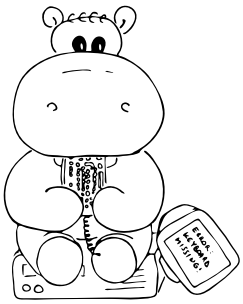
Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

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

Řešení třetí série začátečnické kategorie 36. ročníku KSP

36-Z3-1 Kebab



Na vstupu jsme dostali pro každou restauraci dvojici údajů – vzdálenost od Kevinova domu a cenu, za kterou je zde kebab k dostání. Pro každou restauraci tak snadno spočítáme celkové výdaje – vzdálenost krát cena jízdy za kilometr plus cena kebabu. Zároveň si průběžně udržujeme nejnižší celkovou cenu za kebab, kterou pak na konci vypíšeme.

Celkově jsme tedy úlohu zvládli vyřešit při jednom průchodu vstupem a to znamená lineární časovou složitost $\mathcal{O}(N)$ s počtem restaurací.

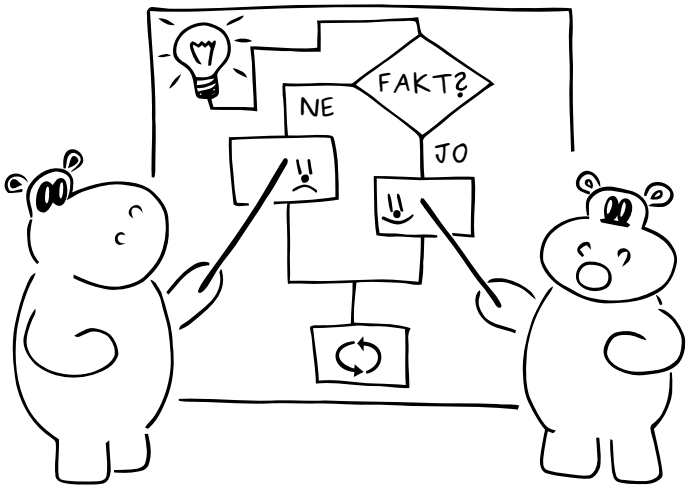
Program (Python 3):
`http://ksp.mff.cuni.cz/viz/36-Z3-1.py`

Úlohu připravili: Petr Budai, Adam Jahoda, Vojta Lančarič

36-Z3-2 Přednášky

Při rozmýšlení řešení bývá užitečné uvážit nejprve postup *hrubou silou*, tedy vyzkoušet všechny přípustné možnosti. To vede nejen na relativně snadnou implementaci, ale především na triviální důkaz správnosti – nemohli jsme vynechat optimální možnost, když jsme vyzkoušeli všechny.

Časová složitost ale může trpět a ouvej, tady máme dokonce nekonečně mnoho možností, kdy se na přejezd vydat. Ani omezení na celočíselné časy odjezdu nepomůže dostatečně. Stačí dostatečně dlouhá pauza na oběd a algoritmus nám nedoběhne ani do konce semestru.



Pojďme naši *hrubou sílu* trochu zjemnit. Máme-li mezi dvěma povinnými přednáškami v různých budovách 50 minut volného času, máme celkem 11 celočíselných možností, kdy na přejezd vyrazit. Kterou zvolíme, je ale úplně jedno. Což takhle předepsat si nějaké pravidlo, podle kterého jednu vybereme? Pak sice nevyzkoušíme všechny možnosti, ale stále jich bude dost na to, aby mezi nimi byla i některá z optimálních.

Nejsnazším pravidlem je odjet hned po konci nějaké přednášky. Tím určitě nepřijdeme o nic zajímavého – pokud jsme se během čekání nudili, je jedno ve které budově. Pro každou volitelnou přednášku v mezeře mezi povinnými přednáškami v různých budovách tak spočítáme, kolik volitelných přednášek je možné stihnout, pokud bychom se vydali na cestu po jejím konci. Musíme si také dát pozor, zda po konci této volitelné přednášky stále ještě stihneme povinnou přednášku v další budově. V případě, že bychom povinnou přednášku nestíhali, nemůžeme tento čas odjezdu brát v potaz.

Jak algoritmus vypadá detailněji? Skáčeme po mezerách mezi sousedními povinnými přednáškami (ty jistě stihneme všechny). Pokud jsou ve stejné budově, zapíšeme si všechny volitelné v mezidobí (přednášky se nepřekrývají). Pokud jsou v různých budovách, zkusíme naplánovat přejezd po konci libovolné volitelné a té dřívější z povinných. Pro každý spočítáme, kolik přednášek se kvůli tomu nestihne. Nesmíme samozřejmě zvolit takový čas odjezdu, že pozdější povinnou bychom nestihli. Z časů odjezdu vybereme ten, který nás bude nejméně mrzet a k celkovému počtu absolvovaných přednášek přičteme volitelné v mezidobí bez těch v kolizi s přejezdem. Ještě se sluší poznamenat, že volitelné přednášky před první povinnou stihneme všechny, stačí ráno začít na správné budově.

Spotřeba paměti je jistě v $\mathcal{O}(P + V)$. Jak je to s časem? Po každé přednášce plánujeme nejvýše jeden odjezd. Musíme spočítat, kolik přednášek koliduje s naším přejezdem. Protože se ale přednášky nepřekrývají, může jich kolidovat nejvýše 40, tudíž to stihneme v konstantním čase. I časová složitost tedy leží v $\mathcal{O}(P + V)$.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/36-Z3-2.py`

Úlohu připravili: Honza Černohorský, Janek Hlavatý, Vojta Káně, Honza Kotovský

36-Z3-3 Peřeje

Řeka je obdélníková mřížka $W \times H$. Na každém políčku mřížky se buďto nachází, nebo nenachází hroch. Naším cílem je najít cestu skrz tuto mřížku vedoucí pouze přes políčka bez hrochů.

Mřížku budeme procházet do hloubky. Zadání dovoluje začít s loďkou na kterékoliv pozici v prvním řádku, prohledávat tak začneme ze všech těchto možných startovních pozic.

Označme pozici loďky $[r, s]$ jako r – řádek a s – sloupec, na kterém se loďka zrovna nachází. V každém kroku se r zvýší o jedna – proud nás nese dolů – a s buďto může zůstat stejné, nebo se může změnit o 1, nebo -1.

V každém kroku algoritmu se podíváme na políčka, na která by mohla loďka doplout v příštím kroku, a vybereme ta, na kterých nejsou hroši. Pokračovat v prohledávání mřížky budeme odtud. Mřížku prohledáváme, dokud nedoplujeme na konec řeky.

Pokud jsme skončili s prohledáváním, ale nedošli jsme na konec řeky, žádná bezpečná cesta řekou neexistuje a vypíšeme `NEEXISTUJE`.

Nakonec musíme vypsat celou cestu, kterou loďka proplula. Musíme si tak během prohledávání v každém bodě pamatovat, odkud jsme do něj přišli. Na konci pak stačí jen projít tyto záznamy a vypsat je v opačném pořadí.

Na každé políčko vstoupíme nejvýše jednou, takže časová složitost je $\mathcal{O}(W \cdot H)$. Paměti potřebujeme stejně.

Program (Python 3):
`http://ksp.mff.cuni.cz/viz/36-Z3-3.py`

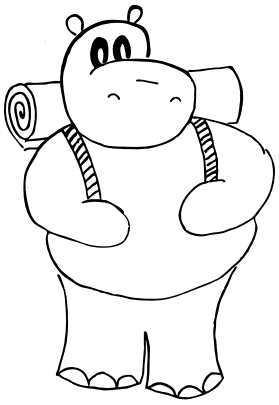
Úlohu připravili: Kačka Doubková, Janek Hlavatý, Prokop Randáček, Dan Skýpala

36-Z3-4 Výlet

O hoře víme, že nejdříve stoupá, pak následuje hledaný vrchol a poté klesá. Použijeme binární vyhledávání. Když si stáhneme výšku bodu b_i, tak bohužel nezjistíme, jestli je vrcholem hory, nebo není. Můžeme si ale stáhnout i výšku bodů b_{i-1} a b_{i+1} okolo něj. Pokud jejich výška bude menší, než výška b_i, tak jsme úspěšně nalezli vrchol a můžeme skončit. Pokud je posloupnost rostoucí, tak víme že se b_i nachází nalevo od hledaného vrcholu. Pokud je klesající tak napravo.

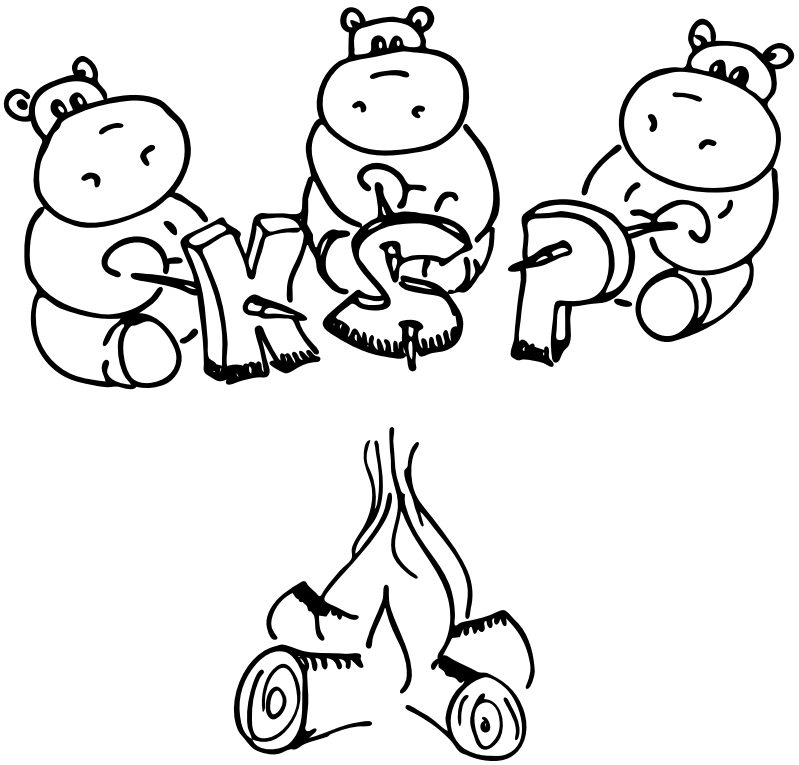
Pořídíme si tedy dva ukazatele $l = 0$ a $p = n - 1$, kde n je šířka hory. Ty nám budou ukazovat na aktuální prohledávaný úsek. Spočítáme prostřední bod $i = \frac{p+n}{2}$ a stáhneme si trojici vrcholů b_{i-1}, b_i a b_{i+1}. Porovnáme je a binární vyhledávání ukončíme, nebo posuneme odpovídající ukazatel a spustíme vyhledávání znovu.

Takto vrchol určitě najdeme, protože při posouvání ukazatelů zahazujeme jen ty body, ve kterých se vrchol hory nenachází.



Časová složitost je logaritmická, protože v každém vyhledávání zmenšíme prohledávaný úsek na polovinu. Prostorová je konstantní, protože si potřebujeme pamatovat pouze několik ukazatelů na okraje a mezivýpočty.

Úlohu připravili: Kačka Doubková, Janek Hlavatý



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/*