Piętnastka

Jan Bełczącki, 180503

2014 / 2015

Prowadzący: mgr inż. Michał Pryczek

Contents

1 Cel

Celem projektu było napisanie programu rozwiązującego problematykę przeszukiwania przestrzeni stanów. Realizacja opierała się o prostą w założeniach zagadkę logiczną zwaną „piętnastką".

2 Wprowadzenie

„Piętnastka" to nazwa płaskich prostokątnych puzzli składających się z 15 klocków i jednego pustego miejsca na klocek. Klocki rozstawione są na planszy o rozmiarach 4 x 4. Rozwiązanie polega na takim przesuwaniu klocków na planszy, aby w efekcie ułożyć je rosnąco według numerów. Klocki można przesuwać tylko w miejsce pustego miejsca. Zagadnienie „piętnastki" jest jednym z modelowych zagadnień, do rozwiązania którego stosuje się algorytmy przeszukiwania ścieżek w grafie. W szczególności, gdy jako stan przyjmiemy aktualne ułożenie klocków w danym momencie, który to będziemy przechowywać w wierzchołku przeszukiwanego grafu. Program realizujący powyższe zagadnienie napisany został w języku C++. Do rozwiązania zagadki wykorzystano trzy metody przeszukiwania grafu:

- Przeszukiwanie „w głąb" (Depth-First Search): Algorytm polega na przeszukiwaniu badaniu krawędzi ostatnio odwiedzonego wierzchołka, z którego jeszcze wychodzą nie zbadane krawędzie. Gdy wszystkie krawędzie opuszczające ten wierzchołek są zbadane, algorytm powraca do wierzchołka, z którego dany wierzchołek został odwiedzony.
- Przeszukiwanie „wszerz" (Breadth-First Search): W tym algorytmie granica między wierzchołkami odwiedzonymi i nieodwiedzonymi jest przekraczana jednocześnie na całej jej szerokości. Dodatkowo algorytm ten zapewnia znalezienie najkrótszej ścieżki do węzła zawierającego stan końcowy, jeśli jest to możliwe.
- Przeszukiwanie algorytmem A*: Jest to algorytm heurystyczny, wprowadzający funkcję kosztu dla każdego wierzchołka w grafie. Decyduje on, który wierzchołek zostanie odwiedzony w pierwszej kolejności. Heurystyka użyta w programie wykorzystuję metodę Manhattan dla określania odległości klocków od ich docelowej pozycji na planszy. Obliczane jest to poprzez sumowanie odległości w osi poziomej i odległości w osi pionowej.

3 Sposób działania i implementacja

Użytkownik może wybrać na początku ile wynosić ma maksymalna głębokość, w której znajdzie się węzeł końcowy grafu. Następnie algorytm pseudolosujący generuje zagadkę, której rozwiązania podejmie się program. Aby być pewnym tego, że wygenerowany układ klocków prowadzi do rozwiązania, algorytm pseudolosujący generuje go od stanu końcowego. Wykonuje tyle kroków ile wpisał

użytkownik, co prowadzi do utworzenia zagadki której rozwiązanie znajduje się na głębokości co najwyżej równej liczbie wykonanych kroków.

Sama plansza przetrzymywana jest w globalnej macierzy 4 x 4, natomiast stan zagadki w strukturze, przechowującej między innymi zaszyfrowany stan planszy, identyfikator stanu, czy odległości poszczególnych klocków od ich docelowych miejsc. Stan planszy zaszyfrowany jest za pomocą czterech zmiennych typu short odpowiadających czterema rzędami planszy. Przeprowadzając na nich operacje bitowe program koduje i dekoduje stany planszy.

3.1 DFS

Algorytm przeszukiwania „w głąb" wykorzystuje stos, w którym przechowywane są stany zagadki do przetworzenia. W każdym kroku pętli, sprawdzane jest czy dany wierzchołek jest wierzchołkiem końcowym. Jeśli nie, na stos odkładane wierzchołki, do których prowadzą krawędzie wychodzące ze sprawdzanego wierzchołka.

3.2 BFS

Algorytm przeszukiwania „wszerz" wykorzystuje kolejkę. Podobnie jak dla poprzedniego algorytmu, umieszczane są w niej stany zagadki, które czekają na przetworzenie. Różnica jest jednak w samym działaniu kontenera – kolejka działa na zasadzie bufora typu FIFO, w przeciwieństwie do stosu, który wykorzystuje działanie bufora LIFO. W ten sposób zapewniamy, że wszystkie wierzchołki leżące na wspólnym poziomie grafu przetwarzane są jeden po drugim.

3.3 A*

Algorytm A* oparty jest o kolejkę priorytetową, która w przypadku biblioteki standardowej c++, w momencie pobrania z niej elementu przy pomocy funkcji top() zwraca element o najwyższym (określonym przez programistę) priorytecie. W tym wypadku jest to najniższa waga węzła. Przy odwiedzaniu wierzchołków sąsiadujących, określane są odległości klocków w stanie bazując na heurystyce Manhattan. Wartości dla każdego klocka są sumowane co składa się na wagę, która ostatecznie zwiększana jest o wagę wierzchołka-rodzica. W przypadku gdy algorytm określi niższą wagę dla konkretnego wierzchołka niż ta określona wcześniej, przeprowadzona zostaje relaksacja.

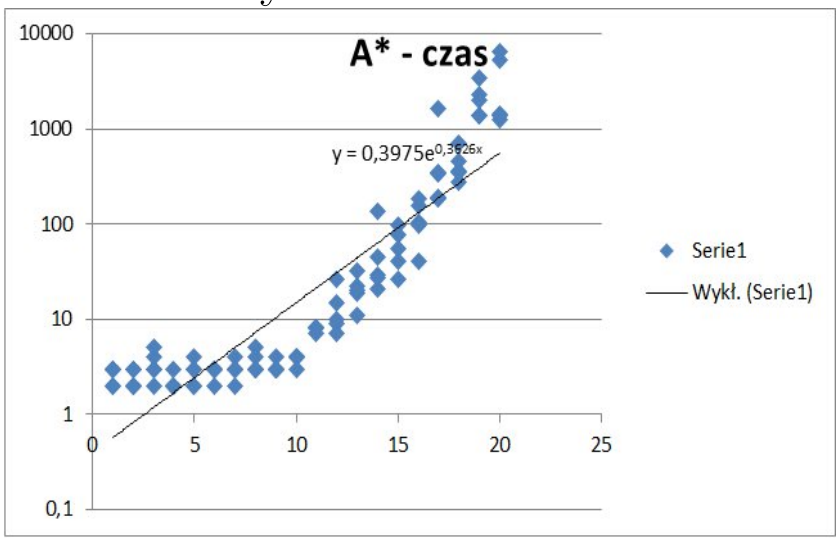
4 Wyniki

Dla każdego algorytmu przeprowadzono odpowiednie testy. Polegały one na zbadaniu ilości czasu i wykorzystanej pamięci (ilość odwiedzonych węzłów) w trakcie rozwiązywania zagadki o konkretnym poziomie trudności. Dla każdego poziomu trudności wykonano 5 testów. Warunkiem powodzenia było zmieszczenie się w 100000 kroków. Dodatkową ważną informacją jest wielkość struktury przechowującej stan zagadki – 84 bajty.

Sprzęt na którym przeprowadzano testy:

- Procesor: Intel Core i7-3630QM 2,4 GHz
- RAM: 16 GB
- OS: Windows 8.1

A* - czas wykonania

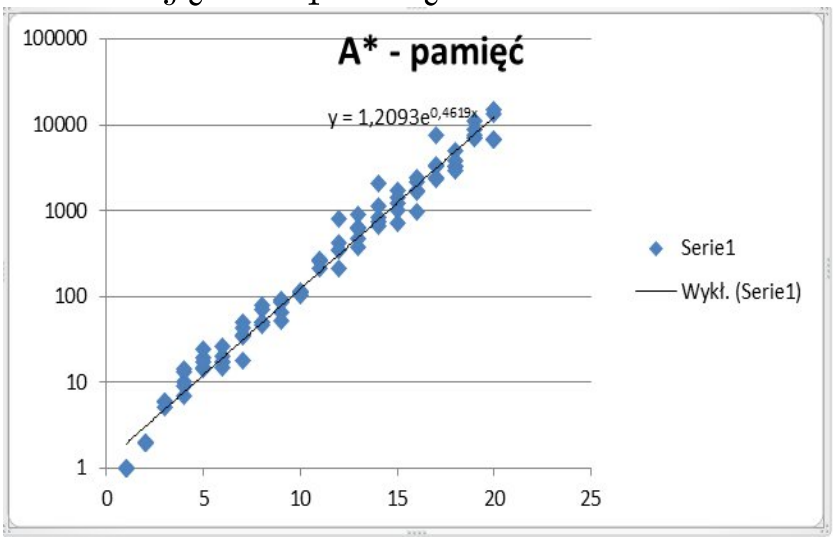


Wzór przybliżenia wykładniczego:

$$0.3975 * e^{0.3626*x}$$

Współczynniki: a = 0.3975, b = 0.3626

A* - zajętość pamięci

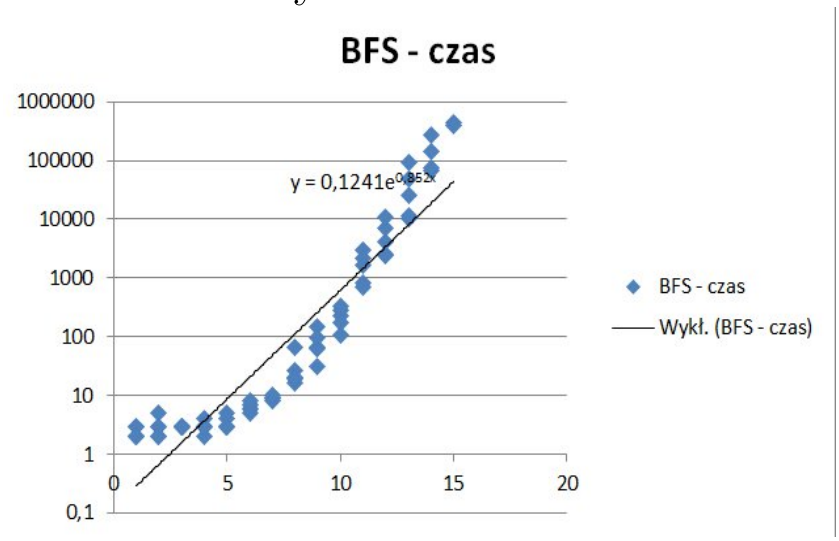


Wzór przybliżenia wykładniczego:

$$1.2093 * e^{0.04619*x}$$

Współczynniki: a = 1.2093, b = 0.4619

BFS - czas wykonania

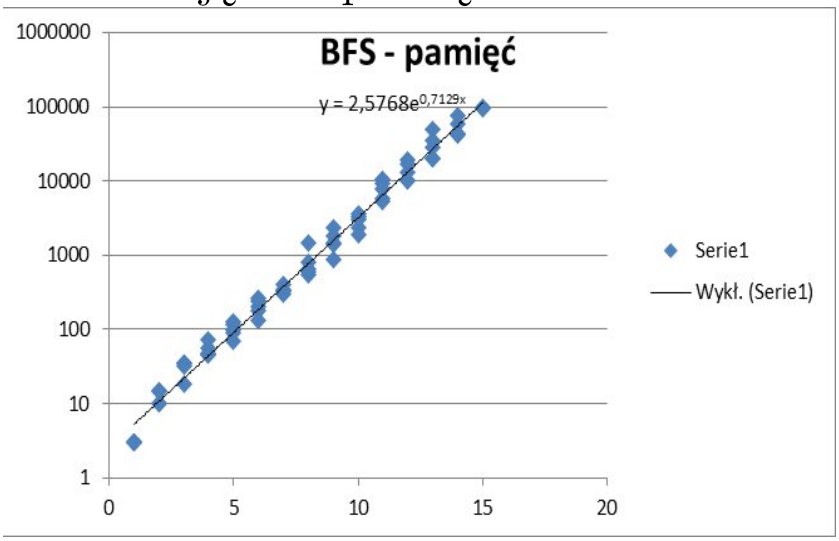


Wzór przybliżenia wykładniczego:

$$0.1241 * e^{0.852*x}$$

Współczynniki: a = 0.1241, b = 0.852

BFS - zajętość pamięci

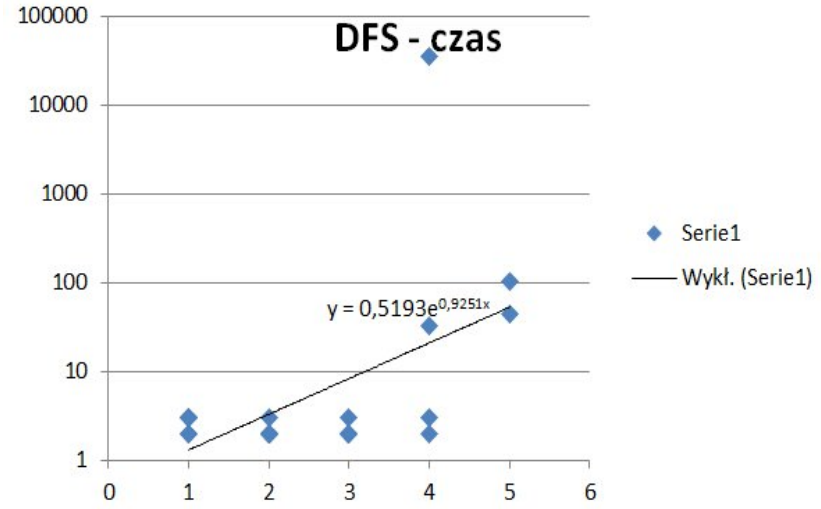


Wzór przybliżenia wykładniczego:

$$2.5768 * e^{0.7129x}$$

Współczynniki: a = 2.5768, b = 0.7129

DFS - czas wykonania

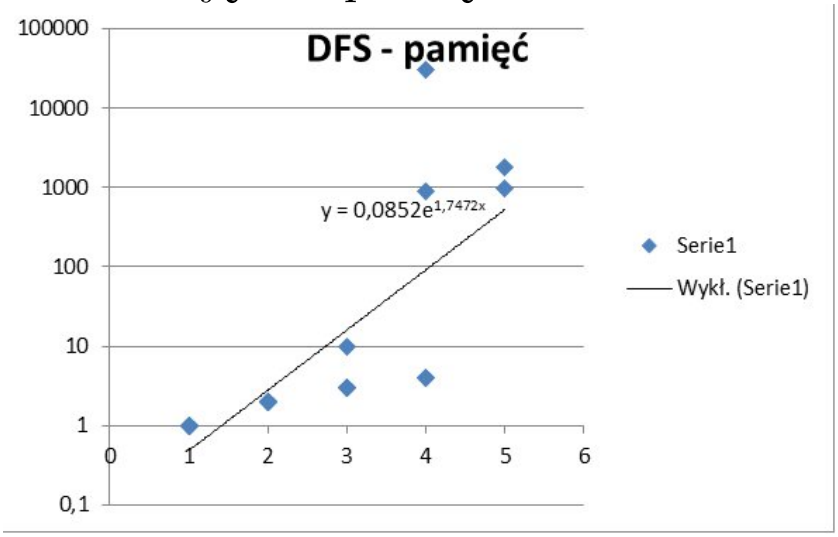


Wzór przybliżenia wykładniczego:

$$0.5193 * e^{0.9251*x}$$

Współczynniki: a = 0.5193, b = 0.9251

DFS - zajętość pamięci



Wzór przybliżenia wykładniczego:

$$0.0852 * e^{1.7472 * x}$$

Współczynniki: a = 0.0852, b = 1.7472

5 Wnioski

Wyniki przedstawiają wyraźną przewagę algorytmu A* nad pozostałymi użytymi w programie. Nie tylko potrzebował mniejszej ilości węzłów do odwiedzenia, aby znaleźć węzeł końcowy, ale też wykazał się najwyższą szybkością działania. Algorytm BFS zawsze znajdował rozwiązanie aż do poziomu trudności 15, jednak trwało to dłużej niż w przypadku A* oraz wymagało większego zużycia pamięci. Dla wyższych poziomów trudności zajmowało mu to już zbyt dużo czasu i zasobów, co dyskwalifikowało go. Gdyby jednak nie ograniczenia, algorytm BFS za każdym razem znajdowałby rozwiązanie. Algorytm DFS wypadł najgorzej, głównie z powodu swojej charakterystyki. Czasem udawało mu się znaleźć rozwiązanie szybko, jednak częściej rozpoczynał przeszukiwanie od niekorzystnego węzła, co prowadziło do bezcelowego przeszukiwania obszernych gałęzi grafu. Nie gwarantuje on także znalezienia rozwiązania gdy takowe istnieje.

6 Bibliografia

- Thomas Cormen - Wprowadzenie do algorytmów
- $http://pl.wikipedia.org/wiki/Algorytm_{A*}$
- $http://dydaktyka.polsl.pl/kwmimkm/MH$