Vzorové řešení soutěže Kasiopea 2013

1 Skořápky

K úloze pravděpodobně není co dodat, vyřešili jste ji úspěšně všichni. K vyřešení stačilo pouze udržovat si pozici kuličky a měnit ji společně se skořápkářem.

Lukáš Folwarczný

2 Permutace

Jak si předávání představit? Bude se nám hodit rozklad na cykly. Otužilce si představme jako vrcholy grafu. Hrana z u do v vede právě tehdy, když otužilec u předává nápoj otužilci v.

Jak graf vypadá? Z každého vrcholu vede právě jedna hrana, do každého vrcholu vstupuje právě jedna hrana, vrcholů je konečně mnoho. Graf se nám tedy rozpadne na soustavu cyklů. Cyklus je posloupnost vrcholů $v_0, \ldots, v_{k-1}$ taková, že z v_i vede hrana do v_{i+1} pro každé $i \in \{0, \ldots, k-2\}$ a navíc ještě vede hrana z v_{k-1} do v_0. Takovýto cyklus má k vrcholů, délkou cyklu budeme rozumět právě tuto hodnotu k.

Při hromadném předávání nápojů se nápoj posune o jednu pozici na cyklu. Tedy nápoj je u svého majitele právě tehdy, když počet provedených předávání je dělitelný k. Odtud už vidíme jak nalézt řešení – jedná se o nejmenší společný násobek délek všech cyklů.

Pro efektivní nalezení délek cyklů použijeme šikovný průchod. U každého otužilce si poznačíme, zda už jsme navštívili cyklus, v němž se nachází. Pak stačí procházet seznamem otužilců – pro otužilce, jehož cyklus ještě nebyl navštíven, následujeme předávku nápojů, dokud se nevrátíme. Takovýto způsob nalezení délek cyklů pak bude mít časovou složitost $\Theta(N)$.

Jak si můžete rozmyslet, nejmenší společný násobek čísel a, b je roven $a \cdot b/\mathrm{nsd}(a, b)$, kde $\mathrm{nsd}(a, b)$ je největší společný dělitel těchto dvou čísel. Ke spočtení největšího společného dělitele slouží stařičký Euklidův algoritmus. Pokud jste se s ním ještě nesetkali, nahlédněte do kuchařky o teorii čísel.[1]

Zbývá určit celkovou časovou složitost algoritmu. Složitost Euklidova algoritmu je $\mathcal{O}(\log l)$, kde l je menší z obou čísel. Označme délky cyklů jako $l_0, \ldots, l_{m-1}$. Pro každé i je $\log l_i \leq l_i$. Čas strávený výpočtem Euklidova algoritmu je $\sum \log l_i$, což je díky odhadu menší nebo rovno N. Protože jsme již určili časovou složitost první části jako $\Theta(N)$, je taková i celková časová složitost.

Závěrem ještě poznamenejme, že tabulka předávání nápojů reprezentuje v matematické řeči permutaci, jedno předání nápojů reprezentuje složení s touto permutací, opakované skládání jedné permutace se často nazývá mocninou per-

[1] `http://ksp.mff.cuni.cz/viz/kucharky/teorie-cisel`

mutace. Pokud bychom pokračovali v aritmetické analogii, zadáním úlohy by nejspíš bylo určit nejmenší kladný logaritmus identické permutace při daném základu.

Lukáš Folwarczný

3 Barevné proužky

Úlohu vyřešíme metodou, které se říká zametací přímka (anglicky sweeping line). Začátky a konce intervalů si setřídíme vzestupně a pak pomyslnou svislou přímkou pojedeme zleva doprava a budeme si pamatovat, kolik různě barevných intervalů zrovna protíná. Tento počet se může změnit pouze v začátcích a koncích intervalů, takže už jen stačí navrhnout, jak efektivně upravovat stav přímky v těchto bodech.

Na to si pořídíme datovou strukturu, která umí vkládat a vyhledávat prvky v logaritmickém čase na dotaz. Takovou je například binární vyhledávací strom, v našem případě použijeme jeho implementaci `set` z `C++` knihovny `STL`. V instanci S této struktury si budeme uchovávat, které barvy aktuálně naše přímka protíná. Pak si pořídíme ještě jednu datovou strukturu – buď další vyhledávací strom, nebo třeba `map` z knihovny `STL` a v té si budeme udržovat, kolik které barvy aktuálně máme.

Zpracování jednoho kraje intervalu bude vypadat následovně:

1. Potkáme-li začátek intervalu: Přidáme barvu do S a dané barvě zvýšíme počet o jedna.
2. Potkáme-li konec intervalu: Snížíme dané barvě počet o jedna a pokud jsme se dostali na nulu, tak ji odebereme z S.
3. Po každé operaci se podíváme, kolik různých barev v setu zrovna máme.

Ještě poznamenejme, že jelikož souřadnice udávají čísla sloupečků, je třeba v rámci jednoho sloupce nejdříve zpracovávat začátky intervalů a potom teprve konce. Časová složitost algoritmu je počet zpracovávaných krajů intervalů krát čas na jeden dotaz, tedy $\mathcal{O}(NlogN)$.

Karel Tesař

4 Nulový strom

K řešení této úlohy použijeme průchod do hloubky, pokud jej neznáte, nahlédněte do příslušné kuchařky.[2]

Strom zakořeníme ve vrcholu 0. Provedeme úvodní pozorování: chceme-li přičíst nebo odečíst jedničku od ohodnocení nějakého vrcholu, je tato jednička

[2] `http://ksp.mff.cuni.cz/viz/kucharky/prochazky-po-grafech`

přičtena či odečtena i od všech vrcholů, které jsou na cestě mezi tímto vrcholem a vrcholem 0. Myšlenka řešení bude tedy zhruba následující: při zpracování vrcholu se zeptáme jeho synovských vrcholů, kolik vyžadují operací přičtení a odečtení, zjistíme dopad těchto operací na ohodnocení vrcholu a navýšíme počty provedení tak, aby se ohodnocení vrcholu vynulovalo, výsledek pak předáme otcovskému vrcholu.

Nyní si projděme celý postup znovu a doplňme jej o detaily tak, abychom měli jistotu, že získáme optimální výsledek.

Rekurzivní funkce průchodu do hloubky na vrcholu x vrátí hodnoty M_x a P_x, které značí nejmenší nutný počet provedení operace odečtení a přičtení jedničky k vynulování ohodnocení vrcholu x a všech jeho potomků.

Průchod vrcholu x bude vypadat následovně: pokud je vrchol x listem, je situace jednoduchá, vrátíme $P_x = 0, M_x = v_x$, nebo $P_x = -v_x, M_x = 0$ v závislosti na tom, zda je v_x kladné, nebo záporné.

Pokud se nejedná o list, má vrchol x svých m potomků p_0 až p_{m-1}, na ně zavoláme naši funkci a získáme hodnoty $M_{p_0}, \ldots, M_{p_{m-1}}$ a $P_{p_0}, \ldots, P_{p_{m-1}}$. Nyní uvažme, jak můžeme zvolit podstromy, abychom obsloužili všechny synovské vrcholy. Všechny tyto podstromy budou obsahovat vrchol x, kdežto syny můžeme vybírat zcela libovolně, na sobě nezávisle. Z toho vyplývá, že nemůžeme nic získat použitím operace přičtení či odečtení na daného potomka vícekrát, než kolik potomek vyžaduje.

Na druhou stranu každou operaci musíme provést alespoň tolikrát, kolikrát syn požaduje. Počet provedení každé operace tedy zvolíme jako maximum ze všech M_{p_0} až $M_{p_{m-1}}$ a P_{p_0} až $P_{p_{m-1}}$.

Jednoznačně jsme tedy pro daný vrchol zjistili nutný počet provedení operace přičtení a odečtení. Naši rekurzivní funkci zavoláme na vrchol 0 a jako výsledek vrátíme $M_0 + P_0$. Časová složitost bude stejná jako u obyčejného průchodu do hloubky, ta je $\mathcal{O}(M + N)$, kde N je počet vrcholů a M počet hran. Protože pro strom platí $M = N - 1$, je výsledná časová složitost našeho algoritmu $\mathcal{O}(N)$.

Lukáš Folwarczný

5 Průchod přes dvě patra

Úlohu vyřešíme dynamickým programováním.[3] Pro každý počet přechodů mezi patry (těch může být nejvíce L) a patro si budeme pamatovat, kam až se v tomto patře můžeme dostat na tolik přechodů. Na začátku se můžeme dostat na nula přechodů na začátky obou pater, jinak se na žádný počet přechodů nemůžeme dostat nikam.

Označme D_i hodnoty řešení pro prvních i klíčů. Nyní předpokládejme, že máme spočítané hodnoty D_{k-1}, z toho budeme chtít spočítat hodnoty D_k. To zvládneme jednoduše – všechny hodnoty řešení D_{k-1} zkusíme vylepšit pomocí k-tého klíče, vždy vyzkoušíme možnost kdy přecházíme i možnost kdy nepře-

[3] `http://ksp.mff.cuni.cz/viz/kucharky/dynamicke-programovani`

cházíme. Z výsledků si pak zapamatujeme vždy ten lepší (ten, který dojde dál).

Při programování, pokud možnosti budeme vylepšovat postupně od nejvyššího počtu přechodů, vše zvládneme s jedním polem o velikosti $2 \times L$. Časová složitost algoritmu je $\mathcal{O}(NL)$.

Karel Tesař

6 Obchod s grafy

Tuto úlohu jsme převzali z celostátního kola 58. ročníku Matematické olypiády kategorie P, kde se objevila jako úloha P-III-4 „Výlet do Švýcarska“. Vzorové řešení spolu se vzorovým kódem můžete najít na stránkách olympiády.[4]

Karel Tesař

[4] `http://mo.mff.cuni.cz/p/58/zadani-3b.html`