Fakulta informatiky a informačných technológií

Slovenská technická univerzita

Umelá inteligencia - zadanie č. 2

# Prehľadávanie stavového priestoru :

# Riešenie bláznivej križovatky pomocou BFS a DFS

Michaela Gubovská

Streda 13:00

Ing. Ivan Kapustík

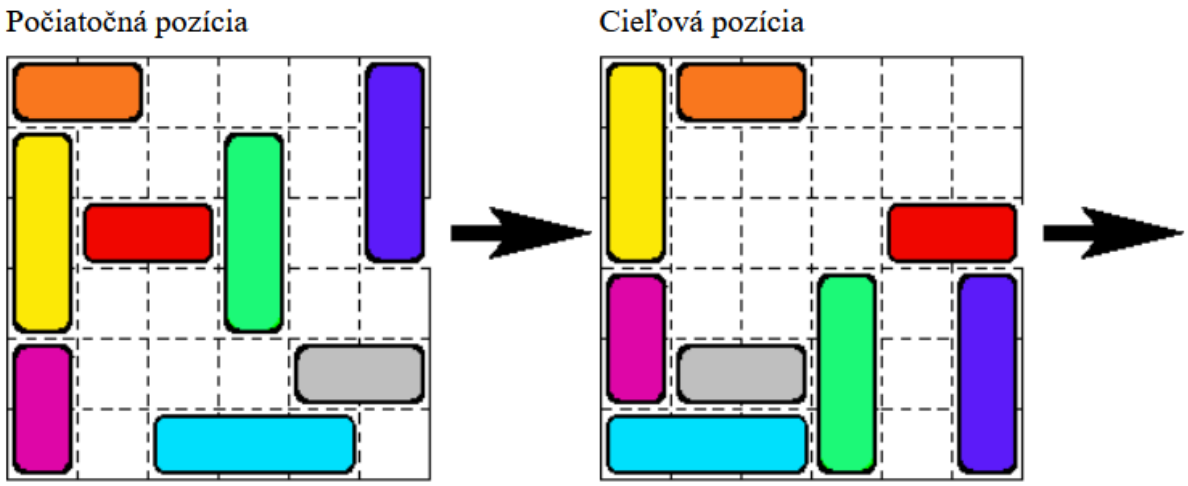
Akad. Rok 2020/21

Michaela Gubovská
ID: 102925
akad.r. 2020/21

## 1. Zadanie

Úlohou je nájsť riešenie hlavolamu **Bláznivá križovatka**. Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6 krát 6 políčok a obsahuje niekoľko vozidiel (áut a nákladiakov) rozložených na mriežke tak, aby sa neprekrývali. Všetky vozidlá majú šírku 1 políčko, autá sú dlhé 2 a nákladiaky sú dlhé 3 políčka. V prípade, že vozidlo nie je blokované iným vozidlom alebo okrajom mriežky, môže sa posúvať dopredu alebo dozadu, nie však do strany, ani sa nemôže otáčať. V jednom kroku sa môže pohybovať len jedno vozidlo. V prípade, že je pred (za) vozidlom voľných *n* políčok, môže sa vozidlo pohnúť o 1 až *n* políčok dopredu (dozadu). Ak sú napríklad pred vozidlom voľné 3 políčka (napr. oranžové vozidlo na počiatočnej pozícii, obr. 1), to sa môže posunúť buď o 1, 2, alebo 3 políčka.

Hlavolam je vyriešený, keď je červené auto (v smere jeho jazdy) na okraji križovatky a môže sa z nej dostať von. Predpokladajte, že červené auto je vždy otočené horizontálne a smeruje doprava. Je potrebné nájsť postupnosť posunov vozidiel (nie pre všetky počiatočné pozície táto postupnosť existuje) tak, aby sa červené auto dostalo von z križovatky alebo vypísať, že úloha nemá riešenie. Príklad možnej počiatočnej a cieľovej pozície je na obr. 1.




Obr. 1 Počiatočná a cieľová pozícia hlavolamu Bláznivá križovatka.

**Problém 1 a) : Použite algoritmus prehľadávania do šírky a do hĺbky. Porovnajte ich výsledky.**

Michaela Gubovská
ID: 102925
akad.r. 2020/21

## 2. Opis riešenia a reprezentácia údajov

Pre riešenie bláznivej križovatky je potrebné postupne pre každé auto prehľadať všetky stavy, teda pohyby jednotlivých áut. Je k tomu potrebné neustále vedieť, v akom stave sa aktuálne nachádza priestor. Pri riešení preto využívam nasledovné konštrukcie:

**Autá** predstavujú objekty, ktorých parametre sú farba, x-ová, y-ová súradnica, veľkosť a smer.

```
class Car:      #kazde auto je jeden objekt s vlastnou
    def __init__(self, color, size, x, y, direction):
        self.color = color
        self.size = size                #"2" alebo "3"
        self.x = x
        self.y = y
        self.direction = direction      #"h" alebo "v"
```

**Uzly** sú takisto objekty, ktoré obsahujú stav pomocou poľa objektov áut, rodiča uzla, operátor a mapu aktuálne priestoru.

```
class Node:     #kazdy uzol je objekt s polom aut, ro
    def __init__(self, cars, parent, operator, mapa):
        self.cars = cars                #pole aut s i
        self.parent = parent            #uzol, z ktor
        self.operator = operator        #o aky posun
        self.mapa = mapa                #mapa aktualn

    def code_name_of_node(self, mapa):  #zakoduje map
        map_string = ""
        for i in range(rows):
            for j in range(columns):
                map_string += mapa[i][j]
        return map_string
```

**Mapa priestoru** je dvojrozmerné 6x6 pole charov, v ktorom je na políčkach zabratých autami prvé písmeno farby daného auta a na zvyšných políčkach je „-„, symbolizujúca prázdne políčko.

**Operátor** sa skladá z názvu pohybu, čiže LEFT/RIGHT/UP/DOWN, farby auta a počtu krokov. Operátor môže vyzerať napríklad takto: **LEFT(orange, 1)**

**Rodič** je objekt triedy *uzol* (teda má všetky parametre uzla) ktorý reprezentuje uzol, z ktorého bol nový uzol vytvorený. Teda pomocou rodičov vieme na konci reverzne zrekonštruovať postupnosť pohybov, ktorou sme dostali červené auto do cieľa.

Michaela Gubovská
ID: 102925
akad.r. 2020/21

Na začiatku si program vloží do poľa cars[] autá, ktoré prečíta zo súboru zvoleného užívateľom. Zo vstupných áut potom vytvorí počiatočnú mapu priestoru, na ktorej sú zaznačené jednotlivé polohy áut. Vytvorí sa počiatočný *root* uzol, ktorého parametrami sú:

- pole áut s ich počiatočnými súradnicami
- rodič je definovaný ako None, nakoľko sa jedná o počiatočný stav
- operátor je takisto None, nakoľko nedošlo k žiadnemu pohybu
- mapa s počiatočnými polohami áut

Tento uzol následne vojde do funkcie na prehľadávanie stavového priestoru a je pridaný do **zásobníka**. Zo zásobníka je následne vybraný a začne sa preňho príslušné prehľadávanie okolia.

## 2.1 Prehľadávanie do hĺbky

DFS spočíva v prehľadávaní priestoru postupne do hĺbky, čo znamená, že pre každý uzol algoritmus postupuje stále nižšie a prehľadáva deti nejakého uzla. Algoritmus začína v **root** uzle a nájde všetky jeho deti, ktoré popridáva na koniec zásobníka. Pod deťmi rozumieme všetky možné uzly s pohybmi áut, ktoré sú „hýbateľné“ pre mapu prehľadávaného uzla (na začiatku uzla root). Do zásobníka sa však pridávajú na ďalšie prehľadanie iba také uzly, ktoré sme predtým ešte nenavštívili, aby sme sa tak vyhli cykleniu. Následne po nájdení všetkých detí vyberieme zo zásobníka uzol na poslednom mieste, čím zabezpečíme prehľadanie do hĺbky, nakoľko na posledných miestach zásobníka budú vždy deti predošlého uzla (ak existujú) alebo ďalší neprehľadaný uzol na danej úrovni. Program takto pokračuje v prehľadávaní detí až pokým buď a) nevyprázdni zásobník, čo by znamenalo, že sa červené auto nedokáže dostať do cieľa, alebo b) nájde uzol, v ktorom sa červené auto nachádza v cieli.

## 2.2 Prehľadávanie do šírky

Podobne ako pri DFS, aj v BFS sa prehľadávajú deti uzlov, avšak po úrovniach, čiže sa neponárame do jedného dieťaťa a z neho rovno hlbšie do jeho ďalších detí. Algoritmus začína v počiatočnom uzle **root** a preňho nájde všetky možné pohyby každého auta. Tieto nové uzly pridáva do zásobníka na jeho začiatok. Ďalší uzol na prehľadanie vytiahne z konca zásobníka. Takto sa zabezpečí postupné prehľadanie priestoru po úrovniach, čiže do šírky. Rovnako ako pri DFS, aj tu je ošetrené cyklenie stavov a to spôsobom, že do stacku nebude pridaný na ďalšie prehľadanie uzol, pre ktorý sme už predtým priestor prehľadali, teda sme ho navštívili. Program rovnako končí buď nájdením uzla v ktorom je červené auto v cieli, alebo úplným vyprázdnením zásobníka, kedy nebolo nájdené riešenie.

V oboch algoritmoch je pri nájdení uzla s červeným autom v cieli vypísaná postupnosť posunov áut, ktorou sa program dostal k riešeniu. Funkcia na výpis tejto cesty zoberie finálny uzol s červeným autom v cieli a pomocou parametra **parent** zistí, ktorým uzlom sme sa do tohto výsledného stavu dostali. Takto pomocou rodičov prechádza cez všetky uzly výslednej cesty a vkladá ich operátory do poľa, ktoré po vojdení do **root** uzlu program vypíše odzadu, zobraziac tak výslednú postupnosť operátorov.

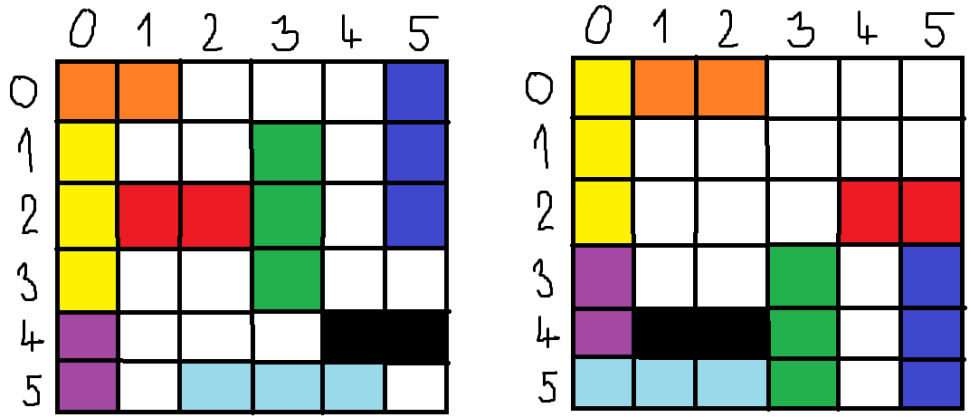
Michaela Gubovská
ID: 102925
akad.r. 2020/21

## 3. Spôsob testovania

Svoje riešenie som testovala na mapách s rozmerom 6x6 s tým, že som vytvorila rôzne scenáre s rôznym pôvodným rozložením áut. Niektoré scenáre boli jednoduchšie, iné zložitejšie a testovala som aj také scenáre, pri ktorých neexistovalo riešenie. Program po skončení vždy vypísal očakávané cesty a v prípadoch, že neexistovalo riešenie, správne to zhodnotil a vypísal hlášku „Neexistuje riesenie“.

Jednotlivé scenáre sa skladajú z dvoch častí – ako prvé opisujem vždy BFS s jeho výsledkom a ako druhé DFS so svojím výsledkom. Scenáre sú očíslované rovnako, ako aj v programe textové súbory s autami.

**Scenár 1 – príklad zo zadania**



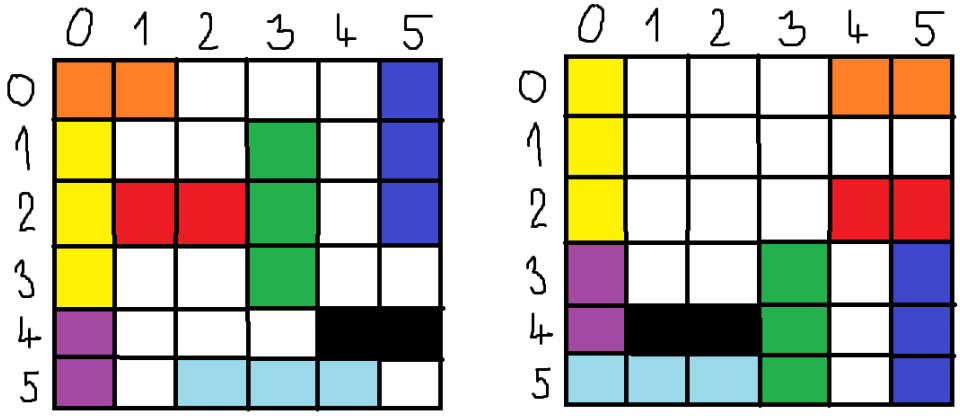

```
bfs:
Red car in final destination, sequence of moves:
RIGHT(orange, 1)
UP(yellow, 1)
UP(purple, 1)
LEFT(lightblue, 1)
LEFT(lightblue, 1)
LEFT(black, 1)
LEFT(black, 1)
LEFT(black, 1)
DOWN(green, 1)
DOWN(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
RIGHT(red, 1)
```

Program pomocou **BFS** našiel ideálne riešenie v **16 krokoch**.

Toto riešenie sa nezhoduje na 100% s tým v zadaní kvôli tomu, že som v algoritmoch posúvala autami iba o jeden krok, preto nebolo možné spraviť v jednom kroku hneď premiestnenie niektorého auta o viac ako jedno políčko, takže potom ani v ďalšom kroku nebolo možné iným autom pohnúť o toľko, ako by bolo možné keby sa auto pred ním pohne o viac krokov. Napríklad to vidíme na posledných 6 riadkoch od konca postupnosti.

**Runtime: 0.816s**

Michaela Gubovská
ID: 102925
akad.r. 2020/21




```
dfs:
Red car in final destination, sequence of moves:
DOWN(darkblue, 1)
LEFT(black, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
LEFT(black, 1)
LEFT(black, 1)
UP(darkblue, 1)
UP(darkblue, 1)
UP(darkblue, 1)
RIGHT(black, 1)
RIGHT(lightblue, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
UP(green, 1)
RIGHT(black, 1)
UP(darkblue, 1)
UP(darkblue, 1)
RIGHT(black, 1)
RIGHT(orange, 1)
DOWN(darkblue, 1)
LEFT(lightblue, 1)
LEFT(black, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
LEFT(black, 1)
LEFT(black, 1)
UP(darkblue, 1)
UP(darkblue, 1)
UP(darkblue, 1)
RIGHT(black, 1)
RIGHT(lightblue, 1)
```

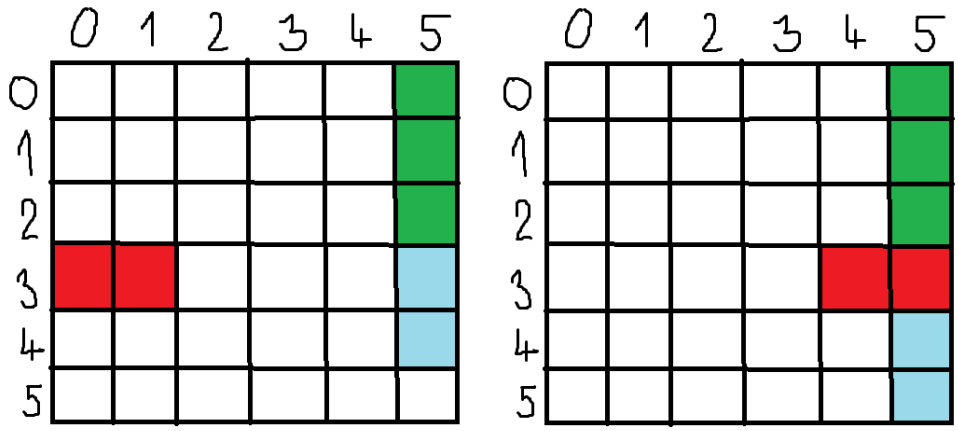
Program pomocou **DFS** tak isto našiel riešenie pre zadanú križovatku, avšak môžeme si všimnúť, že je menej výhodné ako to, ktoré našlo BFS. Celkovo sa skladá až zo **174 krokov**. Táto skutočnosť vyplýva práve z toho, že DFS nájde prvé možné riešenie, ktoré sa môže nachádzať vo väčšej hĺbke ako je to ideálne, na ktoré potom už nenarazí. Algoritmus ide „bezhlavo“ do hĺbky až kým nenájde riešenie a preto riešenia nájdené pomocou DFS nebývajú ideálne.

Na obrázku nie sú všetky kroky, nakoľko by sa do záberu nezmestili.

**Runtime: 0.4s**

Michaela Gubovská
ID: 102925
akad.r. 2020/21

**Scenár 2**

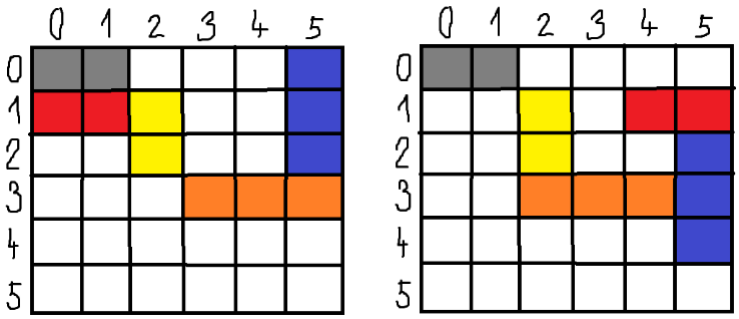



```
bfs:
Red car in final destination, sequence of moves:
RIGHT(red, 1)
RIGHT(red, 1)
RIGHT(red, 1)
DOWN(lightblue, 1)
RIGHT(red, 1)
5

Time taken by BFS:  0.002958536148071289

dfs:
Red car in final destination, sequence of moves:
DOWN(lightblue, 1)
DOWN(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
UP(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
7

Time taken by DFS:  0.0020329952239990234
```

V tomto príklade sa jedná o naozaj jednoduchú križovatku a slúži hlavne na odpozorovanie správnosti riešenia. Znova vidíme, že aj na takto jednoduchej mape je riešenie, vytvorené pomocou DFS zbytočne zložitejšie, aj keď je o niečo málo rýchlejšie.

Michaela Gubovská
ID: 102925
akad.r. 2020/21

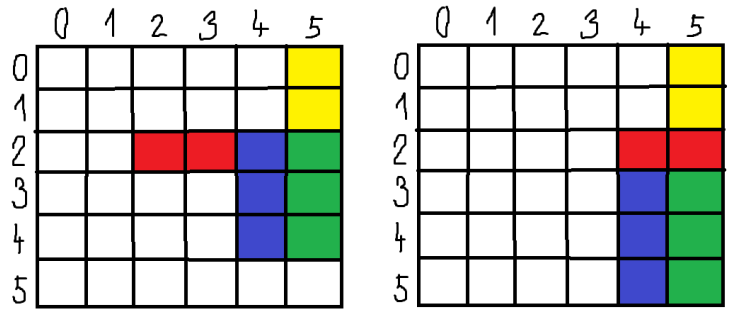
**Scenár 3**



```
bfs:
Red car in final destination, sequence of moves:
DOWN(yellow, 1)
RIGHT(red, 1)
RIGHT(red, 1)
RIGHT(red, 1)
UP(yellow, 1)
LEFT(orange, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
RIGHT(red, 1)
9

Time taken by BFS:  0.09873032569885254
```

```
UP(darkblue, 1)
UP(darkblue, 1)
UP(darkblue, 1)
RIGHT(orange, 1)
UP(yellow, 1)
UP(yellow, 1)
RIGHT(red, 1)
UP(yellow, 1)
UP(yellow, 1)
LEFT(orange, 1)
DOWN(darkblue, 1)
DOWN(darkblue, 1)
RIGHT(red, 1)
69

Time taken by DFS:  0.08038735389709473
```

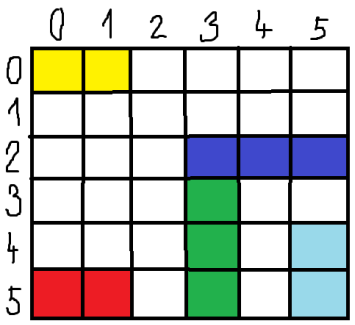
**Scenár 4**

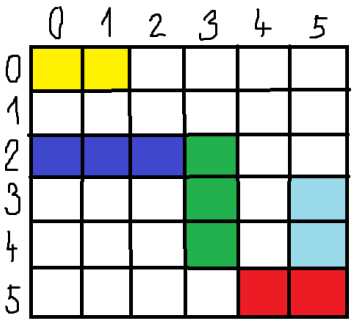


```
bfs:
Red car in final destination, sequence of moves:
DOWN(green, 1)
DOWN(darkblue, 1)
RIGHT(red, 1)
RIGHT(red, 1)
4

Time taken by BFS:  0.009972333908081055

dfs:
Red car in final destination, sequence of moves:
DOWN(darkblue, 1)
DOWN(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
4

Time taken by DFS:  0.012995243072509766
```

Michaela Gubovská
ID: 102925
akad.r. 2020/21

**Scenár 5**

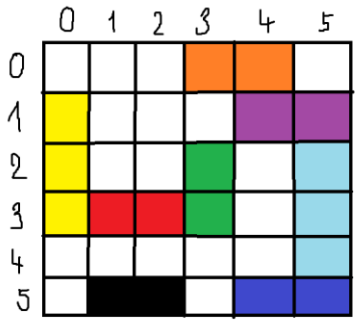


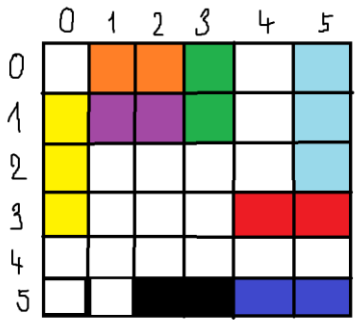




```
bfs:
Red car in final destination, sequence of moves:
RIGHT(red, 1)
UP(lightblue, 1)
LEFT(darkblue, 1)
LEFT(darkblue, 1)
LEFT(darkblue, 1)
UP(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
RIGHT(red, 1)
9

Time taken by BFS:  0.1485588550567627
```

```
RIGHT(yellow, 1)
RIGHT(yellow, 1)
RIGHT(yellow, 1)
RIGHT(yellow, 1)
DOWN(lightblue, 1)
UP(green, 1)
RIGHT(red, 1)
UP(lightblue, 1)
RIGHT(red, 1)
91

Time taken by DFS:  0.06680631637573242
```

**Scenár 6**



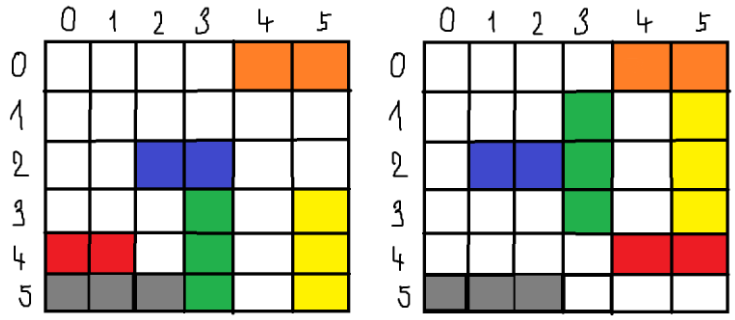




```
bfs:
Red car in final destination, sequence of moves:
LEFT(orange, 1)
LEFT(orange, 1)
LEFT(purple, 1)
LEFT(purple, 1)
LEFT(purple, 1)
UP(green, 1)
UP(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
UP(lightblue, 1)
UP(lightblue, 1)
RIGHT(red, 1)
12
```

```
RIGHT(darkblue, 1)
RIGHT(black, 1)
DOWN(yellow, 1)
UP(lightblue, 1)
RIGHT(darkblue, 1)
RIGHT(black, 1)
UP(lightblue, 1)
UP(lightblue, 1)
LEFT(black, 1)
LEFT(darkblue, 1)
LEFT(purple, 1)
RIGHT(red, 1)
RIGHT(red, 1)
671

Time taken by DFS:  1.064706563949585
```

Michaela Gubovská
ID: 102925
akad.r. 2020/21

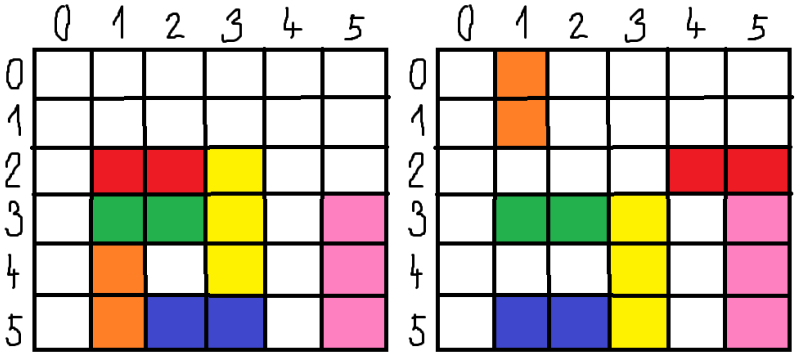
**Scenár 7**



```
UP(yellow, 1)
LEFT(darkblue, 1)
RIGHT(black, 1)
RIGHT(black, 1)
RIGHT(darkblue, 1)
UP(green, 1)
UP(yellow, 1)
RIGHT(red, 1)
146

Time taken by DFS:  0.1371448040008545
```

```
bfs:
Red car in final destination, sequence of moves:
RIGHT(red, 1)
UP(yellow, 1)
UP(yellow, 1)
LEFT(darkblue, 1)
UP(green, 1)
UP(green, 1)
RIGHT(red, 1)
RIGHT(red, 1)
RIGHT(red, 1)
9

Time taken by BFS:  0.4482755661010742
```

**Scenár 8**

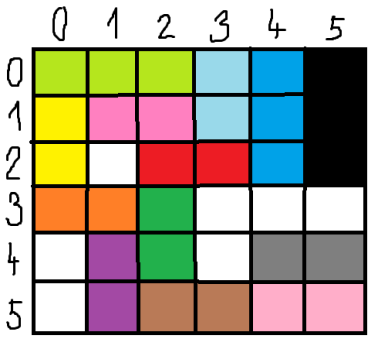


```
LEFT(green, 1)
DOWN(yellow, 1)
DOWN(yellow, 1)
DOWN(yellow, 1)
RIGHT(red, 1)
RIGHT(red, 1)
DOWN(pink, 1)
DOWN(pink, 1)
DOWN(pink, 1)
RIGHT(red, 1)
33

Time taken by BFS:  0.5186178684234619
```

```
DOWN(pink, 1)
RIGHT(blue, 1)
DOWN(pink, 1)
UP(orange, 1)
LEFT(blue, 1)
RIGHT(green, 1)
DOWN(orange, 1)
RIGHT(red, 1)
RIGHT(red, 1)
175

Time taken by DFS:  0.1930224895477295
```

Michaela Gubovská
ID: 102925
akad.r. 2020/21

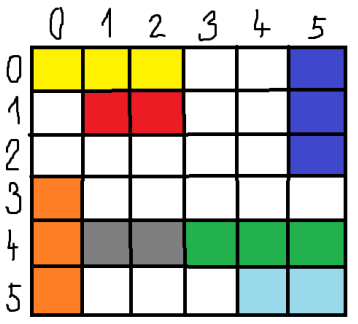
**Scenár 9 – najťažšia známa verzia križovatky**

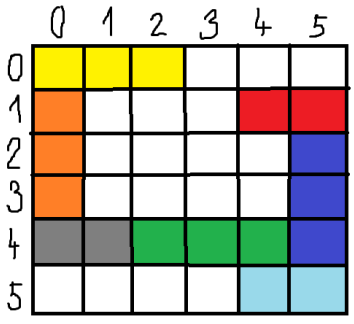


```
LEFT(Gray, 1)
DOWN(emerald, 1)
RIGHT(red, 1)
RIGHT(red, 1)
LEFT(salmon, 1)
DOWN(black, 1)
LEFT(salmon, 1)
DOWN(darkblue, 1)
RIGHT(red, 1)
RIGHT(red, 1)
93

Time taken by BFS:  61.163612365722656
```

```
DOWN(emerald, 1)
RIGHT(Gray, 1)
DOWN(yellow, 1)
UP(Purple, 1)
RIGHT(pink, 1)
UP(emerald, 1)
DOWN(black, 1)
RIGHT(red, 1)
2978

Time taken by DFS:  7.487545967102051
```

**Scenár 10**





```
bfs:
Red car in final destination, sequence of moves:
RIGHT(red, 1)
RIGHT(red, 1)
DOWN(darkblue, 1)
UP(orange, 1)
UP(orange, 1)
LEFT(black, 1)
LEFT(green, 1)
DOWN(darkblue, 1)
RIGHT(red, 1)
8

Time taken by BFS:  0.33011770248413086
```

```
RIGHT(yellow, 1)
UP(orange, 1)
RIGHT(lightblue, 1)
DOWN(darkblue, 1)
DOWN(orange, 1)
RIGHT(red, 1)
RIGHT(red, 1)
83

Time taken by DFS:  0.07104277610778809
```

Michaela Gubovská
ID: 102925
akad.r. 2020/21

## 4. Zhodnotenie riešenia

Na testovacej vzorke som dokázala, že moje riešenie je správne. Niektoré mapy som si vytvorila sama a skontrolovala vlastným krokovaním správnosť riešení. Iné mapy som zasa čerpala zo zdrojov na internete, ktoré sa venujú práve bláznivej križovatke a skontrolovala som si počet krokov riešenia môjho programu s počtom krokov riešenia uvedeného na týchto stránkach.

Z testovania som dospela k záveru, že pre optimálne riešenie je vhodnejšie používať BFS algoritmus, pretože naozaj nájde to najlepšie riešenie. Hlavne pre zložitejšie križovatky je riešenie poskytnuté BFS kratšie. **Výhodou BFS je teda nájdenie optimálneho, najlepšieho (najkratšieho) riešenia.**

Pre rýchlejšie nájdenie riešenia je však vhodnejšie použiť DFS, avšak za cenu častokrát veľmi neoptimálneho riešenia. Postupnosť totiž oproti BFS môže obsahovať rádovo až o stovky krokov viac ako riešenie, ktoré nájde BFS pre tú istú križovatku. Trvá však kratšie, pretože ide rad za radom do hĺbky, až kým nenájde riešenie, na rozdiel od BFS, ktoré prehľadáva priestor do šírky a preto mu trvá dlhšie, kým nájde prvé správne riešenie. **Výhodou DFS je preto menšia časová náročnosť.**

Tieto riešenia sú závislé od programovacieho prostredia hlavne čo sa týka práce s pamäťou. Ja som svoje riešenie implementovala v jazyku Python, ktorý nemá až takú výhodnú pamäťovú réžiu ako napríklad jazyk C. Preto by možno bolo toto riešenie rýchlejšie v jazyku C, hlavne čo sa týka zložitejších križovatiek.

## 5. Porovnanie vlastností pre rôzne riešenia

Z pozorovaní vyplynulo, že časová efektivita oboch algoritmov sa zhoršuje, čím zložitejšia je križovatka. Pri zložitejších križovatkách s viacerými autami totiž oba algoritmy musia prejsť viac stavov. Zložitosť križovatky však neovplyvňuje kvalitu riešenia, ktoré algoritmus vytvorí pri BFS. Pri DFS už býva riešenie pri zložitých križovatkách veľmi obsiahle a znižuje sa tak jeho kvalita, čo sa počtu krokov týka.

## 6. Implementačné prostredie

Program je implementovaný v jazyku Python, verzia 3.7 na platforme Windows 10 Pro. Pracovala som v prostredí Pycharm Professional 2020.1.3. Využila som knižnice collections (zásobník pomocou dátovej štruktúry **deque**, pre ľahkú obojstrannú prácu s ním), copy pre využitie funkcionality deepcopy a time pre časovanie runtime-u oboch algoritmov.