Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Umelá Inteligencia
Zadanie 2 – Problém 1, a)

Martin Rudolf

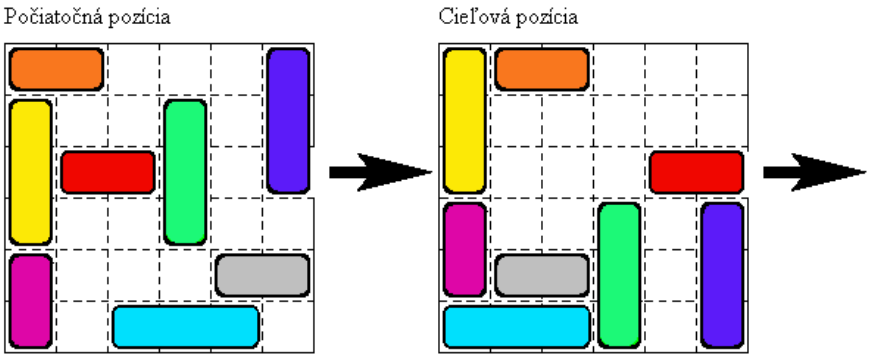
Cvičenie: Štvrtok 16.00
Cvičiaci: Ing. Boris Slíž

2020/2021

Definovanie problému 1

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.

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 ku bráne. Farba vozidla sa môže vynechať, ak ho konkrétna implementácia nevyžaduje. Veľkosť je vždy 2 alebo 3. Súradnice zodpovedajú ľavému hornému rohu automobilu a v tomto príklade sú súradnice počítané od ľavého horného rohu križovatky a začínajú od jednotky, prvá určuje riadok. Smer možného pohybu automobilu určuje **h** (horizontálny) pre pohyb vľavo a vpravo a **v** (vertikálny) pre pohyb hore a dole.

Vstupom algoritmov je začiatočný stav. Cieľový stav je definovaný tak, že červené auto je na najpravejšej pozícii v riadku. To vo všeobecnosti definuje celú množinu cieľových stavov a nás nezaujíma, ktorý z nich bude vo výslednom riešení.

OPERÁTORY

Operátory sú len štyri:

```
(VPRAVO stav vozidlo počet) (VLAVO stav vozidlo počet) (DOLE stav vozidlo
počet) a (HORE stav vozidlo počet)
```

Operátor dostane nejaký stav, farbu (poradie) vozidla a počet políčok, o ktoré sa má vozidlo posunúť. Ak je možné vozidlo s danou farbou o zadaný počet políčok posunúť, vráti nový stav. Ak operátor na vstup nie je možné použiť, výstup nie je definovaný. V konkrétnej implementácii je potrebné výstup buď vhodne dodefinovať, alebo zabrániť volaniu nepoužiteľného operátora. **Všetky operátory pre tento problém majú rovnakú váhu.**

Príklad použitia operátora VPRAVO, pre oranzove auto a posun o 1:

Vstupný 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))
```

Výstupný stav:

```
((cervene 2 3 2 h)(oranzove 2 1 2 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))
```

Riešenie problému

Na riešenie problému boli použité algoritmy na prehľadávanie do šírky (BFS) a na prehľadávanie do hĺbky (DFS).

Pri BFS algoritme využívame FIFO (first in fist out) metódu, čo znamená že vyhodnocujeme uzol ktorý sa nachádza vo fronte na prvom mieste (fronta[0]). Ak tento uzol vyhodnotíme ako nie koncový uzol a vo fronte už nie je žiaden iný uzol tak sa nepodarilo nájť riešenie. No ak fronta nie je prázna a aktuálny uzol sa nenachádza už v navštívených tak z aktuálneho uzla vygenerujeme uzly so všetkými možnými krokmi a priradíme ich do fronty. Aktuálny uzol priradíme k navštíveným a vyradíme ho fronty. Vraciame sa ku kroku 1. Ak aktuálny uzol je koncový tak vypíšeme akcie koncového uzla a aj jeho predchodcov.

DFS algoritmus využíva metódu LIFO (last in last out), pracujeme s uzlom ktorý ako posledný pridal do stacku, ak to je koncový stav vypíš sa postupnosť krokov, ako sa k nemu algoritmus dostal, a ukončí sa. Ak aktuálny stav nie je koncový a stack je prázdny, algoritmus vypíše nezdar a ukončí sa. Inak ak aktuálny uzol nebol ešte prehľadávaný, algoritmus ho priradí do prehľadavaných uzlov, vygeneruje uzly so všetkými možnými krokmi a priradí ich do stacku a vracia sa k prvému kroku.

Na riešenie daného problému som si vytvoril triedu reprzentujúcu auto na križovatke. Zoznam atribútov: farba, veľkosť, y, x, smer, unik(informácia či to je auto ktoré má opustiť križovatku). Triedu Graph na reprezentovanie uzla s atribútmi: pohyb (instancia triedy pohyb ktorá nesie data o poslednom pohybe), unikoveAuto (inštancia auta ktoré ma atribút unik = True), parent (referencia na rodiča, na uzol z ktorého vznikol), sirka, dlzka, cars(pole aut v danom uzle), krizovatka (dvojrozmerné pole reprezentujúce aktuálnu križovatku). Ako som už spomínal algoritmus využíva metódu triedy Graph na vygenerovanie uzlov so všetkými možnými krokmi z daného uzla, táto metóda prechádza pole cars v danom uzle a pre každé urobí následovné: zistí či sa auto pohybuje vertikálne alebo horizontálne, potom zisťuje či sa môže pohnuť hore, dole, vpravo, vlavo, ak áno vytvorí kópiu aktuálneho uzla a na novom uzle vykoná možný pohyb nový uzol pridá do pola ktoré potom vráti ako vygenerované uzly. Tieto kroky sa opakujú pre všetký auta z poľa cars a pre všetký vzdialenosti.

Testovanie

Testovanie prebiehalo na rôznych testovacích vzorkách. Vzorky pozostávali z náhodne porozmiesťnovaných áut na križovatke, pričom auta nesmú byť na pozícií iného auta, inak program funguje nepresne. Testovaná bola taktiež možnosť preplnenej križovatky kedy sa nemohlo pohnuť ani jedno auto, pričom auto ktoré malo opustiť križovatku nebolo pri východe. V ďalšej testovacej vzorke bola preplnená križovatka no unikové auto bolo pri východe. Príklad testovacej vyorky:

Formátovanie" farba veľkosť x, y smer

! 2 3,2 h

g 2 5,1 v

o 3 1,1 h

z 3 2,1 v

m 3 6,4 h

x 2 6,2 h

k 3 1,4 h

i 2 2,6 v

h 2 4,6 v

Zhodnotenie riešenia

Riešenie problému pomocou BFS je uplné, optimálne, prisetorová a Časová zložitosť závisí od vetviaceho faktora (x) a od hĺbky najplytšieho cieľového uzla (n), to znamená že môžme zložitosť vyjadriť O(x^n). Hľadanie pomocou DFS algoritmu nie je optimálne a je úplné len pre konečný strom hľadania. Časová a priestorová zložitosť závisí od vetviaceho faktora (x) a od maximálnej hĺbky uzla(m), teda Časovú zložitosť môžeme vyjadriť O(b^m), priestorovú zložitosť vzjadríme O(bm). Môžeme teda tvrdiť že prehľadávanie do hĺbky (DFS) je rýchlejšie no menej optimálne ako prehľadávanie do šírky (BFS). BFS nájde cieľový uzol za dlhší čas ako DFS no počet krokov napríklad pri probléme križivatky je menší, čo dokázali aj výsledky mojej implementácie.

BFS vzorového riešenia prebehlo za priemerný čas 1,005s a cieľový uzol bol v hĺbke 7. Pričom pri DFS bol priemerný čas hľadania 0,483s, cieľový uzol bol v hĺbke 882.