ALGORYTMY

I

STRUKTURY DANYCH

Projekt inżynierski

Autor: Dominik Czajka

Numer indeksu: 173131

Temat: Zaimplementuj algorytm sortowania grzebieniowego oraz algorytm sortowania poprzez wybór

Opiekun pracy: dr inż. Mariusz Borkowski, prof. PRz

Rzeszów, 2022

Spis treści

Temat projektu

Porównanie działania, budowy, wydajności sortowania grzebieniowego oraz poprzez wybór, w przypadku różnych wartości elementów, a także różnych rodzajów danych

Opis algorytmów

Sortowanie grzebieniowe

Sortowanie grzebieniowe to algorytm sortowania, w którym dane są porównywane za pomocą kilku pasm grzebienia. Każde pasmo składa się z elementów, które są sortowane w kolejnym kroku algorytmu. Sortowanie grzebieniowe jest często używane do sortowania dużych ilości danych, ponieważ jest wydajne i łatwe w implementacji.

Podczas sortowania grzebieniowego elementy są porównywane za pomocą kilku grzebieni, zaczynając od najgrubszego, aż do najcieńszego. Dzięki temu dane są sortowane w sposób iteracyjny, co znacznie zwiększa wydajność algorytmu.

Aby zaimplementować sortowanie grzebieniowe, należy utworzyć kilka grzebieni o różnej liczbie zębów. Następnie należy przejść przez wszystkie elementy w zbiorze danych i umieścić je w odpowiednim grzebieniu w zależności od ich wartości. Gdy wszystkie elementy zostaną umieszczone w grzebieniach, należy przejść przez grzebienie po kolei i wyciągnąć elementy w kolejności rosnącej.

Sortowanie grzebieniowe jest wydajną metodą sortowania, ale ma pewne ograniczenia. Na przykład nie jest dobrze dostosowane do sortowania danych o dużych wartościach, ponieważ wymaga dużej liczby grzebieni. Ponadto wymaga dużo pamięci, ponieważ każdy grzebień musi mieć odpowiednią liczbę elementów. Dlatego zazwyczaj jest używane tylko w określonych sytuacjach, gdy inne metody sortowania nie są wystarczająco wydajne.

Sortowanie poprzez wybór

Sortowanie przez wybór to algorytm sortowania, w którym dane są posortowane przez wybieranie najmniejszego elementu z całego zbioru danych i przenoszenie go na początek zbioru. Ten proces jest powtarzany dla każdego elementu w zbiorze, aż wszystkie elementy zostaną posortowane.

Sortowanie przez wybór jest bardzo prostym algorytmem, ale jest również dość wolnym w porównaniu z innymi metodami sortowania, takimi jak sortowanie przez scalanie czy sortowanie szybkie. Jego złożoność obliczeniowa wynosi O(n^2), co oznacza, że czas sortowania rośnie kwadratowo wraz ze wzrostem liczby elementów w zbiorze danych.

Jednak sortowanie przez wybór ma pewne zalety w porównaniu do innych algorytmów sortowania. Na przykład jest dość łatwe do zaimplementowania i nie wymaga dużo pamięci, ponieważ nie wymaga dodatkowych tablic do przechowywania danych podczas sortowania. Dlatego jest często używane w sytuacjach, w których dane są małe lub wymagana jest prosta implementacja.

Aby zaimplementować sortowanie przez wybór, należy przejść przez każdy element w zbiorze danych i wybrać najmniejszy element. Następnie należy zamienić miejscami wybrany element z pierwszym elementem w zbiorze danych. Ten proces jest powtarzany dla pozostałych elementów w zbiorze, aż wszystkie elementy zostaną posortowane.

Pseudokod programu:

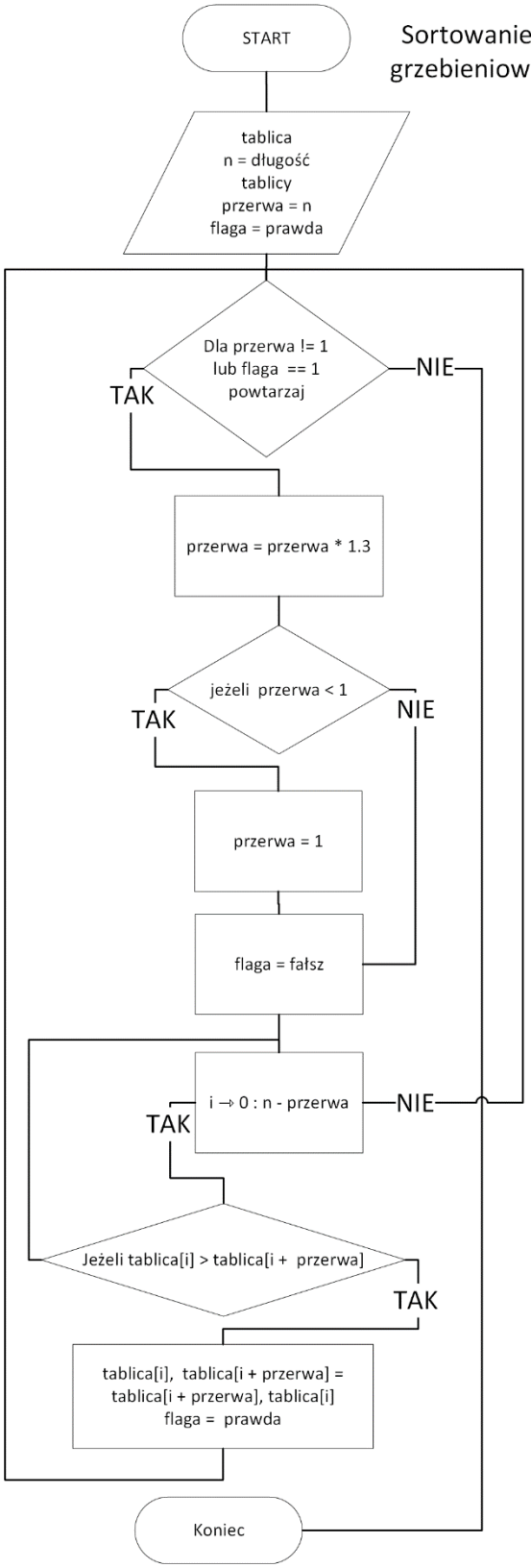
Algorytm sortowania grzebieniowego

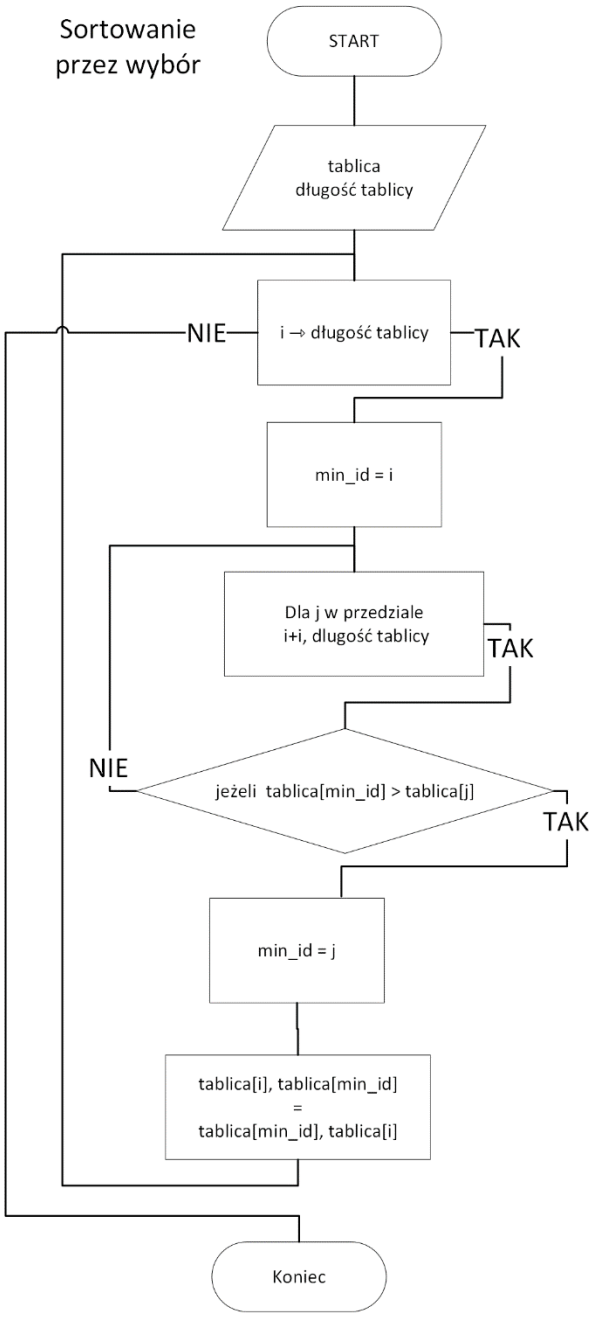
1. Pobierz długość tablicy przechowującej dane do posortowania (tabA) i zapisz ją jako zmienną n.
2. Ustaw zmienną gap na wartość n.
3. Ustaw zmienną swapped na wartość True.
4. Utwórz pętlę, która będzie się wykonywać, gdy zmienna gap jest różna od 1 lub zmienna swapped jest równa True:
    - Oblicz nową wartość zmiennej gap jako iloczyn gap i 10, podzielony przez 13.
    - Jeżeli nowa wartość zmiennej gap jest mniejsza niż 1, zwróć 1 i zakończ działanie funkcji.
    - Ustaw zmienną swapped na wartość False.
        - Utwórz pętlę, która będzie się wykonywać dla każdego indeksu od 0 do n - gap: Jeżeli element o indeksie i jest większy od elementu o indeksie i + gap, zamień je miejscami.
        - Ustaw zmienną swapped na wartość True.
5. Zakończ działanie funkcji.

Algorytm sortowania przez wybieranie:

1. Przejdź przez każdy indeks i w tablicy tabB:
2. Ustaw zmienną min_id na i.
3. Przejdź przez każdy indeks j w zakresie od i+1 do końca tablicy:
    - Jeśli element pod indeksem min_idx jest większy od elementu pod indeksem j, ustaw min_idx na j.
4. Jeśli indeks min_idx jest różny od i, zamień elementy pod indeksami „i” i min_id.
5. Zakończ działanie algorytmu.

Schematy blokowe





Rysunek 1 Schematy blokowe

Działanie Programu

Ogólnie o programie

Uruchomiony program wita użytkownika menu z polem wyboru, jakie dane, oraz jak dużą listę danych ma wygenerować lub czy ma czytać dane z pliku, następnie uruchamiana jest odpowiednia funkcja pobierania danych. W kolejnym kroku pobrane dane są przekazywane do funkcji która pozwala na zliczenie czasu działania algorytmów. Uruchamia ona algorytmy sortowania. Końcowo program wpisuje otrzymane dane w terminalu, a także zapisuje je do pliku

Złożoność obliczeniowa

Sortowanie grzebieniowe

Złożoność algorytmu sortowania grzebieniowego wynosi O(n^2) w najgorszym przypadku i O(n log n) w przypadku przeciętnym.

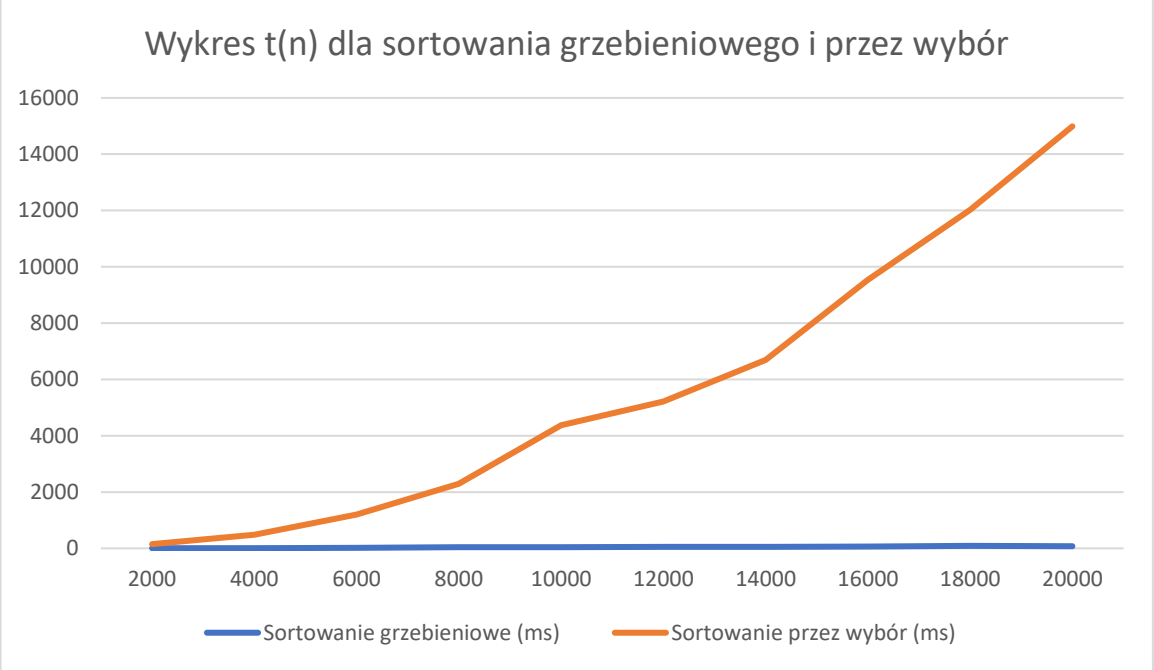
W najgorszym przypadku algorytm zachowuje się jak sortowanie bąbelkowe, co oznacza, że porównuje i potencjalnie zamienia sąsiednie elementy w tablicy wejściowej w każdej iteracji zewnętrznej pętli. Ponieważ zewnętrzna pętla działa przez maksymalnie n-1 iteracji, a wewnętrzna pętla działa przez maksymalnie n-1 iteracji w każdej iteracji zewnętrznej pętli, całkowita liczba porównań i zamian wykonanych w najgorszym przypadku wynosi (n -1) * (n-1) = n^2 - 2n + 1, czyli O(n^2).

W przeciętnym przypadku algorytm działa znacznie lepiej niż sortowanie bąbelkowe, ponieważ wykorzystuje malejącą lukę między porównywanymi elementami w każdej iteracji zewnętrznej pętli. Oznacza to, że wewnętrzna pętla działa przez mniej iteracji w każdej iteracji zewnętrznej pętli, a całkowita liczba przeprowadzonych porównań i zamian jest mniejsza. Dokładna liczba porównań i zamian wykonywanych w przypadku średnim zależy od sekwencji przerw używanej przez algorytm, ale zazwyczaj jest znacznie mniejsza niż n^2, co oznacza, że złożoność algorytmu w przypadku średnim wynosi O(n log n).

Należy zauważyć, że algorytm sortowania grzebieniowego nie jest stabilnym algorytmem sortowania, co oznacza, że może nie zachowywać względnej kolejności elementów o tej samej wartości w tablicy wejściowej. Na przykład, jeśli tablica wejściowa to [2, 3, 1, 3, 2], to tablica wyjściowa utworzona przez algorytm sortowania grzebieniowego może mieć postać [1, 2, 2, 3, 3] lub [2, 2, 1 , 3, 3], w zależności od dokładnej sekwencji porównań i zamian wykonywanych przez algorytm. W przeciwieństwie do tego stabilny algorytm sortowania, taki jak sortowanie przez scalanie lub sortowanie przez wstawianie, w tym przypadku zawsze generowałby tę samą tablicę wyjściową, a mianowicie [1, 2, 2, 3, 3].

Sortowanie przez wybór:

- Złożoność obliczeniowa sortowania przez wybór wynosi O(n^2), co oznacza, że czas sortowania rośnie kwadratowo wraz ze wzrostem liczby elementów w zbiorze danych. To oznacza, że sortowanie przez wybór jest dość wolnym algorytmem w porównaniu z innymi metodami sortowania, takimi jak sortowanie przez scalanie czy sortowanie szybkie. Dlatego sortowanie przez wybór jest często używane tylko w sytuacjach, gdy dane są małe lub gdy wymagana jest prosta implementacja.



Rysunek 2 Wykres t(n) algorytmów

Podsumowanie:

- Sortowanie grzebieniowe jest nieporównywalnie szybsze
- Rodzaj danych (ciąg zer i jedynek, ciąg tych samych liczb, zakres liczb, a także to czy są zmiennno-przecinkowe) nie wpływa w sposób zauważalny na czas sortowania

Kod programu

```
import random  # Biblioteka losowania
import time  # Biblioteka sluzaca do pomiaru czasu

############## Algorytmy ##############

def sortowanieGrzebieniowe(tabA):
    n = len(tabA)
    gap = n # przypisanie przerwy pomiedzy grzebieniami sortowania
    swapped = True
    while gap != 1 or swapped == 1:
        # oblicza przerwe pomiedzy zamienianymi elementami
        gap = (gap * 10) // 13
        if gap < 1:
            return 1
        swapped = False
        # porównywanie elementu itego z itym + przerwa i zamiana
        for i in range(0, n - gap):
            if tabA[i] > tabA[i + gap]:
                tabA[i], tabA[i + gap] = tabA[i + gap], tabA[i]
                swapped = True

def sortowaniePoprzezWybieranie(tabB):
    for i in range(len(tabB)):
        # Znalezienie najmniejszego elementu
        min_idx = i
        for j in range(i + 1, len(tabB)):
            if tabB[min_idx] > tabB[j]:
                min_idx = j
        # zamiana najmniejszego elementu z pierwszym elementem
        tabB[i], tabB[min_idx] = tabB[min_idx], tabB[i]
```

```
#####################################################################

def sortowanie(poczatkowa):
    tabA = poczatkowa
    tabB = poczatkowa

    with open('WynikTablicaPoczatkowa.txt', 'w') as plik:
        plik.write("Tablica poczatkowa:\n")
        for poczatkowa_el in poczatkowa:
            plik.write(str(poczatkowa_el) + '\n')

    startA = time.time()
    sortowanieGrzebieniowe(tabA)
    czasA = str(round(((time.time() - startA) * 1000), 8)) + " ms"
    # Wyswietlanie danych w konsoli
    print("Sortowanie grzebieniowe\nCzas: ", czasA, "\n", tabA)
    startB = time.time()
    sortowaniePoprzezWybieranie(tabB)
    czasB = str(round(((time.time() - startB) * 1000), 8)) + " ms"
    # Wyswietlanie danych w konsoli
    print("\nSortowanie poprzez wybieranie\nCzas: ", czasB, "\n", tabB)
    zapis(tabA, tabB, czasA, czasB)

def zapis(tabA, tabB, czasA, czasB):
    with open("Wynik.txt", 'w') as plik:
        # Dla każdego elementu w tablicach
        i = 0
        plik.write("Tablica początkowa, Sortowanie Grzebieniowe,  Sortowanie poprzez wybór\n" +
                   "czas sortowania: " + czasA + "| czas sortowania: " + czasB + "\n")

        for tabA_el, tabB_el in zip(tabA, tabB):
            i = i + 1
            # Konwertujemy elementy na ciągi znaków i zapisujemy
            # je do pliku w formacie "tabA[i] | tabB[i]"
            plik.write("   Element nr " + str(i) + ": "
                       + str(tabA_el) + ' | ' + str(tabB_el) + '\n')
        plik.write("\n\nProgram napisany z wykorzystaniem wersji:" +
                   "\n Python 3.10.6 (Linux)")
```

```
def samodzielneWprowadzanieDanych():
    poczatkowa = []
    min = int(input("\nPodaj wartosc minimalna losowanej liczby\n"))
    max = int(input("\nPodaj wartosc maksymalna\n"))
    elementy = int(input("Podaj ilosc elementow\n"))
    # losowanie elementow tablicy
    for i in range(elementy):
        poczatkowa.append(random.randint(min, max))
    print("Tablica poczatkowa:\n", poczatkowa, "\n")
    sortowanie(poczatkowa)

def losowanieNiecalkowitychElementow():
    poczatkowa = []
    min = float(input("\nPodaj wartosc minimalna losowanej liczby\n"))
    max = float(input("Podaj wartosc maksymalna\n"))
    elementy = int(input("Podaj ilosc elementow\n"))
    # losowanie elementow tablicy
    for i in range(elementy):
        poczatkowa.append(random.uniform(min, max))
    # wypisywanie tablicy poczatkowej
    print("Tablica poczatkowa", poczatkowa, "\n")
    sortowanie(poczatkowa)

def daneZpliku():
    poczatkowa = []
    with open('tablica.txt', 'r') as plik:
        for i, line in enumerate(plik):
            poczatkowa.append(line)
            poczatkowa[i] = int(poczatkowa[i])
    print("Tablica początkowa\n", poczatkowa, "\n")
    sortowanie(poczatkowa)

def wprowadzanieDanych(metoda_wprowadzania):
    metoda_wprowadzania = input(
        "Podaj metode wprowadzania danych\n " +
        "1: Wprowadzanie wartosci dla losowanych " +
        "liczb calkowitych \n 2: " +
```

```
        "Wprowadzanie danych niecalkowitych \n " +
        "3: Wprowadzanie danych z pliku\n\n")
    slicz(metoda_wprowadzania)

def slicz(metoda_wprowadzania):
    match metoda_wprowadzania:
        case "1":
            samodzielneWprowadzanieDanych()
        case "2":
            losowanieNiecalkowitychElementow()
        case "3":
            daneZpliku()
        case _:
            print("Poprawnie wprowadz dane")
            wprowadzanieDanych(metoda_wprowadzania)

##########################################################################
############################ MAIN ##################################

metoda_wprowadzania = 0
print("Sortowanie poprzez wybor, oraz grzebieniowe\n")
wprowadzanieDanych(metoda_wprowadzania)
```

Spis rysunków i wykresów