

Fakulta informatiky a informačných technológií STU v Bratislave

Umelá inteligencia

Prehľadávanie stavového priestoru

Zadanie 2

Zsolt Kiss

Utorok 14:00-15:50

Ing. Ivan Kapustík

2020/2021

Obsah

Hlavná myšlienka	3
O algoritme	3
Použité heuristiky.....	3
Manhatanské vzdialenosť	3
Linear conflicts	3
O zdrojovom kóde.....	4
Opis dôležitých častí kódu.....	4
Testovanie	5
Test 1: Hlavlom 3x3 bez linear conflict heuristiky – Ak netreba riešiť hlavlom	5
Test 2: Hlavlom 3x3 bez linear conflict heuristiky - Je potrebný len jeden krok.....	6
Test 3: Hlavlom 3x3 bez linear conflict heuristiky – Je potrebné urobiť len dve kroky	7
Test 4: Hlavlom 3x3 bez linear conflict heuristiky – všeobecný.....	8
Test 5: Hlavlom 3x3 bez linear conflict heuristiky – nemá riešenie	9
Test 6: Hlavlom 4x4, bez linear conflict heuristiky – ťažký	10
Test 7: Hlavlom 4x4, linear conflict + Manhattan distance.....	11
Zhrnutie.....	12
Zdroje:	12

Hlavná myšlienka

Zadanie je vypracované v programovacom jazyku Java. Za úlohu som dostal vyriešiť problém č. 2 použitím algoritmu obojsmerného vyhľadávania.

Hlavolam je zložený z 8 očíslovaných políček a jedného prázdneho miesta. Políčka je možné presúvať hore, dole, vľavo alebo vpravo, ale len ak je tým smerom medzera. Máme zadaný nejaký začiatkový stav z ktorého sa musíme dostať do cieľového stavu ktorý máme tiež v zadaní, pritom musíme vypísať postupnosť krokov cez ktoré sme sa dostali k cieľu.

O algoritme

Algoritmy obojsmerného vyhľadávania prekladajú dva samostatné normálne vyhľadávania dopredu od počiatkového stavu a vyhľadávanie dozadu (t. j. pomocou operátorov spätného chodu) z cieľa. Obojsmerné vyhľadávanie nahrádza jeden vyhľadávací graf (ktorý bude pravdepodobne rásť exponenciálne) dvoma menšími čiastkovými grafmi - jedným vychádzajúcim z počiatkového vrcholu a druhým vychádzajúcim z cieľového vrcholu. Vyhľadávanie sa ukončí, keď sa pretnú dva grafy. Meets in the middle je technika vyhľadávania, ktorá sa používa, keď je veľkosť vstupu malá, ale nie dosť malá na to, aby bolo možné použiť priamu hrubú silu.

Použité heuristiky

Manhatanské vzdialenosť

Predpokladajme, že každá pozícia dosky je reprezentovaná ako usporiadaný pár. Vzdialenosť medzi polohami (i, j) a (k, l) je definovaná ako $|i - k| + |j - l|$. Táto vzdialenosť je známa ako Manhatanská vzdialenosť. Pridanie Manhatanskej vzdialenosti medzi doskou x a konečnými pozíciami na doske bude minimálnym odhadcom počtu pohybov potrebných na transformáciu dosky x na dosku riešenia.

Linear conflicts

Predpokladajme, že dva diely, x a y , sú umiestnené v správnom rade, ale obrátene: nazýva sa to lineárny konflikt. Jeden z nich bude musieť zmeniť riadky, aby sa druhý mohol presunúť do svojej konečnej polohy, a po prechode cez stĺpce, ako naznačuje jeho MD, sa kus bude musieť vrátiť do svojho cieľového riadku. To isté sa dá použiť aj na stĺpce. Pre každý lineárny konflikt teda bolo možné k vzdialenosti Manhattanu na palube pridať ďalšie 2 pohyby.[1]

$$MDLC(t) = MD(t) + LC(t)$$

Kde t predstavuje aktuálnu dosku, MD je funkcia, ktorá počíta vzdialenosť na Manhattan pre t , a $LC(t) = 2 * (\text{množstvo lineárnych konfliktov v } t)$. [1]

O zdrojovom kóde

Program na začiatok načíta textové súbory, z ktorého dostaneme rozmery. Program je schopný vyriešiť hlavolam 3x3 alebo 4x4. Následne sa nás opýta, či chceme používať aj linear conflicts heuristiku pri riešení. Po zadaní reťazca "ano" spustíme algoritmus s heuristikami manhattan distance a linear conflict, ak zadáme hocijaký iný reťazec alebo charakter, spustíme algoritmus len s heuristikou Manhattan distance.[1]

Opis dôležitých častí kódu

V triede Algoritmus.java

```
public static Uzol[] start(Stav zaciatok, Stav ciel, boolean linearConflict)
```

- Táto funkcia vráti pole dvoch uzlov, kde sa stretávajú. Cestu dostaneme tzv. "backtrackovaním", používame metódu zoznamy susednosti. To znamená, že každý uzol má predchádzajúceho, a takým spôsobom sa môžeme dostať až k začiatkovej pozícii. Keďže v programe sme používali obojsmerné prehľadávanie znamená, že máme dva uzly. Prvý uzol má smerník, ktorý ukazuje na predchádzajúci uzol až k začiatkovej pozícii. Druhý uzol má predchádzajúceho, ktorý ukazuje až k cieľovej pozícii. Otvorený set obsahuje všetky uzly, ktorý sme ešte nekontrolovali a zatvorený set teda uzly, ktoré sme už kontrolovali.

V programe som používal minimálnu haldy na to, aby som odstránil uzol s najmenšou prioritou z otvoreného setu. Java poskytuje Priority queue, takže to som nemal implementovať manuálne. Hash mapy som používal na uloženie informácií o stavoch uzlov. Zistíme, ktorý uzol má najnižšiu prioritu, a následne ho premiestnime z otvoreného setu do zatvoreného a odstránime ho z haldy. Následne pre všetky štyri smery vytvoríme vstavy, ktoré sú výsledkom posunu. Uzol vložíme do 3 hald, a do hash tabulky. Na koniec vytvoríme Uzol, ktorý ak má nenulovú hodnotu, znamená, že sme cestu našli, a potom len vypíšeme jednotlivé stavy, a kroky, ktoré boli potrebné pre riešenie problému. Sú aj prípady, keď existujú viaceré riešenia, v tom prípade vypíšeme všetky.

V triede Stav.java

```
public Stav posun(Operator operator)
```

- V tejto funkcii skúsime aplikovať operátora a následne vytvoriť si novú tabuľku, kde je už táto zmena aplikovaná. Funkcia vráti null, ak nie je možné aplikovať operátora.

```
public Pozicia[] vratitSpravnuPoziciu(Stav ciel)
```

- Funkcia podľa parametra zistí, kde sa nachádza správna pozícia dlaždice, ktorú aktuálne sledujeme. Vráti pole pozícií pre každú dlaždicu.

```
public int vypovitajManhattanDistance(Stav ciel)
```

- Táto funkcia vráti integer hodnotu Manhattanskej vzdialenosti medzi dvoma bodmi.

```
public int linearConflict(Stav goal, boolean linearConflict)
```

- Táto funkcia vypočíta koľko krokov treba urobiť aby sme sa zbavili lineárneho konfliktu. Vráti integer, ktorá je čiastka Manhattanskej vzdialenosti a počet potrebných krokov, charakterizované dole.

```
private int potrebneKrokyPreRiesenieKonfliktu(int[] pocetKonfliktov)
```

- Funkcia vráti počet zvyšných krokov, koľko je potrebné na riešenie konfliktu. Vypočítal som to podľa vzorca, ktoré som spomínal hore v bode Použité heuristiky – linear conflicts

V triede Uzol.java

```
public String cestaZoStartuKaktualnej()
```

- Funkcia vráti reťazec od začiatočného uzla k aktuálnemu uzlu.

```
public String cestaNaspatVynechajPrvu()
```

- Funkcia vráti reťazec od cieľového uzla k aktuálnej (kde sa stretli uzly) ale vynechá prvý stav

```
public String cestaNaspat()
```

- Funkcia vráti reťazec od cieľového uzla k aktuálnej (kde sa stretli uzly).

```
public int compareTo(Uzol otherUzol)
```

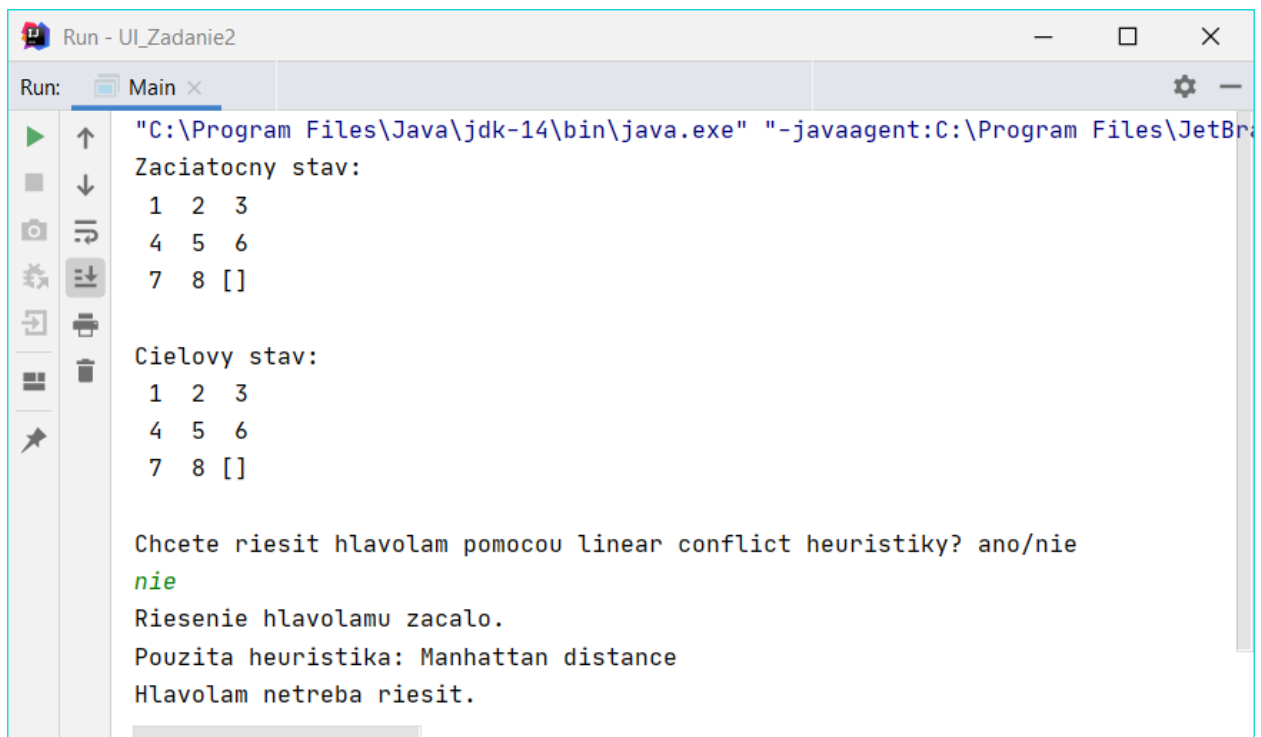
- Pomocná funkcia na zistenie, ktorý uzol má vyššiu prioritu

Testovanie

POZNÁMKA: scenáre načítavam z textových súborov. V main-e treba zmeniť miesto, odkiaľ tieto súbory načítavame. Súbor zaciatok.txt obsahuje začiatočný stav hlavolamu.

Test 1: Hlavolam 3x3 bez linear conflict heuristiky – Ak netreba riešiť hlavolam

Súbor: txt/3x3_01.txt



```
Run - UI_Zadanie2
Run: Main x
"C:\Program Files\Java\jdk-14\bin\java.exe" "-javaagent:C:\Program Files\JetBr
Zaciatocny stav:
1 2 3
4 5 6
7 8 []

Cielovy stav:
1 2 3
4 5 6
7 8 []

Chcete riesit hlavolam pomocou linear conflict heuristiky? ano/nie
nie
Riesenie hlavolamu zacalo.
Pouzita heuristika: Manhattan distance
Hlavolam netreba riesit.
```

Test 2: Hlavlom 3x3 bez linear conflict heuristiky - Je potrebný len jeden krok

Súbor: txt/3x3_02.txt

```
Run - UI_Zadanie2
Run: Main x
Zaciatocny stav:
1 2 3
4 5 6
7 8 []

Cielovy stav:
1 2 3
4 5 []
7 8 6

Chcete riesit hlavolam pomocou linear conflict heuristiky? ano/nie
nie
Riesenie hlavolamu zacalo.
Pouzita heuristika: Manhattan distance
Smer: Dopredu
Hlbka dopredu:1
Hlbka dozadu: 0

Zaciatocny stav:
1 2 3
4 5 6
7 8 []

Dole
1 2 3
4 5 []
7 8 6

Uzly sa stretli tu. Cielovy stav prveho uzla je zaciatocny stav druheho uzla.

Druhy uzol nevykonal ziadne kroky.

Vytvorene uzly: 5 (4 otvorene/1 zatvorene)
Dlžka cesty: 1
Beh programu: PT0.0180007S
```

Test 3: Hlavalam 3x3 bez linear conflict heuristiky – Je potrebné urobiť len dve kroky

Súbor: txt/3x3_03.txt

```
Run - UI_Zadanie2
Run: Main x
Zaciatocny stav:
1 2 3
4 5 6
7 8 []

Cielovy stav:
1 2 3
4 [] 5
7 8 6

Chcete riesit hlavalam pomocou linear conflict heuristiky? ano/nie
nie
Riesenie hlavalamu zacalo.
Pouzita heuristika: Manhattan distance
Smer: Dopredu
Hlbka dopredu:2
Hlbka dozadu: 0

Zaciatocny stav:
1 2 3
4 5 6
7 8 []

Dole
1 2 3
4 5 []
7 8 6

Vpravo
1 2 3
4 [] 5
7 8 6

Uzly sa stretli tu. Cielovy stav prveho uzla je zaciatocny stav druheho uzla.

Druhy uzol nevykonal ziadne kroky.

Vytvorene uzly: 7 (5 otvorene/2 zatvorene)
Dlzska cesty: 2
Beh programu: PT0.0180008S
```

Test 4: Hlavolam 3x3 bez linear conflict heuristiky – všeobecný

Súbor: txt/3x3_08.txt

Zaciatocny stav:

```
1 2 3
4 5 6
7 8 []
```

Cielovy stav:

```
1 2 3
[] 4 8
7 6 5
```

```
Run - UI_Zadanie2
Run: Main x
Zaciatocny stav:
1 2 3
4 5 6
7 8 []

Vpravo
1 2 3
4 5 6
7 [] 8

Dole
1 2 3
4 [] 6
7 5 8

Vlavo
1 2 3
4 6 []
7 5 8

Hore
1 2 3
4 6 8
7 5 []

Uzly sa stretli tu. Cielovy stav prveho uzla je zaciatocny stav druheho uzla.

Vpravo
1 2 3
4 6 8
7 [] 5

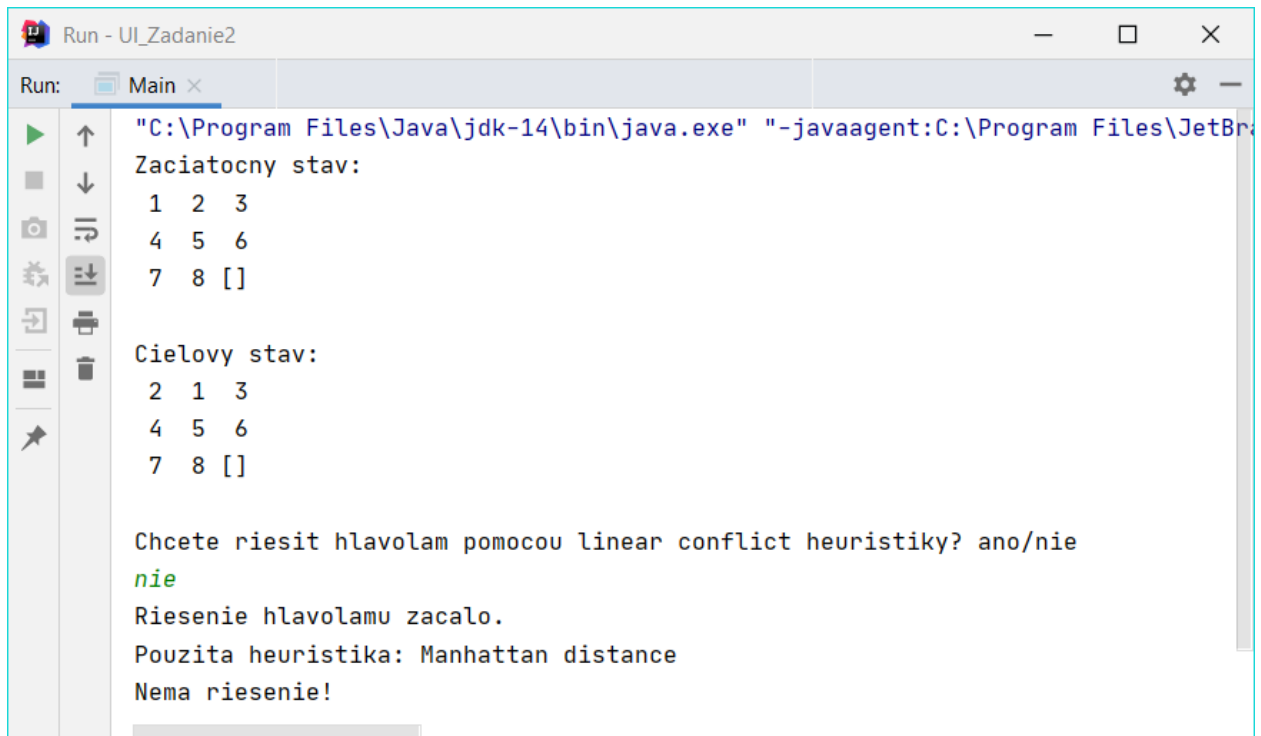
Dole
1 2 3
4 [] 8
7 6 5

Vpravo
1 2 3
[] 4 8
7 6 5

Vytvorene uzly: 31 (19 otvorene/12 zatvorene)
Dlžka cesty: 7
Beh programu: PT0.0219979S
```


Test 5: Hlavoľam 3x3 bez linear conflict heuristiky – nemá riešenie

Súbor: txt/3x3_09.txt



```
Run - UI_Zadanie2
Run: Main x
"C:\Program Files\Java\jdk-14\bin\java.exe" "-javaagent:C:\Program Files\JetBr
Zaciatocny stav:
 1 2 3
 4 5 6
 7 8 []

Cielovy stav:
 2 1 3
 4 5 6
 7 8 []

Chcete riesit hlavoľam pomocou linear conflict heuristiky? ano/nie
nie
Riesenie hlavoľamu zacalo.
Pouzita heuristika: Manhattan distance
Nema riesenie!
```

Test 6: Hlavalam 4x4, bez linear conflict heuristiky – ťažký

Súbor: 4x4_08.txt

Zaciatocny stav:

```
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 []
```

Cielovy stav:

```
2 10 15  4
14 11 []  6
9 13  8  3
5  7 12  1
```

Chcete riesit hlavalam pomocou linear conflict heuristiky? ano/nie

nie

Riesenie hlavalamu zacalo.

Pouzita heuristika: Manhattan distance

Smer: Dopredu

Hlbka dopredu:25

Hlbka dozadu: 24

Algoritmus zvládol riešiť hlavalam, avšak mu trvalo 6 minút a 31 sekúnd.

Vytvorene uzly: 3344129 (1551445 otvorene/1792684 zatvorene)

Dlzska cesty: 49

Beh programu: PT6M19.9615881S

Test 7: Hlavalom 4x4, linear conflict + Manhattan distance

Súbor: 4x4_08.txt

Začiatocny stav:

```
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 []
```

Cielovy stav:

```
2 10 15  4
14 11 []  6
9 13  8  3
5  7 12  1
```

Chcete riesit hlavalom pomocou linear conflict heuristiky? ano/nie

ano

Riesenie hlavalamu zacalo.

Pouzite heuristiky: Manhattan distance + Linear conflict

Smer: Dopredu

Hlbka dopredu:25

Hlbka dozadu: 24

Začiatočný a cieľový stav ja ten istý, ako v teste 6, ale teraz sme používal aj druhú heuristiku, linear conflicts – na vylepšenie prvej heuristiky, tz. Manhattan distance.

Vytvorene uzly: 2142441 (997110 otvorene/1145331 zatvorene)

Dlžka cesty: 49

Beh programu: PT2M21.4393242S

Z výsledku vidíme, že použitím linear conflicts heuristiky dĺžku cesty sme nezmenili, ale počet vytvorených uzlov ako aj čas potrebný na vyriešenie hlavalamo sa zredukovalo.

Zhrnutie

Myslím si, že tento algoritmus je veľmi zaujímavý. Jedna veľká výhoda prečo používať obojsmerný prehľadávací algoritmus namiesto algoritmu, ktorý prehľadáva priestor len z jednej strany je čas. Simultánnym vyhľadávaním drasticky skracuje čas hľadania. Šetrí tiež zdroje pre používateľov, pretože na ukladanie všetkých vyhľadávaní vyžaduje menšiu kapacitu pamäte. V poslednom teste sme preukázali na to, že použitím oboch heuristikou môžeme optimalizovať časovú a pamäťovú náročnosť.

Zdroje:

[1] (PDF) *Parallel processing Puzzle N-2-1 on cluster architectures performance analysis*.

Available from:

https://www.researchgate.net/publication/224323439_Parallel_processing_Puzzle_N-2-1_on_cluster_architectures_performance_analysis [accessed Oct 25 2020].