Relatório - Heap

1. Descrição do Algoritmo

A estrutura trabalhada neste tópico é uma variação de árvore, no caso do projeto em questão, especificamente do tipo binária completa (todos nós folhas no mesmo nível, com uma possível exceção quanto ao último nível, que no caso de não estar completo, deve estar preenchido da esquerda para a direita). Uma de suas características é que a raíz deve ser o nó de maior valor dentre todos os filhos, sendo que a mesma regra deve ser aplicada para os outros nós, ou seja, os pais devem possuir obrigatoriamente valores maiores do que os dos seus filhos (Max Heap). Outra propriedade importante da Heap é o fato de ser uma estrutura plenamente balanceada, devido aos seus métodos de inserção, remoção e sift.

2. Pseudo-código

```
class Node(value)

        initialize value to value parameter
        initialize previous to null
        initialize next to null

class Heap

        initialize nodes' array to empty array

        method getParentIndex (parameters: number to node index): integer
                return index – 1)/2

        method getLeftChildIndex (parameters: number to node index): integer
                return index*2 + 1

        method getRightChildIndex (parameters: number to node index): integer
                return index*2 + 2

        method getLastParentIndex(): integer
                return the result of getParentIndex function for the last node

        method insert(parameters: number to value) : void
                if the value parameter was passed
                        create a new instance of node with the passed value
                        push to nodes' array the new node
                        organize the structure calling the siftUp function
                endif

        method siftUp():void
                creates a variable initialized with the last node index to current node
                while current node has a parent and it's value be bigger than it's parent value
                        swap the node value with it's parent value
```

set current node as it's parent value
endwhile

method remove() : void
if the nodes' array is empty
return false

otherwise
get the value of the node to be removed, in this case, the root
change the root value with last node value
decrements the array size (ensures that the element will not be in the array)

organize the structure calling the siftDown function
endif

method siftDown():void
creates a variable initialized with first node index (zero) to the current node

while current node left child exists
creates a variable to the node with the biggest value (firstly initialized with the left child node)
if current node right child exists and it's value be bigger than the left child
sets the biggest value to the right child node
endif

if current node value be less than the biggest node value
swap the current node and biggest node values and indexes
otherwise
leave
endif
endwhile

method search(paramaters: number to value): boolean

for index in range of 0 to length of nodes' array, compare current element and value
if they're equals return true
endfor

if the values was not found, return false

method clear(): void
set nodes' array to empty array

3. Vantagens e desvantagens

3.1. Vantagens

- É muito eficiente quanto a operação de inserção e remoção, uma heap binária possui sempre altura logarítmica, e, portanto, leva o tempo classificado em O (log2n) (podendo ser até O(1) no melhor caso)

- Também é eficiente para operação de busca do maior valor (em MaxHeaps, tipo trabalhado neste projeto), já que este é a sua raiz (O(1))
- Seu tamanho é flexível
- Fácil representação: Pode ser representado por meio de um array com n elementos, sendo n o número de nós inseridos

3.2. Desvantagens

- Busca geral lenta, levando o tempo de O(n)