Antoni Karwowski 229809 229908@edu.p.lodz.pl
Michał Gebel 229879 229879@edu.p.lodz.pl
Kacper Wiśniewski 230037 230037@edu.p.lodz.pl

Zadanie 2.: Metody przeszukiwania grafów wszerz i w głąb

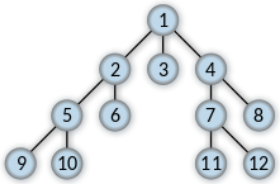
1. Cel

Istotą zadania jest porównanie sposobu działania dwóch algorytmów (BFS, DFS) w kryteriach:
1. Typu implementacji grafu
2. Ilości węzłów
3. Ilości połączeń na węzeł
4. Skierowanych / nieskierowanych połączeń

2. Wprowadzenie

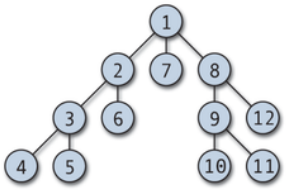
Przeszukiwanie wszerz (breadth-first search [BFS]) i wzdłuż (depth-first search [DFS]) to jedne z najprostszych algorytmów przeszukiwania grafów. Algorytmy te działają prawidłowo zarówno dla grafów skierowanych jak i nieskierowanych.

BFS polega na rozpoczęciu przeszukiwania od pewnego węzła i odwiedzeniu wszystkich osiągalnych z niego węzłów sąsiadujących. Jeśli na danym poziomie nie znajdzie rozwiązania, przechodzi o poziom głębiej i znów przeszukuje wszystkie węzły sąsiadów na danym poziomie.



Rysunek 1: BFS - kolejność przeszukiwania węzłów

DFS polega na badaniu wszystkich krawędzi wychodzących z podanego wierzchołka. Po zbadaniu wszystkich krawędzi wychodzących z danego wierzchołka algorytm powraca do wierzchołka, z którego dany wierzchołek został odwiedzony.



Rysunek 2: DFS - kolejność przeszukiwania węzłów

3. Opis implementacji

Algorytmy zostały zaimplementowane w języku programowania Python. Grafy wykorzystywane do przeszukiwań były generowane przy pomocy biblioteki networkx, która dostarczała metody pozwalające przedstawić grafy w różnych postaciach, które następnie były konwertowane na klasy utworzone przez nas do przeszukiwania.

Główną klasą reprezentującą graf jest Graph. Zawiera on informację o ilości węzłów, węźle początkowym i końcowym oraz metodą is_end, która jest pomocniczo wykorzystywana w implementacji algorytmów BFS i DFS.

Istnieją cztery reprezentacje grafów - tablica, lista sąsiedztwa, macierz incydencji oraz macierz sąsiedztwa. Są to kolejno klasy:

ArrayGraph,

NeighourhoodListGraph,

NeighbourhoodMatrixGraph

IncidentMatrixGraph

Wszystkie te klasy dziedziczą z Graph i różnią się implementacją metody inicjalizującej graf (biblioteka networx), oraz poszukwania sąsiadów dla węzła.

Ponadto istnieją dwie metody, które wykonują przeszukiwanie poprzez zastosowanie właściwego (podanego w nazwie metody) algorytmu:

breadth_first_search

depth_first_search

Sposób ich działania jest zgodny z pseudokodem przedstawionym na wykładzie

4. Materiały i metody

Każda wartość obliczana w eksperymentach (zarówno czasy wykonania algorytmów jak i długość rozwiązania końcowego) jest średnią z obliczeń wykonanych na od 100 do 500 losowych, wygenerowanych grafach o identycznych parametrach.

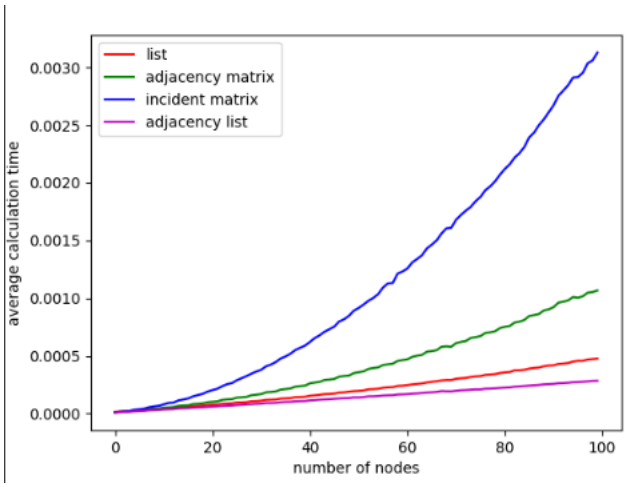
Przy eksperymentach 2 (porównanie implementacji) oraz 4 (porównanie połączeń skierowanych i nieskierowanych) przedstawiono jedynie wyniki dla algorytmu BFS ponieważ nie zarejestrowano widocznej różnicy w wynikach działania obu algorytmów w prezentowanych aspektach.

Wykonano porównanie:

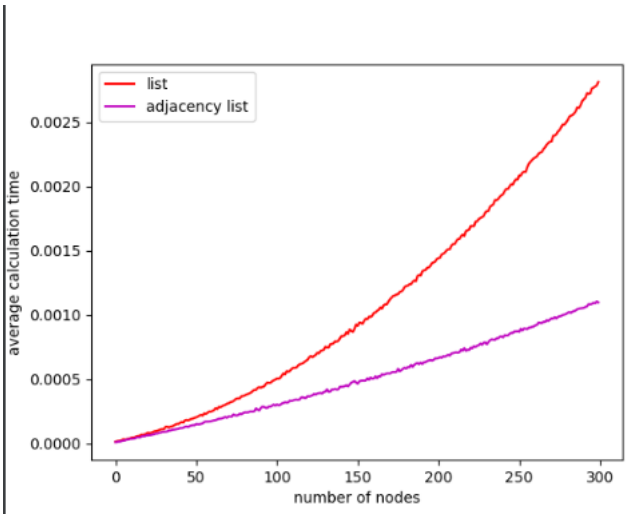
1. Algorytmów BFS i DFS względem średniej długości rozwiązania
2. Typu implementacji grafu względem złożoności obliczeniowej algorytmów i średniego czasu obliczeń
3. Algorytmów BFS i DFS względem czasu wykonania przy różnych ilościach połączeń
4. Grafów skierowanych i nieskierowanych pod kątem wpływu na prędkość wykonywania algorytmów

5. Wyniki

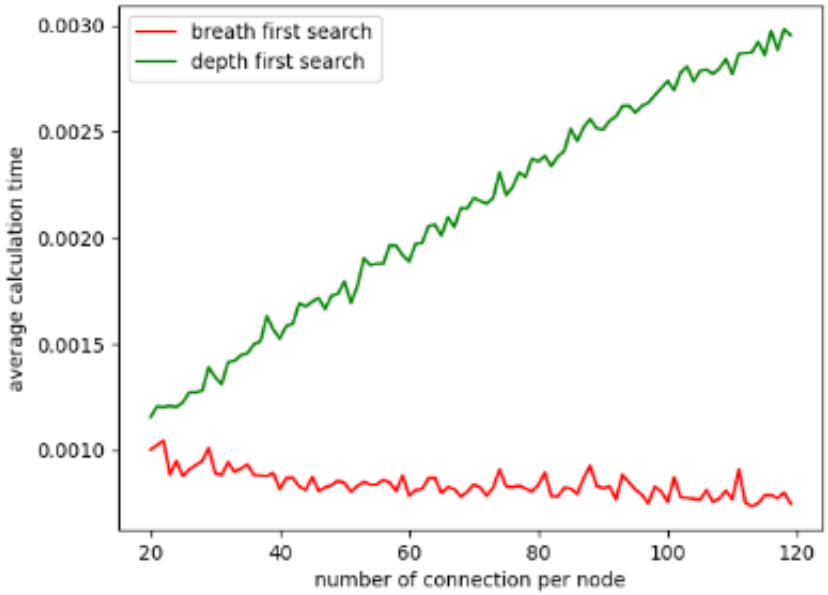
Rysunek 3: Różnica pomiędzy BFS i DFS pod kątem średniej długości rozwiązania w zależności od liczby węzłów



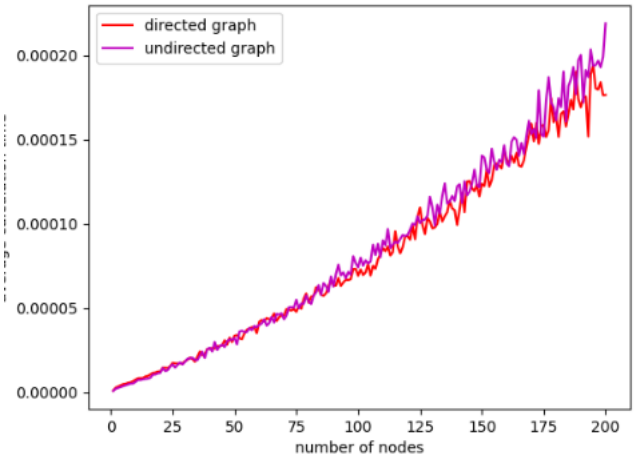
Rysunek 4: złożoność obliczeniowa poszczególnych implementacji grafu w zależności od liczby węzłów



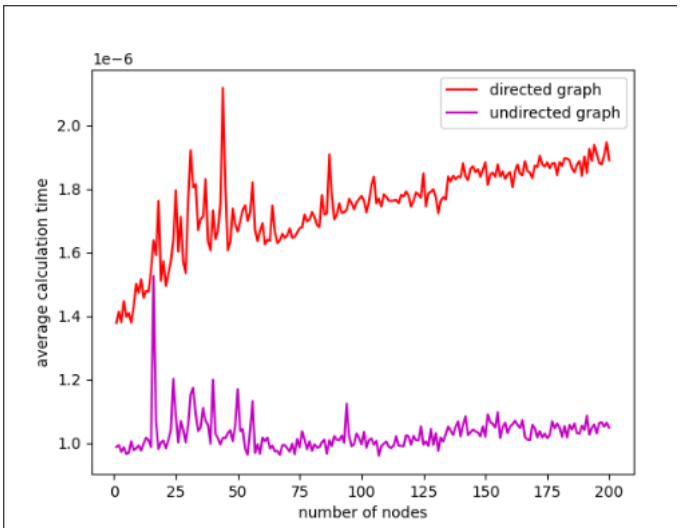
Rysunek 5: złożoność obliczeniowa listy sąsiedztwa oraz tablicy jako implementacji grafu w zależności od liczby węzłów (fragment rysunku 4)



Rysunek 6: różnica w czasie wykonania BFS i DFS dla stałej liczby węzłów (300) w zależności od średniej liczby połączeń przypadającej na jeden węzeł



Rysunek 7: różnica w czasie wykonania algorytmów dla grafów skierowanych i nieskierowanych przy średnio 20 połączeniach na węzeł względem liczby węzłów



Rysunek 8: różnica w czasie wykonania algorytmów dla grafów skierowanych i nieskierowanych przy średnio 3 połączeniach na węzeł względem liczby węzłów

6. Wnioski

1. Przy losowych grafach (szukany węzeł znajduje się w losowym miejscu) metoda BFS generuje średnio krótsze (mniejsza liczba kroków), a więc i bardziej optymalne rozwiązania [rysunek 3].
2. Sposób implementacji grafu wpływa znacząco na złożoność obliczeniową i prędkość wykonywania obu algorytmów. liniową złożoność obliczeniową ma implementacja listy sąsiedztwa. pozostałe implementacje mają złożoność wielomianową. implementacje przedstawione w kolejności od najbardziej wydajnej:
   1. Lista Sąsiedztwa [rysunek 4 oraz 5]
   2. Tablica [rysunek 4 oraz 5]
   3. Macierz Sąsiedztwa [rysunek 4]
   4. Macierz Incydencji [rysunek 4]

3. Przy stałej ilości punktów w grafie (300 punktów) czas wykonywania algorytmów zmienia się w zależności od rosnącej średniej liczby połączeń przypadającej na jeden punkt:
   1) czas wykonywania algorytmu DFS rośnie [rysunek 6]
   2) czas wykonywania algorytmu BFS nieznacznie maleje [rysunek 6]

   Czas wykonywania algorytmu DFS oraz BFS zmienia się względem siebie w zależności od proporcji liczby punktów do liczby połączeń oraz od typu grafu, jednak zależności te nie zostały zbadane [rysunek 6]
4. Nie zarejestrowano zauważalnego wpływu wykorzystania grafów nieskierowanych / skierowanych na prędkość wykonywania obliczeń w żadnym z poniższych aspektów:
   1) porównanie BFS / DFS
   2) zmienna liczba węzłów
   3) typ implementacji grafów

   Zauważono jedynie nieznaczny wzrost wydajności obu algorytmów dla wszystkich implementacji w zależności od liczby połączeń przypadającej na węzeł [rysunek 7 oraz 8]

Literatura

[1] *Wykład 2*, `https://ftims.edu.p.lodz.pl/pluginfile.php/176866/mod_resource/content/1/Metaheurystyki%20wyklad02.pdf`.

[2] Angielska wikipedia `https://en.wikipedia.org/wiki/Breadth-first_search`

[3] Geeks For Geeks BFS `https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/`