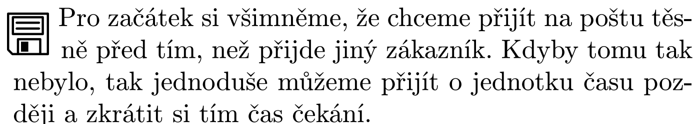
Korespondenční Seminář z Programování

36. ročník | KSP | Prosinec 2023

Vzorová řešení druhé série třicátého šestého ročníku KSP

36-2-1 Fronty na poště



Pro začátek si všimněme, že chceme přijít na poštu těsně před tím, než přijde jiný zákazník. Kdyby tomu tak nebylo, tak jednoduše můžeme přijít o jednotku času později a zkrátit si tím čas čekání.

Co se stane, když přijdeme na poštu před zákazníkem i, oproti tomu, kdybychom na poštu nepřišli? Úředník ve frontě 0 místo původního zákazníka pošle do fronty nás. Zákazník za námi potom skončí ve stejné frontě jako původně zákazník za ním a tak dál. To, co se původně dělo zákazníkovi j (pro $j \geq i$), se nyní děje zákazníkovi $j+1$, a my nahrazujeme zákazníka i.

A tedy všichni před zákazníkem i zůstávají nezměněni. Pozor, toto funguje jenom díky tomu, že zákazníci nebudou nikdy posláni zpět do fronty 0. Např. pro:

```
1 7
3 N
4 N
5 N
6 P
0 0
7 V
0
8 V
0
9 V
0
```

Pokud bychom přišli před prvním zákazníkem, tak by se nás vůbec nedostala řada.

To znamená, že když přijdeme před zákazníkem i, tak potom odejdeme v čase, kdy by býval odešel zákazník i, pokud bychom vůbec nepřišli.

Stačí si tedy odsimulovat celý proces na poště bez nás. Po jednotlivých frontách požadujeme, aby se dalo rychle přidat na konec a odebrat ze začátku, takže na to můžeme *(překvapivě)* použít frontu. Jednotlivé druhy událostí budeme obsluhovat následovně:

- Když přijde nový zákazník, tak ho dáme na konec fronty 0 a poznačíme si, v jakém čase přišel.
- Když je zákazník přesunut, tak ho z fronty j přemístíme do fronty k.
- A když zákazník odejde, tak ho z fronty odebereme a poznačíme si, v jakém čase odešel.

Na konci si spočítáme pro každého zákazníka s vyřízenou žádostí, kolik času na poště strávil, a přijdeme hned před ním. Na poště potom strávíme o tuto jednotku více času.

Simulaci i projití všech zákazníků stihneme v $\mathcal{O}(N)$ (kde N je počet událostí). Paměťová složitost bude také $\mathcal{O}(N)$.

Program (Python):
`http://ksp.mff.cuni.cz/viz/36-2-1.py`

Úlohu připravili: Michal Kodad, Vojta Lančarič, Dan Skýpala

36-2-2 Záchrana mravenců

Vyplatí se zachraňovat jen ty mravence, na které něco může dopadnout. Dále když na mravence dopadne jedna šiška, tak je jednodušší ho zachránit, než když na něj dopadnou dvě. Tedy budeme nejprve chtít zachraňovat ty mravence, na které dopadne nejméně šišek.

Jak to spočítat? Naivně můžeme pro každého mravence projít všechny šišky a podívat se, jestli na něj dopadnou. To je ale hodně pomalé, až $\mathcal{O}(N^2 S)$.

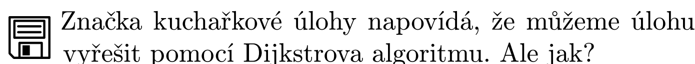
Můžeme to vylepšit tak, že použijeme (inverzní) dvourozměrné prefixové součty, ty sestavíme tak, aby nám říkaly, kolik šišek dopadne na každé políčko. V levém horním rohu šišky přičteme 1, v pravém horním rohu šišky 1 zase odečteme (protože tam šiška končí). Taktéž v levém dolním rohu odečteme 1. Jenže takto bychom jedničku pro políčka pod a za šiškou odečetli dvakrát, takže v pravém dolním rohu zase 1 přičteme.

Takto nejprve do prefixových součtů započítáme všechny šišky a posléze z nich vypočítáme, kolik šišek dopadne na každé políčko. Celkem to stihneme v čase $O(N^2 + S)$.

Nyní, když známe, kolik šišek dopadne na každé políčko, umíme si najít všechna políčka, na která dopadne určitý počet šišek (přihrádkovým řazením). Nejprve budeme krabičky přiřazovat všem mravencům, na které dopadne jedna šiška, pak všem, na které dopadnou dvě a tak dále. Těchto přiřazovacích kroků může být nejvýše K. Také si můžeme všimnout, že přiřazovacích kroků nemůže být víc než N^2, protože to bychom už zcela jistě zachránili všechny mravence.

Takže celkově nám algoritmus běží v čase $O(N^2 + S)$.

36-2-3 Ztracená bunda



Značka kuchařkové úlohy napovídá, že můžeme úlohu vyřešit pomocí Dijkstrova algoritmu. Ale jak?

Dijkstrův algoritmus se často představuje v kontextu plánovačů cest. Vrcholy pak představují křižovatky a body zájmu a hrany cesty mezi nimi. Dijkstrův algoritmus však můžeme využít i k řešení obecnějších problémů, které s hledáním nejkratší cesty na první pohled nijak nesouvisí. Stačí, abychom měli nějakou konečnou množinu stavů a různě drahých způsobů, jak mezi nimi přecházet. Poté můžeme pomocí Dijkstrova algoritmu najít způsob, jak se co nejlevněji dostat z jednoho stavu do druhého.

Prvním krokem tedy bude reprezentovat jízdní řády pomocí grafu.

Stav cestujícího se skládá ze dvou hodnot, a to zastávky a času (0:00 – 23:59). Vrcholy tedy budou reprezentovat právě dvojice zastávek a časů. Nemusíme však vytvářet vrcholy pro všechny možné dvojice, stačí ty, kde a kdy se děje něco zajímavého, neboli příjezd nebo odjezd vlaku. Také přidáme vrchol pro čas a místo začátku cesty.

Dále přidáme hranu pro každou akci, kterou může cestující provést. Akce jsou dvou druhů:

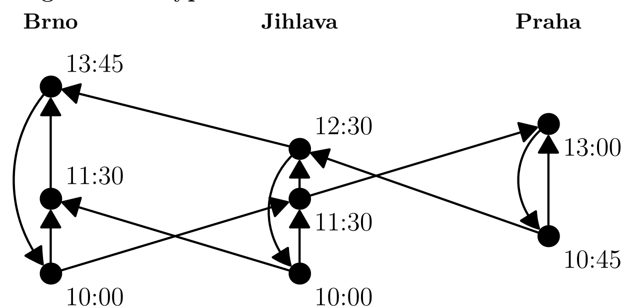
- jízda vlakem na další zastávku,
- čekání do dalšího zajímavého času.

Pro každý úsek každého vlaku tedy přidáme hranu vedoucí ze zastávky a času odjezdu do zastávky a času příjezdu. Při cestě vlakem není třeba mrznout na nástupišti, proto bude mít tato hrana nulovou cenu. Dále vrcholy v rámci jedné zastávky pospojujeme do kruhu podle jejich časů. Každé z těchto hran přiřadíme cenu odpovídající rozdílu časů, které spojuje.

Uvažujme například tento jízdní řád:

R547: Brno 10:00, Jihlava 11:30, Praha 13:00
EC243: Jihlava 10:00, Brno 11:30
Ex75: Praha 10:45, Jihlava 12:30, Brno 13:45

Jeho graf bude vypadat takto:



Pokud se o vlaky zajímáte, tak vám možná tento graf připomíná grafikon.[1]

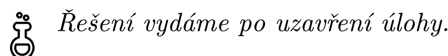
Ke každé „vlakové“ hraně si také poznamenáme, jaký je její vlak a cílová stanice, abychom později mohli snadno nalezenou cestu vypsat.

Poté stačí použít Dijkstrův algoritmus, abychom našli nejkratší cestu z počátečního vrcholu do všech ostatních vrcholů. Jirka může dorazit domů v libovolný čas, takže poté projdeme všechny vrcholy v cílové stanici a vybereme z nich ten s nejnižší vzdáleností. Dijkstrův algoritmus nám také ke každému vrcholu určí seznam hran, po kterých se do něj dostaneme co nejkratší cestou. Projdeme tedy nejkratší cestu do zvoleného cílového vrcholu a vypíšeme informace o všech vlakových hranách, které potkáme.

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

Úlohu připravili: Jirka Kalvoda, Dan Skýpala, Ben Swart

36-2-4 Nejlepší programovací jazyk, část II.



Řešení vydáme po uzavření úlohy.

Úlohu připravili: Filip Hejsek, Standa Lukeš, Kuba Pelc, Jirka Sejkora, Dan Skýpala

[1] `https://cs.wikipedia.org/wiki/Grafikon_dopravy`

matfyz

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/*