Informatyka, studia dzienne, inż I st. semestr IV

Sztuczna inteligencja i systemy ekspertowe 2024/2025

Prowadzący: dr inż. Krzysztof Lichy Wtorek, 12:15

Data oddania: 07-04-2025 Ocena: _____

Aleksander Gencel 251517
Bartosz Żelaziński 251670

Zadanie 1: Piętnastka

Spis treści

1. Cel

Celem zadanie było zapoznanie się z algorytmami przechodzenia (bądź też przeszukiwania) grafów. Takimi są bowiem algorytmy Deep-First Search (DFS) oraz Breadth-First Search (BFS). Ten pierwszy to algorytm przeszukiwania grafu w głąb, natomiast ten drugi przeszukuje graf wszerz.

2. Wprowadzenie

Piętnastka to gra, która polega na rozwiązaniu układanki w taki sposób, żeby spośród rozsypanych liczb przesunąć cyfrę 0 tak, aby cała gra była ułożona po kolei z zerem na końcu. Aby rozwiązać przykładową grę w tabeli 1 należy zwyczajnie wykonać dwa ruchy w dół.

1	2	3	**0**
5	6	7	4
9	10	11	8
13	14	15	12

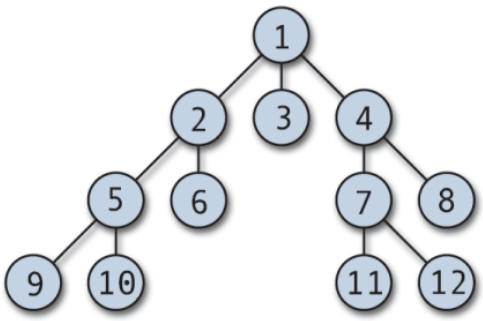
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	**0**

Tabela 1. Gra w piętnastkę. Gra początkowa po lewej, gra rozwiązana po prawej. [1]

Zaprezentowane poniżej algorytmy mogą być wykorzystywane nie tylko do rozwiązywania problemu piętnastki, ale także do rozwiązywania innych problemów, które można przedstawić w postaci grafu. Każdy z nich ma swoje charakterystyczne cechy, różnią się między innymi wydajnością.

2.1. Algorytm BFS

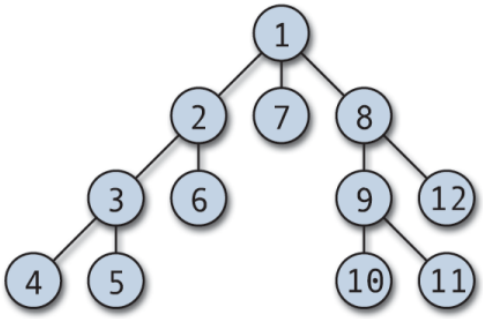
Algorytm przeszukiwania wszerz (ang. *Breadth-First Search*, BFS) jest algorytmem przeszukiwania grafu, który gwarantuje na znalezienie najkrótszej ścieżki w grafie. Wykorzystuje on w implementacji kolejkę FIFO, co pozwala na przeszukiwanie grafu poziomami. Charakteryzuje go przede wszystkim szybkość działania i nieduże zużycie pamięci. [4]



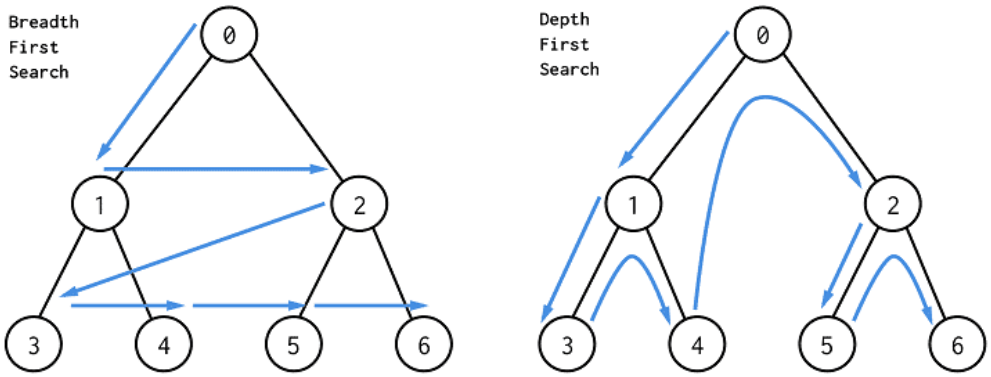
Rysunek 1. Kolejność przeszukiwania drzewa w algorytmie BFS. [2]

2.2. Algorytm DFS

Algorytm ten przeszukuje graf w sposób charakterystyczny – poprzez zagłębianie się w głąb struktury grafu aż do osiągnięcia zadanej maksymalnej głębokości lub do momentu, w którym znajdzie rozwiązanie. Stąd wywodzi się jego nazwa algorytmu przeszukiwania w głąb (ang. *Deep-First Search*, DFS). W przeciwieństwie do algorytmu BFS korzysta on ze stosu LIFO i w tej implementacji jest wolniejszy. [4]



Rysunek 2. Kolejność przeszukiwania drzewa w algorytmie DFS. [3]



Rysunek 3. Porównanie sposobu przeszukiwania algorytmów BFS (lewo) i DFS (prawo).

2.3. Algorytm A-star (A*)

Ten algorytm jest algorytmem heurystycznym. Posługując się funkcją

$$f(x) = g(x) + h(x), \tag{1}$$

gdzie $g(x)$ to koszt dotarcia do węzła x, a $h(x)$ to koszt dotarcia z węzła x do celu, algorytm A-star przeszukuje te wierzchołki, które mają najmniejszą wartość tej funkcji. [5] Wybrano dwie heurystyki:

1. metryka Hamminga – wartość obliczana na podstawie liczby elementów, które nie są na swoim miejscu,
2. metryka Manhattan – wartość obliczana na podstawie sumy odległości w poziomie i pionie.

3. Opis implementacji

To sprawozdanie dotyczy wersji 1.0 programu. Algorytmy BFS, DFS oraz A-star zostały zaimplementowane w Javie w wersji 8, choć początkowo były pisane w wersji 21. Jest to bardzo prosta aplikacja konsolowa, uruchamiana poleceniem

```
java -jar ./target/fifteen-game-1.0.jar <algorytm> <strategia>
              <plik_gry> [plik_rozw] [plik_stat],
```

co jest zgodne z przykładowymi wywołaniami opisanymi w instrukcji. Wykorzystywane są również skrypty do weryfikacji poprawności rozwiązań otrzymanych przez każdy algorytm oraz programy, które zostały udostępnione przez autora ćwiczenia. Każde wykonanie algorytmu zapisuje statystyki jego działania w obiekcie GameStat. Gra przechowuje wysokość i szerokość układanki oraz jednowymiarową listę szesnastu liczb rzeczywistych, odpowiadającej ułożeniu liczb. Implementacje algorytmów w tym programie nie należą do najbardziej zaawansowanych, natomiast w zdecydowanej większości przypadków działają poprawnie.

Diagram programu został przedstawiony na rysunku 4.

3.1. Implementacja BFS

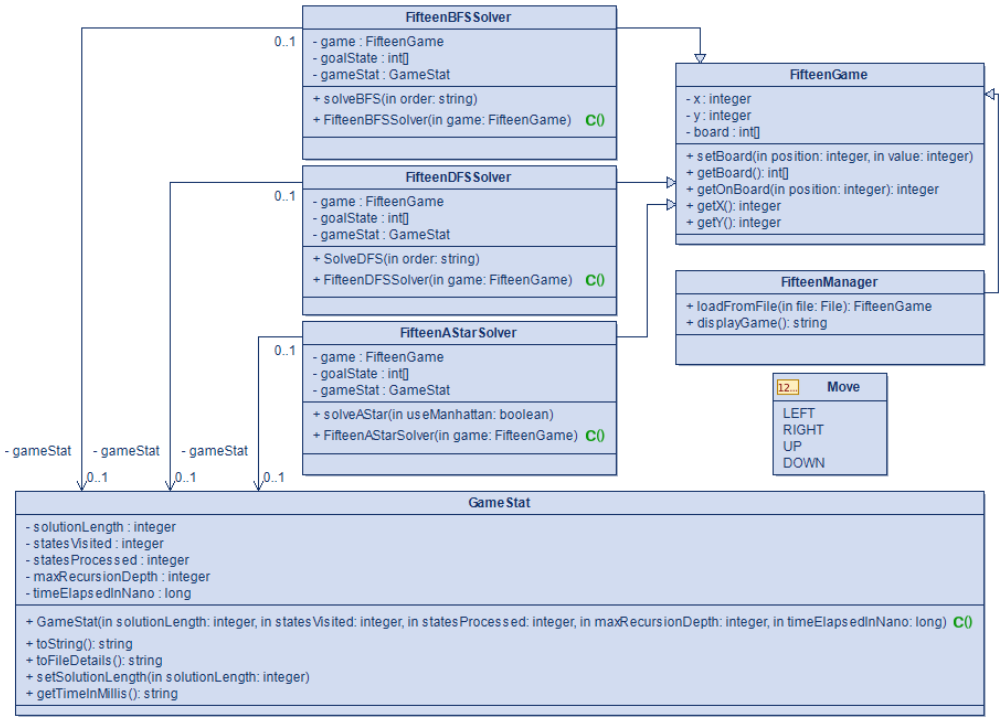
Algorytm BFS został zaimplementowany w klasie `FifteenBFSSolver.java`. Wykorzystuje on kolejkę (Queue), która jest kolekcją FIFO – First In First Out.

3.2. Implementacja DFS

Algorytm DFS został zaimplementowany w klasie `FifteenDFSSolver.java`. Wykorzystuje on stos (Stack), który jest kolekcją LIFO – Last In First Out. Pozwala to zatem na cofanie się oraz założono na stałe maksymalną głębokość schodzenia algorytmu do 20.

3.3. Implementacja A-star

Algorytm A-star został zaimplementowany w klasie `FifteenAStarSolver.java`. Wykorzystuje ona kolejkę priorytetową (PriorityQueue), która, jak opisano w rozdziale 2.3, korzysta z funkcji 1 by obliczyć do pobrania kolejnego stanu o najmniejszej jej wartości.



Rysunek 4. Diagram programu.

4. Materiały i metody

Metodą sprawdzania poprawności algorytmów do pewnego stopnia zaawansowania gry było rozpisanie tych gier ręcznie, w celu weryfikacji rozwiązań, gdyż nie są one zbytnio skomplikowane. Zgodnie z instrukcją, przy pomocy załączonych programów, tj. `puzzlegen.jar` wygenerowano gry o rozmiarze 4×4 i głębokości 7. Następnie wykorzystano program `fifteen-game-1.0.jar` do rozwiązania tych gier, a potem `puzzleval.jar` do sprawdzenia ich poprawności. Proces cały zautomatyzowano, wykorzystując kolejno skrypty: `runprog.sh` oraz `runval.sh`. Do przeprowadzenia analizy, użyto skryptu `extdata.sh`, który podsumował wszystkie wyniki z wygenerowanych plików przez algorytmy. W języku python wykonano wykresy, które zostaną omówione w rozdziale 5.

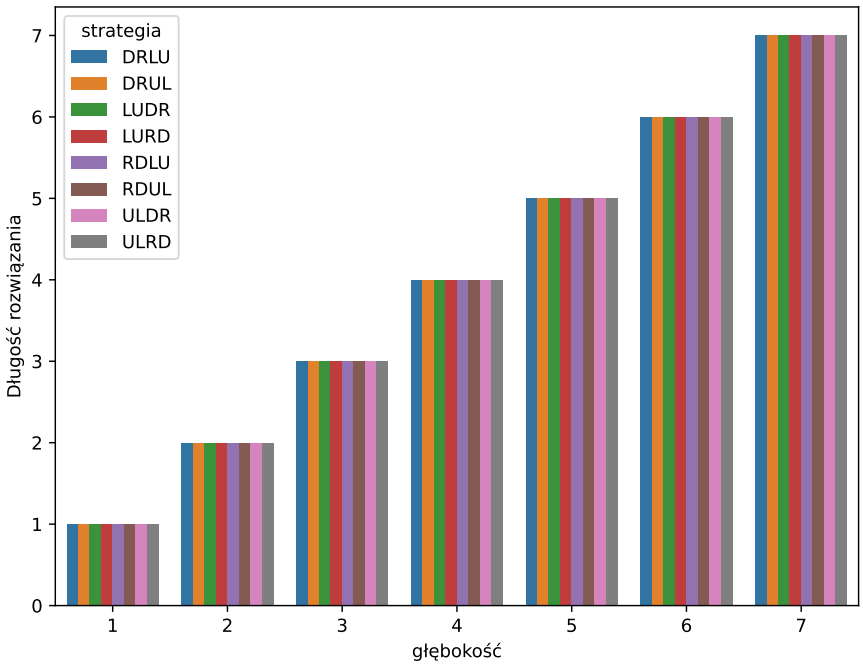
5. Wyniki

W tej sekcji, dla każdego algorytmu (astr, bfs, dfs) przedstawione zostaną wykresy przedstawiające:
— długość znalezionego rozwiązania,
— liczbę stanów odwiedzonych,
— liczbę stanów przetworzonych,
— maksymalną osiągniętą głębokość rekursji,
— czas trwania procesu obliczeniowego.
Dodatkowo ze skryptu runval.sh wiemy, że:

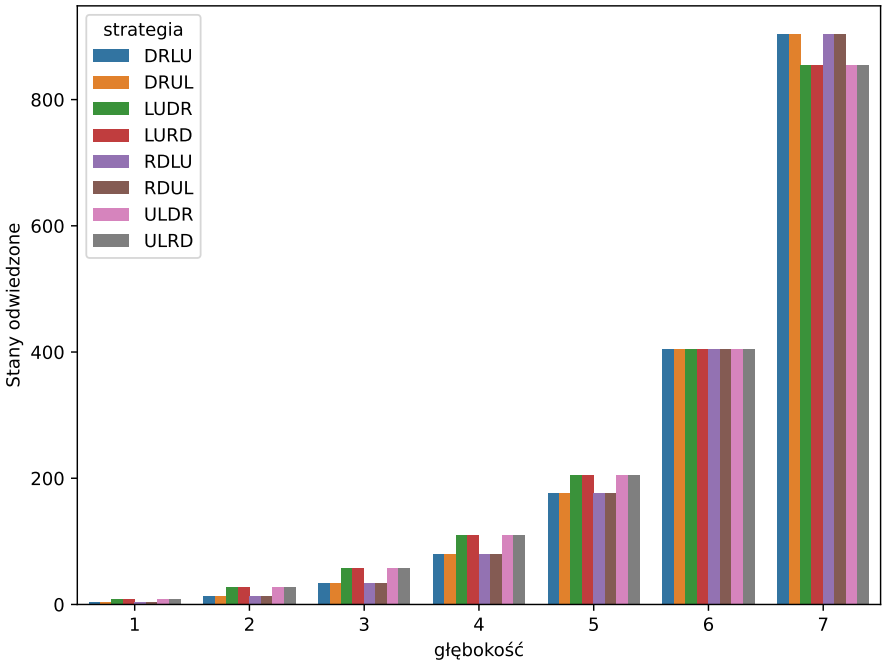
```
----- Summary -----
Correct solutions: 6436
Incorrect solutions: 998
```

co oznacza, że 13,4% rozwiązań było niepoprawnych. Wszystkie niepoprawne wyniki należały do DFS, który zwyczajnie ich nie znalazł, tj. zapisane zostało -1.

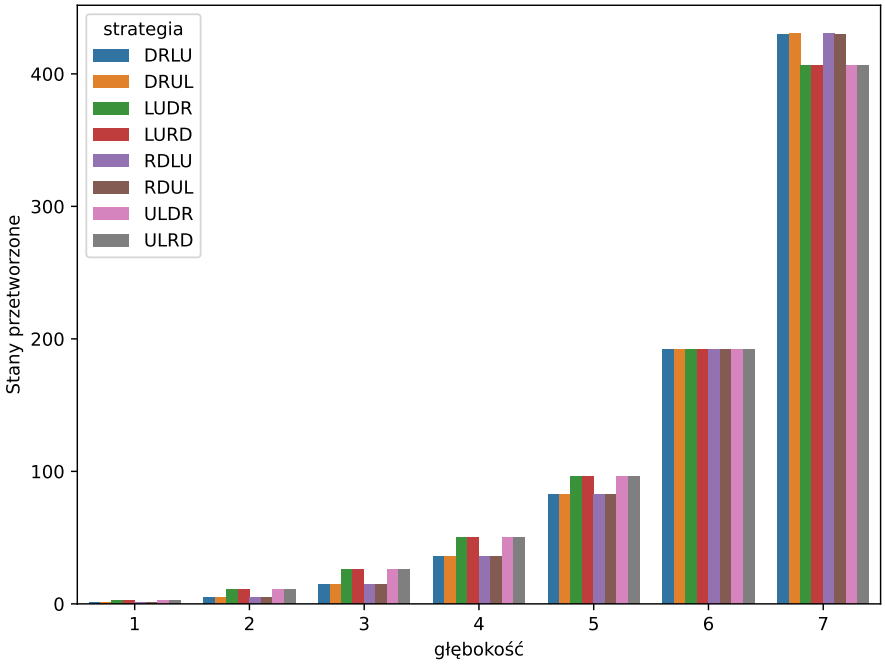
5.1. BFS (bfs)



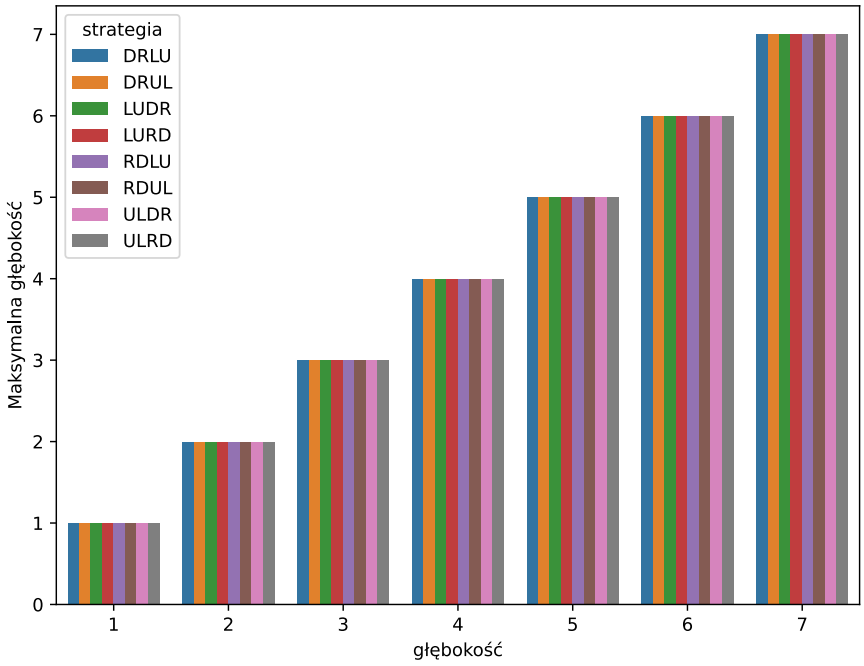
Rysunek 5. Długość znalezionego rozwiązania w algorytmie BFS.



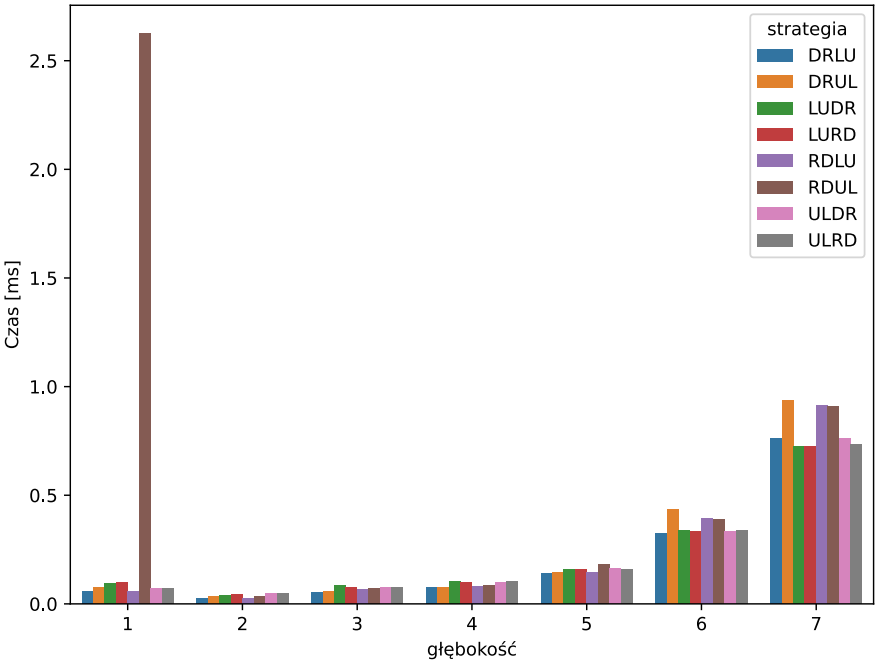
Rysunek 6. Odwiedzone stany w algorytmie BFS.



Rysunek 7. Przetworzone stany w algorytmie BFS.

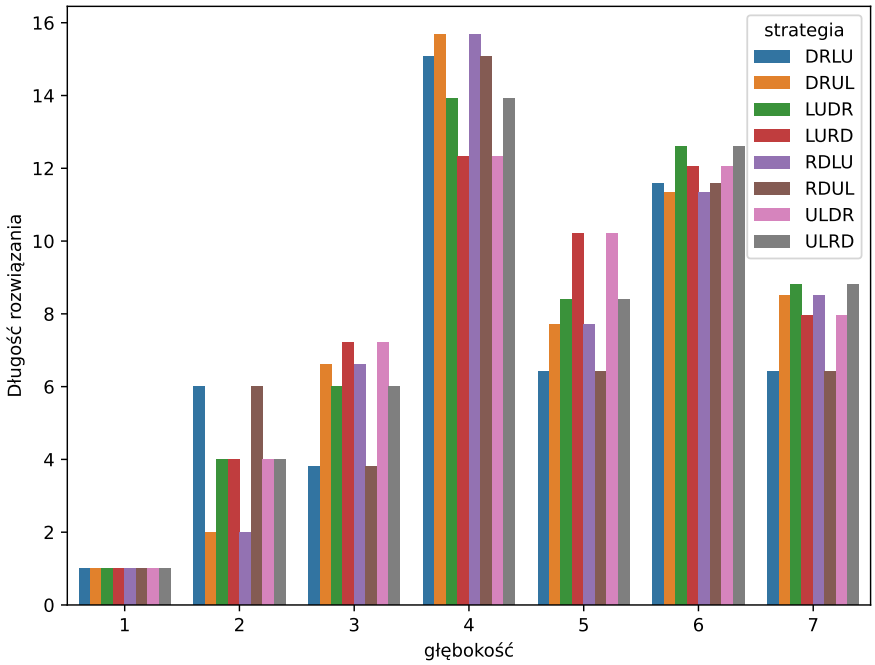


Rysunek 8. Maksymalna osiągnięta głębokość rekursji w algorytmie BFS.

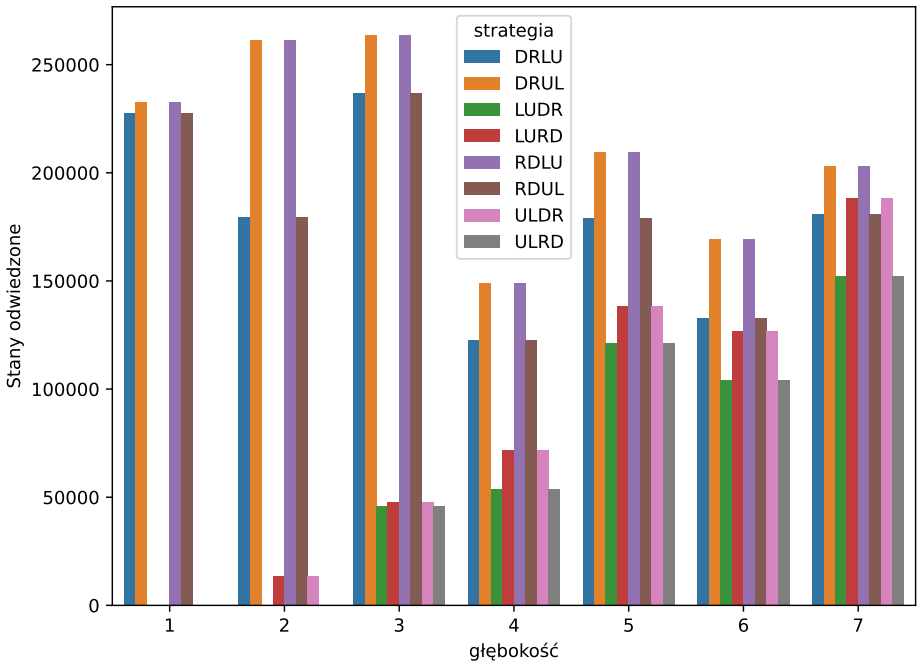


Rysunek 9. Czas trwania procesu obliczeniowego w algorytmie BFS.

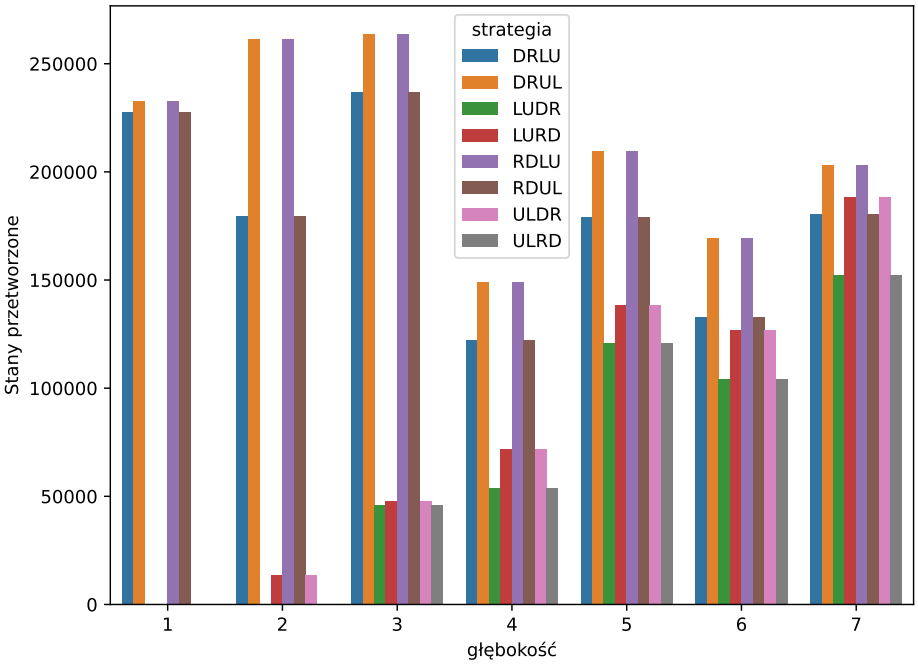
5.2. DFS (dfs)



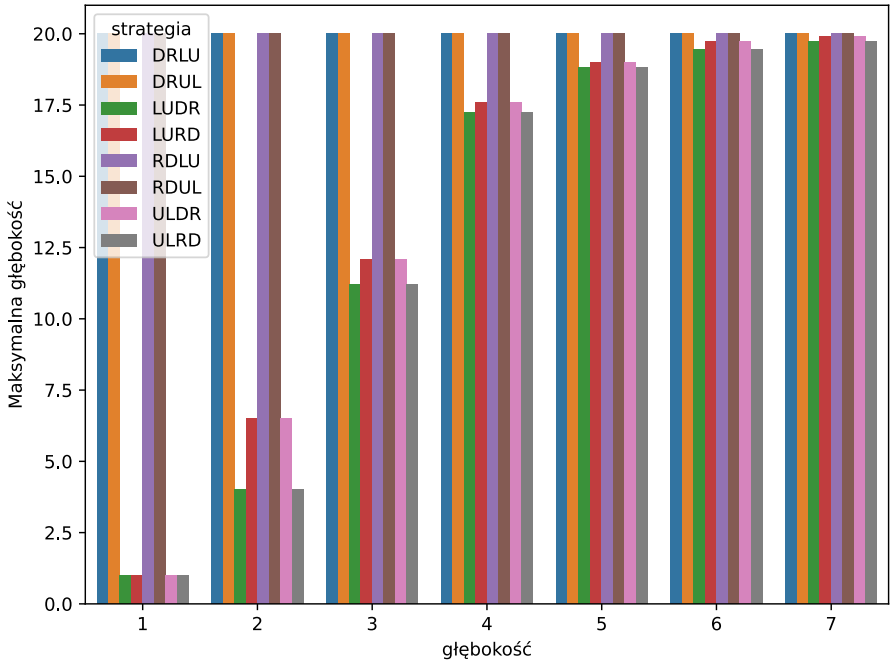
Rysunek 10. Długość znalezionego rozwiązania w algorytmie DFS.



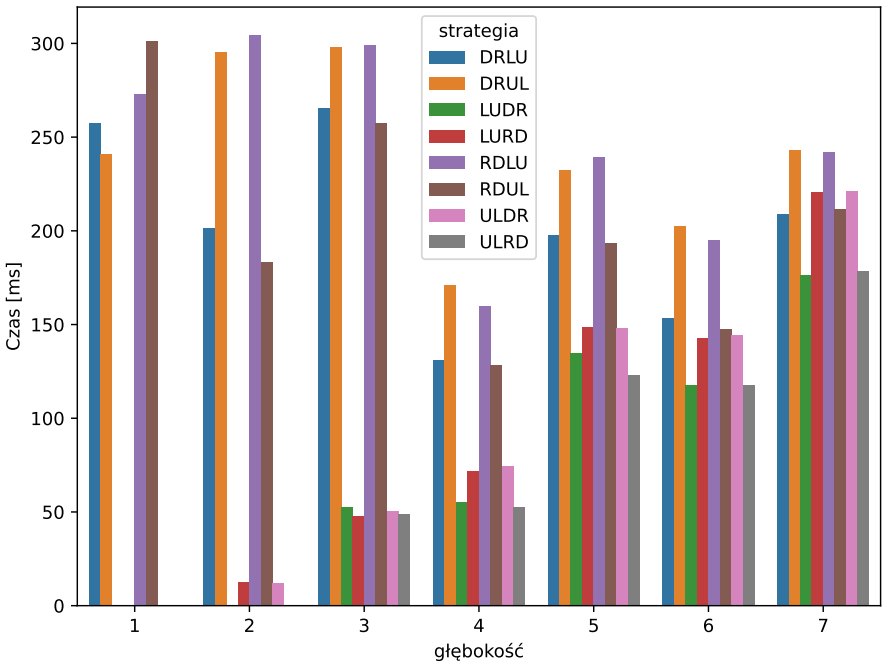
Rysunek 11. Odwiedzone stany w algorytmie DFS.



Rysunek 12. Przetworzone stany w algorytmie DFS.

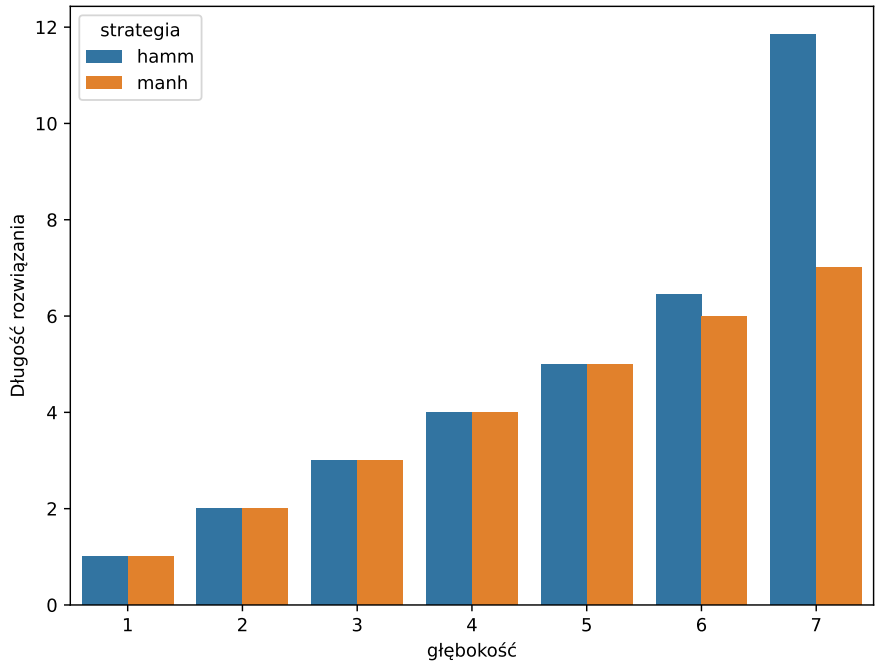


Rysunek 13. Maksymalna osiągnięta głębokość rekursji w algorytmie DFS.

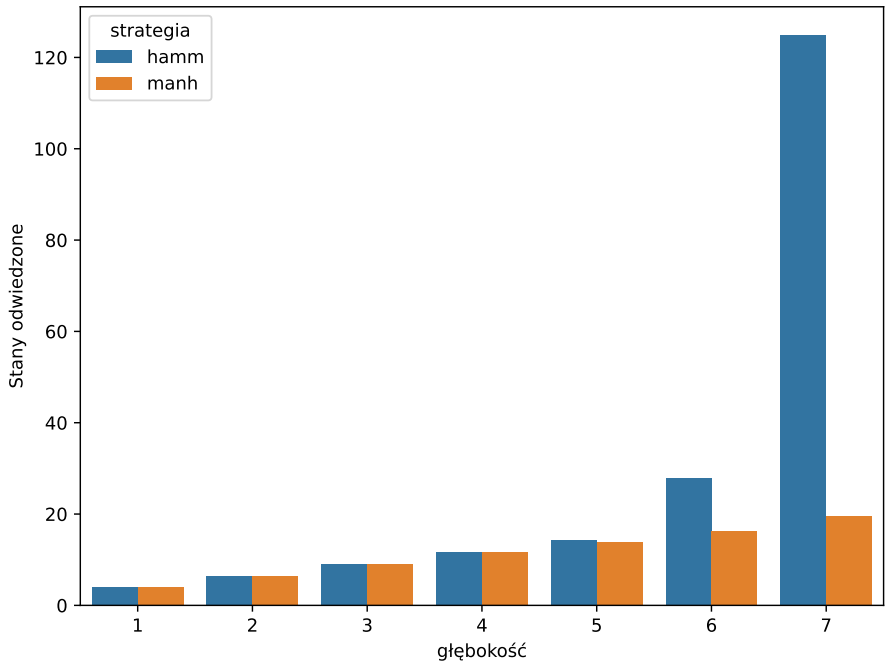


Rysunek 14. Czas trwania procesu obliczeniowego w algorytmie DFS.

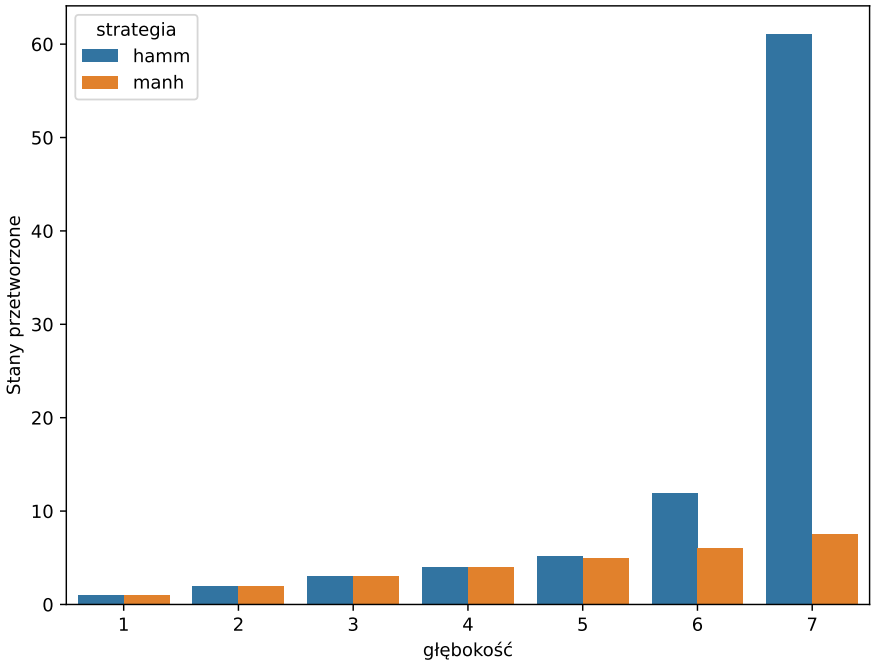
5.3. A-star (astr)



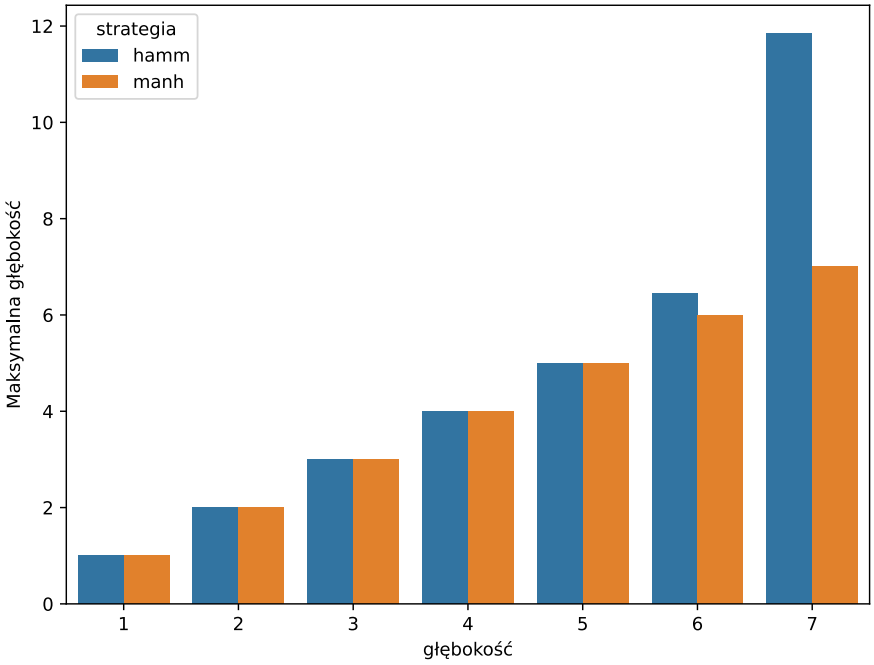
Rysunek 15. Długość znalezionego rozwiązania w algorytmie A-star.



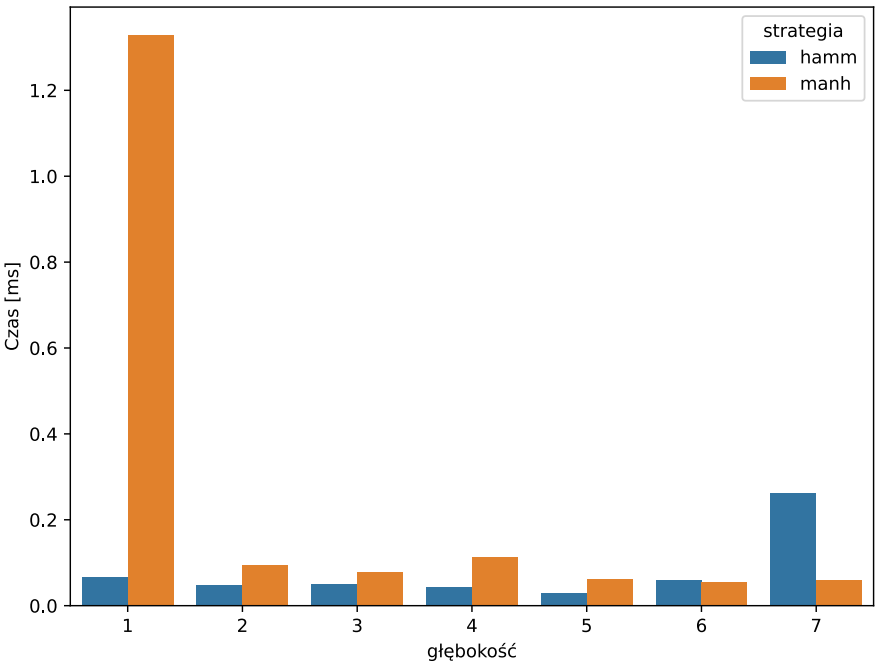
Rysunek 16. Odwiedzone stany w algorytmie A-star.



Rysunek 17. Przetworzone stany w algorytmie A-star.



Rysunek 18. Maksymalna osiągnięta głębokość rekursji w algorytmie A-star.



Rysunek 19. Czas trwania procesu obliczeniowego w algorytmie A-star.

6. Dyskusja

6.1. BFS

Jeden z szybszych algorytmów, który za każdym razem podał wynik. Widać, że długości rozwiązań są takie same jak głębokość gry. Zauważa się pewny błąd w przypadku czasu trwania algorytmu BFS w kolejności RDUL, co widać na rysunku 9. Wartość ta jest najwyższa z możliwych, bo aż dwie i pół sekundy i nie była ona przewidywana.

6.2. DFS

Zdecydowanie najgorszy algorytm do rozwiązywania piętnastki. Prawdopodobnie wynika to z błędnej, a raczej nieoptymalnej implementacji, gdyż algorytm często zapętla lub ostatecznie nawet nie znajduje rozwiązania. Zauważa się również, że temu algorytmowi najdłużej zajmuje znalezienie jakiegokolwiek rozwiązania. Nie ma również pewności, iż takowe zostanie odnalezione, a jeśli nawet, to niekoniecznie będzie ono optymalne bądź poprawne.

6.3. A-star

Od razu można zauważyć, że algorytm jest zdecydowanie najszybszy wśród pozostałych, choć przy odległości Manhattan widać problem podobny do BFS, gdzie czas działania algorytmu jest znacznie dłuższy od pozostałych, bo aż sześciokrotnie od średniej. Zauważono pewny błąd co do strategii Hamminga, gdzie algorytm często przegrywał w wydajności w porównaniu do strategii Manhattana. Szczególnie widać to na wykresie odwiedzonych stanów 16, gdzie są one sześciokrotnie większe niż w przypadku drugiej strategii przy głębokości równej 7.

7. Wnioski

1. Najszybszym algorytmem okazuje się algorytm A-star, w badanym przypadku, z heurystyką odległości Manhattana.
2. Algorytm DFS spisał się najgorzej z całej trójki. Istnieją przypadki, gdzie nie znajduje on rozwiązania, a jeśli je odnajdzie, to nie jest ono optymalne.
3. Algorytmy BFS i A-star zawsze znalazły rozwiązanie.
4. Różnica odwiedzonych oraz przetworzonych stanów w algorytmie DFS kontra BFS czy A-star jest diametralna.
5. Algorytm A-star z heurystyką Hamminga działał gorzej niż z heurystyką Manhattana.
6. Java jest dobrym językiem do implementacji tych algorytmów.

Literatura

[1] Gupta, K. (2019, luty 6). What is the difference between BFS and DFS algorithms. FreelancingGig. https://freelancinggig.com/...

[2] Przeszukiwanie wszerz. (2024, lipiec 15). Wikipedia, wolna encyklopedia. Dostęp 14:53, kwiecień 7, 2025, Dostępny w Internecie: https://pl.wikipedia.org/...

[3] Przeszukiwanie w głąb. (2024, sierpień 21). Wikipedia, wolna encyklopedia. Dostęp 14:53, kwiecień 7, 2025, Dostępny w Internecie: https://pl.wikipedia.org/...

[4] Chrobot, A. (2019, czerwiec 2). Podstawy Programowania 2 - Algorytmy dfs i bfs. https://achilles.tu.kielce.pl/...

[5] Raczyńska, J. Algorytm A*. https://elektron.elka.pw.edu.pl/...