

Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií

UMELÁ INTELIGENCIA

Akademický rok 2018/19

**Riešenie problémov, využitie procedúr hľadania**

2019

Bratislava

riešiteľ

**Daniel Minárik**

rok štúdia: **druhý**

---

# Obsah

<b>Zadanie .....</b>	<b>4</b>
<b>1 A* algoritmus .....</b>	<b>5</b>
1.1 Reprezentácia uzla.....	5
1.2 Opis riešenia .....	5
1.3 Heuristická funkcia.....	6
1.4 Porovnanie heuristík .....	6
1.5 Spôsob testovania .....	6
<b>2 Zhodnotenie .....</b>	<b>7</b>

## Zadanie

Našou úlohou je nájsť riešenie 8-hlavolamu. Hlavolam je zložený z 8 očíslovaných políčok a jedného prázdneho miesta. Políčka je možné presúvať hore, dole, vľavo alebo vpravo, ale len ak je tým smerom medzera. Je vždy daná nejaká východisková a nejaká cieľová pozícia a je potrebné nájsť postupnosť krokov, ktoré vedú z jednej pozície do druhej.

Niektoré z algoritmov potrebujú k svojej činnosti dodatočnú informáciu o riešenom probléme, presnejšie odhad vzdialenosti od cieľového stavu. Pre náš problém ich existuje niekoľko, môžeme použiť napríklad

1. Počet políčok, ktoré nie sú na svojom mieste
2. Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície
3. Kombinácia predchádzajúcich odhadov

Tieto odhady majú navyše mierne odlišné vlastnosti podľa toho, či medzi políčka počítame alebo nepočítame aj medzeru.

Použite A\* algoritmus, porovnajte výsledky heuristík 1. a 2.

# 1 A\* algoritmus

## 1.1 Reprezentácia uzla

Uzol je reprezentovaný pomocou triedy s názvom *Uzol*, ktorá ma ako parametre premenne, ktoré daný uzol reprezentujú. Týmito premennými sú:

- **Dvojrozmerné byte pole**, v ktorom sa uchováva stav daného uzla. Tento stav je konkrétna permutácia hlavolamu.
- **Inštancia triedy rodiča**. V tejto premennej je predchodcu daného uzla.
- **Hĺbka**, ktorá reprezentuje level vnárania sa do hĺbky pri prehľadávaní.
- **Posledná operácia**
- **Hodnota heuristiky**

Pre uchovávanie stavu uzla je zvolená možnosť uchovania v dvojrozmernom poli, aby bola jednoduchšia práca s týmto uzlom. Toto pole je pole byte, z dôvodu, že hodnoty pozícií nikdy nepretočia hodnotu 127 a tak môžeme znížiť priestorovú zložitosť danej reprezentácie uzla.

## 1.2 Opis riešenia

Pri riešení bol použitý algoritmus A\*. Tento algoritmus slúži na nájdenie optimálnej cesty z počiatočného stavu do koncového.

Algoritmus začína so štartovacím stavom, na ktorý sa aplikujú jednotlivé operácie. Použitím operácií sa vytvorí nový stav, ktorý sa vkladá do prioritného radu. Následne ak sa vyskúša použitie všetkých možných operácií sa z prioritného radu vyberie ten najvhodnejší. Pod pojmom najvhodnejší sa myslí uzol, ktorý ma najmenší súčet hodnoty heuristiky a hĺbky uzla. Po vybratí z prioritného radu sa uzol porovnáva, či už to nie je cieľový stav. Ak je to cieľový stav, tak sa algoritmus ukončí a vypíšu sa výsledky hľadania. Ak sa nenájde zhoda s cieľovým stavom, tak algoritmus pokračuje rovnakým spôsobom. V priebehu algoritmu môže nastať situácia, kedy je prioritný rad prázdny, vtedy sa výsledok nepodarilo nájsť, resp. nie je možné vytvorenie cesty z počiatočného do koncového stavu.

Algoritmus na svojej optimalizácii používa to, že stav ktorý sa už rozvíjal sa už ďalej rozvíjať nebude. Tento princíp je v implementácii riešený pomocou hashovania stringovej podoby stavu uzla do hashset.

### 1.3 Heuristická funkcia

Heuristická funkcia slúži na ohodnotenie stavu uzla. Vďaka tomuto hodnoteniu je možné lepšie vybrať uzol, ktorý sa bude rozvíjať. Je potrebné, aby táto funkcia vedelo čo najlepšie opísať daný stav. V tomto riešení sú použité dve rôzne heuristické funkcie.

Prvá heuristická funkcia je založená na počte políčok, ktoré nie sú na správnom mieste. Táto heuristická funkcia môže nadobúdať hodnoty od 0 po súčin výšky a šírky hlavolamu, napr.  $3 \times 3 = 9$ .

Druhá heuristická funkcia je založená na účte vzdialeností jednotlivých políčok od ich cieľovej pozície. Táto vzdialenosť sa taktiež nazýva ja manhattanovska vzdialenosť. Táto heuristická funkcia môže nadobúdať väčšiu škálu hodnôt ako prvá, čo môžeme považovať za jej výhodu.

### 1.4 Porovnanie heuristik

Na základe testovania som prišiel na zhodnotenie, že použitím druhej heuristickej funkcie sa spracováva menej uzlov ako použitím tej prvej. Tento jav môže byť spôsobený tým, že druhá heuristika môže nadobúdať väčšie množstvo hodnôt ako tá prvá. Tým môžeme povedať, že táto heuristika vie lepšie opísať daný stav.

V jednom z testov(test1), bol tento rozdiel tento rozdiel veľmi výrazný. Použitím heuristiky jedna bol počet spracovaných uzlov presne 126 936, pričom použitím heuristiky číslo dva na rovnaký hlavolam bol počet spracovaných uzlov len 7 945. V tomto príklade môžeme vidieť, že použitím lepšej heuristiky bol počet spracovaných uzlov znížený takmer 16 násobne.

V ostatných testoch bol tento rozdiel podobný, ale už nie v takej výraznej podobe, avšak stále táto heuristika znižovala počet spracovaných uzlov.

### 1.5 Spôsob testovania

Testovanie prebiehalo použitím algoritmu na rôzne počiatočné a cieľové stavy, pričom každý hlavolam bol riešený pomocou dvoch odlišných heuristik, aby bolo možné ich porovnávanie.

## 2 Zhodnotenie

Podľa môjho názoru tento algoritmus je predstaviteľom jedného z najlepších algoritmov prehľadávania. Avšak tento algoritmus je len tak dobrý ako heuristická funkcia, ktorá je v ňom použitá. V prípade, že táto heuristická funkcia nedokáže dostatočne ohodnotiť stav, tak toto prehľadávanie sa môže rovnať prehľadávaniu do šírky. Vďaka kvalitnej heuristickej funkcii je možné výrazne znížiť počet prehľadávaných uzlov.

Implementácia v programovacom jazyku Java nie je najlepšou možnosťou. Java ako vysokoúrovňový jazyk umožňuje programátorovi využívať funkcie, ktoré mu uľahčia prácu (napríklad prioritný rad, hashset), avšak na takéto výpočtové úlohy je podľa môjho názoru pomalý. Táto slabosť je môže byť spôsobená tým, že počas behu programu sa vytvára množstvo inštancií objektov, ktoré ďalej nie sú používané, preto Java využíva Garbage Collector, ktorý môže bežať aj polovicu celého behu programu. Taktiež moja implementácia je obmedzená veľkosťou plochy. Pri veľkosti  $4 \times 3$  je množstvo rôznych uzlov taký veľký, že presiahne limit Javy pre veľkosť prioritného radu 4 GB.

Dokumentácia implementácie je taktiež v podobe JavaDoc, ktorá je priložená pri zdrojovom kóde.