FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z předmětu
Programovací techniky

Necháme to bloudovi s. r. o.

1. prosince 2022

Autoři:
Pavlov Volodymyr
A21B0235P
shark32@students.zcu.cz

Lember Jakub
A21B0196P
jlembe22@students.zcu.cz

Obsah

Kapitola 1

Zadání

Standardní zadání semestrální práce pro KIV/PT 2022/2023

Zadání je určeno pro dva studenty. Práce zahrnuje dvě dílčí části - vytvoření funkčního programu diskrétní simulace a napsání strukturované dokumentace.

Zadání

Zásobovací společnost Necháme to bloudovi s. r. o. se specializuje na přepravu zboží do saharských oáz. Její majitel Harpagon Dromedár je však vyhlášená držgrešle, a tak nejraději využívá jako přepravní prostředky velbloudy, kteří nejsou nároční na údržbu a provoz. Přepravované zboží je uskladněno ve speciálních koších, které jsou přichycovány na velbloudí hrby. Pro svoji živnost Harpagon využívá jak velbloudy jednohrbé, známé též jako dromedáry, tak velboudy dvouhrbé, kteří jsou někdy označováni jako drabaři. V poslední době se však starému Harpagonovi moc nedaří a spousta zvířat mu v poušti, částečně i vlivem změny klimatu, uhynula, což je pro něj citelná finanční rána, která mu dělá vrásky na čele. Rozhodl se tedy, že je čas dát prostor moderním technologiím, a proto si chce nechat vytvořit software, který mu pomůže rozplánovat přepravu všeho poptávaného zboží do oáz tak, aby nepřišel o nějaké další zvíře, dodržel závazky a neztratil klientelu, maximálně využil nosnosti zvířat a zároveň zvířata zbytečně neunavil delší cestou, než kterou opravdu musí jít. Tyhle požadavky můžeme jednodušše označit jako snahu o minimalizaci „ceny“ přepravy. Vytvořme pro Harpagona simulační program, který mu pomůže naplánovat přepravu, známe-li:

- počet skladů S,
- každý sklad bude definován pomocí:
  - kartézských souřadnic skladu x_s a y_s,
  - počtu košů k_s, které jsou do skladu vždy po uplynutí doby t_s doplněny. Na začátku simulace předpokládejte, že došlo k doplnění skladů, tj. ve skladu je k_s košů,

  - doby t_n, která udává, jak dlouho trvá daný typ koše na velblouda naložit/vyložit (každý sklad může používat jiný typ koše, se kterým může být různě obtížná manipulace),

- počet oáz O,
- kartézské souřadnice každé oázy x_o a y_o,
- počet přímých cest v mapě C,
- seznam cest, přičemž každá cesta je definována indexy i a j, označujících místa (oáza, sklad) v mapě, mezi kterými existuje přímé propojení, a platí: $i \in \{1, ..., S, S+1, ...S+O\}, j \in \{1, ..., S, S+1, ...S+O\}$, tj. sklady jsou na indexech od 1 do S a oázy na indexech od $S+1$ do $S+O$ Pozn. pokud existuje propojení z místa i do místa j, pak existuje i propojení z místa j do místa i,
- počet druhů velbloudů D,
- informace o každém druhu velblouda, kterými jsou:
  - slovní označení druhu velblouda, které bude uvedeno jako jeden řetězec neobsahující bílé znaky,
  - minimální v_{min} a maximální v_{max} rychlost, kterou se může daný druh velblouda pohybovat, přičemž jedinec se pohybuje konstantní rychlostí (obecně reálné číslo), která mu bude vygenerována v daném rozmezí pomocí rovnoměrného rozdělení,
  - minimální d_{min} a maximální d_{max} vzdálenost, kterou může daný druh velblouda překonat na jedno napojení, přičemž pro jedince je tato doba opět konstantní a jedná se o reálné číslo vygenerované v daném rozmezí pomocí normálního rozdělení se střední hodnotou $\mu = (d_{min} + d_{max})/2$ a směrodatnou odchylkou $\sigma = (d_{max} - d_{min})/4$, (pozn. velbloudi se mohou napít pouze v oázách a nebo ve skladech, jinde není voda k dispozici),
  - doba t_d, udávající kolik času daný druh velblouda potřebuje k tomu, aby se napil,
  - počet košů k_d udávající maximální zatížení daného druhu velblouda,
  - hodnota p_d udávající poměrné zastoupení daného druhu velblouda ve stádě, hodnota je zadána jako desetinné číslo z intervalu $\langle 0, 1 \rangle$, přičemž platí $\sum_{d=1}^{D} p_d = 1.0$,
- počet požadavků k obsloužení P,
- každý požadavek bude popsán pomocí:
  - času příchodu požadavku t_z (pozn. požadavek přichází doopravdy až v čase t_z, tzn. nemůže se stát, že by jeho obsluha začala dříve, a že by v době příchodu požadavku byl náklad již na cestě),

– indexu oázy $o_p \in \{1, ...O\}$, do které má být požadavek doručen,
– množství košů k_p, které oáza požaduje,
– doby t_p udávající, za jak dlouho po příchodu požadavku musí být koše doručeny (tj. nejpozději v čase $t_z + t_p$ musí být koše vyloženy v oáze).

Za úspěšně ukončenou simulace se považuje moment, kdy jsou všechny požadavky obsloužené a všichni velbloudi se vrátili 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 výpisů použijte jednu z následujících variant (formát je závazný, indexace od jedné):

- Příchod požadavku:
  `Cas: <t>, Pozadavek: <p>, Oaza: <o>, Pocet kosu: <k>,`
  `Deadline: <t+t`$_p$`>`
- Ve skladu se začíná připravovat velbloud na cestu:
  `Cas: <t>, Velbloud: <v>, Sklad: <s>, Nalozeno kosu: <k>,`
  `Odchod v: <t+k · t`$_n$`>`
- Velbloud došel do oázy, kde bude něco vykládat (pozn. výpis nikde v průběhu nebude odřádkován, časovou rezervou t_r je myšlen rozdíl mezi časem, kdy má být náklad nejpozději vyložen a časem, kdy k vyložení opravdu došlo):
  `Cas: <t>, Velbloud: <v>, Oaza: <o>, Vylozeno kosu: <k>,`
  `Vylozeno v: <t+k · t`$_n$`>, Casova rezerva: <t`$_r$`>`
- Velbloud došel do oázy/skladu, kde se musí napít před další cestou:
  `Cas: <t>, Velbloud: <v>, Oaza: <o>, Ziznivy <druh>, Pokracovani`
  `mozne v: <t+t`$_d$`>`
  nebo:
  `Cas: <t>, Velbloud: <v>, Sklad: <s>, Ziznivy <druh>, Pokracovani`
  `mozne v: <t+t`$_d$`>`
- Velbloud prochází oázou, ale nemá zde žádný zvláštní úkol:
  `Cas: <t>, Velbloud: <v>, Oaza: <o>, Kuk na velblouda`
- Velbloud dokončil cestu a vrátil se do skladu:
  `Cas: <t>, Velbloud: <v>, Navrat do skladu: <s>`
- Požadavek se nepodařilo (z jakéhokoli důvodu) obsloužit včas:
  `Cas: <t>, Oaza: <o>, Vsichni vymreli, Harpagon zkrachoval,`
  `Konec simulace`

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

```
...
Cas: 12, Pozadavek: 2, Oaza: 2, Pocet kosu: 3, Deadline: 30
Cas: 12, Velbloud: 5, Sklad: 3, Nalozeno kosu: 3, Odchod v: 18
Cas: 21, Velbloud: 5, Oaza: 1, Ziznivy dromedar,
Pokracovani mozne v: 22
Cas: 23, Velbloud: 5, Oaza: 10, Kuk na velblouda
Cas: 24, Velbloud: 5, Oaza: 2, Vylozeno kosu: 3, Vylozeno v: 30,
Casova rezerva: 0
...
```

Čas ve výpisu bude zaokrouhlen dle pravidel zaokrouhlení na celé číslo.

Minimální požadavky

- Funkcionalita uvedená v zadání výše je **nutnou podmínkou pro finální odevzdání práce**, tj. simulační program při finálním odevzdání musí splňovat veškeré požadavky/funkcionalitu výše popsanou, **jinak bude práce ohodnocena 0 body**.
- V případě, že bude program poskytovat výše popsanou přepravu, ale **řešení bude silně neefektivní**, bude uplatněna bodová **penalizace až 20 bodů**.
- **Finální odevzdání práce** v podobě **.ZIP** souboru (obsahujícím zdrojové kódy + přeložené soubory + dokumentace) bude nahráno **na portál**, a to **dva celé dny před stanoveným termínem** předvedení práce, tj.:
  - studenti, kteří mají **termín předvedení** stanoven na **pondělí**, nahrají finální verzi práce na portál **nejpozději v pátek ve 23:59**,
  - studenti, kteří mají **termín předvedení** stanoven na **čtvrtek**, nahrají finální verzi práce na portál nejpozději **v pondělí ve 23:59**.

  Při nedodržení tohoto termínu bude uplatňována **bodová penalizace 30 bodů**, navíc studentům nemusí být umožněno předvedení práce ve smluveném termínu.

Vytvoření funkčního programu:

- Seznamte se se strukturou vstupních dat (polohou skladů a oáz, informacemi o cestách, velbloudech, požadavcích ...) a načtěte je do svého programu. Formát souborů je popsán přímo v záhlaví vstupního souboru `tutorial.txt` **(5 bodů)**.

- Navrhněte a implementujte vhodné datové struktury pro reprezentaci vstupních dat, důsledně zvažujte časovou a paměťovou náročnost algoritmů pracujících s danými strukturami **(10 bodů)**.

- Proveďte základní simulaci jedné obslužné trasy včetně návratu velblouda do skladu. Vypište celkový počet doručených košů > 0 a celkový počet obsloužených požadavků > 0. Trasa velblouda musí být smysluplná. **(10 bodů)**.

Výše popsaná část bude váš minimální výstup při kontrolním cvičení cca v polovině semestru

- Vytvořte prostředí pro snadnou obsluhu programu (menu, ošetření vstupů včetně kontroly vstupních dat) - nemusí být grafické, během simulace umožněte manuální zadání nového požadavku na zásobování některé oázy či odstranění některého existujícího **(5 bodů)**.

- Umožněte sledování (za běhu simulace) aktuálního stavu přepravy. Program bude možné pozastavit, vypsat stav přepravy, krokovat vpřed a nechat doběhnout do konce, podobně jako je tomu v debuggeru **(5 bodů)**.

- Proveďte celkovou simulaci a vygenerujte do souborů následující statistiky (v průběhu simulace ukládejte data do vhodných datových struktur, po jejím skončení je uložte ve vhodném formátu do vhodně zvolených souborů) **(10 bodů)**:

  - přehled jednotlivých velbloudů - základní údaje o velbloudovi (druh; id domovského skladu; rychlost; max. vzdálenost, kterou urazí na jedno napojení), uskutečněné trasy (čas, kdy opustil sklad; kudy šel; kolik toho vezl; kam a kdy doručoval zboží; kde a kdy se zastavil na napojení; kdy se vrátil do svého domovského skladu), jak dlouho za celou dobu simulace odpočíval (tj. byl ve skladu a čekal na přiřazení požadavku) a celkovou vzdálenost, kterou ušel,
  - přehled jednotlivých oáz - čas a velikost vzniklého požadavku; kdy musel být nejpozději doručen; kdy byl skutečně doručen; ze kterého skladu a kterým velbloudem byl obsloužen,
  - přehled jednotlivých skladů - časy, kdy došlo k doplnění skladu; kolik košů v té době ve skladu zbývalo a kolik jich je k dispozici po doplnění,
  - délka trvání celé simulace, celková doba odpočinku všech použitých velbloudů, celková ušlá vzdálenost, kolik velbloudů od jednotlivých druhů bylo použito. Nemá-li úloha řešení, vypište, kdy a kde došlo k problému.

- Vytvořte generátor vlastních dat. Generátor bude generovat vstupní data pomocí rovnoměrného rozdělení, přičemž volte vhodně rozsah hodnot pro jednotlivé veličiny. U seznamu cest se vyhněte duplikátům. Data budou generována do souboru (nebudou přímo použita programem) o stejném formátu jako již dodané vstupní soubory. Při odevzdání přiložte jeden dataset s řešitelnou úlohou a jeden dataset, kdy nebude možné obsloužit všechny požadavky včas. **(5 bodů)**.
- Vytvořte dokumentační komentáře ve zdrojovém textu programu a vygenerujte programovou dokumentaci (Javadoc) **(10 bodů)**.
- Vytvořte kvalitní dále rozšiřitelný kód - pro kontrolu použijte softwarový nástroj PMD (více na `http://www.kiv.zcu.cz/ herout/pruzkumy/pmd/pmd.html`), soubor s pravidly `pmdrules.xml` najdete na portálu v podmenu Samostatná práce **(10 bodů)**
  - mínus 1 bod za vážnější chybu, při 6 a více chybách nutno opravit,
  - mínus 2 body za 10 a více drobných chyb.
- **V rámci strukturované dokumentace (celkově 20 bodů):**
  - připojte zadání **(1 bod)**,
  - popište analýzu problému **(5 bodů)**,
  - popište návrh programu (např. jednoduchý UML diagram) **(5 bodů)**,
  - vytvořte uživatelskou dokumentaci **(5 bodů)**,
  - zhodnoťte celou práci a vytvořte závěr **(2 body)**,
  - uveďte přínos jednotlivých členů týmu (včetně detailnějšího rozboru, za které části byli jednotliví členové zodpovědní) k výslednému produktu **(2 body)**.

Kapitola 2

Analýza problému

Ze zadání vyplývá, že se jedná o grafovou úlohu pro hledání nejkratší cesty mezi vrcholy v grafu. Tento typ úlohy lze řešit několika známými algoritmy, mezi které patří například Dijkstrův algoritmus, Floyd–Warshallův algoritmus nebo A* algoritmus. Pokud budeme analyzovat jednotlivé algoritmy z pohledu jejich časové náročnosti a náročnosti na paměť, můžeme dojít k výsledku, že Floyd–Warshallův algoritmus není příliš vhodný. Důvodem je jeho velká časová náročnost, která je ve všech možných případech $\Theta(|V|^3)$, kde V je počet vrcholů, a také náročnost na paměť, která je v nejhorším případě $\Theta(|V|^2)$ z důvodu vytváření matice sousednosti pro všechny vrcholy V. Dobrým kandidátem tedy může být Dijkstrův algoritmus, jehož časová náročnost je v nejhorším případě $\Theta(|E| + |V| \log|V|)$, kde V je počet vrcholů a E je počet hran. Tato časová náročnost je výsledkem toho, že nevytváříme matici sousednosti, ale je implementován prioritní frontou. Dalším dobrým kandidátem může být A* algoritmus, což je v podstatě "vylepšený"algoritmus Dijkstrův. Ten navíc využívá tzv. heuristickou funkci, která vyhledávání cesty urychluje.

Nakonec jsme se rozhodli použít právě A* algoritmus s Manhattanskou vzdáleností jako heuristickou funkcí. S volbou tohoto algoritmu se zřejmě pojí struktura celého projektu a použité výpočty.

Vstupní data

Nejprve jsme dostali sadu tří souborů pro otestování jejich načtení a zpracování. Jelikož se ve validních datech mohly vyskytovat jak řádkové, tak blokové komentáře, ve kterých se navíc také mohly objevit číselné hodnoty a bílé znaky, bylo nasnadě vytvořit takový algoritmus pro zpracování souborů, který by tato nežádoucí data "ignoroval"a vytvořil korektně reprezentovaný graf.

Graf načítáme ze zdrojového textové souboru po řádcích a následně zásobníkovou metodou procházíme jednotlivé řádky, ve kterých hledáme znaky uvozující a ukončující řádkové, potažmo blokové, komentáře. Tento problém jsme vyřešili tak, že pokud algoritmus pro zpracování souborů narazí na znak uvozující komentář (znak velblouda), hledá dále znak, který komentář ukončuje (znak pouště). Veškerý text uvnitř komentáře včetně obou těchto znaků vymaže. Takto upravená vstupní data dále použijeme pro vytvoření veškerých entit nutných pro simulaci. Vytváření entit grafu se řídí pokyny v zadání o tom, jak jsou data v souborech uspořádána.

Jelikož je tento algoritmus naprosto obecný pro vstupní data určená k této úloze,

fungoval i nadále po zveřejnění ostrých dat pro testování naší práce. Při načítání vstupních dat jsme nepočítali nejkratší vzdálenosti mezi jednotlivými body v grafu. Toto rozhodnutí jsme udělali kvůli dosažení větší čistoty kódu a také kvůli zvolenému algortimu pro jejich hledání.

Reprezentace grafu

Graf lze v paměti ukládat dvěma způsoby: maticí sousednosti nebo spojovým seznamem sousedů. My jsme zvolili druhou možnost v souvislosti se zvoleným algoritmem pro hledání nejkratší cesty.

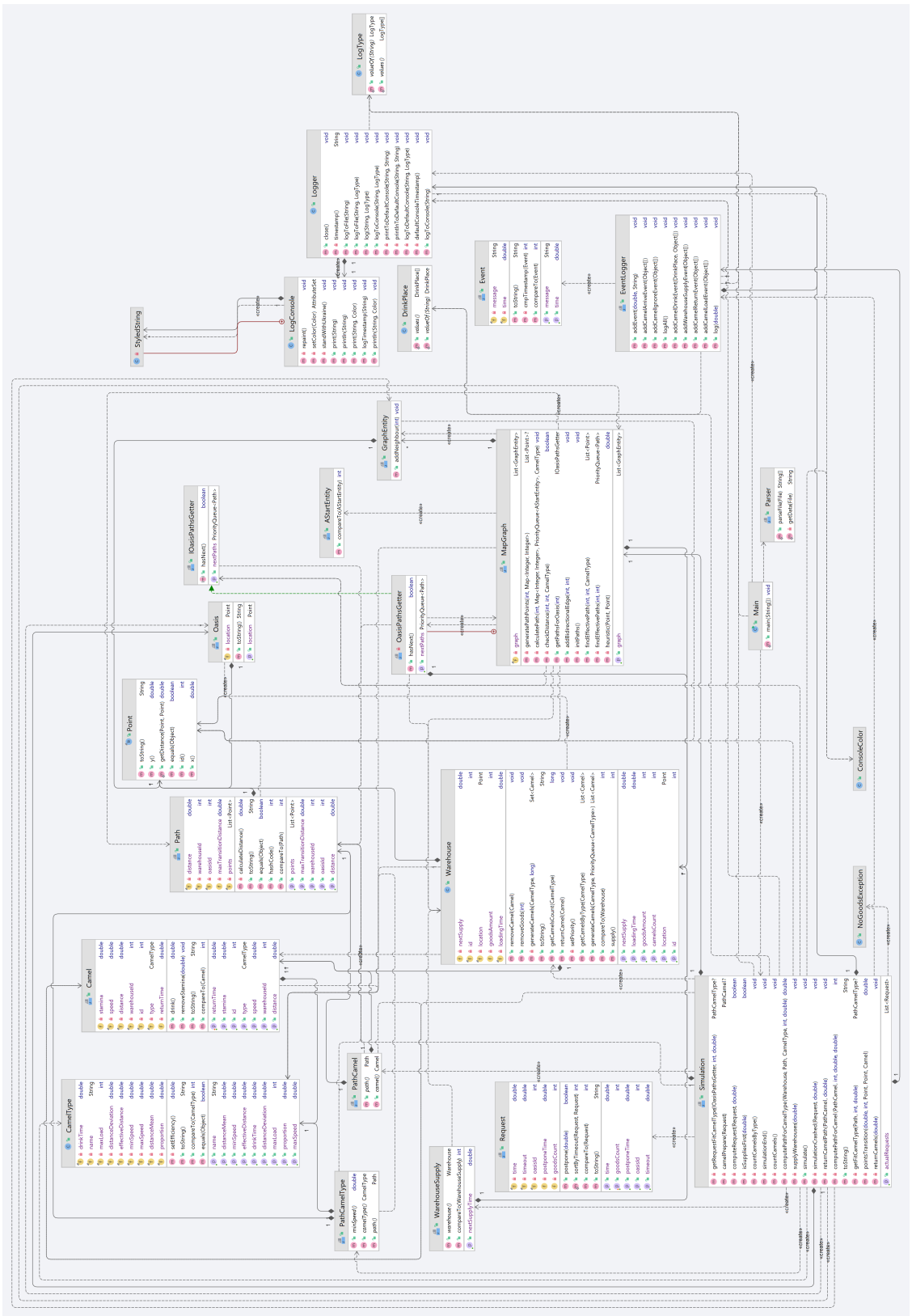
Graf je tvořen seznamem jednotlivých vrcholů grafu, kterými mohou být obecně buď sklad nebo oáza, přičemž každý z těchto vrcholů je definován svou pozicí a seznamem sousedů. Třída reprezentující graf obsahuje mimo jiné také mapu nejkratších cest mezi jednotlivými sklady, jejímž klíčem je číslo skladu a hodnotou další mapa, jejímž klíčem je daný sklad a hodnotou prioritní fronta cest tohoto skladu (tedy ty "nejvýhodnější"cesty); seznam skladů a jejich počet a prioritní frontu velbloudů, která je seřazena podle jejich typu, respektive jejich efektivity.

Kapitola 3

Návrh programu

Program je rozdělen do několika tříd: loggovací třídy, které mají na starosti zaznamenávat veškerý průběh simulace; třídy pro reprezentaci grafu; simulační, které mají na starosti samotnou simulaci požadavků a třídy tvořící model programu.

Detailní popis tříd programu je zobrazen na straně 11 pomocí UML diagramu:



Obrázek 3.1: UML diagram tříd

Kapitola 4

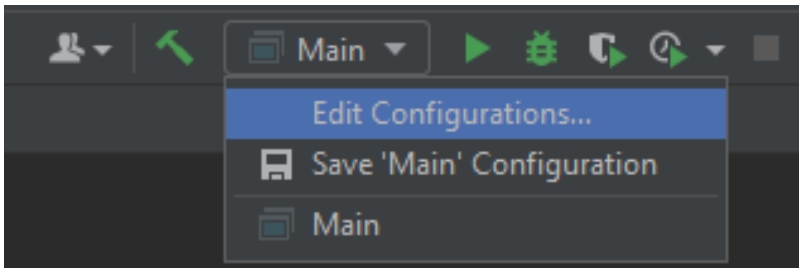
Uživatelská dokumentace

Program nemá žádné uživatelské rozhraní pro komunikaci s uživatelem, je tedy nutné jej spustit přímo z nějakého IDE, přičemž byl zkompilován pomocí JDK verze 17. Pro jeho spuštění je nutno přidat do argumentů programu název souboru, jehož scénář chcete spustit. Buďto můžete mít uložené textové soubory přímo v kořenovém adresáři projektu, nebo ve složce pojmenované podle Vaší libosti. Pak je ovšem nutné zapsat jméno i této složky do argumentů programu.

Program po jeho spuštění vypíše do konzole průběh dané simulace a také v kořenovém adresáři projektu vytvoří soubor s příponou `.log`, ve kterém se nachází tentýž výpis. Tento soubor lze otevřít v jakémkoliv textovém editoru.

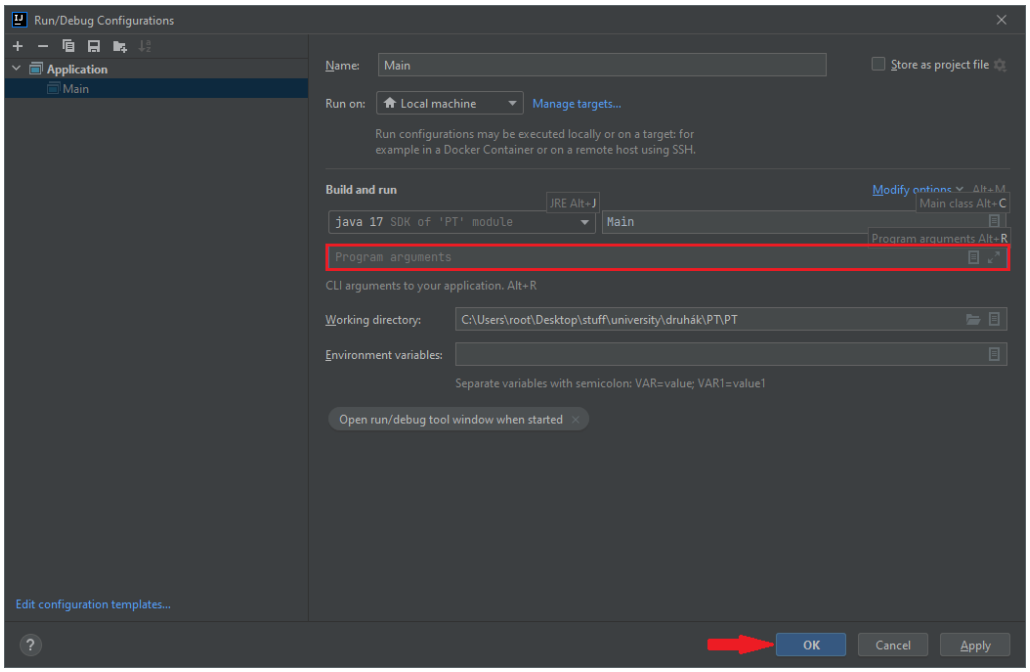
Zde je krátký návod pro spuštění programu v IntelliJ IDEA:

1. Vpravo nahoře rozklikněte nabídku s názem třídy "Main"a klikněte na "Edit Configurations..."



Obrázek 4.1: Kontextová nabídka konfigurace třídy Main

2. V políčku "Program arguments"zadejte název textového souboru, popřípadě cestu k němu s názvem složky, kde je uložen, a vpravo dole klikněte na "OK"



Obrázek 4.2: Argumenty programu

3. Vpravo nahoře vedle již zmíněné nabídky s názvem třídy klikněte na zelenou šipku, čímž program spustíte

Kapitola 5

Závěr

Námi zhotovený program nesplňuje v plném rozsahu všechny body zadání. Konkrétně program nelze v průběhu simulace pozastavit, přidat nový požadavek či nějaký odebrat a nelze program krokovat. Není tedy možné běh simulace jakkoliv modifikovat.

Na druhou stranu program simulaci úspěšně provede a to i v případě vstupních souborů obsahujících velké množství dat. To se nám povedlo díky optimalizaci hledání nejkratší cesty, jejímž výsledkem je doběhnutí programu v rekordně krátkém čase i pro tato velká data.

Za zmínku ovšem stojí logger. I přesto, že program nelze v průběhu simulace pozastavit, lze díky němu sledovat její průběh. V loggeru můžeme vidět postupné načítání dat, vytváření jednotlivých objektů simulace a také vyřízené požadavky. Lze tedy konstatovat, že bod zadání vyžadující tuto funkcionalitu, je splněn částečně. Výstup tohoto loggeru je také převeden do souboru, který obsahuje všechny důležité informace o průběhu simulace

Kapitola 6

Rozdělení práce mezi členy týmu

Na zhotovení programu se podíleli oba členové přiměřeně rovnoměrným dílem. V počáteční fázi vývoje jsme si rozdělili práci následovně: Jakub Lemberk připravoval model programu, tedy potřebné třídy a metody pro reprezentaci skladů, oáz, velbloudů apod. Ve chvíli, kdy byl základní model hotov, Volodymyr Pavlov vytvořil algoritmus pro zpracování vstupních dat programu. Následně jsme si rozdělili práci tak, že Jakub Lemberk pracoval na pomocných metodách pro simulaci běhu programu a Volodymyr Pavlov vytvářel A* algoritmus pro procházení grafu. Po krátké době jsme ovšem narazili na problém, a to sice, že jeden z nás potřeboval metody, které psal ten druhý a naopak. Rozhodli jsme se tedy, že bude pro větší efektivitu a možnost dokončení programu v rozumném čase lepší, když se budeme pravidelně scházet ve studovně a pracovat na řešení problému paralelně v jeden čas. Takto jsme byli schopni dosáhnout rychlejšího postupu, rychlejšího odchytávání chyb v programu a zdárného dokončení semestrální práce.

Seznam obrázků