Slovenská Technická Univerzita v Bratislave
Fakulta Informatiky a Informačných Technológií

Počítačové a komunikačné siete

Zadanie 2 – Prehľadávanie stavového priestoru

Marek Adamovič

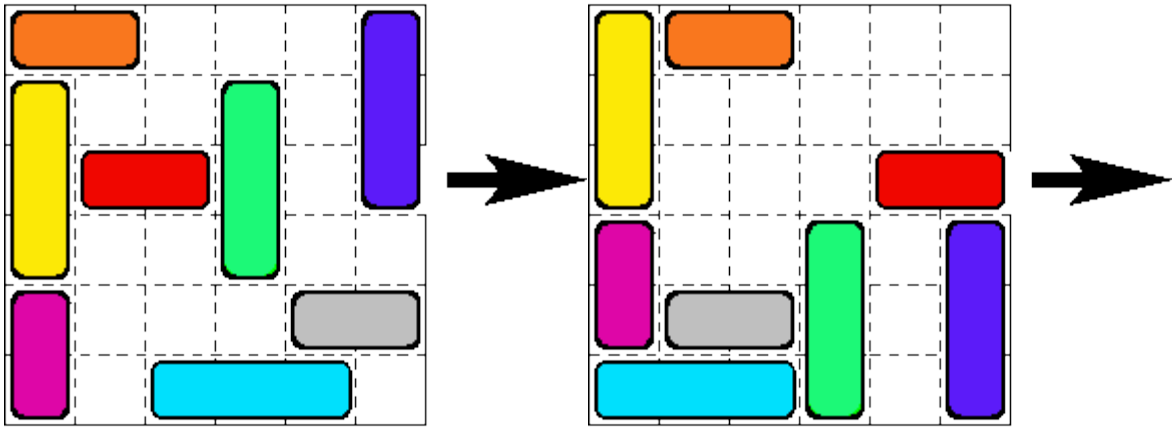
Cvičenie: Utorok 16:00 – 17:40
Ing. Boris Slíž

2020/2021

Zadanie úlohy

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.

Opis riešenia

K riešeniu sa dostanem pomocou prehľadávania stavového priestoru, konkrétne algoritmami prehľadávania do hĺbky (DFS) a prehľadávania do šírky (BFS). Začínam v počiatočnom stave, z ktorého pri oboch algoritmoch vytváram nové stavy a zaradím ich medzi nespracované, do spájaného zoznamu. Potom z jeho začiatku/konca (záleží na algoritme, pri hĺbkovom vyberám posledný pridaný, pri šírkovom zas ten najstarší) vyberiem stav, ktorý idem riešiť. Počas riešenia znovu vytváram nové stavy a kontrolujem, či náhodou nie sú duplicitné s už vyriešenými (vyriešené ukladám do hash tabuľky). Pri každom stave na začiatku pozriem, či to nie je výsledný stav. Ak takýto stav nájdem, ukončujem program s vypísanou

postupnosťou operátorov. Ak ho nenájdem, končím s výpisom, že takáto križovatka nemá riešenie.

Reprezentácia údajov

Stav = reprezentujem ho v štruktúre, obsahuje informácie o všetkých autách a ich pozíciách

```
typedef struct state{
    CAR *red_car;
    unsigned short depth;
    struct state *parrent;
    struct state *next;
    struct state *previous;
    unsigned char last_used_car_id;
    unsigned char last_used_operator;
}STATE;
```

Auto = reprezentujem ho v štruktúre, obsahuje jedinečné ID (0 je vždy určená červenému autu), čo nahrádza farbu, súradnice pozície, veľkosť (či sa jedná o nákladiak) a smerovanie

```
typedef struct car{
    unsigned char id;    //id nam nahradza farbu, 0 je cervene auto
    unsigned char size;
    unsigned char x;
    unsigned char y;
    char direction;
    struct car *next;
}CAR;
```

Vyriešené stavy sú uložené v hash tabuľke, čakajúce sú zreťazené v spájanom zozname. Autá sa nachádzajú v štruktúrach stavov.

Využité algoritmy a dátové štruktúry

Algoritmy: Prehľadávanie do šírky, Prehľadávanie do hĺbky, Hash funkcia

Dátové štruktúry: Hash tabuľka, Jednosmerný/Obojsmerný spájaný zoznam

Testovanie

Testoval som pomocou vlastných križovatiek, obsahujúcich rôzne možné prípady, ktoré mohli nastať a meral som čas, za ktorý sú oba algoritmy schopné tieto prípady vyriešiť. Medzi rôzne prípady patrila križovatka, ktorej prvý stav bol hneď koncový,

križovatky bez riešenia, zložité križovatky a preplnená križovatka. Pre rýchlu kontrolu som využíval funkciu výpisu aktuálneho stavu, ktorá mi „vykreslila“ stav križovatky do konzoly a taktiež hĺbku daného stavu (teda koľko krokov bolo treba, aby som sa k nej dopracoval zo štartovacieho stavu).

```
Ktory subor chces spustit? Zadaj cislo v rozmedzi 1 - 9
5
Ktory algoritmus chces pouzit?
1 => hladanie do hlbky
2 => hladanie do sirky
1

0 0 1 * 5 *
7 * 1 * 5 *
7 * 1 3 6 6
2 2 2 3 * *
* 8 * 4 4 *
* 8 * * * *

Time: 7.00

* * 1 * 0 0
* * 1 3 * *
6 6 1 3 * *
7 8 2 2 2 *
7 8 4 4 5 *
* * * * 5 *
Depth = 155

koniec
```

vykreslený štartovací a koncový stav s hĺbkou

Zhodnotenie riešenia

Ak existuje riešenie, oba algoritmy nám ho nájdu. Sú tu však 2 hlavné rozdiely, čo sa týka výstupu, prehľadávanie do hĺbky nájde hocijaké riešenie (nemusí byť ideálne, najkratšie), avšak v oveľa lepšom čase ako prehľadávanie do šírky (ktoré nájde vždy perfektné riešenie, keďže prechádza stavový priestor po vrstvách). Takže ak nám ide o čas, použijeme hľadanie do hĺbky, ak o perfektné riešenie, tak použijeme prehľadávanie do šírky. Veľkú časovú úsporu nám zabezpečila hash tabuľka, vďaka ktorej nemusíme kontrolovať duplicitu so všetkými doterajšími stavmi, ale máme viac menej priamy prístup k stavom, ktoré by mohli byť duplicitnými. Možné rozšírenia sú akceptovanie väčších áut, ako sú o veľkosti 3, prípadne

zväčšenie križovatky. Zrýchliť riešenie by sa dalo zlepšením hash funkcie pre lepší rozptyl do tabuľky a menej výpočtov. Taktiež by sa dalo rozšíriť rozhranie a zabezpečiť možnosť spúšťania viacerých súborov za sebou. Moja implementácia nie je závislá na konkrétnom implementačnom prostredí.

Porovnanie algoritmov

Prehľadávanie do šírky je časovo náročnejšie, avšak vždy nájde najlepší možný výsledok. Prehľadávanie do hĺbky je rýchlejšie, ale nevieme zaručiť, že výsledok, čo nájde, bude najlepší (je veľmi pravdepodobné, že od najlepšieho bude mať naozaj ďaleko).

Časová zložitosť prehľadávania do šírky: exponenciálna, postupne narastajúca v závislosti od hĺbky prvého dobrého riešenia

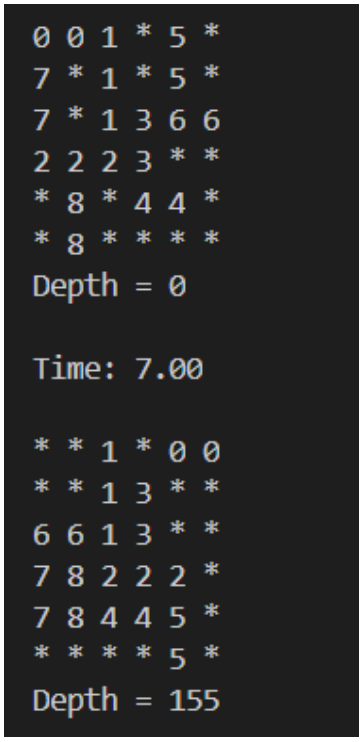
Pamäťová zložitosť prehľadávania do šírky: exponenciálna, keďže na každej ďalšej vrstve si musíme pamätať xkrát viacej stavov

```
0 0 1 * 5 *
7 * 1 * 5 *
7 * 1 3 6 6
2 2 2 3 * *
* 8 * 4 4 *
* 8 * * * *
Depth = 0

Time: 253.00

* * * * 0 0
7 * * * 5 *
7 * 6 6 5 *
* * 1 2 2 2
* 8 1 3 4 4
* 8 1 3 * *
Depth = 16
```

Na obrázku vidíme výsledok prehľadávania do šírky, čas 253 (používal som funkciu clock, čas teda nie je v sekundách) a nájdené perfektné riešenie na 16. vrstve (teda 16x sme posunuli s autom).

Časová zložitosť prehľadávania do hĺbky: lineárna, zväčšujúca sa v závislosti na vrstve, v ktorej „natrafíme“ na prvé správne riešenie

Pamäťová zložitosť prehľadávania do šírky: lineárna, keďže pri každej vrstve si zapamätáme len hŕstku novovytvorených stavov



Na obrázku vidíme výsledok prehľadávania do hĺbky, čas 7 a nájdené neperfektné riešenie na 155. vrstve (teda 155x sme posunuli s autom).

S komplikovanejšou križovatkou vidíme stále sa zväčšujúci rozdiel časovej zložitosti týchto dvoch riešení. Predpokladám, že pri obrovských komplikovaných križovatkách bude prehľadávanie do šírky skoro nepoužiteľné, práve kvôli časovej zložitosti.

Implementačné prostredie

Program som implementoval v jazyku C.