SEMESTRÁLNÍ PRÁCE – PT

2. PROSINCE 2023

Václav Prokop (prokopv@students.zcu.cz, A22B0330P)
Filip Valtr (valtrf@students.zcu.cz, A22B0107P)

Obsah

Obsah

Zadání

Zásobovací společnost Sto roku v šachtě žil s. r. o., se specializuje na přepravu uhlí ze svých skladů do různých domácností a firem. Její majitel Petr Beznoha je ale velký skrblík, a tak nejraději využívá jako přepravní prostředky zahradní kolečka, která nejsou náročná na údržbu a provoz jako například Avie, které používá konkurence. Přepravované uhlí je uskladněno v speciálních pytlích, které jsou nakládány na kolečka, aby se zaměstnanci neumazali. Beznoha používá jak kolečka s železnou korbou, s pozinkovanou korbou, také kolečka s korbou plastovou a různé další alternativy. V poslední době se však panu Beznohovi moc nedaří a, částečně i vlivem toho, že po cestě mu spousta kovových koleček orezla vlivem deště, ho přeprava stojí spoustu peněz navíc. Rozhodl se tedy, že je čas dát prostor moderním technologiím, a proto si chce nechat vytvořit program, který mu pomůže rozplánovat přepravu uhlí k zákazníkům tak, aby nepřišel o nějaká další kolečka, dodržel závazky a neztratil klientelu, maximálně využil nosnosti koleček a zároveň kolečka zbytečně neopotřeboval zbytečně dlouhými cestami. Tyto požadavky můžeme jednoduše označit jako snahu o minimalizaci "ceny" přepravy.

Formální popis 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 pytle, 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 definována 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 vmin a maximální vmax rychlost, kterou se může daný druh kolečka pohybovat, přičemž kolečko se pohybuje konstantní rychlostí, která mu bude vygenerována 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 konstantní a je vygenerována v daném rozmezí pomocí normálního rozdělení se střední hodnotou $\mu = (dmin + dmax) / 2$ a směrodajnou 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: Σd=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 ∈ {1, . . . , 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 simulaci 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). V rámci výpisu použijte jednu z následujících variant (formát je závazný, indexace od jedné):

- Příchod požadavku:

Cas: <t>, Pozadavek: <p>, Zakaznik: <z>, Pocet pytlu: <p>, Deadline: <t+t_p>

- Ve skladu se začíná připravovat kolečko:

Cas: <t>, Kolecko: <k>, Sklad: <s>, Nalozeno pytlu: <p>, Odjezd v: <t+k*t_n>

- Zaměstnanec firmy dovezl kolečko k zákazníkovi, kde bude něco vykládat (pozn. výpis nikde v průběhu nebude odřádkován, časovou rezervou tr je myšlen rozdíl mezi časem, kdy má být náklad nejpozději vyložen a časem, kdy k vyložení opravdu došlo):

Cas: <t>, Kolecko: <k>, Zakaznik: <z>, Vylozeno pytlu: <p>, Vylozeno v: <t+k*t_n>, Casova rezerva: <t_r>

- Kolečko dojelo do skladu, kde ale na další cestu vyžaduje údržbu (nelze servisovat na cestě ani u zákazníka):

Cas: <t>, Kolecko: <k>, Zakaznik: <z>, <druh> kolecko vyzaduje udrzbu, Pokracovani mozne v: <t+t_d>

- Kolečko projelo okolo zákazníka, ale nemá zde žádný zvláštní úkol:

Cas: <t>, Kolecko: <k>, Zakaznik: <z>, Kuk na <druh> kolecko

- Kolečko dokončilo cestu a vrátilo se do skladu:

Cas: <t>, Kolecko: <k>, Navrat do skladu: <s>

Požadavek se nepodařilo (z jakéhokoli důvodu) obsloužit včas:

Cas: <t>, Zakaznik <z> umrzl zimou, protoze jezdit s koleckem je hloupost, konec

Výstup Vašeho programu bude do standardního výstupu a bude vypadat například následovně:

...
Cas: 12, Pozadavek: 2, Zakaznik: 2, Pocet pytlu: 3, Deadline: 30
Cas: 12, Kolecko: 5, Sklad: 3, Nalozeno pytlu: 3, Odjezd v: 18
Cas: 21, Kolecko: 5, Zakaznik: 1, Pozinkovane kolecko vyzaduje udrzbu, Pokracovani mozne v: 22
Cas: 23, Kolecko: 5, Zakaznik: 10, Kuk na pozinkovane kolecko
Cas: 24, Kolecko: 5, Zakaznik: 2, Vylozeno pytlu: 3, Vylozeno v: 30, Casova rezerva: 0
...

Čas ve výpise bude zaokrouhlen dle pravidel zaokrouhlování na celé číslo.

Analýza problému

Nejprve bylo zapotřebí napsat program, který dokáže jednotlivé soubory zpracovat tak, abychom z něj dostali potřebné údaje, a přitom odstraňuje nepotřebné komentáře a bílé znaky.

Program jsme se rozhodli zpracovat za pomocí grafu, jež hrany budou neorientované, jelikož kolečko se po cestě vedoucí do cíle může i vracet zpátky, odkud vyrazil. Dále bylo potřeba rozlišit, zdali ze vstupních dat bude vytvořen řídký, nebo hustý graf, což se odvíjí od počtu vrcholů ku počtu hran, pokud byl počet vrcholů mnohem vyšší, nežli počet hran, rozhodli jsme se využít matici sousednosti řešící husté grafy, pokud ale počet vrcholů a hran byl obdobný, využili jsme výhod seznamu sousednosti, a to hlavně toho, že není tak paměťově náročný, protože neobsahuje prázdné prvky na rozdíl od matice.

Složitost paměti při implementaci:

Seznamem sousednosti:	Maticí sousednosti:
O (|V|+|E|)	**Ω (|V|^2)**

… kde V je počet vrcholů a E počet hran

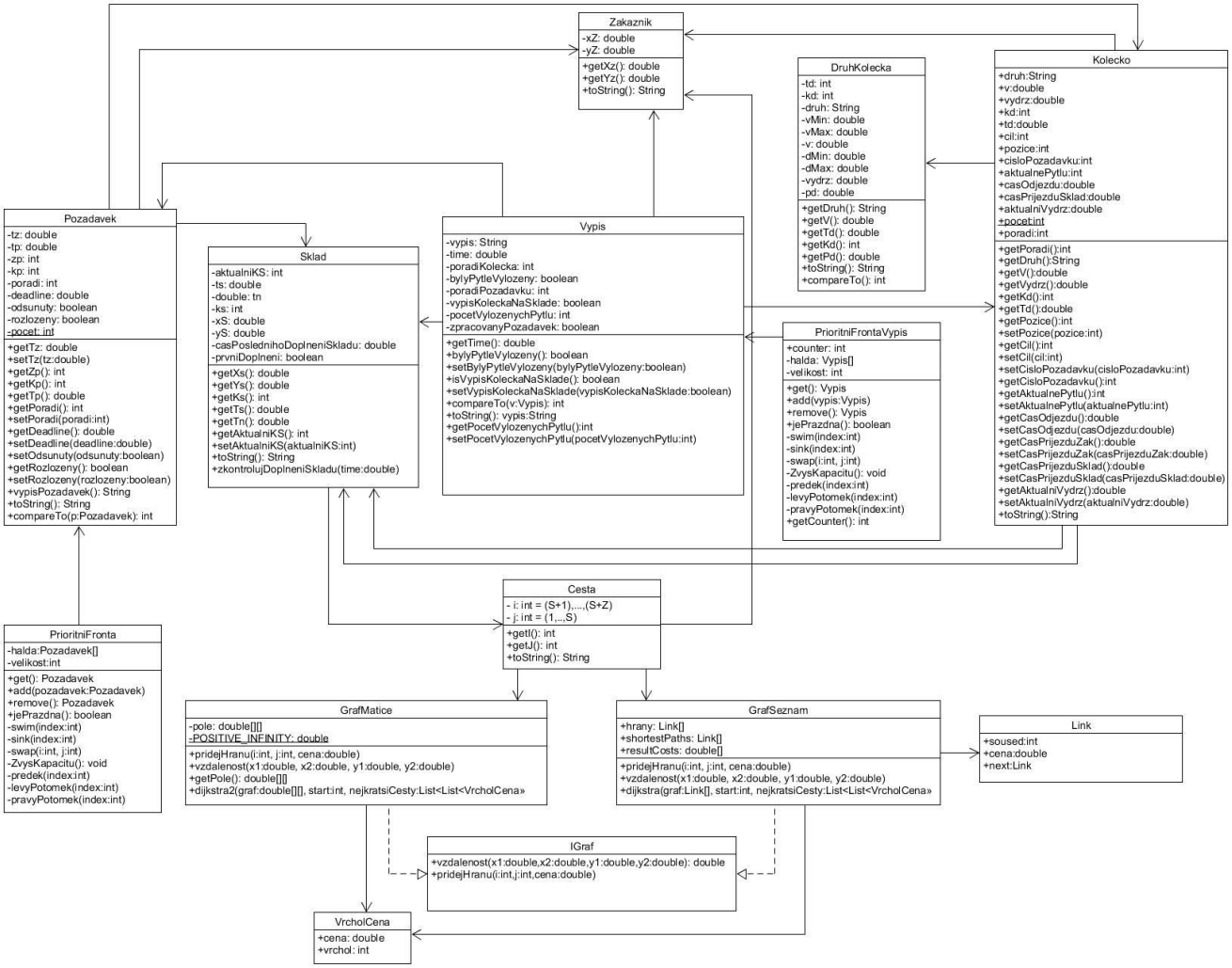
Dále jsme se museli rozhodnout, který algoritmus zvolit pro získání nejkratší cesty ze skladů k zákazníkům, rozhodli jsme se využit **Dijkstrův algoritmus**, který lépe zpracovává soubory obsahující větší počet vrcholů než hran, a vypočítává cestu pouze z jednoho vrcholu ke všem ostatním, zatímco například **Floyd-Warshallův algoritmus** jej tvoří mezi všemi dvojicemi vrcholů, což se nám ale v některých případech nehodí, protože pouze potřebujeme nejkratší cestu ze skladu k cílovému zákazníkovi.

Navíc jsme tento algoritmus museli vylepšit, a to kvůli tomu, aby nám ukládal vrcholy vedoucí nejkratší cestou k cílovému zákazníkovi a s ním i jejich ceny, díky tomu jsme dále byli schopní zhotovit kuk na kolečko, jak pro cestu k cílovému zákazníkovi, tak i na cestu zpět do skladu.

Dalším problémem bylo řešení času v simulaci, rozhodli jsme se ji nejprve řešit pomocí for cyklu, kde jsme postupně po sekundě zvyšovali čas simulace, nepočítali jsme ale s problémem, který mohl nastat, a tím byl příchod požadavku v setinách času, tudíž jsme museli čas implementovat jinak. Napadlo nás tento problém řešit prioritní frontou, do které jsme ukládali požadavky, a v každém průběhu cyklu hlavního programu se čas odvíjel od nejnižšího času vyskytujícího se právě na začátku této fronty. Požadavky, které nešly zpracovat v danou chvíli se nám vyřešili díky prioritní frontě taky, stačilo je předpočítat dopředu a následně do této fronty vložit, a postupem programu jsme se k nim vrátili.

Na výpisy programu jsme opět využili prioritní frontu, a to proto, aby byly výpisy vypsány v lineárním čase, navíc pokud program zjistí, že nestihne požadavek zpracovat, běží program do deadline tohoto požadavku a zpracovává ostatní požadavky až do deadlinu nezpracovatelného požadavku a až poté se program ukončí.

Návrh programu (UML diagram)



UML diagram

Uživatelská dokumentace

Soubor se spouští z příkazové řádky za použití příkazu „java Main“ v adresáři, ve kterém se nachází kompilované soubory dané aplikace (.class), za tento příkaz v příkazové řádce ještě následuje i parametr, kterým se zadá i soubor, nad kterým se má daná simulace provést, takže například:

```
C:\Users\Václav\IdeaProjects\Newest\out\production\PT>java Main C:\Users\Václav\IdeaProjects\Newest\tutorial.txt
Cas: 0, Pozadavek 1, Zakaznik: 4, Pocet pytlu: 5, Deadline: 10080
Cas: 0, Kolecko: 1, Sklad: 1, Nalozeno pytlu: 2, Odjezd v: 6
```

Správné spuštění programu

Program přijímá pouze jeden argument, pokud není zadán soubor žádný, či více, program to nahlásí.

```
C:\Users\Václav\IdeaProjects\Newest\out\production\PT>java Main
Uživatel nezadal vstupní soubor, či jich zadal více.
```

Spuštění bez argumentu

A pokud se v příkazu nachází soubor, ale není programem zpracovatelný, vyhodí program chybu.

```
C:\Users\Václav\IdeaProjects\Newest\out\production\PT>java Main xd
CHYBA PŘI ČTENÍ SOUBORU
java.io.FileNotFoundException: xd (Systém nemůže nalézt uvedený soubor)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:213)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:152)
        at java.base/java.util.Scanner.<init>(Scanner.java:645)
        at Main.nactiData(Main.java:135)
        at Main.main(Main.java:268)
Exception in thread "main" java.lang.NumberFormatException: Cannot parse null string
        at java.base/java.lang.Integer.parseInt(Integer.java:623)
        at java.base/java.lang.Integer.parseInt(Integer.java:777)
        at Main.vytvorInstance(Main.java:204)
        at Main.main(Main.java:270)
```

Spuštění s chybným souborem

Závěr

Zpracování programu bylo nakonec obtížnější, než jsme čekali, například zpracování dovážení pytlů v sobě nese více úskalí, než jsme si při návrhu programu mysleli. Myslíme si, že jsme nakonec vymysleli vcelku slušně efektivní algoritmus pro posílání koleček a zpracování jednotlivých požadavků.

Program nám zabral i hodně času, jelikož jsme oba začátečníci, a tak ještě nemáme tolik zkušeností, zároveň se pro nás oba jedná o náš největší projekt, na kterém jsme pracovali a myslíme si, že nám přinesl celkem dost vědomostí.

Jelikož jsme se semestrální prací celkem bojovali, nezbyl nám čas pro dodělání bonusových úkolů, které si myslíme, že by nám také hodně dali.

Největším problémem programu je jeho neefektivita, kdy třída Main je velice složitá, a tak nedokáže zpracovat rozsáhlejší soubory, na tom bychom chtěli zapracovat i po odevzdání semestrální práce a dovést práci do lepší podoby. Zároveň si myslíme, že jsme se dobře vypořádali s prioritní frontou v programu, jež nám hodně usnadnila další práci.

Ve dvojici se nám pracovalo výborně, a to hlavně z důvodu, že jsme oba právě začátečníci, a tak jsme na práci nechali stejnou míru úsilí a mohli jsme si tak oba z práce vzít co nejvíc, určitě spolu budeme chtít spolupracovat na dalších projektech i dále. Kód jsme psali společně a vzájemně jsme o kódu a různých možnostech zpracování daného problému debatovali.