Priebeh krajského kola

Krajské kolo 34. ročníka Olympiády v informatike, kategória A, sa koná 22. januára 2019 v dopoludňajších hodinách. Na riešenie úloh majú súťažiaci **4 hodiny čistého času**. Rôzne úlohy riešia súťažiaci na samostatné listy papiera. Akékoľvek pomôcky okrem písacích potrieb (napr. knihy, výpisy programov, kalkulačky) sú zakázané.

Čo má obsahovať riešenie úlohy?

- Slovne popíšte algoritmus.
  Slovný popis riešenia musí byť jasný a zrozumiteľný i bez nahliadnutia do samotného algoritmu/programu.
- Zdôvodnite správnosť vášho algoritmu.
- Uveďte a zdôvodnite jeho časovú a pamäťovú zložitosť.
- Podrobne uveďte dôležité časti algoritmu, ideálne vo forme programu v nejakom bežnom programovacom jazyku (napr. C++, Python, Java, Pascal).
- V prípade, že používate vo svojom programovacom jazyku knižnice, ktoré obsahujú implementované dátové štruktúry a algoritmy (napr. STL pre C++), v popise algoritmu stručne vysvetlite, ako by ste napísali program s rovnakou časovou zložitosťou bez použitia knižnice.

Hodnotenie riešení

Za každú úlohu môžete získať od 0 do 10 bodov.

Pokiaľ nie je v zadaní povedané ináč, najdôležitejšie dve kritériá hodnotenia sú v prvom rade **správnosť** a v druhom rade **efektívnosť** navrhnutého algoritmu. Na výslednom počte bodov sa môže prejaviť aj kvalita popisu riešenia a zdôvodnenie tvrdení o jeho správnosti a efektívnosti.

Efektívnosť algoritmu posudzujeme vypočítaním jeho časovej zložitosti – funkcie, ktorá hovorí, ako dlho vykonanie algoritmu trvá v závislosti od veľkosti vstupných parametrov. Nezávisí pri tom na konštantných faktoroch, len na rádovej rýchlosti rastu tejto funkcie.

V zadaní úloh uvádzame časť „Hodnotenie“, v ktorej nájdete približné limity na veľkosť vstupných údajov. Pod pojmom „efektívne vyriešiť“ chápeme to, že váš program spustený na modernom počítači by mal dať odpoveď nanajvýš do niekoľkých sekúnd.

Údaje z tejto časti zadania by mali slúžiť hlavne na to, aby ste o riešení, ktoré vymyslíte, vedeli približne povedať, koľko bodov zaň dostanete.

A-II-1 Tulipány

Úlohu budeme riešiť pomerne priamočiarym dynamickým programovaním. Existujú síce aj pažravé riešenia, tie sú však väčšinou náročné na efektívnu implementáciu a vyžadujú rozbor prípadov.
Predstavme si, že ideme popri záhone zľava doprava a vždy, keď stretneme nejaké voľné políčko, rozhodneme sa nejako, či tam zasadiť tulipán alebo nie. Samozrejme, ukončiť skupinu tulipánov (a teda nechať políčko voľné) smieme len vtedy, ak táto skupina má nepárnu veľkosť.
Malo by byť zjavné, že po tom, ako sme práve spracovali prvých k políčok, sú len tri principiálne rôzne situácie, v ktorých sa môžeme nachádzať:

0. Doteraz spracovaný úsek končí skupinou tulipánov párnej dĺžky.
1. Doteraz spracovaný úsek končí skupinou tulipánov nepárnej dĺžky.
2. Doteraz spracovaný úsek končí voľným políčkom.

V našom riešení postupne pre každé k a každú situáciu zistíme, či ju vieme dosiahnuť, a ak áno, koľko najmenej tulipánov na to treba zasadiť. Riešením úlohy je potom najlacnejší spôsob ako spracovať všetkých n políčok a dosiahnuť stav 2 alebo 3.

Označme $b_{i,j}$ najmenší počet tulipánov potrebný na to, aby sme po spracovaní prvých i políčok boli v stave j. Ak sa to vôbec nedá dosiahnuť, položíme $b_{i,j} = \infty$.
Na začiatku zjavne stačí uvažovať $b_{0,2} = 0$ a $b_{0,0} = b_{0,1} = \infty$, teda akoby naľavo od záhonu bolo voľné políčko, ktoré sme nechali voľné.
Ak i-te políčko zľava obsahuje tulipán, vieme dosiahnuť len stavy 0 a 1. Najlepší spôsob, ako dosiahnuť stav 0 je rovný najlepšiemu spôsobu ako dosiahnuť o políčko skôr stav 1. No a najlepší spôsob ako dosiahnuť stav 1 (teda úsek nepárnej dĺžky) môže byť buď z predchádzajúceho stavu 0 (pokračujeme v úseku párnej dĺžky ďalším tulipánom) alebo z predchádzajúceho stavu 2 (tento tulipán začína nový úsek). Formálne, $b_{i,0} = b_{i-1,1}$, $b_{i,1} = \min(b_{i-1,0}, b_{i-1,2})$ a $b_{i,2} = \infty$.
Ak je i-te políčko zľava prázdne, vieme dosiahnuť stav 2 tak, že ho necháme prázdne. Toto ale môžeme spraviť len vtedy, ak sme po $i-1$ políčkach boli v stave 1 alebo 2 (teda mali sme buď úsek nepárnej dĺžky, ktorý teraz môžeme ukončiť, alebo nebol žiaden bežiaci úsek tulipánov). Ak sa na toto políčko rozhodneme zasadiť tulipán, dostaneme stav 0 alebo 1. Tu platí rovnaká úvaha ako v predchádzajúcom odseku, len ešte nesmieme zabudnúť na $+1$ za práve zasadený tulipán. Formálne, $b_{i,0} = 1+b_{i-1,1}$, $b_{i,1} = 1+\min(b_{i-1,0}, b_{i-1,2})$ a $b_{i,2} = \min(b_{i-1,1}, b_{i-1,2})$.

Listing programu (Python)

```
zahon = input()
infinity = float('inf')

B = [ [infinity,infinity,0] ]

for policko in zahon:
    stare = B[-1]
    nove = [infinity, infinity, infinity]
    if policko == 'T':
        nove[0] = stare[1]
        nove[1] = min( stare[0], stare[2] )
    else:
        nove[0] = 1 + stare[1]
        nove[1] = 1 + min( stare[0], stare[2] )
        nove[2] = min( stare[1], stare[2] )
    B.append( nove )

print( min( B[-1][1], B[-1][2] ) )
```

A-II-2 Ohrada

Hlavným pozorovaním potrebným pre efektívne riešenie tejto úlohy je skutočnosť, že pre $n = 5$ riešenie vždy existuje. Toto si dokážeme rozborom prípadov. Presnejšie, pozrieme sa na to, ako môže vyzerať konvexný obal našich piatich bodov.

- Ak všetkých päť bodov tvorí vrcholy konvexného päťuholníka, môžeme zobrať ľubovoľné štyri z nich.
- Ak na konvexnom obale ležia štyri z piatich bodov, označme ich po obvode A, B, C, D tak, aby piaty bod E ležal v trojuholníku určenom bodmi A, B a priesečníkom AC a BD. Potom môžeme zobrať body $AECD$ (alebo body $BCDE$).
- Ak na konvexnom obale ležia len tri z našich piatich bodov, označme tie na obvode ABC a tie vnútri DE. Zoberieme D, E a tie dva body spomedzi A, B, C ktoré ležia na rovnakej strane priamky DE.

Všeobecné riešenie teraz vyzerá nasledovne:

1. Pre $n = 4$ overíme, či body tvoria vrcholy konvexného štvoruholníka. Ak áno, zoberieme ho, ak nie, riešenie neexistuje.
2. Pre $n \geq 5$ v lineárnom čase vyberieme 5 bodov s najmenšou x-ovou súradnicou. Zvyšných $n - 5$ bodov zahodíme.
3. Pre 5 bodov, ktoré nám ostali, vyššie popísaným spôsobom nájdeme riešenie.

Všimnite si, že ak sme v kroku 3 našli riešenie pre päť bodov, ktoré nám zostali, toto je nutne riešením aj pre pôvodný vstup. Body, ktoré sme v kroku 2 zahodili, totiž zjavne nemôžu ležať v nami zostrojenom štvoruholníku.

Listing programu (C++)

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

void fail() { cout << "neda sa" << endl; exit(0); }

struct point { int x, y; };

// komparator pre body, primarne podla x-ovej suradnice
bool operator<(const point &A, const point &B) {
    if (A.x != B.x) return A.x < B.x;
    return A.y < B.y;
}

// vypis zoznamu bodov
void output(const vector<point> &P) {
    for (const point p : P) cout << p.x << " " << p.y << endl;
    exit(0);
}

// ked ideme z A cez B do C, zatacame dolava?
bool ccw(const point &A, const point &B, const point &C) {
    int ux = B.x - A.x, uy = B.y - A.y, vx = C.x - B.x, vy = C.y - B.y;
    return (ux*vy - vx*uy) > 0;
}

// lezi bod A v konvexnom stvoruholniku P?
bool is_in_quad(const point &A, const vector<point> &P) {
    int c = ccw(P[0], P[1], A);
    for (int i=1; i<4; ++i) if (c != ccw(P[i], P[(i+1)%4], A)) return false;
    return true;
}

// tvori stvorica bodov v tomto poradi konvexny stvoruholnik?
// (pozor, takyto test je korektny len pre troj- a stvoruholniky!)
bool is_quad(const vector<point> &P) {
    bool c = ccw( P[0], P[1], P[2] );
    for (int i=1; i<4; ++i) if (c != ccw( P[i], P[(i+1)%4], P[(i+2)%4] )) return false;
    return true;
}

// vyries ulohu pre styri body
void solve_four(vector<point> P) {
    do { if(is_quad(P)) output(P); } while (next_permutation(P.begin(),P.end()));
    fail();
}

// vyries ulohu pre pat bodov
void solve_five(vector<point> P) {
    do {
        point A = P[0];
        vector<point> Q( P.begin()+1, P.end() );
        if (is_quad(Q) && !is_in_quad(A,Q)) output(Q);
```

```
    } while (next_permutation(P.begin(),P.end()));
    assert(false); // <-- sem by sa nikdy nemal dostat
}

int main() {
    int N;
    cin >> N;
    vector<point> P;
    int x, y;
    for (int n=0; n<N; ++n) { cin >> x >> y; P.push_back( {x,y} ); }
    if (N == 4) {
        solve_four(P);
    } else {
        nth_element( P.begin(), P.begin()+5, P.end() );
        P.resize(5);
        solve_five(P);
    }
}
```

A-II-3 Úrady

Nech M je množina tých miest, do ktorých nevedie žiadna zjazdovka.
Ak by sme postavili úrady vo všetkých mestách z množiny M, zjavne by sme pokryli celú krajinu – ak máme mesto x, ktoré v M neleží, tak keď z neho pôjdeme dostatočne dlho hore zjazdovkami, dostaneme sa časom nutne do nejakého mesta y, ktoré v M leží – a teda z mesta y vieme do mesta x poslať kontrolóra na lyžiach.
Ak teda žiadne dve mestá v množine M nie sú spojené cestou, našli sme riešenie.
Čo ale robiť, ak nejaké dve mestá u a v v množine M susedia? Ľahká pomoc. Odstránime jedno z nich a začneme od začiatku – teda vyriešime pôvodnú úlohu pre krajinu, ktorá má o mesto menej. Riešenie pre novú krajinu je potom aj riešením pre tú pôvodnú.
Prečo vyššie popísaný postup funguje? Povedzme, že u sme v krajine nechali a v sme z nej vyhodili. Keď nájdeme riešenie pre novú menšiu krajinu, sú dve možnosti: buď v u je úrad, alebo tam nie je. Ak v u je úrad, keď pridáme naspäť v, budeme vedieť poslať z u do v kontrolóra po ceste a teda je všetko v poriadku. No a ak v u úrad nie je, znamená to, že do u vie z nejakého iného mesta w prísť kontrolór. No a keďže do u nevedie žiadna zjazdovka, musí tento kontrolór prísť po ceste. Keďže ale aj u a v sú spojené cestou, vie kontrolór z mesta w prísť po ceste aj do mesta v. V oboch prípadoch teda sada úradov, ktorá pokrývala menšiu krajinu, pokrýva aj tú väčšiu.
Na záver podotkneme, že je zjavné, že vyššie popísaný postup je konečný. V každej iterácii buď už nájdeme riešenie, alebo začneme odznova s krajinou s menej mestami. No a keďže miest je len konečný počet, časom sa minú – teda časom určite dostaneme krajinu, v ktorej žiadne dve mestá z jej množiny M nie sú spojené cestou. (Rozmyslite si, že z toho vyplýva, že nejaké riešenie vždy existuje – v každej krajine sa teda nejak dajú rozmiestniť úrady.)

Priamočiara implementácia

Pre danú krajinu vieme množinu M zostrojiť v čase $O(n + z)$: jednoducho prejdeme zoznam zjazdoviek a pre každé mesto si spočítame, koľko z nich doň viedlo. Následnú kontrolu množiny M vieme spraviť v čase $O(c)$ tak, že prejdeme zoznam ciest a pre každú cestu skontrolujeme, či nanajvýš jeden jej koniec leží v množine M. Ak nájdeme cestu, ktorej oba konce ležia v M, jeden z nich z krajiny odstránime. Toto vieme spraviť v konštantnom čase – stačí mať pole boolovských premenných, v ktorom si pamätáme, ktoré mestá už boli odstránené. Tie potom ignorujeme pri konštrukcii aj kontrole novej množiny M v ďalších iteráciách.
Vyššie popísaný postup budeme postupne niekoľkokrát opakovať. Keďže začíname s n mestami, po nanajvýš $n - 1$ iteráciách skončíme a nájdeme riešenie. Dokopy teda dostávame časovú zložitosť $O(n(n + z + c))$.

Efektívna implementácia

Ako vieme predchádzajúce riešenie implementovať efektívnejšie?
Namiesto toho, aby sme množinu M zostrojovali vždy odznova, budeme si ju priebežne upravovať. Náš nový algoritmus bude fungovať nasledovne:

1. Vyššie popísaným spôsobom zostrojíme množinu M pre celú krajinu. O každom meste, ktoré do M nepatrí, si pamätáme, koľko zjazdoviek doň vedie.

2. Všetky mestá aktuálnej množiny M zaradíme do fronty na kontrolu.

3. Kým máme neprázdnu frontu na kontrolu, opakujeme: Vyber z fronty mesto u. Postupne prejdi všetky cesty vedúce z u. Ak žiadna nekončí v inom meste aktuálnej množiny M, je všetko v poriadku.

   V opačnom prípade ideme u z krajiny odstrániť. Odstránením u nastane jedna dôležitá zmena: z krajiny zmiznú aj zjazdovky, ktoré začínali v u. Prejdeme teda ich zoznam a ich koncovým mestám zmenšíme počet vchádzajúcich zjazdoviek. Ak takto nejakému mestu klesne počet vchádzajúcich zjazdoviek na nulu, pridáme ho do množiny M a zaradíme do fronty na kontrolu.

4. Akonáhle sa fronta vyprázdni, aktuálna množina M je riešením pre aktuálnu, a teda aj pre pôvodnú krajinu.

Vyššie popísaný algoritmus je korektný. Je zjavné, že si správne udržiavame množinu M. No a keď cyklus v kroku 3 dobehne, vieme, že každé mesto, ktoré je teraz v množine M, sme niekedy kontrolovali, a pri tej kontrole sme určite skontrolovali, že z neho nevedie cesta do žiadneho mesta, ktoré bolo do M pridané skôr. Preto naozaj v kroku 4 máme platné riešenie.
No a akú má tento postup časovú zložitosť? Na každú zjazdovku sa pozrieme nanajvýš dvakrát: raz v kroku 1, keď zostrojujeme prvú množinu M, a nanajvýš raz v kroku 3, ak jej začiatok niekedy odstránime z množiny M. Na každú cestu sa tiež pozrieme nanajvýš dvakrát: raz keď kontrolujeme jeden jej koniec, druhýkrát keď druhý. Ľahko teda nahliadneme, že vyššie popísaný algoritmus má časovú zložitosť $O(n + z + c)$, čiže lineárnu od veľkosti vstupu.

Listing programu (Python)

```
from collections import defaultdict
from queue import Queue

N, Z, C = [ int(_) for _ in input().split() ]
M = set()
zjazdovky = defaultdict(list)
cesty     = defaultdict(list)
vchadza_zjazdoviek = defaultdict(int)

for z in range(Z):
    x, y = [ int(_) for _ in input().split() ]
    zjazdovky[x].append(y)
    vchadza_zjazdoviek[y] += 1

for c in range(C):
    x, y = [ int(_) for _ in input().split() ]
    cesty[x].append(y)
    cesty[y].append(x)

Q = Queue()
for n in range(1,N+1):
    if vchadza_zjazdoviek[n] == 0:
        M.add(n)
        Q.put(n)

while not Q.empty():
    kde = Q.get()
    treba_zahodit = any( sused in M for sused in cesty[kde] )
    if not treba_zahodit: continue
    M.remove(kde)
    for sused in zjazdovky[kde]:
        vchadza_zjazdoviek[sused] -= 1
        if vchadza_zjazdoviek[sused] == 0:
            M.add(sused)
            Q.put(sused)

print(*list(M))
```

A-II-4 Online algoritmy 2

Ako vyzerá optimálne riešenie offline verzie našej úlohy? Človek, ktorý presne vie, že brigáda bude trvať n dní, má na výber dve možnosti: buď si zľavovú kartu kúpi alebo nie. Ak si ju nekúpi, dokopy zaplatí $2n$ eur. Ak si ju kúpi, zjavne si ju kúpi hneď na začiatku a potom dokopy zaplatí $k+n$ eur. Jasnovidec by si samozrejme vybral vždy lepšiu z týchto dvoch možností, a teda by dokopy na cestovnom minul $\min(2n, n+k)$ eur.

Pri návrhu algoritmu pre online verziu úlohy príliš nemáme na výber. Pred každým dňom sa musíme rozhodnúť, či už zľavovú kartu kúpiť alebo nie. No a akonáhle ju kúpime, už sa nerozhodujeme vôbec o ničom ďalšom. Preto majú všetky možné riešenia majú presne rovnakú formu: „Prvých d dní si každé ráno kúp lístok za 2 eurá. Po d dňoch si večer kúp zľavovú kartu a odvtedy si už každé ráno kupuj lístok za 1 euro.“ Potrebujeme si len správne zvoliť hodnotu d.
Je zjavné, že nechceme $d = 0$. Ak by sme hneď prvý večer zaplatili k za zľavovú kartu a bolo by $n = 0$ (čiže brigáda hneď skončila), minuli sme „nekonečnekrát“ viac peňazí ako by minul jasnovidec, ktorý n pozná.
Naopak, algoritmus pre $d = \infty$ (čiže algoritmus „nikdy si nekúp zľavovú kartu“) je zjavne 2-kompetitívny. Ak brigáda trvala nanajvýš k dní, zaplatíme rovnako ako jasnovidec. Ak trvala dlhšie, jasnovidec zaplatí $k+n$ eur, zatiaľ čo my zaplatíme $2n$ eur, no a $ALG = 2n < 2(k+n) = 2 \cdot OPT$.

Teraz si dokážeme, že žiaden online algoritmus pre túto úlohu nemôže byť lepší ako (3/2)-kompetitívny.
Pre algoritmus, ktorý nikdy kartu nekúpi, uvažujme prípad, kedy brigáda trvá $n = 3k$ dní. Algoritmus zaplatí $6k$ eur, zatiaľ čo optimálne offline riešenie je kúpiť hneď kartu a zaplatiť len $4k$ eur.
Pre všetky ostatné možné algoritmy uvažujme prípad, kedy brigáda skončí po presne d dňoch – teda tesne po tom ako kúpime kartu. Náš algoritmus v tomto prípade dokopy za cestovné zaplatí $2d + k$ eur.
Rozoberme dva prípady:

- Ak $d < k$, optimálny offline algoritmus kartu nekúpi vôbec a zaplatí $2d$ eur. Platí teda

$$\frac{ALG}{OPT} = \frac{2d+k}{2d} > \frac{2d+d}{2d} = \frac{3}{2}$$

- Ak $d \geq k$, optimálny offline algoritmus kartu kúpi hneď na začiatku a zaplatí $d + k$ eur. Tu dostávame

$$\frac{ALG}{OPT} = \frac{2d+k}{d+k} = \frac{4d+2k}{2d+2k} \geq \frac{3d+3k}{2d+2k} = \frac{3}{2}$$

Na záver si ukážeme, že algoritmus pre $d = k$ je naozaj (3/2)-kompetitívny. Inými slovami, optimálny riešením súťažnej úlohy je nasledovný algoritmus: „Prvých k dní si kúp celý lístok. Ak ešte stále brigáda neskončila, večer po k-tom dni si kúp zľavovú kartu a odvtedy až do konca brigády jazdi na polovičné lístky.“
Ak brigáda trvá menej ako k dní (čiže ak $n < k$), zaplatí tento algoritmus za cestovné presne rovnako ako by zaplatil jasnovidec, ktorý vopred pozná n.
Ak bude brigáda trvať k alebo viac dní, zaplatíme $2k$ eur za prvých k dní, potom k eur za kartu a na záver $n - k$ eur za zvyšné dni. Dokopy teda zaplatíme $2k + n$ eur. Optimálnym riešením bolo kúpiť kartu hned na začiatku a zaplatiť len $k + n$ eur.
Aký je najhorší možný pomer ceny medzi naším a optimálnym riešením? Keďže $k \leq n$, je $4k + 2n \leq 3k + 3n$, a odtiaľ dostaneme $(2k + n)/(k + n) \leq 3/2$. Tento algoritmus teda vždy zostrojí riešenie, ktoré je nanajvýš 1.5-krát horšie ako optimálne offline riešenie.

TRIDSIATY ŠTVRTÝ ROČNÍK OLYMPIÁDY V INFORMATIKE

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