Obraz zawierający tekst, logo, Czcionka, symbol

Opis wygenerowany automatycznie

Imię: **Paweł**

Nazwisko: **Janas**

Nr indeksu: **272939**

Prowadzący: **dr inż. Dariusz Banasiak**

Termin zajęć: **czwartek NP 11:15 – 13:00**

Data utworzenia sprawozdania: **20.04.2024**

**Algorytmy i złożoność obliczeniowa**

**Zadanie projektowe 1**

Temat: „Badanie efektywności wybranych algorytmów sortowania ze względu na złożoność obliczeniową”

**I. Szczegółowy opis zadania projektowego oraz konfiguracja sprzętowa**

W ramach projektu należało wybrać oraz zaimplementować określone algorytmy sortowania w ybranym języku programowania (tutaj C++), a następnie przeprowadzić analizę ich efektywności. Wybrałem wariant na ocenę 5, dlatego badaniu poddałem następujące algorytmy sortowania:

* Przez wstawianie (Insertion Sort)
* Przez kopcowanie (Heap Sort)
* Shella z 2 różnymi wzorami tworzącymi algorytmy o różnych złożonościach
* Szybkie (Quick Sort) ze sposobem podziału pivota na skrajny lewy, skrajny prawy, środkowy oraz losowy

Analiza polegała na zmierzeniu czasów działania poszczególnych algorytmów w zależności od liczby sortowanych elementów tablicy (alokowanej dynamicznie zgodnie z badanym rozmiarem). Pomiary były wykonywane wielokrotnie, a dla celów sprawozdania wyznaczono ich wartości średnie. Analizę przeprowadzono dla 7 reprezentatywnych wartości rozmiaru tablicy oraz dla różnych rozkładów początkowych danych (w zakresie 1 – 30000), tj. dla tablicy całkowicie losowej, tablicy posortowana rosnąco, tablicy posortowanej malejąco i tablicy posortowanej częściowo (33% i 66% początkowych elementów już posortowanych). Elementami sortowanymi były, zgodnie z założeniami projektowymi: 4 bajtowa liczba całkowita ze znakiem (int) oraz liczba zmiennoprzecinkowa o podwójnej precyzji (double), gdzie dla typu double mierzono czas sortowania tylko jednego algorytmu (tutaj dla sortowania przez wstawianie). Dla dokładnego pomiaru czasu sortowania użyto funkcji: *std::chrono::high\_resolution\_clock* z biblioteki <chrono>.

Konfiguracja sprzętowa komputera, na którym przeprowadzano analizę:

Procesor: **AMD Ryzen 5 1600X**

Pamięć: **16 GB pamięci RAM**

System operacyjny: **Microsoft Windows 10 Home**

**II. Analiza algorytmów sortowania**

1. **Sortowanie przez wstawianie (Insertion Sort)**
   1. **Opis algorytmu:**

Algorytm sortowania przez wstawianie polega na umieszczaniu kolejnych elementów w odpowiednich miejscach już posortowanej części zbioru. Rozpoczyna się od uznania pierwszego elementu za posortowany. Następnie, każdy kolejny element jest porównywany z posortowanymi i wstawiany na swoje miejsce. Ten sposób sortowania jest skuteczny, gdy ilość danych jest niewielka.

* 1. **Wyniki pomiarów:**



Dodatkowo przeprowadzona została analiza algorytmu sortowania przez wstawianie dla liczb zmiennoprzecinkowych podwójnej precyzji (double):



* 1. Sortowanie przez wstawianie ma złożoność obliczeniową:

- optymistyczną:

- typową:

- pesymistyczną:

1. **Sortowanie przez kopcowanie (Heap Sort)**
   1. **Opis algorytmu:**

Sortowanie przez kopcowanie (Heap Sort) wykorzystuje strukturę zwaną kopcem binarnym. Proces ten dzieli się na dwie fazy:

- Budowa kopca: Przekształcenie danych w kopiec binarny, gdzie największy element jest na szczycie. Przy każdym rozszerzeniu, elementy są reorganizowane, aby spełniały właściwości kopca.

- Sortowanie: Największy element z kopca zamieniany jest z ostatnim, a następnie usuwany, co umieszcza go na końcu posortowanej tablicy. Kopiec jest naprawiany, aby zachować właściwości kopca, a proces jest powtarzany do posortowania całej tablicy.

* 1. **Wyniki pomiarów:**



* 1. **Złożoność obliczeniowa:**

Sortowanie przez kopcowanie ma złożoność obliczeniową:

- optymistyczną:

- typową:

- pesymistyczną:

1. **Sortowanie Shella (Shell Sort)**
   1. **Opis algorytmu:**

Sortowanie Shella to zaawansowana wersja sortowania przez wstawianie, dopuszczająca porównania i zamiany elementów położonych o określoną odległość od siebie. Na początku sortuje on elementy tablicy położone daleko od siebie, a następnie stopniowo zmniejsza odstępy między sortowanymi elementami.

Pomiary wykonano dla 2 różnych wariantów korzystających z innego wzoru na odstęp porównywanych elementów:

- Shella:

- Hibbarda:

* 1. **Wyniki pomiarów:**

Dla wariantu Shella:



Dla wariantu Hibbarda:



* 1. **Złożoność obliczeniowa:**

Sortowanie Shella w wariancie Shella ma złożoność obliczeniową:

- optymistyczną:

- typową:

- pesymistyczną:

Sortowanie Shella w wariancie Hibbarda ma złożoność obliczeniową:

- optymistyczną:

- typową:

- pesymistyczną:

1. **Sortowanie Szybkie (Quick Sort)**
   1. **Opis algorytmu:**

Algorytm szybkiego sortowania polega na wyborze elementu zwanego pivotem, który dzieli tablicę na dwie podtablice: jedna zawiera elementy równie lub mniejsze niż pivot, a druga zawiera elementy większe. Następnie każda z tych podtablic jest rekurencyjnie sortowana oddzielnie. Proces rekurencji kończy się, kiedy podzielona część tablicy składa się z jednego elementu, co wynika z faktu, że tablica jednoelementowa jest uznawana za już posortowaną.

Pomiary wykonano dla 4 wariantów wyboru pivota:

- skrajny lewy

- skrajny prawy

- środkowy

- losowy

* 1. **Wyniki pomiarów:**

Dla lewego skrajnego pivota:



Można na powyższym wykresie wyraźnie zauważyć pesymistyczny przypadek, gdy elementy w tablicy są posortowane rosnąco. Złożoność obliczeniowa jest wtedy zbliżona do . W pozostałych przypadkach złożoność zbliżona jest do .

Dla prawego skrajnego pivota:



Dla środkowego pivota:



Dla losowego pivota:



* 1. **Złożoność obliczeniowa:**

Sortowanie szybkie ma złożoność obliczeniową:

- optymistyczną:

- typową:

- pesymistyczną:

**Bibliografia:**

- <https://eduinf.waw.pl/inf/alg/003_sort/0025.php>

- <https://pl.wikipedia.org/>