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

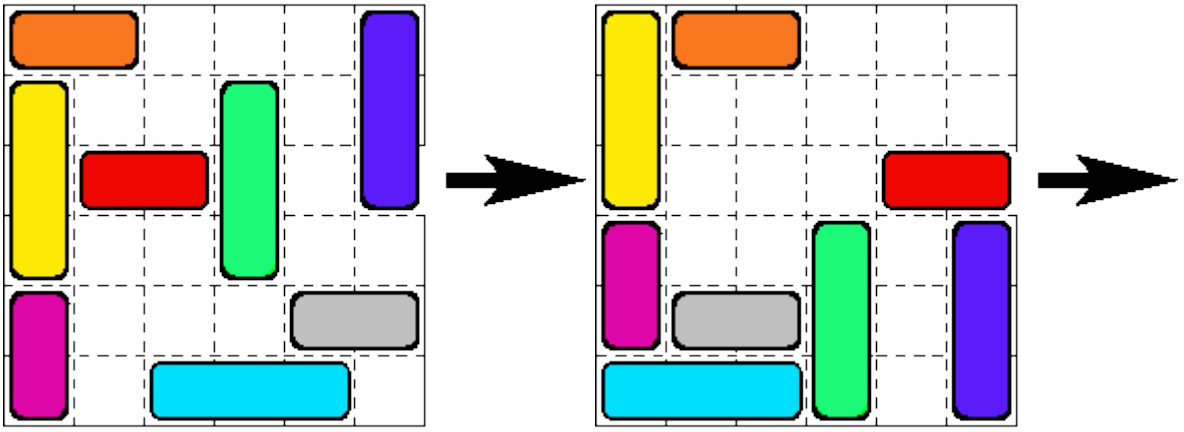
Zadanie úlohy

Úlohou je nájsť riešenie hlavolamu **Bláznivá križovatka**. Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6 krát 6 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.

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(oranzove, 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 vyhľadaní 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 – rezprezentuje auto. Sú tu uložené všetky informácie o aute ktoré sa nachádza v križovatke.

Stav je teda reprezentovaný ako pole triedy Car. Na výpočet kolízii 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) Choď 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) Choď na krok 2.

Spôsob testovania:

Na testovanie boli použité vstupné súbory v projektovom adresáry “Input”. Na overenie efektívnosti algoritmov som použil čas potrebný na vykonanie programu a cena výslednej cesty pre obe algoritmy.

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.

Výstup z tohto súboru je korektný. Tento hlavolam naozaj nemá riešenie.

Vstup4.txt
BFS
Riešenie:
HORE(zlte, 1)
VLAVO(zelene, 2)
HORE(oranzove, 1)
VPRAVO(cervene, 2)

Cena cesty: 6
Výpočet trval 0.003989458084106445 sekúnd.

DFS
Riešenie:
HORE(zlte, 1)
VLAVO(zelene, 1)
DOLE(zlte, 1)
VLAVO(zelene, 1)
HORE(oranzove, 1)
VPRAVO(cervene, 1)
HORE(zlte, 1)
VPRAVO(cervene, 1)

Cena cesty: 8
Výpočet trval 0.0009968280792236328 sekúnd.

Týmto testom som dokázal, že program je schopný pracovať s hlavolam s menším rozmerom ako 6x6.

Vstup5.txt
BFS
Riešenie:
DOLE(fialove, 1)
VLAVO(modre, 1)
DOLE(fialove, 1)
VLAVO(zelene, 1)
DOLE(zlte, 1)
VLAVO(zelene, 1)
DOLE(oranzove, 2)
VPRAVO(cervene, 3)

Cena cesty: 11
Výpočet trval 32.993247985839844 sekúnd.

DFS
Riešenie:
Je príliš dlhé na výpis do dokumentácie.

Cena cesty: 587
Výpočet trval 0.19148755073547363 sekúnd.

Týmto testom som dokázal, že program je schopný pracovať s hlavolam s väčším rozmerom ako 6x6.

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. Program by sa dal rozšíriť pridaním algoritmu cyklicky sa prehlbujúceho hľadania.

Porovnanie vlastností použitých metód pre rôznu dĺžku riešenia:

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 vyhľadá najkratšiu cestu. Pri veľkých mriežkach je hľadanie riešenia veľmi zdĺhavé.

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