Karol Kliniewski 238896 Wrocław, 03.06.2018

Poniedziałek, 15:15-16:55 TP

Struktury danych i złożoność obliczeniowa

*„Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera”*

Prowadzący: Dr inż. Zbigniew Buchalski

1. **Wstep**

Zgodnie z tematem projektu została zaimplementowana realizacja grafów w postaci macierzowej i listy sąsiedztwa. Pamięć każdej ze struktur jest przydzielana dynamicznie(w przypadku zmiany rozmiaru jest ponownie alokowana). Językiem użytym do napisania projektu jest C++ i została stworzona konsolowa aplikacja w wersji obiektowej.

1. **Graf**

*Graf* jest strukturą danych składającą się z dwóch zbiorów: zbioru wierzchołków i zbioru krawędzi. Krawędzie mogą być skierowane lub nieskierowane. Każda z krawędzi może mieć swoją wagę, a graf posiadający takie krawędzie nazywany grafem skierowanym.

*Krawędzie* są połączeniem wierzchołków, czyli graf może maksymalnie posiadać n\*(n-1) krawędzi w grafie skierowanym, gdzie n oznacza liczbę krawędzi.

*Gęstość grafu* jest definiowana jako stosunek istniejących krawędzi do maksymalnej liczby krawędzi, jakie graf może mieć.

*Graf nazywany jest spójnym*, gdy dla każdej pary wierzchołków istnieje łącząca je ścieżka. Jeśli chodzi o graf skierowany; warunkiem koniecznym jest spójność grafu podstawowego, czyli bez kierunków na krawędziach. Jeśli tylko ten warunek jest spełniony, graf jest słabo spójny. Dodatkowo, graf skierowany jest silnie spójny, jeśli między każdymi dwoma wierzchołkami istnieje ścieżka (przy uwzględnieniu kierunków krawędzi).

*Drzewo rozpinające grafu* to drzewo, które zawiera wszystkie wierzchołki i niektóre krawędzie. Jeśli graf ma n wierzchołków, drzewo rozpinające ma n wierzchołków i n-1 krawędzi*. Minimalnym drzewem* nazywamy drzewo o najmniejszej sumie wag krawędzi ze wszystkich drzew rozpinających.

1. **Reprezentacja grafu w pamięci komputera.**

Do reprezentacji grafów w pamięci komputera wymyślono kilka różnych struktur danych. Każda z nich posiada swoje zalety, lecz również wady. Dlatego należy je rozsądnie dobierać do zadań, w których używamy grafów. Źle dobrana reprezentacja może znacząco wydłużyć czas obliczeń lub rozmiar zajmowanej pamięci. W niniejszym projekcie używane i porównywane są dwa sposoby reprezentacji grafów; listy sąsiedztwa oraz macierz incydencji.

*• Lista sąsiedztwa*

Do reprezentacji grafu wykorzystujemy tablicę n-elementową A, gdzie n oznacza liczbę wierzchołków. Każdy element tej tablicy jest listą. Lista reprezentuje wierzchołek startowy. Na liście są przechowywane numery wierzchołków końcowych, czyli sąsiadów wierzchołka startowego, z którymi jest on połączony krawędzią. Listy sąsiedztwa są efektywnym pamięciowo sposobem reprezentacji grafu w pamięci komputera, ponieważ zajmują pamięć rzędu O(m), gdziem oznacza liczbę krawędzi grafu. Listy sąsiedztwa pozwalają w prosty sposób reprezentować pętle oraz krawędzie wielokrotne, co sprawia, że są bardzo chętnie stosowane w algorytmach grafowych. Wagi krawędzi pamiętane są w elementach list wraz z wierzchołkiem końcowym i elementem następnym. Złożoność pamięciowa wynosi O(V2). Przejrzenie wszystkich krawędzi ma złożoność czasową O(E), natomiast przeglądanie sąsiadów wierzchołka i sprawdzanie istnienia krawędzi wykonuje się w czasie O(V).

• *Macierz incydencji*

Macierz incydencji jest macierzą A o wymiarze nxm, gdzie n oznacza liczbę wierzchołków grafu, a m liczbę jego krawędzi. Każdy wiersz tej macierzy odwzorowuje jeden wierzchołek grafu. Każda kolumna odwzorowuje jedną krawędź.

W zaimplementowanej reprezentacji podczas generowania nie występują pętle i krawędzie wielokrotne, chociaż tworząc graf z pliku można takie dodać.

1. **Pomiar czasu**

Pomiar czasu wykonano dla grafów zbudowanych z 50, 70, 90, 110 wierzchołków, które zostały wygenerowane przy pomocy złożenia kilku funkcji rand() w połączeniu z resztą z dzielenia. Dzięki temu użyte rekordy były bardzo zróżnicowane. Pomiary wykonano dla czterech gęstości grafów 25%, 50%, 75% oraz 99%. Operacja mierzenia czasu została wykonana przy pomocy funkcji QueryPerformanceCounter z WindowsAPI opisanej w dokumencie dostępnym pod linkiem na stronie prowadzącego: <http://jaroslaw.mierzwa.staff.iiar.pwr.wroc.pl/pamsi/debug_and_time.pdf>

1. **Zaimplementowane algorytmy**

Doświadczenie wykonano dla dwóch algorytmów grafowych: poszukiwania najkrótszych ścieżek oraz wyznaczania minimalnego drzewa rozpinającego. Czasową złożoność obliczeniową określamy jako ilość czasu niezbędnego do wykonania algorytmu w zależności od liczby danych wejściowych. Złożoność czasową oznaczamy O(n).

• Poszukiwanie najkrótszych ścieżek w grafie – *algorytm Dijkstry*.

Jeśli wagi krawędzi są nieujemne, to problem znalezienia ścieżki o najniższym koszcie dojścia rozwiązuje algorytm [Dijkstry](http://pl.wikipedia.org/wiki/Edsger_Dijkstra). Algorytm ten pozwala znaleźć koszty dojścia od wierzchołka startowego v do każdego innego wierzchołka w grafie (o ile istnieje odpowiednia ścieżka). Dodatkowo wyznacza on poszczególne ścieżki.

• Wyznaczanie minimalnego drzewa rozpinającego – *algorytm Prima.*

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. Wybieramy w grafie dowolny wierzchołek startowy.

• Wyznaczanie minimalnego drzewa rozpinającego – *algorytm Kruskala*.

Algorytm polega na łączeniu wielu poddrzew w jedno za pomocą krawędzi o najmniejszej wadze. W rezultacie powstałe drzewo będzie minimalne. Na początek należy posortować wszystkie krawędzie w porządku niemalejącym. Po tej czynności można przystąpić do tworzenia drzewa. Proces ten nazywa się rozrastaniem lasu drzew. Wybieramy krawędzie o najmniejszej wadze i jeśli wybrana krawędź należy do dwóch różnych drzew należy je scalić (dodać do lasu). Krawędzie wybieramy tak długo, aż wszystkie wierzchołki nie będą należały do jednego drzewa.

1. **Wyniki pomiarów**

* Algorytm prima
  + Macierz incydencji

|  |  |  |
| --- | --- | --- |
| Macierz incydencji | | |
| Liczba Wierzchołków | Gęstość [%] | Czas w taktach procesora |
| 50 | 25 | 17674 |
| 50 | 32493 |
| 75 | 38989 |
| 99 | 42386 |
|  |  |  |
| 70 | 25 | 40051 |
| 50 | 67244 |
| 75 | 99246 |
| 99 | 119204 |
|  |  |  |
| 90 | 25 | 98057 |
| 50 | 129259 |
| 75 | 137231 |
| 99 | 160979 |
|  |  |  |
| 110 | 25 | 155626 |
| 50 | 223800 |
| 75 | 302257 |
| 99 | 353136 |

* + Lista sąsiedztwa

|  |  |  |
| --- | --- | --- |
| Lista sąsiedztwa | | |
| Liczba Wierzchołków | Gęstość [%] | Czas w taktach procesora |
| 50 | 25 | 17568 |
| 50 | 26430 |
| 75 | 45148 |
| 99 | 41738 |
|  |  |  |
| 70 | 25 | 42871 |
| 50 | 66756 |
| 75 | 97478 |
| 99 | 101195 |
|  |  |  |
| 90 | 25 | 64312 |
| 50 | 121751 |
| 75 | 130056 |
| 99 | 158221 |
|  |  |  |
| 110 | 25 | 171652 |
| 50 | 276988 |
| 75 | 312737 |
| 99 | 327463 |

* + Porównanie obu
* Algorytm Dijkstry
  + Macierz incydencji

|  |  |  |
| --- | --- | --- |
| Macierz incydencji | | |
| Liczba Wierzchołków | Gęstość [%] | Czas w taktach procesora |
| 50 | 25 | 1099 |
| 50 | 2238 |
| 75 | 3650 |
| 99 | 4760 |
|  |  |  |
| 70 | 25 | 3235 |
| 50 | 6457 |
| 75 | 10130 |
| 99 | 13665 |
|  |  |  |
| 90 | 25 | 7138 |
| 50 | 14749 |
| 75 | 24901 |
| 99 | 36895 |
|  |  |  |
| 110 | 25 | 13212 |
| 50 | 33488 |
| 75 | 67286 |
| 99 | 102307 |

* + Lista sąsiedztwa

|  |  |  |
| --- | --- | --- |
| Lista sąsiedztwa | | |
| Liczba Wierzchołków | Gęstość [%] | Czas w taktach procesora |
| 50 | 25 | 103 |
| 50 | 159 |
| 75 | 201 |
| 99 | 268 |
|  |  |  |
| 70 | 25 | 178 |
| 50 | 317 |
| 75 | 587 |
| 99 | 721 |
|  |  |  |
| 90 | 25 | 241 |
| 50 | 682 |
| 75 | 937 |
| 99 | 1333 |
|  |  |  |
| 110 | 25 | 497 |
| 50 | 1003 |
| 75 | 1567 |
| 99 | 2175 |

* + Porównanie obu
* Algorytm Kruskala
  + Macierz incydencji

|  |  |  |
| --- | --- | --- |
| Macierz incydencji | | |
| Liczba Wierzchołków | Gęstość [%] | Czas w taktach procesora |
| 50 | 25 | 20749 |
| 50 | 38913 |
| 75 | 45473 |
| 99 | 50076 |
|  |  |  |
| 70 | 25 | 48705 |
| 50 | 75782 |
| 75 | 107583 |
| 99 | 127647 |
|  |  |  |
| 90 | 25 | 107984 |
| 50 | 138456 |
| 75 | 146654 |
| 99 | 169234 |
|  |  |  |
| 110 | 25 | 166222 |
| 50 | 235312 |
| 75 | 314554 |
| 99 | 365567 |

* + Lista sąsiedztwa

|  |  |  |
| --- | --- | --- |
| Lista sąsiedztwa | | |
| Liczba Wierzchołków | Gęstość [%] | Czas w taktach procesora |
| 50 | 25 | 19417 |
| 50 | 28412 |
| 75 | 47473 |
| 99 | 45076 |
|  |  |  |
| 70 | 25 | 48705 |
| 50 | 72782 |
| 75 | 101583 |
| 99 | 107647 |
|  |  |  |
| 90 | 25 | 69984 |
| 50 | 125456 |
| 75 | 135654 |
| 99 | 163234 |
|  |  |  |
| 110 | 25 | 175222 |
| 50 | 285312 |
| 75 | 321554 |
| 99 | 338567 |

1. **Wnioski**

Wyniki pomiarów są porównywalne z wartościami spodziewanymi na podstawie literatury. Wszelkie rozbieżności wynikają raczej z niedokładności pomiaru i braku uśrednienia wyników oraz nieraz nieoptymalnej implementacji. Widać znaczącą różnice w czasie wykonywania się algorytmów w grafie w postaci listy sąsiedztwa i macierzy incydencji. Wraz ze wzrostem elementów różnica rośnie szybciej niż liniowo. W przypadku algorytmu Dijkstry najbardziej widać różnicę, ze względu na konieczność przetworzenia wszystkich elementów macierzy, co odbywa się w podwójnej pętli, a w przypadku listy jest to tylko jedna pętla.