Politechnika Wrocławska

Projektowanie i analiza algorytmów

Projekt 1

ArtsyKimiko

4 kwietnia 2024

1 Wstęp

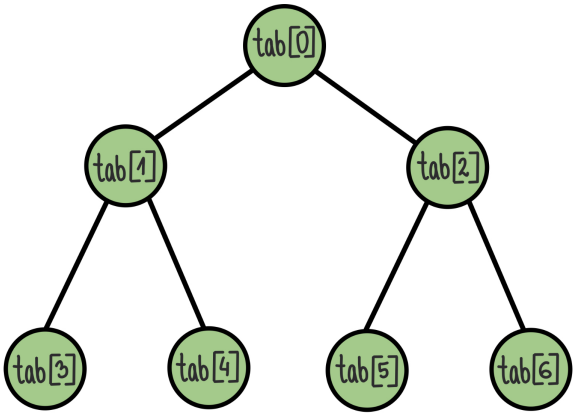
Problemem, który należało rozwiązać podczas wykonywania tego projektu było zapewnienie możliwości odczytywania danych uporządkowanych według priorytetu. Celem było zagwarantowanie, że wiadomości, nawet te dostarczone w niewłaściwej sekwencji, będą poprawnie interpretowane przez odbiorcę, a także opracowanie systemu, który efektywnie zarządzałby kolejnością tych danych na podstawie określonego priorytetu.

2 Opis wybranej struktury

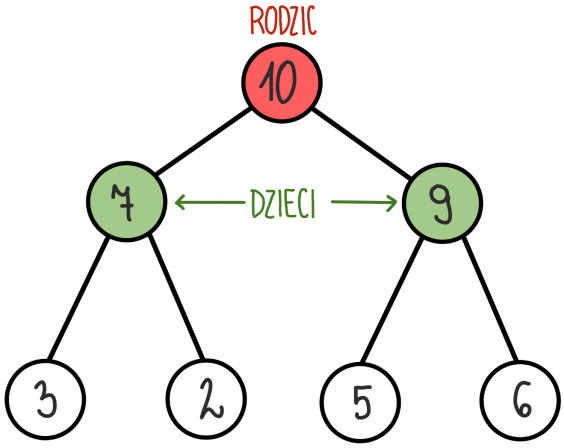
Aby skutecznie zrealizować główne założenia projektu, wybór sortowania przez kopcowanie wydaje się być najlepszym rozwiązaniem.

Kopiec, jako struktura oparta na drzewie binarnym, wyróżnia się tym, że w odróżnieniu od zwykłego drzewa binarnego, jego elementy muszą być uporządkowane według określonego warunku. W przypadku kopca maksymalnego, ten warunek mówi, że każdy rodzic musi być większy lub równy swoim dzieciom. Ten prosty warunek zapewnia, że największy element znajduje się na szczycie kopca, co ułatwia sortowanie danych. Na rysunku 2 zilustrowano graficznie zależność rodzica wraz z jego dziećmi.

Przy sortowaniu danych rosnąco z użyciem kopca, funkcja zamienia największy element (znajdujący się na szczycie kopca) z ostatnim elementem, a następnie układa strukturę kopca poprzez wywołanie funkcji inicjalizującej kopiec, pomijając ostatni element - czyli ten, który został już umieszczony na właściwym miejscu.



Rysunek 1: Drzewo binarne - nie uzależnione od wartości tablicy o danym indeksie.



Rysunek 2: Zależność rodzic-dzieci w kopcu na przykładzie losowych wartości - uzależnione od wartości.

3 Proponowane rozwiązanie problemu

Zaproponowane rozwiązanie polega na wprowadzeniu przez użytkownika wielkości generowanej wiadomości, która będzie składać się z danych oraz priorytetów. Priorytety są losowo generowane, a następnie tworzony jest kopiec, który jest sortowany. Użytkownik ma możliwość wyboru, czy chce zobaczyć priorytety przed sortowaniem, po sortowaniu, lub jedynie czas realizacji sortowania.

Poniżej znajduje się bardziej szczegółowe omówienie funkcji oraz ich działania.

3.1 Kopiec

W przedstawionej funkcji stosowane jest podejście zwane przesiewaniem w dół (heapify down), które ma na celu utrzymanie struktury kopca (`heap`) poprzez porównywanie wartości rodzica z wartościami jego dzieci. Jeśli wartość rodzica jest mniejsza od któregoś z dzieci, następuje zamiana miejscami rodzica z tym większym dzieckiem. Proces ten kontynuowany jest rekurencyjnie dla nowo zamienionego węzła, aż do momentu, gdy rodzic będzie większy od swoich dzieci lub stanie się ostatnim węzłem w kopcu. Dzięki temu jest pewność, że największa wartość w kopcu znajduje się na szczycie.

```
void heap(Packet tab[], int size, int i)
{
    int max=i; //parent
    int left=2*i+1; //left child
    int right=2*i+2; //right child

    if(left<size&&tab[left].priority>tab[max].priority)
        max=left;

    if(right<size&&tab[right].priority>tab[max].priority)
        max=right;

    if(max!=i)
    {
        swap(tab[i],tab[max]);
        heap(tab,size,max);
    }
}
```

Dodatkowo, w głównej funkcji programu wprowadzono ulepszenie, które eliminuje zbędne porównania rodziców z dziećmi. Każdy węzeł o indeksie większym lub równym `size/2-1` jest traktowany jako liść, co oznacza, że nie ma on dzieci. Dlatego też nie ma potrzeby wykonywania operacji naprawiania kopca na tych węzłach. Przechodząc od indeksu `size/2-1` do 0, naprawiamy kopiec od dolnej do górnej części, co gwarantuje, że każde wywołanie `heap` naprawia aktualny węzeł i zapewnia, że jego dzieci są już kopcem.

```
    for (int i=size/2-1;i>=0;i--)
        heap(tab,size,i);
```

3.2 Sortowanie

Poniższa funkcja sortująca w pierwszej kolejności zamienia największy element z góry kopca z ostatnim. Ta operacja umożliwia nam "wyciągnięcie"największego elementu z nieposortowanego fragmentu i umieszczenie go na jego właściwym miejscu w posortowanej części. Następnie wywołując funkcje `heap` sprawdzamy poprawność struktury kopca pomijając ostatni element, który został już umieszczony na właściwym miejscu.

```
void sort(Packet tab[], int i, int size)
{
    swap(tab[0],tab[i]);
    heap(tab,i,0);
}
```

4 Złożoność obliczeniowa

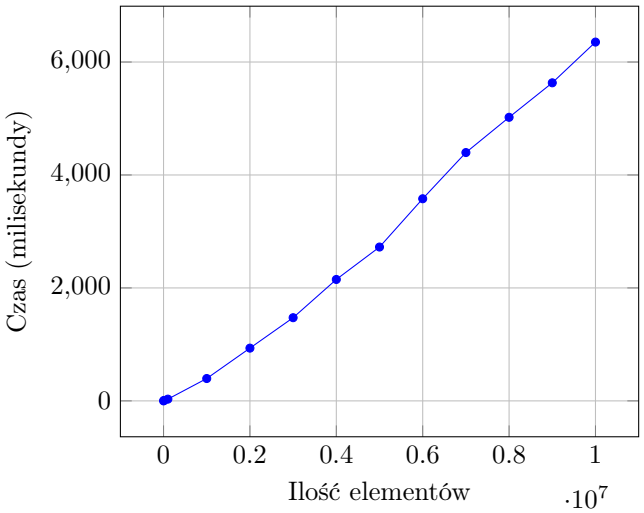
Funkcja `heap()` implementująca kopiec ma złożoność równą $O(\log n)$, ponieważ przy każdym wywołaniu rekurencyjnym rozmiar kopca zmniejsza się o połowę dzięki czemu złożoność obliczeniowa staje się logarytmiczna.

Natomiast funkcja `sort()`, wykorzystująca funkcję `heap()`, jest wywoływana w pętli, co skutkuje tym, że jej złożoność jest `n` razy większa od złożoności funkcji `heap()`. Co daje nam złożoność obliczeniową równą $O(n \log n)$.

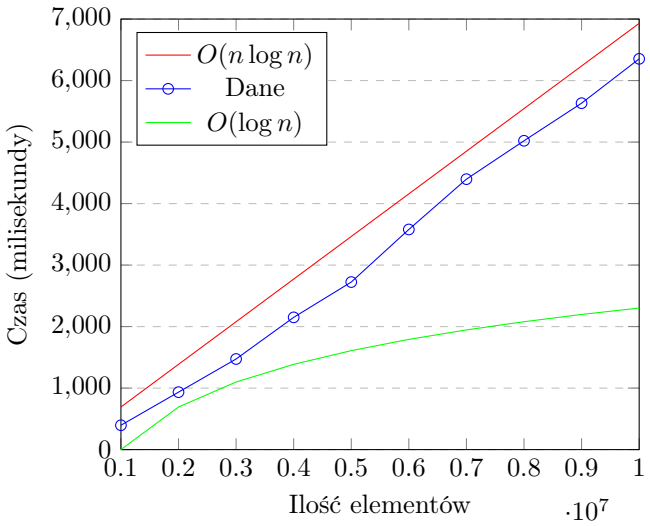
Jest to jednocześnie złożoność obliczeniowa całego programu. Poniżej znajdują się dane oraz wykresy z wartościami zmierzonymi przez program.

Tabela 1: Dane czasowe

Ilość elementów	Czas (ms)
10000000	6353
9000000	5632
8000000	5021
7000000	4397
6000000	3579
5000000	2724
4000000	2150
3000000	1473
2000000	934
1000000	396



Rysunek 3: Wykres zależności czasu wykonywania sortowania od ilości elementów w tablicy.



Rysunek 4: Porównanie wyników zmierzonych przez program ze złożonością czasową $O(n \log n)$ i $O(\log n)$

5 Wnioski

Wykorzystanie struktury kopca do sortowania danych pozwoliło na zachowanie optymalnej kolejności według określonego priorytetu. Dzięki temu możliwe jest szybkie odczytywanie danych, nawet w przypadku ich dostarczenia w niewłaściwej kolejności.

Wprowadzone optymalizacje, takie jak eliminacja zbędnych porównań dla węzłów, pomogły w zoptymalizowaniu działania algorytmu sortowania przez kopcowanie. Dzięki temu program jest bardziej wydajny, co wpływa na szybkość przetwarzania danych.

Analiza złożoności obliczeniowej wykazała, że całkowita złożoność programu wynosi $O(n \log n)$, co jest zgodne z oczekiwaniami dla algorytmu sortowania przez kopcowanie.

Algorytm sortowania przez kopcowanie jest efektywnym rozwiązaniem do sortowania danych, zwłaszcza w przypadkach, gdzie istotne jest utrzymanie optymalnej kolejności danych oraz szybkie przetwarzanie ich, co potwierdzają przeprowadzone analizy złożoności obliczeniowej oraz skuteczność zastosowanych optymalizacji.