Dokumentácia druhého zadania z UI

Adam Štuller

March 16, 2019

1 Riešený problém – Zadanie

Zadanie, ktoré nasleduje je priamo okopírované zo stránky predmetu. Poznámka 1, o ktorej sa hovorí v zadaní g sa týka limitu, ktorý je možno zadať na vstup programu a po uplynutí, ktorého sa program skončí neúspechom. Je to z toho dôvodu, že pre niektoré štartovacie body neexistuje riešenie.

1.1 Eulerov kôň

Úlohou je prejsť šachovnicu legálnymi ťahmi šachového koňa tak, aby každé políčko šachovnice bolo prejdené (navštívené) práve raz. Riešenie treba navrhnúť tak, aby bolo možné problém riešiť pre štvorcové šachovnice rôznych veľkostí (minimálne od veľkosti 5 x 5 do 20 x 20) a aby cestu po šachovnici bolo možné začať na ľubovoľnom východziom políčku.

1.2 Zadanie g

Pre riešenie problému Eulerovho koňa existuje veľmi dobrá a pritom jednoduchá heuristika, skúste na ňu prísť sami. Ak sa vám to do týždňa nepodarí, pohľadajte na dostupných informačných zdrojoch heuristiku (z roku 1823!), prípadne konzultujte na najbližšom cvičení cvičiaceho. Implementujte túto heuristiku do algoritmu prehľadávania stromu do hĺbky a pre šachovnicu 8x8 nájdite pre 10 rôznych východzích bodov jedno (prvé) správne riešenie (pre každý východzí bod). Algoritmus s heuristikou treba navrhnúť a implementovať tak, aby bol spustiteľný aj pre šachovnice iných rozmerov než 8 x 8. Treba pritom zohľadniť upozornenie v Poznámke 1. Je preto odporúčané otestovať implementovaný algoritmus aj na šachovnici rozmerov 7x7, 9x9, prípadne 20x20 (máme úspešne odskúšaný aj rozmer 255x255) a prípadné zistené rozdiely v úspešnosti heuristiky analyzovať a diskutovať.

2 Stručný Opis Riešenia

Tento problém som riešil pomocou prehľadávacieho algoritmu a to konkrétne prehľadávania do hĺbky. Vytvoril som si vlastnú reprezentáciu stavov a potom

som začal prehľadávanie v začiatočnom bode a skončil som ho keď som narazil na koncový stav.

Celý čas som si udržoval a zväčšoval slovník predchodcov kam som pre každý preskúmaný stav uložený ako kľúč uložil ako hodnotu jeho predchodcu. Tento slovník predchodcov som po úspešnom ukončení prehľadávania použil na zostrojenie pôvodnej cesty od začiatočného do koncového políčka na šachovnici.

Program je rozdelený do troch zdrojových súborov, chessboard.py, node.py a config.py. Prvé dve obsahujú rovnomenné triedy a posledný obsahuje konfiguráciu programu, kde sa dá ovplyvniť správanie programu. Width a height reprezentujú rozmery šachovnice, limit je pre vyhľadávanie. Ďalej sa tam dá nastaviť, či sa použije heuristika, či sa bude vykresľovať šachovnica a či sa každý výsledok podrobí testu.

3 Reprezentácia Údajov

Na reprezentáciu údajov som použil dve triedy, Chessboard pre šachovnicu a Node pre uzol v grafe, ktorý prehľadávam.

3.1 Šachovnica

Šachovnicu ako takú som reprezentoval iba jej rozmermi, pri jej inicializácii boli parametre x-ový a y-ový rozmer a limit, koľko krát mal hlavný cyklus prehľadávania do šírky prebehnúť pred tým ako sa skončil neúspešne.

S každým políčkom šachovnice som pracoval ako s číslom od 0 do $(x * y - 1)$. V triede šachovnica som mal potom metódy na prevod tohto poradového čísla alebo pozície na x-ovú a y-ovú zložku.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Figure 1: Šachovnica rozmeru 5x5

V šachovnici boli ako atribúty uložené aj pomocné dátové štruktúry na prehľadávanie. Konkrétne to boli, už spomínaný slovník predchodcov a rad na prehľadávanie, implementovaný ako jednoduchý pythonovský zoznam. Do radu som si zasa ukladal novo objavené uzly v každej iterácií. Z neho som tiež na začiatku každej iterácií prehľadávania vybral posledne vložený uzol a ten potom prehľadal. Takto som urobil vyhľadávanie do hĺbky, na vyhľadávanie do

šírky by som urobil to isté, iba by som vyberal najdávnejšie vložené uzly (prvé zľava).

3.2 Uzol

Uzol som reprezentoval dvomi informáciami, stavom a pozíciou. Pozícia je iba poradové číslo políčka, ktoré reprezentuje konkrétny uzol na šachovnici (pre jedno políčko existuje $2(x * y)$ uzlov).

Stav je informácia o tom, ktoré pozície už boli pred touto navštívené. Stav bol reprezentovaný ako číslo o veľkosti x*y bitov. Pre každú pozíciu, ktorá bola uz navštívená v tomto stave bola na prislúchajúcom bite stavu 1 a na ešte nenavštívenej pozícií zasa 0. Z toho vyplýva, že koncový stav, ktorý som chcel dosiahnuť bolo číslo ktoré malo na každej pozícií 1.

Trieda Node mala v sebe aj metódy na prácu s týmto stavom a pozíciou. Dokopy, tieto dve informácie predstavovali identifikačné číslo uzlu. Toto čislo som si napríklad pamätal aj v slovníku predchodcov, aby som si nemusel pamätať celý objekt.

Implementoval som aj metódu na zistenie či v konkrétnom stave už bola nejaká pozícia navštívená (pozrel som sa na prislúchajúci bit). Túto metódu som potom využíval pri vyhľadávaní aby som sa vyhol zacykleniu.

4 Algoritmus

Algoritmus, ktorý som použil bolo teda prehľadávanie do hĺbky. Tento algoritmus je prakticky celý v metóde depth_first_search. Pracuje v cykle, pričom v každej iterácií vyberie naposledy pridaný uzol do radu a ten preskúma pomocou samostatnej metodý find_neighbors. Tato metóda vráti pole všetkých možných uzlov, ktoré môžem navštíviť. Podľa toho, či je v configu nastavená heuristika je potom toto pole susedov utriedené a postupne vložené do radu tak, že uzol, ktorý má byť navštívený ako ďalší je vložený ako posledný. V prípade že na začiatku iterácia podmienka zistí že už sme v koncovom stave, cyklus sa ukončí a prehľadávanie vráti poradie navštívených pozícií, ktoré vypľuje metóda backtrace.

4.1 Hľadanie susedov

Susedov nejakého uzla hľadáme tak, že sa postupne na terajšiu pozíciu aplikujem všetkých osem operátorov (možných skokov koňa) a zistím, či novovzniknuté políčko nie je mimo šachovnice alebo či už v terajšom stave nebolo navštívené. Ak nie, vytvorím nový uzol s novou pozíciou a a upraveným stavom a pridám ho do pola, ktoré potom vrátim vyhľadávaniu.

4.2 Heuristika

Používam Warnsdorfove pravidlo, čo je heuristika z roku 1823. Táto heuristika hovorí, že najbližšie by mal byť prehľadaný uzol, ktorý má najmenej susedov. V mojej implementácií je to tak, že keď vrátim pole susedov a spustím na nich heuristiku, ona pre každého z týchto susedov najde ich vlastných susedov a zoradí ich podľa počtu susedných uzlov. Zoradený sú tak, že sused s najmenším počtom susedov je v novom zozname ako posledný a teda ako posledný bude aj pridaný do radu na prehľadávanie.

4.3 Operátory

Operátorov na pohyb koňa po šachovnici je osem a sú to tieto: (-1 , -2) , (-2 , -1) , (-2 , 1) , (-1 , 2) , (1 , 2) , (2 , 1) , (2 , -1) , (1 , -2). Čísla znamenajú o koľko sa má kôň pohnúť v x-ovom respektíve y-ovom smere aby prišiel na novú pozíciu.

5 Testovanie

Testovanie som riešil dvoma spôsobmi. Manuálne a automatizovane. Manuálne tak že si pri každom správnom výsledku nakreslím aj prislúchajúcu šachovnicu a potom výsledná cesta dá veľmi jednoducho skontrolovať. Automaticky som to kontroloval pomocou testovacej metódy test_tour, ktorá po najdení každej cesty skontroluje či je to validná eulerovska cesta.

6 Zhodnotenie Riešenia

V zadaní bolo, že pre šachovnicu veľkosti 8x8 máme najsť prvých 10 riešení. Toto riešenie som nepripojil ako samostatny textový súbor, pretože program je implementovaný všeobecne, nie len pre určité rozmery a je veľmi jednoduché ho nastaviť na konkrétne rozmery (v konfiguráku) a spustiť. Pre šachovnicu veľkosti 8x8 dokonca nájde 13 riešení (pre každé párne políčko ako štartovacie políčko). Rýchlosť a úspešnosť programu veľmi závisí od toho ako funguje heuristika. Ak zaúčinkuje a nájde cestu na prvý krát algoritmus je veľmi rýchly. Napríklad pri rozmere 8x8 to pravdepodobne urobí vždy. Pri rozmeroch 7x7 je to o dosť pomalšie a v niektorých prípadoch treba nastaviť väčší limit aby sa cesta vôbec našla. Bez heuristiky je algoritmu oveľa pomalší. Program som testoval pre rozmery 4x4 až 200x200 a viem, že pre tieto rozmery funguje.

Pamäťovo je program efektívny, dlhodobo si pamätám zoznam predchodcov, kde mam uložené iba čísla - je ich tam toľko koľko uzlov som prehľadal. Ani rad v ktorom mam uložené uzly na prehľadávanie nie je príliš veľký, každý prehľadaný uzol z neho vyhodím.

Vedel by som si predstaviť že algoritmus ešte zefektívnim, minimálne po pamäťovej stránke. V rade by som si napríklad nemusel vôbec pamätať všetky

tie údaje. V prípade že by som sa s riešením viacej hral tak by som to upravil tak, že aj v rade by boli iba identifikačné čísla uzlov.