Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 2, 842 16 Bratislava 4

**Stanislav Jakúbek**

**Zadanie č.2**

Študijný program: Informatika

Ročník: 2.

Predmet: Umelá inteligencia

Vedúci projektu: Mgr. Irina Malkin Ondik, PhD.

Ak. rok: 2017/2018

# Zadanie

Úlohou je nájsť riešenie hlavolamu **Bláznivá križovatka** pomocou algoritmu cyklicky sa prehlbujúceho hľadania. 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.

Keď chceme túto úlohu riešiť algoritmami prehľadávania stavového priestoru, musíme si konkretizovať niektoré pojmy. Uvádzame príklad reprezentácie stavu, opis operátorov a cieľového stavu.

## Stav

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

((cervene 2 3 2 h)(oranzove 2 1 1 h)(zlte 3 2 1 v)(fialove 2 5 1 v)

(zelene 3 2 4 v)(svetlomodre 3 6 3 h)(sive 2 5 5 h)(tmavomodre 3 1 6 v))

V tomto zápise je prvé vozidlo červené auto, ktoré sa má dostať ku bráne. Farba vozidla sa môže vynechať, ak ho konkrétna implementácia nevyžaduje. Veľkosť je vždy 2 alebo 3. Súradnice zodpovedajú ľavému hornému rohu automobilu a v tomto príklade sú súradnice počítané od ľavého horného rohu križovatky a začínajú od jednotky, prvá určuje riadok. Smer možného pohybu automobilu určuje **h** (horizontálny) pre pohyb vľavo a vpravo a **v** (vertikálny) pre pohyb hore a dole.

**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 posnúť, 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.**

Príklad použitia operátora VPRAVO, pre oranzove auto a posun o 1:

Vstupný stav:

((cervene 2 3 2 h)(oranzove 2 1 1 h)(zlte 3 2 1 v)(fialove 2 5 1 v)

(zelene 3 2 4 v)(svetlomodre 3 6 3 h)(sive 2 5 5 h)(tmavomodre 3 1 6 v))

Výstupný stav:

((cervene 2 3 2 h)(oranzove 2 1 2 h)(zlte 3 2 1 v)(fialove 2 5 1 v)

(zelene 3 2 4 v)(svetlomodre 3 6 3 h)(sive 2 5 5 h)(tmavomodre 3 1 6 v))

## 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.

Uvedené kroky sú len všeobecné a pre jednotlivé algoritmy ich treba ešte vždy rôzne upravovať a optimalizovať.

# Opis riešeného algoritmu

Na vyriešenie tejto úlohy bol použitý algoritmus cyklicky sa prehlbujúceho hľadania. Tento algoritmus postupne prehľadáva údaje po úrovniach, teda najskôr prehľadá všetky údaje do hĺbky 1, potom všetky údaje do hĺbky 2, potom 3 atď.

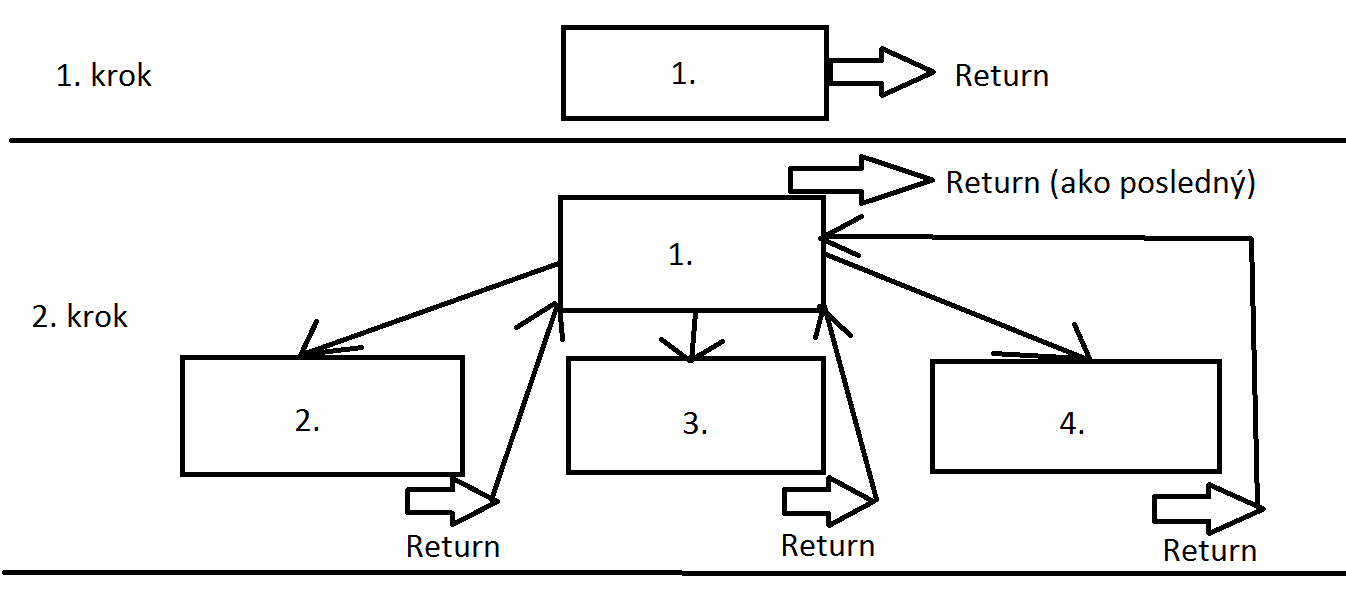
# Implementácia

Program bol zimplementovaný v programovacom jazyku C++. Algoritmus funguje rekurzívne – nájde všetky možné platné pohyby vozidlami a opäť sa spustí s údajmi po posune vozidla.

## Funkcia IDS

UZOL \*IDS(UZOL \*uzol, unsigned char maxHlbka, unsigned char pocetVozidiel, unsigned char cervene)

Funkcia IDS je hlavnou funkciou programu. Funkcia dostane ako vstupné parametre stav križovatky *uzol*, maximálnu hĺbku do ktorej treba prehľadávať *maxHlbka*, počet vozidiel v križovatke *pocetVozidiel* a identifikačné číslo červeného vozidla *cervene*. Funkcia postupne prehľadá všetky možné posuny vozidiel do hĺbky maxHlbka a keď sa mu podarí nájsť výsledok tak skončí a vráti výsledný uzol ako návratovú hodnotu, inak vráti *NULL*.



## Štruktúra *pozicia*

struct pozicia {

unsigned char farba;

unsigned char dlzka;

unsigned char x;

unsigned char y;

char smer;

}POZICIA;

Táto štrukutúra reprezentuje pozíciu vozidla na mape – jeho farbu, dĺžku, smer (horizontálne alebo vertikálne) a jeho súradnice x a y.

## Štrutúra *tah*

struct tah {

unsigned char farba;

unsigned char dlzka;

unsigned char oldx;

unsigned char oldy;

unsigned char newx;

unsigned char newy;

char smer;

}TAH;

Slúži na zapamätanie si práve vykonaného ťahu (pohybu vozidlom). Ukladá farbu, dĺžku aj smer vozidla a aj posun po súradnicovej osi – z oldx do newx a z oldy do newy.

## Štruktúra *krizovatka*

struct krizovatka {

unsigned char mapa[xSize][ySize];

};

Slúži na reprezentáciu stavu križovatky. Ukladá pozície jednotlivých vozidiel.

## Štruktúra *uzol*

typedef struct uzol {

krizovatka krizovatka;

pozicia \*\*pozicie;

tah \*tah;

unsigned char hlbka;

struct uzol \*pred;

}UZOL;

V štruktúre sa ukladajú údaje o aktuálnom stave križovatky, pozíciách všetkých vozidiel, práve vykonanom ťahu a predchádzajúcom stave pred jeho vykonaním, a takisto aj aktuálnu hĺbku v strome prehľadávania.

### Pomocné funkcie

* void init(krizovatka \*k) – inicializuje jednotlivé miesta križovatky *k* na hodnotu 0
* void vypis(krizovatka k) – vypíše križovatku *k* na obrazovku
* void debugVypis(krizovatka k) – vypíše križovatku *k* spolu so súradnicami. Pri bežnom behu program sa nevyužíva, slúži len na pomocné (debug) výpisy
* bool areEqual(krizovatka k1, krizovatka k2) – slúži na porovnanie zhodnosti dvoch križovatiek *k1* a *k2*
* bool isResult(krizovatka k, unsigned char cervene) – skontroluje, či sa križovatka *k* nachádza v koncovom stave, teda či sa *cervene* vozidlo nachádza pri pravej hrane križovatky
* bool isNotNavstivene(krizovatka k, unsigned char h) – zistí či už križovatka *k* bola navštívená v hĺbke menšej alebo rovnej *h*
* bool overTah(krizovatka k, tah \*tah) – funkcia kontroluje či je možné nad križovatkou *k* vykonať ťah *tah*
* void vykonajTah(krizovatka input, krizovatka \*output, tah \*tah) – nad križovatkou *input* vykoná ťah *tah* a túto upravenú križovatku vráti ako *output*

### Vstupné údaje

Na vstupe treba najskôr zadať mená farieb jednotlivých vozidiel, pričom im bude pridelené identifikačné číslo. Tento vstup treba ukončiť zadaním textu „OK“ a následne treba zadať rozmiestnenie jednotlivých vozidiel v križovatke. Tieto údaje treba zadať vo formáte *„identifikačné číslo farby (medzera) dĺžka vozidla (medzera) umiestnenie na X-ovej osi (medzera) umiestnenie na Y-ovej osi (medzera) písmeno „v“ (pre vertikálne umiestnenie) alebo „h“ (pre horizontálne)“.* Aj tento vstup treba ukončiť zadaním textu „OK“. Následne treba zadať maximálnu hĺbku do ktorej má program hľadať výsledok.

Ukážkový vstup:

*Modra* (program pridelí id 1)

*Hneda* (id 2)

*Cervena* (id 3)

*OK* (ukončenie načítavania farieb)

*1 3 1 1 v* (Modré vozidlo (id 1), dĺžky 3, pozícia 1x1 vertikálne)

*3 2 2 2 h* (Červené vozidlo (id 3), dĺžky 2, pozícia 2x2 horizontálne)

*2 3 6 2 v* (Hnedé vozidlo (id 2), dĺžky 3, pozícia 6x2 vertikálne)

*OK* (ukončenie načítavania rozmiestnenia)

5 (hľadanie do hĺbky 5)

### Výstupné údaje

Výstupom programu je postupnosť krokov spojená s výpisom priebežných stavov, vrátane počiatočnej a výslednej križovatky. Táto postupnosť je v programe zapísaná zdola nahor:

Výsledná križovatka

Vykonaný ťah

Priebežný stav

Vykonaný ťah

...

Priebežný stav

Vykonaný ťah

Počiatočná križovatka

### Testovanie

Program bol testovaný na viacerých kombináciách umiestnenia áut a kamiónov (vrátane ukážkového testu na stránke predmetu). Počas testovania boli využité pomocné výpisy, ktoré je možné aktivovať nastavením premennej *debug* na hodnotu *true*. Ukážkový test:

Cervena

Modra

Zlta

OK

1 2 2 3 h

2 3 4 2 v

3 2 6 2 v

OK

4

Počiatočný stav: Výsledné pole:

0 0 0 0 0 0 0 0 0 0 0 3

0 0 0 2 0 3 0 0 0 0 0 3

0 1 1 2 0 3 0 0 0 0 1 1

0 0 0 2 0 0 0 0 0 2 0 0

0 0 0 0 0 0 0 0 0 2 0 0

0 0 0 0 0 0 0 0 0 2 0 0

Výsledná postupnosť krokov pre ukážkový test:

(modra [2] dole 2)(zlta [3] hore 1)(cervena [1] doprava 3)