Resolva cada exercício em folhas separadas

4 p^tos

1. Considere as seguintes classes: classe **Mensagem** que representa uma mensagem de e-mail e classe **Servidor** que possui duas caixas de e-mail para o envio e receção de mensagens. Ambas as caixas do Servidor podem conter várias mensagens de um mesmo remetente.

```
public class Mensagem {
  String emailRemetente;
  Set<String> lst_destinatarios;
  String assunto;
  String texto;

  String getRemetente()
               {return emailRemetente;}
  Set<String> getLista_destinatarios()
               {return lst_destinatarios;}
  ...
}
```

```
public class Servidor {
  List<Mensagem> caixaIn; // INBOX
  List<Mensagem> caixaOut; // OUTBOX

  public List<Mensagem> getCaixaIn ()
                              { return caixaIn;}
  public List<Mensagem> getCaixaOut()
                              { return caixaOut;}

  ...
}
```

Implemente um método na classe Servidor que devolve um map com o número de mensagens enviadas por cada remetente para cada destinatário.

```
public Map<String, Map<String, Integer>> NumberMessagesSent()
```

3 p^tos

2. Considere o seguinte trecho de código:

```
public String[] complexAnal (int a[]){
       String [] r = new String[a.length];

       for (int i=0; i < a.length; i++){
          r[i] = doWork(a[i]);
       }
       return r;
}

public String doWork(int n){
       if (n == 0)
           return "";
       return doWork(n / 2) + n % 2;
}
```

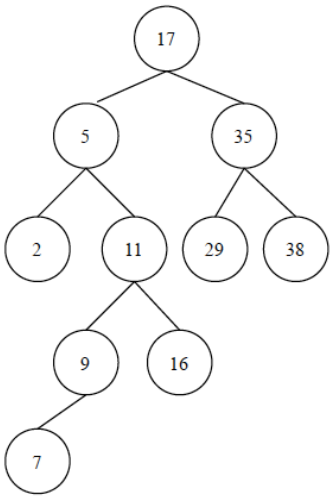
a) Explique o que o trecho de código faz.

b) Analise o trecho de código quanto à sua complexidade temporal utilizando a notação Big-Oh. Justifique adequadamente.

Resolva cada exercício em folhas separadas

5 ptos

3. Adicione à classe TREE<E> um método genérico que, sem recurso a qualquer método de travessia da classe BST (inOrder, preOrder, posOrder), devolva o predecessor de um elemento passado por parâmetro. Por exemplo:

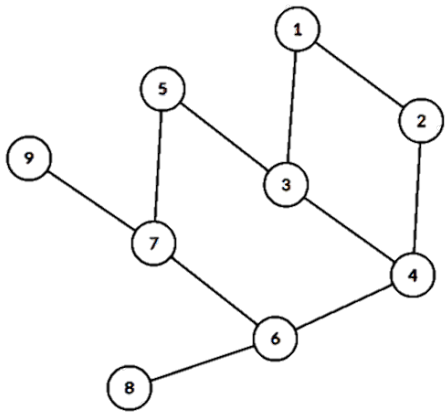


predecessor(17) => 16
predecessor(7) => 5
predecessor(29) => 17
predecessor(2) => null
predecessor(14) => null

```
public E predecessor(E elem)
```

5 ptos

4. Considere as cidades de um país representadas num grafo conexo e não direcionado com N nós e M ramos. O Joaquim e o António estão localizados na cidade correspondente a um dado nó S. O Joaquim quer viajar para o nó A, enquanto o António quer viajar para o nó B, com A != B. Cada um deles quer fazer o caminho mais curto até ao seu destino. Calcule o número máximo de ramos que os dois amigos conseguem percorrer juntos. Por exemplo, dada a cidade inicial S=1 e destinos A=8, B=9, o número máximo de ramos é 1.




```
int longestJourney(Graph<String, Integer> g, String s, String a, String b)
```

3 ptos

5. Implemente o método na classe HeapPriorityQueue<K,V> que verifica se um ArrayList<Entry<K,V>> representa uma heap.

```
public boolean isPriorityQueue(ArrayList<Entry<K,V>> lstEntrys)
```