CENTRO UNIVERSITÁRIO UNIVEL
Av. Tito Muffato, n. 2317, Bairro Santa
Cruz CEP: 85806-080, Cascavel (PR)
Fone: (45) 3036-3653 | Fax: (45) 3036-3638
www.univel.br

Profº: Daniel

Disciplina: Estrutura de dados

Turma: 3bN

Aluno: Vicente Joel Freitas de Souza

1 – O que é Heap em estrutura de dados?

Uma Heap é uma estrutura de dados especial que satisfaz a propriedade heap: o pai de um nó é sempre menor (ou maior) que os seus filhos. Isso significa que o nó raiz da Heap é sempre o menor (ou maior) elemento da estrutura. Existem dois tipos de Heaps: Min-Heap e Max-Heap.

2 – Quais são as características principais?

As principais características de uma Heap são:

Ordem: Os elementos da Heap estão ordenados de acordo com a propriedade heap.

Raiz: O nó raiz da Heap é sempre o menor (ou maior) elemento da estrutura.

Inserção e remoção: É possível inserir e remover elementos da Heap de forma eficiente.

Busca: É possível buscar um elemento específico na Heap, mas não é tão eficiente quanto em outras estruturas de dados.

3 – Crie dois códigos, um em python e o outro em javascript.

Exemplos diferentes e comente o código.

```python
# Classe MinHeap que implementa uma Min-Heap
class MinHeap:
    def __init__(self):
        # Inicializa a Heap como uma lista vazia
        self.heap = []

    # Método para inserir um elemento na Heap
    def insert(self, valor):
        # Adiciona o elemento ao final da lista
```

```
        self.heap.append(valor)
        # Chama o método _heapify_up para
garantir que a propriedade heap seja mantida
        self._heapify_up(len(self.heap) - 1)

    # Método para garantir que a propriedade heap
seja mantida após a inserção de um elemento
    def _heapify_up(self, indice):
        # Enquanto o elemento não estiver na raiz
da Heap
        while indice > 0:
            # Calcula o índice do pai do elemento
            pai = (indice - 1) // 2
            # Se o pai for menor ou igual ao
elemento, não é necessário fazer nada
            if self.heap[pai] <=
self.heap[indice]:
                break
            # Troca o elemento com o pai
            self.heap[pai], self.heap[indice] =
self.heap[indice], self.heap[pai]
            # Atualiza o índice para o pai
            indice = pai
```

```
    # Método para remover o menor elemento da
Heap
    def remove_min(self):
        # Se a Heap estiver vazia, retorna None
        if len(self.heap) == 0:
            return None
        # Se a Heap tiver apenas um elemento,
retorna o elemento e remove-o da lista
        if len(self.heap) == 1:
            return self.heap.pop()
        # Armazena o menor elemento
        minimo = self.heap[0]
        # Substitui o menor elemento pelo último
elemento da lista
        self.heap[0] = self.heap.pop()
        # Chama o método _heapify_down para
garantir que a propriedade heap seja mantida
        self._heapify_down(0)
        # Retorna o menor elemento
        return minimo

    # Método para garantir que a propriedade heap
seja mantida após a remoção do menor elemento
    def _heapify_down(self, indice):
```

```
        # Enquanto o elemento não estiver na raiz
da Heap
        while True:
            # Calcula os índices dos filhos do
elemento
            filho_esquerdo = 2 * indice + 1
            filho_direito = 2 * indice + 2
            # Armazena o índice do menor filho
            menor = indice
            # Se o filho esquerdo for menor que o
elemento, atualiza o índice do menor filho
            if filho_esquerdo < len(self.heap)
and self.heap[filho_esquerdo] < self.heap[menor]:
                menor = filho_esquerdo
            # Se o filho direito for menor que o
elemento, atualiza o índice do menor filho
            if filho_direito < len(self.heap) and
self.heap[filho_direito] < self.heap[menor]:
                menor = filho_direito
            # Se o elemento for o menor, não é
necessário fazer nada
            if menor == indice:
                break
            # Troca o elemento com o menor filho
```

```
            self.heap[indice], self.heap[menor] =
self.heap[menor], self.heap[indice]
            # Atualiza o índice
```

Agora em Javascript

```
// Classe MinHeap que implementa uma Min-Heap
class MinHeap {
  // Construtor que inicializa a Heap como uma
lista vazia
  constructor() {
    this.heap = [];
  }

  // Método para inserir um elemento na Heap
  insert(valor) {
    // Adiciona o elemento ao final da lista
    this.heap.push(valor);
    // Chama o método _heapifyUp para garantir
```

```
que a propriedade heap seja mantida
    this._heapifyUp(this.heap.length - 1);
  }

  // Método para garantir que a propriedade heap
seja mantida após a inserção de um elemento
  _heapifyUp(indice) {
    // Enquanto o elemento não estiver na raiz da
Heap
    while (indice > 0) {
      // Calcula o índice do pai do elemento
      const pai = Math.floor((indice - 1) / 2);
      // Se o pai for menor ou igual ao elemento,
não é necessário fazer nada
      if (this.heap[pai] <= this.heap[indice]) {
        break;
      }
      // Troca o elemento com o pai
      [this.heap[pai], this.heap[indice]] =
[this.heap[indice], this.heap[pai]];
      // Atualiza o índice para o pai
      indice = pai;
    }
  }
```

```
  // Método para remover o menor elemento da Heap
  removeMin() {
    // Se a Heap estiver vazia, retorna null
    if (this.heap.length === 0) {
      return null;
    }
    // Se a Heap tiver apenas um elemento,
retorna o elemento e remove-o da lista
    if (this.heap.length === 1) {
      return this.heap.pop();
    }
    // Armazena o menor elemento
    const minimo = this.heap[0];
    // Substitui o menor elemento pelo último
elemento da lista
    this.heap[0] = this.heap.pop();
    // Chama o método _heapifyDown para garantir
que a propriedade heap seja mantida
    this._heapifyDown(0);
    // Retorna o menor elemento
    return minimo;
  }
```

```
  // Método para garantir que a propriedade heap
seja mantida após a remoção do menor elemento
  _heapifyDown(indice) {
    // Enquanto o elemento não estiver na raiz da
Heap
    while (true) {
      // Calcula os índices dos filhos do
elemento
      const filhoEsquerdo = 2 * indice + 1;
      const filhoDireito = 2 * indice + 2;
      // Armazena o índice do menor filho
      let menor = indice;
      // Se o filho esquerdo for menor que o
elemento, atualiza o índice do menor filho
      if (filhoEsquerdo < this.heap.length &&
this.heap[filhoEsquerdo] < this.heap[menor]) {
        menor = filhoEsquerdo;
      }
      // Se o filho direito for menor que o
elemento, atualiza o índice do menor filho
      if (filhoDireito < this.heap.length &&
this.heap[filhoDireito] < this.heap[menor]) {
        menor = filhoDireito;
      }
```

```
      // Se o elemento for o menor, não é
necessário fazer nada
      if (menor === indice) {
        break;
      }
      // Troca o elemento com o menor filho
      [this.heap[indice], this.heap[menor]] =
[this.heap[menor], this.heap[indice]];
      // Atualiza o índice
      indice = menor;
    }
  }
}

// Exemplo de uso
const heap = new MinHeap();
heap.insert(5);
heap.insert(3);
heap.insert(8);
heap.insert(1);
console.log(heap.removeMin()); // imprime 1
console.log(heap.removeMin()); // imprime 3
```

Entrega: daniel.silva@univel.br