Fakulta informatiky a informačných technológií

Slovenská technická univerzita

Zadanie 02
Prehľadávanie stavového priestoru dokumentácia

Meno: Samuel Hanák

Cvičenia: pondelok 16:00

Predmet: Umelá inteligencia

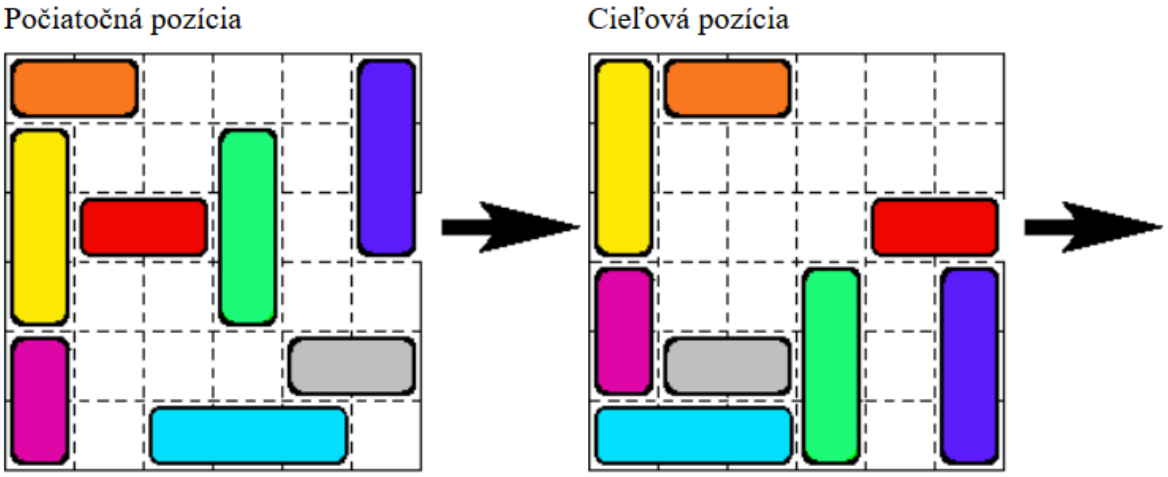
Cvičiaci: Jozef Kováč

Ak. rok: 2018/19

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 z nej teda 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.

Použite algoritmus prehľadávania do šírky a do hĺbky. Porovnajte ich výsledky.

2. Špecifikácia požiadaviek

Moje riešenie som implementoval pomocou programovacieho jazyku Python 3.6.3 na platforme Windows 10 Pro. Ako vývojové prostredie som použil PyCharm 2017 2.4. Ako prídavné moduly do môjho zadanie som použil json a copy.

3. Návrh riešenia

Úlohu som riešil v obidvoch prípadoch podobne, jediným rozdielom bolo prideľovanie ďalšej bunky na preskúmanie. Vytvoril som strom všetkých možných ťahov a ukončil som prehľadávanie, keď som našiel taký stav pri ktorom bolo červené autíčko na pravej strane hlavolamu. Počas behu som riešil aj cyklenie stavov, čo spôsobovalo vytváranie rovnakých stavov dookola.

3.1. Stavy a operátory

V mojom zadaní reprezentoval stav množinu áut v hlavolame napr:

((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))

Formát dát je nasledovný ($auto $Y-axis $X-axis $velkost $poloha)

Operátory sú nasledovné:

- VPRAVO($stav, $auto, $počet) ➔ nový_stav
- VLAVO($stav, $auto, $počet) ➔ nový_stav
- HORE($stav, $auto, $počet) ➔ nový_stav
- DOLE($stav, $auto, $počet) ➔ nový_stav

3.2. Uzol

V uzle som si reprezentoval údaje nasledovným spôsobom:

Prehľadávanie do šírky:

- Stav križovatky
- Vykonaný ťah
- Index rodiča

- Farba auta

Prehľadávanie do hĺbky:

- Stav križovatky
- Vykonaný ťah
- Index rodiča
- Farba auta
- List indexov detí
- Práve prehľadávaný index dieťaťa

3.3. Popis riešenia

Vstupy som si uložil do príslušných súborov vo formáte JSON, ktoré si následne vyberám a uložím do štruktúr. Následne idem vypočítať postupnosť ťahov prehľadávaním do šírky a hĺbky, výsledky vypíšem a určím lepšiu cestu.

3.3.1. Prehľadávanie do šírky

Prvým ťahom je pripnutie vstupného stav do koreňa stromu a nastavenie premenných na počiatočné stavy.

Následne prejdem všetky uzly v danej úrovni a pre každý uzol vytvorím všetky možné ťahy autami, okrem auta ktoré sa hýbalo v predchádzajúcom ťahu. Ťah je legálny, ak mu na ceste neprekáža nijaké auto a neprejde za križovatku. Každý ťah je poistený proti slučkám, teda prejde cestu až po koreň a ak sa tam nachádza rovnaký stav, tak našiel slučku a daný ťah nepripojí. Po prejdení všetkých uzlov prejdem o úroveň nižšie a opakujem proces. V prípade, ak sa v danej úrovni nenašiel ťah, riešenie neexistuje.

Algoritmus končí ak sa červené auto nachádza na pravom kraji hlavolamu.

3.3.2. Prehľadávanie do hĺbky

Prvým ťahom je pripnutie vstupného stav do koreňa stromu a nastavenie premenných na počiatočné stavy.

Následne prejdem nespracovaný stav z danej úrovne a nájdem ku nemu všetky možné ťahy. Ťahy sú podobne ako v prehľadávaní do šírky poistené proti slučkám a vykonaný je len ak je legálny. Následne si vyberiem náhodný detský uzol a posuniem sa o úroveň nižšie. V prípade ak nemá dieťa, posúva sa o úroveň nižšie pokiaľ nenájde nespracovaný detský proces. Ak ho nenájde, riešenie neexistuje.

3.4. Spôsob testovania

Zadanie som si testoval na rôznych vstupoch, na jednoduchých aj zložitejších. Avšak pri príliš zložitých vstupoch moje riešenie nebolo dostačujúce. Testoval som aj na riešeniach, ktoré nemali riešenie a každé riešenie dodalo požadovaný výsledok. Testovacie vstupy sú priložené ku riešeniu ako dvojice:

CarsX.json – CrossroadsSizeX.json (X = číslo)

Dané súbory sa definujú pri spustení programu.

Príklad vstupu:

```
Enter file with sizes(e.g CrossroadsSize.json): CrossroadsSize1.json
Enter file with state of croosroads(e.g Cars.json): Cars1.json
```

Príklad výstupu:

```
DFS output:  VPRAVO(cervene, 1), HORE(fialove, 1), VLAVO(oranzove, 1), HORE(fialove, 1), VLAVO(zlte, 1), DOLE(modre, 2), VPRAVO(cervene, 1)
BFS output:  VPRAVO(cervene, 1), HORE(fialove, 2), VLAVO(zlte, 1), VLAVO(oranzove, 1), DOLE(modre, 2), VPRAVO(cervene, 1)
Path return from BFS is better,  6  moves only

Process finished with exit code 0
```

4. Zhodnotenie

Moje riešenie je korektné, ale nepoužiteľné pre zložitejšie vstupy a pre lepšiu efektivitu potrebuje dodatočné rozšírenia, lebo jeho zložitosť rastie exponenciálne. Snažil som sa implementovať rozšírenie, ktoré by nepovoľovalo ťahy, ktoré nemajú váhu, napr. viem autom pohnúť o 1,2,3,4 políčka, ale všetky ťahy majú rovnaký dopad na ostatné autá, tak ho vykonám iba raz. Bohužiaľ po niekoľkých pokusoch sa mi dané rozšírenie implementovať nepodarilo.

Pre algoritmus je podľa môjho zhodnotenia dôležité mať nejakú pomôcku, podľa ktorej sa rozhodne, ktorý ťah je najvyhovujúcejší, pretože ak sa rozhoduje náhodne dostáva sa do veľkých hĺbok, pričom pri prehľadávaní do šírky by sa dostal maximálne do 1/3 hĺbky. Tu som chcel implementovať mechanizmus, ktorý by dal do priority autá, ktoré zavadzajú červenému autu prejsť do cieľa. Avšak pri implementácií som sa viac-krát dostal do zložitej situácie, ktorú som nevedel vyriešiť.