

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 zmien

Termin zajęć: PN 18:45-20:35

Numer indeksu: 252936

Prowadzący: Dr inż. Piotr Ciskowski

2 Zadanie

Postawionym przed 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 przyczyn w losowej kolejności do użytkownika B. Nasz program powinien zatem:

1. Podzielić naszą wiadomość na szereg pakietów
2. Przelosować pakiety w celu zasymulowania sytuacji
3. Wczytać do struktury danych nasze pakiety
4. Przesyłać pakiety w formacie [klucz, wartość]
 - (a) klucz powinien mieć unikalną wartość
 - (b) wartość powinna być dowolnego typu (użytkownik powinien mieć możliwość przesłania dowolnej wiadomości)
5. Złączyć wiadomość w jedną, prawidłową całość i wyświetlić użytkownikowi

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
- Złożoność czasową lepszą niż $O(n)$ dla uzyskania min oraz max
- Dynamiczną alokację danych

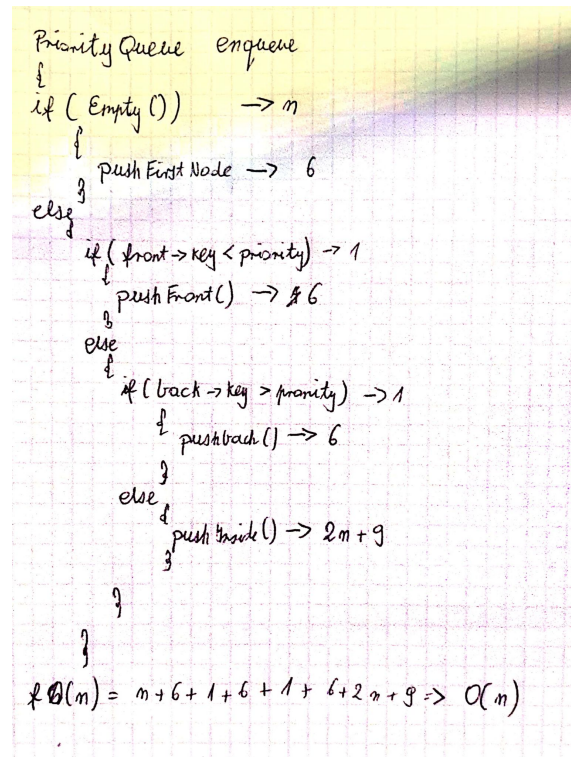
3.3 Implementacja

```
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 obsługi 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 znajdują się klasa kolejki priorytowej, reszta kodu dostępna jest na repozytorium na Githubie lub dołączonym skompresowanym pliku.

3.4 Złożoności obliczeniowe

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



```

Priority Queue enqueue
{
  if ( Empty () )      → n
  {
    push First Node    → 6
  }
  else
  {
    if ( front → key < priority ) → 1
    {
      push Front ()      → 6
    }
    else
    {
      if ( back → key > priority ) → 1
      {
        push back ()      → 6
      }
      else
      {
        push back ()      → 2n + 9
      }
    }
  }
}

T(n) = n + 6 + 1 + 6 + 1 + 6 + 2n + 9 ⇒ O(n)

```

Rysunek 1: Złożoność obliczeniowa dla metody enqueue

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

//tabelka

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 sytuację w której użytkownik B otrzymuje od użytkownika A pakiety w złej kolejności.

```

Priority Queue enqueue
{
  if ( Empty () ) → n
  {
    push Front Node → 6
  }
  else
  {
    if ( front → key < priority ) → 1
    {
      push Front () → 6
    }
    else
    {
      if ( back → key > priority ) → 1
      {
        push back () → 6
      }
      else
      {
        push inside () → 2n + 9
      }
    }
  }
}
T T(n) = n + 6 + 1 + 6 + 1 + 6 + 2n + 9 => O(n)

```

Rysunek 2: Złożoność obliczeniowa dla metody enqueue

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

```

Priority Queue Enqueue
{
  if ( Empty () ) → n
  {
    push Front Node → 6
  }
  else {
    if ( front → key < priority ) → 1
    {
      push Front () → 6
    }
    else {
      if ( back → key > priority ) → 1
      {
        push back () → 6
      }
      else {
        push inside () → 2n + 9
      }
    }
  }
}

T(n) = n + 6 + 1 + 6 + 1 + 6 + 2n + 9 => O(n)

```

Rysunek 3: Złożoność obliczeniowa dla metody enqueue

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

4 Wnioski

5 Bibliografia

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