B-I-1 Kamenná cesta

Ak si zoberieme všetky dni, v ktorých vieme prejsť na druhý breh rieky, zadanie od nás chce, aby sme našli posledný (najväčší) deň, v ktorom je to možné. Ako prvé je preto dobré si rozmysliet, ako veľké takéto číslo môže byť, čo nám ohraničí výsledok.
Existuje pre každý vstup aspoň nejaké riešenie? Kedže pre deň, v ktorom je i-ty kameň ešte použiteľný, k_i platí, že $k_i \geq 1$, tak ľahko vidieť, že v prvý deň vieme určite použiť všetky kamene a dostať sa na druhú stranu. Existuje aj horné ohraničenie? Inými slovami, existuje deň, kedy už nevieme cez rieku v žiadnom prípade prejsť? Takéto ohraničenie by neexistovalo iba keby nepotrebujeme žiadny kameň na prejdenie rieky, kedže po určitom čase bude každý zaplavený. Kedže vieme preskočiť iba jeden kameň a pre počet kameňov n platí nerovnosť $n \geq 2$ tak máme zaručené, že musíme použiť aspoň jeden kameň na prejdenie rieky. Vieme teda, že rieku určite neprejdeme v deň, kedy už bude zatopený aj posledný kameň. Môžeme si uvedomiť, že tento deň je o jedna väčší ako maximum z hodnôt $k_1, k_2, \dots k_n$.
Zistili sme, že správna odpoveď sa nachádza niekde medzi hodnotami 1 a $\max(k_1, k_2 \dots k_n) + 1$. To je super, pretože môžeme postupne vyskúšať všetky tieto hodnoty a nájsť najväčšiu vyhovujúcu.

Riešenie hrubou silou

Povedzme, že máme nejaký konkrétny deň r. Chceme zistiť, či vieme prekročiť rieku iba pomocou doteraz nezatopených kameňov. Teda platí, že kameň j môžeme použiť iba ak $k_j \geq r$. Môžeme si vytvoriť pomocné pole o veľkosti n a doňho si zaznačiť, na ktoré políčka sa vieme dostať z ľavého brehu. Pri vypĺňaní i-teho políčka sa nám stačí pozrieť, či je možné sa dostať buď na políčko $i-1$ alebo $i-2$. Ak áno a platí, že $k_i \geq r$, tak sa vieme dostať v r-tý deň aj na r-té políčko a do pomocného poľa zaznačíme `True`, inak `False`. Na konci vieme z hodnôt na indexoch $n-1$ a n v našom pomocnom poli zistiť, či v daný deň vieme rieku prekročiť. Rieku prekročíme iba ak sa na jedno z týchto políčok vieme dostať.

Listing programu (C++)

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

using namespace std;

bool over(vector<int>& kam, int den)
{
    int n = kam.size();
    // na zaciatku oznacime vsetko ako nedosiahnutelne
    vector<bool> pom(n, false);

    if(kam[0] <= den) pom[0] = true;
    if(kam[1] <= den) pom[1] = true;

    for(int i=2; i<n; ++i) {
        if(kam[i] >= den && (pom[i-1] || pom[i-2])) {
            pom[i] = true;
        }
    }
    if (pom[n-1] == true || pom[n-2] == true) {
        return true;
    }
    else {
        return false;
    }
}

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

    vector<int> kamene(n);
    for(int i=0; i<n; ++i) {
        cin >> kamene[i];
    }

    int maxim = kamene[0];
    for(int i=1; i<n; ++i) {
        maxim = max(maxim, kamene[i]);
    }

    for(int i=maxim; i>=1; --i) {
```

```
        if(over(kamene, i)) {
            cout << i << endl;
            break;
        }
    }
    return 0;
}
```

V pythone si ukážeme mierne inú implementáciu a to bez pomocného poľa. Je založená na pozorovaní, že z nášho pomocného poľa nám vždy stačí poznať iba posledné dve vypočítané hodnoty (pozície $i-2$ a $i-1$).

Listing programu (Python)

```
def over(kam, den):
    pred, pred2 = True, True

    for i in range(len(kam)):
        if (pred == False and pred2 == False):
            break

        pred2 = pred
        pred = False

        if (kam[i] >= den):
            pred = True

    return (pred or pred2)


n = int(input())
kamene = [int(_) for _ in input().split()]

maxim = max(kamene)

for i in range(maxim, 0, -1):
    if over(kamene, i):
        print(i)
        break
```

Toto riešenie je správne, ale má časovú zložitosť závislú od maximálneho k_i, ktoré môže byť oveľa väčšie ako n. Je teda pomalé, ale stačí na riešenie prvej a tretej sady.

Zlepšenie riešenia hrubou silou

Riešenie hrubou silou ide v skutočnosti vylepšiť ďalším pozorovaním. Samotné overovanie nejakého konkrétneho riešenia r nezrýchlime. To čo sme robili doteraz je, že sme overili všetky hodnoty medzi 1 až $\max(k_0, k_1, \dots k_n)$ a vybrali najväčšie riešenie. Myšlienkou zrýchlenia je, že pokiaľ ide prejsť rieku v i-ty deň tak je to možné aj v každý skorší deň. Prečo? Pokiaľ sme v i-ty deň použili nejaké kamene, pomocou ktorých sme vedeli prejsť na druhú stranu rieky. Vieme presne tie isté kamene použiť aj v hociktorý skorší deň. To platí aj opačne, ak nevieme v i-ty deň prejsť rieku, nepôjde to ani neskôr. To preto, že neskôr bude zatopených už len viac kameňov. Toto nám umožnuje použiť techniku známu ako binárne vyhľadanie na nájdenie nášho riešenia.
Pointa binárneho vyhľadávania je, že namiesto overenia hodnôt 1 až $\max(k_0, k_1, \dots k_n)$ nám postačuje overiť riešenie pre číslo, ktoré je v strede, medzi 1 a $\max(k_0, k_1, \dots k_n)$. Označme si ho s. Pokiaľ platí, že v s-tý deň sa dá prejsť cez rieku tak vieme, že naše riešenie bude väčšie alebo rovné ako s. Pokiaľ v s-tý deň nevieme rieku prejsť, riešenie bude menšie ako s. V oboch prípadoch sa nám počet čísel, ktoré môžu byť riešením zmenšil približne o polovicu. Tento postup vieme ďalej opakovať na novo nájdené ohraničenia našeho riešenia až kým nám neostane jedno číslo, ktoré je výsledkom.

Listing programu (C++)

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

using namespace std;

// na overenie mozeme pouzit funkciu z predchadzajuceho riesenia
bool over(vector<int>& kam, int den);

int main()
{
    int n; cin >> n;
```

```
    vector<int> kamene(n);
    for(int i=0; i<n; ++i) {
        cin >> kamene[i];
    }

    int maxim = kamene[0];
    for(int i=1; i<n; ++i) {
        maxim = max(maxim, kamene[i]);
    }

    int l = 0, h = maxim+1;
    while (l+1 < h) {
        int stred = (l + h) / 2;

        if (over(kamene, stred)) {
            l = stred;
        }
        else {
            h = stred;
        }
    }

    cout << l << endl;

    return 0;
}
```

Iné riešenia

V predchádzajúcich riešeniach nám vadilo, že naše riešenia boli závislé od maximálneho k_i. Poďme skúsiť nájsť riešenie, ktoré je závislé len od n. Urobíme to tak, že sa pozrieme bližšie na samotnú hodnotu najlepšieho riešenia.
Vieme, že najlepšie riešenie leží medzi 1 a $\max(k_1, k_2, \dots k_n)$. Tvrdíme však, že toto riešenie musí byť rovná niektorému z k_i. Viete prečo? Povedzme si, že by nebolo. Potom ale určite existujú kamene, po ktorých vieme preskákať v tento deň a jeden z nich, označme si ho j, má najnižšiu hodnotu k_j. Naše riešenie nemôže byť najlepšie pretože od tohto riešenia existuje lepšie riešenie rovné k_j. To používa kamene z nášho predchádzajúceho riešenia a pritom je určite väčšie. Po tejto úvahe už vieme, že riešenie bude určite rovné niektorému z hodnôt $k_1, k_2, \dots k_n$. To znamená, že nám stačí overiť n hodnôt, každú v čase $O(n)$, čo nám prinesie riešenie s časovou zložitosťou $O(n^2)$.

Vzorové riešenie

Síce na vyriešenie úlohy na plný počet bodov stačilo aj riešenie z časti `Zlepšenie riešenia hrubou silou`, my si teraz ukážeme jedno ešte rýchlejšie a elegantnejšie riešenie. Jeho myšlienka je nasledujúca.
Kedy určite rieku nevieme prejsť? Predsa keď budú zatopené dva za sebou ležiace kamene! Takto veľkú medzeru totiž nevieme preskočiť a ostaneme zaseknutý. Dvojica kameňov i a $i+1$ bude zatopená keď sa zatopí ten neskorší, teda v čase $\max(k_i, k_{i+1}) + 1$. Zo všetkých dvojíc však chceme zobrať tú najskoršie zatopenú, prvá dvojica kameňov bude teda zatopená v čase $\min(\max(k_1, k_2) + 1, \max(k_2, k_3) + 1, \dots \max(k_{n-1}, k_n) + 1)$.
My si však musíme odpovedať aj na dôležitejšiu otázku a to, či sa dá prejsť na druhú stranu rieky vždy, keď ešte **neexistuje** dvojica za sebou idúcich zatopených kameňov. Ak totiž platí takéto tvrdenie, znamená to, že vyššie určená hodnota je správnou odpoveďou.
Predstavme si, že žiadne dva za sebou idúce kamene nie sú zatopené. Je jasné, že z ľavého brehu máme kam skočiť. V tom momente sa nachádzame na nezatopenom kameni. Ak je nasledujúci kameň nezatopený, jednoducho naň prejdeme a riešime problém ďalej. Ak však zatopený je, ten ďalší zatopený byť nemôže (lebo to by boli už dva za sebou idúce zatopené kamene). Preskočíme teda jeden zatopený kameň a pokračujeme rovnakou úvahou ďalej.
Práve sme dokázali, že v deň $\min(\max(k_1, k_2), \max(k_2, k_3), \dots \max(k_{n-1}, k_n)) + 1$ sa cez rieku určite nedostaneme, ale v ľubovoľný skorší deň to ešte pôjde, toto číslo mínus jeden je teda odpoveďou.

Listing programu (C++)

```
#include <iostream>
#include <vector>
#include <algorithm> // min(), max()

using namespace std;
```

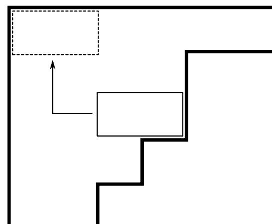
```
int main()
{
    int n; cin >> n;

    vector<int> kamene(n);
    for(int i=0; i<n; ++i) {
        cin >> kamene[i];
    }

    int odp = max(kamene[0], kamene[1]);
    for(int i=1; i<n-1; ++i) {
        odp = min(odp, max(kamene[i], kamene[i+1]));
    }

    cout << odp << endl;
    return 0;
}
```

Listing programu (Python)

```
n = int(input())
kamene = [int(_) for _ in input().split()]

odpoved = max(kamene[0], kamene[1])
for i in range(1, n-1):
    odpoved = min(odpoved, max(kamene[i], kamene[i+1]))
print(odpoved)
```

B-I-2 Baliaci papier

Predstavme si, že máme na stole Aničkin baliaci papier a chceme z neho niekde vystrihnúť obdĺžnik konkrétnych rozmerov. Ak sa to dá, určite sa to dá tak, aby tento obdĺžnik obsahoval ľavý horný roh Aničkinho papiera. Hocijaký obdĺžnik, ktorý leží na Aničkinom papieri, totiž môžeme posúvať doľava až kým nenarazíme na ľavý okraj a následne dohora až kým nenarazíme na horný.

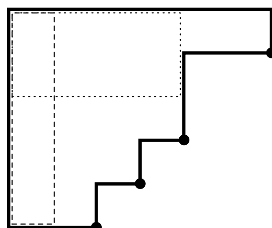


Ukážka posunutia obdĺžnika, ktorý chceme vystrihnúť, do ľavého horného rohu.

Ak chceme vystrihnúť obdĺžnik, ktorý obsahuje ľavý horný roh Aničkinho papiera, je celý obdĺžnik jednoznačne určený tým, ktoré políčko má v pravom dolnom rohu. No a už ľahko nahliadneme, že každé políčko takto určuje platný obdĺžnik a všetky tieto obdĺžniky majú navzájom rôzne rozmery.
Celkový počet obdĺžnikov, ktoré vie Anička vystrihnúť, je preto rovný celkovému počtu políčok na jej baliacom papieri – čiže ide jednoducho o súčet všetkých hodnôt p_i zo vstupu.

Maximálne obdĺžniky

Každý maximálny obdĺžnik zjavne musí obsahovať ľavý horný roh – lebo hocijakému inému obdĺžniku vieme smerom doľava zväčšiť šírku, resp. smerom dohora zväčšiť výšku.
Zostáva teda zistiť, kde môže mať maximálny obdĺžnik svoj pravý dolný roh.
Opäť, musí platiť nutná podmienka, že takýto obdĺžnik nevieme predĺžiť ani dodola, ani doprava. Jeho pravý dolný roh teda musí byť posledným políčkom vo svojom stĺpci, a navyše stĺpec napravo (ak existuje) nesmie byť rovnako dlhý ako tento.



Dva nie-maximálne obdĺžniky: jeden sa dá zväčšiť doprava, druhý dodola.
Zvýraznené body sú pravé dolné rohy maximálnych obdĺžnikov.

Je táto podmienka aj postačujúca? Inými slovami, sú všetky obdĺžniky, ktoré toto spĺňajú, naozaj maximálne? Áno, sú. Majme takýto obdĺžnik, ktorého pravý dolný roh je na poslednom políčku stĺpca i. Obdĺžniky, ktoré majú šírku aspoň i, nemôžu byť od tohto vyššie, lebo stĺpcov s výškou väčšou ako p_i je menej ako i. A obdĺžniky, ktoré majú výšku aspoň p_i, nemôžu byť od tohoto širšie, keďže stĺpcov s aspoň p_i políčkami je len i.
Všetky maximálne obdĺžniky vieme teda nájsť tak, že v cykle prechádzame postupnosťou p_i a o každej z nich vyhodnotíme, či je ostro väčšia ako nasledujúca z hodnôt. (Pritom predpokladáme, že $p_{s+1} = 0$.)
Podotkneme ešte, že počet maximálnych obdĺžnikov by sme tiež vedeli vyjadriť ako počet *rôznych* hodnôt v postupnosti p_i, čiže napr. v Pythone ako `len(set(p))`. Ale keďže sa tak či tak na všetky potrebujeme pozrieť, v nižšie uvedenom listingu ich sčitujeme ručne.

Listing programu (Python)

```
S = int( input() )
P = [ int(_) for _ in input().split() ]

print( sum(P) ) # pocet vsetkych obdlznikov

P.append(0)
maximalnych = 0
najvacsi_obsah = 0

for s in range(S):
    if P[s] > P[s+1]:
        maximalnych += 1
        najvacsi_obsah = max( najvacsi_obsah, (s+1) * P[s] )

print(maximalnych)
print(najvacsi_obsah)
```

B-I-3 Mravčia farma

Začneme riešením ľahšej podúlohy. V tej si stačilo všimnúť, že:

- Vždy, keď sa Ferdo posunie dodola, zdvojnásobí sa počet komôrok, kde môže byť. Totiž z každej, kde mohol byť, mal dve možnosti, kam ísť, a všetky vedú do navzájom rôznych komôrok.
- Vždy, keď sa Ferdo posunie dohora, zmenší sa počet možností na polovicu – keďže je to „undo" predchádzajúcej operácie.

  Jedinou výnimkou je situácia, kedy už presne vieme, kde sa Ferdo nachádza. Ak sa vtedy posunie vyššie, lezie práve na úroveň, na ktorej ešte nebol. Ale aj táto situácia je zjavná: naďalej nám ostala presne jedna možnosť pre Ferdovu polohu.

Stačilo teda zo vstupu načítať len reťazec pohybov a potom podľa jeho písmen násobiť a deliť dvoma.
Pozrime sa teraz na ťažšiu podúlohu.

Poznámka k bodovaniu

Odovzdané riešenia ťažšej verzie úlohy sme bodovali nezvykle tolerantne – presnejšie, za jeden typ chyby sme sa rozhodli nestŕhať body. Dôvod pre toto rozhodnutie je jednoduchý: rovnakú chybu vo svojom pôvodnom riešení spravil aj autor úlohy :)

Podotýkame, že takéto bodovanie nikde nespravilo principiálny rozdiel – totiž riešenia ťažšej verzie úlohy, ktoré majú len tento typ chyby, korektne riešia ľahšiu verziu úlohy.
Nižšie si ukážeme jedno možné **nesprávne** riešenie, vysvetlíme si, v ktorej jeho myšlienke je skrytá záludná chyba a potom si ukážeme komplikovanejšie, ale predsa len korektné riešenie.

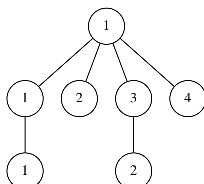
Nesprávne riešenie – skúste sami zistiť, prečo

Tvrdenie ♠: Ak sa kdekoľvek vo Ferdovom „programe" nachádza postupnosť príkazov DH, teda „dole, hore", môžeme ju zjavne vynechať, lebo po nej vie Ferdo zjavne byť presne tam, kde pred ňou, a nikde inde.

Toto pozorovanie nám vie pomôcť – hovorí, že všetky výskyty DH môžeme z Ferdovho programu vymazať.
Pozor, toto treba robiť trochu šikovne. Môže vás pokúšať spraviť jednoducho niečo ako „kým program obsahuje nejaké DH, zmaž všetky výskyty DH", no v najhoršom prípade má takýto postup časovú zložitosť až kvadratickú od dĺžky programu. (Dotyčným najhorším prípadom je program začínajúci veľa D a pokračujúci veľa H. V každej iterácii raz prejdeme celý program, ale nájdeme a zmažeme len jediné DH.)
Šikovnejší spôsob mazania je taký, že program čítame znak po znaku a pamätáme si, čo sme ešte nezmazali. Nezmazaná časť bude vždy mať tvar „najskôr niekoľko H, potom niekoľko D", takže na jej zapamätanie už nepotrebujeme ani reťazec, stačia nám dve celočíselné premenné.
Po vyššie popísanej úprave dostávame nový Ferdov program. Ten vo všeobecnosti najskôr spraví niekoľko krokov hore a potom niekoľko krokov dole. Krok hore vieme odsimulovať v konštantnom čase (keďže vtedy je ešte jednoznačné, kde sa Ferdo nachádza). No a pri krokoch dodola si už môžeme dovoliť pokojne aj vyskúšanie všetkých možností, keďže každú komôrku aj každú chodbičku počas tejto cesty dole navštívime nanajvýš raz.

Kde bola chyba?

No čo, všimli ste si chybu? Nám to vcelku dlho trvalo. Chyba sa skrýva hneď na začiatku, v tvrdení ♠. Problém je v tom, že toto tvrdenie vôbec nemá pravdu. Predstavme si takéto mravenisko:



Povedzme, že sme začali na úplnom vrchu. Keď sa pohneme dodola, môžeme byť hocikde na strednej úrovni. Teraz sa pohneme dole a zase hore… aha. Problém je v tom, že z komôrok 2 a 4 sa dodola ísť nedá. Ak teda začneme na vrchu tohto mraveniska a pohneme sa DDH, musíme skončiť v strednej vrstve, a to buď v komôrke 1 alebo v komôrke 3. Všimnite si explicitne, že v komôrke 2, hoci leží medzi nimi, skončiť nevieme.

Vzorové riešenie

Túto patáliu treba nejakým spôsobom vyriešiť. Ako prvé riešenie nám môže napadnúť udržiavať si zoznam všetkých komôrok, v ktorých sa Ferdo môže nachádzať. Pri každom posune jednoducho prejdeme všetkými týmito komôrkami a pozrieme sa, kam sa z nich Ferdo mohol v danom smere posunúť. Občas sa stane, že sa posunúť nevie, tak ako v prípade uvedenom vyššie, a tieto možnosti jednoducho zahodíme.
Takéto riešenie je síce správne, ale pomalé. Pri každom pohybe sa totiž musíme pozrieť na viacero, potenciálne až n, rôznych komôrok a nejakým spôsobom ich spracovať. To vedie k časovej zložitosť $O(nq)$.
Aj zo zlého riešenia si však vieme odniesť niekoľko užitočných pozorovaní. Napríklad, že ak sa na každú úroveň pozrieme najviac raz, môžeme pokojne spracovať aj všetky komôrky, ktorej na nej ležia. Dokopy sa totiž pozrieme nanajvýš na všetkých n komôrok, čo je dostatočne rýchle.
Ferdovu cestu by sme teda chceli spracovávať postupne po úrovniach. Veľmi ľahko vieme vypočítať, na ktorej úrovni sa nachádzal v jednotlivých krokoch svojej cesty. Vieme totiž kde začínal, každý pohyb dohora `H` ho posunie o úroveň vyššie, pohyb dodola `D` o úroveň nižšie. Vďaka tomu vieme napríklad zistiť aj to, na ktorej najvyššej úrovni sa počas svojho pohybu nachádzal. Navyše si uvedomme, že na najvyššej úrovni je práve jeden vhodný vrchol, v ktorom mohol stáť. Toto pozorovanie sme si všimli už v riešení jednoduchšej podúlohy.

Okrem začiatku máme teda ďalšie jednoznačné miesto, kde sa Ferdo mohol nachádzať, a navyše je najvyššie ako Ferdo v mravenisku bol. Je teda dobrý nápad začať od neho. Zamyslime sa, čo vieme o tejto komôrke a ďalších Ferdových pohyboch povedať. Vieme, že keď bol v tejto komôrke, jediný pohyb, ktorý mohol spraviť je `D`, vyššie totiž nešiel. Presunul sa teda do komôrok na úrovni o jedna nižšej, ktoré susedia s touto komôrkou.
Sú však všetky tieto komôrky pre Ferda vyhovujúce? Ferdo sa bude z nich predsa hýbať ďalej a ako sme videli v protipríklade vyššie, občas to nie je možné. Našťastie, je pomerne ľahké zistiť, kedy to nepôjde. Vieme totiž, ako najnižšie počas tohto pohybu klesol. A ak sa z tejto komôrky nedá klesnúť dostatočne hlboko, tak do nej predsa nemohol vojsť.
Toto sa pekne ilustruje na vyššie uvedenom obrázku a postupnosti `DDH`. Najvyššie bol na úrovni 0 v komôrke 1. Pohybom dole sa posunul na úroveň 1, nie všetky komôrky sú však vyhovujúce. Ferdo totiž vie, že najnižšie ako sa počas svojej cesty dostane je úroveň 2 (pri ďalšom pohybe dodola). Z vrchola 2 a 4 na úrovni 1 však nevedie žiadna cesta na úroveň 2, a preto nie sú vyhovujúce.
Začína to vyzerať sľubne, ale niekoľko detailov nám predsa len uniká. Predstavme si, že na obrázku vyššie, začínajúc úplne na vrchu, išiel postupnosťou `DDHHD`. V tomto prípade mohol skončiť v ľubovoľnej komôrke na úrovni 1. V prvej časti síce musel ísť buď do komôrky 1 alebo 3 aby mohol spraviť druhý krok dodola, potom sa však aj tak vrátil na samý vrch a to ho dostalo opäť do začiatočnej pozície.
Predchádzajúca úvaha bude teda platiť iba v tom prípade, keď si povieme, že Ferdo sa nachádza v najvyššej komôrke jeho cesty **poslednýkrát**. A najnižší bod jeho cesty nepočítame z celej postupnosti, ale iba z tej časti, ktorá nasledovala po tomto poslednom navštívení najvyššej komôrky. Teraz však už naozaj vieme, do ktorých komôrok nižšej úrovne sa mohol posunúť, a do ktorých nie.
Zvyšok riešenia je už v podstate rovnaký. Aj pre tieto komôrky totiž platí rovnaká úvaha. Keďže dohora už nemôže ísť (krok predtým tam bol posledný krát) a ak ešte nie je na úrovni kde má skončiť, bude sa musieť pohnúť dodola. Najskôr teda preskočíme na ten moment jeho cesty, keď bude na tejto úrovni posledný krát. Následne sa pre každú komôrku, v ktorej mohol byť na tejto úrovni, pozrieme na susedné komôrky o úroveň nižšie a zistíme, či je pod nimi dostatok priestoru na zvyšok Ferdovej cesty, teda či sa v nich dá dostatočne klesnúť. Ak áno, budeme s nimi pracovať v ďalšom kroku.
Takýmto spôsobom postupne prejdeme každou úrovňou a spracujeme vrcholy na nej, zastavíme sa keď budeme na úrovni, v ktorej Ferdo skončil. Všetky vrcholy, ktoré nám ostali sú výsledkom. Časová zložitosť takéhoto riešenie je $O(n+q)$.
Bohužiaľ, aj implementácia takéhoto riešenie je o niečo zložitejšia ako sme zamýšľali, nie však natoľko aby ste ju nezvládli, akurát si v nej treba predpočítať všetky dôležité informácie. Napríklad pre každú komôrku v mravenisku potrebujeme vedieť, ako najnižšie sa z nej dá dostať, ak pôjdeme iba dodola. Taktiež potrebujeme pre každú úroveň vedieť, kedy naposledy sa na nej Ferdo vyskytne a pre každý bod Ferdovej cesty potrebujeme vedieť ako hlboko pôjde v zvyšku cesty. Tieto výpočty však nie sú také zložité a môžete si ich bližšie prezrieť v priloženej implementácii.

Listing programu (Python)

```
n = int(input())

strom = []
for _ in range(n):
    strom.append([int(_)-1 for _ in input().split()][1:])

# pre kazdu komorku si spocitame ako najhlbsie sa z nej vieme dostat cestou dodola
hlbka = []
for i in range(n):
    hlbka.append([i for _ in strom[i]])

for i in range(n-1, 0, -1):
    for j in range(len(strom[i])):
        otec = strom[i][j]
        hlbka[i-1][otec] = max(hlbka[i-1][otec], hlbka[i][j])

zac_u, zac_k, q = map(int, input().split())

cesta = input()
urovne_cesty = [zac_u]
posledna_navsteva = [-1]*n

# spocitame uroven, na ktorej bol Ferdo v kazdom kroku cesty a pre kazdu uroven
# si zapamatame posledny krat, ked na nej bol
for i in range(q):
```

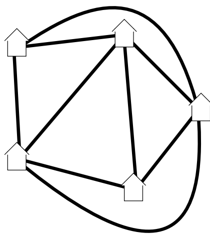
```
    if cesta[i] == 'D':
        urovne_cesty.append(urovne_cesty[-1] + 1)
    else:
        urovne_cesty.append(urovne_cesty[-1] - 1)
    posledna_navsteva[urovne_cesty[-1]] = i+1

# pre kazdu cast cesty potrebujeme vediet ako najhlbsie pojde v jej zvysku
najvyssia_uroven = min(urovne_cesty)
for i in range(q-1, -1, -1):
    urovne_cesty[i] = max(urovne_cesty[i], urovne_cesty[i+1])

while zac_u != najvyssia_uroven:
    zac_k = strom[zac_u][zac_k-1]
    zac_u -= 1

pripustne_komorky = [zac_k]
pozicia = posledna_navsteva[najvyssia_uroven]
# postupne ideme od najvyssej urovne az po tu, kde skoncil
while zac_u != urovne_cesty[-1]:
    nove_komorky = []
    p = 0
    # prechadzame komorky na o jedno nizsej urovni, zistujeme ci sa Ferdo vedel
    # dostat do komorky priamo nad nimi a ci su vyhovujuce
    for i in range(len(strom[zac_u + 1])):
        while p < len(pripustne_komorky) and pripustne_komorky[p] < strom[zac_u + 1][i]:
            p += 1
        if p < len(pripustne_komorky) and pripustne_komorky[p] == strom[zac_u + 1][i] and \
           hlbka[zac_u + 1][i] >= urovne_cesty[pozicia]:
            nove_komorky.append(i)
    zac_u += 1
    pozicia = posledna_navsteva[zac_u]
    pripustne_komorky = nove_komorky

print(len(pripustne_komorky))
```

B-I-4 Domy na lúke

Ide o problém z teórie grafov súvisiaci s kreslením grafu do roviny. Domy predstavujú vrcholy grafu a cesty jeho hrany. Graf sa snažíme do roviny nakresliť tak, aby sa jeho hrany nikde nekrižovali.

Podúloha A (2 body): pre päť domov nakresli deväť ciest

Jedno možné riešenie je na nasledujúcom obrázku:



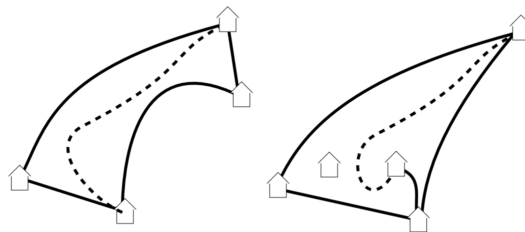
Podotkneme, že viac ako deväť ciest sa postaviť nedá. Dvojíc domov je síce desať, všetkých desať ciest však nevieme naraz nakresliť do roviny tak, aby sa žiadne dve nekrižovali. Odborne teda hovoríme, že päťvrcholový kompletný graf K_5 nie je rovinný. (Keď si vyriešime aj zvyšné podúlohy, budeme vedieť, prečo tomu tak je.)

Podúloha B (2 body): ako vyzerajú časti lúky pri maximálnom počte ciest

Keďže vieme, že na lúke pri Bardejove je dokopy postavených najviac ciest, ako sa len dalo, znamená to, že nijakú ďalšiu cestu už nikde nevieme postaviť. No a z toho pomerne zjavne vyplýva, že Zuzankin (aj každý iný) kus lúky musí obsahovať presne tri domy.

Presnejšie zdôvodnenie: Menej ako tri domy nikdy na žiadnom kuse lúky nie sú – to by vyžadovalo cestu, ktorá spája menej ako dva rôzne domy, alebo dve cesty spájajúce tú istú dvojicu domov, ale také cesty nemáme.

Ak by na nejakom kuse lúky (či už na obvode tohto kusu alebo vo vnútri) ležali viac ako tri domy, vedeli by sme ešte postaviť ďalšiu cestu. Stačí si vybrať dva domy, ktoré ešte nie sú cestou spojené, a spojiť ich vnútrom nášho kusu lúky. Ak sa teda už žiadna cesta nedá postaviť, každý kus lúky musí byť „trojuholník“ ohraničený tromi domami a tromi cestami medzi nimi.



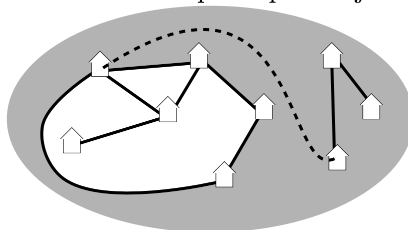
Dva príklady pridania novej (čiarkovanej) cesty na priveľkú lúku.
Vľavo spájame nesusedné domy na jej obvode, vpravo dom na obvode s nesusedným domom vo vnútri.

Podúloha C (2 body): pridávame cesty ku kružnici

Pozrime sa na ľubovoľný okamih, kedy na lúke pri Cíferi staviame jednu novú cestu. Táto cesta povedie vnútrom nejakej časti lúky a prepojí dva domy, ktoré boli na jej obvode a doteraz spolu nesusedili. Tým teda táto cesta rozdelí jednu časť lúky na dve – čiže vždy, keď pridáme novú cestu, zväčší sa o jednu aj počet častí lúky.
Nech m je počet ciest a f počet častí lúky. Na začiatku máme $m = n$ a $f = 2$. No a práve sme si odvodili, že hodnota $m - f$ sa už nikdy nezmení. Vždy teda bude platiť vzťah $m - f = n - 2$.

Podúloha D (2 body): budujeme cesty na zelenej lúke

Teraz potrebujeme spraviť ešte jedno pozorovanie: ak pridáme cestu, ktorá spojí dve rôzne skupiny domov, tak sa nezmení počet častí lúky – lebo na oboch stranách práve pridanej cesty je naďalej tá istá časť lúky.



Každým pridaním cesty teda *buď* zmenšíme počet skupín domov, *alebo* zväčšíme počet častí lúky. Ak si počet skupín domov označíme k, dostávame teda, že hodnota $m + k - f$ sa počas stavby ciest nikdy nemení.
Na začiatku, keď ešte nemáme postavené žiadne časti, máme $m + k - f = n - 1$, lebo máme 0 ciest, n skupín domov a jednu lúku. Tento vzťah teda musí platiť aj po postavení ľubovoľného počtu ľubovoľných ciest. Počet častí lúky teda vieme vo všeobecnosti vyjadriť vzťahom $f = m + k + 1 - n$.

Pre zaujímavosť, tento vzťah medzi počtom vrcholov, hrán a oblastí v rovinnom grafe sa nazýva Eulerov vzťah. Veľmi podobný vzťah (aký a prečo?) platí aj pre počet vrcholov, hrán a stien mnohostenu (v trojrozmernom priestore).

Vzťah, ktorý sme si odvodili v podúlohe C, platí pre všetky súvislé *rovinné grafy – teda pre ľubovoľnú lúku, na ktorej už všetky domy tvoria jednu skupinu, platí $f = m + 2 - n$. Dôkaz vyzerá v podstate rovnako ako naše riešenie podúlohy C. Následne vieme tento vzťah zovšeobecniť na nesúvislé rovinné grafy aj tak, že spravíme nasledovné pozorovanie: keď na lúku pridáme ďalšiu skupinu, ktorá sama o sebe mala n domov, m ciest a delila lúku na f oblastí, tak sa počet domov naozaj zvýši o n, počet ciest sa zvýši o m, ale počet častí lúky sa zvýši len o $f - 1$, lebo tá časť lúky, v ktorej sa nová skupina domov zjavila, je teraz zároveň „vonkajšou“ časťou lúky pre novo pridanú skupinu domov.*

Podúloha E (2 body): najväčší počet ciest a oblastí

Keď je postavený najväčší možný počet ciest, zjavne platia viaceré skutočnosti:

- Je len jedna skupina domov, teda $k = 1$.
- Každá cesta oddeľuje od seba dve rôzne časti lúky.

- (Podúloha B:) Každá časť lúky je „trojuholník“.

Ak máme f častí lúky a každá z nich má na obvode tri cesty, znamená to, že ciest musíme mať presne $m = 3f/2$. (Do súčtu $3f$ sme každú cestu zarátali dvakrát, lebo každá cesta od seba oddeľuje dve lúky.)
Keď toto dosadíme do vzorca z podúlohy D, dostávame $f = 3f/2 + 2 - n$, alebo ekvivalentne $f = 2n - 4$. A následne vieme dopočítať, že $m = 3(2n - 4)/2 = 3n - 6$.
Na veľkej lúke pri Egreši je teda medzi n domami postavených $3n - 6$ ciest a tie delia lúku na $2n - 4$ častí.

Inými slovami, rovinný graf s $n \geq 3$ vrcholmi môže mať nanajvýš $3n - 6$ hrán.

TRIDSIATY PIATY ROČNÍK OLYMPIÁDY V INFORMATIKE

Príprava úloh: Michal Anderle, Michal Forišek, Andrej Korman
Recenzia: Michal Forišek
Slovenská komisia Olympiády v informatike
Vydal: IUVENTA – Slovenský inštitút mládeže, Bratislava 2019