## Struktury danych i złożoność obliczeniowa

Projekt nr. 2: Badanie algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.

**Bartosz Czerwiński**

**Nr. Indeksu: 218215**

1. **Wstęp**

**Graf jest to struktura złożona z wierzchołków i krawędzi. Grafy mogą być skierowane tzn. że krawędź ma określony zwrot, oraz nieskierowane gdzie krawędź jest obustronnie przepustowa.**

**Graf w pamięci komputera można przechowywać na wiele sposobów. W moim projekcie, zgodnie z założeniami jest to lista sąsiedztwa i macierz sąsiedztwa.**

**Lista sąsiedztwa została zaimplementowana jako tablica list par:**list<pair <int, int>> \*TabList;   
//Tablica list Tablist[wierzchołek(skąd)].first-dokąd, .second -waga

**Macierz sąsiedztwa została zaimplementowana jako tablica dwuwymiarowa o wielkości V x V:** int \*\*Tab;

1. **Algorytmy**

**W swoim projekcie zaimplementowałem trzy algorytmy, Prima, Kruskala i Dijkstry. Wszystkie te algorytmy są zaimplementowane podwójnie. Jedna implementacja dla listy i druga dla macierzy.**

1. **Algorytm Prima i Kruskala**

**Algorytmy te poszukują minimalnego drzewa spinającego, tzn. takiego drzewa który zawiera wszystkie wierzchołki grafu. Krawędziami drzewa są krawędzie grafu dobrane tak by koszt drzewa był najmniejszy. Algorytmy te różnią się reprezentacją wewnętrznej struktury.**

1. **Algorytm Dijkstry**

**Algorytm ten poszukuje najkrótszej drogi w grafie z początkowego wierzchołka do każdego innego. Algorytm po skończonej pracy wyświetla koszt przejścia ścieżki oraz drogę jaką trzeba pokonać.**

1. **Testy**

**Testy algorytmów zostały przeprowadzone dla losowych grafów o wielkościach 80, 100, 120 i 140 wierzchołków, dodatkowo każda ta instancja została przetestowana pod innym zapełnieniem procentowym 25, 50, 75, i 99%. Każdy z testów został wykonany 100 razy, a wyniki zostały uśrednione średnią arytmetyczną.**

1. **Algorytm Prima – Lista**

|  |  |  |
| --- | --- | --- |
| Liczba wierzchołków | Gęstość [%] | Czas[ms] |
| 80 | 25% | 20,209 |
| 50% | 23,112 |
| 75% | 24,111 |
| 99% | 24,999 |
| 100 | 25% | 41,347 |
| 50% | 42,061 |
| 75% | 42,7 |
| 99% | 43,799 |
| 120 | 25% | 65,54 |
| 50% | 68,592 |
| 75% | 72,751 |
| 99% | 74,567 |
| 140 | 25% | 97,049 |
| 50% | 98,083 |
| 75% | 99,142 |
| 99% | 100,075 |
| 160 | 25% | 161,019 |
| 50% | 172,034 |
| 75% | 174,012 |
| 99% | 177,625 |

1. **Algorytm Prima – Macierz**

|  |  |  |
| --- | --- | --- |
| Liczba wierzchołków | Gęstość [%] | Czas[ms] |
| 80 | 25% | 19,276 |
| 50% | 19,645 |
| 75% | 24,14 |
| 99% | 28,897 |
| 100 | 25% | 35,623 |
| 50% | 39,144 |
| 75% | 53,341 |
| 99% | 67,032 |
| 120 | 25% | 72,154 |
| 50% | 74,217 |
| 75% | 76,791 |
| 99% | 85,934 |
| 140 | 25% | 117,154 |
| 50% | 128,99 |
| 75% | 139,678 |
| 99% | 155,243 |
| 160 | 25% | 173,815 |
| 50% | 177,226 |
| 75% | 180,735 |
| 99% | 211,003 |

1. **Algorytm Kruskala – lista**

|  |  |  |
| --- | --- | --- |
| Liczba wierzchołków | Gęstość [%] | Czas[ms] |
| 80 | 25% | 0,906 |
| 50% | 1,131 |
| 75% | 1,208 |
| 99% | 1,24 |
| 100 | 25% | 0,869 |
| 50% | 1,49 |
| 75% | 2,126 |
| 99% | 2,562 |
| 120 | 25% | 1,235 |
| 50% | 1,462 |
| 75% | 1,635 |
| 99% | 2,124 |
| 140 | 25% | 1,853 |
| 50% | 2,124 |
| 75% | 2,851 |
| 99% | 3,448 |
| 160 | 25% | 1,78 |
| 50% | 1,795 |
| 75% | 3,189 |
| 99% | 4,391 |

1. **Algorytm Kruskala – Macierz**

|  |  |  |
| --- | --- | --- |
| Liczba wierzchołków | Gęstość [%] | Czas[ms] |
| 80 | 25% | 0,984 |
| 50% | 1,161 |
| 75% | 1,253 |
| 99% | 1,432 |
| 100 | 25% | 0,888 |
| 50% | 1,17 |
| 75% | 1,378 |
| 99% | 2,852 |
| 120 | 25% | 1,216 |
| 50% | 1,341 |
| 75% | 1,487 |
| 99% | 1,772 |
| 140 | 25% | 1,116 |
| 50% | 1,813 |
| 75% | 2,417 |
| 99% | 3,113 |
| 160 | 25% | 1,478 |
| 50% | 1,551 |
| 75% | 3,008 |
| 99% | 4,153 |

1. **Algorytm Dijkstry - lista**

|  |  |  |
| --- | --- | --- |
| Liczba wierzchołków | Gęstość [%] | Czas[ms] |
| 80 | 25% | 0,037 |
| 50% | 0,051 |
| 75% | 0,128 |
| 99% | 0,151 |
| 100 | 25% | 0,075 |
| 50% | 0,143 |
| 75% | 0,23 |
| 99% | 0,219 |
| 120 | 25% | 0,092 |
| 50% | 0,163 |
| 75% | 0,318 |
| 99% | 0,546 |
| 140 | 25% | 0,117 |
| 50% | 0,241 |
| 75% | 0,369 |
| 99% | 0,403 |
| 160 | 25% | 0,225 |
| 50% | 0,502 |
| 75% | 0,673 |
| 99% | 0,824 |

1. **Algorytm Dijkstry – macierz**

|  |  |  |
| --- | --- | --- |
| Liczba wierzchołków | Gęstość [%] | Czas[ms] |
| 80 | 25% | 0,036 |
| 50% | 0,045 |
| 75% | 0,056 |
| 99% | 0,068 |
| 100 | 25% | 0,065 |
| 50% | 0,081 |
| 75% | 0,089 |
| 99% | 0,113 |
| 120 | 25% | 0,075 |
| 50% | 0,091 |
| 75% | 0,128 |
| 99% | 0,161 |
| 140 | 25% | 0,114 |
| 50% | 0,136 |
| 75% | 0,176 |
| 99% | 0,243 |
| 160 | 25% | 0,124 |
| 50% | 0,158 |
| 75% | 0,171 |
| 99% | 0,206 |

1. **Wnioski**

Powyższe testy pokazują jak wyglądają faktyczne dane czasowe dla poszczególnych algorytmów i reprezentacji. Biorąc pod uwagę różnice w szybkości działania algorytmu ze względu na jego reprezentacje mogli byśmy przewidywać, że reprezentacja listowa zachowa się lepiej. Niestety testy pokazały coś odwrotnego w przypadku algorytmu Dijkstry. Powodów takiej rozbieżności teorii z praktyka może być szybkość działania iteratora struktur listy w c++.

Algorytm Prima zachował się bardzo przewidywalnie i jego reprezentacja listowa jest minimalnie szybsza od macierzowej.

Algorytm Kruskala okazał się bardzo niestabilny czasowo dla obu reprezentacji co widać na wykresach. Trudno mi stwierdzić dlaczego tak jest.