Zadanie 2 – Problém 2 d) – Adam Benovič

Opis riešeného problému:

Problém sa nazýva 8 puzzle alebo v našom prípade aj N puzzle. Ide o nájdenie postupnosti krokov zo začiatočného stavu do koncového stavu. Stavy sú reprezentácia pozícii políčok a vyzerajú takto: ((1 2 3)(4 5 6)(7 8 m)), ((7 8 6)(5 4 3)(2 m 1)). Inými slovami, cieľom je dostať všetky políčka zo začiatočných pozícii na cieľové pozície. Realizuje sa to pomocou 4 operátorov: hore, vľavo, vpravo, dole. Zo zadania d) vyplýva použitie algoritmu lačného hľadania, ktorý je informovaným vyhľadávaním. Informáciou je heuristická hodnota stavu. Použil som 2 zadané heuristiky - počet políčok, ktoré nie sú na svojom mieste a súčet vzdialeností políčok od cieľovej pozície. Zadanie d) tiež hovorí o porovnaní výsledkov medzi použitými heuristikami, čo som urobil manuálne – viac v časti Testovanie

Opis riešenia problému:

Problém som riešil v jazyku Java verzie 8. Spúšťal som na i5 6600K@4GHz. Okrem reprezentácie stavu neopisujem použité reprezentácie keďže je to skôr vecou použitého implementačného prostredia ako samotného riešenia.
Začiatkom je načítanie stavov zo súborov – je možné zadať viacero začiatočných aj koncových stavov naraz.

Stavy sú reprezentované triedou Puzzle, ktorá si v parametroch pamätá súčasný stav políčok v 2 rozmernom poli čísiel(int mapa), rozmery mapy(výška a šírka), pozícia prázdneho políčka(naše m, ktoré sa zmenilo na 0) odkaz na predchodcu, operátor, ktorým stav vznikol, cenu do cieľa(vypočítaná pomocou heuristiky) a zoznam stavov, ktoré sa z neho dajú dosiahnuť aplikovaním niektorého z operátorov.

Po načítaní stavov sa nastaví zvolená heuristika a spustí sa samotné riešenie problému. To sa realizuje v cykle, kde každá iterácia je pre jednu dvojicu začiatočný stav – koncový stav. V samotnej iterácii nastáva táto postupnosť krokov:

1. Inicializácia premenných na „nulové hodnoty“, spustenie časovaču, výpočet celkového počtu stavov – faktoriál(výška * šírka)
2. While cyklus – kontrola či je fronta prázdna, ak áno pokračujeme krokom 9
3. Výber prvého stavu z fronty a nastavenie ako aktuálny stav
4. Kontrola či je prvý prvok fronty koncový stav, ak áno while cyklus sa končí a vieme, že sme už našli riešenie a pokračujeme krokom 9
5. Vygenerovanie stavov dosiahnuteľných použitím niektorého z operátorov na aktuálny stav – negeneruje sa spätný krok(teda ak bol aktuálny stav dosiahnutý operátorom hore, stav operátorom dole nebude vytvorený)
6. Kontrola vygenerovaných stavov v cykle, či už neboli navštívené alebo ich už nemáme zaradené do fronty – ak neboli navštívené ani nie sú vo fronte, pridáme si ich do fronty na príslušné miesto podľa ceny do cieľa(z heuristiky), fronta je zoradená podľa heuristiky od najmenšieho
7. Kontrola presiahnutia časového limitu nastaveného pri spustení a počtu navštívených stavov proti celkovému počtu stavov – ak aspoň jedno z nich presiahlo očakávanú hodnotu, končíme while cyklus a pokračujeme krokom 9
8. pridáme si aktuálny stav do navštívených a odstránime z fronty, prechádzame na krok 2
9. Ukončenie časovaču a vytvorenie postupnosti stavov od začiatku do konca, ak riešenie nebolo nájdené bude postupnosť prázdna, ak bolo nájdené tak sa cesta generuje od posledného stavu iterovaním cez predchádzajúce stavy a ich zapisovaním do riešenia

Po poslednej iterácii dvojicami sa množina riešení vypíše do súboru zadaného ako 3. argument programu. Vypíše sa spolu s ním takisto čas potrebný na riešenie.

Testovanie

Program som testoval na vstupoch, ktoré sa nachádzajú v priečinku puzzles, začiatočné stavy v súbore start.txt a cieľové stavy v súbore finish.txt. Výstup z nich sa nachádza v solutionDisplaced.txt pre heuristiku políčok, ktoré nie sú na svojom mieste, a solutionManhattan.txt pre heuristiku súčtu vzdialeností políčok od cieľovej pozície.

Prvým cieľom testovania bolo overiť funkčnosť implementovaného algoritmu pre rôzne veľké vstupy, ktoré z definície problému môžu ale nemusia byť riešiteľné. Nepodarilo sa mi, žiaľ, implementovať riešenie úplne všetkých vstupov. Implementoval som aj funkciu na overenie riešiteľnosti vstupu, ktorá je však platná iba pre vstupy kde m=n. Jej problémom je, že označí za neriešiteľné aj vstupy, ktoré majú riešenie tak som ju z algoritmu vyhodil. Asi som sa niekde zmýlil.

Druhým cieľom testovania bolo porovnanie rýchlosti algoritmu pri použití rozličnej heuristiky. Použitie heuristiky políčok, ktoré nie sú na svojom mieste sa ukázalo ako pomalšie, keďže na vyriešenie potrebovalo vygenerovať mnohonásobne viac stavov(čím dlhšie algoritmus beží, tým viac stavov sa muselo vygenerovať). Druhá heuristika, súčet vzdialeností políčok od cieľovej pozície, sa ukázala byť omnoho efektívnejšia. Riešenie väčších vstupov zbehlo rýchlejšie v rádovo stovkách percent.

Porovnanie vlastností použitých heuristík

Obe použité heuristiku sú závislé od veľkosti vstupu – čím väčší stav tým väčšie hodnoty budú generovať.

Výhodou heuristiky súčtu vzdialeností políčok od cieľovej pozície je však to, že čím bližšie riešeniu sa stav nachádza, tým nižšie v poradí fronty sa stav zaradí – tým pádom nemusím prehľadávať frontu do takej hĺbky ako je to pri heuristike počtu políčok, ktoré nie sú na svojom mieste.

V priemernom prípade heuristika súčtu vzdialeností políčok od cieľovej pozície aj navštívi menej stavov, práve kvôli tomu, že navštevuje najskôr stavy, ktoré sú bližšie riešeniu. Takisto aj generuje menej stavov. Toto má za výsledok, že použitím tejto heuristiky beží riešenie rýchlejšie.

Pri malých vstupoch môže byť 1. heuristika lepšia o niekoľko stavov ale nerobí to výrazný rozdiel v rýchlosti.

Pri väčších vstupoch má výrazne navrch 2. heuristika. 1. heuristika nie je schopná pri väčších vstupoch nájsť riešenie pod 60 sekúnd, čo som si pri testovaní stanovil ako smerodajný limit.

Heuristika/číslo vstupu	1	2	3	4	5	6
Počet políčok, ktoré nie sú na cieľovej pozícii	4ms,56	32ms, 499	21ms,305	22658ms, 11022	>60000ms, >8691	>60000ms, 10209
Súčet vzdialeností políčok od cieľovej pozície	4ms, 60	10ms, 213	2ms, 105	101ms, 969	258ms, 1297	1040ms, 3604

Tabuľka obsahuje čas a počet navštívených uzlov použitím jednotlivej heuristiky na vstup

Zhodnotenie riešenia

Myslím si, že moje riešenie je pomerne efektívne a jednoducho implementované. Súvisí to s použitím jazyka Java. Ten umožňuje jednoduchú reprezentáciu stavov v triede, jednoduché porovnávanie stavu systémovými metódami ako aj jednoduchú implementáciu spájaných zoznamov, ktoré sú dôležité pre moje riešenie.

Drobné vylepšenie, ktoré som implementoval už bolo spomenuté v časti opis riešenia problému – negeneruje sa stav, ktorý by vznikol použitím opačného operátoru ako ten, ktorým vznikol súčasný stav.

Riešenie by sa dalo ešte zrýchliť použitím hashovania stavov, o čo som sa snažil ale, žiaľ, bez úspechu. Toto by bolo riešením na problémy pri riešení niektorých vstupov, ktoré momentálne nefungujú.

Spustenie programu

Spúšťa sa program run.jar, ktorý sa nachádza v zadanie2_benovic/puzzles/. Je však nutné mať nainštalovanú Javu aspoň vo verzii 8 a otvoriť na danom mieste príkazový riadok. Spúšťa sa následovne:

java -jar run.jar arg1 arg2 arg3 arg4 arg5

Kde:

- arg1 – cesta k súboru so štartovacími stavmi, napr. start.txt
- arg2 – cesta k súboru s cieľovými stavmi, napr. finish.txt
- arg3 – cesta k výstupnému súboru, napr. soluition.txt
- arg4 – voľba heuristiky, 1 pre počet políčok, ktoré nie sú na cieľovej pozícii
  2 pre súčet vzdialeností políčok od cieľovej pozície
- arg5 – časový limit behu v sekundách, napr. 60

Teda spustenie môže vyzerať ako: java -jar run.jar start.txt finish.txt solution.txt 2 100