Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Umelá inteligencia

Prehľadávanie stavového priestoru

Artúr Kozubov

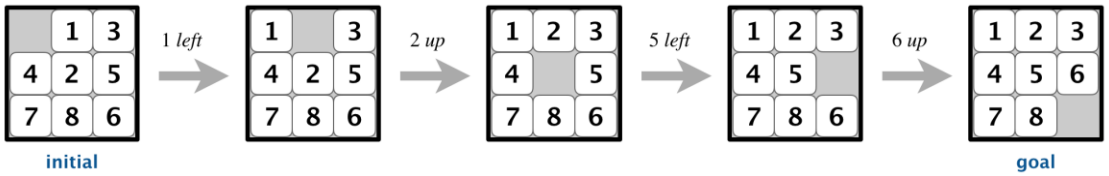
Meno cvičiaceho: Ing. Ivan Kapustík

Čas cvičení: Št 18:00

Dátum vytvorenia: 09. 10. 2023

Zadanie úlohy *(e) Problém 2. A* algoritmus*

Našou úlohou je nájsť riešenie 8-hlavolamu. Hlavolam je zložený z 8 očíslovaných políčok a jedného prázdneho miesta. Políčka je možné presúvať hore, dole, vľavo alebo vpravo, ale len ak je tým smerom medzera. Je vždy daná nejaká východisková a nejaká cieľová pozícia a je potrebné nájsť postupnosť krokov, ktoré vedú z jednej pozície do druhej.



Implementačné prostredie

Program je vytvorenej v `Python 3.10.11` a na správne fungovanie sa využíva knižnica `time` a `deepcopy`.

Priebeh programu

Po spustenie programu bude možnosť zadať rozmery stavov:

```
1. > Enter a state sizes, like m n, or ENTER if default:  3 3
2.
```

Ho rozmery:

```
1. > Enter a initial state of the puzzle 1-8, and 0 as an empty one:
2. 5 1 0 2 4 6 3 7 8
```

Napísať cieľový stav, alebo vybrať štandardne:

```
1. > Enter a goal state of the puzzle 1-8, and 0 as an empty one, or ENTER if default:
2. "ENTER"
```

A vybrať method pre kalkuláciu heuristicku hodnotu:

```
1. > Select method for calculating a Heuristic score:
2. 2
```

Tu su par možnosti kalkulácii tejto hodnoty:

1. (Hamming distance) Počet políčok, ktoré nie sú na svojom mieste.
2. (Manhattan distance) Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície.

Po zadanie hodnôt, program bude vypisovať aktuálni hodnoty, ako:

- Počet všetkých vytvorených uzlov.
- Aktuálna hĺbka (najlepšieho uzla).
- Skóre (kde -> 0 je ciel).
- Čas spustenia.

```
1. > Nodes: 4215 | Depth: 28 | Score: 0 | Time: 1.8902 secc
```

A po ukončeniu aj prehodí, ako:

```
1. > left left down right up left down down right up left down right right up left up right down
down left left up up right down down right
```

Príklad priebehu programu:

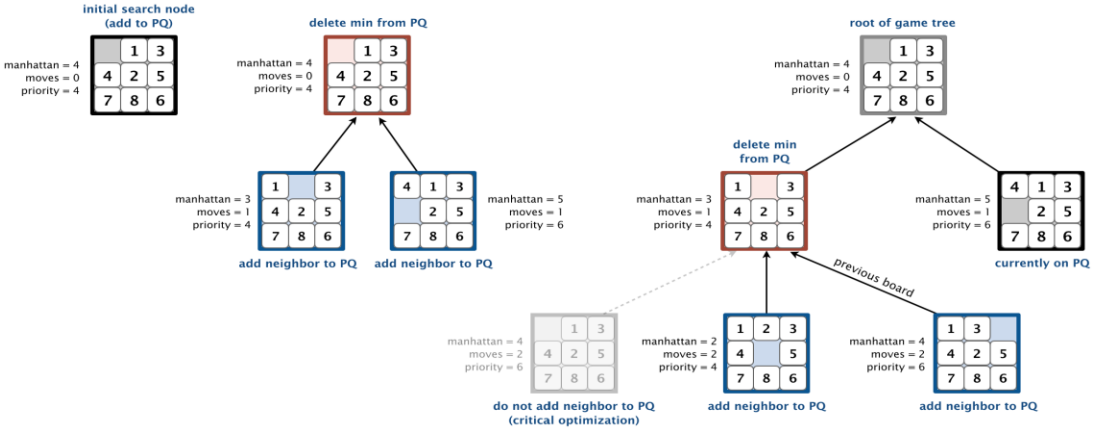
```
1. ( venv) justAnArthur on { V:/justAnArthur/university/UI/assignments/01 8-puzzle AI solver  main
2.  # python main.py
3.  Enter a state sizes, like m n, or ENTER if default:
4.  3 3
5.  Enter a initial state of the puzzle 1-8, and 0 as an empty one:
6.  5 1 0 2 4 6 3 7 8
7.  Enter a goal state of the puzzle 1-8, and 0 as an empty one, or ENTER if default:
8.
9.  Select method for calculating a Heuristic score:
10. 2
11. Nodes: 4215 | Depth: 28 | Score: 0 | Time: 1.6266 sec
12. left left down right up left down down right up left down right right up left up right down down
left left up up right down down right
```

Algoritmus

Po zadaní všetkých dat, program spravi začiatočný uzol a začne pokračovať v bezkonečnom cykle pokiaľ nenájde riešenie (pokiaľ heuristicka hodnota nebude `0`).

Cykle

1. Nájsť uzol z najlepšou (najmenšou) heuristickoj hodnotou (ak takých uzlov je vela zobrať najstaršej).
    1. Ak rovná sa `0` - tak skonči cyklus.
    2. Ak počet uzlov bude rovná sa null - tak program výpise, ze taky stav nemožno vyriešiť.
2. Generovať jeho (4) potomkov.
    1. Nájsť pre každú z nich ich heuristicku hodnoty.



Algoritmus pre heuristicku hodnotu

Program poskytuje logiku pre dve metódy výpočtu tejto hodnoty:

Hamming distance (Počet políčok, ktoré nie sú na svojom mieste)

> Hammingová vzdialenosť je pojem z oblasti informačnej teórie a používa sa na meranie rozdielov medzi dvoma reťazcami rovnakej dĺžky. Nazýva sa podľa Richarda Hamminga, ktorý prvýkrát definoval tento pojem. Hammingová vzdialenosť medzi dvoma reťazcami je rovná počtu pozícií, na ktorých sa reťazce líšia. Napríklad, Hammingová vzdialenosť medzi slovami „karban“ a „kanban“ je 1, pretože sa líšia iba na jednom mieste.

Výhody:

- Je jednoduchá na výpočet.
- Je ľahko pochopiteľná, pretože sa jednoducho porovnávajú pozície dlaždíc.

Nevýhody:

- Je menej presná než Manhattanova vzdialenosť, pretože ignoruje skutočnú vzdialenosť, ktorú musia dlaždice prejsť, aby dosiahli cieľový stav.

Tu je môj implementovaný kód:

```
1. score = 0  # 0 is goal (filtered by min)
2.
3. for i, row in enumerate(state):
```

```
4.    for j, element in enumerate(row):
5.        if state[i][j] != goal_state[i][j]:
6.            score += 1
```

Manhattan distance (Súčet vzdialeností jednotlivých políčok od ich cieľovej pozície)

> Manhattanská vzdialenosť, niekedy nazývaná aj taxikárska metrika, je spôsob merania vzdialenosti medzi dvoma bodmi v mriežkovom systéme. Je to celkový počet krokov po horizontálnych a vertikálnych trasách medzi dvoma bodmi, bez akýchkoľvek diagonálnych pohybov.

Výhody:

- Poskytuje presnejší odhad vzdialenosti ku cieľovému stavu než Hammingova vzdialenosť.
- Môže poskytnúť rýchlejšie riešenie, pretože lepšie určuje, ktoré stavy sú bližšie k cieľu.

Nevýhody:

- Je zložitejší na výpočet než Hammingova vzdialenosť.
- Môže byť menej efektívna pri hľadaní riešení, ktoré vyžadujú veľa krokov, pretože výpočet Manhattanovej vzdialenosti môže byť časovo náročný.

Tu je môj implementovaný kód:

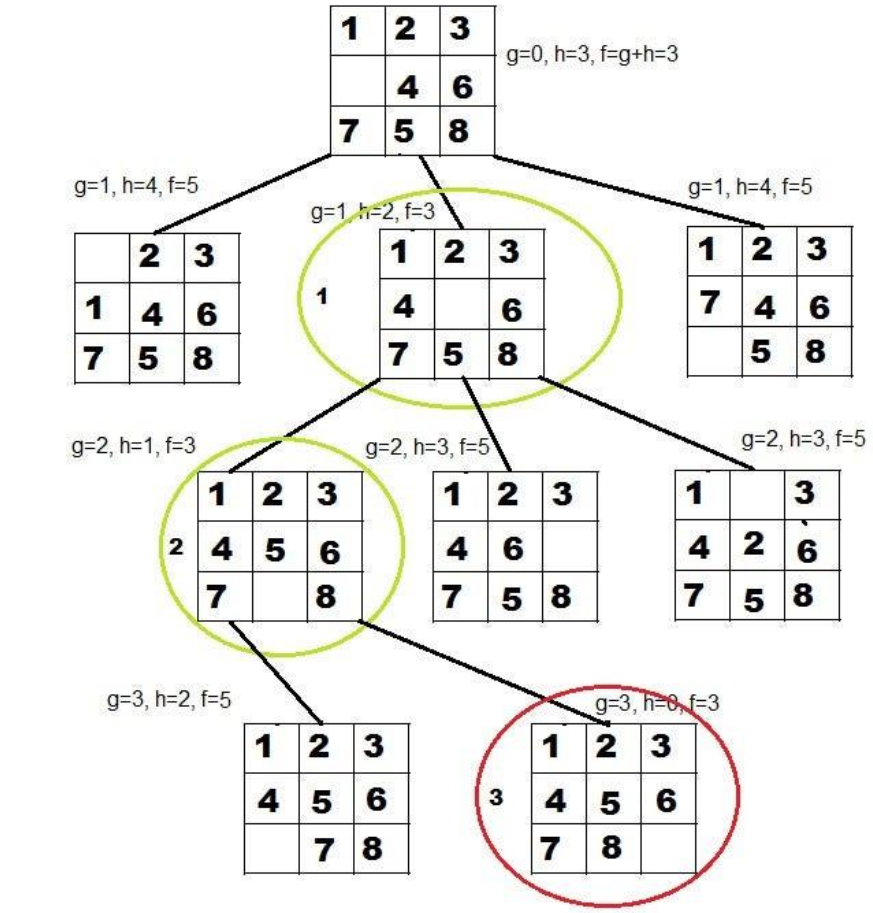
```
1. score = 0  # 0 is goal (filtered by min)
2.
3. for i, row in enumerate(state):
4.    for j, element in enumerate(row):
5.        x, y = find_position(goal_state, element)
6.        score += abs(i - x)
7.        score += abs(j - y)
```

Zhodnotenie riešenia

Možnosti rozšírenia

Rôzne spôsoby rozšírenia algoritmu A* v hre s 8 puzzle môžu zahŕňať:

- **Heuristika**: Je možne použiť rôzne heuristické funkcie a porovnať ich výkonnosť. Môžete napríklad porovnať Manhattanovu vzdialenosť a vzdialenosť blokov.
- **Optimalizácia**: Môžete vyskúšať rôzne stratégie na optimalizáciu algoritmu, napríklad použitie prioritného frontu na udržiavanie otvorených uzlov alebo implementáciu rôznych stratégií na riadenie veľkosti pamäte.
- **Vizualizácia**: Implementácia vizualizácie, ktorá ukazuje, ako algoritmus prechádza hrou, môže byť účinným nástrojom na pochopenie toho, ako A* funguje. Vizualizácia môže pomôcť aj pri ladení algoritmu.



–

- **Paralelizácia**: A* sa dobre hodí na paralelizáciu, pretože mnohé výpočty možno vykonávať nezávisle. Ak máte k dispozícii viacero procesorov alebo plánujete používať cloudové výpočty, môžete preskúmať, ako paralelizovať A*.

Testovanie

Otestoval som tie dve metody z kalkulácie Heuristic hodnoty:

Hodnota	Hamming nodes/depth/time	Manhattan nodes/depth/time	Rozdiel v čase (1/2)
6 3 0 4 2 8 7 5 1	3478/28/0.7038	1515/28/0.3129	2.2493
1 8 0 2 5 3 6 7 4	4670/28/1.2428	4145/28/0.9183	1.3534
4 5 0 2 6 8 1 3 7	28877/28/28.5796	2717/30/0.4698	60.8335
3 2 5 7 1 8 0 4 6	2705/30/0.4506	12482/22/7.6542	0.0588
6 3 2 5 8 7 4 1 0	69479/29/192.6848	3913/30/0.9281	207.6121

Výsledky ukazujú, že druhý spôsob výpočtu (Manhattan) je Oveľa lepší, ako prvý. Ale aj tak niekedy tento spôsob prehráva.