Projektowanie Algorytmów i Metody Sztucznej Inteligencji	
Kierunek *Automatyka i Robotyka*	**Termin** *Wtorek 13.15*
Imię, nazwisko, numer albumu *Łukasz Walczak 259278*	**Ocena:**
Prowadzący projekt *Dr hab. inż. Andrzej Rusiecki*	**Data oddania sprawozdania** *17 marca 2023*

Projekt 1 - Zadania na ocenę bdb

Spis treści

Link do repozytorium

`https://github.com/CrassusXY/Pamsi-1`

1 Zadanie do wykonania

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 Interpretacja zadania

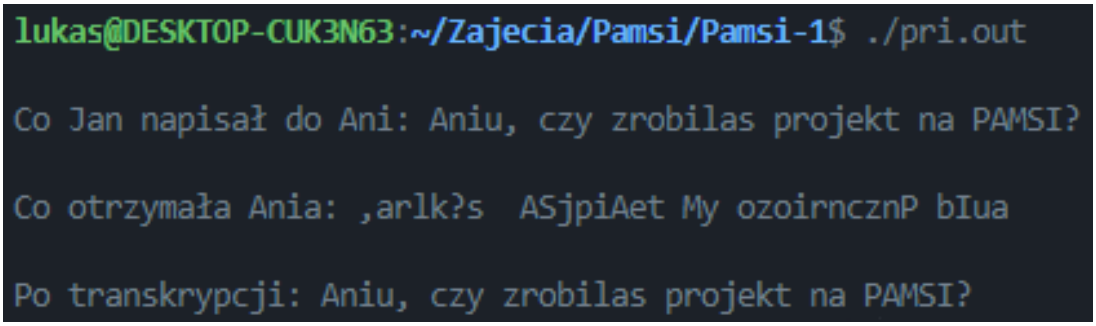
1. Użytkownik wpisuje wiadomość w terminalu
2. Program dzieli wiadomość na n pakietów, gdzie n to liczba znaków w wiadomości
3. Każdy pakiet otrzymuje kolejny numer ID reprezentujący jego priorytet w kolejce priorytetowej
4. Pakiety są dodawane w losowej kolejności do kolejki
5. Metoda push kolejki, wrzuca pakiety w kolejności posortowanej względem priorytetów
6. Przy ściaganiu elementów zawsze sciąga się najpierw ten z najmniejszym priorytetem
7. Program ściąga wszystkie pakiety znajdujące się w kolejce i wypisuje je w kolejności posortowanej

3 Wybrane struktury danych

Lista jednokierunkowa – struktura danych w której elementy są poukładane w porządku liniowym. Każdy element zawiera adres następnego elementu. Mój wybór padł na listę jednokierunkową ze względu na prostotę w implementacji i fakt, że w użyciu przy moim sposobie implementacji kolejki, nie potrzebuję nigdy odwoływać się do poprzedniego elementu w kolejce.
Kolejka priorytetowa – abstrakcyjnych typ danych służący do reprezentowania zbioru elementów, gdzie każdy element reprezentowany jest przez węzeł(node), który zawiera adres następnego elementu, pakiet wiadomości oraz priorytet danego pakietu. Kolejkę priorytetową wybrałem, ponieważ do realizacji zadania polegającego na sortowaniu względem priorytetów pakietów wydaje mi się to najbardziej efektywna struktura danych.

4 Implementacja

Program składa się z drivera main, makefile oraz 2 klas:
class Node - jest to podstawowy element kolejki. W private klasy znajduje się wskaźnik na kolejny węzeł, znak char i priorytet danego pakietu. Klasa ta dostarcza metody pozwalające na pobranie wiadomości, priorytetu i adresu następnego pakietu oraz metode pozwalającą na modyfikacje wskaźnika na następny pakiet.
class Priority_queue - kolejka priorytetowa. W private klasy znajdziemy wskaźnik na początek kolejki oraz jej rozmiar. Klasa ta dostarcza metody pozwalające na pobranie rozmiaru, zapytanie czy kolejka jest pusta, zwrócenie elementu o najmniejszym priorytecie, który jest również pierwszym elementem listy i metodę pop_all polegającą na zwróceniu wszystkich elementów danej listy. Najważniejszą metodą dla tej klasy, jest metoda push, która polega na dodawaniu do kolejki nowych węzłów. Przeszukuje ona listę w poszukiwaniu pierwszego elementu o prirytecie niższym niż priorytet nowego węzła, a następnie dodaje nowy węzęł przed ten o wyższym priorytecie.
main - main pobiera od użytkownika wiadomość, nastepnie generuje dwie talice o elementach od 0 do n, gdzie n to długość wiadomości. Jedna z tych tablic jest permutowana przy użyciu funkcji std::random_shuffle z biblioteki algorithm, druga z tych służy jako lista priorytetów kolejnych pakietów. Pakiety są dodawane do kolejki zgodnie z kolejnością pochodzącą z wymieszanej listy. Następnie cała kolejka jest opróżniana na terminal.



Rysunek 1: Przykład działania programu

5 Analiza złożoności obliczeniowej

Kolejnym etapem była analiza złożoności obliczeniowej oraz podanie jej wyniku w notacji dużego O. Analizę złożoności obliczeniowej przedstawię w na metodzie pop oraz push z kolejki priorytetowej.

5.1 push():

```
int Priority_queue::push(const uint & prio, const char & item){    Ilość operacji:
    Node *add = new Node(prio, item);                               2
    Node *tmp = head;                                               1
    if(!is_empty()){                                                2
        if(tmp->get_priority() > prio){                             2
            add->set_next(tmp);                                     1
            head = add;                                             1
            length++;                                               1
            return 0;                                               1
        }
        while(tmp->get_next() != nullptr &&                         2n
        tmp->get_next()->get_priority() < prio){                    3n
            tmp = tmp->get_next();                                  2n
        }
        add->set_next(tmp->get_next());                             2
        tmp->set_next(add);                                         1
        length++;                                                   1
        return 0;                                                   1
    }
    else{                                                           1
        add->set_next(tmp);                                         1
        head = add;                                                 1
        length++;                                                   1
        return 0;                                                   1
    }
    return -1;                                                      1
}                                                                   SUMA:
                                                                    7n+22
```

W najgorszym przypadku zostanie wykonane 7n+22 operacji. Zatem złożoność obliczeniowa wynosi O(n)

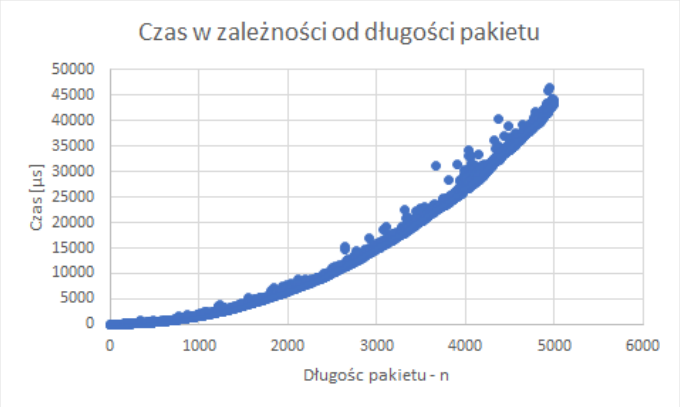
5.2 pop():

```
int Priority_queue::pop(){                                  Ilość operacji:
    if(!is_empty()){                                        1
        Node *tmp = head;                                   1
        print();                                            1
        head = head->get_next();                            2
        delete(tmp);                                        1
        return 0;                                           1
    }
    else{                                                   1
        throw std::logic_error("Queue is empty!");          1
    }                                                       SUMA:
}                                                           9
```

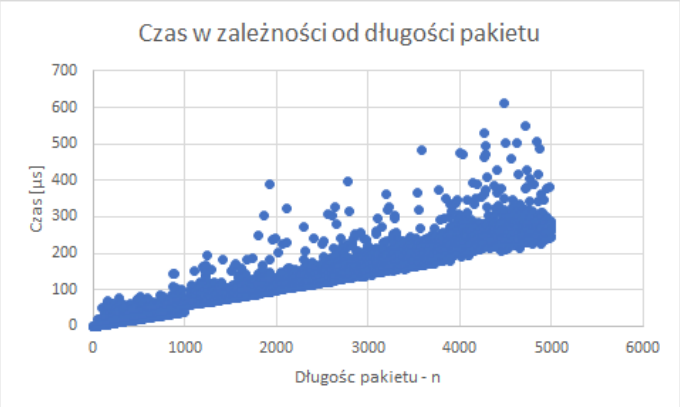
W najgorszym przypadku zostanie wykonane 9 operacji. Zatem złożoność obliczeniowa wynosi O(1).

5.3 Czas wykonania operacji w zależności od długości pakietu

Dodatkowym testem było eksperymentalne sprawdzenie zależności czasowej funkcji push oraz pop polegające na wykonaniu jej dla pakietów o długości n od 0 do 5000. Zatem przy wywołaniu funkcji push oraz pop n razy, ich złożoność obliczeniowa powinna wynieść odpowiednio $O(n^2)$ oraz $O(n)$.



Rysunek 2: Zależność czasowa dla sortowania przez wstawianie



Rysunek 3: Zależność czasowa dla wykonania n operacji ściągnięcia najmniejszego priorytetu

Patrząc na wykresy można łatwo stwierdzić, że zależność czasowa dodawania elementów, to funkcja kwadratowa, natomiast operacja ściągania to zależność liniowa. Co jest zgodne z oczekiwaniami.