Projektowanie algorytmów i metod sztucznej inteligencji
Projekt II

Michał Wypustek 232339
Norbert Cyran 235212

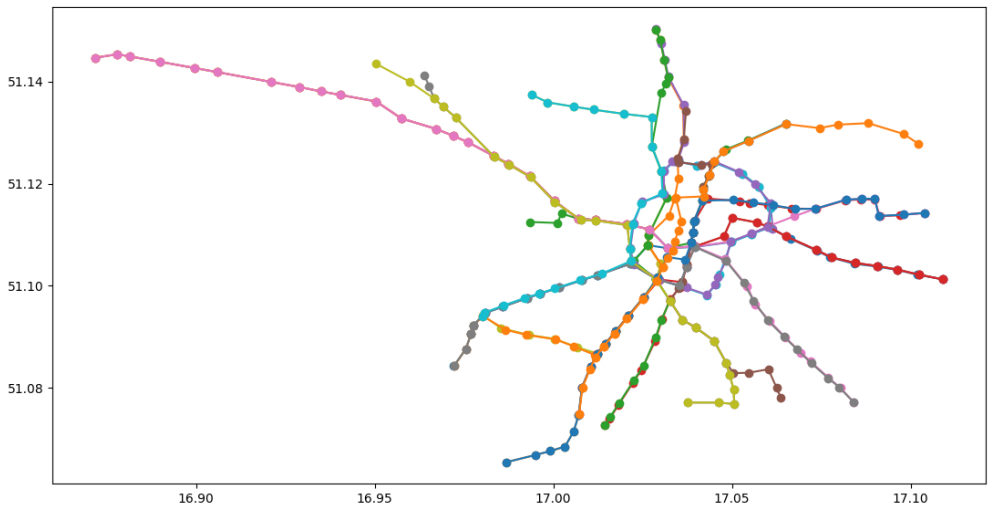
27 kwietnia 2018

Spis treści

1 Wstęp

Zadanie polegało na stworzeniu programu działającego na zasadzie popularnej aplikacji "JakDojade". Przygotowany program po podaniu przystanku początkowego oraz końcowego pokazuje optymalną trasę którą możemy pokonać za pomocą tramwaju. Aplikacja potrafi opracować trasę z przesiadkami. Jedynym warunkiem jest to że przesiadka następuje w czasie rzeczywistym (nie brano pod uwagę czasu oczekiwania na następny tramwaj). Wierzchołkami grafu są przystanki, krawędziami grafu są linie, a waga to czas. Algorytmy jakie zostały wykorzystane na potrzeby projektu:

1. Przeszukiwanie w głąb (ang. Depth-first search, w skrócie DFS).
2. Przeszukiwanie wszerz (ang. breadth-first search, BFS).
3. Algorytm heurystyczny A*.



Rysunek 1: Graf ukazujący mape przystanków tramwajowych ułożony według współrzędnych

2 Wymagania

Do działania programu wymagany jest Python 3 oraz biblioteki:

- PyQt5 - biblioteka na licencji GPL odpowiadająca za GUI programu.
- matplotlib - biblioteka służąca do generowania wykresów.

Projekt zawiera plik *requirements.txt*, za pomocą którego można zainstalować wszystkie potrzebne zależności.

3 Algorytmy

3.1 Depth-first search

Depth-first search, czyli przeszukiwanie w głąb. Algorytm startując z pewnego wierzchołka wrzuca na stos wszystkich jego sąsiadów. Następnie ściąga element ze stosu i powtarza operację. Algorytm może zostać zaimplementowany rekurencyjnie, jednak w tym przypadku nie zdecydowano się na to rozwiązanie. Dzięki zastosowaniu stosu, po zbadaniu wszystkich krawędzi wychodzących z danego wierzchołka algorytm powraca do wierzchołka, z którego dany wierzchołek został odwiedzony(ang. backtracing). Algorytm kończy swoje działanie w chwili znalezienia szukanego elementu.

3.2 Breadth-first search

Przechodzenie grafu rozpoczyna się od zadanego wierzchołka i polega na odwiedzeniu wszystkich osiągalnych z niego wierzchołków. W praktyce implementacja BFS wygląda tak samo jak w przypadku DFS, różnicą jest zastosowanie kolejki zamiast stosu. Algorytm przeszukuje graf poziom po poziomie, w wyniku czego czas potrzebny na odszukanie drogi zazwyczaj jest dłuższy niż w przypadku DFS. Przeszukiwanie wszerz daje gwarancję odszukania drogi, w dodatku zawsze będzie ona najkrótsza względem przebytych wierzchołków, co daje przewagę temu algorytmowi nad przeszukiwaniem wgłąb.

3.3 A*

Algorytm stosujący heurystykę w celu znalezienia najkrótszej ścieżki w grafie ważonym z dowolnego wierzchołka. Algorytm jest optymalny, w tym sensie, że znajduje ścieżkę, jeśli tylko taka istnieje, i przy tym jest to ścieżka najkrótsza. Algorytm, ze względu na heurystykę może być wykorzystany tylko jeśli graf jest zbudowany tak, żeby było możliwe oszacowanie realnej odległości między wierzchołkami. Mapa miasta spełnia ten warunek dzięki współrzędnym geograficznym w jakie wyposażony jest każdy przystanek (wierzchołek). Algorytm A* w odróżnieniu od innych algorytmów, dzięki heurystyce nie przeszukuje wierzchołków które oddalają go od celu. Do implementacji algorytmu zastosowano kolejkę priorytetową.

4 Implementacja

W poniższych podrozdziałach przedstawiono szczegóły implementacyjne wykonanego programu.

4.1 Graf

Graf zaimplementowano metodą list sąsiedztwa (ang. *adjacency list*). Zbudowany jest za pomocą słownika, który mapuje nazwy przystanków odpowiadającym im wierzchołkom grafu. Wierzchołek grafu jest zaimplementowany w osobnej strukturze, która zawiera informacje takie jak nazwa przystanku, listę sąsiedztwa, współrzędne geograficzne czy linie tramwajowe przejeżdżające przez dany przystanek.
Każdy z algorytmów wyszukiwania zwracał znalezioną przez siebie ścieżkę oraz liczył ilość zakolejkowań oraz zdjęć z kolejki w celu statystyk. Na podstawie ścieżki ustalano

przesiadki oraz weryfikowano optymalność.
Dane do grafu pozyskane zostały z otwartych danych miasta Wrocław, następnie sparsowane przy użyciu biblitek xml.etree oraz csv języka Python.

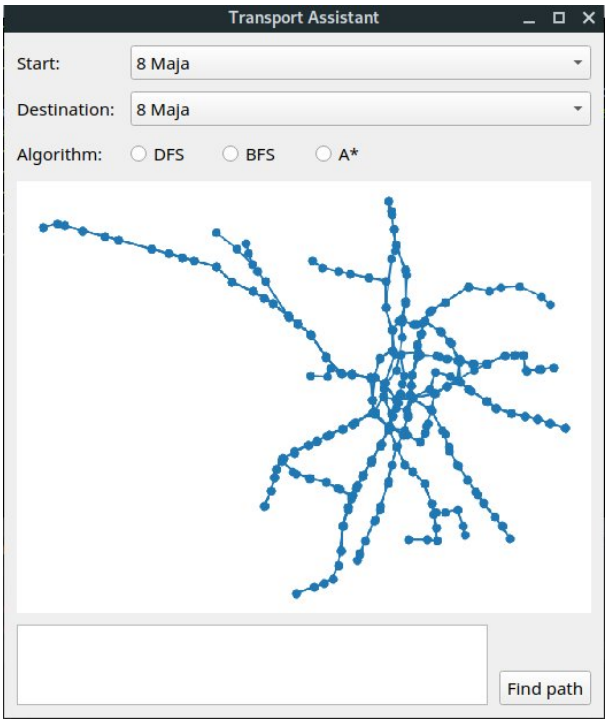
4.2 Realizacja przesiadek

Po wyznaczeniu trasy należało poinformować użytkownika jakimi tramwajami może dostać się do celu. Zaprojektowano algorytm, który wykorzystuje fakt, że każdy z wierzchołków przechowuje informacje o liniach tramwajowych, które przez niego przejeżdżają. Dla każdego kolejnego przystanku w wyznaczonej trasie obliczano intersekcję (za pomocą funkcji STL - std::set_intersection) linii z poprzednimi przystankami. Jeśli intersekcja okaże się zbiorem pustym, oznacza to przesiadkę na poprzednim przystanku. Operację powtarza się aż do końca trasy.

4.3 Wizualizacja grafu

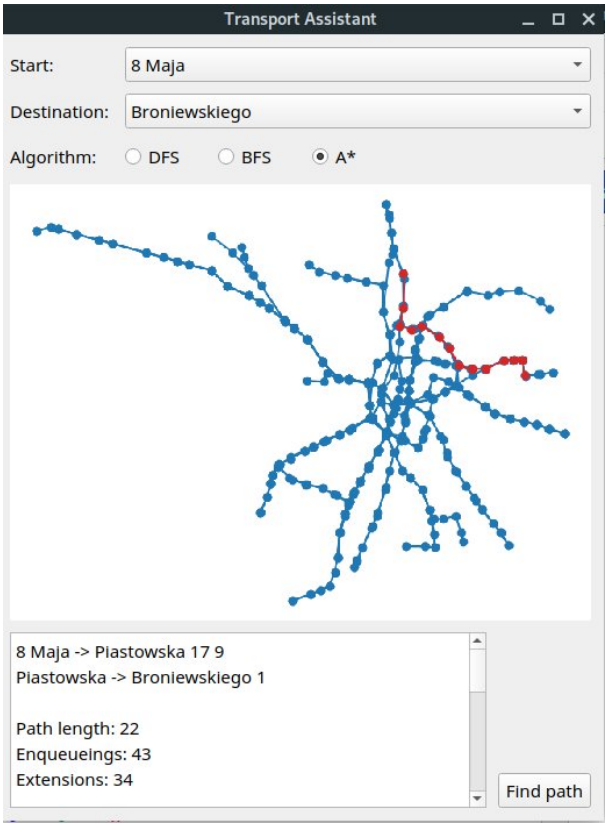
Wszelkie wizualizacje dokonano za pomocą biblioteki matplotlib języka Python.

4.4 GUI (Graphical User Interface)



Rysunek 2: GUI programu którego zadaniem jest wyszukanie linii tramwajowej przez 3 różne algorytmy przeszukiwania

GUI programu zostało wykonane za pomocą języka Python i biblioteki PyQt5. U góry okna programu są trzy opcje do wyboru. Pierwszą z nich jest przystanek początkowy, następną jest przystanek końcowy, a ostatnią jest wybór algorytmu przeszukiwania. Po odpowiednim ustawieniu tych opcji oraz nacisnięciu przycisku „Find path” na oknie wyświetli się graf z zaznaczoną wybraną trasą, wymaganymi przesiadkami, listą wszystkich przystanków, oraz długością trasy.



Rysunek 3: Przykładowe użycie programu.

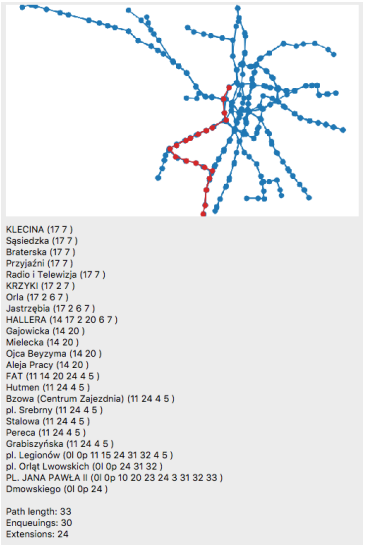
5 Analiza

W tym rozdziale zostały ukazane zaproponowane trasy opracowane przez każdy algorytm oraz porównanie czasów przeszukania grafu.

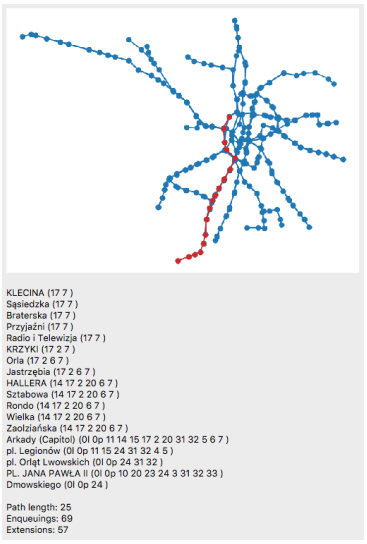
Na początku została wybrana trasa od przystanku „Klecina” do przystanku „Dmowskiego”. Na obrazach poniżej zostały zaprezentowane poszczególne trasy zaproponowane przez poszczególne algorytmy.

Algorytm przeszukiwania w szerz oraz A* wybrały najbardziej optymalną trasę. Algorytm przeszukiwania wgłąb również wybrał dobrą trasę, ale waży ona więcej więc jest mniej optymalna niż trasa BFS i A*.

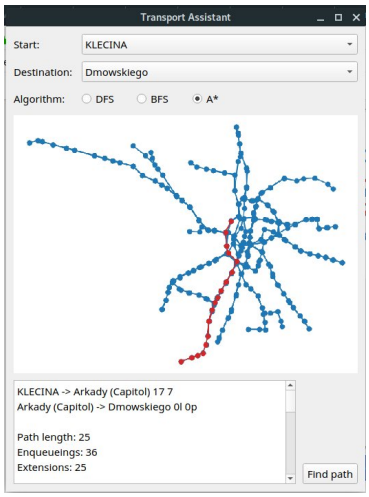
Najszybszy algorytm jeśli chodzi o wyznaczenie trasy był A*. Następnie był DFS, a na samym końcu BFS.



(a) Trasa zaproponowana przez DFS



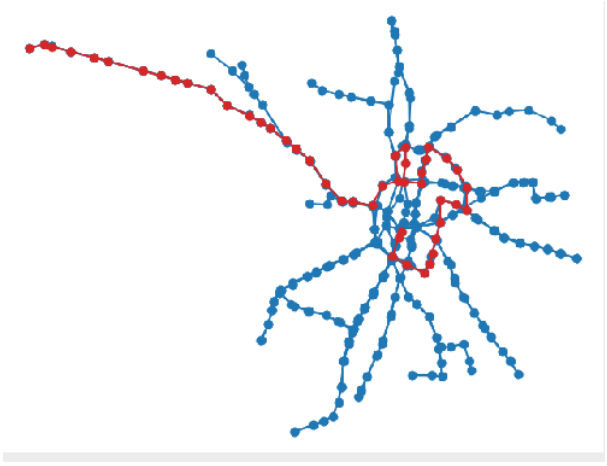
(b) Trasa zaproponowana przez BFS



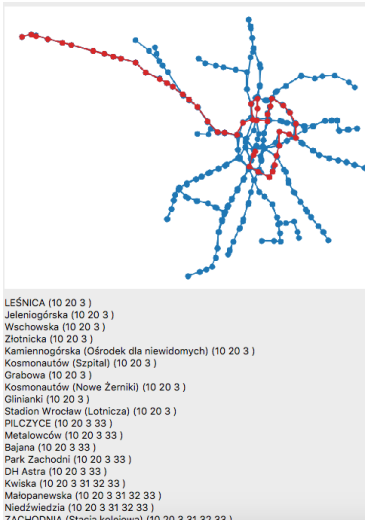
(c) Trasa zaproponowana przez A*

Rysunek 4: Trasa z przystanku „Klecina” do przystanku „Dmowskiego”

Następną trasą jaka została wybrana była droga od przystanku „Leśnica” do przystanku „Świdnicka”

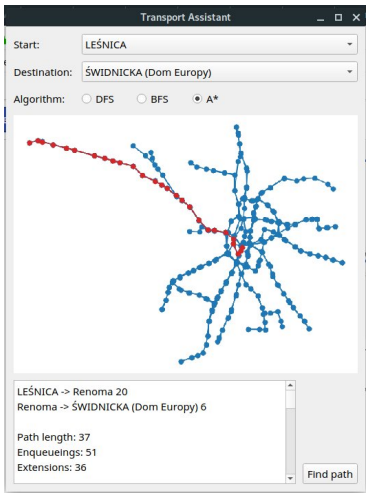


Rysunek 5: Trasa z przystanku „Leśnica”-„Świdnicka” wyliczona za pomocą DFS.



(a) Trasa zaproponowana przez DFS

(b) Trasa zaproponowana przez BFS



(c) Trasa zaproponowana przez A*

Rysunek 6: Trasa z przystanku „Leśnica” do przystanku „Świdnicka”

6 Wnioski

Dla obu przypadków w poprzedniej sekcji można zauważyć:

- Algorytm DFS szybko odnajduje swoją trasę, jednak nie jest ona ani optymalna ani nawet rozsądna.
- Algorytm BFS zawsze odnajduje rozsądną trasę, nawet optymalną, pod warunkiem, że wszystkie krawędzie grafu mają taką samą wagę.
- Algorytm A* w tym przypadku deklasuje pozostałe dwa algorytmy wyszukując zawsze optymalną trasę, rozwijając przy tym swoją ścieżkę do 5 razy rzadziej niż w przypadku BFS (dla trasy Leśnica-Stadion Olimpijski A* zanotował 37 rozszerzeń, BFS aż 156.). Niestety ze względu na heurystykę tego algorytmu nie można użyć w każdym przypadku.

Nie zmierzono czasów działania poszczególnych algorytmów, ponieważ dysponowano zbyt małym zestawem danych, żeby czasy zauważalnie się różniły.
Porównano obliczone trasy oraz wyznaczone przesiadki z trasami wyznaczonymi przez większe aplikacje tego typu (Google Maps, jakdojade), uzyskano wynik niemal identyczny, różniący się najprawdopodobniej w wyniku synchronizacji przesiadek.