Informatyka, studia niestacjonarne, inż I st. semestr VI

Sztuczna inteligencja i systemy ekspertowe 2013/2014

Prowadzący: dr inż. Krzysztof Lichy sobota, 15:30

Data oddania: _____ Ocena: _____

Łukasz Ochmański 183566

Przemysław Szwajkowski 173524

Zadanie 1 - przeszukiwanie przestrzeni stanów*

1. Wprowadzenie

Celem niniejszego zadania jest napisanie dwóch programów. Pierwszy z nich ma za zadanie odnaleźć rozwiązanie łamigłówki zwanej "Piętnastką", a drugi ma na celu wizualizację rozwiązywania łamigłówki.

2. Uruchamianie programu

Program można uruchomić z lini poleceń w systemie z zainstalowaną wirtualną maszyną Java'y wersji 7 lub nowszej. Program przyjmuje jeden parametr:

— `alogorytm`
a do wyboru: *dfs, bfs, dijkstra, a1, a2, a3*

Przed uruchomieniem należy spakować projekt wraz z bibliotekami do formatu *.jar. Metoda main() znajduje się w pliku Solver.java

Następnie uruchomić polecenie:

```
java -jar Zadanie1.jar bfs;
```

* SVN: `https://sise-lukasz-ochmanski.googlecode.com/svn/trunk/`

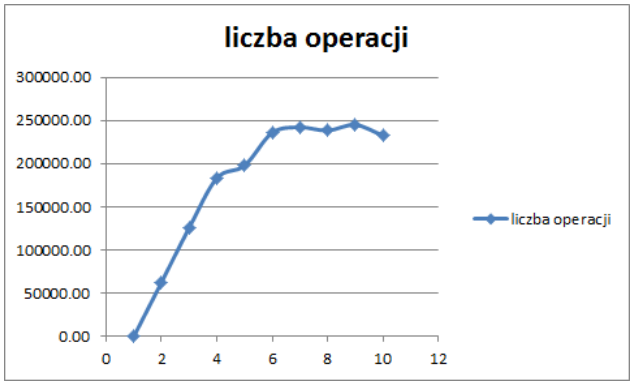
poziom	liczba operacji	odwiedzone węzły	czas wykonania	przeciętne rozwiązanie
1	28	29	0.005ms	28
2	62660	61952	0.758ms	468
3	125773	124339	1.415ms	14494
4	183119	181038	1.820ms	5388
5	197815	195536	2.128ms	5610
6	235743	233053	2.503ms	5640
7	242122	239335	2.391ms	3930
8	238376	235654	2.664ms	2906
9	244965	242148	2.612ms	2935
10	232649	229995	2.890ms	1108

Tabela 1. Depth-First Search

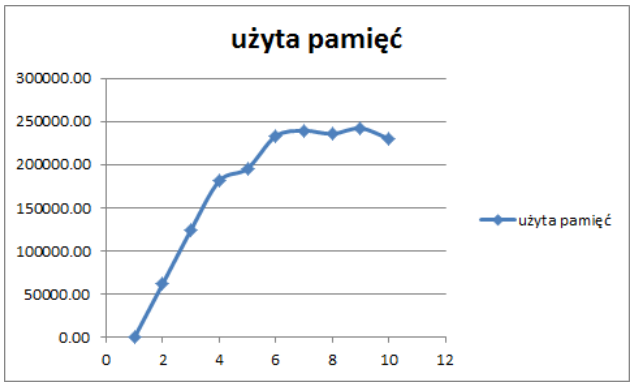
3. Analiza danych

3.1. DFS

Algorytm przeszukiwania grafu "w głąb" (*ang. Depth-First Search*) jest algorytmem rekurencyjnym i polega na tym, że plansza przeszukiwana jest od korzenia wzdłuż jednej gałęzi, w zadanym z góry kierunku. Kiedy algorytm dojdzie do końca gałęzi, wraca do rodzica i przechodzi do kolejnego dziecka, które także jest zdefiniowane przez kolejność.



Rysunek 1. Depth-First Search



Rysunek 2. Depth-First Search

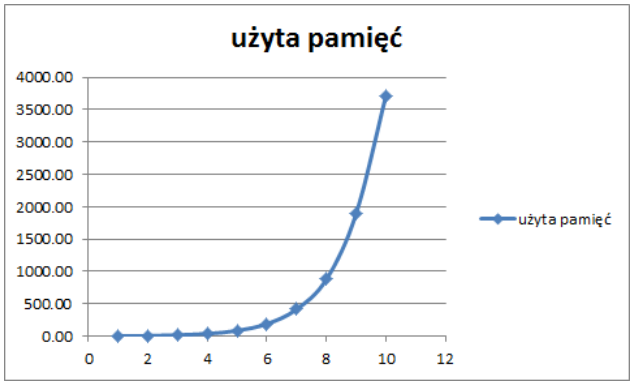
poziom	liczba operacji	odwiedzone węzły	czas wykonania	przeciętne rozwiązanie
1	2	3	0.001ms	1
2	7	8	0.001ms	2
3	18	19	0.001ms	3
4	40	41	0.002ms	4
5	86	87	0.002ms	5
6	191	192	0.002ms	6
7	417	418	0.005ms	7
8	880	881	0.009ms	8
9	1894	1895	0.020ms	9
10	3709	3710	0.041ms	10

Tabela 2. Breadth-first search

3.2. BFS

BFS (*ang. Breadth-first search*) czyli algorytm przeszukiwania grafu "wszerz". Algorytm ten przeszukuje cały graf wszerz aż do znalezienia węzła docelowego. W przypadku rozwiązywania łamigłówki, jego działanie polega na tym, że pobiera stan początkowy i sprawdza czy spełnia on warunek stopu. Jeżeli nie, to przechodzi do wyznaczenia wszystkich możliwych stanów potomnych zgodnie z podaną kolejnością. BFS nie wykorzystuje żadnej heurystyki.

Rysunek 3. Breadth-first search



Rysunek 4. Breadth-first search

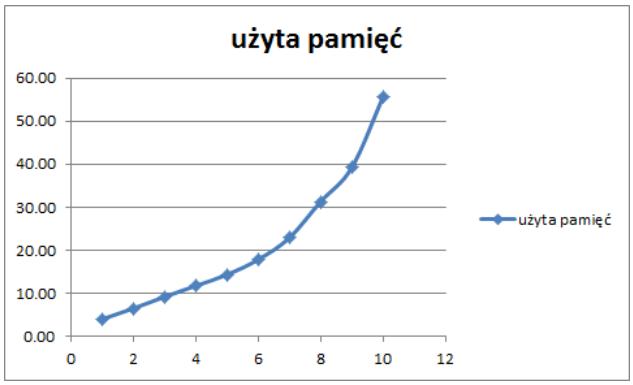
poziom	liczba operacji	odwiedzone węzły	czas wykonania	przeciętne rozwiązanie
1	3	4	221µs	1
2	7	7	380µs	2
3	10	9	233µs	3
4	14	12	224µs	4
5	18	14	210µs	5
6	23	18	230µs	6
7	30	23	309µs	7
8	43	31	348µs	8
9	55	39	431µs	9
10	79	56	622µs	10

Tabela 3. A* Odleglosc taksowkowa

3.3. A* Odległość taksówkowa

Algorytm ten wykorzystuję heurystykę. Jego działanie polega na tym, że wybierana jest ścieżka pomiędzy wierzchołkiem początkowym, a końcowym. Wybór kolejnego elementu ścieżki uzależniony jest od wartości funkcji $f(x) = g(x) + h(x)$. Odległość taksówkowa to odległość od pustego klocka do jego docelowej pozycji. Odległość taksówkowa jest równa sumie współrzędnych w układzie kartezjańskim.

Rysunek 5. A* Odleglosc taksowkowa



Rysunek 6. A* Odleglosc taksowkowa

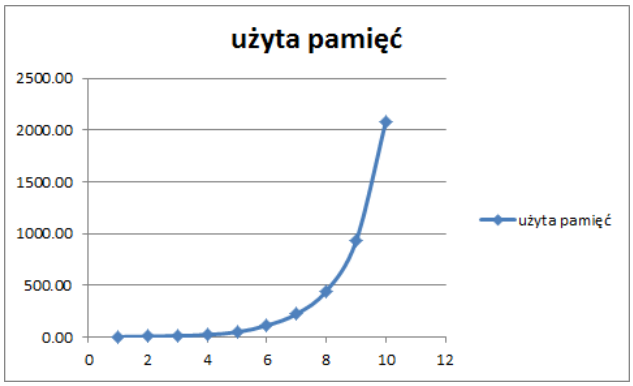
poziom	liczba operacji	odwiedzone węzły	czas wykonania	przeciętne rozwiązanie
1	3	4	1839µs	1
2	9	8	763µs	2
3	17	14	474µs	3
4	33	24	558µs	4
5	73	50	1570µs	5
6	170	114	1540µs	6
7	341	226	3376µs	7
8	680	446	8783µs	8
9	1425	930	33083µs	9
10	3213	2085	214685µs	10.05

Tabela 4. A* Odleglosc Hamminga

3.4. A* Odległość Hamminga

Algorytm ten wykorzystuję heurystykę. Liczba klocków nie znajdująca się na swojej właściwej pozycji.

Rysunek 7. A* Odleglosc Hamminga



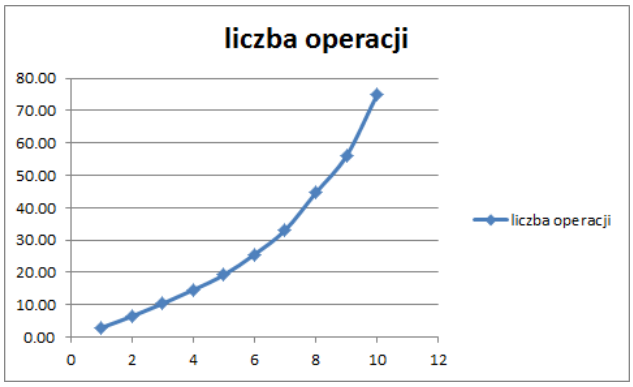
Rysunek 8. A Odleglosc Hamminga

poziom	liczba operacji	odwiedzone węzły	czas wykonania	przeciętne rozwiązanie
1	3	4	260µs	1
2	7	7	460µs	2
3	10	9	454µs	3
4	15	12	271µs	4
5	19	15	215µs	5
6	26	20	265µs	6
7	33	25	330µs	7
8	45	32	381µs	8
9	56	40	528µs	9
10	75	52	681µs	10

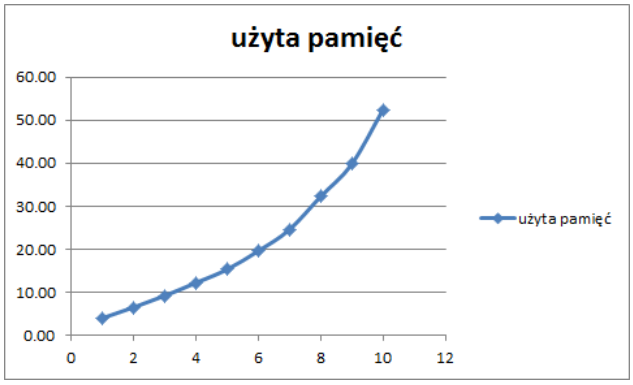
Tabela 5. A* Suma odleglosci taksowkowych

3.5. A* Suma odległości taksówkowych

Algorytm ten wykorzystuję heurystykę. Odległość taksówkowa wszystkich klocków do swojej docelowej pozycji. Odległość taksówkowa jest równa sumie współrzędnych w układzie kartezjańskim.

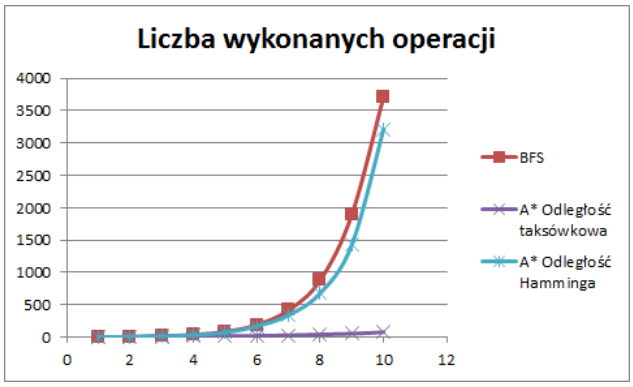


Rysunek 9. A* Suma odleglosci taksowkowych

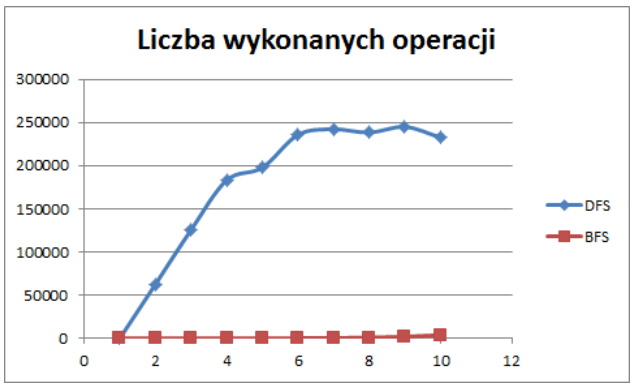


Rysunek 10. A* Suma odleglosci taksowkowych

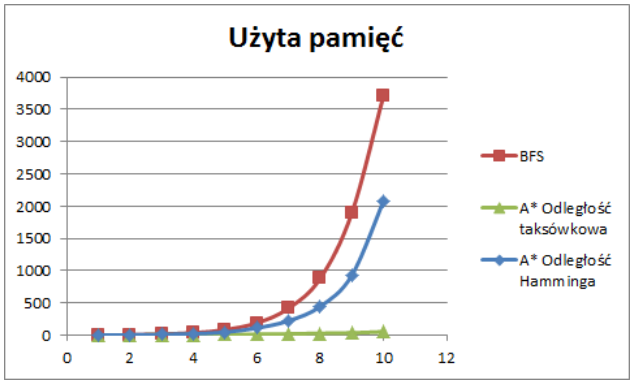
4. Porównanie



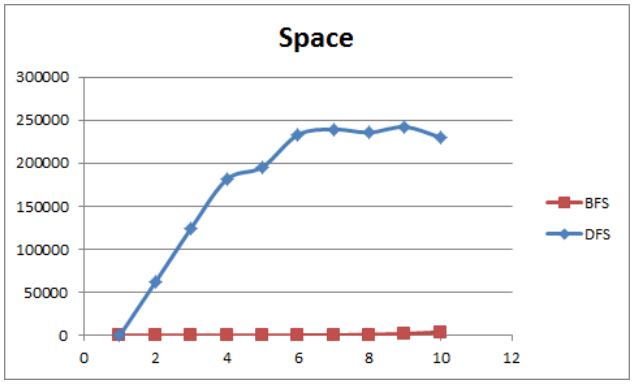
Rysunek 11. BFS vs A*



Rysunek 12. BFS vs DFS



Rysunek 13. BFS vs A*



Rysunek 14. BFS vs DFS

5. Wnioski

Najwydajniejszymi algorytmami okazały się, zgodnie z oczekiwaniami, algorytmy wykorzystujące heurystykę. Spowodowane jest to tym, że wybierane są tutaj drogi, które są najbliżej rozwiązania porzucając pozostałe. Funkcjonalności tej pozbawione są algorytmy nie wykorzystujące heurystyk, przez co rośnie czas poszukiwania. Najmniej wydajnym algorytmem okazał się DFS, ponieważ liczba rekurencji jest tutaj największa.

Literatura

[1] T. Oetiker, H. Partl, I. Hyna, E. Schlegl. *Nie za krótkie wprowadzenie do systemu LaTeX2e*, 2007, dostępny online.

[2] Przemysław Klęsk. *Algorytmy przeszukiwania grafów i drzew dla gier i łamigłówek*, `http://wikizmsi.zut.edu.pl/uploads/b/be/2_search.pdf`

[3] Wikipedia, wolna encykolpedia *Breadth-first search*, `http://en.wikipedia.org/wiki/Breadth-first_search`

[4] Wikipedia, wolna encyklopedia *Algorytm A**, `http://pl.wikipedia.org/wiki/Algorytm_A*`