Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

PAMSI

Sprawozdanie nr 1

Projekt - marzec

Prowadzący:
dr hab. inż. Andrzej Rusiecki

Wykonał:
Jakub Kusz

Wrocław, 25 marca 2022r.

1. Zadanie

Załóżmy, że Jan chce wysłać przez Internet wiadomość W do Anny. Z różnych powodów musi podzielić ją na n pakietów. Każdemu pakietowi nadaje kolejne numery i wysyła przez sieć. Komputer Anny po otrzymaniu przesłanych pakietów musi poskładać je w całą wiadomość, ponieważ mogą one przychodzić w losowej kolejności. Państwa zadaniem jest zaprojektowanie i zaimplementowanie odpowiedniego rozwiązania radzącego sobie z tym problemem. Należy wybrać i zaimplementować zgodnie z danym dla wybranej struktury ADT oraz przeanalizować czas działania - złożoność obliczeniową proponowanego rozwiązania.

2. Rozwiązanie

Kod źródłowy znajduje się **tutaj**.

2.1. Idea

W celu rozwiązania wyżej postawionego zadania napisany został program, którego działanie polega na odczytaniu treści wiadomości z pliku „message.txt", podzieleniu jej na 10-cio znakowe pakiety, nadanie im kluczy definiujących położenie w treści wiadomości, zasymulowaniu wysłania pakietów, zasymulowaniu odebrania pakietów, posortowania i złożenia ich w jedną całą odebraną wiadomość zapisaną w pliku „rec_message.txt".

2.2. Struktura danych

2.2.1. Kolejka priorytetowa

W celu przechowywania pakietów w pamięci komputera została zastosowana kolejka priorytetowa. Zaimplementowana została struktura danych „t_priority_queue " działająca na podstawie listy jednokierunkowej, dodatkowo z możliwością sortowania poprzez wstawianie elementów w odpowiednie miejsca, porównując ich klucze.

2.2.2. Uzasadnienie wyboru

Wybór takiej struktury danych wynika ze specyfiki zadnia - wiadomości posiadające klucz identyfikacyjny mogą być dostarczane w losowej kolejności. Ich liczba jest uzależniona od długości wiadomości (argument za zastosowaniem czegoś co działa na podstawie listy) i posiadają klucz (konieczność sortowania). Do rozwiązania tak postawionego problemu najlepiej nadaje się kolejka priorytetowa. Oto jej definicja:

```
#pragma once
#include <iostream>
#include <stdexcept>
#include"struct_for_message.hpp"


template<typename T>

class t_priority_queue{
```

```
private:

    static constexpr int initial_size = 0;
    int quantity; //quantity of nodes
    struct str_of_data {
        T T_type;
        str_of_data *next = nullptr;
        str_of_data *previous = nullptr;
        int key;
        void operator=(const str_of_data &val);
    };
    str_of_data *data;

    bool comprasion_ascending(int  x, int y){return x > y;};




public:
    t_priority_queue(): quantity(initial_size), data(nullptr){;};
    void insert(const T &val, const int &x);
    void pop();
    void pop_all();
    bool empty(){return data == nullptr;};
    T top();
    int size(){return quantity;};
    void print();

};
```

Metodami pozwalającymi wykonywać operacje na obiektach klasy „t_priority_queue" są:

- insert() - dodaje element do kolejki ustawiając jego położenie na podstawie klucza,
- pop() - usuwa element z początku kolejki,
- pop_all() - usuwa całą kolejkę,
- empty() - orzeka, czy kolejka jest pusta,
- top() - zwraca wartość pierwszego elementu,
- size() - zwraca ilość elementów w kolejce,

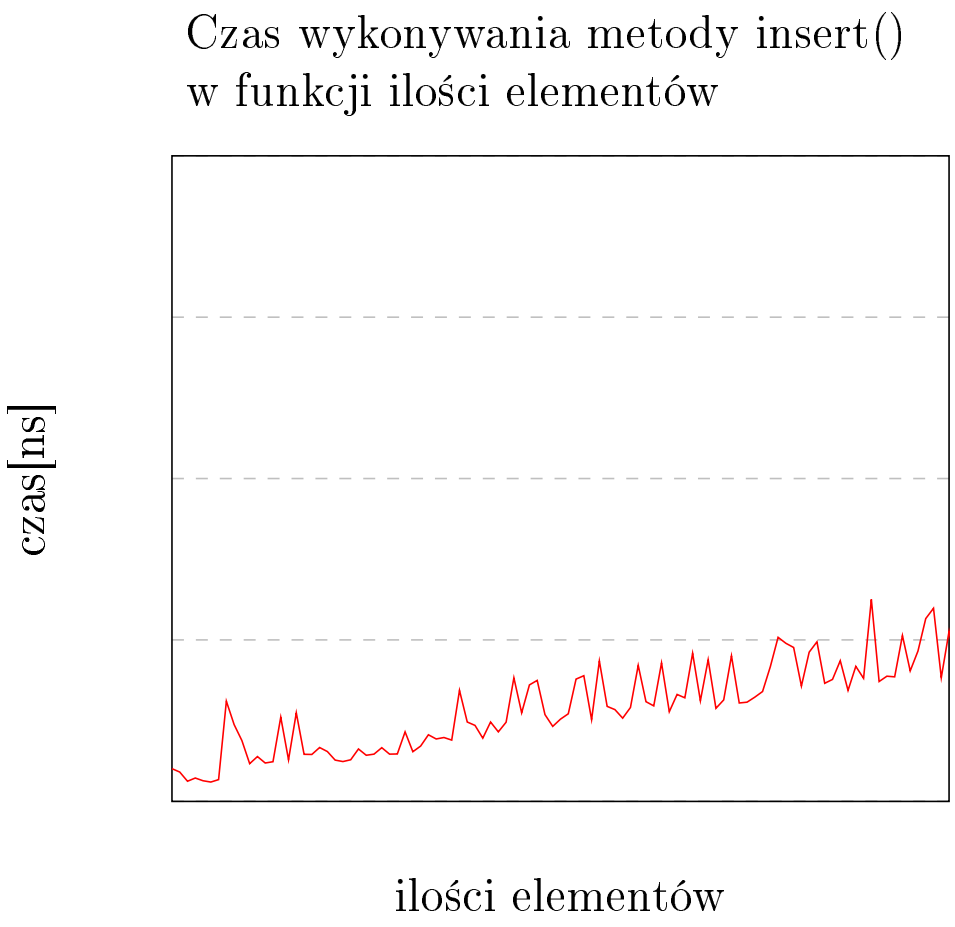
2.3. Sortowanie

Sortowanie danych odbywa się w niezwykle prosty sposób. Do kolejki wysyłana jest dana wraz z kluczem, metoda insert() przesuwa wskaźnik po kolejnych elementach kolejki. Jeśli funkcja orzekająca określająca relacje pomiędzy kolejnymi kluczami kolejki a podanym do insert() kluczem stwierdzi, iż klucz podany przestaje spełniać określoną relację (>,<), insert() utworzy nowy węzeł i umieści nowy element we właściwym miejscu. Poniżej znajduje definicja insert():

```
    void t_priority_queue<T> :: insert(const T &val, const int &x){

    if(data == nullptr){
        data = new str_of_data;
        data->T_type = val;
        quantity++;
        data->key = x;
    }else{
        str_of_data *tmp;
        str_of_data *tmpnew;
        str_of_data *tmpprev;


        tmp = data;
        while(this->comprasion_ascending(x,tmp->key) && tmp->next != nullptr){

            tmp = tmp->next;

        }

        if(this->comprasion_ascending(x,tmp->key)){
            tmp->next = new str_of_data;
            tmp->next->T_type = val;
            tmp->next->key = x;
            tmp->next->previous = tmp;
            quantity++;
        }else{
            if(tmp->previous == nullptr){
                tmpnew = new str_of_data;
                tmpnew->T_type = val;
                tmpnew->key = x;
                tmpnew->next = tmp;
                tmp->previous = tmpnew;
                data = tmp->previous;
            }else{
            tmpnew = new str_of_data;
            tmpnew->T_type = val;
            tmpnew->key = x;
            tmpnew->next = tmp;
            tmpnew->previous = tmp->previous;
            tmp->previous->next = tmpnew;
            tmp->previous = tmpnew;
            if(tmp == data)
                data = tmp->previous;
            quantity++;
            }
        }

    }

}
```

3. Złożoności obliczeniowe

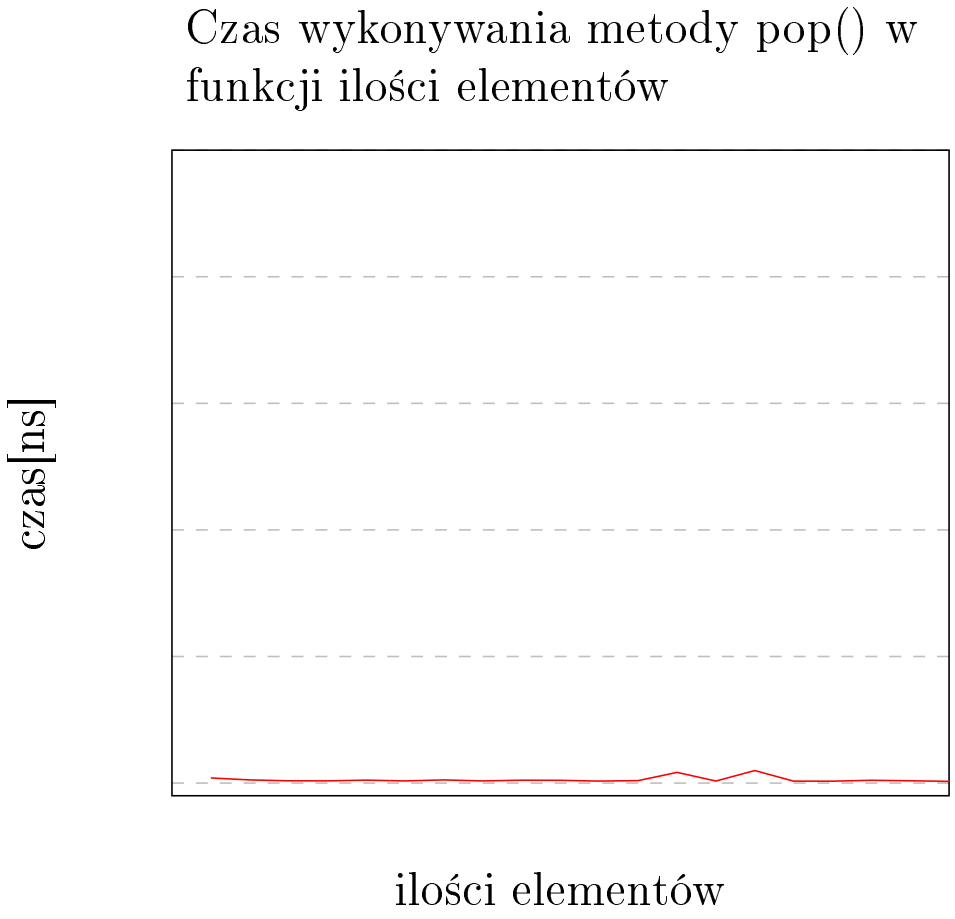
3.1. insert()



Metoda insert() w celu dodania elementu we właściwe miejsce, musi wykonać odpowiednio wiele przejść po elementach kolejki aby natrafić na element, którego klucz nie spełnia właściwej relacji (>,<). Jeśli element, przy którym relacja (>,<) przestaje być spełniona znajduje się w odległości n węzłów od początku kolejki, to kolejka musi wykonać n przejść przez swoją listę. Oznacza to, iż ilość przejść jest wprost proporcjonalna do odległości ów węzła. Z tego wynika, iż złożoność obliczeniowa jest określona **funkcją liniową**. Notacja dużego O:

$$O(n)$$

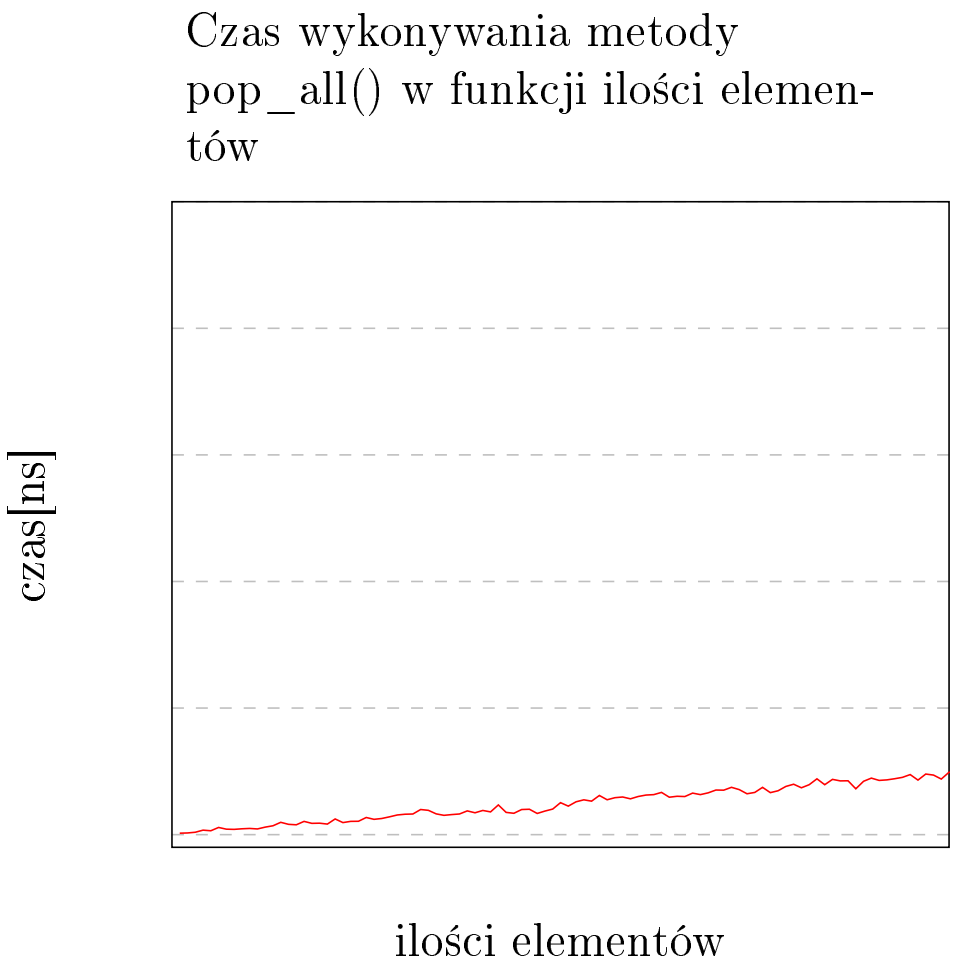
3.2. pop()



Metoda pop() w celu zdjęcia pierwszego elementu musi wykonać zawsze tylko jedną operację, więc jej złożoność jest stała.

$$O(1)$$

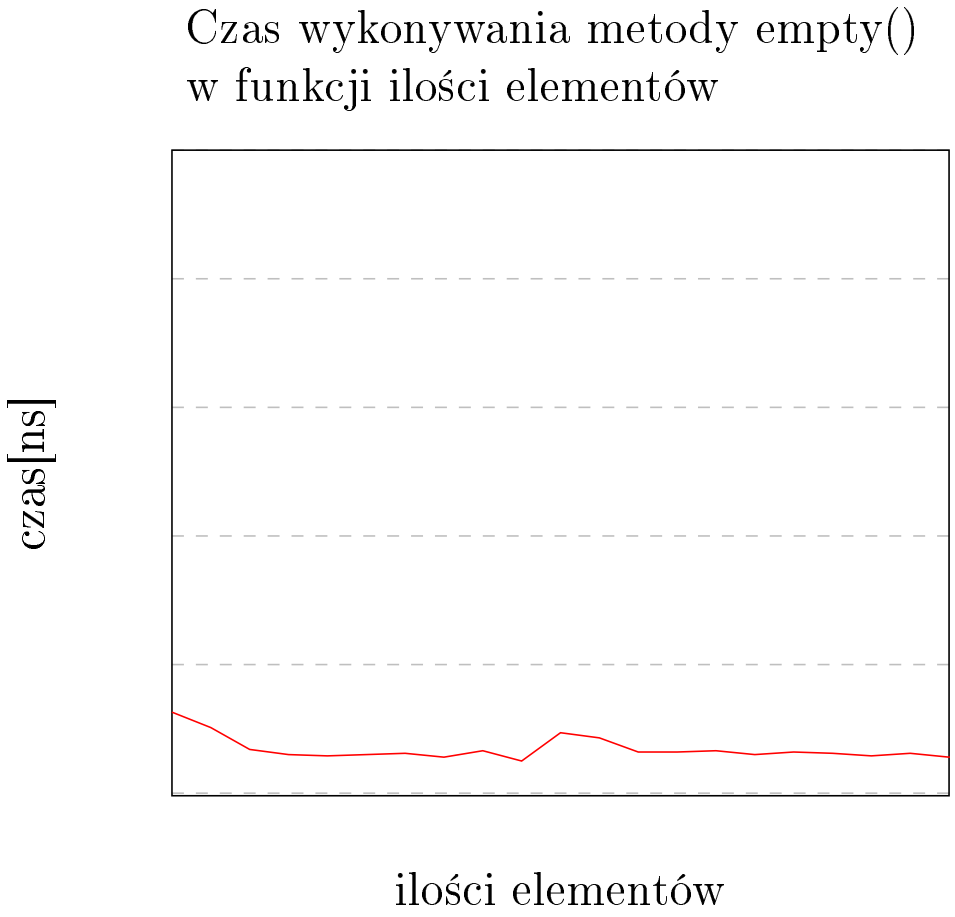
3.3. pop_all()



Metoda pop_all() w celu usunięcia wszystkich elementów musi wykonać tyle operacji ile jest węzłów w kolejce, z czego wynika że jej złożoność obliczeniowa jest liniowa.

$$O(n)$$

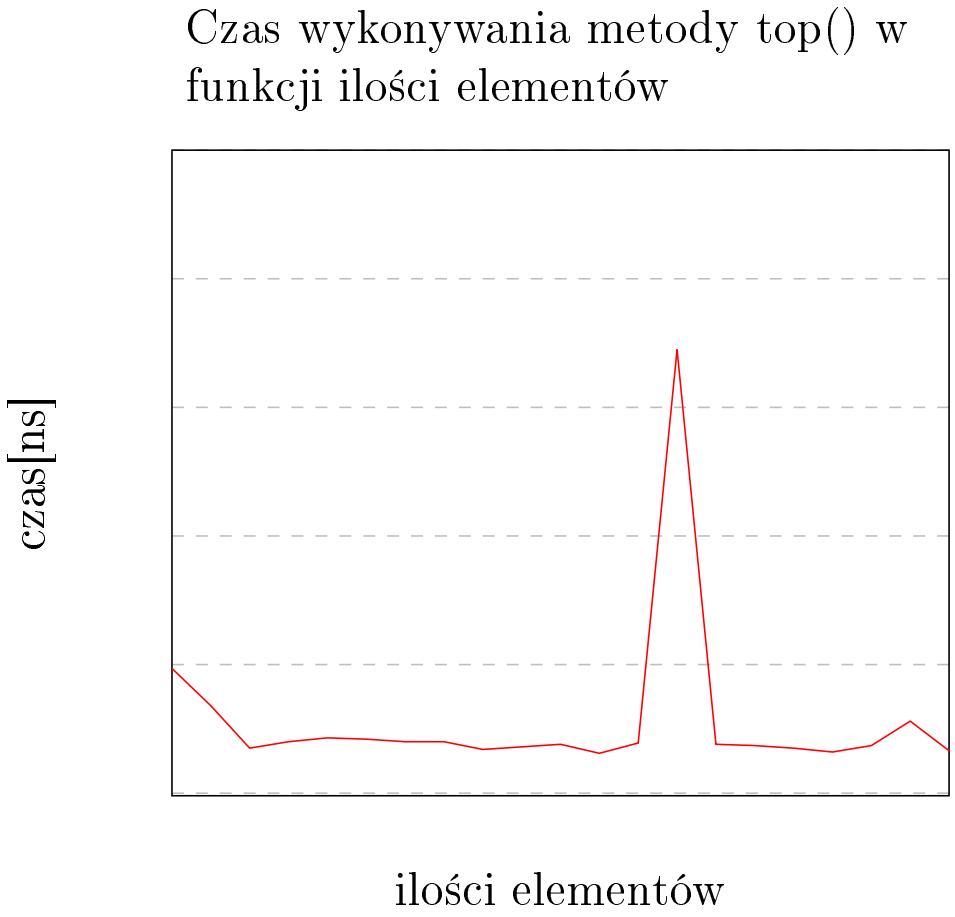
3.4. empty()



Metoda empty() w celu zwrócenia informacji o tym czy kolejka jest pusta musi wykonać zawsze tylko jedną operację, więc jej złożoność jest stała.

$$O(1)$$

3.5. top()



Metoda top() w celu zwrócenia wartości elementu znajdującego się na początku kolejki zawsze musi wykonać tylko jedną operacje, wiec złożoność obliczeniowa jest stała.

$$O(1)$$

3.6. size()

4. Wnioski