

# Dokumentace knihovny pro práci s grafy

Jan Kleprlík

## Zadání

Knihovna pro práci s grafy slouží k jednoduchému vytváření a manipulaci grafů v jiných programech. Součástí knihovny pak jsou také základní algoritmy na průchod grafu, hledání nejkratších cest, či nalezení komponent souvislosti. Tuto knihovnu poté aplikovat na hru Pacman, konkrétně řídit pohyb příšer v bludišti.

## Práce s knihovnou

### Práce s grafem

Základem je vytvoření grafů. Na výběr je ze čtyř možností. Každá z možností splňuje požadavky, které nese ve svém názvu. Knihovna je psaná v anglickém jazyce, takže i ovládání knihovny je v angličtině.

- Orientovaný ohodnocený graf – *DirectedWeightedGraph*
- Orientovaný neohodnocený graf – *DirectedUnweightedGraph*
- Neorientovaný ohodnocený graf – *UndirectedWeightedGraph*
- Neorientovaný neohodnocený graf – *UndirectedUnweightedGraph*

U každého grafu se dají vyvolat následující proměnné.

Proměnná	Popis
<i>directed</i>	Boolean – true, když je graf orientovaný, jinak false
<i>weighted</i>	Boolean – true, když je graf ohodnocený, jinak false
<i>verticesCount</i>	Int – počet vrcholů v grafu
<i>edgesCount</i>	Int – počet hran v grafu
<i>multigraph</i>	Boolean – true, když je graf může obsahovat multihran
<i>vertices</i>	List vrcholů v grafu
<i>edges</i>	List hran v grafu

Zároveň se dají vyvolat následující metody.

Metoda	Popis
<i>AddVertex</i>	Přidá do grafu vrchol.
<i>RemoveVertex</i>	Odebere z grafu vrchol
<i>ExistVertex</i>	Vrátí true, pokud vrchol je v grafu, jinak false.
<i>GetVertex</i>	Vrátí objekt vrchol.
<i>AddEdge</i>	Přidá hranu do grafu.
<i>RemoveEdge</i>	Odebere hranu z grafu.
<i>ChangeWeight</i>	Změní ohodnocení hrany.
<i>GetEdgeWeight</i>	Vrátí ohodnocení hrany.

Clear	Vymaže graf
-------	-------------

## Práce s algoritmy

Pro algoritmy lze použít vhodné grafy. To znamená, že například, že Dijkstrův algoritmus nelze aplikovat na graf se záporně ohodnocenou hranou. U algoritmů, které se dají využít k získání nejkratší cesty mezi dvěma vrcholy lze zavolat metodu *GetShortestPath*, která vrátí zásobník s vrcholy nejkratší cesty. V takovém případě se spustí výjimka s popisem, co je špatně. Knihovna obsahuje celkem osm algoritmů.

Algoritmus	Popis	Předpoklady
<i>BreathFirstSearch</i>	Prohledá graf do šířky.	-
<i>DepthFirstSearch</i>	Prohledá graf do hloubky.	-
<i>Cycle</i>	Vrátí True, pokud graf obsahuje cyklus.	Graf je orientovaný.
<i>DijkstraShortestPath</i>	Aplikuje na graf Dijkstrův algoritmus pro hledání nejkratších cest.	Graf je orientovaný. Graf je ohodnocený. Graf neobsahuje záporně ohodnocenou hranu.
<i>BellmanFordShortestPath</i>	Aplikuje na graf Bellman-Fordův algoritmus pro hledání nejkratších cest.	Graf je ohodnocený. Graf neobsahuje záporný cyklus.
<i>FloydWarshallShortestPath</i>	Aplikuje na graf Floydův-Warshallův algoritmus pro hledání nejkratších cest.	Graf je ohodnocený. Graf neobsahuje záporný cyklus.
<i>TopologicalSort</i>	Topologicky uspořádá vrcholy grafu.	Graf je orientovaný. Graf je acyklický.
<i>StronglyConnectedComponents</i>	Vrátí silně souvislé komponenty grafu.	Graf je orientovaný.

## Aplikace knihovny na hru Pacman

Knihovna je aplikovaná na hru Pacman, kterou jsem vytvořil v průběhu druhého semestru, při přepracování objektového návrhu z úlohy „Příšera v bludišti“. Cílem bylo tedy bylo přidat do hry přidat algoritmus, podle kterého by se příšery pohybovali. Doposud příšery zatačeli náhodně a hráče nepronásledovali.

Aplikaci jsem vzhledem k vytváření jednotlivých kol a jejich načítání udělal následovně. Každá pozice v matici, která představuje mapu, na kterou se může příšera, nebo hráč dostat jsem označil jako vrchol grafu. Hrany pak vedou mezi každými dvěma sousedními pozicemi dostupné hráči, či příšeře.

Na hledání cesty jsem použil jednoduché prohledání do šířky a do hloubky. Modrá příšera hledá nejkratší cestu k hráči každé ušlé políčko. Žlutá a fialová příšera se podívají kde je hráč a na jeho

pozici dojdou. Pak hráče najdou znovu a tento proces se opakuje. Žlutá a fialová příšera jdou na poslední zhlédnutou pozici hráče ze dvou důvodů. Jednak kvůli tomu, aby se eventuálně všechny příšery nesešly na jednom místě a nehonili hráče. Druhý důvod je, kvůli menším nárokům na neustálé přepočítávání a procházení grafu. Fialová příšera jde po cestě vypočítané průchodem do hloubky. Žlutá jde po nejkratší cestě vypočítané průchodem do šířky.

## Detaily knihovny

Knihovna je rozdělena na dvě hlavní části, *DataStructures* a *Algorithms*. Ke každé pak existuje složka s unit testy, která obsahuje testy na správnost fungování datových struktur a algoritmů.

### Zvolené datové struktury

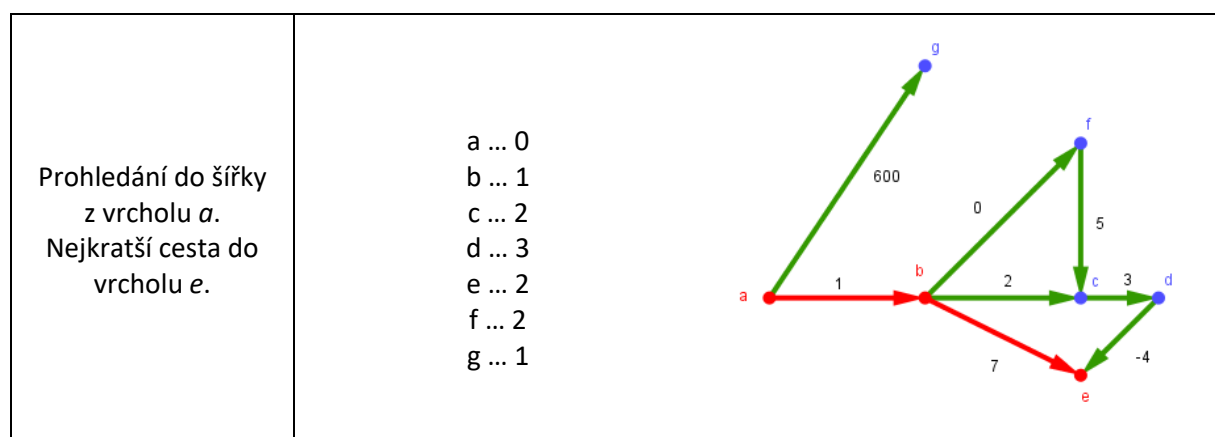
Čtyři hlavní datové struktury grafů dědí po jedné společné *GraphBase*, která řeší společnou implementaci metod a proměnných. Metody, které se u jednotlivých druhů grafů řeší jinak, jsou přepsány v samostatné datové struktuře. Všechny grafy zároveň dědí interface *IGraph*. Tak je zaručené, že základní metody budou u všech grafů. Další dvě klíčové datové struktury jsou *Vertex* a *Edge*. *Vertex* se využívá pro reprezentaci vrcholů v grafech a *Edge* pro reprezentaci hran. Poslední datová struktura je *MinHeap*. To je varianta haldy, kde na vrcholu je vždy prvek s nejmenším klíčem. Využívá se v Dijkstrově algoritmu.

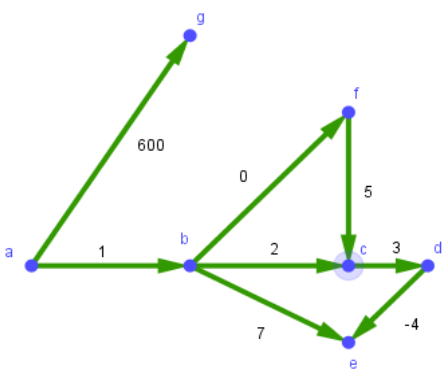
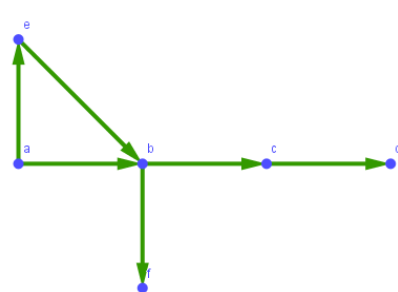
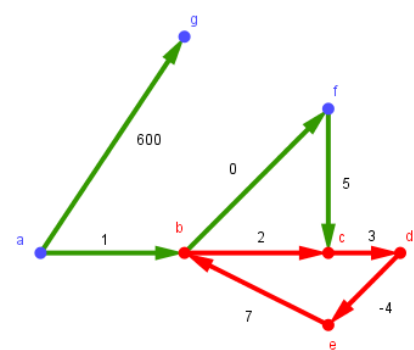
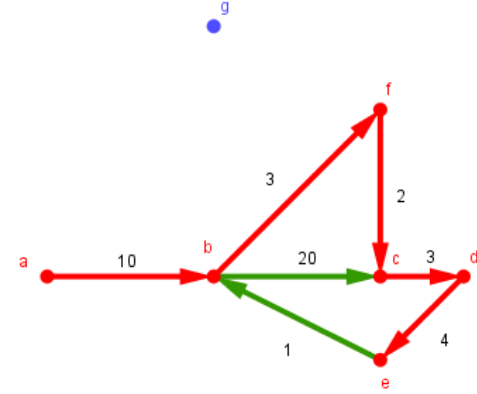
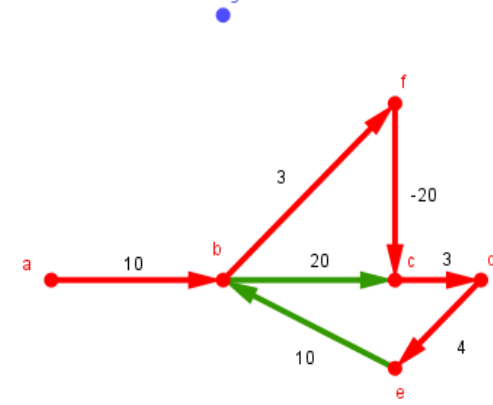
### Zvolené algoritmy

Zvolené algoritmy v knihovně jsou ty, které byli probrané na přednáškách „Algoritmy a datové struktury I“. Až na výjimky algoritmy upravují proměnnou *distance* u vrcholů grafů. Ta určuje vzdálenost určitého vrcholu od zvoleného vrcholu, ze kterého algoritmus začal. Výjimkou je algoritmus *Cycle*, ten pouze hledá cykly v grafu a vrátí boolean. Další výjimkou je Floydův-Warshallův algoritmus. Ten počítá vzdálenosti mezi každými dvojicemi vrcholů. Vrátí tedy matici vzdáleností a matici následníků, pomocí kterých se dají cesty rekonstruovat. Tento algoritmus upraví *distance* u vrcholů tak, že hodnota odpovídá místu vrcholu v matici. Poslední výjimkou je *StronglyConnectedComponents*. Tento algoritmus vrátí list obsahující listy, které obsahují vrcholy jednotlivých silně souvislých komponent.

### Unit testy

Ke každé datové struktuře i algoritmu je napsaný unit test, který ověří správnost metod. Níže jsou graficky zpracované grafy, na kterých se testuje správnost jednotlivých algoritmů. Čísla u hran jsou jejich váhy. Červeně jsou vyznačené vrcholy a hrany, po kterých vede nejkratší cesta, na kterou se v testech dotazuje. U každého grafu jsou ještě vypsány hodnoty proměnné *distance* pro všechny vrcholy.



<p>Prohledání do hloubky</p>	<div data-bbox="606 291 686 548"> <p>a ... 0 b ... 1 c ... 2 d ... 3 e ... 4 f ... 2 g ... 1</p> </div> 
<p>Detekování cyklu v ohodnoceném a neohodnoceném orientovaném grafu.</p>	<div data-bbox="654 649 813 694"> <p>Bez kružnice</p> </div>  <div data-bbox="1085 649 1244 694"> <p>S kružnicí</p> </div> 
<p>Dijkstrův algoritmus z vrcholu <i>a</i>. Nejkratší cesta do vrcholu <i>e</i>.</p>	<div data-bbox="606 1164 718 1422"> <p>a ... 0 b ... 10 c ... 15 d ... 18 e ... 22 f ... 13 g ... inf</p> </div> 
<p>Bellmanův-Fordův algoritmus z vrcholu <i>a</i>. Nejkratší cesta do vrcholu</p>	<div data-bbox="606 1612 718 1870"> <p>a ... 0 b ... 10 c ... -7 d ... -4 e ... 0 f ... 13 g ... inf</p> </div> 

<p>Floydův-Warshallův algoritmus. Nejkratší cesta z vrcholu <i>a</i> do vrcholu <i>e</i>.</p>	<table><tr><th></th><th>a</th><th>b</th><th>c</th><th>d</th><th>e</th><th>f</th><th>g</th></tr><tr><th>a</th><td>0</td><td>10</td><td>-7</td><td>-4</td><td>0</td><td>13</td><td>inf</td></tr><tr><th>b</th><td>inf</td><td>0</td><td>-17</td><td>-14</td><td>-10</td><td>3</td><td>inf</td></tr><tr><th>c</th><td>inf</td><td>17</td><td>0</td><td>3</td><td>7</td><td>20</td><td>inf</td></tr><tr><th>d</th><td>inf</td><td>14</td><td>-3</td><td>0</td><td>4</td><td>17</td><td>inf</td></tr><tr><th>e</th><td>inf</td><td>10</td><td>-7</td><td>-4</td><td>0</td><td>13</td><td>inf</td></tr><tr><th>f</th><td>inf</td><td>-3</td><td>-20</td><td>-17</td><td>-13</td><td>0</td><td>inf</td></tr><tr><th>g</th><td>inf</td><td>inf</td><td>inf</td><td>inf</td><td>inf</td><td>inf</td><td>0</td></tr></table>		a	b	c	d	e	f	g	a	0	10	-7	-4	0	13	inf	b	inf	0	-17	-14	-10	3	inf	c	inf	17	0	3	7	20	inf	d	inf	14	-3	0	4	17	inf	e	inf	10	-7	-4	0	13	inf	f	inf	-3	-20	-17	-13	0	inf	g	inf	inf	inf	inf	inf	inf	0
	a	b	c	d	e	f	g																																																										
a	0	10	-7	-4	0	13	inf																																																										
b	inf	0	-17	-14	-10	3	inf																																																										
c	inf	17	0	3	7	20	inf																																																										
d	inf	14	-3	0	4	17	inf																																																										
e	inf	10	-7	-4	0	13	inf																																																										
f	inf	-3	-20	-17	-13	0	inf																																																										
g	inf	inf	inf	inf	inf	inf	0																																																										
<p>Topologické uspořádání</p>	<table><tr><td>a ...</td><td>3</td></tr><tr><td>b ...</td><td>5</td></tr><tr><td>c ...</td><td>7</td></tr><tr><td>d ...</td><td>8</td></tr><tr><td>e ...</td><td>2</td></tr><tr><td>f ...</td><td>6</td></tr><tr><td>g ...</td><td>4</td></tr><tr><td>h ...</td><td>1</td></tr></table>	a ...	3	b ...	5	c ...	7	d ...	8	e ...	2	f ...	6	g ...	4	h ...	1																																																
a ...	3																																																																
b ...	5																																																																
c ...	7																																																																
d ...	8																																																																
e ...	2																																																																
f ...	6																																																																
g ...	4																																																																
h ...	1																																																																
<p>Silně souvislé komponenty</p>	<p>komponenty: (a,b) (c,d) (e)</p>																																																																

## Co by šlo udělat jinak a co přidat

Jednou z hlavních věcí, které by šli udělat jiným způsobem, než jsem zvolil je reprezentace grafu v paměti. Nyní je zvolená reprezentace pomocí seznamu sousedů. Jiná možnost by byla reprezentace pomocí matice sousednosti. Případně by se daly vytvořit další datové struktury pro husté grafy, které by byli reprezentovány pomocí matice sousednosti.

Přidat by šli především další algoritmy. Například algoritmus A\*, algoritmy na nalezení koster jako je Jarníkův, nebo Borůvkův algoritmus. Jistě ještě existuje více algoritmů, které existují, jen o nich zatím nevím.

## Průběh práce

Na začátku jsem velice dlouho přemýšlel, jakým způsobem budu graf reprezentovat, jestli využiju Interface, nebo jen dědičnost. Nakonec jsem se rozhodl pro kombinaci obojího a ukázalo se to jako dobrý krok. Celkově dobré rozmyšlení celé knihovny byl dobrý krok. Datové struktury se pak implementovali o dost rychleji a snáze. Díky dobrým datovým strukturám pak byla snazší i implementace algoritmů. Aplikování prvků knihovny do hry Pacman bylo také dobré. Algoritmy fungovali a počítali správně tak jak měli. Problém byl s přepsáním určitých prvků pohybu příšer a hráče. Ukázalo se, že objektový návrh příšery v bludišti předělaný na pacmana nebyl nejpovedenější. Zároveň jsem si také uvědomil v čem jsem se od té doby zlepšil v používání jazyka C#.

## Závěr

Posledním slovem bych jen řekl, že pro aplikaci grafové knihovny bych příště použil spíše nějaký typ vizualizace průchodu grafem u jednotlivých algoritmů. Aplikace na hru pacman je sice v jistém slova smyslu užitečná, ale není na ní úplně vidět rozdíl mezi spočítáním nejkratší cesty průchodem do šířky, nebo Dijkstrovým algoritmem.