Long Contest Editorial

November 13, 2015

Moscow International Workshop ACM ICPC, MIPT, 2015

A. Rabbit Lunch

Parts of the bipartite graph have sizes m and n, and each vertex has a degree upper limit. Construct a bipartite graph with maximal number of edges. Multiple edges are disallowed.

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.
Build a network as follows:

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.
Build a network as follows:

- Add edges from source to vertices of left half with capacity m_i

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.
Build a network as follows:

- Add edges from source to vertices of left half with capacity m_i
- Add all edges between left and right half with capacity 1

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.
Build a network as follows:

- Add edges from source to vertices of left half with capacity m_i
- Add all edges between left and right half with capacity 1
- Add edges from vertices of right half to sink with capacity n_i

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.
Build a network as follows:

- Add edges from source to vertices of left half with capacity m_i
- Add all edges between left and right half with capacity 1
- Add edges from vertices of right half to sink with capacity n_i

The answer is indeed the value of max-flow in this network.

A. Rabbit Lunch

One possible approach to this problem is to view it as a max-flow problem.
Build a network as follows:

- Add edges from source to vertices of left half with capacity m_i
- Add all edges between left and right half with capacity 1
- Add edges from vertices of right half to sink with capacity n_i

The answer is indeed the value of max-flow in this network.
Of course, we cannot build it explicitly since network is too large.

A. Rabbit Lunch

Due to Ford-Falkerson theorem, the value of max-flow is equal to the value of the minimal $S - T$ cut.

A. Rabbit Lunch

Due to Ford-Falkerson theorem, the value of max-flow is equal to the value of the minimal $S - T$ cut.
Let source's part of the cut contain a vertices of left half with total capacity A, and $n - b$ vertices of right half with total capacity B.

A. Rabbit Lunch

Due to Ford-Falkerson theorem, the value of max-flow is equal to the value of the minimal $S - T$ cut.
Let source's part of the cut contain a vertices of left half with total capacity A, and $n - b$ vertices of right half with total capacity B.
Similarly, let sink's part of the cut contain $m - a$ vertices of left half with total capacity C, and b vertices of right half with total capacity D.

A. Rabbit Lunch

Due to Ford-Falkerson theorem, the value of max-flow is equal to the value of the minimal $S - T$ cut.
Let source's part of the cut contain a vertices of left half with total capacity A, and $n - b$ vertices of right half with total capacity B.
Similarly, let sink's part of the cut contain $m - a$ vertices of left half with total capacity C, and b vertices of right half with total capacity D.
Then, capacity of the cut if $B + C + a \cdot b$.

A. Rabbit Lunch

Due to Ford-Falkerson theorem, the value of max-flow is equal to the value of the minimal $S - T$ cut.
Let source's part of the cut contain a vertices of left half with total capacity A, and $n - b$ vertices of right half with total capacity B.
Similarly, let sink's part of the cut contain $m - a$ vertices of left half with total capacity C, and b vertices of right half with total capacity D.
Then, capacity of the cut if $B + C + a \cdot b$.
It is evident that B and C should be as small as possible, thus the corresponding parts should contain vertices of smallest capacities.

A. Rabbit Lunch

Due to Ford-Falkerson theorem, the value of max-flow is equal to the value of the minimal $S - T$ cut.

Let source's part of the cut contain a vertices of left half with total capacity A, and $n - b$ vertices of right half with total capacity B.

Similarly, let sink's part of the cut contain $m - a$ vertices of left half with total capacity C, and b vertices of right half with total capacity D.

Then, capacity of the cut if $B + C + a \cdot b$.

It is evident that B and C should be as small as possible, thus the corresponding parts should contain vertices of smallest capacities.

The above reasoning can be considered a proof of the greedy algorithm which matches the largest vertices of the left part with the smallest vertices of the right part.

A. Rabbit Lunch

The problem is now equivalent to the following: we have two sorted arrays a and b. We have to choose non-negative numbers x and y such that

$$\sum_{i=1}^{x} a_i + \sum_{j=1}^{y} b_j + (m-x)\cdot(n-y)$$

is minimal.

A. Rabbit Lunch

The problem is now equivalent to the following: we have two sorted arrays a and b. We have to choose non-negative numbers x and y such that

$$\sum_{i=1}^{x} a_i + \sum_{j=1}^{y} b_j + (m - x) \cdot (n - y)$$

is minimal.

Suppose x is fixed.

A. Rabbit Lunch

The problem is now equivalent to the following: we have two sorted arrays a and b. We have to choose non-negative numbers x and y such that

$$\sum_{i=1}^{x} a_i + \sum_{j=1}^{y} b_j + (m - x) \cdot (n - y)$$

is minimal.
Suppose x is fixed.
Changing y to $y - 1$ changes answer by $m - x - b_y$.

A. Rabbit Lunch

The problem is now equivalent to the following: we have two sorted arrays a and b. We have to choose non-negative numbers x and y such that

$$\sum_{i=1}^{x} a_i + \sum_{j=1}^{y} b_j + (m - x) \cdot (n - y)$$

is minimal.

Suppose x is fixed.

Changing y to $y - 1$ changes answer by $m - x - b_y$. Since b is sorted, y should be decreased while the answer gets better.

A. Rabbit Lunch

The problem is now equivalent to the following: we have two sorted arrays a and b. We have to choose non-negative numbers x and y such that

$$\sum_{i=1}^{x} a_i + \sum_{j=1}^{y} b_j + (m - x) \cdot (n - y)$$

is minimal.

Suppose x is fixed.

Changing y to $y - 1$ changes answer by $m - x - b_y$. Since b is sorted, y should be decreased while the answer gets better.

Moreover, if we increase x, optimal y cannot increase. That means that "two-pointers" approach will work here: increase x, decrease y while answer gets better.

A. Rabbit Lunch

The problem is now equivalent to the following: we have two sorted arrays a and b. We have to choose non-negative numbers x and y such that

$$\sum_{i=1}^{x} a_i + \sum_{j=1}^{y} b_j + (m - x) \cdot (n - y)$$

is minimal.

Suppose x is fixed.

Changing y to $y - 1$ changes answer by $m - x - b_y$. Since b is sorted, y should be decreased while the answer gets better.

Moreover, if we increase x, optimal y cannot increase. That means that "two-pointers" approach will work here: increase x, decrease y while answer gets better.

Together with sorting the arrays, this makes for an $O((n + m) \log(n + m))$ solution.

B. Snuke

Given a string, remove exactly k letters “s” so that the resulting string is lexicographically smallest possible.

B. Snuke

If there are several "s" letters such that their removal makes the string smaller, we should remove them going from left to right. If at some point we removed exactly k letters, we have to stop.

B. Snuke

If there are several "$\mathtt{s}$" letters such that their removal makes the string smaller, we should remove them going from left to right. If at some point we removed exactly k letters, we have to stop.
If there are less then k letters $\mathtt{s}$ that make our string better, we have to remove some more letters. Note that at this point each removal makes string *worse*, so we have to pick rightmost letters over others. Thus, go from right to left and remove all letters "$\mathtt{s}$" until we have removed enough.

B. Snuke

If there are several "s" letters such that their removal makes the string smaller, we should remove them going from left to right. If at some point we removed exactly k letters, we have to stop.
If there are less then k letters s that make our string better, we have to remove some more letters. Note that at this point each removal makes string *worse*, so we have to pick rightmost letters over others. Thus, go from right to left and remove all letters "s" until we have removed enough.
This is easily implemented in $O(n)$.

C. Supermarket

n types of food are sold at the market. Each month, each type is either present or not. We have ordered all types of food from most to least favourite, and each month we pick most favourite food among available types. Find maximal number of food types we can purchase over 12 months, if we can choose preferences arbitrarily.

C. Supermarket

Given the preferences, we can find the number of different types we purchase as follows:

C. Supermarket

Given the preferences, we can find the number of different types we purchase as follows:

- Pick the most favourable food. If it is not sold at all, skip it. Otherwise, increase answer by 1 and mark all months when it's sold as "decided".

C. Supermarket

Given the preferences, we can find the number of different types we purchase as follows:

- Pick the most favourable food. If it is not sold at all, skip it. Otherwise, increase answer by 1 and mark all months when it's sold as "decided".
- Pick the next favourable food. If it is not sold, or all the months when it's sold are already "decided", then we can't buy it and should skip. Otherwise, increase the counter and mark the months when the current type is sold as "decided".

C. Supermarket

Given the preferences, we can find the number of different types we purchase as follows:

- Pick the most favourable food. If it is not sold at all, skip it. Otherwise, increase answer by 1 and mark all months when it's sold as "decided".
- Pick the next favourable food. If it is not sold, or all the months when it's sold are already "decided", then we can't buy it and should skip. Otherwise, increase the counter and mark the months when the current type is sold as "decided".
- Do the same for the next favourable food, and so on.

C. Supermarket

Denote d_S the maximal number of food types that can be bought for some preference list such that the set of "decided" months becomes equal to S.

C. Supermarket

Denote d_S the maximal number of food types that can be bought for some preference list such that the set of "decided" months becomes equal to S.

We can compute d_S with DP. Clearly, the base case is $d_\varnothing = 0$.

C. Supermarket

Denote d_S the maximal number of food types that can be bought for some preference list such that the set of "decided" months becomes equal to S.

We can compute d_S with DP. Clearly, the base case is $d_\varnothing = 0$.

For every S, iterate over all food types that extend the set of decided months (that is, types that are sold on still undecided months).

C. Supermarket

Denote d_S the maximal number of food types that can be bought for some preference list such that the set of "decided" months becomes equal to S.

We can compute d_S with DP. Clearly, the base case is $d_\varnothing = 0$. For every S, iterate over all food types that extend the set of decided months (that is, types that are sold on still undecided months). Let S' be the resulting set of decided months after choosing the current food type. Then we try to improve $d_{S'}$ with $d_S + 1$.

C. Supermarket

Denote d_S the maximal number of food types that can be bought for some preference list such that the set of "decided" months becomes equal to S.

We can compute d_S with DP. Clearly, the base case is $d_\varnothing = 0$.

For every S, iterate over all food types that extend the set of decided months (that is, types that are sold on still undecided months). Let S' be the resulting set of decided months after choosing the current food type. Then we try to improve $d_{S'}$ with $d_S + 1$.

Finally, the answer is the maximal value of d_S for all S.

C. Supermarket

Denote d_S the maximal number of food types that can be bought for some preference list such that the set of "decided" months becomes equal to S.

We can compute d_S with DP. Clearly, the base case is $d_\varnothing = 0$.

For every S, iterate over all food types that extend the set of decided months (that is, types that are sold on still undecided months). Let S' be the resulting set of decided months after choosing the current food type. Then we try to improve $d_{S'}$ with $d_S + 1$.

Finally, the answer is the maximal value of d_S for all S. The total complexity is $O(n2^k)$, where k is the number of months (12 in our case).

D. Subsequence

Count the number of pairs of string (s, t) such that:

- s consists of a zeros and b ones
- t consists of c zeros and d ones
- t is a subsequence of s

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.

Let i be the index of first unmatched symbol of t.

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.
Let i be the index of first unmatched symbol of t. Iterate over all symbols of s, increment i if t_i is equal to current sumbol of s.

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.
Let i be the index of first unmatched symbol of t. Iterate over all symbols of s, increment i if t_i is equal to current sumbol of s.
t is a subsequence of s if at the end of the process index i points after the last symbol of t.

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.
Let i be the index of first unmatched symbol of t. Iterate over all symbols of s, increment i if t_i is equal to current sumbol of s.
t is a subsequence of s if at the end of the process index i points after the last symbol of t.

Denote $\overline{c}$ symbol different from c ($\overline{0} = 1$, $\overline{1} = 0$).

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.
Let i be the index of first unmatched symbol of t. Iterate over all symbols of s, increment i if t_i is equal to current sumbol of s.
t is a subsequence of s if at the end of the process index i points after the last symbol of t.

Denote $\overline{c}$ symbol different from c ($\overline{0} = 1$, $\overline{1} = 0$).
If t is a subsequence of s, then s can be uniquely represented as follows (according to the algorithm above):

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.
Let i be the index of first unmatched symbol of t. Iterate over all symbols of s, increment i if t_i is equal to current sumbol of s.
t is a subsequence of s if at the end of the process index i points after the last symbol of t.

Denote $\overline{c}$ symbol different from c ($\overline{0} = 1$, $\overline{1} = 0$).
If t is a subsequence of s, then s can be uniquely represented as follows (according to the algorithm above):

$$\overline{t_1}^* t_1 \overline{t_2}^* t_2 \ldots \overline{t_n}^* t_n (whatever)$$

(here c^* is c repeated any number of times (possibly, zero))

D. Subsequence

Checking if a string is a subsequence of another string

Let us decide if t is a substring of s.
Let i be the index of first unmatched symbol of t. Iterate over all symbols of s, increment i if t_i is equal to current sumbol of s.
t is a subsequence of s if at the end of the process index i points after the last symbol of t.

Denote $\overline{c}$ symbol different from c ($\overline{0} = 1$, $\overline{1} = 0$).
If t is a subsequence of s, then s can be uniquely represented as follows (according to the algorithm above):

$$\overline{t_1}^* t_1 \overline{t_2}^* t_2 \ldots \overline{t_n}^* t_n (whatever)$$

(here c^* is c repeated any number of times (possibly, zero))
We will call the substring $\overline{t_i}^* t_i$ the *i-th block*.

D. Subsequence

Assume that t is fixed.

D. Subsequence

Assume that t is fixed.
Let k be the total number of 1's in the blocks corresponding to symbols $t_i = 0$, and l be the total number of 0's in the blocks corresponding to symbols $t_i = 1$.

D. Subsequence

Assume that t is fixed.

Let k be the total number of 1's in the blocks corresponding to symbols $t_i = 0$, and l be the total number of 0's in the blocks corresponding to symbols $t_i = 1$. The number of strings s corresponding to the numbers k and l is:

D. Subsequence

Assume that t is fixed.
Let k be the total number of 1's in the blocks corresponding to symbols $t_i = 0$, and l be the total number of 0's in the blocks corresponding to symbols $t_i = 1$. The number of strings s corresponding to the numbers k and l is:

$$\binom{k+d-1}{d-1} \text{ (distribute } k \text{ ones over } d \text{ blocks)}$$

D. Subsequence

Assume that t is fixed.
Let k be the total number of 1's in the blocks corresponding to symbols $t_i = 0$, and l be the total number of 0's in the blocks corresponding to symbols $t_i = 1$. The number of strings s corresponding to the numbers k and l is:

$$\binom{k+d-1}{d-1} \text{ (distribute } k \text{ ones over } d \text{ blocks)}$$
$$\times \binom{l+c-1}{c-1} \text{ (distribute } l \text{ zeros over } c \text{ blocks)}$$

D. Subsequence

Assume that t is fixed.
Let k be the total number of 1's in the blocks corresponding to symbols $t_i = 0$, and l be the total number of 0's in the blocks corresponding to symbols $t_i = 1$. The number of strings s corresponding to the numbers k and l is:

$$\begin{gathered}\binom{k+d-1}{d-1} \text{ (distribute } k \text{ ones over } d \text{ blocks)}\\ \times\binom{l+c-1}{c-1} \text{ (distribute } l \text{ zeros over } c \text{ blocks)}\\ \times\binom{a-c-k+b-d-l}{a-c-k} \text{ (shuffle all unused symbols arbitrarily)}\end{gathered}$$

D. Subsequence

Assume that t is fixed.
Let k be the total number of 1's in the blocks corresponding to symbols $t_i = 0$, and l be the total number of 0's in the blocks corresponding to symbols $t_i = 1$. The number of strings s corresponding to the numbers k and l is:

$$\binom{k+d-1}{d-1} \text{ (distribute } k \text{ ones over } d \text{ blocks)}$$
$$\times \binom{l+c-1}{c-1} \text{ (distribute } l \text{ zeros over } c \text{ blocks)}$$
$$\times \binom{a-c-k+b-d-l}{a-c-k} \text{ (shuffle all unused symbols arbitrarily)}$$

The number of s — supersequences of t is the sum of these products over all valid values of k and l.

D. Subsequence

It is evident that the number of s's does not depend on t, so we can simply multiply the answer by $\binom{c+d}{c}$.

D. Subsequence

It is evident that the number of s's does not depend on t, so we can simply multiply the answer by $\binom{c+d}{c}$.
The complexity is $O(ab)$.

E. Tournament

Count the total number of strongly connected components over all tournaments on n vertices.

E. Tournament

Fact

The condensation of any tournament is a path.

E. Tournament

Fact

The condensation of any tournament is a path.

Let S, T be any partition of the set of vertices V into two non-empty parts.

E. Tournament

Fact

The condensation of any tournament is a path.

Let S, T be any partition of the set of vertices V into two non-empty parts.
Call S, T a *one-directional cut* if all edges between S and T are directed towards T.

E. Tournament

Fact

The condensation of any tournament is a path.

Let S, T be any partition of the set of vertices V into two non-empty parts.
Call S, T a *one-directional cut* if all edges between S and T are directed towards T.

Corollary of the fact

The number of SCC's of a tournament is the total number of one-directional cuts plus one.

E. Tournament

Fact

The condensation of any tournament is a path.

Let S, T be any partition of the set of vertices V into two non-empty parts.
Call S, T a *one-directional cut* if all edges between S and T are directed towards T.

Corollary of the fact

The number of SCC's of a tournament is the total number of one-directional cuts plus one.

Thus, the answer is the total number of one-directional cuts over all graphs, plus the total number of tournaments ($2^{\frac{n(n-1)}{2}}$).

E. Tournament

How many one-directional cuts are there in total?

E. Tournament

How many one-directional cuts are there in total?
The total number of one-direcational cuts such that $|S| = k$ is

E. Tournament

How many one-directional cuts are there in total?
The total number of one-direcational cuts such that $|S| = k$ is

$$\binom{n}{k} \text{ (choose partition into } S \text{ and } T\text{)}$$

E. Tournament

How many one-directional cuts are there in total?
The total number of one-direcational cuts such that $|S| = k$ is

$$\binom{n}{k} \text{ (choose partition into } S \text{ and } T\text{)}$$
$$\times 2^{\frac{k(k-1)}{2}+\frac{(n-k)(n-k-1)}{2}} \text{ (edges between } S \text{ and } T \text{ are directed towards } T\text{, direct all other edges arbitrarily)}$$

E. Tournament

How many one-directional cuts are there in total?
The total number of one-direcational cuts such that $|S| = k$ is

$$\binom{n}{k} \text{ (choose partition into } S \text{ and } T)$$
$$\times 2^{\frac{k(k-1)}{2}+\frac{(n-k)(n-k-1)}{2}} \text{ (edges between } S \text{ and } T \text{ are directed towards } T\text{, direct all other edges arbitrarily)}$$

To obtain the answer, sum these products over all k, and add $2^{\frac{n(n-1)}{2}}$.

E. Tournament

How many one-directional cuts are there in total?
The total number of one-direcational cuts such that $|S| = k$ is

$$\binom{n}{k} \text{ (choose partition into } S \text{ and } T)$$
$$\times 2^{\frac{k(k-1)}{2} + \frac{(n-k)(n-k-1)}{2}} \text{ (edges between } S \text{ and } T \text{ are directed towards } T\text{, direct all other edges arbitrarily)}$$

To obtain the answer, sum these products over all k, and add $2^{\frac{n(n-1)}{2}}$.
The complexity is $O(n \log n)$ (to find powers of 2 on each step).

F. Lake

There are n ports on the border of a circular lake. We can walk around the lake, or move from one port to another immediately. Add k new ports in such a way that maximal length of shortest path between any two points is minimized.

F. Lake

Let us restate the problem:

F. Lake

Let us restate the problem:

- We are given a set of segments. We can move along the segment or instantly jump between any two ends of any segments.

F. Lake

Let us restate the problem:

- We are given a set of segments. We can move along the segment or instantly jump between any two ends of any segments.
- For k times we can choose any segment and break it into two parts arbitrarily.

F. Lake

Let us restate the problem:

- We are given a set of segments. We can move along the segment or instantly jump between any two ends of any segments.
- For k times we can choose any segment and break it into two parts arbitrarily.
- It is evident that the largest distance between two points is equal to half of the sum of lengths of two longest segments.

F. Lake

Let us restate the problem:

- We are given a set of segments. We can move along the segment or instantly jump between any two ends of any segments.
- For k times we can choose any segment and break it into two parts arbitrarily.
- It is evident that the largest distance between two points is equal to half of the sum of lengths of two longest segments.

So, we have to break segments into parts and minimize the sum of two largest lengths.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.
  We assert that a is the smallest segment from the original set that is greater or equal to b.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.
  We assert that a is the smallest segment from the original set that is greater or equal to b.
  Indeed, if there is an original segment a' such that $a > a' \geqslant b$, we can replace a with a' and cut a the same way we cut a'.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.
  We assert that a is the smallest segment from the original set that is greater or equal to b.
  Indeed, if there is an original segment a' such that $a > a' \geqslant b$, we can replace a with a' and cut a the same way we cut a'.
  After that, the length of the second longest segment will not increase, therefore answer wasn't optimal.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.
  We assert that a is the smallest segment from the original set that is greater or equal to b.
  Indeed, if there is an original segment a' such that $a > a' \geqslant b$, we can replace a with a' and cut a the same way we cut a'.
  After that, the length of the second longest segment will not increase, therefore answer wasn't optimal.
  It follows that if we are given $s = a + b$, a has to be the smallest segment not shorter than $s/2$.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.
  We assert that a is the smallest segment from the original set that is greater or equal to b.
  Indeed, if there is an original segment a' such that $a > a' \geqslant b$, we can replace a with a' and cut a the same way we cut a'.
  After that, the length of the second longest segment will not increase, therefore answer wasn't optimal.
  It follows that if we are given $s = a + b$, a has to be the smallest segment not shorter than $s/2$.
- Suppose that a is a proper part of an original segment.

F. Lake

Let's take the optimal answer and look at two largest segments, a and b.

- Suppose that a is whole segment from original set.
  We assert that a is the smallest segment from the original set that is greater or equal to b.
  Indeed, if there is an original segment a' such that $a > a' \geqslant b$, we can replace a with a' and cut a the same way we cut a'.
  After that, the length of the second longest segment will not increase, therefore answer wasn't optimal.
  It follows that if we are given $s = a + b$, a has to be the smallest segment not shorter than $s/2$.
- Suppose that a is a proper part of an original segment.
  In that case we can make a and b equal (while preseving their sum) by contracting a and extending all other segments.

F. Lake

If we want to check whether an answer with $a + b$ at most x exists, we can check two options:

F. Lake

If we want to check whether an answer with $a + b$ at most x exists, we can check two options:

- Break all segments in such a way that all parts are not greater than $x/2$.

F. Lake

If we want to check whether an answer with $a + b$ at most x exists, we can check two options:

- Break all segments in such a way that all parts are not greater than $x/2$.
- Find the shortest segment a not shorter than $x/2$, than break all other segments so that all parts are not longer than $x - a$.

F. Lake

If we want to check whether an answer with $a+b$ at most x exists, we can check two options:

- Break all segments in such a way that all parts are not greater than $x/2$.
- Find the shortest segment a not shorter than $x/2$, than break all other segments so that all parts are not longer than $x-a$.

If any of the options produces partition with at most k cuts, then an answer with $a+b \geqslant x$ exists.

F. Lake

If we want to check whether an answer with $a+b$ at most x exists, we can check two options:

- Break all segments in such a way that all parts are not greater than $x/2$.
- Find the shortest segment a not shorter than $x/2$, than break all other segments so that all parts are not longer than $x-a$.

If any of the options produces partition with at most k cuts, then an answer with $a+b \geqslant x$ exists.
We can now invoke binary search on x to find the answer.

F. Lake

If we want to check whether an answer with $a + b$ at most x exists, we can check two options:

- Break all segments in such a way that all parts are not greater than $x/2$.
- Find the shortest segment a not shorter than $x/2$, than break all other segments so that all parts are not longer than $x - a$.

If any of the options produces partition with at most k cuts, then an answer with $a + b \geqslant x$ exists.
We can now invoke binary search on x to find the answer.
Complexity is $O(n \log^{-1} \varepsilon)$.

G. Medals

n people are participating in a competition. For some pairs of people we know that one performed better then the other. Count the number of ways to give gold, silver and bronze medals (that is, choose the best, the second best and the third best) according to given information.

G. Medals

Call a vertex *a source* if its in-degree is zero.

G. Medals

Call a vertex *a source* if its in-degree is zero.
Define *depth* $d(v)$ of a vertex v as follows:

G. Medals

Call a vertex *a source* if its in-degree is zero.
Define *depth* $d(v)$ of a vertex v as follows:

- Depth of a source is 0.

G. Medals

Call a vertex *a source* if its in-degree is zero.
Define *depth* $d(v)$ of a vertex v as follows:

- Depth of a source is 0.
- Depth of a non-source vertex is $\max_{(u,v)\text{ — edge}} d(u) + 1$.

G. Medals

Call a vertex *a source* if its in-degree is zero.
Define *depth* $d(v)$ of a vertex v as follows:

- Depth of a source is 0.
- Depth of a non-source vertex is $\max_{(u,v)\text{ — edge}} d(u) + 1$.

That is, vertices of depth 1 are dominated only by sources, vertices of depth 2 are dominated by sources and 1-depth vertices, and so on.

G. Medals

Call a vertex *a source* if its in-degree is zero.
Define *depth* $d(v)$ of a vertex v as follows:

- Depth of a source is 0.
- Depth of a non-source vertex is $\max_{(u,v)\text{ — edge}} d(u) + 1$.

That is, vertices of depth 1 are dominated only by sources, vertices of depth 2 are dominated by sources and 1-depth vertices, and so on.
Clearly, no medal can be given to a vertex with depth 3 and more.

G. Medals

Several configurations of medals distributions are possible.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.
- Two medals are given to sources v and u, and one to a 1-depth vertex w.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.
- Two medals are given to sources v and u, and one to a 1-depth vertex w.
  Clearly, w must not be dominated by any vertices other than v and u.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.
- Two medals are given to sources v and u, and one to a 1-depth vertex w.
  Clearly, w must not be dominated by any vertices other than v and u. To count such configurations, iterate over all possible w and handle different cases depending on the number of sources dominating w.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.
- Two medals are given to sources v and u, and one to a 1-depth vertex w.
  Clearly, w must not be dominated by any vertices other than v and u. To count such configurations, iterate over all possible w and handle different cases depending on the number of sources dominating w.
- One medal is given to a source v, and two to 1-depth vertices u and w.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.
- Two medals are given to sources v and u, and one to a 1-depth vertex w.
  Clearly, w must not be dominated by any vertices other than v and u. To count such configurations, iterate over all possible w and handle different cases depending on the number of sources dominating w.
- One medal is given to a source v, and two to 1-depth vertices u and w.
  No sources must dominate u and w besides v.

G. Medals

Several configurations of medals distributions are possible.

- All medals are given to sources. The number of such configurations if $s(s-1)(s-2)$, where s is the number of sources.
- Two medals are given to sources v and u, and one to a 1-depth vertex w.
  Clearly, w must not be dominated by any vertices other than v and u. To count such configurations, iterate over all possible w and handle different cases depending on the number of sources dominating w.
- One medal is given to a source v, and two to 1-depth vertices u and w.
  No sources must dominate u and w besides v. To count these, iterate over possible v and count k — the number of 1-depth vertices dominated by v and nothing else. Add $k(k-1)$ to the answer.

G. Medals

- Finally, medals can be given to a source v, 1-depth vertex u, and 2-depth vertex w.

G. Medals

- Finally, medals can be given to a source v, 1-depth vertex u, and 2-depth vertex w.
  In this case, u is not dominated by vertices other than v, and w is not dominated by vertices other than v and u.

G. Medals

- Finally, medals can be given to a source v, 1-depth vertex u, and 2-depth vertex w.
  In this case, u is not dominated by vertices other than v, and w is not dominated by vertices other than v and u.
  Each non-source vertex can be a part of at most one such configuration, so there are many simple ways to count these efficiently.

G. Medals

- Finally, medals can be given to a source v, 1-depth vertex u, and 2-depth vertex w.
  In this case, u is not dominated by vertices other than v, and w is not dominated by vertices other than v and u.
  Each non-source vertex can be a part of at most one such configuration, so there are many simple ways to count these efficiently.

All these confiurations can be counted in linear ($O(n+m)$) time.

H. Snuke Density

Check if $\frac{c!}{a!b!}$ is an integer.

H. Snuke Density

Some obvious cases:

H. Snuke Density

Some obvious cases:

- $c < a$ or $c < b$ — not an integer, since numerator is less than denominator.

H. Snuke Density

Some obvious cases:

- $c < a$ or $c < b$ — not an integer, since numerator is less than denominator.
- $c \geqslant a + b$ — an integer, since $\frac{c!}{a!b!} \vdots \binom{a+b}{a}$ — the binomial coefficient, which is an integer.

H. Snuke Density

For an integer n and a prime p, denote $d(n, p)$ the number of times $n!$ can be divided by p.

H. Snuke Density

For an integer n and a prime p, denote $d(n, p)$ the number of times $n!$ can be divided by p.
It can be seen that $d(n, p) = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \dots$.

H. Snuke Density

For an integer n and a prime p, denote $d(n,p)$ the number of times $n!$ can be divided by p.
It can be seen that $d(n,p) = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \dots$.
Using this formula, $d(n,p)$ is calculated straightforwardly in $O(\log n / \log p)$ time.

H. Snuke Density

For an integer n and a prime p, denote $d(n, p)$ the number of times $n!$ can be divided by p.
It can be seen that $d(n, p) = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \ldots$.
Using this formula, $d(n, p)$ is calculated straightforwardly in $O(\log n / \log p)$ time.

Observation

$\frac{c!}{a!b!}$ is an integer iff for any prime p $d(a, p) + d(b, p) \leqslant d(c, p)$ holds.

H. Snuke Density

For an integer n and a prime p, denote $d(n,p)$ the number of times $n!$ can be divided by p.
It can be seen that $d(n,p) = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \dots$.
Using this formula, $d(n,p)$ is calculated straightforwardly in $O(\log n / \log p)$ time.

Observation

$\frac{c!}{a!b!}$ is an integer iff for any prime p $d(a,p) + d(b,p) \leqslant d(c,p)$ holds.

Thus, it suffices to check all primes that can possibly fail this condition.

H. Snuke Density

For an integer n and a prime p, denote $d(n,p)$ the number of times $n!$ can be divided by p.
It can be seen that $d(n,p) = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \ldots$.
Using this formula, $d(n,p)$ is calculated straightforwardly in $O(\log n / \log p)$ time.

Observation

$\frac{c!}{a!b!}$ is an integer iff for any prime p $d(a,p) + d(b,p) \leqslant d(c,p)$ holds.

Thus, it suffices to check all primes that can possibly fail this condition.
Since $\frac{c!}{a!b!} = \frac{\binom{a+b}{a}}{(a+b)\ldots(c+1)}$, a prime p that may fail the condition is a divisor of a number from $[c+1; a+b]$.

H. Snuke Density

For an integer n and a prime p, denote $d(n, p)$ the number of times $n!$ can be divided by p.

It can be seen that $d(n, p) = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \dots$.

Using this formula, $d(n, p)$ is calculated straightforwardly in $O(\log n / \log p)$ time.

Observation

$\frac{c!}{a!b!}$ is an integer iff for any prime p $d(a, p) + d(b, p) \leqslant d(c, p)$ holds.

Thus, it suffices to check all primes that can possibly fail this condition.

Since $\frac{c!}{a!b!} = \frac{\binom{a+b}{a}}{(a+b)\dots(c+1)}$, a prime p that may fail the condition is a divisor of a number from $[c + 1; a + b]$.

However, we have to deal with the case when then segment $[c + 1; a + b]$ is large.

H. Snuke Density

Observation

$$d(a+b,2) - d(a,2) - d(b,2) = O(\log(a+b)).$$

H. Snuke Density

Observation

$d(a+b,2) - d(a,2) - d(b,2) = O(\log(a+b)))$.

Proof

A fairly well-known number theory exercise: the number of times $\binom{a+b}{a}$ can be divided by 2 equals the number of times carrying happens when adding a and b in the binary numeral system.

H. Snuke Density

Observation

$$d(a+b,2) - d(a,2) - d(b,2) = O(\log(a+b)).$$

Proof

A fairly well-known number theory exercise: the number of times $\binom{a+b}{a}$ can be divided by 2 equals the number of times carrying happens when adding a and b in the binary numeral system. Clearly, the number of bits is $O(\log(a+b))$.

H. Snuke Density

Observation

$d(a+b,2) - d(a,2) - d(b,2) = O(\log(a+b)).$

Proof

A fairly well-known number theory exercise: the number of times $\binom{a+b}{a}$ can be divided by 2 equals the number of times carrying happens when adding a and b in the binary numeral system. Clearly, the number of bits is $O(\log(a+b))$.

Segment $[c+1; a+b]$ contains at least $\frac{a+b-c}{2}$ even numbers. It follows that if $d(a+b,2) - d(a,2) - d(b,2) < \frac{a+b-c}{2}$, the prime 2 clearly fails the divisiblity condition.

H. Snuke Density

Observation

$d(a+b,2) - d(a,2) - d(b,2) = O(\log(a+b)))$.

Proof

A fairly well-known number theory exercise: the number of times $\binom{a+b}{a}$ can be divided by 2 equals the number of times carrying happens when adding a and b in the binary numeral system. Clearly, the number of bits is $O(\log(a+b))$.

Segment $[c+1; a+b]$ contains at least $\frac{a+b-c}{2}$ even numbers. It follows that if $d(a+b,2) - d(a,2) - d(b,2) < \frac{a+b-c}{2}$, the prime 2 clearly fails the divisiblity condition.
So, is the segment $[c+1; a+b]$ is too large, the answer is "no".
Otherwise, factorize all the numbers from $[c+1; a+b]$ and check their prime divisors.

H. Snuke Density

Observation

$d(a+b,2) - d(a,2) - d(b,2) = O(\log(a+b))$.

Proof

A fairly well-known number theory exercise: the number of times $\binom{a+b}{a}$ can be divided by 2 equals the number of times carrying happens when adding a and b in the binary numeral system.
Clearly, the number of bits is $O(\log(a+b))$.

Segment $[c+1; a+b]$ contains at least $\frac{a+b-c}{2}$ even numbers. It follows that if $d(a+b,2) - d(a,2) - d(b,2) < \frac{a+b-c}{2}$, the prime 2 clearly fails the divisiblity condition.
So, is the segment $[c+1; a+b]$ is too large, the answer is "no".
Otherwise, factorize all the numbers from $[c+1; a+b]$ and check their prime divisors.
Complexity is $O(\sqrt{c} \log^2 c)$.

I. Convex Polygon

Construct a strictly convex polygon on n vertices with non-negative integer coordinates not exceeding 10^6, or determine that none exists.

I. Convex Polygon

That problem is not easier than constructing the polygon with maximal possible number of vertices, which we'll discuss.

I. Convex Polygon

That problem is not easier than constructing the polygon with maximal possible number of vertices, which we'll discuss.

"Natural" assumptions

- The polygon *mostly* consists of four monotonous polylines, each a rotated copy of another.

I. Convex Polygon

That problem is not easier than constructing the polygon with maximal possible number of vertices, which we'll discuss.

"Natural" assumptions

- The polygon *mostly* consists of four monotonous polylines, each a rotated copy of another.
- Consider a copy of the polyline that connects the bottom point and the rightmost point, and denote (X, Y) the vector from one end of the polyline to another. Then the polyline is chosen so that to maximize number of points with $X + Y \leqslant 10^6$.

I. Convex Polygon

How to construct a convex polyline with $X + Y \leqslant 10^6$?

I. Convex Polygon

How to construct a convex polyline with $X + Y \leqslant 10^6$?
Let (x_i, y_i) be vector corresponding to i-th side, ordering from beginning of the polyline.

I. Convex Polygon

How to construct a convex polyline with $X + Y \leqslant 10^6$?
Let (x_i, y_i) be vector corresponding to i-th side, ordering from beginning of the polyline.
It follows from convexity that $x_i y_{i+1} - x_{i+1} y_i > 0$.

I. Convex Polygon

How to construct a convex polyline with $X + Y \leqslant 10^6$?
Let (x_i, y_i) be vector corresponding to i-th side, ordering from beginning of the polyline.
It follows from convexity that $x_i y_{i+1} - x_{i+1} y_i > 0$.
If we have chosen pairs, we can order them by the above comparison if there are no codirectional vectors.

I. Convex Polygon

How to construct a convex polyline with $X + Y \leqslant 10^6$?
Let (x_i, y_i) be vector corresponding to i-th side, ordering from beginning of the polyline.
It follows from convexity that $x_i y_{i+1} - x_{i+1} y_i > 0$.
If we have chosen pairs, we can order them by the above comparison if there are no codirectional vectors.
Together with the requirement of $X + Y$ minimization, we conclude that x_i and y_i must be coprime for all sides, and each pair can be used at most once.

I. Convex Polygon

How do we generate coprime pairs?

I. Convex Polygon

How do we generate coprime pairs?
Here's a brief description of a construction named *Stern-Brocot tree*:

I. Convex Polygon

How do we generate coprime pairs?
Here's a brief description of a construction named *Stern-Brocot tree*:

- Start with two pairs $(0, 1)$ and $(1, 0)$.

I. Convex Polygon

How do we generate coprime pairs?
Here's a brief description of a construction named *Stern-Brocot tree*:

- Start with two pairs $(0, 1)$ and $(1, 0)$.
- For every two consecutive pairs (a, b) and (c, d) write the pair $(a + c, b + d)$ between them.

I. Convex Polygon

How do we generate coprime pairs?
Here's a brief description of a construction named *Stern-Brocot tree*:

- Start with two pairs $(0, 1)$ and $(1, 0)$.
- For every two consecutive pairs (a, b) and (c, d) write the pair $(a + c, b + d)$ between them.
- It can be shown that every coprime pair will be generated exactly once throughout the whole process.

I. Convex Polygon

How do we generate coprime pairs?
Here's a brief description of a construction named *Stern-Brocot tree*:

- Start with two pairs $(0, 1)$ and $(1, 0)$.
- For every two consecutive pairs (a, b) and (c, d) write the pair $(a + c, b + d)$ between them.
- It can be shown that every coprime pair will be generated exactly once throughout the whole process.

We can implement the process with priority queue so that the pairs are generated in the order of increasing $a + b$, until the total of $a + b$ does not exceed 10^6.

I. Convex Polygon

How do we generate coprime pairs?
Here's a brief description of a construction named *Stern-Brocot tree*:

- Start with two pairs $(0, 1)$ and $(1, 0)$.
- For every two consecutive pairs (a, b) and (c, d) write the pair $(a + c, b + d)$ between them.
- It can be shown that every coprime pair will be generated exactly once throughout the whole process.

We can implement the process with priority queue so that the pairs are generated in the order of increasing $a + b$, until the total of $a + b$ does not exceed 10^6.
This indeed maximizes the number of vertices in the polygon formed by four similar polylines.

I. Convex Polygon

WA #6

I. Convex Polygon

What did we miss?

I. Convex Polygon

What did we miss?

We cannot add more sides to all four polylines.

I. Convex Polygon

What did we miss?

We cannot add more sides to all four polylines.

However, we can find two vectors (x, y) and $(-x, -y)$ such that $X + Y + \max(x, y) \leqslant 10^6$, and (x, y) is not codirectional to any of the vectors in the existing polyline.

I. Convex Polygon

What did we miss?
We cannot add more sides to all four polylines.
However, we can find two vectors (x, y) and $(-x, -y)$ such that $X + Y + \max(x, y) \leqslant 10^6$, and (x, y) is not codirectional to any of the vectors in the existing polyline.
This is surely an improvement, and quite probably a best solution.
It's pretty hard to analyse the situation rigorously, but the best guess is that the optimal way to insert four more vectors would probably be to extend the polylines, and inserting three more vectors is hardly optimal since their sum should be zero and that would be hard to maintain with all the constraints.

I. Convex Polygon

What did we miss?
We cannot add more sides to all four polylines.
However, we can find two vectors (x, y) and $(-x, -y)$ such that $X + Y + \max(x, y) \leqslant 10^6$, and (x, y) is not codirectional to any of the vectors in the existing polyline.
This is surely an improvement, and quite probably a best solution.
It's pretty hard to analyse the situation rigorously, but the best guess is that the optimal way to insert four more vectors would probably be to extend the polylines, and inserting three more vectors is hardly optimal since their sum should be zero and that would be hard to maintain with all the constraints.
And it's accepted!

I. Convex Polygon

What did we miss?
We cannot add more sides to all four polylines.
However, we can find two vectors (x, y) and $(-x, -y)$ such that $X + Y + \max(x, y) \leqslant 10^6$, and (x, y) is not codirectional to any of the vectors in the existing polyline.
This is surely an improvement, and quite probably a best solution.
It's pretty hard to analyse the situation rigorously, but the best guess is that the optimal way to insert four more vectors would probably be to extend the polylines, and inserting three more vectors is hardly optimal since their sum should be zero and that would be hard to maintain with all the constraints.
And it's accepted!
If the side of the square is T, we can fit polygon with at most $\sim T^{2/3}$ sides. Therefore, the complexity to generate the polygon is $O(T^{2/3} \log T)$ and can be optimized to $O(T^{2/3})$.

J. Mixed Drinks

We are given a number of points (x_i, y_i, z_i) in the 3D space. For a subset S compute $(\max(x_i), \max(y_i), \max(z_i))$. Count the number of different points obtained this way.
All x_i, y_i and z_i are distinct.

J. Mixed Drinks

- Clearly, every set of one point produces an obtainable maximum.

J. Mixed Drinks

- Clearly, every set of one point produces an obtainable maximum.
- How many new maximums we obtain when we consider two-point sets?

J. Mixed Drinks

- Clearly, every set of one point produces an obtainable maximum.
- How many new maximums we obtain when we consider two-point sets?
  The set is new is no point *dominates* (is coordinate-wise greater) the other

J. Mixed Drinks

- Clearly, every set of one point produces an obtainable maximum.
- How many new maximums we obtain when we consider two-point sets?
  The set is new is no point *dominates* (is coordinate-wise greater) the other
  That is, the number of new maximums is
  $\binom{n}{2}$ − dominating pairs
- How many new maximums we obtain when we consider three-point sets?

J. Mixed Drinks

- Clearly, every set of one point produces an obtainable maximum.
- How many new maximums we obtain when we consider two-point sets?
  The set is new is no point *dominates* (is coordinate-wise greater) the other
  That is, the number of new maximums is
  $\binom{n}{2}$ − dominating pairs
- How many new maximums we obtain when we consider three-point sets?
  The set is new if no point is dominated by another, and no point is dominated by coordinate-wise maximum of other two points.

J. Mixed Drinks

Without further involvement, we mention that it suffices to know the following things to know the answer:

J. Mixed Drinks

Without further involvement, we mention that it suffices to know the following things to know the answer:

- For each point, the number of points it dominates.

J. Mixed Drinks

Without further involvement, we mention that it suffices to know the following things to know the answer:

- For each point, the number of points it dominates.
- Project all points on a coordinate plane (one of Oxy, Oxz, Oyz). For each projection and for each point, the number of points it dominates in the projection.

J. Mixed Drinks

Without further involvement, we mention that it suffices to know the following things to know the answer:

- For each point, the number of points it dominates.
- Project all points on a coordinate plane (one of Oxy, Oxz, Oyz). For each projection and for each point, the number of points it dominates in the projection.

We note that the reference solution (by *Makoto Soejima*) works in $O(n \log^2 n)$, takes ~ 120 lines in C++ and doesn't use complex data structures.

K. Hull Marathon

We can locate n points in the plane, i-th point not farther than r_i from origin. Maximize area of convex hull.

K. Hull Marathon

Suppose that we have fixed the subset of points lying on the border of convex hull, as well as their order around the origin.

K. Hull Marathon

Suppose that we have fixed the subset of points lying on the border of convex hull, as well as their order around the origin.
Obviously, each point has to be located at the maximal distance, r_i.

K. Hull Marathon

Suppose that we have fixed the subset of points lying on the border of convex hull, as well as their order around the origin.
Obviously, each point has to be located at the maximal distance, r_i.
It suffices to choose directions to the points from the origin.
Denote α_i the angle between the Ox axis and the vector Op_i. Put $\alpha_0 = 0$ by definition.

K. Hull Marathon

Suppose that we have fixed the subset of points lying on the border of convex hull, as well as their order around the origin.
Obviously, each point has to be located at the maximal distance, r_i.
It suffices to choose directions to the points from the origin.
Denote α_i the angle between the Ox axis and the vector Op_i. Put $\alpha_0 = 0$ by definition.
Also for convenience put $\alpha_n = 2\pi$, $r_n = r_0$.

K. Hull Marathon

Suppose that we have fixed the subset of points lying on the border of convex hull, as well as their order around the origin.
Obviously, each point has to be located at the maximal distance, r_i.
It suffices to choose directions to the points from the origin.
Denote α_i the angle between the Ox axis and the vector Op_i. Put $\alpha_0 = 0$ by definition.
Also for convenience put $\alpha_n = 2\pi$, $r_n = r_0$.
Area of the convex hull is then equal to

$$S(\alpha_0, \ldots, \alpha_n) = \sum_{i=0}^{n-1} \frac{1}{2} r_i r_{i+1} \sin(\alpha_{i+1} - \alpha_i)$$

K. Hull Marathon

At the global maximum $S'_{\alpha_i} = 0$ for all i from 1 to $n-1$.

K. Hull Marathon

At the global maximum $S'_{\alpha_i} = 0$ for all i from 1 to $n-1$.
$S'_{\alpha_i} = r_i r_{i+1} \cos(\alpha_{i+1} - \alpha_i) - r_{i-1} r_i \cos(\alpha_i - \alpha_{i-1})$

K. Hull Marathon

At the global maximum $S'_{\alpha_i} = 0$ for all i from 1 to $n-1$.
$S'_{\alpha_i} = r_i r_{i+1} \cos(\alpha_{i+1} - \alpha_i) - r_{i-1} r_i \cos(\alpha_i - \alpha_{i-1})$
It follows that $cos(\alpha_{i+1} - \alpha_i) = \frac{r_{i-1}}{r_{i+1}} \cos(\alpha_i - \alpha_{i-1})$.

K. Hull Marathon

At the global maximum $S'_{\alpha_i} = 0$ for all i from 1 to $n-1$.
$S'_{\alpha_i} = r_i r_{i+1} \cos(\alpha_{i+1} - \alpha_i) - r_{i-1} r_i \cos(\alpha_i - \alpha_{i-1})$
It follows that $cos(\alpha_{i+1} - \alpha_i) = \frac{r_{i-1}}{r_{i+1}} \cos(\alpha_i - \alpha_{i-1})$.
Denote $\beta_i = \alpha_{i+1} - \alpha_i$. By the above formula, we can choose numbers t_i such that $\cos(\beta_i) = t_i \cos(\beta_0)$.

K. Hull Marathon

At the global maximum $S'_{\alpha_i} = 0$ for all i from 1 to $n-1$.
$S'_{\alpha_i} = r_i r_{i+1} \cos(\alpha_{i+1} - \alpha_i) - r_{i-1} r_i \cos(\alpha_i - \alpha_{i-1})$
It follows that $cos(\alpha_{i+1} - \alpha_i) = \frac{r_{i-1}}{r_{i+1}} \cos(\alpha_i - \alpha_{i-1})$.
Denote $\beta_i = \alpha_{i+1} - \alpha_i$. By the above formula, we can choose numbers t_i such that $\cos(\beta_i) = t_i \cos(\beta_0)$.
We have the requirement $\sum_{i=0}^{n-1} \beta_i = 2\pi$, and the sum of angles is monotonous over $\cos(\beta_0)$. By doing binary search on $\cos(\beta_0)$, we obtain optimal values of β_i.

K. Hull Marathon

At the global maximum $S'_{\alpha_i} = 0$ for all i from 1 to $n-1$.
$S'_{\alpha_i} = r_i r_{i+1} \cos(\alpha_{i+1} - \alpha_i) - r_{i-1} r_i \cos(\alpha_i - \alpha_{i-1})$
It follows that $cos(\alpha_{i+1} - \alpha_i) = \frac{r_{i-1}}{r_{i+1}} \cos(\alpha_i - \alpha_{i-1})$.
Denote $\beta_i = \alpha_{i+1} - \alpha_i$. By the above formula, we can choose numbers t_i such that $\cos(\beta_i) = t_i \cos(\beta_0)$.
We have the requirement $\sum_{i=0}^{n-1} \beta_i = 2\pi$, and the sum of angles is monotonous over $\cos(\beta_0)$. By doing binary search on $\cos(\beta_0)$, we obtain optimal values of β_i.
Note that $|\cos(\beta_i)|$ cannot exceed 1.

K. Hull Marathon

It suffices to choose optimal subset of points and order and around the origin.

K. Hull Marathon

It suffices to choose optimal subset of points and order and around the origin.
Let's do it straightforwardly, by trying all subsets and all permutations for every mask.

K. Hull Marathon

It suffices to choose optimal subset of points and order and around the origin.
Let's do it straightforwardly, by trying all subsets and all permutations for every mask.
A very simple optimization of this approach is to permute all elements except the first, since only the cyclic order matters.

K. Hull Marathon

It suffices to choose optimal subset of points and order and around the origin.
Let's do it straightforwardly, by trying all subsets and all permutations for every mask.
A very simple optimization of this approach is to permute all elements except the first, since only the cyclic order matters.
The complexity of this approach $O(2^n(n-1)!\log^{-1}\varepsilon)$.