

Zadanie projektowe nr 2

Badanie efektywności algorytmów grafowych w zależności od rozmiaru instancji oraz sposobu reprezentacji grafu w pamięci komputera.

Należy zaimplementować oraz dokonać pomiaru czasu działania wybranych algorytmów grafowych rozwiązujących następujące problemy:

- a. wyznaczanie minimalnego drzewa rozpinającego (MST) - algorytm Prima oraz algorytm Kruskala,
- b. wyznaczanie najkrótszej ścieżki w grafie – algorytm Dijkstry oraz algorytm Forda-Bellmana,
- c. wyznaczanie maksymalnego przepływu – algorytm Forda Fulkersona (tylko na ocenę 5 i 5.5).

Algorytmy te należy zaimplementować dla obu poniższych reprezentacji grafu w pamięci komputera:

- reprezentacja macierzowa (macierz sąsiedztwa),
- reprezentacja listowa (lista sąsiadów).

Należy przyjąć następujące założenia:

- wszystkie struktury danych powinny być alokowane dynamicznie,
- przepustowość krawędzi jest liczbą całkowitą,
- po zaimplementowaniu każdego z algorytmów dla obu reprezentacji należy dokonać pomiaru czasu działania algorytmów w zależności od rozmiaru grafu oraz jego gęstości (liczba krawędzi w stosunku do maksymalnej liczby możliwych krawędzi dla grafu o danej liczbie wierzchołków). Badania należy wykonać dla 7 różnych (reprezentatywnych) liczb wierzchołków V oraz następujących gęstości grafu: 20%, 60% oraz 99%. Dla każdego zestawu: reprezentacja, liczba wierzchołków i gęstość należy wygenerować po 50 losowych instancji, zaś w sprawozdaniu umieścić wyniki uśrednione,
- aby wygenerować graf należy najpierw wygenerować drzewo rozpinające, a dopiero potem pozostałe krawędzie,
- dodatkową funkcją programu musi być możliwość sprawdzenia poprawności zaimplementowanych operacji i zbudowanej struktury (szerzej w na ten temat w dalszej części dokumentu),
- do dokładnego pomiaru czasu w systemie Windows w c++ można skorzystać z funkcji `QueryPerformanceCounter` lub `std::chrono::high_resolution_clock` (opis na stronie <http://cpp0x.pl/forum/temat/?id=21331>)

- dopuszczalnymi językami programowania są języki kompilowane do kodu natywnego (np. C, C++), a nie interpretowane lub uruchamiane na maszynach wirtualnych (np. JAVA, .NET, Python); dopuszczalne jest odstępstwo od tej reguły za zgodą prowadzącego
- używanie okienek nie jest konieczne i nie wpływa na ocenę (wystarczy wersja konsolowa),
- nie wolno korzystać z gotowych bibliotek np. STL, Boost lub innych – wszystkie algorytmy i struktury muszą być zaimplementowane przez studenta (nie kopiować gotowych rozwiązań) – oprócz pewnych wyjątków zapisanych przy ocenianiu,
- implementacja projektu powinna być wykonana w formie jednego programu,
- kod źródłowy powinien być komentowany.

Sprawdzenie poprawności zbudowanej struktury/operacji obejmuje:

- wczytanie struktury grafu z pliku tekstowego (zapytać o nazwę pliku – nie wpisywać nazwy pliku na sztywno do programu). Plik zawiera opis poszczególnych krawędzi według wzoru: początek krawędzi, koniec krawędzi oraz waga/przepustowość. Struktura pliku jest następująca:
 - a. w pierwszej linii zapisana jest liczba krawędzi oraz liczba wierzchołków (rozdzielone spacją) oraz w zależności od algorytmu dodatkowe parametry (patrz niżej)
 - b. w przypadku problemu najkrótszej ścieżki w pierwszej linii dodatkowo wpisany jest nr wierzchołka początkowego, a dla problemu przepływu nr wierzchołka startowego i końcowego,
 - c. wierzchołki numerowane są w sposób ciągły od zera,
 - d. w kolejnych liniach znajduje się opis krawędzi (każda krawędź w osobnej linii) w formie trzech liczb przedzielonych spacjami (wierzchołek początkowy, wierzchołek końcowy oraz waga/przepustowość),
 - e. dla problemu MST pojedynczą krawędź traktuje się jako nieskierowaną, natomiast dla algorytmów najkrótszej drogi i maksymalnego przepływu jako skierowaną,
- losowe wygenerowanie grafu (jako dane podaje się liczbę wierzchołków oraz gęstość w %). Graf z pliku i wygenerowany losowo zajmują tę samą zmienną (czyli ostatnia operacja generowania lub wczytywania z pliku nadpisuje poprzednią),
- możliwość wyświetlenia na ekranie wczytanego lub wygenerowanego grafu w formie reprezentacji listowej i macierzowej,
- uruchomienie algorytmu dla obu reprezentacji i wyświetlenie wyników na ekranie

Poniższe operacje należy zrealizować w formie menu dla każdego problemu z osobną wykonując operację od razu na dwóch strukturach jednocześnie (na początku programu wybieramy problem a dopiero później właściwe menu - tak jak w projekcie pierwszym najpierw wybieraliśmy strukturę):

1. Wczytaj z pliku (od razu wyświetlić strukturę listową i macierzową po wczytaniu).
2. Wygeneruj graf losowo (od razu wyświetlić strukturę listową i macierzową po wczytaniu).
3. Wyświetl listowo i macierzowo na ekranie.
4. Algorytm 1 (np. Prima) macierzowo i listowo z wyświetleniem wyników
5. Algorytm 2 (np. Kruskala) macierzowo i listowo z wyświetleniem wyników

Wyświetlanie wyników:

- a) w przypadku MST wyświetlić listę krawędzi drzewa rozpinającego z wagami oraz sumaryczną wartość
- b) dla problemu najkrótszej drogi dla każdego wierzchołka wyświetlić wartość (koszt) drogi oraz drogę w postaci ciągu wierzchołków o wierzchołka startowego do każdego pozostałego

Uwaga!

Po uruchomieniu program powinien zapytać, który problem chcemy rozwiązywać i przejść do odpowiedniego menu.

Sprawozdanie (w formie papierowej) powinno zawierać:

- krótki wstęp zawierający oszacowanie złożoności obliczeniowej poszczególnych problemów na podstawie literatury,
- plan eksperymentu czyli założenia co do wielkości struktur, sposobu generowania ich elementów, sposobu pomiaru czasu, itp.
- opis metody generowania grafu. Sposób powinien zapewnić spójność oraz zmienną strukturę grafu,
- wyniki - należy przedstawić w tabelach oraz w formie wykresów dla każdego problemu osobno (oddzielnie MST i najkrótsza droga w grafie). Dla każdego problemu (MST oraz najkrótsza ścieżka) należy przygotować następujące wykresy:
 - a. wykresy typ1 (osobne wykresy dla każdej reprezentacji grafu) - w formie linii (połączonych punktów), których parametrem jest gęstość grafu i typ algorytmu (Kruskal/Prim lub Dijkstra/Bellman) - czyli $2 \times 3 = 6$ linii na rysunek (2 algorytmy \times 3 gęstości)
 - b. wykresy typ2 (osobne wykresy dla każdej gęstości grafu) – w formie linii których parametrem jest typ algorytmu (Kruskal/Prim lub Dijkstra/Bellman) i typ

reprezentacji grafu (czyli 4 linie na każdy rysunek – 2 algorytmy x 2 reprezentacje),

c. analogicznie jest dla algorytmu maksymalnego przepływu.

- wszystkie wykresy (ich osie) to czas wykonania algorytmu (oś Y) w funkcji ilości wierzchołków (oś X),
- nie umieszczać na jednym rysunku wyników działania algorytmów z różnych problemów,
- wnioski dotyczące efektywności poszczególnych struktur. Wskazać (jeśli są) przyczyny rozbieżności pomiędzy złożonościami teoretycznymi a uzyskanymi eksperymentalnie,
- załączony kod źródłowy w formie elektronicznej (cały projekt wraz z wersją skompilowaną programu) na dysku google'a oraz sprawozdanie w formie papierowej.
- w przypadku algorytmu Kruskala, który operuje na liście krawędzi należy, po wczytaniu grafu z pliku, utworzyć macierz sąsiedztwa/listę sąsiedztwa. Tak utworzona struktura ma być parametrem algorytmu Kruskala w którym nastąpi transformacji na listę krawędzi i właściwe wykonanie algorytmu.

Ocena projektu:

3.0 – po jednym algorytmie z każdego problemu: Dijkstra + Prim lub Kruskal (możliwość korzystania z biblioteki STL) – po 0,5 za każdy algorytm

3.5 - trzy algorytmy: Dijkstra + Prim + Kruskal (możliwość korzystania z biblioteki STL) – po 0,5 za każdy algorytm

4.0 – pod dwa algorytmy z każdego problemu (możliwość wykorzystania z biblioteki STL) – po 0,5 za każdy algorytm

4.4 – pod dwa algorytmy z każdego problemu (bez STL) – po 0,6 za każdy algorytm

5.0 – wersja obiektowa (bez STL) + algorytm znajdowania maksymalnego przepływu Forda-Fulkersona (wynajdywanie ścieżek metodą przeszukiwania grafu w głąb) – j.w.+0,6 za FF

5.5- wersja obiektowa (bez STL) + algorytm znajdowania maksymalnego przepływu Forda-Fulkersona (wynajdywanie ścieżek metodą przeszukiwania grafu w głąb i wszerek).

Uwaga!

1. Jeden z problemów należy oddać wcześniej (patrz warunki zaliczenia przedmiotu).

2. Naiwna implementacja kolejki priorytetowej (np. zwykła tablica z szukaniem wartości minimalnej – optymalną strukturą dla kolejki priorytetowej jest kopiec) lub naiwna implementacja zbiorów rozłącznych (tzw. tablica kolorów-liczb całkowitych) w przypadku algorytmu Kruskala powoduje obniżenie oceny.

3. Dla dla każdej reprezentacji grafu ma być osobny algorytm, a nie wspólny interfejs algorytmu pod który podpinamy raz macierz raz listę.

4. Przy szacowaniu złożoności nie zamieniać ilości krawędzie (E) na kwadrat ilości wierzchołków pomnożony przez stałą

5. Aby wygenerować spójny graf dla grafu nieskierowanego należy najpierw wygenerować sobie drzewo rozpinające i następnie dokładać do niego krawędzie – gwarantuje to, że graf będzie spójny. W przypadku grafów skierowanych należy dodatkowo zapewnić warunek, że do każdego wierzchołka dochodzi co najmniej jedna krawędź (w przeciwnym przypadku do tego wierzchołka nigdy nie da się dojść od innego wierzchołka) – można to zapewnić przez wygenerowanie pętli przechodzącej przez wszystkie wierzchołki, a następnie dokładać to tej pętli krawędzie skierowane.

6. Aby wygenerować graf o gęstości 99% należy wygenerować graf o gęstości 100% a następnie usunąć losowo 1% krawędzi

Przykładowe linki do nienaiwnej implementacji zbiorów rozłącznych (ang. disjoint sets) :

<http://algorytmika.wikidot.com/find-union>

<https://www.cise.ufl.edu/~sahni/cop3530/slides/lec222.pdf>