PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI – GRAFY

Wtorek 15:15

Rafał Rzewucki 248926

Prowadzący:

Mgr inż. Marcin Ochman

# Wstęp

Celem projektu było zbadać efektywność jednego z powyższych algorytmów w zależności od s

# Opis Algorytmu Bellmana-Forda

# Algorytm służący do znalezienia najkrótszej ścieżki w grafie wazony z wierzchołka źródłowego do wszystkich pozostałych wierzchołków. Działanie algorytmu opiera się na metodzie relaksacji (sprawdzaniu, czy przy przejściu dana krawędzią grafu, nie otrzymamy krótszej ścieżki niż dotychczasowa).

# Algorytm Bellmana-Forda w porównaniu z algorytmem Dijkstry jest wolniejszy, jednakże bardziej uniwersalny. Algorytm ten jest bowiem w stanie obsługiwać grafy z ujemnymi wartościami krawędzi oraz wykrywać ujemne cykle, które być może zostały stworzone przez te ujemne wartości.

# 3. Opis programu

Program składa się z pliku głównego main.cpp oraz dodatkowych plików nagłówkowych w tym z plików MergeSort.h, QuickSort.h oraz IntroSort.h, w których zostały zaimplementowane klasy z metodami statycznymi odpowiedzialnymi za poprawne przeprowadzanie algorytmu sortowania. Każda klasa zawiera dodatkowo metodę sort(), która przyjmuje tablicę do posortowania oraz odpowiednie parametry i nie zwraca żadnej wartości oraz metodę sortAll(), która przyjmuje wszystkie tablice, jakie zostały przygotowane do badania czasu sortowania i wywołuje metodę sort() swojej klasy dla każdej tabeli z osobna. Dodatkowo metoda sortAll() zwraca liczbę typu double, która jest czasem wykonywania algorytmu dla wszystkich tablic podanym w sekundach.

Do badania czasu wykonywania algorytmu została wykorzystana biblioteka „chrono” a dokładniej zegar w niej zawarty o nazwie „high\_resolution\_clock”, który gwarantuje największą możliwą na danym komputerze rozdzielczość pomiaru czasu.

W pliku main.cpp został również zaimplementowany mechanizm umożliwiający uruchomienie testowania każdego algorytmu z osobna równocześnie na osobnych wątkach, co umożliwia pełne wykorzystanie możliwości obliczeniowych komputera i znacznie przyśpiesza testowanie zaimplementowanych algorytmów.

Dodatkowo została zaimplementowana klasa CVSaver, która umożliwia w prosty sposób zebrać czasy poszczególnych sortowań i zapisać je w pliku z rozszerzeniem .csv, który mogą bez problemu odczytać programy kalkulacyjne.

# Prezentacja wyników

Dla jak najlepszego wyeliminowania błędów pomiarów związanych z uruchomionymi dodatkowymi programami w tle, pomiary zostały powtórzone trzy razy i została wyciągnięta z nich średnia arytmetyczna.

Na podstawie przedstawionych danych widać, że algorytmy zachowują swoją teoretyczną złożoność obliczeniową. W sortowaniu szybkim jak i introspektywnym dla 50% początkowego posortowania danych występuje dziwna anomalia, której obecnie nie potrafię wyjaśnić.

Mimo zaistniałej sytuacji można zauważyć, że czas potrzebny na posortowanie danych zmniejsza się tym bardziej im większa część danych jest wstępnie posortowana. Dla każdego z algorytmów najkrótszy czas sortowania otrzymano dla tablicy posortowanej w odwrotnej kolejności.

Dodatkowo na podstawie zebranych danych można porównać wydajność algorytmów pomiędzy sobą dla poszczególnych danych.

Z powyższych danych można wyciągnąć następujące wnioski:

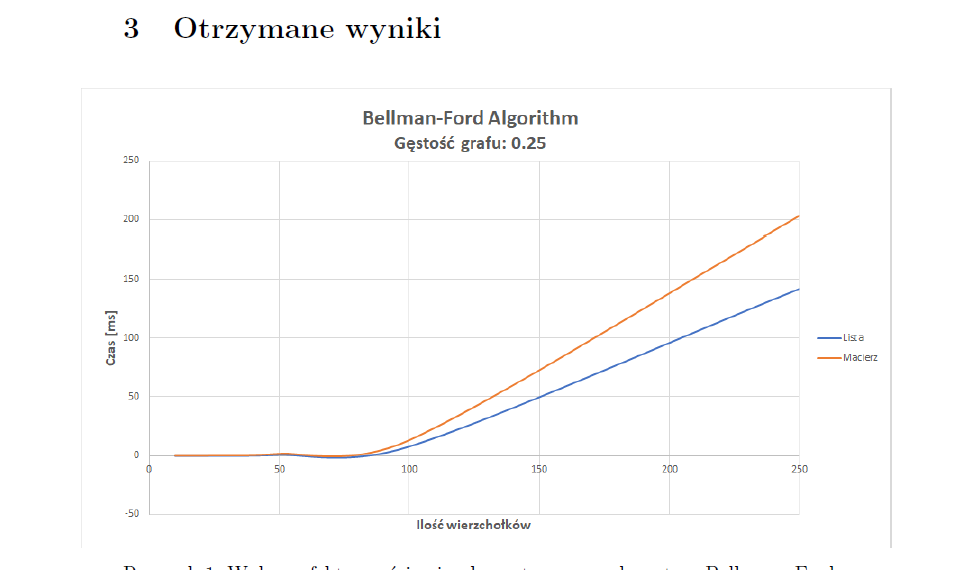
* Algorytm sortowania szybkiego był najszybszy w większości przypadków – może to być spowodowane nie natrafieniem na najgorszy przypadek dla tego algorytmu, przez co złożoność obliczeniowa była znacznie mniejsza.
* Algorytm sortowania introspektywnego zawsze był najwolniejszym – jest to z pewnością spowodowane niedoskonałą implementacją sortowania części tablicy poprzez kopcowanie. Jest to cześć programu, która może zostać w przyszłości zoptymalizowana poprzez usunięcie tablic tymczasowych.

# Podsumowanie

Mimo iż nazwa algorytmu szybkiego, jak i powyższe wykresy sugerują, że jest on najszybszy, to nie zawsze jest to prawdą. W ogólnym przypadku dość szybkim algorytmem jest sortowanie przez scalanie, gdyż jego złożoność obliczeniowa dla każdego przypadku jest taka sama. Ważne jest również wybranie odpowiedniego elementu rozdzielającego. Sortowanie introspektywne na ogół jest nieco szybsze od sortowania szybkiego, jednak w mojej implementacji występują nadmiarowe operacje co powoduje, że w powyższych testach wypada on najgorzej. Należy zaznaczyć, że ogromną zaletą sortowania szybkiego i introspektywnego (w normalnej implementacji) jest to, że nie wymagają one żadnych dodatkowych tablic pomocniczych, przez co są one bardziej oszczędne w korzystaniu z dostępnej pamięci.

# Literatura

* <http://www.algorytm.org/algorytmy-sortowania/sortowanie-przez-scalanie-mergesort.html>
* <https://www.geeksforgeeks.org/merge-sort/>
* <http://algorytmy.ency.pl/artykul/sortowanie_przez_scalanie>
* <https://www.geeksforgeeks.org/heap-sort/>
* <https://www.geeksforgeeks.org/quick-sort/>
* <https://www.geeksforgeeks.org/introsort-or-introspective-sort/>
* <https://pl.wikipedia.org/wiki/Sortowanie_introspektywne>
* <https://www.youtube.com/watch?v=HFpo5ox11aA&list=PL0NHOHZVAjfCioTcerGybx123MRMFBS5w&index=8&t=286s>
* <https://www.youtube.com/watch?v=iJyUFvvdfUg&list=PL0NHOHZVAjfCioTcerGybx123MRMFBS5w&index=9&t=590s>
* <https://www.youtube.com/watch?v=cKJx2nrAmx0&list=PL0NHOHZVAjfCioTcerGybx123MRMFBS5w&index=10&t=0s>
* <https://www.youtube.com/watch?v=M2bKENbdnI4&list=PL0NHOHZVAjfCioTcerGybx123MRMFBS5w&index=11&t=23s>
* <https://www.youtube.com/watch?v=82XxdhRCMbI&list=PL0NHOHZVAjfCioTcerGybx123MRMFBS5w&index=12&t=1s>
* <https://www.youtube.com/watch?v=SKmsk2Rd7oE&list=PL0NHOHZVAjfCioTcerGybx123MRMFBS5w&index=13&t=439s>



Takie wykresy jeszcze trzeba porobić, ale mi się już nie chce. Jak zrobisz szybciej to podeślij