ABC 133 解説

drafear, yosupo, potetisensei, yuma000, camypaper, evima

2019 年 7 月 7 日

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。$k \le Y_{ij}$ 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。

```cpp
#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] $k^2 \le Y_{ij}$ 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。