S. 148.

Horcsin Bálint
Budapest, Németh László Gimnázium, 12. o. t.
horcsinbalint@gmail.com

Programnyelv: C++17

1. Megoldás elvi menete

A feladat lényegében a következő: Eltávolítunk egy levelet a gráfból. Mennyi lesz az átmérők minimális száma, és ehhez az eltávolítást hányféleképpen végezhetjük el.
Jelölések: P pont(ok): az átmérő(k) közös pontja(i)
H pont: egy random kiválasztott pont, amelyet nem módosítunk futás közben
A pont: H ponttól egy legtávolabb eső pont
B pont(ok): A ponttól legtávolabb eső pont(ok)

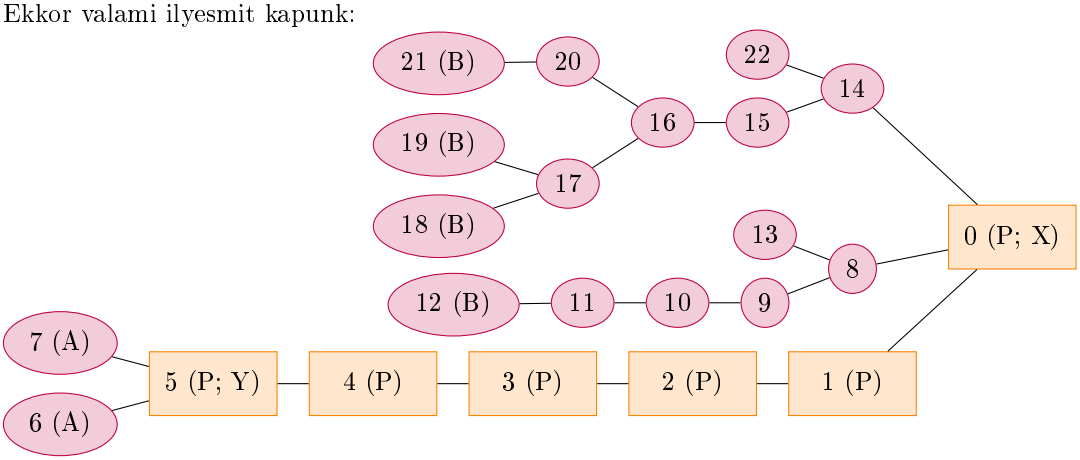
Keressünk meg egy A pontot, amely a fagráf valamely átmérőjének valamely végpontja!
Ezt az ismert módon meg tudjuk tenni. (Egy H ponttól megkeressük a legtávolabbi pontot. I. mélységi bejárás)

Keressük meg B ponto(ka)t! Ezt az A pontból indított mélységi bejárással tudjuk megtenni. Közben figyeljünk rá, hogy melyik az a legtávolabbi X pont A-tól, amely rajta van az összes AB úton! (II. mélységi bejárás)

Ezután egy B pontból indítsunk mélységi bejárást, és keressük meg az A_{tipusu} pontokat! Ezek a B ponttól legtávolabb eső pontok. Közben figyeljünk rá, hogy melyik az a legtávolabbi Y pont B-től, amely rajta van az összes $A_{tipusu}B$ úton! (III. mélységi bejárás)

Legyenek az A_{tipusu} pontok A-k innentől.

Ekkor valami ilyesmit kapunk:



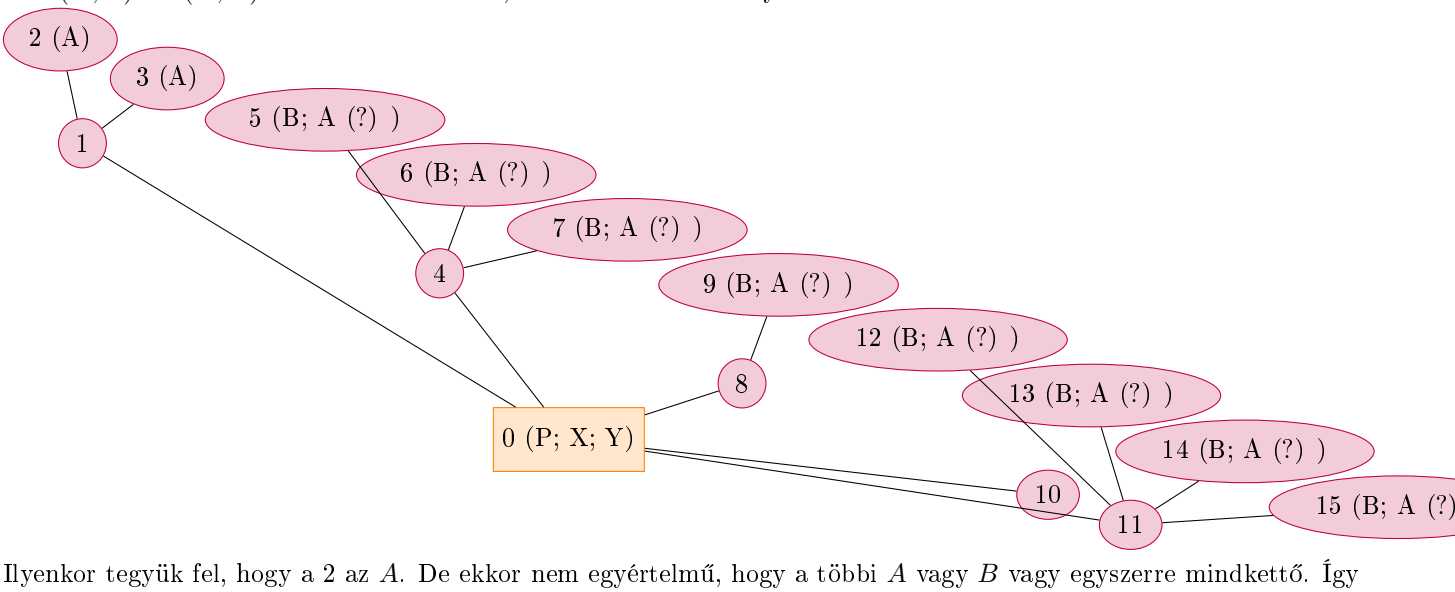
Ismert, hogy biztosan van legalább egy P pont.

Az is ismert, hogy ez(ek) a P pont(ok) az XY úton van(nak) rajta, valamint X és Y is P pontok.

A B pont(ok) távolsága X-től biztosan egyenlő, mivel $d(A,P)$, és $d(A,B)$ nem változnak A és/vagy B függvényében, így $d(A,B)-d(A,X)=d(X,B)$ minden B-re egyforma. (X rajta van az összes AB úton.)

Az A pont(ok) távolsága X-től biztosan egyenlő, mivel $d(B,P)$, és $d(B,A)$ nem változnak A és/vagy B függvényében, így $d(B,A)-d(B,X)=d(X,A)$ minden A-ra egyforma. (X rajta van az összes BA úton.)

Ha $d(X,A)=d(X,B)$ az kicsit kellemetlen, mivel akkor valami ilyesmi lesz az ábra:



Ilyenkor tegyük fel, hogy a 2 az A. De ekkor nem egyértelmű, hogy a többi A vagy B vagy egyszerre mindkettő. Így ezt az esetet vizsgálni szükséges. Ilyenkor $X=Y$.

Legyen DB_A az A pontok száma, DB_B pedig a B pontok száma. Tudjuk, hogy egy fának van átmérője, így $DB_A > 0$ és $DB_B > 0$.

Csináljunk esetszétválasztást.

Ha $DB_A = 1$ és $DB_B > 1$, akkor, ha A-t távolítjuk el, akkor a keletkező átmérők száma legalább DB_B lesz, ha egy B-t távolítunk el, akkor pontosan $DB_B - 1$ lesz az átmérők száma. Ha se nem valamelyik B-t vagy A-t távolítjuk el, akkor az átmérők száma marad DB_B. Így az egyik B-t érdemes eltávolítani, $DB_B - 1$ átmérő lesz az eredmény, amit DB_B-féleképpen érhetünk el. - Ilyenkor nem léphet fel az előbb ábrázolt probléma.

Ha $DB_A > 1$ és $DB_B = 1$, akkor hasonlóan az egyik A-t kell eltávolítani, így $DB_A - 1$ átmérő lesz az eredmény, amit DB_A-féleképpen érhetünk el.

Ha $DB_A = 1$ és $DB_B = 1$, akkor a következő a helyzet. Ha van másik levél, akkor azok közül bármelyiket eltávolíthatjuk, nem változik az átmérők száma, 1 marad. Az A, illetve B eltávolítását pedig leszimulálhatjuk egy-egy mélységi bejárással, és megnézhetjük, hogy ők eltávolíthatók-e (1 átmérő lesz-e úgy is). Így ilyenkor a válasz biztosan az, hogy 1 átmérő van, a lehetőségeket meg lehet határozni. - Ilyenkor sem léphet fel az előbb ábrázolt probléma.
És, ha nincs levél akkor is elérhető az 1 átmérő, mivel ekkor egy vonal mentén helyezkednek el a pontok, és a szélén lévő A-t vagy B-t eltávolítva is 1 átmérő marad.

Ha $DB_A > 1$ és $DB_B > 1$, akkor ha $X = Y$, akkor az elég kellemetlen. Ekkor meg kell nézni, hogy P-ből nézve melyik részgráf(ok) A vagy B pontját érdemes eltávolítani.
Ha $X \neq Y$, akkor meg kell nézni, hogy az egyik A-t vagy egyik B-t érdemes eltávolítani DB_A és DB_B alapján.

2. Futásidő

A program futásideje $\mathcal{O}(N * log\sqrt{N})$, mivel néhány mélységi bejárás van a programban, amik $\mathcal{O}(N)$-ben lefutnak, de ha $X = Y$, akkor a következő kódrész fut le:

```
p[t2.lastMerge].banned=1;
map<int,int> m;
int mx=0;
long long o=0;
for(int i : p[t2.lastMerge].hova){
    triple t = p[i].dfs(-1);
    if(t.tav>mx){
        m.clear();
        mx=t.tav;
        o=0;
    }
    if(t.tav==mx){
        ++m[t.db];
        o+=t.db;
    }
}
long long er = 0;
auto it = m.rbegin();
auto it2 = m.rend();
--it2;
for(;it!=it2;++it){
    er+=it->first*(o-it->first-1)*it->second;
}
er+=it->first*(o-it->first-1)*(it->second-1);
er+=(it->first-1)*(o-it->first);
cout << er/2 << ' ' << it2->first*(long long)it2->second << '\n';
```

m a következő adatokat tartalmazza:
Egy részfa akkor jelenik meg benne, ha szerepel benne A vagy B levél. key az ilyen levelek darabszáma, $value$ az ilyen részfák darabszáma.

Az m-be kerülő key-k mennyisége maximum $\mathcal{O}\sqrt{N}$ lehet, mivel a $\sum(key * value) \leq N$, $1 \leq value$, $key \in \mathbb{Z}^+$. Így ennek a kódnak a futásideje futásideje $\mathcal{O}(N * log\sqrt{N})$.

Így összességében a program $\mathcal{O}(N * log\sqrt{N})$ futásidejű.

3. Megfelelőség

A program a megadott limiteken belül elméletben lefut.

A program random tesztesetekre helyes eredményeket adott (a kontrolt, tesztgenerátort lásd lentebb). A tesztelés a Polygon rendszerrel történt, a legrosszabb futásidő 109 ms volt. Alább tekinthetők meg a tesztelés adatai.

Listing 1. Kontrol (TLE vagy correct megoldás)

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

using namespace std;

class pont{
public:
    vector<int> hova;
    int index;
    bool banned = 0;
    int dfs(int szulo, int cel);
};

vector<pont> p;

int n, a, b;

int pont::dfs(int szulo, int cel){
    //cout << index << ' ' << szulo << ' ' << cel << '\n';
    if(banned){
        return -1;
    }
    if(cel == index){
        return 1;
    }
    int mx = -1;
    for(int i : hova){
        if(i!=szulo){
            int dd = p[i].dfs(index,cel);
            if(dd>-1){
                mx = max(mx, dd+1);
            }
        }
    }
    return mx;
}

int main(){
    cin >> n;
    p.resize(n);
    for(int i=0;i<n-1;i++){
        cin >> a >> b;
        p[a].hova.push_back(b);
        p[b].hova.push_back(a);
    }
    for(int i=0;i<n;i++){
        p[i].index=i;
    }
    map<int,int> err;
    for(int i=0;i<n;i++){
        if(p[i].hova.size()==1){
            p[i].banned=1;
            map<int,int> tmp;
            for(int j=0;j<n;j++){
                for(int k=0;k<n;k++){
                    ++tmp[p[j].dfs(-1,k)];
                }
            }
            auto it = tmp.end();
            --it;
            ++err[it->second];
            cout << i << ' ' << it->second << '\n';
            p[i].banned=0;
        }
```

```
    }
    cout << err.begin()->first/2 << ' ' << err.begin()->second << '\n';
return 0;
}
```

Listing 2. Tesztgenerátor

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

#include"testlib.h"

using namespace std;

vector<pair<int,int>> pp;

int main(int argc, char* argv[]) {
    registerGen(argc, argv, 1);
    int n = atoi(argv[1]);
    cout << n << '\n';
    for(int i=1;i<n;i++){
        int x;
        do{
            x=rnd.next(0, i-1);
        } while(x==i);
        pp.push_back({x,i});
        if(rnd.next(0,1)){
            swap(pp[i-1].first,pp[i-1].second);
        }
    }
    int db = rnd.next(n,1000*n);
    for(int i=0;i<db;i++){
        int a = rnd.next(0,n-2);
        int b;
        do{
            b=rnd.next(0,n-2);
        } while (a==b);
        //cout << a << ' ' << b;
        swap(pp[a],pp[b]);
    }
    for(int i=0;i<n-1;i++){
        cout << pp[i].first << ' ' << pp[i].second << '\n';
    }
}
```

Listing 3. Tesztek készítéséhez generátor script generáló python script

```
for i in range(100):
    print("S148 3",i,"> $")
    print("S148 6",i,"> $")
    print("S148 9",i,"> $")
    print("S148 10",i,"> $")
    print("S148 100",i,"> $")
    print("S148 1000",i,"> $")
    print("S148 10000",i,"> $")
    print("S148 100000",i,"> $")
```

Listing 4. (Hibás) validátor - A validátor lényege a tesztesetgenerálás ellenőrzése. Jelen esetben nem volt szükség egy korrekt ellenőrzésre.

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

#include"testlib.h"

using namespace std;
```

```
int main(int argc, char* argv[]) {
    registerValidation(argc, argv);
    int n = inf.readInt(3, 100000, "N");
    inf.readEoln();
    for (int i = 0; i < n-1; i++) {
        cout << i << '\n';
        int x = inf.readInt(0, n-1, "x_i");
        inf.readSpace();
        int y = inf.readInt(0, n-1, "y_i");
        assert(x!=y);
        inf.readEoln();
        cout << i << '\n';
    }

    inf.readEof();
}
```