NAIPC 2018, GP of America

March 25, 2018

Mikhail Tikhomirov (MIPT)

Moscow Pre-finals ACM ICPC Workshop 2018

A. Cut It Out!

You are given two convex polygons A and B, and B is completely inside A. We are allowed to make straight cuts through sides of B, and remove the part of A that doesn't contain B. The cost of a cut is the length of the cut segment. Find the smallest cost to cut out everything except for B.

A. Cut It Out!

Let us apply subsegment dynamic programming. Let $dp_{i,j}$ be the smallest cost for the problem when B is reduced to points $B_i, \ldots, B_j$ in clockwise order (note that j may be smaller than i, in which case we loop around through the first point to obtain the part), and the cuts along segments $B_{i-1}B_i$ and B_jB_{j+1} were made (indices of points are considered to be looped, that is, $B_{m+1} = B_1$).

A. Cut It Out!

Let us apply subsegment dynamic programming. Let $dp_{i,j}$ be the smallest cost for the problem when B is reduced to points $B_i, \ldots, B_j$ in clockwise order (note that j may be smaller than i, in which case we loop around through the first point to obtain the part), and the cuts along segments $B_{i-1}B_i$ and B_jB_{j+1} were made (indices of points are considered to be looped, that is, $B_{m+1} = B_1$).

Suppose that the next cut to be made is along the segment B_kB_{k+1}. To make an impact, both points B_k and B_{k+1} must be in the range $B_i, \ldots, B_j$. The optimal answer is the sum of the current cut cost and $dp_{i,k} + dp_{k+1,j}$), since these parts become independent.

A. Cut It Out!

The only problem is to compute the cost of the cut. To do this, we find the closest intersection point of the ray $B_k B_{k+1}$ with the polygon A, as well as the cut $B_j B_{j+1}$; also do the same for the ray $B_{k+1} B_k$ with A, and the cut $B_i B_{i-1}$. This boils down to intersecting a bunch of straight lines.

A. Cut It Out!

The only problem is to compute the cost of the cut. To do this, we find the closest intersection point of the ray $B_k B_{k+1}$ with the polygon A, as well as the cut $B_j B_{j+1}$; also do the same for the ray $B_{k+1} B_k$ with A, and the cut $B_i B_{i-1}$. This boils down to intersecting a bunch of straight lines.

To initialize our DP, try all options for the first cut $B_i B_{i+1}$, compute its cost as before, and add $dp_{i+1,i}$ to the answer.

A. Cut It Out!

The only problem is to compute the cost of the cut. To do this, we find the closest intersection point of the ray $B_k B_{k+1}$ with the polygon A, as well as the cut $B_j B_{j+1}$; also do the same for the ray $B_{k+1} B_k$ with A, and the cut $B_i B_{i-1}$. This boils down to intersecting a bunch of straight lines.

To initialize our DP, try all options for the first cut $B_i B_{i+1}$, compute its cost as before, and add $dp_{i+1,i}$ to the answer.

Current complexity is $O(nm^3)$ since there are $O(m^3)$ transitions in the DP, and for each of them we have to intersect cuts with the polygon A in $O(n)$ times. However, there are only $O(m)$ different cuts to consider, so by precomputing these intersections we get $O(nm + m^3)$ solution.

B. Double Clique

For a graph $G = (V, E)$, count the number of sets $S \subseteq V$ such that S is a clique, and $V \backslash S$ is an independent set.

B. Double Clique

Let us show that if an answer exists, then there is a number d such that all vertices with degree $\geqslant d$ form a double clique. Let $deg(v)$ denote the degree of v, and $\delta_-(X)$ and $\delta_+(X)$ denote the smallest and the largest degree in a vertex set X respectively.

B. Double Clique

Let us show that if an answer exists, then there is a number d such that all vertices with degree $\geqslant d$ form a double clique. Let $deg(v)$ denote the degree of v, and $\delta_-(X)$ and $\delta_+(X)$ denote the smallest and the largest degree in a vertex set X respectively.

Consider any double clique S. Note that $\delta_-(S) \geqslant |S| - 1$, and $\delta_+(V \backslash S) \leqslant |S|$. First, let's obtain a double clique S' such that $\delta_+(V \backslash S') < |S'|$.

B. Double Clique

Let us show that if an answer exists, then there is a number d such that all vertices with degree $\geqslant d$ form a double clique. Let $deg(v)$ denote the degree of v, and $\delta_-(X)$ and $\delta_+(X)$ denote the smallest and the largest degree in a vertex set X respectively.

Consider any double clique S. Note that $\delta_-(S) \geqslant |S| - 1$, and $\delta_+(V \backslash S) \leqslant |S|$. First, let's obtain a double clique S' such that $\delta_+(V \backslash S') < |S'|$.

If $\delta_+(V \backslash S) < |S|$, then we are done. Otherwise, let $v \in V \backslash S$ have degree $|S|$, in this case $S' = S + v$ fits.

B. Double Clique

Let us show that if an answer exists, then there is a number d such that all vertices with degree $\geqslant d$ form a double clique. Let $deg(v)$ denote the degree of v, and $\delta_-(X)$ and $\delta_+(X)$ denote the smallest and the largest degree in a vertex set X respectively.

Consider any double clique S. Note that $\delta_-(S) \geqslant |S| - 1$, and $\delta_+(V \backslash S) \leqslant |S|$. First, let's obtain a double clique S' such that $\delta_+(V \backslash S') < |S'|$.

If $\delta_+(V \backslash S) < |S|$, then we are done. Otherwise, let $v \in V \backslash S$ have degree $|S|$, in this case $S' = S + v$ fits.

Now, if $\delta_-(S') > \delta_+(V \backslash S)$, we are done with $d = \delta_-(S')$. The other case can happen only if $\delta_-(S') = \delta_+(V \backslash S') = |S'| - 1$.

B. Double Clique

Consider $u \in V \backslash S'$ with $deg(u) = |S'| - 1$. All vertices in S' adjacent to u have degree $\geqslant |S'|$. But since $\delta_{-}(S') = |S'| - 1$, there must exist a unique vertex $v \in S'$ with $deg(v) = |S'| - 1$, which must not be adjacent to any vertex of $V \backslash S'$.

B. Double Clique

Consider $u \in V \backslash S'$ with $deg(u) = |S'| - 1$. All vertices in S' adjacent to u have degree $\geqslant |S'|$. But since $\delta_-(S') = |S'| - 1$, there must exist a unique vertex $v \in S'$ with $deg(v) = |S'| - 1$, which must not be adjacent to any vertex of $V \backslash S'$.

Let $S'' = S' - v$. We finally have $\delta_-(S'') \geqslant |S'|$ and $\delta_+(V \backslash S'') < |S'|$, hence S'' is the desired set with $d = \delta_-(S'')$.

B. Double Clique

Let's try all values of $d = \delta_{-}(S'')$. We have to check if all vertices with degree $\geqslant d$ form a clique (by counting edges), and if all vertices with degree $< d$ form an independent set (in order for that to happen, every edge must have at least one endpoint with degree $\geqslant d$).

B. Double Clique

Let's try all values of $d = \delta_{-}(S'')$. We have to check if all vertices with degree $\geqslant d$ form a clique (by counting edges), and if all vertices with degree $< d$ form an independent set (in order for that to happen, every edge must have at least one endpoint with degree $\geqslant d$).

Every double clique S differs from S'' in at most one element. When we've found suitable d, try to include all elements of $V \setminus S''$ and exclude all elements of S'', and count the number of correct double cliques.

B. Double Clique

Let's try all values of $d = \delta_{-}(S'')$. We have to check if all vertices with degree $\geqslant d$ form a clique (by counting edges), and if all vertices with degree $< d$ form an independent set (in order for that to happen, every edge must have at least one endpoint with degree $\geqslant d$).

Every double clique S differs from S'' in at most one element. When we've found suitable d, try to include all elements of $V \setminus S''$ and exclude all elements of S'', and count the number of correct double cliques.

The complexity of this approach is $O(n)$ (but could possibly be $O(n \log n)$).

C. Flashing Fluorescents

There is a row of n lights, with a button corresponding to each light. At integer time moments we may press a button, which results in flipping the state of the corresponding light on the next second, flipping the next light on the subsequent second, and so on. The effects from pressing buttons stack together. Find the shortest time needed to make all lights on.

C. Flashing Fluorescents

Consider the process from the end. If we are done at time t, then the button pressed at time $t-1$ only flips its corresponding light, the button pressed at time $t-2$ flips two adjacent lights (or only one light if its the last one), and so on.

C. Flashing Fluorescents

Consider the process from the end. If we are done at time t, then the button pressed at time $t-1$ only flips its corresponding light, the button pressed at time $t-2$ flips two adjacent lights (or only one light if its the last one), and so on.

Hence, we can do this: let S_k be the set of masks of lit lights reachable in k steps. S_0 contains only the initial states. Start increasing l; if S_l contains $1 \ldots 1$, then we are done with the answer l. Otherwise, let $S_{l+1} = \{s \oplus I_{j,\min(j+l-1,n)}\} | s \in S_l, i = 1, \ldots, n\}$, where $I_{l,r}$ is the mask with 1's in positions $l, l+1, \ldots, r$.

C. Flashing Fluorescents

Consider the process from the end. If we are done at time t, then the button pressed at time $t-1$ only flips its corresponding light, the button pressed at time $t-2$ flips two adjacent lights (or only one light if its the last one), and so on.

Hence, we can do this: let S_k be the set of masks of lit lights reachable in k steps. S_0 contains only the initial states. Start increasing l; if S_l contains $1 \ldots 1$, then we are done with the answer l. Otherwise, let $S_{l+1} = \{s \oplus I_{j,\min(j+l-1,n)}\} | s \in S_l, i = 1, \ldots, n\}$, where $I_{l,r}$ is the mask with 1's in positions $l, l+1, \ldots, r$.

It can easily be shown that the answer is at most n, hence this procedure can be implemented as an $O(n^2 2^n)$ time submask DP.

D. Missing Gnomes

Find the lexicographically smallest permutation that contains a given sequence A as a subsequence.

D. Missing Gnomes

If the smallest number not present in A is less than A_1, then it is optimal to place it in the beginning of the permutation and continue. Otherwise, placing anything before A_1 will make the answer worse, so we should place A_1 at the start, remove it from A and continue.

D. Missing Gnomes

If the smallest number not present in A is less than A_1, then it is optimal to place it in the beginning of the permutation and continue. Otherwise, placing anything before A_1 will make the answer worse, so we should place A_1 at the start, remove it from A and continue.

Simulate this process in $O(n)$ time.

E. Prefix Free Code

You are given a dictionary of n words such that no word is a prefix of another word. You are also given a string t that is a concatenation of k distinct words from the dictionary. Find the number of t in the lexicographical order among all possible concatenations of k dictionary words.

E. Prefix Free Code

Let us order the words in the dictionary in their lexicographical order: $s_1 < \ldots < s_n$.

E. Prefix Free Code

Let us order the words in the dictionary in their lexicographical order: $s_1 < \ldots < s_n$.

Note that since the dictionary is prefix-free, there is at most one way to split any string into dictionary words: take the unique prefix of the string present in the dictionary, erase it, take the next prefix, and so on. There aren't any ambiguities since no two words can simultaneously be prefixes of a string.

E. Prefix Free Code

Let us order the words in the dictionary in their lexicographical order: $s_1 < \ldots < s_n$.

Note that since the dictionary is prefix-free, there is at most one way to split any string into dictionary words: take the unique prefix of the string present in the dictionary, erase it, take the next prefix, and so on. There aren't any ambiguities since no two words can simultaneously be prefixes of a string.

Observe further that concatenations are lexicographically compared the same as index sequences of the words in their unique splits. Indeed, if the first words do not match, then the comparison is already settled (since otherwise one of the words is a prefix of the other). If the first words are equal, we drop them and proceed to comparing the rest in the same way.

E. Prefix Free Code

Thus, we just have to convert t into a index sequence and compute its number among sequence of k distinct numbers from 1 to n. The latter is a standard combinatorics task, and can be done in $O(k \log n)$ with range-sum data structures.

E. Prefix Free Code

Thus, we just have to convert t into a index sequence and compute its number among sequence of k distinct numbers from 1 to n. The latter is a standard combinatorics task, and can be done in $O(k \log n)$ with range-sum data structures.

One easy way to convert t into the index sequence is to construct a trie of all dictionary words, and store their numbers in corresponding trie vertices. Start descending the trie using the letters of t; whenever we arrive at a terminal vertex, write down its number and return back to the root.

E. Prefix Free Code

Thus, we just have to convert t into a index sequence and compute its number among sequence of k distinct numbers from 1 to n. The latter is a standard combinatorics task, and can be done in $O(k \log n)$ with range-sum data structures.

One easy way to convert t into the index sequence is to construct a trie of all dictionary words, and store their numbers in corresponding trie vertices. Start descending the trie using the letters of t; whenever we arrive at a terminal vertex, write down its number and return back to the root.

The total complexity is $O(T + k \log n)$, where T is the total length of all strings (also we use $O(T\alpha)$ memory, where α is the size of the alphabet).

F. Probe Droids

The points in an $n \times m$ grid are ordered in counter-clockwise order with respect to the point $(1, 1)$, the point on the same ray are order by the distance. Answer queries "what is the k-th point in this order".

F. Probe Droids

Let us solve an intermediate problem: how many points (x, y) are there with $0 < x \leqslant n$ and $y/x \leqslant a/b$ for some numbers $a, b > 1$? We can assume that a and b are coprime.

F. Probe Droids

Let us solve an intermediate problem: how many points (x, y) are there with $0 < x \leqslant n$ and $y/x \leqslant a/b$ for some numbers $a, b > 1$? We can assume that a and b are coprime.

We can solve this with a modified Euclid's algorithm. Suppose that $a \geqslant b$. Let $t = \lfloor a/b \rfloor$. Rewrite $y/x \leqslant a/b$ as $by \leqslant ax$, and then as $by \leqslant (tb + a')x$ for $a' = a \bmod b$.

F. Probe Droids

Rewrite further: $b(y - tx) \leqslant a'x$. There are two kinds of solutions to this: with $y - tx \geqslant 0$ and $y - tx < 0$. Note that $y - tx < 0$ automatically ensures the inequality, and the number of such points under $x \leqslant n$ is $\sum_{x=1}^{n} tx = tn(n+1)/2$. The number of solutions with $y - tx \geqslant 0$ is the solution to a smaller problem with n, a', b.

F. Probe Droids

Rewrite further: $b(y - tx) \leqslant a'x$. There are two kinds of solutions to this: with $y - tx \geqslant 0$ and $y - tx < 0$. Note that $y - tx < 0$ automatically ensures the inequality, and the number of such points under $x \leqslant n$ is $\sum_{x=1}^{n} tx = tn(n+1)/2$. The number of solutions with $y - tx \geqslant 0$ is the solution to a smaller problem with n, a', b.

If $a < b$, then we rewrite similarly as $b'y \leqslant a(x - ty)$ for $t = b \bmod a$ and $t = \lfloor a/b \rfloor$. Further, let $y_0 = \lfloor an/b \rfloor$. The solutions with $x \leqslant n - ty_0$ are the solutions of an instance with parameters $n - ty_0, a, b'$, and solutions with $x > n - ty_0$ can be counted roughly as before as a sum of certain arithmetic progression. The complexity of this approach is the same as the Euclid's algorithm: $O(\log a + \log b)$.

F. Probe Droids

Assume that the target point is below or on the ray with the slope m/n. Then our problem can be solved by "binary searching" on the largest slope a/b such that there are $k' < k$ solutions to $0 < x \leqslant n$, $y/x < a/b$. The answer then is $((k-k')a, (k-k')b)$. If the target point has slope larger than m/n, we can flip the dimensions and solve for $k' = nm - k$.

F. Probe Droids

Assume that the target point is below or on the ray with the slope m/n. Then our problem can be solved by "binary searching" on the largest slope a/b such that there are $k' < k$ solutions to $0 < x \leqslant n$, $y/x < a/b$. The answer then is $((k-k')a, (k-k')b)$. If the target point has slope larger than m/n, we can flip the dimensions and solve for $k' = nm - k$.

This binary search can be implemented with a so-called *Stern-Brocot tree*. The idea is that any irreducible fraction can be obtained by the following process: start with fractions $a/b = 0/1$ and $c/d = 1/0$. Then on each step we change either a/b or c/d to $(a+c)/(b+d)$, which lies strictly between a/b and c/d.

F. Probe Droids

Now, to binary search for a largest fraction with denominator $\leqslant n$ and satisfying some monotone criterion $f(x/y)$, we "descend" the tree by checking if $f((a+c)/(b+d))$ holds, and then moving one of the boundaries of the search accordingly. Once the next fraction would have denominator largest than n, we output a/b as the answer.

F. Probe Droids

Now, to binary search for a largest fraction with denominator $\leqslant n$ and satisfying some monotone criterion $f(x/y)$, we "descend" the tree by checking if $f((a+c)/(b+d))$ holds, and then moving one of the boundaries of the search accordingly. Once the next fraction would have denominator largest than n, we output a/b as the answer.

In the worst case this process can take up to n steps if there are long chains of descending in the same direction (for instance, when the answer is $1/n$). We can avoid this problem by binary searching the number of descents in the same direction on each step. The complexity then becomes $O(\log^2 n)$ times the time needed to compute $f(x/y)$. In our case the function is computable in $O(\log n)$ (that was our intermediate problem), for the total complexity $O(\log^3 n)$.

G. Rainbow Graph

A weighted undirected graph is given, with every edge colored red, blue or green. For every k find the smallest weight of a set of exactly k edges which is spanning both in the graph without red edges, and in the graph without blue edges (or report that there is no such set).

G. Rainbow Graph

Consider two families of subsets of edges M_r and M_b, where M_c consists of *complements* of spanning subsets without the edges of color c (that is, edges of color c are irrelevant).

G. Rainbow Graph

Consider two families of subsets of edges M_r and M_b, where M_c consists of *complements* of spanning subsets without the edges of color c (that is, edges of color c are irrelevant).

Both of these are easily seen to be *matroids*. Thus, our problem is the *weighted matroid intersection* problem.

G. Rainbow Graph

The solution is a modification of the usual matroid intersecion algorithm. Let $S \in M_r \cup M_b$ be a solution for a number of edges k. Construct the directed *exchange graph* $G(S)$ with vertices being edges of the original graph, and edges added as follows:

G. Rainbow Graph

The solution is a modification of the usual matroid intersecion algorithm. Let $S \in M_r \cup M_b$ be a solution for a number of edges k. Construct the directed *exchange graph* $G(S)$ with vertices being edges of the original graph, and edges added as follows:

- For $e_1 \in S$, $e_2 \notin S$ an arc (e_1, e_2) is present iff $S - e_1 + e_2 \in M_r$;

G. Rainbow Graph

The solution is a modification of the usual matroid intersecion algorithm. Let $S \in M_r \cup M_b$ be a solution for a number of edges k. Construct the directed *exchange graph* $G(S)$ with vertices being edges of the original graph, and edges added as follows:

- For $e_1 \in S$, $e_2 \notin S$ an arc (e_1, e_2) is present iff $S - e_1 + e_2 \in M_r$;
- For $e_1 \in S$, $e_2 \notin S$ an arc (e_2, e_1) is present iff $S - e_1 + e_2 \in M_b$.

G. Rainbow Graph

The solution is a modification of the usual matroid intersecion algorithm. Let $S \in M_r \cup M_b$ be a solution for a number of edges k. Construct the directed *exchange graph* $G(S)$ with vertices being edges of the original graph, and edges added as follows:

- For $e_1 \in S$, $e_2 \notin S$ an arc (e_1, e_2) is present iff $S - e_1 + e_2 \in M_r$;
- For $e_1 \in S$, $e_2 \notin S$ an arc (e_2, e_1) is present iff $S - e_1 + e_2 \in M_b$.

Put costs to vertices of $G(S_k)$ as $w(e)$ for $e \in S_k$, and $-w(e)$ for $e \notin S_k$.

G. Rainbow Graph

The solution is a modification of the usual matroid intersecion algorithm. Let $S \in M_r \cup M_b$ be a solution for a number of edges k. Construct the directed *exchange graph* $G(S)$ with vertices being edges of the original graph, and edges added as follows:

- For $e_1 \in S$, $e_2 \notin S$ an arc (e_1, e_2) is present iff $S - e_1 + e_2 \in M_r$;
- For $e_1 \in S$, $e_2 \notin S$ an arc (e_2, e_1) is present iff $S - e_1 + e_2 \in M_b$.

Put costs to vertices of $G(S_k)$ as $w(e)$ for $e \in S_k$, and $-w(e)$ for $e \notin S_k$.

Find a shortest (with respect to vertex weights) path between sets A and B, where $A = \{v \in S \mid S + v \in M_r\}$, and $B = \{v \in S \mid S + v \in M_b\}$. Flip the state of all edges in the path; the new set S' is optimal for $k + 1$.

G. Rainbow Graph

Finding the shortest path can be done with Ford-Bellman algorithm, for the total complexity of $O(n^2 m)$.

G. Rainbow Graph

Finding the shortest path can be done with Ford-Bellman algorithm, for the total complexity of $O(n^2 m)$.

For further reading on matroid intersection refer to [Frank, András. "A weighted matroid intersection algorithm." Journal of Algorithms 2.4 (1981): 328-336.].

H. Recovery

Recover a $0-1$ matrix by the parities of the sums in each row and column. The matrix has to have the largest total number of 1's, and out of them should be lexicographically smallest (comparing elements in row-major order).

H. Recovery

If the total parities by rows and columns are different, there is obivously no answer.

H. Recovery

If the total parities by rows and columns are different, there is obivously no answer.

Otherwise, compute the parities of the numbers of 0's in each row and column. The number of 0's in a suitable matrix is at least $\max(s_r, s_c)$, where s_r and s_c is the number of rows (columns) with an odd number of 0's.

H. Recovery

If the total parities by rows and columns are different, there is obivously no answer.

Otherwise, compute the parities of the numbers of 0's in each row and column. The number of 0's in a suitable matrix is at least $\max(s_r, s_c)$, where s_r and s_c is the number of rows (columns) with an odd number of 0's.

Let $r_1 < \ldots < r_{s_r}$ and $c_1 < \ldots < c_{s_c}$ be the indices or rows (columns) with an odd number of 0's. Place 0's at positions (r_{s_r-i}, c_{s_c-i}) for $i = 0, \ldots, \min(s_r, s_c) - 1$, assuming s_i and r_i to be 1 when $i < 1$. It can be easily seen that the resulting matrix meets the parity conditions, has $\min(s_r, s_c)$ 0's, and is lexicographically minimal.

H. Recovery

If the total parities by rows and columns are different, there is obivously no answer.

Otherwise, compute the parities of the numbers of 0's in each row and column. The number of 0's in a suitable matrix is at least $\max(s_r, s_c)$, where s_r and s_c is the number of rows (columns) with an odd number of 0's.

Let $r_1 < \ldots < r_{s_r}$ and $c_1 < \ldots < c_{s_c}$ be the indices or rows (columns) with an odd number of 0's. Place 0's at positions (r_{s_r-i}, c_{s_c-i}) for $i = 0, \ldots, \min(s_r, s_c) - 1$, assuming s_i and r_i to be 1 when $i < 1$. It can be easily seen that the resulting matrix meets the parity conditions, has $\min(s_r, s_c)$ 0's, and is lexicographically minimal.

The total complexity is $O(nm)$.

I. Red Black Tree

There is a rooted tree with n vertices, out of which m are red. For each $k = 0, \ldots m$ count the number of sets of vertices which do not contain a vertex together with any of its ancestors, and have exactly k red vertices.

I. Red Black Tree

Let us compute $dp_{v,k}$ — the answer for the problem for the subtree of v. First count the sets not containing v. Start with $dp_{v,0} = 1$, and $dp_{v,k} = 0$ for all other k. Compute answers for children successively; whenever an answer for the new child u is computed, the new values $dp'_{v,k}$ are equal to $\sum_{i=0}^{k} dp_{v,i} dp_{u,k-i}$.

I. Red Black Tree

Let us compute $dp_{v,k}$ — the answer for the problem for the subtree of v. First count the sets not containing v. Start with $dp_{v,0} = 1$, and $dp_{v,k} = 0$ for all other k. Compute answers for children successively; whenever an answer for the new child u is computed, the new values $dp'_{v,k}$ are equal to $\sum_{i=0}^{k} dp_{v,i} dp_{u,k-i}$.

The only set not accounted for contains only the vertex v; add 1 to $dp_{v,k}$, where $k = 1$ if v is red, or $k = 0$ otherwise.

I. Red Black Tree

Let us compute $dp_{v,k}$ — the answer for the problem for the subtree of v. First count the sets not containing v. Start with $dp_{v,0} = 1$, and $dp_{v,k} = 0$ for all other k. Compute answers for children successively; whenever an answer for the new child u is computed, the new values $dp'_{v,k}$ are equal to $\sum_{i=0}^{k} dp_{v,i} dp_{u,k-i}$.

The only set not accounted for contains only the vertex v; add 1 to $dp_{v,k}$, where $k = 1$ if v is red, or $k = 0$ otherwise.

If done inefficiently, this can take up to $O(nm^2)$ time. However, if we limit i and $k - i$ in the "gluing" up of $dp_{v,.}$ and $dp_{u,.}$ to be at most the number of red vertices in corresponding sets, a standard argument shows that the total complexity becomes $O(nm + m^2) = O(nm)$.

I. Red Black Tree

However, we may still have issues with memory limit since the total sizes of $dp_{v,\cdot}$ can still be $\Omega(nm)$. Let us store $dp_{v,\cdot}$ in dynamic arrays of appropriate sizes, and only store them while they are needed in recursion.

I. Red Black Tree

However, we may still have issues with memory limit since the total sizes of $dp_{v,\cdot}$ can still be $\Omega(nm)$. Let us store $dp_{v,\cdot}$ in dynamic arrays of appropriate sizes, and only store them while they are needed in recursion.

Observe that at any time any red vertex "contributes" to the size of an array of at most one vertex. It means that the memory usage becomes $O(n + m)$ in the worst case.

J. Winter Festival

Assign costs between 0 and 2 to edges of a graph, so that no two adjacent edges have total cost 1 (mod 3), and the sum of costs along any simple cycle is odd.

J. Winter Festival

Assume that there is an edge that belongs to two different simple cycles. Then we can find three simple cycles C_1, C_2, C_3 such that $C_3 = C_1 \triangle C_2$, where cycles are treated as sets of edges, and $X \triangle Y$ is the symmetric difference of X and Y. Clearly, there is no coloring such that all C_1, C_2, C_3 have odd sum. It follows that every edge must lie on at most one simple cycle, that is, the graph must be a *cactus*.

J. Winter Festival

Assume that there is an edge that belongs to two different simple cycles. Then we can find three simple cycles C_1, C_2, C_3 such that $C_3 = C_1 \triangle C_2$, where cycles are treated as sets of edges, and $X \triangle Y$ is the symmetric difference of X and Y. Clearly, there is no coloring such that all C_1, C_2, C_3 have odd sum. It follows that every edge must lie on at most one simple cycle, that is, the graph must be a *cactus*.

First, check if the graph is a cactus, and if so, decompose it into biconnected components, each of which is either a cycle or a single edge. This can be done with Tarjan's algorithm in linear time.

J. Winter Festival

Next we "root" the cactus. To do that, consider a vertex that shares adjacent edges from different bicomponents. Out of them, choose a single *parent bicomponent*, and mark all others as its *child components*. The markings are good if any bicomponent (expect a unique *root bicomponent*) is a child bicomponent with respect to a unique vertex (call this a *root vertex* of a bicomponent), and making child-parent transitions always leads us to the root bicomponent. This can done with a single DFS starting from an arbitrary component.

J. Winter Festival

Next we "root" the cactus. To do that, consider a vertex that shares adjacent edges from different bicomponents. Out of them, choose a single *parent bicomponent*, and mark all others as its *child components*. The markings are good if any bicomponent (expect a unique *root bicomponent*) is a child bicomponent with respect to a unique vertex (call this a *root vertex* of a bicomponent), and making child-parent transitions always leads us to the root bicomponent. This can done with a single DFS starting from an arbitrary component.

Now we apply a subtree-like DP to this rooted structure. The result for a bicomponent is going to be the smallest value of a suitable coloring of a "sub-cactus" hanging on the bicomponent so that edges adjacent to the root of the bicomponent take values in a specified set, for all sets of colors.

J. Winter Festival

If our bicomponent is an edge, we simply merge the answers for components parented at both ends of the edge, while making sure that no color collisions happen at that vertex. After that, try all colors of the only edge in the bicomponent and all situations at the vertices, and choose the best ones.

J. Winter Festival

If our bicomponent is an edge, we simply merge the answers for components parented at both ends of the edge, while making sure that no color collisions happen at that vertex. After that, try all colors of the only edge in the bicomponent and all situations at the vertices, and choose the best ones.

If the bicomponent is a cycle, we apply linear DP on the cycle. Start at the root vertex of the cycle, and choose the color of the first edge arbitrarily. The DP will have the following parameters: the number of edges of the cycle processed, current parity of decided edges, the set of colors adjacent to the root vertex, and the set of colors adjacent to the last processed vertex (including adjacent edges of children bicomponents).

J. Winter Festival

To make transitions, choose the color of the next edge, as well as sets of colors for the children components (this requires an intermediate DP that processes children one by one). When all edges are considered, we have to make sure there are no conflicts in the root.

J. Winter Festival

To make transitions, choose the color of the next edge, as well as sets of colors for the children components (this requires an intermediate DP that processes children one by one). When all edges are considered, we have to make sure there are no conflicts in the root.

In total, this solution still has $O(n+m)$ complexity (in fact, $m = O(n)$ in a cactus).

K. Zoning Houses

There are n points in the plane. Answer q queries "what is the smallest axes-parallel square size that encloses all points with indices $l \leqslant i \leqslant r$ except one?".

K. Zoning Houses

We want to solve the problem with segment tree. In order to do that, we need to maintain information to obtain the answer, as well as merge the information for disjoint subproblems.

K. Zoning Houses

We want to solve the problem with segment tree. In order to do that, we need to maintain information to obtain the answer, as well as merge the information for disjoint subproblems.

Note that it is only optimal to remove a point if it is extreme by at least one coordinate. Let us maintain the bounding box of the points in the (sub)problem, as well as the bounding boxes when removing a point with the smallest/largest x/y-coordinate. Ties can be broken arbitrarily since the coordinate won't be affected by removal of one point anyway.

K. Zoning Houses

We want to solve the problem with segment tree. In order to do that, we need to maintain information to obtain the answer, as well as merge the information for disjoint subproblems.

Note that it is only optimal to remove a point if it is extreme by at least one coordinate. Let us maintain the bounding box of the points in the (sub)problem, as well as the bounding boxes when removing a point with the smallest/largest x/y-coordinate. Ties can be broken arbitrarily since the coordinate won't be affected by removal of one point anyway.

Merging the data is trivial when no points are erased. If we want to erase, say, a point with the smallest x-coordinate, we choose the best option of merging all points in one subproblem with all but x-smallest point in another.

K. Zoning Houses

The solution has complexity $O(n \log n)$ (with a large constant factor for recomputing bounding boxes and trying different options for merging).