PROJEKT 2

Sortowanie bąbelkowe.

Sortowanie przez zliczanie

Albert Gawin

15.01.2020

AiSD, Projekt 2

WPROWADZENIE

Program wykorzystuje funkcje sortujące (bąbelkowe, przez zliczanie) tablice zapisane w plikach tekstowych. Oblicza również czasy sortowania obu metod aby na ich podstawie można było porównać złożoność czasową obu sposobów sortowania.

OPIS ALGORYTMÓW

Sortowanie bąbelkowe

Polega na porównywaniu par liczb oraz zamianie ich jeżeli nie są ułożone w odpowiedniej kolejności. Koniec sortowania zachodzi wtedy kiedy porównano wszystkie możliwe pary w tablicy.

Z powodu porównywania każdej z par złożoność czasowa tego algorytmu wynosi **O(n²)** natomiast złożoność pamięciowa wynosi **O(1)**. Oznacza to, że algorytm ten jest bardzo wolny dla większych próbek. Jego plusami są niskie zapotrzebowanie pamięciowe oraz prostota.

W każdym **kroku** algorytm porównuje następne pary liczb. Na koniec **kroku** ostatnia liczba jest liczbą posortowaną. Przykład opisany dla tablicy **[4, 2, 5, 1, 7]**.

KROK 1 [4, 2, 5, 1, 7] -> [2, 4, 5, 1, 7] -> [2, 4, 5, 1, 7] -> [2, 4, 1, 5, 7]

KROK 2 [2, 4, 1, 5, 7] -> [2, 4, 1, 5, 7] -> [2, 1, 4, 5, 7]

KROK 3 [2, 1, 4, 5, 7] -> [1, 2, 4, 5, 7]

KROK 4 [1, 2, 4, 5, 7]

Czerwona para oznacza, że liczby nie są odpowiednio ułożone tj. należy je zamienić miejscami.
Zielona para oznacza, że liczby są odpowiednio ułożone.
Niebieska liczba oznacza że liczba jest posortowana w tablicy.

Sortowanie przez zliczanie

Przelicza ilość wystąpień każdej z liczb w tablicy. Następnie wypisuje każdą z liczb od [min do max] tyle razy ile razy wystąpiła w tablicy. Koniec sortowania zachodzi po wypisaniu każdej z liczb.

Z powodu konieczności utworzenia zapisu ilości wystąpień do tablicy liczników (k) oraz wypisaniu wszystkich liczb do tablicy (n) złożoność czasowa wynosi **O(n+k).** Obie tablice musimy zapisać w pamięci wskutek czego złożoność pamięciowa również wynosi **O(n+k).** Oznacza to, że algorytm ten nie jest najoptymalniejszy jeżeli chodzi o potrzebną pamięć. Jego ogromnym plusem jest fakt, że jest to jeden z najszybszych algorytmów sortujących.

KROK 1: algorytm inicjalizuje tablicę liczników, szukając wcześniej liczby minimalnej oraz maksymalnej, aby następnie przypisać dla każdej z liczb zero wystąpień.

Wyszukiwanie liczb min oraz max ma na celu optymalizację zajmowanej pamięci np. jeżeli istniałaby tablica [123, 52] aby tablica liczników nie zaczynała się od zera tylko od liczby minimalnej, a kończyła na maksymalnej.

KROK 2: algorytm (skanując tablicę od początku do końca) zapisuje ilość wystąpień liczb w tablicy.

KROK 3: liczby z tablicy liczników wypisywane są odpowiednio tyle razy ile występowały w tablicy np. dla 1:2 liczba jeden zostanie wypisane odpowiednio dwa razy. Przykład opisany dla tablicy **[4, 2, 5, 1, 7].**

KROK 1 [1:0, 2:0, 3:0, 4:0, 5:0, 6:0, 7:0]

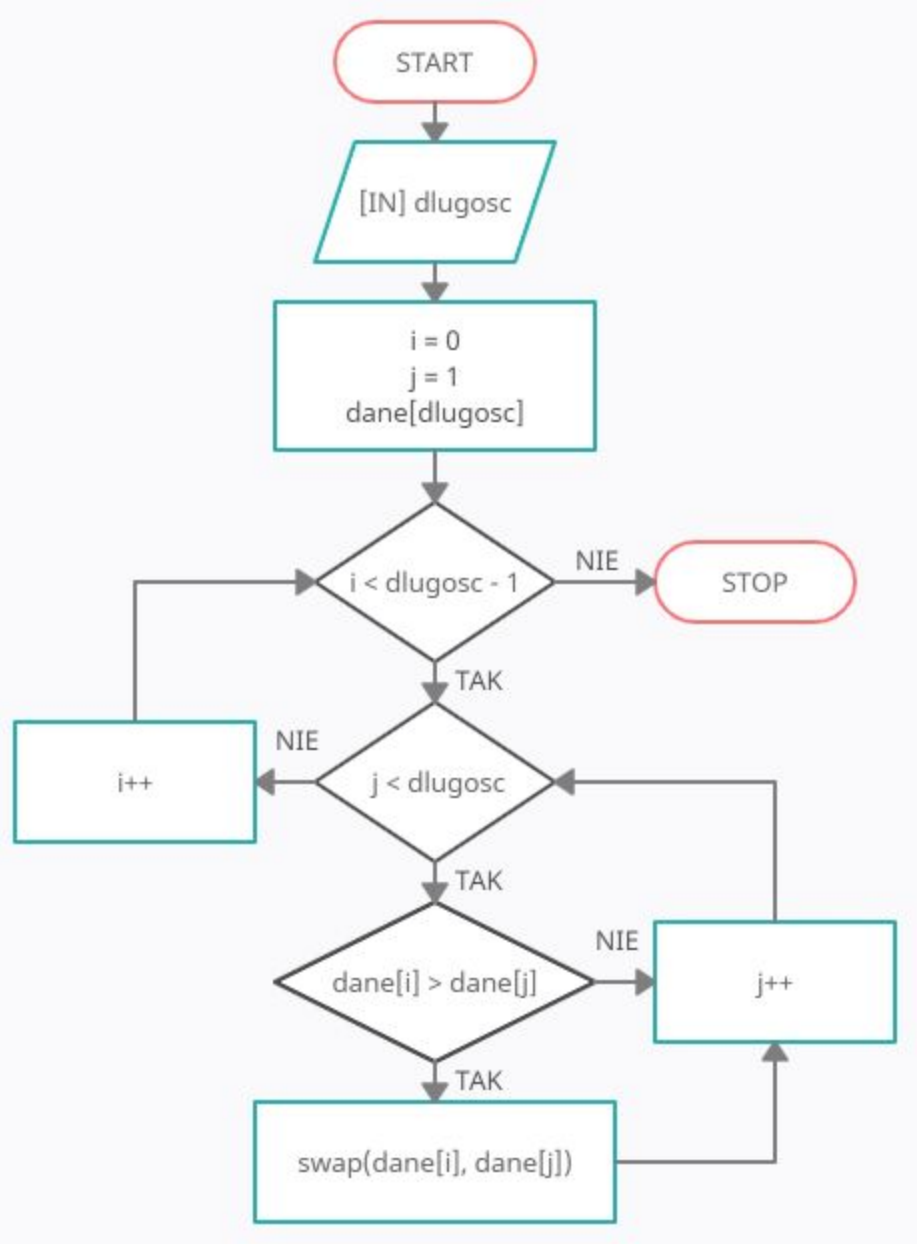
KROK 2 [1:1, 2:1, 3:0, 4:1, 5:1, 6:0, 7:1]

KROK 3 [1, 2, 4, 5, 7]

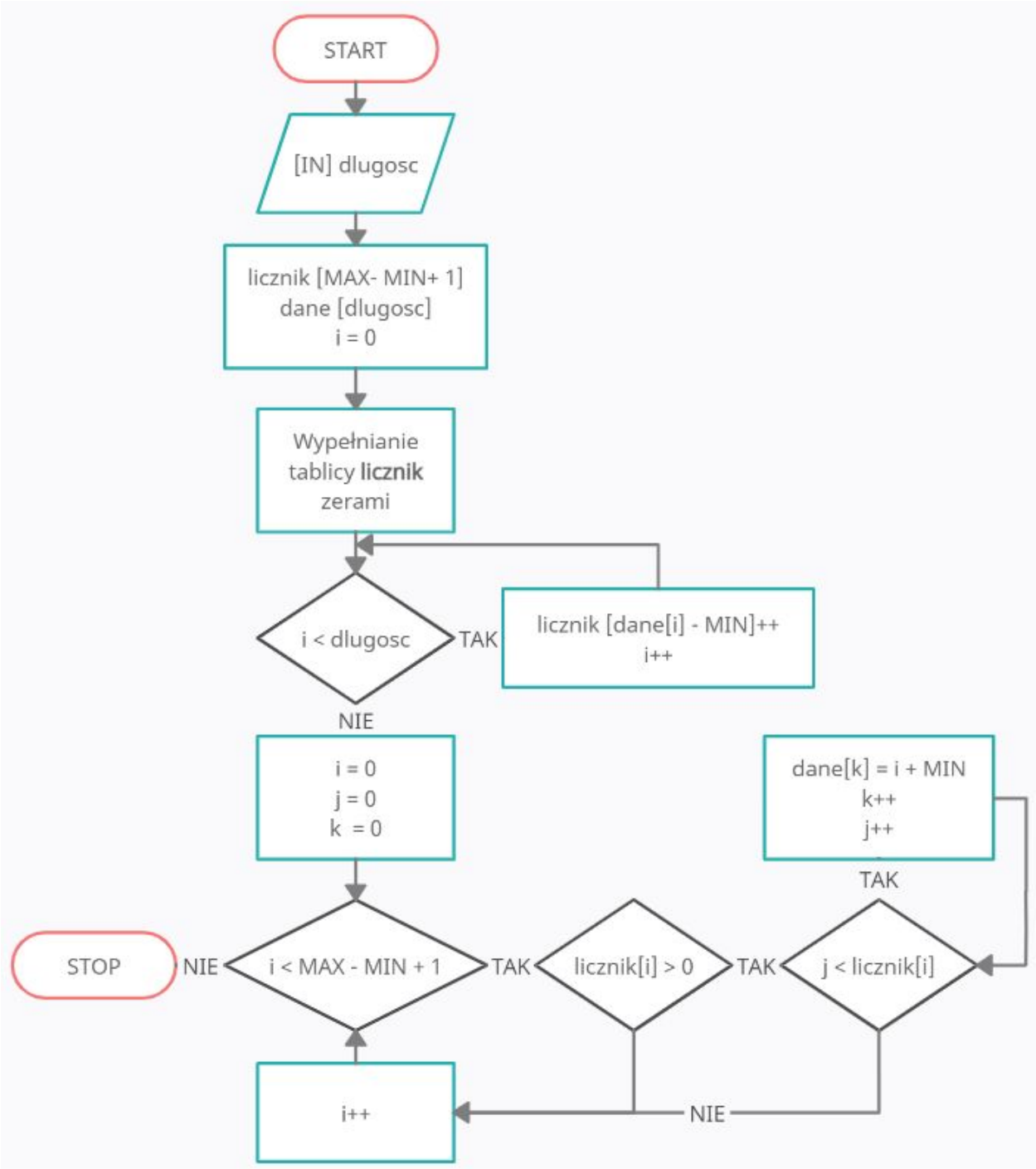
Czerwona oznacza, że liczba nie występuje w tablicy.
Zielona oznacza że liczba liczba występuje więcej niż jeden raz w tablicy.
Niebieska liczba oznacza że liczba jest posortowana w tablicy.

SCHEMATY BLOKOWE

Sortowanie bąbelkowe



Sortowanie przez zliczanie



PSEUDOKOD

Sortowanie bąbelkowe

dlugosc = liczba_elementow(tab)

i = 0

j = 0

KROK 1 JEŻELI (i < dlugosc - 1) **ZRÓB** KROK 2

KROK 2 **JEŻELI** (j < dlugosc) **ZRÓB** KROK 3

KROK 3 **JEŻELI** (dane[i] > dane[j]) **ZRÓB** KROK 4

KROK 4 zamień (dane[i], dane[j])

KROK 5 j++ **PRZEJDŹ** do KROK 2

KROK 6 i++ **PRZEJDŹ** do KROK 1

KONIEC

Sortowanie przez zliczanie

dlugosc = liczba_elementow(tab)

licznik [MAX - MIN + 1]

i = 0

KROK 1 Wypełnienie tablicy licznik zerami

KROK 2 JEŻELI (i < dlugosc) **ZRÓB** KROK 3

KROK 3 licznik[dane[i] - MIN]++

i++

[i = 0, j = 0, k = 0]

KROK 4 JEŻELI (i < MAX - MIN + 1) **ZRÓB** KROK 5

KROK 5 **JEŻELI** (licznik[i] > 0) **ZRÓB** KROK 6

KROK 6 **JEŻELI** (j < licznik[i]) **ZRÓB** KROK 7

KROK 7 dane[k] = i + MIN

k++

j++ **PRZEJDŹ** do KROK 6

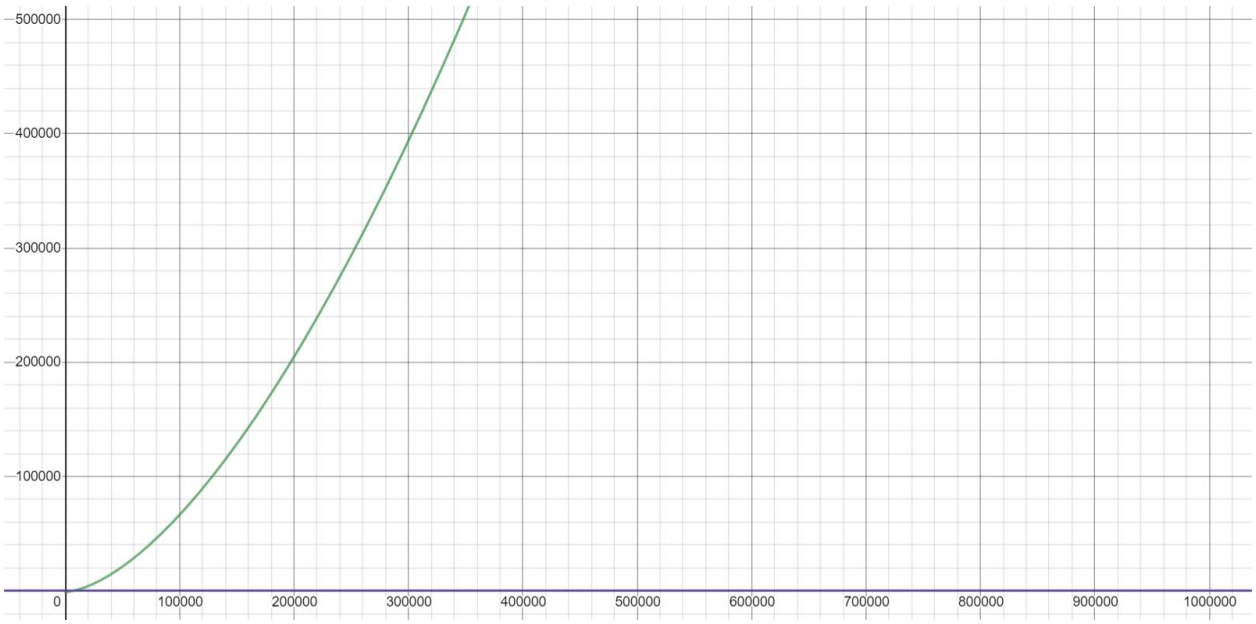
KROK 8 i++ **PRZEJDŹ** do KROK 4

KONIEC

PORÓWNANIE

Dokonałem porównania czasu wykonania obydwu algorytmów. Jak można zauważyć sortowanie przez zliczanie praktycznie pokrywa się z osią OX, ponieważ sam ten algorytm jest najszybszy. Natomiast sortowanie bąbelkowe rośnie kwadratowo co można zauważyć na zamieszczonej grafice.

Testy robione na procesorze AMD Ryzen 5 (4 rdzenie, 8 wątków, 2,1 GHz).



Zielona funkcja - sortowanie bąbelkowe
Fioletowa funkcja - sortowanie przez zliczanie

W znalezieniu funkcji pomógł program do znalezienia funkcji na podstawie próbki danych [1]. Narysowałem funkcję w programie ogólnodostępnym programie do rysowania funkcji [2].

DANE

PRÓBKA DANYCH	SORTOWANIE BĄBELKOWE	SORTOWANIE PRZEZ ZLICZANIE
10^2	0 ms	0 ms
10^3	10 ms	0 ms
10^4	604 ms	0 ms
10^5	40-60 s	2 ms
10^6	około 25 min	38 ms

PODSUMOWANIE

Sortowanie bąbelkowe jest bardzo prostym rozwiązaniem, niestety bardzo zawodnym jeżeli chodzi o czas wykonania. Mając na uwadze czas lepiej byłoby użyć sortowania przez zliczanie. Jeżeli natomiast nie pracujemy na dużych próbkach danych a nie możemy sobie pozwolić na zajmowanie pamięci najlepszym rozwiązaniem będzie sortowanie bąbelkowe.

Jak widać każdy z algorytmów ma swoje plusy i minusy, a to tylko od nas programistów zależy jak je wykorzystamy.

ŹRÓDŁA

[1] Function Equation Finder from Points Table - Online Calculator (dcode.fr)

[2] Desmos | Kalkulator graficzny