Sprawozdanie
Struktury danych i złożoność obliczeniowa

Temat:
Badanie efektywności operacji dodawania, usuwania oraz wyszukiwania elementów w różnych strukturach danych

Politechnika Wrocławska
Dawid Szeląg 264008
Prowadzący: Mgr. inż. Antoni Sterna
24.04.2023r.

Politechnika
Wrocławska

Spis treści

1 Wstęp teoretyczny

Zadanie projektowe polegało na zaimplementowaniu oraz dokonaniu pomiaru czasu działania operacji takich jak:

- dodawanie
- usuwanie,
- wyszukiwanie elementu,

na następujących strukturach danych:

- tablica dynamiczna,
- lista dwukierunkowa,
- kopiec binarny,
- drzewo czerwono-czarne.

Przyjęte założenia w projekcie:

- elementem wszystkich struktur jest 4 bajtowa liczba całkowita ze znakiem;
- wszystkie struktury danych są alokowane dynamicznie i zajmują jak najmniej miejsca (relokacja przy dodawaniu/usuwaniu elementów);
- kopiec zaimplementowany w wariancie z tablicą a nie jako drzewo ze wskaźnikami;
- dla tablicy i listy rozpatrzono osobno operacje dodawania i usuwania elementu na pierwszej pozycji, końcowej oraz dowolnej wybranej;
- dla kopca zastosowano tylko usuwanie elementu ze szczytu;

- pomiary zależności czasu wykonywania poszczególnych operacji, wykonywano **20 razy**, następnie wyliczano z tego średnią ważoną dla następujących rozmiarów:

Tablica, lista oraz kopiec

1. 10	**6**. 50 000	**11**. 2 000 000	**16**. 10 000 000
2. 100	**7**. 100 000	**12**. 3 000 000	**17**. 20 000 000
3. 1 000	**8**. 300 000	**13**. 4 000 000	**18**. 30 000 000
4. 5 000	**9**. 600 000	**14**. 5 000 000	**19**. 40 000 000
5. 10 000	**10**. 1 000 000	**15**. 7 500 000	**20**. 50 000 000

Drzewo czerwono-czarne

1. 10	**6**. 50 000	**11**. 1 500 000	**16**. 5 000 000
2. 100	**7**. 100 000	**12**. 2 000 000	**17**. 7 000 000
3. 1 000	**8**. 300 000	**13**. 2 500 000	
4. 5 000	**9**. 600 000	**14**. 3 000 000	
5. 10 000	**10**. 1 000 000	**15**. 4 000 000	

Liczba elementów została zmieniona ze względu na długi czas wykonywania testów. Wynika to z zapętlonej operacji insert na drzewie, która trwa dłużej niż stworzenie tablicy o określonym rozmiarze oraz stworzenie listy za pomocą addLast.

1.1 Złożoności według literatury

Tablica dynamiczna

Operacja	**Złożoność**
Dodawanie elementu na początek	O(n)
Dodawanie elementu w dowolne miejsce	O(n)
Dodawanie elementu na koniec	O(n)
Usuwanie elementu z początku	O(n)
Usuwanie elementu z dowolnego miejsca	O(n)
Usuwanie elementu z końca	O(n)
Wyszukiwanie elementu	O(n)

Tabela 1: Złożoność obliczeniowa tablicy dynamicznej

Lista dwukierunkowa

Operacja	**Złożoność**
Dodawanie elementu na początek	O(1)
Dodawanie elementu w dowolne miejsce	O(n)
Dodawanie elementu na koniec	O(1)
Usuwanie elementu z początku	O(1)
Usuwanie elementu z dowolnego miejsca	O(n)
Usuwanie elementu z końca	O(1)
Wyszukiwanie elementu	O(n)

Tabela 2: Złożoność obliczeniowa listy dwukierunkowej

Kopiec binarny

Operacja	**Złożoność**
Dodawanie elementu	O($log(n)$)
Usuwanie elementu z wierzchołka	O($log(n)$)
Wyszukiwanie elementu	O(n)

Tabela 3: Złożoność obliczeniowa kopca binarnego

Drzewo czerwono-czarne

Operacja	Złożoność
Dodawanie elementu	O($log(n)$)
Usuwanie elementu	O($log(n)$)
Wyszukiwanie elementu	O($log(n)$)

Tabela 4: Złożoność obliczeniowa drzewa czerwono-czarnego

1.2 Sposoby generowania liczb oraz liczenie czasu

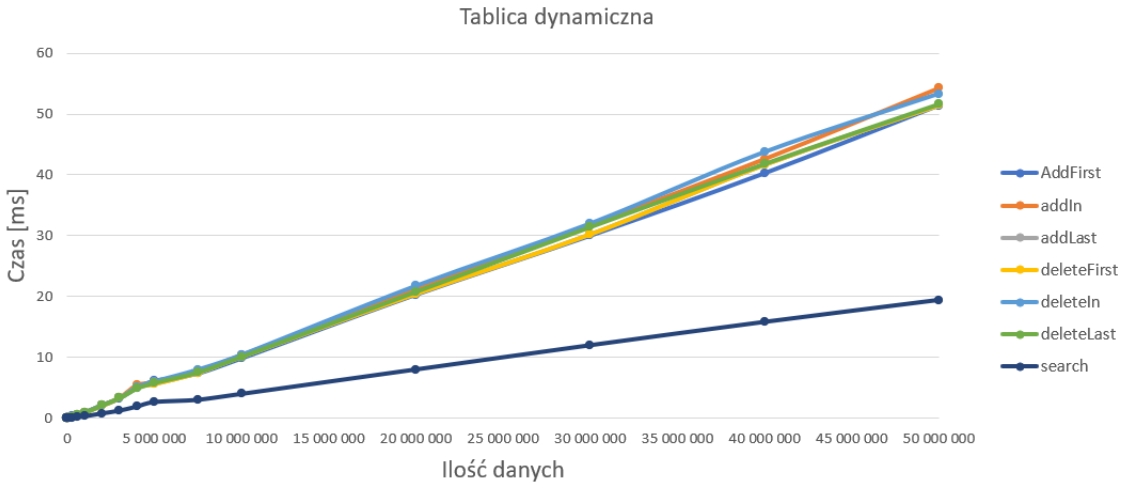
Generowanie liczb pseudolosowych zostało zrealizowane za pomocą generatora mt19937 z biblioteki <random>. Liczby zostały wylosowane w zakresie <-2147483648, 2147483647>.

```
//Genarator.h
#include <random>
#include <iostream>
using namespace std;

class Generator {
    random_device rd;
    mt19937 gen;
    uniform_int_distribution<> dist;
public:
    Generator();
    int getNumber();
    int getNumber(int min, int max);
};

//Genarator.cpp
Generator::Generator():rd(),gen(rd()), dist(INT32_MIN, INT32_MAX)
{}
int Generator::getNumber() {
    return dist(gen);
}
int Generator::getNumber(int min, int max) {
    dist.param(uniform_int_distribution<>::param_type(min,max));
    int valReturn = dist(gen);
    dist.param(uniform_int_distribution<>::param_type(INT32_MIN,INT32_MAX));
    return valReturn;}
}
```

Czas liczony był za pomocą funkcji QueryPerformanceCounter().

```
#include <windows.h>
#include <iostream>
#include <iomanip>
using namespace std;

Timer::Timer() : frequency(0), start(0), elapsed(0)
{
}

void Timer::run() {
    QueryPerformanceFrequency((LARGE_INTEGER *)&frequency);
    start = read_QPC();
}

void Timer::stop() {
    elapsed = read_QPC() - start;
}

long long int Timer::read_QPC()
{
    LARGE_INTEGER count;
    DWORD_PTR oldmask = SetThreadAffinityMask(GetCurrentThread(),
        0);
    QueryPerformanceCounter(&count);
    SetThreadAffinityMask(GetCurrentThread(), oldmask);
    return((long long int)count.QuadPart);
}

double Timer::getTimeMs()
{
    return (1000.0 * elapsed) /frequency;
}
```

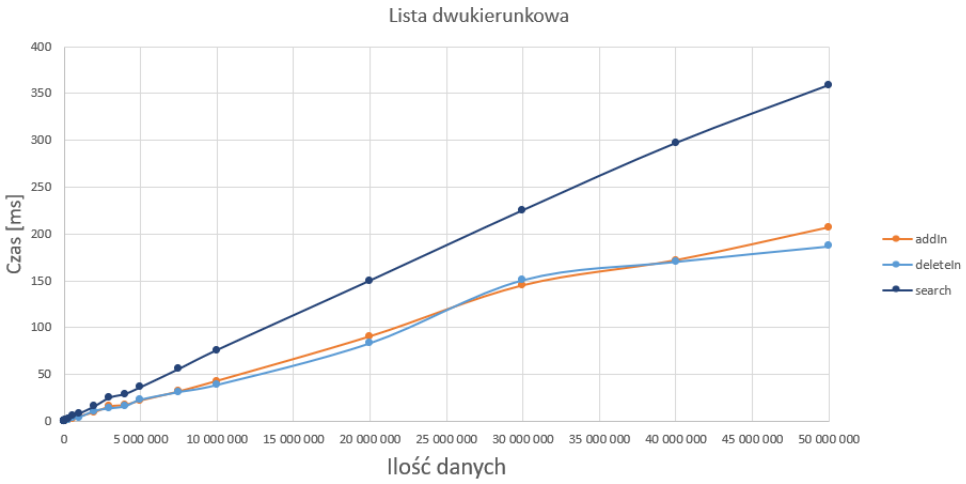
2 Tablica dynamiczna



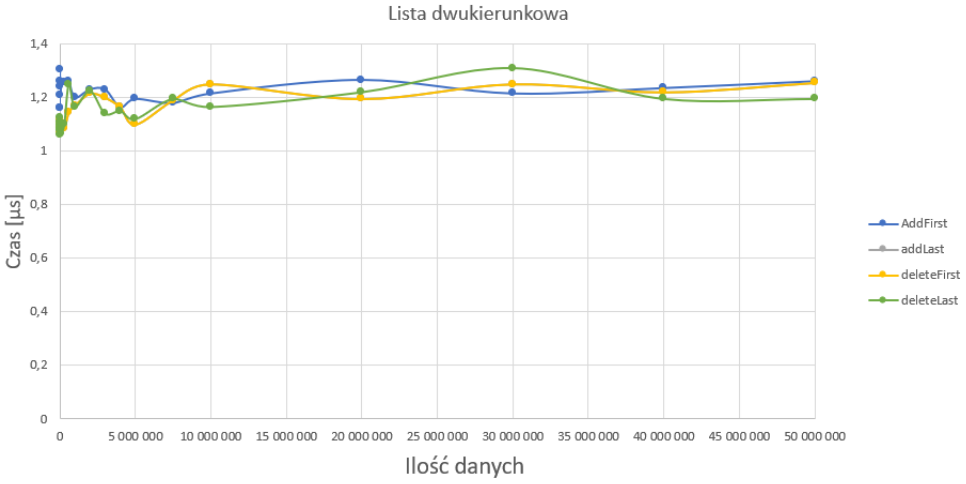
Rysunek 1: Tablica dynamiczna - wykres

Wszystkie pomiary zgadzają się z teoretyczną złożonością O(n), a mianowicie tworzą wykres liniowy.

3 Lista dwukierunkowa



Rysunek 2: Lista dwukierunkowa (addIn, deleteIn, search) - wykres

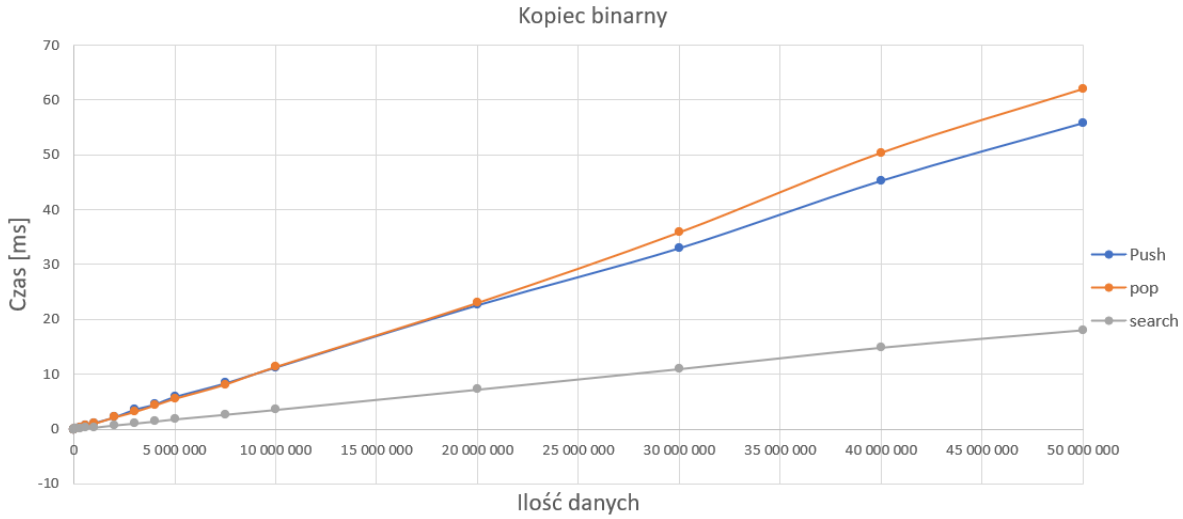


Rysunek 3: Lista dwukierunkowa (addFirst, addLast, deleteFirst, deleteLast) - wykres

Pomiary operacji addFirst, addLast, deleteFirst, deleteLast krążą w okolicach 1,2μs, co zgadza się z zakładaną złożonością O(1).

Pomiary operacji addIn, deleteIn w przybliżeniu mają charakter liniowy, co daje nam złożoność O(n), co jest jak najbardziej prawidłowe. Pomiar operacji search ma dokładny charakter liniowy, co zgadza nam się z zakładaną złożonością O(n).

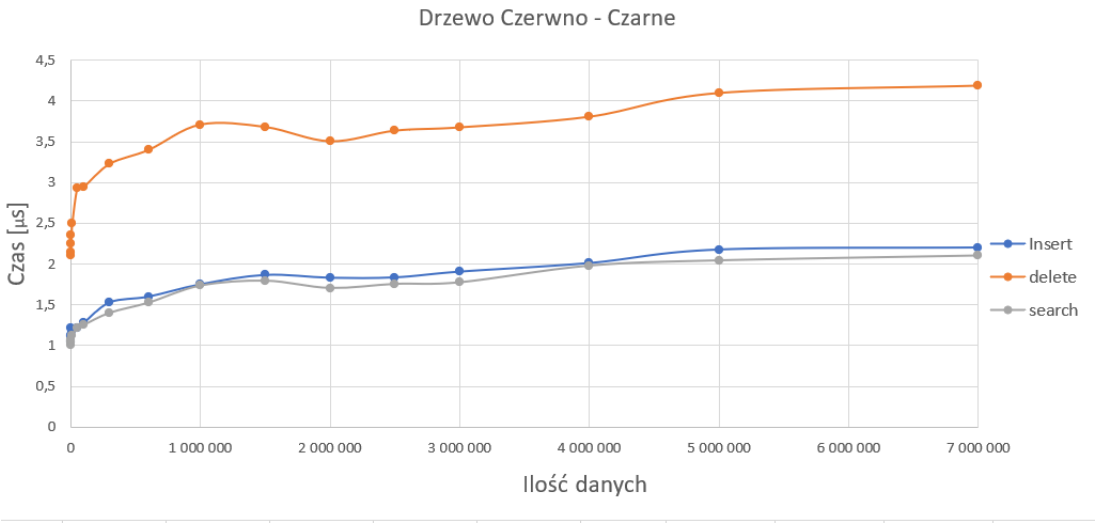
4 Kopiec binarny



Rysunek 4: Kopiec binarny - wykres

Pomiary operacji nie są zgodne z literaturą. Według pomiarów, złożoność obliczeniowa ma charakter O(n), a powinna wynosić O($log(n)$).
Wynika to z założeń projektu. Kopiec binarny jest zaimplementowany jako tablica, która jest relokowana z każdą operacją dodawania i usuwania. Operacja relokacji ma charakter liniowy, który dominuje nad zakładanym logarytmem.
Aby kopiec uzyskał zakładane złożoności, należałoby zaimplementować go w wersji tablicowej, lecz z jakimś dodatkowym buforem na kolejne liczby, aby nie relokować ich co operację.

5 Drzewo czerwono-czarne



Rysunek 5: Drzewo czerwono-czarne - wykres

Pomiary operacji są zbliżone do zakładanej złożoności O($log(n)$). Niestety końcowy wzrost czasu wykonywania operacji jest stosunkowo mały w porównaniu do ilości danych (zgodnie z charakterystką log(n)), jaką trzeba przetworzyć.
Próby wyłączenia optymalizacji kompilatora oraz testy na większych ilościach danych, dawały niestety podobny efekt.

6 Wnioski

Większość operacji wyszła zgodnie z oczekiwaniami. Po zrealizowaniu badań można dojść do następujących wniosków:

- Kopiec binarny lepiej implementować jako drzewo ze wskaźnikami lub jako tablicę, lecz z pewnym zapasem danych, na kolejne elementy. W przeciwnym wypadku, nie wykorzystujemy zbytnio właściwości kopca (oprócz wyszukiwania największego elementu, które z pewnością będzie szybsze niż w zwykłej tablicy).

- Lista dwukierunkowa znakomicie nadaję się do dodawania/usuwania elementów z ostatniej i pierwszej pozycji. W pozostałych operacjach,

gdzie trzeba przechodzić od wskaźnika do kolejnego wskaźnika (search, deleteIn, addIn), lista osiąga znacznie gorsze wyniki niż tablica i kopiec.

- Drzewo czerwono-czarne uzyskało najlepsze czasy wykonywania operacji. Nie są co prawda one zgodne w całości, z teoretyczną złożonością, lecz pomimo tego i przyjęciu nawet niepewności +/- rzędu 100% ($+/-3.5\mu s$), to i tak są one znacznie szybsze od tablicy, kopca oraz listy (z pominięciem operacji na pierwszych i ostatnich elementach, gdzie lista wygrawa nieznacznie).

7 Literatura

1. https://eduinf.waw.pl/inf/ - mgr Jerzy Wałaszek I LO w Tarnowie
2. Wprowadzenie do algorytmów - Cormen Thomas H., Leiserson Charles E., Rivest Ronald L, Clifford Stein