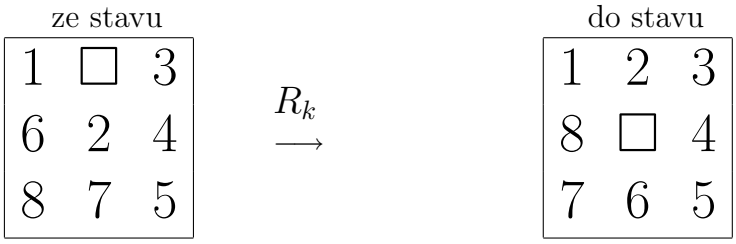
Semestrální práce z předmětu KIV/UIR

Tomáš Maršálek
marsalet@students.zcu.cz

12. května 2012

1 Zadání

V programovacím jazyce C zpracujte jednoduchý výukový program, který plně demonstruje nalezení řešení následující úlohy: Jde o převedení hlavolamu "8" (při respektování pořadí aplikace produkčních pravidel uvedeného na přednášce)



metodou prohledávání grafu s využitím heuristické funkce

$$\begin{aligned} \hat{f}(n_i) &= \hat{g}(n_i) + \hat{h}(n_i) \\ \hat{g}(n_i) &= d(n_i) \quad (\textit{délka cesty z } n_0 \textit{ do } n_i) \\ \hat{h}(n_i) &= P(n_i) + 3Q(n_i) \end{aligned}$$

- $P(n_i)$ je součet vzdáleností každého kamene hlavolamu od svého cílového místa (v možných posuvech).
- $Q(n_i)$ je míra porušení pořadí kamenů zahrnutá tak, že
  - přičítáme hodnotu 2 za každý kámen nenacházející se ve středu pole a jenž není následován správným kamenem,
  - za kámen ve středu pole přičítáme 1.

2 Analýza úlohy

Cílem úlohy je zjistit posloupnost tahů, která řeší výše uvedený hlavolam. Při hledání jednotlivých tahů je možné postupně vypisovat postup algoritmu.

Řešení tohoto hlavolamu je možné provést prohledáváním stavového prostoru, tzn. všech možných pozic kamenů. Jednou takovou možností je použití

slepých strategií pro prohledávání grafu stavového prostoru (prohledávání do hloubky, prohledávání do šířky), nicméně efektivnější strategií, co se týče velikosti podmnožiny prohledaných stavů hlavolamu, jsou cílené strategie. Pro tuto úlohu je zvolena strategie heuristického prohledávání grafu stavového prostoru s výše uvedenou heuristickou funkcí.

Oproti slepým strategiím je výhoda v menším počtu prohledaných stavů, na druhou stranu může být nevýhodou složitý výpočet heuristické funkce.

3 Implementace

Protože je zvolená cílená strategie prohledávání, primární datovou strukturou není graf ani strom, ale prioritní fronta. V každém kroku algoritmu nás zajímá stav hlavolamu s nejnižším ohodnocením. Grafová struktura stavového prostoru je tvořena až dodatečně pomocí rodič-potomek ukazatelů. Aby nedocházelo k cyklení v případě neexistujícího řešení a omezení zbytečného generování stejných stavů, je implementována datová struktura množina, ve které jsou uchovávány vygenerované stavy. Nikdy se tedy nestane, že by se stejný stav vygeneroval vícekrát.

Jazyk implementace je, dle zadání, ANSI C. Je zajištěna správná manipulace s pamětí, nedochází k paměťovým únikům a při skončení programu je rovněž čistá halda.

3.1 Prioritní fronta

I když je velikost zadané úlohy poměrně malá, je prioritní fronta implementována jako binární halda, přitom by stačilo použít jednoduchý seznam.

3.2 Množina

Implementace pomocí rozptylové tabulky s řetězením záznamů při kolizi.

4 Uživatelská příručka

Program je odevzdán jako zip soubor obsahující zdrojové soubory a makefile pro přeložení pomocí gcc.

4.1 Uživatelské módy

Bez specifikování jakýchkoliv dobrovolných argumentů je uživateli pouze představena posloupnost řešících tahů. Je zde ale možnost podívat se jak je generován stavový prostor s možností krokování jednotlivých iterací nebo ne.

Argument (-h) vypíše do konzole nápovědu a skončí program.

Volitelný verbózní mód (-v) vypisuje číslo iterace, hloubku právě vytaženého stavu a všechny dosud neobjevené potomky, které z tohoto stavu vedou dál. Pro každý stav je zobrazena hodnota heuristické funkce.

Dalším volitelným módem je krokování (-s), které umožňuje uživateli programu krokovat jednotlivé iterace prohledávání v případě, že je zvolen verbózní mód.

Dodatečně je implementována možnost uložení výsledného výpisu do souboru, standardně pomocí přepínače (-o) a názvu souboru.

4.2 Příklad použití

Spuštění programu s detailním výpisem vygenerovaných stavů hlavolamu:

```
./8puzzle -v
```

Ukázka výstupu programu - vygenerované stavy:

```
2.
Depth: 1
Grid:
1 2 3
6   4
8 7 5
f: 23

Children:
1 2 3    1 2 3    1 2 3
  6 4    6 4      6 7 4
8 7 5    8 7 5    8   5
f: 21    f: 41    f: 29
------------------------
```

Výstup naznačuje, že stav má hloubku 1 od počátečního stavu a hodnotu heuristické funkce 23. Počet potomků je tři, přestože z prostředního stavu lze jít čtyřmi směry. Poslední stav byl ale objeven již dříve, proto není vygenerován vícekrát.

Ukázka výstupu programu - posloupnost stavů vedoucích k řešení:

```
1   3
6 2 4
8 7 5

1 2 3
6   4
8 7 5

1 2 3
  6 4
8 7 5

1 2 3
8 6 4
  7 5

1 2 3
8 6 4
7   5

1 2 3
8   4
7 6 5
```

5 Závěr

Cílem bylo splnit zadání co možná nejpřesněji.

Překvapením byl nízký počet kroků nutných k nalezení řešení a výsledná krátká posloupnost tahů vedoucích k řešení. Program je tedy možná řešen zbytečně složitě, nicméně má potenciál být rozšířen na stejnou úlohu o větších rozměrech.

Jednoduchým rozšířením by byla implementace možnosti zadávat vlastní počáteční a konečný stav hlavolamu, nicméně zadání tohle nevyžadovalo.

Možné vylepšení by bylo zavedení grafického výstupu namísto tisknutí do konzole, například formou animace generování stavového grafu.