Rekurzivni otoki

```
#include <iostream>
#include <vector>
#include <deque>
#include <unordered_set>
#include <algorithm>
using namespace std;

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int W, H;
    cin >> W >> H;
    vector<string> grid(H);
    for (int i = 0; i < H; i++){
        cin >> grid[i];
    }
    vector<vector<int>> comp(H, vector<int>(W, -1));
    int compCount = 0;
    vector<char> compType;
    vector<bool> compBorder;

    int dx4[4] = {1, -1, 0, 0};
    int dy4[4] = {0, 0, 1, -1};
    int dx8[8] = {-1,-1,-1,0,0,1,1,1};
    int dy8[8] = {-1,0,1,-1,1,-1,0,1};

    for (int i = 0; i < H; i++){
        for (int j = 0; j < W; j++){
            if(comp[i][j] == -1){
                char t = grid[i][j];
                compType.push_back(t);
                bool touchesBorder = false;
                deque<pair<int,int>> dq;
                dq.push_back({i,j});
                comp[i][j] = compCount;
                if(t == '#'){
                    while(!dq.empty()){
                        auto [x,y] = dq.front();
                        dq.pop_front();
                        if(x==0 || x==H-1 || y==0 || y==W-1)
                            touchesBorder = true;
                        for (int d = 0; d < 4; d++){
                            int nx = x + dx4[d];
                            int ny = y + dy4[d];
                            if(nx < 0 || ny < 0 || nx >= H || ny >= W)
                                continue;
                            if(comp[nx][ny] == -1 && grid[nx][ny] == t){
                                comp[nx][ny] = compCount;
                                dq.push_back({nx,ny});
                            }
```

```
                    }
                }
            } else { /
                while(!dq.empty()){
                    auto [x,y] = dq.front();
                    dq.pop_front();
                    if(x==0 || x==H-1 || y==0 || y==W-1)
                        touchesBorder = true;
                    for (int d = 0; d < 8; d++){
                        int nx = x + dx8[d];
                        int ny = y + dy8[d];
                        if(nx < 0 || ny < 0 || nx >= H || ny >= W)
                            continue;
                        if(comp[nx][ny] == -1 && grid[nx][ny] == t){
                            comp[nx][ny] = compCount;
                            dq.push_back({nx,ny});
                        }
                    }
                }
            }
            compBorder.push_back(touchesBorder);
            compCount++;
        }
    }
}
vector<unordered_set<int>> adj(compCount);
for (int i = 0; i < H; i++){
    for (int j = 0; j < W; j++){
        int cid = comp[i][j];
        if(grid[i][j] == '#'){
            for (int d = 0; d < 4; d++){
                int ni = i + dx4[d];
                int nj = j + dy4[d];
                if(ni < 0 || nj < 0 || ni >= H || nj >= W)
                    continue;
                int ncid = comp[ni][nj];
                if(ncid != cid){
                    adj[cid].insert(ncid);
                }
            }
        }
    }
}
for (int i = 0; i < H; i++){
    for (int j = 0; j < W; j++){
        if(grid[i][j] == '.'){
            int cid = comp[i][j];
            for (int d = 0; d < 8; d++){
                int ni = i + dx8[d];
                int nj = j + dy8[d];
                if(ni < 0 || nj < 0 || ni >= H || nj >= W)
                    continue;
                int ncid = comp[ni][nj];
                if(ncid != cid && grid[ni][nj]=='#'){
                    adj[cid].insert(ncid);
                }
            }
        }
    }
}
```

```
    vector<vector<int>> compAdj(compCount);
    for (int i = 0; i < compCount; i++){
        for (auto nb : adj[i]){
            compAdj[i].push_back(nb);
        }
    }
    const int INF = 1e9;
    vector<int> depth(compCount, INF);
    deque<int> dq;
    for (int i = 0; i < compCount; i++){
        if(compType[i]=='#' && compBorder[i]){
            depth[i] = 0;
            dq.push_back(i);
        }
    }

    while(!dq.empty()){
        int cur = dq.front();
        dq.pop_front();
        for(auto nb : compAdj[cur]){
            int cost = (compType[cur]=='.' ? 1 : 0);
            int newDepth = depth[cur] + cost;
            if(newDepth < depth[nb]){
                depth[nb] = newDepth;
                if(cost == 0)
                    dq.push_front(nb);
                else
                    dq.push_back(nb);
            }
        }
    }

    int ans = 0;
    for (int i = 0; i < compCount; i++){
        if(compType[i]=='#' && !compBorder[i]){
            ans = max(ans, depth[i]);
        }
    }

    cout << ans << "\n";
    return 0;
}
```

Urejanje z abakom

- Ali je v neki vrstici in stolpcu kroglica, ugotivimo z bitnimi operatorji: `seznam[vrstica] & (1 << stolpec)`
- V vsakem stolpcu najdlje drsi vrhnja kroglica. Na koncu je v vrstici $n - m_i$, kjer je m_i število kroglic v i-tem stolpcu. Čas drsenja je kar razlika med končnim in začetnim položajem kroglice.
- Skupni čas je maksimum časov drsenja vrhnjih kroglic v posameznih stolpcih.

Burek King

Rešitev, ko je $N \leq 10$

V tem podnalogi lahko simuliramo vseh 2^N možnih načinov gradnje burekdžinic in nato za vsako mesto poiščemo najbližjo restavracijo v $O(N)$. Skupna časovna zahtevnost je $O(2^N \cdot N^2)$.

Rešitev, ko v vsakem mestu živi enako število ljudi

Naj bo število prebivalcev vsakega mesta g. Opazujmo dve obstoječi zaporedni restavraciji. Naj bosta ti restavraciji v mestih i in j in naj med njima obstaja vsaj eno mesto. Opazimo, da lahko z gradnjo dveh dodatnih restavracij, ene v $i+1$ in druge v $j-1$, Admin prevzame stranke iz vseh mest v (cikličnem) intervalu $[i+1, j-1]$. Torej, če zgradimo dve restavraciji v (cikličnem) intervalu $[i+1, j-1]$, bomo prevzeli vse stranke iz tega intervala. Če zgradimo le eno, bomo prevzeli polovico strank, natančneje: $g \cdot \left(\frac{j-i-2}{2} + 1\right)$ strank.

Nalogo rešujemo z **požrešnim algoritmom** (*greedy algorithm*). V vsakem koraku izberemo interval, v katerem je optimalno zgraditi novo restavracijo. Opazimo, da če gre za prvo restavracijo na intervalu, z njeno postavitvijo pridobimo $g \cdot \left(\frac{j-i-2}{2} + 1\right)$ novih strank, medtem ko z drugo pridobimo $g \cdot \left(\frac{j-i-2}{2}\right)$ novih strank. Zato je dovolj, da uredimo vseh $2K$ dobičkov padajoče in izberemo najkrajši prefiks, katerega vsota je strogo večja od polovice skupnega števila prebivalcev. Ta rešitev deluje v časovni zahtevnosti $O(K \log K)$.

Rešitev, ko je $N \leq 3000$

Za to rešitev je dovolj, da modificiramo prejšnjo rešitev tako, da za vsak ciklični interval poiščemo optimalni položaj za prvo restavracijo, ki bo v njem zgrajena. To lahko dosežemo v časovni zahtevnosti $O(N^2)$, tako da preverimo, koliko strank nam prinese vsaka izbira.

Opazimo, da je **nujen pogoj**, da bi požrešni algoritem deloval, ta, da prva restavracija, ki je zgrajena na intervalu, prinese vsaj toliko strank, kolikor jih bo prinesla tudi druga. Opazimo tudi, da je ta pogoj tukaj izpolnjen, saj bo prva restavracija prevzela vsaj polovico strank iz intervala. Natančneje, opazite, da z gradnjo na poziciji $i+1$ prevzamemo celotno prvo polovico intervala, medtem ko z gradnjo na poziciji $j-1$ prevzamemo celotno drugo polovico intervala. Ena od teh dveh vrednosti je vsaj polovica vsote celotnega intervala.

Glavna rešitev

Za glavno rešitev je dovolj, da optimiziramo rešitev prejšnjega podnaloga. Z uporabo **predpone vsot** (*prefix sums*) je mogoče v $O(N)$ preveriti, koliko strank iz ustreznega intervala bo prevzela prva restavracija, ki bo v njem zgrajena.

Sušenje rjuh

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <limits.h>

using namespace std;
using ii = pair<int, int>;

template<typename F, typename S> ostream& operator<<(ostream& os, const pair<F, S>& par) {
    return os << par.first << "/" << par.second;
}

template<typename T> ostream& operator<<(ostream& os, const vector<T>& v) {
    bool prvic = true;
    os << "[";
    for (const T& e: v) {
        if (!prvic) os << ", ";
        prvic = false;
        os << e;
    }
    return os << "]";
}

int main() {
    long stVrvi, stRjuh, maxCezKoliko;
    cin >> stVrvi >> stRjuh >> maxCezKoliko;
    maxCezKoliko = min(maxCezKoliko, stVrvi);

    long imenovalec = 232'792'560;  // lcm(1, 2, ..., 20)

    // dp[rj]: optimalni "cas su"senja za <rj> rjuh
    vector<long> dp(stRjuh + 1);

    // "ce imamo manj kot (stVrvi / maxCezKoliko) rjuh, se vse posu"sijo v
    // "casu (1 / maxCezKoliko)
    long d = min(stVrvi / maxCezKoliko, stRjuh);
    fill(dp.begin() + 1, dp.begin() + d + 1, imenovalec / maxCezKoliko);

    // "ce imamo ve"c rjuh, moramo poiskati optimalno re"sitev
    for (long rj = stVrvi / maxCezKoliko + 1; rj <= stRjuh; rj++) {

        // Za k = 1, 2, ..., maxCezKoliko izra"cunaj "cas su"senja, "ce
        // najve"cje mo"zno "stevilo rjuh obesi"s "cez k vrni.

        long stevec = imenovalec * rj;
        for (int k = 1; k <= maxCezKoliko; k++) {
            // preostalih (rj - stVrvi / k) rjuh obesi optimalno
            long preostanek = (rj > stVrvi / k) ? (dp[rj - stVrvi / k]) : (0);
            stevec = min(stevec, imenovalec / k + preostanek);
        }
        dp[rj] = stevec;
    }

    long g = gcd(dp[stRjuh], imenovalec);
```

```
    cout << dp[stRjuh] / g << "/" << imenovalec / g << endl;

    return 0;
}
```

Vijačniki

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <queue>

using namespace std;
using pq = priority_queue<long, vector<long>, greater<long>>;

template<typename F, typename S> ostream& operator<<(ostream& os, const pair<F, S>& par) {
    return os << par.first << "/" << par.second;
}

template<typename T> ostream& operator<<(ostream& os, const vector<T>& v) {
    bool prvic = true;
    os << "[";
    for (const T& e: v) {
        if (!prvic) os << ", ";
        prvic = false;
        os << e;
    }
    return os << "]";
}

int main() {
    int stVijakov, stTipov, stPrimerkovVsakegaTipa, casVijacenja, casHlajenja;
    cin >> stVijakov >> stTipov >> stPrimerkovVsakegaTipa >> casVijacenja >> casHlajenja;

    // pripravljenost[i]: prioritetna vrsta, ki vsebuje "case, ko se ohladijo
    // vija"cniki tipa i
    vector<pq> pripravljenost(stTipov);
    for (int i = 0; i < stTipov; i++) {
        for (int j = 0; j < stPrimerkovVsakegaTipa; j++) {
            pripravljenost[i].push(0);
        }
    }

    // teko"ci "cas
    long cas = 0;

    for (int i = 0; i < stVijakov; i++) {
        int tip;
        cin >> tip;

        // ugotovimo, kdaj se bo ohladil prvi izmed vija"cnikov tipa <tip>
        long casOhladitveVijacnika = pripravljenost[tip].top();
        pripravljenost[tip].pop();

        // posodobimo teko"ci "cas
        cas = max(cas, casOhladitveVijacnika);
        cas += casVijacenja;

        // izbrani vija"cnik postavimo na hladno
        pripravljenost[tip].push(cas + casHlajenja);
    }
```

```
    cout << cas << endl;

    return 0;
}
```

Ljubljanske trole

```
#include<iostream>
#include<vector>
#include<queue>
#include<stack>
#include<algorithm>
#include<limits.h>
#include<math.h>
#include<map>
#include<set>
#include<unordered_map>
#include<unordered_set>
#include<iomanip>
#include<cstring>
#include<random>
#include<cassert>
#include<sstream>
typedef long long ll;
typedef unsigned long long ull;
typedef long double ld;
using namespace std;

vector<pair<int,int>>nodes[200000];


// dp[i] = (j, k) i==1 ce imas na voljo endpoint, j = najmanjsa vsota, k = koliko jih pride gor
 (1 ali 2)
vector<pair<ll,int>>dfs(int node,int par){
    vector<pair<ll,int>>res(2);
    if(nodes[node].size()==1){
        // is a leaf
        res[0].second=1;
        res[1].second=2;
        return res;
    }
    vector<vector<pair<ll,int>>>ch_res;
    for(auto[ch,w]:nodes[node]){
        if(ch==par)
            continue;
        ch_res.push_back(dfs(ch,node));
        for(int i=0;i<2;i++){
            ch_res.back()[i].first+=1LL*w*ch_res.back()[i].second;
        }
    }

    for(auto i:ch_res){
        res[0].first+=i[0].first;
        res[0].second+=i[0].second;
    }
    res[0].second=(res[0].second+1)%2+1;
    res[1].second=res[0].second%2+1;

    res[1].first=res[0].first;
    for(auto i:ch_res){
        ll nres=res[0].first-i[0].first+i[1].first;
        res[1].first=min(res[1].first,nres);
```

```
    }

    return res;
}

void solve(){
    int n;
    cin>>n;
    for(int i=0;i<n;i++){
        nodes[i].clear();
    }
    for(int i=1;i<n;i++){
        int a,b,w;
        cin>>a>>b>>w;
        a--;b--;
        nodes[a].emplace_back(b,w);
        nodes[b].emplace_back(a,w);
    }
    int root=-1;
    int nl=0;
    for(int i=0;i<n;i++){
        if(nodes[i].size()==1){
            nl++;
        }else{
            root=i;
        }
    }
    cout<<(nl+1)/2<<" "<<dfs(root,root)[nl%2].first<<"\n";
}

int main(){
    ios::sync_with_stdio(false);cout.tie(0);cin.tie(0);
    int t=1;cin>>t;
    while(t--)solve();
    return 0;
}

/*

 */
```