Bláznivá križovatka

Zadanie UI Filip Agh 2018

21. marca 2018

# 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. Hlavolam je vyriešený, keď je červené auto (v smere jeho jazdy) na okraji križovatky a môže z nej teda dostať von.

# Stručný opis riešenia a jeho podstatných častí

Riešenie tohto problému som sa rozhodol naprogramovať v objektovom jazyku Java.

Vytvoril som si hlavne triedy

* AI
* Uzol
* Vozidlo
* Príkaz

### AI

Hlavná logistická trieda, rieši čítanie zo vstupného súboru, vygenerovanie začiatočného uzla

Obsahuje nasledujúce metódy

Boolean ries(int sposob) – metóda ktorá spusti celý algoritmus prehľadávania vstupný parameter spôsob (0 – do šírky //// 1 – do dĺžky)

Ak sa nájde uzol kde je správny výsledok zavolá sa metóda daného uzla na výpis cesty a ukonči sa hľadanie

void pridajvozidlo(Vozidlo voz) - metóda na pridanie vozidla do Listu (použite pri načítavaní vstupu)

### Uzol

public boolean hladaj(List<Uzol> imp\_uzle,int sposob) – metóda hľadaj spracováva daný uzol čiže z listu vozidiel vytvorí maticu a na nej skúša ci hlavne vozidlo sa vie dostáť z parkoviska ak nie tak postupne urobí s každým vozidlom

* Posunie vozidlo
* Zapíše novy stav do zoznamu vozidiel (vozpark\_exp)
* Zahashuje ho a skontroluje ci taký stav už nebol
* Ak nie tak vytvorí novy príkaz následne uzol
* Potom podľa potreby ho zaradí na správne miesto v liste zoznamu

Keď uzol správy všetky možnosti zmaže sám seba z listu uzlov, vypíše výslednú maticu

void vypiscestu()

metóda sa vola z AI iba ak z daného uzlu sa vie hlavne auto sa dostáť von, metóda vypíše list príkazov daného uzla

### Vozidlo/Príkaz

Obsahujú atribúty potrebne pre záznam údajov

# Použitý algoritmus

Zo zadania vyplýva že mam naprogramovať prehľadávanie

Do šírky

Mam list všetkých uzlov a vždy berme 1. uzol v rade ten rozviniem a nove uzly dám na koniec radu

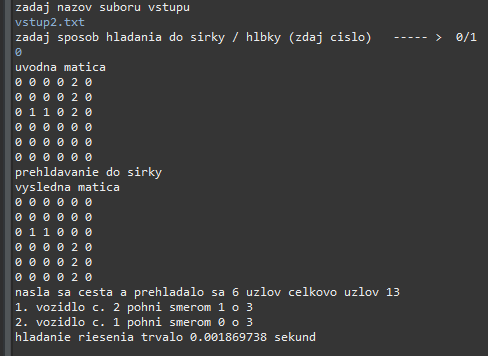
Do hĺbky

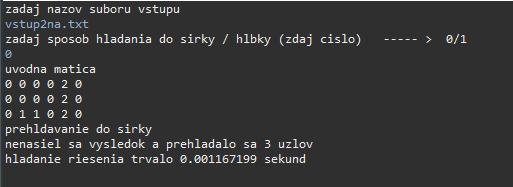
Mam list všetkých uzlov a vždy berme 1. uzol v rade ten rozvinom a nove uzly dám na začiatok radu

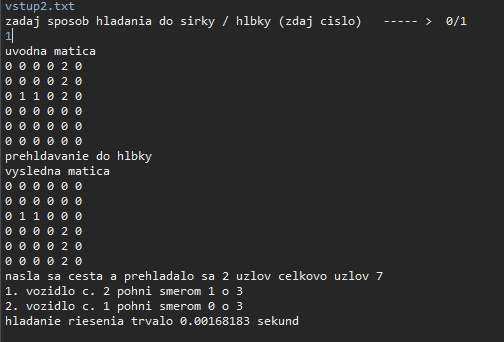
# Spôsob testovania

Svoj program som testoval na základnom teste

* 2 vozidla







Ďalšie testy sa dajú spraviť (nepridal som ich sem ale testoval som ich)

Všeobecný vstup musí obsahovať

Int int int // počet vozidiel, rozmer x, rozmer y

Int int int int int // id vozidla (od 1 do n), dĺžka vozidla , x sur, ysur, orientácia (0 – horizontálne 1 – vertikálne)

Ukážka vstupu

2 6 6

1 2 1 2 0

2 3 4 0 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9 6 6

1 2 3 2 0

2 2 3 0 0

3 3 2 0 1

4 3 5 0 1

5 3 3 3 1

6 2 4 3 0

7 2 1 4 0

8 2 4 5 0

9 2 0 4 1

Výpis príkazu smer môže byt (0 – vľavo 1- dole 2 – vpravo 3 - hore)

# Zhodnotenie riešenia

Výhody

Môj algoritmus naraz vie posunúť vozidlo aj o ‘n‘ krokov

Hracia doska môže byt rôznej veľkosti

Používam hashtabulku na uchovávanie stavov ktoré som už prešiel aby som nerobil duplicity

Dĺžka vozidla sa da nastaviť

Nevýhody

Prehľadávanie do hĺbky trvá dlho koly prepisu veľkému zoznamu príkazov

Program je závislí na knižniciach ktoré ponuka Java

# Porovnanie vlastností použitých metód pre rôznu dĺžku riešenia

Ak sa nejedna o ozaj triviálny problém tak prehľadávanie do šírky je rýchlejšie výraznejšie ale spracuje viacej uzlov a zábere viac pamäte