B-II-1 Nepexeso

Pomerne rýchlo vieme prísť s jednoduchým riešením, ktoré pre každú akčnú kartičku prehľadá celý úsek $\langle l, r\rangle$ a hľadá v ňom inú farbu ako f. Toto riešenie je však pomalé, pretože pri každej otázke musíme prejsť prinajhoršom celým radom. Jeho časová zložitosť je $O(qn)$. Na druhej strane implementácia tohto riešenie je priamočiara a bol to jednoduchý spôsob ako zaručene získať aspoň 3 body.

Napriek všetkému však predchádzajúci prístup vôbec nepôsobil tak pomaly. Áno, v najhoršom prípade musí prejsť všetky karty v zadanom úseku, to však nastane, iba ak sú farby v ňom rovnaké ako farba akčnej kartičky. Ak sú farby kartičiek zvolené náhodne, nájdenie inej farby je pomerne rýchlo. Častokrát sa totiž stane, že už kartička na pozícii l má inú farbu ako f, a preto ju môžeme vypísať.

To vedie k dôležitému pozorovaniu. Ak je farba kartičky na pozícii l iná ako f, stačí (v konštantnom čase) vrátiť túto pozíciu. Inak hľadáme kartičku, ktorej farba je iná ako farba l-tej kartičky. Všimnite si, že v druhej časti podmienky už farba f nefiguruje. Tým pádom, zadávané otázky buď vieme vyriešiť hneď, alebo ich farba zodpovedá farbe kartičky v rade. A na takýto typ otázok sa vieme veľmi ľahko pripraviť vopred, skôr ako ich vôbec dostaneme.

Aby sme otázky typu „Nájdi pozíciu v úseku $\langle l, r\rangle$, na ktorej je kartička inej farby ako kartička na pozícii l.“ vedeli odpovedať rýchlo, pre každú pozíciu si predpočítame pozíciu najbližšej kartičky napravo od nej, ktorej farba je iná. Túto pozíciu pre i-tu kartičku si označme $P[i]$. Odpovedať na otázky (f, l, r) je teraz jednoduché. Najskôr sa pozrieme, či je farba kartičky l iná ako farba f. Ak áno, vypíšeme l. Ak nie, tak sa pozrieme, či je $P[i] \leq r$ a ak áno, vypíšeme hodnotu $P[i]$, inak vypíšeme -1. Uvedomme si, že $P[i]$ je najbližšia kartička, ktorej farba je rozdielna od l-tej kartičky. Ak je teda prvá iná kartička ďalej ako r, celý úsek $< l, r >$ pozostáva z kariet rovnakej farby.

Ostáva nám už len vypočítať hodnoty $P[i]$. Budeme pritom postupovať odzadu. Posledná kartička nemá napravo od seba nič, preto povieme, že $P[n] = n+1$. A keď poznáme všetky hodnoty $P[i+1]$ až $P[n]$, ľahko vypočítame hodnotu $P[i]$. Buď sú farby i-tej a $i+1$-vej karty rôzne a vtedy $P[i] = i+1$, alebo sú rovnaké a preto je k nim najbližšia iná karta rovnaká a $P[i] = P[i+1]$.

Predpočítať pole $P[]$ nám trvá čas $O(n)$, pretože ním iba raz prejdeme a následne už každú otázku zodpovedáme v konštantnom čase. Celková časová zložitosť je teda $O(n+q)$. Pamäťová zložitosť je $O(n)$.

Listing programu (Python)

```
n, q = map(int, input().split())
F = list(map(int, input().split()))
P = [n+1 for i in range(n)]

for i in range(n-2,-1,-1):
    if F[i] == F[i+1]:
        P[i] = P[i+1]
    else:
        P[i] = i+1

for i in range(q):
    f, l, r = map(int, input().split())
    # pozor, l a r sú číslované od 1 po n, pole je ale indexované od 0 po n-1
    if f != F[l-1]:
        print(l)
    else:
        if P[l-1] < r:
            print(P[l-1]+1)
        else:
            print(-1)
```

B-II-2 Topiaci sa sneh

Táto úloha sa dala riešiť rôznymi spôsobmi, s využitím pomerne známych algoritmov a dátových štruktúr ako binárne vyhľadávanie a vyhľadávacie stromy (napr. `set` v C++), ale aj bez ich použitia iba pomocou triedenia. Vo vzorovom riešení si predstavíme dva hlavné prístupy, akými sa dalo na úlohu pozerať.

Riešenie pomocou binárneho vyhľadávania

Samozrejme najzaujímavejšou otázkou o kope snehu je, ktorý deň sa roztopí úplne. Všetky predchádzajúce dni

sa z nej totiž stopí toľko snehu, aká je aktuálna teplota t_i, čo sa počíta ľahko. Iba ten posledný deň to bude inak. Zoberme si teda kopu snehu, ktorá na záhrade pribudla i-ty deň a pokúsme sa zistiť, kedy sa roztopila.
Ako prvé môžeme simulovať jej roztápanie. Postupne z nej budeme odčítavať hodnoty $t_i, t_{i+1}, \ldots$ až kým nedostaneme záporné číslo (alebo nulu aby sme boli presný). Všimnime si však, že pomerne ľahko vieme odpovedať aj na otázku, či sa v j-ty deň z i-tej kopy ešte niečo stopilo. Ak by i-ta kopa bola neobmedzene veľká, po j-ty deň by sa z nej stopilo práve $t_i + t_{i+1} + \cdots + t_{j-1} = T$ snehu. Táto kopa má však veľkosť v_i, preto vieme, že ak $T < v_i$, tak do j-teho dňa sa ešte celá nestihla stopiť. Môžeme preto postupne skúšať, ktorý deň i-ta kopa ešte existovala až kým prvýkrát nedostaneme odpoveď nie.
Takéto riešenie však navádza na použitie binárneho vyhľadávania. Ak tento algoritmus nepoznáte, nevadí, je vskutku jednoduchý. Namiesto toho, aby sme skúšali hodnoty j postupne budeme niektoré hodnoty preskakovať. Na začiatku vieme, že i-ta kopa sa stopí niekedy medzi dňom i až n. Vyberieme si preto j v strede tohto intervalu a zistíme, či v tento deň na kope ešte bol sneh. Ak áno, nemusíme skúšať žiadne menšie j, pretože deň keď sa táto kopa stopí bude určite za dňom j. A naopak, ak zistíme, že táto kopa sa stopila niekedy predtým, tak už nebudeme skúšať väčšie hodnoty ako j. V každom prípade sme sa po jednej otázke zbavili polovice možností. Takto budeme pokračovať aj v ďalších krokoch, vždy si zvolíme deň, ktorý je zhruba v strede zostávajúceho intervalu. Vďaka tomu namiesto n skúšaní musíme spraviť iba $\log n$ overení.
Pri tomto riešení sme zanedbali ešte jednu vec, spôsob ako rýchlo zistiť, koľko snehu sa roztopilo medzi i-tym a j-tym dňom. Na to použijeme takzvané prefixové polia. Ešte predtým ako budeme binárne vyhľadávať si spočítame pole $P[]$, pre ktoré platí, že $P[i] = t_1 + t_2 + \cdots + t_i$. Toto pole vieme vyplniť jedným prechodom zľava doprava, pretože platí, že $P[i] = P[i-1] + t_i$. Vďaka tomuto poľu teraz vieme odpovedať na otázky, koľko snehu sa roztopilo medzi dňami i až j. Odpoveďou bude $P[j-1] - P[i-1]$. Skúste sa zamyslieť prečo.
Ak už pre každú kopu vieme zistiť, ktorý deň sa roztopí, ľahko skonštruujeme riešenie. Vytvoríme si dve polia – $S[i]$, do ktorého si budeme značiť, koľko snehu sa stopilo v i-ty deň z kôp, ktoré v tento deň zmizli a $C[i]$, kde si budeme pamätať počet kôp, ktoré zmizli v i-ty deň. Pre každú kopu nájdeme pomocou binárneho vyhľadávania deň, v ktorom sa roztopila a upravíme hodnoty v poliach $S[]$ a $C[]$.
Následne budeme prechádzať deň po dni a rovno vypisovať odpoveď. Pritom si budeme pamätať, koľko kôp je ešte na Sysľovej záhrade. i-ty deň pribudne jedna nová kopa a $C[i]$ sa ich roztopí. Z nich stečie $S[i]$ snehu a zo všetkých zvyšných kôp, ktoré sú na Sysľovej záhrade sa roztopí t_i snehu.
Spočítanie prefixových polí a odpovede má zložitosť $O(n)$. Pre každú kopu však musíme ešte vypočítať, ktorý deň sa roztopila, čo robíme binárnym vyhľadávaním, zložitosť tohto kroku je $O(n \log n)$, čo je aj výsledná časová zložitosť. Pamäťová zložitosť je $O(n)$.

Listing programu (Python)

```
n = int(input())
V = list(map(int, input().split()))
T = list(map(int, input().split()))

# prefixove pole
P = [0]
for i in range(n):
    P.append(P[i] + T[i])

S = [0]*n
C = [0]*n
for i in range(n):
    # binarne vyhladavanie
    zac, kon = i-1, n
    while kon - zac > 1:
        stred = (kon + zac) // 2
        if V[i] > P[stred+1] - P[i]:
            zac = stred
        else:
            kon = stred
    if kon != n: # kopa sa do konca neroztopila
        C[kon] += 1
        S[kon] += V[i] - (P[kon] - P[i])

pocet = 0
for i in range(n):
    pocet += 1
    pocet -= C[i]
    print(S[i] + pocet*T[i], end = ' ' if i != n-1 else '\n')
```

Riešenie pomocou triedenia

V tejto časti si predstavíme riešenie, ktoré nepotrebovalo žiadne zložité algoritmy či dátové štruktúry, vystačilo si s jednoduchým triedením.
Keď si zoberieme nejaké jednoduché riešenie, je veľmi ľahké si držať prehľad o tom, ktoré kopy sú aktuálne na Sysľovej záhrade. Tá ťažká, a pomalá, operácia však je, keď sa snažíme zistiť, ktoré sa roztopili. Na základe ich veľkostí sa totiž môžu rôzne predbiehať. Pomohlo by nám teda, keby sme si ich vedeli usporiadať podľa toho, v akom poradí sa roztopia. Nemusíme vedie, kedy sa tak stane, stačí ak budeme vedieť, v akom poradí.
Zoberme si dve kopy i a j, pričom predpokladajme, že $i < j$. Vieme zistiť, či sa kopa i roztopí neskôr ako kopa j? Áno, stačí ak bude platiť, že v j-ty deň bude na i-tej kope stále viac snehu ako na kope j-tej. To znamená, že musí platiť $v_i - t_i - t_{i+1} - \ldots t_{j-1} > v_j$, čo vieme ľahko prepísať na $v_i > v_j + t_i + t_{i+1} + \ldots t_{j-1}$. Uvedomme si, že kopu j sme zväčšili tak, akoby ju nasnežilo takisto v i-ty deň. A pri kopách, ktoré nasnežilo rovnaký deň vieme ľahko zistiť, ktorá sa roztopí skôr – tá väčšia.
Aby sme vedeli jednoducho porovnali všetky kopy snehu, budeme sa tváriť, že všetky nasnežilo v prvý deň. K veľkosti každej kopy v_i preto pripočítame $t_1 + t_2 + \ldots t_{i-1}$. Tieto čísla teraz vieme jednoducho usporiadať, čím dostaneme poradie, v ktorom sa budú kopy roztápať.
Zvyšok riešenia je už len o počítaní výsledku deň po dni. Počas toho si budeme pamätať, koľko kôp snehu je aktuálne na Sysľovej záhrade. Každý deň pribudne jedna nová kopa. Následne musíme zistiť, ktoré kopy sa roztopili a koľko snehu z nich stieklo. Práve na to použijeme pole, v ktorom sú kopy zoradené podľa toho, v akom poradí sa roztápajú. Hľadané kopy totiž môžu byť iba na začiatku a roztopili sa vtedy, ak je súčet teplôt do daného dňa väčší ako ich upravená veľkosť, keďže sa tvárime, že všetky nasnežilo v prvý deň. Ak už o nejakej kope zistíme, že sa roztopila, posunieme sa v poli ďalej a späť sa už nevraciame, čo si vieme pamätať jedným ukazovateľom. Každý deň tento ukazovateľ posunieme o toľko kôp, koľko sa ich daný deň roztopilo a popritom si spočítame, ako veľké pritom boli a zmenšíme počet kôp snehu, ktoré sú aktuálne u Sysľa na záhrade. Nakoniec k výsledku pripočítame t_i za každú kopu, ktorá v záhrade ostala aj na konci tohto dňa.
Takmer všetko v tomto riešení je vyriešené prejdení jedného poľa, čo má lineárnu zložitosť. Jedinou výnimkou je triedenie a preto je zložitosť tohto riešenia taktiže $O(n \log n)$.

Listing programu (Python)

```
n = int(input())
V = list(map(int, input().split()))
T = list(map(int, input().split()))

N = [] # nove velkosti kop
t = 0
for i in range(n):
    N.append(t + V[i])
    t += T[i]
N.sort()

pocet = 0
kde = 0
t = 0
for i in range(n):
    pocet += 1
    vys = 0
    while kde < n and N[kde] <= t + T[i]:
        vys += N[kde] - t
        pocet -= 1
        kde += 1
    t += T[i]
    print(vys + pocet*T[i], end = ' ' if i != n-1 else '\n')
```

B-II-3 O Miškovej sestre

Použijeme podobnú techniku ako v domácom kole: dynamické programovanie.
Predstavme si, že ideme popri rade kociek zľava doprava a postupne sa o každej kocke rozhodneme, či ju necháme alebo zahodíme. Čo si potrebujeme pamätať o už spracovaných kockách? Zjavne jediné, čo nás zaujíma, je posledné písmeno na poslednej z kociek, ktorú sme si nechali. Len od toho totiž závisí, či nasledujúcu kocku musíme zahodiť alebo ju môžeme nechať v rade.

Aby sme nemuseli ošetrovať špeciálny prípad, kedy sme zatiaľ všetky kocky zahodili a žiadnu nenechali, môžeme si predstaviť, že úplne na začiatku radu je ešte jedna kocka, ktorej slovo končí na nejaké úplne nové písmeno (také, ktoré nikde inde nie je použité) a ktorú necháme na mieste. Toto špeciálne písmeno si označíme napr. `@`.

Označme $p_\star$ počet spôsobov, ako sa dá z doteraz spracovaných kociek niektoré vyhodiť a niektoré nechať tak, aby sme ako posledné písmeno mali práve písmeno $\star$. Úplne na začiatku, pred tým ako začneme spracúvať kocky, je teda $p_@ = 1$ a všetky ostatné hodnoty v p sú nuly, lebo iným písmenom ako `@` končiť nevieme.

Predstavme si teraz, že sme začali na začiatku radu, spracovali sme nejaké kocky a poznáme im zodpovedajúce hodnoty p. Ako ďalšia v poradí nech je kocka, ktorá začína písmenom x a končí písmenom y. Ako sa zmenia hodnoty p po spracovaní tejto kocky?
Ak chceme vyrobiť rad kociek končiaci písmenom $z \neq y$, musíme to spraviť tak, že ho vyrobíme z predchádzajúcich kociek a potom túto novú kocku zahodíme. Počet spôsobov p_z sa teda nezmenil. Jediné, čo sa zmení, je hodnota p_y.
Ak chceme vyrobiť rad končiaci písmenom y, máme na výber dva rôzne scenáre. Prvý je taký, že novú kocku zahodíme a rad končiaci písmenom y vyrobíme z predchádzajúcich kociek. Aktuálna hodnota p_y nám hovorí, koľkými spôsobmi vieme spraviť toto. Druhý scenár vyzerá tak, že z predchádzajúcich kociek vyrobíme rad končiaci *ľubovoľným iným písmenom ako* x a zaň pridáme našu kocku. Dokopy teda platí:

$$\text{nové } p_y = \text{staré } p_y + \text{súčet všetkých hodnôt } p_\star \text{ okrem } p_x$$

Na to, aby sme pomocou tohto vzorca vypočítali novú hodnotu p_y, potrebujeme čas priamo úmerný veľkosti abecedy, čiže $O(a)$. Dokopy teda n kociek spracujeme v čase $O(na)$, a teda máme 8-bodové riešenie.

Aby sme toto riešenie ešte zrýchlili, použijeme ešte jeden drobný trik. Okrem všetkých hodnôt $p_\star$ si budeme pamätať aj hodnotu s rovnú ich súčtu. Potom vieme v konštantnom čase zistiť, o koľko treba zväčšiť aj hodnotu p_y, aj hodnotu s: presne o $(s - p_x)$.
Takto každú kocku spracujeme v konštantnom čase, a teda dostávame riešenie s časovou zložitosťou $O(n + a)$.

Listing programu (Python)

```
from string import ascii_lowercase

s = 1
p = { x:0 for x in ascii_lowercase }
p['@'] = 1

n = int( input() )
for i in range(n):
    slovo = input()
    x, y  = slovo[0], slovo[-1]
    zmena = s - p[x]
    p[y]  += zmena
    s     += zmena

print(s)
```

B-II-4 Šimon a vesmírna stanica

Prvé dve podúlohy sú vlastne návodom k tomu, ako potom riešiť tretiu hlavnú podúlohu. Prejdeme si detailne cez prvú podúlohu a potom rovno ukážeme všeobecné riešenie tretej.
Vo všetkých riešeniach budeme dva smery pohybu pre jednoduchosť volať „doprava“ a „doľava“, netreba však zabúdať, že stanica je cyklická – teda keď Šimon pôjde dostatočne dlho „doprava“, dostane sa určite na miesto, kde začínal.

Podúloha A: je modul jeden?

Keďže Šimon nevie rozlíšiť jeden modul od iného inak, musí si nejak pomôcť svetlom. Ale ako spoznať, či je to naozaj nový modul a nie iný, v ktorom je tiež zasvietené?

Trik je v tom, že ak ide o dva rôzne moduly, prepnutie svetla v jednom z nich sa neprejaví v druhom. Keď si toto uvedomíme, už ľahko vymyslíme spôsob kontroly. Jedno možné riešenie vyzerá nasledovne:

```
1. rozsvieť
2. pohni sa doprava
3. ak tam je zhasnuté, je to iný modul, a teda sú aspoň dva, koniec
4. ak tam je rozsvietené, zhasni
5. pohni sa naspäť doľava
6. ak tam ostalo rozsvietené, je to iný modul, a teda sú aspoň dva, koniec
7. ak je aj tam zhasnuté, muselo ísť o ten istý modul, a teda je len jeden, koniec
```

Iné možné riešenie vyzerá

Podúlohy B a C: koľko je modulov?

Jedno možné všeobecné riešenie podúlohy C bude vyzerať tak, že postupne otestujeme, či sú moduly aspoň dva (to už vieme robiť), či sú aspoň tri, aspoň štyri, a tak ďalej. Akonáhle takýto test prvýkrát zlyhá, budeme presne vedieť, koľko modulov má naša stanica.
Teraz si rozmyslíme, ako skontrolovať, či je modulov aspoň k. Postup bude zovšeobecnením toho z podúlohy A: najskôr dostatočne veľa modulov v rade rozsvietime a potom sa pozrieme, čo sa stane, keď ich pôjdeme postupne zhášať. Ukážeme si najskôr samotné riešenie:

```
1. rozsvieť v module, kde začínaš
2. (k-1)-krát sa posuň doprava a rozsvieť aj tam (ak bolo zhasnuté)
3. zhasni v module, kde práve si
4. (k-1)-krát sprav nasledovné:
       posuň sa naspäť doľava
       ak si prišiel do modulu, kde je zhasnuté, je modulov menej ako k
       zhasni v module, kde práve si
5. ak si sa dostal až sem, modulov bolo aspoň k
```

Prečo toto riešenie funguje? Rozmyslime si, čo sa stane, ak máme modulov aspoň k a čo sa stane, ak ich je menej.
Ak máme modulov aspoň k, v prvých dvoch krokoch postupne navštívime k rôznych modulov a v každom z nich rozsvietime, a potom v krokoch 3 a 4 cez ne prejdeme naspäť a v každom zhasneme. Všetko zafunguje ako má a dáme správnu odpoveď.
Ak máme modulov menej ako k, niekedy počas kroku 2 sa „zacyklíme“ – teda prídeme znova do modulu, v ktorom sme už boli. Špeciálne teda aj modul, v ktorom skončíme krok 2, sme určite navštívili (aspoň) dvakrát. V kroku 3 v ňom zhasneme. Ale potom keď sa v kroku 4 vraciame naspäť, určite sa ešte (aspoň) raz do tohto modulu vrátime. No a keď sa tak stane, uvidíme, že v ňom už zhasnuté a z toho vieme, že sme v ňom už boli, a teda že modulov musí byť menej ako k.

Podúloha D: efektívne riešenie

Koľko zhruba krokov urobí Šimon pri predchádzajúcom riešení? Ak je skutočný počet modulov n, Šimon postupne spraví 1 krok doprava, 1 doľava, 2 doprava, 2 doľava, ..., až nakoniec n doprava a potom n doľava. Dokopy teda spraví rádovo n^2 krokov (a rádovo toľko isto prepnutí svetla).
Ako vieme tento postup urýchliť? Všimnime si, čo sa stane, ak použijeme ten istý postup, len nebudeme skúšanú veľkosť modulu zväčšovať vždy o 1, ale rýchlejšie. Napríklad sa zamyslime, čo sa stane, keď vyššie popísaný postup použijeme pre $k = 10$, ale modulov je len 7. V krokoch 1 a 2 postupne navštívime 10 modulov a v každom zažneme ak treba. V krokoch 3 a 4 sa nám potom ale podarí zhasnúť len 7-krát (lebo len toľko modulov naozaj máme) a po nasledujúcom pohybe už nájdeme v module tmu.
Ak teda použijeme vyššie použitý postup a budeme si navyše počítať, v koľkých moduloch sa nám naozaj podarilo zhasnúť, vieme presne povedať, či bol modul len jeden, či boli presne dva, presne tri, ..., či ich bolo presne $k - 1$ alebo či ich muselo byť aspoň k.

No a po tomto vylepšení už môžeme skúšať len niektoré vhodne zvolené hodnoty k. Celé riešenie teda bude vyzerať napríklad tak, že vyššie popísaným postupom budeme postupne zisťovať presný počet modulov ak je menší ako 2, ak je menší ako 4, potom 8, 16, 32, a tak ďalej. (Rovnako dobre sme mohli použiť napríklad mocniny 10, alebo inú postupnosť ktorá rastie exponenciálne.)

Prečo má tento postup lepšiu časovú zložitosť? Počítajme. V poslednom kole (v tom, kedy už odhalíme správny počet modulov) prejdeme celú stanicu nanajvýš dvakrát smerom tam a presne raz celú smerom späť. V predposlednom kole prejdeme nanajvýš celú stanicu tam a späť. O kolo skôr to už muselo byť nanajvýš pol stanice tam a späť, predtým štvrtina, a tak ďalej. No a keď toto celé spočítame, dostaneme, že dokopy spravíme nanajvýš $7n$ krokov. A keďže každému kroku zodpovedá najviac jedno prepnutie svetla, je aj tých len lineárne veľa – a teda celé toto riešenie má časovú zložitosť lineárnu od počtu modulov v stanici.
Formálne, k ľubovoľnému n vieme nájsť t také, že $2^{t-1} \leq n < 2^t$. Pre toto n spraví vyššie popísaný algoritmus postupne t kontrol, pričom bude postupne voliť $k = 2^1, 2^2, \ldots, 2^t$. Počas týchto kontrol spraví menej ako $2^1 + 2^2 + \cdots + 2^t < 2^{t+1}$ krokov smerom doprava a menej ako $2^1 + 2^2 + \cdots + 2^{t-1} + n < 2^t + n$ krokov smerom doľava. Dokopy teda spraví menej ako $2^{t+1} + 2^t + n = 6 \cdot 2^{t-1} + n \leq 6n + n = 7n$ krokov.

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