

FIIT STU

Prehľadávanie stavového priestoru

Dokumentácia

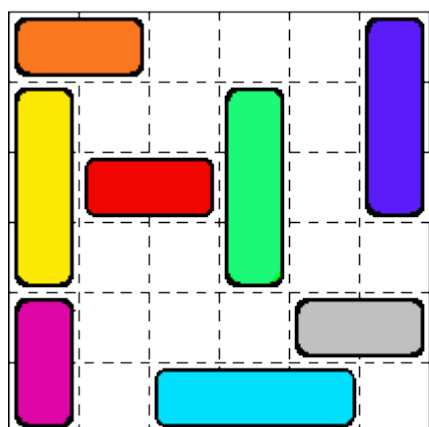
Meno: Martin Šváb
Študijný program: Informatika
Ročník: 2, cvičenie: štvrtok 14:00
Predmet: Umelá Inteligencia
Cvičiaci: Ing. Ivan Kapustík
Akademický rok: 2020/2021

Zadanie úlohy

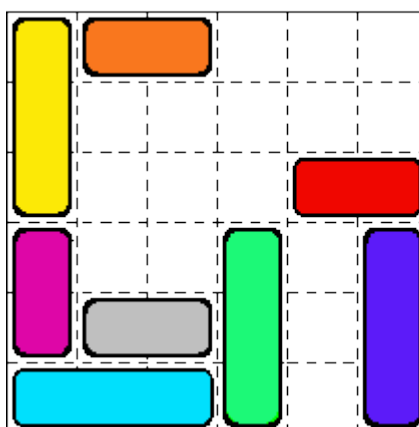
Úlohou je nájsť riešenie hlavolamu **Bláznivá križovatka**. Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6 krát 6 políček a obsahuje niekoľko vozidiel (áut a nákladiakov) rozložených na mriežke tak, aby sa neprekrývali. Všetky vozidlá majú šírku 1 políčko, autá sú dlhé 2 a nákladiaky sú dlhé 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 políček, môže sa vozidlo pohnúť o 1 až n políček 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.

Počiatočná pozícia



Cieľová pozícia



Obr. 1 Počiatočná a cieľová pozícia hlavolamu Bláznivá križovatka.

Postupnosť posunov vozidiel z počiatočnej do cieľovej pozície z obr.1 je:

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

Riešenie úlohy

Stručný opis riešenia:

Na riešenie tohto problému som použil algoritmus breadth-first search a depth-first search. Teda náhodne som pohyboval autami a tým prechádzal medzi rôznymi stavmi až kým som sa nedostal do finálneho stavu. Navštívené a preskúmané stavy sa ukladajú do polí aby sa program nezacyklil a neprechádzalo by sa dookola tými istými stavmi. Poradie postupného navštevovania a prehľadávania uzlov je dané zvoleným algoritmom. Výpočet algoritmu sa končí vtedy, keď už neexistujú žiadne nenavštívené uzly alebo bol nájdený uzol s finálnym stavom. Po vyhľadání cesty sa vypíšu použité inštrukcie, ich celková cena a čas výpočtu.

Program číta vstupný stav a rozmer hlavolamu z externého súboru. Výstup je vypísaný do konzoly.

Reprezentácia údajov:

V tomto programe boli použité 3 triedy:

- 1) Car – reprezentuje auto. Sú tu uložené všetky informácie o aute ktoré sa nachádza v križovatke.

Stav je teda reprezentovaný ako pole triedy Car. Na výpočet kolízií používam aj dvojrozmerné pole triedy Car – križovatku (ak je na pozícii None, auto tam nie je).

```
class Car:
    def __init__(self, color, size, pos_y, pos_x, is_horizontal):
        self.color = color
        self.size = size
        self.pos_y = pos_y
        self.pos_x = pos_x
        self.is_horizontal = is_horizontal
```

- 2) StateNode – reprezentuje uzol. Obsahuje stav, predošlý uzol, cenu cesty a použitú inštrukciu.

```
class StateNode:
    def __init__(self, state, instruction, prev, cost):
        self.state = state
        self.instruction = instruction
        self.prev = prev
        self.cost = cost
```

- 3) Instruction – reprezentuje použitú inštrukciu. Obsahuje použitý smer pohybu, farba presunutého auta a prejdenú vzdialenosť.

```
class Instruction:
    def __init__(self, direction, color, distance):
        self.direction = direction
        self.color = color
        self.distance = distance
```

Použitý algoritmus:

Breadth-first search:

- 1) Vytvor počiatočný uzol a umiestni ho medzi navštívené uzly
- 2) Ak neexistuje navštívený neprehľadaný uzol, ukonči s neúspechom
- 3) Vyber 1. uzol z poľa navštívených uzlov a označ ho ako aktuálny
- 4) Ak tento uzol obsahuje cieľový stav ukonči s úspechom
- 5) Pohni všetkými autami dopredu a dozadu, pričom skontroluj či vzniknutý stav už nebol navštívený alebo preskúmaný, resp. či je pohyb povolený (auto nejde mimo mriežku a nie je zablokované iným autom). Ak je stav tohto uzla nový, tak ho vlož do poľa navštívených uzlov.
- 6) Presuň aktuálny uzol do poľa preskúmaných uzlov.
- 7) Chod' na krok 2.

Depth-first search:

- 1) Vytvor počiatočný uzol a umiestni ho medzi navštívené uzly
- 2) Ak neexistuje navštívený neprehľadaný uzol, ukonči s neúspechom
- 3) Vyber 1. uzol z poľa navštívených uzlov a označ ho ako aktuálny
- 4) Ak tento uzol obsahuje cieľový stav ukonči s úspechom
- 5) Pohni všetkými autami dopredu a dozadu, pričom skontroluj či vzniknutý stav už nebol navštívený alebo preskúmaný, resp. či je pohyb povolený (auto nejde mimo mriežku a nie je zablokované iným autom). Ak je stav tohto uzla nový, tak ho vlož do poľa navštívených uzlov. **Tento prvok hneď preskúmaj – kroky algoritmu 4, 5 a 6!**
- 6) Presuň aktuálny uzol do poľa preskúmaných uzlov.
- 7) Chod' na krok 2.

Spôsob testovania:

Na testovanie boli použité vstupné súbory v projektovom adresári "Input". Na overenie efektívnosti algoritmov som použil čas potrebný na vykonanie programu a cena výslednej cesty pre obe algoritmy. Program by sa dal rozšíriť pridaním algoritmu cyklicky sa prehlbujúceho hľadania.

Vstup1.txt:

BFS:

Riešenie:

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

Cena cesty: 16

Výpočet trval 3.3799662590026855 sekúnd.

DFS

Riešenie:

Je príliš dlhé na výpis do dokumentácie.

Cena cesty: 923

Výpočet trval 0.6143569946289062 sekúnd.

Vstup2.txt:

BFS:

Riešenie:

HORE(fialove, 1)
VPRAVO(modre, 1)
HORE(zelene, 2)
VPRAVO(cervene, 2)
HORE(zelene, 1)
VLAVO(modre, 2)
HORE(zlte, 2)
VPRAVO(cervene, 2)

Cena cesty: 13

Výpočet trval 0.15761089324951172 sekúnd.

DFS

Riešenie:

Je príliš dlhé na výpis do dokumentácie.

Cena cesty: 124

Výpočet trval 0.044879913330078125 sekúnd.

Vstup3.txt

BFS:

Riešenie nebolo nájdené!

Výpočet trval 0.004960060119628906 sekúnd.

DFS

Riešenie nebolo nájdené!

Výpočet trval 0.005017757415771484 sekúnd.

Zhodnotenie riešenia

Algoritmy bfs a dfs sú totožné až na 5. krok, v ktorom dfs ihneď preskúma nový stav aj všetkých jeho potomkov. Moja implementácia je schopná pracovať s rôznymi rozmermi mriežky hlavolamu.

Porovnanie vlastností použitých metód:

Z výsledkov testovania je vidieť značný rozdiel medzi použitými algoritmami breadth-first search a depth-first search.

Breadth-first search je pomalší algoritmus, ktorý však vždy vyhladá najkratšiu cestu.

Depth-first search je síce veľmi rýchly algoritmus ale vo väčšine prípadov vyhladá veľmi zdĺhavé a neefektívne riešenie problému.