FAKULTA ústav
STROJNÍHO automatizace
INŽENÝRSTVÍ a informatiky

8 - puzzle

technická zpráva k řešiči 8 - puzzlu

18. 5. 2024

David Bystroň, 228676

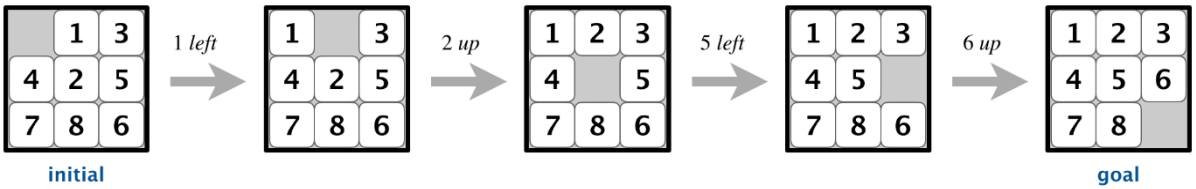
Úvod

Tento projekt se zabývá řešením 8 – puzzlu, v České republice taky známého jako „Lišák“. K jeho řešení je použita informovaná metoda prohledávání, konkrétně algoritmus A*. Technická zpráva obsahuje jak teoretický popis problému, tak řešení. Nakonec je popsán program s kompletní implementací problému, který byl napsán v programovacím jazyce python.

8 – puzzle

8 – puzzle je typická logická hra skládající se z mřížky o rozměrech 3x3, která obsahuje osm očíslovaných polí a jedno prázdné pole. Mřížka může mít samozřejmě i větší rozměry, nejtypičtější je 15 – puzzle (4x4).

Cílem hry je seřadit „dlaždice“ do správného pořadí pomocí přesouvání jednotlivých polí na prázdné místo. Konec hry nastává, když se nám podaří dosáhnout cílové konfigurace, ve které jsou všechny pole seřazeny vzestupně od 1 do 8 a prázdné místo je v pravém horním rohu. Cílová konfigurace však může být čistě na hráči. Například cílem může být vzestupné pořadí čísel po obvodu a prázdné místo uprostřed.



Obr.1: 8 – puzzle

Kromě rozměrových variací existují i grafické varianty, kde je cílem poskládat z polí obrázek.

A* algoritmus

Informované metody prohledávání

Ačkoliv by byl problém řešitelný i pomocí neinformovaných metod, informované metody pracuji mnohem efektivněji. Tyto metody pomocí hodnotící funkce definují výhodnost výběru dalšího stavu.

Hodnotící funkce má tvar

$$f(n) = h(n) + g(n),$$

kde h(n) je odhad ceny cesty do cílového stavu, g(n) je cena cesty z počátečního stavu do aktuálního.

Čím nižší hodnotu f(n) stav má, tím je výhodnější pro expanzi.

Funkce h(n) je odhadem a metodu jejího výpočtu určuje člověk na základě nějakých zkušeností. Správná volba heuristické funkce přímo ovlivňuje efektivnost řešení.

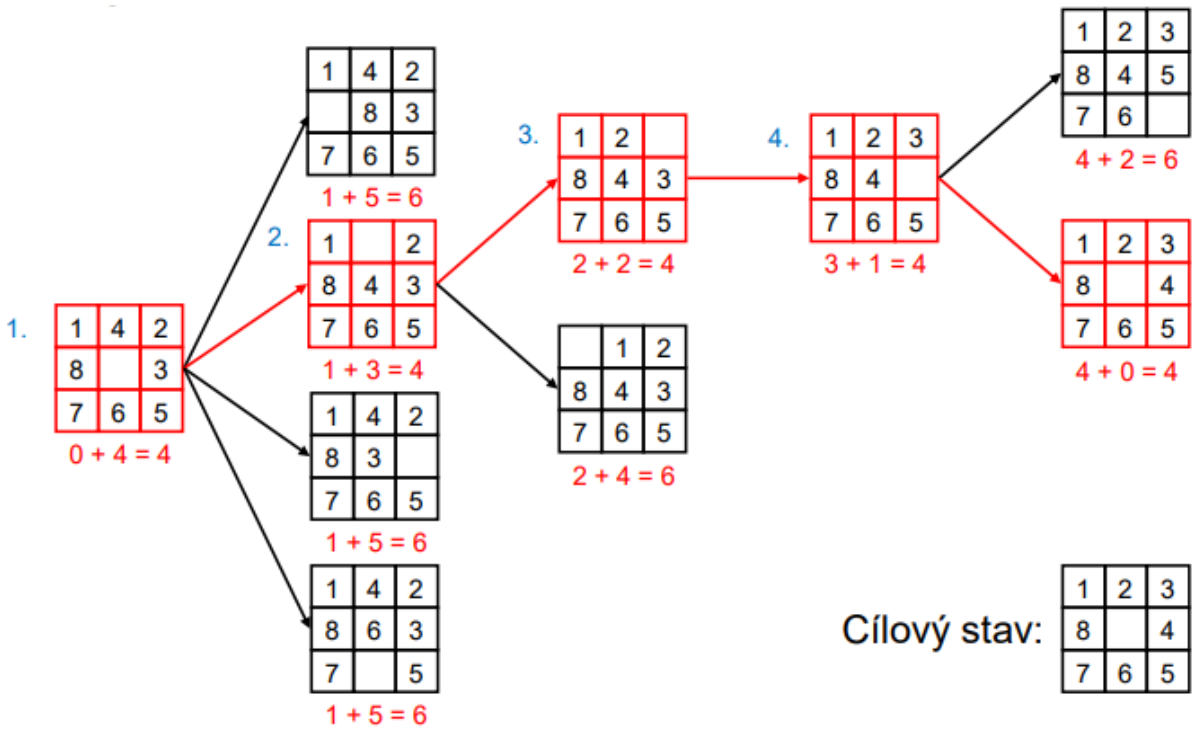
A*

Algoritmus A* (A – star) je efektivní algoritmus pro vyhledávání cest a řešení problémů v grafech. Jedná se vlastně o algoritmus best – first search s mírnou modifikací.

V úloze pro řešení 8 – puzzlu je nejlepší volba heuristické funkce h(n), součet Manhattanských vzdáleností všech polí od jejich cílové pozice. Manhattanská vzdálenost je hojně užívaná v úlohách s mřížkou, protože odpovídá počtu kroků v horizontálním i vertikálním směru. V prostředí, kde je pohyb možný ve všech směrech má největší uplatnění Euklidovská vzdálenost.

Funkce g(n) je pak počet tahů, který byl potřeba od počátečního stavu do aktuálního stavu.

Algoritmus A* začíná tím, že umístí počáteční uzel na otevřený seznam OPEN, který obsahuje všechny uzly, které jsou připraveny k prohledání. Uzly z otevřeného seznamu jsou vybírány na základě nejnižší hodnoty f(n). Jakmile je uzel vybrán, je přesunut na uzavřený seznam CLOSED, který obsahuje uzly, které byly již prozkoumány. Pro každý sousední uzel aktuálního uzlu se vypočítá jeho hodnota g(n), h(n) a f(n). Pokud sousední uzel není na otevřeném ani uzavřeném seznamu, je přidán na otevřený seznam. Pokud je již na otevřeném seznamu s vyšší hodnotou f(n), je jeho hodnota aktualizována a uzel je znovu zařazen do prohledávání. Proces se opakuje, dokud není nalezen cílový uzel nebo jsou vyčerpány všechny možnosti.



Obr. 2: Řešení 8 – puzzlu pomocí A*

David Bystroň, 228676

Python řešič

Pro řešení 8 – puzzlu byl vytvořen program napsaný v programovacím jazyce python. Výstup programu je pak konzolová aplikace.

Celý kód je psán objektově. Nejprve byla vytvořena třída uzlu *Node*, která obsahuje vždy daný stav, jeho předchůdce a hodnotící funkci.

```
class Node:

def __init__(self, state, prev=None, g=0, h=0):

        self.state = state

        self.prev = prev

        self.g = g

        self.h = h

        self.f = 0
```

Hlavní část kódu pak tvoří třída *Puzzle*. Při vytváření musíme zadat počáteční a cílový stav formou matice.

```
initial_state = np.array([[1, 3, 6], [5, 8, 2], [0, 7, 4]])

gl_state = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 0]])

solver = Puzzle(initial_state, gl_state)
```

V této třídě pak můžeme najít metodu *possible_moves*, která zjistí, jaké možné posuny můžeme ve hře udělat, díky pozici 0, která reprezentuje prázdné pole ve hlavolamu. Výstupem je pak list možných posunů.

Metoda *move* pak vytvoří nový stav na základě možného pohybu.

Heuristika v podobě součtu všech Manhattanských vzdáleností od polí od cíle je spočítána pomocí metody *distance.*

```
def distance(self, state):
distance = 0
        for i in range(3):
                for j in range(3):
                        value = state[i][j]
                        # 0 is empty position
                        if value != 0:
                                goal_position = np.where(self.goal == value)  # get correct position
                                distance += abs(goal_position[0] - i) + abs(goal_position[1] - j)
        return distance
```

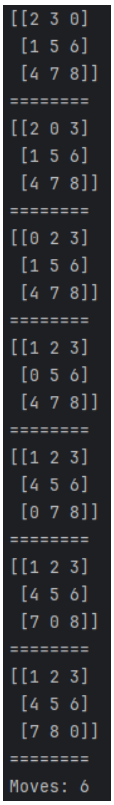
Kostrou celého algoritmu je pak *a_star*, jehož kód je obdobou popisu A*, viz výše.

```
def a_star(self):
        open = []
        closed = []
        first_node = Node(self.start, h = self.distance(self.start))
        first_node.f = first_node.g + first_node.h
        open.append(first_node)
        while len(open):
                open.sort(key = lambda node: node.f)
                current = open.pop(0)
                if np.array_equal(current.state, self.goal):
                        return current
                closed.append(current)
                moves = self.possible_moves(current.state)
                for move in moves:
                        new = self.move(current.state, move)
                        state_in_closed = any(np.array_equal(new, node.state) for node in closed)
                        if not state_in_closed:
                                next_node = Node(new, current, g = current.g + 1, h = self.distance(new))
                                next_node.f = next_node.g + next_node.h
                                open.append(next_node)
    return None
```

Závěr

Tato práce se zabývala řešením 8 – puzzle, pomocí A* algoritmu. V této zprávě je pak teoretický popis problematiky a vysvětlení informovaného prohledávání pomocí A*.

Výsledkem je pak program v pythonu, kde po zadání počátečního stavu a cílového stavu se v konzoli zobrazí jednotlivé stavy vedoucí k řešení a počet nutných tahů k dosáhnutí cíle.



Obr.3: Výstup programu v konzoli

Zdroje

[1] Informované heuristické programování. Online. Matematikabiologie. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--prohledavani-stavoveho-prostoru--metody-prohledavani--informovane-heuristicke-prohledavani. [cit. 2024-05-19].

*[2] A**. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2001-. Dostupné z: https://cs.wikipedia.org/wiki/A*. [cit. 2024-05-19].

[3] 8 puzzle. Dostupné také z: https://www.cs.princeton.edu/courses/archive/spring18/cos226/assignments/8puzzle/index.html. [cit. 2024-05-19].

[4] DVOŘÁK, Jiří a BŘEZINA, Tomáš. *Prohledávání stavového prostoru - informované metody*.