Samuel Řeřicha
AIS ID 103110

Zadanie 2

Prehľadávanie stavového priestoru

Umelá Inteligencia

Samuel Řeřicha

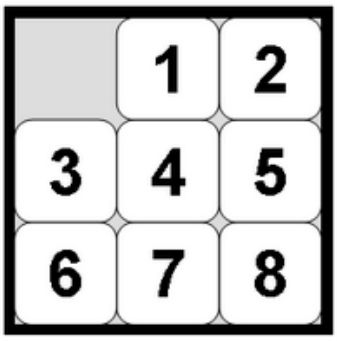
Cvičenie: Štvrtok 14:00

Cvičiaci: Ing. Juraj Vincúr

Samuel Řeřicha
AIS ID 103110

Zadanie:

Prehľadávanie stavového priestoru, respektíve riešenie 8-puzzle pomocou A-star algoritmu s použitím **displacement count** a **sum of Manhattan distances** heuristík.



Algoritmus:

A-star algoritmus je informovaný prehľadávajúci algoritmus typu best-first, pričom poradie prehľadávania dostupných stavov závisí od ich vzdialenosti od počiatočného stavu a odhadovenej vzdialenosti od finálneho stavu, pričom druhú hodnotu sprostredkúva vybraná heuristická funkcia. Rekonštrukcia cesty je realizovaná prostredníctvom referencií novonájdených uzlov na uzol, z ktorého sme ich našli.

Stavový priestor:

Stavový priestor predstavujú všetky rozloženia 8-puzzle dostupné z počiatočného stavu, pričom zmeny stavov sú prevedené prostredníctvom posunutí do strán (UP/DOWN/LEFT/RIGHT).

Heuristika 1 (Displacement count):

Pre skúmaný stav zisťuje počet "políčok", ktorých hodnota sa nezhoduje s hodnotou "políčka" (ďalej "tile) finálneho stavu. Prázdne políčko sa môže, ale nemusí brať do úvahy (v mojej implementácii sa do úvahy neberie). Pre nižšie zobrazený aktuálny a finálny stav je hodnota **h1(akt_state, final state) == 5.**

Aktuálny stav

Finálny stav

Samuel Řeřicha
AIS ID 103110

Heuristika 2 (Súčet Manhattanských vzdialeností):

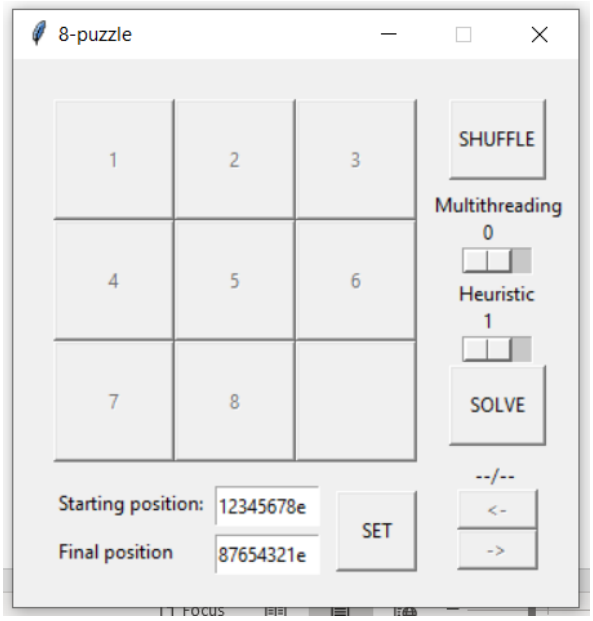
Pre každý tile zistí vzdialenosť medzi jeho súčasnou a želanou pozíciou, pričom "pohyb" je uvažovaný iba v horizontálnej a vertikálnej podobe, a teda hypotetický diagonálny pohyb by mal vzdialenosť 2, nie ~1.4). Podobne ako pri predchádzajúcej heuristike sa prázdny tile môže, ale nemusí brať do úvahy (v mojej implementácii tejto heuristiky sa do úvahy neberie). Pre nižšie zobrazený aktuálny a finálny stav je pre tile "5" vzdialenosť 2 a celková hodnota je **(0 + 2 + 2 + 0 + 1 + 1 + 0 + 0 + 1) == 7**.

Aktuálny stav

Finálny stav

Implementácia:

Implementácia je realizovaná prostredníctvom jazyka **python**, špecificky v prostredí IDE **PyCharm.** Implementácia je limitovaná na hlavolam 3x3, pričom som sa pre to rozhodol preto, že iné veľkosti sú bonus, a značná časť funkcionality a GUI by bola komplikovaná zmenou veľkosti.



3x3 grid buttonov na vizualizáciu aktuálneho stavu.

2 text entry polia pod ním slúžia na manuálne zadanie vybranej kombinácie stavov.

SET button slúži na ich uloženie ako aktuálnych stavov, ktoré bude program spracovávať.

SHUFFLE vygeneruje "náhodnú" riešiteľnú kombináciu štartovacej a koncovej pozície.

Multithreading slider slúži na kontrolovanie toho, či je **solve** funkcia vykonávaná v novom threade (aby GUI ostalo responzívne), alebo nie. Je možné túto možnosť vypnúť, pretože neviem s istotou vylúčiť, že to nespomaľuje vykonávanie.

Heuristic slider kontroluje výber heuristiky.

SOLVE button spustí vykonávanie riešenia.

-> a **<-** buttony slúžia na navigáciu medzi krokmi riešenia. (vizualizácia prostredníctvom GUI)

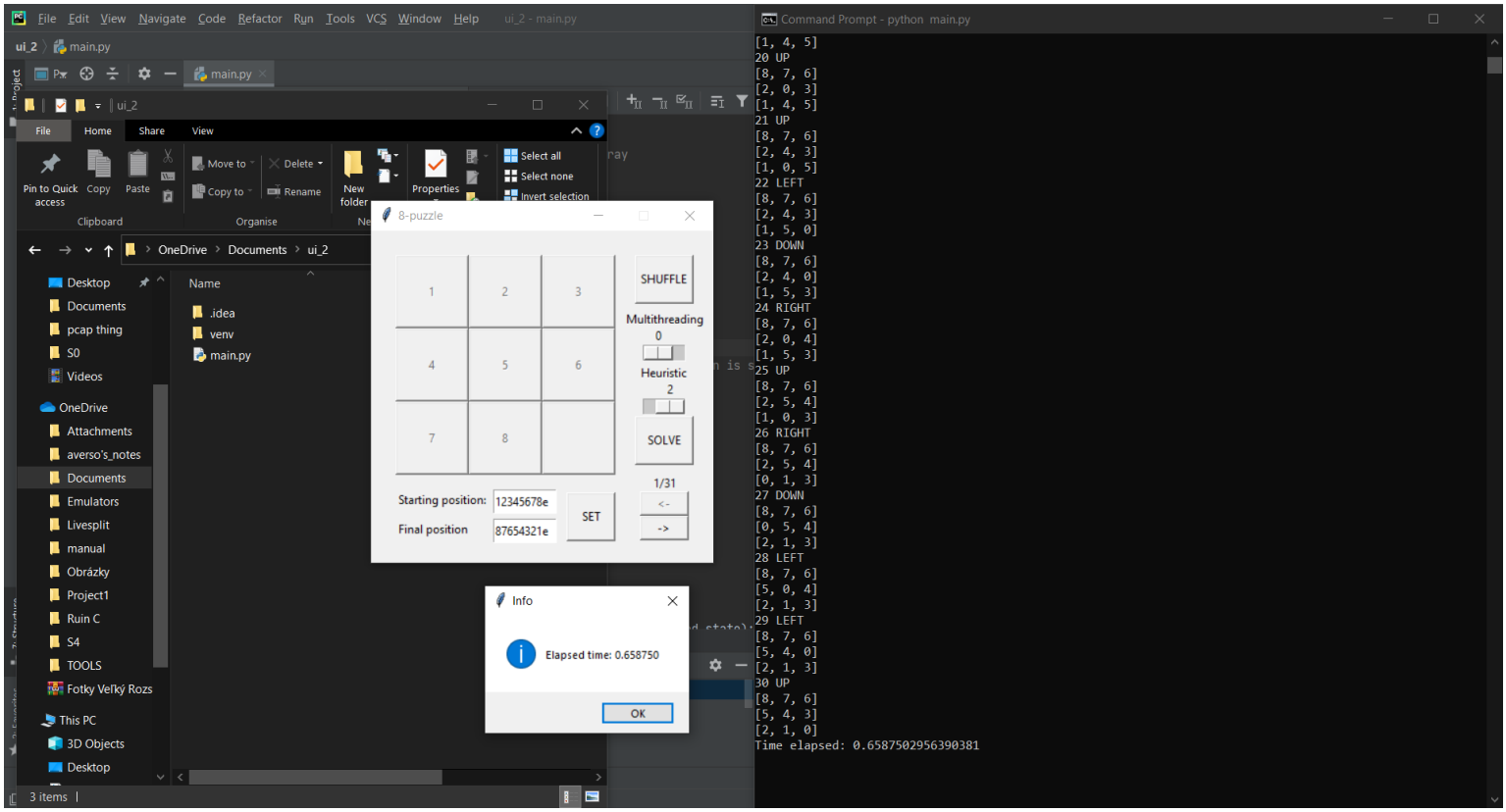
Samuel Řeřicha
AIS ID 103110

Je možné tabovať a shift-tabovať medzi jednotlivými prvkami GUI a pomocou **enter** key použiť momentálne focusnutý widget (button alebo text entry pole). Boli implementované isté opatrenia, ktoré by mali zabrániť konfliktujúcej funkcionalite pri multithreadingu. **Program taktiež do konzoly vypíše jednotlivé kroky riešenia a počas výpisu kontroluje, či podniknuté kroky boli legálne.**

```
global solution
solution = copy.deepcopy(solution_temp)

for i in range(len(solution_temp)):
    solution_temp[i] = [0 if x == 'e' else x for x in solution_temp[i]]
    if i == 0:
        print(solution_temp[i][0:3])
        print(solution_temp[i][3:6])
        print(solution_temp[i][6:9])
        continue
    curr_pos = get_position(solution[i], 'e')
    prev_pos = get_position(solution[i - 1], 'e')
    if prev_pos - curr_pos == 3:
        print("%2d" % i, "DOWN")
        if not can_move_down(solution[i - 1]):
            tkinter.messagebox.showinfo("BAD", "%d ILLEGAL MOVE" % i)
    if prev_pos - curr_pos == -3:
        print("%2d" % i, "UP")
        if not can_move_up(solution[i - 1]):
            tkinter.messagebox.showinfo("BAD", "%d ILLEGAL MOVE" % i)
    if prev_pos - curr_pos == 1:
        print("%2d" % i, "RIGHT")
        if not can_move_right(solution[i - 1]):
            tkinter.messagebox.showinfo("BAD", "%d ILLEGAL MOVE" % i)
    if prev_pos - curr_pos == -1:
        print("%2d" % i, "LEFT")
        if not can_move_left(solution[i - 1]):
            tkinter.messagebox.showinfo("BAD", "%d ILLEGAL MOVE" % i)
    print(solution_temp[i][0:3])
    print(solution_temp[i][3:6])
    print(solution_temp[i][6:9])

print("Time elapsed:", time, "\n")
```



Samuel Řeřicha
AIS ID 103110

Reprezentácia údajov:

Nájdené a spracované rozloženia tiles sú uložené v objektoch **State**, ktoré majú atribúty:

- **distance** vzdialenosť, resp. počet ťahov vykonaných pri zmene zo štartovného na súčasný stav
- **map[]** array obsahujúci reprezentáciu poľa, čísla **1-8** a znak **'e'** reprezentujú rozloženie
- **previous** referencia na stav, priamo z ktorého sme sa na tento stav dostali

Konkrétna implementácia algoritmu (funkcia *solve*):

```
if not solvable:

      showmessage(<not solvable>)
      return
< GUI stuff >
queue = Queue()            # queue s momentalne dostupnymi uzlami, cez heap
start = State(start_state) # start state je global variable
queue.insert(curr_heuristic(start.map), start)
searched = dict()          # obsahuje tie co som uz nasiel - bud su v queue
                                 alebo uz boli prejdene
searched.add(start.map)
while not queue.is_empty():
      top = queue.pop() # momentalne najlacnejsi dostupny uzol
      if top.map == wanted_state:
            < GUI stuff >
            printsolution()
            return

      if (can_move_down(top.map))
            temp_map = copy(top.map)
            move_down(temp_map)
            if temp_map not in searched
                  add_state = State(temp_map, top.distance + 1, top)
                  searched.add(add_state.map)
                  queue.insert(add_state.distance + curr_heuristic(add_state.map))
      < to iste pre zvysne 3 pohyby >

showmessage(<unsuccesful>)
```

searched je dictionary obsahujúci prvky, ktoré už boli dosiahnuté, pričom implementácia využíva dictionary built-in v pythone. Jednotlivé keys sú mapy tranformované na integer, hodnoty sú 0 (podstatná je prítomnosť kľúča – rozloženia určitého stavu). Time complexity **O(log n)**

queue je zoznam prvkov, ktoré už boli dosiahnuté, ale ešte neboli rozvité, pričom podľa definície **a-star** algoritmu je zoznam zoradený podľa funkcie **f(n) = g(n) + h(n)** kde
g(n) je doterajší distance uzla od počiatočného stavu a
h(n) je odhad vzdialenosti uzla od cieľového stavu (hodnota je poskytutá práve vybranou heuristikou)**.** Je implementovaný pomocou **heapq** library pythonu, konkrétne binárnou haldou. V prípade, že hodnota **f(n)** je pre 2 prvky v **queue** rovnaká je uprednostňovaný prvok s vyššou doterajšou vzdialenosťou **g(n).** Time complexity **O(log n).**
Toto môže mať vo veľmi špecifických prípadoch za následok nájdenie mierne neoptimálnej cesty. Výhodou tohto rozhodnutia je však lepší čas vykonávania, pretože sa jedná o lačnejší algoritmus. Toto rozhodnutie závisí od jedného znamienka pri porovnávaní. To, že toto rozhodnutie dokáže viesť k mierne horšiemu riešeniu som si uvedomil až po testovaní. Odovzdaný zdrojový kód obsahuje toto znamienko na line 25, kde to je tiež do určitej miery okomentované.

Samuel Řeřicha
AIS ID 103110

Testovanie:

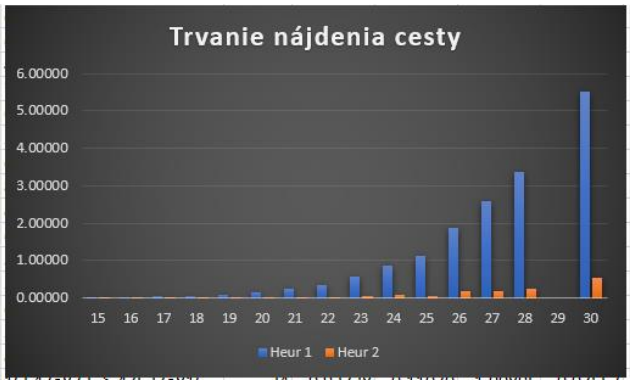
Testovanie počas implementácie prebiehalo najmä na kombinácii vstupov 12345678e -> 87654321e, s ktorým sa program aj vytvorí. Daná kombinácia je zvolená preto, že to je najzložitejšie/najdlhšie riešenie (30 operácií/krokov), ktoré sa mi v čase implementácie podarilo nájsť. Okrem tejto kombinácie boli taktiež použité kombinácie poskytnuté implementovaným **shufflom**. Okrem manuálnej kontroly legality krokov je implementovaná aj už spomenutá kontrola, či bol vykonaný krok možný, vykonávaná pri spracovaní výstupu.

Po implementácii bolo testovanie vykonávané na kombináciách vstupov generovaných **SHUFFLE** button funkciou na vzorke 30 kombinácií obojstranne (dokopy 60).

Prvá časť zobrazuje priemerné hodnoty pre daný počet krokov, druhá časť zobrazuje vyskúšané kombinácie a dosiahnuté časy, tretia časť zobrazuje pomer dĺžok trvania vykonávania kombinácie **A -> B** a kombinácie **B -> A.** Program bol v čase testov spustený prostredníctvom **PyCharmu**, bez multitaskingu, na 4core 8thread CPU bežiacom na 2.3 - 2.5 GHz.

Length	Count	Heur 1	Heur 2
15	1	0.01623	0.00499
16	2	0.03268	0.00825
17	1	0.04238	0.00597
18	1	0.05633	0.00999
19	1	0.08935	0.00906
20	2	0.13836	0.01696
21	4	0.26561	0.02693
22	4	0.34593	0.02919
23	2	0.57281	0.05685
24	4	0.87440	0.08234
25	2	1.13025	0.05335
26	3	1.87574	0.18587
27	1	2.59318	0.18785
28	1	3.35959	0.24663
30	1	5.50034	0.53650

Kombinácia	Length	H1	H2	H1 opak	H2 opak	H1/H1o	H2/H2o
61482e537 -> 564138e27	15	0.01651	0.00499	0.01595	0.00499	1.03452	0.99971
132e57648 -> 6e1523748	16	0.03391	0.00901	0.02493	0.01101	1.36001	0.81864
2416e5837 -> 43258167e	16	0.03195	0.00798	0.03993	0.00499	0.80013	1.60077
e64875321 -> 43578e261	17	0.03191	0.00596	0.05286	0.00598	0.60377	0.99570
e76854321 -> 745328e61	18	0.05086	0.01000	0.06179	0.00997	0.82313	1.00301
476521e38 -> 2e1568734	19	0.08880	0.01114	0.08990	0.00698	0.98778	1.59544
78e142653 -> 26487315e	20	0.12766	0.01696	0.14259	0.01695	0.89526	1.00042
6e8324571 -> 6e3721485	20	0.15159	0.01795	0.13157	0.01599	1.15217	1.12258
2687345e1 -> e74865123	21	0.18263	0.02693	0.22723	0.01496	0.80373	1.80023
e25841736 -> 384e56172	21	0.18652	0.02493	0.34862	0.02593	0.53501	0.96160
8657432e1 -> 56e871432	21	0.36007	0.03989	0.36004	0.03993	1.00008	0.99909
825e73614 -> 1326e4875	21	0.21742	0.02095	0.24235	0.02194	0.89713	0.95471
e51283647 -> 4287e3651	22	0.30003	0.03391	0.35301	0.02696	0.84990	1.25777
51468e372 -> 637e18542	22	0.37999	0.02194	0.41090	0.02294	0.92479	0.95642
368e24751 -> 5e1387462	22	0.30219	0.01401	0.37497	0.04890	0.80591	0.28649
e64527813 -> 1637e8542	22	0.26932	0.01197	0.37703	0.05290	0.71432	0.22625
38e124675 -> 823e17564	23	0.52667	0.06881	0.70570	0.09173	0.74631	0.75020
87254163e -> 8e3456721	23	0.49868	0.02497	0.56019	0.04189	0.89020	0.59601
16547e832 -> 43527e816	24	0.92728	0.11070	1.00895	0.07553	0.91906	1.46559
e12746583 -> 62384175e	24	0.94148	0.08180	0.76662	0.06906	1.22810	1.18457
458732e16 -> 8175e6342	24	0.71069	0.07941	0.94454	0.18432	0.75241	0.43083
7184365e2 -> 25138e467	24	0.80385	0.02502	0.89176	0.03291	0.90142	0.76015
215e48673 -> 31682745e	25	1.13074	0.03092	1.08250	0.02992	1.04456	1.03328
183574e26 -> 278e45361	25	1.14793	0.12068	1.15984	0.03188	0.98973	3.78514
514e87632 -> 25684e173	26	1.77366	0.08973	1.72678	0.06586	1.02715	1.36242
2413786e5 -> 4e2163857	26	1.89258	0.12967	1.83967	0.21542	1.02876	0.60193
3147e8526 -> 5683e7412	26	1.93997	0.29978	2.08178	0.31478	0.93188	0.95233
1364287e5 -> 847312e65	27	2.42642	0.16111	2.75995	0.21460	0.87915	0.75076
34816275e -> e76485213	28	3.28772	0.20745	3.43146	0.28581	0.95811	0.72581
12345678e -> 87654321e	30	5.52120	0.52920	5.47948	0.54380	1.00761	0.97316
						0.91326	**1.06503**



Samuel Řeřicha
AIS ID 103110

Čas potrebný na vyriešenie pri zvyšujúcom sa počte krokov pri oboch prípadoch rastie exponenciálne s koeficientom 2 (približne).
Mierne odchýlky je možné pripísať tomu, že program je interpretovaný, bežiaci v IDE a bežiaci spolu s množstvom ostatným procesov. Taktiež sa čas medzi rôznymi kombináciami s rovnakým počtom potrebných krokov môže čiastočne líšiť, čo by sa dalo eliminovať iba veľkým množstvom testovaných kombinácií.

Ako posledný stĺpec naznačuje, trvanie riešenia kombinácie **B -> A** sa od trvania pri kombinácii **A -> B** priemerne líši dostatočne málo na to, aby sa to dalo pripísať už spomenutým faktorom.

Testovacie príklady:

#steps	START	-> WANTED
15	**61482e537**	**-> 564138e27**
16	**132e57648**	**-> 6e1523748**
17	**e64875321**	**-> 43578e261**
18	**e76854321**	**-> 745328e61**
19	**476521e38**	**-> 2e1568734**
20	**6e8324571**	**-> 6e3721485**
21	**8657432e1**	**-> 56e871432**
22	**368e24751**	**-> 5e1387462**
23	**87254163e**	**-> 8e3456721**
24	**7184365e2**	**-> 25138e467**
25	**215e48673**	**-> 31682745e**
26	**2413786e5**	**-> 4e2163857**
27	**1364287e5**	**-> 847312e65**
28	**34816275e**	**-> e76485213**
30	**12345678e**	**-> 87654321e**

Porovnanie heuristík:

Z nazbieraných dát poskytnutých meraním vyplýva, že heuristika 2 je v priemere mnohonásobne lepšia (približne 4- až 16- krát). Tento pomer sa zdá byť konzistentný, ak berieme do úvahy nevyhnutne značné odchýlky pri jednoduchších kombináciách. Zaktiež to pri všetkých zložitostiach závisí aj od konkrétnych kombinácií a "sklonov" heuristík vyberať určité stavy.

Zhrnutie a ohodnotenie:

Dátové štruktúry použité na implementáciu by mali zaručiť dostatočne rýchle vykonanie.

Pre stavy som si vybral kombináciu údajov **map, distance** a **previous**. Prvý a posledný pre riešenie v podstate nevyhnutné a **distance** je atribútom, pretože alternatívou by bolo pri každom uzle vkladanom do zoznamu dostupných uzlov traverzovať pomocou **previous** referencií, aby sme zistili hĺbku/distance daného uzla, čo by program výrazne spomalilo.
Predchádzajúce, resp. posledný operátor som sa rozhodol nevyužiť, pretože check, či daný stav už bol navštívený je dostatočne lacný. Iné informácie sa nezdali potrebné.

Samuel Řeřicha
AIS ID 103110

Pre zoznam už objavených stavov som použil **dictionary,** ktorý je súčasťou štandardnej knižnice pythonu. S časovou zložitosťou **O(log n)** je rýchlejší než lineárny search v lineárnom poli a je veľmi jednoduchý na implementáciu.

Zoznam dostupných stavov je implementovaný prostredníctvom **haldy**, čo by malo znovu zaručiť primeranú rýchlosť vykonania svovou časovou zložitosťou **O(log n)**. Ako už bolo spomenuté, poradie stavov v halde vie značne ovplyvniť riešenie.

Program nie je veľmi otvorený k rozšíreniu, najmä kvôli grafickému rozhraniu. Je to však možné, špecifické časti programu viazané na veľkosť 3x3 by sa rozumne dobre dali prepísať na nezávislé od veľkosti, vstup by mohol byť zmenený zo znakov **12345678e** na **0123456789abcdefgh…z** – 36 znakov, čo by sprístupnilo väčšinu veľkostí hlavolamu.

Program využíva celkom hojný počet globálnych premenných, väčšina z nich sa týka GUI, tie by sa dali pre väčšiu prehľadnosť "upratať" do classy alebo štruktúry. Zvyšné by sa dali posúvať funkciami ako argumenty, avšak som to považoval za nepotrebné.