Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

37. ročník | **KSP-Z** | **Březen 2025**

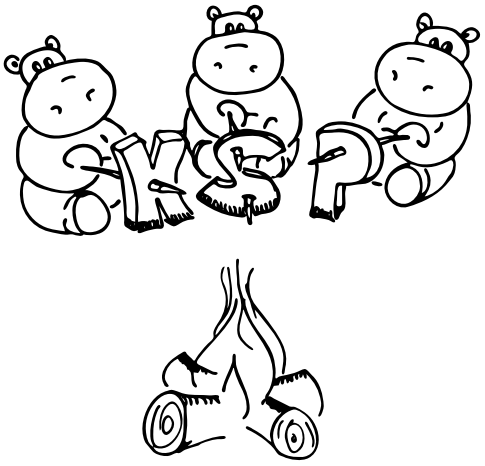
Řešení čtvrté série začátečnické kategorie 37. ročníku KSP

37-Z4-1 Kevinova dieta

Ze zadání víme, kde se čokoláda rozlámala, ale bohužel nám nikdo neslibuje, že budou zlomy setříděné. V prvním kroku si tedy musíme zlomy v každém směru zvlášť setřídit. Potom už dokážeme z rozdílu dvou sousedních zlomů odečíst, jakou šířku (pro horizontální zlomy výšku) budou mít dílky mezi zlomem i a $i+1$. To si zaznamenáme do pole v pro vertikální zlomy a h pro horizontální zlomy. Pozor na to, že musíme zvlášť přidat dílky před prvním zlomem a také po posledním, protože je nedostaneme jako rozdíl nějakých dvou sousedních zlomů.

Když si teď zvolíme dvojici prvků z v a h, určuje nám rozměry konkrétního dílku. Můžeme tedy přímočaře projít všechny dvojice, tím projdeme všechny dílky čokolády, a pro každý zkontrolovat, jestli je čtvercový. Pokud ano, aktualizujeme případně maximální nalezenou délku strany.

Jakou časovou složitost má tento algoritmus? Označme si počet zlomů jako v zadání úlohy V a H. Nejprve nám třídění zabere $O(V \log V) + O(H \log H)$, což můžeme také zapsat jako $O(K \log K)$, kde K je maximum z čísel V a H. Následně ovšem ještě přidáme při procházení všech dílků čas $O(V \cdot H)$. Celkem tedy dostaneme $O(K \log K + V \cdot H)$.



Nešlo by druhou část zrychlit, aby nám už nic nepřibylo k časové složitosti? Ale ovšem. Když si velikosti dílků v polích v a h setřídíme od největšího po nejmenší, můžeme zefektivnit procházení dílků pomocí metody dvou jezdců. Jak na to? Začneme obě pole procházet od největších hodnot, tedy od začátku. Pokud jsou hodnoty v obou polích stejné, máme výsledek a skončíme. V opačném případě se posuneme v tom poli, ve kterém je větší hodnota, o jeden prvek dál a znovu porovnáváme. Tento algoritmus opakujeme, dokud nenarazíme na stejné prvky. To se určitě jednou stane, protože zadání nám slibuje existenci nějakého čtverce. Že jako první narazíme na ten největší už zřejmě plyne z toho, že postupujeme od největších hodnot.

A jakou nakonec máme časovou a prostorovou složitost? Metodou dvou jezdců jsme prošli pole v a h jednou, máme tedy $O(V + H)$. Celkem dostaneme $O(V + H + K \log K)$, což je ale $O(K \log K)$. Paměti jsme spotřebovali $O(K)$ na uložení polí v, h a vstupu.

Na závěr zmiňme, že třídění na začátku algoritmu lze alternativně realizovat přihrádkovým tříděním. Takový přístup by se vyplatil, kdyby počet zlomů byl řádově rovný velikosti tabulky. Potom bychom dostali časovou složitost $O(R+S)$, kde je R počet řádků a S počet sloupců tabulky.

Úlohu připravili: Říša Hladík, Vojta Lančarič

37-Z4-2 Poker

Pro každého hráče známe pět karet, které tvoří jeho ruku. Naším úkolem je jednotlivé ruky seřadit podle síly. K tomu můžeme použít třídící algoritmus dostupný v našem zvoleném programovacím jazyce. Musíme však algoritmu říci, co znamená nejslabší a co nejsilnější ruka. Třídící funkce typicky podporuje jednu z následujících možností, jak toho docílit:

- Třídící funkci lze předat *porovnávací funkci*, která dostane dva prvky a řekne, jestli je první „menší“ než druhý.
- Třídící funkci lze předat *klíč*, což je funkce, která pro daný prvek vrátí nějakou hodnotu, podle které se bude třídit.

V našem případě bude jednodušší použít klíč. Pokud třídící funkce ve vašem zvoleném jazyce vyžaduje porovnávací funkci, můžete si ji snadno vyrobit porovnáním klíčů.

Každé výherní kombinaci přiřadíme číslo: high card dostane nejmenší číslo (třeba 1), straight flush největší (třeba 9). Kromě toho však musíme být schopní rozhodnout o pořadí dvou rukou stejné výherní kombinace. Naším klíčem tedy nebude jedno číslo, ale seznam několika čísel. Pokud jsou první čísla dvou klíčů shodná, porovnají se druhá čísla, atd. První číslo v seznamu bude číslo výherní kombinace a další čísla určíme podle pravidel pro porovnávání kombinací:

- Pro high card to budou hodnoty karet seřazené od nejvyšší,
- pro one pair bude první číslo hodnota páru, pak hodnoty zbývajících karet,
- pro postupku nám stačí hodnota nejvyšší karty,
- pro zbylé kombinace obdobně.

Ohodnocování se dá hezky naprogramovat tak, že si pro každou výherní kombinaci napíšeme jednu funkci, které předáme ruku jako argument. Pokud ruka odpovídá dané kombinaci, vrátí funkce seznam čísel, který bude klíčem pro třídění. Pokud ruka kombinaci neodpovídá, vrátí jinou hodnotu, například `None`. Poté můžeme procházet kombinace od nejsilnější po nejslabší a použít první funkci, která vrátí platnou hodnotu.

Úlohu připravili: Říša Hladík, David Klement, Dan Skýpala

37-Z4-3 Turistka na blátě

Podle zadání hledáme cestu z Udolí do Vrcholu hory. Naivně si můžeme pomyslet, že na to bude stačit BFS, které najde nejkratší cestu, ale je zde háček v podobě bláta. Bláto přidává další rozměr do našeho problému, protože musíme platit za každý krok do něj ponožkami. BFS-ko najde nejkratší cestu, ale pozor, ta nemusí být ta, která použije jen K ponožek. Zkusme se zamyslet, jak dokážeme tento problém řešit.

Prohledávání

Představme si, že budeme prohledávat horu tak, že nejdřív projdeme všechna políčka bez bláta, do kterých se umíme dostat, a když pak vstoupíme do bláta, proces opakujeme. Tohle nám vždy najde cestu do konce s nejmenším počtem použitých ponožek, protože vždy budeme hledat cestu, která nás dostane co nejdál s nejmenší spotřebou ponožek.

Proč najdeme správnou cestu? Protože pokud bychom šli do bláta dřív, tak bychom mohli překročit povolený počet ponožek. Můžeme se na tento algoritmus podívat jako na vlnu, která se šíří přes zdi (bláto) a nejdřív musí vyplnit prostor mezi nimi, a pak dokáže projít dál. Protože nehledáme nejkratší cestu, tak nás vůbec nezajímá, kdy se tam dostaneme, ale chceme použít co nejméně ponožek.

Úprava BFS

Jak si ale pamatovat cestu? Nejdřív si musíme uvědomit, co potřebujeme ukládat pro každou souřadnici. Potřebujeme si pamatovat, jak daleko jsme od startu a jestli je na ní bláto. Takže si můžeme naši mapu předělat na 2D pole, kde pro každý index si budeme pamatovat boolean, jestli je na něm bláto. Pak si vytvoříme druhé 2D pole, kde si zapamatujeme počet ponožek od startu. Tohle nám taky poslouží na zjištění, jestli jsme vrchol již navštívili. Teď se zaměříme na to, jak bude prohledávání vypadat.

Jak zaručíme, že nejdřív projdeme všechny vrcholy bez bláta a pak teprve ty s ním? Můžeme si vytvořit obousměrnou frontu, kde budeme dopředu ukládat vrcholy bez bláta a na její konec vrcholy s ním. Takže když budeme vybírat vrchol z fronty, tak budeme mít jistotu, že jsme už prošli všechny vrcholy bez bláta. Takhle prohledáme celou mapu a skončíme, když se dostaneme do cíle.

Složitost

Složitost tohoto algoritmu je $O(R \cdot S)$, kde R je počet řádků a S počet sloupců mapy. Prohledáváme každý vrchol jen jednou a pro každý vrchol si pamatujeme jen konstantní množství informací. Takže celková paměťová i časová složitost je $O(R \cdot S)$.

Úlohu připravili: Daniel Culliver, Vladimír Sklenár, Dan Skýpala

37-Z4-4 Velikonoční zajíčci

Představme si, že už víme, že zajíc a má rád čistá vajíčka. Potom všichni zajíci, se kterými se hádal, určitě musí mít rádi malovaná vajíčka. A všichni zajíci, se kterými se hádali oni mají rádi čistá. A tak dále.

Celou situaci si můžeme představit jako graf – zajíci tvoří vrcholy a hrany vedou mezi dvojicemi, kteří se pohádali. Pokud se grafovou terminologii moc neznáte, doporučujeme nahlédnout do kuchařky.[1]

Začněme tedy se zajícem a, který má rád čistá vajíčka a dejme ho do fronty. Poté opakujme:

- Vezměme prvního zajíce ve frontě a označme ho a.
- Pro všechny zajíce b (tedy sousední vrcholy), se kterými se a hádal:
  - Zaznačme si, že b má rád opačná vajíčka než a. (Pokud už máme zaznačeno, že má rád stejnou, zajíce do skupin rozdělit nejde.)
  - Přidejme do fronty ty z nich, u kterých jsme neměli nic zaznačeno.

Tento algoritmus je vlastně jen poupravené prohledávání do šířky (bfs), které používá dvě značky na navštívenost vrcholů.

A co s tím, že nevíme, kteří zajíci mají rádi která vajíčka? Mohli bychom vzít jednoho zajíce a zkusit obě možnosti. Ale všimněme si, že potom dostaneme pouze prohozené skupiny, takže stačí vyzkoušet jen jednu možnost. (Zajíc a bude mít opačná vajíčka, stejně tak jeho sousedi, jejich sousedi, atd.)

Bohužel ale prohledávání do šířky nemusí určit rozdělení všech zajíců, ale jen těch v aktuální komponentě. Proto budeme zajíce postupně procházet, a pokud aktuální zajíc nemá přiřazená vajíčka, prohlásíme, že má rád čistá a pustíme na něj bfs.

A jakou tohle má složitost? Prohledávání do šířky má samo o sobě složitost $\mathcal{O}(N + M)$, pro počet vrcholů a hran z aktuální komponenty. My ho pouštíme opakovaně, ale protože ho pouštíme jen jednou na každou komponentu, tak každý vrchol a hranu uvidí jen jednou. A tedy celková složitost je $\mathcal{O}(N + M)$ pro vrcholy a hrany z celého grafu. Procházení zajíců bude trvat $\mathcal{O}(N)$, což se schová do výše zmíněné složitosti.

Úlohu připravili: Adam Jahoda, Dan Skýpala

[1] `http://ksp.mff.cuni.cz/viz/kucharky/grafy`

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*