Problem A. Building Bombing

Problem idea: *Dongkyu Han* (*queued_q*), *Won Park* (*chunghan*)

Preparation: *Dongkyu Han* (*queued_q*)

First solver: 아아악!!!원더에게서도망쳐야해요!!!티원따운!!! (*Onsite*, *91 min.*), *aeren* (*Open*, *30 min.*)

Total solved team: *7* (*Onsite*), *7* (*Open*)

For buildings on the left to building L, we only have to blow up the buildings that block building L from being visible. The main problem is to solve for the buildings on the right. From now on, we will assume $L = 1$.

We can solve the problem using dynamic programming. Let $D[i, k, h]$ be the minimum number of buildings to blow up only considering buildings $1 \dots i$, and after blowing up, the number of visible buildings becomes k and the greatest height among them becomes h.

We will add buildings one by one and calculate the corresponding DP values. There are three cases to consider when adding building i to the set. We can make it visible by taking $k - 1$ visible buildings on the left, where the greatest height is less than h_i. Or, we can make it invisible either by blowing it up or hiding it behind another building. The recurrence relation is as follows.

$$D[i,k,h] = \begin{cases} D[i-1,k,h]+1 & \text{for } h < h_i \\ \min(DP[i-1,k,h], \min_{h'<h} D[i-1,k-1,h']) & \text{for } h = h_i \\ D[i-1,k,h] & \text{for } h > h_i \end{cases}$$

This takes $O(N^2K)$ naively, so we have to do it efficiently. Suppose we have added buildings $1 \dots i$ so far. We will store K arrays $D[i, k, *]$ for $k = 1 \dots K$ in K segment trees. Now, the task of adding a building to the set becomes adding 1 to a range, querying the minimum in a range, and updating a point in a segment tree. Each task can be done in $O(\log N)$ per segment tree using lazy propagation. Since there are K segment trees and we will add N buildings in total, the time complexity is $O(NK \log N)$.

Problem B. Connecting Cables

Problem idea: Eunsoo Choe (gs18115)

Preparation: Eunsoo Choe (gs18115)

First solver: bitKOIn (Onsite, 13 min.), tlwpdus (Open, 10 min.)

Total solved team: 12 (Onsite), 11 (Open)

Let $[(x_1, x_2), (y_1, y_2)]$ represents the area has a boundary with (x_1, y_1) as a lower left corner and (x_2, y_2) as a upper right corner.

Minimum cost for cable between $[(x_1, x_2), (y_1, y_2)]$ and $[(x'_1, x'_2), (y'_1, y'_2)]$ is calculated as $\max\{\min\{x_1, x'_1\} - \max\{x_2, x'_2\}, 0\} + \max\{\min\{y_1, y'_1\} - \max\{y_2, y'_2\}, 0\}$.

Since the expression is independent of axis, we can calculate the sum of cost by adding costs in each axis.

The problem in 1D is solved by sorting and sweeping the intervals.

Problem C. Double-Colored Papers

Problem idea: Jongyoung Lee (moonrabbit2)

Preparation: Jongyoung Lee (moonrabbit2)

First solver: N/A (Onsite), N/A (Open)

Total solved team: 0 (Onsite), 0 (Open)

There exists a very complicated solution only using suffix array, but here we explain the intended solution using suffix automaton.

We build a suffix automaton for S and T, A and B. Then, all possible double-colored paper is a combination of path in automaton of S starting from the root, and path in automaton of T starting from the root.

Our strategy is to fix the characters one by one: if we have found the prefix pfx of the answer string ans, we can find the next character of ans, or conclude that $pfx = ans$ if we can calculate the number of double-colored papers with having $pfx + c$ as prefix, where c is a arbitrary character.

To do this, we should analyze the possible pair of paths given the string pfx. Let u be the vertex in A reached when following the path pfx until there is no edge of desired direction, and v be the vertex in B reached when following the path pfx, following the link edge while there is no edge of desired direction.

All possible endpoints of the first path are the vertices in the path in A from root to u, and all possible endpoints of the second path are the vertices in the path in link tree of B from root to v. The two pair of vertices are a valid pair if sum of the lengths they present is $|pfx|$. Be careful that vertices in B correspond may present multiple lengths.

Note that we are finding the maximum possible length of prefix and suffix of pfx that is included in S and T by this procedure, and we can maintain u and v while adding characters to the end of pfx.

With dynamic programming in A and B, we can calculate the number of paths starting from each vertex in the automaton, *cnt*.

To count the number of pair of paths, note that the vertices in B that are included in a valid pair forms a suffix of the path: we can also store the first vertex not included in the suffix. Let's call it w. Then, the number of pair of paths is sum of *cnt* values multiplied by the number of lengths the vertex is representing, for the vertices in the path $w - v$ in the link tree of B. w and v should have extra care, as only some suffix of w is counted and some prefix of v is counted, depending on minimum and maximum length of possible suffix of pfx.

With dynamic programming in link tree of B, we can calculate this value after adding character c to the end, in a method similar to prefix sum. If we update u, v and w after each update, we can solve the whole problem in $O(26(|S| + |T|))$.

Problem D. Histogram Sequence 4

Problem idea: Jongyoung Lee (moonrabbit2)

Preparation: Jongyoung Lee (moonrabbit2)

First solver: 🍖 (Onsite, 5 min.), tlwpdus (Open, 20 min.)

Total solved team: 12 (Onsite), 25 (Open)

If $NL > A$, then there is no answer.

Otherwise, there should exist some rectangle inside the histogram with width w and height $\frac{A}{w}$ where $w \le N$ and $\frac{A}{w} \le R$. If such w exists, we can always construct the solution: first w rectangle have height $\frac{A}{w}$ and others have height L. Note that $L \le \frac{A}{w}$, so this construction always works.

When checking $NL > A$, you should be careful of overflows. There are various ways: using python, using 128-bit integers, or checking if $L > \lfloor \frac{A}{N} \rfloor$.

Problem E. Making Number

Problem idea: *Jimin Ahn* (*retro3014*)

Preparation: *Jongyoung Lee* (*moonrabbit2*)

First solver: 새쿼리와 내기 (*Onsite*, *101 min.*), *tlwpdus* (*Open*, *166 min.*)

Total solved team: *3* (*Onsite*), *4* (*Open*)

Let's define N_i as the i-th digit of N.

There is two cases for Z:

- $Z = Y$.
- There exists an unique index i, $Z_j = Y_j$ for $1 \leq j \leq i - 1$ and $Z_i > Y_i$.

The first case is checked easily by comparing counts of each digit.

For the second case, we can find an answer by binary searching the index i.

Let's define $I(j)$ as the maximum index satisfies following, corresponding to a given index j:

- $I(j) \geq j$
- $Y_k \geq Y_{k+1}$ for $j \leq k \leq I(j) - 1$

i.e. $I(j)$ is the maximum index such that $N_{j \sim I(j)}$ is non-increasing sequence.

It can be checked that $i \leq j$ for given index j with following steps.

1. Calculate $C_I(d)$, the count of digit d in $Y_{j \sim I(j)}$, for $0 \leq d \leq 9$.
2. Calculate $C_P(d)$, the count of digit d in $Y_{1 \sim j-1}$, for $0 \leq d \leq 9$
3. Calculate $C_X(d)$, the count of digit d in X, for $0 \leq d \leq 9$.
4. If $C_X(d) < C_P(d)$ for some $digit$, result $i < j$.
5. Find the maximum digit d satisfies $C_P(d) \neq C_X(d) - C_I(d)$.
6. If there is no such d or $C_P(d) > C_X(d) - C_I(d)$, result $i < j$.
7. Otherwise, check the existance of d' satisfies $d' < d$ and $C_P(d') > 0$.
8. If d' exists, result $i \geq j$. If not, result $i < j$.

Each step can be proceed in the node of segment tree where the node of segment tree is consists of count of each digit in each interval, l to r and l to $min\{I(l), r\}$.

Time complexity is $O(D(N + Q \log N))$ where D is the number of digits.

Problem F. One Path

Problem idea: Eunsoo Choe (gs18115)

Preparation: Eunsoo Choe (gs18115)

First solver: 아아악!!!원더에게서도망쳐야해요!!!티원따운!!! (*Onsite, 136 min.*), *tlwpdus* (*Open, 236 min.*)

Total solved team: 1 (Onsite), 1 (Open)

Problem G. Permutation Arrangement

Problem idea: Kyung Chan Lee (kclee2172)

Preparation: Kyung Chan Lee (kclee2172)

First solver: cCc (Onsite, 95 min.), hyperbolic (Open, 182 min.)

Total solved team: 4 (Onsite), 1 (Open)

Suppose that there are non-adjacent 8 empty spaces left in the permutation. Then, each remaining number has at least 4 squares that it can go to and each remaining square has at least 4 numbers that can be in the square. Hence, from Hall's marriage theorem, there exist the valid arrangement of remaining numbers.

Hence, when there are 15 or more empty spaces it is always possible to arrange the remaining numbers. Therefore, the solution of greedily assigning numbers until there are 15 numbers left and using bit DP to find the optimal assignment of last 15 numbers works.

From casework it is possible to bring the minimum number of empty spaces needed down and hence the brute force solution of testing all the permutations in lexicographical order also works when implemented well.

Time complexity is $O(n)$.

Problem H. Set and Sequence and Query

Problem idea: *Won Park (chunghan)*

Preparation: *Won Park (chunghan)*

First solver: 🍖 (*Onsite, 16 min.*), *ainta* (*Open, 15 min.*)

Total solved team: *12* (*Onsite*), *15* (*Open*)

Problem I. Similarity Graph

Problem idea: Eunsoo Choe (gs18115)

Preparation: Eunsoo Choe (gs18115)

First solver: N/A (Onsite), ainta (Open, 253 min.)

Total solved team: 0 (Onsite), 2 (Open)

Problem J. Squirrel Game

Problem idea: *Dongkyu Han* (*queued_q*)

Preparation: *Dongkyu Han* (*queued_q*)

First solver: 아아악!!!원더에게서도망쳐야해요!!!티원따운!!! (*Onsite*, *58 min.*), *tlwpdus* (*Open*, *59 min.*)

Total solved team: *8* (*Onsite*), *7* (*Open*)

Let d_i be the sequence of distances between the squirrels. That is, $d_i = x_i - x_{i-1} - 1$, assuming there also is a squirrel at $x_0 = 0$. A single move decreases d_i by some amount and increases d_{i+K} by the same amount for some i. (If d_{i+K} does not exist, we just ignore it.) We can decompose the game into K distinct chains of distances in the form d_{Kq+r}, for each r in $0 \dots K-1$. Each chain is equivalent to a game where $K = 1$. If we can calculate the Grundy number of each chain, we can solve the entire game using the Sprague-Grundy theorem. However, it turns out that we don't even need the Sprague-Grundy theorem to solve this problem.

Let us first solve for $K = 1$. From the right, label the squirrels with R and L alternatingly. We can pair up two consecutive squirrels with the label L on the left and R on the right. We claim that the game is equivalent to a Nim game where the heap sizes are the distances between each squirrel pair.

Suppose you have a winning strategy and have played accordingly. Your opponent has two choices: Move a squirrel labeled L or labeled R. If your opponent moves a squirrel labeled R, it is equivalent to reducing the heap size in the corresponding Nim game. Therefore, you are still in a winning position. If your opponent moves a squirrel labeled L, which is increasing the heap size, you can move the paired squirrel of label R so that the heap returns to the original size. Therefore, you are back in the winning position. Note that the game has to end in finite turns because each move reduces the sum of x_i. It means that your opponent cannot indefinitely stall the game by always increasing the heap size.

When $K > 1$, the game is equivalent to playing K distinct Nim games, which is just a single Nim game with all the heaps from each game.

Problem K. Two Paths

Problem idea: Jongyoung Lee (moonrabbit2)

Preparation: Jongyoung Lee (moonrabbit2)

First solver: N/A (Onsite), N/A (Open)

Total solved team: 0 (Onsite), 0 (Open)

First, with dynamic programming, we can find the result of following query $F(u, v, w)$ in $O(1)$: the farthest vertex from u except for the directions v and w, where v and w are either vertex connected to u or 0(no direction).

For a single query, let P be the path from u to v. Let w_1 be the last vertex of P_1 inside P, and w_2 be the last vertex of P_2 inside P. If we fix some vertex c inside P, then we can divide the cases by three: both w_1 and w_2 is in c - u path, both w_1 and w_2 are in c - v path, or w_1 is in c - u path and w_2 is in c - v path.

With centroid decomposition, we make c be the root of the tree. Note that while P is inside the current tree, now P_1 and P_2 may not be fully included in the tree, so we use F query to find the actual farthest vertex in the tree.

First, let's solve the case where w_1 or w_2 is c, and the case where w_1 is in c - u path and w_2 is in c - v path. In this case, optimal choice of w_1 and w_2 is independent, so by calculating optimal w for each vertex using dynamic programming(Using the F query), this case can be solved in $O(1)$.

Now we solve the case where both w_1 and w_2 is in c - u path. The case where both w_1 and w_2 is in c - v path can be solved symmetrically. In this case, if w_1 is fixed, the optimal w_2 is fixed, and can be found by dynamic programming(Again using the F query). Also, the result value with w_1 and w_2 fixed can be expressed as a line $Ax + By + C$ for A, B values in each query, and C being a constant fixed for each query. Therefore, we can solve this case with Convex Hull Trick in tree. You should be careful with the case $u = w_1$.

The time complexity is $O(N \log^2 N + Q \log N)$.

Problem L. Village Planning

Problem idea: Jongyoung Lee (moonrabbit2)

Preparation: Jongyoung Lee (moonrabbit2)

First solver: N/A (Onsite), N/A (Open)

Total solved team: 0 (Onsite), 0 (Open)

$K = 0$ is easy: answer is $A_0^{\binom{N}{2}}$ for each N.

$K = 3$ is a trick: the answers are equal with the case $K = 2$.

Now we only need to solve $K = 1$ and $K = 2$. First, we solve the case where $A_i = 1$ for all i: we just need to count the number of possible graphs.

For both $K = 1$ and $K = 2$, we first count the number of possible connected components for each size, c_i. Then, we can count the number of ways to merge the components.

Let's fix the number of components k, and label each component from 1 to k. Let size of component i be n_i, where $\sum_{i=1}^{k} n_i = N$. Then, the number of possible ways would be $\binom{N}{n_1 \cdots n_k} \prod_{i=1}^{k} c_{n_i}$. Therefore, the number of graphs with k connected components is $\sum_{n_1+\cdots+n_k=N} \binom{N}{n_1 \cdots n_k} \prod_{i=1}^{k} c_{n_i}$. Finally, to remove the order of the components, we divide the result by $k!$.

The above analysis tells us that if the exponential generating function of c is $P(x)$, then answer's exponential generating function is $\sum_{k=0}^{\infty} \frac{P(x)^k}{k!} = \exp(P(x))$.

When $K = 1$, each connected component is a tree. The number of connected components of size N is N^{N-2}.

When $K = 2$, each connected component is a tree or a unicyclic graph. Let's count the number of connected unicyclic graph of size N.

The unicyclic graph is formed by a cycle and a rooted trees with each vertex in the cycle being a root. Let $Q(x)$ be the exponential generating function of N^{N-1}. By a similar analysis to above, we can find the exponential generating function of the number of unicyclic graphs as $\sum_{k=3}^{\infty} \frac{Q(x)^k}{k!} \times \frac{(k-1)!}{2} = \sum_{k=3}^{\infty} \frac{Q(x)^k}{2k} = \frac{1}{2}(-\ln(1 - Q(x)) - Q(x) - \frac{Q(x)^2}{2})$.

Now, the general case where $1 \leq A_i < 998244353$ is not too different: for each step, we are merging components. Note that the number of simple paths with both vertices in the same component is fixed, and the number of simple paths with two vertices in different components is fixed. Using this fact, you can just modify the generating function before and after each step by multiplying $A_i^{\binom{N}{2}}$ and $A_i^{-\binom{N}{2}}$ in each terms appropriately.

Problem M. Window Arrangement

Problem idea: *Kyung Chan Lee* (*kclee2172*)

Preparation: *Kyung Chan Lee* (*kclee2172*), *Eunsoo Choe* (*gs18115*)

First solver: 아아악!!!원더에게서도망쳐야해요!!!티원따운!!! (*Onsite*, *53 min.*), *ainta* (*Open*, *57 min.*)

Total solved team: *5* (*Onsite*), *5* (*Open*)

This problem can be modeled as MCMF as following.

1. Create one source vertex, one sink vertex, one vertex for each room, and one vertex for each places where we can put windows
2. Make edges connecting source vertex and rooms and give it capacity according to number of windows needed and cost zero.
3. Make edges connecting rooms and potential window locations. Give the edge capacity one and cost zero.
4. For each potential window location, connect it and the sink vertex twice. The first edge has capacity one and cost zero and the second one has capacity one and cost equal to the discomfort we get when we put two windows in this location (i.e. product of numbers of people in two rooms connected to this window location)

And now running MCMF algorithm on this graph solves the problem. The time complexity is $O(n^2m^2)$.