[SDPB] Wyznaczanie trasy

Dokumentacja końcowa projektu

Michał Aniserowicz, Jakub Turek

Temat projektu

Napisać aplikację wyznaczającą i porównującą trasę przejazdu na terenie Warszawy środkami komunikacji miejskiej i samochodem z opcją ustawienia godziny wyjścia z domu żeby zdążyć na czas. Aplikacja może być aplikacją mobilną lub przeznaczoną na komputer klasy PC.

Założenia

Temat został uszczegółowiony poniższymi założeniami:

- Aplikacja zaprojektowana w architekturze klient-serwer.
- Serwer posiada dwie odpowiedzialności:
  - Przechowuje dane.
  - Dostarcza logikę związaną z przeliczaniem tras.
- Klient jest aplikacją dostępową umożliwiającą wprowadzenie następujących danych:
  - Lokalizacja początkowa i docelowa (punkty na mapie).
  - Docelowy czas przyjazdu.
  - Przejazd komunikacją miejską lub samochodem.
- Klient na wyjściu prezentuje najszybszą trasę dla podanych danych wejściowych oraz godzinę wyjścia/wyjazdu, która pozwala zdążyć na czas.
- Dane nie są dostarczane z systemów zewnętrznych (np. *Google Maps*, *Jak dojadę*), ale składowane w bazie danych opracowanej w ramach projektu.
- Uproszczenie algorytmu wyznaczania trasy:
  - Założenie, że czas przejazdu tego samego odcinka drogi komunikacją miejską i samochodem jest identyczny.
  - Brak uwzględnienia informacji o godzinach przyjazdu środków komunikacji miejskiej na przystanki. Zakłada się, że czas wymagany na przesiadkę jest stały i jest parametrem algorytmu.

  - Wyznaczana jest zawsze pojedyncza najszybsza trasa dla wybranego środka komunikacji.
  - Jako punkt początkowy i końcowy wybierane są te lokalizacje spośród danych znajdujących się w bazie, które są najbliższe lokalizacjom wskazanym przez użytkownika.
  - W przypadku wskazania komunikacji miejskiej jako środka transportu punktem początkowym i końcowym podróży są zawsze przystanki najbliższe wskazanym lokalizacjom. Nie jest uwzględniana możliwość dojścia do przystanku.

Model danych

W aplikacji został wykorzystany sieciowy model danych:

- Miasto jest opisane przy pomocy węzłów oraz łuków:
  - Współrzędne węzła (punkty) są opisane długością oraz szerokością geograficzną.
  - Węzły opisują różne lokalizacje miejskie: przystanki autobusowe, skrzyżowania, etc.
  - Łuki reprezentują połączenia drogowe pomiędzy węzłami. Są skierowane, więc można na nich opisać ruch jednokierunkowy.
  - Informacja o relacjach pomiędzy dwuwymiarowymi strukturami nie jest przechowywana.
- Nie ma ograniczeń odnośnie planarności sieci:
  - Testowy zbiór danych jest siecią planarną.
  - Aplikacja bez modyfikacji zadziała dla sieci nieplanarnych (tunele, przejazdy wielokondygnacyjne).

Reprezentacja logiczna modelu danych

Relacyjny model baz danych nie jest dobrze przystosowany do opisu modelu sieciowego. Informacje o stukturze grafowej można przechowywać na dwa sposoby:

- W postaci znormalizowanych tabel - osobne tabele dla wierzchołków i krawędzi.
- W postaci zdenormalizowanych tabel - rekord tabeli zawiera pełną informację o węźle i wszystkich połączonych z nim węzłach (wejściowych i wyjściowych). Węzły wyjściowe i wejściowe przechowywane są w postaci dwóch osobnych list.

W przypadku drugiego podejścia problemem jest przechowywanie informacji o koszcie danej krawędzi. Wymaga ona opakowania informacji w słownikową strukturę danych, co z kolei uniemożliwia wykonywanie zapytań języka SQL. Alternatywą może być przechowywanie tylko informacji o ścieżkach. Przy tym podejściu każdy rekord zawiera pełne

informacje o węźle początkowym i końcowym. Jest to jednak duży narzut pamięciowy, gdyż informacja o każdym węźle jest przechowywana n razy, gdzie n to stopień danego węzła.

Algorytm wyszukiwania połączeń w sieci wymaga dokonania wielu złączeń. Przykładowo dla postaci znormalizowanej, po wejściu do każdego węzła musimy najpierw pobrać informację o łukach, które ten węzeł tworzy, a następnie pobrać informacje o połączonych z nim węzłach. Jeżeli na etapie oszacowania kosztu potrzebna jest informacja o węzłach poprzedzających, wtedy wykonywane będą kolejne złączenia. Jest to istotny narzut obliczeniowy. Zakładając, że w ramach modelu danych będzie ujęte wiele typów relacji problem potęguje się.

Opisany w powyższym akapicie narzut nie wyklucza wykorzystania relacyjnej bazy danych do rozwiązania zadanego problemu. Autorzy projektu postanowili jednak skorzystać z innego podejścia i wykorzystać bazy danych o reprezentacji logicznej lepiej nadającej się do modelowania dziedziny problemu.

Interesującą alternatywę zapewnia ruch **Not Only SQL**, a konkretnie jego podzbiór - grafowe bazy danych. Ich logiczny model danych jest spójny z sieciowym modelem struktury miasta, dlatego bardzo dobrze nadają się do rozwiązania postawionego problemu. Jednym z najbardziej dojrzałych projektów grafowych baz danych jest **Neo4j** (`http://www.neo4j.org`). Zaletą tej bazy w kontekście tematyki przestrzennych baz danych jest posiadanie warstwy abstrakcji do obliczeń przestrzennych - **Neo4j Spatial**. Ze względu na wymienione zalety i odpowiedniość struktury do dziedziny problemu, do implementacji projektu autorzy skorzystali właśnie z tej bazy danych.

Schemat modelu danych miasta

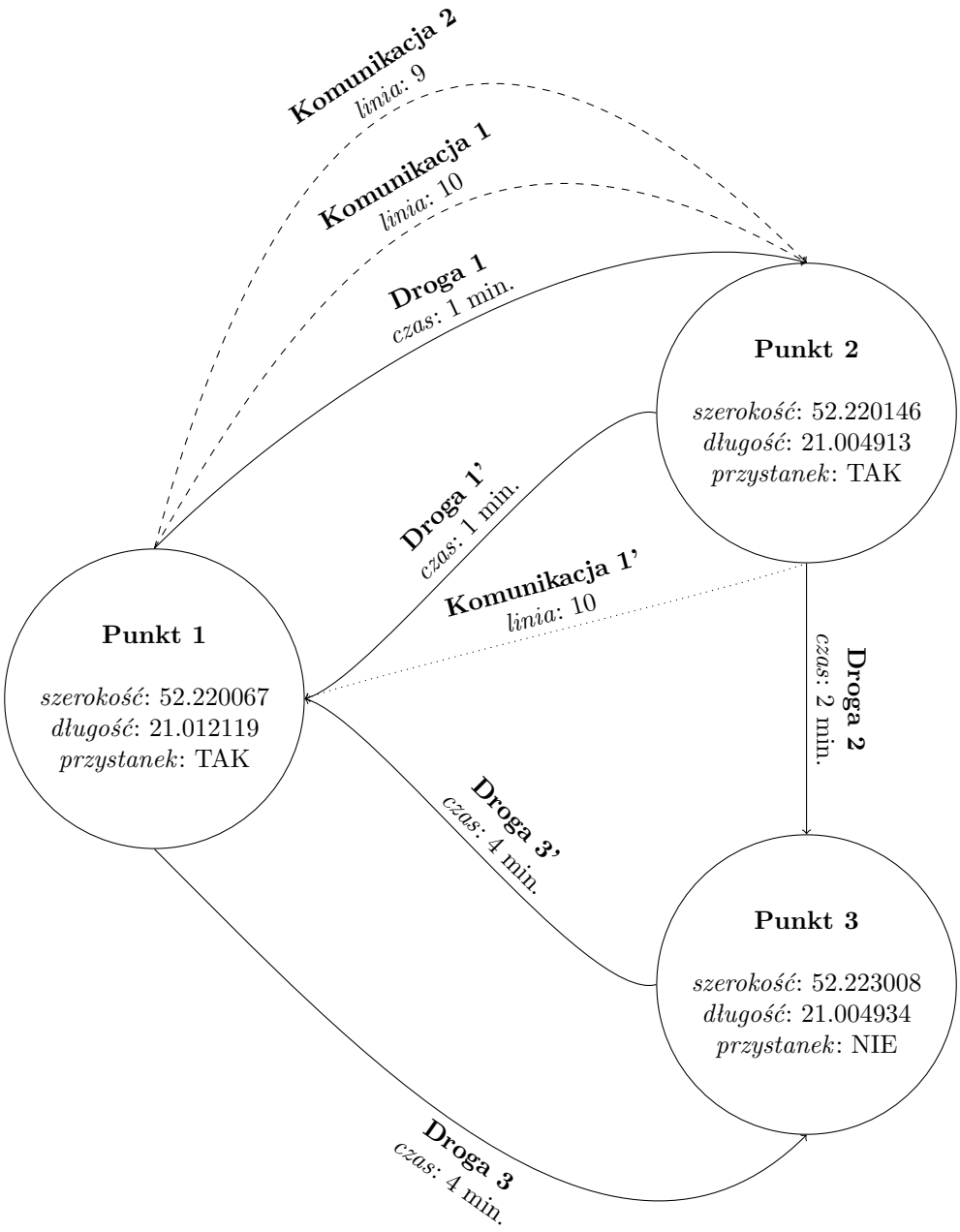
Wykorzystany model danych został schematycznie przedstawiony na rysunku 1.

Na schemacie pokazane są trzy punkty:

1. O współrzędnych $(52.220067, 21.012119)$. Przystanek komunikacji miejskiej.
2. O wpółrzędnych $(52.220146, 21.004913)$. Przystanek komunikacji miejskiej.
3. O współrzędnych $(52.223008, 21.004934)$.

Na schemacie pokazane są dwa typy relacji. Relacja narysowana pełną linią o nazwie **Droga** oznacza istnienie drogi pomiędzy punktami. Relacja narysowana linią przerywaną o nazwie **Komunikacja** oznacza natomiast istnienie ścieżki przejazdu komunikacji miejskiej jednej linii pomiędzy tymi punktami. Z relacji przedstawionych na schemacie można więc odczytać:

- Z punktu 1 do punktu 2 prowadzi droga 1, której przebycie zajmuje jedną minutę. Istnieje droga powrotna $1'$, która prowadzi z punktu 2 do punktu 1.
- Z punktu 1 do punktu 2 można dojechać dwoma liniami komunikacji miejskiej: 9 i 10. Z punktu 2 do punktu 1 wraca tylko linia numer 10.
- Punkty 2 i 3 połączone są drogą jednokierunkową, której przebycie zajmuje 2 minuty.



Rysunek 1: Schematyczne przedstawienie modelu danych miasta.

- Punkty 1 i 3 są połączone drogą dwukierunkową, której przejechanie zajmuje 4 minuty.

Indeksy

W modelu danych założone są dwa indeksy:

- Indeks bitmapowy na polu *przystanek*.
- Indeks przestrzenny na polach *szerokość* i *długość* geograficzna[1].

Metoda wyznaczania trasy

Algorytm wyznaczania najszybszej trasy jest dwukrokowy:

1. W bazie danych odnajdywane są dwa punkty: najbliższe początkowi i końcowi trasy spośród wszystkich punktów zdefiniowanych w bazie danych. W przypadku wyznaczania trasy komunikacji miejskiej wyznaczane są takie najbliższe punkty, że są one przystankami komunikacji.
2. Za pomocą algorytmu Dijkstry wyznaczana jest najszybsza droga pomiędzy znalezionymi punktami:
    - Dla trasy samochodowej/pieszej pod uwagę brane są tylko i wyłącznie łuki, które reprezentują relację **Droga**. Ewaluacja kosztu polega na dodawaniu czasu podróży dla każdego segmentu drogi.
    - Dla podróży komunikacją miejską pod uwagę brane są tylko i wyłącznie łuki, które reprezentują relację **Komunikacja**. Dla każdego segmentu wykonywana jest ewaluacja kosztu:
        - Koszt jest zwiększany o czas podróży dla danego segmentu.
        - Wykonywane jest sprawdzenie czy do węzła wejściowego analizowanego łuku prowadzi łuk, który obsługuje ta sama linia komunikacyjna. Jeżeli tak to przesiadka jest zbędna i ewaluacja kosztu kończy się. W przeciwnym razie koszt całkowity ścieżki jest powiększany o zadany czas oczekiwania na przesiadkę.

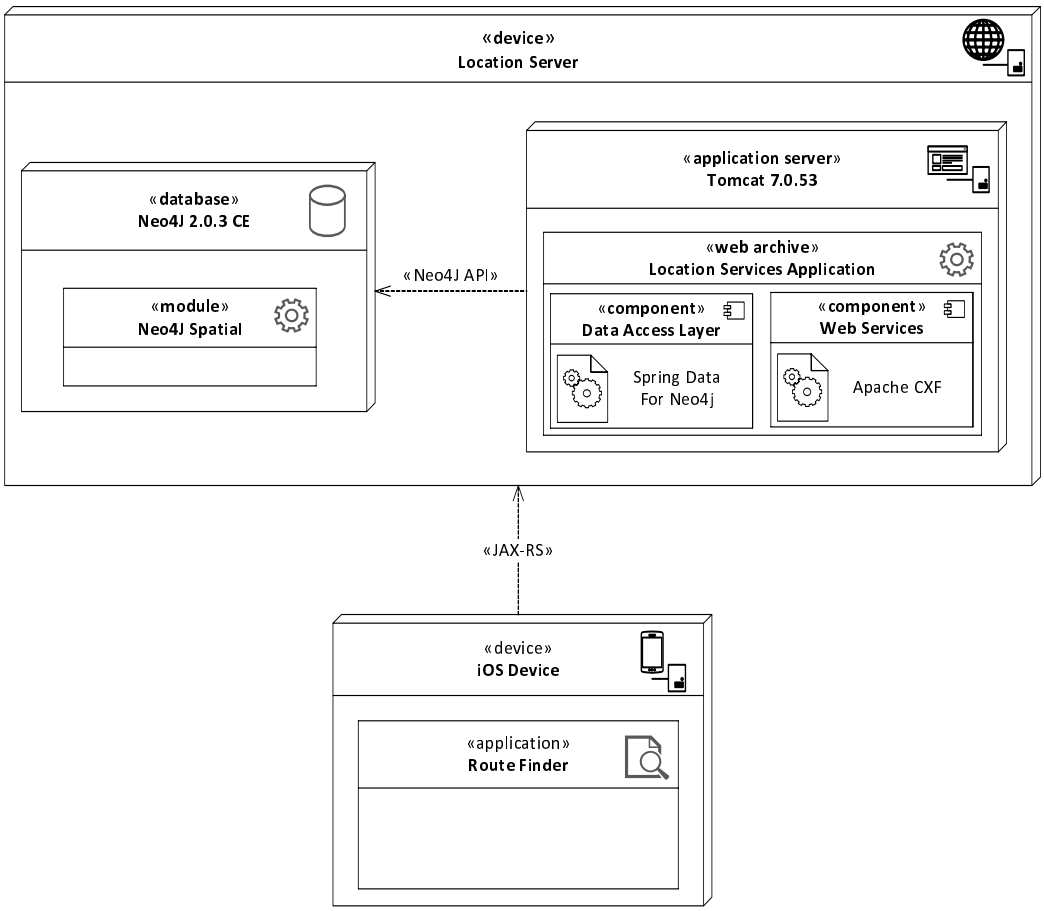
Architektura aplikacji

Aplikacja została zaprojektowana w architekturze klient-serwer. Szczegółowy schemat architektury przedstawia rysunek 2.

Budowanie i uruchamianie aplikacji

Moduł serwerowy oraz kliencki aplikacji budowane są osobno. Część serwerowa jest budowana przy użyciu aplikacji Maven (`http://maven.apache.org`). Uruchomienie polecenia **mvn clean install** w ścieżce projektu spowoduje ściągnięcie wszystkich zależności,

[1] W bazie danych Neo4j punkt jest reprezentowany jako łańcuch znaków WKT (Well-Known Text). Przykład: `POINT( 52,223008 21,004934 )`.



Rysunek 2: Schemat architektury aplikacji.

uruchomienie testów jednostkowych oraz integracyjnych, a także wygenerowanie pliku web-archive (**.war**), który następnie trzeba *zdeployować* na serwer aplikacji Java.

Budowanie aplikacji klienckiej jest realizowane przy pomocy menadżera zależności CocoaPods (`http://cocoapods.org`). Uruchomienie polecenia **pod install** w podfolderze z klientem spowoduje ściągnięcie wszystkich zależności oraz wygenerowanie pliku projektu programu Xcode, za pomocą którego można skompilować i wyeksportować aplikację.