PROJEKTOWANIE ALGORYTMÓW I METODY SZTUCZNEJ INTELIGENCJI – ALGORYTMY SORTOWANIA

Wtorek 15:15

Rafał Rzewucki 248926

Prowadzący:

Mgr inż. Marcin Ochman

# Wstęp

Celem projektu, było zaimplementowanie wybranych algorytmów w języku C++ oraz analiza ich efektywności. Wybrano 3 algorytmy: sortowanie przez scalanie, sortowanie szybkie oraz sortowanie introspektywne. Cechą istotną każdego algorytmu jest jego złożoność obliczeniowa. Przy jej określaniu bierzemy pod uwagę operacje, które mają największy wpływ na czas wykonywania algorytmu, czyli na przykład operacje porównania, operacje arytmetyczne, czy nawet operacje zamiany danych miejscami w tablicy. Określenie złożoności obliczeniowej umożliwia wyznaczenie czasu wykonywania algorytmu dla danej ilości danych niezależnie od komputera, na którym jest on uruchomiony.

Analiza efektywności algorytmów była przeprowadzana dla 100 tablic różniących się rozmiarami, kolejno: 10 000, 50 000, 100 000, 500 000, 1 000 000 elementów.

Poszczególne warianty rozmiarów tablic został dodatkowo rozpatrzony w kilku przypadkach wstępnego posortowania:

* Wszystkie elementy losowe
* 25%, 50% 75%, 95%, 99%, 99,7% posortowanych elementów
* Wszystkie elementy posortowane w odwrotnej kolejności

# Opis zaimplementowanych algorytmów

**Sortowanie przez scalanie** – jest to algorytm, który wykorzystuje zasadę „dziel i zwyciężaj”, dzieli on zbiór danych na mniejsze zbiory aż do uzyskania zbiorów jednoelementowych.

Złożoność czasowa sortowania przez scalanie dla każdego przypadku (najgorszy, średni, najlepszy) wynosi O(nlog(n)). Wynika ona z tego, że głębokość drzewa wywołań funkcji wynosi log(n) dla tablicy o rozmiarze n, przy czym złożoność operacji scalania tablic jest liniowa.

Złożoność pamięciowa algorytmu jest równa O(n), co wynika z faktu, że w pamięci operacyjnej jest potrzebne dodatkowe miejsce na tymczasowe tablice tworzone przy scalaniu.

Należy również zauważyć, że algorytm sortowania przez scalanie jest stabilny, oznacza to, że w przypadku takich samych elementów w zbiorze wejściowym zostanie zachowana ich kolejność w zbiorze wyjściowym.

**Sortowanie szybkie** – podobnie jak w algorytmie sortowania przez scalanie, ten algorytm dzieli zbiór danych na dwa fragmenty. Wybierany jest element rozdzielający (pivot), który rozdziela dane na mniejsze i większe od elementu rozdzielającego. Następnie ponownie wykonuje się sortowanie tablic tak z elementami mniejszymi, jak i większymi od elementu rozdzielającego.

Złożoność obliczeniowa algorytmu sortowania szybkiego zależy od wyboru elementu rozdzielającego:

* Dla przypadku najlepszego złożoność obliczeniowa wynosi O(nlog(n)). Występuje ona wtedy, gdy wybierając element rozdzielający trafimy na medianę z sortowanego fragmentu tablicy.
* Dla przypadku średniego, czyli wtedy, kiedy rozkład prawdopodobieństwa wyboru elementu z tablicy jest równomierny, to złożoność obliczeniowa wynosi O(1,39nlog(n)). Wynika ona z tego, że dla danych losowych mamy 1,39nlog(n) porównań.
* Dla przypadku najgorszego złożoność obliczeniowa wynosi O(n2). Występuje on wtedy, gdy element rozdzielający za każdym razem znajduję się na brzegu tabeli.

Sortowanie szybkie w przeciwieństwie do sortowania przez scalanie nie potrzebuje pamięci na przechowywanie dodatkowych danych.

**Sortowanie introspektywne –** jest to tak zwany algorytm hybrydowy, ponieważ łączy w sobie sortowanie szybkie i sortowanie przez kopcowanie. Ma on na celu zapobiec najgorszemu przypadkowi złożoności obliczeniowej dla sortowania szybkiego. W tym celu bada się maksymalną głębokość rekurencji ze wzoru 𝑀=2log(2𝑛), gdzie n to długość tablicy. Dla M mniejszego bądź równego zero wykonuje się sortowanie przez kopcowanie. Jest to działanie algorytmu, pozwalające uniknąć klasy złożoności obliczeniowej O(𝑛2). Dla M większego od zera sortowanie introspektywne działa podobnie jak sortowanie szybkie. Wywoływana jest procedura, dzieląca tablice na dwa fragmenty o wartościach mniejszych i większych od elementu rozdzielającego. Następnie dla obu tych podzbiorów wykonywana jest rekurencyjnie funkcja sortowania introspektywnego z parametrem M pomniejszonym o 1. Powoduje to, że maksymalna dozwolona głębokość rekurencyjnych wywołań dąży do zera, a gdy je osiągnie, dla pozostałej części tablicy wywoływany jest już algorytm sortowania przez kopcowanie.

Złożoność obliczeniowa sortowania introspektywnego wynosi zawsze O(nlog(n)), ponieważ został wyeliminowany najgorszy przypadek dla sortowania szybkiego.

Sortowanie introspektywne potrzebuje takiej samej ilości pamięci jak sortowanie szybkie.

# 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>