Uživatelská a programátorská dokumentace

Vojtěch Vlachovský

1. ročník MFF UK, obor Informatika, kruh 33
Zimní semestr 2024/2025
Programování I (NPRG030)

30. ledna 2025

1 Úvod

Tento dokument popisuje uživatelskou a programátorskou dokumentaci k programu **VizuAlgo** který slouží k grafické vizualizaci algoritmu nejkratší cesty v orientovaném grafu.

2 Uživatelská dokumentace

2.1 Způsob použití

Program je předvybaven "defaultním" grafem kde se uživatel může seznámit s ovládáním programu, následně může uživatel vložit pomocí integrovaného průzkumníka souborů graf vlastní a nastavit zdrojový a cílový vrchol.

Po načtení libovolných parametrů nebo použití výchozích je program ovládán dvěmi tlačítky intuitivně v levém a pravém spodním rohu.

Tato tlačítka změní v tabulce hodnot jednotlivých uzlů jejich hodnotu postupně od výchozího uzlu až po cílový.

V případě že by uživatel zašel na 0. krok programu nebo na $n-1$ krok programu, jsou jejich tlačítka znepřístupněna aby nedošlo k neočekávané vyjímce.

Po "odkrokování hodnot" nebo kdykoliv v průběhu je zvýrazněna nejkratší cesta mezi zvolenými uzly, toto zvýraznění lze vypnout/zapnout pomocí tlačítka v levém horním rohu "Show SP".

2.2 Formát vstupních dat

Data jsou v ustáleném formátu grafů *.DOT* který využívají nástroje jako je například GraphWiz, formát vypadá například takto, kde vždy první hodnota je zdroj a druhá je cíl tedy $A \to B, B \to D$ etc.

```
graph G {
  A -- B [weight=3];
  B -- D [weight=3.5];
  B -- E [weight=2.8];
  C -- A [weight=3];
  C -- E [weight=2.8];
  C -- F [weight=3.5];
  D -- C [weight=9];
  D -- G [weight=10];
  E -- G [weight=7];
  F -- G [weight=2.5];
  G -- G [weight=0]
}
```

Je velice důležité aby hlavička obsahovala graph G, v případě že uživatel chce mít tzv. slepý uzel t.j. uzel kde $\deg^+(n) = 0$, pak přidá hranu z daného uzlu do daného uzlu s váhou hrany 0, viz. poslední řádek ukázkového vstupu.

Vstupní data je možné uložit do libovolné prázdné složky a následně v průběhu programu vybrat preferované parametry. Daný graf je možné také vizualizovat v nástroji GraphWiz v případě nesrovnalostí nebo nepřehlednosti.

3 Programátorská dokumentace

3.1 Anotace

Program VizuAlgo, dále jen program je nástroj umožňující dynamickou vizualizaci Dijkstrova algoritmu nejkratší cesty v grafu.

Program je vybaven nástroji které automaticky přečtou
zformátovaný graf (formátu .DOT) a následně uživateli umožní hledání cesty od libovolných uzlů v grafu pomocí integrovaného GUI (graphical user interface).

Využití závisí zcela na uživateli a to ve smyslu pokud jediné co ho zajímá je korektní nejkratší cesta grafem nebo jednotlivé mezivýpočtové hodnoty.

3.2 Struktura programu

- `main.py` - hlavní soubor programu určený jako spouštěč
- `dijkstra.py` - soubor který provádí kalkulace dijkstrova algoritmu a vrací hodnoty potřebné pro vizualizaci
- `visualization.py` - soubor využivívající knihovny `pygame,pygame GUI,networkX` určené k veškeré grafické reprezentaci

3.3 Použitý algoritmus

Program vizualizuje *Dijkstrův algoritmus* který slouží k hledání nejkratší cesty v souvislém grafu v našem případě jsou vstupní grafy vždy orientované.

Algoritmus docílí výsledku pomocí předem nastavených "vah" hran které určují v reálném smyslu například délku komunikace nebo čas který nám zabere dostat se z bodu A do bodu B pomocí dané komunikace.

3.4 Volba vizualizační knihovny

Pro program byla zvolena knihovna pygame primárně kvůli předchozím zkušenostem s její implementací a kvůli mému osobnímu zájmu co se týče grafických softwarových aplikací, herních enginů a knihoven.

Alternativy pro pygame může být například pyOpenGL ale tato knihovna co se týče její složitosti a křivky učení je naprosto nepřijatelná pro škálu ročníkového projektu a v tak krátkém pracovním intervalu by nebyla ani zdaleka využita.

3.5 Hlavní objekty a funkce

- `main.py`
  - funkce obsahuje pouze funkci main kde se nachází výchozí odladěný graf určený pro seznámení se s nástrojem a jeho fungováním, slouží výhradně k startu programu a volání funkce pro vizualizaci.
- `dijkstra.py`
  - `class Graph()`
    Funkce obsahuje výchozí konstruktor, třídu pro čtení vstupu a převodu vstupu na třídu graf, a samotnou implementaci algoritmu
  - `dijkstra(self, source: str, target: str)`
    Funkce využívá prioritní frotnu z knihovny `heapq` která zaručuje rychlý a optimalizovaný průběh při hledání cesty.
    Samotný algoritmus byl částečně modifikován a to tak aby ukládal průběžné kroky, změny hodnot a výpočty které funkce společně s ostatními hodnotami vrací.

- `shortest_path(self,source: str, target: str)`
  Funkce obdrží při volání jako parametry hodnoty vypočítané funkcí `dijksra()` a následně pomocí iterativního backtrackingu najde nejkratší cestu mezi předem zvolenými uzly. Funkce vrací pouze List posloupností uzlů které jsou nejkratší cestou.

- `visualization.py`

  - `class Object`
    Objekt je třída která reprezentuje jednotlivé vrcholy grafu, je ale zároveň vytvořená tak aby z ní mohly ostatní třídy dědit nejdůležitější informace tedy : poloměr,(x,y) středového bodu, typ, jméno, váhu hrany.
  - `class Arrow(Object)`
    Tato funkce je funkcí dědící z funkce Object a tedy pouze rozšiřuje funkci o dodatečné parametry potřebné k renderování šipek mezi uzly.
  - `render(self,surface,font,background_color, color)`

    Tato funkce využívá trigonometrických výpočtů pro vykreslení šipek pod správným úhlem, jelikož šipka jako taková není standardizovaný tvar v knihovně pygame je potřeba vytvořit uniformní polygony tvaru šipek a tedy i práce s nimi je značně náročná.

    Funkce vypočítává zdrojové a cílové pozice které jsou počítány od konce šipky po její hrot, samotná délka šipky mimo její hrot je označená proměnnou `shaft_length` která vezme pozice jejího cílového a zdrojového vrcholu ^{2} sečte je s rozdílem pozic na ose y a výsledkem je vždy přesná velikost mezi dvěma vrcholy.

    Samotný hrot byl již vytvářen pomocí opakovaného hledání vhodných škálovacích parametrů.
  - `parseInput(input_string)`
    Tato funkce ač se již objevila v jiném souboru je modifikovaná funkce která pomocí "regulárních výrazů" izoluje potřebné hodnoty ze vstupu. Je používána při načtení vstupu ze souboru v případě že je program spuštěn.

– `renderGraph(graph, surface, font, screen, distances, source_node, opacity=255)`
  Jelikož rozmístění vrcholů grafu při jeho renderování je složitostí program hodný ročníkového projektu využívám k rozmístění vrcholů externí knihovnu `networkX`, která přiřadí každému vrcholu grafu jeho vektor (x,y).
  Poté jsou dopočítány jeho další hodnoty jako například průměr kružnice nebo její pozice vůči ostatním.

  Dalším krokem je vytvoření seznamů hran a vrcholů, to je implementováno iterativním voláním konstruktorů daného objektu uvnitř inicializace seznamu a delegování již vypočítaných a konstantních hodnot konstruktoru daného objektu.

– `visualize(graph, source_node,target_node,select_source, select_target)`
  Tato funkce je srdcem celého programu `visualization.py`, stará se o průběh programu a obstarává všechny interní mechaniky knihovny `pygame`.

  Obsahuje veškeré inicializace GUI prvků knihovny jako jsou tlačítka, interní hodiny, "labely", tabulku hodnot a průzkumníka souborů. Veškerá logika funkce se nachází uvnitř a okolo tzv. "GameLoopu" (náš program sice není hrou ale využíváme herní engine). Uvnitř herní smyčky se nachází veškerá logika a opatření pro správný chod programu tedy tlačítka a co má program dělat po jejich stisknutí případně jiné interakci.

3.6 Průběh práce

Nad projektem jsem přemýšlel od jeho oznámení, zprvu mi nebylo příliš jasné, co bych mohl vytvořit, abych splňoval podmínky pro uznání, ale následně při přednášce Algoritmizace mi v hlavě utkvěl nápad vytvořit nástroj, který by mi pomohl představit si, jak nějaký algoritmus funguje. Nakonec jsem se rozhodl že nejlepší algoritmus pro takový program by byl právě Dijkstrův a pustil jsem se do práce.

Hrubou kostru program jsem měl zhruba za týden intenzivní práce a pak nastalo dlouhé téměř měsíční období optimalizace, zkrášlování a testů. S výsledkem jsem velmi spokojený, jelikož toto byla má první zkušenost s pygame programem takového rozsahu, a i když by někdo mohl namítat, že by program mohl být detailnější, tak si myslím, že program, který jsem sepsal ve specifikaci, je naprostým obrazem programu, který je finálním výsledkem.