Mendelova univerzita v Brně
Provozně ekonomická fakulta

Názorná demonstrace principů a funkčnosti vybraných algoritmů a metod umělé inteligence: A* a jeho alternativy

Diplomová práce

Vedoucí práce:
doc. Ing. Jan Žižka, CSc.

Bc. Jan Hauschwitz

Brno 2017

Tímto bych rád poděkoval mému vedoucímu práce panu doc. Ing. Janu Žižkovi, CSc. za odborné vedení, vstřícný přístup a mnoho cenných rad při zpracování závěrečné práce.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Názorná demonstrace principů a funkčnosti vybraných algoritmů a metod umělé inteligence: A* a jeho alternativy**
vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 19. května 2017 _____

Abstract

Hauschwitz, J. This thesis focuses on the project and realization of the implementation of the algorithm A*. The created result is used in animated graphic and interactive demonstration of the qualities A*, The other algorithms and the comparison of their functioning, results and efficacy are also part of the thesis. In the end, the results are evaluated.

Keywords

Artifical intelligence, algorithm, A*, application.

Abstrakt

Hauschwitz, J. Diplomová práce se zabývá návrhem a realizací implementace algoritmu A*. Vytvořený výsledek je použit v názorné animované grafické a uživatelsky interaktivní demonstraci vlastností A*. Součástí implementace jsou i vybrané další prohledávací algoritmy a vzájemné srovnání jejich průběžné činnosti, výsledků a efektivity. V závěru práce jsou vyhodnoceny dosažené výsledky.

Klíčová slova

Umělá inteligence, algoritmus, A*, aplikace.

Obsah

1 Úvod a cíl práce

1.1 Úvod

V poslední letech se počítače staly běžnou součástí našeho každodenního života. Od svého samotného počátku se lidem snaží usnadňovat jejich práci a stále více se stávají také prostředkem zábavy. S jejich masivním využíváním se již běžně setkáváme v řadě oblastí, aniž bychom už nad tím přemýšleli. V současnosti již totiž většina lidí bere využívání počítačů jako samozřejmost.

Vývoj výpočetních systémů odborná literatura rozděluje do několika etap běžně již označovaných jako počítačové generace. První generace počítačů započala s objevem elektronky, druhá pak nastoupila s vynálezem tranzistoru. Integrované obvody jsou charakteristické pro počítače třetí generace a čtvrtá generace se zapsala do povědomí lidí svými mikroprocesory a osobními počítači. Hudbou budoucnosti je i přes pokusy japonských vědců zatím pátá generace počítačů. Tato generace si klade za cíl rozšiřování umělé inteligence. Umělá inteligence má být nadále schopná úlohy nejen řešit, ale také nalézat algoritmy zadávaných řešení. I když v současné době umělá inteligence proniká do stále více oblastí lidské činnosti, existuje ještě mnoho problémů, které na tomto poli zbývá řešit.

1.2 Cíl práce

Cílem této práce je návrh a realizace implementace algoritmu A*. Vytvořený výsledek je použit v názorné animované grafické a uživatelsky interaktivní demonstraci vlastností A*. Součástí implementace jsou, v souladu se zadáním práce, i vybrané další prohledávací algoritmy a vzájemné srovnání jejich průběžné činnosti, výsledků a efektivity. V rámci práce bylo dále navrženo několik vhodných typických demonstračních problémů pro zobrazení vývoje jejich řešení za pomoci implementovaných algoritmů. Součástí systémů je nápověda a vyhodnocení výsledků realizovaného úkolu včetně vlastností jednotlivých implementovaných algoritmů ve vztahu k navrženým úlohám pro řešení. Obsahem je také návrh dalšího možného rozšíření vytvořeného demonstračního systému z hlediska uživatelského rozhraní, včetně připojení nových algoritmů a charakteristických úloh pro řešení umělou inteligencí.

2 Teoretická část

2.1 Inteligence

Inteligence je vrozená schopnost některých živých organismů ke správnému jednání při řešení různých situací, úkolů nebo vykonávání zadané práce. Lidé se navzájem liší svým intelektovým výkonem a člověk, který dosahuje vyšších rozumových schopností, je ve společnosti považován za inteligentního. Novými zkušenostmi a vystavováním se neznámým situacím a podmínkám lze inteligenci rozvíjet. Vyšší inteligence je předpokladem osobního úspěchu, neboť člověk svými správnými reakcemi na stále se měnící podmínky dosahuje svých cílů, které zároveň může využívat ve svůj prospěch.

Nástup výpočetní techniky a její rozvoj po druhé světové válce vedly ke snaze napodobit schopnosti člověka. Lidé se tak čím dále více pokouší vtisknout inteligenci i věcem. Některé známky, které s inteligentními bytostmi běžně spojujeme, mohou proto v současné době vykazovat i počítače nebo organismy, tedy subjekty postrádající mozek a centrální nervovou soustavu.

Kognitivní vědy nebo informatika formulují inteligentní systém obecně, bez nutného vztahu k živým organismům. Aby byl systém považován za inteligentní, musí být schopný reagovat na měnící se podmínky prostředí. Předpokladem také je, že si musí umět zajistit vlastní schopnost přežití a schopnost reprodukce. Každý inteligentní systém musí být též orientovaný na cíl, mít snahu stanovený cíl dosáhnout a být schopný se učit.

Pojem inteligence nebyl dosud přesně vymezen. Ve vztahu k rozumu ho poprvé použil Francis Galton, bratranec Charlese Darwina, na počátku 18. století. Teorie Francise Galtona byla založena na předpokladu, že inteligence je určena přesností smyslů a vnímání. O co nejvýstižnější definici inteligence se v minulosti pokoušela celá řada uznávaných světových psychologů jako např. William Stern, Joy Paul Guilford nebo David Wechsler. Inteligence je však jev natolik složitý, že ani v současnosti nemáme dokonalou definici, která by tento pojem kompletně popsala.

2.2 Umělá inteligence

Nejsme-li dosud schopni přesně definovat inteligenci, tím spíše je složitější vymezit obsah technické umělé inteligence. „Umělá inteligence je obor informatiky zabývající se tvorbou strojů vykazujících známky inteligentního chování“ (Wikipedia, 2017). Činnost těchto mechanismů směřuje k napodobování chování člověka. I ve vztahu k umělé inteligenci existuje řada definic snažící se o co nejpřesnější zachycení tohoto pojmu. Přesto však výstižná definice zatím neexistuje. Mezi nejdůležitější definice však řadíme definici, kterou vytvořil Marvin Minsky v roce 1967: „Umělá inteligence je věda o vytváření strojů nebo systémů, které budou při řešení určitého úkolu užívat takového postupu, který, kdyby ho dělal člověk, bychom považovali za projev jeho inteligence“ (Marvin Minsky, 1967). Nelze opomenout také definici Elanie Richové: „Umělá inteligence se zabývá tím, jak počítačově řešit úlohy, které zatím zvládají lidé lépe” (Elania Richová, 1991). Zajímavá je i definice Zdeňka Kotka: „Umělá inteligence je vlastnost člověkem uměle vytvořených systémů vyznačujících se schopností rozpoznávat předměty, jevy a situace, analyzovat vztahy mezi nimi, a tak vytvářet vnitřní modely světa, ve kterých tyto systémy existují, a na tomto

základě pak přijímat účelná rozhodnutí a objevovat nové zákonitosti mezi různými modely nebo jejich skupinami" (Zdeněk Kotek, 1993). Počátky umělé inteligence jsou úzce spjaty se jmény Alana Turinga a Johna von Neumana, které můžeme považovat za zakladatele moderní informatiky. Jejich přičiněním odborná veřejnost změnila pohled na počítače a téma umělé inteligence se tak dostalo do popředí zájmů. Objevily se velmi ambiciózní předpovědi rozvoje umělé inteligence. V průběhu roku 1956 v rámci konání konference odborníků zabývajících se tématem umělé inteligence bylo „předpovězeno, že v roce 1970 počítač:

- bude velmistrem v šachu,
- odhalí nové významné matematické teorémy,
- porozumí přirozenému jazyku a bude sloužit jako překladatel,
- bude schopen komponovat hudbu na úrovni klasiků" (Mařík a kol., 2013, str. 19).

Protože se však tato předpověď nenaplnila, následovala krize umělé inteligence vystřídaná obdobím objevů pro umělou inteligenci velmi důležitých. Jednalo se například o objev Franka Frosenblatta, který vyvinul perceptron jako nejjednodušší model nervové buňky – neuronu. Velmi významným se stal i vynález programovacího jazyka LISP určeného pro matematické výpočty Johna McCartyho, stejně jako jazyk PROLOG navržený Alainem Colmerauerem. „Tento jazyk je dodnes upřednostňován jako jazyk pro umělou inteligenci v Evropě a v Japonsku" (Mařík a kol., 2013).

Výzkumem umělé inteligence, který je vysoce odborný a specializovaný, se za jeho existenci zabývalo velké množství renomovaných vědců. I přesto má umělá inteligence stále své slabiny. Hlavními problémy u umělé inteligence jsou uvažování, plánování, učení, zpracovávání přirozeného jazyka, vnímání a schopnost se pohybovat či manipulovat s předměty. Dosažení umělé inteligence srovnatelné s lidskou je stále jedním z hlavních cílů oboru (Wikipedia, 2017). Vytvořit obecnou umělou inteligenci srovnatelnou s inteligencí lidskou se však ukázalo značně obtížné. V posledních padesáti letech byla vyvinuta řada postupů, které dosahují jednotlivých úspěchů v individuálních problémech. Jedním z výpočetních modelů používaným v umělé inteligenci, který v této souvislosti nelze opomenout, je neuronová síť. „Jejím vzorem je chování odpovídajících biologických struktur. Umělá neuronová síť je struktura určená pro distribuované paralelní zpracování dat. Skládá se z umělých (nebo také formálních) neuronů, jejichž předobrazem je biologický neuron. Neurony jsou vzájemně propojeny a navzájem si předávají signály a transformují je pomocí určitých přenosových funkcí. Neuron má libovolný počet vstupů, ale pouze jeden výstup. Neuronové sítě se používají mimo jiné i pro rozpoznávání a kompresi obrazů nebo zvuků, předvídání vývoje časových řad (např. burzovních indexů), někdy dokonce k filtrování spamu. V lékařství slouží k prohlubování znalostí o fungování nervových soustav živých organismů. Například perceptonová síť vznikla původně jako simulace fyziologického modelu rozpoznávání vzorů na sítnici lidského oka" (Wikipedia, 2017). Neuronová síť se chová tak, jak si lidé myslí, že se chová lidský mozek. Vzájemně propojené elektronické neurony reagují na podněty a přenášejí signály po síti dál. Největší předností neuronové sítě je schopnost učit se. V tomto případě však pojem učit se představuje způsobilost algoritmu zapamatovat si kombinace, jejichž užitím bylo dosaženo požadovaného výstupu. U nových vstupů je zase jeho úkolem obracet se na svou paměť a na základě již získaných zkušeností odhadovat nový výsledek. Neuronová síť se využívá zejména pro řešení problémů, u kterých klasické programování nevedlo ke správným výsledkům. Často uváděným příkladem je tak určování objektů na obrázku nebo

rozpoznávání tváří. Obecným programátorským postupem, který místo toho, aby sepsal pro řešení úkolu konkrétní algoritmus, hledá cestu evolučními metodami, je genetické programování. Jde o metodu, pomocí které vlastně počítače programují počítače. U genetického programování jsou opět používány principy známé z biologie, jako je např. přirozený výběr, křížení nebo mutace. S nadsázkou se dá říci, že pomocí evolučních technik vědci navrhují, jak dělat nějakou činnost, aby byla co nejefektivnější. Z hlediska výzkumu umělé inteligence se důležitými staly také pojmy jako expertní systém, prohledávání stavového prostoru, dobývání znalostí, nebo strojové učení (Wikipedia, 2017).

2.3 Expertní systémy

Problém získávání, prezentace a využívání speciálních expertních znalostí je úkolem počítačových programů, které jsou známy pod názvem expertní systémy. Expertní systémy jsou asi nejznámější teorií umělé inteligence. Jde o „programy simulující rozhodovací činnost specialistů (expertů) při řešení složitých úloh rozhodování a využívající vhodně zakódovaných speciálních znalostí převzatých od expertů s cílem dosahovat ve zvolené problémové oblasti kvality rozhodování na úrovni experta" (Mařík a kol., str. 21). Expertní systémy jsou navrženy tak, aby mohly zpracovávat nenumerické a neurčité informace a řešit tak úlohy, které nejsou řešitelné tradičními algoritmickými postupy. „Jedná se o systém, který nevyužívá znalostí nabytých vlastní činností, ale využívá znalostí (myšlení a rozhodování) špičkových odborníků v dané oblasti. Avšak bez lidských omylů. Cílem činnosti expertního systému je dosáhnout co nejlepší odezvy na reálná data" (Ikaros, 1999). Expertní systémy se liší od klasicky naprogramovaných počítačových systémů, u kterých je činnost počítače dána přesnou posloupností úkonů, algoritmů nebo definicí objektů a metodami jejich zpracování. Znalostní systémy nemají přesně definované postupy, jejich hlavním vodítkem je výsledek. A výsledkem je pomoc experta spočívající v nalezení hledaného závěru, rady, doporučení a nebo požadavku na rozšíření databáze znalostí. K typickým znakům expertních systémů patří oddělení znalostních bází od vlastního vyhodnocovacího programu. Jedná se o zásadní odlišnost těchto systémů ve srovnání s klasickými programy. Dalším znakem je schopnost systému rozhodovat i v případech neurčitostí a nejasností a dokládat dosažené výsledky uživateli náležitými podklady a vysvětleními (Mendelu, 2017).

2.3.1 Úkoly a vlastnosti expertních systémů

Expertní systémy plní svým uživatelům celou řadu funkcí. Pomáhají jim s vyhledáváním informací potřebných pro řešení jejich problémů, takto získané informace systematicky uspořádávají a uživatelům vysvětlují. Od expertních systémů se ve srovnání se znalostními systémy vyžaduje právě i schopnost svá rozhodnutí a řešení problémů uživatelům náležitě zdůvodnit. Expertní systémy také shromažďují a poskytují specializované expertní znalosti, přičemž tyto znalosti mohou být zároveň zpřístupněny více uživatelům současně a to na kterémkoliv místě na světě. Znalosti jsou trvale uchovávány a mohou být průběžně doplňovány. Neméně důležitou funkcí těchto systémů je také zabezpečování řešení některých stereotypních úloh. Uživatelé tak mají umožnost věnovat se složitějším stránkám řešení úkolu.

Expertní systémy jsou určeny k řešení složitých problémů. Jsou charakteristické dostupností svých expertiz a sníženými náklady na jejich provedení. Expertizy a všechny znalosti odborníků přitom mají trvalý charakter a dají se opakovaně využívat. Za výhodu expertních systémů je považována skutečnost, že mohou zároveň sloužit jako trénovací nástroj pro začátečníky. I přes velkou řadu výhod mají expertní systémy i některé své slabiny. Nevýhodou těchto systémů je nebezpečí selhání expertního systému při jeho užití za změněných podmínek a také neschopnost systému rozpoznat hranice své použitelnosti (Mendelu, 2017).

2.3.2 Složení expertních systémů.

Expertní systém se skládá ze tří relativně na sobě nezávislých základních součástí, kterými jsou báze znalostí, báze dat a řídící mechanismus pro odvozování závěrů (Mendelu, 2017).

Báze znalostí

Počátky vzniku báze znalostí se datují od 60. a 70. let minulého století a tento termín je znám a začal se užívat právě ve spojení s rozvojem umělé inteligence. Sestavování báze znalostí představuje dlouhodobou záležitost. Bázi znalostí totiž tvoří veškeré znalosti experta potřebné pro řešení daného problému. Bázi znalostí lze přirovnat ke zdroji informací, ve kterém jsou shromážděny informace všech úrovní od základních informací až po úzce specializované údaje. Samostatnou kapitolou jsou znalosti soukromé, označované také jako znalosti nejisté. Nejnovější expertní systémy současně využívají i větší počet bází znalostí. Bylo prokázáno, že úspěšnost expertního systému závisí právě na kvalitě báze znalostí. „Podstatnou součástí expertních znalostí jsou znalosti nepřesné, neurčité, vágní. Proto je teorie zpracování neurčitosti ve znalostech i v datech velmi důležitou partií teorie umělé inteligence" (Mařík a kol., 2013, str. 21). Nejistými znalostmi označujeme ty znalosti, které expert postupně získal v průběhu své praxe. Jedná se o vědecky neprokázané znalosti, na jejichž základě se ani nemusí dospět k nalezení správného řešení. Běžnou součástí expertních systémů se v současné době staly elektronické slovníky, encyklopedie a katalogy.

Báze dat

Údaje poskytnuté uživatelem k řešenému případu jsou zaznamenány v bázi dat. Báze dat je zpravidla uložena na pevných discích nebo jiných externích paměťových zařízeních a odpovídá množině dat klasického programu (Mařík a kol., 2013).

Řídící mechanismus

Řídící mechanismus představuje vlastní program, ve kterém jsou shromážděny myšlenkové, usuzovací a odvozovací algoritmy a který na základě znalostí upravuje bázi dat takovým způsobem, aby byla schopna najít požadované výsledky. Tento mechanizmus umožňuje odvozování nových poznatků, prohledávání báze znalostí a zpracování neurčitostí. Je schopen dedukce, indukce, abdukce i heuristiky. Řídící mechanismus má za úkol vyhodnocovat stav, který je ovlivněn expertními znalostmi uloženými v bázi znalostí a informacemi získanými od uživatele zaznamenanými v bázi dat. Na základě toho expertní systém samostatně rozhoduje o tom, zda je již schopen poskytnout expertní radu

nebo zda je pro řešení úkolu potřebné získat ještě nějaké další informace. Úkolem expertního systému je tedy co nejlépe se dotazovat na informace k řešené problematice, analyzovat odpovědi uživatele a na základě těchto rozborů a obecných znalostí uložených v bázi znalostí navrhnout uživateli konkrétní řešení případu. „Řídící mechanismus určuje, jak a v jakém pořadí aplikovat pravidla na bázi dat. Principiálně rozlišujeme přímý režim řízení, kdy při aplikaci produkčních pravidel postupujeme ve směru od počátečního stavu k některému ze stavů cílových, a zpětný režim řízení, kdy se vychází od cíle (cílů) ve směru počátečního stavu. V prvním případě se také často hovoří o strategii řízené daty (data-driven strategy), v druhém pak o strategii řízené cílem (goal-driven strategy). Oba režimy řízení lze vhodným způsobem kombinovat“ (Mařík a kol., 2013, str. 35).

Využívání expertního systému u pravidelných nebo často řešených problémů podobného typu je výhodnější než užití experta lidského. Jde především o nižší dlouhodobou nákladnost a lepší časovou efektivitu tohoto systému ve srovnání s lidským expertem. Expertní systém totiž na rozdíl od člověka může pracovat i 24 hodin denně. Nebývá unavený a nepotřebuje přestávky, odpočinek ani dovolenou (Mařík a kol., 2013).

2.3.3 Komerční využívání expertního systémů

K prvnímu komerčnímu využívání expertního systému došlo v roce 1980. Jednalo se o systém pro konfiguraci počítačů a vzhledem k tomu, že tento expertní systém od svého počátku přináší obrovské finanční úspory, došlo k hektickému rozvoji expertních systémů i v mnoha dalších oblastech. Jednalo se především o sféru zdravotnictví, dopravy a průmyslu. Velká poptávka po expertních systémech však vznikla i u pracovníků zabývajících se vědou, veřejnou bezpečností, obchodem nebo bankovnictvím. Předpokládá se, že se technologie umělé inteligence v nejbližší době zásadním způsobem projeví i v oblasti podnikových informačních systémů. V uplynulých 20 letech dokázala umělá inteligence porazit člověka i na poli her. Došlo k tomu postupně a jednalo se například o šachy nebo dámu. Na konci ledna 2016 však počítač překonal člověka i ve staré strategické hře Go. Jednalo se o velký úspěch, neboť tato hra byla donedávna považována pouze za doménu lidskou, protože se jedná o jednu z nejsložitějších her světa. Hlavním nástrojem vývojářů daného softwaru byly právě tzv. hluboké neuronové sítě. V prosinci a listopadu minulého roku bylo dokonce zaznamenáno vítězství počítače nad týmem složených ze 33 specializovaných hráčů pokeru. Jeden z tvůrců algoritmu předmětného počítače pro tisk uvedl: „Algoritmus DeepStacku je přelomový, protože se nám podařilo přenést myšlenky, které byly klíčové v hrách s úplnou informací, do světa her s neúplnou informací. Doposud nebylo jasné, zda je podobný přístup vůbec možný” (EuroZprávy.cz, 2017). Tento úspěch je podle vědců velmi důležitý, dá se nazvat i průlomovým, právě z hlediska jeho praktického uplatnění. Lidé se totiž běžně ve svém životě musí rozhodovat právě na základě neúplných informací. Zveřejnění tohoto vítězství proto okamžitě vyvolalo poptávku po algoritmu daného počítače, který by tak mohl být využíván pro potřeby řešení komplikovaných problémů například v rámci obchodního vyjednávání, vojenské strategie nebo třeba bezpečnosti v kyberprostoru (Mařík a kol., 2013).

Umělá inteligence je považována za zvláštní, netypickou vědní disciplínu, jejíž hranice nejsou jasně určeny a neustále se mění. Na počátku se výzkum umělé inteligence věnoval „vývoji jednotlivých technik, modelů či algoritmů, později (v 70. a 80. letech) je patrný posun směrem ke tvorbě opakovaně použitelných a samostatných (stand-alone) systémů,

které obvykle využívaly kombinaci některých základních technik (prohledávání stavového prostoru, modely pro práci s neurčitou informací atd.). Základním vývojovým směrem pro 90. léta je integrace nejrůznorodějších softwarových systémů s podporou filozofie a metodologie umělé inteligence a znalostního inženýrství (jako příslušné inženýrské disciplíny)" (Mařík a kol., 2013, str. 29).

Dá se tedy předpokládat, že v několika následujících letech připraví čtvrtá průmyslová revoluce, nazývaná někdy také revolucí robotickou, o práci velké množství lidí, kteří budou nahrazeni stroji a ve větší míře právě počítačovými programy. Používání programů umělé inteligence tak může výrazně a všestranně zefektivnit proces rozhodování téměř v každém oboru lidské činnosti (Mařík a kol., 2013). Představme si tedy důležité pojmy, které jsou klíčové pro tuto práci a které jsou s umělou inteligencí pevně spojeny.

2.4 Algoritmus

Algoritmem rozumíme přesnou a jednoznačnou sekvenci příkazů pro řešení daného typu úlohy. Algoritmy byly známé už dávno před tím, než člověk sestavil první počítače. Jednoduché i složitější návody a postupy řešení totiž vždy byly a neustále jsou součástí každodenního běžného života. Algoritmy však jsou využívány i při řešení úkolů v nejrůznějších vědních oborech, nejčastěji se s nimi setkáváme právě při programování. Algoritmus představuje teoretický princip řešení problému, jde o popis elementární operace, která má být vykonána a na základě které je realizováno požadované řešení daného problému. Algoritmus udává, za jakých podmínek lze převádět vstupní údaje na údaje výstupní.

Základními prvky algoritmu jsou sekvence, větvení a cyklus. Sekvencemi rozumíme příkazy k provádění, které následují po sobě jeden za druhým v přesně stanoveném pořadí, a větvením schopnost vybrat si z několika možných příkazů v závislosti na nějaké určité podmínce. Cyklus potom znázorňuje opakování jedné či více instrukcí do doby, než nenastanou podmínky pro ukončení příkazu.

Algoritmus lze vyjádřit různými způsoby. Cílem je, aby ten, kdo bude podle algoritmu postupovat, ať již to bude člověk nebo počítač, vyjádření rozuměl. Algoritmus tak lze zachytit popisem přirozeného jazyka, strukturovaným jazykem, programovacím jazykem nebo graficky. Strukturovaný jazyk přestavuje v podstatě přirozený jazyk, při jehož užití však platí určitá předem dohodnutá omezující pravidla. Programovací jazyk má již jasně danou množinu povolených slov s jasně daným významem. Pokud se jedná o grafický zápis algoritmu, tento je velmi názorný, ale nehodí se pro vyjádření složitých algoritmů, které je vhodné rozdělit na dílčí části. Grafické znázornění algoritmu nazýváme vývojovým diagramem (Mendelu, 2017).

2.4.1 Vlastnosti algoritmu

Algoritmem se rozumí pouze takové postupy, které splňují přesně stanovené požadavky. Tyto požadavky představují vlastnosti algoritmu. Rozlišujeme pět základních vlastností algoritmu:

1. Elementárnost algoritmu

Tato vlastnost je založena na skutečnosti, že algoritmus se skládá z konečného počtu jednoduchých, tzv. elementárních, kroků.

2. Konečnost (finitnost) algoritmu

Každý algoritmus musí skončit v konečném počtu kroků, který ale může být libovolně velký.

3. Obecnost (univerzálnost, hromadnost) algoritmu

Algoritmus neřeší jeden konkrétní problém, ale obecnou třídu obdobných problémů, z čehož vyplývá široká množina možných vstupů.

4. Determinovanost algoritmu

Každý krok uvedený v algoritmu musí být jednoznačně a přesně definován. V každé situaci tak musí být naprosto zřejmé, co a jak se má provést a jak má provádění algoritmu dále pokračovat. Vzhledem k tomu, že běžný jazyk obvykle neposkytuje naprostou přesnost a jednoznačnost vyjadřování, neboť některé výrazy lze vyložit i více významy, byly pro zápisy algoritmů navrženy programovací jazyky, ve kterých má každý příkaz jasně definovaný smysl.

5. Výstup (resultativnost) algoritmu

Algoritmus musí mít alespoň jeden výstup. Musí tedy vytvářet veličinu, která je v požadovaném vztahu k zadaným vstupům, a tím představuje odpověď na řešený problém. Algoritmus tak vede od zpracování hodnot k výstupu (Mendelu, 2017).

2.5 Stavový prostor

Mezi základní úkoly umělé inteligence patří metody pro strojové řešení úloh. Vzhledem k tomu, že není přijatelné, aby stroj hledal řešení každého jednotlivého úkolu postupným testováním všech možností, vznikla proto potřeba hledání nějak efektivně řídit. Inteligentní systémy jsou proto nadány schopností vytvářet si vnitřní strojový model prostředí (světa) a s tímto pracovat. Za situace, kdy je určen počáteční a koncový model prostředí, systém umělé inteligence vyhledává vhodnou posloupnost akcí, v rámci umělé inteligence nazvanou plánem, na základě kterého se z výchozího bodu lze dostat do cíle. „Každému modelu odpovídá jistý stav prostředí, množina všech stavů představuje stavový prostor" (Mařík a kol., 2013, str. 33). Cílový stav nemusí být pouze jeden a v rámci řešení úlohy může být známá pouze představa o tom, jak by měl cílový stav vypadat. „Stavovým prostorem se v informatice rozumí konfigurace diskrétních stavů sloužící jako výpočetní model. Formálně může být stavový prostor definován jako čtveřice [N, A, S, G], kde:

N je množina stavů,
A je množina přechodů mezi stavy,
S je neprázdná podmnožina N obsahující počáteční stavy,

G je neprázdná podmnožina N obsahující cílové stavy.

Na procházení stavového prostoru je založena metoda řešení úloh zvaná Prohledávání stavového prostoru" (Wikipedia, 2017).

Je zaznamenána řada příkladů formalizace úloh právě pomocí stavového prostoru. Jedná se např. o popis hry lišák, bludiště, přelévání vody nebo šachů.

2.6 Prohledávání stavového prostoru

Jednou ze základních technik umělé inteligence je prohledávání stavového prostoru. Stavový prostor poskytuje návod pro výběr pravidel z konfliktní množiny pravidel a to v rámci každého jednotlivého úkonu vykonaného v rámci prohledávání stavového prostoru. Jedná se o postup řešení úloh umělé inteligence. Jeho princip spočívá ve vhodném procházení stavů řešené domény za účelem nalezení požadovaného stavu. „Řídící mechanismus realizuje řídící strategii, je to tedy algoritmus, poskytující návod pro výběr pravidel z konfliktní množiny pravidel v každém kroku prohledávání stavového prostoru" (Mařík a kol., 2013, str. 38). Aby byla řídící strategie úspěšná, musí vést k prohledávání a být systematická. „Prohledávání lze omezit využitím znalostí o řešeném problému. Tyto znalosti mají někdy charakter empirický, mohou to být neexaktní poznatky, o nichž víme, že jsou „často" užitečné při řešení, přičemž mnohdy ani nezaručují, že nalezneme řešení" (Mařík a kol., 2013, str. 38). Uvedené znalosti se nazývají heuristikami a jsou využívány především v případech absence dokonalého algoritmu. Heuristiky se vyskytují v různých podobách, podstatně však přispívají ke snadnějšímu a rychlejšímu nalezení řešení případu, neboť jejich využitím je prohledávána menší část stavového prostoru, postupuje se přímočařeji k cíli a způsob řešení se tak jeví jako „inteligentnější" (Mařík a kol., 2013).

2.6.1 Metody prohledávání

Protože stavový prostor může být velmi rozsáhlý nebo obecně nekonečný, není často v silách programu příslušného počítače ve stanoveném termínu přezkoumat všechny možné stavy a přechody mezi nimi. Systematické prohledávání stavového prostoru, kdy se prohledává i značná část stavového prostoru nevedoucí k cíli, je totiž velmi neefektivní. V těchto případech proto dochází k využívání nejrůznějších způsobů pro prohledávání stavového prostoru, které se snaží o výběr co nejlepší varianty ze všech možných existujících variant, snaží se tedy tento proces optimalizovat. Z uvedeného důvodu byly vytvořeny různé metody prohledávání stavového prostoru s různými výhodami a nevýhodami, přičemž nelze s určitostí stanovit, jestli je některá metoda výrazně lepší ve srovnání s jinou metodou. Pro výběr konkrétní metody je totiž zapotřebí znát dobře nejen povahu řešené úlohy, požadavky na řešení ale i dostupné prostředky. Základní metody prohledávání stavového prostoru existují v řadě úprav. Obecně ale můžeme metody pro prohledávání stavového prostoru dělit na metody neinformované a informované a to na základě skutečností, zda využívají znalosti o dané úloze nebo ne. Úkolem metod prohledávání je snaha zabránit hledacím algoritmům v bloudění (Mařík a kol., 2013).

Pro potřebu aplikace bylo nutné důkladné nastudování prohledávacích algoritmů a to především algoritmů prohledávání do šířky, do hloubky, hladového vyhledávání a algoritmu A*.

2.6.2 Způsob hodnocení metod

V současné době jsou zaznamenány tři základní způsoby hodnocení metod prohledávání stavového prostoru. V prvé řadě se jedná o metodu, která je založená na časové složitosti. V tomto případě je tedy sledován čas a to ve své minimální, maximální a průměrné hodnotě. Jedná se tedy o dobu, která je potřebná pro vyřešení konkrétní úlohy. Druhým hlediskem pro hodnocení zvolené metody je paměťová náročnost. Zde se kontroluje množství operační paměti, která je nutná k vyřešení zadaného úkolu. Pro hodnocení metody prohledávání stavového prostoru je však důležitá také kvalita získaných výsledků, tedy skutečnost, zda zvolená metoda nalezne řešení vždy, když řešení daného problému existuje, a zda nalezené řešení je optimální (Wikipedia, 2017).

2.7 Neinformované metody

Neinformované metody prohledávání jsou ve své podstatě metody jednoduché nevyžadující výpočet žádné složité ohodnocovací funkce. Z uvedeného důvodu jsou používány pouze v triviálních případech. Tyto metody procházejí stavový prostor bez jakékoliv dopředu známé informace, některá literatura hovoří o procházení stavového prostoru „hrubou silou". Jde o situaci, kdy je naslepo prohledáván jeden stav za druhým, tedy zbytečně příliš velká část stavového prostoru. Tyto metody žádným způsobem nehodnotí, zda je aktuální stav či cesta k němu výhodná, a nesnaží se ani odhadovat, zda je cíli bližší či vzdálenější. Z hlediska časového a prostorového tak jde o metody vysoce neefektivní a metody, které nevyhledávají nejlépe vyhovující řešení. „Metody neinformované dělíme z hlediska pořadí, ve kterém jsou uzly expandovány, na slepé prohledávání do hloubky a slepé prohledávání do šířky. Hloubkou uzlu ve stromu řešení rozumíme počet hran na cestě od počátečního uzlu k danému uzlu. Při slepém prohledávání do šířky (breadth-first search) se nejdříve expanduje uzel s minimální hloubkou" (Mařík a kol., 2013, str. 38). Při slepém prohledávání do hloubky (depth-first search) se naopak přednostně expanduje uzel s největší hloubkou. V současné době však již existují i metody prohledávání do hloubky s omezením (depth-limited search), iterativní prohledávání do hloubky (iterative deepening search) a obousměrné prohledávání (bidirectional search).

2.7.1 Prohledávání do šířky

Prohledávání do šířky, anglicky breadth-first search, je algoritmus určený k prohledávání grafu. Dokáže projít všechny uzly, které jsou dostupné ze startovního. Algoritmus prochází graf po vrstvách.

Výhodou prohledávání grafu do šířky je skutečnost, že kromě samotného řešení nalezne zároveň i nejkratší cestu, která k řešení vede. Tento algoritmus nepoužívá při svém prohledávání žádnou heuristickou analýzu, pouze postupně prochází všechny uzly a také všechny jejich následovníky. Přitom si zaznamenává předchůdce jednotlivých uzlů a tím následně vytváří nejkratší cesty k jednotlivým uzlům z uzlu startovního. Ve srovnání s algoritmem prohledávání do hloubky prohledávání do šířky klade také velký nárok na prostorovou složitost. Proto je velmi nepraktický pro řešení rozsáhlejších problémů (Russell & Norvig, c2014).

Existuje více způsobů implementace algoritmu prohledávání do šířky, v aplikaci je použit následně popsaný postup.

Popis algoritmu

Algoritmus prohledávání do šířky nejčastěji používá pro ukládání uzlů následníků prozkoumaných uzlů frontu (dále OPENED) a pole pro ukládání již expandovaných uzlů (dále CLOSED).

V prvním kroku algoritmu se vloží do OPENED počáteční uzel. Potom následuje cyklus, který se opakuje do té doby, dokud není OPENED prázdná nebo dokud není nalezen výsledek. Prvním krokem cyklu je tedy vyjmutí uzlu (dále U) z fronty a přiřazení U do CLOSED. Následuje kontrola výsledku. V případě, že U je cílovým uzlem, je prohledávání ukončeno a cílový uzel je vrácen. Pokud se však o cílový uzel nejedná, dostaneme se k dalšímu kroku, při kterém do OPENED postupně vkládáme všechny následníky uzlu U, které nejsou v CLOSED.

Implementace algoritmu, která je použita ve vytvořené aplikaci, je uvedena v příloze.

Ukázka prohledávání jednoduchého stromového grafu

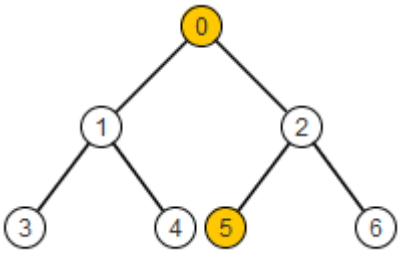
Počáteční stav:

Uzel 0 je počátek.

Uzel 5 je cílový.

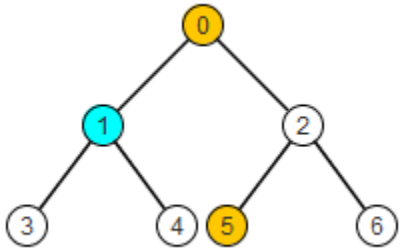
Krok 1

V prvním kroku podle výše uvedeného postupu expandujeme uzel 0 a jeho následníky uzly 1 a 2 přidáme do fronty. Uzly ukládáme v tomto případě zleva.



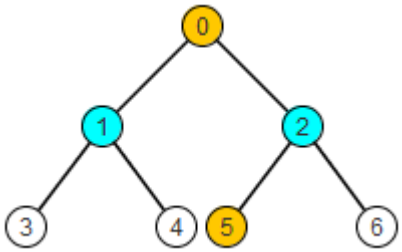
Obr. 1 Prohledávání do šířky krok 1

Krok 2
V kroku dva vybereme první uzel 1 z fronty a jeho následníky uzly 3 a 4 přidáme do fronty.



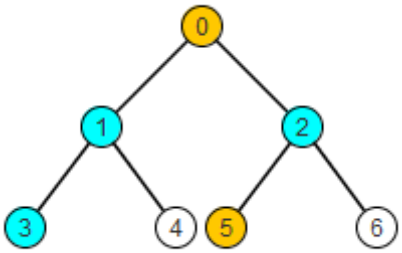
Obr. 2 Prohledávání do šířky krok 2

Krok 3
V kroku tři vybereme první uzel z fronty a jeho následníky uzly 5 a 6 přidáme do fronty.



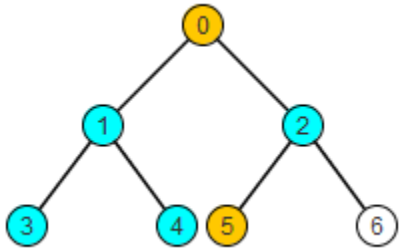
Obr. 3 Prohledávání do šířky krok 3

Krok 4
V kroku čtyři vybereme první uzel 3 z fronty, a protože tento uzel už nemá žádné další následníky do fronty, nic nepřidáváme.



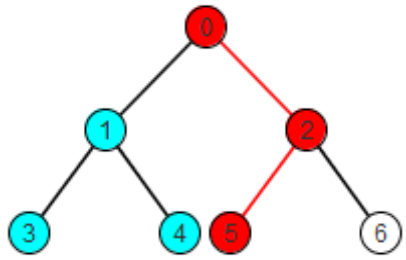
Obr. 4 Prohledávání do šířky krok 4

Krok 5
V kroku pět vybereme první uzel 4 z fronty, a jelikož i tento uzel nemá žádné další následníky do fronty, nic nepřidáváme.



Obr. 5 Prohledávání do šířky krok 5

Krok 6
V kroku šest vybereme první uzel 5 z fronty a zjistíme, že je cílový. Ukončíme proto prohledávání. Výslednou cestu tvoří uzly 0 – 2 – 5.



Obr. 6 Prohledávání do šířky krok 6

Vlastnosti prohledávání do šířky:

Algoritmus je kompletní. Najde vždy první výsledek, který se nachází v nejmenší hloubce.

Tento algoritmus splňuje všechny předpoklady, které jsou na algoritmy kladeny. Jednotlivé kroky jsou jednoduché, jejich počet je konečný. Vytvořený algoritmus do šířky řeší obecnou třídu postupů a každý jeho krok je přesně stanoven. V rámci uvedeného postupu byl nalezen požadovaný výsledek.

Uvedený algoritmus není časově ani paměťově náročný. Použitím tohoto algoritmu bylo dosaženo výsledku.

2.7.2 Prohledávání do hloubky

Prohledávání do hloubky, v anglickém překladu depth-first search, je algoritmus určený k prohledávání grafu. Dokáže projít všechny uzly, které jsou dostupné ze startovního. Algoritmus vždy expanduje uzel, který je nejhlouběji, a vrací se pouze, když nalezne uzel, který už nemá další následníky (Russell & Norvig, c2014).

Jedná se o algoritmus pro procházení grafů do hloubky metodou backtrackingu. Jeho výhodu představují nižší nároky na paměť. V paměti algoritmu prohledávání do hloubky se

totiž uchovávají pouze uzly nacházející se na cestě od počátečního stavu k právě expandovanému stavu. Tento algoritmus vždy expanduje prvního následníka každého vrcholu, pokud jej ještě nenavštívil. Vrcholy určené k procházení ukládá do zásobníku. Pokud narazí na vrchol, ze kterého už nelze dále pokračovat, neboť tento vrchol již nemá žádné následníky nebo všichni následníci už byli navštíveni, vrací se zpět backtrackingem. Backtracking představuje zpětné vyhledávání. Jde o způsob řešení algoritmických problémů založený na metodě pokus – omyl. Tento postup představuje vylepšené hledání řešení hrubou silou. Velké množství potenciálních řešení může být při jeho použití vyloučeno bez přímého vyzkoušení. Algoritmus je úplný, protože vždy najde řešení, které je představováno určitým cílovým vrcholem. Řešení však naleze pouze za předpokladu, že nějaké možné řešení existuje. Tento algoritmus však není optimální, protože nemusí najít nejkratší možnou cestu vedoucí k cíli. Jeho asymptotická složitost je O(|V| + |E|), přičemž V představuje množinu vrcholů a E množinu hran daného grafu (Kott, 2017).

Popis algoritmu

Tento algoritmus používá struktury pro ukládání uzlů následníků prozkoumaných uzlů, nejčastěji zásobník (dále OPENED) a pole pro ukládání již expandovaných uzlů (dále CLOSED).

V prvním kroku je do OPENED uložen počáteční uzel. Potom následuje cyklus, který se opakuje do té doby, dokud není OPENED prázdná nebo dokud není nalezen výsledek. Prvním krokem cyklu je tedy vyjmutí uzlu (dále U) z fronty a přiřazení U do CLOSED. Vždy je prováděna kontrola výsledku zkoumaného U a pokud je U cílovým uzlem, je prohledávání ukončeno a tento cílový uzel je vrácen. Pokud cílový uzel nalezen nebyl, dostaneme se k následujícímu kroku, při kterém postupně vložíme do OPENED všechny následníky uzlu U, které nejsou v CLOSED.

Implementace algoritmu, která je použita ve vytvořené aplikaci, je uvedena v příloze.

Ukázka prohledávání jednoduchého stromového grafu:

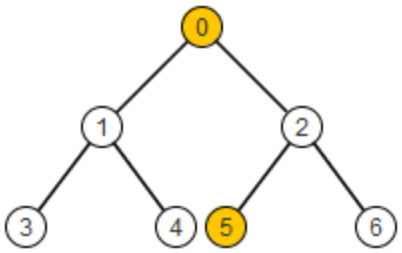
Počáteční stav:

Uzel 0 je počátek.

Uzel 5 je cílový.

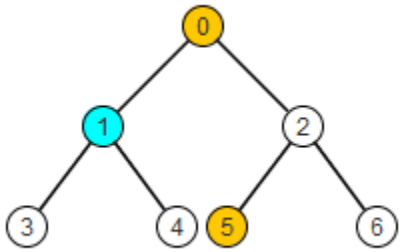
Krok 1

V prvním kroku podle výše uvedeného postupu expandujeme uzel 0 a jeho následníky uzly 1 a 2 přidáme do zásobníku.



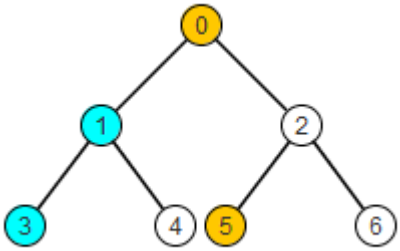
Obr. 7 Prohledávání do hloubky krok 1

Krok 2
V kroku dva vybereme uzel 1 ze zásobníku, expandujeme ho a jeho následníky uzly 3 a 4 přidáme do zásobníku.



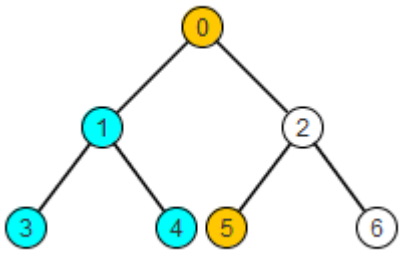
Obr. 8 Prohledávání do hloubky krok 2

Krok 3
V kroku tři vybereme uzel 3 ze zásobníku a expandujeme ho. Zjistíme, že tento uzel již nemá žádné následníky, a proto do zásobníku nic nepřidáváme.



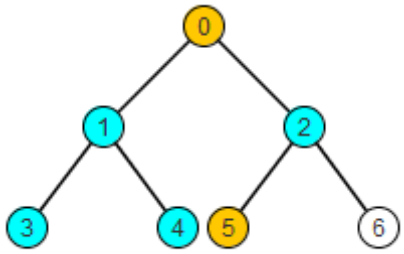
Obr. 9 Prohledávání do hloubky krok 3

Krok 4
V kroku čtyři vybereme uzel 4 ze zásobníku a expandujeme ho. Zjistíme, že také tento uzel už nemá žádné následníky, a proto do zásobníku nic nepřidáváme.



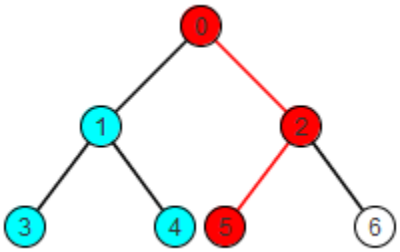
Obr. 10 Prohledávání do hloubky krok 4

Krok 5
V kroku pět vybereme uzel 2 ze zásobníku, expandujeme ho a jeho následníky uzly 5 a 6 přidáme do zásobníku.



Obr. 11 Prohledávání do hloubky krok 5

Krok 6
V kroku šest vybereme první uzel ze zásobníku 5 a zjistíme, že je cílový. Ukončíme proto prohledávání. Výslednou cestou tvoří uzly 0 – 2 – 5.



Obr. 12 Prohledávání do hloubky krok 6

Vlastnosti prohledávání do šířky:

Algoritmus je úplný. Vždy najde řešení, které je představováno určitým cílovým vrcholem.

Také tento algoritmus splňuje všechny předpoklady, které jsou na algoritmy kladeny. Jednotlivé kroky jsou jednoduché a jejich počet je konečný. Řeší obecnou třídu postupů a každý krok tohoto algoritmu je přesně stanoven. V rámci uvedeného postupu byl výsledek nalezen.

Uvedený algoritmus není časově ani paměťově náročný. Použitím tohoto algoritmu bylo dosaženo výsledku.

2.8 Informované metody

Na rozdíl od neinformovaných metod informované metody pracují navíc se znalostmi o stavovém prostoru. Ve srovnání s neinformovanými metodami se proto jedná o metody chytřejší, využívající nějakého vodítka nebo nápovědy k řešenému problému. A právě na základě tohoto se počítač odhodlává k dalším krokům. Neprohledává jako za použití neinformované metody všechny možné stavy, ale využitím nějaké další poskytnuté znalosti

o řešeném problému a použitím heuristické informace dojde k rychlejšímu nalezení řešení s co nejmenšími prostředky. Není totiž zatížen prohledáváním prostoru, který nevede k cíli. „Informované metody prohledávání mají navíc znalosti o stavovém prostoru, které jim umožňují odhadnout, jak daleko se nachází řešení od aktuálního stavu. Tento odhad reprezentuje tzv. heuristická funkce h(n). Čím nižší hodnoty h(n) nabývá, tím spíše povede cesta k řešení skrze stav n. Heuristickou funkci dodává na základě znalostí člověk a informované metody jsou na ní kriticky závislé. Čím lepší heuristika je k dispozici, tím rychleji a s menším zatížením paměti dojde k nalezení řešení" (Wikipedia, 2017).

K informovaným algoritmům patří horolezecký algoritmus, nazývaný také algoritmem gradientním (v anglickém překladu hill-climbing algorithm). Jedná se o jednoduchou a rychlou verzi, která je založena na skutečnosti, že algoritmus expanduje vždy uzel vyhodnocený pomocí hodnotící funkce f jako ten nejlepší. Tento algoritmus však uchovává v paměti právě jen rozvíjený uzel. Jeho další nevýhodou je, že algoritmus nemusí ani nalézt globální maximum, může uvíznout jak v globálním, tak v lokálním extrému a při jeho použití není vyloučeno zacyklení, tedy pohyb po nekonečně dlouhé cestě. Výrazně účinnější prohledávání nabízí algoritmus uspořádaného prohledávání (best-first search), nazývaný také metodou prvního nejlepšího. Tento algoritmus má celou řadu modifikací. Velmi známý je obecný algoritmus, algoritmus paprskového uspořádání, algoritmus A nebo algoritmus A*. Mezi další metody informovaného prohledávání řadíme pravidla se složitějšími předpoklady, metaznalosti, metodu generování a testování, metodu užití analogie a rozklad úlohy na podúlohy (Mařík a kol., 2013).

2.8.1 Heuristická funkce

Heuristická funkce představuje zkusmé řešení problému. Heuristiky se používají v případě, že nejsou k dispozici exaktní algoritmy. Mohou mít různou podobu. Heuristické funkce jsou vždy specifické pro daný problém, vycházejí z konkrétního programu. Heuristiky nalezení optimálního řešení obecně nezaručují, často je však jejich užitím nalezeno v přiměřeném čase dobré řešení. Heuristika totiž využívá znalostí o řešeném problému. Podle toho, jestli jsou heuristické funkce při prohledávání stavového prostoru použity, rozlišujeme algoritmy informované, tedy ty, které heuristiky užívají, a algoritmy neinformované. Heuristické funkce mnohdy vychází z náhody, intuice, analogie či zkušeností. Často je při hledání vhodné heuristické funkce zapotřebí experimentovat. Za nejjednodušší heuristiku je považována metoda pokusu a omylu (Mařík a kol., 1993).

2.8.2 Hledání prvního nejlepšího

Mezi informované metody prohledávání stavového prostoru řadíme, jak bylo shora uvedeno, například metodu hledání prvního nejlepšího (v anglickém překladu best-first search). V tomto případě jde o metodu využívající uspořádání uzlů. Uzly, které se zdají být nejlépe hodnoceny, jsou při užití této metody expandovány jako první. Jde vlastně o verzi gradientního algoritmu rozšířeného o paměť. Best-first-search algoritmy se snaží najít optimální řešení, typicky k tomu používají nějakou odhadovací míru pro nalezení nejlepšího výsledku a snaží se minimalizovat zatížení. Aby bylo hledání přímo zaměřeno na cíl, musí tedy použitá míra v sobě zahrnovat nějaký odhad. Například u hledání optimální ceny se jedná o odhad ceny cesty z nějakého stavu do nejbližšího cílového stavu. V daném případě lze použít nejméně dva základní přístupy k nalezení uvedeného řešení.

Jedním je pokus expandovat uzel nejbližší k cíli a dalším pokus expandovat uzel na nejlevnější cestě k řešení (Russell & Norvig, c2014).

2.8.3 Hladové hledání prvního nejlepšího

Jednou z nejjednodušších metod hledání prvního nejlepšího za současné minimalizace odhadované ceny dosažení cíle je hladové (neboli lačné) vyhledávání minimalizující odhadovanou cenu dosažení cíle. Výhodou lačnosti je rychlé nalezení cíle, i když tento cíl nemusí být vždy optimální. Optimalita by totiž vyžadovala důkladnější analýzu z hlediska uvažování o celé dlouhé vzdálenosti, nikoliv pouze bezprostřední nejlepší výběr. Metoda lačného hledání spočívá právě v tom, že se program vůbec nestará o to, jestli je řešení úkolu ze všech možných řešení to nejlepší. Zohledňuje pouze odhadnutou cenu nejlevnější cesty ze stavu v uzlu n do stavu cílového. Za použití této metody je vždy nejdřív expandován uzel, který se zdá být nejblíže cíli. Pro většinu (reálných) problémů lze náklady na dosažení cíle z nějakého okamžitého stavu jen odhadnout, nelze je přesně stanovit. Funkci, která počítá takové odhady nákladů, nazýváme funkcí heuristickou a běžně ji označujeme jako funkci h. Jako každá metoda prohledávání stavového prostoru má lačné hledání své přednosti a zároveň i své nedostatky. K přednostem patří rychlé nalezení řešení, za nedokonalost této metody je však nutno požadovat skutečnost, že nalezené řešení nemusí být optimální, kompletní a že tato metoda je velmi citlivá na její chybný počátek. Lačné hledání bývá také časově i prostorově složité. Kvalitní heuristika však může složitost této metody podstatně zmírnit. Pro nalezení optimální cesty z daného počátečního uzlu do požadovaného koncového uzlu bývá používán algoritmus A*, který bude následně popsán (Russell & Norvig, c2014).

Popis algoritmu hladové hledání prvního nejlepšího

Stejně jako v algoritmech neinformovaného hledání budeme používat OPENED a CLOSED (v aplikaci implementováno jako arraylist).

V prvním kroku je do OPENED uložen počáteční uzel. Potom následuje cyklus, který se opakuje do té doby, dokud není OPENED prázdná nebo dokud není nalezen výsledek. Na začátku cyklu vyjmeme nejlépe ohodnocený uzel z OPENED a uložíme ho do CLOSED. Za nejlépe ohodnocený je považován ten uzel, který je nejlépe ohodnocený použitou heuristickou funkcí. V tomto případě je použita heuristika straight-line distance, takže se jedná o uzel, který je „leteckou“ dráhou nejblíže k cíli. Všechny následníky tohoto uzlu, kteří ještě nejsou v CLOSED, ohodnotíme a přidáme do OPENED.

Implementace algoritmu, která je použita ve vytvořené aplikaci, je uvedena v příloze.

Ukázka prohledávání jednoduchého grafu:

Byl vybrán graf, na kterém je zřetelně viditelný rozdíl mezi algoritmem A*. Porovnávání s algoritmy prohledávání do hloubky a do šířky nemá velký význam už jen z důvodu, že v tomto případě se bavíme o informovaném hledání, jak bylo popsáno v teoretickém úvodu.

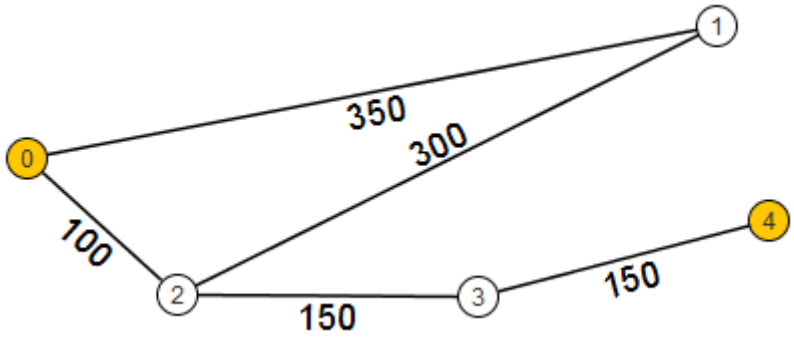
Počáteční stav:

Uzel 0 je počátek.

Uzel 5 je cílový.

Krok 1

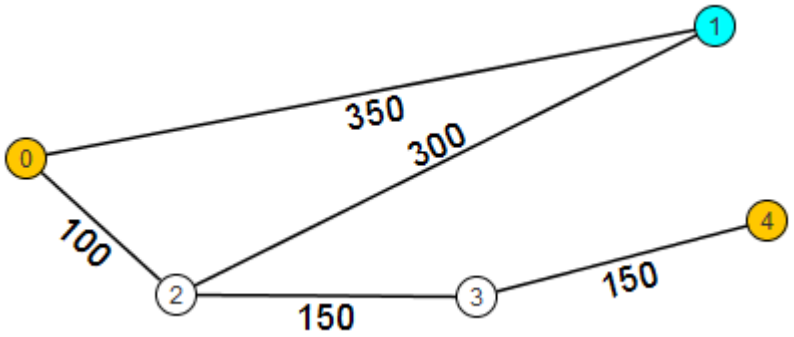
V prvním kroku podle výše uvedeného postupu expandujeme uzel 0 a jeho následníky uzly 1 a 2 přidáme do OPENED.



Obr. 13 Hladové hledání prvního nejlepšího krok 1

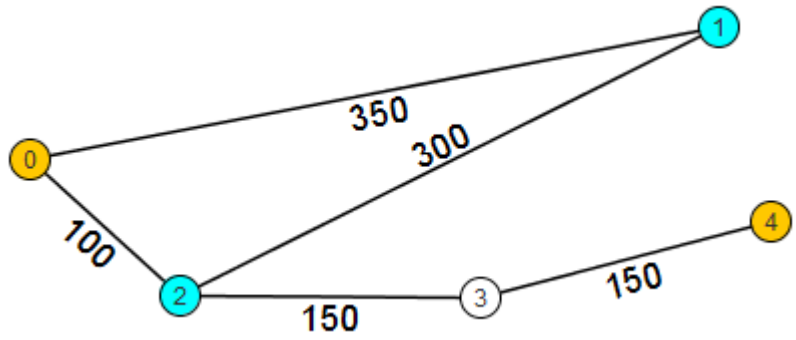
Krok 2

V kroku dva vybereme nejlépe ohodnocený uzel 1 z OPENED, expandujeme ho a jeho následníka uzel 2 přidáme do zásobníku.



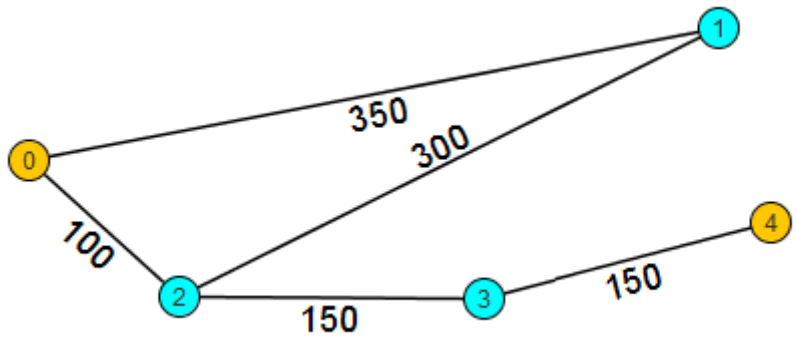
Obr. 14 Hladové hledání prvního nejlepšího krok 2

Krok 3
Protože podle zdroje se hladový algoritmus vrací jen pokud najde mrtvý bod, prohledávání pokračuje v uzlu 2.



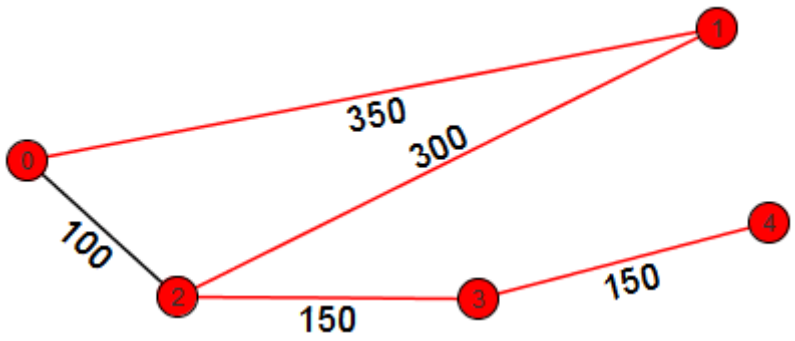
Obr. 15 Hladové hledání prvního nejlepšího krok 3

Krok 4
Čtvrtým krokem expandujeme uzel 3.



Obr. 16 Hladové hledání prvního nejlepšího krok 4

Krok 5
Pátým krokem byl nalezen cíl. Cestu tvoří uzly 0, 1, 2, 3 a 4.



Obr. 17 Hladové hledání prvního nejlepšího krok 5

2.8.4 Vlastnosti hladového hledání prvního nejlepšího:

Algoritmus je kompletní. Vždy najde výsledek, který odpovídá jeho heuristické funkci.

Také tento algoritmus splňuje všechny předpoklady, které jsou na algoritmy kladeny. Jednotlivé kroky jsou jednoduché, jejich počet je konečný. Vytvořený algoritmus hladového hledání prvního nejlepšího řeší obecnou třídu postupů a každý jeho krok je přesně stanoven. V rámci uvedeného postupu byl nalezen požadovaný výsledek.

Uvedený algoritmus není časově ani paměťově náročný. Použitím tohoto algoritmu bylo dosaženo výsledku.

2.8.5 Metoda A*

A* (A star) je algoritmus, který se používá pro vyhledávání optimálních cest v kladně ohodnocených grafech. Jeho autory jsou Peter Hart, Nils Nilsson a Bertram Raphael (Wikipedia, 2017). Jedná se o vylepšené řešení Dijkstrova algoritmu, tedy algoritmu používaného k nalezení nejkratší cesty v grafu, o heuristický prvek. Za nejlépe vyhovující cestu může být považována podle návrhu zadaných hodnot vah hran v grafu cesta nejkratší, nejrychlejší, nejlevnější apod. V případě, že je hledaná nejkratší cesta, je vstupem tohoto algoritmu ohodnocený graf, počáteční uzel a cílový uzel. Výstupem je pak nejkratší cesta vedoucí z počátečního uzlu do uzlu cílového nebo informace o tom, že žádná taková cesta neexistuje.

Hledání A* představuje hledání prvního nejlepšího s použitím f jako vyhodnocovací funkce a h jako přijatelné funkce. Omezením je pouze výběr takové funkce h, která nikdy nepřesáhne cenu dosažení cíle. Takovou funkci h nazýváme přijatelnou heuristikou. Přijatelné heuristiky jsou ve své podstatě funkce optimistické, protože předpokládají, že cena řešení problému je menší, než je ve skutečnosti.

Základ tohoto algoritmu je shodný s algoritmem prohledávání do šířky. Odlišuje se pouze tím, že místo obyčejné fronty používá frontu prioritní, ve které jsou cesty seřazeny podle hodnoty speciální funkce f. Tato funkce je definována pro každou cestu p a je součtem tzv. heuristické funkce (h) posledního uzlu cesty p a její zbývající délky (g). Čím je hodnota funkce f(p) nižší, tím vyšší má daná cesta p prioritu. Zjednodušeně je možno říct, že algoritmus se dívá do „minulosti", čímž máme na mysli, že sleduje, jak daleko musel ujít, než nalezl konec cesty p, a zároveň se dívá i do „budoucnosti", což zase znamená, že kontroluje, jak daleko ještě přibližně zbývá ujít z posledního uzlu cesty p do cíle.

Při použití tohoto algoritmu se předpokládá, že cesty ve frontě neobsahují kružnice a pro každý cílový uzel se v ní nachází nejvíce jedna cesta, a to právě ta nejkratší zatím nalezená.

Pořadí cest ve frontě je určeno následující funkcí:

- f(x) - předpokládaná délka cesty x,
- h(x) - hodnota heuristické funkce pro koncový uzel cesty x,
- g(x) - délka cesty x.

Použitá heuristická funkce musí odpovídat důležitým předpokladům. V prvním případě musí být větší než nula a dále musí být tzv. přípustná (admissible), což znamená, že její hodnota pro libovolný uzel musí být nižší nebo rovna skutečné vzdálenosti z daného uzlu do cíle. Její hodnota tedy nikdy nemůže být větší, než je skutečná vzdálenost z daného uzlu do cíle. Je nutné také zdůraznit, že heuristická funkce vzniká pouze na základě (alespoň částečné) znalosti struktury daného problému (Hordějčuk, 2017).

*Vlastnosti metody A**

Z hledání pomocí A* vyplývá, že podél libovolné cesty z kořene, f-cena nikdy neklesá. Není to náhodou a heuristiky, které tuto vlastnost nenarušují, se nazývají heuristikami monotónními. V roce 1984 bylo dokázáno, že heuristika je monotónní tehdy a právě jen tehdy, pokud splňuje pravidlo tzv. trojúhelníkové nerovnosti. Přímočaré vzdálenosti tuto podmínku samozřejmě splňují (Russell & Norvig, c2014).

Časová složitost:

Asymptotická složitost algoritmu A star závisí především na použité heuristické funkci. Tato funkce podstatným způsobem ovlivňuje množství cest uvažovaných během výpočtu. Lze však konstatovat, že jeho asymptotická složitost nikdy nebude horší než složitost prohledávání do šířky.

Podobnost s ostatními algoritmy:

Prohledávání do šířky (breadth-first search)

- všechny hrany mají stejné ohodnocení, heuristická funkce je konstantní
- h(x) = C
- cost(x,y) = C

Dijkstrův algoritmus (Uniform Cost Search)

- heuristická funkce je konstantní
- h(x) = C

Hladový algoritmus (Greedy Search)

- všechny hrany mají stejné ohodnocení
- cost(x,y) = C

(Hordějčuk, 2017).

Popis algoritmu A:*

Opět jako v algoritmech neinformovaného hledání budeme používat OPENED a CLOSED (v aplikaci implementováno jako arraylist).

V prvním kroku je do OPENED uložen počáteční uzel a je vypočítána jeho funkce f(x). Potom následuje cyklus, který se opakuje do té doby dokud není OPENED prázdná, nebo do doby, než je nalezen výsledek. Na začátku cyklu vyjmeme nejlépe ohodnocený uzel z OPENED a uložíme ho do CLOSED. Nejlépe hodnocený je uzel, který je nejlépe ohodnocený heuristikou. V tomto případě je použita heuristika straight-line distance. Jak již bylo popsáno v teoretickém úvodu, bereme také v potaz funkci g(x), která reprezentuje délku cesty z počátku. Dále vezmeme všechny následníky aktivního uzlu. Pokud ještě nejsou v CLOSED, vypočítáme jejich ohodnocení a v případě, že ještě nejsou obsaženy v OPENED, tak je do OPENED vložíme. Pokud již jsou obsaženy v OPENED, ale jejich aktuální hodnocení je lepší než obsažené v OPENED, tak hodnocení nahradíme.

Implementace algoritmu, která je použita ve vytvořené aplikaci, je uvedena v příloze.

Ukázka prohledávání jednoduchého grafu:

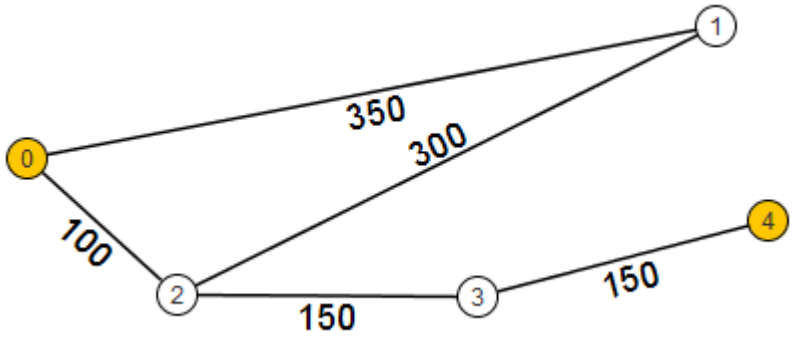
Počáteční stav:

Uzel 0 je počátek.

Uzel 5 je cílový.

Krok 1

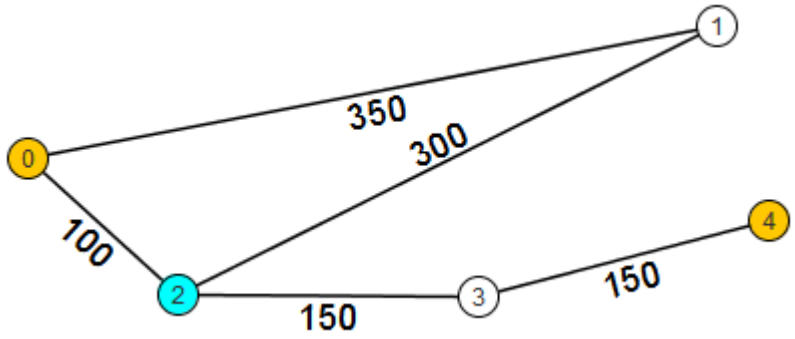
V prvním kroku podle výše uvedeného postupu expandujeme uzel 0 a jeho následníky uzly 1 a 2 přidáme do OPENED.



Obr. 18 A* krok 1

Krok 2

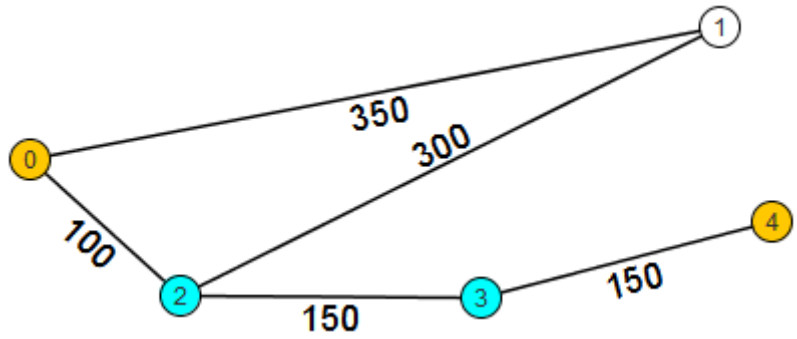
V kroku dva vybereme nejlépe ohodnocený uzel 2 z OPENED, expandujeme ho a jeho následníka uzel 3 přidáme do OPENED.



Obr. 19 A* krok 2

Krok 3

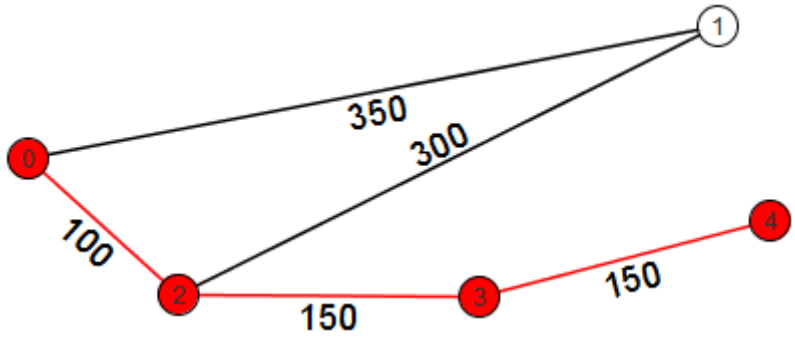
V kroku tři vybereme nejlépe ohodnocený uzel 3 z OPENED, expandujeme ho a jeho následníka uzel 4 přidáme do OPENED.



Obr. 20 A* krok 3

Krok 4

V kroku čtyři vybereme nejlépe ohodnocený uzel 4 z OPENED, zjistíme, že je to výsledek, a prohledávání proto ukončíme. Cestu tvoří uzly 0, 2, 3 a 4.



Obr. 21 A* krok 4

2.8.6 Vlastnosti prohledávání algoritmu A::*

Algoritmus *A*:* je úplný. Vždy najde výsledek, který odpovídá jeho vnitřní hodnotící funkci.

Také tento algoritmus splňuje všechny předpoklady, které jsou na algoritmy kladeny. Jednotlivé realizoavané kroky jsou jednoduché a jejich počet je konečný. Řeší obecnou třídu postupů a každý krok tohoto algoritmu je přesně stanoven. V rámci uvedeného postupu byl výsledek nalezen.

Uvedený algoritmus není časově ani paměťově náročný. Použitím tohoto algoritmu bylo dosaženo výsledku.

2.9 Závěr kapitoly

Obsahem této kapitoly se stalo vysvětlení pojmů důležitých pro pochopení zpracovávané problematiky, bez kterých by nebylo možné zadanou práci vytvořit.

V úvodu teoretické části práce jsou objasňovány pojmy inteligence a umělé inteligence. Dále se práce podrobně věnuje popisu expertních systémů včetně jejich úkolů, složení a komerčního využívání expertních systémů. Vybrané kapitoly jsou věnovány objasnění termínu stavového prostoru a prohledávání stavového prostoru. Samostatný díl se zabývá způsoby hodnocení metod prohledávání stavového prostoru. Závěr teoretické části je pak tvořen podrobným popisem vybraných algoritmů, a to algoritmu lačného prohledávání, algoritmu prohledávání do šířky, prohledávání do hloubky a algoritmu A*.

3 Vývoj aplikace

3.1 Účel aplikace

Účelem aplikace je vytvoření animované a uživatelsky interaktivní ukázky, která by měla dopomoci snadnému pochopení grafových algoritmů. Ovládání vytvořené interaktivní ukázky by mělo být velmi jednoduché a zároveň by mělo uživatelům umožňovat sledování a následné porovnávání výsledků daných algoritmů. Aplikace by měla být také snadno přenositelná a jednoduchá na instalaci a spuštění.

Shrnutí požadavků:

- Jednoduchost.
- Snadné ovládání.
- Bezchybné uživatelské prostředí.
- Nápověda.

3.2 Výběr programovacího jazyka

Především z důvodu snadné přenositelnosti byl zvolen programovací jazyk Java. Díky tomu je vytvořená aplikace funkční na nejznámějších desktopových platformách iOS, Windows i Linux. Jedinou nutností bude instalace Javy verze SE 7, případně novější verze.

3.2.1 Programovací jazyk Java

Na počátku 90. let se podařilo pracovníkům společnosti Sun Microsystems pod vedením Jamese Goslinga vyvinout jednoduchý avšak velmi efektivní jazyk určený pro spotřební elektroniku. Ve svých začátcích byl tento jazyk označován názvem Oak, později však byl z důvodu konkurence názvu přejmenovaný na programovací jazyk Java. Název Java ve svém překladu znamená americké slangové označení pro kávu. Tento programovací jazyk našel uplatnění zejména prostřednictvím internetu, neboť je velmi dobře použitelný na webu a je zdarma dostupný pro různé operační systémy, jako např. pro Windows, Linux nebo Solaris. První oficiální verze Java Development Kit (JDK) 1.0 vyšla 26. srpna 1996 a stala se populární zejména díky Java Appletům. Následně byly vyvíjeny další verze tohoto programovacího jazyka, poslední oficiální verze Java 8 byla zveřejněna 18. března 2014. V současné době se již pracuje na verzi Java SE 9, která měla být veřejnosti představena již počátkem letošního roku, doposud k tomu však nedošlo. Mezi výhody Javy patří zejména její čitelnost a jednoduchost. Dalším prvkem, který Javě zajišťuje oblibu, je dlouhodobá kompatibilita, díky které je zaručeno, že její starší aplikace pojedou i v budoucnu.

Java je podle cílového zařízení rozdělena do tří edic. Základní edice Java SE (Standard Edition) slouží k vývoji aplikací pro stolní počítače. Její podmnožinou je Java ME (Micro Edition), která je určena pro vestavná zařízení, jakými jsou mobily, domácí elektronika atp. Poslední edicí, která přidává do Javy SE funkcionalitu pomocí dodatečných knihoven, je Java EE (Enterprise Edition). Tato edice má rozsáhlé uplatnění na poli serverových aplikací, ať již se jedná o bankovní aplikace, informační systémy nebo pouze o dynamické webové stránky (Neckář, 2016).

V současné době jde v případě programovacího jazyka Java o jeden z nejpoužívanějších programovacích jazyků na světě. Podle TIOBE indexu je Java nejpopulárnější programovací jazyk (Tiobe, 2013).

3.3 Výběr frameworků

Framework představuje prostředí, ve kterém se vyvíjejí aplikace. Jedná se o softwarovou strukturu sloužící jako podpora při programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API představující rozhraní pro programování aplikací, podporu pro návrhové vzory nebo doporučené postupy při vývoji.

V rámci této práce byla pro tvorbu grafického uživatelského rozhraní zvolena knihovna Swing. Pro modelování grafů bylo možno volit ze dvou hlavních frameworků a to JUNG a JGraph. Nakonec byl zvolen Jung především pro lepší vizualizaci.

3.3.1 Swing

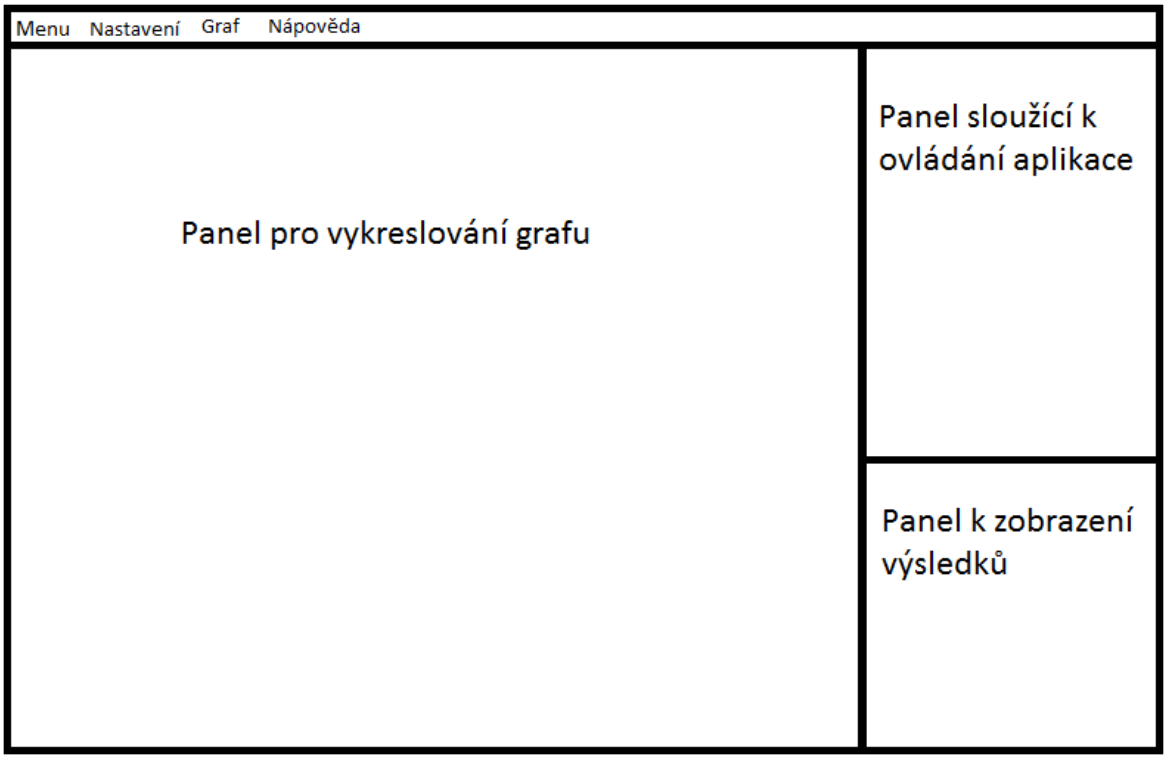
Swing představuje standardní knihovnu pro tvorbu grafického rozhraní v programovacím jazyce Java. Jedná o tzv. lightweightknihovnu, která provádí (téměř) veškeré kreslení komponent sama. Kreslení se realizuje v programovacím jazyce Java, nedochází k žádnému delegování na systémové binárky. Tento postup je výhodný z hlediska dobré přenositelnosti samotné knihovny i programů v ní napsaných. Knihovna se také nemusí zabývat problémem, jestli daný systém konkrétní funkcionalitu podporuje. Z uvedeného důvodu nabízí větší množství poskytovaných funkcí (Neckář, 2016).

3.3.2 Jung

Jung je volně šiřitelná softwarová knihovna napsaná v programovacím jazyce Java. Je určená k modelování, analýzám a vizuálnímu zobrazování grafů. V rámci této práce byla použita jeho poslední verze 2.1.1.

Knihovna je dostupná na adrese: https://github.com/jrtom/jung

3.4 Návrh uživatelského prostředí



Obr. 22 Návrh uživatelského prostředí

Podle zadání bylo nutné navrhnout na první pohled velmi jednoduché uživatelské prostředí. Rozvržení nejdůležitějších částí aplikace bylo zvoleno následujícím způsobem:

Panel pro vykreslování by měl být velikostně flexibilní, aby měl uživatel možnost měnit velikost vykreslovací plochy podle aktuální potřeby.

Panel sloužící k ovládání by měl pro dosažení jednoduchého ovládání obsahovat minimum ovládacích prvků.

Panel k zobrazení výsledků by měl sloužit k jednoduchému výpisu výsledků průběhů vybraných algoritmů.

V horní části aplikace by mělo být zobrazeno klasické menu, které by obsahovalo:

- Záložku **Menu** sloužící především pro možnosti ukončení aplikace, její ukládání a nahrávání.
- Záložku **Nastavení** pro pokročilé možnosti úpravy generování a vykreslování grafu.
- Záložku **Graf** pro vytváření grafů a vyčištění kreslící plochy.
- Záložku **Nápověda** určenou pro vysvětlení ovládání aplikace.

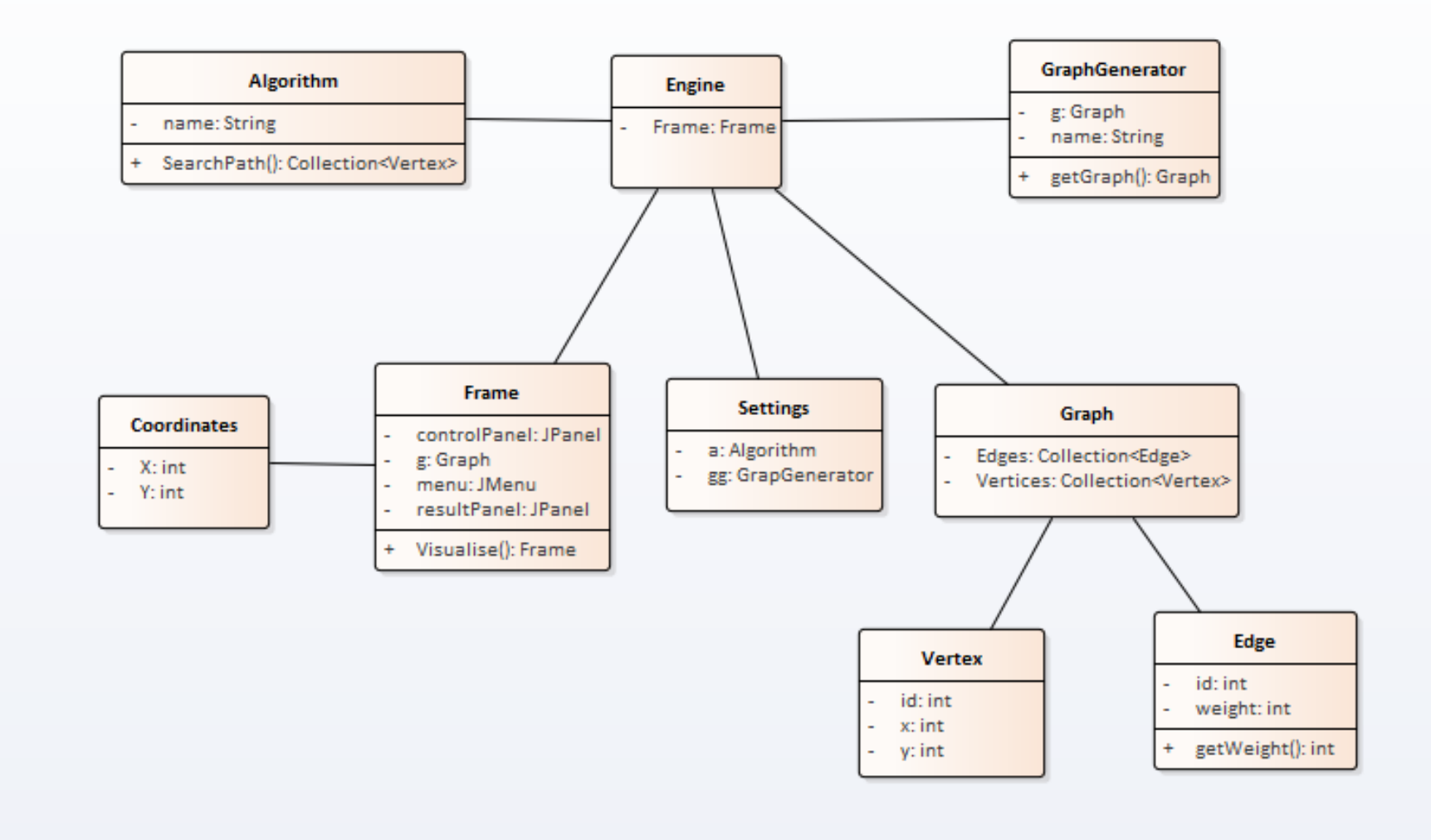
3.5 Návrhový diagram tříd

Diagram tříd byl vytvořen v jazyce UML.

UML, Unified Modeling Language v překladu unifikovaný modelovací jazyk, představuje soubor grafických notací, který se používá při vývoji softwaru. Jedná se o velmi užitečný nástroj programátorů pro usnadnění návrhu a vývoje informačního systému. Vznikl v důsledku vzrůstající složitosti informačních systémů, na základě které vyvstala potřeba velmi dobré komunikace mezi programátory. V průběhu 90. let se podařilo firmě Rational Software pomocí sjednocení několika metodik vytvořit standard UML, který výrazně usnadnil právě komunikaci při vytváření programovacích systémů.

UML lze obecně použít třemi způsoby. V prvním případě můžeme UML diagramy používat ve velmi jednoduché podobě jako náčrt či koncept. Obvykle se tak jedná o ručně kreslené diagramy na tabuli nebo do sešitu vznikající ze společného jednání vývojářů s klientem a následně usnadňující komunikaci mezi samotnými vývojáři. Dále je možné UML použít jako plán, který je ve srovnání s náčrtem o mnoho detailnější. Jedná se totiž o zaznamenání komplexního návrhu, na jehož základě by již měl být programátor schopen vytvořit samotný program. Třetím významem je UML jako programovací jazyk. Z detailního UML diagramu lze vygenerovat šablonu kódu, která slouží jako základ pro implementaci. Vývojářem jsou nakresleny UML diagramy, ze kterých je následně vygenerován přímo spustitelný kód.

Diagramy dělíme na dvě základní skupiny a to na diagramy struktury a diagramy chování. Diagramy struktury popisují strukturu systému. Objasňují nám tedy, z čeho je systém složený. Diagramy chování zase zaznamenávají chování systému, tedy to, jak systém funguje (Čapka, 2017).

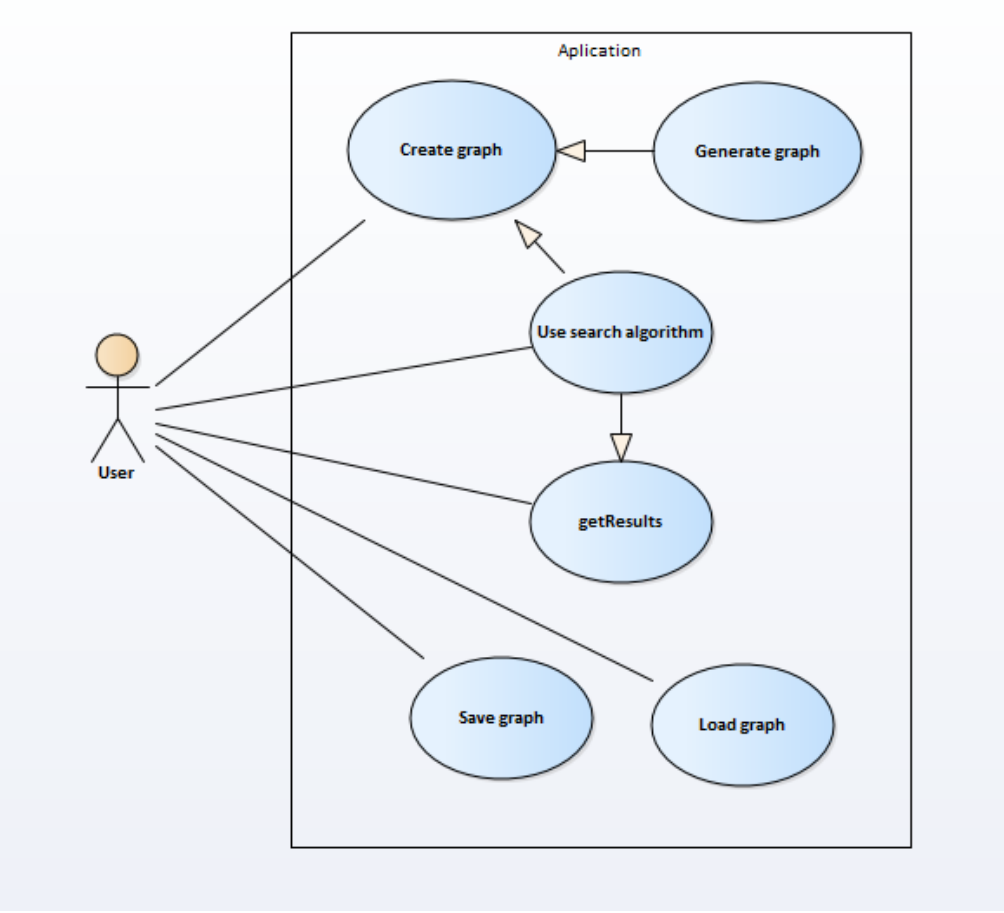


Obr. 23 Diagram tříd

3.6 Use Case

Use Case v překladu znamená případ užití. Jedná se o definici jednoho úkolu, který by měl navrhovaný systém zvládnout, přičemž tento úkol v sobě může zahrnovat další dílčí akce, které již v diagramu zachyceny nebudou. UML často hovoří o tzv. Blackboxu, tedy černé skříňce, ve které je ukryta vnitřní logika, přičemž programátor pracuje pouze s jednotlivými součástmi. Tento princip využívá právě Use Case diagram. Use Case diagram zobrazuje chování systému z pohledu uživatele. Jde většinou o první diagram, který je programátorem při návrhu informačního systému vytvářen. Proto je velmi důležité zachytit a popsat vše, co bude uživatelem od systému požadováno, a tím přesně vymezit, k čemu bude navrhovaný systém určen.

Případy užití spolu s aktéry a jednotlivé vztahy mezi nimi spolu tvoří Use Case diagram (Wikipedia, 2017).



Obr. 24 Use case diagram

V diagramu je naznačen způsob používání systému aktérem. Vzhledem k tomu, že jsou v diagramu uvedeny jen názvy činností, je zapotřebí přesněji

vymezit další podrobnosti vztahující se k navrhovanému systému, tedy vložit do jednotlivých Use Case textové specifikace, případně je možné podrobnosti namodelovat dalším navázaným diagramem.

Use Case vychází z požadavků uživatele. Uživatel by tedy měl mít možnost vytvořit graf a to buď graf vlastní nebo generovaný. Měl by mít také možnost projít graf pomocí vybraného algoritmu a sledovat výsledky hledání. Z důvodu uchovávání příkladových grafů je pro uživatele také velmi důležité ukládání a načítání grafu (Wikipedia, 2017).

3.7 Samotná implementace

Vývoj návrhu implementace probíhal v prostředí Eclipse.

Eclipse představuje vývojové prostředí určené pro programování v jazyce Java. Návrh tohoto vývojového prostředí je velmi adaptabilní a umožňuje rozšiřování seznamu podporovaných programovacích jazyků za pomoci pluginů, tedy zásuvných modulů, které pracují pouze jako doplňkový modul jiné aplikace, čímž rozšiřují funkčnost této aplikace. Nevýhodou pluginů je, že nepracují samostatně (Wikipedia, 2017).

Projekt byl pro jednodušší buildování strukturovaný jako maven.

Maven byl vytvořen jako prostředek pro zjednodušení buildů v rámci projektu Jakarta Turbine. Podnětem pro jeho zhotovení byl pokus směřující ke standardizaci a také znovupoužitelnosti buildovacích skriptů. Je určený především pro usnadnění práce při buildování aplikací a agendy s buildováním spojenou. Jeho úkolem je proto zejména usnadnění procesu buildování a poskytování kvalitních informací o projektu. Svými tvůrci byl určen také k zavedení jednotného systému buildování. Další jeho neméně důležitou funkci představuje poskytování příkazů pro osvědčené postupy a poskytnutí transparentního přidávání nových funkcí.

Maven pracuje na principu popsání projektu za pomoci modelu nazvaného Project Object Model (konceptu popisu projektu jako objektu). Jde o model, který popisuje softwarový projekt z hlediska jeho zdrojového kódu a zároveň i z pohledu závislostí na externích knihovnách, popisu procesu buildování a různých funkcí s tím spojených. Maven pracuje na základě volání jednotlivých pluginů a sám obstarává pouze dodání a spuštění nadefinovaných pluginů. Nemá žádné vlastní grafické uživatelské rozhraní (Wikipedia, 2017).

Nejprve bylo implementováno generování grafu a to pomocí algoritmů Barabási–Albert, Eppstein Power Law a Kleinberg SmallWorld.

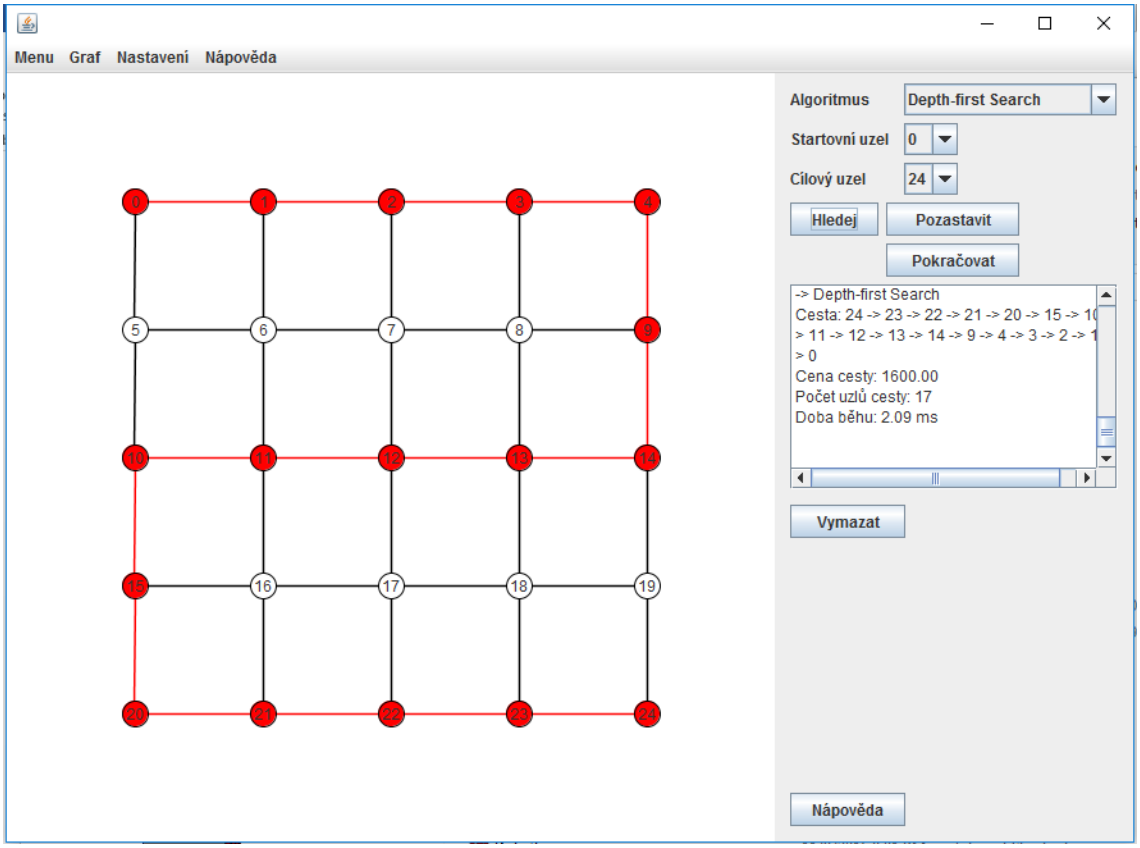
V rámci zachování jednoduchého ovládání byly některé parametry ponechány jako konstanty.

Podrobná dokumentace je dostupná zde:
http://jung.sourceforge.net/doc/api/edu/uci/ics/jung/algorithms/generators/random/package-summary.html

Následovalo vytvoření uživatelského rozhraní za pomoci frameworku Swing. Snahou bylo přiblížit se co nejvíce návrhu.

Podstatnou část práce představuje vykreslení grafu do JPanelu frameworku Swing. Pro tento účel vznikla třída visualisation, která se stará o zobrazení grafu.

Výsledné uživatelské prostředí s ukázkou grafu:



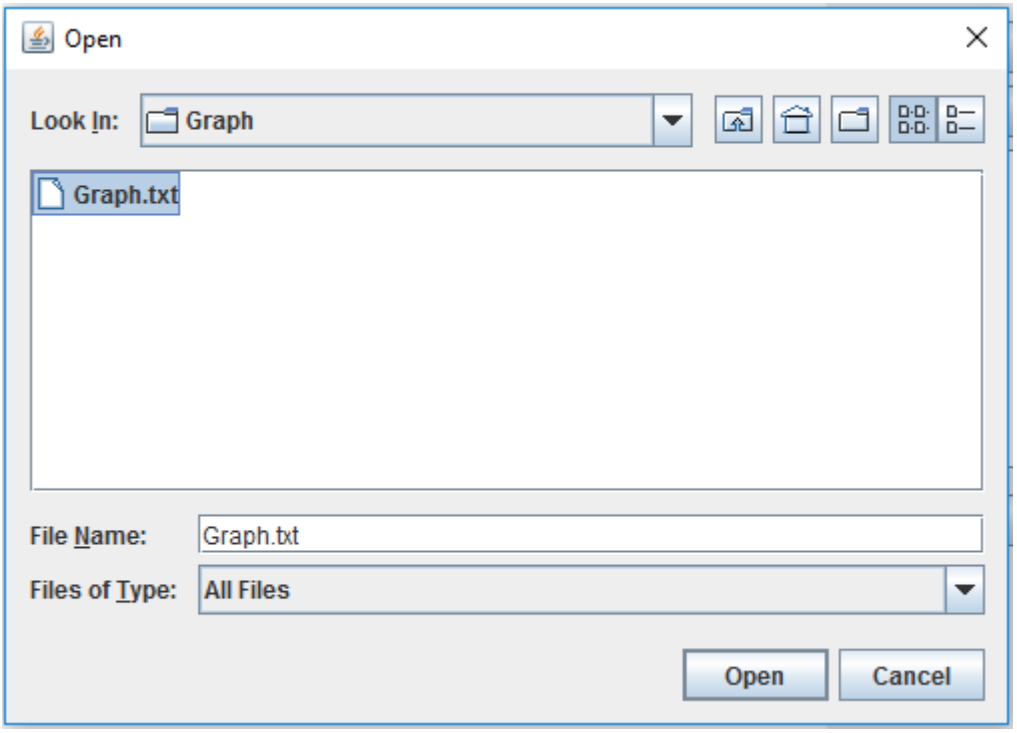
Obr. 25 Výsledné uživatelské prostředí

Následně bylo možné přejít k implementaci prohledávacích algoritmů, které jsou uvedeny a podrobně popsány v teoretické části.

3.8 Funkce pro ukládání

Dalším bodem se stalo vytvoření funkcí pro ukládání a načítání, které byly implementovány pomocí knihovny Swingu JFileChooser. Podrobná dokumentace je zaznamenána zde:
https://docs.oracle.com/javase/tutorial/uiswing/components/filechooser.html

Výsledek:



Obr. 26 Načtení grafu

3.9 Nápověda

Poslední fází implementace bylo vytvoření nápovědy, ošetření vstupů a vytvoření testů.

4 Typické úlohy

Doby běhů algoritmů jsou velmi orientační, záleží zde na stroji a podmínkách. V práci byly použity konstantní podmínky.

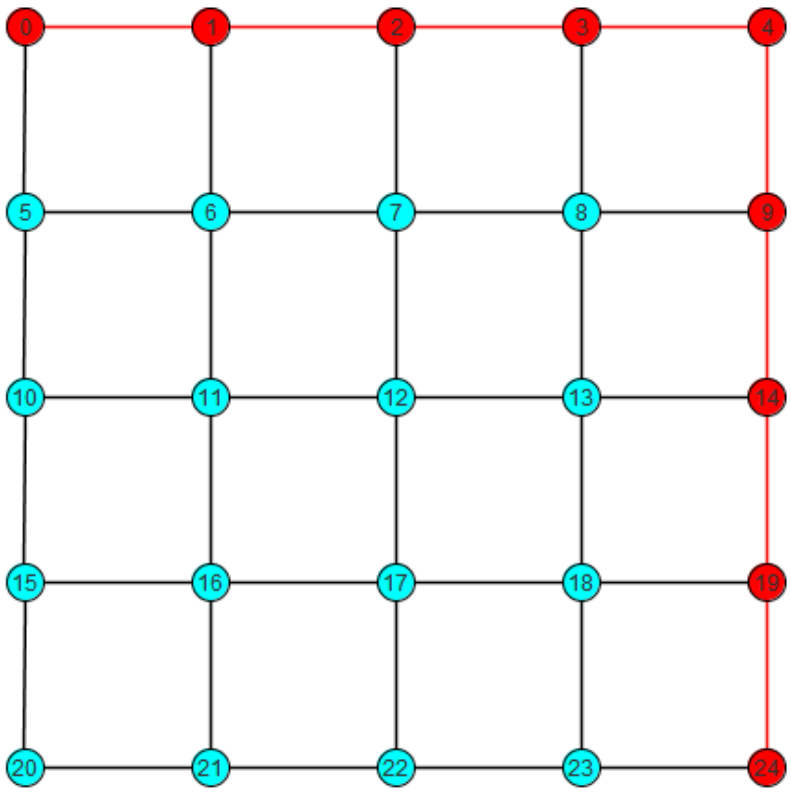
Červeně jsou označeny uzly cesty, tyrkysově expandované uzly.

4.1 Příklad hledání - mřížka

Hledání v grafu, který připomíná „mřížku“, je jeden z typických příkladů.

Je zvolena mřížka s konstantní délkou hran s 25 uzly. Počáteční uzel je 0, cílový 24.

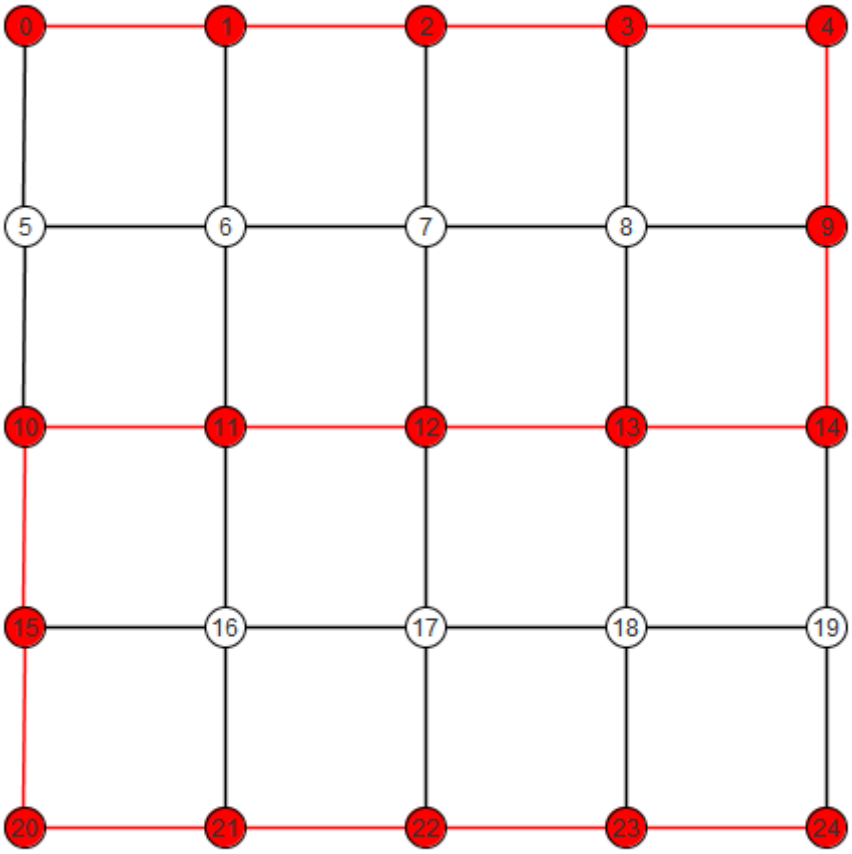
Nalezení cíle pomocí prohledávání do šířky:



Obr. 27 Příklad mřížka 1

Cena cesty: 800.00 jednotek
Počet uzlů cesty: 9
Doba běhu: 0.11 ms

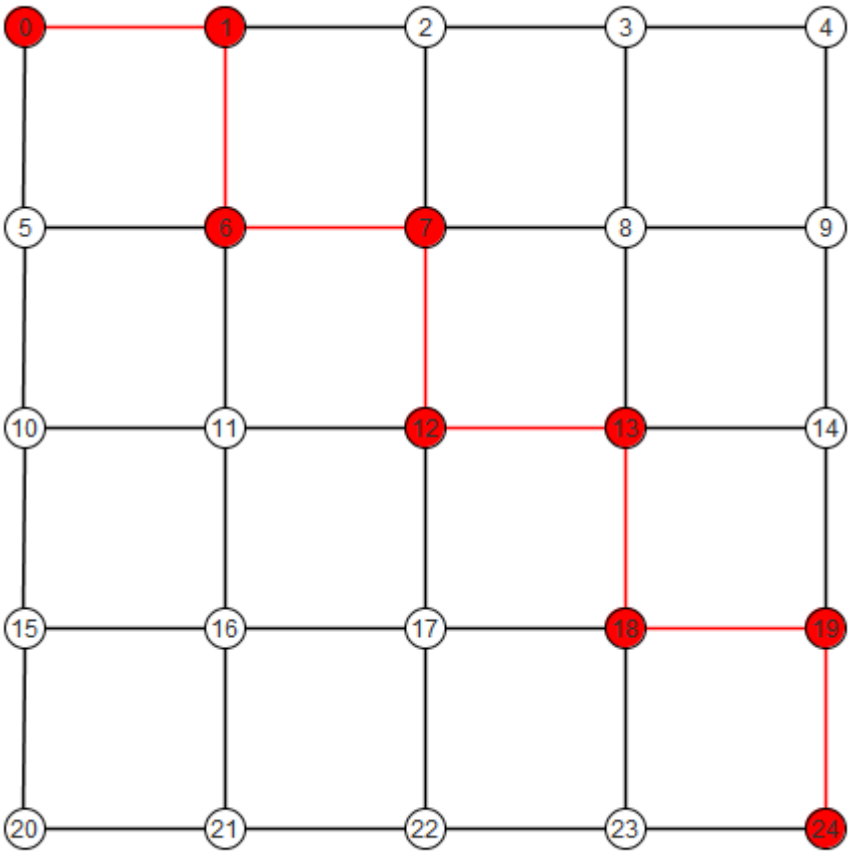
Nalezení cíle pomocí prohledávání do hloubky:



Obr. 28 Příklad mřížka 2

Cena cesty: 1600.00 jednotek
Počet uzlů cesty: 17
Doba běhu: 0.09 ms

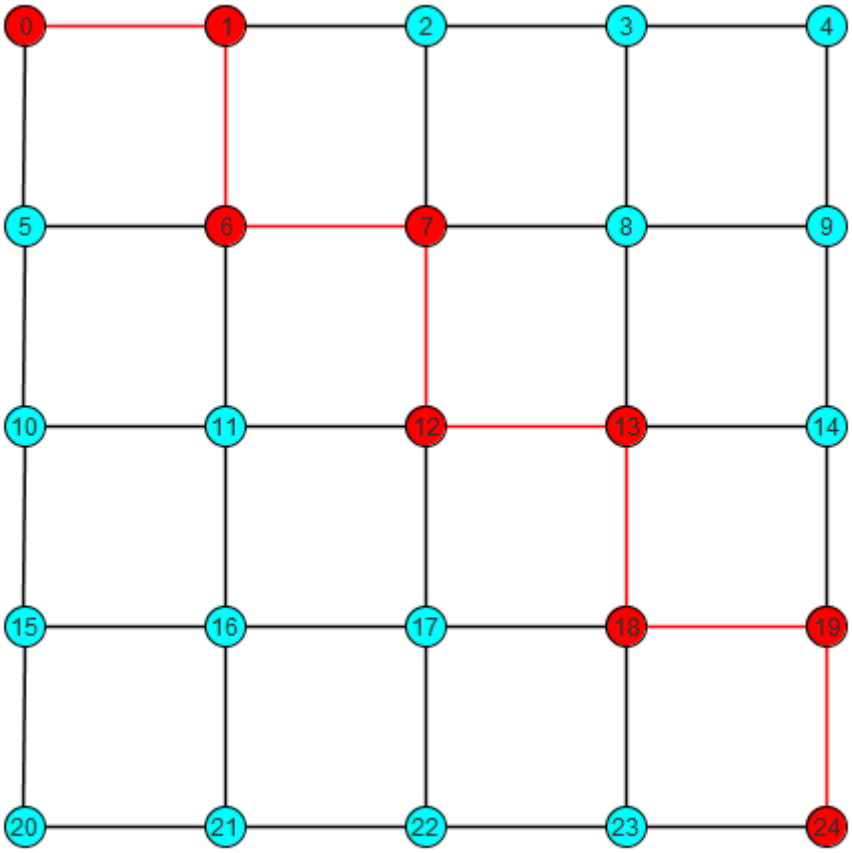
Nalezení cíle pomocí hladového hledání prvního nejlepšího:



Obr. 29 Příklad mřížka 3

Cena cesty: 800.00 jednotek
Počet uzlů cesty: 9
Doba běhu: 0.33 ms

Nalezení cíle pomocí A*:



Obr. 30 Příklad mřížka 4

Cena cesty: 800.00 jednotek
Počet uzlů cesty: 9
Doba běhu: 2.88 ms

Na první pohled se jeví A* a hladové hledání prvního nejlepšího stejné. Z detailního rozboru je však zřejmé, že algoritmus A* prozkoumal všechny uzly, hladový algoritmus však procházel pouze ty uzly, které jsou na cestě, což se projevilo především na době běhu algoritmu A*, která u tohoto příkladu byla přibližně 10x delší.

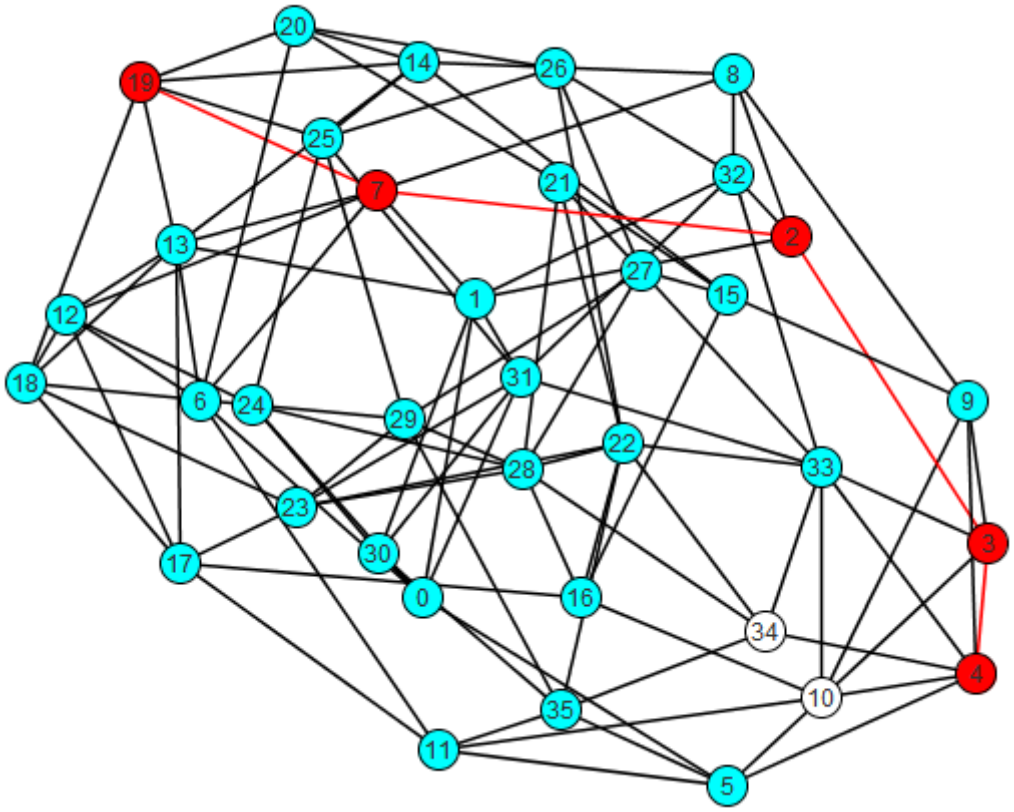
Ukázkový graf je možné přímo vygenerovat prostřednictvím aplikace.

4.2 Příklad hledání – generovaný graf

Jako další ukázkový příklad byl zvolen graf, u kterého není na první pohled zřetelná optimální cesta.

Je to graf vygenerovaný pomocí aplikace s 36 uzly. Počáteční uzel je uzel s číslem 19, cílovým je uzel číslo 4.

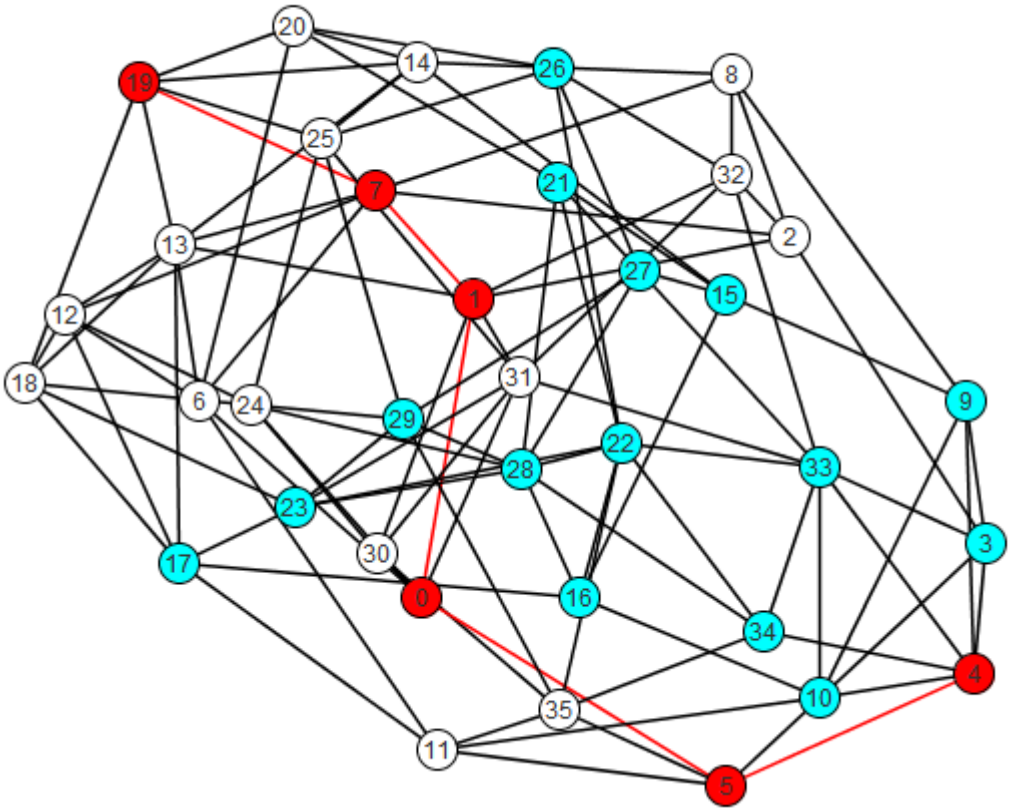
Nalezení cíle pomocí prohledávání do šířky:



Obr. 31 Příklad generovaný graf 1

Cena cesty: 584.92 jednotek
Počet uzlů cesty: 5
Doba běhu: 0.11 ms

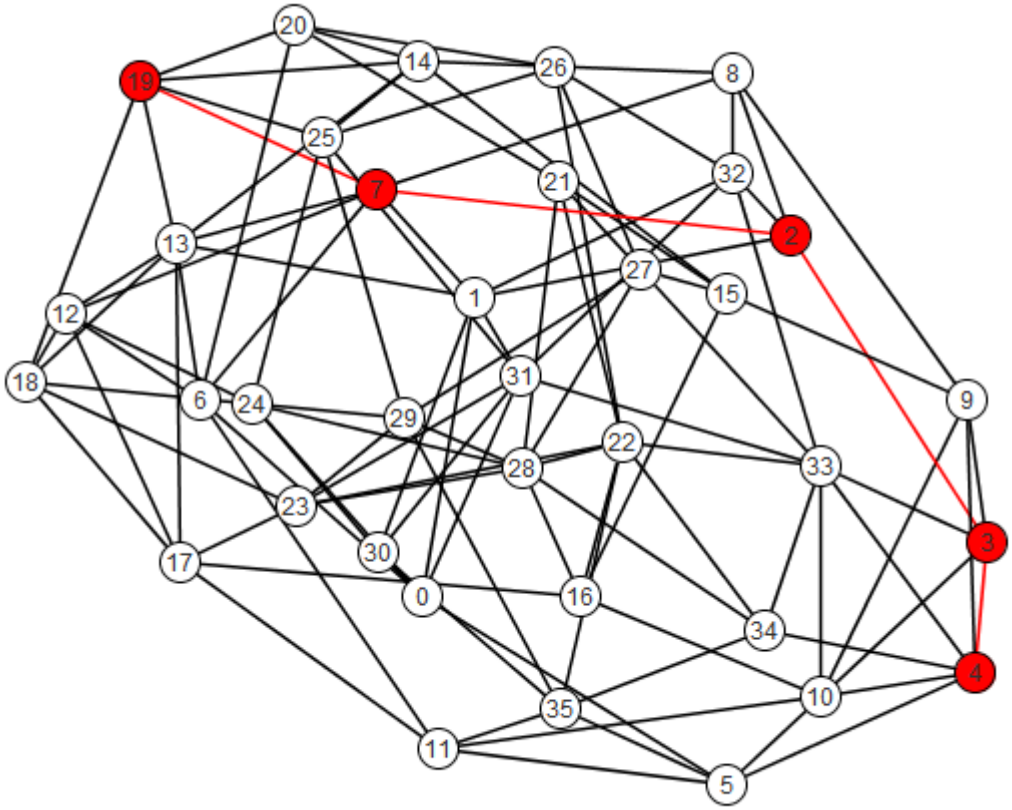
Nalezení cíle pomocí prohledávání do hloubky:



Obr. 32 Příklad generovaný graf 2

Cena cesty: 669.25 jednotek
Počet uzlů cesty: 6
Doba běhu: 0.12 ms

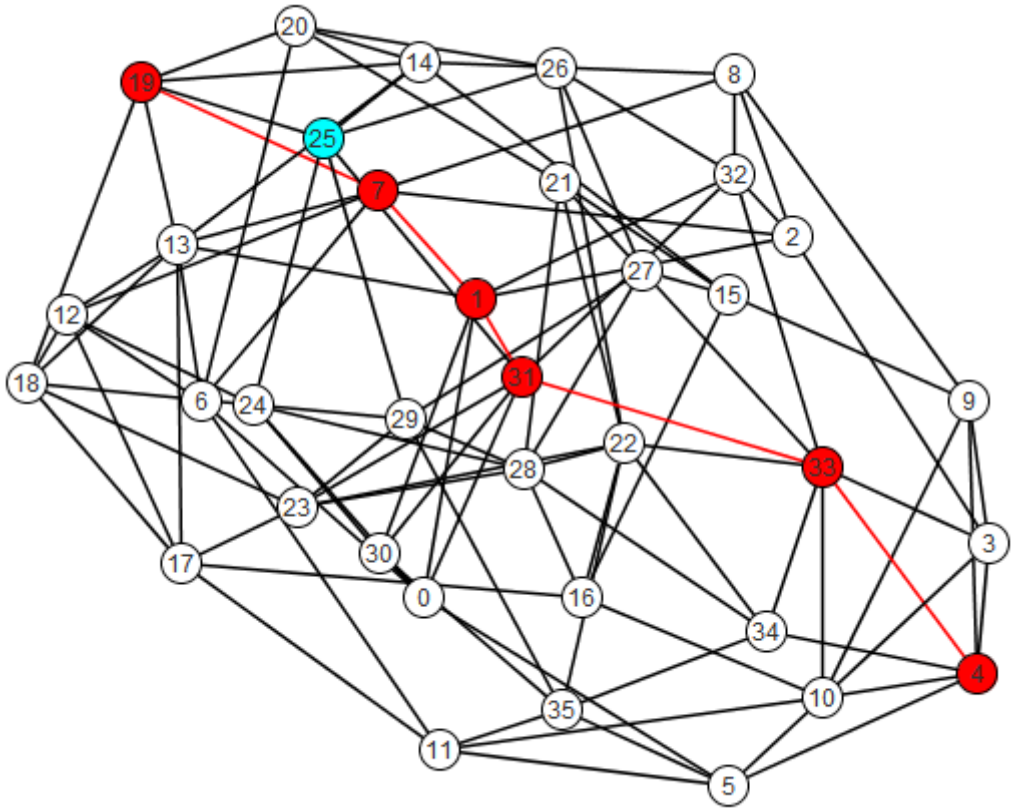
Nalezení cíle pomocí hladového hledání prvního nejlepšího:



Obr. 33 Příklad generovaný graf 3

Cena cesty: 584.92 jednotek
Počet uzlů cesty: 5
Doba běhu: 0.29 ms

Nalezení cíle pomocí A*:

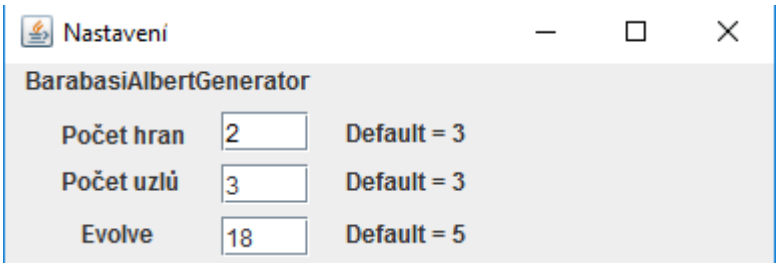


Obr. 34 Příklad generovaný graf 4

Cena cesty: 533.98 jednotek
Počet uzlů cesty: 6
Doba běhu: 1.79 ms
Ukázkový graf je přiložen na připojeném CD.

5 Porovnání algoritmů

Pro porovnání algoritmů bylo za pomoci vytvořené aplikace vygenerováno 100 grafů o dvaceti uzlech a přibližně 30 hranách pomocí algoritmu Barabási–Albert. Přesné nastavení:



Obr. 35 Nastavení generování grafu

Sledována byla délka běhu algoritmu a cena cesty.

5.1 Průměrné doby běhu algoritmu:

Prohledávání do šířky: 99988,93 ns
Prohledávání do hloubky: 122047,87 ns
Hladové hledání prvního nejlepšího: 155592,25 ns
A*: 181314,54 ns

5.2 Průměrné ceny cest:

Prohledávání do šířky: 473,12 jednotek
Prohledávání do hloubky: 642,46 jednotek
Hladové hledání prvního nejlepšího: 498,57 jednotek
A*: 449,52 jednotek
Všechny vygenerované grafy jsou dostupné v příloze.

5.3 Závěr kapitoly

Z uvedeného porovnání je patrné, že informovaná hledání z hlediska ceny cesty dosahují lepších výsledků, ale jsou časově náročnější.

6 Metodika

Cílem této diplomové práce je návrh a realizace implementace algoritmu A* a jeho užití v názorné animované grafické a uživatelsky interaktivní demonstraci vlastností A*.

6.1 Nastudování problematiky

Pro vytvoření této práce bylo nejdříve nezbytné seznámit se s problematikou umělé inteligence, algoritmů a jejich vlastností. Dílčí kapitoly jsou věnovány popisu stavového prostoru, jeho prohledávání a metodami jeho prohledávání. Z hlediska daného zadání bylo také nutné důkladně prostudovat i způsoby hodnocení metod prohledávání stavového prostoru.

6.2 Tvorba aplikace

6.2.1 Výběr programovacího jazyka

Pro vytvoření této práce byl zvolen programovací jazyk Java a to především z důvodu jeho snadné přenositelnosti.

6.2.2 Výběr frameworků

K tvorbě grafického uživatelského rozhraní byla zvolena knihovna Swing. Framework Jung byl použit k názornému zobrazení vytvořených grafů.

6.2.3 Návrh uživatelského prostředí

Podle zadání bylo navrženo na první pohled velmi jednoduché uživatelské prostředí. Pro vytvoření návrhu tohoto prostředí byla použita aplikace MS Paint 6.1.

6.2.4 Návrhový diagram tříd a Use Case

Diagram tříd byl vytvořen v jazyce UML. Pro vytvoření diagramů byla použita aplikace Enterprise architect, verze 12.

6.3 Navržení demonstračních úloh

Pomocí aplikace bylo navrženo několik vhodných typicky demonstračních problémů pro zobrazení vývoje jejich řešení pomocí implementovaných algoritmů.

6.4 Závěr kapitoly

V této kapitole byl zaznamenán postup při vypracování práce, zvolení metody a použití aplikace. Výsledky realizovaného zadání byly v závěru práce v rámci diskuze vyhodnoceny.

7 Diskuze

V této části práce budou rozvedeny hlavní výsledky práce a splnění zadaného úkolu.

Cílem této práce bylo vytvoření animované a uživatelsky interaktivní ukázky směřující ke snadnému pochopení grafových algoritmů. Ovládání vytvořené interaktivní ukázky mělo být velmi jednoduché a zároveň mělo uživatelům umožňovat sledování a následné porovnávání výsledků daných algoritmů. Aplikace měla být také snadno přenositelná a jednoduchá na instalaci a spuštění.

V rámci práce tedy bylo vytvořeno uživatelské prostředí sestávající z panelu pro vykreslování, panelu sloužícího k ovládání aplikace a panelu k zobrazení výsledků. Horní část aplikace obsahuje klasické menu, které uživatelům nabízí záložku menu, nastavení, graf a nápovědu. Vytvořené uživatelské prostředí je, tak jak bylo požadováno, již na první pohled velmi jednoduché, pro uživatele přehledné. Uživateli umožňuje také vhodné intuitivní ovládání názorných ukázek.

Následně byl jazyce UML vytvořen diagram tříd. Tvorba samotného návrhu implementace probíhala v prostředí Eclipse, přičemž projekt byl pro jednodušší buildování strukturovaný jako maven. Bylo implementováno generování grafu a to pomocí algoritmů Barabási–Albert, Eppstein Power Law a Kleinberg SmallWorld. V rámci zachování jednoduchého ovládání pro uživatele byly některé parametry ponechány jako konstanty. Následovalo vytvoření uživatelského rozhraní za pomoci frameworku Swing, protože snahou bylo přiblížit se co nejvíce návrhu. Podstatnou část práce představuje vykreslení grafu do JPanelu frameworku Swing. Pro tento účel vznikla třída visualisation, která je určena k zobrazení grafu.

Následovala realizace vybraných algoritmů a to algoritmu prohledávání do šířky, prohledávání do hloubky, hladového algoritmu a algoritmu A plus.

Bylo vytvořeno výsledné uživatelské prostředí i s ukázkou grafu. V této části práce již bylo možné přejít k implementaci prohledávacích algoritmů, které jsou uvedeny shora a podrobně popsány v teoretické části. Dalším krokem se stalo vytvoření funkcí pro ukládání a načítání grafů, které byly implementovány pomocí knihovny Swingu JfileChooser. Poslední fází implementace se stalo vytvoření nápovědy, ošetření vstupů a vytvoření testů.

V další části práce byla provedena ukázka typických úloh prohledávání grafů a to stromového grafu, „mřížkového“ grafu a grafu náhodně vytvořeného. V rámci takto provedených ukázek byly vyhodnoceny výsledky, které jsou uvedeny pod ukázkami. K porovnání došlo také za pomoci generování náhodných grafů a zprůměrování výsledných hodnot.

7.1 Návrhy na rozšíření aplikace

Velmi snadnou úpravou je možné aplikací přidat libovolný grafový algoritmus.

Například pro algoritmy na hledání nejkratší cesty by bylo vhodné upravit uživatelské prostředí a to například pouze skrytím rozbalovacího seznamu pro

výběr cílového uzlu. Po této jednoduché úpravě již pak stačí pouze implementovat vybraný algoritmus.

Pro algoritmy sloužící k hledání minimální kostry grafu by zase bylo vhodné zakrýt rozbalovací okna pro startovní i cílový uzel.

Při algoritmech barvení grafu by bylo nutné realizování úprav ve třídě visualisation, konkrétně vložení funkce pro *VertexFillPaintTransformer*, která se stará o vybarvování vertexů. Také by bylo nutné ve třídě Vertex přidat atribut barva.

Při úpravě uživatelského rozhraní by bylo vhodné do budoucna přidat možnost přidávat uzly s přesně danými souřadnicemi pro jednodušší tvorbu pravidelných grafů.

Aplikaci by bylo také možné překompilovat jako webový applet.

V rámci práce bylo vytvořeno jednoduché uživatelské prostředí. Další možnou úpravou by mohlo být přepracování designu podle aktuálních potřeb uživatelů.

8 Závěr

V této části práce budou zhodnoceny výsledky práce a splnění zadaného úkolu.

8.1 Zhodnocení výsledků

Diplomová práce se skládá ze dvou částí a to části teoretické zaměřené na vysvětlení pojmů důležitých pro pochopení zpracovávané problematiky a části praktické obsahující tvorbu samotné aplikace. V teoretické části jsou objasňovány pojmy vztahující se k zadané problematice, jakými jsou pojem inteligence a umělé inteligence, tato část je zaměřena také na vysvětlení odborných termínů a to expertního systému a stavového prostoru. Předmětem teoretické části se stalo prohledávání stavového prostoru společně se způsoby a metodami prohledávání stavového prostoru. Jedna kapitola je věnována algoritmu a jeho vlastnostem.

Důležitou součástí teoretické části práce je i podrobný popis algoritmus A* s uvedením jeho výhod a nevýhod a jeho srovnání s jinými typickými algoritmy popsanými v této práci a to algoritmem prohledávání do šířky, do hloubky a hladovým algoritmem. Srovnání bylo zaměřeno na jejich časovou a prostorovou složitost a kvalitu výsledku při hledání optima.

Účelem aplikace bylo vytvoření animované a uživatelsky interaktivní ukázky směřující ke snadnému pochopení grafových algoritmů. Ovládání interaktivní ukázky bylo v souladu se zadáním vytvořeno velmi jednoduché. Uživatelům však umožňuje sledování a následné porovnávání výsledků daných algoritmů. Byla vytvořena aplikace, která je snadno přenositelná a zároveň je i jednoduchá na instalaci a spuštění.

Podle zadání práce byl implementován algoritmus A*. Tento algoritmus byl použit v názorné animované grafické a uživatelsky interaktivní demonstraci A*. Součástí implementace se staly i vybrané další prohledávací algoritmy. Došlo ke vzájemnému srovnání jejich průběžné činnosti, výsledků a efektivity.

Dále byl proveden návrh několika vhodných typických demonstračních problémů pro zobrazení vývoje jejich řešení pomocí implementovaných algoritmů.

Došlo k vyhodnocení výsledků realizovaného zadání včetně vlastností jednotlivých implementovaných algoritmů vzhledem k navrženým úlohám pro řešení. Byl proveden návrh dalšího možného rozšíření vytvořeného demonstračního systému z hlediska uživatelského rozhraní, přidání dalších algoritmů a charakteristických úloh pro řešení umělou inteligencí.

Praktická část práce obsahuje popis jednotlivých kroků nezbytných pro vytvoření samotné aplikace. Tato část se proto skládá z výběru programovacího jazyka a frameworků. Nachází se v ní i samotný návrh uživatelského prostředí a nechybí zde ani návrhový diagram tříd a Use Case. Nezbytnou součástí praktické části se stalo i navržení demonstračních úloh.

V rámci tvorby diplomové práce se požadovanou aplikaci vytvořit podařilo. Vytvořená aplikace je v souladu se zadáním uživatelsky přívětivá, jednoduchá na ovládání klientem a současně, jak bylo také požadováno, názorně demonstruje

ukázky algoritmů zmíněných v zadání. Klientovi umožňuje i vhodné intuitivní ovládání demonstrace. Součástí systému se stala i vytvořená nápověda.

V rámci práce byla popsána problematika stavového prostoru. Byla navržena a vytvořena aplikace sloužící k názorné demonstraci prohledávacích algoritmů. Byly také navrženy typické úlohy. Pro neinformované hledání se jednalo o stromový graf, pro informované hledání byl navržen graf ukázaný v praktické části práce. Všechny popsané algoritmy byly také demonstrovány na vytvořeném vlastním grafu. Grafické uživatelské rozhraní je v rámci možností intuitivní a jeho ovládání je popsáno v nápovědě. Dosažené výsledky byly vyhodnoceny za pomoci průměrování výsledků generovaných grafů. Byla navržena další možná rozšíření aplikace.

Vytvořená aplikace, její zdrojový kód, dokumentace a všechny ukázkové grafy jsou obsaženy na přiloženém CD.

9 Literatura

A* search algorithm. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-12]. Dostupné z: https://en.wikipedia.org/wiki/A*_search_algorithm

Artificial intelligence. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-11]. Dostupné z: https://en.wikipedia.org/wiki/Artificial_intelligence

ČÁPKA, David a . Úvod do UML [online]. [cit. 2017-05-11]. Dostupné z: http://www.itnetwork.cz/navrhove-vzory/uml/uml-uvod-historie-vyznam-a-diagramy

Eclipse [online]. The Eclipse Foundation, 2017 [cit. 2017-05-15]. Dostupné z: https://eclipse.org/ide/

EUROZPRÁVY.CZ. Čeští vědci vyvinuli přelomový algoritmus [online]. 2017 [cit. 2017-05-11]. Dostupné z: http://veda-atechnika.eurozpravy.cz/technika/183791-cesti-vedci-vyvinuli-prelomovy-algoritmus-pocitac-poprve-porazil-hrace-pokeru/

GOSLING, James. The Java language specification [online]. 3rd ed. Upper Saddle River, NJ: Addison-Wesley, c2005 [cit. 2017-05-12]. ISBN 03-212-4678-0. Dostupné z: http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

HORDĚJČUK, Vojtěch. Algoritmus A-Star [online]. [cit. 2017-05-11]. Dostupné z: http://voho.eu/wiki/algoritmus-a-star/

KOTT, Petr. Grafové algoritmy [online]. [cit. 2017-05-11]. Dostupné z: http://statnice.dqd.cz/home:inf:ap17

MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ a Jiří LAŽANSKÝ. Umělá inteligence. Praha: Academia, 2013. ISBN 978-80-200-2276-9.

MENDELU. Algoritmus a jeho vlastnosti [online]. 2017 [cit. 2017-05-14]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7316

MENDELU. Expertní systémy [online]. 2017 [cit. 2017-05-14]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21856

NECKÁŘ, Jan a . Java: Grafické rozhraní [online]. 2016 [cit. 2017-05-11]. Dostupné z: https://www.algoritmy.net/article/39898/Graficke-rozhrani-23

NECKÁŘ, Jan a . Java: Úvod [online]. 2016 [cit. 2017-05-11]. Dostupné z: https://www.algoritmy.net/article/21340/Uvod-1

RUSSELL, Stuart J. a Peter NORVIG. Artificial intelligence: a modern approach. 3rd ed. Harlow: Pearson Education, c2014. ISBN 978-1-29202-420-2.

State space search. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-11]. Dostupné z: https://en.wikipedia.org/wiki/State_space_search

TIOBE Programming Community Index for May 2013 [online]. TIOBE Software BV, květen 2013, [cit. 2017-05-11]

Use case. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-05-12]. Dostupné z: https://en.wikipedia.org/wiki/Use_case

What is Maven? [online]. The Apache Software Foundation, 2017 [cit. 2017-05-15]. Dostupné z: http://maven.apache.org/what-is-maven.html

10 Seznam obrázků

Přílohy

A Komentovaný zdrojový kód: Prohledávání do šířky

```
protected void search() {
      Vertex active = null;
      // přidam počáteční uzel do opened
      opened.add(start);
      // while cyklus hledání cíle
      while (!opened.isEmpty()) {
            // vyberu uzel z opened
            active = opened.poll();
            // přidám ho do closed
            closed.add(active);
            // V následující podmínce kontroluji nalezení cíle
            if (active == goal) {
                  resultV.add(active);
                  break;
            }
            // Pro všechny následniky aktivního uzlu
            for (Vertex vertex : getSucces-
            sors(g.getSuccessors(active))) {
                  // Pokud následník není v opened ani v closed
                  if (!(opened.contains(vertex) ||
                  closed.contains(vertex))) {
                        // Přidám následníka do opened
                        opened.add(vertex);
                        // Nastavím následníkovi aktivní uzel jako
                        předka
                              vertex.setPrevious(active);
                  }
            }
      }
}
```

B Komentovaný zdrojový kód: Prohledávání do hloubky

```
protected void search() {
      Vertex active = null;
      // přidam počáteční uzel do opened
      opened.push(start);
      // while cyklus hledání cíle
      while (!opened.isEmpty()) {
            // vyberu uzel z opened
            active = opened.pop();
            // V následující podmínce kontroluji nalezení cíle
            if (active == goal) {
                  resultV.add(active);
                  break;
            }
            // Pokud není aktivní uzel v closed
            if (!closed.contains(active)) {
                  // Přidej aktivní uzel do closed
                  closed.add(active);
                  // Pro všechny potomky aktivního uzlu
                  for (Vertex vertex : getSucces-
                  sors(g.getSuccessors(active))) {
                        // Pokud následník není v opened ani v
                        closed
                        if (!(opened.contains(vertex) ||
                        closed.contains(vertex))) {
                              // Přidám následníka do opened
                              opened.push(vertex);
                              // Nastavím následníkovi aktivní uzel
                              jako předka
                              vertex.setPrevious(active);
                        }
                  }
            }
      }
}
```

C Komentovaný zdrojový kód: Hladový algoritmus

```
protected void search() {
	Vertex active = null;
	// přidam počáteční uzel do opened
	opened.add(start);
	// ohodnotím počáteční uzel
	evaluateVertex(start);
	// while cyklus hledání cíle
	while (!opened.isEmpty()) {
		// vyberu nejlepší uzel z opened
		active = getBest(opened);
		// V následující podmínce kontroluji nalezení cíle
		if (active == goal) {
			resultV.add(active);
			break;
		}
		// Pokud aktivní uzel není v closed, tak ho přidám do
		closed
		if (!closed.contains(active))
			closed.add(active);
		// odeberu aktivní uzel z Opened
		opened.remove(active);
		// Pro všechny následniky aktivního uzlu
		for (Vertex vertex : g.getSuccessors(active)) {
			// Pokud následník není v closed
			if (!closed.contains(vertex)) {
				// Ohodnotím následníka
				evaluateVertex(vertex);
				// Přidám následníka do opened
				opened.add(vertex);
				// Nastavím následníkovi aktivní uzel jako
				předka
				vertex.setPrevious(active);
			}
		}
	}
}
```

D Komentovaný zdrojový kód: A*

```
protected void search() {
	double pomG = 0;
	Vertex active = null;
	// podmínka složící k určení lepší cesty
	boolean better = false;
	// přidam počáteční uzel do opened
	opened.add(start);
	// ohodnotím počáteční uzel
	evaluateVertex(start);
	start.setG(0);
	start.setF();
	// while cyklus hledání cíle
	while (!opened.isEmpty()) {
		// vyberu nejlepší uzel z opened
		active = getBest(opened);
		// V následující podmínce kontroluji nalezení cíle
		if (goal == active) {
			resultV.add(active);
			break;
		}
		// odeberu aktivní uzel z Opened
		opened.remove(active);
		// Přidám následníka do opened
		closed.add(active);
		// Pro všechny následniky aktivního uzlu
		for (Vertex vertex : g.getSuccessors(active)) {
			// Ohodnotím následníka
			evaluateVertex(vertex);
			// Pokud je následník v closed, tak přeskočím
			zbytek iterace
			if (closed.contains(vertex)) {
				continue;
			}
			// Vypočítám G pro následníka
			pomG = (active.getG() + g.findEdge(active, ver-
			tex).getWeight());
			// Pokud není následník v opened
			if (!opened.contains(vertex)) {
				// přidám ho do opened
				opened.add(vertex);
				// nastavím better na true
				better = true;
```

```
                } else if (pomG < vertex.getG()) {
                    better = true;
                }
                // Pokud je better true
                if (better) {
                    // Nastavím nového předchůdce
                    vertex.setPrevious(active);
                    // nastavím nové G
                    vertex.setG(pomG);
                    // nastavím nove F
                    vertex.setF();
                    // nastavím better na false
                    better = false;
                }
            }
        }
    }
```