Projekt zaliczeniowy z języka Python

„Zastosowanie algorytmu Kruskala w poszukiwaniu najkrótszego połączenia autostradowego pomiędzy wszystkimi wsiami i miastami Polski.”

Autor: Michał Bularz

Język: Python 2.7

Kraków, 14.01.2017

1. Opis plików

- **separateSets.py** – implementacja sumowalnych zbiorów rozłącznych stanowiących podstawę algorytmu Kruskala. Zbiór jest tutaj reprezentowany w postaci listy.

- **graph.py** – implementacja grafu na którym działa algorytm Kruskala w postaci listy wierzchołków będących obiektami i listy krawędzi. Krawędź jest tutaj zaimplementowana jako para wierzchołków (obiektów). Lista wierzchołków jest ostatecznie słownikiem, ponieważ była potrzeba zmapowania wartości w wierzchołku (np. litery) na obiekt tego wierzchołka przy wczytywaniu danych w postaci napisów (nazw miast i wsi).

- **kruskal.py** – implementacja algorytmu Kruskala korzystająca z powyżej wymienionych plików. Na początku wczytywane są dane, a następnie wykonywany jest algorytm Kruskala na stworzonym grafie. Program wypisuje do konsoli ilość krawędzi minimalnych oraz ich sumę, natomiast do pliku **„min_tree.txt”** zapisuje poszczególne krawędzie oraz do pliku **„edges.dat”** zapisuje wierzchołki grafu wraz z współrzędnymi (będzie to potrzebne do wizualizacji).

- **miasta_wspolrzedne.txt** - plik zawierający listę miast (i niekiedy dzielnice tych miast) Polski wraz z współrzędnymi geograficznymi.

- **GeneratorMiast.java** – program który na podstawie pliku „miasta_wspolrzedne.txt” oblicza (z długości sferycznej) w przybliżeniu rzeczywistą odległość [km] każdego miasta z każdym (redundantne przypadki obliczania odległości z A do B, znając już odległość z B do A są pomijane). Program generuje różną ilość danych w postaci listy wierzchołków (razem ze współrzędnymi dziesiętnymi) i listy krawędzi (z odległościami między miastami) w zależności od użytego kroku (opis poniżej) i zapisuje te dane w postaci plików tekstowych w stworzonym folderze „wygenerowaneDane”. *Uwaga: program pochodzi z innego mojego projektu o podobnej tematyce, dlatego zdecydowałem się na ponowne użycie programu.*

- **draw_cities.pg** – skrypt gnuplota rysujący tylko wierzchołki (miasta) w zależności od użytego pliku z listą miast (domyślnie wartość kroku = 1). Wartość kroku trzeba ręcznie zmienić w skrypcie w nazwie pliku z danymi przy poleceniu „plot”. Wynik programu zapisywany jest do pliku **cities.png**.

- **draw_cities._with_edges.pg** – skrypt gnuplota rysujący cały graf pełny (wierzchołki z krawędziami) dla ustalonego kroku. Wartość kroku trzeba ręcznie zmienić w skrypcie w nazwie pliku z danymi przy poleceniu „plot”. Wynik programu zapisywany jest do pliku **cities_edges.png**.

- **draw_min_tree.pg** – skrypt gnuplota rysujący znalezione minimalne drzewo rozpinające zapisane w pliku „min_tree.txt”. Wynik programu zapisywany jest do pliku **min_tree.png.**

- **makefile** – plik makefile ułatwiający korzystanie z programu generatora miast.

- **folder „wygenerowaneDane”** zawierający wszystkie dane wytworzone przez program GeneratorMiast.java, które później są wykorzystywane przez program kruskal.py i inne. Więcej informacji w sekcji „Kompilacja i uruchamianie”.

- **min_tree.txt** – plik powstały po uruchomieniu algorytmu Kruskala. Zawiera listę minimalnych krawędzi potrzebną do wizualizacji minimalnego drzewa rozpinającego.

- **pliki graficzne png** – wyniki wymienionych skryptów gnuplota.

2. Kompilacja i uruchamianie

a) Generowanie danych wejściowych – generator miast

Kompilacja: polecenie ***make***
Uruchomienie generatora:
Wszystkich miast (wierzchołków) w pliku miasta_wspolrzedne.txt jest 2318. Dwa z nich się powtarzają, część nie leży w obrębie Polski dlatego są pominięte przy wczytywaniu.
Ostatecznie wszystkich wierzchołków jest 2273.

Dane można wygenerować na dwa sposoby.

1. Wygenerowanie wszystkich plików z danymi o różnej ilości wierzchołków

Polecenie: ***make runGenAll***

Spowoduje to uruchomienie serii programów, które będą generować pliki z danymi. Parametrem wywołania każdego z tych programów jest wartość kroku.

Dla kroku=1 wszystkie 2273 wierzchołków zostanie użytych w generatorze.
Dla kroku=2 co drugi wierzchołek będzie użyty, czyli będzie 1137 wierzchołków.
Dla kroku=3 co trzeci wierzchołek będzie użyty, czyli będzie 569 wierzchołków.
…

Dla kroku=k co k-ty wierzchołek będzie użyty, czyli będzie użyty, czyli będzie $\lceil \frac{2273}{k} \rceil$ wierzchołków.

Po zakończeniu działania programu w folderze „wygenerowaneDane” pojawią się pliki ***lista_krawedzi_k.txt*** oraz ***lista_miast_k.txt***, gdzie k jest wartością kroku.

2. Wygenerowanie plików z ustaloną wartością kroku

Polecenie: ***make runGen ARGS=k*** (zamiast k wpisać liczbę, $0 < k <= 50$)

Spowoduje to uruchomienie programu, który wygeneruje tylko pliki z zadaną wartością kroku.

b) Poszukiwanie minimalnego drzewa rozpinającego

Polecenie: ***python kruskal.py k*** (zamiast k wpisać liczbę, 0<k<=50). W przypadku nie podania żadnego argumentu program domyślnie wczyta dane z wartością k = 1).

Wartość k określa które dane (o jakiej wielkości) zostaną wczytane do programu. Im mniejsze k tym mniejsze dane.

c) Wizualizacja grafu – rysowanie tylko wierzchołków grafu

Aby wygenerować wykres przedstawiający wszystkie wierzchołki w grafie należy uruchomić skrypt draw_cities.pg poleceniem: ***gnuplot draw_cities.pg***

Domyślnie skrypt wykona się z wartością kroku k=1. Jednak dla lepszej czytelności zaleca się użyć największego kroku k = 50.

d) Wizualizacja grafu – rysowanie całego grafu

Aby wygenerować wykres przedstawiający cały graf pełny należy uruchomić skrypt draw_cities_with_edges.pg poleceniem: ***gnuplot draw_cities_with_edges.pg***

Domyślnie skrypt wykona się z wartością kroku k=1. Jednak dla lepszej czytelności zaleca się użyć największego kroku k = 50. Im mniejsze k, tym graf jest bardziej nieczytelny.

e) Wizualizacja minimalnego drzewa rozpinającego

Aby wygenerować wykres przedstawiający minimalne drzewo rozpinające grafu należy uruchomić skrypt draw_min_tree.pg poleceniem: ***gnuplot draw_min_tree.pg***

Domyślnie skrypt wykona się z wartością kroku k=1. Jednak dla lepszej czytelności zaleca się użyć największego kroku k = 50.

Instrukcja poprawnego wykonania programów:

1. Uruchomić generator danych
2. Uruchomić program kruskal.py
3. Narysować mapę wierzchołków, cały graf i minimalne drzewo rozpinające (dla k przyjętego w ostatnim wywołaniu kruskal.py, wartość k zmienić w skryptach)

3. Użyte algorytmy

W projekcie został użyty **algorytm Kruskala o złożoności czasowej O(E*logV)**, który znajduje minimalne drzewo rozpinające przy użyciu struktury zbiorów rozłącznych dla grafu nieskierowanego, spójnego oraz ważonego. Graf zbudowany z miast i wsi Polski spełnia te 3 założenia ponieważ

- graf pozostaje nieskierowany ponieważ przy wczytywaniu wierzchołków i tworzeniu krawędzi redundantne połączenia z B do A znając już połączenie z A do B są pomijane
- graf jest spójny, co jest sprawdzane dodatkowo przy użyciu struktury zbiorów rozłącznych
- graf jest ważony ponieważ każda krawędź na swoją wagę (odległość między miastami). Dodatkowo każda waga > 0.

Pseudokod:

```
Kruskal(G): // G – graf, N – liczba wierzch., M – liczba kraw.
     foreach v in G.V:                          // O(N)
          create_set(v)

     sort(G.E) // rosnąco wzlędem wag           // O(M*logM) timsort

     foreach (u,v) in G.E:                      // O(M*logN)
          if find_set(u) != find_set(v):
               union(find_set(u), find_set(v))
```

Podsumowanie złożoności:

$M = O(N^2) \rightarrow M <= N(N-1)$ stąd:
$O(M*logM + M*logN) = O(M*(logM + logN)) <= O(M*(2logN + logN)) <=$ **O(M*logN)**

4. Testy

Poniżej przeprowadzono testy programu kruskal.py dla danych o k=1, k=10, k=50.
Wydajność algorytmu dla takich danych przedstawia poniższa tabelka:

Wartość kroku	Liczba wierzchołków	Liczba krawędzi	Czas operacji [s]
k = 1	2 273	2 582 128	42.90
k = 10	228	25 878	0.14
k = 50	46	1035	0.005687

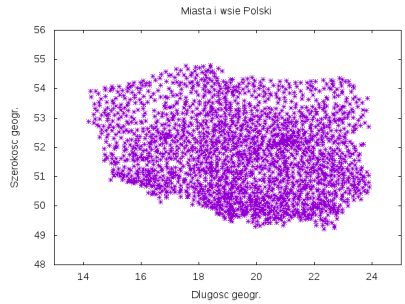
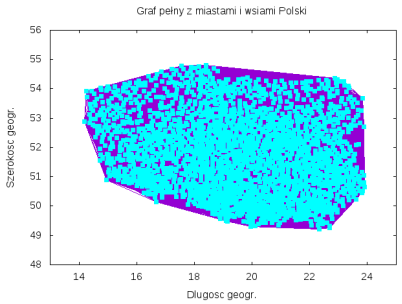
Wyniki testów

- **k = 1**

Wynik w konsoli:

```
Liczba wierzcholkow: 2273
Liczba krawedzi: 2582128

liczba krawedzi minimalnych: 2272
suma wag: 24819.79
```

Graf z samymi wierzchołkami	Graf pełny	Drzewo rozpinające
		

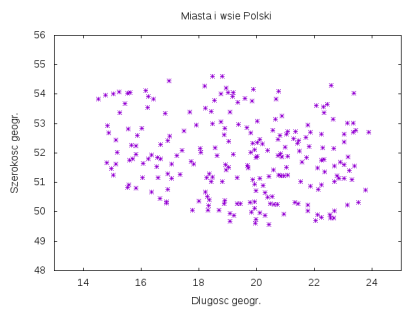
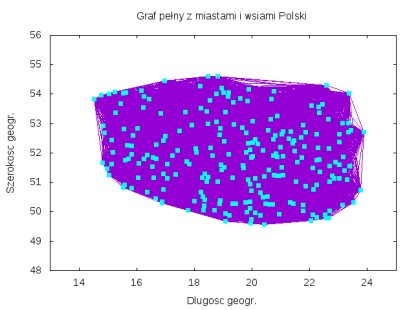
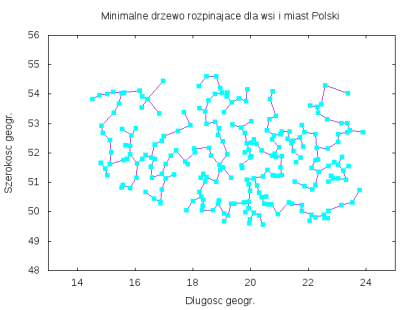
Uwaga: dla drzewa rozpinającego wyłączono wyświetlanie wierzchołków dla lepszej czytelności.

- **k = 10**

Wynik w konsoli:

```
Liczba wierzcholkow: 228
Liczba krawedzi: 25878

liczba krawedzi minimalnych: 227
suma wag: 7057.25
```

Graf z samymi wierzchołkami	Graf pełny	Drzewo rozpinające
		

- **k = 50**

Wynik w konsoli:

```
Liczba wierzcholkow: 46
Liczba krawedzi: 1035

liczba krawedzi minimalnych: 45
suma wag: 3139.64
```

Graf z samymi wierzchołkami	Graf pełny	Drzewo rozpinające