Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

BAKALÁRSKA PRÁCA

Tomáš Novella

Grid-Based Path Planning

Katedra teoretické informatiky a matematické logiky

Vedúci bakalárskej práce: Mgr. Tomáš Balyo

Študijný program: Informatika

Študijný obor: Obecná informatika

Praha 2013

Poďakovanie.

Prehlasujem, že som túto prácu vypracoval samostatne a výhradne s použitím citovaných prameňov, literatúry a ďalších odborných zdrojov.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona a v platnom znení, obzvlášť skutočnosť, že Univerzita Karlova v Prahe má právo na uzavretie licenčnej zmluvy o použití tejto práce ako školského diela podľa §60 odst. 1 autorského zákona.

V .... dne .... Podpis autora

Názov práce: Grid-Based Path Planning

Autor: Tomáš Novella

Katedra: Katedra teoretické informatiky a matematické logiky

Vedúci bakalárskej práce: Mgr. Tomáš Balyo, Katedra teoretické informatiky a matematické logiky

Abstrakt:

Kľúčové slová:

Title: Grid-Based Path Planning

Author: Tomáš Novella

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Tomáš Balyo, Department of Theoretical Computer Science and Mathematical Logic

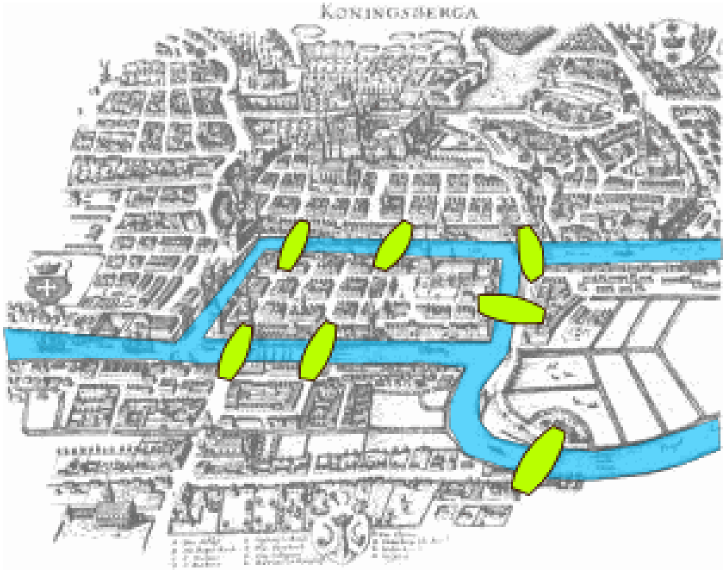
Abstract:

Keywords:

Obsah

Úvod

Slávna Eulerova úloha siedmych mostov v Kaliningrade [1] sa považuje za prvú prácu, ktorá zaviedla teóriu grafov. Úlohou je prejsť po týchto siedmych mostoch tak, aby sme po každom prešli práve raz.



Obr. 1: Sedem mostov v Kaliningrade

Od tej doby sa využitie teórie grafov značne rozšírilo a v dnešnej dobe patrí medzi významné a rozpracované teórie. V modernej dobe je jedným z jej najdôležitejších problémov hľadanie najkratšej cesty. Najčastejšie sa s nimi stretávame pri plánovaní trasy v GPS navigácii. Medzi najvýznamnešie práce považujeme práce od Dijkstru [2] a Floyd-Warshalla.

So začiatkom fenoménu počítačových hier a umelej inteligencie sa do povedomia dostal špeciálny typ grafu – mriežkový graf, využívaný ako herná mapa. V hrách často trebalo nájsť cestu pre počítačom ovládanú postavičku z miesta A do miesta B. Nakoľko je väčšina hier komerčná, algoritmy využívané v hrách boli a sú taktiež komerčné. Dôsledkom toho nie sú verejne publikované a porovnané rôzne prístupy a algoritmy na vyhľadávanie najkratších ciest v mriežkových mapách. A keď už aj sú, tak práce používajú rôzne mapy na bechmarking a teda neexistuje žiadna globálna porovnávacia štúdia týchto prístupov.

Súťaž *Grid-Based Path Planning Competition* [8] sa snaží tento problém vyriešiť tým, že porovnáva rôzne algoritmy na veľkej množine máp použitých v známych počítačových hrách a vyhodnocuje ich úspešnosť v rámci viacerých kategórií.

Cieľom tejto práce je spraviť prehľad doterajších prístupov k tomuto problému a prispieť vlastným algoritmom do sútaže a niekoľkými vylepšeniami k doterajším prístupom hľadania najkratšej cesty na mriežkových grafoch.

V prvej kapitole si zavedieme kľúčové termíny a popíšeme problém formálne. Na konci kapitoly spomenieme súťaž, ktorej sa daný algoritmus zúčastnil.

Druhá kapitola je zameraná na vytvorenie prehľadu kľúčových algoritmov používaných na riešenie problému.

V tretej kapitole navrhneme vlastné riešenie založené na poznatkoch popísaných v druhej kapitole s pridaním vlastných vylepšení.

Vo štvrtej kapitole toto riešenie porovnáme s dosavadnými.

1. Zadanie problému a cieľové požiadavky

1.1 Úvodné definície a značenia

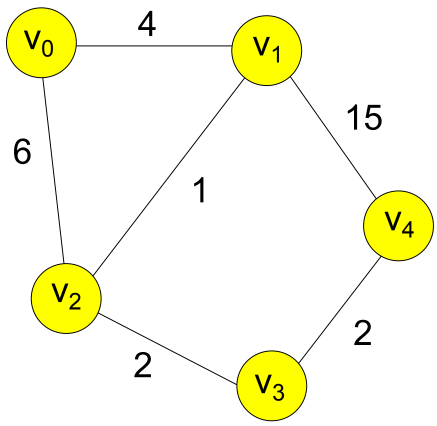
Na začiatok si zaveďme niektoré dôležité pojmy z teórie grafov. Úlohu so všetkými jej špecifikami si ozrejmíme v nasledujúcich podkapitolách.

Definícia. *Graf G je usporiadaná dvojica (V, E), kde V označuje množinu vrcholov(vertices) a $E \subseteq V \times V$ označuje množinu hrán (edges). Značíme $G =$ (V, E).*

Poznámka 1. *Hrana je jednoznačne určená dvojicou vrcholov.*

Definícia. *Ohodnotený graf (G, w) je graf s spolu s reálnou funkciou (tzv. ohodnotením) $w : E(G) \rightarrow \mathbb{R}$, kde w je funkcia, ktorá každej hrane priradí reálne číslo takzvanú* dĺžku, *alebo* hodnotu*hrany.*

Ukážka obecného ohodnoteného grafu je na obrázku 1.1.



Obr. 1.1: Ohodnotený graf

Pri hľadaní najkratšej cesty v grafe pracujeme s pojmamy, ako sú *cesta* a *najkratšia cesta.*

Definícia. *Cesta P z vrcholu v_0 do vrcholu v_n v grafe G je postupnosť $P = (v_0, e_1, v_1, \ldots, e_n, v_n)$, pre ktorú platí $e_i = \{v_{i-i}, v_i\}$ a taktiež $v_i \neq v_j$ pre každé $i \neq j$.*

Všimnime si, že na ceste nenavštívime žiaden vrchol dvakrát a teda cesta neobsahuje kružnice.

Definícia. *Dĺžka cesty P z vrcholu v_0 do vrcholu v_n v ohodnotenom grafe (G, w) je súčet dĺžok hrán, ktoré sa na ceste nachádzajú.*

Samozrejme, medzi dvoma vrcholmi môže existovať viacero ciest.

Definícia. *Najkratšia cesta P z vrcholu v_0 do vrcholu v_n v ohodnotenom grafe (G, w) je cesta z vrcholu v_0 do vrcholu v_n s najmenšou vzdialenosťou.*

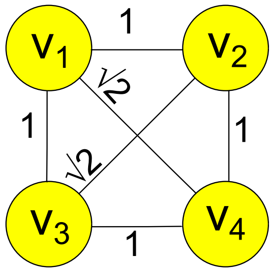
1.2 Mriežkový graf

Keď sme si už zaviedli kľúčové pojmy, prejdime k samotnému zadaniu úlohy. Ako sme už spomínali, problém budeme riešit na tzv. mriežkových grafoch. Čo je mriežkový graf a v čom sa od obecného grafu odlišuje?

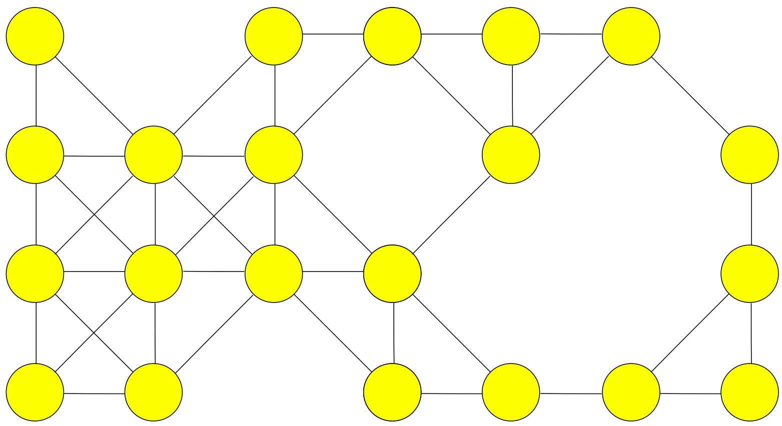
V hernej praxi predstavuje mriežkový graf mapu hracej plochy, s ktorou sa stretávame v najrôznejších hrách, ako je Warcraft, Startcraft, Dragon Age a podobne.

Ide o špeciálny a dosť obmedzený typ grafu. Vizuálne si ho môžme predstaviť ako konečný graf v ktorom sú vrcholy rozostúpené v tvare mriežky a hrana je stále medzi dvojicami susedných vrcholov vo všetkých ôsmych smeroch. Dĺžka vodorovnej alebo zvislej hrany je 1 a dĺžka šikmej hrany je $\sqrt{2}$.

Poznámka 2. *Mriežkový graf patrí medzi riedke grafy, pretože má veľmi malý počet hrán (lineárny od počtu vrcholov).*



Obr. 1.2: Mriežkový graf 2x2



Obr. 1.3: Mriežkový graf bez označenia vrcholov a dĺžok hrán

Zadefinujme si teraz mriežkový graf formálne.

Definícia. *Mriežkový graf rozmerov* $m \times n$ *je ohodnotený graf s ohodnotením* w *s* $m*n$ *vrcholmi očíslovanými od* $v_{1,1}$ *až po* $v_{m,n}$ *s priamymi hranami* j *v tvare* $\{v_{a,b}, v_{a,b+1}\}$, $\{v_{a,b}, v_{a+1,b}\}$, *kde* $w(j) = 1$ *a šikmými hranami* s *v tvare* $\{v_{a,b}, v_{a+1,b+1}\}$, $\{v_{a,b}, v_{a-1,b+1}\}$, *kde* $w(s) = \sqrt{2}$.

Poznámka 3. *Mriežkový graf sa dá reprezentovať ako matica* $m \times n$ *nad telesom* $\mathbb{Z}_2$, *kde jednotky predstavujú vrcholy. Príklad maticovej reprezentácie grafu rozmerov* 4×6 *je na obrázku 1.2.*

$$G = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Obr. 1.4: Maticová reprezentácia mriežkového grafu

1.3 GPPC: Grid-Based Path Planning Competition

Algoritmus navrhnutý a naprogramovaný v tejto práci bol zaradený do súťaže **GPPC**, ktorá sa koná približne 2 krát ročne.

1.3.1 Špecifiká súťaže, limity

Mriežkové grafy budú mať rozmery maximálne 2048 × 2048. Súťaž bude rozdelená do dvoch fáz — fázy predspracovania grafu (pre-processing) a fázy testovania. Na predspracovanie grafu bude vyhradený čas maximálne 30 minút a program si svoje dáta uloží na disk do súboru o veľkosti maximálne 50MB. Potom vo fáze testovania budé dostávať požiadavky na nájdenie najkratšej cesty.

Úlohou súťaže je naimplementovať tieto tri funkcie.

```
void PreprocessMap(std::vector<bool> &bits, int width, int height,
    const char *filename);

void *PrepareForSearch(std::vector<bool> &bits, int width, int
    height,const char *filename);
bool GetPath(void *data, xyLoc s, xyLoc g, std::vector &path);
```

1.3.2 Kritériá súťaže, hodnotenie programov

Na algoritmus môžeme klásť rôzne požiadavky, ktoré si častokrát navzájom protirečia, preto programy posudzujeme podľa viacerých kritérií.

- Celkový čas na nájdenie cesty.
- Čas na nájdenie prvých 20 tich krokov.
- Dĺžka cesty (zohľadnená suboptimalita).
- Maximálny čas vrátenia hociktorej časti cesty.

Testovací počítač má 12 GB RAM pamäti a dva 2.4 Ghz Intel Xeon E5620 procesory.

2. Prehľad algoritmov

Na hľadanie najkratších ciest v grafe poznáme mnoho algoritmov, ktoré rozdeľujeme do týchto troch [3] skupín:

- Point To Point Shortest Path(P2PSP) - hľadajú najkratšiu cestu medzi dvoma zadanými bodmi.
- Single Source Shortest Path(SSSP) - pre daný vrchol v hľadajú najkratšiu cestu do všetkých vrcholov grafu.
- All Pairs Shortest Path (APSP)- skúmajú najkratšiu cestu medzi všetkými dvojicami vrcholov.

Tieto problémy sú na obecných grafoch NP-ťažké. Napriek tomu na mriežkových grafoch (kde sú vzdialenosti medzi vrcholmi vždy kladné) existujú algoritmy v polynomiálnom čase.

V práci budeme ďalej zaoberať riešením prvého problému (Point to Point Shortest Path).

V tejto kapitole popíšeme algoritmy, ktoré sú použiteľné na grafoch s nezápornými dĺžkami hrán. TODO?? prerob

TODO?? POTIAL VSETKO PREPISANE a myslim, ze clekom prijemne citatelne.

2.1 Kritéria efektivity algoritmu

Na porovnanie efektivity algoritmou slúži v teoretickej informatike odhad asymptotickej složitosti [7]. Tento odhad je veľmi užitočný v teoretickej informatike a veľmi často algoritmus s lepšou zložitosťou je v praxi rýchlejší. Nie je to ale pravidlom a teda potrebujeme zaviesť ďalšie kritériá, ktoré presnejšie popíšu a porovnajú správanie algoritmov praxi. Kritéria, podľa ktorých budeme porovnávať efektivitu algoritmov sú teda nasledovné:

- Asymptotická zložitosť.
- Počet navštívených vrcholov.
- Reálny čas behu algoritmu.

2.2 Dijkstrov algoritmus

Medzi základné algoritmy typu SSSP patrí Dijkstrov algoritmus [2] popísaný už v roku 1959. Miernu modifikáciu pôvodného algoritmu môžeme vidieť na (Algoritmus 1). ASK?? spravil som citacie spravne? Patrí medzi relaxačné algoritmy a zbehne korektne na grafoch s nezápornými hranami.

Pri hľadaní cesty z vrcholu s do vrcholu t prechádzame postupne vrcholy s neklesajúcou vzdialenosťou od s, až dokým sa nedostaneme k cieľovému vrcholu t.

ASK?? rozlisujeme tri stavy vrcholov, co je halda blablabla mam popisat aj slovne???

Algoritmus 1 Dijkstra: nájdi najkratšiu cestu medzi dvoma bodmi s a t

Vstup: graf G

Výstup: dĺžková funkcia d obsahujúca najkratšie cesty z vrcholu s do vrcholov grafu

$d(*) \leftarrow \infty$
$stav(*) \leftarrow$ NENAVŠTÍVENÝ
{pridám počiatok}
$d(s) \leftarrow 0$
$stav(s) \leftarrow$ OTVORENÝ
Heap H
$Insert(H, s)$
while H not empty **do**
  // vyberiem v – najbližší otvorený vrchol
  $v \leftarrow ExtractMin(H)$
  while $stav(v) \neq$ OTVORENÝ **do**
    $v \leftarrow ExtractMin(H)$
  end while
  // zrelaxujeme vrchol v
  for all e, $e = (v, u)$ **do**
    if $d(u) > d(v) + l(v, u)$ **then**
      $Insert(H, v)$
      $stav(u) \leftarrow$ OTVORENÝ
      $d(u) \leftarrow d(v) + l(v, u)$
    end if
  end for
end while

Veta 1. *V dijkstrovom algoritme uzatvárame každý dosiahnuteľný vrchol práve raz.*

2.2.1 Zložitosť

Na každý vrchol zavoláme operáciu $Insert$ maximálne $deg(v)$-krát a teda počet všetkých zavolaní tejto operácie bude rádovo $\mathrm{O}\big(\sum_v deg(v)\big) = \mathrm{O}\big(m\big)$. Zo štruktúry môžme vybrať maximálne toľko prvkov, koľko sme tam vložili a teda aj volania $ExtractMin$ trvajú $\mathrm{O}\big(m\big)$.

Algoritmus teda zbehne v čase $\mathrm{O}\big(mT_i + mT_e\big)$, kde T_i odpovedá času na vloženie prvku a T_d odpovedá času na vybranie najmenšieho prvku.

To znamená, že zložitosť algoritmu závisí od zložitosti operácií $Insert$ a $ExtractMin$. Na riedke grafy je obecne v praxi najvýhodnejšie použiť binárnu haldu, ktorej obe operácie trvajú $\mathrm{O}\big(\log n\big)$ a celkový čas je teda $\mathrm{O}\big(m \log n\big)$ Prehľad štruktúr aj so zložitosťami operácií $Insert$ a $ExtractMin$ sa nachádza napr. na [3].

2.2.2 Halda na mriežkovom grafe

Nakoľko mriežkový graf je veľmi špeciálny typ grafu, vieme niektoré jeho vlastnosti využiť na to, aby sme vytvorili štruktúru, ktorá zvládne obe operácie v konštantnom čase.

Na konštrukciu tejto štruktúry (viď [10]) budeme potrebovať nasledujúcu vetu.

Veta 2. *Pokiaľ sme v Dijkstrovom algoritme uzavreli vrchol u so vzdialenosťou $d(u)$ a najkratšia hrana v grafe má dĺžku ϵ, tak môžme taktiež uzavrieť všetky vrcholy v so vzdialenosťami $d(v) \in (d, d+\epsilon)$.*

Dôkaz. Do haldy vieme pridávať len vrcholy so vzdialenosťami aspoň $d+\epsilon$ (kratšia hrana tam už nie je), ale tie už cestu k vrcholom so vzdialenosťami $d_v \in (d, d+\epsilon)$ skrátiť nemôžu. □

Dôsledok 1. *Keď uzavrieme vrchol so vzdialenosťou d_u, môžme uzavrieť aj vrcholy vo vzdialenosťami menšími, ako $d_u + \epsilon$ pričom poradie je nezávislé od skutočnej vzdialenosti vrcholov.*

Príklad 1. *Dĺžka ϵ najkratšej hrany v mriežkovom grafe je 1. Je to dĺžka akejkoľvek vodorovnej, alebo zvislej hrany. Keď teda uzavrieme vrchol so vzdialenosťou d_u, môžme uzavrieť aj vrcholy so vzdialenosťami menšími, ako je $d_u + 1$ a to v ľubovoľnom poradí.*

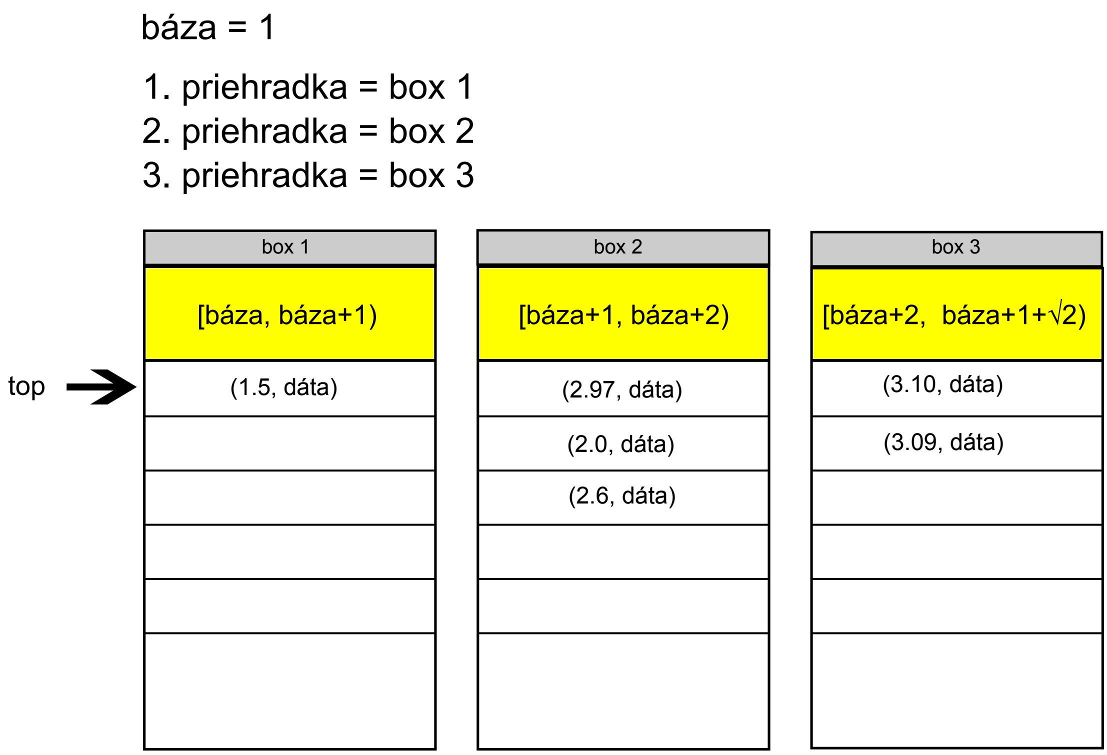
Tieto veci vieme výborne využiť pri konštrukcii štruktúry zvanej *priehradková halda*. Tá, využijúc vyššieuvedenú vetu, uzatvára a pridáva vrcholy bez porušenia akejkoľvek konzistencie behu algoritmu.

ASK?? mam to supnut to kapitoly o 'mojom' algoritme? Totizto o priehradkovej strukture sa da docitat vsade, neni to moj patent, ale na druhu stranu ja som tu strukturu implenmentoval konkretne ma moj mriezkovy graf...

2.2.3 Popis haldy

Majme haldu s tromi priehradkami, pričom rozdiel najmenšieho a najväčšieho čísla bude ostro menší, ako 1. To nám zaručí, že pokiaľ uzavrieme akýkoľvek vrchol v priehradke, môžme uzavrieť aj všetky ostatné vrcholy. Viď Príklad 1. Pre jednoduchosť implementácie nech priehradka uchováva vrcholy so vzdialenosťami v intervale $[k, k+1), k \in \mathbb{N}$. Problémom ostalo ešte sa presvedčiť, že tri priehradky sú postačujúce. Predstavme si najhorší prípad, totižto, že vyberáme vrchol z prvej priehradky a otvárame všetkých jeho susedov. Pokiaľ sa bude jednať o suseda priečneho, tak vieme, že jeho vzdialenosť od začiatku je menšia, ako $k + 1 + \sqrt{2}$ a tento vrchol patrí to tretej priehradky.

Ilustrujme si to na obrázkových príkladoch. Príklad troj priehradkovej haldy vidíme na obrázku 2.1. ASK?? troj–priehradkovej? Halda uchováva premennú *baza*, ktorá definuje bázu od ktorej sa ostatné intvarvaly odvíjajú, ktorá môže byť do haldy vkladaná. Okrem nej, uchováva tri smerníky na tri po sebe idúce priehradky priehradky a smerník na vrchol haldy, zvaný *top*.

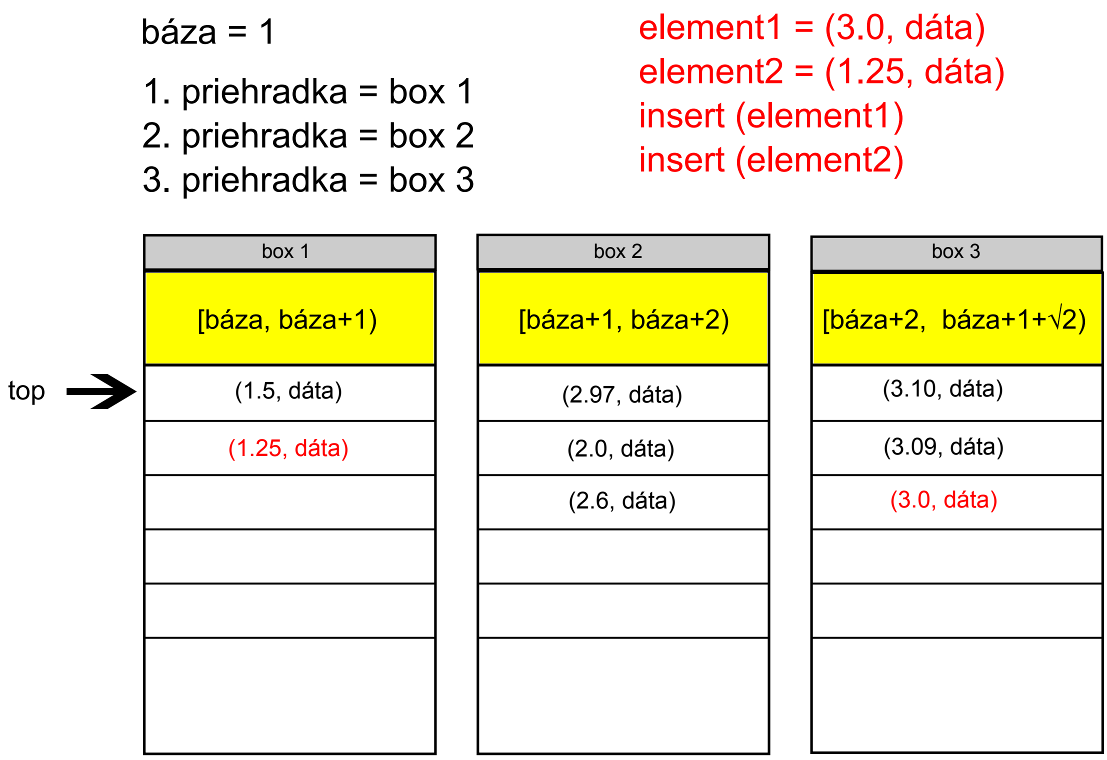


Obr. 2.1: Priehradky

Pridanie dvoch prvkov je ilustrované na obrázku 2.2. Priehradka, do ktorej má byť prvok s danou vzdialenosťou vložený sa vypočíta podľa vzorca: $\lfloor vzdialenostPrvku - baza + 1 \rfloor$.

Zmazanie prvku vidíme na obrázku 2.3. Celé zmazanie spočíva v inkrementácii ukazovateľa na vrchol priehradky.

Pokiaľ sa v priehradke nachádza jediný prvok a ten chceme vybrať, tak nám inkrementácia premennej *top* v priehradke nestačí. Musíme prehodiť priehradky. Druhú priehradku presunúť na miesto prvej, tretiu na miesto druhej a prvú umiestniť nakoniec. Viď obrázok 2.4. Nakoniec musíme zvýšit bázickú vzdialenosť. Presunutie týchto priehradok sa samozrejme uskutočňuje cez prehodenie



Obr. 2.2: Insert

smerníkov. Keďže máme konštatný počet priehradok, tak aj táto operácia trvá konštantný čas.

2.3 A*

Ďalší algoritmus, ktorým sa budeme zaoberať je algoritmus A* [4] prvykrát popísaný v roku 1968.

Tento algoritmus vychádza z Dijkstrovho algoritmu a je mu veľmi podobný. Hlavný rozdiel medzi týmito algoritmami je, že kým Dijkstrov algoritmus vyberá z haldy vrcholy s neklesajúcou vzdialenosťou $d(v)$ od počiatku, tak algoritmus A* vyberá prvky s neklesajúcou vzdialenosťou $f(v) := d(s, v) + h(v, t)$, kde $h(v, t)$ značí heuristickú funkciu, ktorá je dolným odhadom vzdialenosti od vrcholu v do cieľa t. Obrátene, Dikstrov algoritmus si vieme predstaviť ako algoritmus A*, kde $\forall v \in G : h(v, t) = 0$.

Použitá heuristická funkcia má dopad na počet prehľadaných vrcholov a teda do zásadnej miery ovplyvňuje výkon algoritmu.

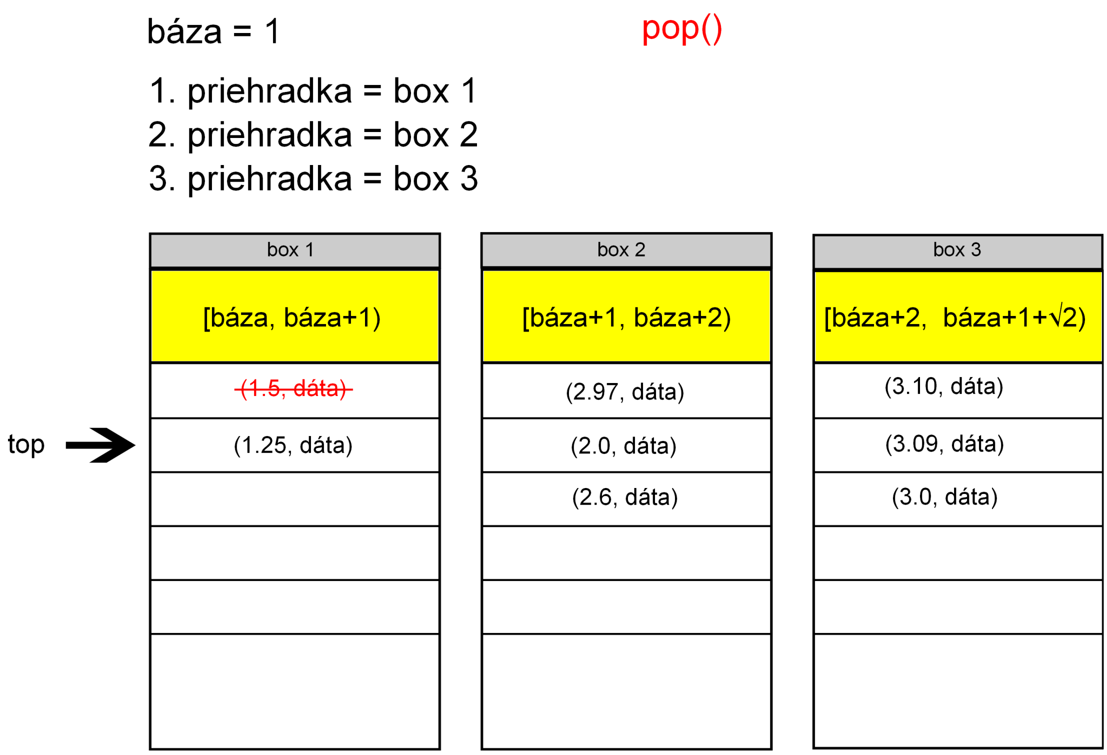
2.3.1 Heuristická funkcia

Heuristická funkcia nemôže byť ľubovoľná a musí splňovať niektoré podmienky [5] [6].

TODO?? potencialy blablabla

Ako sme už spomínali, voľba heuristickej funkcie je kľúčová časť pri implementácii algoritmu.

Najčastejšie možnosti voľby heuristickej funkcie sú tieto:



Obr. 2.3: Pop (1)

- Euklidovská vzdialenosť.
- Trojuholníková nerovnosť, tzv. *landmarks*.

Euklidovská vzdialenosť Euklidovská vzdialenosť je najjednoduchšie implementovateľná heuristická funkcia. Na väčšine jednoduchých grafov s malým počtom prekážok vracia dobré výsledky. Problém nastáva na grafoch, kde začiatok a koniec cesty sú geometricky blízko seba, hoci ich skutočná vzdialenosť je veľká. Viď napr.

Landmarks a trojuholníková nerovnosť Nevýhoda použitia euklidovskej heuristickej funkcie je na obecných mapách zjavná. To motivovalo vymyslieť heuristiku, ktorá lepšie odráža vzdialenosti v grafe.

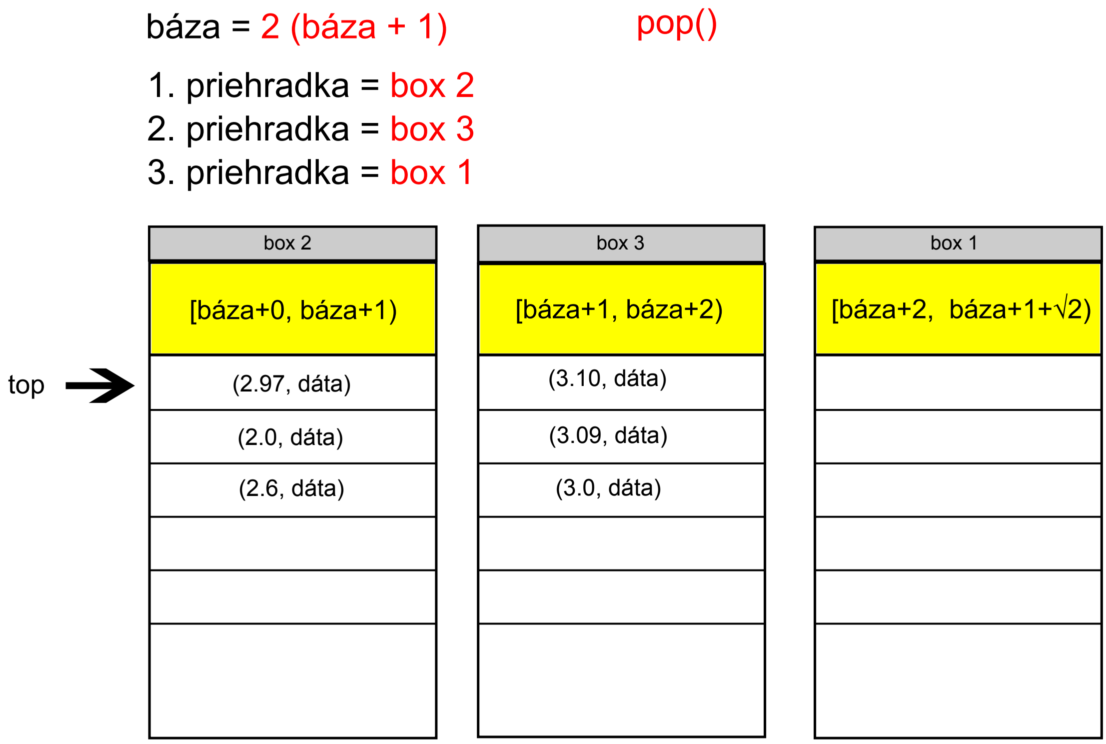
Jednou z týchto heuristík je počítanie dolného odhadu pomocou tzv. *landmarks*.

Landmarks sú vybrané vrcholy v grafe, z ktorých je následne prepočítaná najkratšia vzdialenosť do všetkych ostatných vrcholov grafu. Kedže jeden prechod grafu vieme Dijkstrovým algoritmom s priehradkami vykonať za lineárny čas, predpočítanie k landmarkov trvá $\mathrm{O}(kn)$, kde n značí počet vrcholov grafu.

TODO?? ako to funguje...

Možnosti voľby landmarkov Pri voľbe landmarkov sú dva faktory: počet a rozmiestnenie.

Príklad 2 (na počte záleží)**.** *Pokiaľ zvolíme málo landmarkov, tak dolný odhad nebude presný. Pokiaľ ich zvolíme priveľa, tak prepočet vzdialenosti cez každý landmark pre každý vrchol zaberie veľa času.*

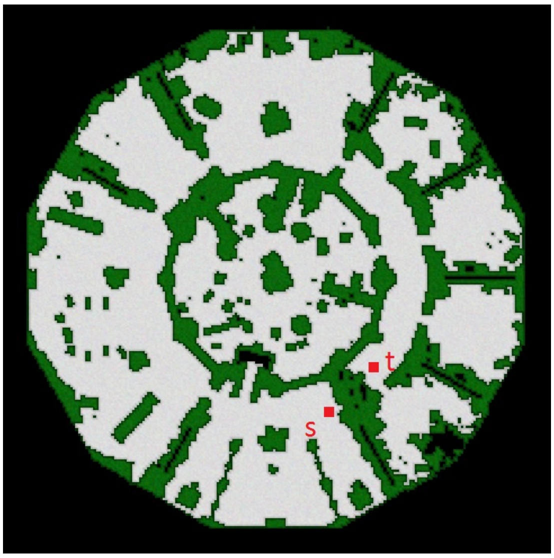


Obr. 2.4: Pop (2)

Príklad 3 (na rozmiestnení záleží)**.** *Pokiaľ zvolíme všetky landmarky hneď pri sebe, tak heuristika nám nebude vraciať dostatočne presné dolné odhady na vzdialenosť vrcholov, ktoré sú ďaleko od landmarkov.*

ASK?? jak spravi dalsie hlbsie cislovanie odstavcov???

TODO??potencialy euklid landmarks landmark selection priblem s priehradkami



Obr. 2.5: Mapa, na ktorej euklidovská heuristika zlyhá.

3. Nový algoritmus: NovellA*

3.1 Zlepšenie výkonu v niektorých prípadoch

Nie všetky najkratšie cesty ale musia obchádzať veľa prekážok. V mnohých prípadoch nemusí medzi počiatočným a koncovým bodom ležať žiadna prekážka, a teda cesty sú veľmi priamočiare. To sa pokúsime využiť na zlepšenie výkonu algoritmu.

Pre ľahšie vyjadrovanie si zaveďme definíciu *mriežkového grafu bez prekážok*.

Definícia. *Mriežkový graf je bez prekážok pokiaľ medzi každými dvoma susednými vrcholmi existuje hrana.*

Kvôli lepšej prehľadnosti budeme mriežkový graf bez prekážok nazývať aj *obdĺžnik*. Intuitívne, kvôli jeho vizuálnej interpretácii.

Majme mriežkový graf bez prekážok a hľadajme najkratšiu cestu medzi bodmi $s = (x_s, y_s), t = (x_t, y_t)$. V tomto prípade vieme nájsť najkratšiu cestu veľmi jednoducho.

Algoritmus 2 Nájdi najkratšiu cestu medzi dvoma bodmi s a t na mriežkovom grafe bez prekážok

---

Vstup: $s = (x_s, y_s), t = (x_t, y_t)$
Výstup: *path*

path.append((x_s, y_s)) {pridám počiatok}
while $x_s \neq x_t \vee y_s \neq y_t$ **do**
    if $x_s < x_t$ **then**
        $x_s \leftarrow x_s + 1$
    else if $x_s > x_t$ **then**
        $x_s \leftarrow x_s - 1$
    end if
    if $y_s < y_t$ **then**
        $y_s \leftarrow y_s + 1$
    else if $y_s > y_t$ **then**
        $y_s \leftarrow y_s - 1$
    end if
    path.append((x_s, y_s))
end while

Teda, jednoducho povedané: keď sa počiatočný a koncový bod líšia v jednej súradnici, tak sa posúvame priamočiaro, keď sa líšia v oboch, tak sa posúvame šikmo.

Pokiaľ si zadefinujeme $dx := |x_t - x_s|$ a $dy := |y_t - y_s|$, tak počet vrcholov, ktorými cesta prechádza vieme zhora odhadnúť, ako $\max(dx, dy)$. Jej vzdialenosť vieme zistiť v čase $\mathrm{O}\big(\max(dx, dy)\big)$ Na zistenie vzdialenosti v každom kroku nám stači konštantná pamäť.

TODO?? Poznamka, ze nepotrebujem na to obdlzniky, mozem robit aj komplikovanejsie utvary, ale by to sa blbo hladalo... ledaze...

Skúsme to teda nejak využiť. Pokiaľ vieme, že počiatočný aj koncový bod ležia v jednom obdĺžniku, tak máme problém vyriešený. Jediným problémom ostalo takéto obdĺžniky nájsť.

3.2 Hľadáme obdĺžniky

3.2.1 Proporcie obdĺžnikov

Dôležitou otázkou je, na akých vlastnostiach obdĺžnikov záleží. Uvažujme nasledujúci motivačný príklad.

Príklad 4. *Majme na mape nájdené dva obdĺžniky, ktorých celkový obsah je 10. Predstavme si tieto dva prípady. V prvom prípade je obsah prvého 9 a druhého 1, v druhom prípade sú obsahy 6 a 4. Chceme maximalizovať pravdepodobnosť toho, aby pri voľbe dvoch náhodných bodov boli obe v rovnakom obdĺžniku.*

Úlohu vieme zobecniť na klasickú pravdepodobnostno-optimalizačnú úlohu.

Príklad 5. *Majme k ekvivalenčných tried na množine s n prvkami. Ako zvoliť ekvivalenčné triedy tak, aby pri voľbe dvoch náhodných prvkov bola pravdepodobnosť toho, že oba prvky budú v tej istej ekvivalenčnej triede čo najvyššia?*

Poznámka 4. *Ekvivalenčnú triedu predstavuje obdĺžnik a množinu predstavuje množina vrcholov grafu. Alternatívne sa môžeme na úlohu pozerať ako na problém farbenia guličiek čo najmenším počtom farieb.*

Zapíšme túto úlohu formálne.

Majme n-prvkovú množinu $Prv = \{x_1, \ldots, x_n\}$, k-prvkovú množinu ekvivalenčných tried $Ek = \{ek_1, \ldots, ek_k\}$, veľkosť triedy $\|ek_i\|$ označme k_i a zaveďme funkciu $f\colon Prv \to Ek$ ktorá roztriedi prvky do ekvivalenčných tried.

Označme výberový priestor $\Omega = \{(x_a, x_b) | x_a, x_b \in Prv, a \neq b\}$ Udalosťou A_i nazveme jav, v ktorom oba prvky patria do tej istej ekvivalenčnej triedy ek_i, teda $A_i = \{(x_a, x_b) | x_a, x_b \in Prv, a \neq b, f(x_a) = f(x_b) = ek_i\}$ Jav $A = \bigcup_{i=1}^{k} A_i$ teda nastáva práve vtedy, keď oba vybrané prvky patria do rovnakej triedy.

Úlohou je teda navrhnúť funkciu f tak, aby pravdepodobnosť $P[A]$ bola čo najvyššia. Keďže udalosti A_i sú nezlučiteľné, môžme písať $P[A] = P[\bigcup_{i \in Ek} A_i] = \sum_{i \in Ek} P[A_i]$.

Ak si pravdepodobnosť každého javu rozpíšeme, dostaneme $\sum_{i \in Ek} P[A_i] = \sum_{i=1}^{k} \frac{\binom{k_i}{2}}{\binom{|Prv|}{2}}$.

nakoľko chceme nejak rozvrhnúť prvky v triedach ek_i, a menovateľ je len konštanta, môžme ho vynechať.

Maximalizujeme teda hodnotu výrazu $\sum_{i=1}^{k} \binom{k_i}{2} = \sum_{i=1}^{k} \frac{k_i!}{(k_i-2)!2!} = \sum_{i=1}^{k} \frac{k_i(k_i-1)}{2}$. Po vyškrtnutí konštanty a roznásobení sme dostali nasledujúcu optimalizačnú úlohu: maximalizovať $\sum_{i=1}^{k} k_i^2 - k_i$ za podmienok $\sum_{k=1}^{k} k_i = n$, kde $k_i \in \mathbb{N}_0$.

Sumu si vieme rozpísať ako $\sum_{i=1}^{k} k_i^2 - k_i = \sum_{i=1}^{k} k_i^2 + \sum_{i=1}^{k} -k_i$ druhá suma sa nasčíta $-n$, čo je konštanta, takže nám stačí maximalizovať $\sum_{i=1}^{k} k_i^2$.

Teraz nám už len zostáva použiť nerovnosť $(a+b)^2 \geq a^2 + b^2$ ktorá platí pre $a, b \geq 0$, z ktorej jasne vyplýva, že potrebujeme spraviť ľubovoľné k_i čo najväčšie. Ekvivalenčné triedy musia teda byť čo najväčšie a problém sa transformuje na problém hľadania obdĺžnikov s najväčším možným obsahom. V programe tento problém rieši trieda Colorizator.

3.2.2 Nájdenie najväčšej jednotkovej podmatice

Ako sme si v úvode povedali, mriežkovú mapu vieme reprezentovať ako maticu a teda problém môžeme ekvivalentne zapísať ako problém hľadania najväčšej jednotkovej podmatice. Tento problém má riešenie v čase lineárnom od počtu vrcholov a teda nájdenie k najväčších jednotkových matíc trvá $\mathrm{O}\big(k * n\big)$, kde n je počet vrcholov matice.

Slovný popis algoritmu 3: V prvom prechode maticou si u každého vrcholu zapamätáme počet jedničiek naľavo od neho, vratane daneho vrcholu. Tento prechod trvá lineárny čas.

V druhom prechode treba prejsť zaradom všetky stĺpce zľava doprava

Algoritmus 3 Nájdenie najväčšej jednotkovej podmatice v matici mxn

Vstup: matica M rozmerov mxn nad telesom $\mathbb{Z}_2$
Výstup: pravý dolný roh podmatice aj s jej rozmermi

for all prvok p, $p \in M$ **do**
  if $p = 0$ **then**
    nalavoOdPrvku(p) $\leftarrow$ 0
  else
    nalavoOdPrvku(p) $\leftarrow$ najdlhšia súvislá postupnosť jednotiek končiaca prvkom p
  end if
end for
for stĺpec s, $s \in M$ **do**
  Vytvor nový zásobník dvojíc (riadok, nalavoOdPrvku)
  for riadok r, $r \in M$ **do**
    $p \leftarrow (s, r)$
    while Zásobník je neprázdny **do**
      vyberiem prvok *top* z vrcholu zásobníka
      if *top* . nalavoOdPrvku > nalavoOdPrvku(p) **then**
        prvokZoZasobnika $\leftarrow$ (*top*.r, s)
        DlzkaSekvencie(prvokZoZasobnika) = r - prvokZoZasobnika.r
      else
        Vložím prvok $prvokZoZasobnika$ do zásobníka
        break
      end if
    end while
    Do zásobníka vložím dvojicu (r, nalavoOdPrvku(p))
  end for
  while Zásobník je neprázdny **do**
    vyberiem prvok *top* z vrcholu zásobníka
    prvokZoZasobnika $\leftarrow$ (*top*.r, s)
    DlzkaSekvencie(prvokZoZasobnika) = m - prvokZoZasobnika.r
  end while
end for

4. Porovnanie NovellA* s ostatnými algoritmami.

4.1 Vstupné dáta

Častým problémom pri vzájomnom porovnávaní algoritmov je nájsť tetsovaciu vyorku, ktor... [9]

Na porovnávanie využijeme benchmark

TODO?? bibliografia styl - priezvisko,meno - alebo naopak???

5. T-maps - Vizuálna predstava

Pre názornejšiu predstavu behu algoritmu sme navrhli softvér **T-maps**, ktorý beh algoritmov ilustruje graficky.

5.1 Použitie

Program T-maps má funčnosť veľmi podobnú programu Goole Maps. Na začiatku do neho nahrajeme mapu, nad ktorou algoritmus beží a taktiež dáta tohto algoritmu (prehľadané vrcholy a najkratšiu cestu) a program tieto hodnoty graficky znázorní. Medzi možnosti programu patrí export mapy a viditeľného poľa do súboru. TODO?? uzivatelska dokumentacia Mozno popis zaujimavej casti

Obr. 5.1: Logo MFF UK

5.1.1 Ukázka LaTeXu

V této krátké části ukážeme použití několika základních konstrukcí LaTeXu, které by se vám mohly při psaní práce hodit.

Třeba odrážky:

- Logo Matfyzu vidíme na obrázku. 5.1
- Tato subsekce má číslo 5.1.1.
- Odkaz na literaturu [**?**].

Druhy pomlček: červeno-černý (krátká), strana 16–22 (střední), 45 − 44 (minus), a toto je — jak se asi dalo čekat — vložená věta ohraničená dlouhými pomlčkami. (Všimněte si, že jsme za `a` napsali vlnovku místo mezery: to aby se tam nemohl rozdělit řádek.)

„České uvozovky.“

Definícia. *Strom je souvislý graf bez kružnic.*

Veta 3. *Tato věta neplatí.*

Dôkaz. Neplatné věty nemají důkaz. □

[**?**]

Záver

Zoznam použitej literatúry

[1] L. Euler. *Solutio problematis ad geometriam situs pertinentis.* Commentarii academiae scientiarum Petropolitanae, 8:128–140, 1741.

[2] E. W. Dijkstra. *A note on two problems in connexion with graphs.* Numerische Mathematatik, 1(1):269–271, 1959.

[3] M. Mareš. *Nejkratší cesty.* Krajinou grafových algoritmů, 2007.

[4] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *SIGART Bull.*, (37):28–29, December 1972.

[5] A. V. Goldberg. *Shortest Path Algorithms: Engineering Aspects.* In Proc. ESAAC '01, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[6] A. V. Goldberg and C. Harrelson. *Computing the Shortest Path: A* Search Meets Graph Theory.* In Proc. 16th ACM-SIAM Symposium on Discrete Algorithms, pages 156–165, 2005.

[7] J. Hartmanis, R. Stearns. *On the computational complexity of algorithms.* Transactions of the American Mathematical Society, vol. 117, 285–306, 1965.

[8] N. Sturtevant. *GPPC: Grid-Based Path Planning Competition.* Dostupné z: `http://http://movingai.com/GPPC/`

[9] N. Sturtevant. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144 – 148, 2012.

[10] A. Goldberg, C. Silverstein. *Implementations of Dijkstra's Algorithm Based on Multi-Level Buckets.* Springer Berlin Heidelberg, 1997.

Zoznam tabuliek

Zoznam použitých skratiek

Prílohy