Sprawozdanie

Projekt nr 2. z przedmiotu

„Algorytmy i struktury Danych”

Temat

„Zaimplementuj sortowanie bąbelkowe oraz sortowanie grzebieniowe.”

Wiktor Misiaszek, PRz semestr 2020/2021,

Inżynieria i Analiza Danych

Spis treści

[1. Podstawy teoretyczne algorytmów sortowania 4](#_Toc61463915)

[1.1 Sortowanie bąbelkowe 4](#_Toc61463916)

[1.1.1 Opis algorytmu 4](#_Toc61463917)

[1.1.2 Sortowanie 4](#_Toc61463918)

[1.1.3 Złożoność obliczeniowa 5](#_Toc61463919)

[1.2 Sortowanie grzebieniowe 5](#_Toc61463920)

[1.2.1 Opis algorytmu 5](#_Toc61463921)

[1.2.2 Sortowanie 5](#_Toc61463922)

[1.2.3 Złożoność obliczeniowa 6](#_Toc61463923)

[2. Schematy blokowe 6](#_Toc61463924)

[2.1 Sortowanie bąbelkowe 6](#_Toc61463925)

[2.2 Sortowanie grzebieniowe 7](#_Toc61463926)

[3. Pseudokod 8](#_Toc61463927)

[3.1 Sortowanie bąbelkowe 8](#_Toc61463928)

[3.2 Sortowanie grzebieniowe 8](#_Toc61463929)

[4. Testy porównujące działanie algorytmów 9](#_Toc61463930)

[4.1 Liczby pseudolosowe 9](#_Toc61463931)

[4.2 Przypadek optymistyczny 10](#_Toc61463932)

[11](#_Toc61463933)

[4.3 Przypadek pesymistyczny dla obu sortowań 11](#_Toc61463934)

[5. Wnioski 12](#_Toc61463935)

# Podstawy teoretyczne algorytmów sortowania

## Sortowanie bąbelkowe

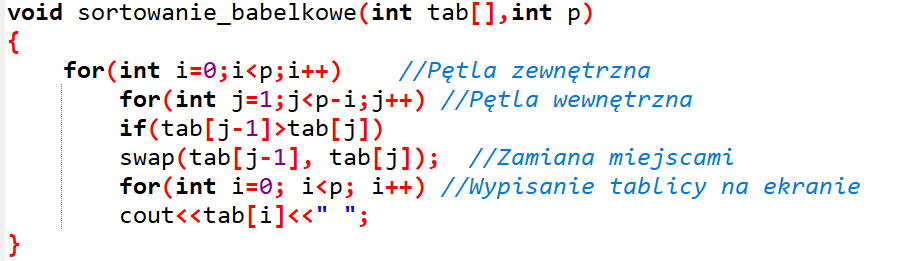
Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących. Można go potraktować jako ulepszenie opisanego w poprzednim rozdziale algorytmu sortowania głupiego. Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany

### 1.1.1 Opis algorytmu

Sortowanie wykonywane jest w dwóch zagnieżdżonych pętlach. Pętla zewnętrzna nr 1 kontrolowana jest przez zmienną *j*. Wykonuje się ona *n* - 1 razy. Wewnątrz pętli nr 1 umieszczona jest pętla nr 2 sterowana przez zmienną *i*. Wykonuje się również *n* - 1 razy. W efekcie algorytm wykonuje w sumie: obiegów pętli wewnętrznej, po których zakończeniu zbiór zostanie posortowany.

### 1.1.2 Sortowanie

Sortowanie odbywa się wewnątrz pętli nr 2. Kolejno porównywany jest *i*-ty element z elementem następnym. Jeśli elementy te są w złej kolejności, to zostają zamienione miejscami. W tym miejscu jest najważniejsza różnica pomiędzy algorytmem sortowania bąbelkowego a algorytmem sortowania głupiego. Ten drugi w momencie napotkania elementów o złej kolejności zamienia je miejscami i rozpoczyna cały proces sortowania od początku. Algorytm sortowania bąbelkowego wymienia miejscami źle ułożone elementy sortowanego zbioru i przechodzi do następnej pary zwiększając indeks *i* o 1.



Rysunek Sortowanie bąbelkowe

### 1.1.3 Złożoność obliczeniowa

Złożoność obliczeniowa wynosi . Gdzie n jest liczbą elementów

## 1.2 Sortowanie grzebieniowe

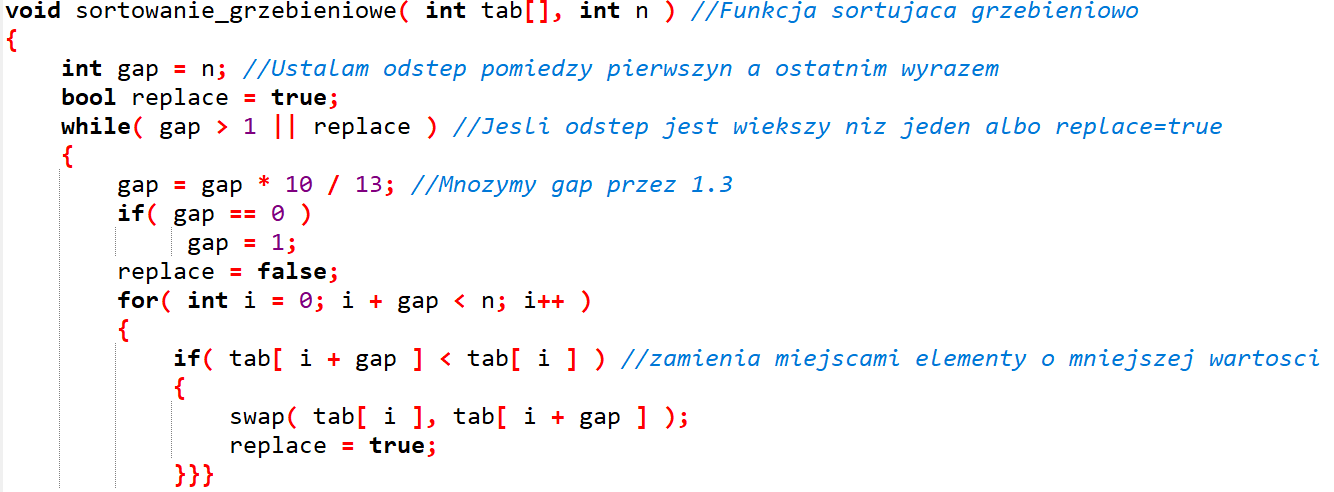
Oparte jest na sortowaniu bąbelkowym. Za rozpiętość (gap) przymuje się wielkość tablicy, którą dzięli się przez 1.3 i odrzuca część ułamkową. Następnie bada się kolejno wszystkie pary obiektów odległych o rozpiętość. Gdy rozpiętość osiągnie wartość 1 sortowanie zostanie zakończone. Do sprawdzenia czy zaszła zmiana podczas sortowania, można użyć dodatkowej zmiennej typu bool, jak w przypadku sortowania bąbelkowego. Przerywane jest wykonywanie algorytmu, gdy podczas przejścia przez całą tablice nie nastąpiła żadna zmiana.

### 1.2.1 Opis algorytmu

Sortowanie wykonywane jest w dwóch zagnieżdżonych pętlach. Pętla zewnętrzna nr 1. wykonuje się dopóki zmienna gap jest większa od 1 albo zmienna replace zmieni wartość na false. W pętli nadawana jest wartość zmiennej gap=1.3, następnie funkcja if odpowiada za sprawdzenie czy gap ma odpowiedni rozmiar. Pętla wewnętrzna nr2. jest kontrolowana przez zmienna i, która na początku ma wartość 0 jest jednak inkrementowana aż osiągnie wartość i+gap<n, gdzie n jest liczba elementów w tablicy.

### 1.2.2 Sortowanie

Do sortowania dochodzi w pętli wewnętrznej for. Wewnątrz niej jest zainkrementowana funkcja if która sprawdza czy wartość liczby stojącej na indexie i+gap jest mniejsza od liczby stojącej na indexie i. Jeśli tak to zamieniane są one miejscami i pętla zostaje przerwana.



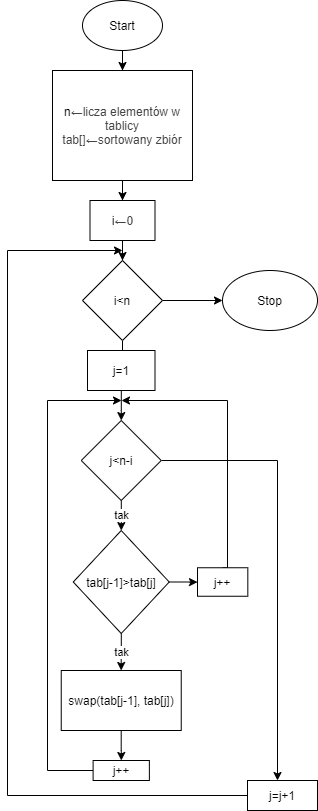
Rysunek Sortowanie Grzebieniowe

### 1.2.3 Złożoność obliczeniowa

Złożoność obliczeniowa tego algorytmu w najlepszym przypadku wynosi pesymistyczny wariant to . Gdzie n jest liczbą elementów.

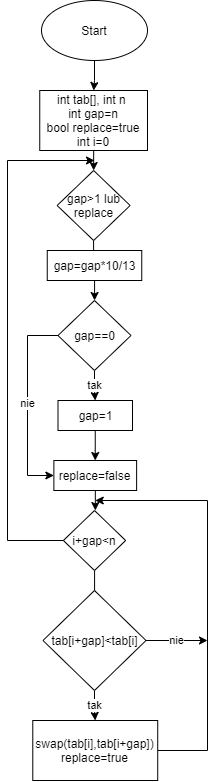
# Schematy blokowe

## 2.1 Sortowanie bąbelkowe



Rysunek Schemat blokowy Sortowanie bąbelkowe

## 2.2 Sortowanie grzebieniowe



# Pseudokod

## 3.1 Sortowanie bąbelkowe

Void Sortowanie\_babelkowe ( tabllica , n)

{

n=liczba\_elementów(tablica)

i=0

Dopóki n>1

Wykonuj:

Dopóki i<n-1

Wykonuj: Jeśli tablica[i]>tablica[i+1] to

Zamień miejscami A[i] z A[i+1]

}

## 3.2 Sortowanie grzebieniowe

void sortowanie\_grzebieniowe ( tablica [] ,n)

{

replace= true

dopóki gap > 1 albo replace

Wykonuj: gap=gap\*10/13

Jeśli gap==0

To gap=0

Replace=false

I=0

Dopóki i+gap<n

Wykonuj: Jeśli tab[i+gap] < tab[i]

To Zamień miejscami T[i] z T[i+gap]

Replace=true

}

# Testy porównujące działanie algorytmów

## 4.1 Liczby pseudolosowe

W pierwszym przypadku algorytm sortuje tablice wypełnioną liczbami pseudolosowymi. Wykorzystane zostały tablice dynamiczne co pozawala na sprawdzanie algorytmów na tablicach małych jak i dużych. Program będzie od nas wymagał podania wielkości tablicy.

Dzięki tablicy wypełnionej liczbami pseudolosowymi otrzymamy oczekiwanązłożoność czasową działania algorytmów.

Tabela Pomiar czasów dla tablic 100-10000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Rozmiar tablicy | 100 | | 1000 | | 10000 | |
| Typ sortowania | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe |
| Pomiar 1 | 0,02s | 0,013s | 0,241s | 0,171s | 1,633s | 1,593s |
| Pomiar 2 | 0,021s | 0,013s | 0,255s | 0,204s | 1,689s | 1,547s |
| Pomiar 3 | 0,02s | 0,014s | 0,301s | 0,156s | 1,701s | 1,564s |
| Średnia pomiarów | 0,0203s | 0,0133s | 0,2656s | 0,177s | 1,6743s | 1,568s |

Jak przedstawia wykres powyżej czasy są porównywalne, wynika to z fakty małego rozmiaru tablicy.

Tabela Pomiary czasów dla tablic 20000-100000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Rozmiar tablicy | 20000 | | 50000 | | 100000 | |
| Typ sortowania | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe |
| Pomiar 1 | 3,902 | 3,157 | 11,2253 | 10,609 | 20,512 | 18,884 |
| Pomiar 2 | 3,821 | 3,415 | 11,115 | 10,601 | 21,214 | 19,142 |
| Pomiar 3 | 3,802 | 3,213 | 11,632 | 10,355 | 20,701 | 18,924 |
| Średnia pomiarów | 3,8416 | 3,2617 | 11,3333 | 10,5217 | 20,809 | 18,9833 |

W tym przypadku widać, że wyniki się rozjeżdżają, wynika to z różnej złożoności obliczeniowej algorytmów.

4.2 Przypadek optymistyczny  
 Optymistyczną złożoność czasową otrzymamy wtedy, gdy tablica, którą trzeba posortować, jest już posortowana. Do tego celu użyjemy możliwości pobrania danych do posortowania z pliku tekstowego.

Tabela Pomiary czasów dla tablic 1000-100000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Rozmiar tablicy | 1000 | | 20000 | | 100000 | |
| Typ sortowania | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe |
| Pomiar 1 | 0,21s | 0,156s | 4,21s | 3,031 s | 21,32s | 16,5s |
| Pomiar 2 | 0,213s | 0,187s | 4,345s | 2,984s | 21,127s | 16,537s |
| Pomiar 3 | 0,221s | 0,156s | 4,113s | 2,996s | 22,132s | 16,524s |
| Średnia pomiarów | 0,2146s | 0,1663s | 4,2226s | 3,00367s | 21,526 | 16,5203s |

## 

Gdy tablice są posortowane to czas pracy znacznie się zmniejszył co widać na wykresie wyżej.

## 4.3 Przypadek pesymistyczny dla obu sortowań

Pesymistyczną złożoność czasową otrzymamy wtedy, gdy tablica, którą trzeba posortować, jest posortowana odwrotnie. Do tego celu użyjemy możliwości pobrania danych do posortowania z pliku tekstowego.

Tabela Pomiary czasów dla tablic 1000-100000

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Rozmiar tablicy | 1000 | | 20000 | | 100000 | |
| Typ sortowania | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe | Bąbelkowe | Grzebieniowe |
| Pomiar 1 | 0,16s | 0,156s | 3,982s | 3,981s | 19,611s | 19,653s |
| Pomiar 2 | 0,161s | 0,156s | 4s | 3,99s | 19,676s | 19,741s |
| Pomiar 3 | 0,15s | 0,156s | 3,893s | 3,894s | 19,932s | 19,541s |
| Średnia pomiarów | 0,157s | 0,156s | 3,958s | 3,955s | 19,739s | 19,645s |

Jak widać pesymistyczny czas obliczeniowy jest praktycznie taki sam dla obu algorytmów.

# Wnioski

Projekt został zrealizowany, działanie algorytmu jest poprawne. W programie główny algorytm został zaimplementowany w osobnej funkcji, która jest wywoływana w późniejszych etapach działania programu. Projekt posiada możliwość odczytu z pliku tekstowego jak i późniejsze zapisanie wyników do pliku tekstowego. W programie zostały umieszczone stosowane komentarze, które pomagają w zrozumieniu kodu.