THE MODE OF NULL-A:
COMPOSITIONAL COMPUTATION OF A GENERALIZED INVERSE

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a novel algorithm for calculating the preimage of an affine space through a product $J = J_{T-1} \cdots J_0$ of matrices J_t of special form: finding the largest input space $\mathbf{X}$ such that $\mathbf{x} \in \mathbf{X}$ implies $J\mathbf{x} \in \mathbf{Y}$, where $\mathbf{Y}$ is a given output affine space. These special matrices arise in AD, where the Jacobians J describing the linearized computation have precisely this structure: the product of a series of linearized primitive numeric operations. This allows us to use the new algorithm to formulate Null-A mode preimage AD, which finds the affine preimage through the Jacobian or Jacobian transpose of a numeric computation. This is a generalization of the inverse AD problem of solving $J\acute{\mathbf{x}}^* = \acute{\mathbf{y}}^*$ or $J^\mathsf{T}\grave{\mathbf{y}}^* = \grave{\mathbf{x}}^*$. The key is to represent affine spaces in a fashion which lends itself to efficient preimage calculation, in a compositional and *quasi-local* fashion, through a succession of matrices J_t.

Unlike previous methods, Null-A preimage mode AD allows the J_t matrices to be non-square, corresponding to a computer program whose number of active variables swells and shrinks during the computation. When J is square and the initial affine space is a single point, this finds the conventional inverse. But in the more general case, having the entire affine space provides freedom which can be leveraged in a problem-specific manner. We apply the method to small problems on-CPU where the J_t are linearized scalar unary or binary numeric functions; and to larger problems on-GPU where the J_t are linearized aggregate array operations like convolution and attention.

1 INTRODUCTION

Automatic Differentiation (AD) has become widely used in machine learning, mainly to compute gradients for gradient descent. Forward and reverse mode AD compute

$$\acute{\mathbf{y}} = J\acute{\mathbf{x}} \qquad \text{and} \qquad \grave{\mathbf{x}} = J^\mathsf{T}\grave{\mathbf{y}} \tag{1}$$

where J is the Jacobian of a primal computation $\mathbf{y} = f(\mathbf{x})$. The problem of inverting these, solving for $\acute{\mathbf{x}}^*$ and $\grave{\mathbf{y}}^*$ in

$$\acute{\mathbf{y}}^* = J\acute{\mathbf{x}}^* \qquad \text{and} \qquad \grave{\mathbf{x}}^* = J^\mathsf{T}\grave{\mathbf{y}}^* \tag{2}$$

is *inverse AD*, useful for calculations such as steps of the Newton-Raphson method for finding roots and inverse-Hessian-vector products required for many advanced optimization algorithms. Known algorithms for inverse AD assume J to be invertible and have issues when the J_t are not square. In this work, we formulate *preimage AD*, a generalization of inverse AD which expands the space of programs to which it can be efficiently applied by generalizing the problem: propagating affine sets instead of point inverses, and introducing novel data structures and algorithms to support this.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1.1 AD FORMULATION AND NOTATION

AD starts with the observation that, once control flow is resolved and the resulting computation is expressed in single-assignment form, programs that compute $\mathbf{y} = f(\mathbf{x})$ have the following structure:[1]

$$\begin{aligned} \mathbf{x}_0 &:= \mathbf{x} \\ \mathbf{x}_1 &:= f_0(\mathbf{x}_0) \\ &\vdots \\ \mathbf{x}_T &:= f_{T-1}(\mathbf{x}_{T-1}) \\ \mathbf{y} &:= \mathbf{x}_T \end{aligned} \tag{3}$$

where each function f_t denotes a step in the program and transforms one program state to the next, typically by modifying only a few values, while the rest just pass through. The entire program can be viewed as a composition,: $f = f_{T-1} \circ \cdots \circ f_0$ We refer to f as the *primal* computation and the values $\mathbf{x}_t$ as the primal values. Each $\mathbf{x}_t$ is the program state at time t as a vector of all of the active[2] floating-point values in all of the variables and all of the slots of all data structures.

By the chain rule, the Jacobian J of the whole program f is simply the product $J = J_{T-1} \cdots J_0$ of the Jacobians J_t of the individual steps.[3] Further: $J^\mathsf{T} = {J_0}^\mathsf{T} \cdots {J_{T-1}}^\mathsf{T}$ Since f_t varies from the identity function only along a few dimensions of its input and output, each stepwise Jacobian J_t is sparse, differing from the identity matrix in just a few entries. While the stepwise Jacobians J_t are sparse, their product J is in general dense, and of size $O(n^2)$ if the program state is of size $O(n)$, and thus in general intractable to compute and store. Forward and reverse AD do not compute and store J. Instead, they take advantage of associativity while computing *Jacobian-vector products* (forward mode)

$$J\acute{\mathbf{x}} = J_{T-1} \cdots J_1 J_0 \acute{\mathbf{x}} = J_{T-1}(\cdots(J_1(J_0\acute{\mathbf{x}}))\cdots) \tag{4}$$

or *vector-Jacobian products* (reverse mode)

$$(\grave{\mathbf{y}}J)^\mathsf{T} = J^\mathsf{T}\grave{\mathbf{y}}^\mathsf{T} = {J_0}^\mathsf{T} \cdots {J_{T-2}}^\mathsf{T}{J_{T-1}}^\mathsf{T}\grave{\mathbf{y}}^\mathsf{T} = {J_0}^\mathsf{T}(\cdots({J_{T-2}}^\mathsf{T}({J_{T-1}}^\mathsf{T}\grave{\mathbf{y}}^\mathsf{T}))\cdots). \tag{5}$$

Here, the column vector $\acute{\mathbf{x}}$ is a *tangent* of the input $\mathbf{x}$ and the row vector $\grave{\mathbf{y}}$ is a *cotangent*[4] of the output $\mathbf{y}$. Each step multiplies a sparse stepwise Jacobian, or its transpose, with a (co)tangent of size $O(n)$. So step $\mathbf{x}_{t+1} := f_t(\mathbf{x}_t)$ in the original program corresponds to an analogous step

$$\acute{\mathbf{x}}_{t+1} := J_t\acute{\mathbf{x}}_t \qquad \text{(forward AD)} \tag{6}$$

$$\grave{\mathbf{x}}_t^\mathsf{T} := {J_t}^\mathsf{T}\grave{\mathbf{x}}_{t+1}^\mathsf{T} \qquad \text{(reverse AD)} \tag{7}$$

in the computation of the Jacobian-vector or vector-Jacobian product. Since the sparsity structure of J_t corresponds to the sparsity of step f_t, each of these steps in computing Jacobian-vector and vector-Jacobian products reads and writes just those elements in $\acute{\mathbf{x}}_t$ or $\grave{\mathbf{x}}_t$ that correspond to the elements in $\mathbf{x}_t$ read and written by f_t, while the rest just pass through.

1.2 INVERSE AD OF CONSTANT-WIDTH COMPUTATIONS

Computing inverse Jacobians can be done in a similar fashion by observing that

$$J^{-1} = J_0^{-1} \cdots J_{T-1}^{-1}. \tag{8}$$

For J to be invertible it must be square, meaning f must have the same number of inputs and outputs. Since J^{-1} is of size $O(n^2)$, and thus intractable to compute and store, Pearlmutter & Siskind (2024) observed that one can instead compute *inverse-Jacobian-vector products* (reverse inverse mode)

$$J^{-1}\acute{\mathbf{y}}^* = J_0^{-1} \cdots J_{T-2}^{-1}J_{T-1}^{-1}\acute{\mathbf{y}}^* = J_0^{-1}(\cdots(J_{T-2}^{-1}(J_{T-1}^{-1}\acute{\mathbf{y}}^*))\cdots) \tag{9}$$

[1] We use lower case for scalars, lower case bold for (row and column) vectors, upper case for matrices, upper case bold for affine spaces, juxtaposition for matrix product, and $M^{-\mathsf{T}} = (M^{-1})^\mathsf{T}$ for matrix inverse transpose.

[2] In the AD literature, an *active* value is one that participates in the derivative computation. For example, when computing the derivative of $f(x_1, x_2)$ wrt x_1, the variable x_1 along with all other values that depend on x_1 would be active, while x_2 would not be.

[3] The Jacobian of f at $\mathbf{x}$ is the matrix of its partial derivatives, $(J)_{ij} = \partial(f(\mathbf{x}))_i/\partial(\mathbf{x})_j$.

[4] This is a dual construction, so $\grave{\mathbf{x}}$ is a linear mapping of $\acute{\mathbf{x}}$ to $\mathbb{R}$. When $\acute{\mathbf{x}}$ is a finite dimensional vector, we can regard $\grave{\mathbf{x}}$ as a vector of the same dimensionality, and the mapping as their dot product. More pedantically, if $\acute{\mathbf{x}}$ is a column vector then $\grave{\mathbf{x}}$ is a row vector, so $\grave{\mathbf{x}}\acute{\mathbf{x}}$ is a 1×1 matrix, *i.e.*, a scalar. The relationship between forward and reverse AD is that if $\mathbf{y} = f(\mathbf{x})$, forward AD takes $\acute{\mathbf{x}}$ to $\acute{\mathbf{y}}$, reverse AD takes $\grave{\mathbf{y}}$ to $\grave{\mathbf{x}}$, and $\grave{\mathbf{x}}\acute{\mathbf{x}} = \grave{\mathbf{y}}\acute{\mathbf{y}}$.

and *vector-inverse-Jacobian products* (forward inverse mode)

$$(\grave{\mathbf{x}}^* J^{-1})^\mathsf{T} = J^{-\mathsf{T}} \grave{\mathbf{x}}^{*\mathsf{T}} = {J_{T-1}}^{-\mathsf{T}} \cdots {J_1}^{-\mathsf{T}} {J_0}^{-\mathsf{T}} \grave{\mathbf{x}}^{*\mathsf{T}} = {J_{T-1}}^{-\mathsf{T}} (\cdots ({J_1}^{-\mathsf{T}} ({J_0}^{-\mathsf{T}} \grave{\mathbf{x}}^{*\mathsf{T}})) \cdots) \quad (10)$$

through appropriate associativity.

This assumes that each stepwise Jacobian J_t is invertible, and therefore square, implying that the computation needs to be *constant width*: the number of active floating-point values does not increase or decrease during execution. This is the case for certain kinds of steps f_t, and further, for these kinds of steps, the inverse Jacobian steps J_t^{-1} are also structurally sparse, with the same structure as the Jacobian steps J_t, writing and reading those elements in $\grave{\mathbf{x}}_t^*$ or $\acute{\mathbf{x}}_t^*$ that correspond to the elements in $\mathbf{x}_t$ read and written by f_t.

However most programs are not constant width, with the number of active floating-point values increasing and decreasing as the computation proceeds. This is called *swell*. Ways to change the commutativity and/or associativity to aggregate program stretches into segments that are constant width, thus eliminating swell, have been explored (Naumann, 2024; Pearlmutter & Siskind, 2024), but this does not work well in practice because the stretches tend to be long, and the number of input/output variables in each aggregate step tends to be large, necessitating explicit representation and inversion of the Jacobians of these aggregate steps.

1.3 Handling swell by affine-space propagation

Here, we introduce a new way to lift the constant-width restriction and handle swell. The essential idea is to generalize from the inverse, which requires invertible functions, to the preimage, which does not. If $\mathbf{y} = f(\mathbf{x})$ but f is not invertible, so $\mathbf{x} = f^{-1}(\mathbf{y})$ is not well defined, one can generalize from inverses to *preimages*. Let $\mathbf{X}$ and $\mathbf{Y}$ be sets. The image $\mathbf{Y}$ of $\mathbf{X}$ under f is

$$\mathbf{Y} = \textsc{im}(f, \mathbf{X}) = \{f(\mathbf{x}) \mid \mathbf{x} \in \mathbf{X}\} \quad (11a)$$

and the preimage $\mathbf{X}$ of $\mathbf{Y}$ under f is the largest set $\mathbf{X}$ such that $\textsc{im}(f, \mathbf{X}) \subseteq \mathbf{Y}$,

$$\mathbf{X} = \textsc{preim}(f, \mathbf{Y}) = \{\mathbf{x} \mid f(\mathbf{x}) \in \mathbf{Y}\}. \quad (11b)$$

This generalizes the inverse, since when f is invertible $\textsc{preim}(f, \{\mathbf{y}\}) = \{f^{-1}(\mathbf{y})\}$. And these are compositional

$$\textsc{im}(g \circ f, \mathbf{X}) = \textsc{im}(g, \textsc{im}(f, \mathbf{X})) \quad (12a)$$

$$\textsc{preim}(g \circ f, \mathbf{Y}) = \textsc{preim}(f, \textsc{preim}(g, \mathbf{Y})) \quad (12b)$$

so we can calculate the preimage of an entire program by taking the preimage through each step in reverse order.

An affine space is a linear subspace plus an origin, and can be represented as an origin vector and a finite set of basis vectors.[5] It denotes the set of vectors that can be represented as the sum of the origin vector and a linear combination of the basis vectors. Note that the images and preimages of affine spaces under linear maps (or more generally, affine maps) are also affine spaces, *i.e.*, the representation is closed under IM and PREIM of linear functions. Further note that when computing Jacobian-vector and vector-Jacobian products, the functions in play are in fact linear. This makes affine spaces a suitable representation of sets of values for calculating $\textsc{preim}(J, \acute{\mathbf{Y}}^*)$. This is the natural generalization of the inverse for non-constant-width f or non-square J or J_t, since it collapses to the ordinary inverse if we let $\acute{\mathbf{Y}}^* = \{\acute{\mathbf{y}}^*\}$ or $\grave{\mathbf{X}}^* = \{\grave{\mathbf{x}}^*\}$ as appropriate, and assume J invertible.

1.4 Efficient representation of affine spaces

We will represent the origin of an affine space as a column vector and the set of basis vectors as columns of a matrix. This representation is very ambiguous, with degrees of freedom available for both the origin and the non-orthogonal possibly-overcomplete basis of un-normalized vectors. We will take advantage of this, resolving the ambiguity as computationally convenient.

At first blush, the image and preimage computations of affine spaces through linear maps seem inherently nonlocal and burdensome. The sets of vectors denoted by the affine space are the (co)tangent vectors of the entire program state, implying that the origin and basis vectors of these

[5] As a technicality, an affine space might be the empty set. We do not need to represent this since when this arises, the preimage is empty, so we raise an exception.

affine spaces will have the dimensionality of the entire program state. For an arbitrary linear map, the derivation of an element of the representation of an affine space for the image or preimage of another affine space can involve computation over all of the elements of the representation of the original affine space. And preimages may have more or fewer rows, since the number of live values in the program can change; and different numbers of columns, as the dimensionality of the affine space, and therefore the number of basis vectors, can change. But surprisingly, for the particular kinds of linear maps that correspond to stepwise Jacobians J_t of functions f_t for the steps in a program, the preimage computation can be made compositional and *quasi-local*.

To give a concrete example, if we take the preimage through an $(n-1) \times n$ matrix, we would in general expect to have to add an extra column to the basis matrix. If the basis matrix is nearly the identity matrix, this extra column can be sparse, and the changes to other columns will also be sparse.

1.5 Affine space slices

The central construct we introduce for making preimage AD compositional, quasi-local, and efficient is a data structure called *affine space slices* that represent rows of the origin vector and basis matrix. Traditional AD distributes the elements (rows) of intermediate Jacobian-vector and vector-Jacobian products so that each element is stored in a different variable or slot, typically associated with the corresponding primal value. These associations of primal values with (co)tangents are often called *dual numbers* or *(co)bundles*. The program-state vector is distributed among these along with the intermediate Jacobian-vector and vector-Jacobian product state. In traditional AD, the values of the distributed elements of the intermediate Jacobian-vector and vector-Jacobian product state are simply numbers. Here, we take them to be affine space slices. This is a natural choice, since each row of the origin vector and basis matrix is associated with an active primal value. And just as in forward or reverse AD, the primal access patterns determine the access patterns of the associated values, in this case the affine space slices. However, the induced access patterns are not entirely local, and the number of columns of the basis matrix, and therefore the size of the affine space slices, changes in lock step during the computation. In order to avoid adjusting them all at each preimage step, we do the non-local parts of the updates lazily, by queuing them up on a global list that is applied as they are accessed later.

1.6 Nesting

When done correctly, traditional AD *nests* (Maclaurin et al., 2015b; Manzyuk et al., 2019; Pearlmutter & Siskind, 2007; 2008; Radul et al., 2012; Siskind & Pearlmutter, 2005; 2008). The representations for dual numbers and (co)bundles can be designed so that they can contain nested dual numbers and (co)bundles so that one can take higher-order derivatives and derivatives of functions that take derivatives of other functions. This allows computing Hessian-vector products (Pearlmutter, 1994; Werbos, 1992). Doing this in a way that avoids *perturbation confusion* (Manzyuk et al., 2019; Siskind & Pearlmutter, 2005; 2008) requires mechanisms to distinguish (co)tangents of different derivative-operator invocations. Similarly, here we allow affine space slices to nest. The origin and basis elements of an affine space slice can themselves be affine space slices. Since our framework includes the same mechanisms to avoid perturbation confusion as is used in traditional AD, our framework similarly allows nesting of preimage AD to compute inverse-Hessian-vector products, as well as other arbitrary combinations of AD operators.

We proceed to describe this new data structure in detail, and to show how it allows efficient preimage calculations through the sparse matrices that correspond to the Jacobians of steps of a numeric computer program.

2 Multiplication of a structurally-sparse matrix represented as a graph with a dense vector

AD can be viewed as operating on the floating-point computation graph underlying a numeric program. In this directed graph, vertices without inedges are inputs and are labeled with input variables, vertices without outedges are outputs and are labeled with output variables, vertices with inedges are labeled with the function they compute, and inedges are labeled with argument positions. A vertex is created for each input variable and primitive-function invocation. All control flow and invocation of nonprimitive functions is abstracted away. Primitive functions might be just addition, subtraction, multiplication, and division, or might include higher-level operations such as those included in PyTorch. Fig. 3 in §A shows the graph created for the simple program of Fig. 2 in §A. This graph can be constructed explicitly by a tracing process, as is done by AD systems for reverse

mode based on overloading (*e.g.*, PyTorch). It might also be implicit in AD systems for forward mode or those based on source-code transformation. §I contains code to do tracing.

The primal function can be computed from this graph by topologically sorting the vertices, labeling each input vertex with an input value, and computing a label for each noninput vertex, in order, by applying the function label to the values in the parents. This corresponds to executing the program in (3).

AD then linearizes this graph. Each inedge labeled i on a vertex with n inedges is labeled with $\frac{\partial f(x_1,\ldots,x_m)}{\partial x_i}\Big|_{x_1,\ldots,x_m}$. Each such linearized vertex can be interpreted as a stepwise Jacobian. The tangent values flow through the graph. As they flow along an edge they are multiplied by the edge label. Vertices add the values from the inedges and fan them out to the outedges. The topologically sorted collection of vertices can be interpreted as computing a Jacobian-vector product through forward mode (4). Each linearized vertex computes a dot product:

$$\nabla_{x_1,\ldots,x_m} f(x_1,\ldots,x_m)|_{x_1,\ldots,x_m} \cdot [\acute{x}_1,\ldots,\acute{x}_m] \tag{13}$$

Fig. 4 in §A shows the linearization of Fig. 3. The code in §I performs linearization simultaneously with tracing. Since forward mode processes the linearized graph in the same order as the primal graph, it need not be explicitly constructed and is not typically done. The code in §I does explicitly construct the graph, for pedagogical purposes, to illustrate the symmetry between all four AD modes. The code in §L traverses this linearized graph to compute Jacobian-vector products.

Computing a vector-Jacobian product through reverse mode (5) corresponds to *edge reversal*, reversing the direction of all edges, swapping inputs and outputs, and processing the vertices in reverse order. When doing this, addition becomes fanout and fanout becomes addition. This can be seen by the relationship between their Jacobians: $(1 \quad 1)^\mathsf{T} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ Fig. 5 in §A shows the edge reversal of Fig. 4. The code in §K performs edge reversal. The code in §M traverses this edge-reversed graph to compute vector-Jacobian products.

We see from the above that, after linearization and possible edge reversal, computing Jacobian-vector and vector-Jacobian products by traversing this graph constitutes multiplication of a structurally-sparse matrix with a dense vector. The structure of the sparsity follows the structure of the primal. The graph is derived compositionally from the primal and all computations are local. Each dot product (13) accesses just those elements of the (co)tangent Jacobian-vector or vector-Jacobian product state for a given stepwise Jacobian as the corresponding program step.

3 The preimage of the multiplication of a structurally-sparse matrix represented as a graph with a dense vector

We compute generalized inverse-Jacobian-vector and vector-inverse-Jacobian products by computing the preimage of vector-Jacobian and Jacobian-vector products. To do this, fanout must be made explicit, because although the image of fanout is a noöp, the preimage of fanout is not. Thus preimage AD operates on a linearization with *explicit fanout.* This is done by taking each vertex in the linearization and replacing it with a pair of vertices, the first having fanin and only computing a sum, and the second having fanout and only fanout. Fig. 6 in §A shows the result of introducing explicit fanout to the linearization in Fig. 4. Note that fanout introduction and edge reversal commute. Applying just fanout introduction allows computing vector-inverse-Jacobian products (reverse inverse mode). Applying both, in either order, allows computing inverse-Jacobian-vector products (forward inverse mode).

For implementation convenience, we wish to avoid computing the preimage of addition with more than two inputs or the preimage of fanout with more than two outputs. So we introduce an additional *binarization* step that replaces each addition vertex with fanin greater than two with a binary tree of two-input addition vertices and replaces each fanout vertex with fanout greater than two with an inverted binary tree of two-output fanout vertices. Fig. 7 in §A shows the result of binarizing the linearization with explicit fanout in Fig. 6.

Our implementation (§N and §O) does not explicitly construct a binarized graph. Binarization is handled implicitly in the graph traversal. Computing a Jacobian-vector product (forward mode) performs the following steps when processing each vertex in forward topological order:

1. Perform one step of fanout of the tangent value $\acute{x_i}$ in the source of the each inedge, if necessary.
2. Multiply each inedge label $\frac{\partial f(x_1,\ldots,x_m)}{\partial x_i}$ by the tangent value $\acute{x_i}$.
3. Add these products together.

Computing a vector-inverse-Jacobian product (reverse inverse mode) performs the preimage computation of these steps in reverse when processing each vertex in reverse topological order:

1. Compute the preimage of the addition.
2. Compute the preimage of each multiplication.
3. Compute the preimage of one step of fanout, if necessary, for each tangent value.

Computing vector-Jacobian products (reverse mode) and inverse-Jacobian-vector products (forward inverse mode) are analogous to computing Jacobian-vector products (forward mode) and vector-inverse-Jacobian products (reverse inverse mode), respectively, by swapping parents with children and the order of the topological sort.

4 THE PREIMAGE OPERATIONS

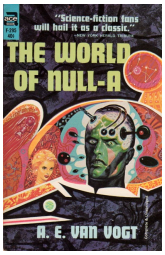
It can be seen from the above that multiplying a structurally-sparse matrix, represented as a graph, with a dense matrix requires just three distinct operations: the two-output fanout in step 1, the multiplication by a constant in step 2, and the two-input addition in step 3. These are all of the operations, and the only operations, for which we need to provide preimage computation for affine spaces. To derive these, we first review the general problem of computing preimages of affine spaces under linear maps. We then specialize this general method for these three particular linear maps.

4.1 THE GENERAL CASE

We represent an affine space $\mathbf{Y}$ as an origin $\mathbf{p}$ (a column vector of dimension n) and a collection of basis vectors (the columns of an $n \times k$ matrix B). Similarly, we represent an affine space $\mathbf{X}$ as an origin $\mathbf{q}$ (a column vector of dimension m) and a collection of basis vectors (the columns of an $m \times l$ matrix C). $\mathbf{Y}$ denotes a set of points in $\mathbb{R}^n$ constructed by adding the origin to a linear combination of the basis vectors, *i.e.*, $\{\mathbf{y} \mid \mathbf{y} = \mathbf{p} + B\mathbf{t}, \mathbf{t} \in \mathbb{R}^k\}$, where $\mathbf{t}$ is a column vector denoting the coefficients of the linear combination. Similarly, $\mathbf{X}$ denotes a set of points in $\mathbb{R}^m$, $\{\mathbf{x} \mid \mathbf{x} = \mathbf{q} + C\mathbf{s}, \mathbf{s} \in \mathbb{R}^l\}$, where $\mathbf{s}$ is a column vector.

Given an $n \times m$ matrix M, the preimage of an affine space $\mathbf{Y}$ under M is the set of all points $\mathbf{x}$ such that $M\mathbf{x} \in \mathbf{Y}$. If the preimage is non-empty, it is also an affine space, $\mathbf{X}$, that satisfies $M(\mathbf{q} + C\mathbf{s}) = \mathbf{p} + B\mathbf{t}$. We can select $\mathbf{q}$ to be any point such that $M\mathbf{q}$ is in $\mathbf{Y}$, by solving $M\mathbf{q} = \mathbf{p} + B\mathbf{t}$:

$$\begin{pmatrix} M & -B \end{pmatrix} \begin{pmatrix} \mathbf{q} \\ \mathbf{t} \end{pmatrix} = \mathbf{p} \tag{14}$$



The preimage is empty if this has no solution. C must be chosen so that M maps the column space of C to the column space of B, *i.e.*, the column space of C must be the set of all vectors $\mathbf{s}$ such that $M\mathbf{s}$ is in the column space of B. In other words, the column space of C is $\{\mathbf{s} \mid \exists \mathbf{t} \text{ s.t. } M\mathbf{s} = B\mathbf{t}\}$. This is equivalent to finding the **null space** of a modified system, $M\mathbf{s} - B\mathbf{t} = \mathbf{0}$, or

$$\begin{pmatrix} M & -B \end{pmatrix} \begin{pmatrix} \mathbf{s} \\ \mathbf{t} \end{pmatrix} = \mathbf{0}. \tag{15}$$

Thus the preimage $\mathbf{X}$ of $\mathbf{Y}$ under M has $\mathbf{q}$ being the first m rows of the solution $\mathbf{z}$ to $A\mathbf{z} = \mathbf{p}$ and C being the first m rows of the kernel of $A = \begin{pmatrix} M & -B \end{pmatrix}$.

In the case of an arbitrary matrix M, computing $\mathbf{q}$ and C is neither compositional nor local. Computing these for a linear map M that reads and writes only a small portion of its inputs and outputs may require reading and writing rows of the affine space that don't correspond to those inputs and outputs. This is the case for all three of the specific linear maps needed: multiplication by a constant, two-input addition, and two-output fanout. But we will mitigate that nonlocality through laziness. We first present the three specific linear maps we need to handle.

4.2 THE SPECIAL CASES NEEDED

Without loss of generality, we can assume that the arguments to the specialized preimage computations are the first one or two columns of a matrix M and the one or two results of the specialized preimage computations are the first one or two rows of M.

Multiplication by a constant corresponds to the $n \times n$ matrix $M_\times$, two-input addition corresponds to the $n \times (n+1)$ matrix M_+, and two-output fanout corresponds to the $(n+1) \times n$ matrix $M_{\texttt{fan}} = {M_+}^\mathsf{T}$.

$$M_\times = \left(\begin{array}{c|c} a & \\ \hline & I_{n-1} \end{array}\right) \qquad M_+ = \left(\begin{array}{c|c} 1 & \\ 0 & \\ \vdots & I_n \\ 0 & \end{array}\right) \qquad M_{\texttt{fan}} = \left(\begin{array}{c} 1\,0\cdots 0 \\ \hline I_n \end{array}\right) \tag{16}$$

The data structure for affine space slices has three slots: a scalar real for the element p at a single row of the origin vector $\mathbf{p}$, a real vector $\mathbf{b}$ for a single row of the basis matrix B, and a timestamp indicating its creation time. This is used to lazily apply nonlocal transformations that have been performed since its creation when it is accessed. §B–§D derive the preimage computations for these three special cases. §S contains code for computing preimages of affine space slices under these specialized linear maps, performing the calculations described in §B–§D.

5 COMPLEXITY

Affine spaces slices are of size $O(k)$. All of the linear algebra operations needed to implement the preimage computations in §B–D (§Q) take $O(k)$ or $O(k^2)$ time. The eager portion and each delayed application of the preimage computations in §B–D makes $O(1)$ calls to these. There are $O(t)$ preimage computations and thus $O(t)$ affine space slices. There can be $O(t)$ delayed applications for each affine spaces slice. Each delayed operation takes $O(k)$ space. The list of delayed operations has $O(t)$ entries. Thus the total space complexity is $O(tk)$ and the total time complexity is $O(t^2k^2)$. When swell is limited, k is bounded. Explicit representation and inversion of the Jacobian would take $O(n^2)$ space and $O(n^3)$ time.

6 CPU BENCHMARK

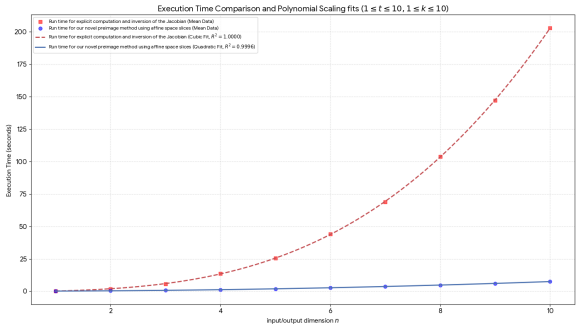


Figure 1: On-CPU Benchmark

To confirm the temporal complexity analysis, we constructed a PYTHON benchmark (§T) that computes the floating point computation graph in §U. The code stacks this graph t times. This benchmark scales independently along the computation time t of the primal, the input/output dimension n, and the swell k. We ran this for $1 \leq t, n, k \leq 10$ (§V), averaged for all t and k for a given n, and plotted measured execution time *vs.* n comparing the computation of an inverse-Jacobian-vector product with Null-A mode preimage AD *vs.* our `jvp` to compute a full Jacobian and `numpy.linalg.inv` and `numpy.matmul` to compute the Jacobian inverse and the matrix-vector product (Fig. 1). This also verified that we computed the exact numerical answer for all 1,000 trials. The point estimates are actual mean run times. The curves are fits to cubic and quadratic respectively. Results show that our benchmark fits the complexity analysis exactly.

7 ARRAY OPERATIONS

The systems we would hope to apply this method to typically involve aggregate array operations implemented in specialized libraries and run on high-performance numeric coprocessors. We therefore implemented Null-A mode preimage AD on the GPU with block (tensor-level) primitives, ran it on optimization problems involving MLPs and GPT-2, benchmarked it against a matrix-free conjugate-gradient (CG) baseline (Hestenes & Stiefel, 1952) built from PYTORCH `jvp` and `vjp`, characterized the tradeoffs between the two, and analyzed numerical stability.

§4.1 gives the preimage of an affine space under a general linear map. The three primitives used in the scalar case suffice when every operation is unary or binary. But a **block** primitive, a whole layer as a single vertex of the computation graph, with a dense Jacobian, does not require any new math, only application of the general case derived above. This is analogous to the development of

AD, which was originally formulated on a handful of scalar primitives, with later production systems hand-coding derivatives of medium-level tensor primitives.

We implemented affine preimage calculations for two new block primitives which suffice for our benchmark of Null-A mode preimage AD *vs.* Conjugate Gradient for finding an input matching a desired output of two networks, an MLP and GPT-2, with either the input data or the weights regarded as the input. See §E, for details, and §F, for benchmark results. To summarize the results, **Null-A mode preimage AD is faster than CG at every point of the GPT-2 sweep except the largest**, by between $2\times$ and $42\times$. For the unbatched MLP, it is faster only at the smallest and most ill-conditioned points. There are also two effects that the benchmarks do not capture: *(a) The two methods do not return the same object.* Null-A mode preimage AD returns the full preimage, while CG just returns one point within that preimage (the Moore-Penrose point). This can be exploited by searching for a solution with particular properties within he preimage; an example of this is given in §G. *(b) Sometimes CG fails.* See the last line of Table 1 in §F.

8 Related work

Algorithms for inverting computational processes has a long history, dating from Newton's Method (Raphson, 1690), and invertible computation and automatic program inversion have been widely studied (Gries, 1981; Kristensen et al., 2024; Matsuda & Wang, 2020; Matsuda et al., 2010). Inverting arbitrary computation is generally intractable and must be applied in highly constrained situations. Generalizing from inversion of programs to preimages requires representing sets of values. When types are discrete, these sets can be represented by streams of elements: this is central to Prolog (Colmerauer & Roussel, 1996), which also generalizes functions to relations.

Griewank (1990), Dixon (1991), Utke (1996), and Hossain (1998, Chapter 4) presented a framework for solving the linear system resulting from the linearization of a function represented as a computer program in which the multiple J_t matrices here were replaced by a single much larger matrix. That framework requires that the computation graph be stored and manipulated in a noncompositional and nonlocal fashion and is thus difficult to do efficiently. Pearlmutter & Siskind (2024) discussed the history of inverse AD, and also presented a compositional and local approach to inverse AD that was limited to constant-width computation and did not handle swell. Naumann (2024) presented a similar constant-width approach, which changed associativity to group Jacobian multiplications and inversions into constant-width stretches. Rahimi (2026) presented an alternative formulation of inverse AD specific to skinny MLPs.

Other special cases have been previously explored: if the primal is stepwise invertible then the tape normally used in reverse mode to store values computed during the forward sweep for use in the reverse sweep can be eliminated as these values can be recomputed during the reverse sweep (Maclaurin et al., 2015a; Pearlmutter, 1995). Invertible differentiable programs are used in a variety of AI systems: BS-Infomax (Bell & Sejnowski, 1995), cICA (Parra et al., 1995; Pearlmutter & Parra, 1996), monotonic neural networks (Wehenkel & Louppe, 2019), normalizing flows (Papamakarios et al., 2021), and stable diffusion (Rombach et al., 2022) use multi-layer structures carefully constructed to be invertible, along with manually-derived inversion procedures. ResNet can be manually inverted (Behrmann et al., 2019), particularly if recast as differential equations (Maleki et al., 2021). But unlike Null-A mode preimage AD, these are all restricted by invertibility.

To our knowledge, there has been no prior work on efficient propagation of affine space to their preimages by linear functions, nor on representations of affine spaces which exploit laziness or deferred execution to efficiently process sequences of operations.

9 Discussion

We presented forward and reverse Null-A mode preimage AD, a method for performing generalized preimage AD. We also exhibited two overloading-based implementations based on dataflow graph traversal. This was done for pedagogical purposes, to explain the method and expose the symmetries. Nothing prevents implementing forward Null-A mode to operate without tracing to construct a graph; it would be analogous to how one implements conventional forward mode without tracing to construct a graph. Moreover, checkpointing (Siskind & Pearlmutter, 2018) can be used to tradeoff time for space for reverse Null-A mode in a fashion analogous to reverse mode. Further, nothing prevents implementing both forward and reverse Null-A preimage mode AD by source-code transformation without tracing; it would be analogous to how one implements conventional forward and reverse

mode by source-code transformation without tracing. While the method as described here is impure—because it uses mutation of a global variable to keep track of the delayed operations that must be performed on affine space slices—it can be made pure simply by threading that list through the computation in A-normal form or continuation-passing style (CPS).

The on-CPU implementation also nests, allowing it to compute Hessian-vector products $\nabla^2_{\mathbf{x}} f(\mathbf{x})\big|_{\mathbf{x}} \acute{\mathbf{x}}$ in three different ways (§P): by computing a Jacobian-vector product of a vector-Jacobian product (forward over reverse, `hvp1`), by computing a vector-Jacobian product of a Jacobian-vector product (reverse over forward, `hvp2`), and by computing a vector-Jacobian product of a vector-Jacobian product (reverse over reverse, `hvp3`). Nesting also allows it to compute inverse-Hessian-vector products $\nabla^{-2}_{\mathbf{x}} f(\mathbf{x})\big|_{\mathbf{x}_1} \acute{\mathbf{x}}_2$ in two different ways: by computing an inverse-Jacobian-vector product of a vector-Jacobian product (reverse inverse over reverse, `ihvp1`) and by computing a vector-inverse-Jacobian product of a vector-Jacobian product (reverse inverse over reverse, `ihvp3`). It further allows the system to compute Newton steps $\nabla^{-2}_{\mathbf{x}} f(\mathbf{x})\big|_{\mathbf{x}_1} \nabla_{\mathbf{x}} f(\mathbf{x})|_{\mathbf{x}_2}$ in two different ways (`newton_step1` and `newton_step3`), by combining either of the two ways of computing an inverse-Hessian-vector product with a vector-Jacobian product. Thus Null-A mode preimage AD also allows optimizing multivariate functions using a generalization of Newton's Method, $x := x - \frac{f'(x)}{f''(x)}$:

$$\dot{\mathbf{x}}^* := \dot{\mathbf{x}}^* - \nabla^{-2}_{\mathbf{x}} f(\mathbf{x})\big|_{\dot{\mathbf{x}}^*} \nabla_{\mathbf{x}} f(\mathbf{x})|_{\dot{\mathbf{x}}^*} \tag{17}$$

10 Limitations & future work

There are a number of limitations intrinsic to inverse or preimage AD in general, the most serious of which is that unlike differentiation, these are not linear operators. The stochastic gradient method relies upon $\nabla\langle f(\mathbf{x})\rangle = \langle\nabla f(\mathbf{x})\rangle$, but nothing like this holds for inverse or preimage, making stochastic methods difficult. Another general issue is that matrix-free methods can be used to solve $J\mathbf{x} = \mathbf{y}$ or $J^{\mathsf{T}}\mathbf{y} = \mathbf{x}$ where conventional AD is used for the product with J or J^{T}. This makes such matrix-free methods a natural comparison to inverse or preimage AD, which after all compute the same result. Precisely delineating the space of problems for which inverse or preimage AD is more efficient than matrix-free methods in concert with conventional AD is a topic we are exploring.

In a more speculative vein, if a function f can be viewed as computing two function in parallel, $f(\mathbf{x}, \mathbf{y}) = (g(\mathbf{x}), h(\mathbf{y}))$, conventional forward and reverse AD of f has complexity equal to the sum of the complexity of applying it to g and h. For Null-A mode preimage AD, we note that if the input affine space is a cross product of affine spaces for g and h, then the output affine space will also be a cross product. Unfortunately our representation does not take advantage of this. This observation hints that more efficient data structures for representing affine spaces, which take advantage of these sorts of symmetries to reduce complexity, might be possible.

11 Conclusion

We presented a general framework, along with a complete Python implementation, for performing Null-A mode preimage AD: a generalization of inverse automatic differentiation (AD), *i.e.*, computing Jacobian-affine-space and Jacobian-transpose-affine-space preimages, even for programs that exhibit increases and decreases in the number of active variables. This allows finding roots of systems of nonlinear equations using the Newton-Raphson Method. Null-A mode preimage AD, when combined with mechanisms that allow nesting, can compute inverse Hessian vector products and thus supports Newton's Method to optimize nonlinear functions. Null-A mode preimage AD implements the requisite computations in a structurally-sparse, compositional, and quasi-local fashion analogous to traditional AD. This means that the same underlying technology that has been previously used to implement traditional AD efficiently in a plethora of languages using both operator overloading and source-code transformation can be used to implement Null-A mode preimage AD in the same languages with analogous increase in efficiency.

The field of machine learning has largely focused on producing either point estimates or probability distributions. Affine spaces can be thought of as something intermediate: more structure than a single point, but much less structure than a distribution. Affine spaces also combine well with convex functions and convex optimization. For this reason, it is our hope that just as reverse-mode AD has been the enabling technique underlying a variety of useful innovations in machine learning, Null-A mode preimage AD might also serve as an enabling method for new machine-learning algorithms.

AI USE STATEMENT

In accordance with the ICLR guidelines, we disclose that Large Language Models (LLMs) were utilized during the research and preparation of this manuscript. Specifically, **Anthropic's Claude** model was used to assist with two tasks:

- **Implementation Sourcing:** The CPU implementation was crafted from scratch entirely by hand (no AI), while the GPU implementation was developed from scratch but with code generated and structured in collaboration with Claude.
- **Drafting Assistance:** Claude was utilized to generate rough initial drafts for paragraphs describing experiments that used the GPU implementation.

All AI-generated code was strictly audited, verified for correctness, and benchmarked by the authors. The core intellectual contributions, experimental design, and final text of the manuscript were directed and approved entirely by the human authors, who assume full responsibility for the content.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our findings, we have taken the following measures:

- **Codebase Availability:** We provide a complete replication package containing both the hand-coded CPU implementation and the AI-assisted GPU implementation. For the review process, this code is included in the anonymous supplementary material as a compressed archive (`.zip`). Upon acceptance, the full repository will be made publicly available on GitHub.
- **Datasets:** All experiments are conducted using the publicly accessible, standardized *Tiny Shakespeare* dataset. No proprietary data is used.
- **Hyperparameters and Environment:** Scripts required to replicate both implementations are fully detailed in §6 and §F and in the supplementary archive. In brief, the CPU benchmarks were run under Ubuntu 26.04 on a laptop and run overnight, while the GPU benchmarks require a DGX Spark running DGX OS 7 and take less than one hour and fifteen minutes to run.

ETHICS STATEMENT

This submission focuses entirely on novel algorithms, benchmarked using small problems and a standard public text dataset (*Tiny Shakespeare*). The work does not involve human subjects, personal data tracking, or hazardous applications. We have reviewed the ICLR Code of Ethics and confirm that this research presents no notable ethical or negative societal concerns.

REFERENCES

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. Invertible residual networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *International Conference on Machine Learning*, volume 97, pp. 573–582, 2019.

Anthony J. Bell and Terrence J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

Alain Colmerauer and Philippe Roussel. The birth of Prolog. In *History of Programming Languages—II*, pp. 331–367. ACM, 1996. doi: 10.1145/234286.1057820.

Laurence C. W. Dixon. Use of automatic differentiation for calculating Hessians and Newton steps. In Andreas Griewank and George F. Corliss (eds.), *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, pp. 114–125. SIAM, 1991.

David Gries. Inverting programs. In *The Science of Programming*, chapter 21, pp. 265–274. Springer, 1981.

Andreas Griewank. Direct calculation of Newton steps without accumulating Jacobians. In Thomas F. Coleman and Yuying Li (eds.), *Large-Scale Numerical Optimization*, pp. 115–137. SIAM, 1990.

Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. doi: 10.6028/jres.049.044.

A.K.M. Shahadat Hossain. *On the Computation of Sparse Jacobian Matrices and Newton Steps*. PhD thesis, Department of Informatics, University of Bergen, 1998. Technical Report 146.

Justin Johnson. Tiny Shakespeare dataset. https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt, 2015. Extracted from the complete works of William Shakespeare.

Andrej Karpathy. nanoGPT: The simplest, fastest repository for training/finetuning medium-sized GPTs. GitHub Repository, 2022. URL https://github.com/karpathy/nanoGPT.

Joachim Tilsted Kristensen, Robin Kaarsgaard, and Michael Kirkedal Thomsen. Jeopardy: An invertible functional programming language. In Torben Ægidius Mogensen and Łukasz Mikulski (eds.), *Reversible Computation*, pp. 124–141. Springer Nature Switzerland, 2024.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In Francis Bach and David Blei (eds.), *International Conference on Machine Learning*, volume 37, pp. 2113–2122, 2015a.

Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: Effortless gradients in numpy. In *ICML AutoML Workshop*, volume 238, pp. 5, 2015b.

Mehrdad Maleki, Mansura Habiba, and Barak A. Pearlmutter. HeunNet: Extending ResNet using Heun's method. In *Irish Signals and Systems Conference*, pp. 1–6, 2021.

O. Manzyuk, B. A. Pearlmutter, A. A. Radul, D. R. Rush, and J. M. Siskind. Perturbation confusion in forward automatic differentiation of higher-order functions. *Journal of Functional Programming*, 29(e12):1–24, September 2019.

Kazutaka Matsuda and Meng Wang. Sparcl: A language for partially-invertible computation. *Proceedings of the ACM on Programming Languages*, 4(ICFP):118:1–118:31, August 2020.

Kazutaka Matsuda, Shin-Cheng Mu, Zhenjiang Hu, and Masato Takeichi. A grammar-based approach to invertible programs. In Andrew D. Gordon (ed.), *European Symposium on Programming*, volume 6012, pp. 448–467. Springer, 2010.

Uwe Naumann. A matrix-free exact Newton method. *Journal on Scientific Computing*, 46(3): A1423–A1440, 2024.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Lucas Parra, Gustavo Deco, and Stefan Miesbach. Redundancy reduction with information-preserving nonlinear maps. *Network: Computation in Neural Systems*, 6(1):61–72, 1995.

B. A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *Transactions on Neural Networks*, 6(5):1212–1228, 1995.

B. A. Pearlmutter and L. C. Parra. A context-sensitive generalization of ICA. In *International Conference on Neural Information Processing, Hong Kong, Sept. 24–27, 1996*, pp. 151–157, 1996.

B. A. Pearlmutter and J. M. Siskind. Lazy multivariate higher-order forward-mode AD. In *Principles of Programming Languages*, pp. 155–160, January 2007.

B. A. Pearlmutter and J. M. Siskind. Using programming language theory to make AD sound and efficient. In *International Conference on Algorithmic Differentiation*, pp. 79–90, August 2008.

B. A. Pearlmutter and J. M. Siskind. Automatic differentiation: Inverse accumulation mode. In *International Conference on Algorithmic Differentiation*, September 2024. arXiv:2411.18786.

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6(1):147–60, 1994.

A. Radul, B. A. Pearlmutter, and J. M. Siskind. AD in Fortran: Implementation via prepreprocessor. In *International Conference on Algorithmic Differentiation*, pp. 273–284, July 2012.

Ali Rahimi. The Hessian of tall-skinny networks is easy to invert. *arXiv:2601.06096*, 2026.

Joseph Raphson. *Analysis Aequationum Universalis*. Typis T. Braddyll, London, 1690.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Computer Vision and Pattern Recognition*, pp. 10674–10685, 2022.

J. M. Siskind and B. A. Pearlmutter. Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD. In *Draft Proceedings of the International Workshop on Implementation and Application of Functional Languages*, September 2005.

J. M. Siskind and B. A. Pearlmutter. Nesting forward-mode AD in a functional framework. *Higher-Order and Symbolic Computation*, 21(4):361–376, December 2008.

J. M. Siskind and B. A. Pearlmutter. Divide-and-conquer checkpointing for arbitrary programs with no user annotation. *Optimization Methods & Software*, 33(03–04):1288–1330, July 2018.

Jean Utke. Efficient Newton steps without Jacobians. In Martin Berz, Christian H. Bischof, George F. Corliss, and Andreas Griewank (eds.), *Computational Differentiation: Techniques, Applications, and Tools*, pp. 253–264. SIAM, 1996.

A. Wehenkel and G. Louppe. Unconstrained monotonic neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (eds.), *Neural Information Processing Systems*, pp. 1534–1544, 2019.

Paul J. Werbos. Neural networks, system identification, and control in the chemical process industries. In D. A. White and D. A. Sofge (eds.), *Handbook of Intelligent Control—Neural, Fuzzy, and Adaptive approaches*, chapter 10, pp. 283–356. Van Norstrand Reinhold, 1992.

A EXAMPLE GRAPH CONSTRUCTIONS

$$
\begin{aligned}
\text{INPUTS} &: x_1, x_2 \\
t_1 &= f(x_1) \\
t_2 &= g(x_1, x_1, x_2) \\
y_1 &= h(t_1, t_2, t_2) \\
y_2 &= p(t_1, t_2, x_2) \\
\text{OUTPUTS} &: y_1, y_2
\end{aligned}
$$

Figure 2: A simple program in single-assignment A-normal form with no control flow. Here, t_1 and t_2 are temporary variables and f, g, h, and p are primitive functions. All variables hold scalar values and all arguments to and results from primitive functions are scalars.

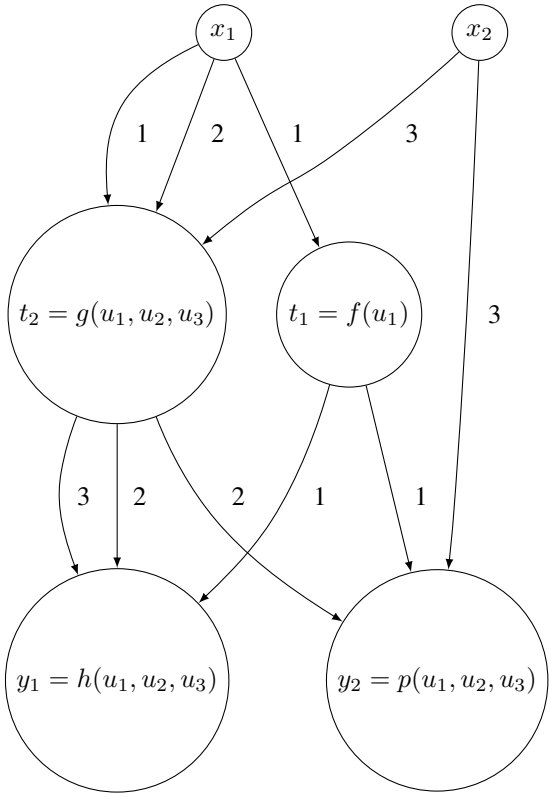


Figure 3: The primal floating-point computation graph corresponding to the program in Fig. 2. Here, u_1, u_2, and u_3 are the formal parameters of the primitive functions. Note that all vertices except for inputs compute some value by calling a primitive function. The edge labels denote the index of the formal parameter of a primitive function a value is passed to.

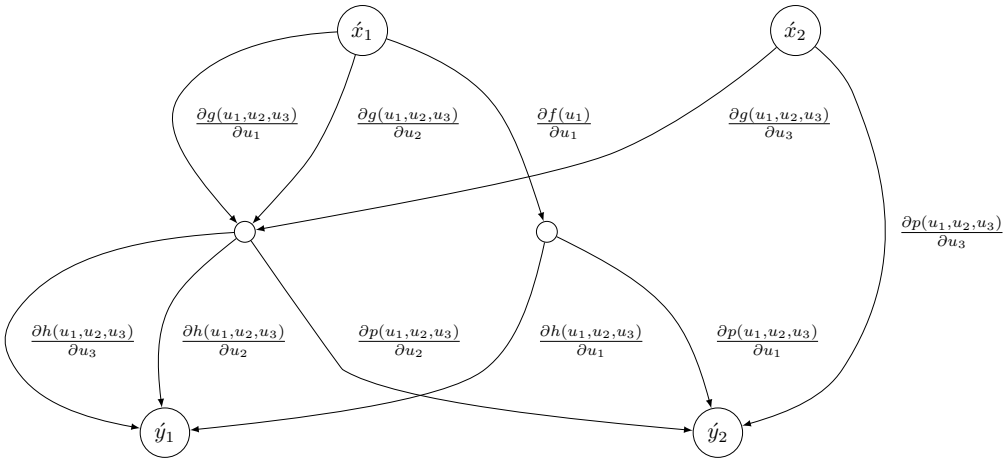


Figure 4: Linearization of the floating-point computation graph in Fig. 3. The inputs and outputs are the tangents associated with the corresponding primal values. Each edge is labeled with the partial derivative of its destination relative to its source, computed at the primal values in the associated primal floating-point computation graph. This graph can be interpreted as a structurally sparse representation of the Jacobian of the program in Fig. 2. It computes a Jacobian-vector product through forward mode by assuming scalar values flow through the network from input tangents to output tangents, interpreting an edge label as multiplication by a constant, and interpreting noninput vertices as summing their inputs. Note that conceptually, this approach could be extended to the case where variables hold vector values and primitive functions accept and return vector values by assuming vector values flow through the network, replacing the partial-derivative edge labels with partial Jacobians, interpreting the edge labels as matrix-vector multiplication, and interpreting noninput vertices as performing vector addition.

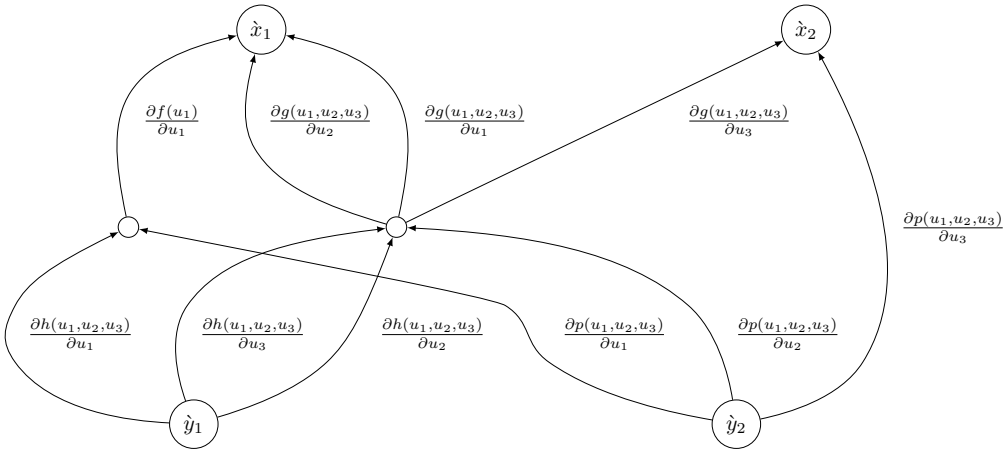


Figure 5: Edge reversal of the linearization in Fig. 4. The inputs and outputs are the cotangents associated with the corresponding primal values. This graph can be interpreted as a structurally sparse representation of the transposition of the Jacobian of the program in Fig. 2, *i.e.*, edge reversal is transposition. It computes a vector-Jacobian product through reverse mode. If one were to extend this to the case where variables hold vector values and primitive functions accept and return vector values, one would have to transpose the partial Jacobian edge labels.

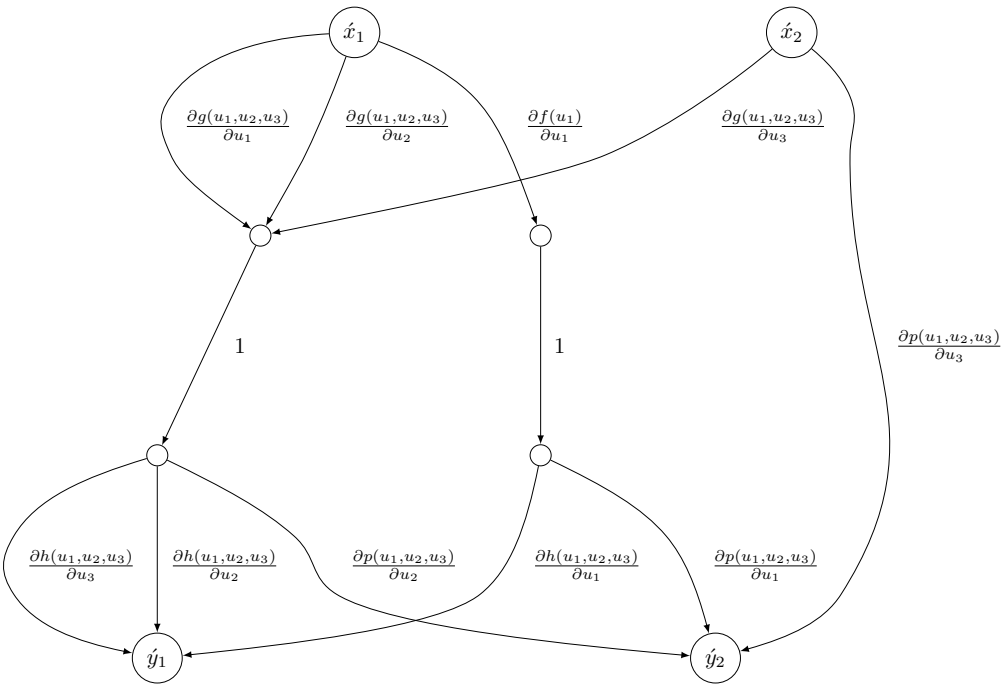


Figure 6: Adding explicit fanout to the linearization in Fig. 4. Explicit fanout must only be added to noninput vertices, *i.e.*, those that correspond to calling primitive functions. Note that in Figs. 4 and 5, inputs have no fanin but may have fanout, outputs have no fanout but may have fanin, and internal vertices may have both fanin and fanout. After adding explicit fanout, internal vertices may have fanin or fanout but not both. Note that edge reversal preserves this property.

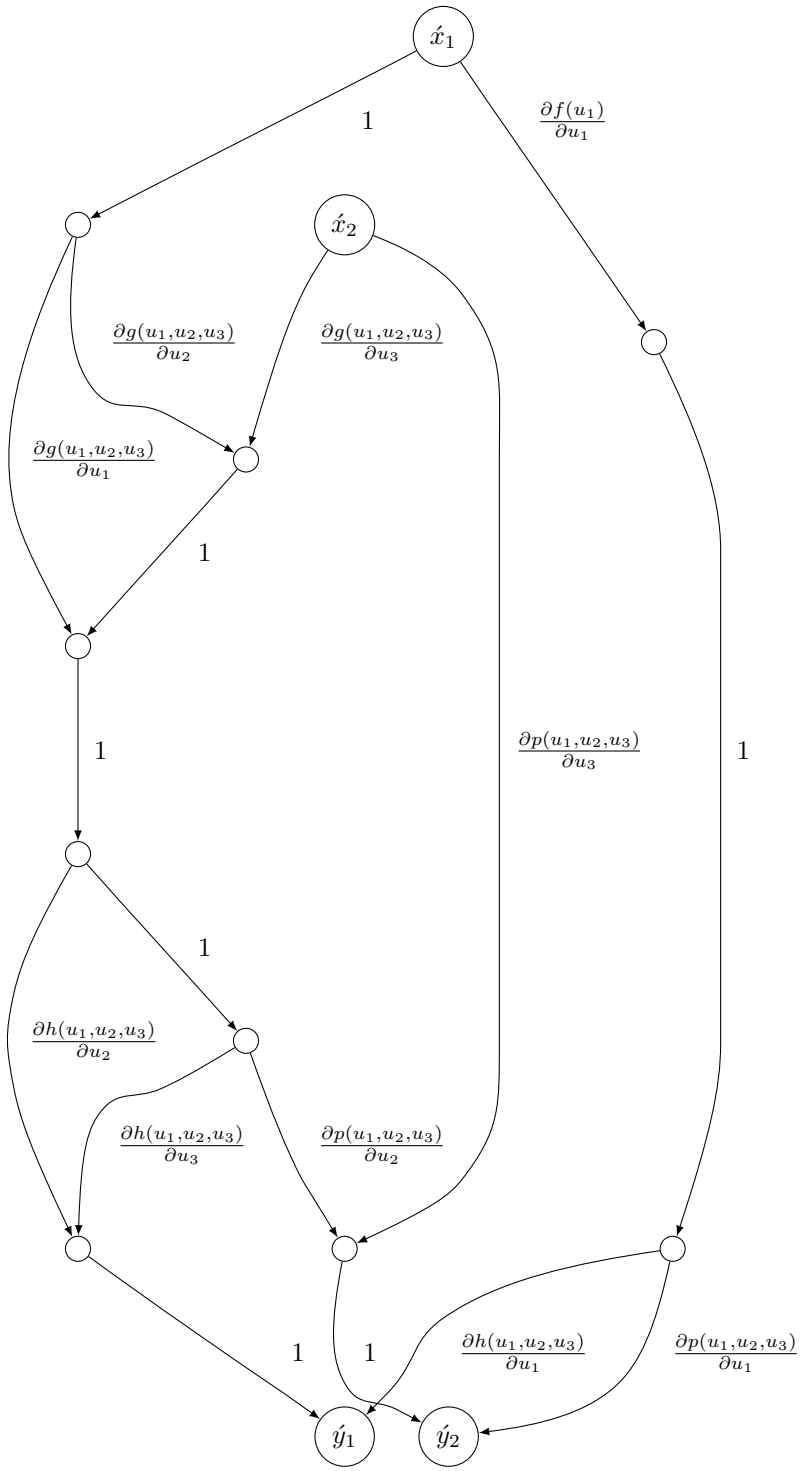


Figure 7: Binarization of the linearization with explicit fanout in Fig. 6. After binarization, each input vertex has no inedges and at most two outedges, each output vertex has no outedges and at most two inedges, and each internal vertex has either one inedge and at most two outedges or one outedge and at most two inedges. Note that edge reversal preserves this property. Edge labels correspond to the preimage computation of $M_\times$ (§B). Vertices with two inedges correspond to the preimage computation of M_+ (§C). Vertices with two outedges correspond to the preimage computation of M_{FAN} (§D).

B THE PREIMAGE OF $M_\times$

Case 1: $a \neq 0$ In this case, $M_\times$ is invertible:

$$M_\times^{-1} = \left(\begin{array}{c|c} 1/a & \\ \hline & I_{n-1} \end{array}\right) \tag{18}$$

The preimage $\mathbf{X}$ of $\mathbf{Y}$ under $M_\times$ is:

$$\{\mathbf{x} \mid \mathbf{x} = M_\times^{-1}(\mathbf{p} + B\mathbf{t}), \mathbf{t} \in \mathbb{R}^k\} \tag{19}$$

Thus:

$$\mathbf{q} = M_\times^{-1}\mathbf{p} = \begin{pmatrix} p_1/a \\ p_2 \\ \vdots \\ p_n \end{pmatrix} \qquad C = M_\times^{-1}\mathbf{B} = \begin{pmatrix} \mathbf{b}_1/a \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} \tag{20}$$

where $\mathbf{b}_i$ is the i^{th} row of B. Note that this is a local operation. This operation returns a new affine space slice with q_1 and $\mathbf{c}_1$. All other affine space slices are unchanged.

Case 2: $a = 0$ **and** $\mathbf{b}_1 \neq \mathbf{0}$ In this case, $A\mathbf{z} = \mathbf{p}$ is underconstrained. We choose a least-squares solution:[6]

$$\mathbf{q} = \mathbf{p} - \tfrac{p_1}{\|\mathbf{b}_1\|^2} B\mathbf{b}_1 \qquad C = \left(\mathbf{e}_1 \middle| B\big(I - \tfrac{\mathbf{b}_1{}^\mathsf{T}\mathbf{b}_1}{\|\mathbf{b}_1\|^2}\big)\right) \tag{21}$$

where $\mathbf{e}_1 = [1, 0, \ldots, 0]^\mathsf{T}$ and $\mathbf{b}_1{}^\mathsf{T}\mathbf{b}_1$ denotes an outer product. This grows the basis by one. This is a quasi-local operation. Equation (21) implies that $q_1 = 0$ and $\mathbf{c}_1 = [1, 0, \ldots]$. These are returned as a new affine space slice. But the remaining affine space slices must be modified according to (21). A delayed operation is queued up to apply (21) to the remaining affine space slices when they are next accessed. This requires storing p_1 and $\mathbf{b}_1$ for use by that delayed operation.

Case 3: $a = 0$**,** $\mathbf{b}_1 = \mathbf{0}$**, and** $p_1 \neq 0$ In this case, the preimage is empty because (14) has no solution, so we raise an exception.

Case 4: $a = 0$**,** $\mathbf{b}_1 = \mathbf{0}$**, and** $p_1 = 0$ In this case, $M_\times$ maps a vector $\mathbf{x}$ to a vector $\mathbf{y}$ where $y_1 = 0$ and $y_i = x_i$ for $i > 1$. Since $y_1 = p_1 + \mathbf{b}_1 \cdot \mathbf{t}$, this always holds. Thus $\mathbf{q} = \mathbf{p}$. A has the form $(I \quad -B)$ but where the first row is all zero. C, the kernel of A, is $(\mathbf{e}_1 \quad B)$ where $\mathbf{e}_1 = [1, 0, \ldots, 0]^\mathsf{T}$. In this case, the origin remains the same and the basis grows by one, a new column with a one for the first row and zeros for all other rows. This is a quasi-local operation where a one is concatenated onto the head of the basis for the returned affine space slice and a delayed operation is queued up to concatenate a zero on the head of the basis of each of the remaining affine space slices when they are next accessed.

C THE PREIMAGE OF M_+

The preimage $\mathbf{X}$ of $\mathbf{Y}$ under M_+ is

$$\mathbf{q} = \begin{pmatrix} 0 \\ p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} \qquad C = \begin{pmatrix} 1 & \mathbf{0} \\ -1 & \mathbf{b}_1 \\ 0 & \mathbf{b}_2 \\ \vdots & \vdots \\ 0 & \mathbf{b}_n \end{pmatrix}. \tag{22}$$

This grows the basis by one. This is a quasi-local operation. The first two rows of $\mathbf{q}$ and C are returned as affine space slices. But a delayed operation must be queued up to concatenate a zero onto the head of the basis of each of the remaining affine space slices when they are next accessed.

[6] Any solution would serve here. Optimizing sparseness or efficiency of representation would also be reasonable choices.

D THE PREIMAGE OF M_{FAN}

The preimage $\mathbf{X}$ of $\mathbf{Y}$ under M_{FAN} is computed as follows: First note that $\mathbf{p}$ and B have $n+1$ rows. Assume that B has k columns. Let $\mathbf{r} = \mathbf{b}_1 - \mathbf{b}_2$. Find a least squares (of $\mathbf{z}$) solution to:

$$\mathbf{r} \cdot \mathbf{t} = p_2 - p_1 \qquad\qquad \mathbf{t} = \frac{(p_2 - p_1)\mathbf{r}}{\|\mathbf{r}\|^2} \tag{23}$$

One exists iff $\|\mathbf{r}\|^2 \neq 0$. This happens when $\mathbf{b}_1 \neq \mathbf{b}_2$.

Case 1: $\mathbf{b}_1 \neq \mathbf{b}_2$ In this case:

$$\mathbf{q} = \begin{pmatrix} p_2 + \mathbf{b}_2 \cdot \mathbf{t} \\ \vdots \\ p_{n+1} + \mathbf{b}_{n+1} \cdot \mathbf{t} \end{pmatrix} \qquad\qquad C = \left(\begin{array}{c} \mathbf{b}_2 \\ \hline \vdots \\ \hline \mathbf{b}_{n+1} \end{array}\right) W \tag{24}$$

where $W : k \times (k-1)$ is the kernel of $\mathbf{r}$ regarded as a one-row matrix. Note that $p_1 + \mathbf{b}_1 \cdot \mathbf{t} = p_2 + \mathbf{b}_2 \cdot \mathbf{t}$, $\mathbf{b}_1 W = \mathbf{b}_2 W$, and the first row is eliminated.

W has a closed form. Since $\mathbf{b}_1 \neq \mathbf{b}_2$, let d index the first element where $r_d \neq 0$. Let $\mathbf{u} = -\frac{\mathbf{r}}{r_d}$ with the d^{th} element removed. (The removed element would be equal to -1.) W is an identity matrix with $\mathbf{u}$ added as additional row d:

$$W = \left(\begin{array}{c|c} I_{d-1} & \\ \hline \mathbf{u} & \\ \hline & I_{k-d} \end{array}\right) = \left(\begin{array}{c|c} \multicolumn{2}{c}{I_{d-1}} \\ \hline -r_1/r_d \quad \cdots \quad -r_{d-1}/r_d & -r_{d+1}/r_d \quad \cdots \quad -r_k/r_d \\ \hline \multicolumn{2}{c}{I_{k-d}} \end{array}\right) \tag{25}$$

$$w_{ij} = \begin{cases} -r_j/r_d & i = d \text{ and } j < d \\ -r_{j+1}/r_d & i = d \text{ and } j \geq d \\ 1 & (i < d \text{ and } j = i) \text{ or } (i > d \text{ and } j = i+1) \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

Since B has k columns, $\mathbf{u}$ has $k-1$ elements, and thus W is $k \times (k-1)$ and C will have $k-1$ columns, This shrinks the basis by one:

$$c_{ij} = \begin{cases} b_{i+1,j} - b_{i+1,d}\frac{r_j}{r_d} & j < d \\ b_{i+1,j+1} - b_{i+1,d}\frac{r_{j+1}}{r_d} & j \geq d \end{cases} \tag{27}$$

This is a quasi-local operation. The first and second rows correspond to the two variables being fanned out. One of them (corresponding to the first row) is dropped. The result (corresponding to the second row) is returned as a new affine space slice. A delayed operation is queued up to apply (27) to the remaining affine space slices when they are next accessed. This requires storing d, $\mathbf{r}$, and $\mathbf{t}$ for use by that delayed operation.

Case 2: $\mathbf{b}_1 = \mathbf{b}_2$ **and** $p_1 = p_2$ In this case:

$$\mathbf{q} = \begin{pmatrix} p_2 \\ \vdots \\ p_{n+1} \end{pmatrix} \qquad\qquad C = \begin{pmatrix} \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_{n+1} \end{pmatrix} \tag{28}$$

In this case, $\mathbf{q}$ and C are the same as $\mathbf{p}$ and B, respectively, with the first row removed. This is a local operation implemented by just dropping one of the two input affine space slices and returning the other.

Case 3: $\mathbf{b}_1 = \mathbf{b}_2$ **and** $p_1 \neq p_2$ In this case, the preimage is empty because (14) has no solution, so we raise an exception.

E ON-GPU IMPLEMENTATION OF NULL-A MODE PREIMAGE AD

- `feedforward`: $\mathbf{y} = \sigma(W\mathbf{x} + \mathbf{b})$, Jacobian derived by hand. It is very wide. Because the layer consumes many more active values than it produces, it is a case of swell.
- `transformer_block`: the entire GPT-2 block (layer norm, causal multi-head self attention, residual, layer norm, GELU feedforward, residual) is one vertex. The Jacobian is obtained by using `torch.func.jacrev` over the block with respect to the input activations and all twelve parameter tensors at once. (A block primitive needs only **some** way to obtain a dense Jacobian, not a hand derivation, so the cost of adding one is low.)

Everything runs in PYTORCH on the GPU (a Dell GB10, aka NVidia DGX Spark). We verified the correctness of the block implementations against versions using the scalar implementations.

E.1 COMPARISON AGAINST CG AT SCALE

We used these block GPU primitives to benchmark Null-A mode preimage AD against CG on two ML-scale models: an MLP and GPT-2, doing parameter hypersweeps to determine when Null-A mode preimage AD is better or worse than CG. We did two sets of benchmarks, one without batching and one with, because batching introduces swell.[7] The baseline solves $J\acute{\mathbf{x}}^* = \acute{\mathbf{y}}^*$ for the minimum-ℓ_2-norm solution $J^\mathsf{T}(JJ^\mathsf{T})^{-1}\acute{\mathbf{y}}^*$ by conjugate gradients on JJ^T, using only `torch.func.jvp` and `torch.func.vjp`, so J is never constructed. That returns the Moore-Penrose point.

[7] Even though a transformer is constant width relative to the input data, it is *not* constant width relative to the weights, which are the active inputs to AD here. This is aggravated with batching. Further, at the micro level, attention units are not constant width due to the fanout to Q, K, and V. Further, convolutions and residual connections are not constant width. Essentially no standard deep-learning model is constant width.

F ON-GPU BENCHMARKS OF NULL-A MODE PREIMAGE AD *vs.* CG

Table 1: MLP batched on-GPU

batch size S	`dim X`	Null-A (s)	CG (s)	$\frac{\text{Null-A}}{\text{CG}}$	CG residual
1	2080	0.0165	0.0074	2.25	8.9×10^{-8}
2	2048	0.5722	0.0851	6.72	1.4×10^{-4}
4	1984	2.0108	0.1373	14.65	2.8×10^{-4}
8	1856	4.8536	0.2966	16.36	9.1×10^{-4}
16	1600	10.1089	1.9339	5.23	6.9×10^{-3}
32	1088	19.3625	44.1483	0.44	1.2×10^{1}

Table 2: MLP unbatched on-GPU

depth	width	`P`	`dim X`	Null-A (s)	CG (s)	$\frac{\text{Null-A}}{\text{CG}}$
1	32	1056	1024	0.0079	0.0050	1.6
2	32	2112	2080	0.0165	0.0074	2.2
3	32	3168	3136	0.0271	0.0072	3.8
4	8	288	280	0.0089	0.0097	0.9
4	16	1088	1072	0.0160	0.0088	1.8
4	24	2400	2376	0.0251	0.0090	2.8
4	32	4224	4192	0.0402	0.0072	5.5
4	40	6560	6520	0.0642	0.0074	8.6
4	48	9408	9360	0.0964	0.0074	13.1
4	55	12320	12265	0.1427	0.0095	15.0
5	32	5280	5248	0.0509	0.0077	6.6
6	32	6336	6304	0.0674	0.0139	4.8
7	32	7392	7360	0.0854	0.0137	6.2
8	32	8448	8416	0.1088	0.0188	5.8
8	40	13120	13080	0.2023	0.0180	11.2
8	48	18816	18768	0.3648	0.0195	18.7
8	55	24640	24585	0.5891	0.0197	29.8

`P` and `dim X` are the number of rows and columns in the affine space returned respectively.

In short, Null-A mode preimage AD is faster on ill-conditioned, wide-output, small-to-moderate problems. CG is faster on well-conditioned, narrow-output, large ones. If one takes the preimage relative to the weights, and the weights are poorly initialized, the system can be ill conditioned, requiring CG to perform more iterations. In the worst case, CG does not converge and returns no usable answer. A well-initialized MLP is the worst case for Null-A mode preimage AD and the best case for CG. Transformers are the opposite.

Null-A mode preimage AD beats CG more when doing sufficiently large **batching**, because it increases the output width and worsens the conditioning at the same time. Thus Null-A mode preimage AD beat CG even on MLP with batching.

At $S = 32$ the baseline **fails**: it exhausts the iteration cap and returns a residual of 12, *i.e.*, no answer at all, while Null-A mode preimage AD returns the whole preimage in a little over a third of the time. Note also that the baseline's accuracy degrades monotonically with S, from 10^{-7} to 7×10^{-3} before it fails outright, so the two methods are not returning comparable quality even at the sizes where the baseline is faster.

Table 3: GPT-2 unbatched on-GPU

n_embd	n_layer	n_head	n_seq	P	dim X	Null-A (s)	CG (s)	$\frac{\text{Null-A}}{\text{CG}}$
8	2	2	8	1744	1680	0.0384	1.6078	0.02
12	2	2	8	3768	3672	0.0552	2.0095	0.03
16	1	2	8	3280	3152	0.0321	0.4107	0.08
16	2	1	8	6560	6432	0.0837	1.0426	0.08
16	2	2	2	6560	6528	0.0429	0.2318	0.19
16	2	2	4	6560	6496	0.0560	0.3815	0.15
16	2	2	8	6560	6432	0.0738	1.0019	0.07
16	2	2	16	6560	6304	0.1470	3.1935	0.05
16	2	2	32	6560	6048	0.3904	13.0296	0.03
16	2	4	8	6560	6432	0.0839	1.3999	0.06
16	2	8	8	6560	6432	0.0732	0.7672	0.10
16	3	2	8	9840	9712	0.1399	1.4513	0.10
16	4	2	8	13120	12992	0.2298	1.3173	0.17
16	5	2	8	16400	16272	0.3364	3.3922	0.10
16	6	2	8	19680	19552	0.4906	3.7047	0.13
24	2	2	8	14448	14256	0.2343	1.2551	0.19
32	2	2	8	25408	25152	0.7070	1.4119	0.50
40	2	2	8	39440	39120	1.6811	0.7474	2.25

P and dim X are the number of rows and columns in the affine space returned respectively.

G NON-ℓ_2 SELECTION OVER THE AFFINE SPACE

A major advantage of returning a space rather than a point is that one can then choose from it by an arbitrary criterion. To demonstrate this, we implemented a method to search the input space of GPT-2 to find an input sequence that produces a target output sequence. This takes the preimage **at the input embeddings** of a pretrained GPT-2 rather than at the weights, and selects from it by linear programming. More specifically: we searched for a point in the preimage that changes only two input tokens to force a desired change in the output tokens. We implemented and evaluated this on Andrej Karpathy's nanoGPT-Shakespeare model (Johnson, 2015; Karpathy, 2022):

1. We want to make the model say "`ther the sta`".
2. Starting with the prompt "`First Ci`" instead gives "`tizen:\nThe s`".
3. We want to change two characters in the prompt so it says "`ther the sta`".
4. Compute the preimage, the whole affine space of prompt perturbations that would produce the wanted text.
5. Search the preimage with both ℓ_1 and ℓ_2 to select two character positions to change.
6. Perform exhaustive search over real characters at their chosen positions to find the changes that yield the closest target.
   - ℓ_1 needs the whole preimage. Edits positions 5 and 6. The best prompt it finds: "`First Oi`". The model then says: "`ther the sta`" — exactly what we wanted.
   - ℓ_2 what CG would give. Edits positions 0 and 5. The best prompt it finds: "`\nirstqCi`". The model then says: "`tion the sta`" — not what we wanted.

ℓ_1 asks for a perturbation concentrated on a small number of token positions (ℓ_0 would give the smallest, but is NP-hard; ℓ_1 is a convex approximation), so its answer identifies candidate characters to change. That question requires a representation of the preimage, because it is a linear program constrained to the preimage. Conjugate gradient returns one point, with minimum-ℓ_2 norm, and that point is spread over every position because ℓ_2 prefers less sparse vectors, so it fails to crisply identify candidate characters for change; the two positions it happens to rank highest are typically the wrong ones.

In appropriate circumstances, as here, the problem space can be huge but the preimage can be small, so a postprocessing search over the preimage can be much faster than a search over the whole problem space. Please keep in mind that this is a proof of concept and not a benchmark, so we did **not** compare it against CG. The point here is that admitting to a selection criterion within an admissible inverse space is possible with Null-A mode preimage AD, but is not really possible with only a single inverse point as would be calculated by CG.

H NUMERICAL STABILITY IN DEEP COMPUTATIONS

Null-A mode preimage AD performs no basis orthogonalization, as that would destroy the sparsity of the representation. So it does not suffer the errors specific to orthogonalization, though it can suffer from near-parallel basis elements.

In short, like in almost all numerical analysis, this is highly problem dependent. We did a detailed analysis of our MLP and GPT-2 benchmarks. A highly batched deep narrow MLP at initialization can lead to a noisy estimate of the origin but not the basis, because of near dependence of the rows of the Jacobian for different samples. But GPT-2 does not suffer from this.

Note that **block primitives are more robust, not just faster.** On our benchmarks, scalar Null-A mode preimage AD makes about 10^5 independent decisions of the form "is this coefficient zero?," and such a decision **cannot** be made correctly in isolation, because a single scalar multiply carries no information about whether its own coefficient is meaningful. A block primitive decides the rank of a whole layer at once, against the largest pivot of that layer's own Jacobian. We have measured this correct out to $\text{cond}(J) = 1.5 \times 10^{13}$.

I PYTHON CODE TO PERFORM TRACING

```
import math
import affine_local as af

epsilon = 0
serial_number = 0

class Vertex:
    def __init__(self, epsilon, primal, partials, parents):
        global serial_number
        serial_number += 1
        self.epsilon = epsilon
        self.primal = primal
        self.parent_partials = partials
        self.parents = parents
        self.serial_number = serial_number
        self.children = []
        self.child_partials = []
        self.tangent_or_cotangent = None
    def __pos__(self): return self
    def __neg__(self): return 0-self
    def __add__(self, y): return plus(self, y)
    def __radd__(self, x): return plus(x, self)
    def __sub__(self, y): return minus(self, y)
    def __rsub__(self, x): return minus(x, self)
    def __mul__(self, y): return times(self, y)
    def __rmul__(self, x): return times(x, self)
    def __div__(self, y): return divide(self, y)
    def __rdiv__(self, x): return divide(x, self)
    def __truediv__(self, y): return divide(self, y)
    def __rtruediv__(self, x): return divide(x, self)
    def __eq__(self, x): return eq(self, x)
    def __ne__(self, x): return ne(self, x)
    def __lt__(self, x): return lt(self, x)
    def __gt__(self, x): return gt(self, x)
    def __le__(self, x): return le(self, x)
    def __ge__(self, x): return ge(self, x)

def earlier(epsilon1, epsilon2):
    return epsilon1<epsilon2

def lift_real_to_real(f, dfdx):
    def me(x):
        if isinstance(x, Vertex):
            return Vertex(x.epsilon,
                          me(x.primal),
                          [dfdx(x.primal)],
                          [x])
        else:
            return f(x)
    return me

def lift_real_cross_real_to_real(f, dfdx1, dfdx2):
    def me(x1, x2):
        if isinstance(x1, Vertex):
            if isinstance(x2, Vertex):
                if x1.epsilon<x2.epsilon:
                    return Vertex(x2.epsilon,
                                  me(x1, x2.primal),
                                  [dfdx2(x1, x2.primal)],
                                  [x2])
                elif x2.epsilon<x1.epsilon:
                    return Vertex(x1.epsilon,
                                  me(x1.primal, x2),
                                  [dfdx1(x1.primal, x2)],
                                  [x1])
                else:
                    return Vertex(x1.epsilon,
                                  me(x1.primal, x2.primal),
                                  [dfdx1(x1.primal, x2.primal),
                                   dfdx2(x1.primal, x2.primal)],
                                  [x1, x2])
```

```
            else:
                return Vertex(x1.epsilon,
                              me(x1.primal, x2),
                              [dfdx1(x1.primal, x2)],
                              [x1])
        else:
            if isinstance(x2, Vertex):
                return Vertex(x2.epsilon,
                              me(x1, x2.primal),
                              [dfdx2(x1, x2.primal)],
                              [x2])
            else:
                return f(x1, x2)
    return me

def lift_real_cross_real_to_boolean(f):
    def me(x1, x2):
        if isinstance(x1, Vertex):
            return me(x1.primal, x2)
        elif isinstance(x2, Vertex):
            return me(x1, x2.primal)
        else:
            return f(x1, x2)
    return me

plus = lift_real_cross_real_to_real(lambda x1, x2: x1+x2,
                                    lambda x1, x2: 1.0,
                                    lambda x1, x2: 1.0)

minus = lift_real_cross_real_to_real(lambda x1, x2: x1-x2,
                                     lambda x1, x2: 1.0,
                                     lambda x1, x2: -1.0)

times = lift_real_cross_real_to_real(lambda x1, x2: x1*x2,
                                     lambda x1, x2: x2,
                                     lambda x1, x2: x1)

divide = lift_real_cross_real_to_real(lambda x1, x2: x1/x2,
                                      lambda x1, x2: 1.0/x2,
                                      lambda x1, x2: -x1/(x2*x2))

sqrt = lift_real_to_real(math.sqrt, lambda x: 1.0/(2.0*sqrt(x)))

exp = lift_real_to_real(math.exp, lambda x: exp(x))

log = lift_real_to_real(math.log, lambda x: 1.0/x)

sin = lift_real_to_real(math.sin, lambda x: cos(x))

cos = lift_real_to_real(math.cos, lambda x: -sin(x))

def atan(x):
    return atan2(x, 1.0)

atan2 = lift_real_cross_real_to_real(math.atan2,
                                     lambda x1, x2: x2/(x1*x1+x2*x2),
                                     lambda x1, x2: -x1/(x1*x1+x2*x2))

tanh = lift_real_to_real(math.tanh, lambda x: 1.0-tanh(x)*tanh(x))

eq = lift_real_cross_real_to_boolean(lambda x1, x2: x1==x2)

ne = lift_real_cross_real_to_boolean(lambda x1, x2: x1!=x2)

lt = lift_real_cross_real_to_boolean(lambda x1, x2: x1<x2)

gt = lift_real_cross_real_to_boolean(lambda x1, x2: x1>x2)

le = lift_real_cross_real_to_boolean(lambda x1, x2: x1<=x2)

ge = lift_real_cross_real_to_boolean(lambda x1, x2: x1>=x2)
```

J PYTHON CODE USED BY ALL MODES

```
def fanout(x):
    return x, x

def memq(vertex, vertices):
    for other in vertices:
        if vertex is other:
            return True
    return False

def all_vertices(y_vertex):
    vertices = []
    def collect(vertex):
        if isinstance(vertex, Vertex) and not memq(vertex, vertices):
            vertices.append(vertex)
            for parent in vertex.parents:
                collect(parent)
    for_walk1(collect, y_vertex)
    return vertices

def topological_sort(vertices):
    vertices.sort(key=lambda x:x.serial_number)
    return vertices

def map_walk1(f, x):
    if isinstance(x, bool):
        return x
    elif isinstance(x, str):
        return x
    elif isinstance(x, int):
        return x
    elif isinstance(x, float) or isinstance(x, Vertex):
        return f(x)
    elif isinstance(x, tuple):
        return tuple((map_walk1(f, xe) for xe in x))
    elif isinstance(x, list):
        return list((map_walk1(f, xe) for xe in x))
    elif isinstance(x, dict):
        return {k: map_walk1(f, xe) for k, xe in x.items()}
    elif isinstance(x, set):
        return set((map_walk1(f, xe) for xe in x))
    else:
        raise RuntimeError("Not walkable: %s"%x)

def for_walk1(f, x):
    if isinstance(x, bool):
        pass
    elif isinstance(x, str):
        pass
    elif isinstance(x, int):
        pass
    elif isinstance(x, float) or isinstance(x, Vertex):
        f(x)
        pass
    elif isinstance(x, tuple):
        for xe in x:
            for_walk1(f, xe)
    elif isinstance(x, list):
        for xe in x:
            for_walk1(f, xe)
    elif isinstance(x, dict):
        for xe in x.values():
            for_walk1(f, xe)
    elif isinstance(x, set):
        for xe in x:
            for_walk1(f, xe)
    else:
        raise RuntimeError("Not walkable: %s"%x)

def for_walk2(f, x, x_prime):
    if isinstance(x, bool) and isinstance(x_prime, bool) and x==x_prime:
        pass
```

```
    elif isinstance(x, str) and isinstance(x_prime, str) and x==x_prime:
        pass
    elif isinstance(x, int) and isinstance(x_prime, int) and x==x_prime:
        pass
    elif ((isinstance(x, float) or isinstance(x, Vertex)) and
          (isinstance(x_prime, float) or isinstance(x_prime, Vertex))):
        f(x, x_prime)
        pass
    elif (isinstance(x, tuple) and
          isinstance(x_prime, tuple) and
          len(x)==len(x_prime)):
        for xe, xe_prime in zip(x, x_prime):
          for_walk2(f, xe, xe_prime)
    elif (isinstance(x, list) and
          isinstance(x_prime, list) and
          len(x)==len(x_prime)):
        for xe, xe_prime in zip(x, x_prime):
          for_walk2(f, xe, xe_prime)
    elif (isinstance(x, dict) and
          isinstance(x_prime, dict) and
          set(x.keys())==set(x_prime.keys())):
        for k in x.keys():
          for_walk2(f, xe[k], xe_prime[k])
    elif (isinstance(x, set) and
          isinstance(x_prime, set) and
          len(x)==len(x_prime)):
          RuntimeError("Not walkable: %s %s"%(x, x_prime))
    else:
        raise RuntimeError("Values don't conform: %s, %s"%(x, x_prime))

def primal(vertex):
    if isinstance(vertex, Vertex) and not earlier(vertex.epsilon, epsilon):
        return vertex.primal
    else:
        return vertex

def tangent_or_cotangent(vertex):
    if isinstance(vertex, Vertex) and not earlier(vertex.epsilon, epsilon):
        if vertex.tangent_or_cotangent is None:
            return 0.0
        else:
            return vertex.tangent_or_cotangent
    else:
        return 0.0

def deaffine_tangent_or_cotangent(vertex):
    if isinstance(vertex, Vertex) and not earlier(vertex.epsilon, epsilon):
        if vertex.tangent_or_cotangent is None:
            raise RuntimeError("Empty preimage")
        else:
            return af.deaffine(vertex.tangent_or_cotangent)
    else:
        raise RuntimeError("Empty preimage")

def set_tangent_or_cotangent(vertex, tangent_or_cotangent):
    if isinstance(vertex, Vertex) and not earlier(vertex.epsilon, epsilon):
        if vertex.tangent_or_cotangent is None:
            vertex.tangent_or_cotangent = tangent_or_cotangent

def set_tangent_or_cotangent_to_affine_slice(vertex, tangent_or_cotangent):
    if isinstance(vertex, Vertex) and not earlier(vertex.epsilon, epsilon):
        if vertex.tangent_or_cotangent is None:
            vertex.tangent_or_cotangent = af.AffineSpaceSlice(
                tangent_or_cotangent, empty_vector)
```

K PYTHON CODE TO PERFORM EDGE REVERSAL

```
def compute_children(vertices):
    for vertex in vertices:
        for parent, partial in zip(vertex.parents, vertex.parent_partials):
            parent.children.append(vertex)
```

```
            parent.child_partials.append(partial)
```

L PYTHON CODE TO PERFORM FORWARD MODE

```
def forward_accumulation_sweep(y_vertex):
    vertices = topological_sort(all_vertices(y_vertex))
    for vertex in vertices:
        # Step 1
        S = []
        for parent in vertex.parents:
            s, parent.tangent_or_cotangent = fanout(
                parent.tangent_or_cotangent)
            S.append(s)
        # Step 2
        T = [s*partial for s, partial in zip(S, vertex.parent_partials)]
        # Step 3
        for t in T:
            if vertex.tangent_or_cotangent is None:
                vertex.tangent_or_cotangent = t
            else:
                vertex.tangent_or_cotangent = vertex.tangent_or_cotangent+t

def jvp(f, x, x_tangent):
    global epsilon
    epsilon += 1
    x_vertex = map_walk1(lambda x: Vertex(epsilon, x, [], []), x)
    for_walk2(set_tangent_or_cotangent, x_vertex, x_tangent)
    y_vertex = f(x_vertex)
    forward_accumulation_sweep(y_vertex)
    result = (map_walk1(primal, y_vertex),
              map_walk1(tangent_or_cotangent, y_vertex))
    epsilon -= 1
    return result
```

M PYTHON CODE TO PERFORM REVERSE MODE

```
def reverse_accumulation_sweep(y_vertex):
    vertices = topological_sort(all_vertices(y_vertex))
    compute_children(vertices)
    for vertex in vertices[::-1]:
        # Step 1
        S = []
        for child in vertex.children:
            s, child.tangent_or_cotangent = fanout(
                child.tangent_or_cotangent)
            S.append(s)
        # Step 2
        T = [s*partial for s, partial in zip(S, vertex.child_partials)]
        # Step 3
        for t in T:
            if vertex.tangent_or_cotangent is None:
                vertex.tangent_or_cotangent = t
            else:
                vertex.tangent_or_cotangent = vertex.tangent_or_cotangent+t

def vjp(f, x, y_cotangent):
    global epsilon
    epsilon += 1
    x_vertex = map_walk1(lambda x: Vertex(epsilon, x, [], []), x)
    y_vertex = f(x_vertex)
    for_walk2(set_tangent_or_cotangent, y_vertex, y_cotangent)
    reverse_accumulation_sweep(y_vertex)
    result = (map_walk1(primal, y_vertex),
              map_walk1(tangent_or_cotangent, x_vertex))
    epsilon -= 1
    return result
```

N PYTHON CODE TO PERFORM REVERSE NULL-A MODE PREIMAGE AD

```
def reverse_null_A_preimage_accumulation_sweep(y_vertex):
    vertices = topological_sort(all_vertices(y_vertex))
    for vertex in vertices[::-1]:
        S = []
        # Preimage of step 3
        for _ in vertex.parents:
            if len(S)==0:
                S.append(vertex.tangent_or_cotangent)
            else:
                X1, X2 = af.preimage_binary_addition(S[-1])
                S = S[:-1]
                S.append(X1)
                S.append(X2)
        # Preimage of step 2
        T = [af.preimage_multiplication_by_constant(s, a)
             for s, a in zip(S, vertex.parent_partials)]
        # Preimage of step 1
        for parent, t in zip(vertex.parents, T):
            if parent.tangent_or_cotangent is None:
                parent.tangent_or_cotangent = t
            else:
                parent.tangent_or_cotangent = af.preimage_binary_fanout(
                    t, parent.tangent_or_cotangent)

def ijvp(f, x, y_tangent):
    global epsilon
    epsilon += 1
    x_vertex = map_walk1(lambda x: Vertex(epsilon, x, [], []), x)
    y_vertex = f(x_vertex)
    for_walk2(set_tangent_or_cotangent_to_affine_slice, y_vertex, y_tangent)
    reverse_null_A_preimage_accumulation_sweep(y_vertex)
    result = (map_walk1(primal, y_vertex),
              map_walk1(deaffine_tangent_or_cotangent, x_vertex))
    epsilon -= 1
    return result
```

O PYTHON CODE TO PERFORM FORWARD NULL-A MODE PREIMAGE AD

```
def forward_null_A_preimage_accumulation_sweep(y_vertex):
    vertices = topological_sort(all_vertices(y_vertex))
    compute_children(vertices)
    for vertex in vertices:
        # Preimage of step 3
        S = []
        for _ in vertex.children:
            if len(S)==0:
                S.append(vertex.tangent_or_cotangent)
            else:
                X1, X2 = af.preimage_binary_addition(S[-1])
                S = S[:-1]
                S.append(X1)
                S.append(X2)
        # Preimage of step 2
        T = [af.preimage_multiplication_by_constant(s, a)
             for s, a in zip(S, vertex.child_partials)]
        # Preimage of step 1
        for child, t in zip(vertex.children, T):
            if child.tangent_or_cotangent is None:
                child.tangent_or_cotangent = t
            else:
                child.tangent_or_cotangent = af.preimage_binary_fanout(
                    t, child.tangent_or_cotangent)

def vijp(f, x, x_cotangent):
    global epsilon
    epsilon += 1
    x_vertex = map_walk1(lambda x: Vertex(epsilon, x, [], []), x)
    for_walk2(set_tangent_or_cotangent_to_affine_slice, x_vertex, x_cotangent)
    y_vertex = f(x_vertex)
```

```
    forward_null_A_preimage_accumulation_sweep(y_vertex)
    result = (map_walk1(primal, y_vertex),
              map_walk1(deaffine_tangent_or_cotangent, y_vertex))
    epsilon -= 1
    return result
```

P PYTHON CODE FOR HESSIAN-VECTOR PRODUCTS AND NEWTON STEPS

```
def hvp1(f, x, x_tangent):
    return jvp(lambda x: vjp(f, x, 1.0)[1], x, x_tangent)[1]

def hvp2(f, x, x_tangent):
    return vjp(lambda x:jvp(f, x, x_tangent)[1], x, 1.0)[1]

def hvp3(f, x, x_tangent):
    return vjp(lambda x: vjp(f, x, 1.0)[1], x, x_tangent)[1]

def ihvp1(f, x, x_tangent):
    return ijvp(lambda x: vjp(f, x, 1.0)[1], x, x_tangent)[1]

def ihvp3(f, x, x_tangent):
    return vijp(lambda x: vjp(f, x, 1.0)[1], x, x_tangent)[1]

def newton_step1(f, x):
    return ihvp1(f, x, vjp(f, x, -1.0)[1])

def newton_step3(f, x):
    return ihvp3(f, x, vjp(f, x, -1.0)[1])
```

Q PYTHON CODE FOR DENSE LINEAR-ALGEBRA COMPUTATIONS

```
empty_vector = []

def dot(u, v):
    return sum([ue*ve for ue, ve in zip(u, v)])

def scalar_product(a, u):
    return [a*ue for ue in u]

def cons_zero(u):
    return [0.0]+u

def cons_one(u):
    return [1.0]+u

def cons_minus_one(u):
    return [-1.0]+u

def e1(u):
    return [1.0]+[0.0]*len(u)

def vector_subtraction(u, v):
    return [ue-ve for ue, ve in zip(u, v)]

def nonzero(x):
    return abs(x)>1e-12

def any_nonzero(u):
    return any([nonzero(ue) for ue in u])

def first_nonzero(u):
    for j, uj in enumerate(u):
        if nonzero(uj):
            return j
    return None

def drop(l, u):
    return [uj for j, uj in enumerate(u) if j!=l]
```

```
def W(factor, b1, b):
    return [sum([bil*((1.0 if l==j else 0.0)-factor*b1l*b1j)
                 for l, (bil, b1l) in enumerate(zip(b, b1))])
            for j, b1j in enumerate(b1)]
```

R PYTHON CODE FOR SPARSE LINEAR-ALGEBRA COMPUTATIONS

```
empty_vector = {}

def dot(u, v):
    return sum([ue*v[key] for key, ue in u.items() if key in v])

def scalar_product(a, u):
    if nonzero(a):
        return {key: a*ue for key, ue in u.items() if nonzero(a*ue)}
    else:
        return {}

def cons_zero(u):
    return {key+1: ue for key, ue in u.items()}

def cons_one(u):
    w = {key+1: ue for key, ue in u.items()}
    w[0] = 1.0
    return w

def cons_minus_one(u):
    w = {key+1: ue for key, ue in u.items()}
    w[0] = -1.0
    return w

def e1(u):
    return {0: 1.0}

def vector_subtraction(u, v):
    keys = set()
    for key in u.keys():
        keys.add(key)
    for key in v.keys():
        keys.add(key)
    return {key:
            u[key]-v[key] if key in u and key in v
            else u[key] if key in u
            else -v[key]
            for key in keys
            if nonzero(u[key]-v[key] if key in u and key in v
                       else u[key] if key in u
                       else -v[key])}

def nonzero(x):
    return abs(x)>1e-12

def any_nonzero(u):
    return any([nonzero(ue) for ue in u.values()])

def first_nonzero(u):
    l = None
    for j, uj in u.items():
        if nonzero(uj) and (l is None or j<l):
            l = j
    return l

def drop(l, u):
    return {j-1 if j>l else j: uj for j, uj in u.items() if j!=l}

def W(factor, b1, b):
    result = {}
    for j, b1j in b1.items():
        entry = 0.0
        for l, b1l in b1.items():
            if l in b:
```

```
                entry += b[l]*((1.0 if l==j else 0.0)-factor*b1l*b1j)
        result[j] = entry
    for j, bj in b.items():
        if j not in result:
            result[j] = bj
    return {key: value for key, value in result.items() if nonzero(value)}
```

S PYTHON CODE FOR COMPUTING PREIMAGES OF AFFINE SPACE SLICES UNDER THE SPECIALIZED LINEAR MAPS

```
delayed_operations = []
if options.dense:
    from dense_vector import *
else:
    from sparse_vector import *

class AffineSpaceSlice:
    def __init__(self, origin, basis):
        self.timestamp = len(delayed_operations)
        self.origin = origin
        self.basis = basis
        if options.print:
            print(basis)

class GrowAffineSpaceConstrained:
    def __init__(self, origin, basis):
        self.origin = origin
        self.basis = basis
    def apply(self, origin, basis):
        factor = 1.0/dot(self.basis, self.basis)
        origin -= self.origin*factor*dot(basis, self.basis)
        basis = cons_zero(W(factor, self.basis, basis))
        return origin, basis

class GrowAffineSpaceUnconstrained:
    def apply(self, origin, basis):
        return origin, cons_zero(basis)

class ShrinkAffineSpace:
    def __init__(self, d, r, t):
        self.d = d
        self.r = r
        self.t = t
    def apply(self, origin, basis):
        origin += dot(basis, self.t)
        if options.dense:
            bd = basis[self.d]
            rd = self.r[self.d]
        else:
            bd = basis[self.d] if self.d in basis else 0.0
            rd = self.r[self.d] if self.d in self.r else 0.0
        basis = drop(
            self.d, vector_subtraction(basis, scalar_product(bd/rd, self.r)))
        return origin, basis

def apply_delayed_operations(Y):
    origin = Y.origin
    basis = Y.basis
    for operation in delayed_operations[Y.timestamp:]:
        origin, basis = operation.apply(origin, basis)
    return AffineSpaceSlice(origin, basis)

def preimage_multiplication_by_constant(Y, a):
    Y = apply_delayed_operations(Y)
    # case 1
    if nonzero(a):
        return AffineSpaceSlice(Y.origin/a, scalar_product(1.0/a, Y.basis))
    # case 2
    elif any_nonzero(Y.basis):
        delayed_operations.append(GrowAffineSpaceConstrained(Y.origin, Y.basis))
```

```
        return AffineSpaceSlice(0.0, e1(Y.basis))
    # case 3
    elif nonzero(Y.origin):
        raise RuntimeError("Empty preimage")
    # case 4
    else:
        delayed_operations.append(GrowAffineSpaceUnconstrained())
        return AffineSpaceSlice(Y.origin, cons_one(Y.basis))

def preimage_binary_addition(Y):
    Y = apply_delayed_operations(Y)
    delayed_operations.append(GrowAffineSpaceUnconstrained())
    return (AffineSpaceSlice(Y.origin, cons_minus_one(Y.basis)),
            AffineSpaceSlice(0.0, e1(Y.basis)))

def preimage_binary_fanout(Y1, Y2):
    Y1 = apply_delayed_operations(Y1)
    Y2 = apply_delayed_operations(Y2)
    r = vector_subtraction(Y1.basis, Y2.basis)
    d = first_nonzero(r)
    # case 1
    if d is not None:
        t = scalar_product((Y2.origin-Y1.origin)/dot(r, r), r)
        delayed_operations.append(ShrinkAffineSpace(d, r, t))
        if options.dense:
            bd = Y1.basis[d]
            rd = r[d]
        else:
            bd = Y1.basis[d] if d in Y1.basis else 0.0
            rd = r[d] if d in r else 0.0
        return AffineSpaceSlice(
            Y1.origin+dot(Y1.basis, t),
            drop(d, vector_subtraction(Y1.basis, scalar_product(bd/rd, r))))
    # case 2
    elif nonzero(Y1.origin-Y2.origin):
        raise RuntimeError("Empty preimage")
    # case 3
    else:
        return Y1

def deaffine(Y):
    Y = apply_delayed_operations(Y)
    if any_nonzero(Y.basis):
        raise RuntimeError("Preimage is not a unique point")
    return Y.origin
```

T PYTHON CODE FOR A BENCHMARK TO TEST THE ASYMPTOTIC COMPLEXITY SCALING OF §5

```
from graph_AD import *
import functools
import time
import sys

def f(x):
    global calls_to_f
    calls_to_f += 1
    return 1.0*x

def g(x, y):
    global calls_to_g
    calls_to_g += 1
    return 1.0*x+0.0*y

def h(x, y):
    global calls_to_h
    calls_to_h += 1
    return 1.0*x+0.0*y

def p(t, k, v):
```

```
    # With the current f, g, and h, this is the identity function on v. This is
    # carefully contrived so that the Jacobian is the identity matrix and thus
    # trivially invertible.
    # This runs the whole thing t times.
    for _ in range(1, t+1):                # O(t(n+k+kn))=O(tkn)
        d = functools.reduce(g, map(f, v)) # O(n)
        # This creates swell of k by making k copies of q(v).
        s = [d for _ in range(k)] # O(k)
        for i in range(k):                          # O(kn)
            v = list(map(lambda x: h(x, s[i]), v)) # O(n)
    return v

def e(i, n):
    v = [0.0 for _ in range(n)]
    v[i] = 1.0
    return v

def explicit_ijvp(f, x, y_tangent):
    n = len(x)
    J = np.array([jvp(f, x, e(i, n))[1] for i in range(n)])
    return list(np.linalg.inv(J)@np.array(y_tangent))

if __name__ == "__main__":
    sys.setrecursionlimit(100000)
    # t is proportional to the number of steps
    # k is proportional to the swell
    # n is the input/output dimension
    #\needswork: I don't know how to measure space.
    global calls_to_f, calls_to_g, calls_to_h
    for t in range(1, 10+1):
        for k in range(1, 10+1):
            def p_tk(v):
                return p(3*t, 3*k, v)
            for n in range(1, 10+1):
                v = list([float(q) for q in range(1, 3*n+1)])
                calls_to_f = 0
                calls_to_g = 0
                calls_to_h = 0
                start_time = time.perf_counter()
                ours = ijvp(p_tk, v, v)[1]
                end_time = time.perf_counter()
                our_elapsed_time = end_time-start_time
                our_calls_to_f = calls_to_f
                our_calls_to_g = calls_to_g
                our_calls_to_h = calls_to_h
                calls_to_f = 0
                calls_to_g = 0
                calls_to_h = 0
                start_time = time.perf_counter()
                theirs = explicit_ijvp(p_tk, v, v)
                end_time = time.perf_counter()
                their_elapsed_time = end_time-start_time
                their_calls_to_f = calls_to_f
                their_calls_to_g = calls_to_g
                their_calls_to_h = calls_to_h
                ours = np.array(ours).flatten()
                theirs = np.array(theirs).flatten()
                norm = np.linalg.norm(ours-theirs)
                if our_calls_to_f!=3*t*3*n:
                    raise RuntimeError("Wrong number of our calls to f")
                if our_calls_to_g!=3*t*(3*n-1):
                    raise RuntimeError("Wrong number of our calls to g")
                if our_calls_to_h!=3*t*3*k*3*n:
                    raise RuntimeError("Wrong number of our calls to g")
                our_calls = our_calls_to_f+our_calls_to_g+our_calls_to_h
                their_calls = their_calls_to_f+their_calls_to_g+their_calls_to_h
                if our_calls!=3*t*((3*k+2)*3*n-1):
                    raise RuntimeError("Wrong number of our calls")
                if 3*n*our_calls!=their_calls:
                    raise RuntimeError("Wrong number of their calls")
                if norm>0.0:
                    raise RuntimeError("Nonzero norm")
                print("t: %d, k: %d, n: %d,"%(t, k, n))
```

```
print("our calls: %d, their calls: %d,"%(
    our_calls, their_calls))
print("our time: %f, their time: %f"%(
    our_elapsed_time, their_elapsed_time))
```

U COMPUTATION GRAPH FOR THE BENCHMARK IN §T

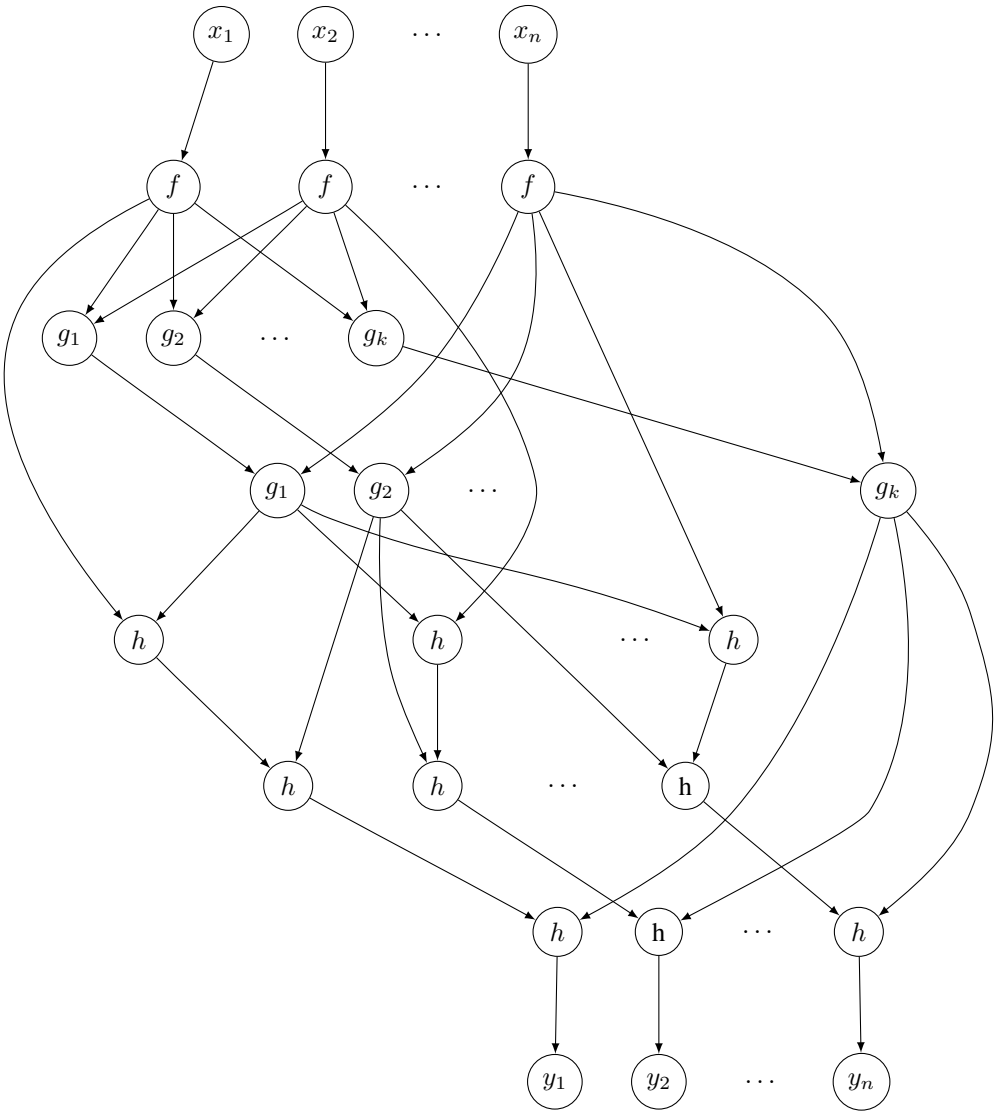


Figure 8: The primal floating-point computation graph corresponding to the program in §T.

V OUTPUT OF THE BENCHMARK THAT DEMONSTRATES THE ASYMPTOTIC COMPLEXITY SCALING OF §5

```
>python3 benchmark.py
t: 1, k: 1, n: 1,
our calls:: 42, their calls: 126,
our time: 0.000987, their time: 0.000603
t: 1, k: 1, n: 2,
our calls:: 87, their calls: 522,
our time: 0.001607, their time: 0.002902
t: 1, k: 1, n: 3,
our calls:: 132, their calls: 1188,
our time: 0.003028, their time: 0.007870
t: 1, k: 1, n: 4,
our calls:: 177, their calls: 2124,
our time: 0.004828, their time: 0.017204
t: 1, k: 1, n: 5,
our calls:: 222, their calls: 3330,
our time: 0.010013, their time: 0.030910
t: 1, k: 1, n: 6,
our calls:: 267, their calls: 4806,
our time: 0.009628, their time: 0.053409
t: 1, k: 1, n: 7,
our calls:: 312, their calls: 6552,
our time: 0.012778, their time: 0.080154
t: 1, k: 1, n: 8,
our calls:: 357, their calls: 8568,
our time: 0.016326, their time: 0.115089
t: 1, k: 1, n: 9,
our calls:: 402, their calls: 10854,
our time: 0.019828, their time: 0.162227
t: 1, k: 1, n: 10,
our calls:: 447, their calls: 13410,
our time: 0.023910, their time: 0.219824
t: 1, k: 2, n: 1,
our calls:: 69, their calls: 207,
our time: 0.001134, their time: 0.001063
t: 1, k: 2, n: 2,
our calls:: 141, their calls: 846,
our time: 0.003340, their time: 0.006251
t: 1, k: 2, n: 3,
our calls:: 213, their calls: 1917,
our time: 0.005817, their time: 0.018754
t: 1, k: 2, n: 4,
our calls:: 285, their calls: 3420,
our time: 0.009274, their time: 0.044581
t: 1, k: 2, n: 5,
our calls:: 357, their calls: 5355,
our time: 0.013717, their time: 0.079979
t: 1, k: 2, n: 6,
our calls:: 429, their calls: 7722,
our time: 0.018908, their time: 0.130709
t: 1, k: 2, n: 7,
our calls:: 501, their calls: 10521,
our time: 0.024791, their time: 0.205447
t: 1, k: 2, n: 8,
our calls:: 573, their calls: 13752,
our time: 0.031545, their time: 0.296166
t: 1, k: 2, n: 9,
our calls:: 645, their calls: 17415,
our time: 0.040299, their time: 0.417423
t: 1, k: 2, n: 10,
our calls:: 717, their calls: 21510,
our time: 0.057991, their time: 0.603951
t: 1, k: 3, n: 1,
our calls:: 96, their calls: 288,
our time: 0.002082, their time: 0.008272
t: 1, k: 3, n: 2,
our calls:: 195, their calls: 1170,
our time: 0.006119, their time: 0.014953
t: 1, k: 3, n: 3,
our calls:: 294, their calls: 2646,
```

```
our time: 0.012296, their time: 0.052787
t: 1, k: 3, n: 4,
our calls:: 393, their calls: 4716,
our time: 0.020838, their time: 0.116873
t: 1, k: 3, n: 5,
our calls:: 492, their calls: 7380,
our time: 0.045765, their time: 0.241909
t: 1, k: 3, n: 6,
our calls:: 591, their calls: 10638,
our time: 0.051881, their time: 0.369099
t: 1, k: 3, n: 7,
our calls:: 690, their calls: 14490,
our time: 0.057873, their time: 0.470744
t: 1, k: 3, n: 8,
our calls:: 789, their calls: 18936,
our time: 0.056810, their time: 0.613940
t: 1, k: 3, n: 9,
our calls:: 888, their calls: 23976,
our time: 0.063767, their time: 0.845461
t: 1, k: 3, n: 10,
our calls:: 987, their calls: 29610,
our time: 0.121005, their time: 1.626909
t: 1, k: 4, n: 1,
our calls:: 123, their calls: 369,
our time: 0.002848, their time: 0.003810
t: 1, k: 4, n: 2,
our calls:: 249, their calls: 1494,
our time: 0.007902, their time: 0.020637
t: 1, k: 4, n: 3,
our calls:: 375, their calls: 3375,
our time: 0.015054, their time: 0.068399
t: 1, k: 4, n: 4,
our calls:: 501, their calls: 6012,
our time: 0.024244, their time: 0.141905
t: 1, k: 4, n: 5,
our calls:: 627, their calls: 9405,
our time: 0.034468, their time: 0.252835
t: 1, k: 4, n: 6,
our calls:: 753, their calls: 13554,
our time: 0.045136, their time: 0.409233
t: 1, k: 4, n: 7,
our calls:: 879, their calls: 18459,
our time: 0.067311, their time: 0.640968
t: 1, k: 4, n: 8,
our calls:: 1005, their calls: 24120,
our time: 0.083671, their time: 1.361368
t: 1, k: 4, n: 9,
our calls:: 1131, their calls: 30537,
our time: 0.136684, their time: 1.417386
t: 1, k: 4, n: 10,
our calls:: 1257, their calls: 37710,
our time: 0.119850, their time: 2.268886
t: 1, k: 5, n: 1,
our calls:: 150, their calls: 450,
our time: 0.004695, their time: 0.004375
t: 1, k: 5, n: 2,
our calls:: 303, their calls: 1818,
our time: 0.009769, their time: 0.028481
t: 1, k: 5, n: 3,
our calls:: 456, their calls: 4104,
our time: 0.020115, their time: 0.096102
t: 1, k: 5, n: 4,
our calls:: 609, their calls: 7308,
our time: 0.031998, their time: 0.201027
t: 1, k: 5, n: 5,
our calls:: 762, their calls: 11430,
our time: 0.043837, their time: 0.358442
t: 1, k: 5, n: 6,
our calls:: 915, their calls: 16470,
our time: 0.059932, their time: 0.592979
t: 1, k: 5, n: 7,
our calls:: 1068, their calls: 22428,
our time: 0.078114, their time: 1.238304
```

```
t: 1, k: 5, n: 8,
our calls:: 1221, their calls: 29304,
our time: 0.168741, their time: 1.732536
t: 1, k: 5, n: 9,
our calls:: 1374, their calls: 37098,
our time: 0.138762, their time: 2.360185
t: 1, k: 5, n: 10,
our calls:: 1527, their calls: 45810,
our time: 0.172509, their time: 3.148620
t: 1, k: 6, n: 1,
our calls:: 177, their calls: 531,
our time: 0.004425, their time: 0.006746
t: 1, k: 6, n: 2,
our calls:: 357, their calls: 2142,
our time: 0.012397, their time: 0.049691
t: 1, k: 6, n: 3,
our calls:: 537, their calls: 4833,
our time: 0.024452, their time: 0.129247
t: 1, k: 6, n: 4,
our calls:: 717, their calls: 8604,
our time: 0.038906, their time: 0.268859
t: 1, k: 6, n: 5,
our calls:: 897, their calls: 13455,
our time: 0.056752, their time: 0.491739
t: 1, k: 6, n: 6,
our calls:: 1077, their calls: 19386,
our time: 0.091312, their time: 0.985925
t: 1, k: 6, n: 7,
our calls:: 1257, their calls: 26397,
our time: 0.149851, their time: 1.667958
t: 1, k: 6, n: 8,
our calls:: 1437, their calls: 34488,
our time: 0.146600, their time: 2.299067
t: 1, k: 6, n: 9,
our calls:: 1617, their calls: 43659,
our time: 0.211282, their time: 3.217775
t: 1, k: 6, n: 10,
our calls:: 1797, their calls: 53910,
our time: 0.311538, their time: 4.449789
t: 1, k: 7, n: 1,
our calls:: 204, their calls: 612,
our time: 0.005710, their time: 0.006930
t: 1, k: 7, n: 2,
our calls:: 411, their calls: 2466,
our time: 0.014236, their time: 0.060648
t: 1, k: 7, n: 3,
our calls:: 618, their calls: 5562,
our time: 0.028384, their time: 0.157154
t: 1, k: 7, n: 4,
our calls:: 825, their calls: 9900,
our time: 0.046864, their time: 0.336959
t: 1, k: 7, n: 5,
our calls:: 1032, their calls: 15480,
our time: 0.087161, their time: 0.788201
t: 1, k: 7, n: 6,
our calls:: 1239, their calls: 22302,
our time: 0.152575, their time: 1.446643
t: 1, k: 7, n: 7,
our calls:: 1446, their calls: 30366,
our time: 0.131520, their time: 1.923188
t: 1, k: 7, n: 8,
our calls:: 1653, their calls: 39672,
our time: 0.262687, their time: 2.851942
t: 1, k: 7, n: 9,
our calls:: 1860, their calls: 50220,
our time: 0.274789, their time: 4.287846
t: 1, k: 7, n: 10,
our calls:: 2067, their calls: 62010,
our time: 0.379850, their time: 5.440876
t: 1, k: 8, n: 1,
our calls:: 231, their calls: 693,
our time: 0.007283, their time: 0.008262
t: 1, k: 8, n: 2,
```

```
our calls:: 465, their calls: 2790,
our time: 0.044150, their time: 0.055957
t: 1, k: 8, n: 3,
our calls:: 699, their calls: 6291,
our time: 0.035063, their time: 0.234202
t: 1, k: 8, n: 4,
our calls:: 933, their calls: 11196,
our time: 0.085412, their time: 0.623296
t: 1, k: 8, n: 5,
our calls:: 1167, their calls: 17505,
our time: 0.165603, their time: 1.045741
t: 1, k: 8, n: 6,
our calls:: 1401, their calls: 25218,
our time: 0.146589, their time: 1.509887
t: 1, k: 8, n: 7,
our calls:: 1635, their calls: 34335,
our time: 0.220816, their time: 2.601891
t: 1, k: 8, n: 8,
our calls:: 1869, their calls: 44856,
our time: 0.230621, their time: 3.857490
t: 1, k: 8, n: 9,
our calls:: 2103, their calls: 56781,
our time: 0.393761, their time: 5.460730
t: 1, k: 8, n: 10,
our calls:: 2337, their calls: 70110,
our time: 0.337616, their time: 7.376869
t: 1, k: 9, n: 1,
our calls:: 258, their calls: 774,
our time: 0.011172, their time: 0.010455
t: 1, k: 9, n: 2,
our calls:: 519, their calls: 3114,
our time: 0.021224, their time: 0.117790
t: 1, k: 9, n: 3,
our calls:: 780, their calls: 7020,
our time: 0.043980, their time: 0.322300
t: 1, k: 9, n: 4,
our calls:: 1041, their calls: 12492,
our time: 0.137303, their time: 0.855915
t: 1, k: 9, n: 5,
our calls:: 1302, their calls: 19530,
our time: 0.145234, their time: 1.178888
t: 1, k: 9, n: 6,
our calls:: 1563, their calls: 28134,
our time: 0.144630, their time: 2.231471
t: 1, k: 9, n: 7,
our calls:: 1824, their calls: 38304,
our time: 0.249266, their time: 3.251581
t: 1, k: 9, n: 8,
our calls:: 2085, their calls: 50040,
our time: 0.367270, their time: 4.650294
t: 1, k: 9, n: 9,
our calls:: 2346, their calls: 63342,
our time: 0.300846, their time: 6.736258
t: 1, k: 9, n: 10,
our calls:: 2607, their calls: 78210,
our time: 0.426181, their time: 9.418243
t: 1, k: 10, n: 1,
our calls:: 285, their calls: 855,
our time: 0.009367, their time: 0.048656
t: 1, k: 10, n: 2,
our calls:: 573, their calls: 3438,
our time: 0.024446, their time: 0.087124
t: 1, k: 10, n: 3,
our calls:: 861, their calls: 7749,
our time: 0.047516, their time: 0.339818
t: 1, k: 10, n: 4,
our calls:: 1149, their calls: 13788,
our time: 0.079403, their time: 0.692823
t: 1, k: 10, n: 5,
our calls:: 1437, their calls: 21555,
our time: 0.164032, their time: 1.774980
t: 1, k: 10, n: 6,
our calls:: 1725, their calls: 31050,
```

```
our time: 0.187960, their time: 2.465354
t: 1, k: 10, n: 7,
our calls:: 2013, their calls: 42273,
our time: 0.387567, their time: 3.978157
t: 1, k: 10, n: 8,
our calls:: 2301, their calls: 55224,
our time: 0.355126, their time: 5.915277
t: 1, k: 10, n: 9,
our calls:: 2589, their calls: 69903,
our time: 0.537016, their time: 8.372925
t: 1, k: 10, n: 10,
our calls:: 2877, their calls: 86310,
our time: 0.513479, their time: 11.639896
t: 2, k: 1, n: 1,
our calls:: 84, their calls: 252,
our time: 0.002032, their time: 0.001950
t: 2, k: 1, n: 2,
our calls:: 174, their calls: 1044,
our time: 0.008317, their time: 0.012442
t: 2, k: 1, n: 3,
our calls:: 264, their calls: 2376,
our time: 0.010475, their time: 0.081122
t: 2, k: 1, n: 4,
our calls:: 354, their calls: 4248,
our time: 0.017960, their time: 0.088352
t: 2, k: 1, n: 5,
our calls:: 444, their calls: 6660,
our time: 0.026928, their time: 0.172113
t: 2, k: 1, n: 6,
our calls:: 534, their calls: 9612,
our time: 0.096698, their time: 0.287824
t: 2, k: 1, n: 7,
our calls:: 624, their calls: 13104,
our time: 0.042214, their time: 0.439973
t: 2, k: 1, n: 8,
our calls:: 714, their calls: 17136,
our time: 0.044338, their time: 0.506560
t: 2, k: 1, n: 9,
our calls:: 804, their calls: 21708,
our time: 0.098042, their time: 0.721823
t: 2, k: 1, n: 10,
our calls:: 894, their calls: 26820,
our time: 0.063284, their time: 1.200413
t: 2, k: 2, n: 1,
our calls:: 138, their calls: 414,
our time: 0.005706, their time: 0.005753
t: 2, k: 2, n: 2,
our calls:: 282, their calls: 1692,
our time: 0.012017, their time: 0.034855
t: 2, k: 2, n: 3,
our calls:: 426, their calls: 3834,
our time: 0.024525, their time: 0.107876
t: 2, k: 2, n: 4,
our calls:: 570, their calls: 6840,
our time: 0.041776, their time: 0.290553
t: 2, k: 2, n: 5,
our calls:: 714, their calls: 10710,
our time: 0.051404, their time: 0.419855
t: 2, k: 2, n: 6,
our calls:: 858, their calls: 15444,
our time: 0.104063, their time: 0.560381
t: 2, k: 2, n: 7,
our calls:: 1002, their calls: 21042,
our time: 0.129178, their time: 0.997301
t: 2, k: 2, n: 8,
our calls:: 1146, their calls: 27504,
our time: 0.129085, their time: 1.834973
t: 2, k: 2, n: 9,
our calls:: 1290, their calls: 34830,
our time: 0.117354, their time: 2.049413
t: 2, k: 2, n: 10,
our calls:: 1434, their calls: 43020,
our time: 0.263664, their time: 2.872287
```

```
t: 2, k: 3, n: 1,
our calls:: 192, their calls: 576,
our time: 0.012682, their time: 0.008859
t: 2, k: 3, n: 2,
our calls:: 390, their calls: 2340,
our time: 0.018470, their time: 0.141220
t: 2, k: 3, n: 3,
our calls:: 588, their calls: 5292,
our time: 0.035144, their time: 0.178186
t: 2, k: 3, n: 4,
our calls:: 786, their calls: 9432,
our time: 0.060093, their time: 0.522083
t: 2, k: 3, n: 5,
our calls:: 984, their calls: 14760,
our time: 0.103271, their time: 0.809704
t: 2, k: 3, n: 6,
our calls:: 1182, their calls: 21276,
our time: 0.092697, their time: 1.155598
t: 2, k: 3, n: 7,
our calls:: 1380, their calls: 28980,
our time: 0.124964, their time: 2.269630
t: 2, k: 3, n: 8,
our calls:: 1578, their calls: 37872,
our time: 0.158786, their time: 2.976017
t: 2, k: 3, n: 9,
our calls:: 1776, their calls: 47952,
our time: 0.200596, their time: 4.169812
t: 2, k: 3, n: 10,
our calls:: 1974, their calls: 59220,
our time: 0.305141, their time: 5.934330
t: 2, k: 4, n: 1,
our calls:: 246, their calls: 738,
our time: 0.008451, their time: 0.068513
t: 2, k: 4, n: 2,
our calls:: 498, their calls: 2988,
our time: 0.022217, their time: 0.074844
t: 2, k: 4, n: 3,
our calls:: 750, their calls: 6750,
our time: 0.042804, their time: 0.284399
t: 2, k: 4, n: 4,
our calls:: 1002, their calls: 12024,
our time: 0.065854, their time: 0.560260
t: 2, k: 4, n: 5,
our calls:: 1254, their calls: 18810,
our time: 0.098629, their time: 1.257081
t: 2, k: 4, n: 6,
our calls:: 1506, their calls: 27108,
our time: 0.187250, their time: 2.160462
t: 2, k: 4, n: 7,
our calls:: 1758, their calls: 36918,
our time: 0.253346, their time: 3.401246
t: 2, k: 4, n: 8,
our calls:: 2010, their calls: 48240,
our time: 0.234081, their time: 4.943105
t: 2, k: 4, n: 9,
our calls:: 2262, their calls: 61074,
our time: 0.427588, their time: 7.411081
t: 2, k: 4, n: 10,
our calls:: 2514, their calls: 75420,
our time: 0.618255, their time: 9.499169
t: 2, k: 5, n: 1,
our calls:: 300, their calls: 900,
our time: 0.013831, their time: 0.019584
t: 2, k: 5, n: 2,
our calls:: 606, their calls: 3636,
our time: 0.037242, their time: 0.146879
t: 2, k: 5, n: 3,
our calls:: 912, their calls: 8208,
our time: 0.095162, their time: 0.564202
t: 2, k: 5, n: 4,
our calls:: 1218, their calls: 14616,
our time: 0.117843, their time: 1.021419
t: 2, k: 5, n: 5,
```

```
our calls:: 1524, their calls: 22860,
our time: 0.142647, their time: 2.056770
t: 2, k: 5, n: 6,
our calls:: 1830, their calls: 32940,
our time: 0.295729, their time: 3.407015
t: 2, k: 5, n: 7,
our calls:: 2136, their calls: 44856,
our time: 0.387944, their time: 5.089032
t: 2, k: 5, n: 8,
our calls:: 2442, their calls: 58608,
our time: 0.445873, their time: 7.679123
t: 2, k: 5, n: 9,
our calls:: 2748, their calls: 74196,
our time: 0.488325, their time: 10.913183
t: 2, k: 5, n: 10,
our calls:: 3054, their calls: 91620,
our time: 0.820959, their time: 14.297458
t: 2, k: 6, n: 1,
our calls:: 354, their calls: 1062,
our time: 0.015172, their time: 0.022049
t: 2, k: 6, n: 2,
our calls:: 714, their calls: 4284,
our time: 0.042476, their time: 0.187470
t: 2, k: 6, n: 3,
our calls:: 1074, their calls: 9666,
our time: 0.210651, their time: 0.739627
t: 2, k: 6, n: 4,
our calls:: 1434, their calls: 17208,
our time: 0.188870, their time: 1.374808
t: 2, k: 6, n: 5,
our calls:: 1794, their calls: 26910,
our time: 0.188854, their time: 2.837746
t: 2, k: 6, n: 6,
our calls:: 2154, their calls: 38772,
our time: 0.280244, their time: 4.540591
t: 2, k: 6, n: 7,
our calls:: 2514, their calls: 52794,
our time: 0.375588, their time: 7.435384
t: 2, k: 6, n: 8,
our calls:: 2874, their calls: 68976,
our time: 0.573464, their time: 11.024536
t: 2, k: 6, n: 9,
our calls:: 3234, their calls: 87318,
our time: 0.719267, their time: 14.995342
t: 2, k: 6, n: 10,
our calls:: 3594, their calls: 107820,
our time: 1.157340, their time: 19.188589
t: 2, k: 7, n: 1,
our calls:: 408, their calls: 1224,
our time: 0.016499, their time: 0.024489
t: 2, k: 7, n: 2,
our calls:: 822, their calls: 4932,
our time: 0.041774, their time: 0.173721
t: 2, k: 7, n: 3,
our calls:: 1236, their calls: 11124,
our time: 0.086252, their time: 0.674602
t: 2, k: 7, n: 4,
our calls:: 1650, their calls: 19800,
our time: 0.273944, their time: 1.909490
t: 2, k: 7, n: 5,
our calls:: 2064, their calls: 30960,
our time: 0.331421, their time: 3.389742
t: 2, k: 7, n: 6,
our calls:: 2478, their calls: 44604,
our time: 0.322916, their time: 5.330986
t: 2, k: 7, n: 7,
our calls:: 2892, their calls: 60732,
our time: 0.737713, their time: 8.650758
t: 2, k: 7, n: 8,
our calls:: 3306, their calls: 79344,
our time: 0.533667, their time: 12.903945
t: 2, k: 7, n: 9,
our calls:: 3720, their calls: 100440,
```

```
our time: 0.699279, their time: 17.870541
t: 2, k: 7, n: 10,
our calls:: 4134, their calls: 124020,
our time: 0.962298, their time: 24.627865
t: 2, k: 8, n: 1,
our calls:: 462, their calls: 1386,
our time: 0.016043, their time: 0.030261
t: 2, k: 8, n: 2,
our calls:: 930, their calls: 5580,
our time: 0.053395, their time: 0.353471
t: 2, k: 8, n: 3,
our calls:: 1398, their calls: 12582,
our time: 0.151025, their time: 1.167198
t: 2, k: 8, n: 4,
our calls:: 1866, their calls: 22392,
our time: 0.238039, their time: 2.012933
t: 2, k: 8, n: 5,
our calls:: 2334, their calls: 35010,
our time: 0.378655, their time: 4.433983
t: 2, k: 8, n: 6,
our calls:: 2802, their calls: 50436,
our time: 0.639304, their time: 7.146409
t: 2, k: 8, n: 7,
our calls:: 3270, their calls: 68670,
our time: 0.568219, their time: 10.919288
t: 2, k: 8, n: 8,
our calls:: 3738, their calls: 89712,
our time: 0.850159, their time: 17.742184
t: 2, k: 8, n: 9,
our calls:: 4206, their calls: 113562,
our time: 1.219247, their time: 25.585861
t: 2, k: 8, n: 10,
our calls:: 4674, their calls: 140220,
our time: 1.784013, their time: 34.137717
t: 2, k: 9, n: 1,
our calls:: 516, their calls: 1548,
our time: 0.019425, their time: 0.037129
t: 2, k: 9, n: 2,
our calls:: 1038, their calls: 6228,
our time: 0.061381, their time: 0.451891
t: 2, k: 9, n: 3,
our calls:: 1560, their calls: 14040,
our time: 0.128912, their time: 1.265761
t: 2, k: 9, n: 4,
our calls:: 2082, their calls: 24984,
our time: 0.349628, their time: 2.498206
t: 2, k: 9, n: 5,
our calls:: 2604, their calls: 39060,
our time: 0.463261, their time: 5.279636
t: 2, k: 9, n: 6,
our calls:: 3126, their calls: 56268,
our time: 0.633205, their time: 8.941855
t: 2, k: 9, n: 7,
our calls:: 3648, their calls: 76608,
our time: 0.757537, their time: 13.765089
t: 2, k: 9, n: 8,
our calls:: 4170, their calls: 100080,
our time: 0.998688, their time: 21.631000
t: 2, k: 9, n: 9,
our calls:: 4692, their calls: 126684,
our time: 1.404674, their time: 31.899674
t: 2, k: 9, n: 10,
our calls:: 5214, their calls: 156420,
our time: 1.588641, their time: 43.682313
t: 2, k: 10, n: 1,
our calls:: 570, their calls: 1710,
our time: 0.034686, their time: 0.062638
t: 2, k: 10, n: 2,
our calls:: 1146, their calls: 6876,
our time: 0.096622, their time: 0.551569
t: 2, k: 10, n: 3,
our calls:: 1722, their calls: 15498,
our time: 0.163928, their time: 1.260758
```

```
t: 2, k: 10, n: 4,
our calls:: 2298, their calls: 27576,
our time: 0.333297, their time: 3.276127
t: 2, k: 10, n: 5,
our calls:: 2874, their calls: 43110,
our time: 0.701094, their time: 6.214402
t: 2, k: 10, n: 6,
our calls:: 3450, their calls: 62100,
our time: 0.624528, their time: 10.684371
t: 2, k: 10, n: 7,
our calls:: 4026, their calls: 84546,
our time: 1.124505, their time: 16.398028
t: 2, k: 10, n: 8,
our calls:: 4602, their calls: 110448,
our time: 1.422556, their time: 24.329206
t: 2, k: 10, n: 9,
our calls:: 5178, their calls: 139806,
our time: 1.370927, their time: 35.056656
t: 2, k: 10, n: 10,
our calls:: 5754, their calls: 172620,
our time: 1.946500, their time: 47.776138
t: 3, k: 1, n: 1,
our calls:: 126, their calls: 378,
our time: 0.002975, their time: 0.009143
t: 3, k: 1, n: 2,
our calls:: 261, their calls: 1566,
our time: 0.007819, their time: 0.020991
t: 3, k: 1, n: 3,
our calls:: 396, their calls: 3564,
our time: 0.015053, their time: 0.065793
t: 3, k: 1, n: 4,
our calls:: 531, their calls: 6372,
our time: 0.026275, their time: 0.151844
t: 3, k: 1, n: 5,
our calls:: 666, their calls: 9990,
our time: 0.037266, their time: 0.458146
t: 3, k: 1, n: 6,
our calls:: 801, their calls: 14418,
our time: 0.049214, their time: 0.444222
t: 3, k: 1, n: 7,
our calls:: 936, their calls: 19656,
our time: 0.062214, their time: 1.065344
t: 3, k: 1, n: 8,
our calls:: 1071, their calls: 25704,
our time: 0.117116, their time: 1.527922
t: 3, k: 1, n: 9,
our calls:: 1206, their calls: 32562,
our time: 0.110658, their time: 2.030833
t: 3, k: 1, n: 10,
our calls:: 1341, their calls: 40230,
our time: 0.178496, their time: 2.765474
t: 3, k: 2, n: 1,
our calls:: 207, their calls: 621,
our time: 0.005734, their time: 0.007605
t: 3, k: 2, n: 2,
our calls:: 423, their calls: 2538,
our time: 0.016465, their time: 0.052828
t: 3, k: 2, n: 3,
our calls:: 639, their calls: 5751,
our time: 0.035121, their time: 0.222009
t: 3, k: 2, n: 4,
our calls:: 855, their calls: 10260,
our time: 0.082187, their time: 0.685796
t: 3, k: 2, n: 5,
our calls:: 1071, their calls: 16065,
our time: 0.113333, their time: 1.147355
t: 3, k: 2, n: 6,
our calls:: 1287, their calls: 23166,
our time: 0.114540, their time: 1.540987
t: 3, k: 2, n: 7,
our calls:: 1503, their calls: 31563,
our time: 0.198513, their time: 2.755059
t: 3, k: 2, n: 8,
```

```
our calls:: 1719, their calls: 41256,
our time: 0.203506, their time: 3.743215
t: 3, k: 2, n: 9,
our calls:: 1935, their calls: 52245,
our time: 0.274607, their time: 5.637827
t: 3, k: 2, n: 10,
our calls:: 2151, their calls: 64530,
our time: 0.306516, their time: 7.581443
t: 3, k: 3, n: 1,
our calls:: 288, their calls: 864,
our time: 0.010663, their time: 0.013533
t: 3, k: 3, n: 2,
our calls:: 585, their calls: 3510,
our time: 0.027718, their time: 0.099132
t: 3, k: 3, n: 3,
our calls:: 882, their calls: 7938,
our time: 0.057066, their time: 0.325995
t: 3, k: 3, n: 4,
our calls:: 1179, their calls: 14148,
our time: 0.094411, their time: 1.016786
t: 3, k: 3, n: 5,
our calls:: 1476, their calls: 22140,
our time: 0.171596, their time: 2.205061
t: 3, k: 3, n: 6,
our calls:: 1773, their calls: 31914,
our time: 0.200616, their time: 3.200221
t: 3, k: 3, n: 7,
our calls:: 2070, their calls: 43470,
our time: 0.500478, their time: 4.910358
t: 3, k: 3, n: 8,
our calls:: 2367, their calls: 56808,
our time: 0.454133, their time: 7.626067
t: 3, k: 3, n: 9,
our calls:: 2664, their calls: 71928,
our time: 0.606025, their time: 10.361397
t: 3, k: 3, n: 10,
our calls:: 2961, their calls: 88830,
our time: 0.943325, their time: 14.177224
t: 3, k: 4, n: 1,
our calls:: 369, their calls: 1107,
our time: 0.015467, their time: 0.023488
t: 3, k: 4, n: 2,
our calls:: 747, their calls: 4482,
our time: 0.319547, their time: 0.198072
t: 3, k: 4, n: 3,
our calls:: 1125, their calls: 10125,
our time: 0.111617, their time: 0.726431
t: 3, k: 4, n: 4,
our calls:: 1503, their calls: 18036,
our time: 0.230794, their time: 1.615601
t: 3, k: 4, n: 5,
our calls:: 1881, their calls: 28215,
our time: 0.502784, their time: 3.228228
t: 3, k: 4, n: 6,
our calls:: 2259, their calls: 40662,
our time: 0.315135, their time: 5.278746
t: 3, k: 4, n: 7,
our calls:: 2637, their calls: 55377,
our time: 0.803356, their time: 8.221292
t: 3, k: 4, n: 8,
our calls:: 3015, their calls: 72360,
our time: 0.747430, their time: 12.187074
t: 3, k: 4, n: 9,
our calls:: 3393, their calls: 91611,
our time: 0.708988, their time: 15.210691
t: 3, k: 4, n: 10,
our calls:: 3771, their calls: 113130,
our time: 0.811602, their time: 20.926160
t: 3, k: 5, n: 1,
our calls:: 450, their calls: 1350,
our time: 0.020936, their time: 0.036195
t: 3, k: 5, n: 2,
our calls:: 909, their calls: 5454,
```

```
our time: 0.060107, their time: 0.469369
t: 3, k: 5, n: 3,
our calls:: 1368, their calls: 12312,
our time: 0.108986, their time: 0.702282
t: 3, k: 5, n: 4,
our calls:: 1827, their calls: 21924,
our time: 0.183227, their time: 2.639703
t: 3, k: 5, n: 5,
our calls:: 2286, their calls: 34290,
our time: 0.279045, their time: 3.871706
t: 3, k: 5, n: 6,
our calls:: 2745, their calls: 49410,
our time: 0.624232, their time: 6.853569
t: 3, k: 5, n: 7,
our calls:: 3204, their calls: 67284,
our time: 0.501649, their time: 11.097712
t: 3, k: 5, n: 8,
our calls:: 3663, their calls: 87912,
our time: 0.964971, their time: 16.149378
t: 3, k: 5, n: 9,
our calls:: 4122, their calls: 111294,
our time: 0.891364, their time: 24.266158
t: 3, k: 5, n: 10,
our calls:: 4581, their calls: 137430,
our time: 1.207220, their time: 33.338510
t: 3, k: 6, n: 1,
our calls:: 531, their calls: 1593,
our time: 0.031377, their time: 0.050461
t: 3, k: 6, n: 2,
our calls:: 1071, their calls: 6426,
our time: 0.083441, their time: 0.384982
t: 3, k: 6, n: 3,
our calls:: 1611, their calls: 14499,
our time: 0.459012, their time: 1.337233
t: 3, k: 6, n: 4,
our calls:: 2151, their calls: 25812,
our time: 0.274721, their time: 3.596858
t: 3, k: 6, n: 5,
our calls:: 2691, their calls: 40365,
our time: 0.409325, their time: 6.205636
t: 3, k: 6, n: 6,
our calls:: 3231, their calls: 58158,
our time: 0.650809, their time: 10.677634
t: 3, k: 6, n: 7,
our calls:: 3771, their calls: 79191,
our time: 0.890661, their time: 16.270804
t: 3, k: 6, n: 8,
our calls:: 4311, their calls: 103464,
our time: 1.186446, their time: 24.344263
t: 3, k: 6, n: 9,
our calls:: 4851, their calls: 130977,
our time: 1.648557, their time: 31.035425
t: 3, k: 6, n: 10,
our calls:: 5391, their calls: 161730,
our time: 1.760596, their time: 43.767255
t: 3, k: 7, n: 1,
our calls:: 612, their calls: 1836,
our time: 0.037931, their time: 0.066355
t: 3, k: 7, n: 2,
our calls:: 1233, their calls: 7398,
our time: 0.105352, their time: 0.461713
t: 3, k: 7, n: 3,
our calls:: 1854, their calls: 16686,
our time: 0.208470, their time: 1.865395
t: 3, k: 7, n: 4,
our calls:: 2475, their calls: 29700,
our time: 0.471395, their time: 4.217123
t: 3, k: 7, n: 5,
our calls:: 3096, their calls: 46440,
our time: 0.521308, their time: 8.002924
t: 3, k: 7, n: 6,
our calls:: 3717, their calls: 66906,
our time: 0.766463, their time: 13.968017
```

```
t: 3, k: 7, n: 7,
our calls:: 4338, their calls: 91098,
our time: 0.981477, their time: 21.856352
t: 3, k: 7, n: 8,
our calls:: 4959, their calls: 119016,
our time: 1.375589, their time: 32.562349
t: 3, k: 7, n: 9,
our calls:: 5580, their calls: 150660,
our time: 2.255163, their time: 44.701811
t: 3, k: 7, n: 10,
our calls:: 6201, their calls: 186030,
our time: 1.855011, their time: 56.135291
t: 3, k: 8, n: 1,
our calls:: 693, their calls: 2079,
our time: 0.037738, their time: 0.069269
t: 3, k: 8, n: 2,
our calls:: 1395, their calls: 8370,
our time: 0.110670, their time: 0.995872
t: 3, k: 8, n: 3,
our calls:: 2097, their calls: 18873,
our time: 0.333155, their time: 2.359112
t: 3, k: 8, n: 4,
our calls:: 2799, their calls: 33588,
our time: 0.400727, their time: 5.129232
t: 3, k: 8, n: 5,
our calls:: 3501, their calls: 52515,
our time: 0.696196, their time: 9.899945
t: 3, k: 8, n: 6,
our calls:: 4203, their calls: 75654,
our time: 0.908603, their time: 15.782834
t: 3, k: 8, n: 7,
our calls:: 4905, their calls: 103005,
our time: 1.837140, their time: 24.404078
t: 3, k: 8, n: 8,
our calls:: 5607, their calls: 134568,
our time: 1.473871, their time: 37.270773
t: 3, k: 8, n: 9,
our calls:: 6309, their calls: 170343,
our time: 1.841874, their time: 49.877936
t: 3, k: 8, n: 10,
our calls:: 7011, their calls: 210330,
our time: 2.488659, their time: 57.944794
t: 3, k: 9, n: 1,
our calls:: 774, their calls: 2322,
our time: 0.041510, their time: 0.077673
t: 3, k: 9, n: 2,
our calls:: 1557, their calls: 9342,
our time: 0.384599, their time: 0.582632
t: 3, k: 9, n: 3,
our calls:: 2340, their calls: 21060,
our time: 0.260570, their time: 2.233604
t: 3, k: 9, n: 4,
our calls:: 3123, their calls: 37476,
our time: 0.445951, their time: 4.795135
t: 3, k: 9, n: 5,
our calls:: 3906, their calls: 58590,
our time: 0.961746, their time: 9.279655
t: 3, k: 9, n: 6,
our calls:: 4689, their calls: 84402,
our time: 0.966342, their time: 16.009449
t: 3, k: 9, n: 7,
our calls:: 5472, their calls: 114912,
our time: 1.297947, their time: 24.649912
t: 3, k: 9, n: 8,
our calls:: 6255, their calls: 150120,
our time: 1.993390, their time: 36.267036
t: 3, k: 9, n: 9,
our calls:: 7038, their calls: 190026,
our time: 2.761555, their time: 60.237252
t: 3, k: 9, n: 10,
our calls:: 7821, their calls: 234630,
our time: 4.494003, their time: 117.813494
t: 3, k: 10, n: 1,
```

```
our calls:: 855, their calls: 2565,
our time: 0.082555, their time: 0.182320
t: 3, k: 10, n: 2,
our calls:: 1719, their calls: 10314,
our time: 0.249270, their time: 1.481447
t: 3, k: 10, n: 3,
our calls:: 2583, their calls: 23247,
our time: 0.515756, their time: 4.236537
t: 3, k: 10, n: 4,
our calls:: 3447, their calls: 41364,
our time: 0.853786, their time: 9.336466
t: 3, k: 10, n: 5,
our calls:: 4311, their calls: 64665,
our time: 1.688145, their time: 18.383685
t: 3, k: 10, n: 6,
our calls:: 5175, their calls: 93150,
our time: 2.058272, their time: 31.479171
t: 3, k: 10, n: 7,
our calls:: 6039, their calls: 126819,
our time: 2.631417, their time: 46.593782
t: 3, k: 10, n: 8,
our calls:: 6903, their calls: 165672,
our time: 3.181986, their time: 73.746955
t: 3, k: 10, n: 9,
our calls:: 7767, their calls: 209709,
our time: 4.742419, their time: 102.117847
t: 3, k: 10, n: 10,
our calls:: 8631, their calls: 258930,
our time: 5.650534, their time: 139.767164
t: 4, k: 1, n: 1,
our calls:: 168, their calls: 504,
our time: 0.005921, their time: 0.015401
t: 4, k: 1, n: 2,
our calls:: 348, their calls: 2088,
our time: 0.015833, their time: 0.049128
t: 4, k: 1, n: 3,
our calls:: 528, their calls: 4752,
our time: 0.032429, their time: 0.171485
t: 4, k: 1, n: 4,
our calls:: 708, their calls: 8496,
our time: 0.054031, their time: 0.392776
t: 4, k: 1, n: 5,
our calls:: 888, their calls: 13320,
our time: 0.085310, their time: 1.034392
t: 4, k: 1, n: 6,
our calls:: 1068, their calls: 19224,
our time: 0.119643, their time: 1.158179
t: 4, k: 1, n: 7,
our calls:: 1248, their calls: 26208,
our time: 0.144978, their time: 2.253784
t: 4, k: 1, n: 8,
our calls:: 1428, their calls: 34272,
our time: 0.183395, their time: 3.049574
t: 4, k: 1, n: 9,
our calls:: 1608, their calls: 43416,
our time: 0.235546, their time: 4.109634
t: 4, k: 1, n: 10,
our calls:: 1788, their calls: 53640,
our time: 0.259308, their time: 6.050588
t: 4, k: 2, n: 1,
our calls:: 276, their calls: 828,
our time: 0.013218, their time: 0.018728
t: 4, k: 2, n: 2,
our calls:: 564, their calls: 3384,
our time: 0.038730, their time: 0.135527
t: 4, k: 2, n: 3,
our calls:: 852, their calls: 7668,
our time: 0.085394, their time: 0.459628
t: 4, k: 2, n: 4,
our calls:: 1140, their calls: 13680,
our time: 0.144348, their time: 1.000090
t: 4, k: 2, n: 5,
our calls:: 1428, their calls: 21420,
```

```
our time: 0.180502, their time: 2.179006
t: 4, k: 2, n: 6,
our calls:: 1716, their calls: 30888,
our time: 0.267563, their time: 3.534033
t: 4, k: 2, n: 7,
our calls:: 2004, their calls: 42084,
our time: 0.351100, their time: 5.253877
t: 4, k: 2, n: 8,
our calls:: 2292, their calls: 55008,
our time: 0.445134, their time: 8.175189
t: 4, k: 2, n: 9,
our calls:: 2580, their calls: 69660,
our time: 0.561406, their time: 11.253125
t: 4, k: 2, n: 10,
our calls:: 2868, their calls: 86040,
our time: 0.618494, their time: 16.151280
t: 4, k: 3, n: 1,
our calls:: 384, their calls: 1152,
our time: 0.022960, their time: 0.034768
t: 4, k: 3, n: 2,
our calls:: 780, their calls: 4680,
our time: 0.063000, their time: 0.273110
t: 4, k: 3, n: 3,
our calls:: 1176, their calls: 10584,
our time: 0.131951, their time: 0.872129
t: 4, k: 3, n: 4,
our calls:: 1572, their calls: 18864,
our time: 0.253604, their time: 2.373507
t: 4, k: 3, n: 5,
our calls:: 1968, their calls: 29520,
our time: 0.344326, their time: 4.307065
t: 4, k: 3, n: 6,
our calls:: 2364, their calls: 42552,
our time: 0.508927, their time: 6.651179
t: 4, k: 3, n: 7,
our calls:: 2760, their calls: 57960,
our time: 0.596764, their time: 9.551541
t: 4, k: 3, n: 8,
our calls:: 3156, their calls: 75744,
our time: 0.820964, their time: 15.137089
t: 4, k: 3, n: 9,
our calls:: 3552, their calls: 95904,
our time: 0.922423, their time: 21.272793
t: 4, k: 3, n: 10,
our calls:: 3948, their calls: 118440,
our time: 1.170651, their time: 28.089483
t: 4, k: 4, n: 1,
our calls:: 492, their calls: 1476,
our time: 0.031251, their time: 0.052315
t: 4, k: 4, n: 2,
our calls:: 996, their calls: 5976,
our time: 0.088511, their time: 0.391176
t: 4, k: 4, n: 3,
our calls:: 1500, their calls: 13500,
our time: 0.185202, their time: 1.563037
t: 4, k: 4, n: 4,
our calls:: 2004, their calls: 24048,
our time: 0.311172, their time: 2.922195
t: 4, k: 4, n: 5,
our calls:: 2508, their calls: 37620,
our time: 0.482086, their time: 6.400180
t: 4, k: 4, n: 6,
our calls:: 3012, their calls: 54216,
our time: 0.704233, their time: 9.961352
t: 4, k: 4, n: 7,
our calls:: 3516, their calls: 73836,
our time: 1.304851, their time: 15.144146
t: 4, k: 4, n: 8,
our calls:: 4020, their calls: 96480,
our time: 1.309888, their time: 24.391061
t: 4, k: 4, n: 9,
our calls:: 4524, their calls: 122148,
our time: 1.504433, their time: 33.245968
```

```
t: 4, k: 4, n: 10,
our calls:: 5028, their calls: 150840,
our time: 1.777474, their time: 44.057885
t: 4, k: 5, n: 1,
our calls:: 600, their calls: 1800,
our time: 0.036628, their time: 0.070124
t: 4, k: 5, n: 2,
our calls:: 1212, their calls: 7272,
our time: 0.117533, their time: 0.533225
t: 4, k: 5, n: 3,
our calls:: 1824, their calls: 16416,
our time: 0.705618, their time: 1.849795
t: 4, k: 5, n: 4,
our calls:: 2436, their calls: 29232,
our time: 0.457300, their time: 4.981543
t: 4, k: 5, n: 5,
our calls:: 3048, their calls: 45720,
our time: 0.743419, their time: 8.026619
t: 4, k: 5, n: 6,
our calls:: 3660, their calls: 65880,
our time: 0.935430, their time: 15.700644
t: 4, k: 5, n: 7,
our calls:: 4272, their calls: 89712,
our time: 1.808186, their time: 23.269912
t: 4, k: 5, n: 8,
our calls:: 4884, their calls: 117216,
our time: 1.688726, their time: 35.816214
t: 4, k: 5, n: 9,
our calls:: 5496, their calls: 148392,
our time: 2.113889, their time: 50.029278
t: 4, k: 5, n: 10,
our calls:: 6108, their calls: 183240,
our time: 3.071037, their time: 66.988189
t: 4, k: 6, n: 1,
our calls:: 708, their calls: 2124,
our time: 0.049760, their time: 0.099603
t: 4, k: 6, n: 2,
our calls:: 1428, their calls: 8568,
our time: 0.158264, their time: 0.765359
t: 4, k: 6, n: 3,
our calls:: 2148, their calls: 19332,
our time: 0.355870, their time: 3.011527
t: 4, k: 6, n: 4,
our calls:: 2868, their calls: 34416,
our time: 0.631923, their time: 6.761772
t: 4, k: 6, n: 5,
our calls:: 3588, their calls: 53820,
our time: 0.926937, their time: 12.261022
t: 4, k: 6, n: 6,
our calls:: 4308, their calls: 77544,
our time: 1.820487, their time: 21.307023
t: 4, k: 6, n: 7,
our calls:: 5028, their calls: 105588,
our time: 1.774309, their time: 32.467784
t: 4, k: 6, n: 8,
our calls:: 5748, their calls: 137952,
our time: 2.557083, their time: 47.521286
t: 4, k: 6, n: 9,
our calls:: 6468, their calls: 174636,
our time: 2.903814, their time: 70.679298
t: 4, k: 6, n: 10,
our calls:: 7188, their calls: 215640,
our time: 3.774666, their time: 94.860506
t: 4, k: 7, n: 1,
our calls:: 816, their calls: 2448,
our time: 0.071146, their time: 0.144200
t: 4, k: 7, n: 2,
our calls:: 1644, their calls: 9864,
our time: 0.218966, their time: 1.547490
t: 4, k: 7, n: 3,
our calls:: 2472, their calls: 22248,
our time: 0.436345, their time: 3.454966
t: 4, k: 7, n: 4,
```

```
our calls:: 3300, their calls: 39600,
our time: 0.756202, their time: 9.408781
t: 4, k: 7, n: 5,
our calls:: 4128, their calls: 61920,
our time: 1.256052, their time: 16.191399
t: 4, k: 7, n: 6,
our calls:: 4956, their calls: 89208,
our time: 1.676405, their time: 26.493132
t: 4, k: 7, n: 7,
our calls:: 5784, their calls: 121464,
our time: 2.368588, their time: 44.408478
t: 4, k: 7, n: 8,
our calls:: 6612, their calls: 158688,
our time: 3.534411, their time: 63.613667
t: 4, k: 7, n: 9,
our calls:: 7440, their calls: 200880,
our time: 3.779238, their time: 88.624367
t: 4, k: 7, n: 10,
our calls:: 8268, their calls: 248040,
our time: 5.214866, their time: 127.499180
t: 4, k: 8, n: 1,
our calls:: 924, their calls: 2772,
our time: 0.072159, their time: 0.171858
t: 4, k: 8, n: 2,
our calls:: 1860, their calls: 11160,
our time: 0.273037, their time: 1.423015
t: 4, k: 8, n: 3,
our calls:: 2796, their calls: 25164,
our time: 1.208583, their time: 4.519555
t: 4, k: 8, n: 4,
our calls:: 3732, their calls: 44784,
our time: 0.971180, their time: 11.100590
t: 4, k: 8, n: 5,
our calls:: 4668, their calls: 70020,
our time: 2.051373, their time: 21.072260
t: 4, k: 8, n: 6,
our calls:: 5604, their calls: 100872,
our time: 2.713678, their time: 36.375229
t: 4, k: 8, n: 7,
our calls:: 6540, their calls: 137340,
our time: 2.936016, their time: 53.095036
t: 4, k: 8, n: 8,
our calls:: 7476, their calls: 179424,
our time: 3.232149, their time: 84.009731
t: 4, k: 8, n: 9,
our calls:: 8412, their calls: 227124,
our time: 4.910369, their time: 114.939647
t: 4, k: 8, n: 10,
our calls:: 9348, their calls: 280440,
our time: 6.486665, their time: 160.327870
t: 4, k: 9, n: 1,
our calls:: 1032, their calls: 3096,
our time: 0.674381, their time: 0.228954
t: 4, k: 9, n: 2,
our calls:: 2076, their calls: 12456,
our time: 0.336769, their time: 1.748315
t: 4, k: 9, n: 3,
our calls:: 3120, their calls: 28080,
our time: 0.759214, their time: 6.355535
t: 4, k: 9, n: 4,
our calls:: 4164, their calls: 49968,
our time: 1.251674, their time: 13.642068
t: 4, k: 9, n: 5,
our calls:: 5208, their calls: 78120,
our time: 1.919648, their time: 26.431477
t: 4, k: 9, n: 6,
our calls:: 6252, their calls: 112536,
our time: 2.682173, their time: 45.770271
t: 4, k: 9, n: 7,
our calls:: 7296, their calls: 153216,
our time: 3.657439, their time: 71.016567
t: 4, k: 9, n: 8,
our calls:: 8340, their calls: 200160,
```

```
our time: 4.690297, their time: 108.896678
t: 4, k: 9, n: 9,
our calls:: 9384, their calls: 253368,
our time: 6.044843, their time: 144.324288
t: 4, k: 9, n: 10,
our calls:: 10428, their calls: 312840,
our time: 7.405956, their time: 197.034963
t: 4, k: 10, n: 1,
our calls:: 1140, their calls: 3420,
our time: 0.680993, their time: 0.271661
t: 4, k: 10, n: 2,
our calls:: 2292, their calls: 13752,
our time: 0.408261, their time: 2.030150
t: 4, k: 10, n: 3,
our calls:: 3444, their calls: 30996,
our time: 0.881889, their time: 7.585611
t: 4, k: 10, n: 4,
our calls:: 4596, their calls: 55152,
our time: 1.503546, their time: 16.677157
t: 4, k: 10, n: 5,
our calls:: 5748, their calls: 86220,
our time: 2.353712, their time: 32.251954
t: 4, k: 10, n: 6,
our calls:: 6900, their calls: 124200,
our time: 3.008514, their time: 50.372221
t: 4, k: 10, n: 7,
our calls:: 8052, their calls: 169092,
our time: 3.833561, their time: 81.710739
t: 4, k: 10, n: 8,
our calls:: 9204, their calls: 220896,
our time: 5.355690, their time: 123.886386
t: 4, k: 10, n: 9,
our calls:: 10356, their calls: 279612,
our time: 6.967958, their time: 178.703919
t: 4, k: 10, n: 10,
our calls:: 11508, their calls: 345240,
our time: 8.459781, their time: 240.960049
t: 5, k: 1, n: 1,
our calls:: 210, their calls: 630,
our time: 0.006766, their time: 0.014934
t: 5, k: 1, n: 2,
our calls:: 435, their calls: 2610,
our time: 0.019051, their time: 0.067188
t: 5, k: 1, n: 3,
our calls:: 660, their calls: 5940,
our time: 0.039394, their time: 0.926211
t: 5, k: 1, n: 4,
our calls:: 885, their calls: 10620,
our time: 0.069279, their time: 0.526925
t: 5, k: 1, n: 5,
our calls:: 1110, their calls: 16650,
our time: 0.109561, their time: 1.047604
t: 5, k: 1, n: 6,
our calls:: 1335, their calls: 24030,
our time: 0.138667, their time: 1.653405
t: 5, k: 1, n: 7,
our calls:: 1560, their calls: 32760,
our time: 0.191677, their time: 3.214385
t: 5, k: 1, n: 8,
our calls:: 1785, their calls: 42840,
our time: 0.232260, their time: 4.423898
t: 5, k: 1, n: 9,
our calls:: 2010, their calls: 54270,
our time: 0.316101, their time: 5.699156
t: 5, k: 1, n: 10,
our calls:: 2235, their calls: 67050,
our time: 0.362243, their time: 7.769076
t: 5, k: 2, n: 1,
our calls:: 345, their calls: 1035,
our time: 0.016471, their time: 0.020844
t: 5, k: 2, n: 2,
our calls:: 705, their calls: 4230,
our time: 0.045468, their time: 0.152162
```

```
t: 5, k: 2, n: 3,
our calls:: 1065, their calls: 9585,
our time: 0.091886, their time: 0.505293
t: 5, k: 2, n: 4,
our calls:: 1425, their calls: 17100,
our time: 0.165698, their time: 1.151082
t: 5, k: 2, n: 5,
our calls:: 1785, their calls: 26775,
our time: 0.235647, their time: 3.264219
t: 5, k: 2, n: 6,
our calls:: 2145, their calls: 38610,
our time: 0.353793, their time: 5.368493
t: 5, k: 2, n: 7,
our calls:: 2505, their calls: 52605,
our time: 0.502691, their time: 8.005604
t: 5, k: 2, n: 8,
our calls:: 2865, their calls: 68760,
our time: 0.606684, their time: 11.461009
t: 5, k: 2, n: 9,
our calls:: 3225, their calls: 87075,
our time: 0.783402, their time: 16.798244
t: 5, k: 2, n: 10,
our calls:: 3585, their calls: 107550,
our time: 0.938154, their time: 20.661162
t: 5, k: 3, n: 1,
our calls:: 480, their calls: 1440,
our time: 0.026904, their time: 0.047908
t: 5, k: 3, n: 2,
our calls:: 975, their calls: 5850,
our time: 0.082419, their time: 0.359708
t: 5, k: 3, n: 3,
our calls:: 1470, their calls: 13230,
our time: 0.174457, their time: 1.195043
t: 5, k: 3, n: 4,
our calls:: 1965, their calls: 23580,
our time: 0.295932, their time: 3.390339
t: 5, k: 3, n: 5,
our calls:: 2460, their calls: 36900,
our time: 0.486178, their time: 5.029775
t: 5, k: 3, n: 6,
our calls:: 2955, their calls: 53190,
our time: 0.672145, their time: 9.871459
t: 5, k: 3, n: 7,
our calls:: 3450, their calls: 72450,
our time: 1.566523, their time: 15.040303
t: 5, k: 3, n: 8,
our calls:: 3945, their calls: 94680,
our time: 1.148911, their time: 23.108721
t: 5, k: 3, n: 9,
our calls:: 4440, their calls: 119880,
our time: 1.382198, their time: 30.423537
t: 5, k: 3, n: 10,
our calls:: 4935, their calls: 148050,
our time: 1.697841, their time: 44.207547
t: 5, k: 4, n: 1,
our calls:: 615, their calls: 1845,
our time: 0.038760, their time: 0.072546
t: 5, k: 4, n: 2,
our calls:: 1245, their calls: 7470,
our time: 0.119673, their time: 0.593541
t: 5, k: 4, n: 3,
our calls:: 1875, their calls: 16875,
our time: 0.272425, their time: 2.698132
t: 5, k: 4, n: 4,
our calls:: 2505, their calls: 30060,
our time: 0.478641, their time: 4.574224
t: 5, k: 4, n: 5,
our calls:: 3135, their calls: 47025,
our time: 0.728549, their time: 9.533176
t: 5, k: 4, n: 6,
our calls:: 3765, their calls: 67770,
our time: 1.041776, their time: 16.722258
t: 5, k: 4, n: 7,
```

```
our calls:: 4395, their calls: 92295,
our time: 1.371902, their time: 24.731314
t: 5, k: 4, n: 8,
our calls:: 5025, their calls: 120600,
our time: 2.527476, their time: 35.487855
t: 5, k: 4, n: 9,
our calls:: 5655, their calls: 152685,
our time: 2.227967, their time: 53.161306
t: 5, k: 4, n: 10,
our calls:: 6285, their calls: 188550,
our time: 2.749924, their time: 70.467916
t: 5, k: 5, n: 1,
our calls:: 750, their calls: 2250,
our time: 0.052076, their time: 0.122598
t: 5, k: 5, n: 2,
our calls:: 1515, their calls: 9090,
our time: 0.151691, their time: 0.715207
t: 5, k: 5, n: 3,
our calls:: 2280, their calls: 20520,
our time: 0.388917, their time: 3.004528
t: 5, k: 5, n: 4,
our calls:: 3045, their calls: 36540,
our time: 0.685472, their time: 7.556628
t: 5, k: 5, n: 5,
our calls:: 3810, their calls: 57150,
our time: 0.992233, their time: 13.452012
t: 5, k: 5, n: 6,
our calls:: 4575, their calls: 82350,
our time: 1.303431, their time: 23.537108
t: 5, k: 5, n: 7,
our calls:: 5340, their calls: 112140,
our time: 1.980954, their time: 36.617953
t: 5, k: 5, n: 8,
our calls:: 6105, their calls: 146520,
our time: 2.165378, their time: 52.865545
t: 5, k: 5, n: 9,
our calls:: 6870, their calls: 185490,
our time: 2.822422, their time: 75.377151
t: 5, k: 5, n: 10,
our calls:: 7635, their calls: 229050,
our time: 4.679744, their time: 107.533727
t: 5, k: 6, n: 1,
our calls:: 885, their calls: 2655,
our time: 0.081573, their time: 0.167051
t: 5, k: 6, n: 2,
our calls:: 1785, their calls: 10710,
our time: 0.251737, their time: 1.246128
t: 5, k: 6, n: 3,
our calls:: 2685, their calls: 24165,
our time: 0.533757, their time: 4.779894
t: 5, k: 6, n: 4,
our calls:: 3585, their calls: 43020,
our time: 0.949974, their time: 10.466380
t: 5, k: 6, n: 5,
our calls:: 4485, their calls: 67275,
our time: 1.428179, their time: 19.153901
t: 5, k: 6, n: 6,
our calls:: 5385, their calls: 96930,
our time: 1.902210, their time: 33.087253
t: 5, k: 6, n: 7,
our calls:: 6285, their calls: 131985,
our time: 2.783994, their time: 52.248331
t: 5, k: 6, n: 8,
our calls:: 7185, their calls: 172440,
our time: 3.128520, their time: 78.183008
t: 5, k: 6, n: 9,
our calls:: 8085, their calls: 218295,
our time: 4.078080, their time: 105.723655
t: 5, k: 6, n: 10,
our calls:: 8985, their calls: 269550,
our time: 5.561894, their time: 148.159111
t: 5, k: 7, n: 1,
our calls:: 1020, their calls: 3060,
```

```
our time: 0.105371, their time: 0.238714
t: 5, k: 7, n: 2,
our calls:: 2055, their calls: 12330,
our time: 0.296564, their time: 1.614012
t: 5, k: 7, n: 3,
our calls:: 3090, their calls: 27810,
our time: 0.683228, their time: 5.600978
t: 5, k: 7, n: 4,
our calls:: 4125, their calls: 49500,
our time: 2.039756, their time: 12.622130
t: 5, k: 7, n: 5,
our calls:: 5160, their calls: 77400,
our time: 2.821321, their time: 24.729062
t: 5, k: 7, n: 6,
our calls:: 6195, their calls: 111510,
our time: 2.641437, their time: 43.961695
t: 5, k: 7, n: 7,
our calls:: 7230, their calls: 151830,
our time: 3.546865, their time: 69.261354
t: 5, k: 7, n: 8,
our calls:: 8265, their calls: 198360,
our time: 4.721655, their time: 102.065067
t: 5, k: 7, n: 9,
our calls:: 9300, their calls: 251100,
our time: 6.850436, their time: 146.320059
t: 5, k: 7, n: 10,
our calls:: 10335, their calls: 310050,
our time: 6.219468, their time: 189.339644
t: 5, k: 8, n: 1,
our calls:: 1155, their calls: 3465,
our time: 0.123556, their time: 0.276050
t: 5, k: 8, n: 2,
our calls:: 2325, their calls: 13950,
our time: 0.407030, their time: 2.917564
t: 5, k: 8, n: 3,
our calls:: 3495, their calls: 31455,
our time: 0.868263, their time: 6.758582
t: 5, k: 8, n: 4,
our calls:: 4665, their calls: 55980,
our time: 1.301905, their time: 15.360379
t: 5, k: 8, n: 5,
our calls:: 5835, their calls: 87525,
our time: 2.061074, their time: 33.693969
t: 5, k: 8, n: 6,
our calls:: 7005, their calls: 126090,
our time: 3.301686, their time: 57.132045
t: 5, k: 8, n: 7,
our calls:: 8175, their calls: 171675,
our time: 4.451577, their time: 88.509913
t: 5, k: 8, n: 8,
our calls:: 9345, their calls: 224280,
our time: 5.858126, their time: 131.931078
t: 5, k: 8, n: 9,
our calls:: 10515, their calls: 283905,
our time: 7.445333, their time: 177.124159
t: 5, k: 8, n: 10,
our calls:: 11685, their calls: 350550,
our time: 9.246159, their time: 256.476365
t: 5, k: 9, n: 1,
our calls:: 1290, their calls: 3870,
our time: 0.153915, their time: 0.335271
t: 5, k: 9, n: 2,
our calls:: 2595, their calls: 15570,
our time: 0.485605, their time: 2.647960
t: 5, k: 9, n: 3,
our calls:: 3900, their calls: 35100,
our time: 1.091047, their time: 9.465430
t: 5, k: 9, n: 4,
our calls:: 5205, their calls: 62460,
our time: 1.608200, their time: 20.789334
t: 5, k: 9, n: 5,
our calls:: 6510, their calls: 97650,
our time: 2.726190, their time: 40.901369
```

```
t: 5, k: 9, n: 6,
our calls:: 7815, their calls: 140670,
our time: 4.247286, their time: 70.962821
t: 5, k: 9, n: 7,
our calls:: 9120, their calls: 191520,
our time: 5.627756, their time: 107.683679
t: 5, k: 9, n: 8,
our calls:: 10425, their calls: 250200,
our time: 7.201222, their time: 162.212744
t: 5, k: 9, n: 9,
our calls:: 11730, their calls: 316710,
our time: 8.641648, their time: 222.065049
t: 5, k: 9, n: 10,
our calls:: 13035, their calls: 391050,
our time: 12.560786, their time: 316.457050
t: 5, k: 10, n: 1,
our calls:: 1425, their calls: 4275,
our time: 1.298192, their time: 0.453135
t: 5, k: 10, n: 2,
our calls:: 2865, their calls: 17190,
our time: 0.651441, their time: 3.348512
t: 5, k: 10, n: 3,
our calls:: 4305, their calls: 38745,
our time: 1.225114, their time: 10.498741
t: 5, k: 10, n: 4,
our calls:: 5745, their calls: 68940,
our time: 2.286321, their time: 25.670357
t: 5, k: 10, n: 5,
our calls:: 7185, their calls: 107775,
our time: 3.521117, their time: 48.298115
t: 5, k: 10, n: 6,
our calls:: 8625, their calls: 155250,
our time: 5.934823, their time: 85.861756
t: 5, k: 10, n: 7,
our calls:: 10065, their calls: 211365,
our time: 6.945241, their time: 136.532422
t: 5, k: 10, n: 8,
our calls:: 11505, their calls: 276120,
our time: 10.234348, their time: 198.255658
t: 5, k: 10, n: 9,
our calls:: 12945, their calls: 349515,
our time: 11.268718, their time: 276.678110
t: 5, k: 10, n: 10,
our calls:: 14385, their calls: 431550,
our time: 13.692086, their time: 387.176759
t: 6, k: 1, n: 1,
our calls:: 252, their calls: 756,
our time: 0.007795, their time: 0.020763
t: 6, k: 1, n: 2,
our calls:: 522, their calls: 3132,
our time: 0.024451, their time: 0.083043
t: 6, k: 1, n: 3,
our calls:: 792, their calls: 7128,
our time: 0.048764, their time: 0.290839
t: 6, k: 1, n: 4,
our calls:: 1062, their calls: 12744,
our time: 0.089631, their time: 0.710476
t: 6, k: 1, n: 5,
our calls:: 1332, their calls: 19980,
our time: 0.135738, their time: 1.354101
t: 6, k: 1, n: 6,
our calls:: 1602, their calls: 28836,
our time: 0.191320, their time: 2.337330
t: 6, k: 1, n: 7,
our calls:: 1872, their calls: 39312,
our time: 0.263060, their time: 4.720976
t: 6, k: 1, n: 8,
our calls:: 2142, their calls: 51408,
our time: 0.342966, their time: 4.634754
t: 6, k: 1, n: 9,
our calls:: 2412, their calls: 65124,
our time: 0.425591, their time: 8.865811
t: 6, k: 1, n: 10,
```

```
our calls:: 2682, their calls: 80460,
our time: 0.531633, their time: 11.725888
t: 6, k: 2, n: 1,
our calls:: 414, their calls: 1242,
our time: 0.019530, their time: 0.034370
t: 6, k: 2, n: 2,
our calls:: 846, their calls: 5076,
our time: 0.062472, their time: 0.273778
t: 6, k: 2, n: 3,
our calls:: 1278, their calls: 11502,
our time: 0.124090, their time: 0.827053
t: 6, k: 2, n: 4,
our calls:: 1710, their calls: 20520,
our time: 0.241214, their time: 2.969142
t: 6, k: 2, n: 5,
our calls:: 2142, their calls: 32130,
our time: 0.346312, their time: 3.436179
t: 6, k: 2, n: 6,
our calls:: 2574, their calls: 46332,
our time: 0.433737, their time: 7.135024
t: 6, k: 2, n: 7,
our calls:: 3006, their calls: 63126,
our time: 0.629727, their time: 11.031614
t: 6, k: 2, n: 8,
our calls:: 3438, their calls: 82512,
our time: 0.811618, their time: 15.047169
t: 6, k: 2, n: 9,
our calls:: 3870, their calls: 104490,
our time: 0.953575, their time: 22.838665
t: 6, k: 2, n: 10,
our calls:: 4302, their calls: 129060,
our time: 1.236845, their time: 31.922113
t: 6, k: 3, n: 1,
our calls:: 576, their calls: 1728,
our time: 0.036473, their time: 0.067879
t: 6, k: 3, n: 2,
our calls:: 1170, their calls: 7020,
our time: 0.110827, their time: 0.508825
t: 6, k: 3, n: 3,
our calls:: 1764, their calls: 15876,
our time: 0.234079, their time: 1.666476
t: 6, k: 3, n: 4,
our calls:: 2358, their calls: 28296,
our time: 0.424397, their time: 4.979169
t: 6, k: 3, n: 5,
our calls:: 2952, their calls: 44280,
our time: 0.604420, their time: 7.357694
t: 6, k: 3, n: 6,
our calls:: 3546, their calls: 63828,
our time: 0.843171, their time: 14.167304
t: 6, k: 3, n: 7,
our calls:: 4140, their calls: 86940,
our time: 1.251346, their time: 21.861705
t: 6, k: 3, n: 8,
our calls:: 4734, their calls: 113616,
our time: 1.562904, their time: 31.148802
t: 6, k: 3, n: 9,
our calls:: 5328, their calls: 143856,
our time: 1.681736, their time: 45.545795
t: 6, k: 3, n: 10,
our calls:: 5922, their calls: 177660,
our time: 2.525163, their time: 61.848088
t: 6, k: 4, n: 1,
our calls:: 738, their calls: 2214,
our time: 0.054332, their time: 0.100311
t: 6, k: 4, n: 2,
our calls:: 1494, their calls: 8964,
our time: 0.162061, their time: 0.832396
t: 6, k: 4, n: 3,
our calls:: 2250, their calls: 20250,
our time: 0.382968, their time: 3.925467
t: 6, k: 4, n: 4,
our calls:: 3006, their calls: 36072,
```

```
our time: 0.640162, their time: 6.563262
t: 6, k: 4, n: 5,
our calls:: 3762, their calls: 56430,
our time: 1.004751, their time: 13.236617
t: 6, k: 4, n: 6,
our calls:: 4518, their calls: 81324,
our time: 1.222180, their time: 22.243098
t: 6, k: 4, n: 7,
our calls:: 5274, their calls: 110754,
our time: 1.775844, their time: 33.777912
t: 6, k: 4, n: 8,
our calls:: 6030, their calls: 144720,
our time: 2.428922, their time: 53.892896
t: 6, k: 4, n: 9,
our calls:: 6786, their calls: 183222,
our time: 3.078911, their time: 74.615418
t: 6, k: 4, n: 10,
our calls:: 7542, their calls: 226260,
our time: 3.922653, their time: 102.271947
t: 6, k: 5, n: 1,
our calls:: 900, their calls: 2700,
our time: 0.076400, their time: 0.169867
t: 6, k: 5, n: 2,
our calls:: 1818, their calls: 10908,
our time: 0.254959, their time: 1.233663
t: 6, k: 5, n: 3,
our calls:: 2736, their calls: 24624,
our time: 0.542793, their time: 5.320720
t: 6, k: 5, n: 4,
our calls:: 3654, their calls: 43848,
our time: 0.925817, their time: 9.364430
t: 6, k: 5, n: 5,
our calls:: 4572, their calls: 68580,
our time: 1.442503, their time: 20.500967
t: 6, k: 5, n: 6,
our calls:: 5490, their calls: 98820,
our time: 2.105196, their time: 35.048452
t: 6, k: 5, n: 7,
our calls:: 6408, their calls: 134568,
our time: 2.908292, their time: 54.402021
t: 6, k: 5, n: 8,
our calls:: 7326, their calls: 175824,
our time: 3.618730, their time: 80.562847
t: 6, k: 5, n: 9,
our calls:: 8244, their calls: 222588,
our time: 4.567117, their time: 108.537347
t: 6, k: 5, n: 10,
our calls:: 9162, their calls: 274860,
our time: 5.537830, their time: 152.827021
t: 6, k: 6, n: 1,
our calls:: 1062, their calls: 3186,
our time: 0.104366, their time: 0.232997
t: 6, k: 6, n: 2,
our calls:: 2142, their calls: 12852,
our time: 0.349762, their time: 1.799358
t: 6, k: 6, n: 3,
our calls:: 3222, their calls: 28998,
our time: 0.767231, their time: 5.798205
t: 6, k: 6, n: 4,
our calls:: 4302, their calls: 51624,
our time: 1.205798, their time: 14.254583
t: 6, k: 6, n: 5,
our calls:: 5382, their calls: 80730,
our time: 1.956840, their time: 26.337820
t: 6, k: 6, n: 6,
our calls:: 6462, their calls: 116316,
our time: 2.849983, their time: 46.772648
t: 6, k: 6, n: 7,
our calls:: 7542, their calls: 158382,
our time: 3.346141, their time: 70.956301
t: 6, k: 6, n: 8,
our calls:: 8622, their calls: 206928,
our time: 4.914671, their time: 112.659709
```

```
t: 6, k: 6, n: 9,
our calls:: 9702, their calls: 261954,
our time: 6.299151, their time: 158.644264
t: 6, k: 6, n: 10,
our calls:: 10782, their calls: 323460,
our time: 7.836864, their time: 218.994661
t: 6, k: 7, n: 1,
our calls:: 1224, their calls: 3672,
our time: 0.127390, their time: 0.317763
t: 6, k: 7, n: 2,
our calls:: 2466, their calls: 14796,
our time: 0.418493, their time: 2.299661
t: 6, k: 7, n: 3,
our calls:: 3708, their calls: 33372,
our time: 0.956503, their time: 9.113930
t: 6, k: 7, n: 4,
our calls:: 4950, their calls: 59400,
our time: 1.746230, their time: 19.777956
t: 6, k: 7, n: 5,
our calls:: 6192, their calls: 92880,
our time: 2.584796, their time: 35.875103
t: 6, k: 7, n: 6,
our calls:: 7434, their calls: 133812,
our time: 3.835997, their time: 63.864342
t: 6, k: 7, n: 7,
our calls:: 8676, their calls: 182196,
our time: 4.614933, their time: 95.355056
t: 6, k: 7, n: 8,
our calls:: 9918, their calls: 238032,
our time: 6.612118, their time: 148.392373
t: 6, k: 7, n: 9,
our calls:: 11160, their calls: 301320,
our time: 8.467298, their time: 205.015859
t: 6, k: 7, n: 10,
our calls:: 12402, their calls: 372060,
our time: 10.248931, their time: 280.852421
t: 6, k: 8, n: 1,
our calls:: 1386, their calls: 4158,
our time: 0.185164, their time: 0.426437
t: 6, k: 8, n: 2,
our calls:: 2790, their calls: 16740,
our time: 1.857332, their time: 3.161987
t: 6, k: 8, n: 3,
our calls:: 4194, their calls: 37746,
our time: 1.280958, their time: 10.198576
t: 6, k: 8, n: 4,
our calls:: 5598, their calls: 67176,
our time: 2.137817, their time: 24.057058
t: 6, k: 8, n: 5,
our calls:: 7002, their calls: 105030,
our time: 3.327835, their time: 45.031005
t: 6, k: 8, n: 6,
our calls:: 8406, their calls: 151308,
our time: 6.291521, their time: 80.564597
t: 6, k: 8, n: 7,
our calls:: 9810, their calls: 206010,
our time: 7.845969, their time: 125.537975
t: 6, k: 8, n: 8,
our calls:: 11214, their calls: 269136,
our time: 8.226374, their time: 186.514991
t: 6, k: 8, n: 9,
our calls:: 12618, their calls: 340686,
our time: 10.966486, their time: 265.687078
t: 6, k: 8, n: 10,
our calls:: 14022, their calls: 420660,
our time: 14.499462, their time: 366.898871
t: 6, k: 9, n: 1,
our calls:: 1548, their calls: 4644,
our time: 0.196880, their time: 0.490184
t: 6, k: 9, n: 2,
our calls:: 3114, their calls: 18684,
our time: 0.695739, their time: 5.155481
t: 6, k: 9, n: 3,
```

```
our calls:: 4680, their calls: 42120,
our time: 1.546667, their time: 12.377581
t: 6, k: 9, n: 4,
our calls:: 6246, their calls: 74952,
our time: 2.649053, their time: 31.284639
t: 6, k: 9, n: 5,
our calls:: 7812, their calls: 117180,
our time: 4.079359, their time: 60.594570
t: 6, k: 9, n: 6,
our calls:: 9378, their calls: 168804,
our time: 5.823162, their time: 101.733785
t: 6, k: 9, n: 7,
our calls:: 10944, their calls: 229824,
our time: 8.626961, their time: 158.145229
t: 6, k: 9, n: 8,
our calls:: 12510, their calls: 300240,
our time: 12.071497, their time: 233.769728
t: 6, k: 9, n: 9,
our calls:: 14076, their calls: 380052,
our time: 13.330487, their time: 335.373129
t: 6, k: 9, n: 10,
our calls:: 15642, their calls: 469260,
our time: 18.060014, their time: 453.888065
t: 6, k: 10, n: 1,
our calls:: 1710, their calls: 5130,
our time: 0.226720, their time: 0.496526
t: 6, k: 10, n: 2,
our calls:: 3438, their calls: 20628,
our time: 0.728900, their time: 4.335770
t: 6, k: 10, n: 3,
our calls:: 5166, their calls: 46494,
our time: 1.619505, their time: 16.244285
t: 6, k: 10, n: 4,
our calls:: 6894, their calls: 82728,
our time: 2.711489, their time: 37.526251
t: 6, k: 10, n: 5,
our calls:: 8622, their calls: 129330,
our time: 5.135960, their time: 70.914237
t: 6, k: 10, n: 6,
our calls:: 10350, their calls: 186300,
our time: 7.645797, their time: 124.649380
t: 6, k: 10, n: 7,
our calls:: 12078, their calls: 253638,
our time: 9.980061, their time: 189.177061
t: 6, k: 10, n: 8,
our calls:: 13806, their calls: 331344,
our time: 11.267256, their time: 287.608415
t: 6, k: 10, n: 9,
our calls:: 15534, their calls: 419418,
our time: 15.969133, their time: 407.830064
t: 6, k: 10, n: 10,
our calls:: 17262, their calls: 517860,
our time: 21.808863, their time: 577.509630
t: 7, k: 1, n: 1,
our calls:: 294, their calls: 882,
our time: 0.026825, their time: 0.016474
t: 7, k: 1, n: 2,
our calls:: 609, their calls: 3654,
our time: 0.033614, their time: 0.123978
t: 7, k: 1, n: 3,
our calls:: 924, their calls: 8316,
our time: 0.068772, their time: 0.409255
t: 7, k: 1, n: 4,
our calls:: 1239, their calls: 14868,
our time: 0.116362, their time: 0.951684
t: 7, k: 1, n: 5,
our calls:: 1554, their calls: 23310,
our time: 0.175643, their time: 1.859761
t: 7, k: 1, n: 6,
our calls:: 1869, their calls: 33642,
our time: 0.253898, their time: 4.699047
t: 7, k: 1, n: 7,
our calls:: 2184, their calls: 45864,
```

```
our time: 0.342156, their time: 5.330021
t: 7, k: 1, n: 8,
our calls:: 2499, their calls: 59976,
our time: 0.429953, their time: 8.931295
t: 7, k: 1, n: 9,
our calls:: 2814, their calls: 75978,
our time: 0.548444, their time: 9.536687
t: 7, k: 1, n: 10,
our calls:: 3129, their calls: 93870,
our time: 0.676222, their time: 15.563566
t: 7, k: 2, n: 1,
our calls:: 483, their calls: 1449,
our time: 0.025458, their time: 0.045859
t: 7, k: 2, n: 2,
our calls:: 987, their calls: 5922,
our time: 0.078774, their time: 0.352462
t: 7, k: 2, n: 3,
our calls:: 1491, their calls: 13419,
our time: 0.173701, their time: 1.188827
t: 7, k: 2, n: 4,
our calls:: 1995, their calls: 23940,
our time: 0.304762, their time: 4.198466
t: 7, k: 2, n: 5,
our calls:: 2499, their calls: 37485,
our time: 0.451229, their time: 4.702644
t: 7, k: 2, n: 6,
our calls:: 3003, their calls: 54054,
our time: 0.594432, their time: 9.163537
t: 7, k: 2, n: 7,
our calls:: 3507, their calls: 73647,
our time: 2.470219, their time: 15.176495
t: 7, k: 2, n: 8,
our calls:: 4011, their calls: 96264,
our time: 1.086719, their time: 23.134761
t: 7, k: 2, n: 9,
our calls:: 4515, their calls: 121905,
our time: 1.419800, their time: 31.638751
t: 7, k: 2, n: 10,
our calls:: 5019, their calls: 150570,
our time: 1.675115, their time: 44.216739
t: 7, k: 3, n: 1,
our calls:: 672, their calls: 2016,
our time: 0.049040, their time: 0.088697
t: 7, k: 3, n: 2,
our calls:: 1365, their calls: 8190,
our time: 0.151741, their time: 0.697610
t: 7, k: 3, n: 3,
our calls:: 2058, their calls: 18522,
our time: 0.328742, their time: 2.316587
t: 7, k: 3, n: 4,
our calls:: 2751, their calls: 33012,
our time: 0.541488, their time: 5.224939
t: 7, k: 3, n: 5,
our calls:: 3444, their calls: 51660,
our time: 0.805295, their time: 11.807928
t: 7, k: 3, n: 6,
our calls:: 4137, their calls: 74466,
our time: 1.154317, their time: 19.465000
t: 7, k: 3, n: 7,
our calls:: 4830, their calls: 101430,
our time: 1.636604, their time: 29.158130
t: 7, k: 3, n: 8,
our calls:: 5523, their calls: 132552,
our time: 1.755977, their time: 42.303697
t: 7, k: 3, n: 9,
our calls:: 6216, their calls: 167832,
our time: 2.608354, their time: 60.691621
t: 7, k: 3, n: 10,
our calls:: 6909, their calls: 207270,
our time: 2.618787, their time: 82.068689
t: 7, k: 4, n: 1,
our calls:: 861, their calls: 2583,
our time: 0.073067, their time: 0.146547
```

```
t: 7, k: 4, n: 2,
our calls:: 1743, their calls: 10458,
our time: 0.230142, their time: 1.134746
t: 7, k: 4, n: 3,
our calls:: 2625, their calls: 23625,
our time: 0.493785, their time: 3.784958
t: 7, k: 4, n: 4,
our calls:: 3507, their calls: 42084,
our time: 0.857068, their time: 10.066326
t: 7, k: 4, n: 5,
our calls:: 4389, their calls: 65835,
our time: 1.300851, their time: 16.127228
t: 7, k: 4, n: 6,
our calls:: 5271, their calls: 94878,
our time: 1.901263, their time: 29.838768
t: 7, k: 4, n: 7,
our calls:: 6153, their calls: 129213,
our time: 2.630014, their time: 47.520882
t: 7, k: 4, n: 8,
our calls:: 7035, their calls: 168840,
our time: 3.252946, their time: 68.053957
t: 7, k: 4, n: 9,
our calls:: 7917, their calls: 213759,
our time: 4.214629, their time: 100.673177
t: 7, k: 4, n: 10,
our calls:: 8799, their calls: 263970,
our time: 7.030279, their time: 135.916417
t: 7, k: 5, n: 1,
our calls:: 1050, their calls: 3150,
our time: 0.097411, their time: 0.217318
t: 7, k: 5, n: 2,
our calls:: 2121, their calls: 12726,
our time: 0.338379, their time: 1.711490
t: 7, k: 5, n: 3,
our calls:: 3192, their calls: 28728,
our time: 0.743662, their time: 7.173618
t: 7, k: 5, n: 4,
our calls:: 4263, their calls: 51156,
our time: 1.251553, their time: 13.383701
t: 7, k: 5, n: 5,
our calls:: 5334, their calls: 80010,
our time: 1.991558, their time: 24.968674
t: 7, k: 5, n: 6,
our calls:: 6405, their calls: 115290,
our time: 2.709511, their time: 45.104885
t: 7, k: 5, n: 7,
our calls:: 7476, their calls: 156996,
our time: 3.234218, their time: 71.191793
t: 7, k: 5, n: 8,
our calls:: 8547, their calls: 205128,
our time: 4.800879, their time: 105.315689
t: 7, k: 5, n: 9,
our calls:: 9618, their calls: 259686,
our time: 6.071962, their time: 150.265350
t: 7, k: 5, n: 10,
our calls:: 10689, their calls: 320670,
our time: 7.498894, their time: 198.587227
t: 7, k: 6, n: 1,
our calls:: 1239, their calls: 3717,
our time: 0.142072, their time: 0.318199
t: 7, k: 6, n: 2,
our calls:: 2499, their calls: 14994,
our time: 0.455191, their time: 3.872020
t: 7, k: 6, n: 3,
our calls:: 3759, their calls: 33831,
our time: 0.980368, their time: 6.781050
t: 7, k: 6, n: 4,
our calls:: 5019, their calls: 60228,
our time: 1.742970, their time: 18.327681
t: 7, k: 6, n: 5,
our calls:: 6279, their calls: 94185,
our time: 4.711157, their time: 34.736189
t: 7, k: 6, n: 6,
```

```
our calls:: 7539, their calls: 135702,
our time: 5.399776, their time: 63.380496
t: 7, k: 6, n: 7,
our calls:: 8799, their calls: 184779,
our time: 5.035989, their time: 101.510741
t: 7, k: 6, n: 8,
our calls:: 10059, their calls: 241416,
our time: 6.668909, their time: 146.676869
t: 7, k: 6, n: 9,
our calls:: 11319, their calls: 305613,
our time: 8.423816, their time: 212.972220
t: 7, k: 6, n: 10,
our calls:: 12579, their calls: 377370,
our time: 9.798779, their time: 291.803939
t: 7, k: 7, n: 1,
our calls:: 1428, their calls: 4284,
our time: 0.169853, their time: 0.401094
t: 7, k: 7, n: 2,
our calls:: 2877, their calls: 17262,
our time: 0.611225, their time: 3.293942
t: 7, k: 7, n: 3,
our calls:: 4326, their calls: 38934,
our time: 1.343099, their time: 10.492624
t: 7, k: 7, n: 4,
our calls:: 5775, their calls: 69300,
our time: 2.271925, their time: 26.804824
t: 7, k: 7, n: 5,
our calls:: 7224, their calls: 108360,
our time: 3.512438, their time: 49.924762
t: 7, k: 7, n: 6,
our calls:: 8673, their calls: 156114,
our time: 5.151301, their time: 85.813252
t: 7, k: 7, n: 7,
our calls:: 10122, their calls: 212562,
our time: 8.712276, their time: 132.256803
t: 7, k: 7, n: 8,
our calls:: 11571, their calls: 277704,
our time: 10.779808, their time: 198.901430
t: 7, k: 7, n: 9,
our calls:: 13020, their calls: 351540,
our time: 11.298109, their time: 282.939902
t: 7, k: 7, n: 10,
our calls:: 14469, their calls: 434070,
our time: 15.803842, their time: 392.946509
t: 7, k: 8, n: 1,
our calls:: 1617, their calls: 4851,
our time: 0.244970, their time: 0.559442
t: 7, k: 8, n: 2,
our calls:: 3255, their calls: 19530,
our time: 0.741282, their time: 5.377186
t: 7, k: 8, n: 3,
our calls:: 4893, their calls: 44037,
our time: 1.381326, their time: 12.795251
t: 7, k: 8, n: 4,
our calls:: 6531, their calls: 78372,
our time: 2.360172, their time: 30.400573
t: 7, k: 8, n: 5,
our calls:: 8169, their calls: 122535,
our time: 4.456230, their time: 62.203947
t: 7, k: 8, n: 6,
our calls:: 9807, their calls: 176526,
our time: 6.345559, their time: 104.893429
t: 7, k: 8, n: 7,
our calls:: 11445, their calls: 240345,
our time: 8.614613, their time: 176.189146
t: 7, k: 8, n: 8,
our calls:: 13083, their calls: 313992,
our time: 11.651840, their time: 255.138583
t: 7, k: 8, n: 9,
our calls:: 14721, their calls: 397467,
our time: 13.835402, their time: 354.362328
t: 7, k: 8, n: 10,
our calls:: 16359, their calls: 490770,
```

```
our time: 18.419844, their time: 501.134763
t: 7, k: 9, n: 1,
our calls:: 1806, their calls: 5418,
our time: 0.225535, their time: 0.501485
t: 7, k: 9, n: 2,
our calls:: 3633, their calls: 21798,
our time: 0.752163, their time: 5.099966
t: 7, k: 9, n: 3,
our calls:: 5460, their calls: 49140,
our time: 1.766058, their time: 16.821765
t: 7, k: 9, n: 4,
our calls:: 7287, their calls: 87444,
our time: 3.543414, their time: 40.984330
t: 7, k: 9, n: 5,
our calls:: 9114, their calls: 136710,
our time: 5.807978, their time: 75.413115
t: 7, k: 9, n: 6,
our calls:: 10941, their calls: 196938,
our time: 10.215846, their time: 130.650907
t: 7, k: 9, n: 7,
our calls:: 12768, their calls: 268128,
our time: 10.082002, their time: 216.953490
t: 7, k: 9, n: 8,
our calls:: 14595, their calls: 350280,
our time: 14.287354, their time: 323.882458
t: 7, k: 9, n: 9,
our calls:: 16422, their calls: 443394,
our time: 18.078825, their time: 466.815101
t: 7, k: 9, n: 10,
our calls:: 18249, their calls: 547470,
our time: 22.881508, their time: 641.610683
t: 7, k: 10, n: 1,
our calls:: 1995, their calls: 5985,
our time: 0.352223, their time: 0.825854
t: 7, k: 10, n: 2,
our calls:: 4011, their calls: 24066,
our time: 1.199963, their time: 8.774007
t: 7, k: 10, n: 3,
our calls:: 6027, their calls: 54243,
our time: 2.504162, their time: 21.251596
t: 7, k: 10, n: 4,
our calls:: 8043, their calls: 96516,
our time: 3.848245, their time: 51.104632
t: 7, k: 10, n: 5,
our calls:: 10059, their calls: 150885,
our time: 6.781715, their time: 98.102852
t: 7, k: 10, n: 6,
our calls:: 12075, their calls: 217350,
our time: 10.100376, their time: 166.520915
t: 7, k: 10, n: 7,
our calls:: 14091, their calls: 295911,
our time: 13.351862, their time: 270.961541
t: 7, k: 10, n: 8,
our calls:: 16107, their calls: 386568,
our time: 17.884457, their time: 408.346053
t: 7, k: 10, n: 9,
our calls:: 18123, their calls: 489321,
our time: 21.986953, their time: 562.243828
t: 7, k: 10, n: 10,
our calls:: 20139, their calls: 604170,
our time: 27.385792, their time: 776.199605
t: 8, k: 1, n: 1,
our calls:: 336, their calls: 1008,
our time: 0.026026, their time: 0.020462
t: 8, k: 1, n: 2,
our calls:: 696, their calls: 4176,
our time: 0.039732, their time: 0.155553
t: 8, k: 1, n: 3,
our calls:: 1056, their calls: 9504,
our time: 0.084800, their time: 0.529398
t: 8, k: 1, n: 4,
our calls:: 1416, their calls: 16992,
our time: 0.145213, their time: 1.241591
```

```
t: 8, k: 1, n: 5,
our calls:: 1776, their calls: 26640,
our time: 0.229062, their time: 2.367815
t: 8, k: 1, n: 6,
our calls:: 2136, their calls: 38448,
our time: 0.305809, their time: 6.190082
t: 8, k: 1, n: 7,
our calls:: 2496, their calls: 52416,
our time: 0.431840, their time: 6.585748
t: 8, k: 1, n: 8,
our calls:: 2856, their calls: 68544,
our time: 0.538806, their time: 9.694024
t: 8, k: 1, n: 9,
our calls:: 3216, their calls: 86832,
our time: 0.659971, their time: 15.617721
t: 8, k: 1, n: 10,
our calls:: 3576, their calls: 107280,
our time: 0.830144, their time: 20.470824
t: 8, k: 2, n: 1,
our calls:: 552, their calls: 1656,
our time: 0.034391, their time: 0.061846
t: 8, k: 2, n: 2,
our calls:: 1128, their calls: 6768,
our time: 0.104711, their time: 0.456903
t: 8, k: 2, n: 3,
our calls:: 1704, their calls: 15336,
our time: 0.209807, their time: 1.478127
t: 8, k: 2, n: 4,
our calls:: 2280, their calls: 27360,
our time: 0.378353, their time: 3.559620
t: 8, k: 2, n: 5,
our calls:: 2856, their calls: 42840,
our time: 0.540077, their time: 6.952729
t: 8, k: 2, n: 6,
our calls:: 3432, their calls: 61776,
our time: 0.843769, their time: 14.464290
t: 8, k: 2, n: 7,
our calls:: 4008, their calls: 84168,
our time: 1.140471, their time: 20.792922
t: 8, k: 2, n: 8,
our calls:: 4584, their calls: 110016,
our time: 1.370840, their time: 25.964472
t: 8, k: 2, n: 9,
our calls:: 5160, their calls: 139320,
our time: 1.477705, their time: 40.172403
t: 8, k: 2, n: 10,
our calls:: 5736, their calls: 172080,
our time: 2.172783, their time: 58.773534
t: 8, k: 3, n: 1,
our calls:: 768, their calls: 2304,
our time: 0.057608, their time: 0.117619
t: 8, k: 3, n: 2,
our calls:: 1560, their calls: 9360,
our time: 0.187141, their time: 0.914369
t: 8, k: 3, n: 3,
our calls:: 2352, their calls: 21168,
our time: 0.411686, their time: 3.008949
t: 8, k: 3, n: 4,
our calls:: 3144, their calls: 37728,
our time: 0.690014, their time: 6.965950
t: 8, k: 3, n: 5,
our calls:: 3936, their calls: 59040,
our time: 1.060039, their time: 16.192095
t: 8, k: 3, n: 6,
our calls:: 4728, their calls: 85104,
our time: 1.523784, their time: 23.583300
t: 8, k: 3, n: 7,
our calls:: 5520, their calls: 115920,
our time: 2.115043, their time: 37.033889
t: 8, k: 3, n: 8,
our calls:: 6312, their calls: 151488,
our time: 2.636528, their time: 56.932530
t: 8, k: 3, n: 9,
```

```
our calls:: 7104, their calls: 191808,
our time: 3.432348, their time: 79.059301
t: 8, k: 3, n: 10,
our calls:: 7896, their calls: 236880,
our time: 4.144304, their time: 110.775367
t: 8, k: 4, n: 1,
our calls:: 984, their calls: 2952,
our time: 0.090312, their time: 0.193765
t: 8, k: 4, n: 2,
our calls:: 1992, their calls: 11952,
our time: 0.298428, their time: 1.534838
t: 8, k: 4, n: 3,
our calls:: 3000, their calls: 27000,
our time: 0.677066, their time: 4.984306
t: 8, k: 4, n: 4,
our calls:: 4008, their calls: 48096,
our time: 1.112166, their time: 11.618020
t: 8, k: 4, n: 5,
our calls:: 5016, their calls: 75240,
our time: 1.703000, their time: 24.120382
t: 8, k: 4, n: 6,
our calls:: 6024, their calls: 108432,
our time: 2.661314, their time: 38.435081
t: 8, k: 4, n: 7,
our calls:: 7032, their calls: 147672,
our time: 3.517899, their time: 60.813784
t: 8, k: 4, n: 8,
our calls:: 8040, their calls: 192960,
our time: 3.730983, their time: 89.366920
t: 8, k: 4, n: 9,
our calls:: 9048, their calls: 244296,
our time: 5.378721, their time: 132.154617
t: 8, k: 4, n: 10,
our calls:: 10056, their calls: 301680,
our time: 5.846350, their time: 181.166367
t: 8, k: 5, n: 1,
our calls:: 1200, their calls: 3600,
our time: 0.131785, their time: 0.313672
t: 8, k: 5, n: 2,
our calls:: 2424, their calls: 14544,
our time: 0.408393, their time: 2.254902
t: 8, k: 5, n: 3,
our calls:: 3648, their calls: 32832,
our time: 0.926407, their time: 7.210264
t: 8, k: 5, n: 4,
our calls:: 4872, their calls: 58464,
our time: 1.323679, their time: 15.917469
t: 8, k: 5, n: 5,
our calls:: 6096, their calls: 91440,
our time: 2.449563, their time: 35.534435
t: 8, k: 5, n: 6,
our calls:: 7320, their calls: 131760,
our time: 3.640947, their time: 57.186868
t: 8, k: 5, n: 7,
our calls:: 8544, their calls: 179424,
our time: 4.042725, their time: 95.290599
t: 8, k: 5, n: 8,
our calls:: 9768, their calls: 234432,
our time: 6.587167, their time: 139.210817
t: 8, k: 5, n: 9,
our calls:: 10992, their calls: 296784,
our time: 7.950667, their time: 199.994253
t: 8, k: 5, n: 10,
our calls:: 12216, their calls: 366480,
our time: 12.110478, their time: 209.587583
t: 8, k: 6, n: 1,
our calls:: 1416, their calls: 4248,
our time: 0.177767, their time: 0.413908
t: 8, k: 6, n: 2,
our calls:: 2856, their calls: 17136,
our time: 0.599873, their time: 3.293600
t: 8, k: 6, n: 3,
our calls:: 4296, their calls: 38664,
```

```
our time: 1.378110, their time: 13.004928
t: 8, k: 6, n: 4,
our calls:: 5736, their calls: 68832,
our time: 2.067860, their time: 25.177651
t: 8, k: 6, n: 5,
our calls:: 7176, their calls: 107640,
our time: 3.550530, their time: 51.367172
t: 8, k: 6, n: 6,
our calls:: 8616, their calls: 155088,
our time: 5.065386, their time: 86.265423
t: 8, k: 6, n: 7,
our calls:: 10056, their calls: 211176,
our time: 7.158496, their time: 138.914658
t: 8, k: 6, n: 8,
our calls:: 11496, their calls: 275904,
our time: 8.828819, their time: 206.792314
t: 8, k: 6, n: 9,
our calls:: 12936, their calls: 349272,
our time: 11.574784, their time: 293.159470
t: 8, k: 6, n: 10,
our calls:: 14376, their calls: 431280,
our time: 16.348717, their time: 401.596867
t: 8, k: 7, n: 1,
our calls:: 1632, their calls: 4896,
our time: 0.235952, their time: 0.548783
t: 8, k: 7, n: 2,
our calls:: 3288, their calls: 19728,
our time: 0.800984, their time: 4.211165
t: 8, k: 7, n: 3,
our calls:: 4944, their calls: 44496,
our time: 1.740812, their time: 14.154516
t: 8, k: 7, n: 4,
our calls:: 6600, their calls: 79200,
our time: 2.950809, their time: 35.460398
t: 8, k: 7, n: 5,
our calls:: 8256, their calls: 123840,
our time: 4.795808, their time: 66.856235
t: 8, k: 7, n: 6,
our calls:: 9912, their calls: 178416,
our time: 6.877698, their time: 116.296695
t: 8, k: 7, n: 7,
our calls:: 11568, their calls: 242928,
our time: 9.130456, their time: 123.515209
t: 8, k: 7, n: 8,
our calls:: 13224, their calls: 317376,
our time: 7.277042, their time: 235.819719
t: 8, k: 7, n: 9,
our calls:: 14880, their calls: 401760,
our time: 15.990456, their time: 322.070416
t: 8, k: 7, n: 10,
our calls:: 16536, their calls: 496080,
our time: 11.294297, their time: 455.506137
t: 8, k: 8, n: 1,
our calls:: 1848, their calls: 5544,
our time: 0.203297, their time: 0.471318
t: 8, k: 8, n: 2,
our calls:: 3720, their calls: 22320,
our time: 0.671610, their time: 3.560565
t: 8, k: 8, n: 3,
our calls:: 5592, their calls: 50328,
our time: 1.430370, their time: 11.790660
t: 8, k: 8, n: 4,
our calls:: 7464, their calls: 89568,
our time: 2.551409, their time: 30.397090
t: 8, k: 8, n: 5,
our calls:: 9336, their calls: 140040,
our time: 3.929208, their time: 57.470076
t: 8, k: 8, n: 6,
our calls:: 11208, their calls: 201744,
our time: 6.065581, their time: 98.009829
t: 8, k: 8, n: 7,
our calls:: 13080, their calls: 274680,
our time: 10.392704, their time: 156.832545
```

```
t: 8, k: 8, n: 8,
our calls:: 14952, their calls: 358848,
our time: 10.630133, their time: 247.198317
t: 8, k: 8, n: 9,
our calls:: 16824, their calls: 454248,
our time: 12.730347, their time: 336.874133
t: 8, k: 8, n: 10,
our calls:: 18696, their calls: 560880,
our time: 16.145833, their time: 461.631723
t: 8, k: 9, n: 1,
our calls:: 2064, their calls: 6192,
our time: 0.236115, their time: 0.599634
t: 8, k: 9, n: 2,
our calls:: 4152, their calls: 24912,
our time: 0.843870, their time: 4.608871
t: 8, k: 9, n: 3,
our calls:: 6240, their calls: 56160,
our time: 1.833394, their time: 18.270781
t: 8, k: 9, n: 4,
our calls:: 8328, their calls: 99936,
our time: 3.261369, their time: 34.700666
t: 8, k: 9, n: 5,
our calls:: 10416, their calls: 156240,
our time: 4.826285, their time: 71.075413
t: 8, k: 9, n: 6,
our calls:: 12504, their calls: 225072,
our time: 10.087073, their time: 125.244472
t: 8, k: 9, n: 7,
our calls:: 14592, their calls: 306432,
our time: 9.715498, their time: 203.842886
t: 8, k: 9, n: 8,
our calls:: 16680, their calls: 400320,
our time: 13.529401, their time: 301.282938
t: 8, k: 9, n: 9,
our calls:: 18768, their calls: 506736,
our time: 16.841659, their time: 430.908943
t: 8, k: 9, n: 10,
our calls:: 20856, their calls: 625680,
our time: 20.157538, their time: 615.295795
t: 8, k: 10, n: 1,
our calls:: 2280, their calls: 6840,
our time: 3.133406, their time: 0.739294
t: 8, k: 10, n: 2,
our calls:: 4584, their calls: 27504,
our time: 1.055881, their time: 5.493435
t: 8, k: 10, n: 3,
our calls:: 6888, their calls: 61992,
our time: 2.193179, their time: 18.555472
t: 8, k: 10, n: 4,
our calls:: 9192, their calls: 110304,
our time: 3.843619, their time: 46.824420
t: 8, k: 10, n: 5,
our calls:: 11496, their calls: 172440,
our time: 6.186951, their time: 88.490002
t: 8, k: 10, n: 6,
our calls:: 13800, their calls: 248400,
our time: 8.916073, their time: 155.045540
t: 8, k: 10, n: 7,
our calls:: 16104, their calls: 338184,
our time: 11.691812, their time: 247.615032
t: 8, k: 10, n: 8,
our calls:: 18408, their calls: 441792,
our time: 16.091603, their time: 366.583614
t: 8, k: 10, n: 9,
our calls:: 20712, their calls: 559224,
our time: 20.518973, their time: 517.101314
t: 8, k: 10, n: 10,
our calls:: 23016, their calls: 690480,
our time: 24.089433, their time: 732.795608
t: 9, k: 1, n: 1,
our calls:: 378, their calls: 1134,
our time: 0.035264, their time: 0.018562
t: 9, k: 1, n: 2,
```

```
our calls:: 783, their calls: 4698,
our time: 0.035027, their time: 0.138827
t: 9, k: 1, n: 3,
our calls:: 1188, their calls: 10692,
our time: 0.075160, their time: 0.463342
t: 9, k: 1, n: 4,
our calls:: 1593, their calls: 19116,
our time: 0.129883, their time: 1.097830
t: 9, k: 1, n: 5,
our calls:: 1998, their calls: 29970,
our time: 0.199594, their time: 2.160203
t: 9, k: 1, n: 6,
our calls:: 2403, their calls: 43254,
our time: 0.281018, their time: 6.802099
t: 9, k: 1, n: 7,
our calls:: 2808, their calls: 58968,
our time: 0.373761, their time: 5.884519
t: 9, k: 1, n: 8,
our calls:: 3213, their calls: 77112,
our time: 0.485265, their time: 8.648171
t: 9, k: 1, n: 9,
our calls:: 3618, their calls: 97686,
our time: 0.604205, their time: 15.272735
t: 9, k: 1, n: 10,
our calls:: 4023, their calls: 120690,
our time: 0.745272, their time: 16.942541
t: 9, k: 2, n: 1,
our calls:: 621, their calls: 1863,
our time: 0.028840, their time: 0.051643
t: 9, k: 2, n: 2,
our calls:: 1269, their calls: 7614,
our time: 0.091211, their time: 3.250300
t: 9, k: 2, n: 3,
our calls:: 1917, their calls: 17253,
our time: 0.199701, their time: 1.370298
t: 9, k: 2, n: 4,
our calls:: 2565, their calls: 30780,
our time: 0.352391, their time: 3.119753
t: 9, k: 2, n: 5,
our calls:: 3213, their calls: 48195,
our time: 0.483691, their time: 5.976877
t: 9, k: 2, n: 6,
our calls:: 3861, their calls: 69498,
our time: 0.708990, their time: 10.282972
t: 9, k: 2, n: 7,
our calls:: 4509, their calls: 94689,
our time: 3.761357, their time: 16.318536
t: 9, k: 2, n: 8,
our calls:: 5157, their calls: 123768,
our time: 1.213265, their time: 27.537168
t: 9, k: 2, n: 9,
our calls:: 5805, their calls: 156735,
our time: 1.566298, their time: 39.469306
t: 9, k: 2, n: 10,
our calls:: 6453, their calls: 193590,
our time: 2.037821, their time: 49.490935
t: 9, k: 3, n: 1,
our calls:: 864, their calls: 2592,
our time: 0.053974, their time: 0.100978
t: 9, k: 3, n: 2,
our calls:: 1755, their calls: 10530,
our time: 0.160840, their time: 0.772492
t: 9, k: 3, n: 3,
our calls:: 2646, their calls: 23814,
our time: 0.344474, their time: 2.572609
t: 9, k: 3, n: 4,
our calls:: 3537, their calls: 42444,
our time: 0.591794, their time: 6.016165
t: 9, k: 3, n: 5,
our calls:: 4428, their calls: 66420,
our time: 0.916564, their time: 14.723963
t: 9, k: 3, n: 6,
our calls:: 5319, their calls: 95742,
```

```
our time: 1.306057, their time: 20.027947
t: 9, k: 3, n: 7,
our calls:: 6210, their calls: 130410,
our time: 1.732646, their time: 34.358167
t: 9, k: 3, n: 8,
our calls:: 7101, their calls: 170424,
our time: 2.351203, their time: 51.795034
t: 9, k: 3, n: 9,
our calls:: 7992, their calls: 215784,
our time: 2.848599, their time: 69.367013
t: 9, k: 3, n: 10,
our calls:: 8883, their calls: 266490,
our time: 3.373445, their time: 99.504631
t: 9, k: 4, n: 1,
our calls:: 1107, their calls: 3321,
our time: 0.078549, their time: 0.163920
t: 9, k: 4, n: 2,
our calls:: 2241, their calls: 13446,
our time: 0.251677, their time: 1.271335
t: 9, k: 4, n: 3,
our calls:: 3375, their calls: 30375,
our time: 0.540983, their time: 4.171967
t: 9, k: 4, n: 4,
our calls:: 4509, their calls: 54108,
our time: 0.984990, their time: 13.002130
t: 9, k: 4, n: 5,
our calls:: 5643, their calls: 84645,
our time: 1.468985, their time: 19.342696
t: 9, k: 4, n: 6,
our calls:: 6777, their calls: 121986,
our time: 2.068933, their time: 36.412820
t: 9, k: 4, n: 7,
our calls:: 7911, their calls: 166131,
our time: 2.717993, their time: 54.671236
t: 9, k: 4, n: 8,
our calls:: 9045, their calls: 217080,
our time: 3.643868, their time: 82.071496
t: 9, k: 4, n: 9,
our calls:: 10179, their calls: 274833,
our time: 4.585229, their time: 122.722142
t: 9, k: 4, n: 10,
our calls:: 11313, their calls: 339390,
our time: 6.062134, their time: 165.817288
t: 9, k: 5, n: 1,
our calls:: 1350, their calls: 4050,
our time: 0.109119, their time: 0.247483
t: 9, k: 5, n: 2,
our calls:: 2727, their calls: 16362,
our time: 0.371037, their time: 1.917925
t: 9, k: 5, n: 3,
our calls:: 4104, their calls: 36936,
our time: 0.787647, their time: 6.302989
t: 9, k: 5, n: 4,
our calls:: 5481, their calls: 65772,
our time: 1.375513, their time: 18.335212
t: 9, k: 5, n: 5,
our calls:: 6858, their calls: 102870,
our time: 2.131620, their time: 29.121136
t: 9, k: 5, n: 6,
our calls:: 8235, their calls: 148230,
our time: 3.045106, their time: 53.874781
t: 9, k: 5, n: 7,
our calls:: 9612, their calls: 201852,
our time: 4.260957, their time: 83.661310
t: 9, k: 5, n: 8,
our calls:: 10989, their calls: 263736,
our time: 5.570659, their time: 124.657217
t: 9, k: 5, n: 9,
our calls:: 12366, their calls: 333882,
our time: 7.005071, their time: 176.559224
t: 9, k: 5, n: 10,
our calls:: 13743, their calls: 412290,
our time: 8.255303, their time: 240.094111
```

```
t: 9, k: 6, n: 1,
our calls:: 1593, their calls: 4779,
our time: 0.145110, their time: 0.348470
t: 9, k: 6, n: 2,
our calls:: 3213, their calls: 19278,
our time: 0.501111, their time: 5.757291
t: 9, k: 6, n: 3,
our calls:: 4833, their calls: 43497,
our time: 1.097443, their time: 9.002280
t: 9, k: 6, n: 4,
our calls:: 6453, their calls: 77436,
our time: 1.895108, their time: 20.722239
t: 9, k: 6, n: 5,
our calls:: 8073, their calls: 121095,
our time: 3.075549, their time: 43.840309
t: 9, k: 6, n: 6,
our calls:: 9693, their calls: 174474,
our time: 4.361295, their time: 73.682898
t: 9, k: 6, n: 7,
our calls:: 11313, their calls: 237573,
our time: 5.863442, their time: 116.603657
t: 9, k: 6, n: 8,
our calls:: 12933, their calls: 310392,
our time: 10.616133, their time: 168.582330
t: 9, k: 6, n: 9,
our calls:: 14553, their calls: 392931,
our time: 9.388467, their time: 248.035466
t: 9, k: 6, n: 10,
our calls:: 16173, their calls: 485190,
our time: 12.237038, their time: 343.145443
t: 9, k: 7, n: 1,
our calls:: 1836, their calls: 5508,
our time: 0.183922, their time: 0.500082
t: 9, k: 7, n: 2,
our calls:: 3699, their calls: 22194,
our time: 0.675067, their time: 7.302830
t: 9, k: 7, n: 3,
our calls:: 5562, their calls: 50058,
our time: 1.569143, their time: 13.416812
t: 9, k: 7, n: 4,
our calls:: 7425, their calls: 89100,
our time: 2.822487, their time: 29.199256
t: 9, k: 7, n: 5,
our calls:: 9288, their calls: 139320,
our time: 3.971369, their time: 58.657360
t: 9, k: 7, n: 6,
our calls:: 11151, their calls: 200718,
our time: 5.603670, their time: 97.666037
t: 9, k: 7, n: 7,
our calls:: 13014, their calls: 273294,
our time: 10.970446, their time: 159.374140
t: 9, k: 7, n: 8,
our calls:: 14877, their calls: 357048,
our time: 10.093523, their time: 242.983459
t: 9, k: 7, n: 9,
our calls:: 16740, their calls: 451980,
our time: 13.420788, their time: 332.632698
t: 9, k: 7, n: 10,
our calls:: 18603, their calls: 558090,
our time: 16.471485, their time: 475.246748
t: 9, k: 8, n: 1,
our calls:: 2079, their calls: 6237,
our time: 0.241530, their time: 0.603261
t: 9, k: 8, n: 2,
our calls:: 4185, their calls: 25110,
our time: 0.829892, their time: 4.709313
t: 9, k: 8, n: 3,
our calls:: 6291, their calls: 56619,
our time: 1.934618, their time: 16.018649
t: 9, k: 8, n: 4,
our calls:: 8397, their calls: 100764,
our time: 3.402746, their time: 41.114667
t: 9, k: 8, n: 5,
```

```
our calls:: 10503, their calls: 157545,
our time: 5.072479, their time: 75.832640
t: 9, k: 8, n: 6,
our calls:: 12609, their calls: 226962,
our time: 7.954519, their time: 131.334090
t: 9, k: 8, n: 7,
our calls:: 14715, their calls: 309015,
our time: 9.672155, their time: 208.817220
t: 9, k: 8, n: 8,
our calls:: 16821, their calls: 403704,
our time: 13.668368, their time: 307.302498
t: 9, k: 8, n: 9,
our calls:: 18927, their calls: 511029,
our time: 16.739509, their time: 451.221900
t: 9, k: 8, n: 10,
our calls:: 21033, their calls: 630990,
our time: 21.535238, their time: 637.058247
t: 9, k: 9, n: 1,
our calls:: 2322, their calls: 6966,
our time: 0.311084, their time: 0.774083
t: 9, k: 9, n: 2,
our calls:: 4671, their calls: 28026,
our time: 1.080265, their time: 5.952658
t: 9, k: 9, n: 3,
our calls:: 7020, their calls: 63180,
our time: 2.377834, their time: 20.104247
t: 9, k: 9, n: 4,
our calls:: 9369, their calls: 112428,
our time: 4.417029, their time: 52.868877
t: 9, k: 9, n: 5,
our calls:: 11718, their calls: 175770,
our time: 6.814485, their time: 98.548477
t: 9, k: 9, n: 6,
our calls:: 14067, their calls: 253206,
our time: 9.984275, their time: 166.219933
t: 9, k: 9, n: 7,
our calls:: 16416, their calls: 344736,
our time: 13.902776, their time: 263.558784
t: 9, k: 9, n: 8,
our calls:: 18765, their calls: 450360,
our time: 16.550717, their time: 408.379764
t: 9, k: 9, n: 9,
our calls:: 21114, their calls: 570078,
our time: 21.351389, their time: 558.888646
t: 9, k: 9, n: 10,
our calls:: 23463, their calls: 703890,
our time: 25.603257, their time: 762.589765
t: 9, k: 10, n: 1,
our calls:: 2565, their calls: 7695,
our time: 0.354864, their time: 5.399044
t: 9, k: 10, n: 2,
our calls:: 5157, their calls: 30942,
our time: 1.353969, their time: 7.542347
t: 9, k: 10, n: 3,
our calls:: 7749, their calls: 69741,
our time: 2.980193, their time: 25.485969
t: 9, k: 10, n: 4,
our calls:: 10341, their calls: 124092,
our time: 5.243526, their time: 64.322606
t: 9, k: 10, n: 5,
our calls:: 12933, their calls: 193995,
our time: 8.287443, their time: 115.042919
t: 9, k: 10, n: 6,
our calls:: 15525, their calls: 279450,
our time: 11.224971, their time: 204.965355
t: 9, k: 10, n: 7,
our calls:: 18117, their calls: 380457,
our time: 19.197281, their time: 317.426751
t: 9, k: 10, n: 8,
our calls:: 20709, their calls: 497016,
our time: 24.139612, their time: 471.371197
t: 9, k: 10, n: 9,
our calls:: 23301, their calls: 629127,
```

```
our time: 27.066385, their time: 725.349442
t: 9, k: 10, n: 10,
our calls:: 25893, their calls: 776790,
our time: 37.079586, their time: 941.580991
t: 10, k: 1, n: 1,
our calls:: 420, their calls: 1260,
our time: 0.025949, their time: 0.022452
t: 10, k: 1, n: 2,
our calls:: 870, their calls: 5220,
our time: 0.042620, their time: 0.175350
t: 10, k: 1, n: 3,
our calls:: 1320, their calls: 11880,
our time: 0.090772, their time: 0.580097
t: 10, k: 1, n: 4,
our calls:: 1770, their calls: 21240,
our time: 0.155097, their time: 1.376944
t: 10, k: 1, n: 5,
our calls:: 2220, their calls: 33300,
our time: 0.239772, their time: 6.787543
t: 10, k: 1, n: 6,
our calls:: 2670, their calls: 48060,
our time: 0.342771, their time: 4.683401
t: 10, k: 1, n: 7,
our calls:: 3120, their calls: 65520,
our time: 0.460600, their time: 7.403778
t: 10, k: 1, n: 8,
our calls:: 3570, their calls: 85680,
our time: 0.592915, their time: 11.008415
t: 10, k: 1, n: 9,
our calls:: 4020, their calls: 108540,
our time: 4.633119, their time: 15.774362
t: 10, k: 1, n: 10,
our calls:: 4470, their calls: 134100,
our time: 0.923233, their time: 25.418550
t: 10, k: 2, n: 1,
our calls:: 690, their calls: 2070,
our time: 0.037875, their time: 0.067591
t: 10, k: 2, n: 2,
our calls:: 1410, their calls: 8460,
our time: 0.112104, their time: 0.514251
t: 10, k: 2, n: 3,
our calls:: 2130, their calls: 19170,
our time: 0.239592, their time: 1.712266
t: 10, k: 2, n: 4,
our calls:: 2850, their calls: 34200,
our time: 0.408156, their time: 4.030631
t: 10, k: 2, n: 5,
our calls:: 3570, their calls: 53550,
our time: 0.627734, their time: 7.509312
t: 10, k: 2, n: 6,
our calls:: 4290, their calls: 77220,
our time: 0.809472, their time: 12.127689
t: 10, k: 2, n: 7,
our calls:: 5010, their calls: 105210,
our time: 5.139084, their time: 19.651154
t: 10, k: 2, n: 8,
our calls:: 5730, their calls: 137520,
our time: 1.435491, their time: 33.329423
t: 10, k: 2, n: 9,
our calls:: 6450, their calls: 174150,
our time: 1.838541, their time: 43.363687
t: 10, k: 2, n: 10,
our calls:: 7170, their calls: 215100,
our time: 6.346992, their time: 59.419203
t: 10, k: 3, n: 1,
our calls:: 960, their calls: 2880,
our time: 0.062874, their time: 0.124015
t: 10, k: 3, n: 2,
our calls:: 1950, their calls: 11700,
our time: 0.200642, their time: 4.736789
t: 10, k: 3, n: 3,
our calls:: 2940, their calls: 26460,
our time: 0.431224, their time: 3.284541
```

```
t: 10, k: 3, n: 4,
our calls:: 3930, their calls: 47160,
our time: 0.749123, their time: 7.629368
t: 10, k: 3, n: 5,
our calls:: 4920, their calls: 73800,
our time: 1.150093, their time: 14.925725
t: 10, k: 3, n: 6,
our calls:: 5910, their calls: 106380,
our time: 1.685043, their time: 30.546709
t: 10, k: 3, n: 7,
our calls:: 6900, their calls: 144900,
our time: 2.158884, their time: 39.886519
t: 10, k: 3, n: 8,
our calls:: 7890, their calls: 189360,
our time: 6.814093, their time: 60.317675
t: 10, k: 3, n: 9,
our calls:: 8880, their calls: 239760,
our time: 3.656349, their time: 94.827924
t: 10, k: 3, n: 10,
our calls:: 9870, their calls: 296100,
our time: 4.232645, their time: 120.134981
t: 10, k: 4, n: 1,
our calls:: 1230, their calls: 3690,
our time: 0.095522, their time: 0.220041
t: 10, k: 4, n: 2,
our calls:: 2490, their calls: 14940,
our time: 0.331503, their time: 1.700221
t: 10, k: 4, n: 3,
our calls:: 3750, their calls: 33750,
our time: 0.715453, their time: 5.605293
t: 10, k: 4, n: 4,
our calls:: 5010, their calls: 60120,
our time: 1.232956, their time: 17.010208
t: 10, k: 4, n: 5,
our calls:: 6270, their calls: 94050,
our time: 1.910645, their time: 25.907287
t: 10, k: 4, n: 6,
our calls:: 7530, their calls: 135540,
our time: 2.756519, their time: 46.939836
t: 10, k: 4, n: 7,
our calls:: 8790, their calls: 184590,
our time: 3.520616, their time: 69.973629
t: 10, k: 4, n: 8,
our calls:: 10050, their calls: 241200,
our time: 4.415596, their time: 104.129142
t: 10, k: 4, n: 9,
our calls:: 11310, their calls: 305370,
our time: 5.712022, their time: 147.591113
t: 10, k: 4, n: 10,
our calls:: 12570, their calls: 377100,
our time: 7.015431, their time: 210.452152
t: 10, k: 5, n: 1,
our calls:: 1500, their calls: 4500,
our time: 0.125042, their time: 0.306219
t: 10, k: 5, n: 2,
our calls:: 3030, their calls: 18180,
our time: 0.450883, their time: 2.372871
t: 10, k: 5, n: 3,
our calls:: 4560, their calls: 41040,
our time: 0.982297, their time: 8.056619
t: 10, k: 5, n: 4,
our calls:: 6090, their calls: 73080,
our time: 1.698113, their time: 23.871644
t: 10, k: 5, n: 5,
our calls:: 7620, their calls: 114300,
our time: 2.791064, their time: 38.460329
t: 10, k: 5, n: 6,
our calls:: 9150, their calls: 164700,
our time: 4.021295, their time: 67.410015
t: 10, k: 5, n: 7,
our calls:: 10680, their calls: 224280,
our time: 5.115541, their time: 104.528710
t: 10, k: 5, n: 8,
```

```
our calls:: 12210, their calls: 293040,
our time: 6.725778, their time: 163.311248
t: 10, k: 5, n: 9,
our calls:: 13740, their calls: 370980,
our time: 8.856298, their time: 227.835352
t: 10, k: 5, n: 10,
our calls:: 15270, their calls: 458100,
our time: 14.909080, their time: 317.610144
t: 10, k: 6, n: 1,
our calls:: 1770, their calls: 5310,
our time: 0.166444, their time: 0.422031
t: 10, k: 6, n: 2,
our calls:: 3570, their calls: 21420,
our time: 0.602174, their time: 3.231008
t: 10, k: 6, n: 3,
our calls:: 5370, their calls: 48330,
our time: 1.308784, their time: 11.233654
t: 10, k: 6, n: 4,
our calls:: 7170, their calls: 86040,
our time: 2.293062, their time: 31.932520
t: 10, k: 6, n: 5,
our calls:: 8970, their calls: 134550,
our time: 3.708469, their time: 51.394009
t: 10, k: 6, n: 6,
our calls:: 10770, their calls: 193860,
our time: 5.220160, their time: 91.449101
t: 10, k: 6, n: 7,
our calls:: 12570, their calls: 263970,
our time: 7.136719, their time: 144.683829
t: 10, k: 6, n: 8,
our calls:: 14370, their calls: 344880,
our time: 9.452138, their time: 223.628029
t: 10, k: 6, n: 9,
our calls:: 16170, their calls: 436590,
our time: 13.193574, their time: 313.121020
t: 10, k: 6, n: 10,
our calls:: 17970, their calls: 539100,
our time: 15.671024, their time: 446.589269
t: 10, k: 7, n: 1,
our calls:: 2040, their calls: 6120,
our time: 0.233258, their time: 0.562355
t: 10, k: 7, n: 2,
our calls:: 4110, their calls: 24660,
our time: 0.799558, their time: 4.361495
t: 10, k: 7, n: 3,
our calls:: 6180, their calls: 55620,
our time: 1.814432, their time: 14.415522
t: 10, k: 7, n: 4,
our calls:: 8250, their calls: 99000,
our time: 3.071698, their time: 39.637418
t: 10, k: 7, n: 5,
our calls:: 10320, their calls: 154800,
our time: 5.015355, their time: 70.343551
t: 10, k: 7, n: 6,
our calls:: 12390, their calls: 223020,
our time: 7.342255, their time: 128.683669
t: 10, k: 7, n: 7,
our calls:: 14460, their calls: 303660,
our time: 10.160491, their time: 204.134228
t: 10, k: 7, n: 8,
our calls:: 16530, their calls: 396720,
our time: 12.602559, their time: 288.430900
t: 10, k: 7, n: 9,
our calls:: 18600, their calls: 502200,
our time: 16.307285, their time: 418.903243
t: 10, k: 7, n: 10,
our calls:: 20670, their calls: 620100,
our time: 20.719813, their time: 600.795990
t: 10, k: 8, n: 1,
our calls:: 2310, their calls: 6930,
our time: 0.305498, their time: 0.777930
t: 10, k: 8, n: 2,
our calls:: 4650, their calls: 27900,
```

```
our time: 1.096966, their time: 6.011972
t: 10, k: 8, n: 3,
our calls:: 6990, their calls: 62910,
our time: 2.400665, their time: 20.007811
t: 10, k: 8, n: 4,
our calls:: 9330, their calls: 111960,
our time: 4.044502, their time: 49.506679
t: 10, k: 8, n: 5,
our calls:: 11670, their calls: 175050,
our time: 6.161371, their time: 94.435006
t: 10, k: 8, n: 6,
our calls:: 14010, their calls: 252180,
our time: 8.994435, their time: 160.209029
t: 10, k: 8, n: 7,
our calls:: 16350, their calls: 343350,
our time: 12.464322, their time: 258.571395
t: 10, k: 8, n: 8,
our calls:: 18690, their calls: 448560,
our time: 21.734955, their time: 407.558710
t: 10, k: 8, n: 9,
our calls:: 21030, their calls: 567810,
our time: 28.487178, their time: 614.626688
t: 10, k: 8, n: 10,
our calls:: 23370, their calls: 701100,
our time: 28.961040, their time: 825.461302
t: 10, k: 9, n: 1,
our calls:: 2580, their calls: 7740,
our time: 0.359051, their time: 1.008296
t: 10, k: 9, n: 2,
our calls:: 5190, their calls: 31140,
our time: 1.549308, their time: 7.935515
t: 10, k: 9, n: 3,
our calls:: 7800, their calls: 70200,
our time: 3.180325, their time: 31.260314
t: 10, k: 9, n: 4,
our calls:: 10410, their calls: 124920,
our time: 5.526895, their time: 63.950530
t: 10, k: 9, n: 5,
our calls:: 13020, their calls: 195300,
our time: 8.729413, their time: 132.044827
t: 10, k: 9, n: 6,
our calls:: 15630, their calls: 281340,
our time: 12.709397, their time: 226.510999
t: 10, k: 9, n: 7,
our calls:: 18240, their calls: 383040,
our time: 17.206182, their time: 330.568460
t: 10, k: 9, n: 8,
our calls:: 20850, their calls: 500400,
our time: 20.114612, their time: 478.012492
t: 10, k: 9, n: 9,
our calls:: 23460, their calls: 633420,
our time: 26.066138, their time: 663.789369
t: 10, k: 9, n: 10,
our calls:: 26070, their calls: 782100,
our time: 34.595405, their time: 925.028892
t: 10, k: 10, n: 1,
our calls:: 2850, their calls: 8550,
our time: 0.452599, their time: 1.097809
t: 10, k: 10, n: 2,
our calls:: 5730, their calls: 34380,
our time: 1.519254, their time: 8.492330
t: 10, k: 10, n: 3,
our calls:: 8610, their calls: 77490,
our time: 3.359205, their time: 28.646532
t: 10, k: 10, n: 4,
our calls:: 11490, their calls: 137880,
our time: 6.123256, their time: 72.685005
t: 10, k: 10, n: 5,
our calls:: 14370, their calls: 215550,
our time: 9.799790, their time: 145.895774
t: 10, k: 10, n: 6,
our calls:: 17250, their calls: 310500,
our time: 13.905580, their time: 239.857377
```

```
t: 10, k: 10, n: 7,
our calls:: 20130, their calls: 422730,
our time: 19.657438, their time: 395.031667
t: 10, k: 10, n: 8,
our calls:: 23010, their calls: 552240,
our time: 25.489824, their time: 602.428832
t: 10, k: 10, n: 9,
our calls:: 25890, their calls: 699030,
our time: 32.609059, their time: 838.798629
t: 10, k: 10, n: 10,
our calls:: 28770, their calls: 863100,
our time: 40.787374, their time: 1160.345273
63796.823u 42.547s 17:50:37.47 99.3%  0+0k 24+0io 3pf+0w
>
```