A-I-1 Najmenejkrát rozsvieť

Začať môžeme tým, že si zistíme, kam všade sa vieme dostať bez toho, aby sme kdekoľvek rozsvietili. Ak sa vieme dostať až do cieľa cesty, rovno tak spravíme a nie je čo ďalej riešiť.
Ak sa do cieľa nevieme dostať, niekde musíme rozsvietiť. Nech r je najväčšie číslo riadku, do ktorého sa dá dostať. Tvrdíme, že nič nepokazíme, keď rozsvietime práve riadok r.
Prečo je to tak? Rozsvietiť v ľubovoľnom skoršom riadku nám môže umožniť dostať sa do *niektorých* nových miestností v riadku r, zatiaľ čo rozsvietenie v riadku r nám umožní dostať sa do *úplne všetkých miestností v tomto riadku, vrátane tých, v ktorých sa teraz nesvieti.* A teda ak sme sa v prvej možnosti vedeli dostať niekam v riadkoch s číslom $\geq r$, vieme sa tam dostať aj v druhej možnosti.
Navyše v okamihu, keď rozsvietime riadok r, je jasné, že sa nikdy nebudeme musieť vrátiť do skorších riadkov – a tým skôr nebudeme potrebovať v žiadnom z nich rozsvietiť. Môžeme teda následne predstierať, že tieto riadky neexistujú. Tým ale dostaneme v podstate tú istú situáciu ako na začiatku, a teda na jej riešenie môžeme znova zopakovať tú istú úvahu.

Úlohu teda vieme vyriešiť tak, že zistíme, kam všade sa vie Matúš dostať, rozsvietime posledný dosiahnuteľný riadok, zistíme, ktoré nové miestnosti sú teraz dosiahnuteľné, zase rozsvietime posledný dosiahnuteľný riadok, a tak ďalej, až kým buď nepríde Matúš do cieľa alebo nezistíme, že riešenie neexistuje.
Na zisťovanie, kam sa Matúš vie dostať, použijeme prehľadávanie grafu – napr. prehľadávanie do šírky alebo do hĺbky. (Nestačí len ísť riadok po riadku, keďže Matúšovi sa občas oplatí aj vrátiť do riadku s menším číslom. Pozrite si príklad vstupu znázornený na nasledujúcom obrázku.)

```
S....
O.OOO
OOO.O
....C
```

Príklad vstupu, v ktorom sa Matúšovi oplatí vrátiť.
(S je štart, C je cieľ, O sú ostatné rozsvietené miestnosti.)

Pri dobrej implementácii bude mať toto riešenie časovú zložitosť $O(n^2)$, čo stačí na vyriešenie všetkých vstupov. Na dosiahnutie tejto časovej zložitosti si treba uvedomiť, že tým, že niekde rozsvietime, nemôžeme spôsobiť, že nejaké miestnosti *prestanú* byť dosiahnuteľné zo štartu. Netreba teda po každom rozsvietení spúšťať prehľadávanie úplne od začiatku, stačí len pokračovať zo stavu, v ktorom sme pred ním prestali. (Alebo explicitne spúšťať prehľadávanie vždy len na riadky od práve rozsvieteného ďalej.)

Listing programu (C++)

```
#include <bits/stdc++.h>
using namespace std;

const int DR[] = {-1,1,0,0}, DC[] = {0,0,-1,1};

int N, M;
vector< vector<bool> > rozsvietene, dosiahnutelne;
queue< pair<int,int> > spracovat;

int prehladaj() {
    // pomocou prehladavania najde vsetky nove dosiahnutelne miestnosti
    // vrati cislo najvacsieho dosiahnutelneho riadku
    int odpoved = -1;
    while (!spracovat.empty()) {
        int cr, cc;
        tie(cr,cc) = spracovat.front();
        spracovat.pop();
        odpoved = max( odpoved, cr );
        for (int d=0; d<4; ++d) {
            int nr = cr+DR[d], nc = cc+DC[d];
            if (!(0 <= nr && nr < N && 0 <= nc && nc < N)) continue;
            if (!rozsvietene[nr][nc]) continue;
            if (dosiahnutelne[nr][nc]) continue;
            dosiahnutelne[nr][nc] = true;
            spracovat.push( {nr,nc} );
        }
```

```
    }
    return odpoved;
}

int main() {
    cin >> N >> M;
    rozsvietene.resize(N, vector<bool>(N, false));
    dosiahnutelne.resize(N, vector<bool>(N, false));
    for (int m=0; m<M; ++m) {
        int r, s;
        cin >> r >> s;
        rozsvietene[r-1][s-1] = true;
    }

    dosiahnutelne[0][0] = true;
    spracovat.push( {0,0} );

    int posledny_rozsvieteny = -1;
    int pocet_rozsvieteni = 0;

    while (true) {
        // zisti, kam vsade sa teraz vie Matus dostat
        int rozsviet = prehladaj();
        // ak vie prist do ciela, vrat odpoved
        if (dosiahnutelne[N-1][N-1]) {
            cout << pocet_rozsvieteni << endl;
            return 0;
        }
        // ak sa nevieme dostat nikam dalej, vrat odpoved
        if (rozsviet == posledny_rozsvieteny) {
            cout << -1 << endl;
            return 0;
        }
        // rozsviet posledny dosiahnutelny riadok
        ++pocet_rozsvieteni;
        posledny_rozsvieteny = rozsviet;
        for (int c=0; c<N; ++c) {
            if (!rozsvietene[rozsviet][c]) {
                rozsvietene[rozsviet][c] = true;
            }
            if (!dosiahnutelne[rozsviet][c]) {
                dosiahnutelne[rozsviet][c] = true;
                spracovat.push( {rozsviet,c} );
            }
        }
    }
}
```

Poznámka na záver: Na zisk plného počtu bodov stačilo vyššie popísané riešenie, prípadne iné v podobnej časovej zložitosti. Existujú aj o trochu pomalšie riešenia, napr. s časovou zložitosťou $O(n^2 \log n)$, ktoré tiež mohli stihnúť vyriešiť všetky vstupy v časovom limite.
Úlohu však vieme riešiť aj v ešte lepšej časovej zložitosti. Na to si treba predspracovať vstup, aby sme vedeli, ktoré rozsvietené miestnosti spolu susedia. Potom pri prehľadávaní si len o týchto miestnostiach budeme pamätať, ktoré už sú dosiahnuteľné. Pri dobrej implementácii vie mať takéto riešenie časovú zložitosť $O(m + n)$.

A-I-2 Vizualizácia firmy

Na začiatok sa môžeme zamyslieť nad tým, ako získať čiastočné body za hierarchie špeciálneho tvaru.

Ak má celá firma hierarchiu „hviezda“ (všetci sú priami podriadení šéfky Marty), tak zjavne máme na výber úplne všetkých $n!$ poradí. Totiž nech zamestnancov rozmiestnime ľubovoľne, následne budeme kresliť vždy to isté: $n-1$ úsečiek vedúcich z jedného vrcholu (toho, kam sme dali Martu). Tieto sa zjavne nikde nebudú pretínať.

Zaujímavejší je druhý špeciálny prípad, ten nám už napovie niečo o tom, ako to bude vyzerať vo všeobecnosti. V tomto prípade máme firmu s hierarchiou „reťaz“: každý (až na toho úplne na spodku) má práve jedného priameho podriadeného.
Poďme zamestnancov umiestňovať do vrcholov zhora dole, teda začínajúc Martou. Pre ňu máme samozrejme n možností: je jedno, ktorý vrchol jej vyberieme. Už pre nasledujúceho zamestnanca však budeme mať možnosti len dve: musíme ho dať hneď naľavo alebo hneď napravo od Marty. Prečo? No lebo keby sme ho dali kamkoľvek inam, úsečka medzi ním a Martou rozdelí mnohouholník na dve nezávislé časti a v každej budú ešte nejaké nepoužité vrcholy. Počas umiestňovania ostatných zamestnancov potom nutne príde chvíľa, kedy budeme musieť prejsť z

jednej časti do druhej, no a to nevieme spraviť – dotyčná úsečka bude určite križovať tú medzi Martou a jej priamym podriadeným.
No a rovnaká úvaha bude teraz platiť aj pre každého ďalšieho zamestnanca. Vždy bude platiť, že tí doteraz umiestnení pokrývajú súvislý kus obvodu mnohouholníka, a následne bude vždy platiť, že pre toho nasledujúceho prichádzajú do úvahy len dve možnosti: buď ho dáme hneď naľavo alebo hneď napravo od doteraz zaplneného úseku. (Obe tieto možnosti fungujú, úsečka medzi práve umiestneným zamestnancom a jeho šéfom zjavne nepretne žiadnu zo skôr nakreslených.)
Takýchto rozhodnutí spravíme postupne presne $n-2$: pre každého ďalšieho zamestnanca okrem posledného. Posledný zamestnanec už bez rozhodovania ide na posledný voľný vrchol. Dokopy teda existuje $n2^{n-2}$ vhodných nakreslení.

Vzorové riešenie

Opäť môžeme začať tým, že niekam umiestnime šéfku Martu, teda koreň stromu. Máme n rovnocenných možností, ktorý vrchol n-uholníka jej priradiť.
Nech k označuje počet jej priamych podriadených. Každému z nich zodpovedá jedno „oddelenie“ firmy – teda jeden podstrom v hierarchii firmy. Kľúčové pozorovanie je, že každému z týchto podstromov musí zodpovedať súvislý úsek obvodu mnohouholníka. Myšlienka je podobná ako vyššie: ak premiešame zamestnancov z rôznych oddelení, niekde sa nám niečo bude križovať.
Dôkaz sporom: Rozoberieme dva prípady.
Prvý je, že sa úseky obvodu mnohouholníka zodpovedajúce nejakým oddeleniam prekrývajú len čiastočne. V takomto prípade vieme nájsť zamestnancov a_1, b_1, a_2, b_2, ktorí ležia na obvode mnohouholníka v tomto poradí, pričom a_1 a a_2 sú z jedného oddelenia (so šéfom a) a b_1 a b_2 z iného oddelenia (so šéfom b). Tým, že nakreslíme celé oddelenie a, určite prepojíme a_1 a a_2 nejakou lomenou čiarou. No a to isté dostaneme pre b_1 a b_2 keď nakreslíme oddelenie b. No a obe tieto lomené čiary ležia celé v mnohouholníku a preto sa určite niekde musia križovať – čo je spor.
Druhý prípad je, že celé jedno oddelenie je „okolo druhého“ – niekde na obvode mnohouholníka je oddelenie b, pričom naľavo aj napravo od neho máme zamestnancov a_1 a a_2, ktorí sú obaja z toho istého oddelenia a. Opäť dostaneme podobný spor: lomená čiara spájajúca a_1 a a_2 totiž niekde musí križovať úsečku spájajúcu šéfku celej firmy so zamestnancom b, ktorý šéfuje tomu „vnútornému“ oddeleniu.

Vieme teda, že ak začneme pri šéfke a pôjdeme postupne v smere hodinových ručičiek okolo mnohouholníka, postupne stretneme k úsekov, z ktorých každý bude zodpovedať jednému z oddelení firmy.
Ako teraz spočítať možnosti ich nakreslenia?
Zjavne existuje $k!$ možností pre poradie, v ktorom týchto k úsekov leží. Rôzne poradia zjavne vedú k navzájom rôznym nakresleniam.
Akonáhle si zvolíme konkrétne poradie, vieme pre každé oddelenie jednoznačne určiť, ktorý úsek vrcholov mnohouholníka mu bude zodpovedať – jeho dĺžka je určená počtom zamestnancov v oddelení. V tejto chvíli teda dostávame k menších problémov: pre každé oddelenie zvlášť si potrebujeme zvoliť jednu z možností, ako ho nakresliť na jeho úsek vrcholov.
Tieto menšie problémy sú zjavne navzájom nezávislé: to, ako nakreslím jedno oddelenie, nemá žiaden vplyv na to, ako môžem alebo nemôžem kresliť ostatné. Obrázky pre jednotlivé oddelenia môžeme teda spolu ľubovoľne kombinovať. Celkový počet možností teda dostaneme jednoducho tak, že vynásobíme počty možných nakreslení pre jednotlivé oddelenia.

Ostáva nám teda otázka, koľkými spôsobmi vieme nakresliť jedno konkrétne oddelenie firmy na kus obvodu mnohouholníka (tvorený presne toľkými vrcholmi, koľko má toto oddelenie dokopy zamestnancov).
Toto už našťastie bude úloha, ktorá bude veľmi podobná tej, ktorú sme práve vyriešili. Pozrime sa na to podrobnejšie. Nech a je zamestnanec, ktorý je šéfom celého oddelenia, ktoré ideme kresliť, a nech s je jeho priamy nadriadený. (Na začiatku našej úvahy je s Marta a a jej konkrétny priamy podriadený.)
Zamestnanca a musíme niekam umiestniť. Po tom, ako ho umiestnime, nakreslíme úsečku as.
Nech má zamestnanec a práve ℓ priamych podriadených. Opäť, každému z nich prislúcha nejaké pod-oddelenie firmy (tvorené ním a všetkými jeho priamymi aj nepriamymi podriadenými).

Podobne ako vyššie, aj teraz tvrdíme, že každému z týchto pod-oddelení musí prislúchať súvislý úsek obvodu. Dôkaz je rovnaký ako vyššie, len navyše ešte zmienime, že žiadne pod-oddelenie nemôže križovať úsečku sa, a teda musí ležať buď celé naľavo alebo celé napravo od nej.
Keď teraz teda chceme nakresliť celé oddelenie, ktorého vedúcim je a, tak chceme na príslušný úsek obvodu umiestniť $\ell + 1$ objektov: ℓ obrázkov jednotlivých pod-oddelení a niekde medzi nimi ešte aj samotného zamestnanca a.
No a už sme vyhrali, lebo spočítať tieto možnosti vieme zopakovaním takmer presne tej istej úvahy, ktorú sme už raz spravili. Začneme tým, že si zvolíme jedno konkrétne spomedzi $(\ell + 1)!$ možných poradí, v ktorom tieto objekty ležia. No a potom pre každé pod-oddelenie zvlášť zistíme počet spôsobov, ako ho nakresliť na súvislý úsek obvodu, a tieto počty medzi sebou vynásobíme.

Časová zložitosť tohto riešenia je $O(n)$, čiže lineárna od počtu zamestnancov. Totiž počas počítania celkového počtu nakreslení práve raz spracujeme každého zamestnanca x – vtedy, keď si kladieme otázku, koľkými spôsobmi vieme na obvod nakresliť pod-oddelenie, ktorému x šéfuje. Spracovať konkrétneho zamestnanca vieme v čase priamo úmernom tomu, koľko má priamych podriadených (nerátajúc čas strávený rekurzívnymi volaniami). No a keďže dokopy existuje len $n - 1$ dvojíc [priamy nadriadený, jeho/jej priamy podriadený], spracovanie celej firmy nám dokopy zaberie len lineárne veľa času.
Hodnoty funkcie faktoriál si môžeme ale nemusíme predrátať. Vo vrchole, v ktorom potrebujeme poznať hodnotu $x!$, aj tak spravíme $O(x)$ iných krokov výpočtu, a teda si môžeme dovoliť príslušnú hodnotu faktoriálu priamo vypočítať.

Listing programu (C++)

```cpp
#include <bits/stdc++.h>
using namespace std;

const long long MOD = 1000000007;

int N;
vector<int> parent;
vector< vector<int> > children;
vector<long long> factorial;

long long solve(int v) {
    long long answer;
    if (v == 0) {
        // specialny pripad: N moznosti kam dat sefku,
        // krat k faktorial moznosti pre poradie postromov
        answer = (N * factorial[ children[v].size() ]) % MOD;
    } else {
        // vseobecny pripad: (l+1) faktorial moznosti pre poradie
        answer = factorial[ 1 + children[v].size() ];
    }
    // pre kazdy podstrom zvlast spocitaj pocet nakresleni
    // a prenasob nim odpoved
    for (int c : children[v]) answer = (answer * solve(c)) % MOD;
    return answer;
}

int main() {
    cin >> N;

    factorial.resize(N+1, 1);
    for (int n=1; n<=N; ++n) factorial[n] = (factorial[n-1] * n) % MOD;

    parent.resize(N, -1);
    children.resize(N);
    for (int n=1; n<N; ++n) {
        cin >> parent[n];
        --parent[n];
        children[ parent[n] ].push_back(n);
    }

    cout << solve(0) << "\n";
}
```

Stručnejšie riešenie

Vyššie popísané riešenie rekurzívne prejde celý strom, v každom jeho vrchole ale vlastne spraví to isté: prenásobí celkovú odpoveď faktoriálom jeho stupňa. Presnejšie, keď sa na hierarchiu firmy dívame ako na *neorientovaný*

strom, tak celkový počet jeho nakreslení je zjavne rovný n-násobku súčinu faktoriálov jeho stupňov vrcholov. To samozrejme vieme vypočítať aj bez akejkoľvek rekurzie:

Listing programu (C++)

```
#include <bits/stdc++.h>
using namespace std;

const long long MOD = 1000000007;

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

    vector<long long> factorial(N+1, 1);
    for (int n=1; n<=N; ++n) factorial[n] = (factorial[n-1] * n) % MOD;

    vector<int> stupne(N+1, 0);
    for (int n=2; n<=N; ++n) { int p; cin >> p; ++stupne[n]; ++stupne[p]; }

    long long odpoved = N;
    for (int n=1; n<=N; ++n) odpoved = (odpoved * factorial[stupne[n]]) % MOD;
    cout << odpoved << endl;
}
```

A-I-3 Nevhodný darček

Výslednú postupnosť pre číslo x budeme pre stručnosť označovať $\mathrm{kod}(x)$. Spojenie po sebe idúcich postupností budeme označovať jednoducho symbolom $+$. Pripomeňme si, že platí $\mathrm{kod}(1) = (1)$ a pre všetky $x > 0$ platí $\mathrm{kod}(2x) = \mathrm{kod}(x) + (0) + \mathrm{kod}(x)$ a $\mathrm{kod}(2x+1) = \mathrm{kod}(x) + (1) + \mathrm{kod}(x)$.

Skúmanie vlastností kódov

Ľahko si môžeme všimnúť, že postupnosť $\mathrm{kod}(y)$ má súčet y, a teda práve y jednotiek.
Intuitívne: Postup zo zadania si môžeme predstaviť tak, že na začiatku máme jedno vrecko s c guličkami a postupne ich prerozdeľujeme do menších vreciek, pričom žiadne guličky nemiznú ani nepribúdajú.
Formálny dôkaz vyššie uvedeného tvrdenia sa ľahko dá spraviť matematickou indukciou.

Druhé užitočné pozorovanie bude zistiť, akú má postupnosť $\mathrm{kod}(x)$ dĺžku. Začnime tým, že si vypíšeme niekoľko prvých postupností:

$$
\begin{aligned}
kod(1) &= (1)\\
kod(2) &= (1,0,1)\\
kod(3) &= (1,1,1)\\
kod(4) &= (1,0,1,0,1,0,1)\\
kod(5) &= (1,0,1,1,1,0,1)\\
kod(6) &= (1,1,1,0,1,1,1)\\
kod(7) &= (1,1,1,1,1,1,1)\\
kod(8) &= (1,0,1,0,1,0,1,0,1,0,1,0,1,0,1)\\
kod(9) &= (1,0,1,0,1,0,1,1,1,0,1,0,1,0,1)
\end{aligned}
$$

Vidíme, že dĺžka postupnosti sa zvyšuje skokovo: napr. $\mathrm{kod}(8)$ aj $\mathrm{kod}(9)$ obe obsahujú dve kópie $\mathrm{kod}(4)$, a teda sú zhruba dvakrát také dlhé ako $\mathrm{kod}(4)$.
Môžeme si tiež všimnúť, že skoky v dĺžke nastávajú práve pri mocninách dvoch. Z toho už vieme sformulovať všeobecné pravidlo.

Rozdeľme si vyššie uvedený zoznam na bloky podľa dĺžky výslednej postupnosti: v bloku 0 bude číslo 1, v bloku 1 budú čísla 2-3, v bloku 2 čísla 4-7, a tak ďalej. Vo všeobecnosti platí, že blok b tvoria čísla z polo-otvoreného intervalu $[2^b, 2^{b+1})$. Každé číslo v bloku b má kód dĺžky $2^{b+1} - 1$.
Aj toto tvrdenie by sme vedeli formálne dokázať matematickou indukciou, najlepšie podľa premennej b.

Posledná vlastnosť kódov, ktorú si všimneme, je, že sú to palindrómy: každý kód vyzerá rovnako odpredu aj odzadu. Aj túto vlastnosť sa dá ľahko dokázať matematickou indukciou. Napr. ak už vieme, že $\text{kod}(x)$ je palindróm, tak aj $\text{kod}(x) + (0) + \text{kod}(x)$ zjavne je palindróm, lebo aj odzadu prečítame $\text{kod}(x) + (0) + \text{kod}(x)$.

Riešenie súťažnej úlohy

Teraz už sme pripravení vyriešiť súťažnú úlohu. Začneme tým, že si vypočítame číslo b bloku, v ktorom naše zadané c leží, a tým aj dĺžku $n = 2^{b+1} - 1$ celej postupnosti $\text{kod}(c)$.
Teraz sa pozrime na otázku, ktorý sme dostali na vstupe. Úsek, ktorý nás zaujíma, môže byť jedného z dvoch typov: buď obsahuje hodnotu v strede kódu alebo nie.

```
kód pre c =      | ... kód pre c/2 ... | 0/1 | ... kód pre c/2 ... |

                    | otázka typu 1 |
                                | ... otázka typu 2 ... |
```

Pri otázke typu 1 zadaný úsek neobsahuje strednú hodnotu, a teda leží celý v jednej polovici – čiže v jednej kópii kódu $\lfloor c/2 \rfloor$. Pre tieto otázky vieme teda pôvodnú otázku previesť na podobnú otázku s rovnako dlhým úsekom a postupnosťou zhruba polovičnej dĺžky.
Pri otázke typu 2 zadaný úsek obsahuje strednú hodnotu. Navyše k tej potrebujeme zistiť celkový počet jednotiek na konci (v sufixe) jedného kódu a na začiatku (v prefixe) druhého kódu.
Riešenie si môžeme zjednodušiť pozorovaním, že keďže kódy sú rovnaké spredu a zozadu, otázka na počet jednotiek v sufixe nejakého kódu má rovnakú odpoveď ako otázka na počet jednotiek v rovnako dlhom prefixe. Otázka typu 2 teda vedie k dvom otázkam na prefix kódu. V oboch týchto otázkach máme kód zhruba polovičnej dĺžky od pôvodného.

Tu by sa mohlo zdať, že sme si až tak nepomohli. Ak by aj naďalej mohlo nastať, že každá otázka nám vyrobí dve nové, tak by počet otázok veľmi rýchlo (exponenciálne) rástol a výsledné riešenie by bolo pomalé.
My však máme šťastie: vyššie popísaný prípad bol jediný, v ktorom sme z jednej otázky dostali dve. Tieto dve nové otázky sú totiž obe výrazne jednoduchšie ako pôvodná všeobecná otázka.
Pozrime sa na to, ako môže vyzerať otázka na počet jednotiek v nejakom prefixe kódu. Opäť sú dve možnosti: buď je krátka, teda neobsahuje hodnotu v strede, alebo je dlhá a stred obsahuje.

```
kód pre c =      | ... kód pre c/2 ... | 0/1 | ... kód pre c/2 ... |
otázky na prefix:
                 | krátka |
                 | ........... dlhá ........... |
```

Krátka otázka vedie k rovnakej otázke pre kód zhruba polovičnej dĺžky. Dlhá otázka obsahuje jeden celý kód (o tom presne vieme, koľko celý obsahuje jednotiek: $\lfloor c/2 \rfloor$) a potom nejaký prefix druhej kópie kódu.
Otázkú na prefix nejakého kódu teda *vždy* vieme previesť na *(nanajvýš) jednu novú otázku* na prefix kódu zhruba polovičnej dĺžky.

Vo všeobecnosti teda celé riešenie bude vyzerať nasledovne: Dostaneme otázku na všeobecný úsek postupnosti $\text{kod}(c)$. Tú prevedieme na nanajvýš dve otázky na prefix postupnosti $\text{kod}(\lfloor c/2 \rfloor)$, každú z nich zodpovieme samostatne.
Otázku na prefix postupnosti $\text{kod}(x)$ vieme vždy buď priamo zodpovedať, alebo ju previesť na jednu otázku na prefix postupnosti $\text{kod}(\lfloor x/2 \rfloor)$.

Keďže pri každom rekurzívnom volaní zmenšíme aktuálne číslo na polovicu, toto riešenie postupne spraví $O(\log c)$ takýchto volaní. V každom volaní funkcie spravíme len konštantný počet aritmetických operácií. Celý program dokopy teda spraví $O(\log c)$ aritmetických operácií.

Listing programu (Python)

```
# dlzka kodu cisla c
def dlzka(c):
    n = 1
    while n < c: n = 2*n+1
    return n

# odpoved pre prefix kod(c) dlzky r
# pomocna premenna n obsahuje dlzku kod(c)
def prefix(c, n, r):
    # prazdny prefix ma vzdy sucet nula
    if r == 0: return 0
    # pozrieme, ci je cela otazka v prvej kopii kodu
    stred = n // 2
    if r <= stred: return prefix(c//2, n//2, r)
    # ak nie, zoberieme celu prvu kopiu a prefix druhej
    return c//2 + c%2 + prefix(c//2, n//2, r-stred-1)

# odpoved na vseobecnu otazku: sucet kod(c)[l:r]
def vseobecna_otazka(c, n, l, r):
    # pozrieme, ci je cela otazka v prvej kopii kodu
    stred = n // 2
    if r <= stred: return vseobecna_otazka(c//2, n//2, l, r)
    # pozrieme, ci je cela v druhej kopii kodu
    if l > stred: return vseobecna_otazka(c//2, n//2, l-stred-1, r-stred-1)
    # ak sme sa dostali sem, otazka obsahuje stred
    # prevedieme ju na dve otazky o prefixoch
    return (c%2) + prefix(c//2, n//2, stred-l) + prefix(c//2, n//2, r-stred-1)

c, l, r = [ int(_) for _ in input().split() ]
print( vseobecna_otazka( c, dlzka(c), l-1, r ) )
```

Detailnejšie o časovej zložitosti

Tento program síce dokopy spraví $O(\log c)$ aritmetických operácií, ale keďže pracujeme s veľkými číslami, nie je úplne OK prehlásiť, že jeho časová zložitosť je $O(\log c)$.
Čísla, s ktorými pracujeme, majú hodnoty nanajvýš rádovo rovné c. Číslo c reprezentované v sústave so základom b má približne $\log_b c$ cifier, preto platí, že všetky čísla, s ktorými pracujeme, majú $O(\log c)$ cifier.
V našom programe s veľkými číslami robíme len veľmi jednoduché operácie: násobíme a delíme ich malými konštantami a porovnávame ich medzi sebou. Každú takúto operáciu vieme ľahko implementovať tak, aby bežala v čase lineárnom od počtu cifier príslušných čísel. Dokopy si teda každá aritmetická operácia vyžiada $O(\log c)$ krokov výpočtu, a teda celková časová zložitosť nášho programu bude $O((\log c)^2)$.

Stručnejšia implementácia

Vzorové riešenie vieme implementovať aj o čosi stručnejšie ako vyššie. Stačí sa inšpirovať technikou, ktorú používame napr. pri implementovaní intervalovej operácie v intervalovom strome. Pri tejto technike nemusíme samostatne riešiť prefixové otázky.

Listing programu (Python)

```
# dlzka kodu cisla c
def dlzka(c):
    n = 1
    while n < c: n = 2*n+1
    return n

def vseobecna_otazka(c, start, end, qstart, qend):
    # mozno je cely nas usek kodu sucastou otazky
    if qstart <= start and end <= qend: return c
    # mozno je cely nas usek kodu mimo otazky
    if end <= qstart or qend <= start: return 0
    # ak nenastal ani jeden pripad, rozbijeme kod na tri casti
    # stred spracujeme rovno, dva mensie kody rekurzivne
    pol = (end - start) // 2
    odpoved = 0
```

```
    odpoved += vseobecna_otazka(c//2, start, start+pol, qstart, qend)
    if qstart <= start+pol and start+pol < qend: odpoved += c%2
    odpoved += vseobecna_otazka(c//2, end-pol, end, qstart, qend)
    return odpoved

# uvodne volanie:
# vseobecna_otazka(c, 0, dlzka(c), l-1, r)
```

A-I-4 Zoznámte sa s Hviezdnym impériom

Postupne uvedieme riešenia jednotlivých podúloh. Začiatok riešenia podúlohy C sa stručne odkazuje na riešenie podúlohy A, inak sú tieto riešenia navzájom nezávislé.

Podúloha A: test kružnice

Každý hviezdny systém vykoná nasledovný algoritmus: Ak máme iný počet susedov ako 2, môžeme rovno odpovedať `NIE`. V opačnom prípade počkáme, aby sme videli, či niekto iný odpovedal `NIE`, a následne odpovieme `ANO`.
Prečo to funguje? Ak má každý systém práve dvoch susedov, znamená to, že každý systém leží na nejakej kružnici. A navyše si spomeňme, že jedna zo záruk, ktoré máme, je, že celé Hviezdne impérium je súvislé. Situácia, v ktorej by sme mali viacero navzájom disjuktných kružníc, teda nemôže nastať.
Toto riešenie má hodnotu k rovnú nule: nepotrebujeme si nič dať veštiť.

```
# ak mame nespravny stupen, urcite to nie je kruznica
if stupen != 2: return NIE

# pockame a ak vesmir nie je ruzovy, nik nemal odpoved NIE, a teda vsetci vratime odpoved ANO
if ruzovy_vesmir(): return NIE
return ANO
```

Podúloha B: existencia párovania

Začneme tým, že si každý systém dá vyveštiť $k = \lceil \log_2 n \rceil$ bitov: ich lokálny index toho suseda, s ktorým majú byť vo dvojici.
Teraz potrebujeme vyveštené údaje skontrolovať. Každý systém teda pošle hodnotu 1 tomu susedovi, ktorý mu bol vyveštený, a hodnotu 0 každému inému susedovi.
Po odoslaní správ rok počkáme na odpovede. Čo očakávame naspäť? Náš partner nám má tiež poslať hodnotu 1, potvrdzujúcu, že aj oni si myslia, že sú v dvojici s nami. No a každý iný z našich susedov nám má poslať 0, teda oznámiť, že nás za svoju dvojicu nepovažujú.
(Ak sme si vyveštenú hodnotu uložili do premennej `par` tak sme najskôr zapísali 1 do `outbox[par]` a potom sme skontrolovali, či sme dostali 1 práve do `inbox[par]` a nikam inam.)
Ak perfektné párovanie existuje, veštci si vedia vybrať jedno konkrétne (teda konkrétnu sadu $n/2$ dvojíc susedných systémov) a každému systému vyveštiť index toho správneho suseda. Ak perfektné párovanie neexistuje, veštci musia vyveštiť nejakú sadu indexov, ktorá nefunguje. V takom prípade musí existovať nejaký systém a, ktorý pošle svoju 1 systému b takému, že b nepošle svoju 1 späť. No a toto práve systém a po roku odhalí, keď v príslušnom inboxe nájde nulu.

```
# ak sme jediny system v celom imperiu, odpoved je nie
if N == 1: return NIE

# vyvestime si index suseda, ktory ma byt nasou dvojickou
par = vyvesti_cislo(stupen)
if par >= stupen: return NIE

# vyrobime spravy vsetkym susedom
for i in range(stupen):
    outbox[i] = 0
outbox[par] = 1

# posleme spravy, pockame rok a potom skontrolujeme, ci prisli rovnake odpovede
for i in range(stupen):
    if outbox[i] != inbox[i]:
        return NIE

# pockame a ak vesmir nie je ruzovy, nik nemal odpoved NIE, a teda vsetci vratime odpoved ANO
```

```
if ruzovy_vesmir(): return NIE
return ANO
```

Podúloha C: existencia Hamiltonovskej kružnice

Tu už sa nemôžeme spoľahnúť na podobný trik ako v podúlohe A. Totiž keďže medzi systémami môžeme mať aj ďalšie hrany okrem našej kružnice, môžu súvislosť Hviezdneho impéria (čiastočne alebo úplne) zabezpečiť tieto hrany. V nejakom okamihu si preto budeme musieť dať pozor, aby nás veštci neoklamali tým, že nám namiesto jednej veľkej kružnice vyveštia niekoľko menších, navzájom disjunktných kružníc.
Ukážeme si postupne dve riešenia.

Pri prvom riešení sa budeme inšpirovať príkladom zo študijného textu. Každý systém si dá vyveštiť dve čísla: jeho poradové číslo na Hamiltonovskej kružnici (od 0 po $n-1$) a tiež lokálny index toho suseda, ktorý leží na tejto Hamiltonovskej kružnici bezprostredne pred ním. (Ak bolo systému vyveštené poradové číslo 0, vyveštený index má ukazovať na suseda, ktorý má na kružnici číslo $n-1$.)
Každý systém potom pošle správnemu susedovi správu, v ktorej mu oznámi, aké číslo si myslí, že tento sused má mať. (Ostatným susedom pošle „prázdnu" správu – napr. s číslom -1.)
Každý systém potom počká na správy od susedov a skontroluje, že dostal práve jednu neprázdnu správu a v tej napísané to správne číslo: to, ktoré mu bolo vyveštené. Ak ktokoľvek odhalí nejakú nezrovnalosť, odpovie `NIE`. Ak všetci všetko úspešne skontrolujú, všetci odpovedia `ANO`.

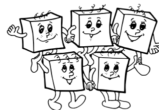
Zdôvodnenie správnosti: Je zjavné, že ak nejaká Hamiltonovská kružnica existuje, veštci si vedia nejakú vybrať a podľa nej vyveštiť všetko potrebné. Potrebujeme teda hlavne ukázať, že ak takáto kružnica neexistuje, tak bez ohľadu na obsah veštieb to aspoň jeden zo systémov vždy odhalí.
Ak niekto dostane neplatné číslo alebo neplatný index suseda, odhalí to triviálne: uvidí hodnotu mimo správneho rozsahu. Ak dva systémy dostanú toho istého suseda x ako svojho predchodcu na kružnici, odhalí to x tak, že dostane dve neprázdne správy. Jediná možnosť, ako prejsť cez všetky tieto kontroly, je teda tá, že každý systém dostane vyvešteného iného predchodcu (t.j. vyveštení predchodcovia tvoria permutáciu systémov). Ako sme už opísali vyššie, v tejto chvíli by sme ešte mohli mať namiesto jednej veľkej kružnice niekoľko menších. Na túto záverečnú kontrolu teraz poslúži druhý údaj, ktorý sme si dali vyveštiť: poradové čísla na kružnici.
Ak nik nedostal vyveštené poradové číslo 0, každý systém pošle svojmu predchodcovi číslo menšie ako svoje. Systém s najmenším poradovým číslom teda svojmu predchodcovi pošle číslo, ktoré je určite menšie ako to správne – a teda príjemca tejto správy odhalí, že vyveštené informácie nesedia.
Ak nejaký systém x dostal vyveštené poradové číslo 0, jeho vyveštený predchodca musel dostať číslo $n-1$ (inak by oznámil chybu), toho predchodca musel dostať číslo $n-2$, a tak ďalej. Ak by čokoľvek z tohoto bolo vyveštené ináč, dotyčný systém by odhalil chybu, keďže by nesedelo číslo, ktoré dostane v správe.
Ak by neboli všetky systémy na tejto konkrétnej kružnici, dostali by sme sa priskoro k tomu systému y, ktorý má za predchodcu systém x (ten s číslom 0, od ktorého sme začínali). Systém y musel dostať vyveštené číslo väčšie ako 1, inak by mu prišla zlá správa od jeho predchodcu. Potom systém y pošle systému x o jedno menšie číslo od svojho. Toto číslo bude ešte stále kladné, a tým sa to celé aj tak pokazí a systém x odhalí chybu (keďže očakáva, že dostane číslo 0).
Máme teda korektné riešenie, ktoré potrebuje vyveštiť $k = 2\lceil \log_2 n \rceil$ bitov.

```
# vyvestime si nase poradove cislo a index suseda, ktory ma byt na kruznici pred nami
poradie = vyvesti_cislo(N)
pred = vyvesti_cislo(stupen)
if poradie >= N or par >= stupen: return NIE

# vyrobime spravy vsetkym susedom
for i in range(stupen):
    outbox[i] = -1
outbox[pred] = (poradie-1) % N

# posleme spravy, pockame rok a pozrieme sa na spravy, co prisli nam
prislo = [ x for x in inbox if x != -1 ]
if len(prislo) != -1: return NIE # ma nam prist prave jedna neprazdna sprava
if poradie != prislo[0]: return NIE # v tej sprave nam malo prist nase cislo

# pockame a ak vesmir nie je ruzovy, nik nemal odpoved NIE, a teda vsetci vratime odpoved ANO
if ruzovy_vesmir(): return NIE
return ANO
```



Lepšie riešenie

Nešlo by ešte jedno z týchto dvoch čísel ušetriť? Ale áno, šlo. V skutočnosti totiž funguje aj ešte priamočiarejšie riešenie: stačí, keď si každý systém dá vyveštiť svoje poradové číslo a to potom rozošle úplne všetkým svojim susedom.

Ako bude potom vyzerať kontrola? Každý systém skontroluje, či medzi číslami od susedov dostal aj nejaké o jedna menšie, aj nejaké o jedna väčšie ako svoje. (Samozrejme, počítajúc modulo n, teda systém s číslom 0 chce vidieť čísla 1 a $n-1$, zatiaľ čo systém $n-1$ chce vidieť čísla $n-2$ a 0.) Ak niekto nedostane obe požadované čísla, odpovie NIE. Ak nik neodpovedal NIE, všetci odpovedia ANO.

Podobne ako vyššie potrebujeme zdôvodniť, že ak všetci dajú odpoveď ANO, tak naozaj existuje Hamiltonovská kružnica. Zdôvodnenie bude vyzerať podobne ako vyššie. Niekto musel dostať vyveštené číslo 0, inak by platilo, že systém s najmenším číslom (ktoré je kladné) sa nedočká od žiadneho suseda menšieho čísla od svojho, a teda odpovie NIE. Niektorý zo susedov vrcholu, ktorý dostal číslo 0, musel dostať číslo 1. Niektorý z jeho susedov musel dostať číslo 2. A tak môžeme pokračovať ďalej až po číslo $n-1$. No a tým už je vynútené, že každé číslo dostal práve jeden systém a aj to, že susedné čísla susedia. A keďže aj systém s vyvešteným číslom 0 musí dostať aj správu s číslom $n-1$ (a naopak), musí naozaj ísť o kružnicu a nie len o cestu.

Toto riešenie si vystačí s $k = \lceil \log_2 n \rceil$ vyveštenými bitmi.

TRIDSIATY ÔSMY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: NIVAM – Národný inštitút vzdelávania a mládeže, Bratislava 2022