Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

36. ročník | **KSP-Z** | **Červen 2024**

Řešení páté série začátečnické kategorie 36. ročníku KSP

36-Z5-1 Alergie

V našem algoritmu budeme chtít pro každou zadanou lokalitu spočítat, kolik Kevinových kamarádů do ní může jet na výlet. Čili budeme chtít pro každého kamaráda jednoduše rozhodnout, jestli je alergický na některý z alergenů dané lokality.

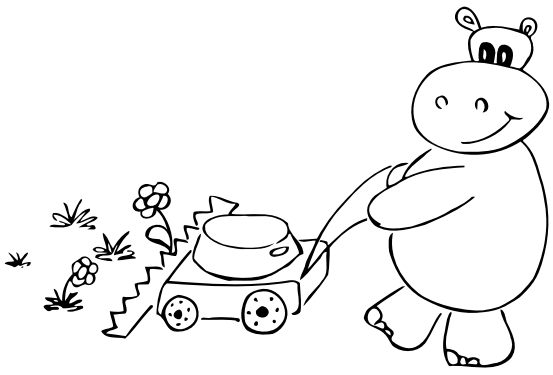
Pro každého kamaráda si tak vytvoříme seznam jeho alergií a pro každou lokalitu seznam jejích alergenů. Následně budeme chtít pro každou lokalitu zjistit počet kamarádů, kteří do ní mohou jet na výlet. To znamená, že pro každou lokalitu budeme muset projít seznam všech kamarádů a pro každého z nich zjistit, zda je některá z jeho alergií obsažena v seznamu alergenů dané lokality. Pakliže žádná taková alergie neexistuje, připočteme jej k počtu kamarádů, kteří mohou jet do dané lokace na výlet.

Jak toto ale zjistíme? Mohli bychom pro každý kamarádův alergen projít celý seznam alergenů lokality, to by ovšem bylo dost pomalé.

Pokud by byly oba seznamy alergenů setřízené, zvládneme to rychleji: Pořídíme si ukazatele i a j, které ukazují na začátky seznamů A a B. Pokud jsou prvky $A[i]$ a $B[j]$ stejné, pak je náš kamarád alergický na některý alergen lokality a můžeme s hledáním skončit. Jinak pokud $A[i] > B[j]$, pak zvýšíme j o 1, jinak zvýšíme i o 1. Pokud bychom někdy měli jeden z ukazatelů posunout mimo rozsah jeho pole, náš kamarád není alergický na žádný alergen lokality.

Jakmile už známe pro každou lokalitu počet Kevinových kamarádů, kteří ji mohou navštívit, stačí vybrat tu, u které je tento uložený počet největší.

Jak je na tom náš algoritmus s časovou složitostí? Pro každou z L lokací musíme projít N kamarádů a pro každého z nich projít nejvýše celý jeho seznam alergií A a seznam alergenů G lokace, které si navíc musíme předem seřadit. Podrobnější rozepsání časové složitosti by vedlo na poměrně nehezký vzorec s mnoha parametry. Uvedeme tak jen, že časová složitost celého algoritmu je kvadratická vůči délce vstupu.



Úlohu připravili: Kačka Doubková, Janek Hlavatý, Andy Mikulová

36-Z5-2 Pletení pomlázky

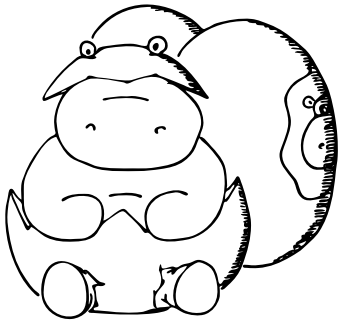
Nejjednoduší řešení je prostě pletení pomlázky odsimulovat. Načteme si popis kroku pletení do pole P, N-krát nastavíme X na $P[X]$ a nakonec vypíšeme výsledek. To ale není optimální. Paměťová složitost je sice $\mathcal{O}(K)$, časová je však $\mathcal{O}(K + N)$.

V každém kroku pletení je proutek na jedné z K pozic. Pokud je N větší než K, musí se nějaká pozice dříve či později zopakovat. Jelikož se plete pořád stejně, proutek pak bude dokola procházet ten samý cyklus pozic. Zároveň platí, že první opakující se pozice musí být ta počáteční. Kdyby to byla jiná, znamenalo by to, že se na ni dorazilo z dvou různých pozic, což není možné.

Stačí tedy simulovat pletení jenom než se proutek vrátí na původní pozici, čímž dostaneme jeden cyklus. Pokud cyklus má délku C, tak pozice po N krocích bude stejná jako v kroku N modulo C cyklu. Takto bude i časová složitost $\mathcal{O}(K)$, protože každá z K pozic se v cyklu nachází maximálně jednou.

Úlohu připravili: Jan Adámek, Vašek Končický

36-Z5-3 Toruslice



Na každom políčku sa môže nachádzať nová oblasť, preto určite musíme prezrieť všetky políčka, aby sme si mohli byť istí výsledkom. Mapu toruslice si uložíme do dvojrozmerného poľa. Na načítanie aj prehľadanie použijeme dva vnorené for cykly, jeden cez každú súradnicu.

Keď pri prechádzaní nájdeme stenu (políčko s bielou farbou), môžeme ju ignorovať. Keď nájdeme voľné políčko, zaujíma nás len vtedy, ak sme ešte nevideli žiadne políčko z jeho oblasti. Aby sme to vedeli rýchlo určiť, budeme si okrem samotnej mapy pamätať aj ďalšie dvojrozmerné pole boolovských hodnôt, ktoré nám vravia, či už sme dané políčko navštívili.

Alebo môžeme tieto dve polia spojiť a uvedomiť si, že keď voľné políčko označíme po navštívení ako stenu, významovo sa nič nezmení – už ho nikdy nechceme navštíviť ani započítavať.

Keď teda nájdeme voľné políčko, poznačíme si, že sme našli novú oblasť a spustíme z neho prehľadávanie do hĺbky – DFS (dalo by sa použiť aj prehľadávanie do šírky – BFS).

Do zásobníka (stack) si budeme ukladať políčka, ktoré ešte chceme navštíviť. Potom vždy jedno z nich vyberieme, označíme ako navštívené a do zásobníka uložíme jeho susedov. Ak sme políčko vybrané zo zásobníka už navštívili (medzitým, čo ležalo v zásobníku), ignorujeme ho. Ak políčko susedí s okrajom mapy, musíme súradnice jeho susedov vypočítať ako zvyšky po delení výškou alebo šírkou celej plochy.

Keď takto prehľadáme a označíme celú oblasť, môžeme pokračovať v prechádzaní celej plochy a už žiadne políčko z danej oblasti nezarátame znova.

Časová aj pamäťová zložitosť sú $\mathcal{O}(SV)$, potrebujeme si zapamätať a prejsť celé dvojrozmerné pole.

Úlohu připravili: Vašek Končický, Ján „Jancí“ Plachý

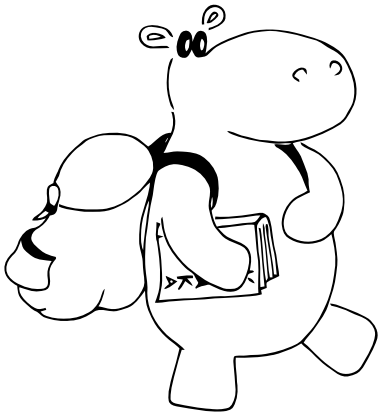
36-Z5-4 Koledování

Nejprve si spočteme největší vzdálenost, kterou zvládne Kevin do poledne urazit. To zvládneme jednoduše tak, že vynásobíme zbylý čas jeho rychlostí.

Proveďme jedno pozorování – každá cesta může vypadat tak, že Kevin prvně vyrazí nalevo, dojde do nějakého domu, tam se otočí a vrátí se domů. Pak udělá to samé směrem vpravo. Proč toto platí? Protože když už se vydá nalevo, tak nedává smysl nedojít až na konec všech levých domů, které chce projít. Jinak by se tam musel vracet, čímž by si cestu určitě prodloužil. Také se určitě otočí na pozici nějaké domu, jinak by cestu od posledního domu k místu své otočky mohl vynechat a prošel by stejně domů, čímž by si cestu zkrátil. A nakonec může určitě prvně vyrazit vlevo a pak vpravo. Kdyby se rozhodl jít nejprve vpravo, tak to můžeme prohodit a nechat ho prvně vyrazit k levým domům, tím se nic nezmění.

Nyní tedy víme, že Kevin bude chtít projít nějaký souvislý úsek domů, který začíná i končí v nějakém domě, obsahuje jeho dům a jehož délka je nejvýše polovina maximální vzdálenosti, kterou stihne urazit. To už vede na jednoduchý kvadratický algoritmus. Vyzkoušíme všechny možné začátky takových úseků a všechny možné konce. Délku každého zjistíme jednoduše odečtením souřadnic, počet domů v něm zase odečtením indexů. Budeme si průběžně pamatovat ten z dostatečně krátkých úseků, který má nejvíce domů, a na konci ho vrátíme. Úseků je $\mathcal{O}(N^2)$, zpracovat jeden trvá $\mathcal{O}(1)$, celková časová složitost je tedy $\mathcal{O}(N^2)$. Toto funguje, ale je to moc pomalé. My budeme mířit na složitost lineární.

Budeme postupovat následovně. Nejprve si zjistíme nejlevější dům, do kterého by Kevin stihl dojít. Prostě vyrazíme z jeho domu a budeme pokračovat doleva, dokud nebude úsek mezi dalším domem a Kevinovým domem větší než polovina největší vzdálenosti. Vyrobíme si dva jezdce, levého a pravého. Levého jezdce umístíme do toho domu, který jsme právě našli, a pravého do Kevinova domu. Nyní budeme opakovat vždy následující kroky. Pokud můžeme úsek o jeden dům protáhnout doprava a úsek stále zůstane dostatečně krátký, tak posuneme pravého jezdce doprava a úsek protáhneme. Jinak posuneme levého jezdce doprava, čímž úsek zkrátíme. Ve chvíli, kdy z úseku vyjede Kevinův dům, tedy když se levý jezdec přesune napravo od něj, tak algoritmus ukončíme. Celou dobu si budeme vedle pamatovat nejlepší úsek, který jsme zatím našli, a na konci jej vypíšeme.



Je vidět, že tento algoritmus běží v lineárním čase. V každém kroku posuneme jeden z ukazatelů doprava, posunutí umíme v $\mathcal{O}(1)$ a posunout každý z nich můžeme nejvýše N-krát. Vůbec ale není zjevné, že algoritmus vrátí správný výsledek. Proto to nyní dokážeme.

Nejprve si všimněme, že v každý moment algoritmu je aktuální úsek dostatečně krátký, aby ho Kevin stihl projít. Také si všimněme, že kdykoliv, když posouváme levého jezdce, tak pravý jezdec je nejpravěji co to jen jde, aby byl úsek validní. Samotný důkaz provedeme nyní. Nechť $a-b$ je úsek s nejvíce domy, který je dost krátký, aby ho Kevin zvládl projít, a zároveň obsahuje Kevinův dům. Protože $a-b$ obsahuje Kevinův dům, tak levý jezdec se někdy musel ocitnout v a a někdy ho musel opustit. Uvažujme moment, kdy ho opouští. Už víme, že v tu chvíli je pravý jezdec nejvíc napravo, co to jen jde. Nyní ukážeme, že to musí být v b. Kdyby byl nalevo od b, tak úsek můžeme ještě prodloužit doprava, což znamená, že neposouváme levého jezdce. Kdyby byl napravo od b, tak úsek od a k němu nemůže být dostatečně krátký, neboť je delší než $a-b$, což je nejlepší validní úsek a tedy i nejdelší, co může být. My víme, že v každý moment našeho algoritmu je zpracovávaný úsek validní, proto napravo od b se dostat nemůžeme. A protože není ani nalevo, ani napravo od b, tak musí být v něm. Úsek $a-b$ jsme tedy během našeho algoritmu museli najít a na konci ho vypíšeme.

Algoritmus běží v čase $\mathcal{O}(N)$ a pamět nám stačí konstatní, pokud do ní nepočítáme pole souřadnic. Vždy si stačí pamatovat jen pozice jezdců, délku zpracovávaného úseku a pozice jezdců v okamžik zatím nejlepšího úseku.

Úlohu připravili: Kačka Doubková, Adam Jahoda

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/*