P-I-1 Potrubní pošta

Nejprve se zamysleme nad okrajovými případy:

- Pokud $k = 0$, nemáme žádné koncovky a tedy neumíme nic propojit. Úloha tedy nemá řešení. Jedinou výjimkou je situace, kdy $n = 1$: jediná stávající kancelář je sama se sebou propojena i bez potrubí.
- Podobně jsme na tom pro $k = 1$. Pro $n = 1$ není třeba dělat nic. Pro $n = 2$ máme dvě kanceláře a v každé jednu koncovku. Pokud jsme na vstupu dostali potrubí mezi nimi, už jsme hotovi; a když ne, toto potrubí postavíme. Pro $n > 2$ řešení opět neexistuje: jakmile jsou libovolné dvě kanceláře propojené potrubím, k ani jedné z nich již nevíme připojit žádnou jinou.

Tím jsme s okrajovými případy skončili. Ukážeme si totiž, že pro $k \geq 2$ řešení vždy existuje a pro všechny tyto vstupy ho umíme sestrojit stejným postupem.

Komponenty souvislosti

Na vstupu máme neorientovaný graf: kanceláře jsou jeho vrcholy, již existující potrubí jsou hrany. Tento graf si můžeme rozdělit na komponenty souvislosti – tedy skupiny kanceláří, které již spolu umí komunikovat.

V zadání jsme obdrželi záruku, že všechna v současnosti existující potrubí jsou nezbytná. Z této podmínky plyne, že v našem grafu nemohou být žádné cykly: Pokud bychom uměli nějakou posloupností různých potrubí poslat zprávu tak, aby skončila tam, kde začínala, znamenalo by to, že jedno libovolné z těchto potrubí můžeme odstranit (nechť toto potrubí například spojuje kanceláře u a v; jeho odstranění nijak neovlivní, které s kanceláří spolu mohou komunikovat, jelikož mezi kancelářemi u a v stále můžeme zprávy posílat přes zbytek cyklu).

Komponenty souvislosti našeho grafu jsou tedy *stromy* – souvislé acyklické grafy.

Každý strom s x vrcholy má přesně $x - 1$ hran: Když postupně mažeme hrany stromu, smazání každé rozpojí jednu část na dvě, a tedy o jedna zvýší počet komponent. Abychom dostali ze souvislého stromu x izolovaných vrcholů, musíme tedy postupně smazat $x - 1$ hran.

Každá hrana má dva konce, proto platí, že každý strom s x vrcholy má součet stupňů vrcholů přesně $2(x - 1) = 2x - 2$. Z toho vidíme, že ve stromu nemohou mít všechny vrcholy velký stupeň.

Přesněji, umíme dokázat, že každý strom s $x > 1$ vrcholy má alespoň dva *listy* – vrcholy stupně 1. Ve stromu s více než jedním vrcholem má každý vrchol stupeň alespoň 1 (neboť žádný vrchol není úplně oddělen od zbytku). Pokud bychom měli nanejvýš jeden vrchol stupně 1 a tedy alespoň $x - 1$ vrcholů, které mají všechny

stupeň alespoň 2, měli bychom součet stupňů vrcholů alespoň $2x - 1$, což už je příliš mnoho.

Vzorové řešení

Nyní máme vše připraveno k tomu, abychom úlohu vyřešili pro $k \geq 2$ terminály v každé místnosti.

Vstupní graf rozdělíme na komponenty souvislosti a tyto očíslujeme od 1 do s. Toto umíme udělat v lineárním čase, například prohledáváním do hloubky nebo do šířky (jelikož graf na vstupu má méně hran než vrcholů, čas lineární ve velikosti celého grafu je totéž, jako čas lineární od počtu jeho vrcholů, tedy $\mathcal{O}(n)$).

V každé komponentě si vybereme dva libovolné vrcholy stupně 1, ty nazveme *terminál A* a *terminál B*. Speciálním případem budou komponenty tvořené jediným izolovaným vrcholem. Ten bude i terminálem A i terminálem B své komponenty.

Nové potrubí nyní přidáme následovně: pro každé i od 1 do $s - 1$ spojíme potrubím terminál B komponenty i s terminálem A komponenty $i + 1$.

Je zjevné, že takto spojíme všechny komponenty do jednoho „řetězu“, a tedy na konci dostaneme souvislou síť potrubí obsahující všechny kanceláře. Také je zjevné, že nikdy nepřekročíme limit k na počet koncovek – z každé kanceláře, která dostala nějaké nové potrubí, na konci vedou nanejvýš dvě potrubí, což je určitě v limitu, jelikož řešíme případ s $k \geq 2$.

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

komponenty = [ [] for f in range(F) ]
for n in range(N):
    komponenty[ barva[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
```

P-I-2 Barevný plot

Najděme si barvu, která má na začátku na plotě nejvíce výskytů, a spočítejme, kolik jich je. Je-li jich v, znamená to, že máme $w = n - v$ latěk jiné barvy. Pokud $d < n - v$, máme málo dní na to, abychom natřeli celý plot jednou barvou, a tedy budeme chtít vyrobit co nejdelší jednobarevný úsek někde na plotě. Pokud $d \geq n - v$, budeme chtít mít celý plot jedné barvy (a skoro vždy toho i budeme umět dosáhnout).

Máme-li dost času

Jediný špatný případ je situace kdy $n > 1$ (máme alespoň dvě laťky), $v = n$ (celý plot už je jedné barvy) a $d = 1$ (je přesně jeden den, kdy musíme něco přebarvit).

V této situaci si musíme plot pokazit. Optimální je samozřejmě pokazit ho co nejméně: přebarvíme laťku na jednom z konců, čímž stále dostaneme souvislý jednobarevný úsek tvořený zbývajícími $n - 1$ laťkami.

(Ve výše uvedeném rozboru je důležité nezapomenout na $n > 1$. Pokud totiž máme jen jednu laťku, tak po jejím přebarvení nebude mít nejdelší jednobarevný úsek délku $n - 1 = 0$. Přebarvená laťka totiž bude novým úsekem délky 1, jen v jiné barvě.)

Ve všech ostatních případech s $d \geq n - v$ umíme přebarvit celý plot na jednu barvu. Máme-li jednobarevný plot a $d > 1$, vybereme si jednu laťku a tu postupně d-krát přebarvíme – např. pokaždé na novou barvu, jen poslední den zpět na tu původní.

Máme-li $w > 0$ latěk špatné barvy, podíváme se, kolik dní máme. Pokud $d = w$, tak je to jednoduché: postupně přebarvíme všechny špatné laťky na dobré. Pokud je d větší, tak předtím budeme $d - w$ dní dělat zbytečnou práci, která nám ale nezkazí řešení. Vybereme si jednu konkrétní laťku špatné barvy a ten v každém z prvních $d - w$ dní přebarvíme – pokaždé na novou špatnou barvu.

Máme-li málo času

Uvažme nyní případ, že $d < n - v$, a tedy nemáme dost času na to plot přebarvit na jednobarevný. V tomto případě barva, které je momentálně na plotě nejvíce, nemusí být barvou, kterou bude na konci mít nejdelší souvislý úsek. Jednoduchý protipříklad pro $d = 0$ je plot $1, 1, 2, 2, 2, 1, 1$. Obecně platí, že méně výskytů, které jsou ale blíže u sebe, nám mohou dát lepší řešení.

Je poměrně zjevné, že v optimálním řešení budeme všechny dny používat tutéž barvu: tu, kterou bude mít na konci souvislý úsek, který vyrábíme. Pokud bychom měli libovolný postup, ve kterém některé dny používáme i jinou barvu, tak bychom uměli dostat lepší řešení tak, že všechny ty dny vynecháme, ostatní provedeme a následně ještě prodloužíme úsek, který jsme po nich dostali.

O daném souvislém úseku tvořeném ℓ laťkami plotu budeme říkat, že jej *stíháme přebarvit*, pokud platí, že některá barva se na něm vyskytuje alespoň $(\ell - d)$-krát. Optimální řešení zjevně dostaneme tak, že najdeme a následně přebarvíme nejdelší úsek, který stíháme přebarvit.

Vlastnost, zda stíháme nějaký úsek plotu přebarvit, je monotónní: má-li ji nějaký úsek, mají ji zjevně i všechny jeho podúseky. Toto pozorování nám pomůže efektivně najít nejdelší takový úsek.

Pomalejší řešení

Pokud bychom znali barvu f, kterou má mít výsledný úsek, uměli bychom naši úlohu vyřešit v lineárním čase.

Představme si místo původního plotu nové pole, ve kterém máme 1 pro úseky nesprávné barvy (různé od f) a 0 pro úseky správné barvy. V takovém poli platí, že součet každého úseku odpovídá počtu dní, které potřebujeme k jeho přebarvení na správnou barvu. Hledáme tedy nejdelší úsek se součtem (nanejvýš) d. Pro každý možný začátek úseku budeme hledat jeho nejvzdálenější možný konec.

Toto můžeme udělat například tak, že si k tomuto novému poli předpočítáme jeho prefixové součty. Pomocí nich pak umíme určit součet libovolného úseku v konstantním čase a pomocí toho víme pro každý konkrétní začátek najít jeho nejvzdálenější možný konec binárním vyhledáváním v čase $\mathcal{O}(\log n)$. Jelikož možných začátků je n, tento algoritmus najde nejlepší řešení pro konkrétní barvu v čase $\mathcal{O}(n \log n)$.

Existuje i ještě lepší řešení pomocí techniky dvou pointerů. Klíčové je uvědomit si, že pokud pro nějaký začátek z známe jeho optimální konec k, tak pro začátek $z + 1$ musí být optimální konec alespoň k – totiž úsek od $z + 1$ po k určitě přebarvit umíme, když jsme uměli obarvit celý úsek od z po k. Můžeme tedy postupovat tak, že v cyklu postupně vyzkoušíme všechny začátky z od 1 do n a pro každý z nich

postupně zvětšujeme k, dokud nenajdeme to optimální. Toto řešení má časovou složitost $\mathcal{O}(n)$, protože dohromady za celý jeho běh proměnnou k zvětšíme nanejvýš n-krát.

Samozřejmě, toto řešení ještě není kompletní, neboť optimální barvu neznáme. Můžeme ale postupně vyzkoušet každou z $\mathcal{O}(n)$ barev, které se na začátku vyskytují na plotě, a vybrat nejlepší z všech těchto řešení. Takto dostáváme korektní řešení s časovou složitostí $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^2 \log n)$, podle toho, kterou technikou řešíme jednotlivé barvy.

Vzorové řešení

Výše popsané řešení umíme zefektivnit. Stále budeme používat techniku dvou pointerů, tentokrát to ale uděláme pro všechny barvy najednou. Abychom to uměli dělat efektivně, budeme si muset pamatovat trochu více informací o právě zpracovávaném úseku pole. Jmenovitě si budeme pamatovat následující údaje:

- Pro každou barvu f si budeme pamatovat její počet výskytů $v[f]$ v aktuálním úseku.
- Abychom věděli, jestli aktuální úsek ještě umíme obarvit, potřebujeme znát kromě jeho délky ještě i maximum z hodnot $v[f]$. Toto si budeme proto udržovat v pomocné proměnné. Jemu odpovídající barva f je ta, která má v aktuálním úseku nejvíc výskytů, a tedy ta, na kterou bychom ho za nejméně dní uměli přebarvit.
- Abychom věděli proměnnou z předchozího bodu efektivně přepočítávat, budeme potřebovat vědět, kdy se změnila. Toto zjistíme tak, že si budeme pamatovat *histogram* hodnot $v[f]$: pro každou hodnotu x ve v si budeme pamatovat $h[x] =$ kolikrát se tato hodnota aktuálně nachází v poli v.

  Například $h[7] = 3$ znamená, že existují právě tři různé barvy f_1, f_2, f_3 které mají v aktuálním úseku po sedmi výskytech.

Když aktuální úsek o políčko prodloužíme, provedeme následující operace:

- Zvýšíme o 1 počet výskytů barvy f, která právě přibyla.
- Upravíme histogram: Např. pokud barva f měla 6 výskytů a nyní jich má 7, zmenšíme $h[6]$ a zvětšíme $h[7]$.
- Pokud počet výskytů aktuální barvy právě překročil aktuální maximum počtu výskytů, upravíme i toto maximum.

Zkrácení úseku o políčko vypadá skoro stejně, s jediným rozdílem: pokud jsme zmenšili počet výskytů barvy, která měla aktuální maximum počtu výskytů, podíváme se na histogram, abychom věděli, zda zmenšit aktuální maximum, nebo zůstává stejné (jelikož ještě existují jiné barvy s tímto počtem výskytů).

Podobně jako u řešení s jednou barvou, i zde celé řešení dohromady běží v lineárním čase – operace, které jsme přidali navíc, provádíme jen $\mathcal{O}(n)$-krát a každou umíme provést v konstantním čase.

V poslední sadě mohou být čísla barev velká, takže nemůžeme na v použít obyčejné pole, ale nějakou implementaci asociativního pole – např. vyvažovaným

binárním stromem nebo hešovací tabulkou. Časová složitost pak může být o něco málo horší než lineární, její přesná podoba bude záviset na zvoleném řešení implementace v.

```
from collections import defaultdict

def nova_barva(barva, zakazana = None):
    # vratí barvu různé od té původní a případně zakázané
    # konstantí časová složitost: cyklus proběhne nejvýše 2x
    while True:
        barva = (barva + 1) % 10**9
        if barva != zakazana: return barva

# načteme vstup

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

# speciální případ N = 1: jen stále přebarvujeme

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

# najdeme barvu, která se vyskytuje nejčastěji, a počet jejích výskytů

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

# speciální případ: musíme pokazit jednobarevný plot

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

# speciální případ: máme jednobarevný plot, ten pokazíme a poté zase opravíme

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

# všechny ostatní případy, kdy můžeme udělat celý plot jednobarevný

if dni >= N - najviac:
    print(N)
    ina = next(i for i in range(N) if plot[i] != najcastejsia)
    # nadbytečné dny strávíme zbytečnou prací
    while dni > N - najviac:
        dni -= 1
        plot[ina] = nova_barva(plot[ina], najcastejsia)
```

```
        print(ina, plot[ina])
    # a poté vše obarvíme správně
    for i in range(N):
        if plot[i] != najcastejsia:
            print(i, najcastejsia)
    exit()

# zbývá případ, kdy nemáme čas na obarvení celého plotu jednou barvou
# musíme tedy najít nejdelší úsek, který umíme přebarvit na jednobarevný

max_od, max_do = 0, 0

# pro každou barvu si pamatujeme, kolikrát se vyskytuje v aktuálním úseku
histogram = defaultdict(int)
for barva in plot: histogram[barva] = 0

# naopak, pro každý počet p si pamatujeme, kolik barev se v aktuálním úseku
# vyskytuje právě p-krát
aggregate = defaultdict(int)
aggregate[0] = len(histogram)

od, do, max_pocet = 0, 0, 0

while True:
    if do - od - max_pocet <= dni:
        # aktuální úsek lze udělat jednobarevný
        # zkontrolujeme, zda nemáme rekord
        if do - od > max_do - max_od: max_od, max_do = od, do
        # zkusíme aktuální úsek prodloužit o laťku 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:
        # aktuální úsek nelze obarvit jednobarevně, musíme ho
        # zkrátit o laťku 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

# už známe nejlepší úsek, je ho tedy potřeba přebarvit

print(max_do - max_od)

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

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

P-I-3 Policajti a zloděj

Ukážeme si nejprve pomalé řešení, které je ale obecnější a dá se použít pro mnoho různých her, a pak efektivní řešení fungující jen pro tuto konkrétní hru.

Obecná technika řešení takových her

Představme si, že na začátku mají všechny možné pozice hry bílou barvu. (Pozice zahrnuje polohy obou policistů i zloděje a informaci, kdo je právě na tahu.) Pozice teď budeme chtít všechny obarvit: ty, ve kterých má vyhrávající strategii policisté, by měli skončit modré, a ty ostatní, ve kterých má vyhrávající strategii zloděj, červené.

Na začátku můžeme modrou obarvit pozice, kde policisté právě vyhráli – tedy některý policista je na stejném políčku jako zloděj.

Následně si můžeme vybrat libovolnou bílou pozici b a udělat pro ni následující úvahu. Podívejme se, kdo je na tahu. Pokud jsou to policisté, vyzkoušejme všechny možnosti, jak mohou udělat první tah. Pokud některá z nich vede do pozice, která je už modrá (tj. víme, že z ní umí policisté vyhrát), i tuto pozici b můžeme obarvit modře. Navíc si umíme zapamatovat i tu možnost prvního tahu, která nás dovedla do modré pozice – toto je začátek optimální strategie pro policisty v této pozici.

Pokud je na tahu zloděj, také vyzkoušíme všechny možnosti, jak umí udělat on svůj první tah. Tentokrát však bude naše úvaha trochu jiná: pokud *všechny možné* zlodějovy tahy vedou do modrých pozic, i aktuální pozici můžeme obarvit modře. Slovně: libovolným tahem, který zloděj může provést, se dostane do pozice, o které už víme, že v ní mají vyhrávající strategii policisté. V takové situaci zloděj neumí zabránit výhře policistů.

Výše popsané úvahy nám dávají induktivní definici modrých pozic. Implementovat ji umíme velmi přímočarou hrubou silou. Obarvování budeme dělat v kolech. V každém kole projedeme v cyklu přes všechny možné pozice a pro každou, která je ještě bílá, použijeme výše popsané pravidla a zjistíme, zda ji již neobarvit modře.

Tento proces zjevně musí časem zkonvergovat - dříve nebo později nastane kolo, ve kterém již žádnou novou pozici neobarvíme. Tehdy pochopitelně celý proces skončíme.

Zbytek obarvování už bude přímočarý: všechny pozice, které zůstaly bílé, obarvíme červeně. Tvrdíme tedy, že ve všech těchto pozicích už má vyhrávající strategii zloděj. Proč je tomu tak? Pokud jsou v takové pozici b na tahu policisté, žádný jejich tah nevede do modré pozice (jinak bychom pozici b také obarvili modře), a tedy všechny jeho možné tahy nyní vedou do červených pozic. A je-li v takové pozici na tahu zloděj, existuje alespoň jeden tah, který nevede do modré pozice, a tedy vede do červené pozice – a toto je tah, který má v dané situaci zloděj provést. Zloděj takto umí zajistit, že je-li začáteční pozice červená, tak všechny pozice během celé hry budou červené, a tedy policisté ho nikdy nechytí.

Toto řešení má polynomiální časovou složitost. Existuje $\mathcal{O}(rs)$ poloh každé osoby, a tedy $\mathcal{O}(r^3s^3)$ pozic v naší hře. Při obarvování víme každou pozici zkontrolovat

v konstantním čase (možností pro první tah každého hráče je jen konstantně mnoho), jedno kolo obarvování tedy proběhne v $\mathcal{O}(r^3 s^3)$.

No a jelikož v každém kole kromě posledního alespoň jednu pozici přebarvíme z bílé na modrou, kol je nanejvýš řádově tolik jako pozici. Dohromady tedy dostáváme hrubý horní odhad časové složitosti $\mathcal{O}(r^6 s^6)$.

Tento odhad je jen hrubý proto, že ve skutečnosti kol obarvování bude řádově méně a ve většině z nich obarvíme spoustu nových pozic modře. Těsnější odhad by však byl výrazně komplikovanější a na nic ho vlastně nepotřebujeme, takže si ho odpustíme.

Vzorové řešení

Ukážeme si nyní, že v této konkrétní hře policisté dokonce vždy umí vyhrát: naprosto všechny pozice skončí modré. Najdeme pro něj navíc jednoduchou vyhrávající strategii, pro kterou nebudeme ani potřebovat procházet přes všechny pozice.

Strategie policistů bude sestávat ze dvou fází. První fázi začneme tím, že policistům v duchu dáme dvě různé čepice s písmeny R a S: bude z nich tedy řádkový a sloupcový policista.

V této fázi bude naším cílem provést tah, po kterém bude řádkový policista ve stejném řádku jako zloděj a sloupcový policista ve stejném sloupci jako zloděj.

Strategie, jak toho dosáhnout, je přímočará: V každém tahu se podíváme, zda je řádkový policista ve správném řádku, a pokud ne, posuneme ho do řádku bližšího ke zloději. Následně uděláme totéž pro sloupce.

Tato fáze skončí do $\max(r, s)$ kroků. Důkaz: Bez újmy na obecnosti ať řádkový policista začíná ve vyšším řádku než zloděj. Potom než tento policista poprvé nedosáhne zlodějův řádek, bude se v každém kole hýbat jen dolů. Po méně než r takových krocích by dosáhl spodní okraj a v nejhorším případě právě tehdy přišel do zlodějova řádku.

Jakmile už jeden policista dosáhl svého cíle a druhý ještě ne, první policista bude v každém kole kopírovat pohyb zloděje, takže i po každém dalším tahu bude tento policista na téže jedné souřadnici jako zloděj.

Jakmile jsme udělali tah, ve kterém řádkový i sloupcový policista obsadili svůj řádek a sloupec, začíná druhá fáze. V této se řádkový policista chce hýbat po svém řádku směrem ke zloději a zároveň sloupcový policista chce udělat to samé ve svém sloupci.

Přesněji budeme ve druhé fázi postupovat následovně: Pokud zloděj zůstane stát na místě, oba policisté se posunou směrem k němu. Pokud se zloděj pohne do jiného řádku, řádkový policista se také pohne do téhož řádku (ve kterém zůstává stejně daleko od zloděje), zatímco sloupcový policista se pohne ve svém sloupci směrem ke zloději. Pohyb zloděje do jiného sloupce vyřešíme analogicky.

Tato druhá fáze zaručeně vždy skončí (tím, že policista chytí zloděje) a bude trvat méně než $r + s$ tahů.

Argument je v podstatě stejný jako v první fázi. Opět máme vlastně dvě samostatné jednorozměrné situace – v první fázi se policista R snažil dostat do zlodějova

řádku a policista S do jeho sloupce, ve druhé fázi se policista R snaží dostat do zlodějova sloupce a policista S do jeho řádku. Rozdíl je ještě v tom, že v první fázi jsme uměli vždy pohnout oběma policisty požadovaným směrem, ve druhé fázi se nám může stát, že jeden policista „zůstane na místě“ (když se díváme jen na tu jeho souřadnici, která nás nyní zajímá) a teprve druhý se pohne.

Formálněji by se tento důkaz dal zformulovat například následovně: Bez újmy na obecnosti předpokládejme, že na začátku fáze je řádkový policista nalevo a sloupcový policista nahoru od zloděje. Potom po každém tahu policistů toto bude opět platit (dokud zloděje nechytí) – zloděj neumí projít kolem policisty a dostat se na jeho druhou stranu. Uvažme nyní součet dvou hodnot: vzdálenost prvního policisty od pravého okraje plus vzdálenost druhého policisty od dolního okraje. V každém tahu se tento součet zmenší (alespoň o 1, možná o 2). A jelikož tento součet nemůže být nikdy záporný, hra musí časem skončit – to se ale stane jen chycením zloděje.

Řešení, které jsme si právě popsali, má časovou složitost (k odehrání celé hry) $\mathcal{O}(r + s)$ – řádově tolik tahů v nejhorším případě odehrajeme, přičemž každý tah umíme provést v konstantním čase. Toto je zjevně optimální.

P-I-4 O Vekslákbotovi a Pokladničce

Podúloha a: Porovnej a přebarvi

V našem řešení nebudeme mít žádná omezení. Pokyny budou vypadat následovně:

1. červená, modrá → 2tyrkysová
2. modrá → tyrkysová
3. červená → růžová
4. růžová, tyrkysová → 2růžová

Dokud máme červené i modré žetony, instrukcí 1 odebereme jeden od každého a za to přidáme dva tyrkysové. Pokud nám žádné modré ani červené nezůstanou, máme, co jsme chtěli. Pokud zůstaly ještě nějaké modré, instrukcí 2 je postupně „přebarvíme“ na tyrkysové a opět skončíme s $c + m$ tyrkysovými.

Naopak, pokud nám po používání instrukce 1 zůstaly nějaké červené žetony (a tedy jich bylo více než modrých), tak je pomocí instrukce 3 vyměníme za růžové. Následně přijde ke slovu instrukce 4, pomocí níž „přebarvíme“ všechny tyrkysové (jež vznikly instrukcí 1) na růžové. Všimněte si, že instrukci 4 lze použít pouze tehdy, kdy již máme alespoň jeden růžový žeton – v situaci se samými tyrkysovými použít nelze.

Podúloha B: Sčítání bez pomocné barvy

Zadání této úlohy se velmi podobá příkladu 2 ze studijního textu. Hlavní rozdíl je v tom, že jsme na začátku nedostali zelený žeton, jenž by nám pomohl rozlišit, který krok řešení právě provádíme.

Pokud se budeme dívat jen na počty žetonů v pokladničce, můžeme si všimnout, že kterákoli instrukce, jež by se dala vykonat za začátku (kdy máme jen červené a

modré žetony), by byla vykonatelná i v koncovém stavu, do nějž se chceme dostat (tehdy máme stejný počet červených i modrých žetonů jako na začátku) – to by ale znamenalo, že výpočet ještě neskončil. Chceme-li tomu zabránit, musíme šikovně využít omezení.

Naše řešení oproti příkladu 2 udělá několik úprav:

- Zrušíme instrukci, která zahazovala žlutý žeton, a prostě si jej na konci řešení ponecháme.
- **Na konec** seznamu instrukcí přidáme novou instrukci $\emptyset \to$ zelená. Tato instrukce se použije v úplně prvním kroku výpočtu, kdy se ještě žádná jiná použít nedá.
- Přidáme omezení zelená + žlutá ≤ 1. Díky tomuto omezení již na konci výpočtu nebudeme moci znovu použít instrukci, jež z ničeho vyrobí zelený žeton.

Náš program bude tedy mít omezení zelená+žlutá ≤ 1 a pokyny budou vypadat následovně:

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

Podúloha C: Násobení

Hlavní myšlenka řešení této úlohy je, že násobení je jen opakované sčítání. Pokud budeme dokola opakovat operaci „odeber jeden červený žeton a potom za každý modrý přidej jeden fialový“, dostaneme na konci právě c-krát m fialových žetonů.

Ve druhém příkladě studijního textu jsme viděli techniku, jak přidat m fialových žetonů a zároveň nepřijít o žádné modré: Nejdříve změníme všechny modré na tmavomodré a potom každý tmavomodrý žeton vyměníme za jeden modrý a jeden fialový.

Na kontrolu toho, kterým směrem právě měníme žetony, používáme jeden žeton jiné barvy: Pokud máme zelený žeton, měníme modré. Když nám modré dojdou, vyměníme zelený žeton za žlutý, a dokud máme žlutý žeton, měníme tmavomodré žetony nazpět. Když nám dojdou tmavomodré žetony, zahodíme i žlutý žeton.

Zde je příslušná posloupnost instrukcí:

1. modrá, zelená $\to$ tmavomodrá, zelená
2. zelená $\to$ žlutá
3. tmavomodrá, žlutá $\to$ modrá, fialová, žlutá
4. žlutá $\to \emptyset$

Pokud se nám odněkud objeví zelený žeton, výše uvedený program přidá tolik fialových žetonů, kolik má na začátku modrých. Teď ještě potřebujeme dořešit spuštění tohoto cyklu. Na to stačí jedna nová instrukce:

5. červená → zelená

Tato instrukce je v pořadí později, takže se vykoná jen tehdy, když nemůžeme použít žádnou z těch předcházejících – tedy pouze když zrovna nemáme ani zelený, ani žlutý žeton. Pokaždé, když tato situace nastane (poprvé je to hned na začátku výpočtu), vyměníme jeden červený žeton za zelený, čímž spustíme jedno kolo kopírování modrých na fialové.

Nyní už máme korektní program na násobení: Když dojdou červené žetony a doběhne poslední kolo kopírování, tento program skončí a budeme mít $c \cdot m$ fialových žetonů – tedy přesně tolik, kolik jsme chtěli – ale také m modrých žetonů. To je však jednoduché vyřešit:

6. modrá → $\emptyset$

Pokud nemůžeme použít žádnou z výše uvedených instrukcí, přijde řada na instrukci 6, která na konci výpočtu zahodí všechny modré žetony.