Slovenská Technická Univerzita v Bratislave
Fakulta informatiky a informačných technológií

Umelá inteligencia

Zadanie 2 - Prehľadávanie stavového priestoru

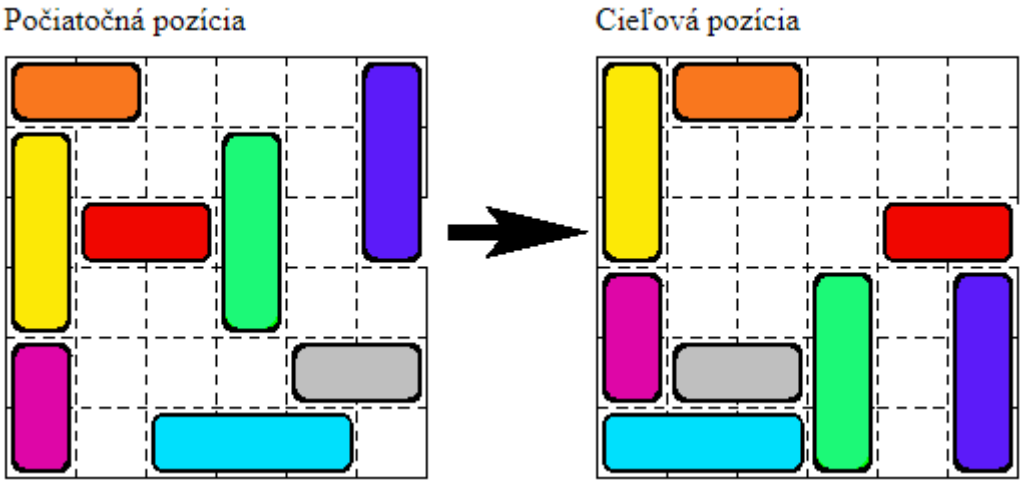
Bláznivá križovatka

Riešený problém – 1. a)

Problém sa týka riešenia hlavolamu s názvom **Bláznivá križovatka (*Unblock Me*)**, ktorý sa skladá z hracej plochy o veľkosti **6x6 políčok**. Na tejto ploche sa nachádzajú autá rôznej **farby**, **dĺžky** a **typu uloženia**, ktoré je možné reprezentovať blokmi o veľkosti **1x2** alebo **1x3**. Podľa typu uloženia sa dané auto(blok) môže pohybovať **HORE** a **DOLE** pri **vertikálnom** uložení, alebo **VĽAVO** a **VPRAVO** pri **horizontálnom** uložení, a to vždy o taký počet políčok, pri ktorom nevznikne žiadna kolízia ani pohyb mimo hracej plochy. Pre autá s **dĺžkou 2** je maximálny rozsah pohybu o **+-4 políčka** a pre autá s **dĺžkou 3** je maximálny rozsah pohybu o **+-3 políčka**.

Na začiatku je zadané štartovanie rozloženie áut(blokov), pričom cieľom je dostať **červené** auto na druhú stranu križovatky. Takže je potrebné nájsť takú postupnosť krokov, pomocou ktorých budú postupne jednotlivé autá uvoľňovať cestu **červenému**, aby sa dostalo do cieľa.

Napríklad:



Postupnosť krokov:

```
VPRAVO(oranzove, 1), HORE(zlte, 1), HORE(fialove, 1), VLAVO(sive, 3),
VLAVO(svetlomodre, 2), DOLE(tmavomodre, 3), DOLE(zelene, 2), VPRAVO(cervene, 3)
```

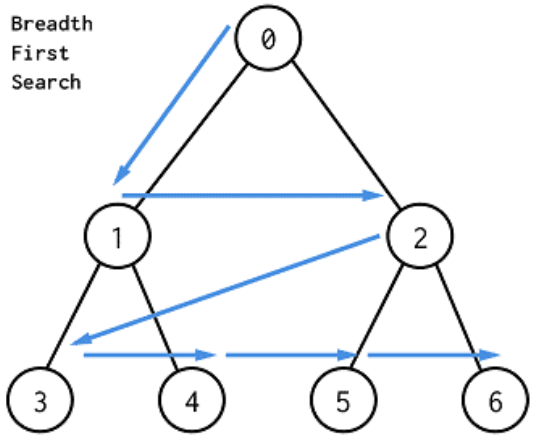
Opis riešenia

Úlohou je použiť algoritmus hľadania do šírky a algoritmus hľadania do hĺbky, a následne porovnať ich výsledky.

V oboch prípadoch sa na začiatku vytvorí štartovací uzol, v ktorom je uložený počiatočný stav (rozloženie) áut. Následne sa pomocou skúšania všetkých možností pohybu pre každé auto postupne od neho rozvíjajú jeho nasledovníci (*childNodes*), z ktorých každý reprezentuje nový stav (rozloženie áut), pričom pri každom "novovzniknutom" stave sa kontroluje vznik duplicity.

Pri oboch prípadoch hľadania je vytvorené poradie (queue), do ktorého sa novovytvorené, neduplicitné a nerozvinuté stavy postupne vkladajú, avšak pre každý algoritmus iným spôsobom.

Pri hľadaní do šírky sa každý novovytvorený stav vkladá na koniec poradia.



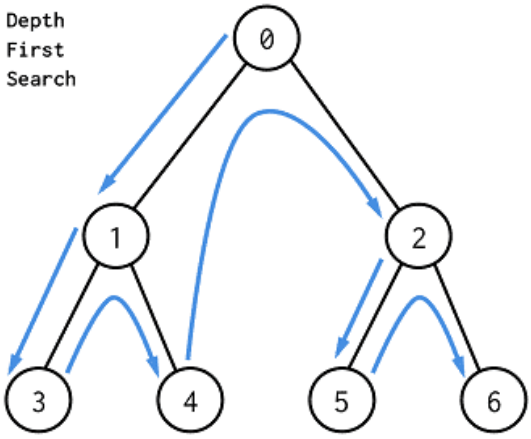
queue = (**0**)

queue = (~~0~~, **1**, 2)

queue = (~~0~~, ~~1~~, **2**, 3, 4)

queue = (~~0~~, ~~1~~, ~~2~~, **3**, 4, 5, 6)

Pri hľadaní do hĺbky sa novovytvorené stavy vkladajú na začiatok



queue =(**0**)

queue =(**1**, 2 ,~~0~~)

queue =(**3**, 4 ,~~1~~, 2, ~~0~~)

queue =(**...**, ~~3~~, 4, ~~1~~, 2, ~~0~~)

V oboch prípadoch sa ale po kompletnom rozvinutí aktuálne rozvíjaného uzla následne tento uzol z poradia(queue) odstráni. Ďalej sa potom postupne rozvíjajú uzol, ktorý sa nachádza ako prvý v poradí až do chvíle, kedy sa buď nájde riešenie hlavolamu (vyhovujúci stav), alebo sa vzniknú všetky možné stavy a v poradí sa nenachádza žiadny ďalší uzol, ktorý by sa dal rozvinúť (riešenie neexistuje).

Reprezentácia údajov a použitý algoritmus

Reprezentácia uzla – trieda **Node**

```
public class Node {
    public State state;
    public Node parentNode;
    public ArrayList<Node> childNodes;
    public int depth;
```

state – reprezentuje samotný stav

parentNode – rodičovský uzol

childNodes – pole jeho nasledovníkov

depth – aktuálna hĺbka uzla

Reprezentácia stavu – trieda **State**

```
public class State {
    public ArrayList<Car> carsArr;
    public String[][] arr;
    public Car lastMoveCar;
```

carsArr – autá s akt. pozíciami

arr – stav hracej plochy

lastMoveCar – auto, ktoré sa hýbalo naposledy

Reprezentácia áut – triedy **Car, CarH, CarV**

```java
public class Car {
    public int len;
    public int headPosX;
    public int headPosY;
    public int maxMoveLen;
    public String color;
    public String lastMove;
    public int jump;
```

```java
public class CarH extends Car {
    public int tailPosX;
    public int tailPosY;
```

```java
public class CarV extends Car {
    public int tailPosX;
    public int tailPosY;
```

Každé auto má svoju dĺžku, X-ovú a Y-ovú pozíciu, ktorá reprezentuje jeho ľavý horný roh, maximálnu možnú dĺžku pohybu, farbu, posledný vykonaný pohyb ako aj dĺžku vykonaného pohybu. Ďalej sú podľa typu auta (horizontálne/vertikálne) vypočítané aj X-ová a Y-ová pozícia jeho konca.

Pre každé auto sú špecifikované funkcie pre výpočet, kontrolu a samotný pohyb. Keďže program využíva polymorfizmus, nie je treba za chodu rozlišovať, o aký typ auta ide.

Reprezentácia hracej plochy – 2-rozmerné pole typu **String[][]**

```
START STATE:
gr  gr  bl  bl  --  lb
--  --  lr  ye  ye  lb
re  re  lr  lg  --  dr
wh  br  br  lg  --  dr
wh  --  --  lg  or  or
wh  pu  pu  pi  pi  --
```

Každé auto má jedinečnú farbu, ktorá sa postupne vyberá z poľa vopred definovaných dostupných farieb. Ako identifikácia auta sa používajú prvé 2 znaky z názvu farby.

Použitý algoritmus

Hlavnou funkciou je funkcia **findSolution()**, ktorá reprezentuje DFS aj BFS podľa zadaného argumentu. Keďže celý princíp je založený na postupnom rozvíjaní, vkladaní a odstraňovaní uzlov z poradia (queue) na príslušné pozície s požadovaným spôsobom.

Pri BFS sa jednotlivé uzly spracúvajú postupne v tej istej hĺbke stromu, preto vznikne oveľa viac stavov ako pri DFS. Za to je ale finálne riešenie hlavolamu jednoduchšie (menši počet krokov).

Pri DFS sa postupne rozvíjajú a spracúvajú uzly v stále väčšej hĺbke pokiaľ to ide. Následne sa postupne vynára tým, že sa za dostanú na rad uzly s menšou hĺbkou (predchodcovia). Môže vzniknúť síce menej stavov ako pri BFS ale za to finálne riešenie hlavolamu bude oveľa zdĺhavejšie (väčší počet korkov).

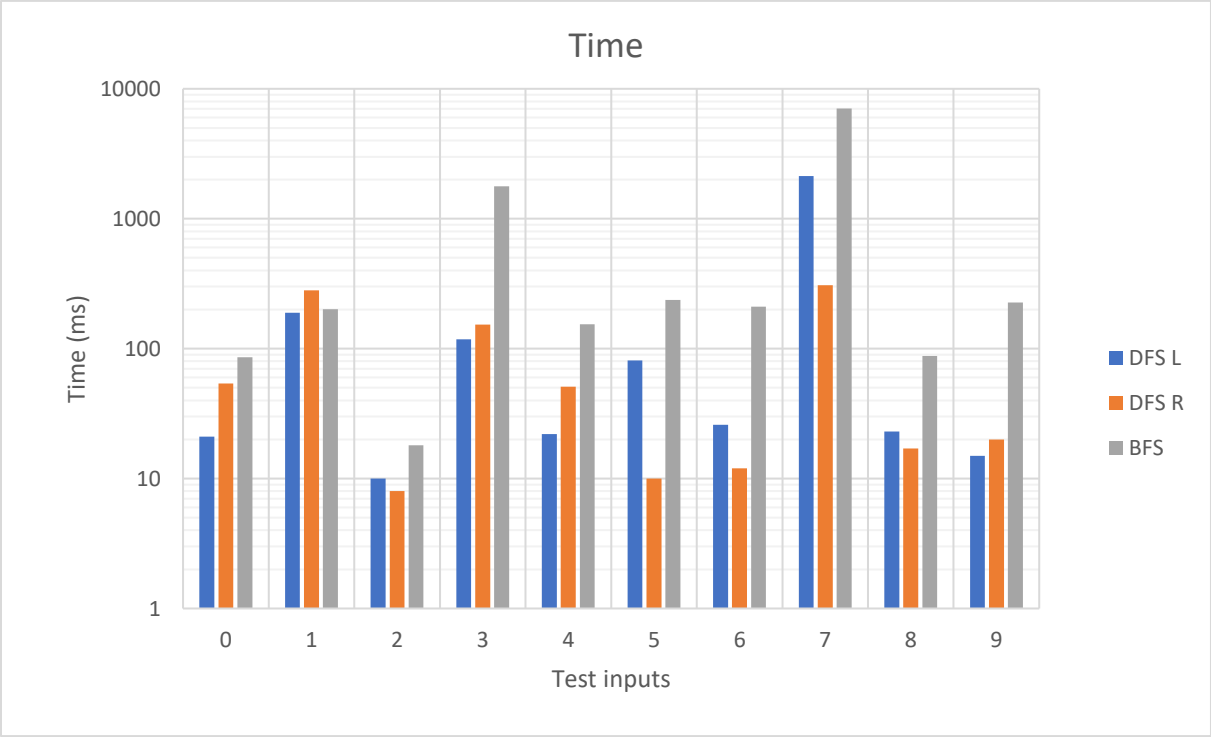
Princíp algoritmu

1. Vytvor počiatočný uzol a umiestni ho medzi vytvorené a zatiaľ nespracované uzly do (queue)
2. Ak neexistuje žiadny vytvorený a zatiaľ nespracovaný uzol – (prázdna queue), skonči s neúspechom – riešenie neexistuje
3. Vyber nasledujúci uzol z vytvorených a zatiaľ nespracovaných (queue), označ ho ako aktuálny
4. Postupne vytvor všetkých nasledovníkov aktuálneho uzla a pri každom skontroluj možnosť duplicity – ak existuje, ignoruj ho
5. Ak nejaký z týchto uzlov predstavuje cieľový stav, skonči s úspechom – vypíš riešenie
6. Ďalej všetky neduplicitné novovytvorené uzly zaraď medzi vytvorené a zatiaľ nespracované uzly do (queue)
7. Aktuálny uzol zaraď medzi spracované uzly – odstráň ho z poradia (queue)
8. Choď na krok 2.

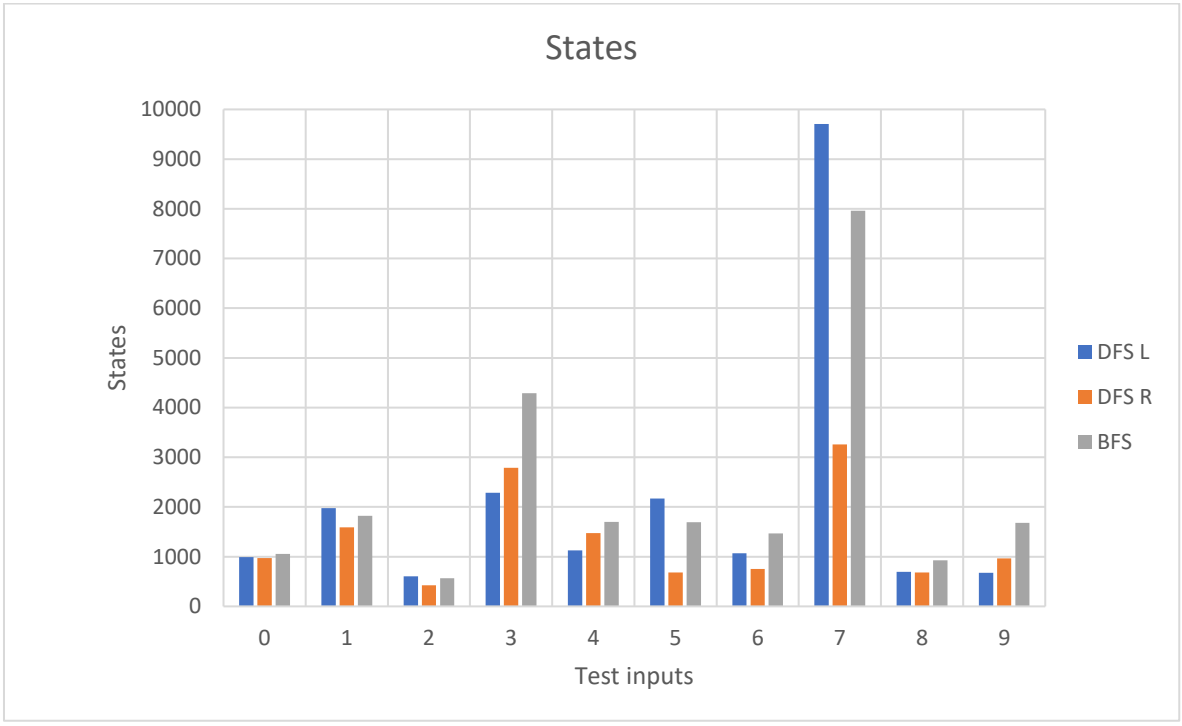
Testovanie

Na testovanie som použil 10 korektných vstupov hlavolamu s existujúcim riešením, ktoré sú priložené v súbore.

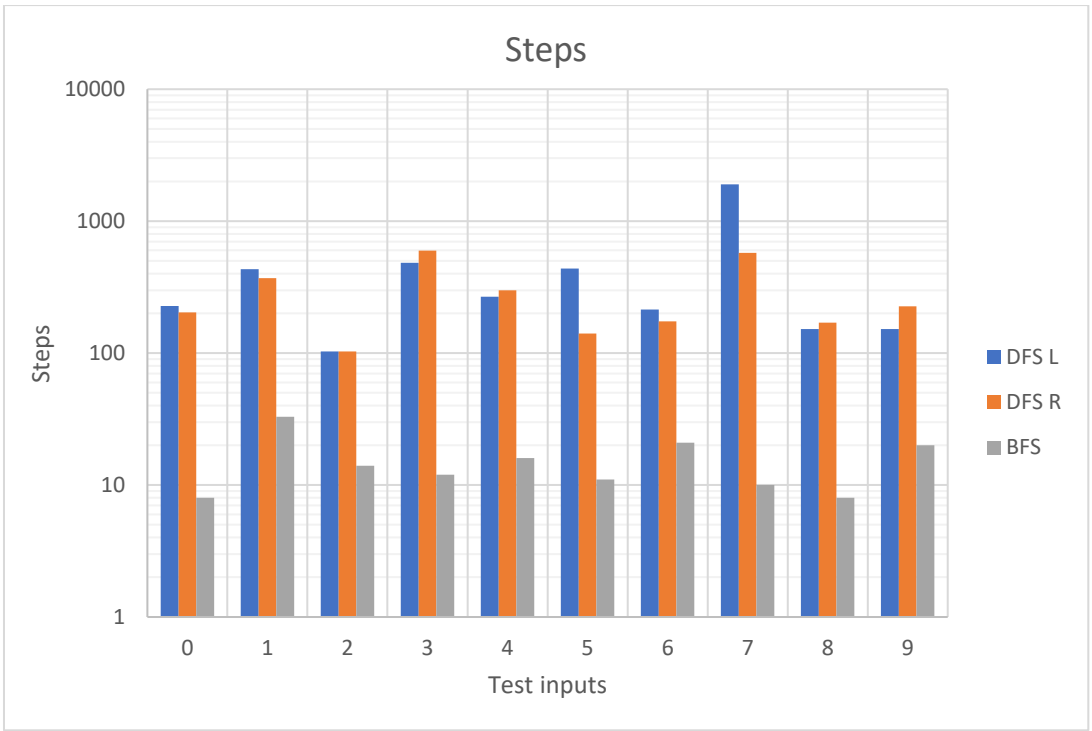
Porovnanie časovej náročnosti



Porovnanie pamäťovej náročnosti – (počet vytvorených stavov)



Porovnanie riešení – (Počet požitých operátorov na vyriešenie hlavolamu)



Rozšíriteľnosť programu

Program by sa pár úpravami v kóde dal rozšíriť aj pre väčší rozmer hlavolamu, väčší počet áut poprípade väčší rozmer áut. Taktiež by sa dal rozšíriť na možnosť zvolenia auta, pre ktoré sa hľadá cieľový stav, ktorí by taktiež bolo možné špecifikovať.

Všeobecné informácie

Program:

- Program bol vyhotovený v jazyku Java (SDK 16)
- Použité bolo IDE IntelliJ IDEA Ultimate 2021.2

Vstupy:

- Nachádzajú sa v priečinku inputs
- Načítavajú sa zo .txt súboru, ktorého názov je možné zadať po spustení programu
- Vstup musí byť v .txt súbore zadaný správne (formát, platné rozloženie, bez kolízií)
- Prvé auto (prvý riadok) reprezentuje **červené auto,** ktoré sa má dostať do cieľa
- Je možné špecifikovať vstup pre max 15 áut
- Formát vstupu: <X-ováPozícia> <Y-ováPozícia> <DĺžkaAuta> <Uloženie>
  0-5 0-5 2/3 V/H

Príklad vstupu: