Sprawozdanie - Laboratorium 08 PAMSI

Artur Gasiński — 218685

25.04.2016

1 Zadanie

1. Implementacja grafu wybraną metodą (wraz z algorytmami przechodzenia w głąb DFS i wszerz BFS).
2. Pomiar czasów przejścia grafu w głąb i wszerz w zależności od liczby wierzchołków n: $n = 10, 10^2, 10^3, 10^6, 10^7$;

2 Wykonanie

1. Struktura programu:
   - interfejs IRunnable,
   - interfejs IGraph (zawiera podstawowe metody ADT grafu),
   - klasa Graph, implementująca interfejs IGraph:
     - graf zbudowany jest z listy wierzchołków reprezentowanych przez liczby naturalne,
     - połączenia między wierzchołkami (krawędzie) reprezentowane są w tablicy list sąsiedztwa dla każdego wierzchołka,
     - w liście zapisane są numery wierzchołków-sąsiadów,
     - główną zaletą takiej implementacji jest mniejsze zapotrzebowanie na pamięć niż w przypadku macierzy sąsiedztwa - wynosi ona $O(n + m)$ dla list sąsiedztwa, gdzie n to liczba wierzchołków, a m to liczba krawędzi, a dla macierzy $O(n^2)$, co jest istotne dla dużych n rzędu 10^6.
     - ponadto implementacja listowa jest łatwo modyfikowalna, tzn. umożliwia przedstawienie wielu rodzajów grafu (np. ważonych),
     - założono, że graf ma być spójny (tzn. nie ma ani jednego węzła nie połączonego z resztą grafu), nieskierowany (krawędzie między dwoma wierzchołkami prowadzą w obie strony) oraz nie ważony (tzn. wszystkie krawędzie mają domyślną wagę równą 1), nie ma też krawędzi wychodzących z wierzchołka i wracających do niego samego.
   - klasa Graph-Test, implementująca intefejs IRunnable dla klasy Graph,
   - pomocnicze interfejsy i implementacje listy dwukierunkowej, kolejki i stosu (odpowiednio klasy IList, BList, IQueue, Kolejka, IStack, Stos),

- klasy Stopwatch i AdvancedStopwatch, wykonujące pomiar czasów korzystając z funkcji gettimeofday(),
- funkcja główna, zarządzająca kolejnością wykonywania zadań.

2. Pomiary wykowano w następujący sposób: Dla każdego n wywoływano metodę $Prepare(n)$, która dodawała n wierzchołków do grafu i tworzyła między nimi spójne powiązanie. Dokonywano tego w następujący sposób: tworzono tymczasową tablicę z numerami wierzchołków i następnie losowo zamieniano komórki tablicy. Następnie dodawano krawędzie zgodnie z wylosowaną kolejnością. Na koniec przygotowania, metoda Prepare() generowała $2n$ losowych krawędzi. Tak przygotowany graf poddawano przejściom wszerz BFS i w głąb DFS. Dla obu przejść zmierzono czas działania. Pomiary przeprowadzono w seriach po 10 razy. Za każdym razem generowano nowy graf, aby pomiary czasu były bardziej wiarygodne. Zmodyfikowano nieco rozmiary problemów: $n = 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7$.

3 Pomiary średnich czasów

Tabela 1: Przejście grafu w głąb (DFS)

n	10	10^2	10^3	10^4	10^5	10^6	10^7
czas [s]	0,00000126	0,00001439	0,00017889	0,00345816	0,06378120	0,75835764	10,63500213

Tabela 2: Przejście grafu wszerz (BFS)

n	10	10^2	10^3	10^4	10^5	10^6	10^7
czas [s]	0,00000147	0,00001381	0,00019207	0,00336020	0,06393588	0,76194568	10,70977833

4 Wnioski

Wyniki pomiarów nie wykazują różnicy w szybkości działania algorytów przechodzenia wszerz BFS i w głąb DFS. Nie jest to niezrozumiałe, gdyż w gruncie rzeczy różnią się tylko wykorzystaniem innej pomocniczej struktury danych: kolejki w BFS i stosu w DFS. Oba algorytmy przechodzą po listach sąsiedztwa kolejnych wierzchołków i wykonują tyle samo operacji.

5 Wykres


Przechodzenie grafu wszerz i w głąb
BFS
DFS
12
10
8
6
4
2
0
czas [s]
0
2000000
4000000
6000000
8000000
10000000
12000000
n (rozmiar problemu)