Wrocław, 11.05.2014r.

Damian Golonka, 200649

Mateusz Karbowiak, 200684

Prowadzący: mgr inż. Agata Rusoń

**Struktury danych i złożoność obliczeniowa**

**Projekt 2** – Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji

oraz sposobu reprezentacji grafu w pamięci komputera.

1. **Wstęp**

Celem projektu było zbadanie efektywności algorytmów grafowych w zależności od instancji oraz sposobu reprezentacji grafu.

Do tego celu użyliśmy algorytmów:

* Algorytm Prima, Kruskala – wyznaczanie minimalnego drzewa rozpinającego
* Algorytm Dijkstry, Forda-Bellmana – wyznaczanie najkrótszej ścieżki

Algorytmy te zaimplementowaliśmy jako:

* Reprezentacja macierzowa
* Reprezentacja listowa

Czasy działania algorytmów:

* Algorytm Prima: , przy użyciu kopca Fibonacciego: 
* Algorytm Kruskala: 
* Algorytm Dijkstry: 
* Algorytm Bellmana-Forda: 

Czasy działania algorytmów zostały opracowane na podstawie książki:   
'Wprowadzenie do algorytmów'[1].

1. **Plan eksperymentu**

Wielkość struktur ustalana jest dynamicznie. Przy losowaniu grafu musieliśmy pamiętać o jego spójności.

W algorytmie losowania, graf generujemy na podstawie wybranej liczby wierzchołków. Umieszczamy je w tablicy i łączymy ze sobą. Krawędzie łączące odpowiednie wierzchołki posiadają swoją wagę, które losujemy podczas przydzielania wierzchołków. Wygenerowany graf wpisujemy do macierzy lub listy.

Do mierzenia czasów operacji użyliśmy funkcji, którą w swoim dokumencie zamieścił   
dr inż. Jarosław Mierzwa[2]. Została zmodyfikowana, aby wyświetlać czas w sekundach.

Do każdej operacji utworzyliśmy pętlę, która wykonywała się dziesięć razy, uśredniając czasy. To pozwoliło nam uzyskać wykresy ukazujące złożoność naszych struktur. Czasy liczyliśmy dla 400, 1000, 1600, 2200 oraz 2800 liczby wierzchołków przy gęstości 30%, 50%, 75% i 90%.

Graf w postaci listy zapisywany jest jako wektor par. Dochodzenie do danego wierzchołka, musimy przechodzić całą listę.

Graf w postaci macierzy zapisywaliśmy jako tablica dwuwymiarowa. Wypełniona jest elementami typu short. Po nieudanych próbach implementacji macierzy incydencji w naszym projekcie, zastosowaliśmy macierz sąsiedztwa, która jak się później okazało prawdopodobnie jest lepsza. W macierzy incydencji koszt znalezienia sąsiadów danego wierzchołka jest największy.

**Specyfikacja laptopa, na którym wykonywane były testy:**

Procesor: Intel i5-3210 @ 2,50GHz

Pamięć RAM: 8GB

System: Windows 7 64-bit

Program potrzebny do badania podstawowych struktur danych został napisany w języku C++ w programie Visual Studio 2013.

1. **Minimalne drzewo rozpinające (MST)**

**3.1 Algorytm Prima**

Algorytm Prima należy do algorytmów zachłannych, w każdym kroku dokonują zachłannego wyboru

rozwiązania. Dla tego algorytmu podajemy spójny i nie skierowany graf.

Działanie algorytmu:

* Wybór wierzchołka i dodanie do grafu
* Umieszczenie na liście krawędzi
* Zdjęcie krawędzi o najmniejszej wadze
* Sprawdzamy czy drugi wierzchołek należy do drzewa

**3.2 Algorytm Kruskala**

Algorytm Kruskala również jest algorytmem zachłannym.

Działanie algorytmu:

* Utworzenie „lasu” z wierzchołków grafu
* Utworzenie zbioru krawędzi
* Łączenie poddrzew przy pomocy najniższych gałęzi
  + Wybór krawędzi
  + Sprawdzamy czy krawędź należy do dwóch różnych drzew
    - Tak – łączymy drzewa
    - Nie - odrzucamy

**Reprezentacja macierzowa**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Wierz.** | **Kruskal 0,3** | **Kruskal 0,5** | **Kruskal 0,75** | **Kruskal 0,9** | **Prim 0,3** | **Prim 0,5** | **Prim 0,75** | **Prim 0,9** |
| **400** | 0,00431 | 0,0071 | 0,01016 | 0,01229 | 0,00218 | 0,00552 | 0,01293 | 0,01056 |
| **1000** | 0,0289963 | 0,0468591 | 0,0696702 | 0,0824803 | 0,012249 | 0,015977 | 0,011962 | 0,041842 |
| **1600** | 0,079756 | 0,138993 | 0,194921 | 0,227654 | 0,025729 | 0,031887 | 0,02505 | 0,079603 |
| **2200** | 0,155551 | 0,250917 | 0,375621 | 0,473629 | 0,051374 | 0,062973 | 0,057713 | 0,119661 |
| **2800** | 0,254611 | 0,426552 | 0,641799 | 0,861797 | 0,08578 | 0,115644 | 0,097533 | 0,184483 |

**Reprezentacja listowa**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Wierz.** | **Kruskal 0,3** | **Kruskal 0,5** | **Kruskal 0,75** | **Kruskal 0,9** | **Prim 0,3** | **Prim 0,5** | **Prim 0,75** | **Prim 0,9** |
| **400** | 0,01725 | 0,029 | 0,04275 | 0,0535 | 0,00317 | 0,00483 | 0,00725 | 0,008 |
| **1000** | 0,280281 | 0,492875 | 0,738344 | 0,913063 | 0,039467 | 0,053272 | 0,068431 | 0,071628 |
| **1600** | 1,29216 | 2,19161 | 3,42804 | 4,18751 | 0,113201 | 0,157847 | 0,226749 | 0,234413 |
| **2200** | 3,6849 | 6,3197 | 9,80063 | 12,0553 | 0,274118 | 0,380502 | 0,576085 | 0,595442 |
| **2800** | 8,12224 | 14,182 | 21,9526 | 26,6374 | 0,47699 | 0,804793 | 1,15394 | 1,241283 |

**3.3 Wnioski**

Algorytm Prima i Kruskala służą do tego samego celu – wyznaczanie MST. Algorytm Prima dla wybranej gęstości grafu działa w podobnych czasach. Reprezentacja grafu w pamięci ma tutaj duże znaczenie w szybkości wykonanie algorytmu. W przypadku reprezentacji listowej czasy wykonania były dużo większe niż w przypadku reprezentacji macierzowej. Czas wykonania algorytmu wzrasta z zwiększeniem ilości wierzchołków. Ma to związek z ilością krawędzi które trzeba porównać.

W algorytmie Kruskala również efektywniejsza okazała się reprezentacja macierzowa grafu. Jednak algorytm ten jest znacznie wolniejszy. Jest to sprzeczne z teorią, ponieważ algorytm Prima i Kruskala powinny mieć taką samą złożoność obliczeniową. W naszej implementacji zastosowaliśmy kopiowanie macierzy do pomocniczego wektora par i prawdopodobnie to przyczyniło się do wydłużenia czasu działania algorytmu Kruskala.

1. **Wyznaczanie najkrótszej ścieżki w grafie**

**4.1 Algorytm Dijkstry**

Algorytm ten należy do grupy algorytmów zachłannych. Warunkiem działania algorytmu jest podanie mu grafu który nie posiada krawędzi o ujemnej wadze.

Działanie algorytmu:

* Utworzenie tablicy D odległości od źródła dla wierzchołków grafu
* Utworzenie zbioru wszystkich wierzchołków Q
* Pobranie ze zbioru, wierzchołka o największej odległości od wybranego wierzchołka s, którego odległość od źródła wynosi 0
* Usunięcie wierzchołka najbliższego źródła
* Dla sąsiada wierzchołka v dokonaj relaksacji poprzez u

**4.2 Algorytm Bellmana-Forda**

Algorytm również służy do rozwiązywania problemów w wyznaczaniu najkrótszej ścieżki w grafie. Ma on jednak przewagę nad Dijkstry, ponieważ nie musi operować na grafach, które nie posiadają krawędzi o wadze ujemnej. Pamiętać trzeba jedynie o tym, że graf nie może posiadać cykli o ujemnej długości. Algorytm ten posługuje się metodą relaksacji.

Działanie algorytmu:

* Macierz A dla każdej pary wierzchołków zawiera wagę krawędzi (u,v)
* Jeśli krawędź nie istnieje ti przyjmujemy, że jej waga wynosi 
* W każdym kroku obliczamy górne oszacowanie odległości s do wszystkich pozostałych wierzchołków.
* Po stwierdzeniu, że D(v)>D(u)+A(u,v) -> polepszamy aktualne oszacowanie
* Algorytm kończymy, gdy żadnego oszacowania nie można już poprawić

**Reprezentacja macierzowa**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Wierz.** | **Dijkstry 0,3** | **Dijkstry 0,5** | **Dijkstry 0,75** | **Dijkstry 0,9** | **Bellman 0,3** | **Bellman 0,5** |
| **400** | 0,00375 | 0,007 | 0,013 | 0,01825 | 3,72675 | 5,14175 |
| **1000** | 0,0135938 | 0,020625 | 0,0247813 | 0,027 | 28,848 | 43,3441 |
| **1600** | 0,0399492 | 0,0407031 | 0,0474727 | 0,055625 | 76,4745 | 128,3 |
| **2200** | 0,0898379 | 0,0906187 | 0,103309 | 0,115078 | 160,803 | 267,73 |
| **2800** | 0,148605 | 0,157452 | 0,170539 | 0,195135 | 290,159 | 485,043 |

**Reprezentacja listowa**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Wierz.** | **Dijkstry 0,3** | **Dijkstry 0,5** | **Dijkstry 0,75** | **Dijkstry 0,9** | **Bellman 0,3** | **Bellman 0,5** |
| **400** | 0,00175 | 0,0035 | 0,00325 | 0,00525 | 45,7578 | 77,4789 |
| **1000** | 0,0142188 | 0,0208125 | 0,0306562 | 0,0352813 | 98,158 | 153,7756 |
| **1600** | 0,0511523 | 0,0851016 | 0,118457 | 0,13816 | 156,72645 | 278,4486 |
| **2200** | 0,128519 | 0,208763 | 0,305182 | 0,368395 | 278,4687 | 467,3369 |
| **2800** | 0,26494 | 0,433595 | 0,619398 | 0,744299 | 490,587 | 685,4056 |

**4.3 Wnioski**

Wykresy dla algorytmu Dijkstry i Forda-Bellmana zawarliśmy osobno, ponieważ jak widać czasy dla algorytmu Bellmana są bardzo duże w porównaniu do algorytmu Dijkstry. Gęstość grafu ma wpływ na algorytm w przypadku dużej liczby krawędzi. Rozkłady dla algorytmów wyznaczania najkrótszej ścieżki w grafie są bardzo podobnego do algorytmów MST. Nasza implementacja algorytmu Bellmana okazała się bardzo kiepska. Czasy są bardzo duże. Dla gęstości powyżej 50% nasza implementacja algorytmu Bellmana nie dała sobie rady. Czas wykonywania był bardzo duży a po 20 minutach pokazywał się błąd pamięci.

1. **Wnioski**

**Osobne wykresy dla każdej gęstości grafu:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Kruskal 0,3** | **Prim 0,3** | **Kruskal 0,3** | **Prim 0,3** |
| 400 | 0,00431 | 0,00218 | 0,01725 | 0,00317 |
| 1000 | 0,0289963 | 0,0122492 | 0,280281 | 0,0394672 |
| 1600 | 0,079756 | 0,0257292 | 1,29216 | 0,113201 |
| 2200 | 0,155551 | 0,0513737 | 3,6849 | 0,274118 |
| 2800 | 0,254611 | 0,0857804 | 8,12224 | 0,47699 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Kruskal 0,5** | **Prim 0,5** | **Kruskal 0,5** | **Prim 0,5** |
| 400 | 0,0071 | 0,00552 | 0,029 | 0,00483 |
| 1000 | 0,0468591 | 0,0159765 | 0,492875 | 0,0532718 |
| 1600 | 0,138993 | 0,031887 | 2,19161 | 0,157847 |
| 2200 | 0,250917 | 0,0629729 | 6,3197 | 0,380502 |
| 2800 | 0,426552 | 0,115644 | 14,182 | 0,804793 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Kruskal 0,75** | **Prim 0,75** | **Kruskal 0,75** | **Prim 0,75** |
| 400 | 0,01016 | 0,01293 | 0,04275 | 0,00725 |
| 1000 | 0,0696702 | 0,0119619 | 0,738344 | 0,0684312 |
| 1600 | 0,194921 | 0,02505 | 3,42804 | 0,226749 |
| 2200 | 0,375621 | 0,057713 | 9,80063 | 0,576085 |
| 2800 | 0,641799 | 0,0975327 | 21,9526 | 1,15394 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Kruskal 0,9** | **Prim 0,9** | **Kruskal 0,9** | **Prim 0,9** |
| 400 | 0,01229 | 0,01056 | 0,0535 | 0,008 |
| 1000 | 0,0824803 | 0,0418416 | 0,913063 | 0,071628 |
| 1600 | 0,227654 | 0,0796028 | 4,18751 | 0,234413 |
| 2200 | 0,473629 | 0,1196606 | 12,0553 | 0,595442 |
| 2800 | 0,861797 | 0,1844827 | 26,6374 | 1,241283 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Dijkstry 0,3** | **Bellman 0,3** | **Dijkstry 0,3** | **Bellman 0,3** |
| 400 | 0,00375 | 3,72675 | 0,00175 | 45,7578 |
| 1000 | 0,0135938 | 28,848 | 0,0142188 | 98,158 |
| 1600 | 0,0399492 | 76,4745 | 0,0511523 | 156,72645 |
| 2200 | 0,0898379 | 160,803 | 0,128519 | 278,4687 |
| 2800 | 0,148605 | 290,159 | 0,26494 | 490,587 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Dijkstry 0,5** | **Bellman 0,5** | **Dijkstry 0,5** | **Bellman 0,5** |
| 400 | 0,007 | 5,14175 | 0,0035 | 77,4789 |
| 1000 | 0,020625 | 43,3441 | 0,0208125 | 153,7756 |
| 1600 | 0,0407031 | 128,3 | 0,0851016 | 278,4486 |
| 2200 | 0,0906187 | 267,73 | 0,208763 | 467,3369 |
| 2800 | 0,157452 | 485,043 | 0,433595 | 685,4056 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Dijkstry 0,75** |  | **Dijkstry 0,75** |  |
| 400 | 0,013 |  | 0,00325 |  |
| 1000 | 0,0247813 |  | 0,0306562 |  |
| 1600 | 0,0474727 |  | 0,118457 |  |
| 2200 | 0,103309 |  | 0,305182 |  |
| 2800 | 0,170539 |  | 0,619398 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Macierz | | Lista | |
|  | **Dijkstry 0,9** |  | **Dijkstry 0,9** |  |
| 400 | 0,01825 |  | 0,00525 |  |
| 1000 | 0,027 |  | 0,0352813 |  |
| 1600 | 0,055625 |  | 0,13816 |  |
| 2200 | 0,115078 |  | 0,368395 |  |
| 2800 | 0,195135 |  | 0,744299 |  |

Jak widzimy na powyższych wykresach algorytm Kruskala wykonuje się najdłużej w reprezentacji macierzowej jak i w reprezentacji listowej. Najszybciej wykonuje się algorytm Prima, czyli algorytm MST.

Algorytm Dijkstry przy zwiększaniu gęstości odczuwa różnice powyżej 75%. Różnica ta jest znacząca jeżeli patrzymy na reprezentacje listową. Algorytm Forda-Bellmana po naszej implementacji wykonuje się najdłużej, co z jednej strony jest przewidujące, ponieważ jest to algorytm ogólny tzn. algorytm ten nie opera się na założeniu że wagi są nieujemne. Algorytm ten nie rozwiązuje jednak problemu cyklu nieujemnego. Stwierdza jedynie fakt, że taki cykl wystąpił i dalej nie może działać.

Przy zadaniu projektowym napotkaliśmy dużo kłopotów z generowanie grafu losowego. Trudne okazało się zachowanie spójność grafu. Jako, że zastosowaliśmy macierz sąsiedztwa, zyskaliśmy miejsce w pamięci, ponieważ macierz ta jest symetryczna dla grafów nie skierowanych. Powoduje to, że możemy przechowywać tylko połowę.

1. **Bibliografia**

[1] Thoma H. Cormen. *Wprowadzenie do algorytmów. Wydanie czwarte.*

Warszawa: WNT, 2001. ISBN 83-204-2665-0.

[2] http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pamsi/debug\_and\_time.pdf