Estruturas de Dados II
Heap

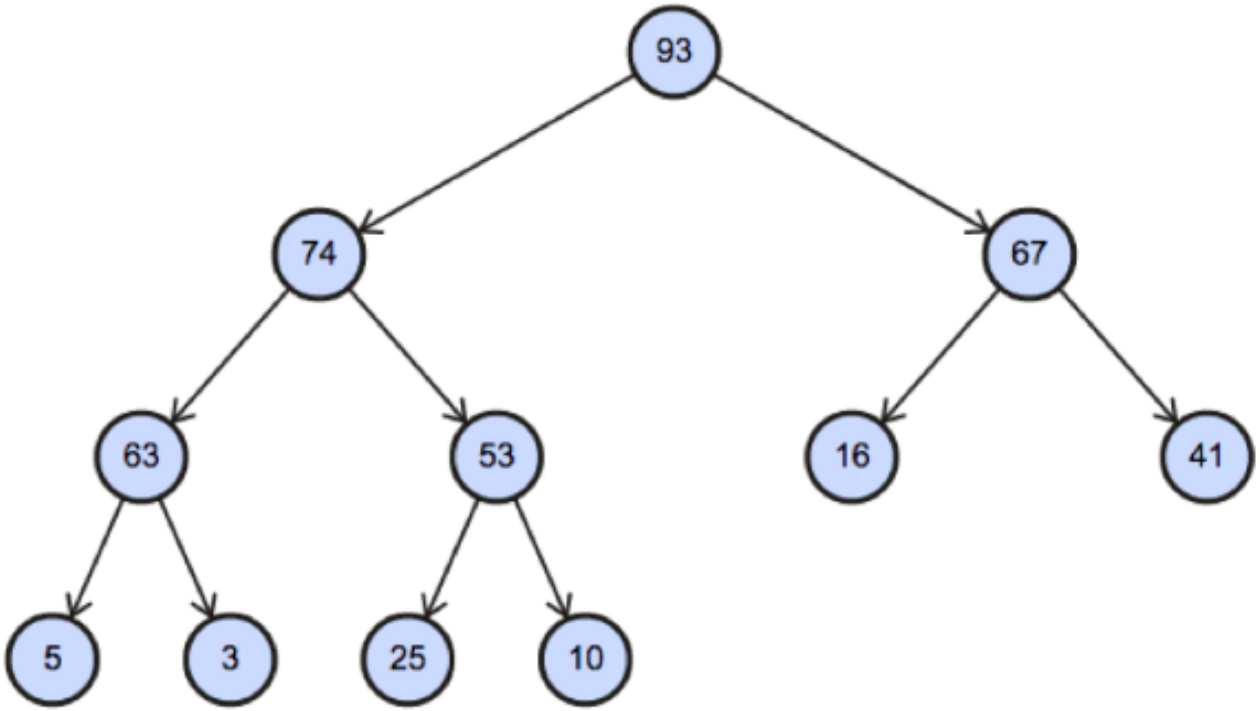
Prof. Leonardo C. R. Soares - *leonardo.soares@ifsudestemg.edu.br*

Instituto Federal do Sudeste de Minas Gerais

7 de março de 2024

Pensando um pouco

Você precisa representar a árvore binária abaixo em um vetor. Como você faria? Qual seria a regra para determinar quem é o antecedente direto de um determinado nó?



Heap

Definição

Heap é uma árvore binária que atende a duas propriedades básicas:

- O valor de um nó deve ser maior (ou menor) ou igual ao valor de seus filhos;
- A árvore binária deve ser completa ou quase-completa da esquerda para a direita.

Heap

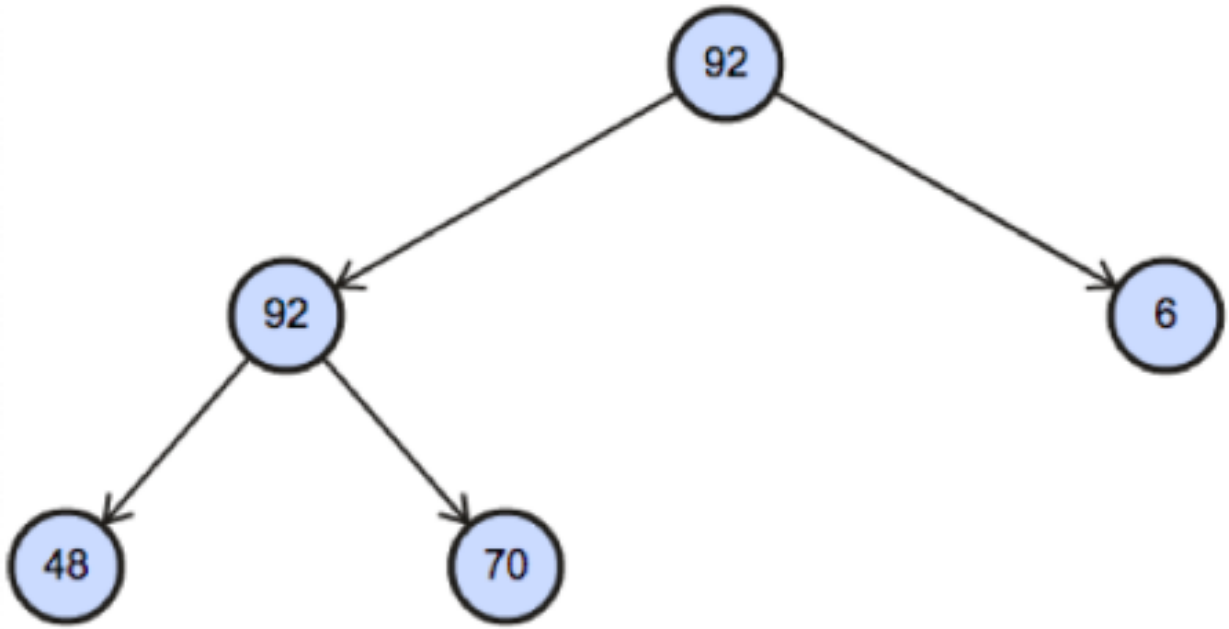
Definição

Heap é uma árvore binária que atende a duas propriedades básicas:

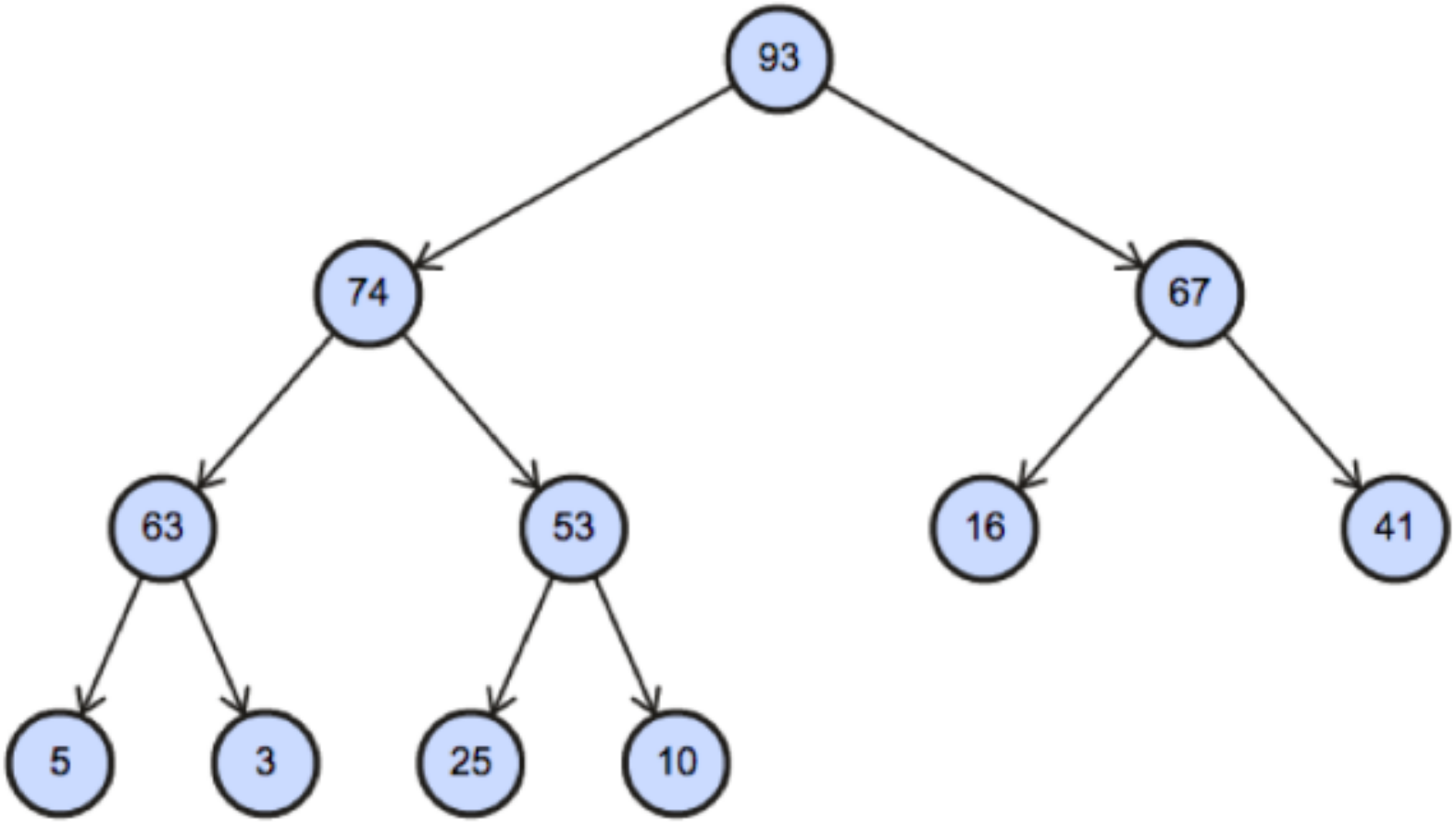
- O valor de um nó deve ser maior (ou menor) ou igual ao valor de seus filhos;
- A árvore binária deve ser completa ou quase-completa da esquerda para a direita.

Caso o valor de um nó seja maior ou igual aos valor de seus filhos, temos um **heap máximo**. Caso o valor seja menor ou igual aos dos filhos, temos um **heap mínimo**.

Heap Máximo



Heap Máximo



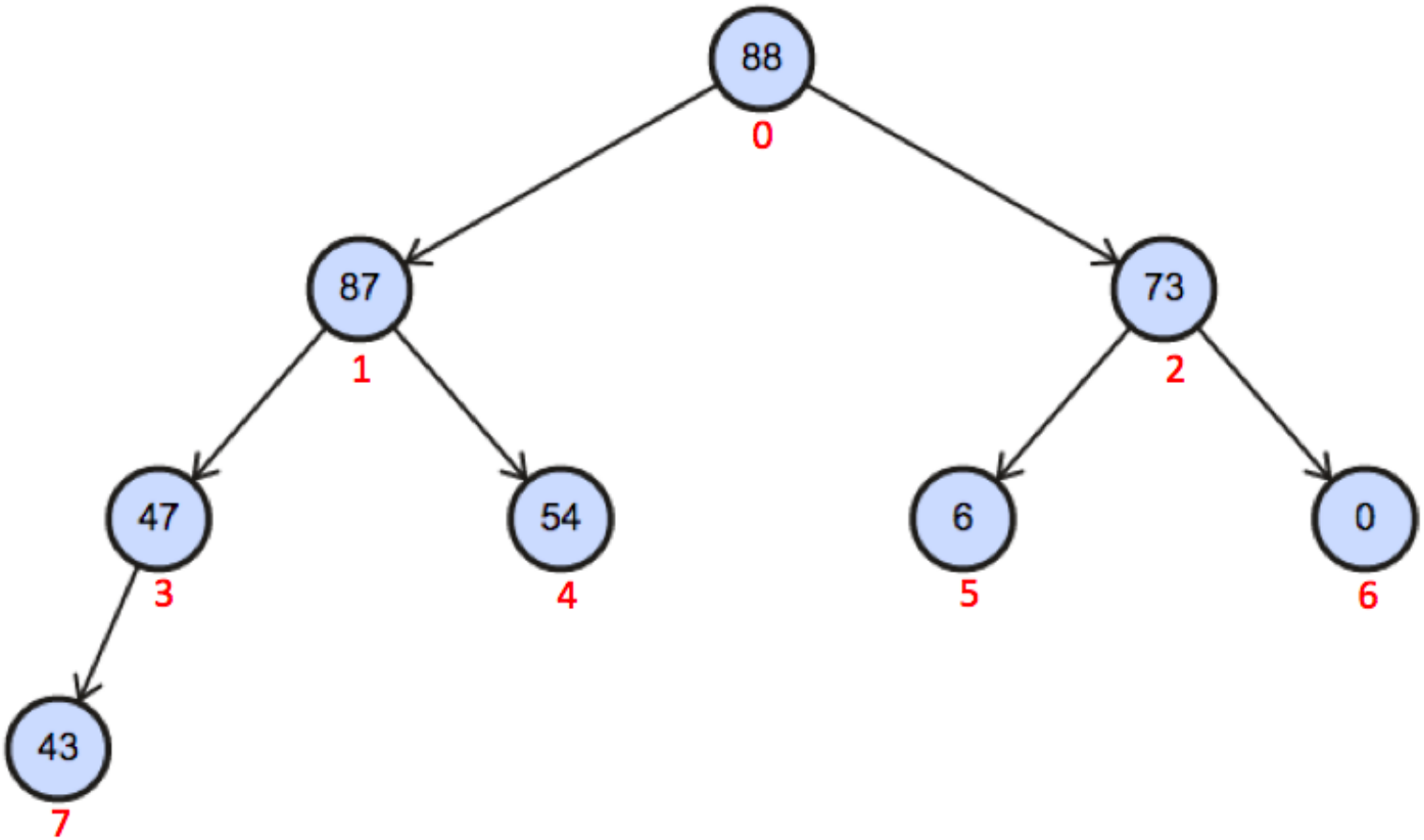
Heap

Dada as propriedades de um Heap, sua altura é $\Theta(\log n)$, visto ser composto por uma árvore binária quase-completa. Essa característica permite que as operações de inserção e remoção sejam eficientes.

Implementação

Utilizaremos um vetor para implementar o Heap. A raiz estará na posição zero do vetor. O **filho da esquerda** de um nó no índice *index* estará sempre na posição $2 * index + 1$. O **filho da direita** de um nó no índice *index* estará sempre na posição $2 * (index + 1)$. O **pai** de um nó no índice *index* estará sempre na posição $\lfloor (index - 1)/2 \rfloor$.

Heap



heap=[88, 87, 73, 47, 54, 6, 0, 43]

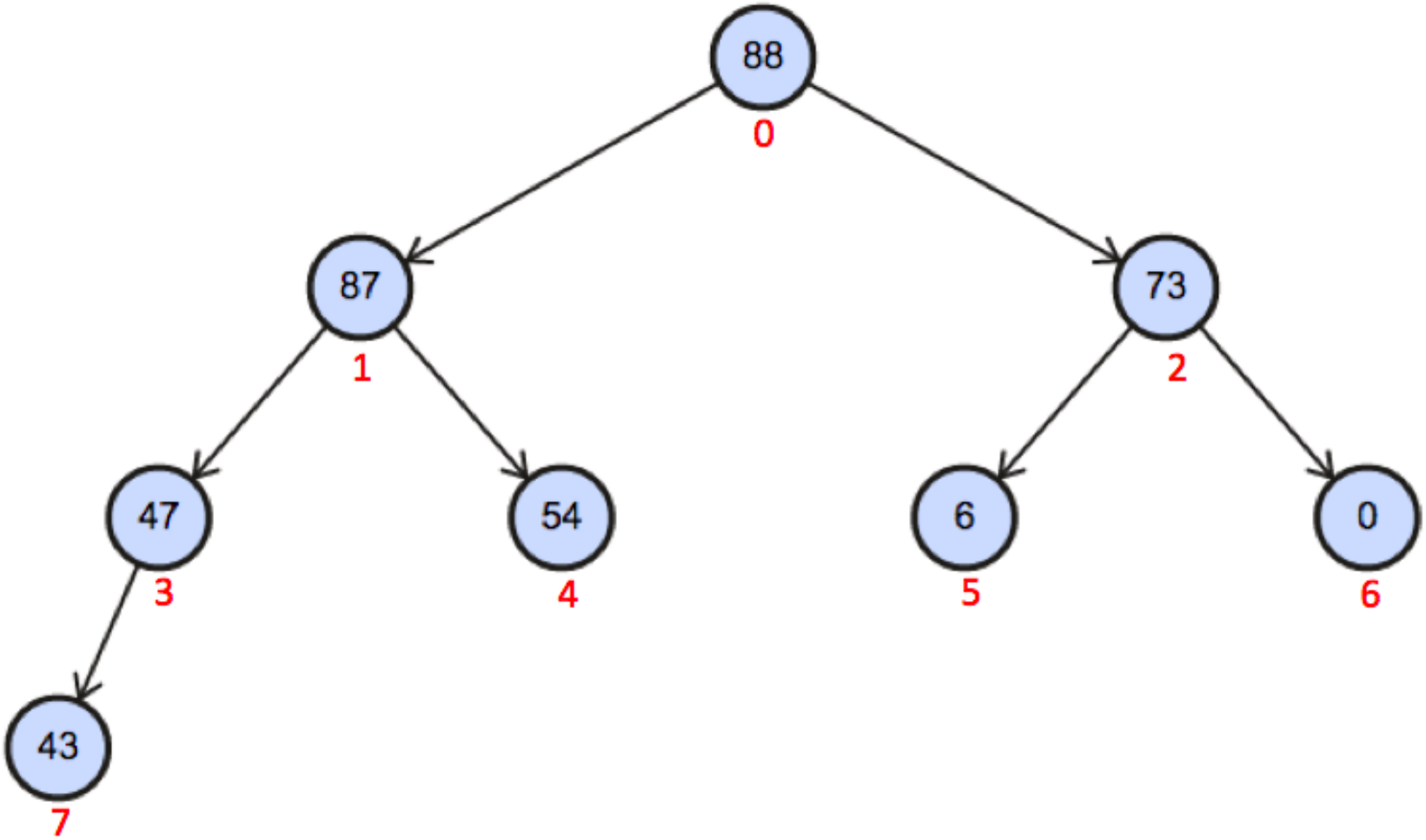
Heap

Inserção

A adição de um novo elemento no heap é sempre feita na próxima posição livre do vetor. Após a inclusão, deve-se mover o nó inserido de modo que a primeira propriedade do heap (o valor de um nó deve ser maior ou igual ao valor de seus filhos) continue verdadeira.

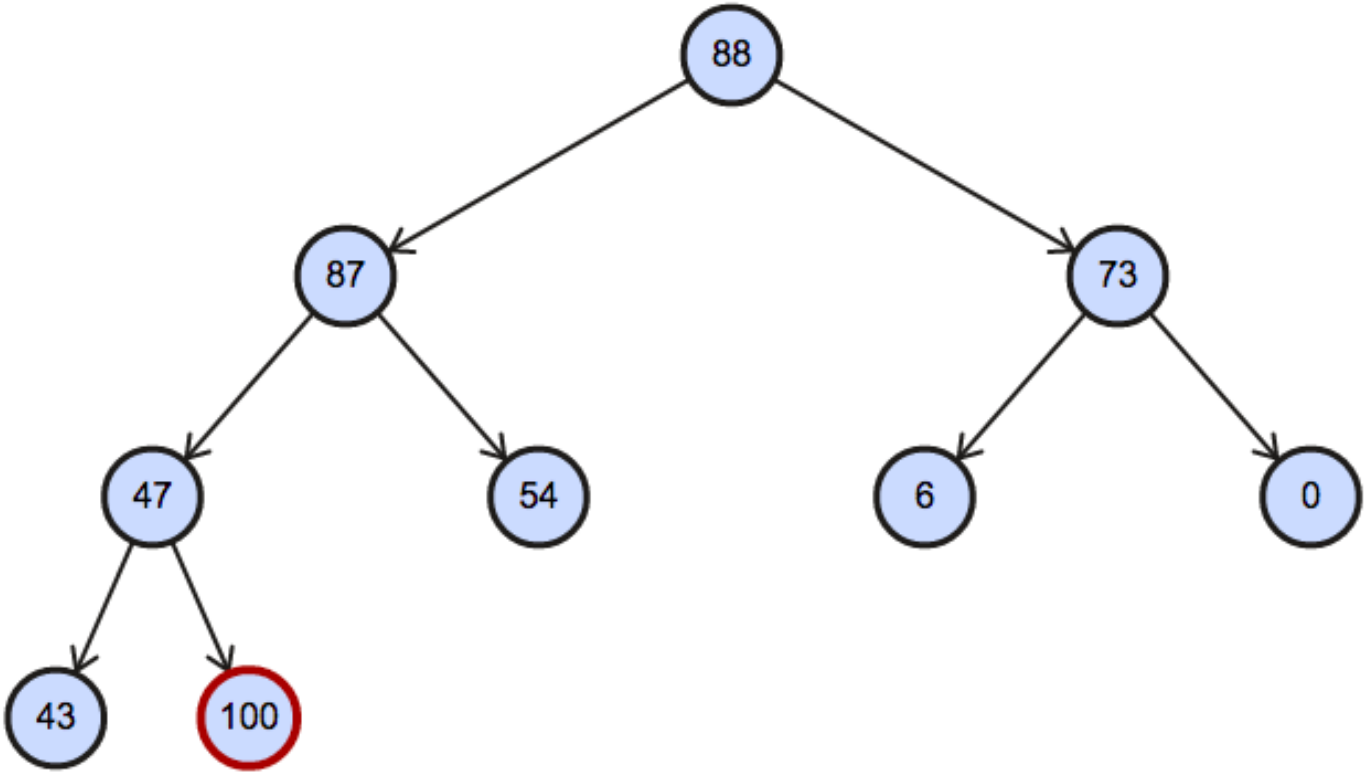
Inserção

Considere que iremos inserir um nó com valor 100 no heap abaixo.



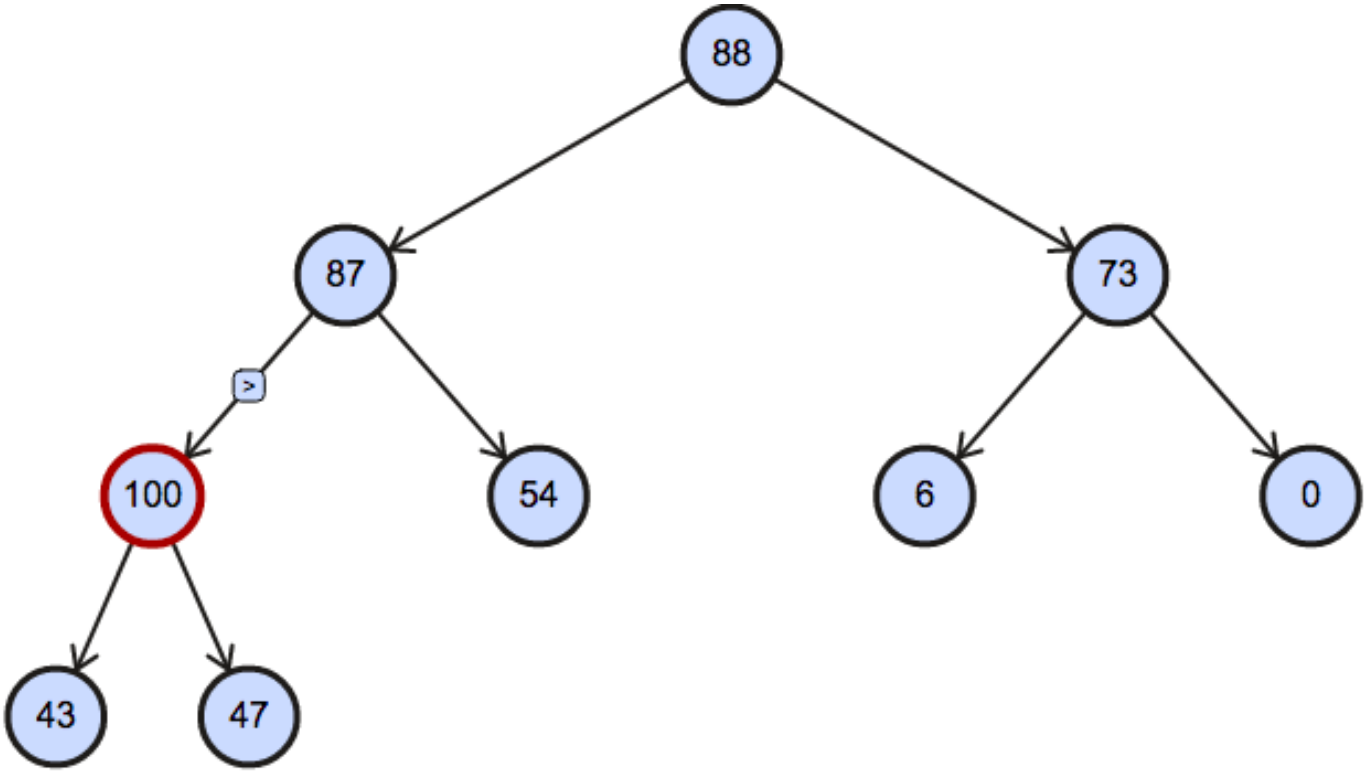
Inserção

A inserção é feita mantendo a árvore quase-completa.



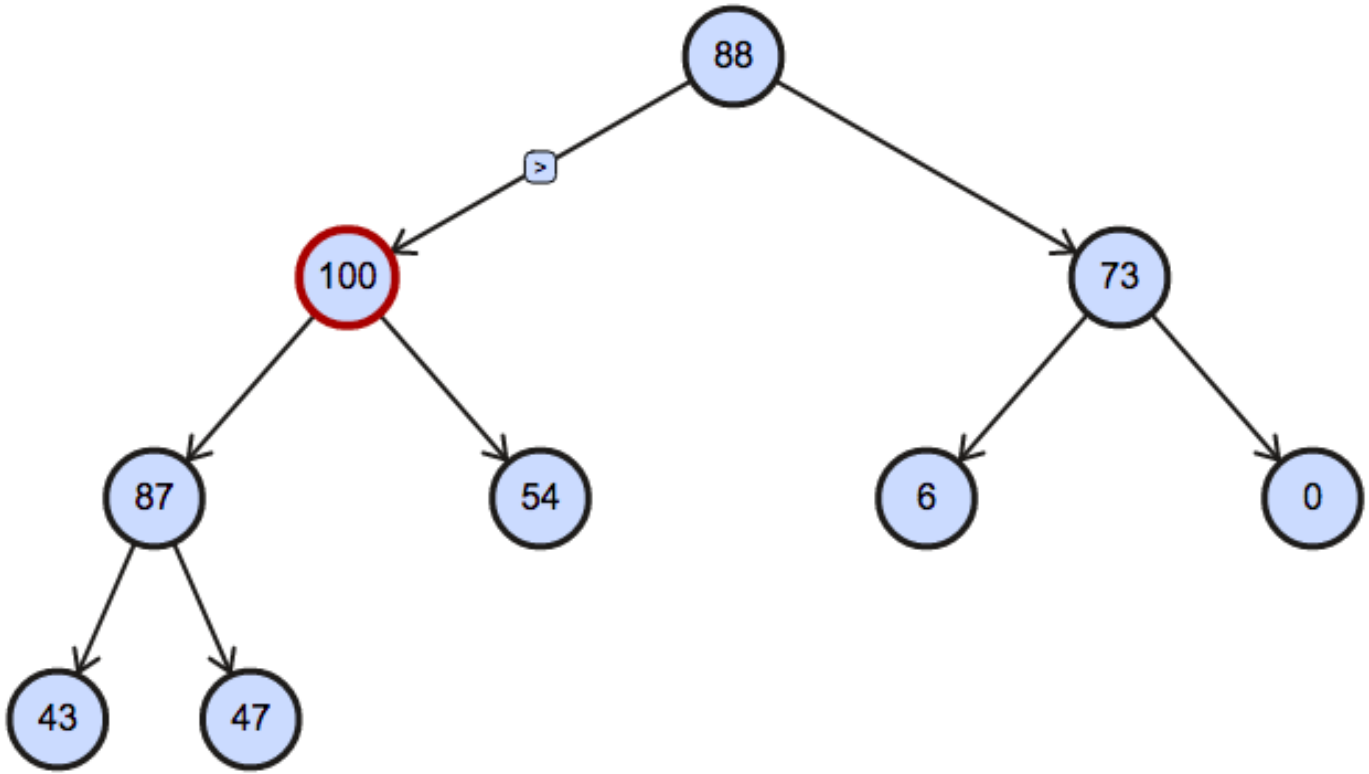
Inserção

Como o nó inserido é maior que seu pai, trocamos os nós de lugar para garantir a primeira propriedade do heap.

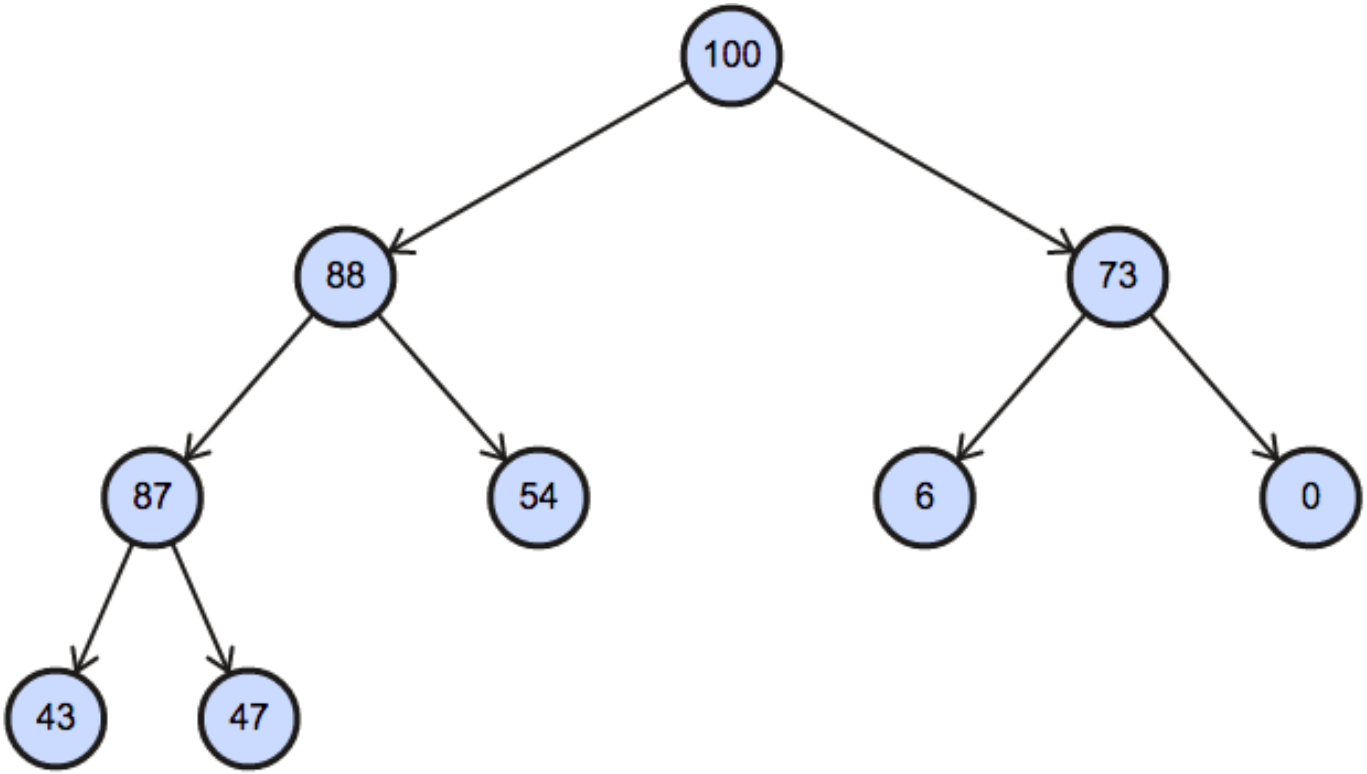


Inserção

Repetimos o passo anterior enquanto for necessário.



Inserção



A inserção de um elemento no heap é O(log n)

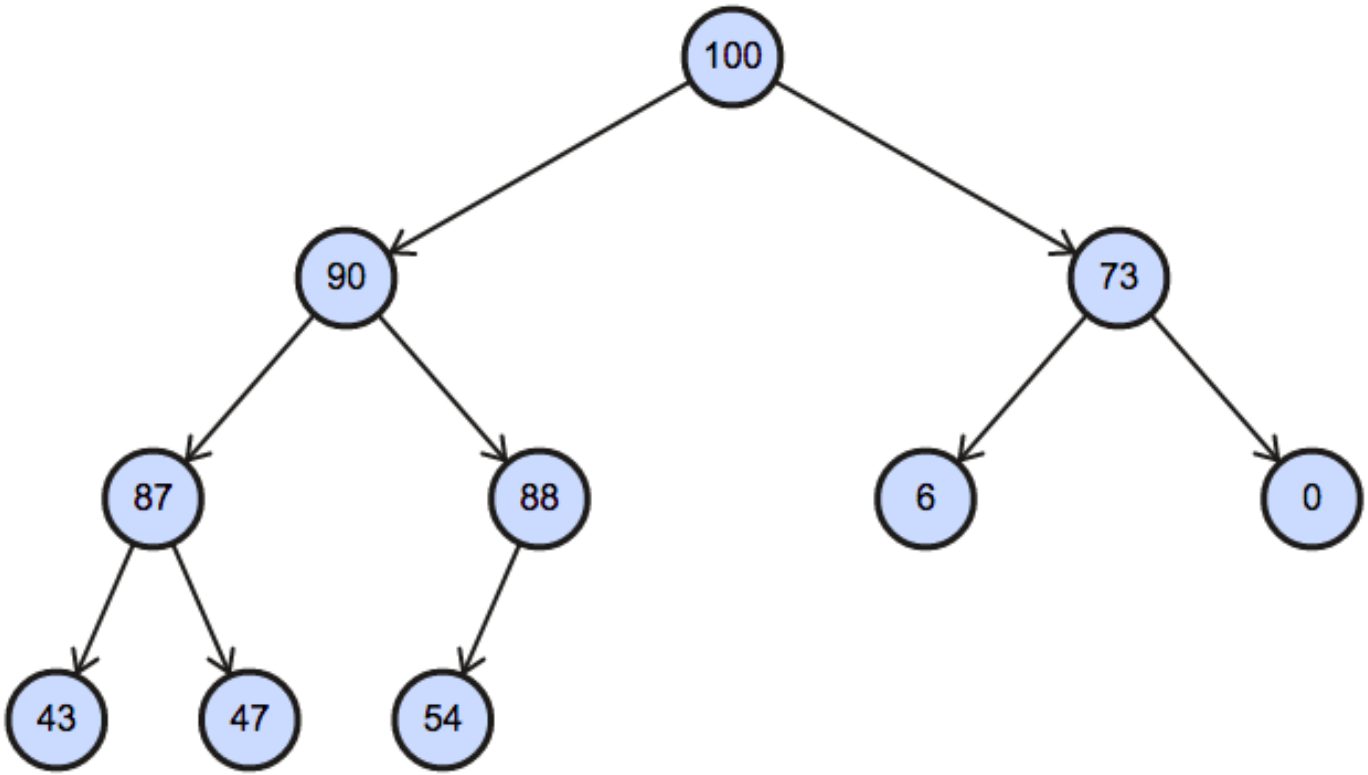
Heap

Remoção

A remoção de um elemento do heap é sempre feita na raiz. Para manter a propriedade de ser uma árvore quase-completa, troca-se o valor da raiz com a última folha e remove-se a folha. Depois disso, devemos corrigir a posição dos elementos do vetor para garantir que a primeira propriedade heap volte a ser verdadeira. Este processo denomina-se **heapify**.

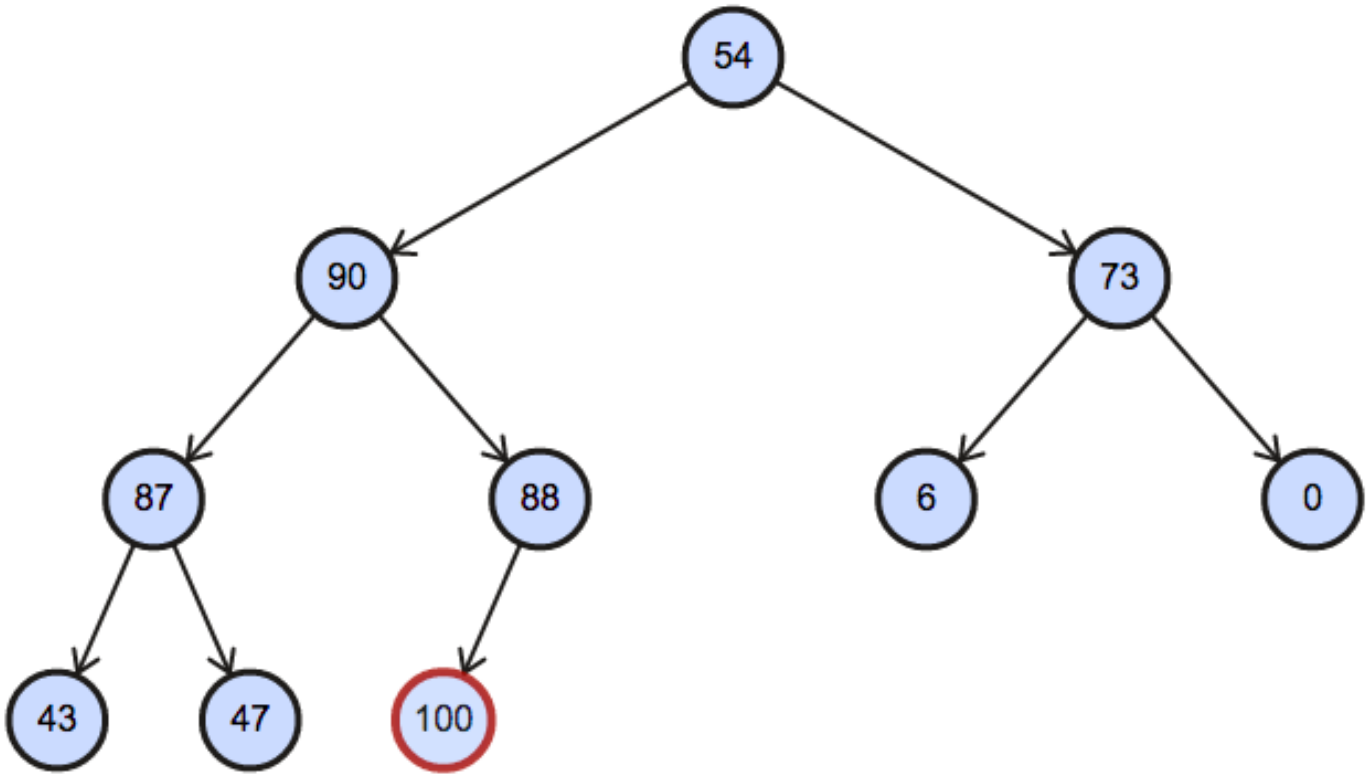
Remoção

Usaremos o heap abaixo para ilustrar o processo de remoção de um nó.



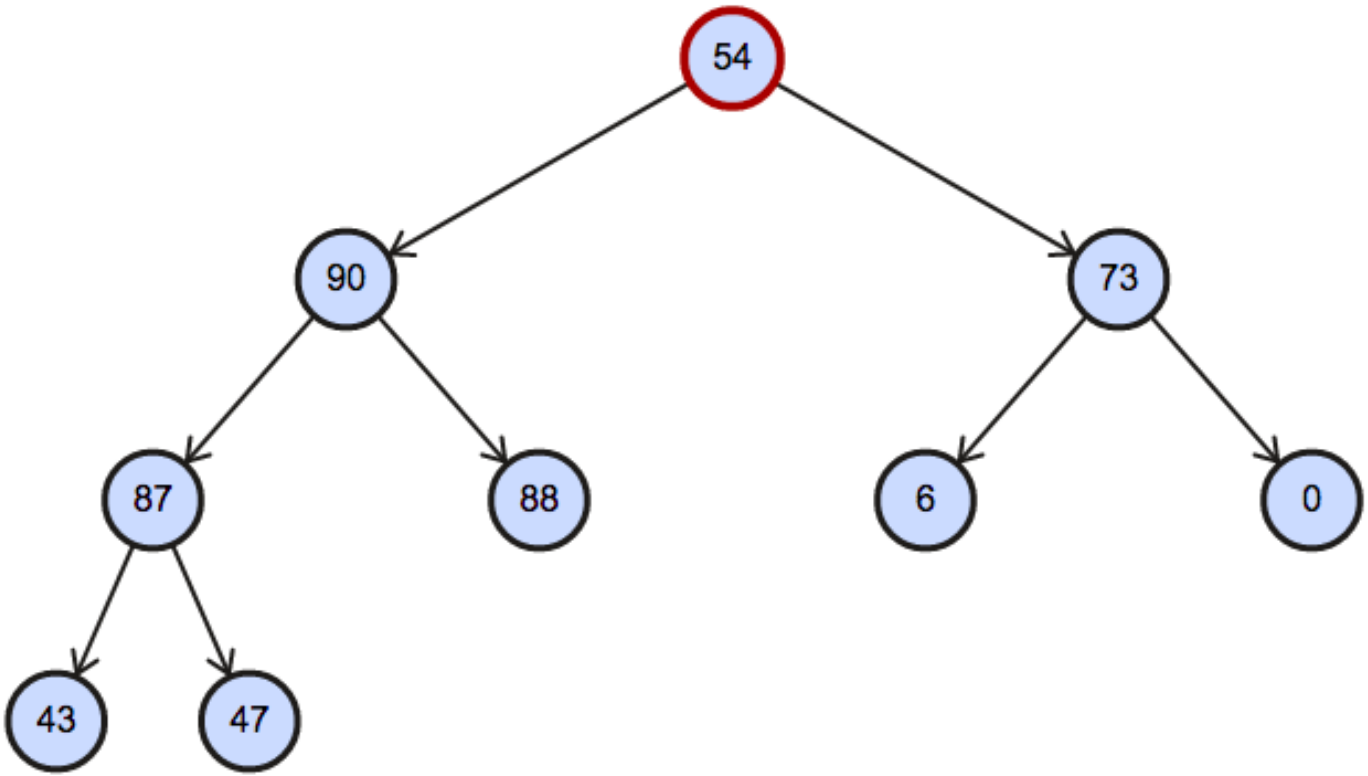
Remoção

Primeiro trocamos a raiz com a última folha.



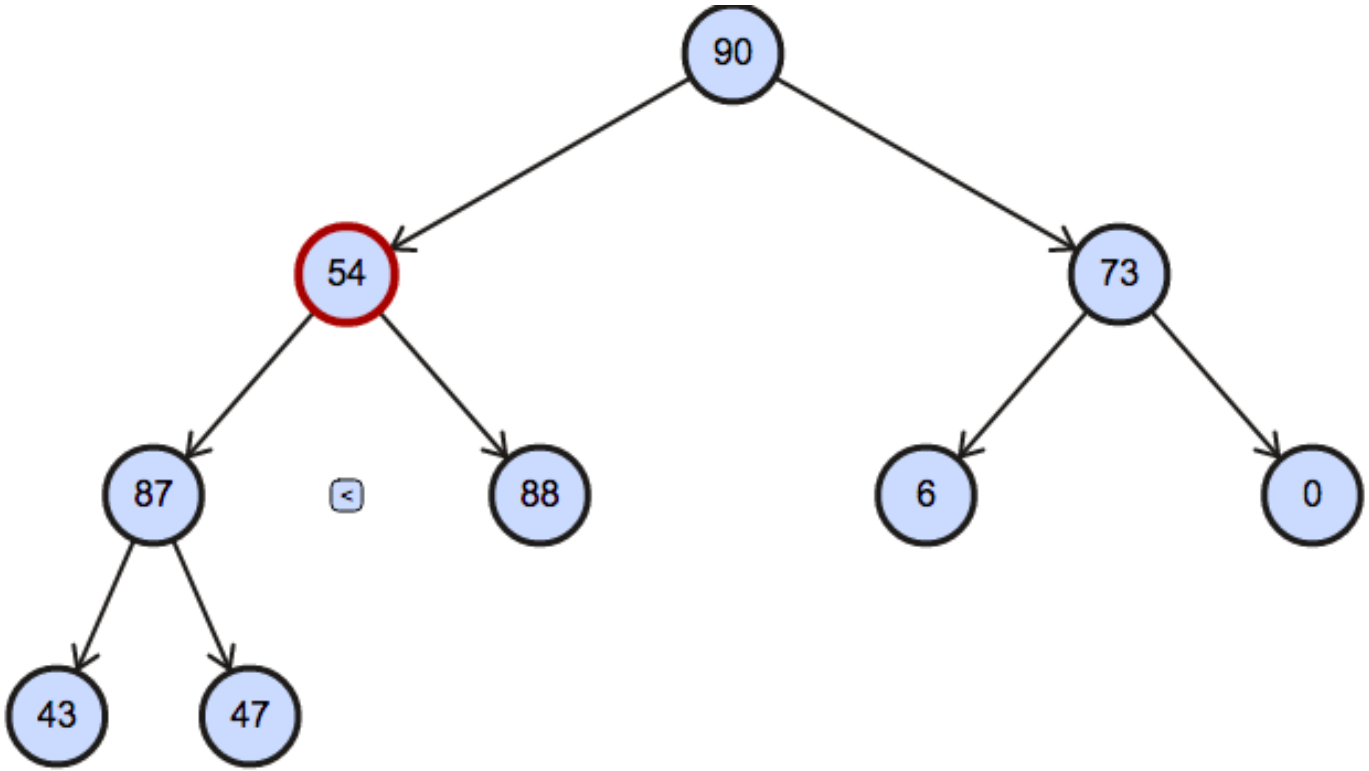
Remoção

Remove-se a última folha.

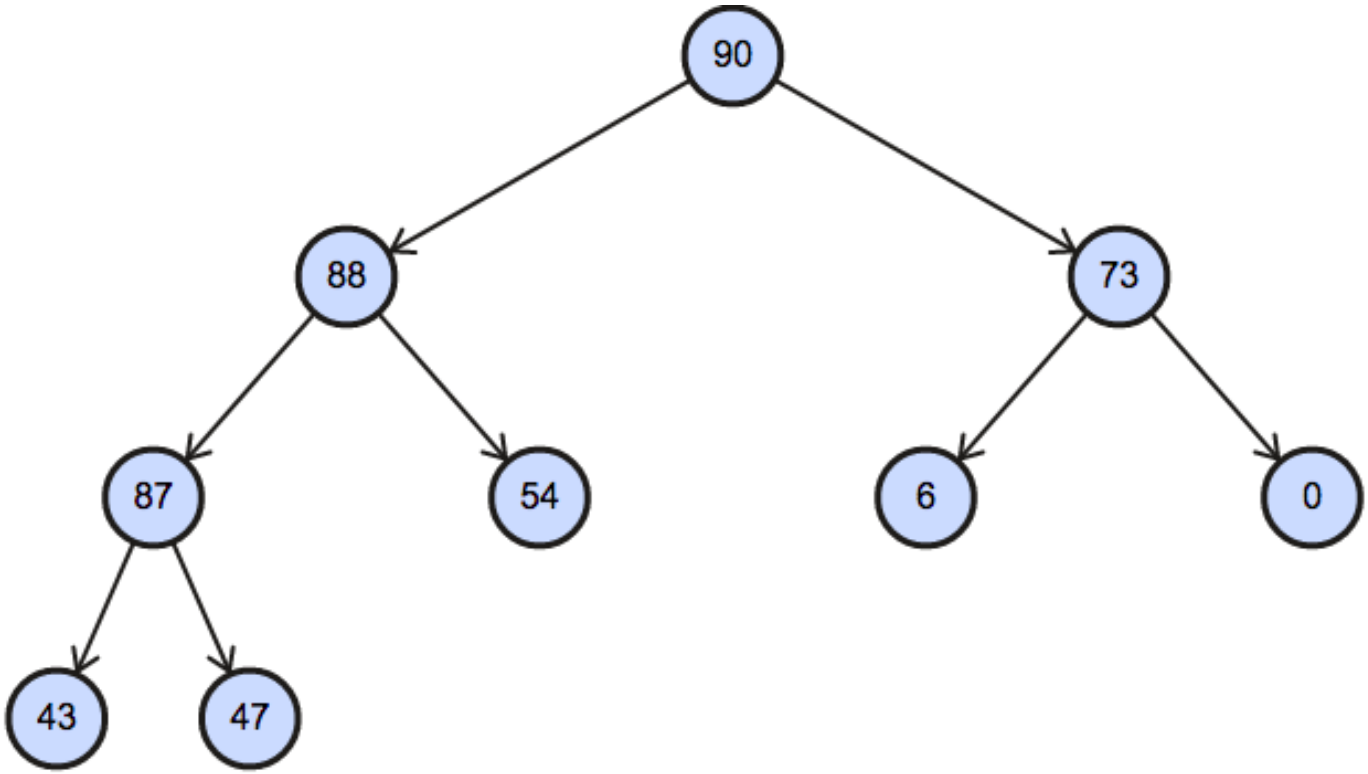


Remoção

O processo de heapify é aplicado à raiz para garantir a primeira propriedade do heap. A raiz é trocada pelo maior de seus filhos. O processo se repete recursivamente até que o heap esteja correto.



Remoção



A remoção de um elemento no heap é O(log n)

Heap

```java
public class Heap{
    private int[] heap;
    private int cauda;
    public Heap(int tam){
        this.heap = new int[tam];
        this.cauda = -1;
    }
    public Heap(int[] v){
        this.heap = v;
        this.cauda = this.heap.length-1;
        this.criaHeap();

    }
```


Heap

```
14      private boolean isVazio(){ return this.cauda == -1;}
15      private int left(int i) { return 2* i + 1; }
16      private int right(int i){ return 2*(i+1); }
17      private int parent(int i){ return (i-1)/2; }
18      private boolean isValido(int i){ return i >=0 && i <= cauda; }
19      private boolean isFolha(int i){ return i > parent(cauda) && i <= cauda; }
20
```

Heap

```
21      public void add(int n) throws Exception {
22          if (cauda >= heap.length-1)
23              throw new Exception(message: "Sem espaço disponível");
24          this.heap[++cauda] = n;
25          int i = cauda;
26          while (i > 0 && this.heap[parent(i)] < this.heap[i]){
27              int aux = this.heap[i];
28              this.heap[i] = this.heap[parent(i)];
29              this.heap[parent(i)] = aux;
30              i = parent(i);
31          }
32      }
33
```

Heap

```
34      public int remove() throws Exception{
35          if (isVazio()) throw new Exception(message: "O Heap está vazio");
36          int elemento = this.heap[0];
37          this.heap[0] = this.heap[cauda--];
38          this.heapify(i: 0);
39          return elemento;
40      }
41
```

Heap

```
42      private void heapify(int i){
43          if (isFolha(i) || !isValido(i))
44              return;
45          int maior = max_index(i, left(i), right(i));
46          if (maior != i){
47              swap(i,maior);
48              heapify(maior);
49          }
50      }
```

Heap

```
51      private int max_index(int i, int esq, int dir){
52          int maior = i;
53          if (isValido(esq) && this.heap[esq] > this.heap[maior])
54              maior = esq;
55          if (isValido(dir) && this.heap[dir] > this.heap[maior])
56              maior = dir;
57          return maior;
58      }
59      private void swap(int i, int j){
60          int aux = this.heap[i];
61          this.heap[i] = this.heap[j];
62          this.heap[j] = aux;
63      }
```

Heap

```
64
65      private void criaHeap(){
66          for(int i=parent(this.cauda); i>=0; i--)
67              heapify(i);
68      }
69
70      public void print(){
71          for (int i=0; i<=cauda;i++)
72              System.out.print(this.heap[i] + " ");
73      }
```

Heap

```
74      public static void main(String[] args) throws Exception{
75          Heap heap = new Heap(tam: 10);
76          for (int i=0;i<10;i++){
77              heap.add((int)((Math.random()+1)*100));
78          }
79          heap.print();
80          System.out.println("\n" + heap.remove());
81          heap.print();
82          System.out.println("\n" + heap.remove());
83          System.out.println();
84          heap.print();
85          System.out.println();
86          int[] x = {1,2,3,4,5,6,7,8,9,10,30};
87          heap = new Heap(x);
88          heap.print();
89      }
90  }
```

HeapSort

Definição

Utilizando-se a estrutura de dados heap é possível implementar um algoritmo eficiente de ordenação **O(n log n)**. O algoritmo é extremamente simples:

1. Construa o heap à partir do vetor original;
2. Troque a raiz com a última folha;
3. Diminua o tamanho do heap em uma unidade;
4. Reconstrua o heap com o método heapfy;
5. Repita os passos 2 a 4 até o heap ter tamanho 1.

HeapSort

```java
public class HeapSort {
    public static void heapSort(int[] v){
        int N = v.length;
        // Transforma o vetor em um heap
        // N / 2 -1 = pai da última folha.s
        for (int i = N /2 -1; i>=0; i--)
            heapify(v,N,i);
        // Trocar a raiz com o último elemento
        // Diminuir o tamanho do vetor
        // Rearranjar o heap
        for (int i = N-1; i > 0; i--){
            int temp = v[0];
            v[0] = v[i];
            v[i] = temp;
            heapify(v,i,i: 0);
        }
    }
```

HeapSort

```java
// vetor, tamanho, raiz
private static void heapify(int arr[], int N, int i) {
    int maior = i; // Considera a raiz como maior;
    int l = 2 * i + 1; // filho da esquerda
    int r = 2 * i + 2; // filho da direita
    // Se o filho da esquerda é maior que a raiz
    if (l < N && arr[l] > arr[maior])
        maior = l;
    // Se o filho da direita é maior que o maior até agora
    if (r < N && arr[r] > arr[maior])
        maior = r;
    // Se maior não é a raiz
    if (maior != i) {
        int swap = arr[i];
        arr[i] = arr[maior];
        arr[maior] = swap;
        // Recursivamente, heapfy a subárvore afetada
        heapify(arr, N, maior);
    }
}
```

HeapSort

```
    public static void main(String[] args){
        int[] vet = {12,11,5,3,9,7,21,13,11};
        heapSort(vet);
        for(int valor : vet)
            System.out.print(valor + " ");
    }
}
```

Exercícios

1. Implemente os exemplos apresentados.
2. Implemente um Heap Mínimo.
3. Como ficariam as fórmulas para definir os filhos de um heap cujo índice inicial do vetor fosse 1?
4. Em qual nível pode estar o segundo maior elemento de um heap máximo?
5. Escreva um método que verifique se um determinado vetor é um heap máximo.


рахмат
danke
謝謝
ngiyabonga
teşekkür ederim
Баярлалаа
спасибо
thank you
gracias
bedankt
dziękuję
obrigado
mochchakkeram
go raibh maith agat
sukriya
kop khun krap
arigatô
dakujem
мерси
terima kasih
grazie
merci
감사합니다
ευχαριστώ
ありがとう
tapadh leat
dank je
mersi
barka
welalin
tack
vinaka
blagodaram
misaotra
matondo
paldies
grazzi
mahalo
хвала
asante
manana
obrigada
tenki
murakoze
chokrane
mamnun
дякую
djiere dieuf
tau
trugarez
shukriya
merce
takk
dhanyavadagalu
diolch
tanemirt
rahmet
xiexie
sulpay
taiku
chnorakaloutioun
gratias ago
gracies
najis tuke
kam sah hamnida
didi madloba
sagolun
mèsi
sobodi
dėkuji
enkosi
bayarlalaa
gracie
hvala
mauruuru
koszonom
dhanyavad
kiitos
dankie
nanni
nandri
faafetai lava
спасибі
kia ora
spas
dankon
aciu
akun
তোমাকে ধন্যবাদ