Korespondenční Seminář z Programování

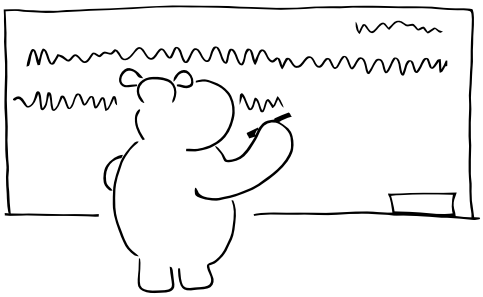
ZAČÁTEČNICKÁ KATEGORIE

36. ročník | **KSP-Z** | **Říjen 2023**

Řešení první série začátečnické kategorie 36. ročníku KSP

36-Z1-1 Nejčastější příjmení

Máme za úkol ze seznamu jmen vybrat takové, které se nejčastěji opakuje. V prvním přístupu si jména lexikograficky setřídíme. Poté už najdeme nejčastější jméno snadno – procházíme seznam a postupně aktualizujeme nejčastěji zastoupené jméno. Časová složitost tohoto řešení závisí nejen na počtu jmen N, ale také na tom, jak dlouho nám zabere porovnat dvě jména. V případě, že nejdelší jméno je dlouhé D znaků, bude porovnání trvat $\mathcal{O}(D)$. Projít seznam zabere $O(DN)$, časové nejnáročnější však bude třídění. To nám bude trvat $\mathcal{O}(DN \log(N))$, což je i výsledná časová složitost.



A nešlo by to zrychlit? Ale jistě. Pořídíme si hešovací tabulku. Pokud ještě hešovací tabulky neznáte, můžete si o nich přečíst v naší kuchařce.[1] Pro každé unikátní jméno si udržujeme záznam v tabulce s počtem jeho výskytů. To tedy konkrétně znamená, že procházíme seznam jmen. Pokud se ještě jméno v tabulce nenachází, inicializujeme jeho záznam na jedničku. V případě, že jméno už v tabulce je, zvýšíme jeho počítadlo o jedna. Na konci projdeme všechny záznamy v tabulce a vybereme z nich maximum. Přidat N jmen o maximální délce D do hešovací tabulky nám zabere v průměru $\mathcal{O}(DN)$. Maximální hodnotu si můžeme aktualizovat průběžně, nebo na závěr ještě celou tabulku projít. V obou postupech však dostaneme průměrnou časovou složitost $\mathcal{O}(DN)$.

Nakonec ještě stojí za zmínku úvaha, zda by nebylo výhodnější si jména, která mohou být potenciálně velmi dlouhá, převést na nějaký úspornější zápis, se kterým by se lépe pracovalo. Tato myšlenka nicméně ztroskotá na tom, že abychom přečetli N až D znakových slov, potřebujeme tak jako tak $O(DN)$ času.

Úlohu připravili: Jirka Kvapil, Vojta Lančarič, Ján Plachý, Jirka Sejkora, Ondra Sladký

36-Z1-2 Buldozer s krabicí

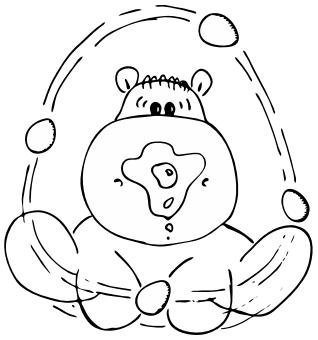
Nejpřímočařejší řešení by bylo si celou mřížku vygenerovat a potom jen průběh simulovat: umístíme bagr a krabici do mřížky, poté v každém kroku pohneme s bagrem a zkontrolujeme, zda na novém políčku neleží krabice. Pokud by ležela, posuneme ji v mřížce ve stejném směru jako bagr.

Náš úvodní krok – zkonstruovat si mřížku – nebude pro větší souřadnice fungovat. Jednak konstrukce potrvá příliš dlouho a jednak se mřížka ani nevleze do paměti. Musíme na to chytřeji, ve skutečnosti totiž úplně stačí, když si v paměti budeme držet pouze souřadnice bagru a krabice. Při změně souřadnice bagru se podíváme, zda se neshoduje se souřadnicí krabice, a případně i tu posuneme stejným směrem. Tato operace zabere čas $\mathcal{O}(1)$ a provedeme ji N-krát. Celkově tedy program poběží v čase $\mathcal{O}(N)$.

Úlohu připravili: Jirka Kalvoda, Jirka Kvapil, Vojta Lančarič, Jirka Sejkora, Ondra Sladký

36-Z1-3 Vejce

Poměrně přímočaré a jednoduché řešení je pozorovat každou slepici a zapisovat si, kdy se vylíhne a kdy snese vejce. Stačilo by zjistit, kdy se vylíhne, a poté zjistit, kolikrát stihne snést vejce, než nastane den D. Musíme si pamatovat přesně, v jaký čas snese jaké vejce, abychom věděli, z jakého času začít, až budeme pozorovat život tohoto vejce a později slepice z něj. Toto řešení nám však velmi rychle spadne na hlavu, když si uvědomíme, jak rychle bude stoupat počet slepic (jak už napovídá v zadání horní hranice počtu slepic).



Co kdybychom si místo každé slepice samostatně drželi časovou osu, na které bychom zapisovali všechny události, co se budou dít? Osu si můžeme vyjádřit pomocí pole, kde každý prvek tohoto pole bude představovat jeden den. Pro jednoduchost si budeme držet jedno pole pro líhnutí nových slepic a jedno pole pro snášení nových vajec. Den bude v poli reprezentován číslem, které určuje, kolik slepic se má ten den vylíhnout, resp. kolik vajíček se má snést. Zároveň si budeme držet po celou dobu proměnnou, která bude určovat aktuální počet vylíhnutých slepic.

Na začátku nastavíme v poli nových slepic na pozici K číslo 1, které označuje vylíhnutí naší první slepice. Jakmile se K-tého dne vylíhne naše první slepice, přičteme ji k celkovému počtu slepic a do pole nových vajec na pozici $K + P$ zapíšeme 1, což označuje snesení vejce naší první vylíhnuté slepice. Pak už jen postupujeme den po dni v obou polích

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

stejně. Přičteme počet vylíhnutých slepic v i-tý den k celkovému počtu slepic, den v poli nových slepic na pozici $i+K$ nastavíme na počet nových vajec v i-tý den a nakonec den v poli nových vajec na pozici $i+P$ nastavíme na počet nových vajec v i-tý den + počet nových slepic v i-tý den. Jakmile takto odsimulujeme D dní, program ukončíme a vypíšeme aktuální počet slepic.

Jaká je časová a paměťová složitost takového algoritmu? Můžeme si rozmyslet, že velikost pole bude $D+K+P$, neboť kdybychom nechali pouze D, tak se může stát, že ke konci algoritmu se pokusí program zapsat mimo rozsah pole. Takové pole budeme potřebovat dvě. Jedno pro líhnutí slepic a jedno pro snášení vajíček. projdeme je každé jednou, kdy v každém kroku udělám konstantní počet operací. Celková paměťová složitost je tedy $\mathcal{O}(D+K+P)$ a časová je obdobně $\mathcal{O}(D+K+P)$.

Úlohu připravili: Jirka Kalvoda, Honza Kotovský, Jirka Sejkora, Ondra Sladký

36-Z1-4 Čaj

Lehčí varianta

V lehčí variantě mají všechny pytlíky čaje stejnou hmotnost. Díky tomu známe maximální počet čajů k, který dokážeme unést, a to $\lfloor M/H_i \rfloor$ (část pytlíku nést nemůžeme). Hodnoty zajímavostí jsou nezáporné, takže k je délka hledané posloupnosti čajů – není nikdy důvod brát kratší.

V řešení si pořídíme okénko velikosti p, kterým budeme jezdit po posloupnosti zajímavostí čajů. Pořídíme si tři proměnné, index levého pytlíku ℓ, index pravého pytlíku p a součet zajímavostí v aktuálním okénku z. Proměnnou ℓ na začátku nastavíme na nejlevější čaj, tedy index 0, p nastavíme na $k-1$ (aby bylo okénko dlouhé k) a z spočítáme sečtením zajímavostí prvních k čajů.

V každém kroku algoritmu posuneme okénko doprava o jeden čaj – k oběma okrajům ℓ, p přičteme 1, z zvětšíme o Z_p a zmenšíme o $Z_{\ell-1}$. Tím jsme získali součet zajímavostí čajů na nové pozici okénka, aniž bychom museli všechny zajímavosti sčítat znovu! V průběhu si budeme ukládat součet a pozici nejzajímavějšího úseku, který jsme zatím viděli, a ten budeme při každém posunutí porovnávat s aktuálním. Když narazíme pravým okrajem na konec provázku, tak vypíšeme pozici nejzajímavějšího úseku z paměti.

Těžší varianta

Oproti lehčí variantě můžou mít pytlíky různé hmotnosti, tím pádem i délka hledané posloupnosti se může lišit. Využijeme stejný postup, jako v lehčí variantě, ale okénko nebude mít nutně stejnou velikost. Budeme ho rozšiřovat doprava a zmenšovat zleva podle toho, kolik váží všechny aktuálně vybrané pytlíky.

Na to si pořídíme čtvrtou proměnnou h, značící aktuální hmotnost čajů, která bude na začátku 0, stejně tak i z, ℓ a p. Krok algoritmu bude vypadat následovně:

- Když $h \leq M$, tedy čaje uneseme, porovnáme zatím nejzajímavější interval s aktuálním. Je-li aktuální zajímavější, tak jej přepíšeme. Pak zvětšíme okénko zprava – zvýšíme p o 1, k z přičteme Z_p a h zvětšíme o H_p.
- Jinak zmenšíme okénko zleva – od z odečteme Z_ℓ, zmenšíme h o H_ℓ a poté k ℓ přičteme 1.

Toto budeme opakovat, dokud nenarazíme pravým okrajem na konec provázku.

Je algoritmus funkční? Rozhodně se zastaví, protože v každém kroku se posune jeden z ukazatelů doprava a zároveň levý se nikdy neposune dál než pravý. Časem tedy musíme pravým narazit na okraj. Při běhu algoritmu uvidíme všechny intervaly, které mají maximální možnou váhu (takovou, že rozšíření o jeden čaj jakýmkoliv směrem by překročilo M) a hledaný nejzajímavější interval je jeden z nich. Tím pádem ho potkáme a zapamatujeme si ho.

Jak je to s časovou složitostí? Při hledání intervalu v každém kroku posuneme ℓ nebo p o jedna dál, čajů je N, každý může být nejvýše jednou označen jako ℓ a nejvýše jednou jako p. Tedy nejpozději po $2N$ krocích dojdeme na konec a ukončíme prohledávání. Celý algoritmus má tedy časovou složitost $\mathcal{O}(N)$.

Úlohu připravili: Janek Hlavatý, Ján Plachý

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/*