Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Umelá inteligencia - Bláznivá križovatka

Róbert Dudáš

Študijný program: Informatika
Ročník: 2, Krúžok: streda 8:00
Predmet: Umelá inteligencia
Cvičiaci: Ing. Ivan Kapustík
Ak. rok: 2013/14

Riešený problém – Bláznivá križovatka

Úlohou je nájsť ťahy ako dostať červené auto do najpravejšej pozícií križovatky tak, aby auta nevyšli z križovatky a neprekrývali sa auta. Riešenie je navrhnute pre hlavolam 6x6. Súradnice áut zodpovedajú ľavému hornému rohu. Program vypíše prvé najedené riešenie. V prípade, že nenájdete riešenie, signalizujte neúspešné hľadanie. V diskusii potom analyzujte pozorované výsledky.

Stručný opis riešenia a podstatných častí

Stavový priestor

V mojej úlohe stavový priestor reprezentovaný ako mapka rozloženia áut. Táto mapka je informácia o tom, kde sa aktuálne nachádzame s daným autom v križovatke (stĺpec, riadok). Na generovanie nových stavov používam štyri operátory, ktoré zodpovedajú ťahom auta v križovatke t.j. posun vľavo, vpravo, hore a dole a daný počet krokov. Všetky operátory majú rovnakú váhu. Nový stav sa vygeneruje len v prípade ak, sa políčko kam sa chceme posunúť, nachádza v križovatke nestojí na ňom iné auto. Cieľovým stavom je taký stav keď červené auto sa dostane na pravý okraj križovatky.

Použitý algoritmus

Na prehľadávanie stavového priestoru som mal použiť algoritmus, *cyklicky sa prehlbujúce hľadanie*. Ako je v prednáške napísane poskytuje to najlepšie z hľadania do šírky a do hĺbky. Hľadanie je úplné a priepustné. M*etóda prehľadávania* vždy nájde riešenie, ak existuje.

Časová zložitosť algoritmu prehľadávania do hĺbky je:

$$(d+1)(1) + db + (d-1)b^2 + \ldots + (1)\, b^d = O(b^d)$$

Priestorová zložitosť:

$$O(bd) \text{ alebo } O(d)$$

Algoritmus je časovo náročný ale priestorovo nenáročný.

Konkrétne riešenie

Program som sa rozhodol implementovať v jazyku *Java.* Ide teda o objektovo orientovaný prístup a jednotlivé časti programu sú reprezentované ako triedy. Nachádzajú sa tu tieto hlavné triedy:

- ***Stav***– trieda, ktorá reprezentuje stav križovatky. Jej poľami sú riadok a stĺpec, kde sa aktuálne nachádzajú autá v križovatke*. Trieda obsahuje aj daný algoritmus a metódy na jeho fungovanie, generovanie nových uzlov a kontrolovanie uzlov.*
- ***Node*** – táto trieda reprezentuje prvky uzla. Nachádza sa tu záznam na rodiča, pole, slovo- postupnosť ťahov.
- ***Posun*** – je interface, ktorý rozvinu dané posuny t.j. pusonvlavo,posunvpravo,posunhore a posundole. Tieto triedy vrátia nový stav križovatky , len ak sa dá vytvoriť. Teda políčko kam sa chceme posunúť sa musí nachádzať v križovatke a nesmie byť obsadené.

Algoritmus teda dostane na vstup počiatočný stav aut (por. číslo, riadok, stĺpec, dĺžka auta, jeho orientácia). Prehľadávanie je implementované pomocou dátovej štruktúry zásobník (LIFO) v ktorej sa nachádzajú uzly.

Algoritmus začne tým, že vytvorí koreň z ktorého zakaždým vytvorí nové uzly , ktoré ukladá do zásobníka. V ďalšom cykle generuje z uzlov nových potomkov a kontroluje ich či nie sú finálne. V týchto dvoch cykloch koluje pokiaľ nenájde finálny stav a postupne zvyšuje hĺbku prehľadávania.

Zložitosť

- **Časová zložitosť**
  - uzol môže mať 4 operátorov
  - prehľadávanie do hĺbky $O(4^m)$
  - vygenerovanie nových možných stavov $O(1)$
  - Celková časová zložitosť**:** $O(4^m)$
- **Pamäťová zložitosť**
  - stav križovatky má konštantnú veľkosť $O(1)$
  - zásobník zo stavmi križovatky obsahuje $O(4m)$ stavov
  - Celková pamäťová zložitosť: $O(4m)$

Testovanie

Prebiehalo na rôznych testovacích úlohách, pri predpoklade dobrého formátu vstupu. Boli použité techniky debugovania a čiastočných výsledkov. Testovaný bol nielen samotný algoritmus, ale aj metódy (napr. metóda na generovanie nových uzlov), samostatne aj spoločne.

Tu sa nachádza ukážkový výstup programu:

```
1 2 3 5 1
2 3 6 1 1
3 2 1 5 1
4 3 1 1 0
5 3 4 4 0
6 2 5 2 1
7 3 4 6 0
8 2 4 1 0
auto 6 vlavo o 3
auto 7 dole o 3
auto 3 vpravo o 4
auto 4 hore o 1
auto 8 hore o 1
auto 2 vlavo o 2
auto 5 dole o 2
auto 1 vpravo o 3
```

Zhodnotenie

Keďže bol program implementovaný v jazyku *Java*, tak by sa dal veľmi jednoducho prerobiť do iného objektovo orientovaného jazyka. Použil som dátové štruktúry (*Stack*), ktorý je dostupný aj v iných programovacích jazykoch. Riešenie funguje na mapku 6x6, avšak s minimálnou úpravou sa dá prerobiť tak aby fungovalo na ľubovoľných rozmerov. Nakoľko riešenie nie je založené na rekurzii, je pomerne dobre škálovateľné. Použité štandardné reprezentácie údajov zaisťujú prijateľnú rýchlosť a odolnosť voči chybám.

Použitá literatúra

[1]Pavol Návrat a kol. – Umelá inteligencia
[2] http://stackoverflow.com/questions/10872398/creating-path-array-using-iddfs
[3] http://sujitpal.blogspot.sk/2006/05/java-data-structure-generic-tree.html