N584 – Projeto e Análise de Algoritmos

Prof. Napoleão Nepomuceno

AV2 - Lab01

Data do Laboratório: 25/09/2019

Márcio Heleno Matrícula: 1814038

Entrega do trabalho: Data da Entrega: 29/09/2019 (enviar arquivo .odt)

Exercício 1 - Av2

- **Passo 1:** Implementar o seguinte código em Java ou equivalente em outra linguagem de programação.

```
import java.util.Random;
public class HeapSort {
    public static void main(String[] args) {
        int[] A = {86, 78, 36, 61, 67, 34, 58, 42, 35, 59, 62, 28, 60, 37, 12};
        System.out.println("Vetor A:");
        imprimeVetor(A);
        System.out.println("Heap A:");
        imprimeHeap(A);
        //int[] B = criaVetorAleatorio(15);
        //System.out.println("-------------------------------------------------------------------
        //System.out.println("Vetor B:");
        //imprimeVetor(B);
        //imprimeHeap(B);
    }

    static int left (int i) {
        // to do
    }

    static int right (int i) {
        // to do
    }

    static void maxheapfy (int[] V, int i) {
        int l = left(i);
        int r = right(i);
        int m = i;
        // to do
```

```
}

static void buildmaxheap (int[] V) {
    // to do
}

static int[] criaVetorAleatorio (int n) {
    Random randomGenerator = new Random();
    int[] A = new int[n];
    for (int i = 0; i < n; i++) {
        A[i] = randomGenerator.nextInt(10 * n);
    }
    return A;
}

static void imprimeVetor (int[] A) {
    for (int i = 0; i < A.length; i++) {
        System.out.printf("%6d", A[i]);
    }
    System.out.print("\n\n\n");
}

static void imprimeHeap (int[] A) {
    int h = (int) (Math.log(A.length) / Math.log(2));
    int espacos = calculaEspacos(h);
    for (int i = 0; i <= h; i++) {
        for (int j = 1; j <= Math.pow(2, i); j++) {
            if ((int) (Math.pow(2, i)) - 1 + (j-1) >= A.length) break;
            imprimeEspacos(espacos);
            System.out.printf("%3d", A[(int) (Math.pow(2, i)) - 1 + (j-1)]);
            imprimeEspacos(espacos);
            if (j < Math.pow(2, i)) {
                System.out.printf("%3s", "");
            }
        }
        espacos = (espacos - 3) / 2;
        System.out.println();
    }
}

static int calculaEspacos (int h) {
    int espacos = 3;
    for (int i = 1; i <= h; i++) {
        espacos = 2 * espacos + 3;
    }
    return espacos;
```

```
    }

    static void imprimeEspacos (int n) {
        for (int i = 1; i <= n; i++) {
            System.out.print(" ");
        }
    }

}
```

Passo 2: Execute o código e escreva a saída do programa aqui. O vetor A é um heap máximo? Justifique sua resposta. (5%)

Saída:

```
Vetor A:
    86    78    36    61    67    34    58    42    35    59    62    28    60    37    12


Heap A:
                                          86
                      78                                        36
          61                      67                  34                      58
    42          35          59          62          28          60          37          12
```

Passo 3: Implemente as funções left e right que retornam, respectivamente, o índice do filho da esquerda e o índice do filho da direita de um nó de índice i. Imprima os elementos A[left(2)] e A[right(4)]. (10%)

Saída:

```
Left A(2):
34
Right A(4):
62
```

Passo 4: Implemente a função maxheapfy. (15%)

```
static void maxheapfy (int[] V, int i) {
  int l = left(i);
  int r = right(i);
  int maior = 0;
  if (l <= V.length && V[l] > V[i]) {
      maior = l;
  } else {
      maior = i;
  }
  if (r <= V.length && V[r] > V[maior]) {
      maior = r;
```

```
  }
  if (maior != i) {
      int trocar = V[i];
      V[i] = V[maior];
      V[maior] = trocar;
      maxheapfy(V, i);
  }
}
```

Passo 5: Antes de imprimir o vetor, aplique o procedimento maxheapfy ao índice 0 do vetor A e escreva a saída do programa aqui. Houve modificação no vetor A? Por que isso aconteceu? (10%)

```
Aplicando MaxHeapFy na raiz do vetor:
                                        86
                    78                                      36
          61                    67                  34                  58
    42          35          59          62          28          60          37          12
```

Não houve mudança no vetor pois a raiz e seu nos filhos ja corespondiam ao requisito de heap máximo ou seja: O maior elemento em um heap máximo e armazenado na raiz, e a subárvore desse nó contém valore menores que o do nó em questão.

Passo 6: Antes de imprimir o vetor, aplique o procedimento maxheapfy ao índice 2 do vetor A e escreva a saída do programa aqui. O vetor A passou a ser um heap máximo? Por que isso aconteceu? (10%)

```
Aplicando MaxHeapFy no indice 2 do vetor:
                                        86
                    78                                      58
          61                    67                  34                  36
    42          35          59          62          28          60          37          12
```

Não, pois as subárvore formado apartir do indice `A[5]` não é um heap máximo.

Passo 7: Antes de imprimir o vetor, aplique o procedimento maxheapfy ao índice 2 do vetor A e, em seguida, ao índice 5. O vetor A passou a ser um heap máximo? Por que isso aconteceu? (10%)

```
Aplicando MaxHeapFy no indice 2 e no indice 5 do vetor:
                                        86
                    78                                      58
          61                    67                  60                  36
    42          35          59          62          28          34          37          12
```

Não, pois a ordem aplicada no aranjo deixo a subárvore de indice `A[2]` em não conformidade com a regra de heap-máximo. Ou seja a subárvore formada apartir do indice `A[2]` tem sua raiz um indice menor que uma de suas folhas.

Passo 8: Antes de imprimir o vetor, aplique o procedimento maxheapfy ao índice 5 do vetor A e, em seguida, ao índice 2. O vetor A passou a ser um heap máximo? Por que isso aconteceu? (10%)

```
Aplicando MaxHeapFy no indice 5 e no indice 2 do vetor:
                                        86
                    78                                      60
          61                    67                  58                  36
    42          35          59          62          28          34          37          12
```

Sim, pois satisfaz a regra de heap máximo, onde toda nó pai é maior que os nós filhos

```
A[Parent(i) >= A[i]
```

Passo 9: Crie um vetor B aleatório, imprima cada um de seus elementos e sua representação em heap (Obs.: código comentado no método main). O vetor B é um heap máximo? (10%)

```
Vetor B:
    79    40     9    81    60   119   113    75    14   113    19    63    78     1    97


                                        79
                    40                                      9
          81                    60                  119                 113
    75          14          113         19          63          78          1           97
```

Não.

Passo 10: Implemente o procedimento buildmaxheap que, dado um vetor qualquer, reorganiza os elementos do vetor para que ele possua a propriedade de maxheap. Apresente seu código aqui. (15%)

```
1. static void buildmaxheap (int[] V) {
2.    for (int i = V.length / 2; i >= 1; i--) {
3.      maxheapfy(V, i - 1);
4.    }
5. }
```

Passo 11: Crie um vetor B aleatório, aplique o procedimento buildmaxheap, imprima cada um de seus elementos e sua representação em heap. O vetor B é um heap máximo? (5%)

```
Aplicando o procedimento de buildMaxHeap no vetor B:
                                        145
                    125                                     133
          92                    119                 129                 85
    89          30          104         75          89          79          84          35
```

Sim o vetor é uma heap máximo, `A[Parent(i) >= A[i]` todo nó pai e maior ou igual ao seus nós filhos.