Robert Dudziński
Michał Zadrożny

ALHE - dokumentacja wstępna

1. Temat projektu

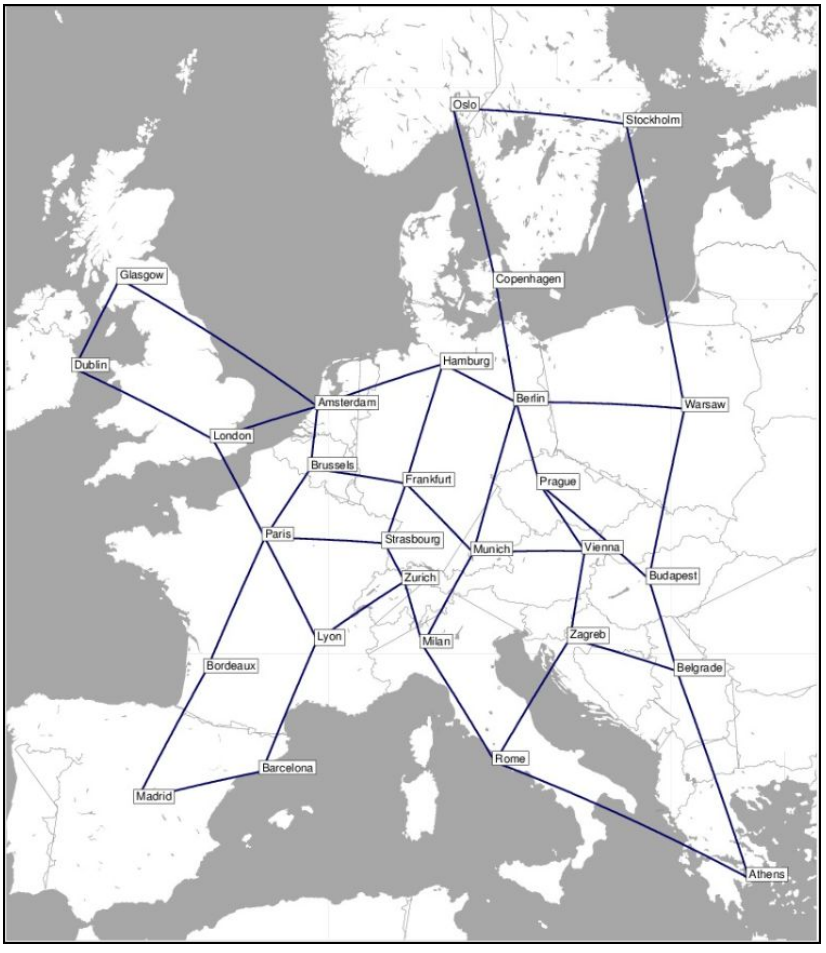
W ramach projektu zaimplementowany i przetestowany zostanie algorytm A*. Zadanie algorytmu będzie polegało na znalezieniu ścieżki o najmniejszej wadze od punktu A do B (jako dane posłużą europejskie miasta). Zaimplementowane zostaną również inne algorytmy wykonujące te zadanie, w celu porównania wydajności i efektywności ich działania.

2. Dane wejściowe

Wejściem aplikacji będzie plik o następującej strukturze:

- liczba miast, liczba bezpośrednich połączeń
- lista miast wraz z ich współrzędnymi geograficznymi (potrzebne do obliczania odległości między miastami)
- lista par identyfikatorów miast posiadających bezpośrednie połączenie
- wagą krawędzi będzie odległość w linii prostej między dwoma miastami

Dane pochodzą ze strony http://sndlib.zib.de/home.action



3. Zastosowane struktury danych

Dane wejściowe zostaną zapisane w programie w postaci grafu. Miasto będzie reprezentowane jako wierzchołek grafu - w wierzchołku zapisane będą takie informacje, jak id miasta oraz jego współrzędne x i y (w zależności od algorytmu w wierzchołku będą przechowywane też dodatkowe informacje). Aby właściwie reprezentować sąsiedztwo miast zdefiniowane w pliku wejściowym, w każdym wierzchołku przechowywany będzie wektor wskaźników na wierzchołki sąsiadów. Graf sprowadza się więc do listy wskaźników na wierzchołki. Struktura będzie budowana na początku działania programu poprzez czytanie kolejnych linii pliku wejściowego.

4. Opis algorytmów

- Brutalne przeszukanie grafu - algorytm Deep First Search

Do poprawnego działania algorytmu konieczne jest przechowywanie w każdym wierzchołku informacji, czy wierzchołek został odwiedzony w ramach aktualnie rozpatrywanej ścieżki (bool visited).
Aby znaleźć rozwiązanie, przechowywana jest najlepsza dotychczas znaleziona ścieżka oraz jej długość.

Algorytm jest rekurencyjny. Przeszukiwanie rozpoczyna od odwiedzenia wierzchołka startowego (jest to pierwszy punkt roboczy). Następnie przeglądani są wszyscy sąsiedzi punktu roboczego. Jeśli któryś z sąsiadów nie został jeszcze odwiedzony (sprawdzana jest wartość pola visited), to

- wierzchołek jest dodawany do aktualnej ścieżki,
- odległość od punktu roboczego do sąsiada dodawana do długości aktualnej ścieżki,
- wartość jego pola visited jest zmieniana na true,
- sąsiad staje się nowym punktem roboczym (wywołanie rekurencyjne).

Warunkiem wyjścia z rekurencji jest dotarcie do wierzchołka końcowego - wtedy porównywana jest długość aktualnej ścieżki z dotychczas najlepszą i w przypadku znalezienia lepszego rozwiązania jest ono zapamiętywane. Drugą możliwością zakończenia wywołań rekurencyjnych jest sytuacja, gdy długość aktualnej ścieżki będzie większa od dotychczas najlepszej - skoro nie ma szans na znalezienie lepszego rozwiązania, to rozwijanie obecnej ścieżki powinno się zakończyć. Po powrocie z rekurencji konieczna jest zmiana wartości pola visited na false w wierzchołku, który był do tej pory rozpatrywany - dzięki temu będzie on mógł wchodzić w skład kolejnych ścieżek budowanych przez algorytm w przyszłości.

W wyniku działania algorytmu rozpatrzone zostaną wszystkie możliwe ścieżki w grafie pomiędzy wierzchołkiem startowym i końcowym (z wyłączeniem tych których rozwijanie przedwcześnie zostało zakończone - ich długość nie dawała szans na znalezienie optymalnego rozwiązania).

- Algorytm A*

  Każdy wierzchołek będzie przechowywał dodatkowo:
  - najmniejszą dotychczas znalezioną odległość od wierzchołka startowego (długość najlepszej znalezionej ścieżki między wierzchołkiem startowym a rozpatrywanym)
  - id wierzchołka poprzedzającego na tej ścieżce (umożliwi to rekonstrukcję optymalnej ścieżki)
  - wartość funkcji f w tym wierzchołku, będącej sumą odległości od wierzchołka startowego (koszt dotarcia do wierzchołka) oraz heurystycznego oszacowania kosztu dojścia do wierzchołka końcowego - odległość miast w linii prostej).

  Algorytm korzysta z dwóch (początkowo pustych) wektorów wskaźników na wierzchołki:
  - wektor wierzchołków do rozpatrzenia (openSet)
  - wektor rozpatrzonych wierzchołków (closedSet)

  Algorytm rozpoczyna swoje działanie od umieszczenia wierzchołka startowego w openSet. Dopóki nie zostanie znaleziony wierzchołek końcowy, algorytm rozpatrywać będzie kolejne wierzchołki z openSet, w każdej iteracji wybierając ten o najmniejszej wartości funkcji f (zdefiniowanej wyżej).
  Dla każdego sąsiada rozpatrywanego wierzchołka sprawdzana jest zapisana w nim najlepsza znaleziona odległość od startu. Jeśli ścieżka przechodząca przez aktualnie rozpatrywany wierzchołek jest krótsza, to pola sąsiada (wartość funkcji f, długość najlepszej ścieżki, wierzchołek poprzedzający) są odpowiednio aktualizowane, a sam wierzchołek (pod warunkiem, że nie trafił wcześniej do closedSet) trafa do openSet i może zostać wybrany przez algorytm jako nowy punkt roboczy. Dotychczasowy wierzchołek roboczy trafia do closedSet.

  Algorytm gwarantuje, że pierwsze natrafienie na wierzchołek końcowy jest optymalnym rozwiązaniem - do wierzchołka doprowadziła najkrótsza ścieżka.

- Algorytm Dijkstry

  Algorytm działa bardzo podobnie do A*, z tą różnicą, że nie korzysta z heurystycznego oszacowania kosztu dotarcia do wierzchołka końcowego. Nowym wierzchołkiem roboczym staje się ten wierzchołek z openSet, do którego koszt dotarcia był najmniejszy.

- Przeszukanie grafu - algorytm Breadth First Search

  BFS nie zostanie wykorzystany, ponieważ w BFS na drzewie odwiedzilibyśmy wszystkie wierzchołki tylko raz. W BFS na grafie też odwiedzilibyśmy tylko raz, więc nie widzielibyśmy różnych ścieżek dojścia.

Można by wykorzystać BFS do podziału wierzchołków na warstwy i potem dla każdego z nich znaleźć najlepszego poprzednika (iterując się od wewnętrznej warstwy), ale wtedy nie zostanie obsłużona sytuacja, w której najkrótsza ścieżka prowadzi "na około", czyli np. najkrótsza ścieżka do punktu z warstwy trzeciej korzysta z jednego punktu warstwy czwartej.

5. Techniczny sposób realizacji oraz testowania algorytmów

Program zostanie zaimplementowany w języku C++.

Każdy z algorytmów zostanie zaimplementowany w oddzielnej klasie. Każda z klas będzie dziedziczyć po klasie bazowej problemu, dzięki czemu wszystkie algorytmy będą miały wspólny interfejs i łatwiej będzie zbierać wyniki działania (weryfikacja poprawności i mierzenie czasu obliczeń).
Każdy z algorytmów będzie mógł zdefiniować specyficzne dla siebie parametry oraz przechowywać dane wejściowe w inny sposób, aby był do nich możliwie najefektywniejszy dostęp.

W celu porównywania wyników zwracanych przez różne algorytmy będzie mierzony czas obliczeń potrzebny do uzyskania optymalnego rozwiązania (nie będzie wliczony w to czas potrzebny na wczytanie i przygotowanie danych wejściowych). Można przetestować wszystkie możliwe trasy A -> B i porównać, który algorytm sumarycznie zwróci najlepszy czas.

Dane wejściowe ze strony sndlib.zib.de są zapisane w pliku XML. Potrzebne dane z tego pliku zostaną eksportowane do pliku tekstowego, aby były łatwe do odczytu przez program.

6. Wyniki

Ze względu na stosunkowo mało graf, nie było możliwe porównanie czasu wykonania programu dla dwóch różnych algorytmów. W celu porównania algorytmu A* oraz DFS zostały zliczone odwiedzone wierzchołki podczas szukania optymalnej ścieżki łączącej dwa dowolne miasta.

Algorytmy zostały uruchomione dla wszystkich możliwych kombinacji miasta początkowego i końcowego. Łącznie jest 765 takich kombinacji. Został porównany stosunek liczby odwiedzonych wierzchołków podczas poszukiwania do liczby wierzchołków, które tworzą optymalną ścieżkę.

Porównanie A* z DFS bez odcięcia:
A* better: 756
DFS better: 0
Same result: 0

W każdym przypadku A* potrzebowała odwiedzić mniej wierzchołków.

How many times more visited vertices than final path length (avg after all tests):
A*: 1.34301
DFS: 8683.13

Porównanie A* z DFS z odcięciem:
A* better: 747
DFS better: 0
Same result: 9

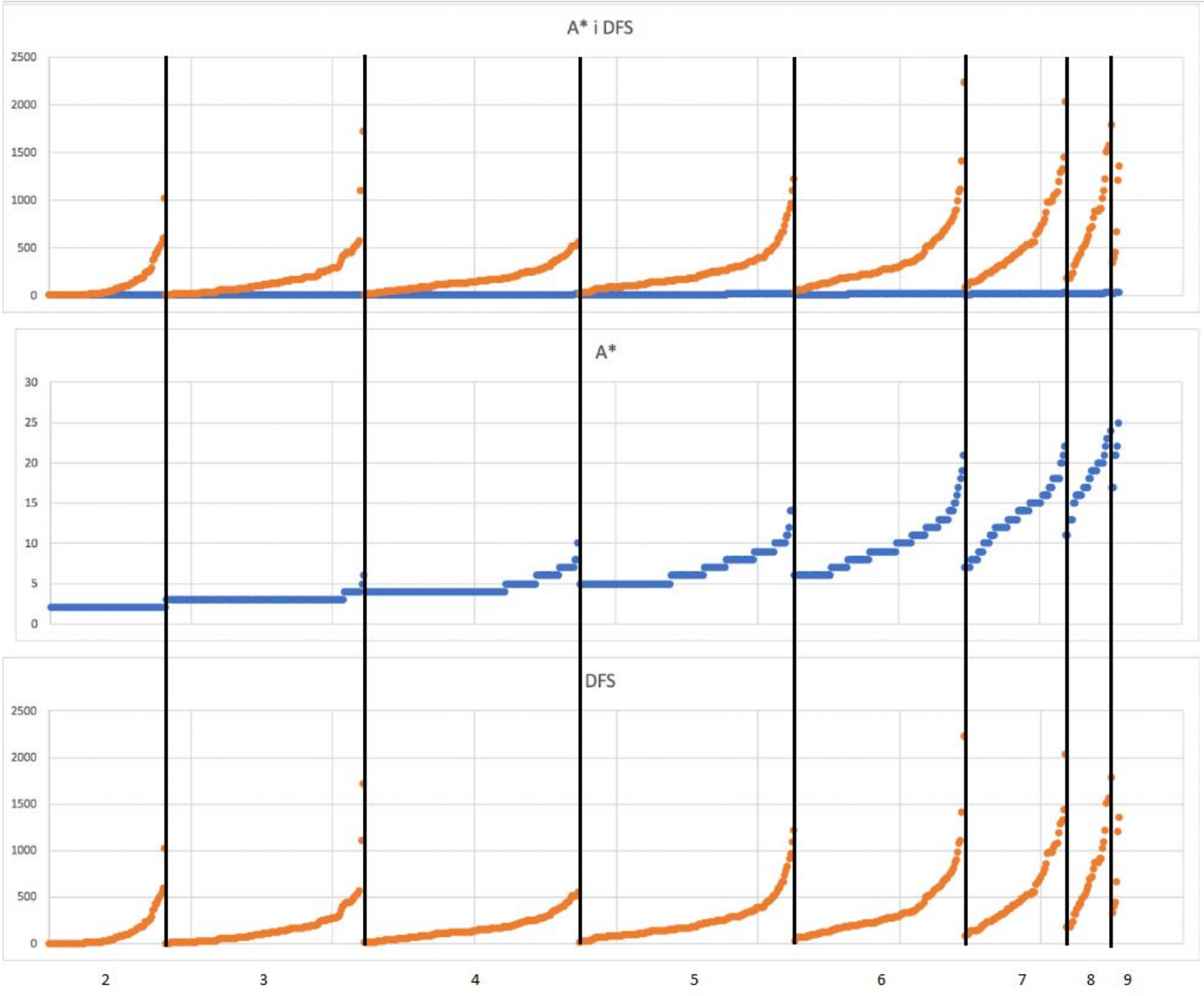
W przypadku DFS z odcięciem również nie było sytuacji w której A* odwiedziłaby większą liczbę wierzchołków niż DFS.

How many times more visited vertices than final path length (avg after all tests):
A*: 1.34301
DFS: 54.4946

Przy zastosowaniu odcięcia w DFS udało się znacząco zmniejszyć liczbę odwiedzonych wierzchołków. Nadal jednak zostało odwiedzonych wiele razy więcej wierzchołków niż miało to miejsce przy A*.

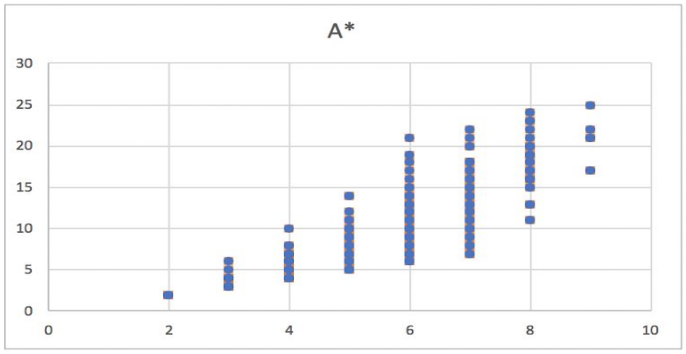
Poniżej znajdują się wykresy przedstawiające ilość odwiedzonych wierzchołków przez A* i DFS dla optymalnych ścieżek zawierających podaną liczbę węzłów. Górny wykres jest połączeniem dwóch niższych wykresów w celu porównania skali.

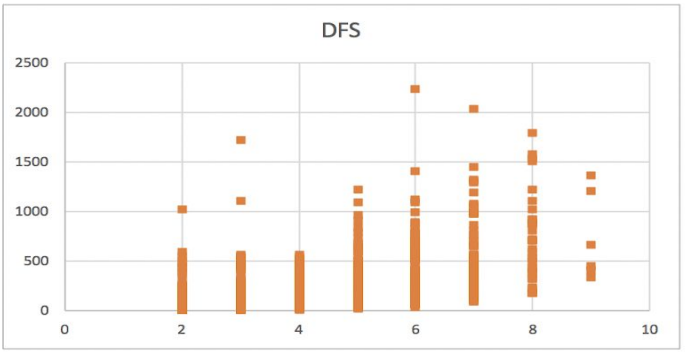
W celu wygodniejszej analizy łatwiejszego wyciąganie informacji z wykresów, ilość odwiedzonych wierzchołków została posortowana w ramach każdego fragmentu, który obrazuje liczbę odwiedzanych wierzchołków dla poszczególnych długości ścieżki optymalnej.



Poniższe wykresy przedstawiają ilość odwiedzin wierzchołków podczas wyszukiwania optymalnej ścieżki. Pozioma oś oznacza ile wierzchołków ma optymalna ścieżka. Dla każdej z wartości są określone punkty oznaczające ilość odwiedzin wierzchołków przy znajdowaniu tej ścieżki. Łącznie jest 756 punktów na każdym z wykresów, czyli są przedstawione dane dla wszystkich możliwych wywołań algorytmów.

W przypadku DFS na ilość odwiedzonych wierzchołków może wpływać kolejność wybierania kolejnych węzłów. Może być to określone np. poprzez wybór wierzchołka np. z największą lub najmniejszą wartością heurystyczną lub może nie być to ustalone, czyli bierzemy pierwszy nieodwiedzony wierzchołek. W tym drugim przypadku na kolejność wybierania węzłów ma wpływ kolejność ich ułożenia w strukturach danych algorytmu, tym samym kolejność wczytywania poszczególnych danych o grafie do struktur algorytmu może mieć wpływ na ostateczną postać wykresu.


A*
30
25
20
15
10
5
0
0
2
4
6
8
10


DFS
2500
2000
1500
1000
500
0
0
2
4
6
8
10