Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 812 19 Bratislava

Dokumentácia k zadaniu z UI:

8-hlavolam

Autor: **Peter Markuš**
Študijný program: INFO-4
Akademický rok: 2016/2017

Obsah

1 Zadanie úlohy

Našou úlohou je nájsť riešenie 8-hlavolamu. Hlavolam je zložený z 8 očíslovaných políčok a jedného prázdneho miesta. Políčka je možné presúvať hore, dole, vľavo alebo vpravo, ale len ak je tým smerom medzera. Je vždy daná nejaká východisková a nejaká cieľová pozícia a je potrebné nájsť postupnosť krokov, ktoré vedú z jednej pozície do druhej.

Príkladom môže byť nasledovná začiatočná a koncová pozícia:

Začiatok:

1	2	3
4	5	6
7	8	

Koniec:

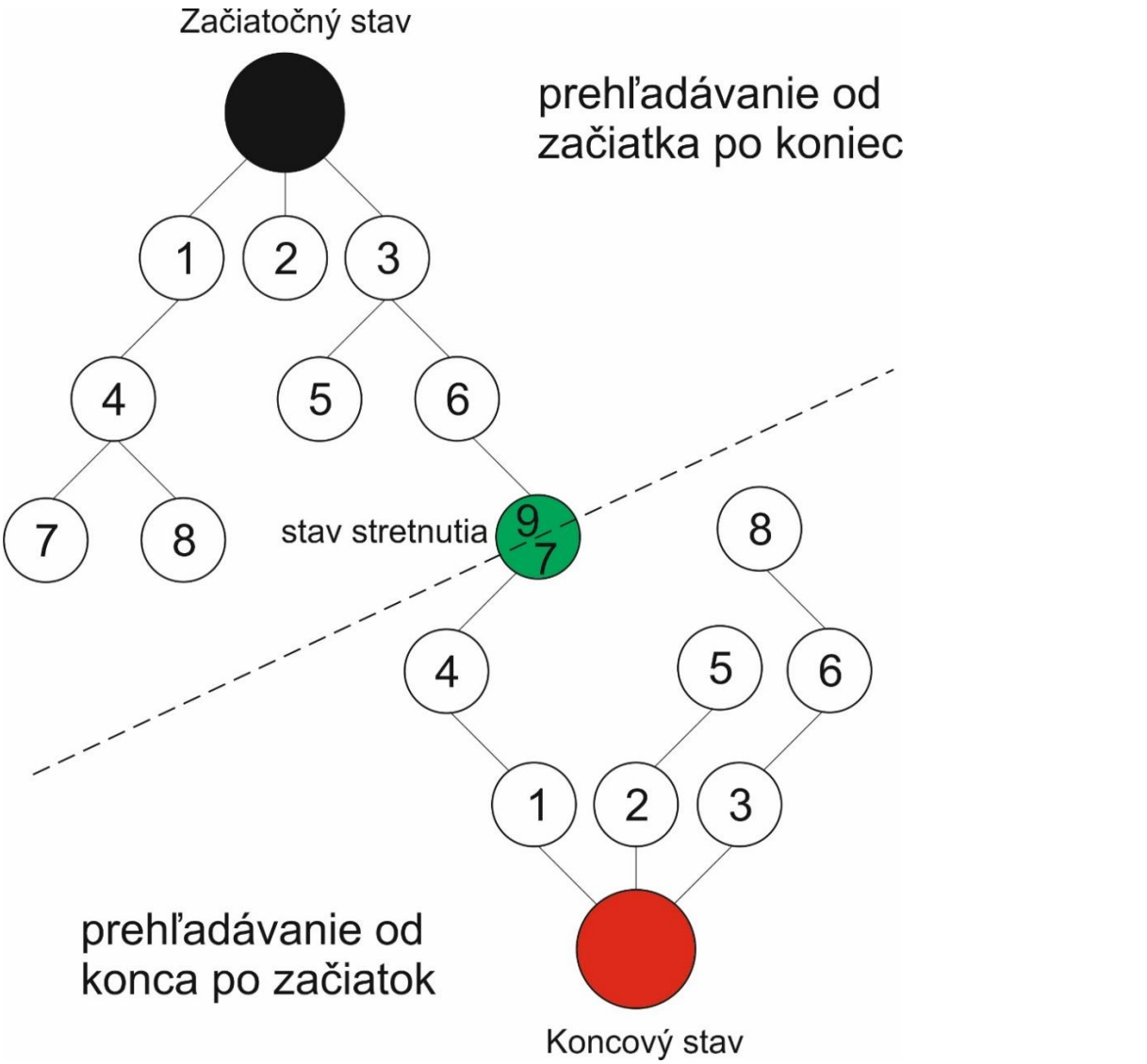
1	2	3
4	6	8
7	5	

Im zodpovedajúca postupnosť krokov je: **VPRAVO, DOLE, VĽAVO, HORE**.

Úloha s variantou c): použite algoritmus obojsmerného hľadania.

2 Riešenie

Na riešenie bolo potrebné podľa zadania použiť algoritmus obojsmerného prehľadávania. Tento algoritmus sa spustí dva krát a to od koncového a začiatočného stavu s využítím prehľadávania do šírky. Hlavný princíp spočíva v tom, že ak sa jeden z algoritmov dostane do stavu, ktorý bol vygenerovaný druhým, tak sa tento stav označí ako spojujúci. Prostredníctvom tohto stavu sa našla cesta medzi začiatočným a koncovým stavom. Na nájdenie postupnosti operácii od začiatočného stavu ku koncovému je potrebné otočiť jednu cestu prehľadávania na to, aby vznikla jednosmerná cesta.
Princíp postupnosti prehľadávania do šírky je vyjadrený číslami na nasledujúcom obrázku. Prehľadávanie od konca sa ku stavu stretnutia dostalo 7. iteráciou a keď ho stretlo prehľadávanie od začiatku v 9. iterácii, prehľadávanie sa ukončilo.



3 Implementácia

3.1 Dátové štruktúry

STATUS – reprezentuje stav hlavolamu. Obsahuje pole typu int veľkosti počtu políčok v hlavolame, pričom každý reprezentuje miesto jedného políčka. Prvkami týchto políčok sú čísla v hlavolame. Medzera je reprezentovaná nulou.
STATUS_HISTORY – je modifikácia štruktúry STATUS. Obsahuje štruktúru STATUS a pole charov, ktoré viedli k aktuálnemu stavu pre zostavovanie jednosmernej cesty cieľa.
TREE_ELE – reprezentuje uzol. Obsahuje štruktúru STATUS a odkazy na štyroch rôznych potomkov ktorý budú obsahovať hodnotu NULL v prípade nemožného aplikovania určitého operátora na rodičovský stav.
HASH_TABLE – uchováva všetky prehľadávacie uzly. Pomocou hashovacej funkcie sa zaraďujú do zoznamu pri ktorom každá položka obsahuje boolovskú hodnotu. Do tabuľky sa vkladajú spracované uzly. Veľkosť tejto tabuľky je 9!, t.j. počet možných stavov hlavolamu.
Na začiatku programu sa tabuľka inicializuje na 0 a postupne popri prehľadávaní a spracovávaní uzlov sa napĺňajú hodnotami 1 prostredníctvom hashovacej funkcie.

Operácie hash tabuľky:

hashIsIn(STATUS s, HASH_TABLE *hashTable) – vráti BOOL hodnotu, ak stav *s* už bol spracovaný v *hashTable*

insertHash(STATUS s, HASH_TABLE *hashTable) – prostredníctvom hashovacej funkcie sa vypočíta index pola, v ktorom sa nastaví hodnota 1 (TRUE)

hash(STATUS act, HASH_TABLE *hashTable) – vráti index pre stav *act*

fact(int i) – funkcia na výpočet faktoriálu čísla *i*

createTable() – vytvorí hashovaciu tabuľku

FRONT – uchováva všetky generované a nespracované uzly. Popri obojsmernom prehľadávaní je potrebné použiť dve fronty pri ktorých sa ukladajú uzly generované od ich začiatku, teda od počiatočného a koncového stavu.

Operácie front štruktúry:

insertFront/2(TREE_ELE *c, FRONT front) – vloženie prvku *c* do fronty

getFront/2(FRONT front) – výber prvku z fronty

emptyFront – vráti hodnotu 1, ak je front prázdny

3.2 Funkcie

TREE_ELE *findTree(TREE_ELE *root, STATUS s, char *ops) – rekurzívna funkcia na vyhľadanie vrchola stromu ktorý obsahuje stav *s* spolu s invertovanými operáciami
STATUS *right(STATUS s) – vráti stav po vykonaní operátora right
STATUS *left(STATUS s) – vráti stav po vykonaní operátora left
STATUS *up(STATUS s) – vráti stav po vykonaní operátora up
STATUS *down(STATUS s) – vráti stav po vykonaní operátora down
void printSummary(STATUS *stav, char msg[255]) – výpiše stav hlavolamu
void addFollower(TREE_ELE *vrchol, TREE_ELE *follow) – pridá child uzol *follow* k rodičovi *vrchol*
char invertOperation(char op) – vráti inverzný operátor (opačný) k zadanému operátoru *op*
int findBlank(STATUS s) – funkcia na nájdenie indexu prázdneho miesta v stave *s*

3.3 Opis algoritmu

1. načítanie počiatočného a koncového stavu
2. vytvorenie hash tabuliek a front štruktúr pre obojsmerné prehľadávanie
3. vloženie počiatočného stavu do druhého frontu a druhej hash tabuľky, a vloženie koncového stavu do prvého frontu a prvej hash tabuľky

prehľadávací cyklus ktorý prehľadáva až pokým nieje jedna z dvoch front prázdna:

4. výber z druhej fronty stav ktorý nieje spracovaný
5. prostredníctvom hash tabuľky zistíme, či sa tento stav už vyskytol v prvom hľadaní.
6. ak nie, tak cyklus pokračuje ďalej. Ak áno, našli sme stav stretnutia a prehľadávací cyklus sa skončí
7. aplikovanie všetkých možných operátorov ktoré nevedú mimo 3x3 tabuľky a nevedú do stavu, v ktorom sa už raz nachádzali
8. novo vytvorené uzly sa pridajú do druhej fronty a druhej hash tabuľky
9. aplikovanie všetkých možných operátorov ktoré nevedú mimo 3x3 tabuľky a nevedú do stavu, v ktorom sa už raz nachádzali
10. novo vytvorené uzly sa pridajú do prvej fronty a prvej hash tabuľky
11. vrátenie sa do 4. kroku
12. ak bol stav stretnutia nájdený, vypíše postupnosť operácií, ktoré vedú od začiatočného stavu skrz stav stretnutia ku koncovému stavu
13. ak tento stav nájdený nebol, výpíše že riešenie nebolo nájdené

4 Testovacie vstupy

Počiatočný stav	Koncový stav	#vytvorených uzlov	#spracovaných uzlov	Stav stretnutia	Postupnosť operácií
1 2 3 4 5 6 7 8 0	1 2 3 4 6 8 7 5 0	22	11	1 2 3 4 0 6 7 5 8	RDLUULLUULL (11)
0 1 2 3 4 5 6 7 8	1 2 3 4 5 6 7 8 0	2798	1729	1 4 2 7 6 3 8 0 5	LURULLDRRUL ... (773)
3 6 8 1 0 4 2 5 7	0 8 1 7 2 6 3 5 4	2590	1631	3 8 1 2 0 6 5 7 4	RULLDRDLURR ... (714)
5 8 6 0 1 3 4 7 2	0 1 2 3 4 5 6 7 8	362880	362880	-	-

5 Zhodnotenie

Implementácia bola zrealizovaná v programovacom jazyku C. Je potrebné použiť kompilátor GNU GCC, prípadne je možné otvoriť projekt prostredníctvom CodeBlocks kompilátora. Program nájde riešenie len ak algoritmus vygeneruje najviac 9! uzlov. Výhodou tohto algoritmu oproti prehľadávania do hĺbky je ten, že je menšia pravdepodobnosť, že prehľadávanie bude trvať donekonečna. Nevýhoda však spočíva v tom, že je má väčšie nároky na pamäť. Obojsmerný algoritmus by bolo možné zlepšiť použitím pri jednom smere iný algoritmus, napr. alg. prehľadávania do hĺbky. Pri zmene koncového a začiatočného stavu dôjde k zmene vytvorenia, spracovávania uzlov a taktiež aj bod stretnutia. Tieto zmeny sú však len zanedbateľne malé. Program pracuje rýchlo, dlhšie trvania programu nastávajú väčšinou v prípade nenájdeného riešenia, nikdy však program netrvá dlhšie ako jednu sekundu.