|  |  |
| --- | --- |
| L3 – środa 13:30 | |
| Aleksander Kamiński | 155840 |
| Olaf Hofman | 155974 |

Obraz zawierający tekst, zegar

Opis wygenerowany automatycznie

Zadanie 3  
Algorytmy Grafowe

Zadanie 3: algorytmy grafowe

Aleksander Kamiński, Olaf Hofman

# Badanie

W niniejszym badaniu sprawdzana jest sprawność czasowa i obliczeniowa oraz zasada działania algorytmów grafowych związanych z różnymi formami przechowywania struktury grafu oraz sortowania topologicznego elementów grafu dla różnych gęstości grafu i liczb elementów.

Badanie zostało przeprowadzone przez zmierzenie czasu wykonania przez program etapu obliczania etykiet grafu oraz liczby łuków powrotnych dla 10 kolejnych liczb wierzchołków, gdzie przy gęstościach grafu .

# Algorytm

Badany algorytm to algorytm sortowania topologicznego. Działa na zasadzie ustalenia etykiet dla każdego wierzchołka grafu w zależności od kolejności odwiedzania, a następnie wychodzenia z wierzchołków przy trawersowaniu grafu i ustawia wierzchołki w kolejności wynikającej z posortowania etykiety wyjścia z danego wierzchołka. Aby dokończyć sortowanie sprawdza również czy nie występują łuki zwrotne, czyli takie, które prowadzą z wierzchołka dalszego wśród posortowanych do jednego z poprzednich wierzchołków w kolejności wynikającej z sortowania, sprawdzając dla każdych dwóch wierzchołków i oraz j, czy , w to indeks wejścia do danego wierzchołka, a y to indeks wyjścia.

Algorytm jest skuteczny tylko dla grafów skierowanych (takich, w których między dwoma wierzchołkami występują łuki każdy o jednym kierunku) acyklicznych (niezawierających takich łuków, które tworzą drogę prowadzącą do jej własnego początku). Oznacza to, że w przypadku grafów nieskierowanych (posiadających wyłącznie krawędzie, po których można poruszać się w dwie strony) i niektórych grafów skierowanych nie można przeprowadzić takiego sortowania, ponieważ dla grafów cyklicznych nie można uszeregować wierzchołków w kolejności ich przechodzenia, gdyż zostanie to zaburzone przez łuki powrotne, nie pozwalające na jasne określenie jaka jest kolejność dwóch, połączonych nimi wierzchołków względem siebie. W grafie nieskierowanym ten sam problem występuje dla wszystkich wierzchołków, ponieważ każdą krawędź można przedstawić jako dwa łuki o przeciwnych kierunkach, tworzące cykl o dwóch wierzchołkach.

Na szybkość działania algorytmów grafowych wpływa wiele czynników związanych z konkretną implementacją oraz używanym sprzętem. Jednym z ważniejszych czynników jest sposób reprezentacji grafu, które mają różne właściwości i przez to są przystosowane do różnych zastosowań. Struktury będą omawiane na przykładowym grafie o n wierzchołkach, będących elementami zbioru V i m łukach, będących elementami zbioru E.

Macierz incydencji jest tablicą dwuwymiarową, która przedstawia strukturę grafu skierowanego. Jej wymiary to n na m. W polu tabeli o indeksie i, j (gdzie i to numery wierzchołków, a j to numery łuków) wpisuje się -1, jeżeli łuk j zaczyna się w wierzchołku i, 1 jeśli łuk j kończy się w wierzchołku i, 0 jeśli nie ma incydencji oraz 2 dla pętli własnej. Oznacza to, że złożoność pamięciowa tej struktury to (m to liczba łuków taka, że ). Sprawdzenie istnienia konkretnego łuku wymaga m operacji, bo wymaga sprawdzenia czy jeden wierzchołek jest początkiem danego łuku, a drugi jego końcem. Przejrzenie wszystkich łuków wychodzących z (albo wchodzących do) danego wierzchołka wymaga m operacji (sprawdzenia związku danego łuku z danym wierzchołkiem dla każdego łuku). Również m operacji wymaga sprawdzenie istnienia konkretnego łuku (przejrzenie czy dla któregoś łuku dane dwa wierzchołki stanowią jego początek i koniec).

Macierz sąsiedztwa wierzchołków jest strukturą podobną do macierzy incydencji. Jest również tablicą dwuwymiarową o wymiarach n na n, czyli o zajętości pamięciowej n2. W danym polu o indeksie i, j (gdzie i, j to numery wierzchołków) wpisuje się liczbę łuków wychodzących z wierzchołka i i wchodzących do wierzchołka j. Dla grafu prostego macierz sąsiedztwa jest zero-jedynkowa z zerami na głównej przekątnej. Dla grafów nieskierowanych jest ona również symetryczna. Złożoność obliczeniowa przejrzenia wszystkich łuków w macierzy sąsiedztwa wynosi n2, a złożoność przejrzenia wszystkich łuków zaczynających (albo kończących) się w danym wierzchołku wynosi n (przejrzenia odpowiednio jednego wiersza lub jednej kolumny macierzy). Sprawdzenie istnienia konkretnego łuku wymaga tylko jednej operacji.

Lista krawędzi jest strukturą tabeli dwuwymiarowej o wymiarach 2 na m, gdzie m to liczba krawędzi w grafie. W pierwszym wierszu tabeli zawiera numer wierzchołka, będącego początkiem danego łuku, a w drugim numer wierzchołka, będącego końcem tego samego łuku. (Może być odwrotnie.) Wyszukanie konkretnego wierzchołka w liście wymaga do m operacji, tak samo jak wypisanie wszystkich łuków wchodzących do (albo wychodzących z) danego wierzchołka. Ma mniejszą złożoność pamięciową niż poprzednie struktury, bo wynosi ona tylko 2m (i ).

Lista incydencji poprzedników jest listą list, która dla każdego z n wierzchołków zawiera listę wszystkich wierzchołków, w których zaczynają się łuki wchodzące do tego łuku. Powoduje to, że maksymalna możliwa złożoność pamięciowa listy incydencji poprzedników to n2. Przejrzenie wszystkich łuków wymaga m operacji. Przejrzenie wszystkich poprzedników danego wierzchołka wymaga maksymalnie n operacji (przejrzenia całej listy poprzedników danego wierzchołka). Przejrzenie następników danego wierzchołka wymaga za to m operacji, bo wymaga sprawdzenia, które wierzchołki mają dany wierzchołek w swojej liście poprzedników. Sprawdzenie istnienia łuku o dowolnym kierunku między dwoma wierzchołkami ma za to złożoność O(n), ponieważ wymaga tylko przejrzenia list poprzedników obu wierzchołków.

Lista incydencji następników również jest listą list, która dla każdego z n wierzchołków przechowuje listę wszystkich wierzchołków, do których wchodzą łuki wychodzące z danego wierzchołka. Oznacza to, że maksymalna możliwa złożoność pamięciowa listy incydencji następników wynosi n2. Przejrzenie wszystkich łuków wymaga m operacji. Przejrzenie wszystkich łuków wychodzących z danego wierzchołka wymaga tylko do n operacji, czyli przejrzenia całej jego listy następników. Przejrzenie wszystkich łuków wchodzących do danego wierzchołka ma natomiast złożoność O(m) (gdzie m to liczba krawędzi), ponieważ wymaga sprawdzenia dla wszystkich wierzchołków czy dany wierzchołek jest na ich liście następników. Potwierdzenie istnienia łuku o dowolnym kierunku między dwoma wierzchołkami ma złożoność O(n), ponieważ wymaga przejrzenia list następników obu wierzchołków.

Macierz grafu jest kolejną strukturą, która dla grafu prostego występuje w postaci tablicy dwuwymiarowej n na n+3, więc jej zajętość pamięciowa to . Każda komórka M[i, j] macierzy grafu (gdzie zawiera liczbę k, należącą do jednego z przedziałów , , , . Dla każdego wierzchołka i istnieje tylko jedna liczba k należąca do danego przedziału, to znaczy, że jeżeli wierzchołek i jest w pewnej relacji do wierzchołka j oraz do wierzchołka j+l (, to to zarówno M[i, j] oraz M[i, j+l] będą zawierały tą samą liczbę, wskazującą na ostatni wierzchołek, pozostający w takiej samej relacji z i, jak j (ostatni następnik, ostatni poprzednik lub ostatni nieincydentny). Liczba ta jest numerem wierzchołka powiększoną o wielokrotność n tak, by znajdowała się w przedziale odpowiadającym relacji między wierzchołkiem i oraz j. K należy do przedziału D, jeśli nie istnieje łuk pomiędzy wierzchołkami i oraz j. K należy do przedziału A, jeśli istnieje łuk wychodzący z wierzchołka i, który wchodzi do wierzchołka j. K należy do przedziału B, jeśli istnieje łuk wchodzący do wierzchołka i, który wychodzi z wierzchołka j. K należy do przedziału C, jeśli istnieje zarówno łuk wychodzący z wierzchołka i, który wchodzi do j oraz łuk wychodzący z wierzchołka j, który wchodzi do i. Ponadto w każdej komórce o indeksie i, n znajduje się numer pierwszego następnika wierzchołka i. W każdej komórce o indeksie i, n+1 znajduje się numer pierwszego poprzednika wierzchołka i. W każdej komórce o indeksie i, n+2 znajduje się numer pierwszego wierzchołka, z którym wierzchołek i jest nieincydentny. Jeśli dany wierzchołek nie ma poprzedników, następników lub wierzchołków nieincydentnych, to we właściwe komórki macierzy wpisuje się zero. Do sprawdzenia istnienia pojedynczej krawędzi w macierzy grafu wystarczy jedna operacja – sprawdzenie pojedynczej komórki. Przejrzenie wszystkich następników i poprzedników danego wierzchołka wymaga do n operacji – sprawdzenia wszystkich relacji zapisanych w wierszu przeznaczonym dla danego wierzchołka. Przejrzenie wszystkich krawędzi w grafie w tej strukturze wymaga natomiast jedynie do m operacji.

Inną rzeczą wpływającą na sprawność i wyniki działania algorytmów grafowych może być sposób przeszukiwania grafu. Wybór między metodą DFS („w głąb”) lub BFS („wszerz”) wpływa przede wszystkim na kolejność odwiedzania wierzchołków grafu, czyli dla algorytmu sortowania topologicznego – na ostateczny wynik sortowania.

Metoda DFS zaczyna przechodzenie grafu nieskierowanego od dowolnego wierzchołka, a skierowanego od wierzchołka, który nie ma żadnych poprzedników. Następnie odwiedza wcześniej nieodwiedzony następnik wybranego wierzchołka i następnik tego następnika aż nie dojdzie do wierzchołka, który nie posiada nieodwiedzonych następników. Przy wejściu do wierzchołka jego numer jest odkładany na stos. Jeśli z danego wierzchołka nie można przejść do żadnego nieodwiedzonego następnika, to algorytm zdejmuje ze stosu numer tego wierzchołka i cofa się do poprzedniego, wskazywanego przez ostatni numer na stosie. Dzieje się tak, dopóki algorytm nie znajdzie znów wierzchołka, który posiada wcześniej nieodwiedzone następniki.

Metoda BFS rozpoczyna się w taki sam sposób jak DFS. Następnie odwiedza po kolei wszystkie uprzednio nieodwiedzone następniki pierwszego wierzchołka. Numer każdego odwiedzonego wierzchołka zostaje odłożony na stos. Po tym algorytm sprawdza czy wierzchołki od dołu stosu mają jeszcze nieodwiedzone następniki i wybiera pierwszy, który takowe ma, a następnie odwiedza po kolei jego następniki i dopisuje ich numery do stosu, po czym znów przegląda stos, żeby znaleźć wierzchołek, mający nieodwiedzone następniki. Dzieje się to tak długo, aż wszystkie wierzchołki zostaną odwiedzone.

Ostatecznie, przy sortowaniu grafu, ważny jest wybór algorytmu, który zdecyduje o szybkości oraz wyniku działania w danym zastosowaniu.

Algorytm Kahna, oparty na usuwaniu wierzchołków niezależnych działa na skierowanych acyklicznych grafach G = (V, E), gdzie V jest niepustym zbiorem wierzchołków, a E niepustym zbiorem łuków. Pierwszym krokiem jego działania jest utworzenie listy L o długości , do której będą wstawiane wierzchołki w porządku topologicznym. Następnie szuka w grafie wierzchołka niezależnego, czyli takiego, że jego stopień wejściowy jest zerowy. Ten wierzchołek zostaje zapisany na koniec listy L, a następnie zostaje usunięty z grafu. Algorytm sprawdza czy w grafie pozostały jeszcze jakieś wierzchołki, i jeśli tak, to znów szuka wierzchołka niezależnego. Kiedy nie zostaną już żadne wierzchołki, istniejąca lista L jest wynikiem sortowania topologicznego grafu G.

Algorytm Tarjana, oparty na metodzie DFS również działa na takich samych danych wejściowych jak algorytm Kahna – Grafie skierowanym acyklicznym G = (V, E), gdzie V jest niepustym zbiorem wierzchołków, a E niepustym zbiorem łuków. Pierwszym krokiem algorytmu jest przypisanie wszystkim wierzchołkom grafu G koloru białego. Następnie tworzy listę L, która będzie zawierała wierzchołki w porządku posortowanym topologicznie. Zależnie od implementacji algorytm może korzystać ze stosu. Algorytm wybiera następnie wierzchołek startowy o przypisanym kolorze białym; najlepiej by był to wierzchołek niezależny, ale może taki nie istnieć. Wierzchołek u zostaje odwiedzony i pokolorowany na szaro. Jeśli istnieją białe następniki wierzchołka u, to jeden z nich zostaje wybrany w miejsce wierzchołka u, odwiedzony i pokolorowany na szaro, a następnie sprawdzane jest czy ma białe następniki. Jeżeli aktualny wierzchołek nie ma białych następników, to zostaje pokolorowany na czarno i odłożony na stos (w przypadku implementacji bez stosu – dopisany na początek listy), a następnie algorytm cofa się do jego szarego poprzednika na już przebytej ścieżce aż nie znajdzie wierzchołka z białym następnikiem. Jeśli dany wierzchołek został pokolorowany na czarno i nie posiada szarego poprzednika, to algorytm szuka od nowa białego wierzchołka początkowego. Jeżeli został znaleziony nowy wierzchołek początkowy, algorytm odwiedza go i koloruje na szaro, zaczynając praktycznie tak, jak od nowa. Jeżeli w grafie nie ma więcej białych wierzchołków, to w implementacji ze stosem wierzchołki są zdejmowane ze stosu od góry i wpisywane na koniec listy wynikowej L. W ten sposób powstaje lista wierzchołków posortowanych topologicznie.

# Wyniki

Wyniki badania zostały przedstawione w tabelach i na odpowiadających im wykresach poniżej. Na ich podstawie zostały wyciągnięte wszystkie wymagane wnioski.