Trabalho – Modelagem Computacional em Grafos

Aluna: Izabela Leme

2021

O processo do HeapSort é um pouco complicado, mas é bem mais rápido do que o QuickSort. Abaixo há o código desenvolvido, que também está em anexo casa queira utilizá-lo.

```c
#include <stdio.h>
#include <stdlib.h>

//Teremos duas funções: heapSort e createHeap
void heapSort (int *vet, int n){
    int i, aux;

    //Criando o heap a partir dos elementos (dados)
    for (i=(n-1)/2; i>=0; i--){
        createHeap(vet, i, n-1);    //O createHeap vai estruturar os elementos no formato da árvore
    }

    for (i=n-1; i>=1; i--){
        //Pegando o maior elemento no topo da árvore e colocando na posição i (última posição do vetor)
        aux = vet[0];
        vet[0] = vet[i];
        vet[i] = aux;
        createHeap(vet, 0, i -1); //Reconstrói o heap
    }

   void createHeap(int *vet, int i, int f){
       int aux = vet[i]; //Posição "pai"
       int j = i*2+1; //Primeiro "filho"

       while (j<=f){ //Se o filho for menor ou igual o final do vetor
           if (j<f){ //Se o filho for menor que o final do vetor
               if(vet[j] < vet[j+1]){ //Verifica qual é o maior dentre os filhos
                   j = j + 1; //Filho maior
               }
           }

           if (aux < vet[j]){ //Verifica se o pai é menor que o filho. Se for correto, ele não está na posição correta da Heap.
               vet[i] = vet[j];
               i = j; //O j se torna o pai
               j = 2*i+1; ///Calcula o primeiro filho do j e repete o processo.
           } else {
               j = f + 1;
           }
       }
       vet[i] = aux; //Antigo pai ocupa o lugar do último filho analisado.
   }

}
```

A explicação já está no código, mas para facilitar o entendimento, vou dar uma breve explicação do que está sendo feito:

Explicando o funcionamento do código:

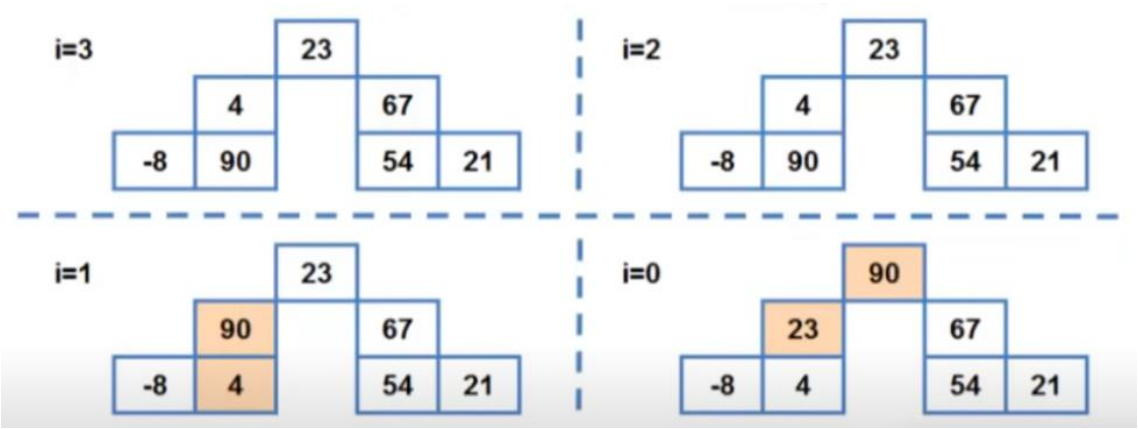
O primeiro comando é o createHeap, que fará com que o elemento pai seja maior que seus filhos.

Na posição em que i é igual a 3, analisamos os elementos da árvore.

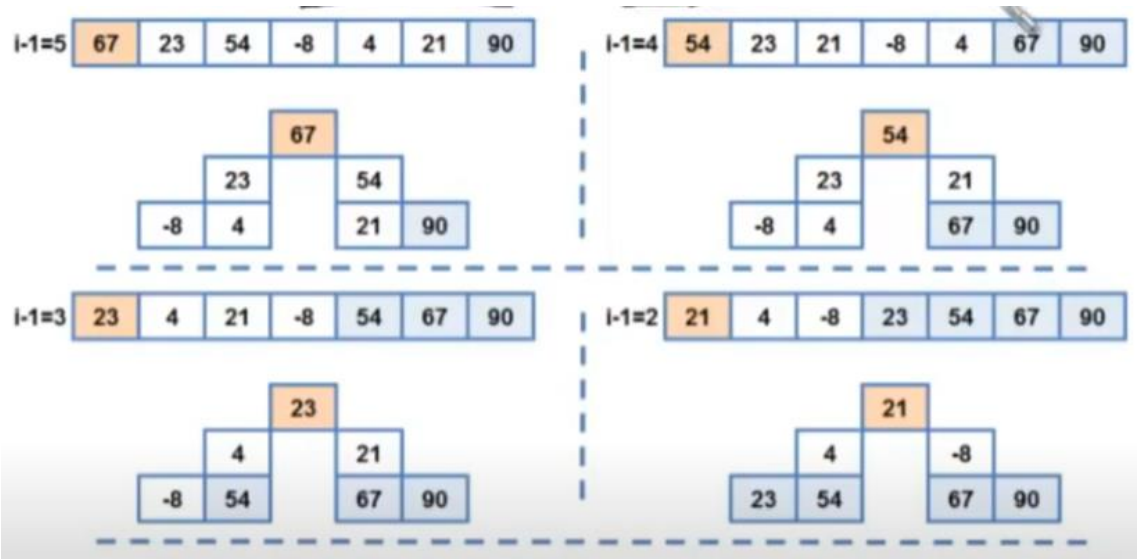
Na posição em que i é igual a 2, conferimos que o pai da direita (67) é maior que seus filhos (54 e 21), portanto não fazemos nada, uma vez que está correto. No entanto, percebe-se que o pai da esquerda (4) é menor que seus filhos (-8 e 90).

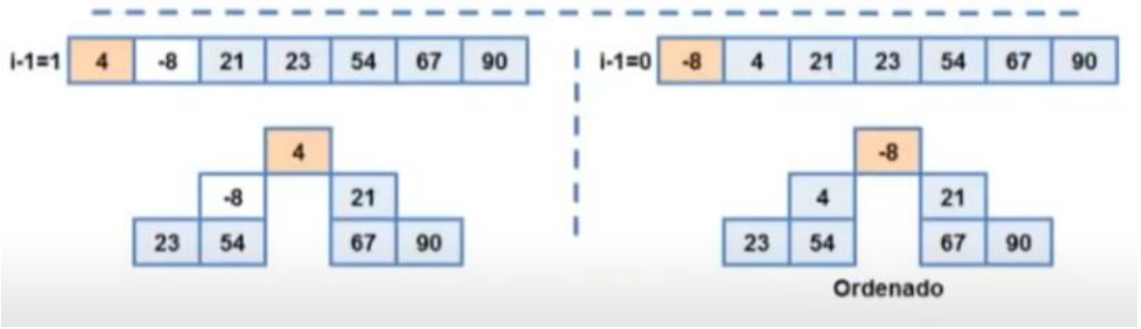
Na posição em que i é igual a 1, trocamos o pai da esquerda, pois o pai deve ser maior que os filhos.

E por fim, na posição em que i é igual a 0, trocamos o pai da esquerda (90) com o pai do topo (23), acertando assim a árvore. Nesse estado, a heap está correta, pois todos os pais são maiores que seus filhos, e no topo, está o maior elemento.



O segundo comando será o heapSort, que moverá o maior elemento para o final e achará o maior do vetor restante.




i-1=1
4
-8
21
23
54
67
90
4
-8
21
23
54
67
90
i-1=0
-8
4
21
23
54
67
90
-8
4
21
23
54
67
90
Ordenado