

Zadanie č. 1 – Prehľadávanie stavového priestoru

Obsah

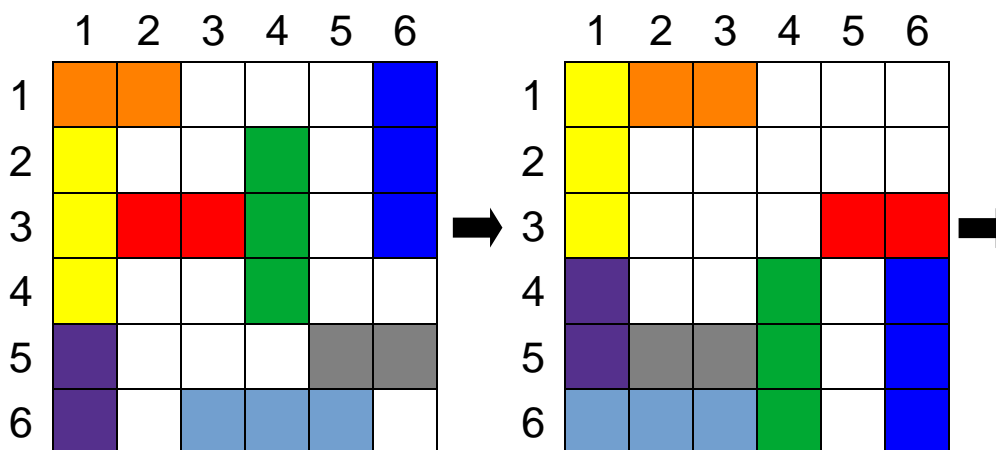
Problém – Bláznivá križovatka	2
Stav	2
Operátory	3
Uzol	3
Algoritmus	3
Prehľadávanie do šírky (BFS)	4
Prehľadávanie do hĺbky (DFS)	5
Moje riešenie	6
Testovanie	7
Príklad 1	8
Príklad 2	10
Príklad 3	12
Príklad 4	13
Príklad 5	14
Zhodnotenie	16

Problém – Bláznivá križovatka

Moje zadanie: a) Použite algoritmus prehľadávania do šírky a do hĺbky. Porovnajme ich výsledky.

Mojou úlohou v zadaní číslo 1 bolo nájsť riešenie hlavolamu *Bláznivá križovatka*. Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6x6 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 alebo za vozidlom voľných n políčok, môže sa vozidlo pohnúť o 1 až n políčok dopredu, prípadne dozadu. Ak sú napríklad pred vozidlom voľné 3 políčka, to sa môže posunúť buď o 1, 2 alebo 3 políčka.

Hlavolam je vyriešený, keď je červené auto na okraji križovatky a môže sa z nej dostať von. Predpokladáme, že červené auto je vždy otočené horizontálne a smeruje doprava. Úlohou je nájsť postupnosť posunov vozidiel, aby sa červené auto dostalo von z križovatky. Nie pre všetky počiatočné pozície postupnosť posunov existuje, v tomto prípade je potrebné vypísať, že úloha nemá riešenie. Príklad možnej počiatočnej a cieľovej pozície je na obrázku nižšie.



Stav

Stav predstavuje aktuálne rozloženie vozidiel. Potrebujeme si pamätať farbu každého vozidla, jeho veľkosť, pozíciu vozidla a či sa môže posúvať vertikálne alebo horizontálne. Počiatočný stav môžeme zapísať napríklad

((cervene 2 3 2 h)(oranzove 2 1 1 h)(zlte 3 2 1 v)(fialove 2 5 1 v)(zelene 3 2 4 v)(svetlomodre 3 6 3 h)(sive 2 5 5 h)(tmavomodre 3 1 6 v))

V tomto zápise je prvé vozidlo červené auto, ktoré sa má dostať von z križovatky. Veľkosť vozidla je vždy 2 alebo 3. Súradnice zodpovedajú ľavému hornému rohu automobilu a súradnice sú počítané od ľavého horného rohu križovatky, začínajúc od jednotky, prvá súradnica určuje riadok, druhá určuje stĺpec. Smer pohybu automobilu určuje **h** (horizontálny), t.j. pohyb vľavo a vpravo a **v** (vertikálny), t.j. pohyb hore a dole.

Vstupom algoritmov je začiatočný stav. Cieľový stav je definovaný tak, že červené auto je na najpravejšej pozícii v riadku.

Operátory

Operátory sú štyri:

- *VPRAVO*(vozidlo, počet)
- *VLAVO*(vozidlo, počet)
- *DOLE*(vozidlo, počet)
- *HORE*(vozidlo, počet)

Operátor dostane poradie vozidla a počet políčok, o ktoré sa má vozidlo posunúť. Operátor posunie vozidlo, ktoré dostane na vstupe o zadaný počet políčok. Všetky operátory pre tento problém majú rovnakú váhu.

Uzol

Uzol (stav) predstavuje nejaký bod v stavovom priestore.

Uzol v mojej implementácii obsahuje:

- **vehicles** – vozidlá a ich pozície, ktoré reprezentujú daný stav
- **predecessor** – odkaz na predchodcu, aby sme vedeli vypísať cestu
- **movement** – správa o tom, ako sme sa do tohto stavu dostali od jeho predchodcu

```
class State:
    def __init__(self, vehicles, predecessor, movement):
        self.vehicles = vehicles
        self.predecessor = predecessor
        self.movement = movement

    #reprezentácia stavu pri výpise
    def __str__(self):
        result = ""
        for v in self.vehicles:
            result += f"{v.color}({v.row}, {v.col}) "
        return result
```

Algoritmus

V mojom riešení využívam dva rôzne algoritmy na nájdenie riešenia hlavolamu *Bláznivá križovatka*:

1. **prehľadávanie do šírky (BFS)**
2. **prehľadávanie do hĺbky (DFS)**

Tieto algoritmy bližšie vysvetlím v nasledujúcich kapitolách.

Prehľadávanie do šírky (BFS)

Tento algoritmus je často používaný na nájdenie najkratšej cesty medzi dvoma bodmi v grafe alebo na prehľadávanie do všetkých možných bodov v grafe od zadaného počiatočného bodu. Jeho princíp spočíva v postupnom prehľadávaní všetkých dostupných uzlov grafu na rovnakej úrovni hĺbky pred presunutím na ďalšiu úroveň.

Fungovanie algoritmu prehľadávania do šírky môže byť popísané v nasledujúcich krokoch:

1. inicializácia

- algoritmus začne v určenom počiatočnom uzle
- tento počiatočný uzol sa pridá do radu, ktorý slúži na uchovávanie uzlov, ktoré ešte čakajú na prehľadanie

2. prehľadávanie

- vyberie sa uzol z radu (teda uzol na prvej pozícii), a preskúma všetky jeho susedné uzly (tie, ktoré ešte neboli pridané do radu)

3. opakovanie kroku 2

- krok 2 sa opakuje, kým rad nie je prázdny (križovatka nemá riešenie) alebo nebolo nájdené riešenie

4. výstup

- výstupom algoritmu je riešenie zadaného intervalu v prípade, že riešenie existuje
- ak riešenie hlavolamu neexistuje, vypíše sa chybová hláška

```
def BFS(state):  
    queue = []  
    queue.append(state)  
    checked_states = {}  
    checked_states[str(state)] = state  
    finished = False  
    final_state = None  
    try:  
        #while-cyklus, ktorý beží, až kým nenájde riešenie  
        while not finished:  
            state = queue.pop(0)  
            #for-cyklus, ktorý prechádza každé vozidlo a pre každý pohyb, ktorý môže vykonať vytvoriť nový stav  
            for i in range(len(state.vehicles)):  
                vehicle = state.vehicles[i]  
                if vehicle.orientation == "h":  
                    ran_r = state.getRange(i, "right")  
                    #stavy pre pohyb doprava  
                    for x in ran_r:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"VPRAVO({vehicle.color}, {x})")  
                        new_board.move(i, "right", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridá ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            queue.append(new_board)  
                            checked_states[str(new_board)] = new_board  
                    ran_l = state.getRange(i, "left")  
                    #stavy pre pohyb doľava  
                    for x in ran_l:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"VĽAVO({vehicle.color}, {x})")  
                        new_board.move(i, "left", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridá ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            queue.append(new_board)  
                            checked_states[str(new_board)] = new_board  
                elif vehicle.orientation == "v":  
                    ran_u = state.getRange(i, "up")  
                    #stavy pre pohyb nahor  
                    for x in ran_u:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"DOHORE({vehicle.color}, {x})")  
                        new_board.move(i, "up", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridá ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            queue.append(new_board)  
                            checked_states[str(new_board)] = new_board  
                    ran_d = state.getRange(i, "down")  
                    #stavy pre pohyb nadol  
                    for x in ran_d:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"DOLE({vehicle.color}, {x})")  
                        new_board.move(i, "down", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridá ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            queue.append(new_board)  
                            checked_states[str(new_board)] = new_board  
    except IndexError:  
        print("Úloha nemá riešenie.")  
    return final_state
```

Výhodou algoritmu prehľadávania do šírky je to, že nájde najkratšiu cestu od počiatočného uzla ku všetkým dosiahnuteľným uzlom v grafe. Tento algoritmus je však náročný na pamäť, pretože si vyžaduje uchovávanie všetkých uzlov v rade.

Prehľadávanie do hĺbky (DFS)

Tento algoritmus sa často používa na nájdenie cesty v grafe alebo na prehľadávanie všetkých možných uzlov grafu. Jeho princíp sa odlišuje od prehľadávania do šírky, pretože sa snaží dosiahnuť čo najväčšiu hĺbku v grafe pred presunutím sa na inú cestu.

Fungovanie algoritmu prehľadávania do hĺbky možno popísať nasledovne:

1. inicializácia

- algoritmus začína v určenom počiatočnom uzle

2. prehľadávanie

- algoritmus sa snaží preskúmať čo najhlbšie uzly v grafe predtým, než sa vráti späť
- začne tým, že vyberie nejaký susedný uzol, ktorý ešte nebol navštívený, a označí ho za aktuálny uzol
- tento krok sa opakuje, kým existujú ďalšie neobjavené susedné uzly

3. návrat

- ak aktuálny uzol nemá viac neobjavených susedných uzlov, algoritmus sa vráti späť na predchádzajúci uzol a pokračuje v prehľadávaní ďalej
- tento proces návratu pokračuje, kým neexistujú ďalšie neobjavené uzly v grafe.

4. opakovanie krokov 2 a 3

- krok 2 a 3 sa opakujú, kým nie je prehľadaný celý graf (úloha nemá riešenie) alebo je dosiahnutý cieľový uzol (riešenie hlavolamu)

```
def DFS(state):  
    stack = []  
    stack.append(state)  
    checked_states = {}  
    checked_states[str(state)] = state  
    finished = False  
    final_state = None  
    try:  
        #while cyklus, ktorý beží, až kým nenájde riešenie  
        while not finished:  
            state = stack.pop(-1)  
            #vyber posledného stavu zo záznamu  
            #for cyklus, ktorý prechádza každé vozidlo a pre každý pohyb, ktorý môže vykonať vytvoriť nový stav  
            for i in range(len(state.vehicles)-1, -1, -1):  
                vehicle = state.vehicles[i]  
                #vyber vozidlo  
                if vehicle.orientation == "r":  
                    ran_r = state.getRange(i, "right")  
                    #stavy pre pohyb doprava  
                    for x in ran_r:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"VPRAVO({vehicle.color}, {x})")  
                        new_board.move(i, "right", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridať ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            stack.append(new_board)  
                            checked_states[str(new_board)] = new_board  
  
                    ran_l = state.getRange(i, "left")  
                    #stavy pre pohyb doľava  
                    for x in ran_l:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"VLAVO({vehicle.color}, {x})")  
                        new_board.move(i, "left", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridať ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            stack.append(new_board)  
                            checked_states[str(new_board)] = new_board  
  
                    ran_u = state.getRange(i, "up")  
                    #stavy pre pohyb nahor  
                    for x in ran_u:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"HOORE({vehicle.color}, {x})")  
                        new_board.move(i, "up", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridať ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            stack.append(new_board)  
                            checked_states[str(new_board)] = new_board  
  
                    ran_d = state.getRange(i, "down")  
                    #stavy pre pohyb nadol  
                    for x in ran_d:  
                        #vytvorenie nového stavu  
                        veh = copy_vehicles(state.vehicles)  
                        new_board = State(veh, state, f"DOLE({vehicle.color}, {x})")  
                        new_board.move(i, "down", x)  
                        if new_board.check_end():  
                            #kontrola, či stav nie je konečný, ak je, tak skončí  
                            final_state = new_board  
                            finished = True  
                        #kontrola, či sa takýto stav ešte nenachádza v už objavených stavoch, ak nie, pridať ho do radu a do objavených stavov  
                        if not str(new_board) in checked_states:  
                            stack.append(new_board)  
                            checked_states[str(new_board)] = new_board  
  
    except IndexError:  
        print("Úloha nemá riešenie.")  
    return final_state
```

Výhodou prehľadávania do hĺbky je jeho jednoduchosť a efektivita pri prehľadávaní grafov, ktoré majú veľa hĺbkových úrovní. Na druhej strane nemusí vždy nájsť najkratšiu cestu v grafe.

Moje riešenie

V mojom riešení tohto problému som implementoval dva algoritmy BFS a DFS. Pri ich zostrojovaní som využil dve triedy:

1. trieda **Vehicle** – reprezentuje jednotlivé vozidlá

- atribúty:
 - *color* – farba vozidla
 - *size* – veľkosť vozidla (auto = 2, nákladiak = 3)
 - *row* – pozícia riadku
 - *col* – pozícia stĺpca
 - *orientation* – pohyb vozidla (v = vertikálne, h = horizontálne)
- metódy pre pohyby do jednotlivých strán

```
class Vehicle:
    def __init__(self, color, size, row, col, orientation):
        self.color = color
        self.size = size
        self.row = row
        self.col = col
        self.orientation = orientation

    #reprezentácia vozidla
    def __str__(self):
        return "(" + self.color + " " + str(self.size) + " " + str(self.row) + " " + str(self.col) + " " + self.orientation + ")"

    #pohyb vpravo
    def move_right(self, fields):
        self.col += fields

    #pohyb vľavo
    def move_left(self, fields):
        self.col -= fields

    #pohyb hore
    def move_up(self, fields):
        self.row -= fields

    #pohyb dole
    def move_down(self, fields):
        self.row += fields
```

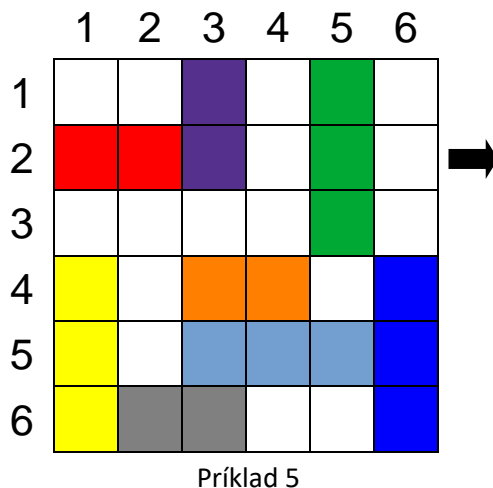
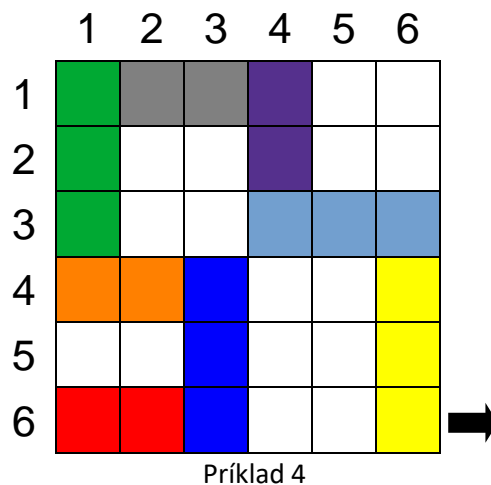
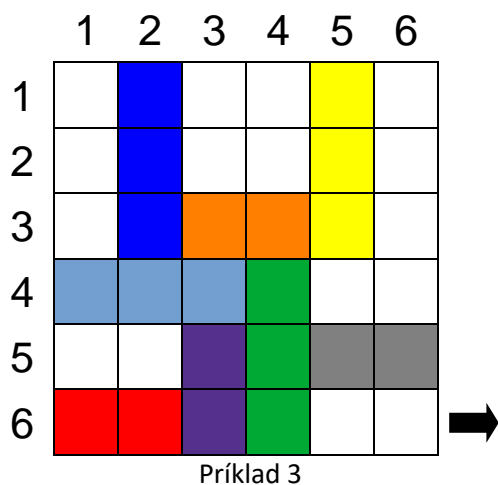
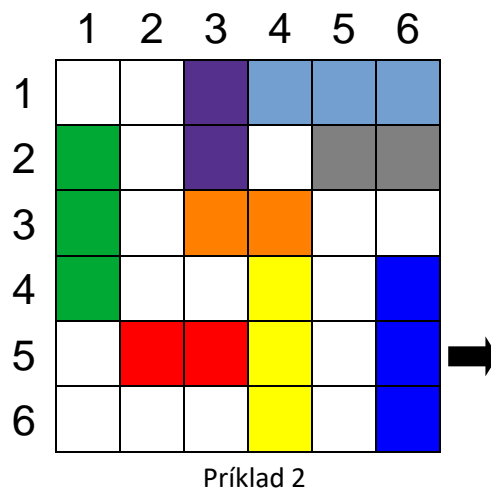
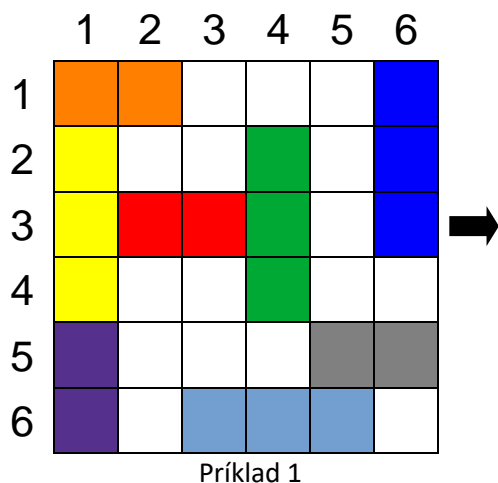
2. trieda **State** – reprezentuje jednotlivé stavy

- atribúty:
 - *vehicles* – vozidlá a ich pozície, ktoré reprezentujú daný stav
 - *predecessor* – odkaz na predchodcu, aby sme vedeli vypísať cestu
 - *movement* – správa o tom, ako sme sa do tohto stavu dostali od jeho predchodcu
- metódy:
 - *check_end(self)* – kontroluje či sa červené auto dostalo von z križovatky
 - *check_border(self, vehicle, dir, fields)* – kontroluje kolízie vozidla s hranicami, a teda kontroluje, či sa vozidlo nepokúša vyjsť z mriežky
 - *check_collision(self, i, dir, fields)* – kontroluje kolízie s ostatnými vozidlami
 - *move(self, vehicle, dir, fields)* – zabezpečuje pohyb vozidla v danom smere o zadaný počet políček
 - *getRange(self, i, dir)* - pre dané vozidlo nájde rozsah polí, kam sa môže pohybovať

Testovanie

V tomto zadaní bolo potrebné aj otestovať správnosť môjho riešenia. Vytvoril som si 5 rôznych križoviek, v ktorých bolo cieľom, aby sa červené autíčko dostalo von z križovatky. Moje testovacie scenáre som si vytvoril tak, aby som ukázal rôzne zložitosti riešenia tohto problému. V jednom z príkladov som ukázal aj križovátku, ktorá nemá riešenie.

Príklady pre otestovanie môjho riešenia:

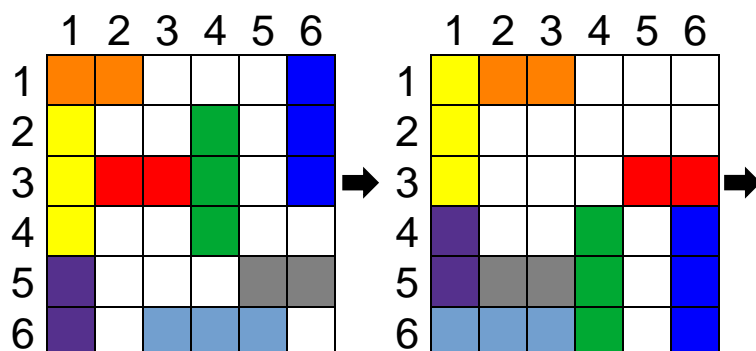


Príklad 1

Počiatkový stav:

(cervene 2 3 2 h)(oranzove 2 1 1 h)(zlte 3 2 1 v)(fialove 2 5 1 v)(zelene 3 2 4 v)(svetlomodre 3 6 3 h) (sive 2 5 5 h)(tmavomodre 3 1 6 v)

BFS:



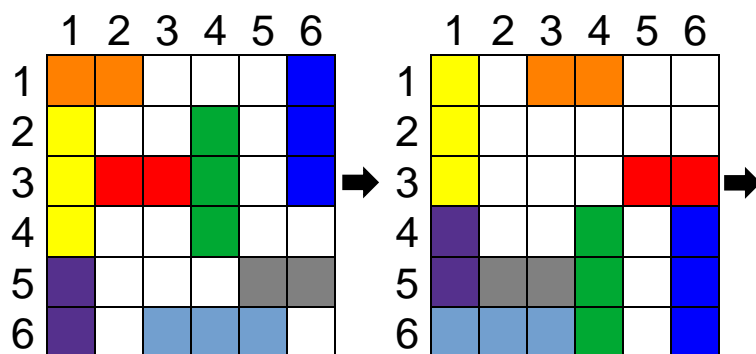
Konečný stav:

(cervene 2 3 5 h)(oranzove 2 1 2 h)(zlte 3 1 1 v)(fialove 2 4 1 v)(zelene 3 4 4 v) (svetlomodre 3 6 1 h)(sive 2 5 2 h)(tmavomodre 3 4 6 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 1 (BFS):

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

DFS:



Konečný stav:

(cervene 2 3 5 h)(oranzove 2 1 3 h)(zlte 3 1 1 v)(fialove 2 4 1 v)(zelene 3 4 4 v)(svetlomodre 3 6 1 h) (sive 2 5 2 h)(tmavomodre 3 4 6 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 1 (DFS):

VPRAVO(oranzove, 3), HORE(zlte, 1), VLAVO(oranzove, 2), HORE(fialove, 1), VPRAVO(oranzove, 1), DOLE(zelene, 1), VPRAVO(oranzove, 1), VLAVO(svetlomodre, 2), VLAVO(oranzove, 2), DOLE(zelene, 1), VPRAVO(cervene, 2), VPRAVO(oranzove, 2), VLAVO(cervene, 1), VLAVO(oranzove, 1), DOLE(tmavomodre, 1), VLAVO(cervene, 1), VPRAVO(oranzove, 2), HORE(zelene, 3), VPRAVO(svetlomodre, 3), DOLE(fialove, 1), DOLE(zlte, 1), DOLE(zelene, 2), VLAVO(oranzove, 4), HORE(zelene, 2), VPRAVO(oranzove, 1), HORE(zlte, 1), HORE(fialove, 1), DOLE(zelene, 2), VPRAVO(oranzove, 2), DOLE(fialove, 1), VLAVO(oranzove, 1), HORE(zelene, 1), DOLE(zlte, 1),

VPRAVO(oranžove, 1), VLAVO(svetlomodre, 2), VLAVO(oranžove, 3), VLAVO(sive, 3), VPRAVO(oranžove, 4), HORE(zlte, 1), VLAVO(oranžove, 3), HORE(fialove, 1), VPRAVO(oranžove, 2), DOLE(zelene, 1), VLAVO(oranžove, 1), DOLE(fialove, 1), DOLE(zlte, 1), VLAVO(oranžove, 1), HORE(zelene, 2), VPRAVO(svetlomodre, 1), VLAVO(oranžove, 1), DOLE(zelene, 2), VPRAVO(oranžove, 4), HORE(zlte, 1), VLAVO(oranžove, 3), HORE(fialove, 1), HORE(zelene, 2), VLAVO(svetlomodre, 2), DOLE(zelene, 3), VPRAVO(cervene, 2), VPRAVO(oranžove, 3), VLAVO(cervene, 1), VLAVO(oranžove, 2), DOLE(tmavomodre, 2), VPRAVO(cervene, 2)

Zhodnotenie Príkladu 1:

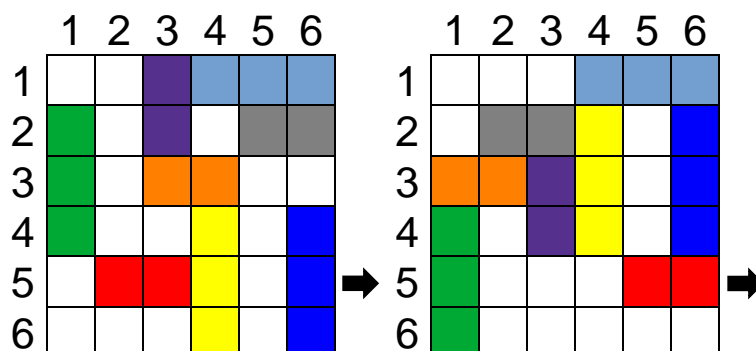
Príklad jedna je prebratý zo zadania úlohy. Na tomto príklade som ako prvom testoval správnosť môjho riešenia. Z uvedeného príkladu môžeme vidieť, že *algoritmus prehľadávania do šírky* našiel optimálne riešenie tohto príkladu, ktorý sa dá vyriešiť v 8 krokoch. *Algoritmus prehľadávania do hĺbky* sa taktiež dopracoval k správne mu riešeniu, na ktoré však potrebuje až 64 krokov.

Príklad 2

Počiatkový stav:

(cervene 2 5 2 h)(oranzove 2 3 3 h)(zlte 3 4 4 v)(fialove 2 1 3 v)(zelene 3 2 1 v)(svetlomodre 3 1 4 h) (sive 2 2 5 h)(tmavomodre 3 4 6 v)

BFS:



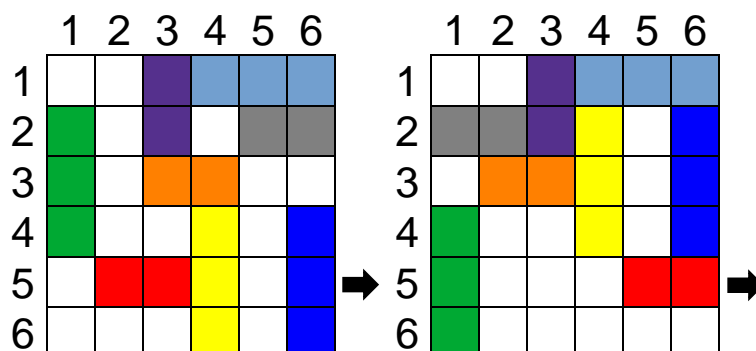
Konečný stav:

(cervene 2 5 5 h)(oranzove 2 3 1 h)(zlte 3 2 4 v)(fialove 2 3 3 v)(zelene 3 4 1 v)(svetlomodre 3 1 4 h) (sive 2 2 2 h)(tmavomodre 3 2 6 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 2 (BFS):

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

DFS:



Konečný stav:

(cervene 2 5 5 h)(oranzove 2 3 2 h)(zlte 3 2 4 v)(fialove 2 1 3 v)(zelene 3 4 1 v)(svetlomodre 3 1 4 h) (sive 2 2 1 h)(tmavomodre 3 2 6 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 2 (DFS):

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

DOLE(fialove, 2), VPRAVO(oranžove, 1), DOLE(zlte, 1), HORE(fialove, 1), VPRAVO(svetlomodre, 2), VLAVO(oranžove, 1), DOLE(fialove, 1), VLAVO(svetlomodre, 1), HORE(fialove, 3), VLAVO(cervene, 2), DOLE(zlte, 2), DOLE(fialove, 1), VPRAVO(svetlomodre, 2), HORE(fialove, 2), VPRAVO(oranžove, 3), DOLE(fialove, 2), HORE(zelene, 3), VLAVO(cervene, 1), DOLE(fialove, 2), VLAVO(oranžove, 2), HORE(zlte, 2), HORE(fialove, 1), DOLE(zlte, 1), DOLE(zelene, 1), DOLE(zlte, 1), VPRAVO(oranžove, 2), HORE(fialove, 3), VLAVO(sive, 1), VPRAVO(cervene, 1), VLAVO(oranžove, 2), HORE(zlte, 1), VLAVO(cervene, 1), HORE(zelene, 1), DOLE(zlte, 1), VPRAVO(oranžove, 1), VPRAVO(cervene, 1), DOLE(zelene, 3), DOLE(tmavomodre, 1), VLAVO(oranžove, 2), HORE(zlte, 1), VPRAVO(oranžove, 1), HORE(zelene, 3), DOLE(zlte, 1), VPRAVO(oranžove, 3), VLAVO(cervene, 1), VLAVO(oranžove, 1), DOLE(fialove, 4), VLAVO(oranžove, 2), HORE(zlte, 1), HORE(fialove, 1), DOLE(zelene, 1), DOLE(zlte, 1), VPRAVO(oranžove, 3), HORE(zlte, 1), DOLE(fialove, 1), HORE(zelene, 1), HORE(fialove, 3), VPRAVO(cervene, 1), DOLE(fialove, 1), DOLE(zelene, 3), DOLE(zlte, 1), VLAVO(oranžove, 1), HORE(fialove, 1), HORE(zelene, 3), VLAVO(svetlomodre, 2), VLAVO(cervene, 1), VPRAVO(oranžove, 1), DOLE(fialove, 3), VLAVO(oranžove, 2), HORE(fialove, 1), VLAVO(oranžove, 1), VPRAVO(svetlomodre, 1), VPRAVO(oranžove, 3), HORE(zlte, 1), HORE(fialove, 1), DOLE(zelene, 1), VPRAVO(cervene, 1), DOLE(zlte, 1), VLAVO(oranžove, 1), VLAVO(cervene, 1), DOLE(fialove, 2), VLAVO(oranžove, 2), HORE(zlte, 1), VLAVO(svetlomodre, 2), VLAVO(sive, 2), DOLE(zlte, 1), VPRAVO(oranžove, 3), HORE(zlte, 3), HORE(fialove, 2), VPRAVO(cervene, 3), DOLE(zlte, 1), VLAVO(cervene, 2), DOLE(zlte, 2), VLAVO(oranžove, 1), VLAVO(cervene, 1), DOLE(fialove, 1), VLAVO(oranžove, 1), VPRAVO(svetlomodre, 3), DOLE(fialove, 1), VPRAVO(oranžove, 1), VLAVO(svetlomodre, 2), HORE(zelene, 1), VLAVO(oranžove, 2), HORE(zlte, 2), HORE(fialove, 1), DOLE(zlte, 1), DOLE(zelene, 1), DOLE(zlte, 1), VPRAVO(oranžove, 3), HORE(zlte, 2), DOLE(fialove, 1), DOLE(zlte, 1), HORE(fialove, 2), VPRAVO(cervene, 1), DOLE(zelene, 2), DOLE(zlte, 1), VLAVO(oranžove, 1), HORE(zelene, 3), VPRAVO(svetlomodre, 2), VLAVO(cervene, 1), VPRAVO(oranžove, 1), HORE(zlte, 2), VPRAVO(cervene, 3), DOLE(fialove, 2), DOLE(zelene, 3), HORE(fialove, 1), HORE(zelene, 2), VLAVO(svetlomodre, 2), DOLE(fialove, 1), DOLE(zelene, 1), HORE(fialove, 2), VLAVO(cervene, 2), HORE(zelene, 2), VPRAVO(cervene, 1), DOLE(zelene, 3), VLAVO(svetlomodre, 1), HORE(zlte, 1), VLAVO(cervene, 1), HORE(zelene, 1), DOLE(zlte, 2), VPRAVO(svetlomodre, 3), DOLE(zlte, 1), VLAVO(svetlomodre, 1), VLAVO(oranžove, 1), VLAVO(sive, 1), HORE(fialove, 1), VPRAVO(oranžove, 1), HORE(zlte, 2), VPRAVO(cervene, 2), DOLE(fialove, 3), DOLE(zelene, 1), HORE(fialove, 2), VLAVO(cervene, 2), DOLE(zlte, 2), VLAVO(svetlomodre, 2), VLAVO(oranžove, 1), HORE(fialove, 1), VPRAVO(svetlomodre, 3), VPRAVO(oranžove, 1), HORE(zlte, 2), VPRAVO(cervene, 2), HORE(fialove, 1), VLAVO(cervene, 1), DOLE(fialove, 2), HORE(zelene, 1), VLAVO(cervene, 1), HORE(fialove, 2), DOLE(zlte, 2), VLAVO(oranžove, 3), HORE(zlte, 2), VPRAVO(cervene, 2), DOLE(zelene, 1), VLAVO(cervene, 1), VLAVO(oranžove, 1), VLAVO(cervene, 1), DOLE(zlte, 2), VPRAVO(oranžove, 2), HORE(tmavomodre, 2), VLAVO(oranžove, 1), HORE(zlte, 2), VPRAVO(cervene, 3)

Zhodnotenie Príkladu 2:


Na tomto príklade som testoval správnosť môjho riešenia. Z uvedeného príkladu môžeme vidieť, že *algoritmus prehľadávania do šírky* našiel optimálne riešenie tohto príkladu, ktorý sa dá vyriešiť v 7 krokoch. *Algoritmus prehľadávania do hĺbky* sa taktiež dopracoval k správne mu riešeniu, na ktoré však potrebuje až 218 krokov. Na tomto príklade môžeme vidieť obrovský rozdiel medzi týmito dvomi algoritmami.

Príklad 3

Počiatkový stav:

(cervene 2 6 1 h)(oranzove 2 3 3 h)(zlte 3 1 5 v)(fialove 2 5 3 v)(zelene 3 4 4 v)(svetlomodre 3 4 1 h) (sive 2 5 5 h)(tmavomodre 3 1 2 v)

	1	2	3	4	5	6
1		blue			yellow	
2		blue			yellow	
3		blue	orange	orange	yellow	
4	lightblue	lightblue	lightblue	green		
5			purple	green	gray	gray
6	red	red	purple	green		



BFS:

Úloha nemá riešenie.

DFS:

Úloha nemá riešenie.

Zhodnotenie Príkladu 3:

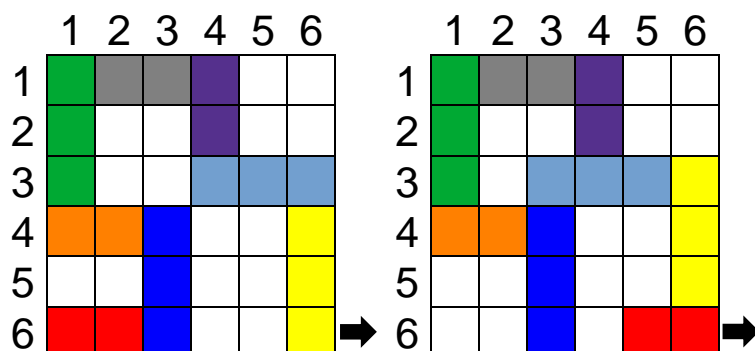
Na tomto príklade som testoval správnosť môjho riešenia, konkrétne som chcel ukázať, že nie každá križovatka má riešenie a čo sa vypíše, keď sa tak stane. Z uvedeného príkladu môžeme vidieť, že *algoritmus prehľadávania do šírky* ani *algoritmus prehľadávania do hĺbky* nenašiel riešenie tohto príkladu a po vyprázdnení radu / zásobníka (prehľadani všetkých stavov), skončil s hlásením, že tento príklad nemá riešenie.

Príklad 4

Počiatkový stav:

(cervene 2 6 1 h)(oranzove 2 4 1 h)(zlte 3 4 6 v)(fialove 2 1 4 v)(zelene 3 1 1 v)(svetlomodre 3 3 4 h) (sive 2 1 2 h)(tmavomodre 3 4 3 v)

BFS:



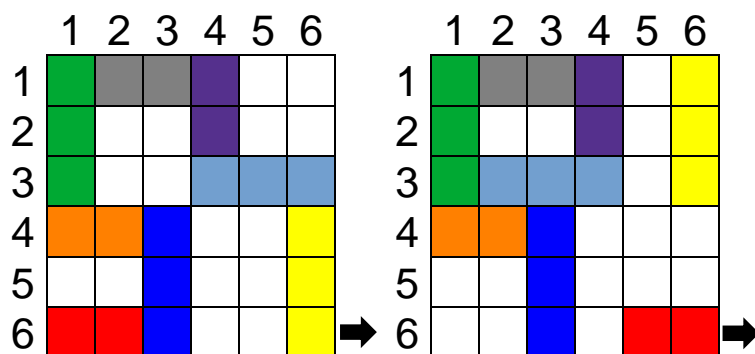
Konečný stav:

(cervene 2 6 5 h)(oranzove 2 4 1 h)(zlte 3 3 6 v)(fialove 2 1 4 v)(zelene 3 1 1 v)(svetlomodre 3 3 3 h) (sive 2 1 2 h)(tmavomodre 3 4 3 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 4 (BFS):

HORE(tmavomodre, 1), VPRAVO(cervene, 3), DOLE(tmavomodre, 1), VLAVO(svetlomodre, 1), HORE(zlte, 1), VPRAVO(cervene, 1)

DFS:



Konečný stav:

(cervene 2 6 5 h)(oranzove 2 4 1 h)(zlte 3 1 6 v)(fialove 2 1 4 v)(zelene 3 1 1 v)(svetlomodre 3 3 2 h) (sive 2 1 2 h)(tmavomodre 3 4 3 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 4 (DFS):

HORE(tmavomodre, 2), VPRAVO(cervene, 3), DOLE(tmavomodre, 2), VLAVO(svetlomodre, 2), HORE(zlte, 3), VPRAVO(cervene, 1)

Zhodnotenie Príkladu 4:

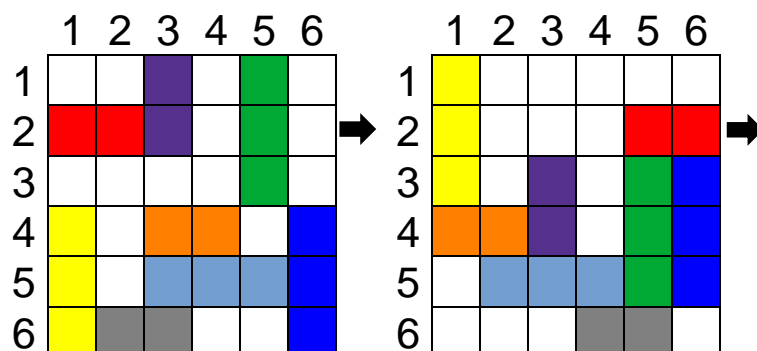
Na príklade číslo 4 som chcel ukázať, že nie vždy musí byť rozdiel v počte posunov pre riešenie problému pri použití algoritmov *prehľadávania do šírky* a *prehľadávania do hĺbky*, taký veľký. Na tomto príklade vidíme, že riešenia sú takmer identické.

Príklad 5

Počiatkový stav:

(cervene 2 2 1 h)(oranzove 2 4 3 h)(zlte 3 4 1 v)(fialove 2 1 3 v)(zelene 3 1 5 v)(svetlomodre 3 5 3 h) (sive 2 6 2 h)(tmavomodre 3 4 6 v)

BFS:



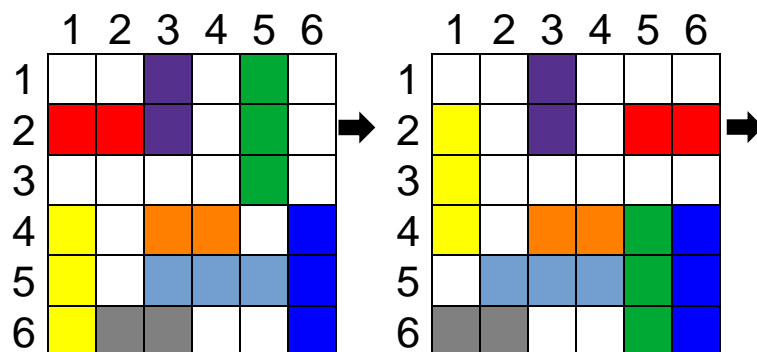
Konečný stav:

(cervene 2 2 5 h)(oranzove 2 4 1 h)(zlte 3 1 1 v)(fialove 2 3 3 v)(zelene 3 3 5 v)(svetlomodre 3 5 2 h) (sive 2 6 4 h)(tmavomodre 3 3 6 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 5 (BFS):

VPRAVO(oranzove, 1), VPRAVO(sive, 2), HORE(tmavomodre, 2), VPRAVO(svetlomodre, 1), DOLE(fialove, 4), VPRAVO(cervene, 1), HORE(zlte, 3), VLAVO(oranzove, 3), HORE(fialove, 2), VLAVO(svetlomodre, 2), DOLE(zelene, 2), DOLE(tmavomodre, 1), VPRAVO(cervene, 3)

DFS:



Konečný stav:

(cervene 2 2 5 h)(oranzove 2 4 3 h)(zlte 3 2 1 v)(fialove 2 1 3 v)(zelene 3 4 5 v)(svetlomodre 3 5 2 h) (sive 2 6 1 h)(tmavomodre 3 4 6 v)

Postupnosť posunov vozidiel z počiatkovej do cieľovej pozície pre Príklad 5 (DFS):

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

VLAVO(oranžove, 1), VLAVO(svetlomodre, 1), HORE(zlte, 1), DOLE(zelene, 1), VPRAVO(oranžove, 1), DOLE(zlte, 1), DOLE(fialove, 1), VLAVO(oranžove, 1), HORE(zelene, 2), VPRAVO(oranžove, 2), HORE(zlte, 1), VPRAVO(svetlomodre, 1), VLAVO(sive, 3), HORE(tmavomodre, 3), VLAVO(oranžove, 2), HORE(fialove, 1), VPRAVO(oranžove, 3), DOLE(fialove, 2), VPRAVO(cervene, 2), VLAVO(oranžove, 1), VLAVO(cervene, 1), HORE(zlte, 2), VPRAVO(oranžove, 1), DOLE(zlte, 1), VLAVO(svetlomodre, 2), VPRAVO(cervene, 1), VLAVO(oranžove, 1), HORE(zlte, 1), VPRAVO(svetlomodre, 3), VPRAVO(oranžove, 1), DOLE(fialove, 2), VLAVO(cervene, 1), VLAVO(oranžove, 4), HORE(fialove, 2), VPRAVO(cervene, 1), DOLE(fialove, 1), DOLE(zelene, 1), VLAVO(cervene, 1), VPRAVO(sive, 4), DOLE(fialove, 1), VPRAVO(cervene, 1), VPRAVO(oranžove, 2), DOLE(zlte, 3), VLAVO(cervene, 2), VLAVO(oranžove, 1), VPRAVO(cervene, 1), HORE(zlte, 2), VPRAVO(cervene, 1), DOLE(zlte, 1), HORE(zelene, 1), VLAVO(cervene, 2), VPRAVO(oranžove, 3), VPRAVO(cervene, 1), VLAVO(oranžove, 2), DOLE(zlte, 1), VPRAVO(oranžove, 1), VLAVO(cervene, 1), HORE(fialove, 4), VLAVO(oranžove, 2), HORE(zlte, 1), VPRAVO(oranžove, 1), DOLE(fialove, 1), VPRAVO(oranžove, 1), DOLE(fialove, 2), VPRAVO(cervene, 2), VPRAVO(oranžove, 1), DOLE(zlte, 1), VLAVO(cervene, 2), HORE(fialove, 2), VLAVO(svetlomodre, 2), VLAVO(oranžove, 3), HORE(zlte, 1), DOLE(zelene, 2), VPRAVO(oranžove, 1), DOLE(zlte, 1), HORE(fialove, 1), VLAVO(oranžove, 1), HORE(zelene, 1), HORE(zlte, 1), VPRAVO(oranžove, 1), VPRAVO(svetlomodre, 1), DOLE(zlte, 1), DOLE(fialove, 1), VLAVO(oranžove, 1), HORE(zlte, 1), VPRAVO(svetlomodre, 1), VLAVO(sive, 4), VPRAVO(oranžove, 1), HORE(fialove, 1), VPRAVO(sive, 3), VLAVO(oranžove, 1), DOLE(zlte, 1), DOLE(fialove, 1), VPRAVO(oranžove, 1), HORE(zelene, 1), VPRAVO(oranžove, 1), DOLE(fialove, 2), VPRAVO(cervene, 2), HORE(zlte, 3), VLAVO(cervene, 1), VPRAVO(oranžove, 1), DOLE(zlte, 3), DOLE(fialove, 1), VLAVO(cervene, 1), VLAVO(oranžove, 3), VPRAVO(cervene, 2), VPRAVO(oranžove, 1), HORE(zlte, 3), VLAVO(cervene, 1), VLAVO(oranžove, 2), HORE(fialove, 1), VLAVO(sive, 2), DOLE(tmavomodre, 1), VPRAVO(cervene, 1), HORE(fialove, 1), DOLE(zelene, 1), VLAVO(cervene, 1), VLAVO(svetlomodre, 3), DOLE(zelene, 2), VPRAVO(cervene, 2), HORE(fialove, 2), VPRAVO(oranžove, 2), DOLE(zlte, 1), VLAVO(oranžove, 1), DOLE(fialove, 1), HORE(zlte, 1), HORE(zelene, 1), VLAVO(oranžove, 1), VPRAVO(svetlomodre, 1), VPRAVO(oranžove, 2), DOLE(zlte, 3), VLAVO(oranžove, 1), HORE(zlte, 2), HORE(fialove, 1), VPRAVO(oranžove, 1), DOLE(zlte, 1), DOLE(zelene, 1), VLAVO(oranžove, 1), DOLE(zlte, 1), VPRAVO(sive, 1), VPRAVO(oranžove, 1), HORE(zlte, 3), VLAVO(oranžove, 2), DOLE(fialove, 2), VLAVO(cervene, 2), HORE(zelene, 3), VPRAVO(cervene, 1), DOLE(zelene, 2), VLAVO(svetlomodre, 1), VLAVO(cervene, 1), VLAVO(sive, 2), VPRAVO(cervene, 2), HORE(fialove, 2), VPRAVO(oranžove, 2), DOLE(zlte, 1), VLAVO(oranžove, 1), DOLE(fialove, 1), VPRAVO(sive, 4), VPRAVO(oranžove, 1), HORE(zlte, 1), VLAVO(sive, 2), DOLE(zelene, 1), VLAVO(oranžove, 2), VLAVO(sive, 2), VPRAVO(svetlomodre, 1), VPRAVO(oranžove, 2), DOLE(zlte, 2), VLAVO(oranžove, 1), HORE(zlte, 1), HORE(fialove, 1), VPRAVO(oranžove, 1), DOLE(tmavomodre, 2), VPRAVO(cervene, 1)

Zhodnotenie Príkladu 5:

Na príklade číslo 5 som chcel ukázať zložitejšie riešenie tohto problému. Algoritmus prehľadávania do šírky našiel optimálne riešenie tohto problému, ktoré sa dá vykonať v 13 krokoch. Riešenie, ktoré našiel algoritmus prehľadávania do hĺbky síce nenašiel optimálne riešenie, ale taktiež sa dopracoval k výsledku, na toto riešenie potreboval 203 krokov.

Zhodnotenie

Mojou úlohou v tomto zadaní bolo implementovať a porovnať algoritmus prehľadávania do šírky a algoritmus prehľadávania do hĺbky.

Prehľadávanie do šírky a prehľadávanie do hĺbky sú dva základné algoritmy prehľadávania stavov v grafoch. Prehľadávanie do šírky (BFS) sa zameriava na objavovanie najkratších ciest a využíva rad na ukladanie stavov. Na druhej strane, prehľadávanie do hĺbky (DFS) sa zameriava na hlboké vetvenie stromu a využíva zásobník na udržiavanie cesty.

BFS má výhodu v nájdení najkratších ciest a objavuje vždy optimálne riešenia. Avšak môže vyžadovať veľa pamäte pre ukladanie stavov v rade. Naopak, DFS je efektívny v stromoch s veľkým vetvením a má nižšie nároky na pamäť, ale nemusí nájsť najkratšiu cestu.

V tabuľke nižšie môžeme vidieť porovnanie týchto algoritmov z hľadiska krokov potrebných na nájdenie riešenia pri jednotlivých príkladoch:

	Prehľadávanie do šírky		Prehľadávanie do hĺbky	
	Počet krokov	Čas [ms]	Počet krokov	Čas [ms]
Príklad 1	8	201	64	16
Príklad 2	7	329	218	57
Príklad 3	-	-	-	-
Príklad 4	6	0	6	0
Príklad 5	13	489	203	56

Z tabuľky môžeme vidieť, že prehľadávanie do šírky nájde vždy optimálne riešenie s najnižším počtom krokov. Na druhej strane prehľadávanie do hĺbky potrebuje, vo väčšine príkladov, tých krokov oveľa viac. V jednom prípade si môžeme všimnúť, že počet krokov je taký istý pre obidva použité algoritmy.

Keď sa však pozrieme na čas, ktorý dané algoritmy potrebujú na ich vykonanie, môžeme vidieť, že prehľadávanie do hĺbky je vo väčšine prípadov rýchlejšie.