Politechnika
Wrocławska

Marcin Karasiewicz
Rafał Kowalski

Projektowanie algorytmów
i metody sztucznej inteligencji

Grafy - "Jak dojadę?"

Spis treści

1 Założenia projektowe

Projekt zakłada stworzenie programu, który wyszukuje najbardziej optymalną czasowo trasę między zadanymi stacjami tramwajowymi we Wrocławiu. Aby to osiągnąć program ma wykorzystywać 3 algorytmy grafowe: Breath First Search, Deep First Search oraz A Star. W projekcie można wyróżnić następujące zadania:

- Pobranie danych dotyczących połączeń tramwajowych ze strony miasta
- Przeparsowanie nadmiaru danych z plików w formacie *.xml na potrzebne dla programu dane w formacie *.csv
- Implementacja tworzenia grafu na podstawie danych z plików *.csv
- Wizualizacja grafu w programie Gnuplot
- Implementacja zadanych algorytmów grafowych
- Wizualizacja działania algorytmów animacją w programie Gnuplot
- Analiza działania algorytmów
- Stworzenie graficznego interfejsu użytkownika
- Wyciagnięcie wniosków

2 Parser danych XML

Parser danych został napisany w Pythonie w wersji 3. Użyto biblioteki Etree, służącej do poruszania się po plikach XML. Parser wyszukuje korzeń, następnie szuka *wariantu* trasy, następnie wyszukuje na danym *wariancie* wszystkie *przystanki*, a następnie odczytuje z nich ich *id*, *nazwę*, *ulicę* oraz *czas*, ile potrzebuję tramwaj na dojechanie do kolejnej stacji na trasie. Na końcu dane te zostają zapisane do pliku **.cvs*

3 Implementacja tworzenia grafu

Przeparsowane dane zostały podzielone na pliki zawierające pojedyńcze trasy tramwajowe. Graf jest tworzony dynamicznie podczas odczytu danych z tych plików. Algorytm tworzenia grafu prezentuje się następująco:

1. Otwórz plik z trasą tramwajową.
2. Wczytaj nazwę tramwaju.
3. Wczytaj dane o pierwszej stacji na trasie z kolejnej linii pliku.
4. Przeszukaj graf w celu sprawdzenia, czy istnieje w grafie już stacja o takich parametrach, jak właśnie wczytane.

5. Jeśli istnieje: ustaw na nią wskaźnik *acctual*; jeśli nie istnieje: utwórz w grafie nową stację o parametrach takich, jak wczytane dane i ustaw na nią wskaźnik *acctual*.
6. Wczytaj dane o kolejnej stacji na trasie z kolejnej linii pliku.
7. Przeszukaj graf w celu sprawdzenia, czy istnieje w grafie już stacja o takich parametrach, jak właśnie wczytane.
8. Jeśli istnieje: ustaw na nią wskaźnik *next*; jeśli nie istnieje: utwórz w grafie nową stację o parametrach takich, jak wczytane dane i ustaw na nią wskaźnik *next*.
9. Utwórz połączenie do sąsiada stacji, na którą wskazuje *acctual* o poarametrach: Sąsiadująca stacja - stacja, na którą wskazuje *next*; Czas podróży do sąsiada - wczytany czas; Nazwa tramwaju realizującego połączenie - wczytana nazwa z początku pliku.
10. Ustaw wskaźnik *acctual* na stację, na którą wskazuje *next*.
11. Powatarzaj kroki od 6. do10. aż do końca danych w pliku.

Algorytm Wykonywany jest dla wszystkich plików zawierających dane o połączeniach tramwajowych.
Kluczowym elementem algorytmu jest krok 10. Zapewnia on tworzenie zawsze odpowiednich połączeń do sąsiadów kolejno wczytywanych stacji.

4 Breadth First Search

4.1 Opis

Algorytm, jak jego nazwa wskazuje, szuka trasy między zadanymi stacjami przeszukując graf wszerz. Zapewnia to znalezienie zawsze najlepszej trasy, ale kosztem zbadania dużej ilości połączeń. Wyszukana przez algorytm trasa stanowi wzorzec do porównania dla tras znalezionych przez pozostałe algorytmy.

4.2 Implementacja

Realizację algorytmu zapewnia kolejka na którą odkładamy, najpierw stację początkową. Kolejne kroki powtarzają się do znalezienia stacji końcowej, lub opustoszenia kolejki:

1. Zdejmij stację z kolejki.
2. Sprawdź, czy zdjęta staja nie jest stacją końcową (jeśli jest, to następny krok nie jest już wykonywany).
3. Dodaj do kolejki wszystkich jej sąsiadów, którzy nie zostali wcześniej dodani.

Sprawdzenie, czy stacja została już odwiedzona zapewnia kolorowanie stacji numerami 0 - stacja nie odwiedzona, 1 - stacja znajduje się już w kolejce, 3 - wszyscy sąsiedzi tej stacji zostali już dodani do kolejki.
Gdy algorytm znajdzie stację końcową odkłada ją na stos, następnie kolejno stacje, z których do niej dojechał, aż do stacji początkowej. Pozwala to na późniejsze wypisanie trasy ze stosu oraz lokalizacji stacji na trasie, potrzebnych do wizualizacji.

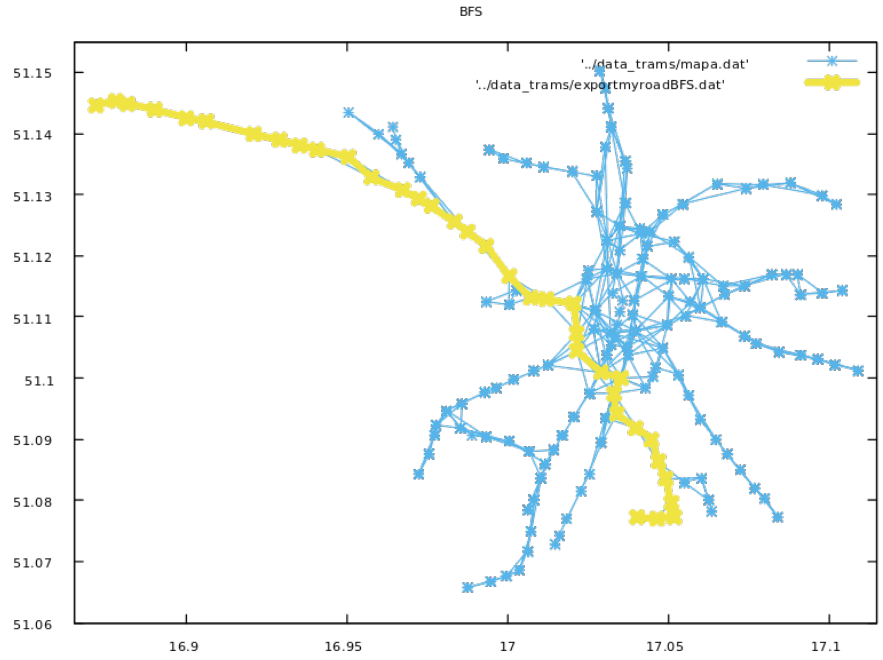
4.3 Testy

Trasa	Czas przejazdu	Zbadane połączenia	Czas algorytmu [ms]
LEŚNICA - GAJ	51	173	$1.5007e-04$
KOZANÓW - BISKUPIN	38	175	$1.6949e-04$
RYNEK - BISKUPIN	20	162	$1.6641e-04$
PL.GRUNWALDZKI - DW GŁÓWNY	9	32	$4.2369e-05$

- Długa trasa między stacjami bardzo odległymi na krańcach grafu

  Stacje: LEŚNICA - GAJ

Id: 21030 - 18201
Czas przejazdu: 51
Liczba zbadanych połączeń: 173
Czas działania algorytmu: 0.000150065 ms



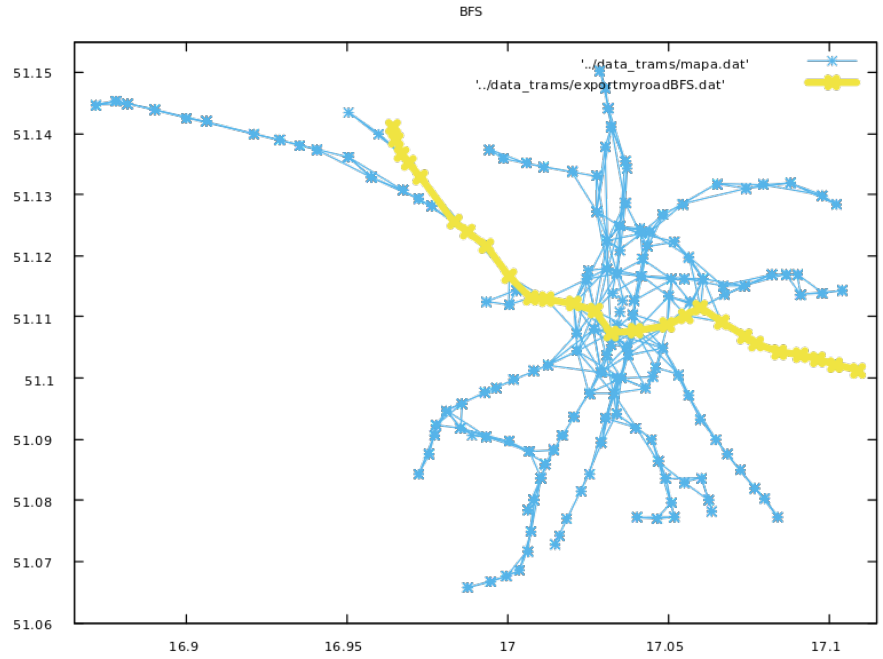
Rysunek 1: Leśnica - Gaj BFS

Czy zaleziona trasa jest optymalna?
Trasa wzorcowa

- Trasa między stacjami krańcowymi grafu

  Stacje: KOZANÓW - BISKUPIN

Id: 12610 - 24416
Czas przejazdu: 38
Liczba zbadanych połączeń: 175
Czas działania algorytmu: 0.000169492 ms



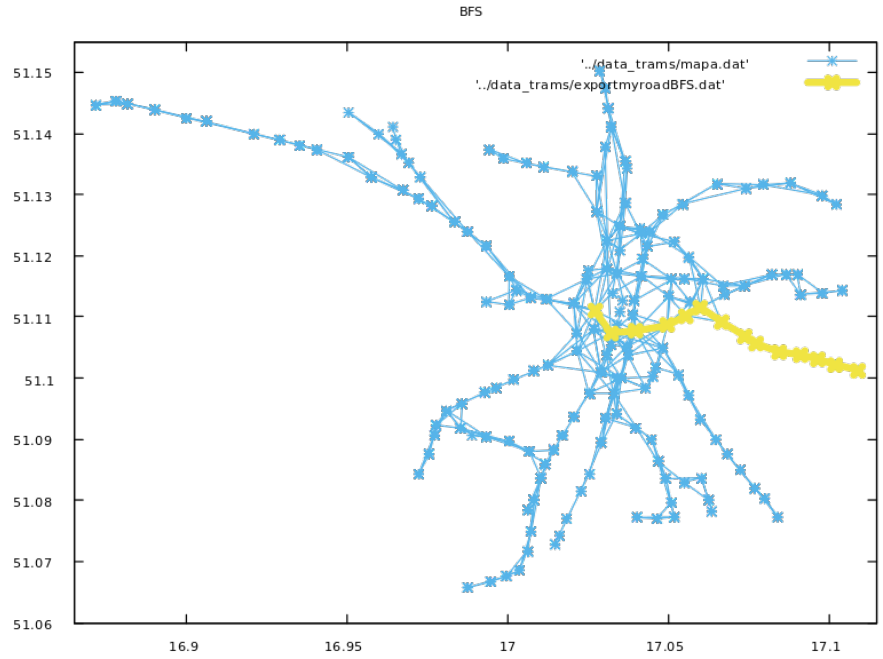
Rysunek 2: Kozanów - Biskupin BFS

Czy zaleziona trasa jest optymalna?
Trasa wzorcowa

- Trasa między stacją z środka do stacji na krańcu grafu

  Stacje: RYNEK - BISKUPIN

Id: 10003 - 24416
Czas przejazdu: 20
Liczba zbadanych połączeń: 162
Czas działania algorytmu: 0.000166407 ms



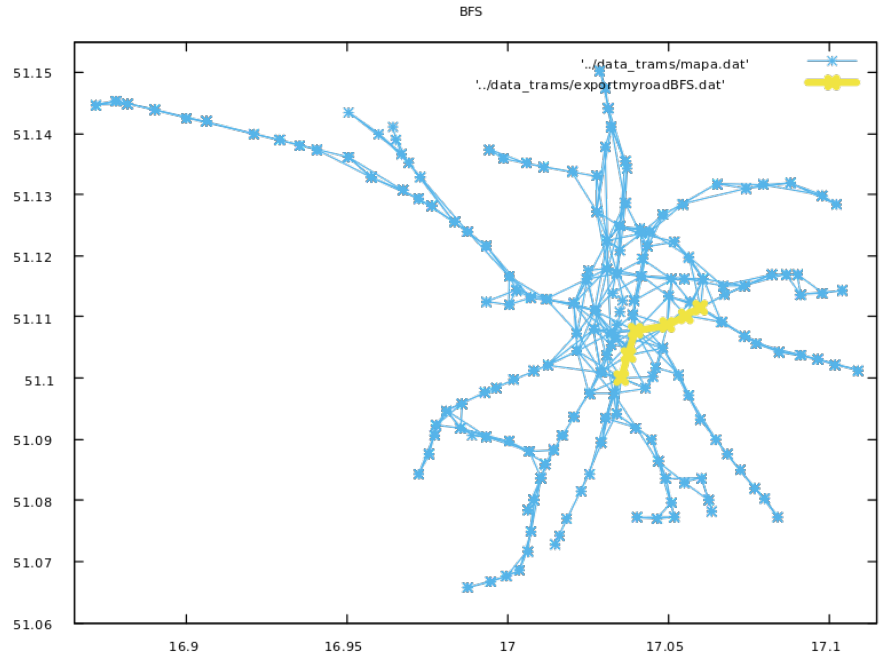
Rysunek 3: Rynek - Biskupin BFS

Czy zaleziona trasa jest optymalna?
Trasa wzorcowa

- Trasa między stacjami znajdującym się w środku grafu

  Stacje: PL.GRUNWALDZKI - DW GŁÓWNY

Id: 20823 - 10226
Czas przejazdu: 9
Liczba zbadanych połączeń: 32
Czas działania algorytmu: $4.2369e-05$ ms



Rysunek 4: PL. Grunwaldzki - Dworzec Gł. BFS

Czy zaleziona trasa jest optymalna?
Trasa wzorcowa

5 Deep First Search

5.1 Opis

Algorytm, jak jego nazwa wskazuje, szuka trasy między zadanymi stacjami przeszukując graf wgłąb. Oznacza to, że w przeciwieństwie do BFS, w pierwszej kolejności przeszukuje trasę, którą zaczął badać, do jej końca i dopiero potem zaczyna przeszukiwać kolejną trasę. Powoduje to, w większości przypadków, znalezienie dłuższej trasy w porównaniu do trasy wzorcowej wyznaczonej przez BFS, jednakże kosztem mniejszej ilości zbadanych połączeń.

5.2 Implementacja

Różnica w implementacji algorytmu DFS względem BFS sprowadza się do zamienienia kolejki na stos.

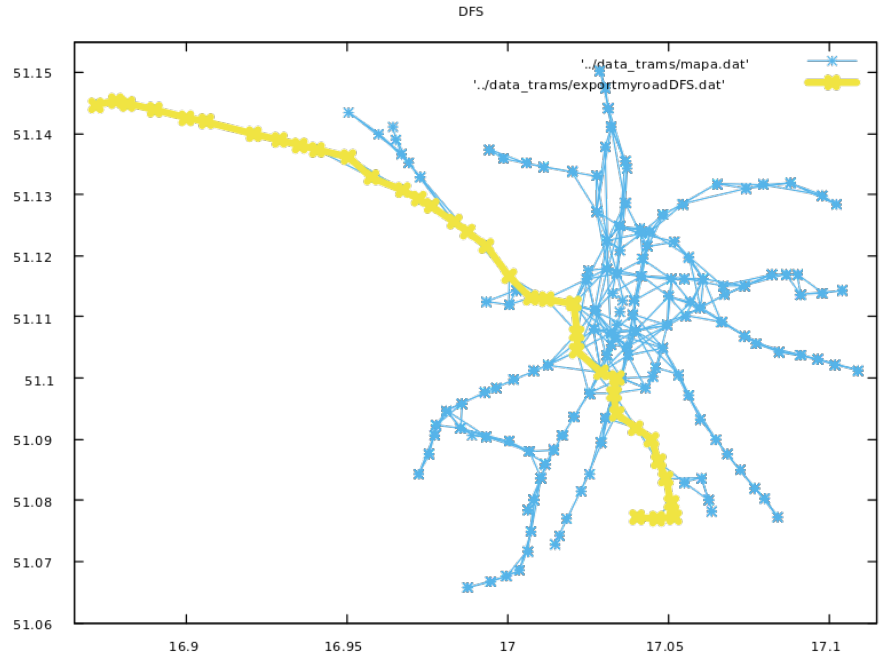
5.3 Testy

Trasa	Czas przejazdu	Zbadane połączenia	Czas algorytmu [ms]
LEŚNICA - GAJ	51	45	$3.0945e-05$
KOZANÓW - BISKUPIN	48	146	$1.1721e-04$
RYNEK - BISKUPIN	30	138	$8.4303e-05$
PL.GRUNWALDZKI - DW GŁÓWNY	24	96	$1.0685e-04$

- Długa trasa między stacjami bardzo odległymi na krańcach grafu

  Stacje: LEŚNICA - GAJ

Id: 21030 - 18201
Czas przejazdu: 51
Liczba zbadanych połączeń: 45
Czas działania algorytmu: $3.0945e-05$ ms



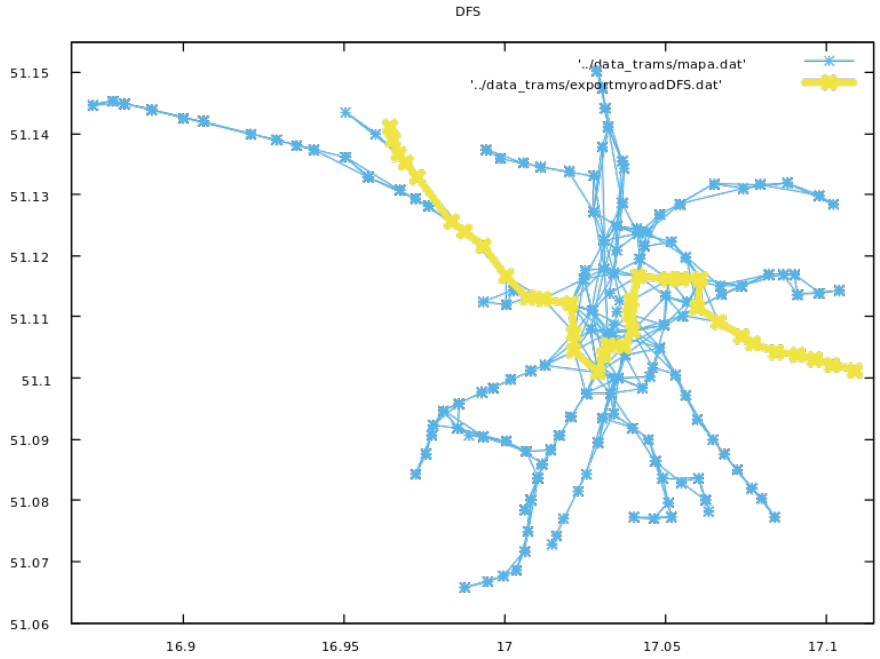
Rysunek 5: Leśnica - Gaj DFS

Czy zaleziona trasa jest optymalna?
Trasa taka sama jak przy BFS. Wynika to z przypadkowego ustawienia danych w grafie i szybkiego przeszukania szukanej trasy, przez co liczba zbadanych połączeń jest kilkukrotnie mniejsza niż w BFS.

- Trasa między stacjami krańcowymi grafu

  Stacje: KOZANÓW - BISKUPIN

Id: 12610 - 24416
Czas przejazdu: 48
Liczba zbadanych połączeń: 146
Czas działania algorytmu: 0.000117204 ms



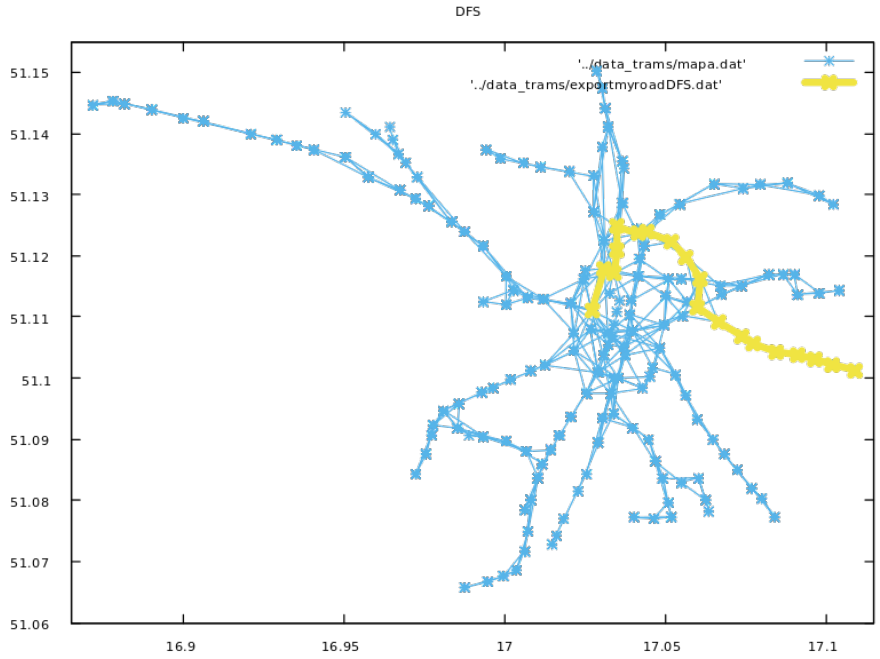
Rysunek 6: Kozanów - Biskupin DFS

Czy zaleziona trasa jest optymalna?
Trasa o 10 minut dłuższa niż trasa znaleziona przez BFS. Różnica 10 minut nie dyskwalifikuje tej trasy z bycia optymalną. Można ją polecić osobą chcącym zwiedzić miasto zza szyby tramwaju. Algorytmowi wystarczyło o 29 mniej zbadanych połączeń niż przy BFS, by wyznaczyć trasę.

- Trasa między stacją z środka do stacji na krańcu grafu

  Stacje: RYNEK - BISKUPIN

Id: 10003 - 24416
Czas przejazdu: 30
Liczba zbadanych połączeń: 138
Czas działania algorytmu: $8.4303e-05$ ms



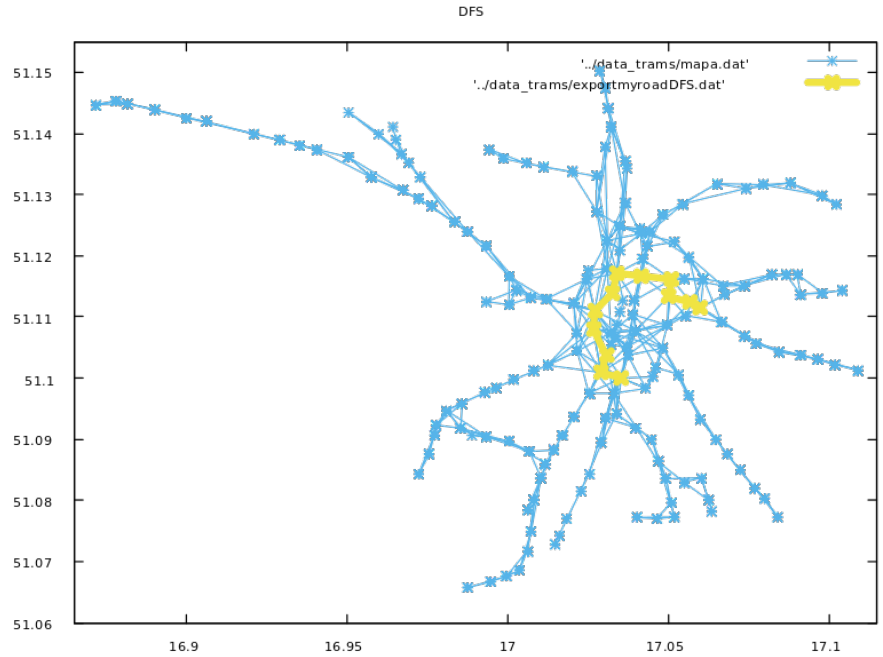
Rysunek 7: Rynek - Biskupin DFS

Czy zaleziona trasa jest optymalna?
Trasa o 10 minut dłuższa niż trasa znaleziona przez BFS. Różnica 10 minut nie dyskwalifikuje tej trasy z bycia optymalną. Algorytmowi wystarczyło o 24 mniej zbadanych połączeń niż przy BFS, by wyznaczyć trasę.

- Trasa między stacjami znajdującym się w środku grafu

  Stacje: PL.GRUNWALDZKI - DW GŁÓWNY

Id: 20823 - 10226
Czas przejazdu: 24
Liczba zbadanych połączeń: 96
Czas działania algorytmu: 0.00010685 ms



Rysunek 8: PL. Grunwaldzki - Dworzec Gł. DFS

Czy zaleziona trasa jest optymalna?
Trasa o 15 minut dłuższa niż trasa znaleziona przez BFS. Różnica 15 minut dyskwalifikuje tą trasę z bycia optymalną. Podróżnik najprawdopodobniej ceni sobie najszybszy przejazd między stacjami w mieście, będącymi stosunkowo blisko siebie. Algorytm potrzebował o 64 więcej zbadanych połączeń niż BFS. Wynika to z badania wgłąb tras nie prowadzących do celu przed zbadaniem szukanej trasy.

6 A Star

6.1 Opis

Algorytm by wybrać połączenie, które w danym kroku będzie badać korzysta z funkcji:
$f = g + 300 * h$, gdzie:

g - suma odległości między stacjami na trasie od stacji początkowej do stacji, na której właśnie jest algorytm,

h - heurystyka, będąca odległością w linii prostej między stacją, na której właśnie jest algorytm, a stacją końcową.

Waga h musiała zostać zwielokrotniona w stosunku do wagi g, ponieważ zrealizowany graf jest mapą miasta ze współrzędnymi geograficznymi, gdzie przy tak małych wartościach g i h, h traciła zupełnie na swojej wartości. Funkcja f jest poprawna dla grafów będących mapą, gdzie długość krawędzi odpowiada odległości między wierzchołkami. W naszym przypadku długość krawędzi stanowi czas przejazdu między stacjami, który jest proporcjonalny do odległości. Pozwala to na używanie tej funkcji, jednak dla dokładności algorytmu stosujemy odległość rzeczywistą.
Algorytm zawsze bada to połączenie, dla którego funkcja f przyjmuje najmniejszą wartość w porównaniu do pozostałych możliwych do zbadania w danym kroku połączeń.

6.2 Implementacja

W przypadku algorytmu A* zamiast kolejki czy stosu wykorzystujemy listę. Wybór ten został podyktowany potrzebą dostępu do jej dowolnego elementu. Procedura algorytmu w stosunku do BFS została wzbogacona o wybór stacji którą wybierzemy do badań trasy i usuniemy z listy. Wybór podyktowany jest warunkiem podanym w opisie algorytmu. Jego realizację zapewnia badanie długości wektora, będącego różnicą wektorów o początku zawsze w punkcie $[0, 0]$ i końcach o współrzędnych geograficznych stacji:

element g - suma długości różnic wektorów sąsiadujących ze sobą stacji na trasie od stacji początkowej, do stacji którą sprawdza algorytm,

element h - długość różnicy wektorów stacji końcowej i stacji, którą sprawdza algorytm.

Algorytm sprawdza w ten sposób wszystkie stacje do których może dojechać ze stacji, na której jest w danej chwili i wybiera przejście do stacji, dla której funkcja f przyjmuje najmniejszą wartość.

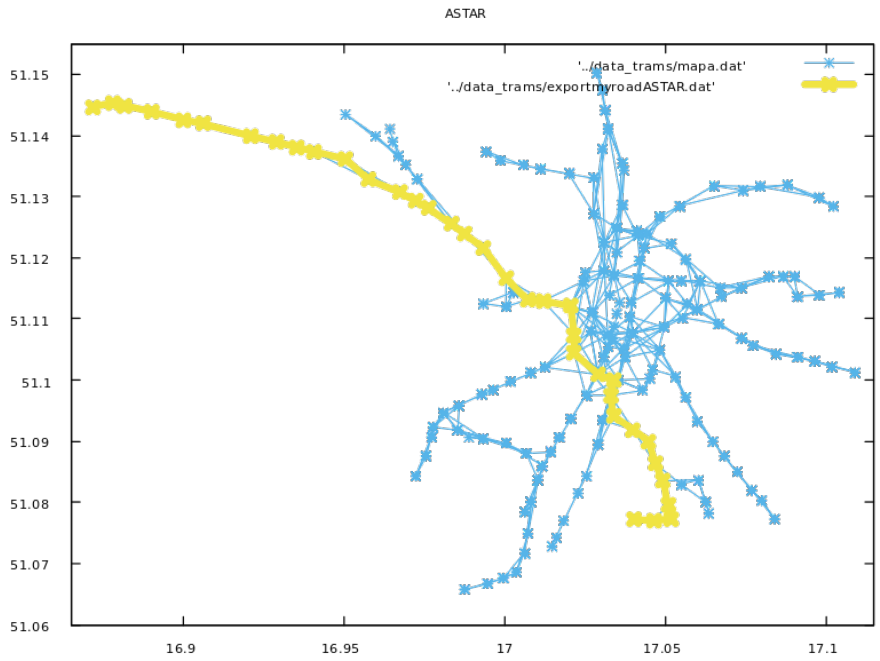
6.3 Testy

Trasa	Czas przejazdu	Zbadane połączenia	Czas algorytmu [ms]
LEŚNICA - GAJ	51	34	$4.2438e-05$
KOZANÓW - BISKUPIN	38	33	$5.6273e-05$
RYNEK - BISKUPIN	20	21	$2.4711e-05$
PL.GRUNWALDZKI - DW GŁÓWNY	9	5	$1.0836e-05$

- Długa trasa między stacjami bardzo odległymi na krańcach grafu

  Stacje: LEŚNICA - GAJ

Id: 21030 - 18201
Czas przejazdu: 51
Liczba zbadanych połączeń: 34
Czas działania algorytmu: $4.2438e-05$ ms



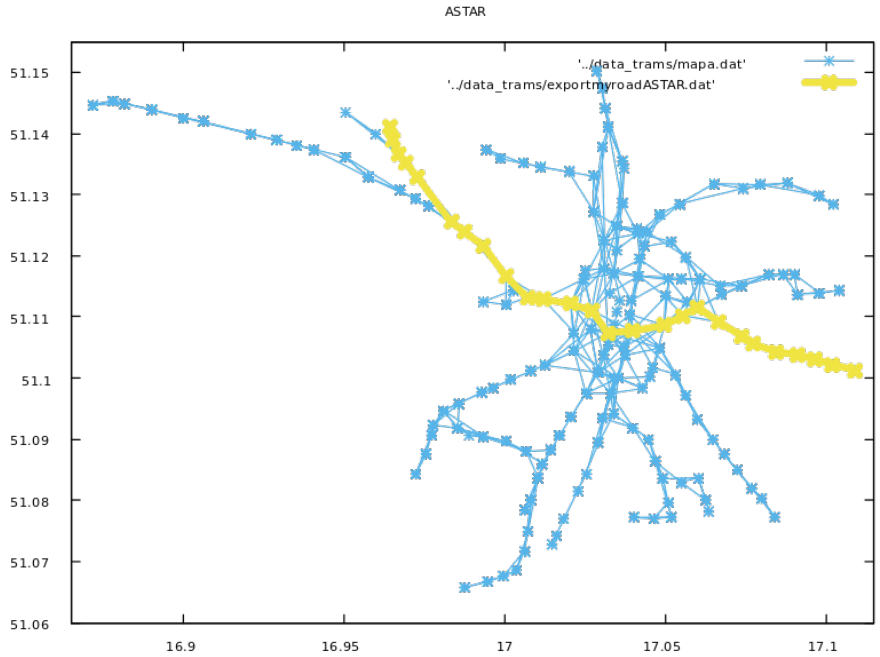
Rysunek 9: Leśnica - Gaj ASTAR

Czy zaleziona trasa jest optymalna?
Trasa taka sama jak przy BFS. Potwierdza to skuteczność algorytmu, który potrzebował wielokrotnie mniej zbabadanych połączeń niż BFS i mniej niż DFS, by wyznaczyć najoptymalniejszą trasę.

- Trasa między stacjami krańcowymi grafu

  Stacje: KOZANÓW - BISKUPIN

Id: 12610 - 24416
Czas przejazdu: 38
Liczba zbadanych połączeń: 33
Czas działania algorytmu: $5.6273e-05$ ms



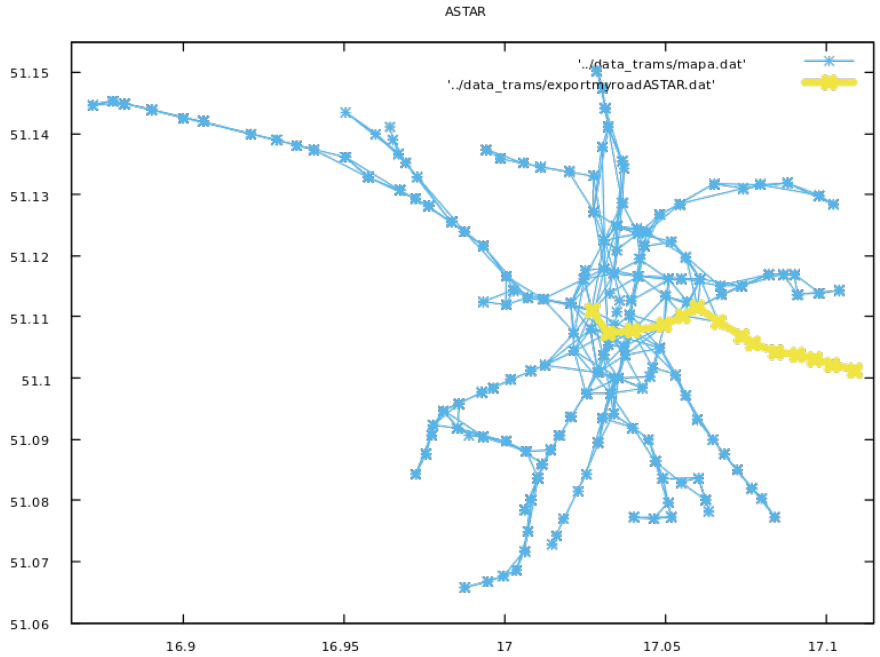
Rysunek 10: Kozanów - Biskupin ASTAR

Czy zaleziona trasa jest optymalna?
Trasa taka sama jak przy BFS. Potwierdza to skuteczność algorytmu, który potrzebował wielokrotnie mniej zbabadanych połączeń niż BFS i DFS, by wyznaczyć najoptymalniejszą trasę.

- Trasa między stacją z środka do stacji na krańcu grafu

  Stacje: RYNEK - BISKUPIN

Id: 10003 - 24416
Czas przejazdu: 20
Liczba zbadanych połączeń: 21
Czas działania algorytmu: $2.4711e-05$ ms



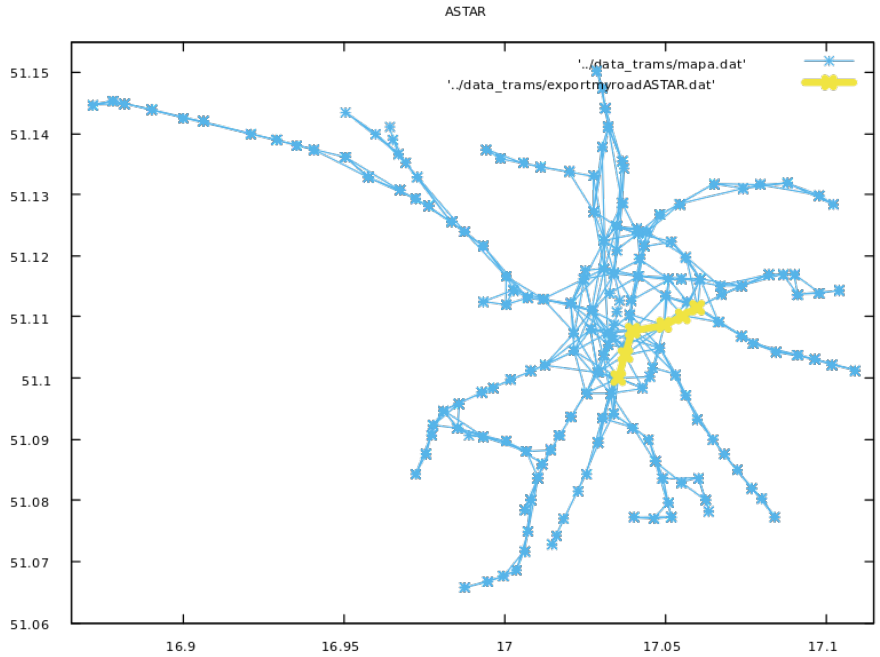
Rysunek 11: Rynek - Biskupin ASTAR

Czy zaleziona trasa jest optymalna?
Trasa taka sama jak przy BFS. Potwierdza to skuteczność algorytmu, który potrzebował wielokrotnie mniej zbabadanych połączeń niż BFS i DFS, by wyznaczyć najoptymalniejszą trasę.

- Trasa między stacjami znajdującym się w środku grafu

  Stacje: PL.GRUNWALDZKI - DW GŁÓWNY

Id: 20823 - 10226
Czas przejazdu: 9
Liczba zbadanych połączeń: 5
Czas działania algorytmu: $1.0836e-05$ ms



Rysunek 12: PL. Grunwaldzki - Dworzec Gł. ASTAR

Czy zaleziona trasa jest optymalna?
Trasa taka sama jak przy BFS. Potwierdza to skuteczność algorytmu, który potrzebował wielokrotnie mniej zbabadanych połączeń niż BFS i DFS, by wyznaczyć najoptymalniejszą trasę.

7 Wizualizacja w Gnuplot

Do wszelkiej wizualizacji został wykorzystany darmowy program Gnuplot. Struktura grafu jest zapisywana przez program do pliku *mapa.dat*. Znajdowane trasy zapisywane są do plików *exportmyroadBFS.dat*, *exportmyroadDFS.dat*, *exportmyroadASTAR.dat*. Zapisane w plikach współrzędne geograficzne stacji są rysowane przez program Gnuplot. Dynamiczna edycja plików zawierających trasy, podczas wykonywania algorytmów szukających, zapewnia animację wyszukiwania trasy i pozwala na obserwację sposobu działania każdego z algorytmów. Należy pamiętać, że przy pomiarach czasu wykonywania algorytmu należy zakomentować części kodu odpowiadające za animację.

8 GUI

Graficzny interfejs użytkownika został napisany w Pythonie w wersji 3. Użyto biblioteki Tkinter, służącej do tworzenia aplikacji okienkowych. Interfejs jest prosty: składa się z dwóch Combobox'ów, z których wybiera się stację początkową i końcową. Kolejnymi elementami są dwa przyciski - pierwszy odpowiedzialny za wyszukanie połącznia oraz drugi odpowiedzialny za poprawne zamknięcie binarki. Dane do binarki przekazywane są za pomocą przechwycenia strumieni wejścia i wyjśćia.

9 Wnioski

- Breadth First Search

Trasa	Czas przejazdu	Zbadane połączenia	Czas algorytmu [ms]
LEŚNICA - GAJ	51	173	$1.5007e-04$
KOZANÓW - BISKUPIN	38	175	$1.6949e-04$
RYNEK - BISKUPIN	20	162	$1.6641e-04$
PL.GRUNWALDZKI - DW GŁÓWNY	9	32	$4.2369e-05$

- Deep First Search

Trasa	Czas przejazdu	Zbadane połączenia	Czas algorytmu [ms]
LEŚNICA - GAJ	51	45	$3.0945e-05$
KOZANÓW - BISKUPIN	48	146	$1.1721e-04$
RYNEK - BISKUPIN	30	138	$8.4303e-05$
PL.GRUNWALDZKI - DW GŁÓWNY	24	96	$1.0685e-04$

- A Star

Trasa	Czas przejazdu	Zbadane połączenia	Czas algorytmu [ms]
LEŚNICA - GAJ	51	34	$4.2438e-05$
KOZANÓW - BISKUPIN	38	33	$5.6273e-05$
RYNEK - BISKUPIN	20	21	$2.4711e-05$
PL.GRUNWALDZKI - DW GŁÓWNY	9	5	$1.0836e-05$

Czasy działania algorytmów BFS i DFS rosną odpowiednio ze wzrostem liczby zbadanych przez nie połączeń. W większości przypadków DFS by znaleźć trasę potrzebował niewiele mniej badań niż BFS, co sprawiało, że wykonywał się szybciej. DFS ustępuje BFS'owi w przypadku szukania trasy między stacjami w środku grafu, gdzie połączenia są najgęstrze, ponieważ jest narażony na tracenie czasu na rozwijaniu złych ścieżek. BFS w tym przypadku wygrywa z DFS'em ponieważ jego szukanie często okazuje się szybsze między bliskimi sobie stacjami.

A Star mimo że poświęca czas na wyszukiwanie najlepszych "stacji kandydatów" do zbadania, nadrabia go małą sumaryczną liczbą przeszukiwań i zarówno w liczbie zbadanych połączeń jak i w czasie pracy pokonuje swą optymalnością pozostałe dwa algorytmy.

Ciekawym odstępstwem od tej reguły jest przypadek trasy Leśnica - Gaj, na której algorytm DFS przypadkiem (związane to było z tym, że akurat ta trasa została przeszukana wgłąb przed innymi) wykonał tylko 45 badań, by znaleźć końcową stację. Mimo że wykonał o 11 badań więcej od A Star'a, to jego czas pracy był zauważalnie krótszy. Wynika to z tego, że A Star musi powięcać czas na selekcję stacji, które może zbadać.

Oprócz tego jednego przypadku A Star zawsze bada wielokrotnie mniej połączeń, by wyznaczyć trasę ze stacji początkowej do stacji końcowej.

Najlepsze czasowo trasy znajduje zawsze BFS i A Star, dla których te czasy są takie same. Mogą się w niektórych przypadkach nieznacznie różnić przez to, że algorytmy kończą działanie po znalezieniu stacji końcowej, nie badając już pozostałych alternatyw.

Utworzony graf jest grafem skierowanym. Nawet jeśli między dwoma stacjami można poruszać się w obie strony, to jest to realizowane przez inne połączenia.

Struktura danych jaką jest graf ma szerokie zastosowanie nie tylko w programowaniu, czego przykładem jest chociażby sieć połączeń tramwajowych we Wrocławiu. Chcielibymy jednak skupić się tutaj na jego programistycznym zastosowaniu, które jest ogromne. Pierwszym przykładem wykorzytania grafu i algorytmów grafowych jest nasz program, który "poprowadzi" podróżnika przez Wrocław. Algorytmy grafowe mają bardzo duże zastosowanie w grach video. Pierwszym tego przykładem jest moment w grze, gdy postać NPC ma nas zaprowadzić do pewnej lokalizacji, a my mamy po prostu za nią iść. Algorytm grafowy wyszukuje wtedy optymalną trasę jaką ma iść NPC. By zobrazować drugi przykład posłużymy się tu konkretną produkcją, którą jest *Heroes Of MIght And Magic III* od *Ubisoftu*. Żeby przemieścić bohatera na mapie wskazujemy miejsce, do którego chcemy się dostać. Następnie gra pokazuje nam najlepszą trasę, którą możemy przejechać, co widać na rysunku 13. Algorytmy grafowe

Rysunek 13: Heroes 3

znajdują również zastosowanie w robotyce. Tego przykładem są roboty poruszające się w labiryncie, które muszą stworzyć jego mapę, będącą grafem, a następnie wyznaczyć najoptymalniejszą drogę do zadanego miejsca.

10 Źródła

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *Wprowadzenie do algorytmów* wyd. VI
- MIT OpenCourseWare (OCW) - wykłady profesora Patricka Winstona, dostępne na platformie *YouTube*
- Ubisoft Entertainment *Heroes of Might and Magic III*