Jakub Radzik
260366
Sprawozdanie z Listy 1

Zadanie 1

a) algorytm wyszukiwania najkrótszej ścieżki z A do B za pomocą algorytmu algorytmem Dijkstry w oparciu o kryterium czasu (10 punktów)
b) algorytm wyszukiwania najkrótszej ścieżki z A do B za pomocą algorytmu A* w oparciu o kryterium czasu (25 punktów)
c) algorytm wyszukiwania najkrótszej ścieżki z A do B za pomocą algorytmu A* w oparciu o kryterium przesiadek (25 punktów)
d) modyfikacja algorytmu A* z punktów (b) lub (c), który pozwoli na zmniejszenie wartości funkcji kosztu uzyskanego rozwiązania lub czasu obliczeń (10 punktów)
    - Dla A* (kryterium czasowe) początkowo funkcja h bazowała tylko na odległości między punktami (funkcja zwracała odległość w kilometrach z dokładnością do 3 cyfr po przecinku)
    - Modyfikacja polegała na założeniu jakiejś prędkości stałej (dla mnie to 15km/h) i zwróceniu czasu podróży w minutach, co pozwoliło zmniejszyć znacząco **czasy obliczeń –** w kosztach nie było znaczących różnic

Czas obliczeń

Trasa	Bazowanie na odległości wyrażonej w kilometrach	Bazowanie na czasie dojazdu wyrażonym w minutach
LEŚNICA –> BISKUPIN	1273 ms	493 ms
LEŚNICA -> Rynek	325 ms	255 ms
LEŚNICA -> KROMERA	832 ms	330 ms
Hala Stulecia -> Jerzmanowska nr 9	1395 ms	497 ms

- **opis teoretyczny**
  Algorytmy znajdowania ścieżek w grafie, czy w naszym przypadku we Wrocławiu.

- **przykładowe zastosowania**
  Znajdowanie najszybszych połączeń między przystankami.

- **opisy bibliotek**
    - typescript – w czymś trzeba pisać
    - csv-parser – jakoś musiałem wczytać te dane
    - ts-node – chciałem jakoś pliki typescript uruchamiać

- moment – biblioteka która w łatwy sposób pozwala operować na obiektach daty i czasu
- jest – pisałem testy jednostkowe do metod, ale potem zainstalowałem moment i praktycznie nie było czego testować

Problemy implementacyjne:

- Ogromną korzyścią tego przedmiotu jest na pewno to, że zmusił mnie do szybkiego nauczenia się używania Debuggera w VSCode i bynajmniej nie przez ciągłe używanie console.log()
- Między niektórymi przystankami czas podróży wynosił 0 – przez przypadek tworzyłem dwa Node-y o różnych nazwach, ale na dane Node'a startowego.
- Operacja na godzinach – przy tworzeniu obiektu moment z godziny musimy podać drugi parametr jakim jest format daty jaką podaliśmy w pierwszym parametrze, bez tego operacje typu liczenie różnic między godzinami zwracały NaN.
  moment('15:20') -> moment('15:20','HH:mm')
- Kilka godzin zabrało mi zrozumienie jak zarządzać czasem przy sprawdzaniu kolejnych node'ów w grafie w algorytmie Dijkstry. Jedna sprawa to taka, że mamy godzinę dotarcia na przystanek i fajnie by było to wielokrotnie używać w jakiś sposób więc na początku mamy inicjalizowany obiekt daty za pomocą godziny startu, a potem z każdą zmianą przystanku dodajemy czas przejazdu do tego obiektu, kiedy skończymy sprawdzać wszystkie Edge w celu znalezienia najtańszego, odejmujemy czas podróży między nimi.
  Kolejnym problemem było to, że na prostych trasach algorytm zaczął działać ale dla takich gdzie trzeba przejechać przez centrum już nie bo za dużo możliwości się tam robiło. Okazało się, że na prostej trasie przesiadaliśmy się co przystanek co było spowodowane złym liczeniem kosztu, ostatecznie kosztem jest suma 3 składowych:
  - Czas jaki mieliśmy na poprzednim przystanku
  - Czas podróży od poprzedniego przystanka
  - Czas jaki upływa od dotarcia na przystanek do odjazdu z niego (gdy się nie przesiadamy wynosi on 0)
- Jeśli godzina jest wcześniej od startowej to trzeba jakoś zaznaczyć, że wcześniejsza godzina jest jutro dopiero
- W A* czasowej też był problem z kosztem – tam po prostu nie użyto czasu na poprzednim przystanku – dodano właściwość dla każdego węzła – currentDuration aby nie mieszać czasu podróży z kosztem i uniknąć jakichś kosmicznych liczb