Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

37. ročník | **KSP-Z** | **Říjen 2024**

Řešení první série začátečnické kategorie 37. ročníku KSP

37-Z1-1 Počet slov

Nejdříve si rozebereme, co máme na vstupu:

- Malý nebo velký znak anglické abecedy, značící slovo.
- Jiný tisknutelný (na displeji viditelný) znak, který nikdy nemůže být ve slově.

Nyní k algoritmu. Budeme procházet znaky na vstupu postupně zleva doprava a pro každý budeme kontrolovat, zda se může nacházet ve slově. Nastat můžou tři situace. Pokud vidíme znak abecedy a do teď jsme slovo nečetli, tak si přičteme jedna k počtu přečtených slov a zapamatujeme, že jsme začali číst slovo. Dále můžeme vidět další znaky z abecedy, ale to už slovo čteme, takže nesmíme znovu připočítat dané slovo do počtu přečtených slov. No a nebo vidíme znak, co v abecedě není, a tedy si pouze zapamatujeme, že slovo aktuálně nečteme.

Nakonec vypíšeme počet přečtených slov.

Složitost algoritmu

Algoritmus má časovou složitost $\mathcal{O}(N)$, kde N je počet znaků na vstupu, jelikož každý znak navštívíme právě jednou. Paměťová složitost je konstantní, pamatujeme si pouze počet přečtených slov a informaci, zda čteme slovo, či nikoliv.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/37-Z1-1.py`

Úlohu připravili: Petr Budai, Kačka Doubková, David Klement

37-Z1-2 Státy

Ve chvíli, když už máme načtený vstup, musíme projít mapu tak, jak ji prošli Kevin se Sárou. Kvůli tomu si budeme udržovat jejich pozici a s každým znakem reprezentujícím pohyb jednoduše upravíme jejich pozici (například pro `>` zvýšíme sloupec, na kterém se nacházíme o 1).

V úloze ale nepotřebujeme směry, kudy Kevin se Sárou šli, ale státy, kudy šli. Budeme si proto po každém kroku ukládat, v jakém státě se nacházíme.

Jenže teď můžeme mít v seznamu navštívených států některé víckrát! Ještě musíme zjistit, kolik je na jejich cestě unikátních států. Toho můžeme docílit mnoha způsoby. Jedním způsobem může být udržování seznamu států, kde už jsme byli. Poté můžeme projít postupně všechny navštívené státy, kde máme dvě možnosti:

- Tento stát už máme v seznamu. V tomto státě už jsme byli. V tom případě nic neuděláme a pokračujeme dál.
- Tento stát ještě nemáme v seznamu. V tomto státě jsme ještě nebyli. Přidáme ho do seznamu navštívených států a pokračujeme dál.

Nakonec skončíme se seznamem unikátních států, které Kevin se Sárou navštívili. A kolik unikátních států tedy navštívili Kevin se Sárou? Stačí zjistit velikost našeho seznamu.

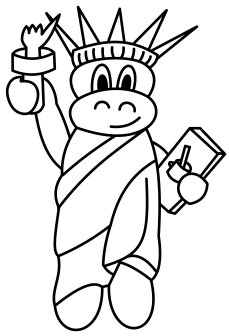
Složitost algoritmu

Teď si pojďme povědět něco o časové složitosti. První musíme načíst mapu – musíme načíst RS prvků do 2D pole. To samotné nám zabere $\mathcal{O}(RS)$ času.

Pro nás nejdůležitější bude poté délka procházky Kevina se Sárou I. Když procházíme, na které státy narazíme na jejich cestě, co uděláme v každém kroku? Zjistíme co je na našem místě na mapě – to dokážeme v konstantním čase. Poté upravíme svou pozici – to také dokážeme v konstantním čase. Sestavit seznam všech navštívených států dokážeme tedy v čase $\mathcal{O}(I)$.

Poté pro každý z navštívených států zjišťujeme, jestli je unikátní. Toho docílíme druhým seznamem – seznamem unikátních navštívených států. Přidávat do seznamu dokážeme v konstantním čase. Co nám ale může dělat problém je zjišťovat, jestli už stát v seznamu máme – proto místo seznamu použijeme množinu, která to dokáže v konstantním čase. V pythonu na toto lze použít `set()`. Sestavit si ze seznamu navštívených států množinu unikátních navštívených států nakonec zvládneme v průměru v čase $\mathcal{O}(I)$.

Nakonec zjišťujeme velikost množiny unikátních států, což zvládneme v konstantním čase. Celková časová složitost našeho programu bude tedy v průměru $\mathcal{O}(RS + I + I)$ – neboli $\mathcal{O}(RS + I)$.



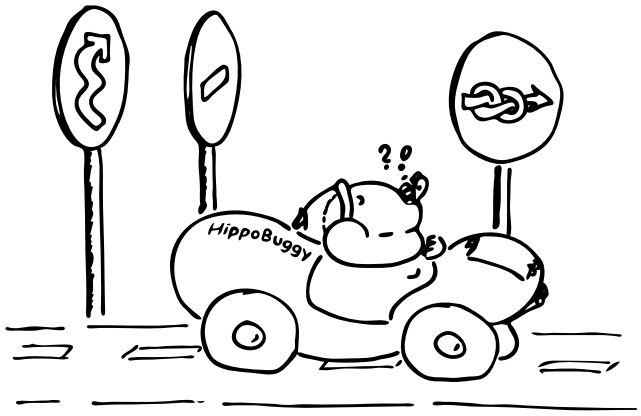
A co prostorová složitost? Kolik paměti potřebujeme? První samozřejmě potřebujeme mapu států – $\mathcal{O}(RS)$. Potom si utvoříme seznam navštívených států – $\mathcal{O}(I)$. Nakonec si utvoříme seznam unikátních navštívených států, kterých bude nejvýše $\mathcal{O}(RS)$, protože se musí vejít do mapy. Můžeme si všimnout, že pokud by například Kevin se Sárou navštívili po cestě celou mapu dvěstěkrát, tak pro nás bude I velké. Můžeme ušetřit prostor tím, že když tvoříme seznam navštívených polí, tak státy rovnou buď přidáváme do seznamu unikátních navštívených států nebo necháváme být. Tím si ušetříme ukládání původního seznamu navštívených států o velikosti I. Prostorová složitost našeho algoritmu bude $\mathcal{O}(RS)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/37-Z1-2.py`

Úlohu připravili: David Klement, Vojta Lančarič, Matúš Púll

37-Z1-3 Čekání na tramvaj

Nejdříve si pojďme vyjasnit, co potřebujeme v úloze zjistit. Hledáme nejdelší interval, kdy nejede žádná linka. Nezajímá nás jaká linka – tím si zjednodušíme práci a nebudeme se vůbec zajímat o čísla linek.



Jak ale najdeme takový nejdelší interval? Dostali jsme časy odjezdů jednotlivých linek, ale mezi časy jedné linky může přijet nějaká jiná. Proto si musíme seřadit všechny časy odjezdů najednou, když už víme, že nás příslušnost k jednotlivým linkám nezajímá.

V řešení si uložíme všechny časy odjezdů do jednoho pole a setřídíme. Poté se podíváme na každý jednotlivý odjezd. Jelikož je máme seřazené, v poli další odjezd v pořadí pojede hned po aktuálním. Odečtením aktuálního času odjezdu od následujícího zjistíme, jak dlouho bychom na další odjezd museli čekat, což přesně hledáme. Potřebujeme si tedy pamatovat vždy největší délku čekání na další tramvaj a čas, kdy toto čekání začíná. Vždy, když najdeme delší úsek, zapamatujeme si ten. Tím na konci zůstaneme s úsekem s nejdelším čekáním a jeho začátkem.

Zbývá ale ještě vyřešit okrajový případ, kde nejdéle musíme čekat do dalšího dne. To dokážeme elegantně obejít tím, že po seřazení všech časů odjezdu přidáme ještě jeden – ten bude stejný jako první odjezd v daný den, ale o den později. Proto stačí přidat do seznamu první prvek zvýšený o $24 \cdot 60 \cdot 60$. Úlohu poté řešíme úplně stejně.

Složitost

První načteme všechny odjezdy do jednoho pole. Pro přehlednost si s dovolením určíme parametr T – celkový počet všech tramvajových odjezdů. Načíst je nám zabere $\mathcal{O}(T)$ času.

Poté musíme odjezdy setřídit podle velikosti. Použijeme libovolný třídící algoritmus s časovou složitostí $\mathcal{O}(T \log T)$. Například pythoní `list` má metodu `sort`.

Nakonec projdeme celé pole, odečteme od sebe každé dva sousedící časy a zapamatujeme si nejdelší čekací úsek. To dokážeme v čase $\mathcal{O}(T)$.

Celková časová složitost algoritmu bude $\mathcal{O}(T \log T)$.

Co se týče prostorové složitosti, tak si pamatujeme hlavně pole všech odjezdů o velikosti $\mathcal{O}(T)$. To nám dává celkovou prostorovou složitost $\mathcal{O}(T)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/37-Z1-3.py`

Úlohu připravili: David Klement, Kuba Pelc, Matúš Púll

37-Z1-4 Hloubka vrcholů ve stromu

Základní princip řešení

Když se snažíme zjistit hloubku vrcholu, tak jelikož máme pro každý vrchol pouze ukazatel na otce, zbývá nám akorát podívat se na něj. Poté máme dvě možnosti:

- Vrchol otce není kořen. V tomto případě si budeme muset pamatovat, že jsme šli přes další vrchol a tento proces aplikujeme znovu na otce tohoto vrcholu.
- Vrchol otce je kořen. V tom případě do původního vrcholu můžeme zapsat, jak daleko je od otce – počet vrcholů, přes které jsme šli.

Takhle prohledáme všechny vrcholy mezi původním a kořenem, zjistíme kolik je mezi nimi hran a zapíšeme toto číslo do původního vrcholu.

Pojďme se podívat, jak dlouho nám toto řešení může trvat v závislosti na počtu vrcholů N. Co kdybychom dostali vstup, který je ušitý přímo proti našemu algoritmu a bude to pouze jeden dlouhý sloupec vrcholů? Potom bychom se z nejnižšího vrcholu museli dostat až ke kořeni – provedeme N kroků. Následně hledáme hloubku vrcholu o 1 výš – prohledáme všechny vrcholy mezi ním a kořenem, tudíž provedeme $N - 1$ kroků. Takhle budeme pokračovat až ke kořeni, pokaždé o 1 méně kroků než předtím. Když to sečteme, vyjde nám $N(N + 1)/2$ kroků, což dává časovou složitost $\mathcal{O}(N^2)$. A co si budeme muset v průběhu algoritmu pamatovat? Pouze jedno číslo – vzdálenost původního vrcholu od toho, na který se právě díváme tedy máme konstantní časovou složitost – $\mathcal{O}(1)$.

Vylepšený princip řešení

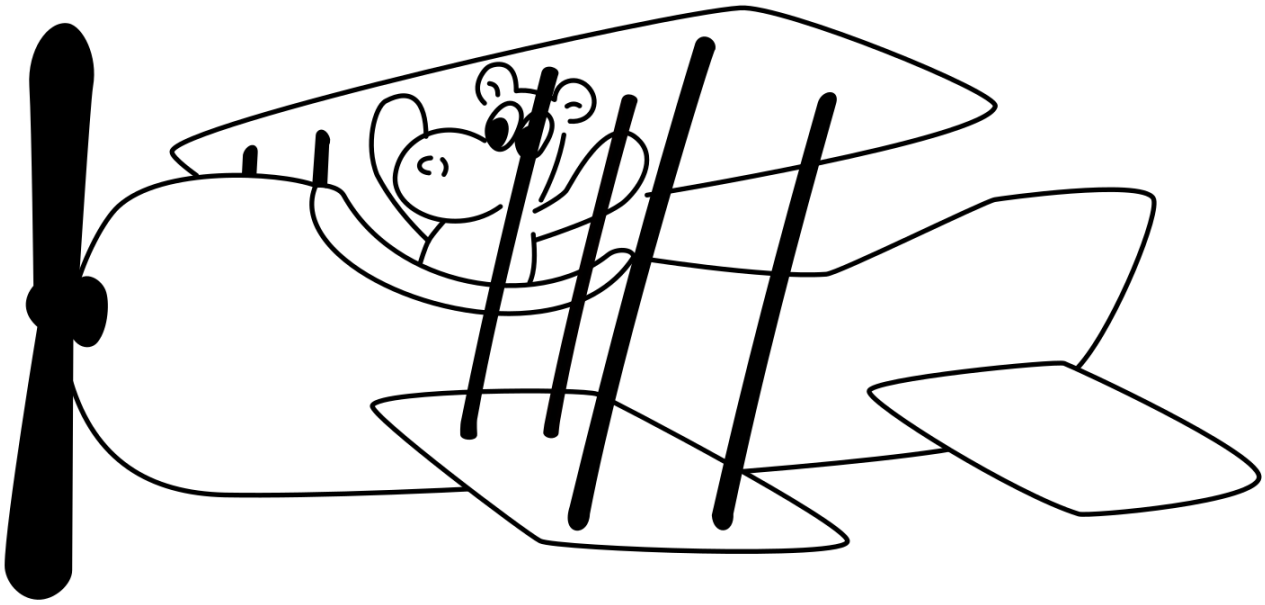
Nyní si ale všimneme, že se s dosavadním algoritmem občas koukáme do vrcholu, přes který už jsme šli. Vylepšíme náš algoritmus tak, abychom při průchodu vrcholem rovnou zjistili, jak je hluboko. Náš nový postup při navštívení vrcholu bude vypadat takto:

- Vrchol má známou hloubku – zapíšeme do předchozího vrcholu hloubku o 1 vyšší.
- Vrchol nemá známou hloubku – pamatujeme si, že jsme přes něj šli a tento proces aplikujeme na otce tohoto vrcholu.

Teď stačí pouze vyřešit, jak poté zapisovat hloubky do vrcholů, přes které jsme se do vrcholu známé hloubky dostali. Toho můžeme dosáhnout například přes rekurzi. Všimněte si, že stačí zavolat jednu rekurzivní funkci na vrchol, pro který zrovna hledáme hloubku a postupně tuto funkci volat na otce vrcholu, dokud se nedostaneme do vrcholu, kde už hloubku známe. Nakonec stačí vrátit jeho hloubku a jak budeme backtrackovat, vždy zapíšeme do vrcholů hloubku o 1 vyšší, než vrácenou a pošleme ji dál.

A teď si rozmysleme, jak bude vypadat složitost nového algoritmu. Kolikrát se podívám na každý vrchol? Každý vrchol navštívíme jednou, když bychom hledali jeho hloubku. Také vždy když pro každý vrchol hledáme hloubku, dojdeme nakonec na jeden vrchol, kde už hloubku známe (což může být i původní vrchol, pro který hloubku hledáme, ale to nám v ničem nevadí). Tím každý vrchol navštívíme jednou a k tomu se podíváme na jeden vrchol navíc, což nám dává časovou složitost $\mathcal{O}(N + N)$, neboli $\mathcal{O}(N)$. A co paměťová složitost? Když bychom dostali vstup, kde by strom byl pouze jeden souvislý řetízek, tak by se mohlo stát, že si musíme pamatovat celý strom, jak jsme ho prohledali, což nám dává paměťovou složitost $\mathcal{O}(N)$.

Úlohu připravili: Kačka Doubková, Matúš Púll



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/*