|  |  |  |  |
| --- | --- | --- | --- |
|  | **Nazwa kursu:** Struktury Danych i Złożoność Obliczeniowa (projekt nr 2.) | | |
| **Temat projektu:** „ Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych.” | | |
| **Autor:**  Piotr Ławniczak 209775 | **Data oddania:** 12.05.2015 | |
| **Grupa:** wtorek 11:15 | |
| **Prowadzący:**  Dr inż. Jarosław Mierzwa | **Ocena:** |

# Cel projektu

Celem projektu jest analiza wydajności algorytmów wyznaczających w grafie: minimalne drzewo rozpinające (algorytm Prima) oraz najkrótszą ścieżkę (algorytm Dijkstry). Polegać ona będzie na badaniach wykazujących zależność ich szybkości wykonania w zależności od rozmiaru, gęstości grafu, a także od rodzaju jego reprezentacji w pamięci komputera.

# Wstęp teoretyczny

Minimalnym drzewem rozpinającym (w skrócie ang. MST) nazywa się drzewo rozpinające danego grafu o najmniejszej z możliwych wag, tj. takie, że nie istnieje dla tego grafu inne drzewo rozpinające o mniejszej sumie wag krawędzi. Sposobów jego wyznaczania jest kilka. Jednym z nich jest algorytm Prima.

Algorytm Prima należy do grupy algorytmów zachłannych. Dysponując grafem spójnym i nieskierowanym, obliczany jest podzbiór ze zbioru krawędzi, dla którego graf nadal pozostaje spójny, jednak suma wag wszystkich takich krawędzi w owym podzbiorze jest najmniejsza z możliwych.

Drugi problem, jaki będzie rozwiązywany przy pomocy algorytmu to problem znajdowania najkrótszych ścieżek w grafie. Do jego rozwiązania służy jeden z popularniejszych algorytmów – algorytm Dijkstry.

Algorytm Dijkstry został opracowany w 1959 roku przez holenderskiego naukowca Edsgera Dijkstrę. Bardzo ważnym warunkiem koniecznym do poprawnego działania tego algorytmu jest to, by wszystkie krawędzie grafu posiadały nieujemne wagi. Mając dany wierzchołek początkowy, metoda znajduje najkrótszą drogę dojścia do wszystkich wierzchołków, wyliczając przy okazji koszt dojścia dla każdej z tych ścieżek. Jest to przykład algorytmu zachłannego.

# Reprezentacja grafów w pamięci komputera

Na potrzeby projektu zostały zaimplementowane dwa sposoby przechowywania grafu:

1. Macierz sąsiedztwa

Sposób ten zakłada stworzenie dwuwymiarowej tablicy liczb całkowitych nieujemnych. Każdy jej element inicjalizowany jest wartością „0”, co oznacza, że między wierzchołkiem reprezentowanym przez numer wiersza a wierzchołkiem reprezentowanym przez numer kolumny nie występuje krawędź. W przypadku kiedy w danej komórce znajduje się liczba różna od zera – oznacza to występowanie krawędzi. Liczba ta jednocześnie jest jej wagą. Wymagania pamięciowe to O(V2). Operacje przejrzenia wszystkich krawędzi, sąsiadów danego wierzchołka i sprawdzenie, czy dana krawędź istnieje wykonują się w czasie O(V).

1. Lista sąsiedztwa

Reprezentacja listowa opiera się na utworzeniu w pierwszej kolejności tablicy zawierającej tyle list ile graf ma wierzchołków. Indeks tablicy odpowiada numerowi wierzchołka, a każdy element listy znajdującej się pod nim – wskazuje na jego sąsiadów (z którymi tworzy krawędź). Sąsiedzi przechowywani są w specjalnie stworzonej do tego celu strukturze, która oprócz numeru sąsiada zawiera kompletne informacje o krawędzi (wierzchołek początkowy, wierzchołek końcowy, waga). Złożoność pamięciowa wynosi O(E). Przejrzenie wszystkich krawędzi ma złożoność czasową O(E), natomiast przeglądanie sąsiadów wierzchołka i sprawdzanie istnienia krawędzi wykonuje się w czasie O(E).

# Implementacja algorytmów

1. Algorytm Prima – wyznaczanie minimalnego drzewa rozpinającego (MST)

Na początku algorytm dodaje do zbioru A reprezentującego drzewo krawędź o najmniejszej wadze, łączącą wierzchołek początkowy v z dowolnym wierzchołkiem. W każdym kolejnym kroku procedura dodaje do A najlżejszą krawędź wśród krawędzi łączących wierzchołki już odwiedzone z nieodwiedzonymi. Jeśli struktura A jest kolejką priorytetową opartą na kopcu binarnym, czasowa złożoność obliczeniowa operacji wynosi O(m \* log2n).

1. Algorytm Dijkstry – znajdowanie najkrótszych ścieżek w grafie

Algorytm Dijkstry znajduje w grafie najkrótsze ścieżki pomiędzy wybranym wierzchołkiem początkowym a wszystkimi pozostałymi, wyliczając również koszt przejścia każdej z tych ścieżek, czyli sumę wag krawędzi na ścieżce. Algorytm ma złożoność czasową O(n2) przy wykorzystaniu wyszukiwania liniowego podczas szukania wierzchołków o najmniejszym koszcie dojścia. Czas ten może być zmniejszony dzięki wykorzystaniu kolejki priorytetowej opartej na kopcu binarnym. Wtedy w korzeniu przechowywany jest wierzchołek o najmniejszej wartości kosztu dojścia. Złożoność czasowa upraszcza się do O(n \* log2n) – tyle, ile wynosi czas przywracania własności kopca.

# Plan eksperymentu

* Czas potrzebny do wykonania danej operacji będzie mierzony przy użyciu licznika cykli procesora wykonanych od jego uruchomienia - TSC (Time Stamp Counter). Dostęp do niego umożliwia funkcja QueryPerformanceCounter. Jego wartość będzie odczytywana i zapisywana tuż przed rozpoczęciem mierzonego polecenia, oraz zaraz po. Różnica podzielona przez ilość cykli jakie procesor wykonuje w czasie jednej sekundy (odczytaną przy użyciu funkcji QueryPerformanceFrequency) daje szczegółowy czas w sekundach do sześciu miejsc po przecinku.
* Pomiary wykonane będą dla grafów o rozmiarach 50, 75, 100, 125 i 150 wierzchołków. Dla każdego z rozmiarów rozpatrywane będą gęstości 25%, 50%, 75% i 99%. Aby pomiary były jak najbardziej wiarygodne, każdy przypadek będzie wykonywany po 100 razy, a wyniki uśredniane.
* W przypadku algorytmu Prima krawędzie są traktowane jako nieskierowane, a w przypadku algorytmu Dijkstry – skierowane.
* Krawędzie wielokrotne oraz pętle nie występują w grafach generowanych przez program. Nie są także dozwolone w przypadku grafów wczytywanych.

# Wyniki eksperymentu

1. Algorytm Dijkstry
2. Macierz sąsiedztwa

|  |  |  |
| --- | --- | --- |
| **Dijkstra - Macierz sąsiedztwa** | | |
| **Liczba wierzchołków** | **Gęstość [%]** | **Czas [ms]** |
| 50 | 99 | 0,033 |
| 75 | 99 | 0,061 |
| 100 | 99 | 0,101 |
| 125 | 99 | 0,166 |
| 150 | 99 | 0,229 |
| 50 | 75 | 0,036 |
| 75 | 75 | 0,065 |
| 100 | 75 | 0,102 |
| 125 | 75 | 0,152 |
| 150 | 75 | 0,216 |
| 50 | 50 | 0,039 |
| 75 | 50 | 0,070 |
| 100 | 50 | 0,111 |
| 125 | 50 | 0,170 |
| 150 | 50 | 0,230 |
| 50 | 25 | 0,032 |
| 75 | 25 | 0,063 |
| 100 | 25 | 0,101 |
| 125 | 25 | 0,149 |
| 150 | 25 | 0,202 |

Tabela 1. Uśrednione (po 100 próbach) czasy wykonania algorytmu Dijkstry na reprezentacji macierzowej

1. Wykres zależności czasu wykonywania algorytmu Dijkstry od wielkości oraz gęstości grafu

1. Lista sąsiedztwa

|  |  |  |
| --- | --- | --- |
| **Dijkstra - Lista sąsiedztwa** | | |
| **Liczba wierzchołków** | **Gęstość [%]** | **Czas [ms]** |
| 50 | 99 | 4,032 |
| 75 | 99 | 8,052 |
| 100 | 99 | 18,033 |
| 125 | 99 | 27,039 |
| 150 | 99 | 83,535 |
| 50 | 75 | 2,761 |
| 75 | 75 | 5,591 |
| 100 | 75 | 9,532 |
| 125 | 75 | 17,993 |
| 150 | 75 | 36,009 |
| 50 | 50 | 1,669 |
| 75 | 50 | 4,543 |
| 100 | 50 | 6,893 |
| 125 | 50 | 12,950 |
| 150 | 50 | 20,227 |
| 50 | 25 | 1,038 |
| 75 | 25 | 2,000 |
| 100 | 25 | 5,147 |
| 125 | 25 | 6,346 |
| 150 | 25 | 10,492 |

Tabela 2. Uśrednione (po 100 próbach) czasy wykonania algorytmu Dijkstry na reprezentacji listowej

2. Wykres zależności czasu wykonywania algorytmu Dijkstry od wielkości oraz gęstości grafu

3. Wykres porównujący szybkości działania dwóch różnych reprezentacji

4. Wykres porównujący szybkości działania dwóch różnych reprezentacji

5. Wykres porównujący obie reprezentacje grafowe dla algorytmu Dijkstry

6. Wykres porównujący obie reprezentacje grafowe dla algorytmu Dijkstry

1. Algorytm Prima
2. Macierz sąsiedztwa

|  |  |  |
| --- | --- | --- |
| **Prim - Macierz sąsiedztwa** | | |
| **Liczba wierzchołków** | **Gęstość [%]** | **Czas [ms]** |
| 50 | 99 | 76,010 |
| 75 | 99 | 196,989 |
| 100 | 99 | 401,148 |
| 125 | 99 | 721,371 |
| 150 | 99 | 1154,419 |
| 50 | 75 | 51,349 |
| 75 | 75 | 130,549 |
| 100 | 75 | 264,935 |
| 125 | 75 | 461,397 |
| 150 | 75 | 745,040 |
| 50 | 50 | 31,726 |
| 75 | 50 | 78,218 |
| 100 | 50 | 152,942 |
| 125 | 50 | 255,062 |
| 150 | 50 | 399,387 |
| 50 | 25 | 14,621 |
| 75 | 25 | 35,551 |
| 100 | 25 | 64,800 |
| 125 | 25 | 104,471 |
| 150 | 25 | 158,689 |

Tabela 3. Uśrednione czasy wykonywania algorytmu Prima na reprezentacji macierzowej grafu

7. Wykres zależności czasu wykonywania algorytmu Prima na macierzy sąsiedztwa w zależności od rozmiaru i gęstości grafu

1. Lista sąsiedztwa

|  |  |  |
| --- | --- | --- |
| **Prim - Lista sąsiedztwa** | | |
| **Liczba wierzchołków** | **Gęstość [%]** | **Czas [ms]** |
| 50 | 99 | 343,599 |
| 75 | 99 | 1154,897 |
| 100 | 99 | 2682,173 |
| 125 | 99 | 5053,283 |
| 150 | 99 | 8640,671 |
| 50 | 75 | 252,270 |
| 75 | 75 | 843,873 |
| 100 | 75 | 2046,533 |
| 125 | 75 | 3811,657 |
| 150 | 75 | 6403,532 |
| 50 | 50 | 163,144 |
| 75 | 50 | 539,531 |
| 100 | 50 | 1253,839 |
| 125 | 50 | 2500,432 |
| 150 | 50 | 4303,310 |
| 50 | 25 | 84,235 |
| 75 | 25 | 268,066 |
| 100 | 25 | 612,451 |
| 125 | 25 | 1177,363 |
| 150 | 25 | 2060,854 |

Tabela 4. Uśrednione czasy wykonywania algorytmu Prima na reprezentacji listowej

8. Wykres zależności czasu wykonania algorytmu Prima w zależności od wielkości i gęstości grafu

9. Wykres porównujący szybkość działania algorytmu Prima w zależności od rodzaju jego reprezentacji

10. Wykres porównujący szybkość działania algorytmu Prima w zależności od rodzaju jego reprezentacji

11. Wykres porównujący szybkość działania algorytmu Prima w zależności od rodzaju jego reprezentacji

12. Wykres porównujący szybkość działania algorytmu Prima w zależności od rodzaju jego reprezentacji

# Wnioski

Na podstawie uzyskanych wyników nasuwają się jednoznaczne wnioski:

* Złożoności obliczeniowe zaimplementowanych przeze mnie algorytmów w większości nie zgadzają się z pierwotnymi założeniami na podstawie danych znalezionych w literaturze. Jest to spowodowane brakiem właściwej optymalizacji algorytmów, zbędne wielokrotne przeszukania list i macierzy, które dałoby się zminimalizować.
* Różnice w czasie wykonywania algorytmów pomiędzy reprezentacjami listowymi a macierzowymi zwłaszcza w przypadku dużych grafów – są bardzo duże. Przemawiają one na korzyść reprezentacji macierzowej, co jest sprzeczne z pierwotnymi założeniami.

# Bibliografia

[1] Robert L. Kruse – „Data Structures and Program Design in C++”

[2] Piotr Wróblewski – „Struktury Danych i Techniki Programowania – Algorytmy”

[3] Thomas H. Cormen – „Wprowadzenie do Algorytmów”

[4] Big-O Cheat Sheet (wersja angielska) - <http://bigocheatsheet.com/>

[5] Algorytm Prima – Wikipedia, wolna encyklopedia - <http://pl.wikipedia.org/wiki/Algorytm_Prima>

[6] Algorytm Dijkstry – Wikipedia, wolna encyklopedia - <http://pl.wikipedia.org/wiki/Algorytm_Dijkstry>