A-I-1 Potrubná pošta

Začnime tým, že sa najskôr zamyslíme nad okrajovými prípadmi.

Ak $k = 0$, nemáme žiadne koncovky a teda nevieme nič prepojiť. Úloha teda nemá riešenie. Jedinou výnimkou je situácia, kedy $n = 1$: jediná existujúca kancelária je sama so sebou prepojená aj bez potrubí.

Podobne sme na tom pre $k = 1$. Pre $n = 1$ netreba robiť nič. Pre $n = 2$ máme dve kancelárie a v každej jednu koncovku. Ak sme na vstupe dostali potrubie medzi nimi, už sme hotoví, ak nie, toto potrubie postavíme. Pre $n > 2$ riešenie opäť neexistuje: akonáhle sú ľubovoľné dve kancelárie prepojené potrubím, k ani jednej z nich už nevieme pripojiť žiadnu inú.

Tým sme s okrajovými prípadmi skončili. Ukážeme si totiž, že pre $k \geq 2$ riešenie vždy existuje a pre všetky tieto vstupy ho vieme zostrojiť tým istým postupom.

Komponenty súvislosti

Na vstupe máme neorientovaný graf: kancelárie sú jeho vrcholy, už existujúce potrubia sú hrany. Tento graf si môžeme rozdeliť na komponenty súvislosti – teda skupiny kancelárií, ktoré už spolu vedia komunikovať.
V zadaní sme dostali záruku, že žiadne z potrubí na vstupe nie je zbytočné. Z tejto podmienky si vieme odvodiť, že v našom grafe nemôžu byť žiadne cykly. Totiž ak by sme vedeli nejakou postupnosťou rôznych potrubí poslať správu tak, aby skončila tam, kde začínala, znamenalo by to, že hociktoré jedno z týchto potrubí môžeme odstrániť. (Nech uv je jedno z týchto potrubí. Ak uv odstránime, nezmení sa nijak, ktoré kancelárie vedia komunikovať, lebo medzi u a v ešte stále vieme posielať správy cez zvyšok cyklu.)
Komponenty súvislosti nášho grafu sú teda *stromy* – súvislé acyklické grafy.

Každý strom s x vrcholmi má presne $x - 1$ hrán. Totiž keď postupne mažeme hrany stromu, zmazanie každej rozpojí jednu časť na dve, a teda o jedno zvýši počet komponentov. Aby sme dostali zo súvislého stromu x izolovaných vrcholov, musíme teda postupne zmazať $x - 1$ hrán.
Každá hrana má dva konce, preto platí, že každý strom s x vrcholmi má súčet stupňov vrcholov presne $2(x-1) = 2x-2$. Z toho vidíme, že v strome nemôžu mať všetky vrcholy veľký stupeň. Presnejšie, vieme dokázať, že každý strom s $x > 1$ vrcholmi má aspoň dva *listy* – vrcholy stupňa 1.
Dôkaz: V strome s viac ako jedným vrcholom má každý vrchol stupeň aspoň 1 (lebo žiaden vrchol nie je izolovaný). Ak by sme mali nanajvýš jeden vrchol stupňa 1 a teda aspoň $x - 1$ vrcholov, ktoré majú všetky stupeň aspoň 2, mali by sme súčet stupňov vrcholov aspoň $2x - 1$, čo už je priveľa.

Vzorové riešenie

Už máme všetko pripravené na to, aby sme vyriešili našu úlohu pre $k \geq 2$ terminály v každej miestnosti.
Graf zo vstupu rozdelíme na komponenty súvislosti a tieto očíslujeme od 1 po s. Toto vieme spraviť v lineárnom čase[1] napr. prehľadávaním do hĺbky alebo do šírky, alebo v skoro lineárnom čase algoritmom union-find.
V každom komponente si vyberieme dva konkrétne vrcholy stupňa 1, tie nazveme *terminál A* a *terminál B*. Špeciálnym prípadom budú komponenty tvorené jediným izolovaným vrcholom. Ten bude aj terminálom A aj terminálom B svojho komponentu.
Nové potrubia teraz pridáme nasledovne: pre každé i (od 1 po $s-1$) spojíme potrubím terminál B komponentu i s terminálom A komponentu $i + 1$.
Je zjavné, že takto spojíme všetky komponenty do jednej „reťaze“, a teda na konci dostaneme súvislú sieť potrubí obsahujúcu všetky kancelárie. Tiež je zjavné, že nikdy neprekročíme limit na počet koncoviek – z každej miestnosti, ktorá dostala nejaké nové potrubie, na konci vedú nanajvýš dve potrubia, čo je určite v limite, keďže riešime prípad s $k \geq 2$.

Listing programu (Python)

[1] Keďže graf na vstupe má zaručene menej hrán ako vrcholov, čas lineárny od veľkosti celého grafu je to isté, ako čas lineárny od počtu jeho vrcholov, teda $O(n)$.

```
from random import randint

N, K, M = [ int(_) for _ in input().split() ]
G = [ [] for n in range(N) ]
stupen = [ 0 for n in range(N) ]
for m in range(M):
    x, y = [ int(_) for _ in input().split() ]
    G[x].append(y)
    G[y].append(x)
    stupen[x] += 1
    stupen[y] += 1

if K == 0:
    if N == 1:
        print(0)
    else:
        print(-1)
    exit()

if K == 1:
    if N == 1:
        print(0)
    elif N == 2:
        if M == 0:
            print(1)
            print(0, 1)
        else:
            print(0)
    else:
        print(-1)
    exit()

F = 0
farba = [ None for n in range(N) ]
for n in range(N):
    if farba[n] is None:
        farba[n] = F
        todo = [n]
        while len(todo) > 0:
            x = todo.pop()
            for y in G[x]:
                if farba[y] is None:
                    farba[y] = F
                    todo.append(y)
        F += 1

komponenty = [ [] for f in range(F) ]
for n in range(N):
    komponenty[ farba[n] ].append(n)

print(F-1)
for f in range(F-1):
    for x in komponenty[f]:
        if stupen[x] <= 1: break
    for y in komponenty[f+1]:
        if stupen[y] <= 1: break
    print(x, y)
    stupen[y] += 1
```

A-I-2 Farebný plot

Nájdime si farbu, ktorá má momentálne na plote najviac výskytov, a spočítajme, koľko ich je. Ak ich je v, znamená to, že máme $w = n - v$ blokov inej farby. Ak $d < n - v$, máme primálo dní na to, aby sme natreli celý plot jednou farbou, a teda budeme chcieť vyrobiť čo najdlhší jednofarebný úsek niekde na plote. Ak $d \geq n - v$, budeme chcieť mať celý plot jednej farby (a skoro vždy to aj budeme vedieť dosiahnuť).

Ak máme dosť času

Jediný zlý prípad je situácia kedy $n > 1$ (máme aspoň dva bloky), $v = n$ (celý plot už je jednej farby) a $d = 1$ (je presne jeden deň, kedy musíme niečo prefarbiť).
V tejto situácii si musíme plot pokaziť. Optimálne je samozrejme pokaziť ho čo najmenej: prefarbíme blok na jednom z koncov, čím ešte stále dostaneme súvislý jednofarebný úsek tvorený zvyšnými $n - 1$ blokmi.
(Vo vyššie uvedenom rozbore je dôležité nezabudnúť na $n > 1$. Ak totiž máme len jeden blok, tak po jeho prefarbení nebude mať najdlhší jednofarebný úsek dĺžku $n - 1 = 0$. Prefarbený blok totiž bude novým úsekom dĺžky 1, len v inej farbe.)

Vo všetkých ostatných prípadoch s $d \geq n - v$ vieme dosiahnuť, aby bol celý plot jednej farby.
Ak máme jednofarebný plot a $d > 1$, vyberieme si jeden blok a ten postupne d-krát prefarbíme – napr. zakaždým na novú farbu, len v posledný deň späť na pôvodnú.
Ak máme $w > 0$ blokov zlej farby, pozrieme sa, koľko dní máme.
Ak $d = w$, tak je to jednoduché: postupne prefarbíme všetky zlé bloky na dobré.
Ak je d väčšie, tak predtým budeme $d - w$ dní robiť zbytočnú prácu, ktorá nám ale nepokazí riešenie. Vyberieme si jeden konkrétny blok zlej farby a ten v každý z prvých $d - w$ dní prefarbíme – zakaždým na novú zlú farbu.

Ak máme málo času

Tu si v prvom rade treba rozmyslieť, že farba, ktorej je momentálne na plote najviac, nemusí byť farbou, ktorú bude na konci mať najdlhší súvislý úsek. Jednoduchý protipríklad pre $d = 0$ je plot $1, 1, 2, 2, 2, 1, 1$. Vo všeobecnosti platí, že menej výskytov, ktoré sú ale bližšie pri sebe, nám môže dať lepšie riešenie.
Ak máme málo času (t.j., nemáme ho dosť na nafarbenie celého plota), je pomerne zjavné, že v optimálnom riešení budeme vo všetky dni používať tú istú farbu: tú, ktorú bude mať na konci súvislý úsek, ktorý vyrábame. (Ak by sme mali ľubovoľný postup, v ktorom niektoré dni používame aj inú farbu, tak by sme vedeli dostať lepšie riešenie tak, že všetky tie dni vynecháme, ostatné vykonáme a následne ešte predĺžime úsek, ktorý sme po nich dostali.)

O danom súvislom úseku tvorenom ℓ blokmi plota budeme hovoriť, že ho *stíhame prefarbiť*, ak platí, že niektorej farby je na ňom aspoň $\ell - d$.
Optimálne riešenie zjavne dostaneme tak, že nájdeme a následne prefarbíme najdlhší úsek, ktorý stíhame prefarbiť.
Vlastnosť, či stíhame nejaký úsek plota prefarbiť, je monotónna: ak ju má nejaký úsek, majú ju zjavne aj všetky jeho podúseky. Toto pozorovanie nám pomôže efektívne nájsť najdlhší takýto úsek.

Pomalšie riešenie

Ak by sme poznali farbu f, ktorú má mať výsledný úsek, vedeli by sme našu úlohu vyriešiť v lineárnom čase.
Predstavme si namiesto pôvodného plota nové pole, v ktorom máme 1 pre úseky nesprávnej farby a 0 pre úseky správnej farby. V takomto poli platí, že súčet každého úseku zodpovedá počtu dní, ktoré potrebujeme na jeho prefarbenie na správnu farbu. Hľadáme teda najdlhší úsek so súčtom (nanajvýš) d.
Spravíme to tak, že postupne pre každý možný začiatok úseku, ktorý stíhame prefarbiť, nájdeme jeho najvzdialenejší možný koniec.
Toto vieme spraviť napríklad tak, že si k tomuto novému poľu predpočítame jeho prefixové súčty. Pomocou nich potom vieme určiť súčet ľubovoľného úseku v konštantnom čase a pomocou toho vieme pre každý konkrétny začiatok nájsť jeho najvzdialenejší možný koniec binárnym vyhľadávaním v čase $O(\log n)$. Keďže možných začiatkov je n, tento algoritmus nájde najlepšie riešenie pre konkrétnu farbu v čase $O(n \log n)$.
Existuje aj ešte lepšie riešenie pomocou techniky dvoch pointrov. Kľúčové je uvedomiť si, že ak pre nejaký začiatok z poznáme jeho optimálny koniec k, tak pre začiatok $z + 1$ musí byť optimálny koniec aspoň k – totiž úsek od $z + 1$ po k určite ofarbiť vieme, keď sme vedeli ofarbiť celý úsek od z po k. Môžeme teda postupovať tak, že v cykle postupne vyskúšame všetky z od 1 po n a pre každé z nich postupne zväčšujeme k, až kým nenájdeme to optimálne. Toto riešenie má časovú zložitosť $O(n)$, keďže dokopy za celý jeho beh premennú k zväčšíme nanajvýš n-krát.
Samozrejme, toto riešenie ešte nie je kompletné, lebo optimálnu farbu nepoznáme. Môžeme ale postupne vyskúšať každú z $O(n)$ farieb, ktoré sa na začiatku vyskytujú na plote, a vybrať najlepšie spomedzi všetkých týchto riešení. Takto dostávame korektné riešenie s časovou zložitosťou $O(n^2)$ alebo $O(n^2 \log n)$, podľa toho, ktorou technikou riešime jednotlivé farby.

Vzorové riešenie

Vyššie popísané riešenie vieme zefektívniť. Stále budeme používať techniku dvoch pointrov, tentokrát to ale spravíme pre všetky farby naraz.

Aby sme to vedeli robiť, budeme si musieť pamätať trochu viac informácií o práve spracúvanom úseku poľa. Menovite si budeme pamätať nasledovné údaje:

- Pre každú farbu f si budeme pamätať jej počet výskytov $v[f]$ v aktuálnom úseku.
- Aby sme vedeli, či aktuálny úsek ešte vieme ofarbiť, potrebujeme poznať okrem jeho dĺžky ešte aj maximum z hodnôt $v[f]$. Toto si budeme preto udržiavať v pomocnej premennej. (Jemu zodpovedajúca farba f je tá, ktorá má v aktuálnom úseku najviac výskytov, a teda tá, na ktorú by sme ho za najmenej dní vedeli prefarbiť.)
- Aby sme vedeli premennú z predchádzajúceho bodu efektívne prepočítavať, budeme potrebovať vedieť, kedy sa zmenila. Toto zistíme tak, že si budeme pamätať *histogram* hodnôt $v[f]$: pre každú hodnotu x vo v si budeme pamätať $h[x]$ = koľkokrát sa táto hodnota nachádza v aktuálnom v.

  Napríklad $h[7] = 3$ znamená, že existujú práve tri rôzne farby f_1, f_2, f_3 ktoré majú v aktuálnom úseku po sedem výskytov.

Keď aktuálny úsek o políčko predĺžime, spravíme nasledovné operácie:

- Zvýšime o 1 počet výskytov farby f, ktorá práve pribudla.
- Upravíme histogram: napr. ak mala farba f doteraz 6 výskytov a už má 7, zmenšíme $h[6]$ a zväčšíme $h[7]$.
- Ak počet výskytov aktuálnej farby práve prekročil aktuálne maximum počtu výskytov, upravíme aj to.

Skrátenie úseku o políčko vyzerá skoro rovnako, s jediným rozdielom: ak sme zmenšili počet výskytov farby, ktorá mala aktuálne maximum počtu výskytov, pozrieme sa na histogram, aby sme vedeli, či zmenšiť aktuálne maximum, alebo ostáva rovnaké (keďže ešte existujú iné farby s tým počtom výskytov).

Podobne ako pri riešení s jednou farbou, aj tu celé riešenie dokopy beží v lineárnom čase – operácie, ktoré sme pridali navyše, robíme len $O(n)$-krát a každú vieme vykonať v konštantnom čase.
(S jednou drobnou výnimkou: v poslednej sade môžu byť čísla farieb veľké, takže nemôžeme na v použiť obyčajné pole ale nejakú implementáciu asociatívneho poľa – napr. vyvažovaným binárnym stromom alebo hešovacou tabuľkou. Časová zložitosť potom bude o chlp horšia od lineárnej, jej presná podoba bude závisieť od zvoleného riešenia implementácie v.)

Listing programu (Python)

```
from collections import defaultdict

def nova_farba(farba, zakazana = None):
    # vrati farbu inu od tej povodnej a pripadnej zakazanej
    # bezi v konstantnom case, cyklus spravi max 2 iteracie
    while True:
        farba = (farba + 1) % 10**9
        if farba != zakazana: return farba

# nacitame vstup

typ = int( input() )
N, dni = [ int(_) for _ in input().split() ]
plot = [ int(_) for _ in input().split() ]

# osetrime pripad N = 1: len dokola prefarbujeme

if N == 1:
    print( 1 )
    for d in range(dni):
        plot[0] = nova_farba( plot[0] )
        print( 0, plot[0] )
    exit()

# zistime najvacsi pocet usekov jednej farby

pocet = defaultdict(int)
for farba in plot: pocet[farba] += 1
najviac = max( pocet.values() )
najcastejsia = next( farba for farba in pocet if pocet[farba] == najviac )
```

```
# osetrime pripad, kedy si treba pokazit jednofarebny plot

if najviac == N and dni == 1:
    print( N-1 )
    print( 0, nova_farba( plot[0] ) )
    exit()

# osetrime pripad, kedy si pokazime a opravime jednofarebny plot

if najviac == N:
    print( N )
    for i in range(dni-1):
        plot[0] = nova_farba( plot[0], najcastejsia )
        print( 0, plot[0] )
    if dni > 0:
        print( 0, najcastejsia )
    exit()

# osetrime vsetky ostatne pripady, kedy mame dost dni na ofarbenie celeho plota

if dni >= N - najviac:
    print( N )
    ina = next( i for i in range(N) if plot[i] != najcastejsia )
    # ak treba, nadbytocne dni minieme zbytocnou pracou
    while dni > N - najviac:
        dni -= 1
        plot[ina] = nova_farba( plot[ina], najcastejsia )
        print( ina, plot[ina] )
    # a teraz vsetko natrieme spravne
    for i in range(N):
        if plot[i] != najcastejsia:
            print( i, najcastejsia )
    exit()

# a teraz nam uz ostava len pripad, ze nemame dost dni na cely plot
# a teda sa snazime najst najdlhsi usek ktory este vieme zmenit na jednofarebny

max_od, max_do = 0, 0

# pre kazdu farbu si pamatame, kolkokrat je v aktualnom useku
histogram = defaultdict(int)
for farba in plot: histogram[farba] = 0

# pre kazdy pocet si pamatame, kolko farieb ho momentalne ma
aggregate = defaultdict(int)
aggregate[0] = len(histogram)

od, do, max_pocet = 0, 0, 0

while True:
    if do - od - max_pocet <= dni:
        # aktualny usek je dobry
        # pozrieme, ci nemame rekord
        if do - od > max_do - max_od: max_od, max_do = od, do
        # skusime ho predlzit o prvok plot[do]
        if do == N: break
        aggregate[ histogram[ plot[do] ] ] -= 1
        histogram[ plot[do] ] += 1
        aggregate[ histogram[ plot[do] ] ] += 1
        max_pocet = max( max_pocet, histogram[ plot[do] ] )
        do += 1
    else:
        # aktualny usek je zly, skratime ho o plot[od]
        aggregate[ histogram[ plot[od] ] ] -= 1
        histogram[ plot[od] ] -= 1
        aggregate[ histogram[ plot[od] ] ] += 1
        if aggregate[max_pocet] == 0: max_pocet -= 1
        od += 1

# uz vieme, ktory je najlepsi usek, a treba ho len prefarbit

print( max_do - max_od )

pocet = defaultdict(int)
for farba in plot[max_od:max_do]: pocet[farba] += 1
najviac = max( pocet.values() )
najcastejsia = next( farba for farba in pocet if pocet[farba] == najviac )

for i in range(max_od, max_do):
    if plot[i] != najcastejsia:
        print( i, najcastejsia )
```

A-I-3 Policajti a zlodej

Ukážeme si najskôr pomalé riešenie, ktoré je ale všeobecnejšie a dá sa použiť pre veľa rôznych hier, a potom efektívne riešenie fungujúce len pre túto konkrétnu hru.

Všeobecnejšia technika riešenia takýchto hier

Predstavme si teraz, že na začiatku majú všetky pozície hry bielu farbu. (Pozícia zahŕňa polohy všetkých figúrok a informáciu, kto je práve na ťahu.)
Pozície teraz budeme chcieť všetky ofarbiť: tie, v ktorých má vyhrávajúcu stratégiu Peter, by mali skončiť modré a tie ostatné, v ktorých má vyhrávajúcu stratégiu Zuzka, červené.
Na začiatku vieme modrou ofarbiť pozície, kde Peter práve vyhral – teda na ťahu je Zuzka a niektorý policajt je na tom istom políčku ako zlodej.
Následne si môžeme vybrať ľubovoľnú bielu pozíciu a spraviť pre ňu nasledujúcu úvahu. Pozrime sa, kto je na ťahu. Ak je to Peter, vyskúšajme všetky možnosti, ako môže spraviť prvý ťah. Ak hociktorá z nich vedie do pozície, ktorá je už modrá (t.j. vieme, že z nej vie Peter vyhrať), aj túto pozíciu môžeme ofarbiť modrou. Navyše si vieme zapamätať aj tú možnosť prvého ťahu, ktorá nás doviedla do modrej pozície – toto je začiatok optimálnej stratégie pre Petra v tejto pozícii.
Ak je na ťahu Zuzka, tiež vyskúšame všetky možnosti, ako vie spraviť ona svoj prvý ťah. Tentokrát však bude naša úvaha trochu iná: ak *všetky možné* Zuzkine ťahy vedú do modrých pozícií, aj aktuálnu pozíciu môžeme ofarbiť na modro. Slovne: bez ohľadu na to, ktorý ťah si Zuzka vyberie, musí vyrobiť pozíciu, o ktorej už vieme, že v nej má Peter vyhrávajúcu stratégiu. V takejto situácii Zuzka teda nevie zabrániť Petrovej výhre.
Vyššie popísané úvahy nám dávajú induktívnu definíciu modrých pozícií. Implementovať ju vieme veľmi priamočiarou hrubou silou. Ofarbovanie budeme robiť v kolách. V každom kole prejdeme v cykle cez všetky možné pozície a pre každú, ktorá je ešte biela, použijeme vyššie popísané pravidlá a zistíme, či ju už neofarbiť na modro.
Tento proces zjavne musí časom skonvergovať – skôr alebo neskôr nastane kolo, v ktorom už žiadnu novú pozíciu neofarbíme. Vtedy pochopiteľne celý proces skončíme.
Zvyšok ofarbovania už bude priamočiary: všetky pozície, ktoré ostali biele, ofarbíme na červeno.
Tvrdíme teda, že vo všetkých týchto pozíciách už má vyhrávajúcu stratégiu Zuzka. Prečo toto môžeme urobiť? Ak je v takejto pozícii na ťahu Peter, žiadny jeho ťah nevedie do modrej pozície (lebo by sme túto ofarbili), a teda všetky jeho možné ťahy teraz vedú do červených pozícii. A ak je v takejto pozícii na ťahu Zuzka, existuje aspoň jeden ťah, ktorý nevedie do modrej, a teda vedie do červenej pozície – a toto je ťah, ktorý má v danej situácii zahrať. Zuzka takto vie zabezpečiť, že ak je začiatočná pozícia červená, tak všetky pozície počas celej hry budú červené, a teda policajti nikdy nechytia zlodeja.

Toto riešenie má polynomiálnu časovú zložitosť. Ak má hrací plán oba rozmery nanajvýš rovné n, tak existuje $O(n^2)$ možností, ako umiestniť jednu figúrku, a teda $O(n^6)$ pozícii v našej hre. Pri ofarbovaní vieme každú pozíciu skontrolovať v konštantnom čase (možností pre prvý ťah každého hráča je len konštantne veľa), jedno kolo ofarbovania teda prebehne v $O(n^6)$.
No a keďže v každom kole okrem posledného aspoň jednu pozíciu prefarbíme z bielej na modrú, kôl je nanajvýš rádovo toľko ako pozícii. Dokopy teda dostávame voľný horný odhad časovej zložitosti $O(n^{12})$.
(Tento odhad je voľný preto, že v skutočnosti kôl ofarbovania bude rádovo menej a vo väčšine z nich ofarbíme veľa nových pozícii na modro. Tesnejší odhad by však bol výrazne komplikovanejší a na nič ho vlastne nepotrebujeme, takže si ho odpustíme.)

Vzorové riešenie

Ukážeme si teraz, že v tejto konkrétnej hre Peter dokonca vždy vie vyhrať: úplne všetky pozície skončia modré. Nájdeme pre neho navyše jednoduchú vyhrávajúcu stratégiu, pre ktorú nebudeme ani len potrebovať prechádzať cez všetky pozície.

Petrova stratégia bude pozostávať z dvoch fáz. Prvú fázu začneme tým, že policatom v duchu dáme dve rôzne čiapky s písmenami R a S: bude z nich teda riadkový a stĺpcový policajt.

V tejto fáze bude našim cieľom spraviť ťah, po ktorom bude riadkový policajt v tom istom riadku ako zlodej a stĺpcový policajt v tom istom stĺpci ako zlodej.
Stratégia, ako toto dosiahnuť, je priamočiara: V každom ťahu sa pozrieme, či je riadkový policajt v správnom riadku, a ak nie, posunieme ho do riadku bližšieho k zlodejovi. Následne spravíme to isté pre stĺpce.
Na hracom pláne ktorého ani jeden rozmer neprekračuje n táto fáza skončí do n krokov. Dôkaz: Bez ujmy na všeobecnosti nech riadkový policajt začína vo vyššom riadku ako zlodej. Potom kým tento policajt prvýkrát nedosiahne zlodejov riadok, bude sa v každom kole hýbať len dodola. Po menej ako n takýchto krokoch by dosiahol spodný okraj a v najhoršom prípade práve vtedy prišiel do zlodejovho riadku.
(Akonáhle už jeden policajt dosiahol svoj cieľ a druhý ešte nie, prvý policajt bude v každom kole kopírovať pohyb zlodeja, takže aj po každom ďalšom ťahu bude tento policajt na tej istej jednej súradnici ako zlodej.)

Akonáhle sme spravili ťah, v ktorom riadkový aj stĺpcový policajt obsadili svoj riadok a stĺpec, začína druhá fáza. V tejto sa riadkový policajt chce hýbať po svojom riadku smerom k zlodejovi a zároveň stĺpcový policajt chce spraviť to isté vo svojom stĺpci.
Presnejšie budeme v druhej fáze postupovať nasledovne: Ak zlodej ostane stáť na mieste, obaja policajti sa posunú smerom k nemu. Ak sa zlodej pohne do iného riadku, riadkový policajt sa tiež pohne do toho istého riadku (v ktorom ostáva rovnako ďaleko od zlodeja), zatiaľ čo stĺpcový policajt sa pohne vo svojom stĺpci smerom ku zlodejovi. Pohyb zlodeja do iného stĺpca vyriešime analogicky.

Táto druhá fáza zaručene vždy skončí (tým, že policajt chytí zlodeja) a bude trvať menej ako $2n$ ťahov.
Argument je v podstate rovnaký ako v prvej fáze. Opäť máme vlastne dve samostatné jednorozmerné situácie – v prvej fáze sa policajt R snažil dostať do zlodejovho riadku a policajt S do jeho stĺpca, v druhej fáze sa policajt R snaží dostať do zlodejovho stĺpca a policajt S do jeho riadku. Rozdiel je ešte v tom, že v prvej fáze sme vedeli vždy pohnúť oboma policajtmi želaným smerom, v druhej fáze sa nám môže stať, že jeden policajt „ostane na mieste“ (keď sa pozeráme len na tú jeho súradnicu, ktorá nás teraz zaujíma) a iba druhý sa pohne.
Formálnejšie by sa tento dôkaz dal sformulovať napr. nasledovne: Bez ujmy na všeobecnosti predpokladajme, že na začiatku fázy je riadkový policajt naľavo a stĺpcový policajt hore od zlodeja. Potom po každom ťahu Petra toto bude opäť platiť (až kým zlodeja nechytí) – zlodej nevie prejsť okolo policajta a dostať sa na jeho druhú stranu. A navyše si všimnime súčet dvoch hodnôt: vzdialenosť prvého policajta od pravého okraja plus vzdialenosť druhého policajta od dolného okraja. V každom ťahu sa tento súčet zmenší (aspoň o 1, možno o 2). A keďže tento súčet nemôže byť nikdy záporný, hra musí časom skončiť – to sa ale stane len chytením zlodeja.

Riešenie, ktoré sme si práve popísali, má časovú zložitosť (na odohranie celej hry) lineárnu od dlhšieho rozmeru hracieho plánu – rádovo toľko ťahov v najhoršom prípade odohráme, pričom každý ťah vieme spraviť v konštantnom čase. Toto je zjavne optimálne.

A-I-4 O Vekslákbotovi a Pokladničke

Podúloha A: porovnaj a prefarbi

V našom riešení nebudeme mať žiadne obmedzenia. Pokyny budú vyzerať nasledovne:

1. červená, modrá → 2 tyrkysové
2. modrá → tyrkysová
3. červená → ružová
4. ružová, tyrkysová → 2 ružové

Kým máme aj červené aj modré žetóny, inštrukciou 1 odstránime po jednom a pridáme dva tyrkysové. Ak nám už neostalo nič, máme, čo sme chceli. Ak nám ostali ešte navyše modré, inštrukciou 2 ich postupne „prefarbíme“ na tyrkysové a opäť dostaneme situáciu so samými tyrkysovými žetónmi. Naopak, ak nám po používaní inštrukcie 1 ostali navyše nejaké červené žetóny (a teda ich bolo viac ako modrých), tieto inštrukciou 3 vymeníme za

ružové. Následne príde k slovu inštrukcia 4. Tou postupne všetky tyrkysové (ktoré vznikli skorším používaním inštrukcie 1) vymeníme za ružové. Všimnite si, že inštrukcia 4 sa dá použiť len ak už máme aspoň jeden ružový žetón – v situácii so samými tyrkysovými sa teda použiť nedá.

Podúloha B: sčítanie bez pomocnej farby

Zadanie tejto súťažnej úlohy sa veľmi podobá na príklad 2 zo študijného textu. Hlavný rozdiel je v tom, že sme na začiatku nedostali zelený žetón, ktorý by nám pomohol rozlíšiť, ktorý krok riešenia práve robíme.
Ak sa pozeráme len na počty žetónov v Pokladničke, tak si môžeme všimnúť, že hocijakú inštrukciu, ktorú by sa dalo použiť v začiatočnej situácii (kedy máme len červené a modré žetóny), sa bude dať použiť aj v želanej koncovej situácii (kedy máme mať tie isté počty červených a modrých žetónov) – to by ale znamenalo, že výpočet ešte neskončil.
Ak tomu chceme zabrániť, musíme nejako šikovne využiť obmedzenia.
Naše riešenie spraví oproti príkladu 2 niekoľko úprav:

- Zrušíme inštrukciu, ktorá vedela zahodiť žltý žetón. Ten si na konci riešenia ponecháme.
- Pridáme **na koniec** zoznamu inštrukcií novú „$\emptyset \to$ zelená“. Táto inštrukcia sa použije úplne v prvom kroku výpočtu, keďže žiadna iná sa ešte použiť nedá.
- Pridáme obmedzenie „zelená + žltá ≤ 1“. Vďaka tomuto obmedzeniu sa už na konci výpočtu nebude môcť znova použiť inštrukcia, ktorá nám z ničoho vyrobí zelený žetón.

Celá postupnosť inštrukcií vyzerá teda nasledovne:

1. červená, zelená → tmavočervená, zelená
2. modrá, zelená → tmavomodrá, zelená
3. zelená → žltá
4. tmavočervená, žltá → červená, fialová, žltá
5. tmavomodrá, žltá → modrá, fialová, žltá
6. $\emptyset$ → zelená

Podúloha C: násobenie

Základná myšlienka riešenia tejto podúlohy je, že násobenie je len opakované sčítanie. Ak budeme dokola opakovať operáciu „odober jeden červený žetón a potom za každý modrý pridaj jeden fialový“, dostaneme na konci zjavne práve c-krát m fialových žetónov.
V druhom príklade v študijnom texte sme videli techniku, ktorou vieme pridať m fialových žetónov a zároveň neprísť o modré: najskôr zmeníme všetky modré na tmavomodré, a potom každý tmavomodrý žetón vymeníme za jeden modrý a jeden fialový.
Na kontrolu toho, ktorým smerom práve meníme žetóny, použijeme jeden žetón inej farby: Ak máme zelený žetón, meníme modré. Keď sa modré minú, vymeníme zelený žetón za žltý. Odkedy máme žltý žetón, meníme tmavomodré žetóny naspäť. Keď sa tmavomodré žetóny minú, zahodíme aj žltý.
Tu je zodpovedajúca postupnosť inštrukcií:

1. modrá, zelená → tmavomodrá, zelená
2. zelená → žltá
3. tmavomodrá, žltá → modrá, fialová, žltá
4. žltá → $\emptyset$

Ak sa nám odniekiaľ zjaví zelený žetón, vyššie uvedený program pridá toľko fialových žetónov, koľko má na začiatku modrých. Teraz ešte potrebujeme doriešiť spúšťanie tohto cyklu. Na to nám stačí jedna nová inštrukcia:

5. červená → zelená

Táto inštrukcia je neskôr v poradí, a teda sa vykoná len vtedy, keď nevieme použiť žiadnu z predchádzajúcich – teda len ak práve nemáme ani zelený, ani žltý žetón. Zakaždým, keď táto situácia nastane (pričom prvýkrát je

to úplne na začiatku výpočtu), zmeníme jeden červený žetón za zelený a tým spustíme jedno kolo kopírovania modrých na fialové.
Teraz už máme korektný program na násobenie. Keď sa nám minú červené žetóny a dobehne posledné kolo kopírovania, tento program skončí a budeme mať $c \cdot m$ fialových žetónov – teda presne to, čo sme chceli – ale ešte aj m modrých žetónov. Tu už je ale ľahká pomoc:

6. modrá $\rightarrow$ $\emptyset$

Ak už nevieme použiť žiadnu z vyššie uvedených inštrukcií, príde na inštrukciu 6, ktorou na konci výpočtu postupne zahodíme všetky modré žetóny.

TRIDSIATY DEVIATY 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 2023