Semestrální práce z KIV/PT

Alternativní zadání semestrální práce pro KIV/PT

Kateřina Kuchynková (A22B0025P), Michael Hladký (A22B02092P)

17.11.2023

Obsah

1. Zadání

Vytvořme pro Beznoha simulační program, který mu pomůže naplánovat přepravu, známe-li:

• počet skladů S,

• každý sklad bude definován pomocí:

– kartézských souřadnic každého skladu xs a ys,

– počtu pytlů ks, které jsou do skladu vždy po uplynutí doby ts doplněny. Na začátku simulace předpokládejte, že došlo k doplnění skladů, tj. ve skladu je ks pytlů uhlí,

– doby tn, která udává, jak dlouho trvá daný typ pytle na kolečko naložit/vyložit (každý sklad může používat jiný typ pytlů, se kterým může být různě obtížná manipulace),

• počet zákazníků Z,

• kartézské souřadnice každého zákazníka xz a yz ,

• počet přímých cest v mapě C,

• seznam cest, přičemž každá cesta je definovaná indexy i a j, označujících místa (zákazník, sklad) v mapě, mezi kterými existuje přímé propojení a platí: $i, j \in \{1, \ldots, S, S+1, \ldots, S+Z\}$, tj. sklady jsou na indexech od 1 do S a zákazníci na indexech od S + 1 do S + Z. Pozn. pokud existuje propojení z místa i do místa j, pak existuje i propojení z místa j do místa i,

• počet druhů koleček D,

• informace o každém druhu kolečka, kterými jsou:

– slovní označení druhu kolečka, které bude uvedeno jako jeden řetězec neobsahující bílé znaky,

– minimální v_{min} a maximální v_{max} rychlost, kterou se může daný druh kolečka pohybovat, přičemž kolečko se pohybuje konstantní rychlostí, která mu bude vygenerovaná v daném rozmezí pomocí rovnoměrného rozdělení,

– minimální dmin a maximální dmax vzdálenost, kterou může daný druh kolečka překonat, dokud nebude potřebovat provést údržbu, přičemž pro každý druh kolečka je tato doba opět konstantním a je vygenerovaná v daném rozmezí pomocí normálního rozdělení se střední hodnotou $\mu = (dmin + dmax)/2$ a směrodatnou odchylkou $\sigma = (dmax - dmin)/4$,

– doba td, udávající kolik času dané kolečko potřebuje pro provedení ´údržby,

– počet pytlů kd udávající maximální zatížení daného druhu kolečka,

– hodnota pd udávající procentuální zastoupení daného druhu kolečka ve vozovém parku firmy, přičemž platí: PD d=1 pd = 100%,

• počet požadavků k obsloužení P,

• každý požadavek bude popsán pomocí:

– času příchodu požadavku tz (pozn. požadavek přichází doopravdy až v čase tz , tzn. nemůže se stát, že by jeho obsluha začala dříve, a že by v době příchodu požadavku byl náklad již na cestě),

– indexu zákazníka $zp \in \{1, \ldots, Z\}$ kterému má být požadavek doručen,

– množství pytlů kp, které zákazník požaduje,

– doby tp udávající, za jak dlouho po příchodu požadavku musí být pytle doručeny (tj. nejpozději v čase tz + tp musí být pytle vyloženy u zákazníka).

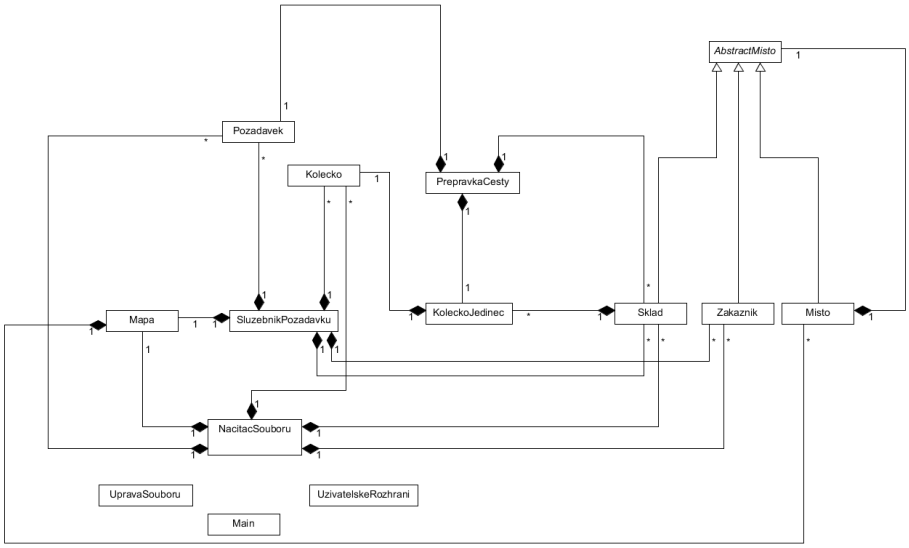
Za úspěšně ukončenou simulace se považuje moment, kdy jsou všechny požadavky obsloužené a všechna kolečka jsou vrácena do svých domovských skladů. V případě, že se některý požadavek nepodaří (z jakéhokoli důvodu) obsloužit včas, pak simulace skončila neúspěchem, o čemž bude program informovat příslušným výpisem (viz níže).

2. Analýza problému

Problém, který má být řešen, lze analyzovat jako problém optimalizace distribuce zásob, přičemž cílem je minimalizovat dobu potřebnou k doručení všech požadavků zákazníků a zároveň minimalizovat nakupování koleček ve skladech firem.

Jako nejdůležitější považujeme výběr vhodné datové struktury pro uložení dat a němu poté vhodný algoritmus na hledání nejkratších cest. Jednou z možností, jak toto řešit by bylo použít matici sousednosti. A k ní poté Floyd-Warshallův algoritmus na prohledání míst, které spolu přímo nesousedí. Slouží k nalezení nejkratších cest mezi všemi dvojicemi uzlů v grafu. Může pracovat s hranami s libovolnými hodnotami, včetně záporných vah. Je vhodný pro grafy s mnoha hranami nebo s hranami s libovolnými hodnotami. Další možnost je možnost využít Graf jako spojový seznam a k němu **Dijkstrův** algoritmus. Je určen pro hledání nejkratších cest z jednoho uzlu do všech ostatních uzlů v grafu. Funguje pouze na grafech s nezápornými váhami hran. Je efektivní pro malé grafy nebo grafy s málo hranami. Původně jsme chtěli pracovat s maticí sousednosti, ale poté jsme si uvědomili, že na velká data by vedla k přetečení haldy. Nakonec jsme se rozhodli využít datovou strukturu graf a Dijkstrův algoritmus.

3. Návrh programu



Třída *main* volá třídu *UpravaSouboru*, která upraví vstupní soubor, abychom je následně mohli načíst. Také volá *NacitacSouboru*, který načte data pro další zpracování. Dále se tu volá *UzivatelskeRozhrani*,

které se stará o to, že si uživatel může načíst soubor dle jeho volby, přidat požadavek anebo jeden určitý odebrat.

Třídy *Sklad*, *Zakaznik*, *Pozadavek, Kolecko* reprezentují jednotlivá data, která jsou potřebná pro spuštění aplikace. *Kolecko* představuje jednotlivé typy koleček a *KoleckoJedinec* představuje už jedno reálné kolečko, které můžeme použít pro převážení pytlů. *Sklad* a *Zakaznik* dědí od Abstraktní třídy *AbstractMisto*. Stejně tak třída Místo, která nám umožňuje sklady a zákazníka dávat dohromady například v jednotlivých polí.

Další důležitou třídou je Mapa, která si vytváří graf s informacemi, kdo s kým sousedí. Také je tu implementována metoda Dijkstrův algoritmus, který najde nejkratší cestu z jednoho určitého místo do ostatních míst.

SluzebnikPozadavku je hlavní metoda, která celý algoritmus ovládá. Má informace o všech dostupných skladech, o požadavcích a zákaznících. Stará se o to, aby našla nejlepší cestu pro kolečko, aby dovezlo pytle v požadovaném čase a také se stará o přímo jeho cestu. Spravuje nakupování či půjčování koleček, o obnovení skladu. Také se tu pracuje s třídou *PrepravkaCesty*, která obsahuje data potřebná pro kolečko a jeho cestu. *SluzebnikPozadavku* se volá v metodě *NacitacSouboru*, až jsou všechna data načtená.

4. Uživatelská příručka

Pro spuštění aplikace je třeba mít spustitelný soubor aplikace a data, jež si přejeme obsloužit. Data musí být ve formátu, který si můžete prohlédnout na obrázku níže. Spuštění větších souborů chvíli trvá, tak prosíme o strpení.

```
❄ Pocet skladu S: ↖
1
❄ Definice skladu (souradnice x,y, pocet pytlu ks, doba doplneni ts a doba nalozeni
tn) ↖
10 10 3 1000 1
❄ Pocet zakazniku Z: ↖
3
❄ Definice zakazniku (souradnice x,y) ↖
0 0
0 10
0 20
❄ Pocet cest C: ↖
4
❄ Definice cest (z jedineho skladu ke kazdemu zakaznikovi) ↖
1 2
1 3
1 4
1 5
❄ Pocet druhu kolecek D: ↖
2
❄ Definice kolecka (nazev, minimalni rychlost, maximalni rychlost, minimalni
vzdalenost, maximalni vzdalenost, doba opravy, maximalni zatizeni a procentualni
pomer zastoupení druhu kolecka)  ↖
Pomalé 1 1 1000 1000 0 1 0.5
Staré 10 10 1 1 10 1 0.5
❄ Pocet pozadavku P: ↖
4
❄ Definice pozadavku (kazdy zakaznik chce jeden pytel, moc na to nespecha) ↖
0 1 1 10000
0 2 1 10000
0 3 1 10000
0 4 1 10000
```

Obrázek 1

Při spuštění je uživatel dotázán na název souboru.

```
Zadejte nazev souboru: tutorial
Soubor neexistuje zadejte soubor znova
Zadejte nazev souboru: tutorial.txt
Data uspesne nactena
```

Obrázek 2

Po jeho správném zadání (při špatném, se ptá dokud není soubor zadaný správně) se zeptá co si přeje dělat dál. Zda chce přidat požadavek, odebrat nebo program spustit. Toto menu můžete vidět na obrázku 3.

```
Jestli si prejete pridat pozadavek, zadejte P
Jestli si prejete odebrat pozadavek, zadejte O
Jestli si prejete zacit simulaci, zadejte S
Zadejte prikaz:
```

Obrázek 3

Po zadání *P* je uživatel dotázán na všechna data, která musí požadavek obsahovat. Jako první je příchod požadavku (musí být kladné číslo, index zákazníka, který požadavek zadal (musí být v rozsahu již zadaných zákazníků), kolik pytlů si zákazník přeje a dokdy má být požadavek splněn.

```
Zadejte prikaz: P
Zadejte cas prichodu pozadavku: 0
Zadejte zakaznika co zadal pozadavek: 1
Zadejte pocet pytlu: 2
Zadejte deadline pozadavku: 1000
```

Obrázek 4

Pokud uživatel zadá *O* si přeje nějaký požadavek odebrat. Poté je dotázán na jeho pořadí a tento požadavek je následně odstraněn. Jako poslední možnost je *S* pro spuštění simulace. Poté se dostaneme k samotnému obsluhování požadavků. Než je ale daný požadavek obsloužen, tak je uživatel pokaždé dotázán, zda si přeje aktuální požadavek odebrat či nikoli a tím ho nechat obsloužit. Toto si můžete prohlédnout na obrázku 5.

```
Cas: 0, Pozadavek: 0, Zakaznik: 1, Pocet pytlu: 1, Deadline: 20
Prejete si pozadavek odebrat? (ano/ne)
```

Obrázek 5

Tímto uživatel nechá obsloužit jen ty požadavky, které si přeje. Simulace se po vyčerpání požadavků ukončí.

5. Závěr

Zadání semestrální práce se nám nepovedlo splnit v požadovaném rozsahu. Program úspěšně obslouží všechny zákazníky, které je možné obsloužit. Což znamená, že k nim vede cesta, existuje kolečko, která je schopné tuto cestu zvládnout a dá se stihnout obsloužit v požadovaném termínu. Avšak pokud jsou data zadávaná s velkým rozsahem cest, tak to náš algoritmus nestihne do požadovaného času 20 min.

Povedlo se nám vytvořit uživatelské prostředí, které dovoluje uživateli, vybrat si, který soubor chce uživatel zpracovat, umožní mu z něj odebrat anebo přidat požadavky. Také v průběhu má možnost aktuální požadavek odebrat anebo nechat obsloužit.

Další bodem zadání bylo, že program se dá v průběhu simulace pozastavit, nicméně tento bod se nám nepodařilo splnit. Vygenerování statistiky a generátor vlastních dat jsme bohužel také nezapracovali do této semestrální práce.

Javadoc si můžete prohlédnout v přiloženém souboru.

Naopak velmi pozitivní je, že program se nám podařilo vytvářet bez velkých kritických chyb a v průběhu programování se nám podařilo odladit program k daleko rychlejším výsledkům, než jsme dostávali na začátku.

Na semestrální práci jsme se podíleli oba rovnoměrným dílem. Kuchynková vymyslela, jak se data budou ukládat, jak bude probíhat Dijkstrův algoritmus a poté několik metod na kontrolování, jakou cestou se může kolečko/a vydat či poslání těchto jednotlivých koleček. Hladký naopak vymyslel, jak se jednotlivé požadavky budou obsluhovat, jak budou vypadat metody vypadat a staral se o celou vizuální stránku dokumentační komentáře.