Problem Tutorial: "Donut"

If a point (x, y) is in the donut of center (x_c, y_c), this is equivalent to the point (x_c, y_c) is in the donut of center (x, y). Thus, we set the input points as the center of the donut, and their area has its score. We can sweep the area to maintain intervals and find the point (x_c, y_c) that maximize the sum. One of the possible methods is segment tree with lazy propagation.

Problem Tutorial: "Circular Arrangement"

There is a well-known problem: Given n positive integer $c_1, c_2, \cdots, c_n$, count the number of arrays containing exactly c_i copies of integer i where no two consecutive elements are same. This problem can solve with inclusion-exclusion principle and DP.

The solution of our problem based on above problem. Let $w_{c,k}$ is the product of the sizes of k group of c elements. Thus, $w_{c,k} = \sum_{j=1}^{c} w_{c-j,k-1} \cdot j$.

If integer i has k $(1 \leq k \leq c_i)$ groups, then $w_{c_i,k}$ be multiplied. Put every value for every k together, and then we can calculate the sum of the cost to make every different possible array.

Dealing with circular is annoying, but not very hard.

Problem Tutorial: "Earthquake"

If we currently look route A, it is optimal way that look only bridges in route A until we could know the route is intact or not.

Also, we should look from the most dangerous bridge to the least dangerous bridge in a route.

we can calculate

$P[i]$:= the probability that route i is safe

$Q[i]$:= the expected number of the inspections, if route i is unsafe.

for each route.

It is easy to solve the remaining part.

Problem Tutorial: "Dynamic Input Tool"

1. preprocessing : calculate $F[c][i]$:= the first location $x > i$ that satisfies $S[i] = c$ for each alphabet c and location i.

2. Use greedy algorithm. Suppose the current location is i (we need to enter $S[i]$) and the current location of the subsequence is $pivot$. if $F[S[i]][pivot] >= i$, then increase the answer by 1 and now $pivot = 0$. otherwise, $pivot = F[S[i]][pivot]$

Problem Tutorial: "Central Lake"

The problem can be transformed into the problem of finding the two points with the largest angle difference.

1. $O(Qlog^2N)$ solution

if there are only build queries, we can process each query per $O(logN)$ by using set. Suppose a house at point x is built in the b-th query and demolished in the e-th query. We can save this in the segment tree. Each query is saved in $O(logN)$ nodes in the segment tree. Then we can manage queries by preorder traversal of the segment tree.

2. $O(QlogN)$ solution

There are 2 sets, $S1$ and $S2$. $S1$ saves the points of the houses. $S1$ contains the points of the houses, $S2$ contains the antipodal points of the houses. If we can find the closest point pair ($p1$, $p2$) that satisfies $p1 \in S1$ and $p2 \in S2$, we can solve the problem.

We can manage this by segment tree : for each range [b,e], save the leftmost point and the rightmost point in this range (for both S1 and S2). Also, save the closest point pair ($p1$, $p2$).

Problem Tutorial: "MST with Metropolis"

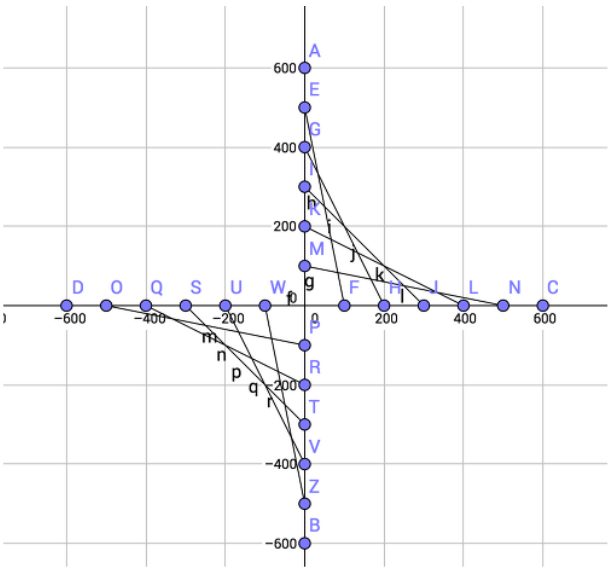
First, make ordinary MST T from G. We can prove that MST with metropolis vertex i will be made by deleting some edges in T and adding every edge end with i.

Let v_i is the set of vertex i and its neighbors, and t_i is the minimal subtree that contains every vertex in v_i. We can prove that deleting some edges in t_i is sufficient to make MST with metropolis vertex i.

t_i can represent as $O(|v_i|)$ branches and $O(|v_i|)$ edge chains. Branches are vertices in v_i and the LCA of the two consecutive vertices in sorted v_i of DFS order. Edge chains are the set of the edges in the branch to branch. We should delete at most one edge in each edge chain. Deleted edge must be the max weight edge in the edge chain.

The actual process starts with delete every max weight edge in each edge chain. And restore deleted edges in non-decreasing order of edge weight.

Problem Tutorial: "Number of Cycles"



For a set of segment S, let $NC(S) :=$ the number of simple cycles in the graph generated by these segment

First, put two long (nearly infinite) segments on the x-axis and y-axis. (segment AB and CD in the above picture)

There are 12 segments in above picture. (segment AB, segment CD, 5 segments in the first quadrant, and 5 segments in the third quadarnt) As you can see, there are lots of cycles in the picture. Also, we can think the cycles in the first quadrant (including segment AB and segment CD) and the cycles in the third quadrant (including segment AB and segment CD) separately.

Let's think about the cycles in the first quadrant. There are 7 segments : segment AB, CD, EF, GH, IJ, KL, MN. For solve this problem, it is enough to find two set containing 7 segments, $S1$ and $S2$, that satisfies $NC(S1) + NC(S2) = N$. (Put $S1$ in the first quadrant and put $S2$ in the third quadrant. it shares AB and CD, so there are 12 segments in total.)

Ignoring the vertex A, B, C and D, there are six vertices in each segment. There are $({}_6C_2)^5$ ways(we don't change segment AB and CD) to reduce the length of the segments by moving their endpoints inside the segment (the straight line created by the extension does not change).

Finding $S1$ and $S2$ for all N between 1 and 1000 is possible.

If we check all the cases, it tooks more than 3 seconds. But we can use some random technique, or branch and bound , or get segments set by preprocessing and just put it in the code. All the methods are good enough to solve this problem.

Problem Tutorial: "Game of Sorting"

Let $A[b,e] = a_b, a_{b+1}, ..., a_e$.

For the initial sequence $A[b,e]$, $Win[b,e] = 1$ if Alice wins the game. otherwise $Win[b,e] = 0$.

If $A[b,e]$ is monotone(nonincreasing or nondecreasing), then $Win[b,e] = 0$. For other cases, $Win[b,e] = (!Win[b,e-1])|(!Win[b+1,e])$

Key idea : Except a few cases, $Win[b,e] = Win[b+1,e-1]$. then we can calculate $Win[b,e]$ easily by its $b+e$ value.

For a sequence S, if there is a contiguous monotone subsequence with length L and there is no contiguous monotone subsequence with length $L+1$, we call S is $L-monotone$.

1. If $A[b,e]$ is $(e-b+1)-monotone$, then $Win[b,e] = 0$.

2. If $A[b,e]$ is $(e-b)-monotone$, then $Win[b,e] = 1$.

3. If $A[b,e]$ is $(e-b-1)-monotone$

3-a. $A[b,e-2]$ is monotone or $A[b+2,e]$ is monontone

without loss of generality, $A[b,e-2]$ is monotone. Then $Win[b,e] = !Win[b+1,e] = Win[b+2,e] = !Win[b+3,e] = ...$ it continues until $Win[i,e]$ which $A[i,e]$ is monotone. So if we precalculate $L(e) :=$ (the minimum value of x that satisfies $A[L(e),e]$ is monotone) for all e, then we can calculate $Win[b,e]$ by the value $L(e)$.

3-b. $A[b+1,e-1]$ is monotone

It is easy to show that $Win[b,e] = 0$

4. For other cases

$Win[b,e] = (!Win[b,e-1])|(!Win[b+1,e])$

$= (Win[b,e-2]\ \ \&\ \ Win[b+1,e-1])|(Win[b+1,e-1]\ \ \&\ \ Win[b+2,e])$

it is easy to show that $Win[b,e] = Win[b+1,e-1]$.

Problem Tutorial: "Subsequence Queries"

There are well-known DP to calculate the number of subsequences of string. In this problem, we use $|\Sigma| = 52$ symbols. But in this tutorial, we will use only 3 symbols. Generalization is straightforward.

Transition in DP can be represented as a matrix. A_0, A_1, A_2 are matrix for each symbol.

$$A_0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For string $S = c_1 c_2 \cdots c_l$, the number of subsequences of S is,

$$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}^T A_{c_l}\cdots A_{c_2}A_{c_1}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

The number of subsequences of substring $c_i \cdots c_j$ can represent as the same way.

$$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}^T A_{c_j}A_{c_{j-1}}\cdots A_{c_{i+1}}A_{c_i}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Each matrix has its inverse.

$$A_0^{-1}=\begin{pmatrix}1&-1&-1&-1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\quad A_1^{-1}=\begin{pmatrix}1&0&0&0\\-1&1&-1&-1\\0&0&1&0\\0&0&0&1\end{pmatrix}\quad A_2^{-1}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\-1&-1&1&-1\\0&0&0&1\end{pmatrix}$$

Thus,

$$\begin{pmatrix}1\\1\\1\\1\end{pmatrix}^T A_{c_j}\cdots A_{c_i}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}=\begin{pmatrix}1\\1\\1\\1\end{pmatrix}^T A_{c_j}\cdots A_{c_i}A_{c_{i-1}}\cdots A_{c_1}A_{c_1}^{-1}\cdots A_{c_{i-1}}^{-1}\begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Let $I_i = A_{c_1}^{-1}\cdots A_{c_i}^{-1}$, $J_j = A_{c_j}\cdots A_{c_1}$, and $I_0 = I$ (Identity matrix) for convenience.

First, $I_i = I_{i-1}A_{c_i}^{-1}$. We represent I_i with four column vector $v_{i,0},\cdots,v_{i,3}$ and assume that $c_i = 0$.

$$\begin{pmatrix}v_{i-1,0}\\v_{i-1,1}\\v_{i-1,2}\\v_{i-1,3}\end{pmatrix}^T\begin{pmatrix}1&-1&-1&-1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}=\begin{pmatrix}v_{i-1,0}\\v_{i-1,1}-v_{i-1,0}\\v_{i-1,2}-v_{i-1,0}\\v_{i-1,3}-v_{i-1,0}\end{pmatrix}^T$$

For reduce calculation complexity, we use common subtracting column vector D_i for I_i. Now,

$$\begin{pmatrix}v_{i-1,0}-D_{i-1}\\v_{i-1,1}-D_{i-1}\\v_{i-1,2}-D_{i-1}\\v_{i-1,3}-D_{i-1}\end{pmatrix}^T\begin{pmatrix}1&-1&-1&-1\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}=\begin{pmatrix}v_{i-1,0}-D_{i-1}\\v_{i-1,1}-v_{i-1,0}\\v_{i-1,2}-v_{i-1,0}\\v_{i-1,3}-v_{i-1,0}\end{pmatrix}^T=\begin{pmatrix}v_{i,0}-D_i\\v_{i,1}-D_i\\v_{i,2}-D_i\\v_{i,3}-D_i\end{pmatrix}^T$$

Let $D_i = v_{i-1,0}$, then $v_{i,0} = 2v_{i-1,0} - D_{i-1}$, $v_{i,1} = v_{i-1,1}$, $v_{i,2} = v_{i-1,2}$ and $v_{i,3} = v_{i-1,3}$. Note that only $v_{i,0}$ is changed.

Second, $J_j = A_{c_j} J_{j-1}$. We represent J_j with four row vector $u_{j,0}, \cdots, u_{j,3}$ and assume that $c_j = 0$. Let $S_j = u_{j,0} + u_{j,1} + u_{j,2} + u_{j,3}$ for convenience.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_{j-1,0} \\ u_{j-1,1} \\ u_{j-1,2} \\ u_{j-1,3} \end{pmatrix} = \begin{pmatrix} S_{j-1} \\ u_{j-1,1} \\ u_{j-1,2} \\ u_{j-1,3} \end{pmatrix} = \begin{pmatrix} u_{j,0} \\ u_{j,1} \\ u_{j,2} \\ u_{j,3} \end{pmatrix}$$

Thus, $S_j = 2S_{j-1} - u_{j-1,0}$. Note that only $u_{j,0}$ is changed.

Now, we can calculate the number of subsequences of substring $c_i \cdots c_j$ easily.

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T A_{c_j} \cdots A_{c_i} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T J_j I_{i-1} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} u_{j,0} \\ u_{j,1} \\ u_{j,2} \\ u_{j,3} \end{pmatrix} \begin{pmatrix} v_{i-1,0} - D_{i-1} \\ v_{i-1,1} - D_{i-1} \\ v_{i-1,2} - D_{i-1} \\ v_{i-1,3} - D_{i-1} \end{pmatrix}^T \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$= \left[\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} u_{j,0} \\ u_{j,1} \\ u_{j,2} \\ u_{j,3} \end{pmatrix} \right] \left[\begin{pmatrix} v_{i-1,0} - D_{i-1} \\ v_{i-1,1} - D_{i-1} \\ v_{i-1,2} - D_{i-1} \\ v_{i-1,3} - D_{i-1} \end{pmatrix}^T \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right] = (u_{j,0} + u_{j,1} + u_{j,2} + u_{j,3}) (v_{i-1,3} - D_{i-1})$$

$$= S_j \left(\begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}^T - D_{i-1} \right)$$

$v_{i,3} = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix}^T$ because this cannot be changed. Note that we don't need to maintain all of u, v vectors, but need to maintain all of D, S vectors.

Overall time complexity is $O\big((N+Q)|\Sigma|\big)$ and memory complexity is $O\big(N|\Sigma| + |\Sigma|^2\big)$

Problem Tutorial: "XOR Transformation "

For $F_K^{2^k}(X) = \big[g_{k,0}(X), g_{k,1}(X), \cdots, g_{k,N-1}(X)\big]$, we can prove that

$$g_{k,i}(X) = \bigoplus_{j=0}^{K-1} x_{(i+2^k \times j) \bmod N}$$

They are XOR of the elements in the step 2^k. Thus, we can get each g values in $O(N)$ time complexity with prefix XOR (in the step 2^k).

And apply repeated squaring, we can get $F_K^T(X)$.