Prowadząca zajęcia:  
dr hab. inż. Małgorzata Sterna, prof. nadzw.

**ALGORYTMY I STRUKTURY DANYCH**

**Ćwiczenie 3**

**Algorytmy grafowe**

Stanisław Jasiewicz  
nr 116753  
Wojciech Regulski  
nr 132312  
  
Informatyka(WI) I1

# Cel

Poznanie zalet oraz wad różnych reprezentacji grafu, implementacja sortowania topologicznego, zapoznanie się z algorytmem przeszukiwania w głąb DFS

# Pomiary i wykresy

## Czas obliczania etykiet

|  |  |  |
| --- | --- | --- |
| **Gęstość d [-]** | **Liczba wierzchołków n [-]** | **Czas obliczania etykiet [s]** |
| 0,2 | 1000 | 0,0010481 |
| 2000 | 0,0034200 |
| 3000 | 0,0071632 |
| 4000 | 0,0130583 |
| 5000 | 0,0192359 |
| 6000 | 0,0269290 |
| 7000 | 0,0362781 |
| 8000 | 0,0464392 |
| 9000 | 0,0588291 |
| 10000 | 0,0699782 |
| 0,4 | 1000 | 0,0016505 |
| 2000 | 0,0064391 |
| 3000 | 0,0135348 |
| 4000 | 0,0224814 |
| 5000 | 0,0363157 |
| 6000 | 0,0509476 |
| 7000 | 0,0689190 |
| 8000 | 0,0926160 |
| 9000 | 0,1165936 |
| 10000 | 0,1397960 |

## Liczba łuków powrotnych

|  |  |  |
| --- | --- | --- |
| **Gęstość d [-]** | **Liczba wierzchołków n [-]** | **Liczba łuków powrotnych [-]** |
| 0,2 | 1000 | 100013 |
| 2000 | 399817 |
| 3000 | 901077 |
| 4000 | 1598934 |
| 5000 | 2499088 |
| 6000 | 3600113 |
| 7000 | 4900051 |
| 8000 | 6399238 |
| 9000 | 8100334 |
| 10000 | 9998804 |
| 0,4 | 1000 | 199630 |
| 2000 | 799085 |
| 3000 | 1799098 |
| 4000 | 3198666 |
| 5000 | 4998579 |
| 6000 | 7197751 |
| 7000 | 9799246 |
| 8000 | 12803603 |
| 9000 | 16196647 |
| 10000 | 19998249 |

## Czas liczenia łuków powrotnych

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Gęstość d [-]** | **Liczba wierzchołków n [-]** | **Czas liczenia łuków powrotnych dla listy następników [s]** | **Czas liczenia łuków powrotnych dla listy łuków [s]** | **Czas liczenia łuków powrotnych dla macierzy sąsiedztwa[s]** |
| 0,2 | 1000 | 0,0009204 | 0,0012234 | 0,0054743 |
| 2000 | 0,0038016 | 0,0043428 | 0,0248095 |
| 3000 | 0,0075682 | 0,0114823 | 0,0475572 |
| 4000 | 0,0133349 | 0,0187042 | 0,0856159 |
| 5000 | 0,0277148 | 0,0309870 | 0,1300095 |
| 6000 | 0,0320190 | 0,0432310 | 0,1905528 |
| 7000 | 0,0440020 | 0,0564093 | 0,2628153 |
| 8000 | 0,0541030 | 0,0746728 | 0,3353503 |
| 9000 | 0,0671396 | 0,0946612 | 0,4326404 |
| 10000 | 0,0880456 | 0,1202503 | 0,5286951 |
| 0,4 | 1000 | 0,0016608 | 0,0025465 | 0,0087435 |
| 2000 | 0,0064328 | 0,0088968 | 0,0357203 |
| 3000 | 0,0159522 | 0,0209063 | 0,0804513 |
| 4000 | 0,0266663 | 0,0350140 | 0,1471797 |
| 5000 | 0,0426746 | 0,0615921 | 0,2252527 |
| 6000 | 0,0618735 | 0,0850202 | 0,3309031 |
| 7000 | 0,0818364 | 0,1263731 | 0,4373816 |
| 8000 | 0,1084640 | 0,1660275 | 0,6002066 |
| 9000 | 0,1349264 | 0,1816642 | 0,7281815 |
| 10000 | 0,1639880 | 0,2326016 | 0,8941303 |

# Wnioski

Program do pozyskania analizowanych danych został napisany w języku C#. Podprocedura obliczająca etykiety czasowe zaimplementowana została dla listy następników. Kryterium branym pod uwagę w wyborze struktury reprezentującej graf była szybkość. Ponieważ obliczanie etykiet czasowych działało w oparciu o wyszukiwanie zbioru następników, lista następników, dla której złożoność czasowa wyznaczania zbioru następników wynosi O(n), była najlepszym wyborem. Wyniki zaprezentowane w tabelach i na wykresach są średnimi z czterech środkowych pomiarów ze zbioru dziesięciu wygenerowanych pomiarów. Pozwoliło to nie brać pod uwagę zdarzających się z różnych powodów mocno skrajnych pomiarów.

Metoda sortowania topologicznego oparta jest na algorytmie przeszukiwania grafu w głąb (ang. depth-first search, w skrócie DFS). Algorytm polega na sprawdzaniu wszystkich wierzchołków grafu sąsiadujących z danym wierzchołkiem. W przypadku, gdy odwiedzone zostały już wszystkie z nich, algorytm cofa się do wierzchołka, z którego wyprowadzona została krawędź na obecny. Sortowanie topologiczne polega na dodatkowym odnotowaniu kolejności odwiedzenia oraz opuszczenia każdego z wierzchołków, co osiągane jest przez zapisywanie kolejności w etykietach wejściowej i wyjściowej każdego wierzchołka. Po uporządkowaniu wierzchołków w odwrotnej kolejności etykiet wyjściowych będą one posortowane topologicznie. Złożoność obliczeniowa sortowania topologicznego uzależniona jest od złożoności algorytmu DFS, który wynosi O(n+m), ponieważ algorytm musi zbadać wszystkie wierzchołki oraz wszystkie łuki z nich wychodzące. Wynika z tego, że korzystniejsza dla złożoności obliczeniowej jest mała gęstość, ponieważ większa gęstość wiąże się z większą liczbą łuków w grafie (co można stwierdzić na podstawie zaprezentowanych wyników). Najniższą złożoność obliczeniową można osiągnąć w przypadku, w którym liczba łuków równa będzie liczbie wierzchołków, wynosić ona będzie wtedy O(n). Należy powiedzieć, że złożoność obliczeniowa zależy również od wykorzystanej reprezentacji grafu. Jak zostało powiedziane, algorytm DFS opiera się na wyszukiwaniu następników, stąd lista następników pozwala osiągnąć możliwie najmniejszą złożoność obliczeniową. Lista poprzedników cechuje się większą złożonością obliczeniową wyznaczenia zbioru następników (O(n+m)), macierz incydencji jeszcze większą (O(n\*m)), co czyni ją najgorszym wyborem. Najlepsze pod kątem szybkości wyznaczania zbioru następników są struktury macierzy sąsiedztw oraz listy następników, obie cechujące się złożonością obliczeniową tej operacji równą O(n). Złożoność listy łuków zależna jest nie od liczby wierzchołków, lecz od liczby łuków i wynosi O(m).

Z podpunktu 2.2. wynika, że wygenerowane grafy były cykliczne. Graf acykliczny cechuje się liczbą łuków powrotnych równą 0, co nie miało miejsca w tym przypadku. Sortowanie topologiczne wymaga, by graf był skierowany oraz acykliczny. W grafie, w którym występują cykle, nie zawsze jest możliwe wyznaczenie poprzednika, który nie jest jednocześnie następnikiem jednego ze swoich następników. Algorytm sortowania topologicznego, choć nie pozwoli posortować wierzchołków grafu cyklicznego, może pomóc w wykryciu łuków powrotnych. Między dwoma wierzchołkami istnieje łuk powrotny, jeżeli spełniony jest warunek: d[v] < d[u] < f[u] < f[v], gdzie u jest wierzchołkiem, z którego wychodzi łuk, v jego następnikiem, d[x] oznacza wartość etykiety rozpoczęcia analizy wierzchołka x, a f[x] oznacza wartość etykiety zakończenia analizy wierzchołka x.

Procedura zliczania liczby łuków powrotnych musi przejść przez daną reprezentację grafu i odnaleźć wszystkie łuki. Z tego powodu najkorzystniejsza jest reprezentacja grafu o najniższej złożoności obliczeniowej wyznaczania zbioru łuków. Macierz sąsiedztwa cechuje się złożonością obliczeniową tej operacji równą O(n2), co czyni ją najgorszym wyborem. Złożoność obliczeniowa w przypadku listy łuków wynosi O(n), co powinno czynić zliczanie szybszym niż w przypadku listy następników, dla której wynosi ona O(n+m). W praktyce zliczanie łuków działa minimalnie szybciej przy liście następników ze względu na brak konieczności odczytywania numeru wierzchołka, z którego wychodzą łuki – w całej pętli dla danego wierzchołka jest on ten sam, zmienia się jedynie numer następnika. W liście łuków odczytany zaś musi być za każdym razem także numer wierzchołka bazowego. Gęstość grafów wpływa na czas zliczania łuków powrotnych ze względu na to, że wpływa na ich ilość. Na podstawie wyników z tabeli w punkcie 2.3. można stwierdzić, że czas zliczania w przypadku wszystkich trzech badanych reprezentacji grafu jest około dwukrotnie większy dla gęstości 0,4 niż dla gęstości 0,2 .

Macierz sąsiedztwa cechuje się stałym czasem testu łuku, o złożoności czasowej O(1), ponieważ aby sprawdzić, czy dany łuk istnieje, wystarczy odwołać się do wybranej komórki macierzy. Zaletą tej reprezentacji jest również bardzo niska złożoność obliczeniowa wyznaczania zbiorów następników oraz poprzedników, O(n) - wystarczy bowiem przeszukać wiersz lub kolumnę. Wadami macierzy sąsiedztwa są wysoka zajętość pamięciowa O(n2), wynikająca z konieczności przechowywania w pamięci macierzy kwadratowej n x n oraz duża złożoność obliczeniowa wyznaczania zbioru łuków, która również wynosi O(n2). Macierz sąsiedztwa jest najlepszą reprezentacją grafu, gdy najczęściej wykonywaną operacją będzie testowanie łuków.

Lista następników również cechuje się złożonością obliczeniową wyznaczenia zbioru następników O(n), lecz w odróżnieniu od macierzy sąsiedztwa, liczba kroków nie będzie zawsze równa n, lecz będzie równa liczbie następników wybranego wierzchołka. Czyni to listę następników najlepszym wyborem reprezentacji grafu w sytuacji, gdy to wyznaczanie następników będzie najczęściej wykonywaną operacją. Zajętość pamięciowa tej struktury to O(n+m), gdzie m potencjalnie osiąga n2, gdy graf jest grafem pełnym. Złożoność obliczeniowa wyznaczania zbioru poprzedników lub zbioru łuków również jest wysoka i wynosi O(n+m), co czyni listę następników nie najlepszym wyborem w pozostałych sytuacjach. Przykładowo, macierz sąsiedztwa cechuje się złożonością obliczeniową O(n) zarówno dla wyznaczania następników, jak i poprzedników.

Lista poprzedników działa analogicznie co lista następników i cechuje się tymi samymi zaletami oraz wadami, lecz w odniesieniu do poprzedników tam, gdzie w liście następników mowa o następnikach, oraz odwrotnie. Z tego powodu jest to najlepsza reprezentacja grafu do wyznaczania zbiorów poprzedników.

Lista łuków posiada zajętość pamięciową O(n), co czyni tę reprezentację najkorzystniejszą pod kątem wykorzystywanej pamięci. Test łuku wymaga sprawdzenia każdego łuku w strukturze, co czyni jego złożoność obliczeniową równą O(m). Z tego samego powodu wyznaczanie następników, poprzedników oraz łuków cechuje się tą samą złożonością obliczeniową, O(m). Jest to najlepsza możliwa złożoność obliczeniowa w przypadku wyznaczania zbioru łuków.

Macierz incydencji cechuje się bardzo wysoką złożonością obliczeniową wyznaczania zbiorów następników, poprzedników, oraz łuków, wynosi ona bowiem O(n\*m). Również zajętość pamięciowa macierzy incydencji wynosi O(n\*m). Test łuku posiada złożoność obliczeniową O(m). Macierze incydencji są niekorzystną reprezentacją grafów w znacznej większości przypadków, lecz sprawdzają się jako reprezentacja hipergrafów.