AtCoder Grand Contest 001 Editorial

July 16, 2016

A. BBQ Easy

N 個のペアを $(a_1, b_1), \ldots, (a_N, b_N)$ とすると，串にさせるものの個数は $min(a_1, b_1) + \cdots + min(a_N, b_N)$ となる．

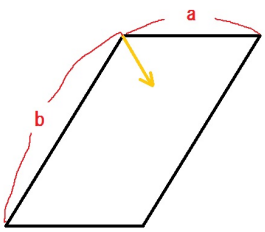
x を入力で与えられた数の最小とし，y が x とペアになっているとする．y の値によらず，この串にさせるものの個数は x となるので，他の串にさせるものの個数を最大化するために y は二番目に小さい数とするのが最適である．同様に，三番目と四番目，五番目と六番目，…をペアにするのが最適となるので，入力をソートして $L_1 \leq \cdots \leq L_{2N}$) とすると，答えは $min(L_1, L_2) + min(L_3, L_4) + \cdots + min(L_{2N-1}, L_{2N})$ となる．

B. Mysterious Light

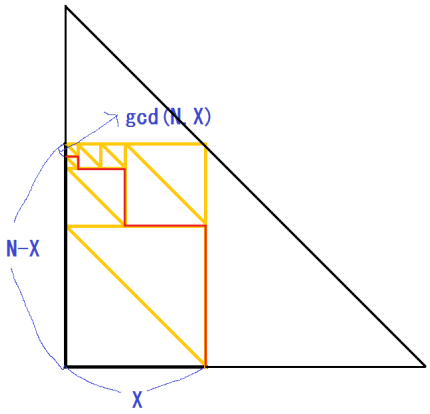
下図で $a * b$ の平行四辺形にこの光を使ったときの長さを $f(a,b)$ とする．

答えは $N + f(N-X, X)$ であり，$a < b$ のとき $f(a,b) = 2a + f(a, b-a)$ であることを使うと $O(N)$ の回答が得られ，300 点を取ることができる．

満点解法では最大公約数を求めるときに使うユークリッドの互助法と似たようなことをする．$a < b$ で b が a の倍数でないとき，$f(a,b) = 2 * floor(b/a) * a + f(a, b\%a)$ となる．これを使うと $O(logN)$ となり満点が得られる．



また，光の軌跡が三角形の集合となり，三角形の辺の長さの和が下図より $N - gcd(N, X)$ となることに気付くと，答えが $3(N - gcd(N, X))$ であることがわかる．



C. Shorten Diameter

木に関する以下の性質を用いる．

木 T の直径を D とする．

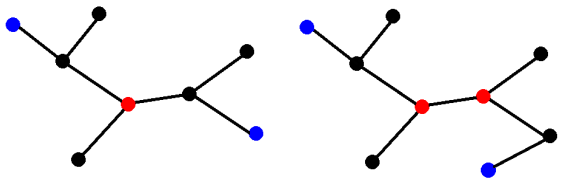
- D が偶数ならば，ある頂点 v が存在して v から他の頂点への距離が $D/2$ 以下となる．
- D が奇数ならば，ある辺 e が存在して e から他の頂点への距離が $(D-1)/2$ 以下となる．

v と e は木の中心とよばれる．

下図で，青い頂点は直径を表している．青い頂点の中点を赤とすると，赤が中心となることが分かる．

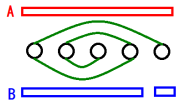
D が偶数のとき，中心の位置を N 通り全て試す．中心から距離 $D/2$ より離れている頂点を全て取り除けばよいので，各頂点から距離 $D/2$ 以上の頂点の個数を数え，その個数の最小値が答えとなる，

同様に，D が奇数のときは全ての辺を中心として試せばよい．この解答は $O(N^2)$ となる．

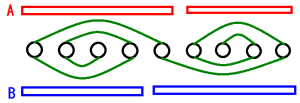


D. Arrays and Palindrome

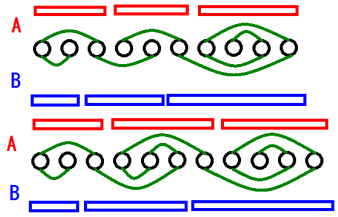
$M = 1$ の場合が最も簡単である．$a = \{x\}$ のとき $b = \{x - 1, 1\}$ とすればよい．



$M = 2$ のときも似たような構成が可能である．$a = \{x, y\}$ のとき $b = \{x - 1, y + 1\}$ とすればよい．



$M \geq 3$ のときも似た構成を試す．つまり，$a = \{x_1, \ldots, x_M\}$ のとき $b = \{x_1 - 1, x_2, \ldots, x_{M-1}, x_M + 1\}$ とする．これは中央の長さの偶奇により，偶数ならうまくいき，奇数ならうまくいかない．



a に奇数が三個以上含まれているとする．a に含まれる奇数の個数を O_a とすると，上図で二箇所を結んでいる辺の個数は $(N - O_a)/2$ となる．同様に，b に含まれる奇数の個数を O_b とすると，上図で二箇所を結んでいる辺の個数は $(N - O_b)/2$ となる．

全ての場所をつなぐためには少なくとも $N - 1$ 本の辺が必要なので，$(N - O_a)/2 + (N - O_b)/2 \geq N - 1$ であり，$O_a + O_b \leq 2$ となるが，これは $O_a > 2$ のとき不可能である．

逆に，a に含まれる個数が二個以下の場合は，奇数が両端になるように並べれば全ての中央の値は偶数となり，上の構成法でよい．

E. BBQ Hard

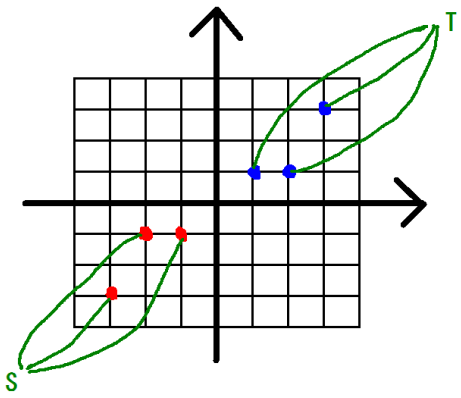
i 番と j 番の集合を選ぶと，串を作る方法は $f(A_i + A_j, B_i + B_j)$ 通りである．ここで $f(x, y) = (x + y)!/x!/y!$ は x 個の同じものと y 個の同じものを並べる方法の個数である．

次の和を計算すればよい:

$$\sum_{1 \le i < j \le N} f(A_i + A_j, B_i + B_j) \tag{1}$$

$f(x, y)$ はグリッド上で $(0, 0)$ から (x, y) への経路数なので，$f(A_i + A_j, B_i + B_j)$ は $(-A_i, -B_i)$ から (A_j, B_j) への経路数となる．

下図で， 点 $(-A_i, -B_i)$ (赤) と点 (A_i, B_i) (青) をプロットしている．赤い点のうちの一つから青い点のうちの一つへの経路の個数を求めるとほとんど答えとなる．



X をその経路数とすると，

$$X = \sum_{1 \le i, j \le N} f(A_i + A_j, B_i + B_j) \tag{2}$$

となるので，答えは

$$\sum_{1\leq i<j\leq N} f(A_i+A_j, B_i+B_j) = (X - \sum_i f(A_i+A_i, B_i+B_i))/2 \tag{3}$$

である．

X は図で S から T への経路数であるので，$O(MAX^2)$ で単純な DP によりもとめられる．f も $O(1)$ でもとめられるので，この解法は $O(MAX^2+N)$ となる．

F. Wide Swap

Q を P の逆の配列とする ($Q_{P_i} = i$ をみたす) と，問題は次のように書き換えられる．

Problem. 順列 Q が与えられる．$|Q_i - Q_j| \geq K$ のとき Q_i と Q_{i+1} をスワップすることができる．1 がなるべく左の場所にあるようにしたい．そのようなな順列のうち，2 がなるべく左にあるようにしたい．そのようなな順列のうち，3 がなるべく左にあるようにしたい…としたときの順列を求めよ．

このとき，Q を別の順列 R に変えることができるためには，任意の $|x-y| < K$ をみたす x, y に対し，Q で x が y の左にあれば，R でも x が y の左になければならない．

逆に，任意の (x, y) に対しこの条件を満たすとき，Q と R の間の転倒数を減らすようなスワップを繰り返すことで Q を R にすることができる．

次の問題を解けばよい：

Problem. 順列 P が与えられる．N 頂点のグラフを作り，$|i-j| < K$ かつ $P_i < P_j$ のとき i から j への辺をはる．このグラフの辞書順最小の topological なラベル付けを求めよ．(s から t への変があるとき，$label(s) < label(t)$ を満たさなければならない．)

一般の DAG に対し，辞書順最小の topological なラベル付けを求めることができる．

ラベル N のつく頂点の出次数は 0 でなければならない．そのような頂点が複数個ある場合は，最も右にある出次数 0 の頂点 (r とする) にラベル N をつければよいことが証明できる．

r のラベルを $x < N$ とすると，r のラベルを N に変えラベル $x+1, \ldots, N$ を $x, \ldots, N-1$ に変えることで辞書順に小さくできるので矛盾する．

よって，次のようにすればよい．

- $k = N$ とする．
- 出次数 0 の頂点のうち最も右にあるものを選び，ラベル k をつける．
- その頂点を取り除き，k を 1 減らして繰り返す．

i 番目の要素を，$P_{i-K+1}, \ldots, P_{i_K-1}$ の中で最大値であるとき極大であると呼ぶことにすると，(出次数が 0 であることに対応する)

$k = N, \ldots, 1$ に対し

- 順列の中で最も右にある極大な要素を見つける．
- その場所に k と書き，その要素を取り除く．

とすればよい．

これは $O(N^2)$ となるので高速化する必要がある．

まず，配列を長さ K ずつのバケットに分割しておく．極大な値はそのバケットの中で必ず最大となっているので，次のような方法でこの問題を解くことができる．

まず，Range Minimum Query (区間の最大値と一点の値の更新を $O(logN)$ でする) を用意しておく．これにより，ある値が極大であるかどうかの判定を $O(logN)$ でできる．

次に，各バケットに対し最大値を求めておき，それが極大である場合は priority queue にいれておく．この priority queue は極大値の集合を含む．

$k = N, \ldots, 1$ に対し，

- priority queue の先頭を取り出しラベル k をつける．
- この要素を削除する．削除によりこのバケットと左右のバケットの三つが影響される可能性がある．
- その三つのバケットに対し，最大値を再び求め，極大である場合は priority queue に追加する．

このアルゴリズムの計算量は $O(NlogN)$ である．

AtCoder Grand Contest 001 Editorial

July 16, 2016

A. BBQ Easy

In this task, you need to divide the given numbers into N pairs. If the pairs are $(a_1, b_1), \ldots, (a_N, b_N)$, the score is $min(a_1, b_1) + \cdots + min(a_N, b_N)$, and your objective is to maximize this score.

Suppose that x is the minimum number among the input, and y is paired with x. Regardless of y, the score of this pair is always x, so y should be the second smallest number in order to maximize the score from other pairs. Thus, the smallest number should be paired with the second smallest number. Similarly, the third smallest number (the minimum after excluding the first pair) should be paired with the fourth, the fifth should be paired with the sixth, ... and so on.

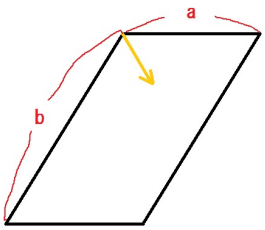
In summary, the solution is as follows: First, let's sort the input numbers in ascending order (i.e., $L_1 \leq \cdots \leq L_{2N}$). Then the answer is $min(L_1, L_2) + min(L_3, L_4) + \cdots + min(L_{2N-1}, L_{2N})$.

B. Mysterious Light

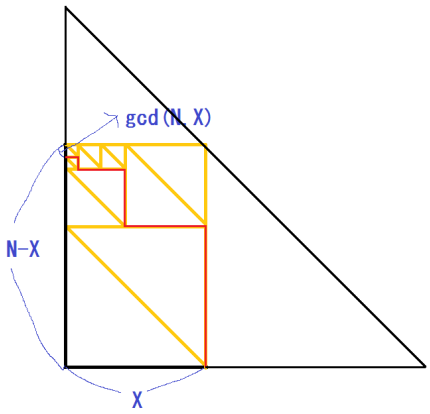
Let $f(a, b)$ be the length of the trajectory when the ray is used in the following parallelogram.

The answer is $N + f(N - X, X)$, and when $a < b$ we can see that $f(a, b) = 2a + f(a, b - a)$. This will lead to an $O(N)$ solution and you can get 300 points.

In order to get the full score, you need to do something similar to Euclid's algorithm of gcd. When $a < b$ and b is not divisible by a, $f(a, b) = 2 * floor(b/a) * a + f(a, b\%a)$. This will lead to an $O(logN)$ solution.



Actually, there is an even simpler solution. The trajectory is a set of triangles, and from the picture below you can see that the sum of lengths of sides is $N - gcd(N, X)$, so the answer is simply $3(N - gcd(N, X))$.



C. Shorten Diameter

We use the following well-known fact about trees.

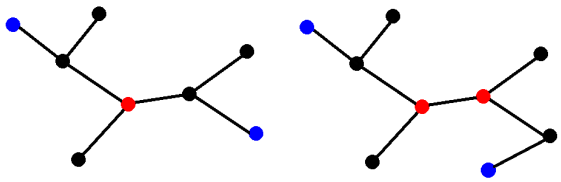
Let T be a tree, and let D be the diameter of the tree.

- If D is even, there exists an vertex v of T such that for each vertex w in T, the distance between w and v is at most $D/2$.
- If D is odd, there exists an edge e of T such that for each vertex w in T, the distance between w and one of the endpoints of e is at most $(D-1)/2$.

Here v and e are called *centers* of the tree.

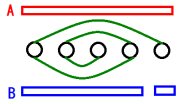
The proof of this fact is not very hard. See the picture below. The blue vertices are the endpoints of the diameters, and the red vertex (or edge) is in the middle of the diameter. This red vertex is the center of the tree; if there is a vertex v such that $dist(v, red) > D/2$, the distance between v and one of blue points will be more than D (because the distance between the red point and each blue point is $D/2$). The proof for odd case is similar.

Now the problem can be solved in the following way (we only describe the solution for the even case, but the odd case is similar). Choose a vertex x in the tree (this will be the center after removal of vertices) and count the number of vertices y such that $dist(x, y) > D/2$. If we remove all such y, the diameter of the remaining graph will be at most D. Thus, we can try all N vertices as x and the answer is the minimum count of such y. The solution works in $O(N^2)$.

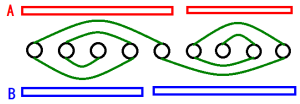


D. Arrays and Palindrome

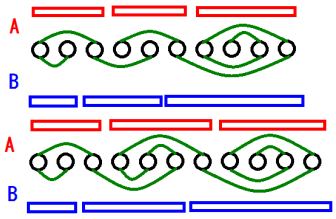
The easiest case is $M = 1$. If $a = \{x\}$ and $b = \{x - 1, 1\}$, the sequences satisfy the condtion as the following picture shows:



When $M = 2$, similar construction works. If $a = \{x, y\}$, choose $b = \{x - 1, y + 1\}$.



What happens when $M \geq 3$ if we construct b in the same manner? I.e., for $a = \{x_1, \ldots, x_M\}$, choose $b = \{x_1 - 1, x_2, \ldots, x_{M-1}, x_M + 1\}$. It depends on the parity of the length of the segment in the middle. When this is even, it works, but when this is odd, it doesn't work.



So, the problem looks difficult when there are many odd elements in a.

When a contains more than two odd numbers, we can prove that there is no solution. Suppose that the sequence a contains O_a odd numbers. Then, the number of arcs that connect two positions is $(N - O_a)/2$. Similarly, if b contains O_b odd numbers, the number of arcs is $(N - O_b)/2$.

In order to satisfy the condition, the arcs must connect all positions. This is the same as connecting N vertices of a graph - and we need at least $N - 1$ arcs.

Thus, $(N - O_a)/2 + (N - O_b)/2 \geq N - 1$, or $O_a + O_b \leq 2$ must be satisfied, but this is impossible when $O_a > 2$.

When a contains at most two odd numbers, we can shuffle the elements of a such that all odd numbers are at the leftmost or the rightmost position. Then the construction we saw above will work.

E. BBQ Hard

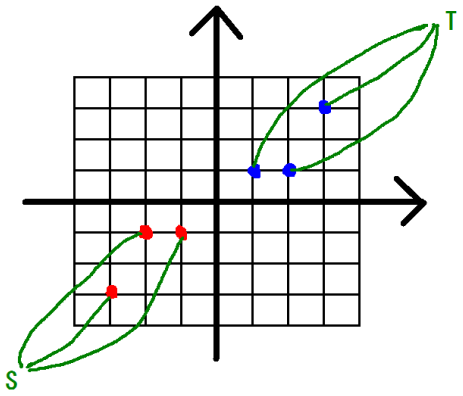
When we choose the i-th pack and the j-th pack, we get $A_i + A_j$ pieces of beef and $B_i + B_j$ pieces of green pepper. There are $f(A_i + A_j, B_i + B_j)$ ways to make a skewer meal using these packs, where $f(x, y) = (x + y)!/x!/y!$ (the number of ways to arrange x identical items and y identical items).

Thus, we want to compute the following sum:

$$\sum_{1 \leq i < j \leq N} f(A_i + A_j, B_i + B_j) \tag{1}$$

The key observation in this task is to find another combinatorial way to define $f(x, y)$. This is the number of ways to go from $(0, 0)$ to (x, y) in a grid. More importantly, $f(A_i + A_j, B_i + B_j)$ is the number of ways to go from $(-A_i, -B_i)$ to (A_j, B_j).

In the following picture, we plot the points $(-A_i, -B_i)$ (red) and the points (A_i, B_i) (blue). If we compute the number of paths from one of red points to one of blue points, we almost get the answer. (In the grid, black edges can be passed only to the right or to the up.)



We need a slight modification. Let X be the number of paths from one of red points to one of blue points, then

$$X = \sum_{1 \le i,j \le N} f(A_i + A_j, B_i + B_j) \tag{2}$$

The answer is

$$\sum_{1 \le i < j \le N} f(A_i + A_j, B_i + B_j) = (X - \sum_i f(A_i + A_i, B_i + B_i))/2 \tag{3}$$

The value of X is the number of paths from S to T in the picture, so it can be computed in $O(MAX^2)$ (where MAX is the maximum coordinate) using a simple DP. The function f can be compute in O(1) with proper pre-calculation. In total, the solution works in $O(MAX^2 + N)$.

F. Wide Swap

Let Q be the inverse of the array P. That is, for each i, $Q_{P_i} = i$. The problem can be restated as follows:

Problem. You are given a permutation Q. The two adjacent elements Q_i and Q_{i+1} can be swapped when $|Q_i - Q_j| \geq K$. You want to move 1 to the leftmost possible position. In case of tie, you want to move 2 to the leftmost possible position. In case of tie, ... and so on. What is the final array?

First, in this version, let's check whether the array Q can be changed into another given array R. Let x and y be two distinct numbers such that $|x-y| < K$. x and y can never be swapped, so if x is to the left of y in Q, x must be to the left of y in R too.

On the other hand, if the relative positions between all such pairs (x, y) are the same in Q and R, we can always change Q into R. This is because when Q and R satisfy this condition and Q and R are not the same, we can always swap two adjacent numbers in Q and reduce the inversion number, and Q will become R after finite number of swaps. (The inversion number is the number of pairs (s, t) such that s is to the left of t in Q but s is to the right of t in R).

Thus, the original problem is equivalent to the following:

Problem. You are given a permutation P. Construct a graph with N vertices, and add an edge from i to j if $|i - j| < K$ and $P_i < P_j$. Find the lexicographically smallest topological labelling of this graph. (In topological labelling, if there is an edge from s to t, $label(s) < label(t)$ must be satisfied.)

There is a general algorithm for finding the smallest topological labelling.

First, determine the vertex that is labelled with N. The out-degree of this vertex must be zero. When there are multiple such vertices, any of those vertices can be labelled with N to find a topological labelling. We can prove that in the lexicographically smallest one, the rightmost vertex among those vertices (let's call it r) should be labelled with N.

Suppose that r is labelled with x. Then, change the label of r to N and change the labels of vertices that are currently labelled as $x + 1, \ldots, N$ to $x, \ldots, N - 1$. This gives a lexicographically smaller topological labelling, so we get a contradiction.

Thus, the following algorithm works:

- Let $k = N$.
- Find the rightmost vertex with out-degree zero, and label it k.
- Remove the vertex.
- Decrement k and go to the second step.

Now, the problem becomes even simpler.

The element at position i is called *local maximum* if P_i is the biggest among $P_{i-K+1}, \ldots, P_{i_K-1}$. (This corresponds to the condition that the out-degree must be zero.)

For each $k = N, \ldots, 1$,

- Find the rightmost local maximum element.
- Label it with k.
- Remove the element.

The straightforward implementation of this algorithm will lead to $O(N^2)$ solution, and we need an efficient data structure to find local maximum elements quickly.

Let's divide the array into buckets. The first bucket contains positions $0, \ldots, K-1$, the second bucket contains positions $K, \ldots, 2K-1$, and so on. Notice that each local maximum element must be the maximum in its bucket.

Thus, the following algorithm works.

First, construct a data structure that supports range minimum query. That is, we can perform the following two types of operations in $O(logN)$:

- For a given position p and a given number x, update the value at position p to x.
- For a given range $[L, R]$, compute the maximum in the indices $[L, R]$.

With this data structure, we can check if a given element is a local maximum or not in $O(logN)$.

Then, for each bucket compute the maximum, and in case it is a local maximum, push it in a priority queue. This priority queue contains the set of local maximum elements.

For each $k = N, \ldots, 1$,

- Pop an element from the priority queue and label this with k.
- Remove this element. Note that the removal of this element affects at most three buckets: the bucket that contains this element and two adjacent buckets.
- For each of those three buckets, recompute the maximum, and in case it is a local maximum, push it into the priority queue.

The time complexity of this solution is $O(NlogN)$.