



Politechnika Wrocławska

Wydział Elektroniki, Fotoniki i Mikrosystemów

PAMSI

Sprawozdanie nr 1

Projekt - marzec

Prowadzący:

dr hab. inż. Andrzej Rusiecki

Wykonał:

Jakub Kusz

Wrocław, 28 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 zadania - 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:

```
1 #pragma once
2 #include <iostream>
3 #include <stdexcept>
4 #include "struct_for_message.hpp"
5 #include <algorithm>
6 #include "how_sort.hpp"
7
8
9
```

```

10
11 template<typename T>
12
13 class t_priority_queue{
14
15     private:
16
17         static constexpr int initial_size = 0;
18         int quantity; //quantity of nodes
19         struct str_of_data {
20             T T_type;
21             str_of_data *next = nullptr;
22             str_of_data *previous = nullptr;
23             int key;
24             void operator=(const str_of_data &val);
25         };
26         str_of_data *data;
27
28         sort how_sort;
29
30         bool comprasion(int x, int y){
31             if(how_sort == sort :: asc)
32                 return x > y;
33             if(how_sort == sort ::des)
34                 return x < y;
35
36         };
37
38
39
40
41     public:
42         t_priority_queue<T>(sort x): how_sort(x),quantity(initial_size), data(nullptr){};
43         t_priority_queue<T>() = default;
44         void insert(const T &val, const int &x);
45         void pop();
46         void pop_all();
47         bool empty(){return data == nullptr;};
48         T top();
49         int size(){return quantity;};
50         void print ();
51
52 };

```

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()`:

```

1      void t_priority_queue<T> :: insert(const T &val, const int &x){
2
3      if (data == nullptr){
4          data = new str_of_data;
5          data->T_type = val;
6          quantity++;
7          data->key = x;
8      }else{
9          str_of_data *tmp;
10         str_of_data *tmpnew;
11         str_of_data *tmpprev;
12
13
14         tmp = data;
15         while( this->comprasion(x,tmp->key) && tmp->next != nullptr){
16
17             tmp = tmp->next;
18
19         }
20
21         if ( this->comprasion(x,tmp->key)){
22             tmp->next = new str_of_data;
23             tmp->next->T_type = val;
24             tmp->next->key = x;
25             tmp->next->previous = tmp;
26             quantity++;
27         }else{
28             if (tmp->previous == nullptr){
29                 tmpnew = new str_of_data;
30                 tmpnew->T_type = val;
31                 tmpnew->key = x;
32                 tmpnew->next = tmp;
33                 tmp->previous = tmpnew;
34                 data = tmp->previous;
35             }else{
36                 tmpnew = new str_of_data;
37                 tmpnew->T_type = val;
38                 tmpnew->key = x;
39                 tmpnew->next = tmp;

```

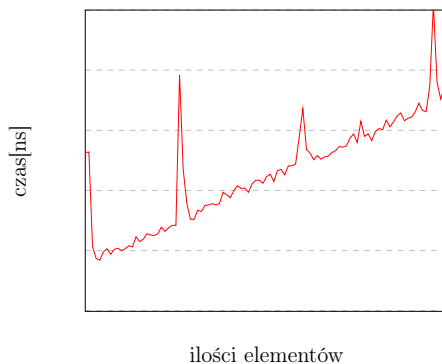
```

40     tmpnew->previous = tmp->previous;
41     tmp->previous->next = tmpnew;
42     tmp->previous = tmpnew;
43     if (tmp == data)
44         data = tmp->previous;
45     quantity++;
46 }
47 }
48
49 }
50
51 }
```

3. Złożoności obliczeniowe

3.1. insert()

Czas wykonywania metody insert()
w funkcji ilości elementów



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:

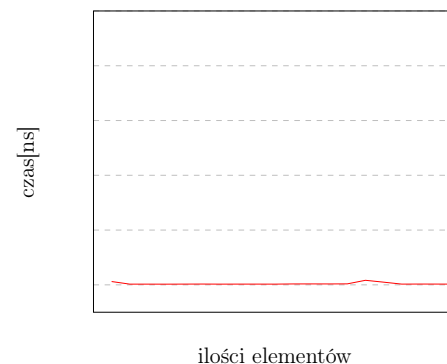
$$O(n)$$

Uwaga. W celu ukazania liniowej złożoności obliczeniowej na każdy kolejny element dodawany jest z

wyższą wartością klucza od największego klucza w kolejce, aby każde kolejne dodanie elementu wiązało się z przebyciem drogi przez całą kolejkę.

3.2. pop()

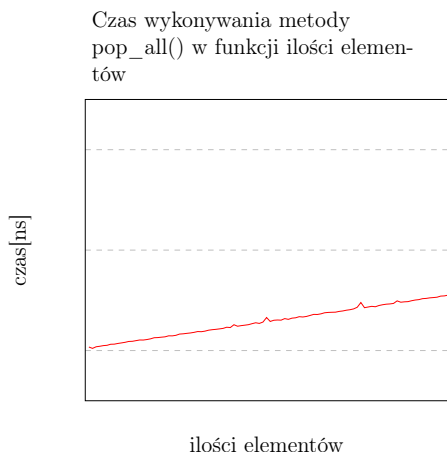
Czas wykonywania metody pop() w
funkcji ilości elementów



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

$$O(1)$$

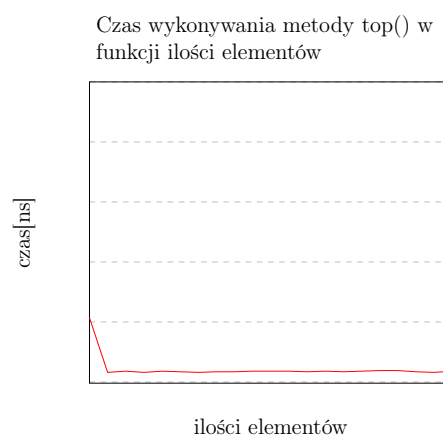
3.3. pop_all()



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. Notacja dużego O:

$$O(n)$$

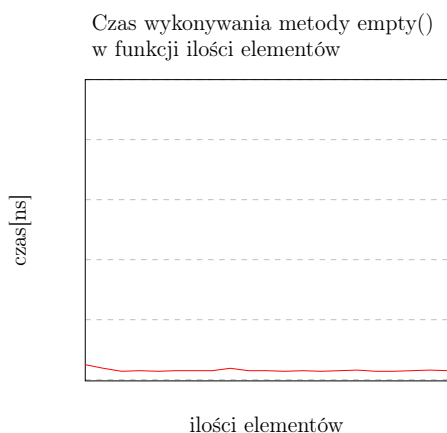
3.5. top()



Metoda top() w celu zwrócenia wartości elementu znajdującego się na początku kolejki zawsze musi wykonać tylko jedną operację, więc złożoność obliczeniowa jest stała. Notacja dużego O:

$$O(1)$$

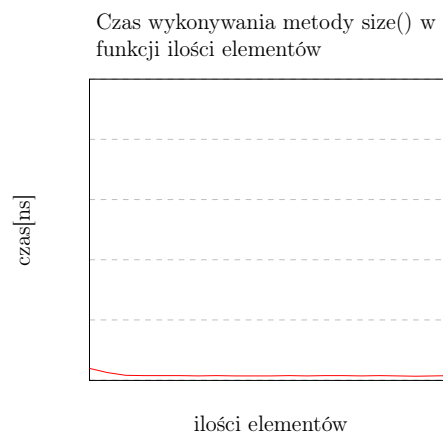
3.4. empty()



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

$$O(1)$$

3.6. size()



Metoda size() w celu zwrócenia wartości ilości elementów w kolejce zawsze musi wykonać tylko jedną operację, więc złożoność obliczeniowa jest stała. Notacja dużego O:

$$O(1)$$

4. Podsumowanie i wnioski

Poprzez zaimplementowanie kolejki priorytetowej udało się rozwiązać zadanie postawione w pkt.(1), co prezentuje driver znajdujący się w podanym w pkt.(2) repozytorium. Wybór kolejki priorytetowej uważam za jak najbardziej uzasadniony, specyfika jej działania idealnie sprawdza się w przypadkach, gdy należy sortować ciągle napływające dane, nie znając do tego ich ilości.

Szybkość działania kolejki jest na bardzo wysokim poziomie, powyższa implementacja gwarantuje liniową zależność czasową podczas dodawania elementu jak i podczas usuwania całej kolejki, pozostałe metody są stałe w czasie.