Umelá inteligencia

Zadanie č. 2 - Bláznivá križovatka

Dávid Šinko, FIIT STU, 3. ročník, ak. rok 2017/2018

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 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.

Implementácia 1

STAV

Stav predstavuje aktuálne rozloženie vozidiel. Potrebujeme si pamätať farbu každého vozidla, jeho veľkosť, pozíciu vozidla a či sa môže posúvať vertikálne alebo horizontálne.

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.

UZOL

Stav predstavuje nejaký bod v stavovom priestore. My však od algoritmov požadujeme, aby nám ukázali cestu. Preto musíme zo stavového priestoru vytvoriť graf. Našťastie to nie je zložitá úloha. Stavy jednoducho nahradíme uzlami.

Čo obsahuje typický uzol?
Musí minimálne obsahovať

- **STAV** (to, čo uzol reprezentuje) a
- **ODKAZ NA PREDCHODCU** (pre nás zaujímavá hrana grafu, reprezentovaná čo najefektívnejšie).

Okrem toho môže obsahovať ďalšie informácie, ako

- **POSLEDNE POUŽITÝ OPERÁTOR**
- **PREDCHÁDZAJÚCE OPERÁTORY**
- **HĹBKA UZLA**
- **CENA PREJDENEJ CESTY**
- **ODHAD CENY CESTY DO CIEĽA**
- Iné vhodné informácie o uzle

Uzol by však nemal obsahovať údaje, ktoré sú nadbytočné a príslušný algoritmus ich nepotrebuje. Pri zložitých úlohách sa generuje veľké množstvo uzlov a každý zbytočný bajt v uzle dokáže spotrebovať množstvo pamäti a znížiť rozsah prehľadávania algoritmu. Nedostatok informácií môže zase extrémne zvýšiť časové nároky algoritmu. ***Použité údaje zdôvodnite***.

ALGORITMUS

Každé zadanie používa svoj algoritmus, ale algoritmy majú mnohé spoločné črty. Každý z nich potrebuje udržiavať informácie o uzloch, ktoré už kompletne spracoval a o uzloch, ktoré už vygeneroval, ale zatiaľ sa nedostali na spracovanie. Algoritmy majú tendenciu generovať množstvo stavov, ktoré už boli raz vygenerované. S týmto problémom je tiež potrebné sa vhodne vysporiadať, zvlášť u algoritmov, kde rovnaký stav neznamená rovnako dobrý uzol.

Činnosť nasledujúcich algoritmov sa dá z implementačného hľadiska opísať nasledujúcimi všeobecnými krokmi:

1. Vytvor počiatočný uzol a umiestni medzi vytvorené a zatiaľ nespracované uzly
2. Ak neexistuje žiadny vytvorený a zatiaľ nespracovaný uzol, skonči s neúspechom - riešenie neexistuje
3. Vyber najvhodnejší uzol z vytvorených a zatiaľ nespracovaných, označ ho aktuálny
4. Ak tento uzol predstavuje cieľový stav, skonči s úspechom - vypíš riešenie
5. Vytvor nasledovníkov aktuálneho uzla a zaraď ho medzi spracované uzly
6. Vytrieď nasledovníkov a ulož ich medzi vytvorené a zatiaľ nespracované
7. Choď na krok 2.

Opis riešenia

K vypracovaniu zadanie som použil programovací jazyk C#. Zadefinoval som si entity auto a uzol a trieda program obsahuje main, ktorý spúšťa celý program. Každá trieda obsahuje špecifické atribúty a metódy, ktoré hlavná inštancia programu volá na vyriešenie zadania.

Auto je definované svojou farbou, dĺžkou, polohou a orientáciou na mapke rozmerov 6x6. Pri každom posune akéhokoľvek vozidla kontrolujem, či sa nenachádza mimo mapky a ak k takémuto vystúpeniu z mapky dôjde, vozidlo sa nemôže daným operátorom posunúť. V triede používam metódy **doprava()**, **dolava()**, **dole()** a **hore()**, ktoré posúvajú vozidlá. Každý posun a zmena polohy reprezentuje nový stav, nové rozloženie vozidiel.

Trieda uzol reprezentuje stavy. Keďže stav je rozloženie vozidiel na mapke, máme rôzne rozloženia, teda stavy a v programe s tým pracujem ako s uzlami, ktoré prechádzam v grafovej štruktúre. Uzol obsahuje zoznam vozidiel, informáciu o tom, či už bol spracovaný, vzdialenosť, ktorá udáva vzdialenosť červeného vozidla od pravého okraja a tiež informáciu o predchodcovi. Táto trieda využíva metódy **vypocitajVzdialenost()**, ktorá ako už názov hovorí, počíta vzdialenosť červeného vozidla od pravého okraja. Je tu použitá mini optimalizácia, že keď tomuto červenému vozidlu stojí v ceste iné vozidlo, vzdialenosť je väčšia ako keď mu nestojí v ceste nič. Je to preto tak, aby sa snažil uvoľňovať cestu von z križovatky. Metóda **uzBolo()** rozhoduje o tom, či taký stav už nastal niekedy predtým v danej vetve stromového grafu alebo nie. Veľmi dôležitá metóda **dalsie()** generuje ďalšie uzly z práve prehľadávaného uzla. Každý posun vozidla znamená iný stav, čiže iný uzol, ktorý je rozvitý a pridaný do radu alebo zásobníku na spracovanie. Tiež dôležitá metóda **jeKolizia()** určuje pri každom posune vozidla, či sa vozidlo môže posunúť daným smerom, či mu v posune nestojí nejaké iné vozidlo. Toto usporiadanie si ukladám v Hashset-e prostredníctvom metódy **obsadene()**. Ďalšie dve metódy **ToString()** a **print()** slúžia viac-menej na vypisovanie usporiadania vozidiel na mapke. ToString() vytvorí slovnú reprezentáciu uzla, teda rozloženia vozidiel na mapke podľa ich farieb a print() mi vypíše toto usporiadanie na konzolu.

Hlavná trieda Program obsahuje Main, v ktorom si vytváram počiatočný uzol. To robím volaním metódy **vytvorPociatocnyStav()**. V tejto metóde načítavam z konzoly potrebné údaje pre vytvorenie stavu ako farbu vozidla, jeho veľkosť, súradnice na mapke a orientáciu. Začínam hlavným, červeným vozidlom a každé si pridávam do zoznamu vozidiel. Keď mám vytvorený počiatočný stav, volám metódu **doSirky()** alebo **doHlbky()** podľa toho, aké prehľadávanie chcem použiť. Treba to vopred v kóde zvoliť a nastaviť. Pri prehľadávaní do šírky používam rad, do ktorého si ukladám novovytvorené uzly a hlavným rozdielom oproti druhému je to, že používam funkcionalitu radu (fifo). To znamená, že tak ako uzly ukladám do radu, v takom poradí ich aj ďalej spracúvam. Ošetrujem tu spracovanie rovnakých stavov, aké už raz boli spracované. Ak nájdem riešenie, teda ako cieľový stav beriem to, keď sa červené vozidlo nachádza úplne vpravo. Ak zatiaľ nemám riešenie, program pokračuje, rozvíja uzly a prehľadáva postupne každý z nich, až kým riešenie nenájde. DoHlbky() používa podobný princíp, má veľa prvkov podobných,

až na to, že namiesto radu používam zásobník. Ten mi umožňuje ukladať doňho stavy a vyberať vždy zvrchu (lifo). Týmto spôsobom prehľadávam samotné vetvy až k listom. V obidvoch prípadoch prehľadávania si nakoniec vypíšem riešenie, buď že riešenie neexistuje alebo ak áno, tak metódou **vypisKonecneRiesenie()**, že koľko stavov rozvilo, spracovalo, koľko krokov potrebovalo, aby sa červené vozidlo dostalo tam kde má a tieto kroky vypíše do konzoly.

Pri porovnaní týchto dvoch spôsobov mi prináša lepšie výsledky spôsob prehľadávania do šírky. Síce potrebuje spracovať o niečo viac uzlov ako prehľadávanie do hĺbky, ale potrebuje podstatne menej operátorov, ktoré dostanú červené vozidlo do cieľovej pozície. Je to kvôli tomu, že keď prehľadávam do hĺbky, vozidlá sa posúvajú rôzne a neprináša to najoptimálnejšie výsledky. Riešenie som testoval na viacerých rôznych vstupoch, ktoré sú aj zakomentované v kóde.