

NIKKEI Programming Contest 2019-2 解説

yutaka1999

2019 年 11 月 09 日

For International Readers: English editorial starts on page 7.

A: Sum of Two Integers

求める場合の数は、 N が偶数のときは $\frac{N}{2} - 1$ 通り、 N が奇数のときは $\frac{N-1}{2}$ 通りです。C++ での実装例を以下に示します。

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4     int N;
5     cin >> N;
6     cout << (N-1)/2 << endl;
7     return 0;
8 }
```

B: Counting of Trees

条件をみたす木が存在するには $D_1 = 0$ が必要なので、そうでない場合答えは 0 です。以下 $D_1 = 0$ とします。

頂点 1 と頂点 i の距離が D_i となるような木を 1 つ考え、頂点 1 以外の各頂点 i に対して、隣接する頂点のうち頂点 1 に最も近い頂点を p_i とすると、 $D_{p_i} = D_i - 1$ となることがわかります。一方で、 $D_{p_i} = D_i - 1$ ($i \geq 2$) となるような (p_2, \dots, p_N) を定めれば、頂点 i と頂点 p_i を結ぶ辺からなる木が一意に定まります。よって、求める場合の数は $D_{p_i} = D_i - 1$ ($i \geq 2$) なる (p_2, \dots, p_N) の個数となります。 $D_i = k$ となる i の個数を c_k としたとき、その個数は各 2 以上の整数 i に対して c_{D_i-1} を掛け合わせた値となるので、線形時間で求めることができます。

C: Swaps

まず適当に要素を並べかえることで $B_1 \leq B_2 \leq \dots \leq B_N$ であるとしてよいです。 (A_1, \dots, A_N) を $(A_{p_1}, \dots, A_{p_N})$ と並べ替えるとします。最終的に $A_i \leq B_i$ になるという条件と swap の回数は $N - 2$ 回までであるという条件をそれぞれ言い換えると、 (p_1, \dots, p_N) のみたすべき条件は以下のようになります。

- $A_{p_i} \leq B_i$
- 置換 (p_1, \dots, p_N) をサイクルに分解したときに、2 つ以上のサイクルに分解される。

まず (p_1, \dots, p_N) を $A_{p_1} \leq A_{p_2} \leq \dots \leq A_{p_N}$ となるようにとります。このとき $A_{p_i} \leq B_i (i = 1, \dots, N)$ が成り立たなければ、1 つめの条件をみたす (p_1, \dots, p_N) が存在しないので、答えは No です。 $A_{p_i} \leq B_i (i = 1, \dots, N)$ となる場合を考えます。このとき (p_1, \dots, p_N) が 2 つめの条件をみたすならば答えは Yes です。よって、 (p_1, \dots, p_N) がサイクル分解したときに 1 つのサイクルからなる場合を考えればよいです。ここで、 $B_i < A_{p_{i+1}} (i = 1, \dots, N - 1)$ であるならば、1 つめの条件をみたす置換が (p_1, \dots, p_N) のみしか存在しないので、答えは No です。一方で、 $B_i \geq A_{p_{i+1}}$ なる i が存在すれば、 $(p_1, \dots, p_{i-1}, p_{i+1}, p_i, p_{i+2}, \dots, p_N)$ という置換がいずれの条件もみたすので、答えは Yes です。

以上によりすべての場合が尽くされているので、答えを求めることができます。時間計算量はソートを行うので $O(N \log N)$ です。

D: Shortest Path on a Line

頂点 1 から頂点 i への最短路の長さを d_i とすると $d_1 \leq d_2 \leq \dots \leq d_N$ であることが分かります。よって、各 i に対して頂点 $i+1$ から頂点 i に長さ 0 の辺を追加したとしても、頂点 1 から各頂点への最短路の長さは変化しません。以下ではこれらの辺はすべてグラフに含まれているものとします。このとき i 回目に追加される辺について考えると、頂点 L_i から頂点 R_i への長さ C_i の辺を追加すれば、その他の辺は最短路に影響しません。これは、 $L_i \leq s < t \leq R_i$ なる頂点 s, t に対して、 s から L_i に長さ 0 のパスがあり、 L_i から R_i に長さ C_i の辺があり、 R_i から t に長さ 0 のパスがあるため、これらを合わせて s から t に長さ C_i のパスがあることから分かります。以上により、 $N + M - 1$ 本の辺のみを考えることで頂点 1 から各頂点への最短路が求まることが分かります。よって、Dijkstra 法を用いることで $O((N + M) \log N)$ の時間計算量で解くことができます。

E: Non-triangular Triplets

条件をみたすような3つ組への分割 $(a_1, b_1, c_1), \dots, (a_N, b_N, c_N)$ が存在するとします。このとき $\sum_{i=1}^N (a_i + b_i) \leq \sum_{i=1}^N c_i$ となるので、 $2 \sum_{i=1}^N c_i \geq \sum_{i=1}^N (a_i + b_i + c_i) = \sum_{i=K}^{K+3N-1} i = \frac{3N(2K+3N-1)}{2}$ となります。 $\sum_{i=1}^N c_i \leq \sum_{i=K+2N}^{K+3N-1} i = \frac{N(2K+5N-1)}{2}$ なので、 $N(2K+5N-1) \geq \frac{3N(2K+3N-1)}{2}$ つまり $2K-1 \leq N$ が必要と分かります。逆に $2K-1 \leq N$ のとき、条件をみたすような3つ組への分割が存在することを示します。 N の偶奇で構成が少し変わるのですが、本質的には同じであるので、ここでは N が奇数の場合のみ構成します。 $N = 2L-1$ ($K \leq L$) とします。このとき、 (a_1, \dots, a_N) を $(K, K+2, K+4, \dots, K+2L-2, K+1, K+3, \dots, K+2L-3)$ とし、 (b_1, \dots, b_N) を $(K+3L-2, K+3L-3, \dots, K+2L-1, K+4L-3, K+4L-4, \dots, K+3L-1)$ とし、 (c_1, \dots, c_N) を $(K+4L-2, \dots, K+6L-4)$ とすればよいです。このようにして、 (a, b) を $(x, y), (x+2, y-1), \dots$ と並べることによって、 $a+b$ を1ずつ増やしていくことができ、無駄の少ない構成をすることができます。

F: Mirror Frame

まず各電球の on と off が決まっているとき、その盤面がきれいかどうかを判定する方法を考えます。まず、各格子点を座標の和の偶奇によって 2 色に塗り分けます。このとき、ある格子点のライン上にある格子点の色はすべて等しいので、各色に対して独立に判定すればよいです。さらに、正方形の対角線に関して対称な位置にある電球の状態は等しい必要があるので、それらの条件をまとめることにより、以下のような問題に帰着されます。

n 頂点からなる完全グラフが与えられる。各辺には on と off の状態がある。ある 2 頂点を選び、それらの少なくとも一方が端点であるような辺すべての on と off を切り替える、という操作を繰り返してすべての辺を off にすることができるか判定せよ。

n が偶数のときは常に可能であり、 n が奇数のときは、各頂点に対し、隣接する on の辺が偶数個であることが条件であることが示せます。 n が奇数の場合、各頂点に対し、一度の操作で on と off が入れ替わる辺の個数が常に偶数なので、必要性は従います。また、これらの場合に実際にすべての辺を off にすることができるのは、適当に操作を行うことで示せます。(3 頂点 a, b, c を選んで、 (a, b) , (b, c) , (c, a) に対して操作を行うと、 (a, b) , (b, c) , (c, a) を結ぶ辺のみの on と off が切り替えることなどが有用です。)

よって、後は上の条件をみたすように各格子点に on と off を割り当てればよいです。先ほどのようにグラフの言葉で書きなおすと以下ようになります。

n 頂点からなる完全グラフが与えられる。各辺の on と off の状態をうまく定めることにより、すべての頂点に対して、on の状態にあって隣接する辺が偶数個あるようにする方法は何通りあるか。ただし、いくつかの辺の状態はすでに定まっている。

これは各連結成分ごとに独立に解くことができます。計算量は $O(N^2)$ です。

A: Sum of Two Integers

The answer is $\frac{N}{2} - 1$ if N is even, and $\frac{N-1}{2}$ if N is odd. Sample C++ implementation follows:

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4     int N;
5     cin >> N;
6     cout << (N-1)/2 << endl;
7     return 0;
8 }
```

B: Counting of Trees

In order for a tree satisfying the condition to exist, $D_1 = 0$ must hold. Otherwise, the answer is 0. Below, we assume $D_1 = 0$.

Consider a tree such that the distance between Vertex 1 and Vertex i is D_i , and for each vertex i other than Vertex 1, let the vertex closest to Vertex 1 among the vertices adjacent to Vertex i be Vertex p_i . Then, we can see $D_{p_i} = D_i - 1$. On the other hand, if we set the values of (p_2, \dots, p_N) so that $D_{p_i} = D_i - 1$ ($i \geq 2$), we have a unique tree formed by edges connecting Vertex i and Vertex p_i . Thus, the answer is the number of sequences (p_2, \dots, p_N) such that $D_{p_i} = D_i - 1$ ($i \geq 2$). Let c_k be the number of indices i such that $D_i = k$. Then, the number of sequences is the product of c_{D_i-1} over all i from 2 to N , which can be found in linear time.

C: Swaps

First, we assume $B_1 \leq B_2 \leq \dots \leq B_N$ by properly arranging the elements beforehand. Now, let us say we do the swaps and permute (A_1, \dots, A_N) into $(A_{p_1}, \dots, A_{p_N})$. By rephrasing each of the restrictions that we must have $A_i \leq B_i$ in the end and that we can do at most $N - 2$ swaps, we see that (p_1, \dots, p_N) must satisfy the following conditions:

- $A_{p_i} \leq B_i$
- The permutation (p_1, \dots, p_N) can be decomposed into two or more cycles.

Let (p_1, \dots, p_N) be a sequence such that $A_{p_1} \leq A_{p_2} \leq \dots \leq A_{p_N}$. If $A_{p_i} \leq B_i (i = 1, \dots, N)$ does not hold, there is no sequence (p_1, \dots, p_N) that satisfies the first condition above, so the answer is No. Now, consider the case $A_{p_i} \leq B_i (i = 1, \dots, N)$. If (p_1, \dots, p_N) satisfies the second condition above, the answer is Yes. What remains is the case that (p_1, \dots, p_N) consists of one cycle. If $B_i < A_{p_{i+1}} (i = 1, \dots, N - 1)$, (p_1, \dots, p_N) is the only permutation satisfying the first condition, so the answer is No. If there exists i such that $B_i \geq A_{p_{i+1}}$, the permutation $(p_1, \dots, p_{i-1}, p_{i+1}, p_i, p_{i+2}, \dots, p_N)$ satisfies both conditions, so the answer is Yes.

Now we have covered all the cases. The time complexity of our solution is $O(N \log N)$, which comes from sorting.

D: Shortest Path on a Line

Let d_i be the length of the shortest path from Vertex 1 to Vertex i . Then, we can see $d_1 \leq d_2 \leq \dots \leq d_N$. Thus, we can add an edge of length 0 from Vertex i to Vertex $i + 1$ for each i without changing those lengths. Below, we assume that the graph contains all of those edges.

Now, consider the edges added in the i -th operation. If we only add the edge of length C_i from Vertex L_i to Vertex R_i , the other edges does not affect the shortest paths. This is because, for a pair of vertices s and t such that $L_i \leq s < t \leq R_i$, we have a path of length 0 from s to L_i , an edge of length C_i from L_i to R_i , and a path of length 0 from R_i to t , so in total we have a path of length C_i from s to t . Therefore, we can find the shortest paths from Vertex 1 to each of the other vertices by only considering $N + M - 1$ edges, which can be done in $O((N + M) \log N)$ time with Dijkstra's algorithm.

E: Non-triangular Triplets

Assume that there exists a partition into triples $(a_1, b_1, c_1), \dots, (a_N, b_N, c_N)$ satisfying the condition. Then, $\sum_{i=1}^N (a_i + b_i) \leq \sum_{i=1}^N c_i$ holds, so we have $2 \sum_{i=1}^N c_i \geq \sum_{i=1}^N (a_i + b_i + c_i) = \sum_{i=K}^{K+3N-1} i = \frac{3N(2K+3N-1)}{2}$. Since $\sum_{i=1}^N c_i \leq \sum_{i=K+2N}^{K+3N-1} i = \frac{N(2K+5N-1)}{2}$, we see that it is necessary for $N(2K+5N-1) \geq \frac{3N(2K+3N-1)}{2}$, that is, $2K-1 \leq N$, to hold. On the other hand, if $2K-1 \leq N$, we can show that there exists a partition satisfying the condition. The construction slightly depends on the parity of N , but not essentially, so here we will only cover the case N is odd. Let $N = 2L - 1$ ($K \leq L$). Now, let (a_1, \dots, a_N) be $(K, K+2, K+4, \dots, K+2L-2, K+1, K+3, \dots, K+2L-3)$, let (b_1, \dots, b_N) be $(K+3L-2, K+3L-3, \dots, K+2L-1, K+4L-3, K+4L-4, \dots, K+3L-1)$, and let (c_1, \dots, c_N) be $(K+4L-2, \dots, K+6L-4)$, and we have a solution. The efficiency of this construction comes from arranging (a, b) in the order $(x, y), (x+2, y-1), \dots$, which allows $a+b$ to increase by 1 at a time.

F: Mirror Frame

Let us first find out the way to determine whether the board is beautiful when the state of each bulb is fixed. We paint the grid points in two colors according to the parity of the sum of the coordinates of each point. Then, all the grid points on the path for some grid point have the same color, so we can examine each color separately. Additionally, two bulbs symmetric with respect to a diagonal of the square need to have the same state. By considering these restrictions all together, it comes down to the following problem:

Given is a perfect graph with n vertices, where each edge has a state - ON or OFF. Determine if we can turn all the edges OFF by repeating the following operation: choose two vertices and swap the states of all the edges incident to at least one of those two vertices.

When n is even, the answer is always Yes. When n is odd, the answer is Yes if and only if, for each vertex, there is an even number of ON edges incident to that vertex. These can be shown as follows. When n is odd, for each vertex, the number of incident edges affected in an operation is always even, from which the necessity follows. To show that we can turn all the edges OFF in those cases, we can actually construct a sequence of operations. (One of the useful facts to do it is that choosing three vertices a , b , and c and doing the operation for (a, b) , (b, c) , and (c, a) only switches the edges (a, b) , (b, c) , and (c, a) .)

Now, what remains is to allocate ON or OFF to each grid point so that the condition above holds. Again in terms of graph theory, we have the following problem:

Given is a perfect graph with n vertices. In how many ways can we assign a state - ON or OFF - to each edge so that, for each vertex, there is an even number of ON edges incident to that vertex? The states of some edges are already fixed.

We can solve this problem separately for each connected component, in a total of $O(N^2)$ time.