ABC 133 解説

drafear, yosupo, potetisensei, yuma000, camypaper, evima

2019 年 7 月 7 日

For International Readers: English editorial starts on page 8.

A: T or T

電車を使った場合は $N \times A$ 円、タクシーを使った場合は B 円かかります。よって、$N \times A$ と B の小さい方を出力すれば良いことになります。これを C++ で実装した例を以下に示します。

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  int N, A, B; cin >> N >> A >> B;
  int ans = min(N * A, B);
  cout << ans << endl;
}
```

B: Good Distance

各 i, j $(i < j)$ について $Y_{ij} = |X_{i1} - X_{j1}|^2 + ... + |X_{iD} - X_{jD}|^2$ を計算します。すると、$k^2 = Y_{ij}$ を満たす整数 k が存在するかを調べれば $\sqrt{Y_{ij}}$ が整数であることを判定できます。$k \leq Y_{ij}$ なので、これは、各 $k = 0, 1, 2, ..., Y_{ij}$ について $k^2 = Y_{ij}$ であるか調べれば十分です[*1]。C++ 言語での実装例を以下に示します。

```
#include <bits/stdc++.h>

using namespace std;

int main() {
int N, D; cin >> N >> D;
vector<vector<int>> X(N, vector<int>(D));
for (int i = 0; i < N; ++i) {
for (int j = 0; j < D; ++j) {
  cin >> X[i][j];
}
}
int ans = 0;
for (int i = 0; i < N; ++i) {
for (int j = i+1; j < N; ++j) {
  int norm = 0;
  for (int k = 0; k < D; ++k) {
    int diff = abs(X[i][k] - X[j][k]);
    norm += diff * diff;
  }
  // check whether norm = k for some k
  bool flag = false;
  for (int k = 0; k <= norm; ++k) {
    if (k * k == norm) {
      flag = true;
    }
  }
  if (flag) ++ans;
}
}
cout << ans << endl;
}
```

[*1] $k^2 \leq Y_{ij}$ であることを使えばより高速に判定できます

C: Remainder Minimization 2019

$R-L$ が非常に大きい時、全ての候補 (i,j) を調べてしまうと、$O((R-L)^2)$ の計算量がかかってしまい、間に合いません。よく考えてみると、2019 で割った余りが同じであるような 2 数 a,b について、$(a\times c) \bmod 2019$ と $(b\times c) \bmod 2019$ は必ず同じ値になります。よって、(i,j) として調べるべき候補は高々 $2019^2 = 4076361$ 通り程度しか存在せず、全て調べても十分間に合います。

また、「$O((R-L)^2)$ の全探索の途中で、解が 0 になった時点で探索を打ち切る」という方法でも解くことが出来ます。これは、R-L が 2019 以上の時、必ず割った余りが 0 となるような値が区間中に存在するため、2019^2 個以上の値を見ることはないからです。

D: Rain flows into dams

i 番目の山に降った雨の量 (答え) を X_i とします。すると、

$$A_1 + ... + A_N = X_1 + ... + X_N$$

が成り立ちます。この和を S とおきます。

また、便宜上 $X_{N+1} = X_1$ とすると

$$\frac{X_i + X_{i+1}}{2} = A_i$$

より

$$X_i + X_{i+1} = 2A_i$$

です。

したがって、

$$X_1 = S - (X_2 + ... + X_N) = S - 2(A_2 + A_4 + ... + A_{N-1})$$

と X_1 を求めることができます。同様に $X_2, X_3, ..., X_N$ を求めることができますが、$O(N^2)$ の時間はかけられないため、実装が少し大変です。

そこで、$X_i + X_{i+1} = 2A_i$ を利用し、$X_{i+1} = 2A_i - X_i$ の漸化式に従って $X_2, X_3, ..., X_N$ を前から順に計算していきます。

このアルゴリズムは時間計算量 $O(N)$ で動作し、十分間に合います。

E. Virus Tree 2(writer : yuma000)

今回の条件は以下のように言い換えられます。

- 全ての頂点に対して、以下の条件が成り立つ。
  - 頂点の色と、それと直接辺で繋がっている全ての頂点の色は、それぞれ異なる。

頂点 0 を根とする根付き木として考えます。すると、以下が成り立つことが分かります。

- ある頂点 x とその親が既に塗られているとき、x の子の数を c_x とすると、x の子の塗り分け方は ${}_{K-2}P_{c_x}$ 通りである。

よって、頂点 0 から DFS 的に色を塗っていくことで、木の塗り分け方の総数を求めることができます。頂点 0 には親がないことに注意して下さい。

以下が、サンプルコードです。

```
#include<iostream>
#include<vector>
using namespace std;

const long long int mod=1e9+7;


long long int dfs(int K,const vector<vector<int>>&graph, int now,int from) {

  int can_use_color_num;
  if (from==-1) {
    can_use_color_num=K-1;
  }
  else {
    can_use_color_num=K-2;
  }
  if (K<graph[now].size()) {
    return 0;
  }
  else {
    long long int case_num=1;
    for (auto e : graph[now]) {
      if(e==from)continue;

      case_num*=can_use_color_num;
      can_use_color_num--;
      case_num%=mod;
    }
    for (auto e : graph[now]) {
      if(e==from)continue;
```

```

      case_num *=dfs(K,graph,e,now);
      case_num %= mod;
    }
    return case_num;
  }
}



int main() {
  int N,K;cin>>N>>K;

  vector<vector<int>>graph(N);
  for (int i = 0; i < N - 1; ++i) {
    int a,b;cin>>a>>b;a--;b--;

    graph[a].push_back(b);
    graph[b].push_back(a);
  }

  long long int answer=K*dfs(K,graph,0,-1);
  answer%=mod;
  cout<<answer<<endl;
  return 0;
}
```

F: Colorful Tree

与えられた木を、頂点 1 を根とする根付き木とみなします。また、任意の頂点 w に対し、根から頂点 w までの距離を $dist_w$ とします。このとき、二頂点 $u.v$ 間の距離は u, v の 最近共通祖先 (英語版 Wikipedia の記事) を a として $dist_u + dist_v - 2dist_a$ と書けます。よって、各問 i に対して次の二つの部分問題を解けばよいことになります。

1. 二頂点 u_i, v_i の最近共通祖先 a_i を求める。
2. 色 x_i のすべての辺の長さが y_i に変更されたときの $dist_{u_i}, dist_{v_i}, dist_{a_i}$ を求める。

部分問題 1 は有名問題です。以下、よく知られている解法を紹介します。任意の頂点 w に対し、$par(w, 0)$ を w、$par(w, 1)$ を w の親、$par(w, 2)$ を w の親の親、... とします (根の親は根とします)。まず、事前にすべての頂点 w と整数 $i = 0, 1, \ldots, \lfloor \log_2 N \rfloor$ に対して $par(w, 2^i)$ を求めておきます。これは、$par(w, 2^{i+1}) = par(par(w, 2^i), 2^i)$ に注意すると $\mathrm{O}(N \log N)$ 時間で行えます。また、すべての頂点 w に対してその深さ $depth_w$ (根までの辺数) も事前に求めておきます。これらを元に、以下のようにして二頂点 u_i, v_i の最近共通祖先をそれぞれ $\mathrm{O}(\log N)$ 時間で求めることができます。

1. $a = u_i, b = v_i$ とする。
2. $depth_a < depth_b$ なら $b = par(b, depth_b - depth_a)$、そうでなければ $a = par(a, depth_a - depth_b)$ とする。
3. $a = b$ なら a を返す。
4. $i = \lfloor \log_2 N \rfloor, \lfloor \log_2 N \rfloor - 1, \ldots, 1, 0$ に対してこの順に次を行う: $par(a, 2^i) \neq par(b, 2^i)$ なら $a = par(a, 2^i), b = par(b, 2^i)$ とし、そうでなければ何もしない。
5. $par(a, 1)$ を返す。

部分問題 2 を解くには、まず事前にすべての頂点 w に対して辺の長さが変更される前の $dist_w$ を求めておきます。色 x_i のすべての辺の長さが y_i に変更されると、$dist_w$ は $num(x_i, w) \times y_i - sum(x_i, w)$ だけ増加します。ここで $num(x_i, w)$ は根と頂点 w の間に存在する色 x_i の辺の数、$sum(x_i, w)$ はそれらの辺の元の長さの総和です。

$num(x_i, w), sum(x_i, w)$ を求めるには、まず木の $N - 1$ 本の辺を DFS を行う際に通過する順に並べたリスト (オイラーツアー) を作ります。例えば、入力例 1 の木に対するオイラーツアーの例は次のようになります: $+e_1, +e_3, -e_3, +e_4, -e_4, -e_1, +e_2, -e_2$。ここで、$i$ 番目の辺を根から遠ざかる方向にたどる際に $+e_i$、根に近づく方向にたどる際に $-e_i$ と表記しています。このリストの e_i を i 番目の辺の元の長さで置き換えた数列を考えると、w と w の親を結ぶ辺が j 番目の辺であるとして、この数列の先頭から $+e_j$ に対応する要素までの和が根と w の間に存在する辺の元の長さの総和と一致します。この数列を順序を保ったまま辺の色ごとに分割し、事前に先頭から各要素までの和を計算しておけば、色 x_i と頂点 w が指定された際に $sum(x_i, w)$ を二分探索により $\mathrm{O}(\log N)$ 時間で求めることができます。$num(x_i, w)$ についても同様です。

以上の処理を行えば、合計 $\mathrm{O}((N + Q) \log N)$ 時間ですべての問いを処理することができます。

ABC 133 Editorial

drafear, yosupo, potetisensei, yuma000, camypaper, evima

July 7, 2019

A: T or T

If you use train it costs $N \times A$ yen, and if you use taxi it costs B yen. So you have to print the smaller value of either $N \times A$ or B. The following is an example implemented in C++.

```
#include <bits/stdc++.h>

using namespace std;

int main() {
  int N, A, B; cin >> N >> A >> B;
  int ans = min(N * A, B);
  cout << ans << endl;
}
```

B: Good Distance

For each i, j $(i < j)$, calculate $Y_{ij} = |X_{i1} - X_{j1}|^2 + ... + |X_{iD} - X_{jD}|^2$. Then you can determine if $\sqrt{Y_{ij}}$ is an integer by checking if there exists an integer k such that $k^2 = Y_{ij}$. Since $k \le Y_{ij}$, it is sufficient if you judged if $k^2 = Y_{ij}$ for each $k = 0, 1, 2, ..., Y_{ij}$[*1]. The following is an implementation example in C++ language.

```
#include <bits/stdc++.h>

using namespace std;

int main() {
int N, D; cin >> N >> D;
vector<vector<int>> X(N, vector<int>(D));
for (int i = 0; i < N; ++i) {
for (int j = 0; j < D; ++j) {
  cin >> X[i][j];
}
}
int ans = 0;
for (int i = 0; i < N; ++i) {
for (int j = i+1; j < N; ++j) {
  int norm = 0;
  for (int k = 0; k < D; ++k) {
    int diff = abs(X[i][k] - X[j][k]);
    norm += diff * diff;
  }
  // check whether norm = k for some k
  bool flag = false;
  for (int k = 0; k <= norm; ++k) {
    if (k * k == norm) {
      flag = true;
    }
  }
  if (flag) ++ans;
}
}
cout << ans << endl;
}
```

[*1] If you used the fact that $k^2 \le Y_{ij}$, you can judge more fast

C: Remainder Minimization 2019

When $R-L$ is very big, trying all candidates of (i,j) needs time complexity $O((R-L)^2$ and does not finish in time. In fact, if two integers a and b have same remainder when divided by 2019, $(a \times c) \bmod 2019$ and $(b \times c) \bmod 2019$ is always the same. Therefore, there are at most $2019^2 = 4076361$ candidats that need to be checked, so it will finish in time if you checked all the candidates.

Also, you can solve the problem with the strategy that "Abort the $O((R-L)^2)$ brute-forcing if one of the solution was 0." This works in time because, if R-L is more than or equal to 2019, there always exist a number whose remainder of 2019 is 0 in the segment so it inspects no more than 2019^2 pairs of values.

D: Rain flows into dams

Let X_i be the amount of rain that the i-th mountain received (the answer). Then it holds that

$$A_1 + ... + A_N = X_1 + ... + X_N$$

. Let S be this sum.

Also, let $X_{N+1} = X_1$ for simplicity, then

$$\frac{X_i + X_{i+1}}{2} = A_i$$

holds, so it holds that

$$X_i + X_{i+1} = 2A_i$$

.

Therefore, you can find X_1 with the equation

$$X_1 = S - (X_2 + ... + X_N) = S - 2(A_2 + A_4 + ... + A_{N-1})$$

. Similarly, you can calculate $X_2, X_3, ..., X_N$, but you cannot spend $O(N^2)$ time, so it's a bit troublesome to implement.

Since $X_i + X_{i+1} = 2A_i$, you can use the recurrence relation $X_{i+1} = 2A_i - X_i$ and calculate $X_2, X_3, ..., X_N$ one by one from the beginning.

This algorithm works in $O(N)$, so it's sufficiently fast.

E. Virus Tree 2(writer : yuma000)

This time, the condition can be rephrased as follows:

- For all vertices, the following condition holds:
  - The color of the vertex and the colors of the vertices directly connected to it are distinct.

Let's regard it as a rooted tree whose root is vertex 0. Then the following property can be found:

- If a vertex x and its parent is already painted, letting c_x be the number of children of x, the way of painting the children of x is ${}_{K-2}P_{c_x}$.

Therefore, by painting the vertices in DFS-like order from vertex 0, you can calculate the number of ways of painting the tree. Note that vertex 0 doesn't have parent.

The following is a sample code.

```cpp
#include<iostream>
#include<vector>
using namespace std;

const long long int mod=1e9+7;


long long int dfs(int K,const vector<vector<int>>&graph, int now,int from) {

  int can_use_color_num;
  if (from==-1) {
    can_use_color_num=K-1;
  }
  else {
    can_use_color_num=K-2;
  }
  if (K<graph[now].size()) {
    return 0;
  }
  else {
    long long int case_num=1;
    for (auto e : graph[now]) {
      if(e==from)continue;

      case_num*=can_use_color_num;
      can_use_color_num--;
      case_num%=mod;
    }
```

```
    for (auto e : graph[now]) {
      if(e==from)continue;

      case_num *=dfs(K,graph,e,now);
      case_num %= mod;
    }
    return case_num;
  }
}



int main() {
  int N,K;cin>>N>>K;

  vector<vector<int>>graph(N);
  for (int i = 0; i < N - 1; ++i) {
    int a,b;cin>>a>>b;a--;b--;

    graph[a].push_back(b);
    graph[b].push_back(a);
  }

  long long int answer=K*dfs(K,graph,0,-1);
  answer%=mod;
  cout<<answer<<endl;
  return 0;
}
```

F: Colorful Tree

Let's regard the given tree as a rooted tree whose root is vertex 1. Also, for each vertex w, let $dist_w$ be the distance from the root to vertex w. Here, the distance between two vertices u, v can be written as $dist_u + dist_v - 2dist_a$, where a is the lowest common ancestor of vertices u, v. Therefore, each query i can be solved by solving the following two subproblems:

1. Find the least common ancestor of two vertices u_i, v_i.
2. Find $dist_{u_i}, dist_{v_i}$ and $dist_{a_i}$ when the length of every edge whose color is x_i is changed to y_i.

Subproblem 1 is a famous problem. The following is a well-known solution. For each vertex w, let $par(w, 0)$ be w, $par(w, 1)$ be the parent of w, $par(w, 2)$ be the parent of the parent of w, and so on (here we assume the parent of root to be the root itself). First, for each vertex w and integer $i = 0, 1, \ldots, \lfloor \log_2 N \rfloor$, precalculate $par(w, 2^i)$. By using the fact that $par(w, 2^{i+1}) = par(par(w, 2^i), 2^i)$, this can be calculated in $O(N \log N)$. Also, for each vertex w, precalculate its depth $depth_w$ (the number of edges to the root). By using them, you can find the least common ancestor of two vertices $u_i and v_i$ with the following steps:

1. Let $a = u_i, b = v_i$.
2. Let $b = par(b, depth_b - depth_a)$ if $depth_a < depth_b$, otherwise $a = par(a, depth_a - depth_b)$.
3. Return a if $a = b$.
4. For each $i = \lfloor \log_2 N \rfloor, \lfloor \log_2 N \rfloor - 1, \ldots, 1, 0$, do the following: let $a = par(a, 2^i), b = par(b, 2^i)$ if $par(a, 2^i) \neq par(b, 2^i)$, and otherwise do nothing.
5. Return $par(a, 1)$.

To solve the subproblem 2, first of all, for each vertex w, calculate $dist_w$ before the length of the edges are changed. After the length of every edge whose color is x_i is changed to y_i, $dist_w$ increases by $num(x_i, w) \times y_i - sum(x_i, w)$, where $num(x_i, w)$ is the number of edges whose color is x_i, and $sum(x_i, w)$ is the sum of the length of those edges.

To find $num(x_i, w), sum(x_i, w)$, first make a list of vertices in the order of DFS traversing (Euler tour). For example, an Euler tour of the tree in Sample Input 1 is like: $+e_1, +e_3, -e_3, +e_4, -e_4, -e_1, +e_2, -e_2$. Here, $+e_i$ denotes moving away from the root, and $-e_i$ denotes getting closer to the root, while passing through the i-th edge. If you substitute each e_i in this sequence with the length of the i-th edge, the sum of the elements of subsequence from the first element to the element corresponding to $+e_j$ is equal to the sum of the length of edges between the root and vertex w if the j-th edge connects two vertices, w and the parent of w. If you split this sequence into subsequences without changing the order so that each subsequence correspond to each color, and precalculate the sum from the beginning to each element, for each specified color x_i and vertex w, you can calculate $sum(x_i, w)$ in time complexity $O(\log N)$ by using binary search. Similarly you can also calculate $num(x_i, w)$.

With the operations above, you can answer to all the queries in $O((N + Q) \log N)$.