# Pocket cube solver

Samuel Krajči

## 1 Úvod

Základné pokyny pre užívateľa sa nachádzajú v `README.md` v git repozitári:

### Pocket cube solver

*Program for finding the solution of pocket cube (2x2x2 Rubik's cube) that requires the least amount of rotations.*

#### 1.1 How to run

It is recommended to compile code with `ghc -O2 cube.hs`. Then simply run `./cube`.

#### 1.2 Input and output

Enter cube as a cube map in the following format with each color being an unique char:

```
UU
UU
FFRRBBLL
FFRRBBLL
DD
DD
```

Example input:

```
gw
or
ybyobwrg
wrbrgobg
ow
yy
```

Example output:

```
R F U' R' F R F R' F U' F'
```

The computation can take up to 1 minute, however most of the cases take only a few seconds. Output is a sequence of moves using traditional Rubik's cube notation.

## 2 Dátová reprezentácia

Je mnoho možných reprezentácií rubikovej kocky, niektoré sú lepšie čitateľné pre človeka, na iných sa lepšie robia rózne operácie, ako napríklad rotácie. Najčitateľnejšia serializovateľná reprezentácia je plášť kocky, pre to sa táto reprezentácia používa na zadávanie vstupu.
Avšak pre prácu s kockou je prektickejšie si ju uložiť do špeciálnej dátovej štruktúry - kocka (`Cube`) je zložená zo šiestich stien (`Side`), pričom jedna stena je zložená zo štyroch znakov.
Takáto reprezentácia je prehľadná a veľmi jednoducho vieme kontrolovať, či je kocka vyriešená.

## 3 Rotácie

Na našej kocke vieme robiť iba tri typy rotácií:

- predná stena,
- pravá stena,
- horná stena,

pričom každú z týchto stien vieme rotovať v kladnom, alebo zápornom smere.
Rotácie zvyšných troch stien vieme nahradiť rotáciou protiľahlej steny do opačného smeru.
Všimnime si, že po aplikovaní ľubovoľnej z týchto operácií ľavý-dolný-zadný diel nezmení polohu ani orientáciu.

## 4 Algoritmus

Zostavne graf, ktorého vrcholy reprezentujú stavy kocky, pričom medzi dvoma vrcholmi vedie hrana práve vtedy ak sa z jedného stavu vieme rotáciou dostať do druhého.
Potom chceme v tomto grafe nájsť najkratšiu cestu medzi vyriešenou kockou a vstupným stavom.
Túto cestu budeme hľadať prehľadávaním do šírky.
Pri BFS (prehľadávaní do šírky) potrebujeme zásobník, ten implementujeme pomocou dátovej štruktúry `Sequence.Seq` a taktiež mapu (`dictionary`) na ukladanie už nájdených ciest ku daným stavom. Na to použijeme štruktúru `Map.Map`.

## 5 Rýchlosť

Samotné BFS trvá lineárny čas vzhľadom na počet vrcholov (resp. vzhľadom na súčet počtu vrcholov a hrán, no keďže z každého vrcholu vedie 6 hrán, tak počet hrán je lineárne závislý od počtu vrcholov). Avšak funkcie `lookup`, `insert` a `member` na štruktúre `Map.Map` trvajú logaritmický čas vzhľadom na počet prvkov v nej, čo môže byť až rovné počtu vrcholov, a keďže pre každý prvok, ktorý vyberieme zo zásobníka musíme spraviť aspoň jednu z týchto operácií, celková zložitosť nám narasie na $O(nlogn)$, kde $n$ je počet vrcholov.
Je známe, že počet všetkých stavov *pocket cube* je 3674160, čo by pre našu časovú zložitosť nemal byť problém.
No, bohužiaľ, dátové štruktúry z `Sequence.Seq` a `Map.Map` sú relatívne pomalé (aj napriek ich dobrej asymtotickej zložitosti) a tak realny čas behu programu je zhruba jedna minúta v najhoršom a zhruba 10 sekúnd v priemernom prípade.