**Slovenská technická univerzita v Bratislave**

Fakulta informatiky a informačných technológii

# Zadanie 2

## Prehľadávanie stavového priestoru

## Umelá Inteligencia

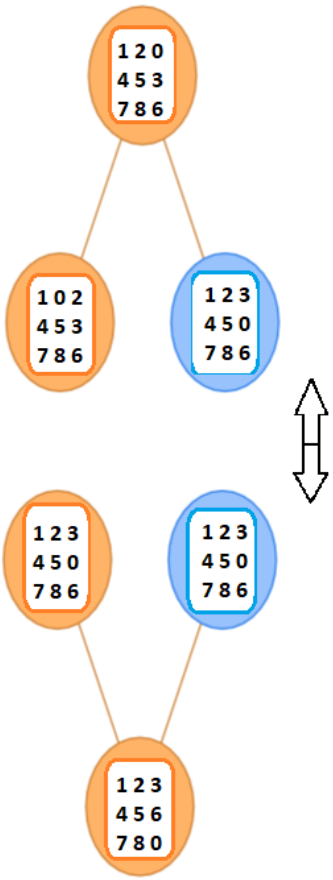
Martin Raffáč

**Zadanie:**

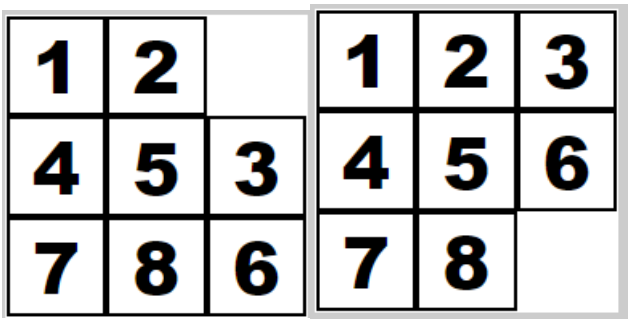
Prehľadávanie stavového priestoru, prostredníctvom logickej hry 8-puzzle použitím algoritmu obojsmerného hľadania.


**Algoritmus:**

Princíp algoritmu spočíva v prehľadávaní priestoru počnúc začiatočným uzlom a zároveň koncovým uzlom. A vrstvu po vrstve sa analyzujú uzly priamo súvisiace so zdrojovým a koncovým uzlom. To znamená že sa prehľadáva stavový priestor z oboch strán súčasne, dokým sa nenájde taký uzol, ktorý je zhodný. Potom vzniká cesta ku koncovému respektíve počiatočnému stavu. No ak zhodný uzol v danej vrstve nebol nájdený prejde sa na ďalšiu vrstvu, tento cyklus sa opakuje dokým nedôjde k zhode alebo sa vyplytvajú všetky možné kombinácie.



V tomto prípade bol počiatočný stav 120,456,786 a koncový stav 123,456,789 a teda spoločný uzol boj nájdený na prvej vrstve. Tento algoritmus je výhodnejší ako obyčajné hľadanie do šírky pretože jeho priestorová zložitosť je $O(B^{n/2})$




**Stavový priestor:**

Stavový priestor predstavujú všetky rozložené 8-puzzle dostupné z počiatočného a koncového stavu. Zmeny stavov sa vykonávajú prostredníctvom posunutí do jednotlivých strán (HORE, DOLE, DOPRAVA, DOĽAVA).

**Implementácia:**

Program je implementovaný prostredníctvom jazyka python v prostredí Visual Studio 2019. Táto implementácia je nezávislá na programovacom prostredí. Moja implementácia je obmedzená na hlavolam o rozmeroch 3x3 políčka. Program funguje na jednoduchých princípoch pričom používateľ na vstupe zadáva počiatočný a koncový stav hracej plochy. Medzera na hracej ploche je reprezentovaná číslicou 0.

```
pre spustenie hry stlačte 1 pre ukončenie 0 : 1
Zadajte vstup (čísla od 1 po 8 a na miesto medzery číslo 0)
Začiatočný stav: 143502786
Zadajte výstup (čísla od 1 po 8 a na miesto medzery číslo 0)
Koncový stav: 123456780
```

Po spustení programu sa vo while cykle vykonáva obojsmerné hľadanie do šírky a podľa nasledovných podmienok sa určujú možné kroky na hracej ploche.

```
if blankIndex_zaciatok!=0 and blankIndex_zaciatok!=1 and blankIndex_zaciatok!=2 and found == False:
    upNode_zaciatok = copy.deepcopy(currentNode_zaciatok)
    upNode_zaciatok[blankIndex_zaciatok] = upNode_zaciatok[blankIndex_zaciatok-3]
    upNode_zaciatok[blankIndex_zaciatok-3] = 0
    upNode_zaciatok.append('HORE')
    found = check_zaciatok(upNode_zaciatok)

if blankIndex_zaciatok!=0 and blankIndex_zaciatok!=3 and blankIndex_zaciatok!=6 and found == False:
    leftNode_zaciatok = copy.deepcopy(currentNode_zaciatok)
    leftNode_zaciatok[blankIndex_zaciatok] = leftNode_zaciatok[blankIndex_zaciatok-1]
    leftNode_zaciatok[blankIndex_zaciatok-1] = 0
    leftNode_zaciatok.append('DOĽAVA')
    found = check_zaciatok(leftNode_zaciatok)

if blankIndex_zaciatok!=6 and blankIndex_zaciatok!=7 and blankIndex_zaciatok!=8 and found == False:
    downNode_zaciatok = copy.deepcopy(currentNode_zaciatok)
    downNode_zaciatok[blankIndex_zaciatok] = downNode_zaciatok[blankIndex_zaciatok+3]
    downNode_zaciatok[blankIndex_zaciatok+3] = 0
    downNode_zaciatok.append('DOLE')
    found = check_zaciatok(downNode_zaciatok)

if blankIndex_zaciatok!=2 and blankIndex_zaciatok!=5 and blankIndex_zaciatok!=8 and found == False:
    rightNode_zaciatok = copy.deepcopy(currentNode_zaciatok)
    rightNode_zaciatok[blankIndex_zaciatok] = rightNode_zaciatok[blankIndex_zaciatok+1]
    rightNode_zaciatok[blankIndex_zaciatok+1] = 0
    rightNode_zaciatok.append('DOPRAVA')
    found = check_zaciatok(rightNode_zaciatok)
```

Program sa snaží objaviť čo najviac nových uzlov na jednej vrstve, ak je pohyb možný vykoná sa a následne sa vo funkcii chcek_zaciatok() skontroluje či náhodou už rovnaké rozpoloženie číslic nebolo objavené ak nie tak sa tento uzol zapíše do zoznamu visitedList_zaciatok. A ak už objavený bol tak funkcia tento uzol už nezapisuje. Na rovnakej báze funguje aj hľadanie do šírky z koncovej strany len s rozdielom, že ak vykonávame pohyb tak zapisujeme opačný smer ako sa vykonal (ak ideme HORE program zapíše DOLE ak ideme DOPRAVA program zapíše DOĽAVA atď.) a to z toho dôvodu, že musíme brať v úvahu že po tejto ceste sa budeme vracať takže je potrebné vykonávať opačný smer krokov aby sme došli ku koncovému stavu.

Ak program nájde spoločný stav znamená to, že sa našla cesta ktorá spojila začiatok a koniec a hlavolam tak bol vyriešený. To či existuje spoločný stav sa porovnáva po prehľadaní danej vrstvy vo for cykloch.

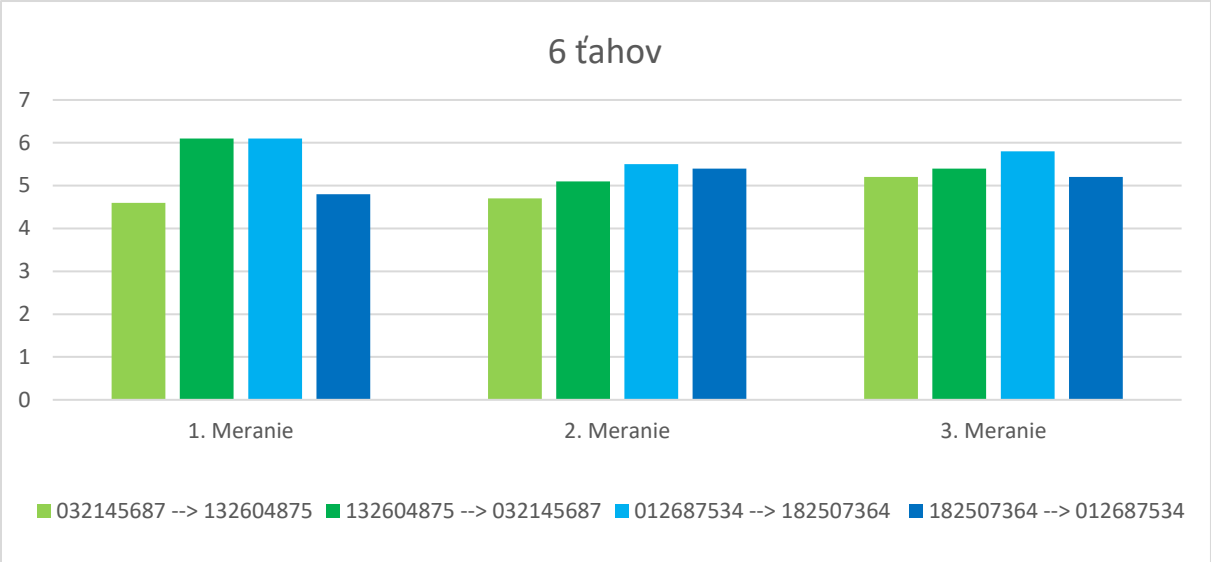
```
for x in range(len(pomocny_zoznam_zaciatok)):
    for y in range(len(pomocny_zoznam_koniec)):
        found = checkFinal(pomocny_zoznam_zaciatok[x],pomocny_zoznam_koniec[y])
        if found == True:
```

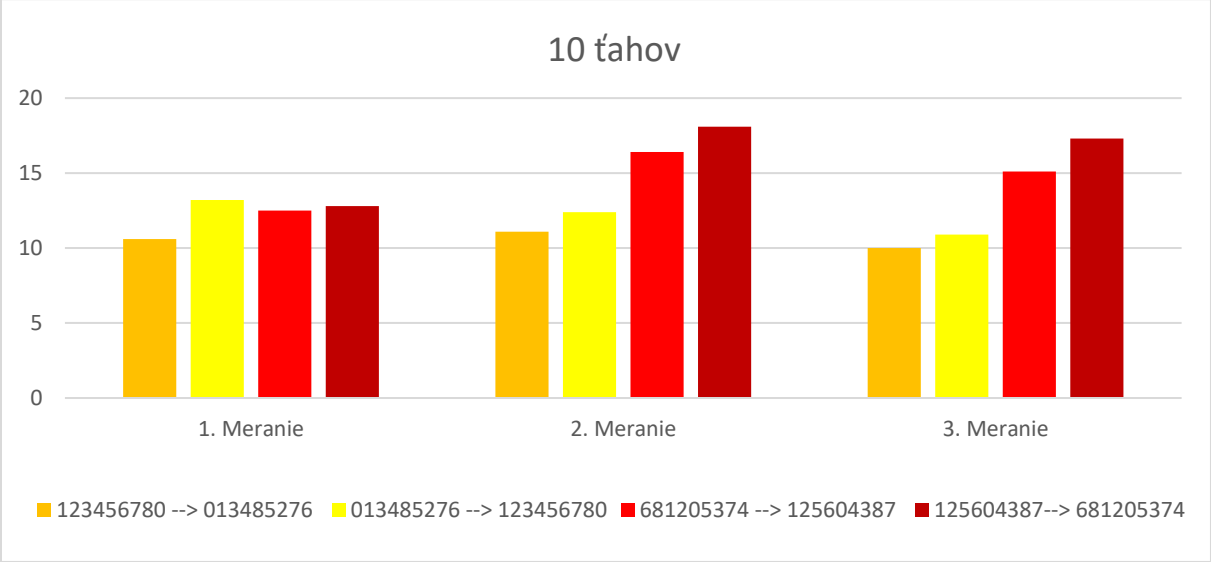
Na konci program vypíše postupnosť krokov, ktoré je treba vykonať aby sa hlavolam dostal z počiatočného ku koncovému stavu. A taktiež sa vypíše počiatočný a koncový stav spolu s časom za ktorý program vyriešil hlavolam a tiež počet krokov riešenia. (v programe sa posúva prázdne políčko)

```
pre spustenie hry stlačte 1 pre ukončenie 0 : 1
Zadajte vstup (čísla od 1 po 8 a na miesto medzery číslo 0)
Začiatočný stav: 012345678
Zadajte výstup (čísla od 1 po 8 a na miesto medzery číslo 0)
Koncový stav: 123456780
0 1 2
3 4 5
6 7 8
------
1 2 3
4 5 6
7 8 0
------
['DOPRAVA', 'DOLE', 'DOĽAVA', 'DOLE', 'DOPRAVA', 'DOPRAVA', 'HORE', 'DOĽAVA', 'DOĽAVA', 'DOLE', 'DOPRAVA']
 ['HORE', 'HORE', 'DOPRAVA', 'DOLE', 'DOLE', 'DOĽAVA', 'DOĽAVA', 'HORE', 'DOPRAVA', 'DOPRAVA', 'DOLE']
Počet ťahov potrebných na vyriešenie:  22
čas riešenia: 77.94867420196533
------
```

**Testovanie:**

Svoju implementáciu som testoval na riešení s 6 krokmi potrebnými na vyriešenie pričom počiatočné a koncové stavy boli 032145687→132604875 a naopak a druhú kombinácia bola 012687534→ 182507364 a naopak. Na druhom grafe je znázornené riešenie hlavolamu na 10 krokov pre dvojice 123456780 → 013485276 a naopak a 681205374 → 125604387 a naopak. Jednotlivé časy sú udané v milisekundách. Zaujímavým zistením bolo že v meraniach pre riešenie so 6 ťahmi bola väčšina časov vyrovnaná zatiaľ čo vo väčšine meraní pre riešenie s 10 ťahmi bola dĺžka času vykonávania programu dlhšia v obrátenom smere a teda od konca k začiatku. Moje testovacie programy sa nachádzajú v samostatných súboroch.




**Zhrnutie a hodnotenie:**

Mnou implementovaný algoritmus obojsmerného hľadania nezaručuje najoptimálnejšiu rýchlosť vyhľadávania no je stále rýchlejší ako obyčajné hľadanie do šírky. Keďže $O(B^{n/2})$ je rýchlejšie ako $O(B^n)$. Ale oproti informovaným algoritmom je pomalší.

Do programu som importoval knižnice queue, copy a time, ktoré boli nevyhnutné pre implementáciu môjho programu. Queue je zoznam prvkov, ktoré už boli dosiahnuté ale ešte neboli rozvinuté. Pomocou time meriam dĺžku trvania programu a copy slúži na skopírovanie jednotlivých uzlov.

Program je možné pomerne jednoducho rozšíriť na iné rozmery ako len 3x3 políčka, a to zmenením podmienok if v cykle while.

Program využíva niekoľko globálnych premenných a to zoznamy, v ktorých sú uložené informácie o objavených uzloch.