Piotr Karoń, 241 626

Wrocław, dn. 29.05.2019

Zadanie projektowe 2.

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.

PROWADZĄCY:

dr Jarosław Mierzwa

Spis treści

1 Założenia projektowe

1.1 Cel

Celem projektu jest zbadanie efektywności algorytmów Kruskala, Prima, Dijkstry i Bellmana-Forda w zależności od sposobu reprezentacji grafu i wielkości instancji.

1.2 Technologie

Do implementacji wymienionych struktur użyto języka ***Kotlin*** w wersji ***Native***, która jest kompilowana do kodu maszynowego danej platformy.

1.3 Przebieg eksperymentu

Badania przeprowadzone zostały dla wierzchołków w liczbie: 10, 100, 200, 400, 800, 1000, 2000 dla każdej liczby w gęstościach 25%, 50%, 75%, 99% osobno dla reprezentacji macierzowej i listowej. Każdy test wykonano 50 razy, a czas uśredniono z wszystkich prób.

2 Złożoności czasowe algorytmów

Przy tworzeniu poniższych opisów korzystano z `Wprowadzenia do algorytmów` autorstwa T. Cormen, C. Leiserson, R. Rivest [1], materiałów udostępnionych na stronie dr Tomasza Kapłona [3] oraz stronie I LO w Tarnowie [5]

2.1 Algorytm Kruskala

Czas działania algorytmu Kruskala dla grafu $G = (V, E)$ zależy od sposobu implementacji struktury zbiorów rozłącznych. W tym projekcie wykorzystano implementację lasu zbiorów rozłącznych z łączeniem według rangi i z kompresją ścieżek.

Całkowity czas działania algorytmu dla reprezentacji listowej wynosi $O(Elog_2E)$. Dla reprezentacji macierzowej – $O(E^2)$.

2.2 Algorytm Prima

Szybkość działania algorytmu Prima zależy od sposobu implementacji kolejki priorytetowej przechowującej koszty dojścia do danego wierzchołka. W tym projekcie wykorzystano kopiec

stworzony w projekcie nr 1, dzięki czemu otrzymano złożoność $O(Elog_2V)$ dla reprezentacji listowej. Dla macierzy sąsiedztwa wynosi ona $O(E^2)$.

2.3 Algorytm Dijkstry

Algorytm Dijkstry działa poprawnie tylko jeśli w grafie nie znajdują się ścieżki o ujemnej wadze. Szybkość algorytmu zależy od sposobu implementacji kolejki priorytetowej przechowującej koszty dojścia do danego wierzchołka. W tym projekcie wykorzystano kopiec stworzony w projekcie nr 1, dzięki czemu otrzymano złożoność $O(Elog_2V)$ dla listy sąsiedztwa. Dla reprezentacji macierzowej otrzymano złożoność $O(E^2)$.

2.4 Algorytm Bellmana-Forda

Algorytm Bellmana-Forda jest prostym algorytmem, który wykonuje $V-1$ iteracji sprawdzających sąsiadów danego wierzchołka i relaksujących krawędzie. W przypadku

3 Wyniki

4 Podsumowanie

Dla reprezentacji macierzowych bardzo dobrze widoczna jest złożoność rzędu $O(E^2)$. Jej testy były bardzo obciążające dla komputera. Dla reprezentacji listowej na pierwszy rzut oka nie widać odpowiedniej krzywej. Biorąc jednak pod uwagę skalę i porównując z reprezentacją macierzową można założyć, że jest to złożoności logarytmiczne.

Dużym problemem w języku Kotlin okazała się inicjalizacja dużej tablicy sąsiedztwa (powyżej 1000 wierzchołków). To było najbardziej czaso chłonne zadanie do wykonania.

Bibliografia

[1] T. Cormen, C. Leiserson, R. Rivest *Wprowadzenie do algorytmów*, Wydawnictwa Naukowo-Techniczne Warszawa, Wyd. IV, 2004

[2] [1] str. 304.

[3] `tomasz.kaplon.staff.iiar.pwr.wroc.pl/`, strona dr Tomasza Kapłona

[4] `kotlinlang.org/docs/reference/native-overview.html`, dokumentacja języka Kotlin/Native

[5] `eduinf.waw.pl`, materiały na stronie I LO w Tarnowie