Grafy

Konrad Łakomy

11 05 2014

Zadanie do wykonania

Implementacja struktury grafu oraz algorytmów przeszukiwania:

1. Przeszukiwanie grafu w głąb DFS (ang. Depth-first search)
2. Przeszukiwanie grafu wszerz BFS (ang. Breadth-first search)

Opis implementacji grafu

Graf (ang. graph) jest strukturą zbudowaną z punktów zwanych wierzchołkami (ang. vertex) oraz łączących te punkty krawędzi (ang. edge). Położenie wierzchołków na płaszczyźnie lub w przestrzeni jest dowolne. Krawędzie symbolizują drogi, którymi możemy się poruszać po grafie.

W celu zaimplementowania struktury grafu wykorzystano **macierz sąsiedztwa (ang. adjacency matrix)** , gdzie poszczególne wiersze oznaczają wierzchołki grafu. Numer wiersza jest numerem początkowego wierzchołka krawędzi. Teraz elementy wiersza informują nas, czy ten wierzchołek jest połączony krawędzią z wierzchołkami grafu, których numery określają kolejne kolumny. Na przykład wiersz 0 definiuje wszystkie krawędzie grafu, które rozpoczynają się w wierzchołku 0.

Opis implementacji metody DFS

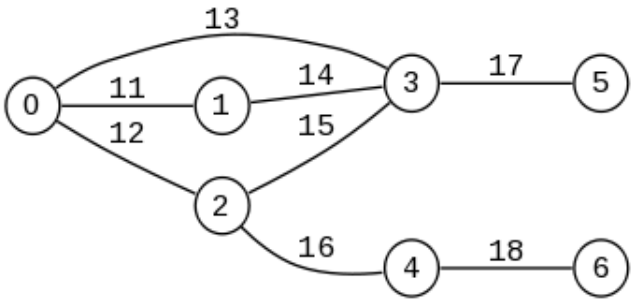
Metoda DFS (ang. Depth First Search - przeszukiwanie najpierw w głąb) rozpoczyna działanie w wybranym wierzchołku grafu, który oznacza jako odwiedzony. Następnie przechodzi wzdłuż dostępnej krawędzi do sąsiada tego wierzchołka, który nie został jeszcze odwiedzony. Przechodzenie jest kontynuowane dalej (w głąb grafu), aż zostanie osiągnięty wierzchołek, który nie posiada nie odwiedzonych sąsiadów. Wtedy procedura wraca do poprzednio odwiedzonego wierzchołka i kontynuuje wzdłuż kolejnej dostępnej krawędzi.

Procedurę DFS realizujemy za pomocą stosu. Stos jest strukturą danych, w której składujemy kolejne elementy, lecz odczyt zawsze odbywa się w kierunku odwrotnym - od ostatnio umieszczonego na stosie elementu.

Opis implementacji metody BFS

Metoda BFS (ang. Breadth First Search - przeszukiwanie najpierw w szerz) wykorzystuje strukturę danych zwaną kolejką (ang. queue). Jest to struktura sekwencyjna. Dane dopisujemy na jej końcu, podobnie jak na stosie. Odczyt danych odbywa się kolejno z początku kolejki tzn. dane odczytujemy z kolejki w takiej samej kolejności, w jakiej zostały zapisane w niej. Kolejkę możemy potraktować jako tzw. bufor, który przechowuje dane. Im więcej danych w kolejce, tym dłużej dany element jest przez nią przechowywany.

Przykłady struktur grafu - Przykład 1 - Graf spójny



Rysunek 1: Przykład nr. 1

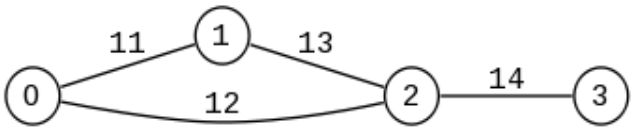
Macierz sąsiedztwa

V	0	1	2	3	4	5	6
0:	0	11	12	13	0	0	0
1:	11	0	0	14	0	0	0
2:	12	0	0	15	16	0	0
3:	13	14	15	0	0	17	0
4:	0	0	16	0	0	0	18
5:	0	0	0	17	0	0	0
6:	0	0	0	0	18	0	0

Algorytmy przeszukiwania:

1. BFS: 0 1 2 3 4 5 6
2. DFS: 0 3 5 2 4 6 1

Przykłady struktur grafu - Przykład 2 - Graf spójny



Rysunek 2: Przykład nr. 2

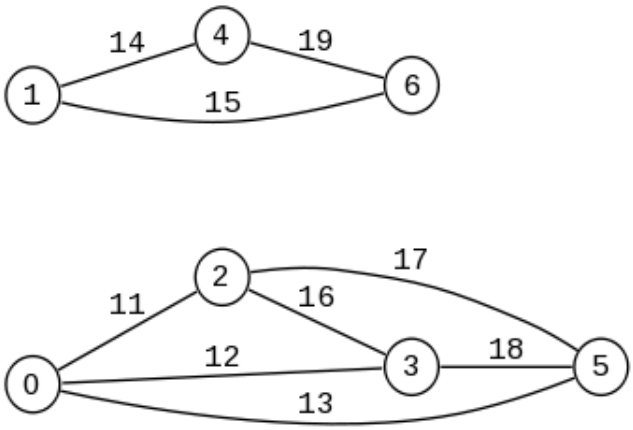
Macierz sąsiedztwa

V	0	1	2	3
0:	0	11	12	0
1:	11	0	13	0
2:	12	13	0	14
3:	0	0	14	0

Algorytmy przeszukiwania:

1. BFS: 0 1 2 3
2. DFS: 0 2 3 1

Przykłady struktur grafu - Przykład 3 - Graf niespójny



Rysunek 3: Przykład nr. 3

Macierz sąsiedztwa

V	0	1	2	3	4	5	6
0:	0	0	11	12	0	13	0
1:	0	0	0	0	14	0	15
2:	11	0	0	16	0	17	0
3:	12	0	16	0	0	18	0
4:	0	14	0	0	0	0	19
5:	13	0	17	18	0	0	0
6:	0	15	0	0	19	0	0

Algorytmy przeszukiwania:

1. BFS: 0 2 3 5
2. DFS: 0 5 3 2

Wnioski

Macierz sąsiedztwa

Reprezentacja grafu za pomocą macierzy sąsiedztwa wymaga zarezerwowania pamięci o rozmiarze $O(n^2)$.Dla dużych grafów reprezentacja może być bardzo kosztowna, a nawet wykraczająca poza możliwości zwykłego komputera PC. Zaletą macierzy sąsiedztwa jest stały koszt O(1) sprawdzenia, czy dwa wierzchołki v_i oraz v_j łączy krawędź. W tym celu wystarczy sprawdzić elementy $a_{i,j}$ oraz $a_{j,i}$ Jeśli jeden z nich jest różny od 0, to krawędź istnieje. Jeśli oba są równe zero, krawędzi brak. Wynika stąd, iż reprezentacja ta jest szczególnie opłacalna w algorytmach, które często muszą sprawdzać występowanie krawędzi pomiędzy wybranymi wierzchołkami grafu, a liczba wierzchołków grafu nie jest zbyt duża.

Złożoność obliczeniowa DFS

Ponieważ każdy wierzchołek jest odwiedzany dokładnie raz, a każda krawędź wychodząca z każdego węzła jest sprawdzana (w celu potencjalnego odwiedzenia sąsiada) również jeden raz, więc algorytm DFS wykonuje się w czasie O(V+E), co jest dobrym wynikiem, gdyż liniowo zależy od rozmiaru grafu.

Złożoność obliczeniowa BFS

Każdy węzeł v ze zbioru wierzchołków V jest dokładnie raz umieszczany w kolejce, a także raz jest z niej wyjmowany. Zatem operacje te potrzebują czasu O(V). Prócz tego wiemy, że podczas usuwania wierzchołka z kolejki sprawdzana jest lista jego sąsiadów (dokładnie raz dla każdego węzła), zatem każda krawędź jest również jednokrotnie analizowana. Wynika z tego, że potrzebujemy na to czasu O(E). W związku z tym całkowity czas działania algorytmu przeszukiwania wszerz jest liniowo proporcjonalny do rozmiaru grafu i wynosi O(V+E). Podsumowując, oba algorytmy, DFS i BFS posiadają identyczną złożoność obliczeniową.