---

1. Uma palavra é denominada um palíndromo se for invertida e a leitura da mesma permanecer sem nenhuma alteração. Algumas palavras que são palíndromos são: aba, radar, reter, rever, rir, rotor, dentre outras. Implemente um algoritmo recursivo de reconhecimento de palíndromos que deteca se uma palavra (string) digitada pelo usuário é ou não um palíndromo.

   Resposta: Implementada em C++

```
bool ehPalindrome(string word) {

      int tam = word.size();

     // Caso Base #1
     if ( word[0] != word[strlen - 1] )
         return false;

     // Caso Base #2
     else if ( strlen <= 1 )
          return true;
     else
          return ehPalindrome(word.substr(1, strlen - 2));

}
```

2. Considere uma implementação de tabelas hash por meio de *linear probing* [Melhorn & Sanders, p. 90]. Apresente algoritmos (em pseudocódigo) para inserir e remover elementos da tabela hash (utilize sentinelas para representar elementos deletados).

   Resposta: A implementação a seguir é feita dentro de uma classe Java.

```
private static int tableSize = 100;

private int getIndex(Object key)
{
    return Math.abs(key.hashCode() % tableSize);
}

private int findEntry(Object key){

    int index = getIndex(key);
```

```
    for (int i = 0; i < tableSize; i++){ //para saber se foi dado um loop completo
      if (data[index] == null){
        return index;
      }
      if (data[index] != DELETED){
        if (data[index].key.equals(key)){
          return index;
        }
      }
      index = (index + 1) % tableSize;
    }
    return -1; //tabela hash cheia
}


public void put(Object key, Object val){
    int index = findEntry(key);
    Entry entry = data[index];
    if (entry == null){
      data[index] = new Entry(key, val);
    }
    else{ //sobrescreve com novo valor
      entry.val = val;
    }
}

public void remove(Object key){
    int index = findEntry(key);
    if ((index != -1) && (data[index] != null)){
      data[index] = DELETED;
    }
  }
}
```

3. Considere uma tabela hash de tamanho 11. Usando a função hash $h(k) = (2k + 5)\%11$,, mostre o resultado de inserção das chaves 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5. Assuma que as colisões são tratadas por:
   (a) listas encadeadas
   (b) *linear probing*

4. Suponha uma floresta com nós isolados 1, 2, 3, 4 e 5. Determine a floresta obtida após a

realização das seguintes operações: $link(1,3)$, $link(2,5)$, $link(3,4)$ e $link(3,2)$ supondo que a operação fundir seja realizada com união por altura.

5. Considere uma árvore binária, ou seja, cada nó possui um filho esquerdo e um direito. Apresente o pseudocódigo de um algoritmo que encontra o primeiro ancestral comum de dois nós da árvore.

   Resposta:

```
public Tree commonAncestor(Tree root, Tree p, Tree q) {
  if (covers(root.left, p) && covers(root.left, q))
    return commonAncestor(root.left, p, q);
  if (covers(root.right, p) && covers(root.right, q))
    return commonAncestor(root.right, p, q);
  return root;
}
private boolean covers(Tree root, Tree p) {
  if (root == null) return false;
  if (root == p) return true;
  return covers(root.left, p) || covers(root.right, p);
}
```

6. Dado um array de inteiros, escreva um algoritmo que crie uma árvore binária com altura mínima.

   Resposta:

```
public static TreeNode addToTree(int arr[], int start, int end){
  if (end < start) {
    return null;
  }
  int mid = (start + end) / 2;
  TreeNode n = new TreeNode(arr[mid]);
  n.left = addToTree(arr, start, mid - 1);
  n.right = addToTree(arr, mid + 1, end);
  return n;
}

public static TreeNode createMinimalBST(int array[]) {
  return addToTree(array, 0, array.length - 1);
}
```

7. É correto afirmar que o algoritmo de Kruskal encontra a árvore geradora de custo máximo caso modifiquemos a ordenação das arestas da ordem crescente de pesos para a ordem decrescente? Justifique.

Resposta: Sim. A prova é similar àquela apresentada para a árvore geradora de custo mínimo.