SLOVENSKÁ TECHNICKÁ UNIVERZITA
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Umelá inteligencia

Zadanie 2 – Prehľadávanie stavového priestoru

Bláznivá križovatka – cyklicky sa prehlbujúce hľadanie

Lucia Murzová
Streda 13:00
25.10.2021

Lucia Murzová
Streda 13:00

Obsah

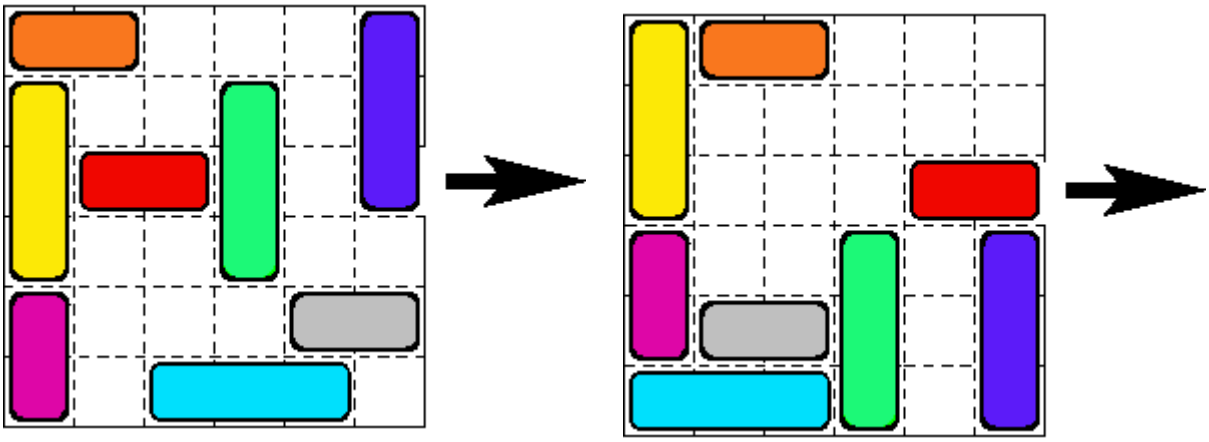
Lucia Murzová
Streda 13:00

1. Riešený problém

V tomto zadaní som sa venovala riešeniu hlavolamu Bláznivá križovatka (Unblock me). Hlavolam je reprezentovaný mriežkou, ktorá má rozmery 6 krát 6 políčok a obsahuje niekoľko vozidiel rozložených na mriežke tak, aby sa neprekrývali. Všetky vozidlá majú šírku 1 políčko a môžu byť dlhé 2 alebo 3 políčka. Vozidlá sa môžu pohybovať len v prípade, že ich nič neblokuje – žiadne auto ani koniec plochy, nikdy však nie do strany. V jednom kroku sa pohybuje vždy len jedno vozidlo o ľubovoľný počet dostupných voľných miest.

Hlavolam je vyriešený, keď je červené auto na pravom okraji križovatky a môže sa z nej dostať von. Červené auto je z tohto dôvodu vždy otočené horizontálne a smeruje doprava. Je potrebné nájsť postupnosť posunov vozidiel tak, aby sa červené auto dostalo von z križovatky alebo vypísať, že úloha nemá riešenie.

Počiatočná pozícia Cieľová pozícia



Obr. 1 Počiatočná a cieľová pozícia hlavolamu Bláznivá križovatka.

Postupnosť posunov vozidiel z počiatočnej do cieľovej pozície z obr.1 je:

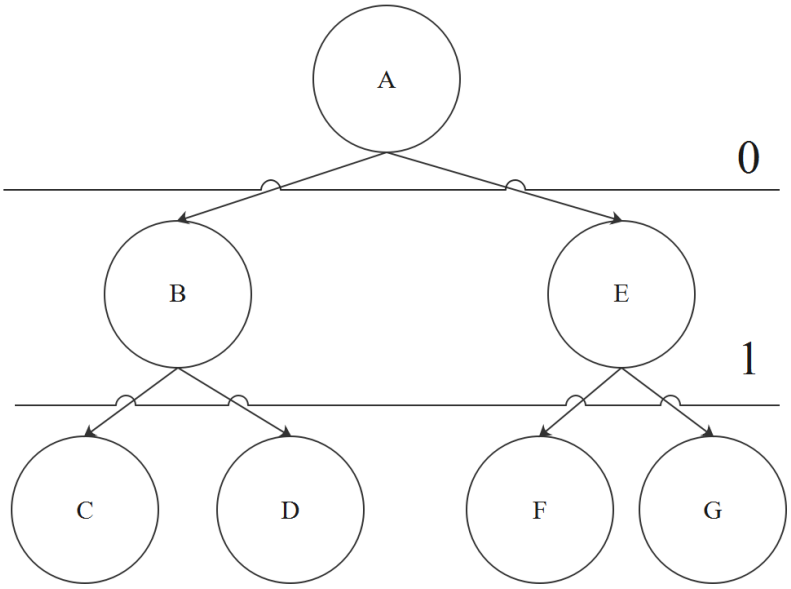
VPRAVO(oranzove, 1), HORE(zlte, 1), HORE(fialove, 1), VLAVO(sive, 3), VLAVO(svetlomodre, 2), DOLE(tmavomodre, 3), DOLE(zelene, 2), VPRAVO(cervene, 3)

Lucia Murzová
Streda 13:00

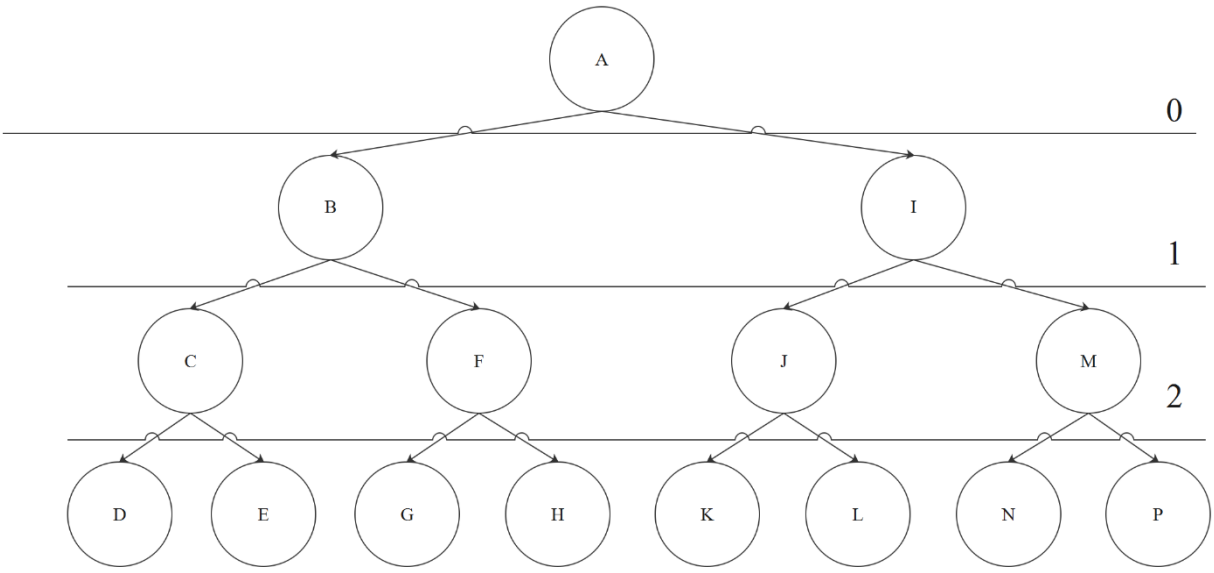
2. Cyklicky sa prehlbujúce hľadanie

Algoritmus cyklicky sa prehlbujúceho hľadania sa správa ako prehľadávanie do hĺbky s tým, že má vždy stanovenú maximálnu hĺbku, do ktorej môže v aktuálnom kroku hľadať. Zvyčajne začína s maximálnou hĺbkou 1 a po každom prehľadaní všetkých uzlov do tejto hĺbky sa postupne navyšuje. Tento algoritmus taktiež vždy pracuje len s malým množstvom uzlov, konkrétne s uzlami v aktuálnej vetve, ktoré pri návrate zabúda.

Z hľadiska výpočtovej náročnosti je tento algoritmus spojením prehľadávania do šírky a prehľadávania do hĺbky. Pri vyváženom strome je časová zložitosť tohto algoritmu rovnaká ako pri prehľadávaní do šírky – O(b^d), kde b je faktor vetvenia a d je hĺbka cieľového stavu. Priestorová náročnosť je zas veľmi podobná prehľadávaniu do hĺbky – O(bd).



0-1 Ukážka prehľadávania keď k=2



0-2 Ukážka prehľadávania keď k=3

Lucia Murzová
Streda 13:00

3. Opis riešenia

Vstup

Program začína načítaným vstupu z definovaného súboru, v ktorom sú zapísané autá na tak, ako sa na začiatku nachádzajú na hracej ploche. Každé auto je zapísané na novom riadku vo formáte: *farba, dĺžka, začiatočná pozícia riadku, začiatočná pozícia stĺpca, smer ktorým sa pohybuje – h / v.* Riešenie je zamerané vždy na červené auto, ktoré musí byť umiestnené horizontále.

Po načítaní vstupu zo súboru program prechádza všetky zadané autá a vytvára z nich hraciu plochu s rozmermi 6x6 (prípadne inak podľa definovania v const.py, označované 0-5), na ktorej sú autá reprezentované začiatočným písmenom ich farby (pri dvoch farbách s rovnakým písmenom je preto dobre pri jednom použiť veľké písmeno). Pri vytváraní plochy kontroluje, či sa autá neprekrývajú a v prípade ak áno, program predčasne skončí.

Stav

Každý stav obsahuje:

- Pole s uloženými pozíciami jednotlivých áut – [[ ]]
- Maticu 6x6, kde je toto rozloženie zapísané – [[ ]]
- Odkaz na svojho predchodcu – Uzol
- Zoznam možných nasledovníkov – [ ]
- Hĺbku uzla – int
- Farbu auta, ktoré sa posunulo – [ ]
- Smer, ktorým sa auto posunulo – int (1 – hore, 2 – vpravo, 3 – dole, 4 – vľavo)
- Dĺžku, o koľko sa auto posunulo – int

Konštanty

Program pracuje s niekoľkými konštantami, pre ktoré využíva súbor const.py. V tomto súbore sú zadefinované rozmery hracej plochy a taktiež poradie, v ktorom sa zo súboru načítavajú informácie o jednotlivých autách (meno = 0, dĺžka auta = 1, riadok = 2, stĺpec = 3, smer pohybu = 4). Tiež sú tu v poli zadefinované jednotlivé pohyby, kde pohyb hore = 1, vpravo = 2, dole = 3 a vľavo = 4. Toto označenie je využité pre prehľadnejší výpis jednotlivých krokov riešenia.

Čerpanie údajov z tohto súboru a používanie názvov týchto konštánt napomáha pri orientácii v samotnom kóde, kedy je prehľadnejšie reprezentovať pozíciu auta ako auto[poloha_riadok], auto[poloha_stlpec] namiesto auto[3], auto[4].

Generovanie potomkov

Pri generovaní potomkov vo funkcii ***generuj_stavy(uzol: Uzol)*** sa program pokúša vygenerovať potomkov pre pohyb každým autom o všetky možné vzdialenosti jeho posunu. Pri generovaní kontroluje, či je nasledujúce políčko pre posun voľné a ak áno, vytvorí sa nový stav – kópia aktuálneho uzla, nad ktorou sa následne zavolá *vykonaj_posun_smer(auto, uzol: Uzol)*, kde sa upraví stav nového potomka a uložia sa informácie o pohybe auta.

Algoritmus

Po načítaní vstupu je inicializované k = 1 a program začína prehľadávanie funkciou ***hladaj_ciel(zaciatocy_stav: Uzol, k: int).*** V tejto funkcii sa overí, či je daný uzol cieľový stav, prípadne či je v maximálnej hĺbke. Pokiaľ nejde o cieľový stav ani o maximálnu hĺbku, vygenerujú sa stavu potomkovia. Zoznam potomkov sa následne potupne prechádza a porovnáva so všetkými

predchodcami. Pokiaľ tento uzol nie je zhodný so žiadnym z predchodcov, rekurzívne sa nad ním zavolá funkcia ***hladaj_ciel(potomok: Uzol, k: int).***

Testovanie a zhodnotenie riešenia

Testovanie bolo vykonané na testoch s rôznym počtom áut a rôznym počtom potrených krokov do cieľa. Taktiež som otestovala rozdiel pri porovnávaní rovnakého stavu so všetkými predchodcami oproti porovnaniu iba s jedným predchodcom. Podstatný rozdiel spôsobuje aj usporiadanie samotných áut vo vstupných súboroch – pokiaľ sú usporiadané tak, ako je potrebné s nimi na ploche pohybovať, riešenie je značne rýchlejšie, keďže rozvíja menej uzlov.

1. Ukážkové riešenie – potrebných 8 krokov, pre program sa ukázalo ako najťažšie (súbor stav.txt0

- Pri vstupe so všetkými autami bol priemerný čas prehľadávania pri porovnaní s jedným predchodcom bol okolo 40 sekúnd, zatiaľ čo pri porovnávaní so všetkými predchodcami bol 50, k je v oboch prípadoch rovné 9

- stačí však odobrať ktorékoľvek auto z plochy a riešenie je v priemere do dvoch sekúnd s k = 8

- pri prehľadávaní po jednotlivých vrstvách boli počty uzlov nasledovné:

- K = 1 – uzly: 7
- K = 2 – uzly: 39
- K = 3 – uzly: 211
- K = 4 – uzly: 1 125
- K = 5 – uzly: 6 023
- K = 6 – uzly: 32 445
- K = 7 – uzly: 175 777
- K = 8 – uzly: 955 299

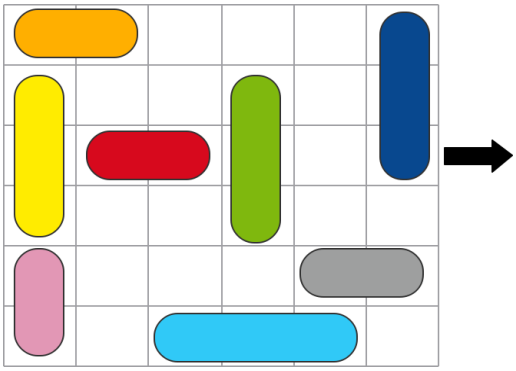
- ako najväčší problém sa preto ukazuje neoptimálna rekurzia, ktorá má v nižších vrstvách za následok uloženie veľkého počtu stavov a plytvanie pamäťou, čo značne spomaľuje čas výpočtu

- Po usporiadaní áut do poradia, v ktorom s nimi treba posúvať, výsledky sa zlepšili - 14 sekúnd

Postupnosť pôvodná:
cervene 2 2 1 h
zelene 3 1 3 v
oranzove 2 0 0 h
Zlte 3 1 0 v
ruzove 2 4 0 v
svetlo_modre 3 5 2 h
tmavo_modre 3 0 5 v
Sive 2 4 4 h

Postupnosť nová :
cervene 2 2 1 h
oranzove 2 0 0 h
Zlte 3 1 0 v
ruzove 2 4 0 v
zelene 3 1 3 v
tmavo_modre 3 0 5 v
Sive 2 4 4 h
svetlo_modre 3 5 2 h

Počty uzlov pri novej postupnosti:
- k = 1: 7 uzlov
- k = 2: 40 uzlov
- k = 3: 222 uzlov
- k = 4: 1 196 uzlov
- k = 5: 6 426 uzlov
- k = 6: 34 719 uzlov
- k = 7: 189 219 uzlov

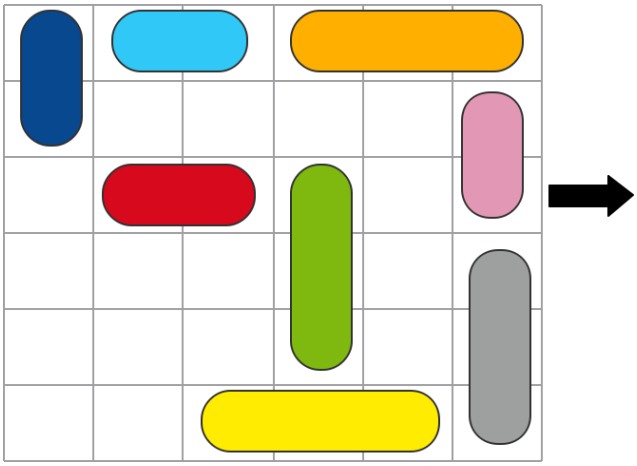


Lucia Murzová
Streda 13:00

2. Potrebných 7 krokov (stav2.txt) -

- riešenie priemerne 0.5 sekundy, k = 9
cervene 2 2 1 h
svetlo_modre 2 0 1 h
tmavo_modre 2 0 0 v
zelene 3 2 3 v
oranzove 3 0 3 h
ruzove 2 1 5 v
sive 3 3 5 v
zlte 3 5 2 h

- riešenie 0.15 sekundy, K = 8
cervene 2 2 1 h
tmavo_modre 2 0 0 v
svetlo_modre 2 0 1 h
oranzove 3 0 3 h
zelene 3 2 3 v
ruzove 2 1 5 v
sive 3 3 5 v
zlte 3 5 2 h



V ukážkovej úlohe je možné vidieť, že tento problém nie je ani tak závislí od veľkosti samotnej plochy alebo počtu áut, ako od počtu potrebných ťahov a hlavne usporiadania áut vo vstupnom súbore. Pri „lepšom“ usporiadaní – podľa potreby pohybu sa síce na jednotlivých vrstvách rozvíja viac uzlov, no samotné riešenie je nájdené rýchlejšie.

Nevýhodou tohto riešenia je použitá rekurzia, ktoré v nižších vrstvách uchováva viacero odkazov a teda plytvá pamäťou, čo program značne spomaľuje. Možnou optimalizáciou by bolo využitie zoznamu všetkých už spracovaných uzlov, prípadne ich uloženie v hash tabuľke, čo by úplne zamedzilo výskytu duplicitných uzlov a tak aj urýchlilo samotné výpočty.