Sztuczna inteligencja i inżynieria wiedzy Lista 1 - Raport

Łukasz Wasilewski

26.03.2024

1 Wstęp

Zadaniem które należało wykonać w ramach realizacji listy było zaimplementowanie konkretnych algorytmów przeszukiwania grafów, które mogły posłużyć do wyszukania optymalnych połączeń wrocławskiej komunikacji miejskiej na bazie rozkładu jazdy. Osobiście zaimplementowałem algorytm Dijkstry oraz A*, których działanie oraz sposób wykonania opiszę w dalszej części raportu.

2 Zbiór danych i jego przetwarzanie

2.1 Zbiór danych

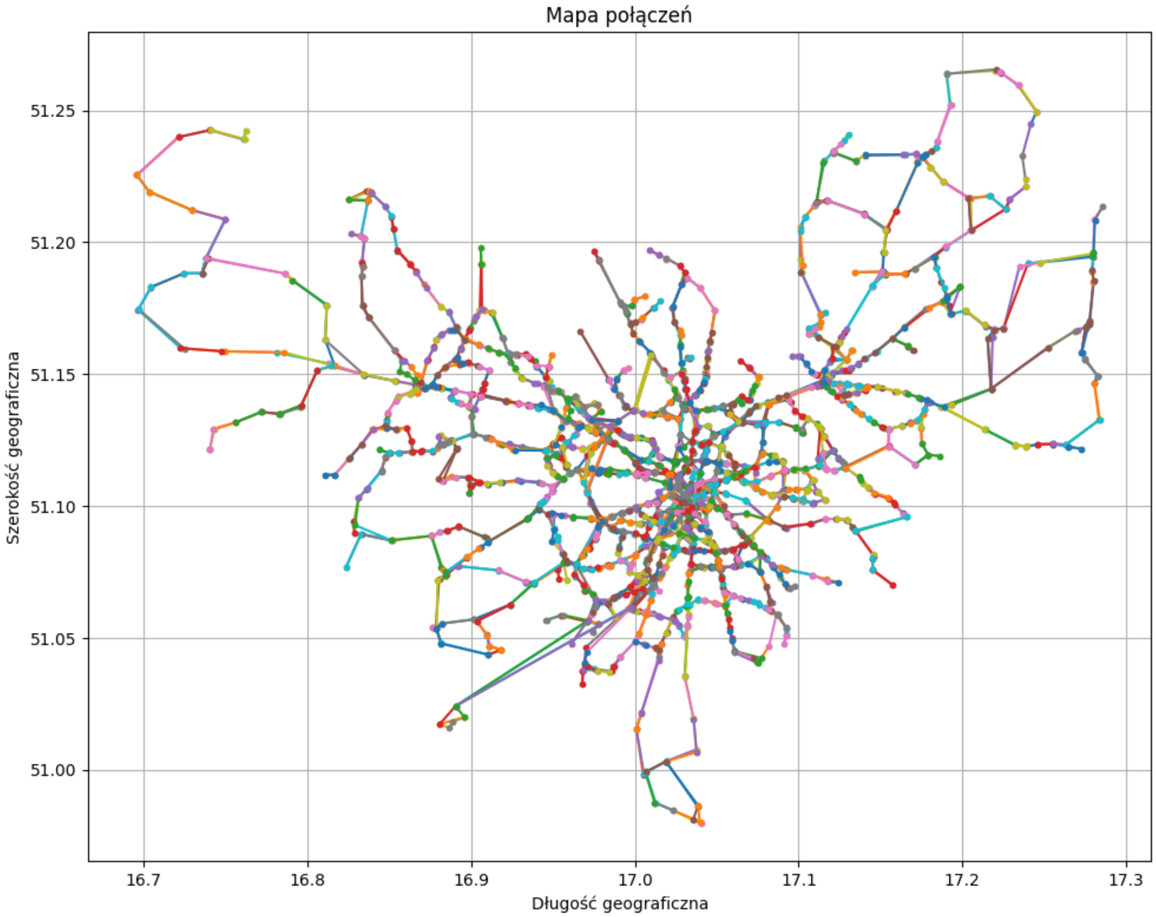
W rozwiązaniu został wykorzystany zbiór danych connection_graph.csv zawierający wstępnie przetworzone dane z rozkładu jazdy komunikacji miejskiej we Wrocławiu z dnia 1 marca 2023. Każda linia w pliku zawiera informacje o przemieszczeniu się tramwaju bądź autobusu między dwoma sąsiednimi przystankami.

2.2 Przetwarzanie danych

Zbiór danych nie posiadał w sobie błędów, które utrudniałyby jednoznaczną interpretację danych. Ważną kwestią nad jaką należało się pochylić był format godziny odjazdu i przyjazdu. Dane te nie są zapisane w standardowym formacie 24-godzinnym, a w pliku możemy natrafić np. na godzinę 25:30:00. Jest to zastosowane w celu jednoznacznej interpretacji czasu przejazdu autobusów nocnych, które mogą przemieszczać się między przystankami równo ze zmianą dnia i wyzerowaniem zegara. Ja jednak w swoim rozwiązaniu szybko zrezygnowałem z tej interpretacji czasu. Wykorzystałem ją do policzenia czasu przejazdu między przystankami, po czym wszystkie czasy sprowadziłem do formatu 24-godzinnego. Co do samego wczytania danych z pliku, na początku przerzuciłem je do listy słowników, tak że każdy słownik odpowiadał jednemu wierszowi z pliku. Następnym krokiem było wyodrębnienie z listy wszystkich niepowtarzalnych nazw przystanków reprezentujących wierzchołki grafu oraz przypisanie im wychodzących z nich krawędzi. Wykorzystałem do tego reprezentację słownika w słowniku, w którym kluczami są nazwy przystanków, a wartościami krawędzie oraz parametry wierzchołków zmieniane w trakcie wykonywania algorytmu.

2.3 Graficzna reprezentacja danych

Jako, że dane przystanków zawierają informacje o współrzędnych geograficzny, to jesteśmy w stanie sprawnie narysować mapę komunikacji miejskiej Wrocławia.



Rysunek 1: Wykres przedstawia wszystkie połączenia między węzłami

3 Zadania

3.1 Algorytm Dijkstry w oparciu o kryterium czasu

Algorytm Dijkstry polega na wyznaczeniu najkrótszej ścieżki od węzła startowego do każdego innego wierzchołka. Jest to metoda niezawodna jeśli chodzi o jak najmniejszą wartość funkcji kosztu, jednak co za tym idzie wykonuje się dłużej niż metody heurystyczne. Często może dojść do sytuacji, w której w praktyce najkrósza ścieżka do wierzchołka końcowego została już znaleziona, jednak jedynym sposobem by to zweryfikować jest sprawdzić resztę ścieżek. Z ciekawych rzeczy jakie udało mi się zauważyć to fakt, że w danych istnieją wierzchołki, które zawsze występują jako przystanek początkowy albo takie, które zawsze występują jako przystanek końcowy. Dlatego zawsze sprawdzam czy węzeł początkowy, który podał użytkownik znajduje się w zbiorze węzłów początkowych, analogicznie z węzłem końcowym. Dzięki temu mam pewność, że zawsze jakaś ścieżka zostanie znaleziona. Kolejnym niuansem jest obliczanie bezwzględengo czasu oczekiwania na przystanku. Nawet połączenie, w którym tramwaj odjechał godzinę temu może okazać się ważne. Jeżeli jest to jedyna ścieżka do interesującego nas węzła, to jedynym sposobem jest poczekać 23 godziny na ponowny przyjazd. Oczywiście nie ma to sensu w realnym świecie ale ma sens jeżeli chcemy mieć pewność w odnalezieniu wszystkich połączeń.

Przykłady użycia algorytmu Dijkstry o następującym schemacie danych wejściowych: dijkstra(przystanek początkowy, przystanek końcowy, godzina przyjścia na przystanek)

dijkstra(”Sucha”, ”KOZANÓW”, ”23:30:00”)

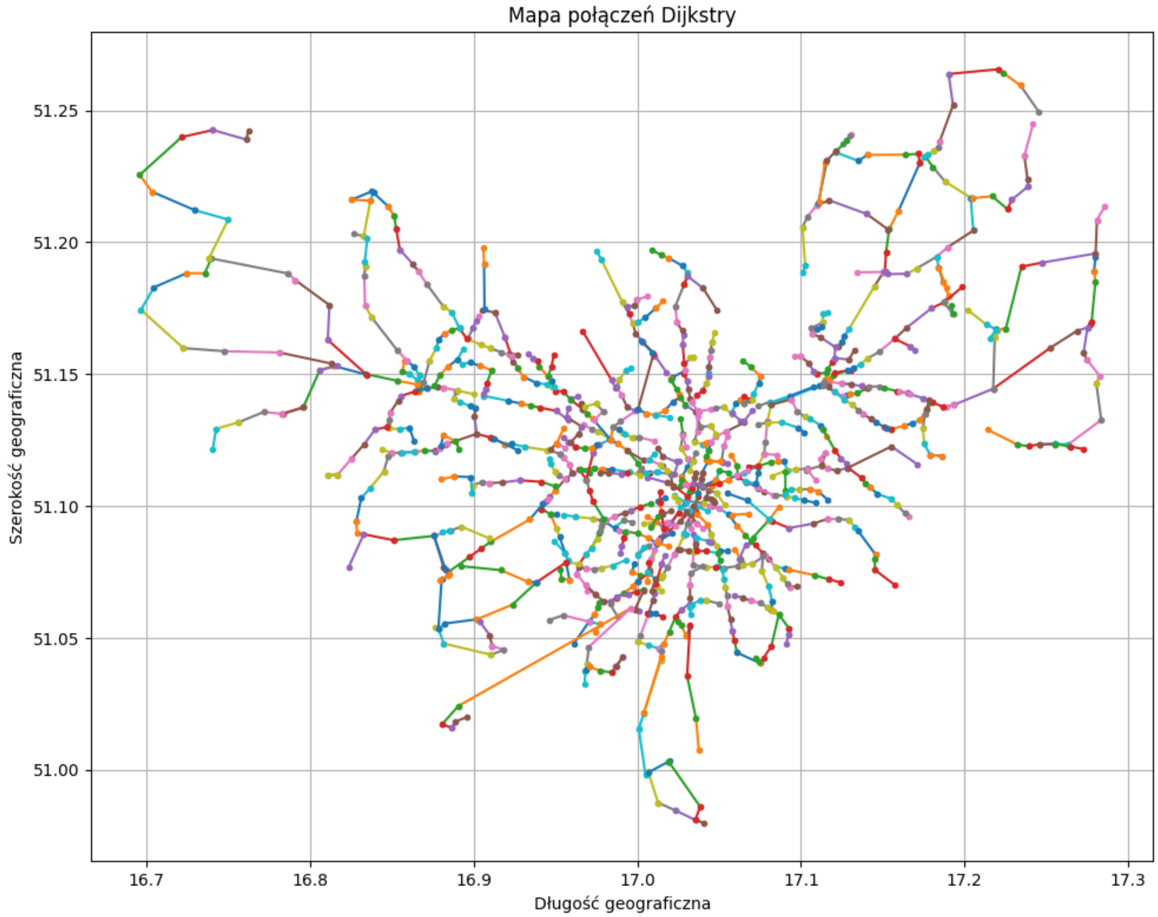
```
Start podrozy: 23:30:00
Czas podrozy: 355.0 minut.
Linia 612, z Sucha [4:53:00] do DWORZEC AUTOBUSOWY [4:54:00]
Linia 5, z DWORZEC AUTOBUSOWY [4:55:00] do Arkady (Capitol) [4:59:00]
Linia 240, z Arkady (Capitol) [5:01:00] do pl. Orląt Lwowskich [5:04:00]
Linia 19, z pl. Orląt Lwowskich [5:04:00] do PL. JANA PAWŁA II [5:06:00]
Linia 21, z PL. JANA PAWŁA II [5:07:00] do Kolista [5:20:00]
Linia 136, z Kolista [5:23:00] do KOZANÓW [5:25:00]

Czas obliczen: 0:00:01.589418
```

dijkstra(”Marchewkowa”, ”Tramwajowa”, ”07:45:00”)

```
Start podrozy: 07:45:00
Czas podrozy: 35.0 minut.
Linia D, z Marchewkowa [7:46:00] do Zimowa [7:52:00]
Linia 933, z Zimowa [7:52:00] do KRZYKI [7:53:00]
Linia 602, z KRZYKI [7:54:00] do GALERIA DOMINIKAŃSKA [8:08:00]
Linia D, z GALERIA DOMINIKAŃSKA [8:09:00] do PL. GRUNWALDZKI [8:13:00]
Linia 19, z PL. GRUNWALDZKI [8:13:00] do Hala Stulecia [8:17:00]
Linia 145, z Hala Stulecia [8:18:00] do Tramwajowa [8:20:00]

Czas obliczen: 0:00:01.837211
```



Rysunek 2: Wykres przedstawia wszystkie najkrótsze połączenia między węzłami, uzyskane przy pomocy algorytmu Dijkstry

Zaimplementowany przeze mnie algorytm Dijkstry średnio wykonuje się między 1,5 a 2 sekundy. Działa on na kryterium czasowym, dlatego tak jak jest to widoczne w drugim użyciu, mogą występować niepotrzebne przesiadki.

3.2 Algorytm A* w oparciu o kryterium czasu

Algorytm A* jest rozszerzeniem algorytmu Dijkstry o dodatkowy element szacowania ścieżki do celu. W tym wariancie w momencie gdy odnajdziemy pewną ścieżkę do węzła końcowego oraz wśród aktualnie badanych węzłów to węzeł końcowy będzie miał najmniejszą wartość funkcji kosztu, działanie algorytmu się kończy. Nie zawsze znajdujemy idealne rozwiązanie ale w stosunkowo krótkim czasie znajdujemy rozwiązanie wystarczająca dobre. Działamy na podobnej zasadzie co w Dijkstrze, z tym wyjątkiem, że całkowita funkcja kosztu jest sumą funkcji kosztu przejścia z jednego wierzchołka do drugiego (ilości czasu) oraz estymacji kosztu z wierzchołka do końcowego celu. Estymacji możemy dokonać np. wyliczając odległość Manhattanową bądź Euklidesową. Rzeczą która rzuciła mi się w oczy było przyjęcie odpowiedniej skali jednostek. Gdybyśmy przyjeli jako jednostkę czasu sekundy, koszt czasu znacząco by przeważał nad wyliczniem odległości.

Przykłady użycia algorytmu A* z kryterium czasu o następującym schemacie danych wejściowych: astar(przystanek początkowy, przystanek końcowy, kryterium, godzina przyjścia na przystanek)

astar(”Sucha”, ”KOZANÓW”, ”t”, ”23:30:00”) (odległość Manhattan)

```
Start podrozy: 23:30:00
Czas podrozy: 1 day, 2:01:00
Linia 612, z Sucha [4:53:00] do 4:54:00 [DWORZEC AUTOBUSOWY]
Linia 241, z DWORZEC AUTOBUSOWY [4:55:00] do EPI [4:57:00]
Linia 248, z EPI [0:05:00] do Rondo [0:08:00]
Linia 134, z Rondo [0:15:00] do Na Ostatnim Groszu [0:31:00]
Linia 243, z Na Ostatnim Groszu [0:33:00] do PL. JANA PAWŁA II [0:41:00]
Linia 245, z PL. JANA PAWŁA II [1:13:00] do KOZANÓW [1:31:00]

Czas obliczen: 0:00:02.062112
```

astar(”Marchewkowa”, ”Tramwajowa”, ”t”, ”07:45:00”) (odległość Manhattan)

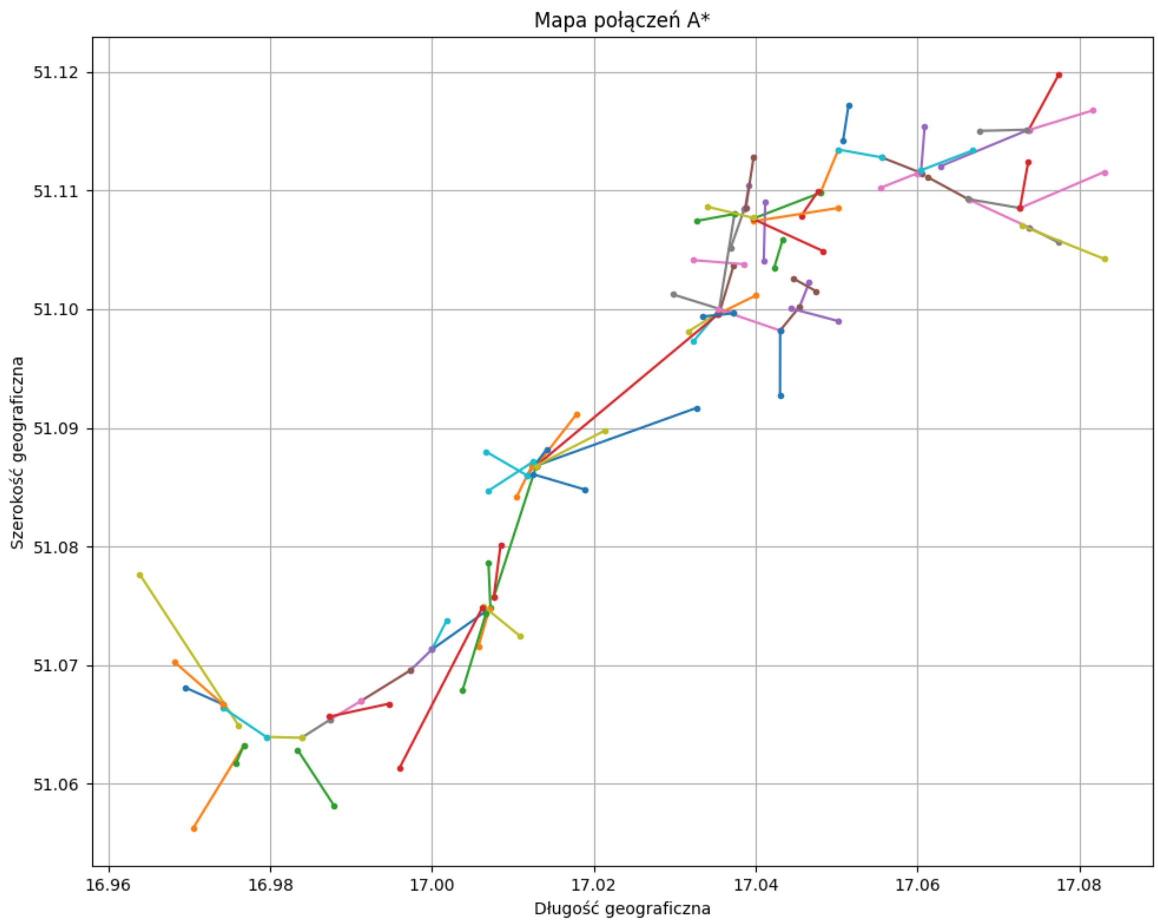
```
Start podrozy: 07:45:00
Czas podrozy: 0:43:00
Linia D, z Marchewkowa [7:46:00] do Zimowa [7:52:00]
Linia 933, z Zimowa [7:52:00] do KRZYKI [7:53:00]
Linia 602, z KRZYKI [7:54:00] do GALERIA DOMINIKAŃSKA [8:08:00]
Linia 2, z GALERIA DOMINIKAŃSKA [8:08:00] do PL. GRUNWALDZKI [8:16:00]
Linia 1, z PL. GRUNWALDZKI [8:16:00] do Hala Stulecia [8:20:00]
Linia 146, z Hala Stulecia [8:26:00] do Tramwajowa [8:28:00]

Czas obliczen: 0:00:00.357004
```

astar(”Marchewkowa”, ”Tramwajowa”, ”t”, ”07:45:00”) (odległość Euklidesowa)

```
Start podrozy: 07:45:00
Czas podrozy: 0:37:00
Linia D, z Marchewkowa [7:46:00] do Zimowa [7:52:00]
Linia 933, z Zimowa [7:52:00] do KRZYKI [7:53:00]
Linia 602, z KRZYKI [7:54:00] do GALERIA DOMINIKAŃSKA [8:08:00]
Linia 2, z GALERIA DOMINIKAŃSKA [8:08:00] do PL. GRUNWALDZKI [8:16:00]
Linia 1, z PL. GRUNWALDZKI [8:16:00] do Tramwajowa [8:22:00]

Czas obliczen: 0:00:00.154522
```



Rysunek 3: Wykres przedstawiający wyszukane połączenia algorytmem A* z parametrami astar(”Marchewkowa”, ”Tramwajowa”, ”t”, ”07:45:00”) (odległość Euklidesowa)

W pierwszym przykładzie widzimy ekstremalny przypadek, w którym algorytm źle sobie poradził. Algorytm zakończył swoje działanie zanim zoptymalizował jedną ze ścieżek na której pasażer rzekomo musiałby czekać aż 20 godzin. W drugim przykładzie porównując z Dijkstrą możemy zobaczyć, że algorytm zadziałał znacznie szybciej, jednak znalazł trasę, której koszt jest o 8 minut większy. W trzecim przypadku zamiast odległości Manhattan użyłem odległości Euklidesowej co nie dość, że pozytywnie wpłyneło na szybkość działania algorytmu to jeszcze udało się znaleźć trasę z mniejszym kosztem o 6 minut niż przy Manhattanie. Wykres dobrze obrazuje jak niewiele ścieżek musieliśmy odkryć aby otrzymać dobrej jakości rozwiązanie oraz, że fukcja estymacji trafnie prowadziła nas w kierunku celu.

3.3 Algorytm A* w oparciu o kryterium przsiadek

Zasady w tym algorytmie są takie jak w poprzednim, z tą różnicą, że zamiast zliczania czasu liczymy odbyte w drodze do węzłów przesiadki.

Przykłady użycia algorytmu A* z kryterium czasu o następującym schemacie danych wejściowych: astar(przystanek początkowy, przystanek końcowy, kryterium, godzina przyjścia na przystanek)

astar("Sucha", "KOZANÓW", "p", "23:30:00") (odległość Euklidesowa)

```
Start podrozy: 23:30:00
Liczba przesiadek: 3
Linia 612, z Sucha [10:03:00] do DWORZEC AUTOBUSOWY [10:04:00]
Linia 122, z DWORZEC AUTOBUSOWY [6:14:00] do Kwiska [6:33:00]
Linia 12, z Kwiska [5:04:00] do Kolista [5:08:00]
Linia 136, z Kolista [6:54:00] do KOZANÓW [6:56:00]

Czas obliczen: 0:00:01.397543
```

astar("Budziszyńska", "KOZANÓW", "p", "15:30:00") (odległość Euklidesowa)

```
Start podrozy: 15:30:00
Liczba przesiadek: 3
Linia 13, z Budziszyńska [6:14:00] do Dolmed [6:26:00]
Linia 20, z Dolmed [15:04:00] do Metalowców [15:16:00]
Linia 123, z Metalowców [9:55:00] do Górnicza [9:58:00]
Linia 126, z Górnicza [19:18:00] do KOZANÓW [19:20:00]

Czas obliczen: 0:00:03.097935
```

Widzimy, że w aktualnym stanie algorytm słabo działa. Biorąc pod uwagę tylko kryterium przesiadek otrzymujemy w wyniku przesiadki na które musimy czekać po kilkanaście godzin. Dzieje się tak, ponieważ algorytm bierze pierwsze lepsze połączenie, w którym liczba przesiadek nie jest większa niż w pozostałych połączeniach. Jeżeli przesiadka jest nieunikniona to algortm bierze pierwsze połączenie z listy niezależnie od czasu wyruszenia z przystanku. Ponadto w drugim wywołaniu widzimy, że czas obliczeń przekroczył nawet przeciętny czas przeszykiwania grafu algorytmem Dijkstry.

3.4 Modyfikacja algorytmu A*, która pozwoli na zmniejszenie wartości funckji kosztu uzyskanego rozwiązania lub czasu obliczeń

3.4.1 Wprowadzenie maksymalnego czas oczekiwania na przystanku

Pierwszą rzeczą, która może pozytywnie wpłynąć na fukcję kosztu jest ustawienie dodatkowego warunku ile pasażer może czakać na przystanku. Jeżeli wprowadzimy maksymalny czas np. 6 godzin, nie dojdzie już do sytuacji, że pasażer musi czekać 20 godzin tramwaj lecz takie połączenia będą automatycznie odrzucane.

Wynik przeszukiwania z pierwszego przykładu punktu 3.2 po dodaniu dodatkowego kryterium:

```
Start podrozy: 23:30:00
Czas podrozy: 5:55:00
Linia 612, z Sucha [4:53:00] do 4:54:00 [DWORZEC AUTOBUSOWY]
Linia 5, z DWORZEC AUTOBUSOWY [4:55:00] do Arkady (Capitol) [4:59:00]
Linia 240, z Arkady (Capitol) [5:01:00] do pl. Orląt Lwowskich [5:04:00]
Linia 19, z pl. Orląt Lwowskich [5:04:00] do PL. JANA PAWŁA II [5:06:00]
Linia 21, z PL. JANA PAWŁA II [5:07:00] do Kolista [5:20:00]
Linia 136, z Kolista [5:23:00] do KOZANÓW [5:25:00]

Czas obliczen: 0:00:02.448321
```

Tym razem została znaleziona przyzwoita trasa.

3.4.2 Zmiana struktury danych oraz przeliczania funckji heurystycznej

Drugą rzeczą, która powinna wpłynąć na czas obliczeń jest operowanie zamiast na liście otwartej i zamkniętej - na zbiorze otwartym i zamkniętym. Operacje dodawania, usuwania i wyszukiwania na zbiorze powinny zachodzić w czasie stałym, a na liście w czasie O(n). Ponadto zmieniłem lekko sposób przeliczania funkcji kosztu. Wcześniej dla takiego kryterium przesiadkowego była to liczba przesiadek dodać odległość Euklidesowa albo Manhattan. Rożnica w lokalizacji przystanków jest stosunkowo niewielka a pierwsza liczba znacząca w wyniku odległości znajduje się kilka miejsc po przecinku. Z tego powodu liczba przesiadek przeważała przy po prównywaniu kosztów. Aby zwiększyć znaczenie funkcji heurystycznej postanowiłem podzielić liczbę przesiadek przez 20 (oszcowałem, że ta liczba daje najlepszy balans)

Wynik przeszukania z drugiego przykładu punktu 3.3 po użyciu zbiorów zamiast list oraz zmniejszeniu znaczenia liczby przesiadek:

```
Start podrozy: 15:30:00
Liczba przesiadek: 3
Linia 13, z Budziszyńska [6:14:00] do Dolmed [6:26:00]
Linia 20, z Dolmed [15:04:00] do Metalowców [15:16:00]
Linia 123, z Metalowców [9:55:00] do Górnicza [9:58:00]
Linia 126, z Górnicza [19:18:00] do KOZANÓW [19:20:00]

Czas obliczen: 0:00:00.636127
```

Udało się znaleźć tą samą trasę w znacznie szybszym czasie.

3.4.3 Wspomaganie kryterium czasowego przesiadkowym oraz odwrotnie

Została jeszcze kwestia niepotrzebnych przesiadek przy przeszukiwaniu grafu kryterium czasowym oraz długiego czasu oczekiwania na przystanku przy przeszukiwaniu kryterium przesiadkowym. A co gdyby poza sprawdzaniem czasu sprawdzać czy zachodzi przesiadka oraz poza liczeniem przesiadek sprawdzać, która daje najlepszy czas? Tak właśnie zrobiłem. W sytuacjach, gdy czas na kilku połączeniach jest identyczny, sprawdzam czy któreś połączenie kontynuuje jazdę tą samą linią i to właśnie wybieram. Przy kryterium przesiadkowym w sytuacjach kiedy liczba przesiadek dla kilku połączeń jest identyczna, wybieram połączenie które daje najlepszy czas. W ten sposób unikam zbyt długiego czasu czekania na przystanku oraz przesiadania się do tej samej lini ale jadącej jakiś czas później

Wynik przeszukiwania grafu z trzeciego przykładu podpunktu 3.2:

```
Start podrozy: 07:45:00
Czas podrozy: 0:37:00
Linia D, z Marchewkowa [7:46:00] do Zimowa [7:52:00]
Linia 933, z Zimowa [7:52:00] do KRZYKI [7:53:00]
Linia 602, z KRZYKI [7:54:00] do GALERIA DOMINIKAŃSKA [8:08:00]
Linia 2, z GALERIA DOMINIKAŃSKA [8:08:00] do Tramwajowa [8:22:00]

Czas obliczen: 0:00:00.136670
```

Udało się znaleźć trasę o tym samym koszcie ale z jedną niepotrzebną przesiadką mniej

Wynik przeszukiwania grafu z pierwszego przykładu podpunktu 3.3:

```
Start podrozy: 23:30:00
Liczba przesiadek: 3
Linia 612, z Sucha [4:53:00] do 4:54:00 [DWORZEC AUTOBUSOWY]
Linia 241, z DWORZEC AUTOBUSOWY [4:55:00] do Kwiska [5:18:00]
Linia 12, z Kwiska [5:20:00] do Kolista [5:24:00]
Linia 136, z Kolista [5:24:00] do KOZANÓW [5:26:00]

Czas obliczen: 0:00:01.305156
```

Liczba przesiadek została ta sama ale usunięto niepotrzebny czas oczekiwania na przystanku.

4 Podsumowanie

W swoich rozwiązaniach wykorzystałem podstawowe biblioteki w Pythonie tj. matplotlib do rysowania wykresów czy csv do wczytania pliku csv. Nie miałem też większych problemów implementacyjnych, schemty algorytmów z instrukcji prawidłowo wskazywały kolejne kroki. Natomiast uważam, że jest potencjał na optymalizcję moich rozwiązań. W swojej implementacji z powodu wygody użyłem wielu struktur słownikowych zagnieżdżonych jedne w drugich. Wielokrotne odwoływanie się do głęboko schowanych elementów zapewne miało negatywny wpływ na szybkość obliczeń. Gdybym miał coś jeszcze dopracować byłoby to właśnie to.

Literatura

[1] ”Encyklopedia algorytmów”, http://algorytmy.ency.pl/artykul/algorytm$_{d}ijkstry$