Wykorzystanie baz danych NoSQL - MongoDB, Redis, Neo4j

Adrian Fabisiewicz (328935), Julia Gomulska (328936)

29 stycznia 2025

1 Projekt 1.

1.1 Cel projektu

Głównym celem projektu było stworzenie aplikacji do analizy danych IMGW z uwzględnieniem danych astronomicznych i położenia stacji pomiarowych w województwach oraz/lub powiatach. Aplikacja miała wykorzystywać bazy NoSQL: Redis oraz MongoDB do przechowywania odpowiednich danych przestrzennych.

1.2 Wstępne przygotowanie danych źródłowych

1.2.1 Dane meteorologiczne

Dane meteorologiczne zostały pobrane z archiwów IMGW. Następnie dane wczytano do struktur modułu *pandas*, który odczytał dane z plików csv. Dane z każdej stacji następnie złączono do jednej dużej ramki danych na podstawie kodu stacji oraz czasu.

	kodSH	B00202A	B00300S	B00305A	B00604S	B00606S	B00608S	B00702A	B00703A	B00714A	B00802A	data	time
0	249190090	94.0	3.29	0.17	NaN	0.0	0.0	0.0	0.9	NaN	92.06	2024-10-01	00:00:00
1	249190090	83.0	3.16	0.11	NaN	NaN	0.0	0.0	0.2	NaN	91.97	2024-10-01	00:10:00
2	249190090	79.0	3.05	0.03	NaN	NaN	0.0	0.0	0.7	NaN	91.98	2024-10-01	00:20:00
3	249190090	70.0	3.01	-0.02	NaN	NaN	0.0	0.2	1.4	NaN	92.45	2024-10-01	00:30:00
4	249190090	70.0	3.07	0.00	NaN	NaN	0.0	0.1	1.5	NaN	92.50	2024-10-01	00:40:00
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1166100	254150060	284.0	11.50	11.00	NaN	NaN	0.0	7.0	9.3	NaN	80.90	2024-10-30	18:10:00
1166101	254150060	277.0	11.40	10.60	NaN	NaN	0.0	6.7	9.2	NaN	80.90	2024-10-30	18:20:00
1166102	254150060	278.0	11.40	10.70	NaN	NaN	0.0	6.8	9.4	NaN	81.70	2024-10-30	18:30:00
1166103	254150060	278.0	11.40	10.90	NaN	NaN	0.0	6.2	8.7	NaN	82.00	2024-10-30	18:40:00
1166104	254150060	283.0	11.50	10.90	NaN	NaN	0.0	6.4	8.2	NaN	80.50	2024-10-30	18:50:00

Rysunek 1: Struktura danych meteo w pandas dataframe

1.2.2 Dane przestrzenne

Dane przestrzenne obejmowały dane na temat poszczególnych stacji pomiarowych, w tym ich współrzędne, w pliku geojson. Powiaty oraz województwa były zawarte w plikach SHP dostarczonych wraz z zadaniem. Wczytano je wykorzystując bibliotekę *geopandas*.
Powiaty połączono z województwami na podstawie analizy atrybutu TERYT. Do połączenia stacji pomiarowych z powiatami wykorzystano spatial join dostępny w module *geopandas*.

	ifcid	name	geometry	powiat	wojewodztwo
0	149180010	Krzyżanowice	POINT (448926.076 236501.65)	raciborski	śląskie
1	149180020	Chałupki	POINT (451760.53 228509.024)	raciborski	śląskie
2	149180030	Łaziska	POINT (460035.038 228718.997)	wodzisławski	śląskie
3	149180040	Gołkowice	POINT (463863.107 228846.279)	wodzisławski	śląskie
4	149180050	Zebrzydowice	POINT (472228.166 223702.104)	cieszyński	śląskie
...	...	...	...	...	...
2647	453220010	Jezioro Rajgrodzkie	POINT (741832.789 659022.551)	grajewski	podlaskie
2648	454170010	Jezioro Jasień	POINT (409980.914 716533.986)	bytowski	pomorskie
2649	454170020	Jezioro Raduńskie Górne	POINT (432806.035 707830.127)	kartuski	pomorskie
2650	454170030	Jezioro Łebsko	POINT (402818.172 765146.628)	słupski	pomorskie
2651	454210010	Jezioro Dejguny	POINT (671048.427 688899.99)	giżycki	warmińsko-mazurskie

Rysunek 2: Struktura danych na temat stacji w geopandas dataframe

1.3 Wczytanie danych do bazy MongoDB

1.3.1 Konfiguracja środowiska

Zdecydowano się na uruchomienie lokalnej bazy danych Mongo na Linuksie. Do połączenia do bazy poprzez Pythona wykorzystano bibliotekę *pymongo*.

1.3.2 Model danych

Do bazy MongoDB zdecydowaliśmy się wczytać dane na temat stacji. Przykładowa struktura utworzonego dokumentu wyglądała jak poniżej:

```
{
"_id": 149180210,
"name": "Zabrzeg",
"location": {
    "type": "Point",
    "coordinates": [ 18.935278, 49.925278 ]
    },
"powiat": "pszczyński",
"wojewodztwo": "śląskie",
"sun_times": [
        {
        "date": "2024-10-01",
        "dawn": 1727755934570,
        "dusk": 1727801661919
        }
    ]
}
```

1.3.3 Przeniesienie danych z geopandas do MongoDB

Aby przenieść dane z ramki danych geopandas do MongoDB, przeiterowano przez każdy z rzędów ramki. W każdej iteracji utworzono odpowiednią strukturę dokumentu, a później dodano dokument do tablicy zawierającej wszystkie dokumenty.
Po przygotowaniu tablicy danych, dodano je do bazy danych wykorzystując metodę *insert_many()*. Utworzono również indeks geoprzestrzenny *2dsphere* na kluczu *location*.

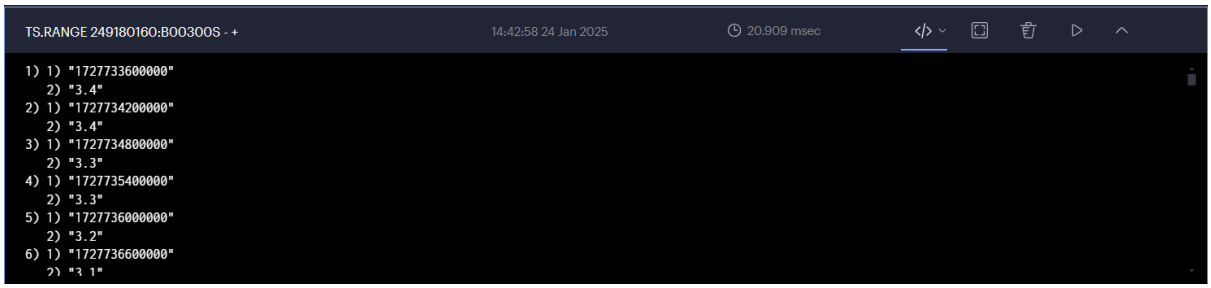
1.4 Wczytanie danych do bazy Redis

1.4.1 Konfiguracja środowiska

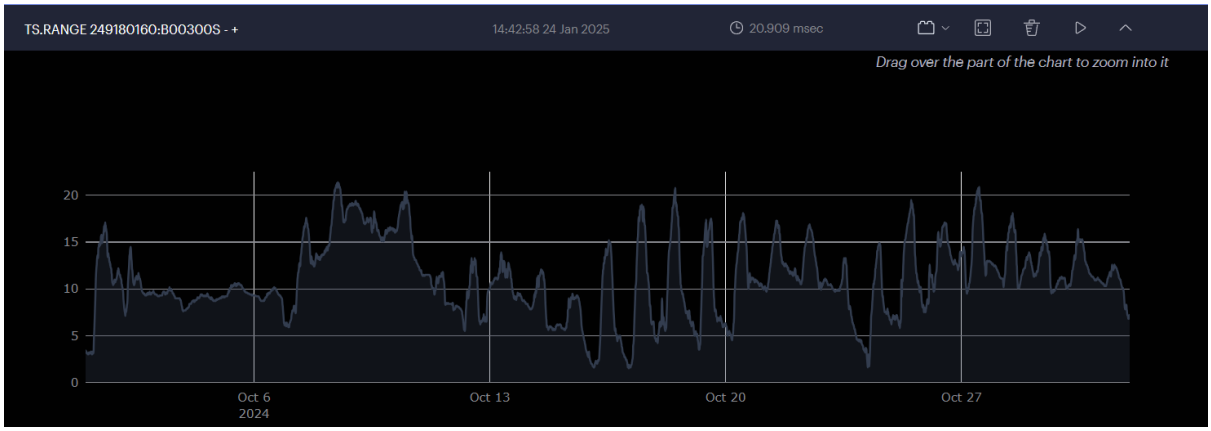
Bazę Redis również uruchomiono lokalnie. Połączono się z nią poprzez Pythona, wykorzystując bibliotekę Redis.

1.4.2 Model danych

Do bazy Redis wczytano dane meteorologiczne. Wykorzystano do tego moduł *time series*. Jako klucz ustawiono string zawierający identyfikator stacji oraz kod, informujący o rodzaju pomiaru. Przykładowo: 350150500:B00604S. Wartością dla takiego klucza była seria danych zawierająca informacje o wartości pomiaru dla określonego timestampu.



Rysunek 3: Widok przykładowej serii danych w Redis Insight - forma tekstowa



Rysunek 4: Widok przykładowej serii danych w Redis Insight - wykres

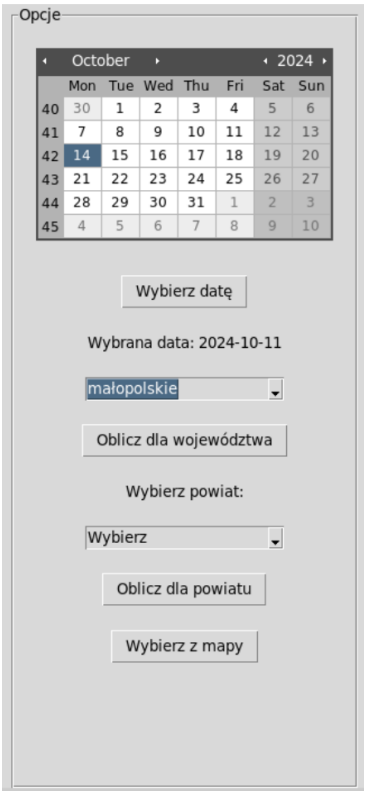
1.4.3 Przeniesienie danych z pandas do bazy Redis

Dane zostały przeniesione, iterując przez każdy rząd z ramki danych meteo. W każdym rzędzie następuje iteracja przez każdą kolumnę z danymi pomiarowymi. Następnie sprawdzamy, czy wartość dla danej kolumny jest liczbą. Jeżeli wartość to liczba, następuje próba dodania klucza dla aktualnego pomiaru dla aktualnej stacji. W przypadku, gdy klucz istnieje, następuje przejście do następnego kroku. Dla utworzonego lub istniejącego wcześniej klucza dodawana jest nowa odczytana wartość do serii. Do wykonywania zapytań wykorzystano metodę *execute_ command()* dla obiektu bazy Redis.

1.5 Aplikacja

1.5.1 Interfejs

Interfejs aplikacji składa się z dwóch ramek. W ramce opcji na górze znajduje się kalendarz, z którego należy wybrać datę, dla jakiej chcemy przeprowadzić obliczenia. Następnie z rozwijalnych menu możemy wybrać województwo oraz powiat. Obliczenia możemy przeprowadzić dla całego województwa bądź dla całego powiatu.



Rysunek 5: Interfejs aplikacji - opcje

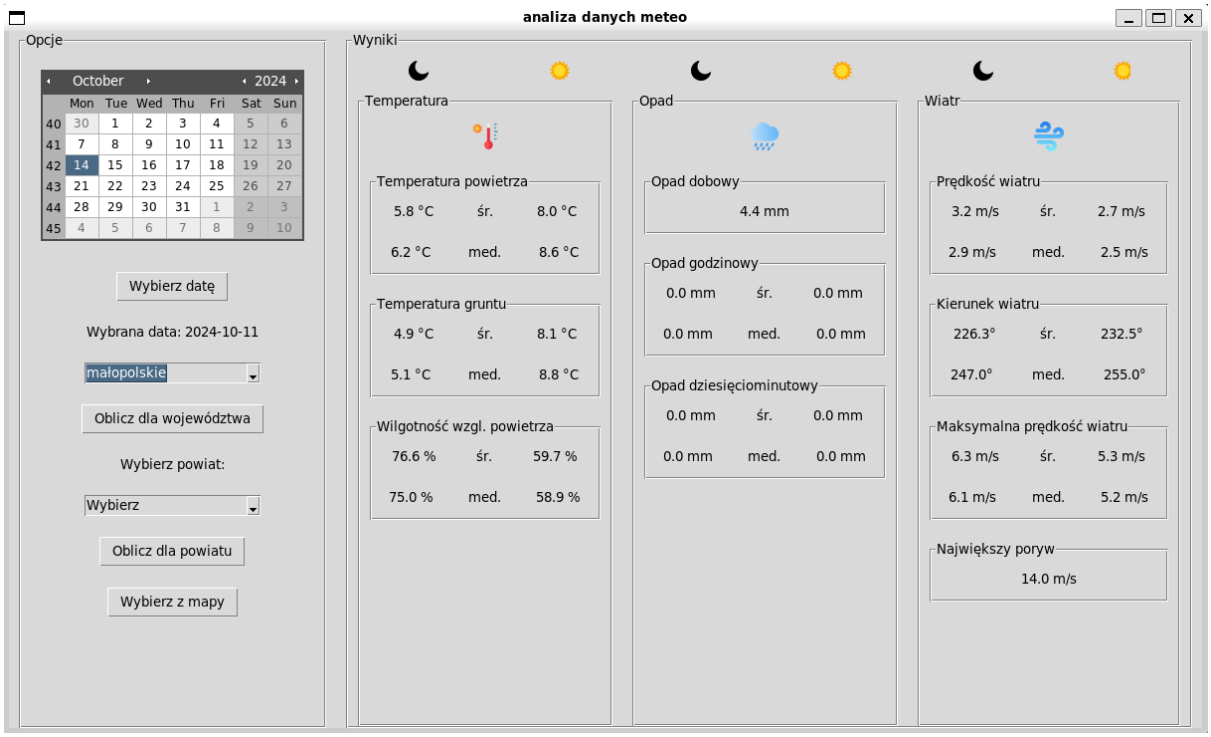
Istnieje również opcja wybrania pożądanego obszaru z mapy, po kliknięciu przycisku znajdującego się na dole ramki. Po wybraniu powiatu lub województwa i zatwierdzeniu wyboru, uruchomione zostaną obliczenia.

Rysunek 6: Interfejs aplikacji - mapa

Ramka wyników jest podzielona na 3 segmenty przedstawiające wartości dla kolejno temperatury, opadu oraz wiatru. W każdym segmencie po lewej stronie przedstawiono wartości dla nocy, a po prawej - dla

dnia.

Rysunek 7: Interfejs aplikacji - wyniki



Rysunek 8: Interfejs aplikacji

1.5.2 Działanie

Aplikacja po uruchomieniu łączy się z bazą danych MongoDB i pobiera z niej każde odrębne województwo oraz powiat. Zwrócone przez bazę wartości zapisuje jako możliwe do wyboru opcje dla list wyboru.

Przy wyborze odpowiedniego województwa lista powiatów zostaje uszczuplona jedynie do powiatów tego województwa, a przy wyborze powiatu zostaje ustawione odpowiednie województwo.
Po kliknięciu przycisków *Oblicz dla województwa* oraz *Oblicz dla powiatu* zostają uruchomione odpowiednie funkcje wybierające z bazy MongoDB stacje znajdujące się w wybranym powiecie lub województwie.
Następnie uruchomiona jest funkcja, odczytująca i zapisująca dane meteorologiczne.
Aby odpowiednio podzielić dane na dzień i noc, najpierw z bazy MongoDB odczytywany jest dla wybranego przez użytkownika dnia timestamp świtu oraz zmierzchu.
Po pobraniu tych danych, z bazy Redis zostają pobrane wszystkie rekordy dla wybranych stacji w zakresie wybranych timestampów i zapisane do odpowiedniego słownika w zależności od pory dnia, w której wykonano pomiar.
Następnie dla każdej pory dnia i każdego pomiaru obliczana jest średnia i mediana. Obliczone wyniki są zapisywane do GUI.

1.6 Wnioski

Wybrane bazy NoSQL: Redis oraz MongoDB mają zastosowanie w projekcie wykorzystującym dane przestrzenne. Dane te możemy przechowywać w obydwu bazach, w zależności od ich struktury oraz naszych potrzeb.

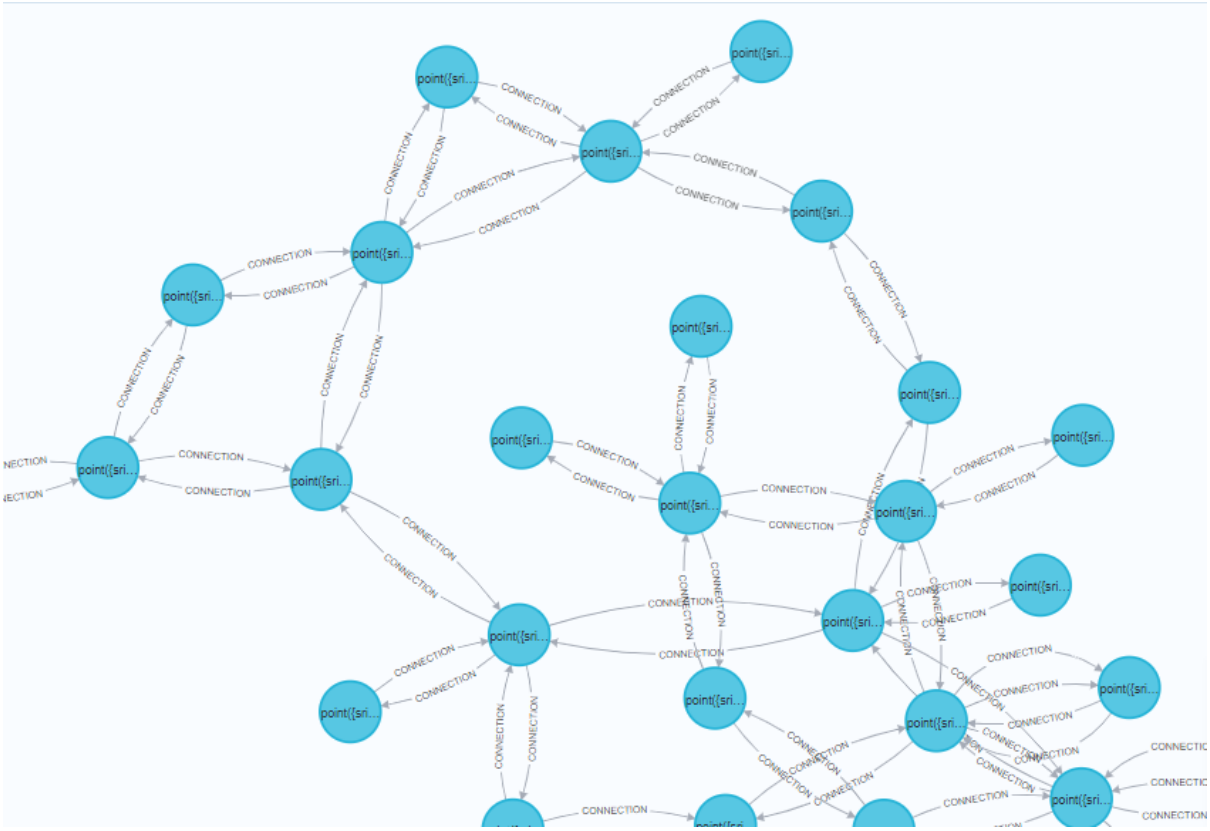
2 Projekt 2

2.1 Cel projektu

Celem tego projektu było uruchomienie algorytmów wyszukujących najkrótszą ścieżkę pomiędzy wybranymi węzłami w grafie z bloku A oraz test innych algorytmów grafowych biblioteki GDS na tym grafie.

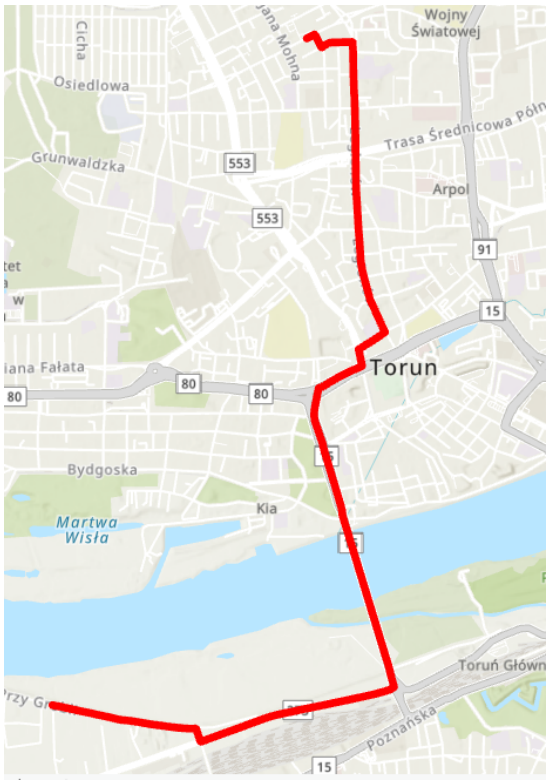
2.2 Wykonanie projektu

Stworzono grafową bazę danych Neo4j. Następnie za pomocą języka Python stworzono w niej graf, który zawiera wierzchołki - punkty początkowe i końcowe dróg z pliku shp oraz połączenia między nimi.

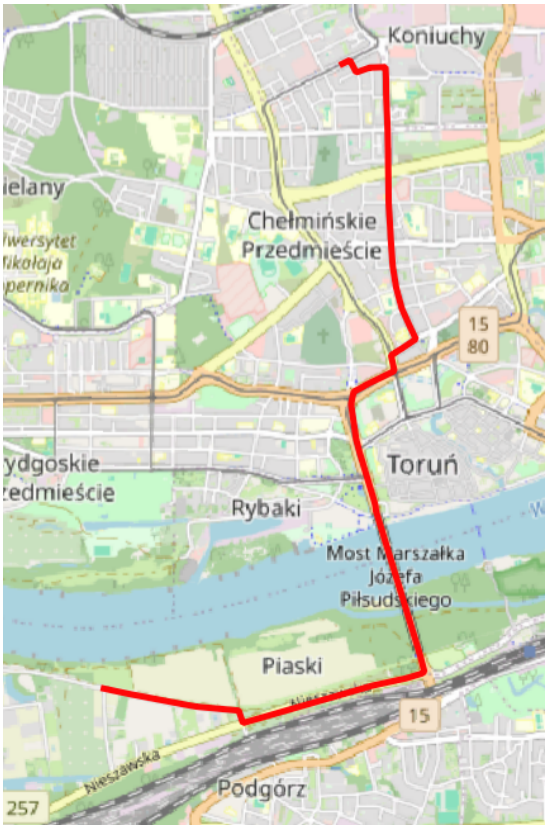


Rysunek 9: Fragment powstałego grafu w bazie Neo4j

Skorzystano z algorytmu Dijkstry by wyznaczyć dwie ścieżki: najkrótszą oraz najszybszą. Dodatkowo wykorzystano algorytm allShortestPaths.dijkstra, który znajduje wszystkie ścieżki, możliwe do przebycia z punktu początkowego w określonym czasie. Aby móc porównać wyniki uzyskane w bloku A oraz B stworzono funkcję, która zapisuje wyniki ww algorytmów do pliku geojson. Otrzymano następujące wyniki:

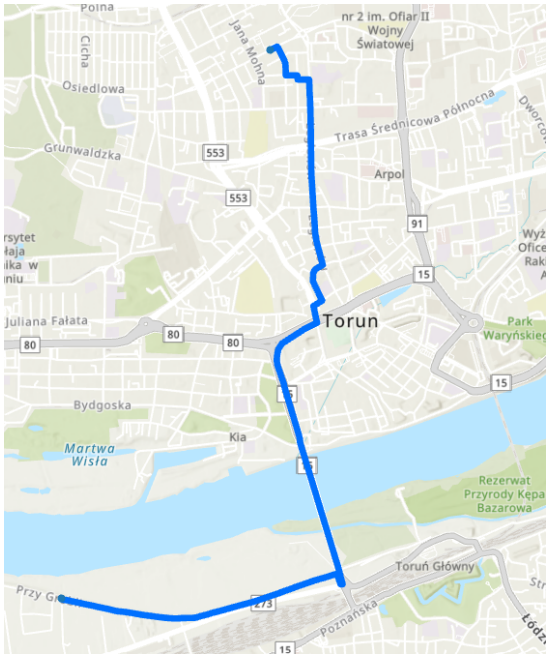


(a) Wynik w bloku A

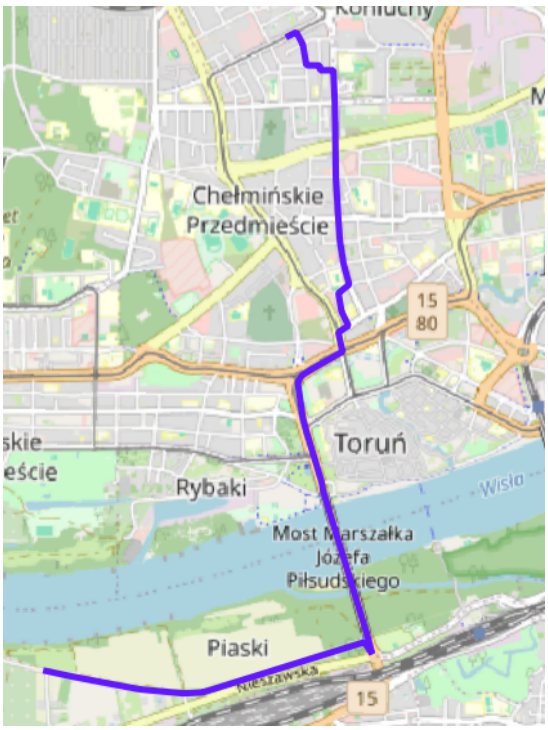


(b) Wynik bloku B

Rysunek 10: Najszybsza ścieżka między dwoma punktami

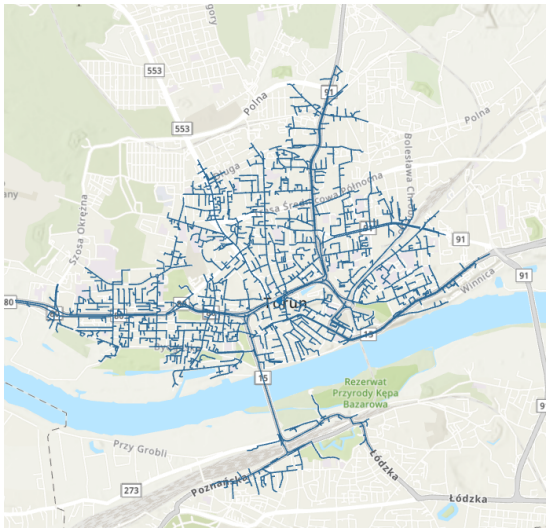


(a) Wynik w bloku A

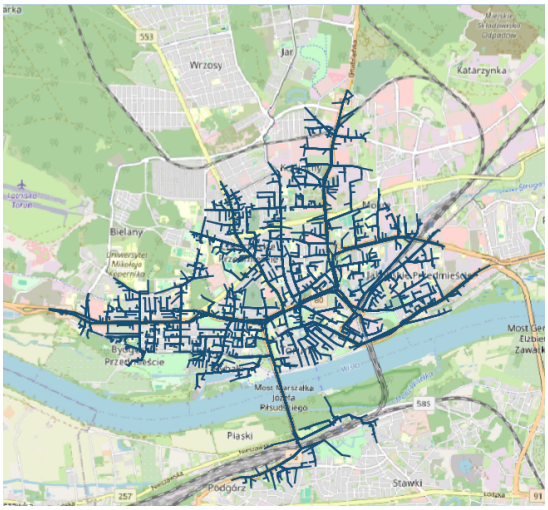


(b) Wynik bloku B

Rysunek 11: Najkrótsza ścieżka między dwoma punktami



(a) Wynik w bloku A



(b) Wynik bloku B

Rysunek 12: Ścieżki możliwe do pokonania w czasie 5 minut od punktu położonego przy Parku Etnograficznym w Toruniu

2.3 Wnioski

Wyniki uzyskane w bloku A oraz B są zgodne. Algorytmy zwracają poprawne wyniki, jednak działanie algorytmów z biblioteki GDS jest dużo szybsze od algorytmów opracowanych przez nas. Największym problemem podczas pracy nad projektem było załadowanie danych do bazy - trwało to stosunkowo długo.