Projektowanie algorytmów i metody sztucznej inteligencji - Raport 1

Damian Ryś

26 kwietnia 2021

Spis treści

1 Wstęp **1**

2 Zadanie **2**

3 Kolejka Priorytetowa **2**
3.1 Czym jest kolejka priorytetowa? 2
3.2 Zalety i wady 2
3.3 Implementacja 3
3.4 Złożoności obliczeniowe 4
3.5 Testy 4

4 Wnioski **6**

5 Bibliografia **6**

1 Wstęp

Grupa lab: E12-93c
Termin zajęć: PN 18:45-20:35
Numer indeksu: 252936
Prowadzący: Dr inż. Piotr Ciskowski

2 Zadanie

Postawionym pzed nami problemem jest wysłanie przez użytkownika A do użytkownika B wiadomości przez Internet. wiadomość ta powinna zostać wysłana w serii pakietów na komputer użytkownika B. Problem polega w tym, że pakiety mogą przychodzić z różnych przycznym w losowej kolejności do użytkownika B. Nasz program powinien zatem:

1. Podzielić naszą wiadomość na szereg pakietów
2. Wysłać je w losowej kolejności
3. Wczytać do struktury danych nasze pakiety
4. Przesyłać pakiety w foramcie [klucz,wartość]
   (a) klucz powininen mieć unikalną wartość
   (b) wartość powinna być dowolnego typu (użytkownik powinine mieć możliwość przesłania dowolnej wiadomości)
5. Wyświetlić użytkownikowi wiadomość w poprawnej kolejności

3 Kolejka Priorytetowa

Wobec postawionego problemu strukturą ADT, którą wybrałem jest kolejka priorytetowa bazowana na liście.

3.1 Czym jest kolejka priorytetowa?

Kolejka priorytetowa (ang. priority queue) jest kolejką, w której elementy są ułożone nie w kolejności wprowadzania, lecz w kolejności priorytetu. Każdy element kolejki posiada dodatkowe pole priority, w którym przechowuje swój priorytet – czyli ważność. Gwarantuje to pobieranie z kolejki jako pierwszych elementów o najwyższym priorytecie. Elementy o priorytetach niższych zostaną pobrane dopiero wtedy, gdy zostaną usunięte wszystkie elementy o priorytetach wyższych.

3.2 Zalety i wady

Strukturę tą wybrałem ze względu na:

- Łatwość implementacji
- Dynamiczną alokację danych
- Przechowywanie danych w formacie [klucz,wartość];

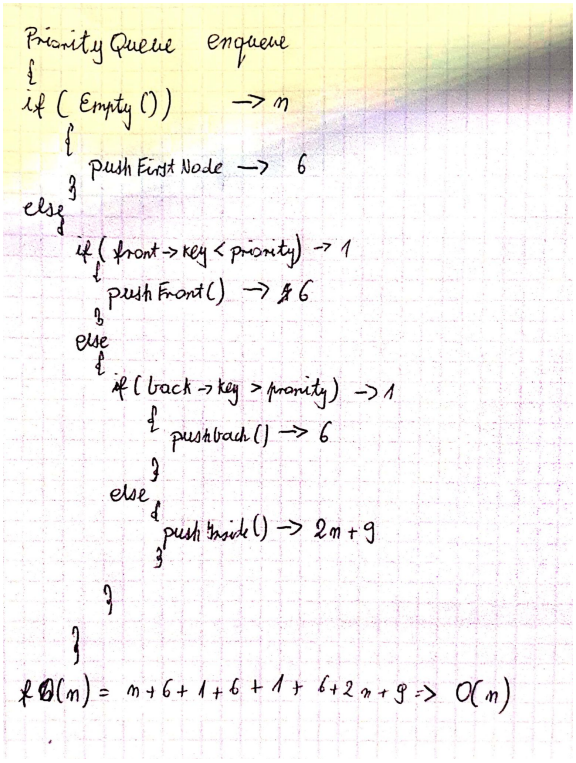
3.3 Implementacja

```
1     template<typename T>
2     class PriorityQueue
3     {
4         typedef struct Node
5         {
6             T field; //wartosc
7             int key; // klucz
8             Node *nextNode;
9         } *nodePointer;
10
11        nodePointer front; //wskaznik na nastepny element
12        nodePointer back; // wskaznik na poprzedni element
13    public:
14        PriorityQueue(); //konstruktor
15        ¬PriorityQueue(); //destruktor
16        // dodanie elementu
17        void enqueue(const T& newElement, int priority);
18        // usuniecie elementu
19        void dequeue();
20        // wyswietlenie kolejki w prawidlowym porzadku
21        void Print();
22        // sprawdzenie czy kolejka nie jest pusta
23        bool Empty();
24    private:
25    //metody do obslugi enqueue i dequeue
26        void pushBack(const T& newElement, int priority);
27        void pushFront(const T& newElement, int priority);
28        void pushFirstNode(const T& newElement, int priority);
29        void pushInside(const T& newElement, int priority);
30
31    };
```

Powyżej znajduję się klasa kolejki priorytowej, reszta kodu dostępna jest na rezpozytorium na Githubie lub dołączonym skompresowanym pliku.

3.4 Złożoności obliczeniowe

Dla naszego programu zgodnie z założeniami projektowymi zostały obliczene złożoności czasowe poszczególnych funkcji naszej struktury:



Rysunek 1: Złożoność obliczeniowa dla metody enqueue

Złożoność została policzona również dla innych metod,co można zobaczyć w tabelce poniżej:

Dodanie elementu	Usuniecie elementu	Wyświetlenie struktury
$O(n)$	$O(n)$	$O(n)$

(1)

Nasze wyniki pokrywają się z danymi dostępnymi w internecie,jak na przykład TUTAJ. Uznaje zatem, że kolejka została zaimplementowana poprawnie.

3.5 Testy

Dla bazy naszych testów zasymulowałem sytuacje w której użytkownik B otrzymuje od użytkownika A pakiety w złej kolejności.

```
[Running] cd "/home/damian/Documents/GitHub/PAMSI/src/" && g++ main.cpp -o main && "/home/damian/Documents/GitHub/PAMSI/src/"
Enqueued element Ala  with priority 1
Enqueued element kota with priority 3
Enqueued element ma with priority 2
Enqueued element a  with priority 4
Enqueued element Ale  with priority 7
Enqueued element kot  with priority 5
Enqueued element ma  with priority 6
Wartość: Ala    Priority: 1
Wartość: ma   Priority: 2
Wartość: kota   Priority: 3
Wartość: a    Priority: 4
Wartość: kot    Priority: 5
Wartość: ma    Priority: 6
Wartość: Ale    Priority: 7


[Done] exited with code=0 in 1.54 seconds
```

Rysunek 2: Test programu dla stringów

Program został przetestowany również dla różnego typu elementów, co widać na poniższym screenie:

```
Enqueued element 112 with priority 1
Enqueued element 23 with priority 3
Enqueued element 2321 with priority 2
Enqueued element 6323 with priority 4
Enqueued element 21332 with priority 7
Enqueued element 745 with priority 5
Enqueued element 24 with priority 6
Wartość: 112   Priority: 1
Wartość: 2321   Priority: 2
Wartość: 23   Priority: 3
Wartość: 6323   Priority: 4
Wartość: 745   Priority: 5
Wartość: 24   Priority: 6
Wartość: 21332   Priority: 7


[Done] exited with code=0 in 1.035 seconds
```

Rysunek 3: Test programu dla intów

Program finalnie przeszedł testy, zatem uznaje, że program działa prawidłowo.

4 Wnioski

Udało nam się zrealizować przedstawiony problem, to co możnaby w nim poprawić to zastosowanie innej wersji kolejki priorytetowej w celu ulepszenia Złożoności czasowej naszej struktury ADT, co wiąże się z większą trudnością implementacji.

5 Bibliografia

Schemat implementacji wraz z schematami blokowymi
Inne rozwiązanie problemu na drzewach binarnych
Złożoności czasowe różnych algorytmów