Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 19 Bratislava 4

umelá inteligencia

zadanie 2

Peter Plevko

2020/2021

Definovanie problému 2

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.

Príkladom môže byť nasledovná začiatočná a koncová pozícia:

Začiatok:

1	2	3
4	5	6
7	8	

Koniec:

1	2	3
4	6	8
7	5	

Im zodpovedajúca postupnosť krokov je: **VPRAVO, DOLE, VĽAVO, HORE**.

c)

Problém 2. Použite algoritmus obojsmerného hľadania.

Opis riešenia a použitý algoritmus

Na vypracovanie tohto zadania som využil obojsmerne prehľadávanie do šírky. Na prehľadávanie do šírky som využil rad(queue). Teda údaje ktoré prvé zapíšem, následne aj ako prvé precitám a spracujem (fifo)

1. Zo súboru graf.txt načítam zadaný 3x3 puzzle a uložím ho do 2D poľa. Následne si vytvorím štruktúru vertex do ktorej načítam toto 2D pole, x a y pozíciu nuly.
2. V tomto momente mi začína prehľadávanie najprv prehľadávam od počiatočného stavu. Pozerám do ktorého smeru sa môže moja nula posunúť mam len 4 smery. Keď zistím že sa môžem posunúť napríklad do lava, vytvorím novy 2d array s posunutím prejdem cele pole visited ci už náhodou mi takáto situácia nenastala ak áno nerobím nič ak je to nová situácia vytvorím objekt nastavím mu nove súradnice nuly nastavím mu predchodcu a jeho rozpoloženie čísel po vykonaní pohybu, a smer do ktorého som išiel. Vložím ho do queue. Následne vytvorím taký istý objekt ale s rozpoložením čísel pred pohybom a vložím ho do poľa navštívených.
3. To iste urobím aj keď prehľadávam do konca
4. Zoberiem nultý prvok z start_queue a uložím ho do premennej start_current, následne zoberiem nultý prvok z end_queue a uložím ho do end_current
5. Prejdem cele pole end_visited a porovnávam ho s mojim start_current ak nájdem že sa tam nachádza môžem skončiť pretože viem že sa s mojej štartovnej pozície viem dostať do koncovej pozície.
6. Algoritmus na hľadanie mi v tejto chvíli konci a už iba vypíšem cestu od štartovného bodu do koncového bodu pomocou objektov a ich predecesorov

Reprezentácia údajov

Stav

Vstupom algoritmu je začiatočný a koncový stav, ten je zadaný v tvare objektu.

3x3 puzzle mam reprezentovanú 2D polom a prázdne políčko mam reprezentovane nulou.

Objekt: Vertex

Moves: aký pohyb som vykonal aby som sa z minulého stavu dostal do toho terajšieho

Predecesor: obsahuje objekt respektíve minulý stav

Stage: obsahuje 2D pole a momentálne rozpoloženie čísel

X : obsahuje x-ovu súradnicu nuly

Y: obsahuje y-ovu súradnicu nuly

Operátory sú len štyri right, left, up, down

1 2 3 1 2 3

4 5 6 left 4 5 6

7 8 0 7 0 8

Užívateľské prostredie

Výpis na terminál vyzerá nasledovne

```
pocet navstivenych: 16
pocet spracovanych: 32
cas vykonania algoritmu: 0.000997304916381836
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
left
[1, 2, 3]
[4, 5, 6]
[7, 0, 8]
up
[1, 2, 3]
[4, 0, 6]
[7, 5, 8]
right
[1, 2, 3]
[4, 6, 0]
[7, 5, 8]
down
[1, 2, 3]
[4, 6, 8]
[7, 5, 0]

Process finished with exit code 0
```

Testovanie

V mojom programe označujem kam ma ísť prázdne políčko to znamená keď mam right znamená že prázdne políčko sa ma vymeniť s číslom napravo od neho.

Vstup zo súboru

Príklad vzoru vstupného súboru:

Test c1:

Počiatočný stav: 0 1 2 koncový stav: 8 0 6

3 4 5 5 4 7

6 7 8 2 3 1

Výstup programu je: right right down left left up right right down downleft left up right down left up up right down right up left downright down left left up up right

Výstup aj s výpisom nájdeme v súbore vystupc1

Čas vykonania programu: 0.398327112197876

Test c2:

Počiatočný stav: 1 0 3 koncový stav: 1 2 3

4 8 6 4 5 6

2 5 7 7 8 0

Výstup programu: right, down, left, down, left, up, right, down, right, up, up, left, down, left down, right, right

Výstup aj s výpisom nájdeme v súbore vystupc2

Čas vykonania programu: `0.009972572326660156`

Test c3:

Počiatočný stav: 5 6 7 koncový stav: 1 2 3

4 0 8 8 0 4

3 2 1 7 6 5

Výstup programu je: left, up, right, right, down, down, left, left, up. up, right, right, down, down, left, left, up, up, right, right, down, down, left, left, up, up, right, right, down, left

Výstup aj s výpisom nájdeme v súbore vystupc3

Čas vykonania programu: `0.009972572326660156`

Test c4:

Počiatočný stav: 2 8 1 koncový stav: 1 2 3

4 6 3 8 0 4

0 7 5 7 6 5

Výstup programu: right, up, left, up, right, down, left, left, up, right, down

Výstup aj s výpisom nájdeme v súbore vystupc4

Čas vykonania programu `0.006981372833251953`

sem urob test nesplniteľného

Test c5:

Nevy riešiteľný vstup

Počiatočný stav: 1 2 3 koncový stav: 7 8 6

4 5 6 5 4 3

7 8 0 2 01

Výstup programu: tato vec nie je vy riešiteľná

Svoj program som testoval ručne na otestovanie som použil príklady ktoré boli na stránke zadania 2 a následne som si to skúšal kontrolovať na tejto stránke https://murhafsousli.github.io/8puzzle/#/

Tato stránka generuje 8 puzzle a cieľom je ho vyriešiť takže som dal do môjho programu očíslovanie z tejto stránky a koncovi stav a skúšal som ci to vyriešim.

Následne som to kontroloval pomocou tejto stránky

http://tristanpenman.com/demos/n-puzzle/

Zhodnotenie

Tento projekt by som nazval ako úspešný riešenie problému je dostatočne rýchle a podľa mojich doterajších testov je aj správne. Možne vylepšenie môjho programu prerobiť že to nebude riešiť iba 3x3 puzzle ale aj rôzne kombinácie napríklad m*n puzzle.

prehľadávania do šírky vs do hĺbky:

Prehľadávanie do šírky sa síce vykonáva dlhšie a ma viac stavov ako do hĺbky ale zato nájde najkratšie riešenie.

Prehľadávanie do hĺbky zase síce nenájde najoptimálnejšie riešenie avšak nájde toto riešenie oveľa rýchlejšie a s menším využitím pamäte.

Výhody obojsmerného prehľadávania sú že už tak efektívne algoritmy prehľadávania ešte zefektívni a skráti ich dobu vykonávania o polovicu čo je logické pretože vykonávam hľadanie z dvoch bodov a keď sa mi niekde pretnú viem že sa zo začiatočného stavu môžem dostáť do stavu koncového. Takže časovú komplexitu O(b^d) skráti na O(b^d/2 + b^d/2). Takéto obojsmerne prehľadávanie je časovo efektívnejšie a aj pamäťovo efektívnejšie pretože si nemusím pamätať toľko stavov.

Samozrejme nezáleží iba na algoritme ale aj na programátorovi preto moja verzia obojsmerného prehľadávania do šírky nie je úplne dokonala, dala by sa aj vylepšiť.

Závislosť na programovacom prostredí:

Keďže python je vysokoúrovňoví programovací jazyk tak je jasne že tento program by rýchlejšie zbehol v tých nižších napríklad c ale v nom by som si musel ostatne veci implementovať sám.