Exercícios de exames anteriores

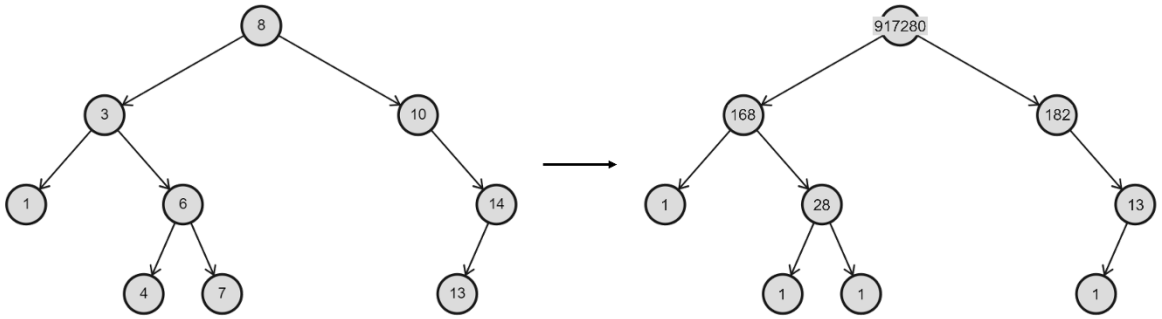
1. Tendo por base a biblioteca de árvores binárias de pesquisa (`bst.hpp`) disponibilizada, implemente as seguintes alíneas (no ficheiro `bst.cpp`).
   a) Implemente o método **`prodReplace()`** que transforma a árvore na correspondente árvore produto.

```
uint prodReplace(Node *tree_node);
```

A função recebe um apontador para um nó da árvore (`tree_node`) e retorna o valor do produto entre o novo conteúdo do nó e o seu antigo valor.

A função é recursiva e deverá percorrer a árvore em pós-ordem. Em cada nó, deverá ser guardado o produto dos valores dos nós das subárvores da esquerda e da direita. O número de nós é mantido e, por conseguinte, o conteúdo de nós que não possuem filhos corresponderá a um.

A figura abaixo representa um exemplo de uma BST (à esquerda) e da sua correspondente árvore produto (à direita).



Quando executado localmente, o ficheiro **`ex1.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
prodReplace():
   Árvore original: 25 20 10 5 1 8 12 15 22 36 30 28 40 38 48 45 50
   Árvore produto: 2306867200 1584000 7200 8 1 1 15 1 1 455446528 28 1 4104000
1 2250 1 1

prodReplace():
   Árvore original: 2 1 4 3 5
   Árvore produto: 60 1 15 1 1
```

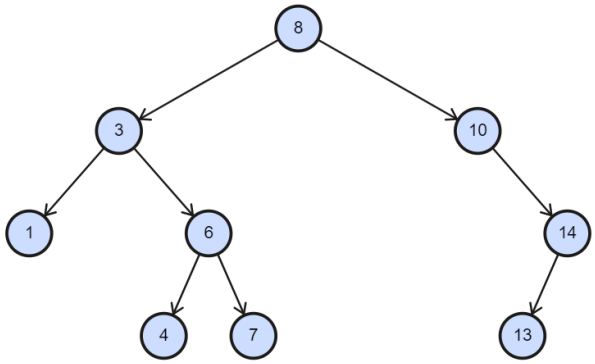
   b) Implemente o método **`printSibling()`** que imprime o irmão de determinado nó de uma BST.

```
void printSibling(Node *search_node, Node *tree_node);
```

A função recebe um apontador para o nó do qual se pretende conhecer o irmão (`search_node`) e um apontador para um nó da árvore (`tree_node`). Deverá verificar se os argumentos são válidos.

A função é recursiva e deverá percorrer a árvore em pré-ordem. Recorra ao método **`getLevel()`** para determinar o nível de um nó na árvore. De notar que o nível da raiz da árvore corresponde a **1**.

Dois nós de uma árvore binária são irmãos caso estejam ao mesmo nível na árvore e tenham o mesmo pai. No exemplo abaixo, o valor do nó irmão do nó de valor **10** é **3**.



Para determinar, por exemplo, o nível de `search_node` deverá escrever:

```
int lev = getLevel(search_node, root, 1);
```

Quando executado localmente, o ficheiro **`ex1.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
printSibling():
   Irmão de ?:

printSibling():
   Irmão de 13:

printSibling():
   Irmão de 8: 4

printSibling():
   Irmão de 12: 5

printSibling():
   Irmão de 50: 45
```

MT2 de 2023

2. Tendo por base a biblioteca STL, implemente as seguintes alíneas associadas à manipulação de filas de prioridade.

a. Implemente a função **`smallestPosElementVector()`** que retorna o k-ésimo menor elemento de um vetor de inteiros.

```
int smallestPosElementVector(vector<int> vec, int k);
```

A função recebe o vetor a analisar (`vec`) e a posição do elemento a encontrar (k-ésimo menor). Deverá verificar se os argumentos são válidos (`k` deve ser um número positivo inferior ou igual ao tamanho do vetor). Em caso de erro, a função retorna o valor **9999**.

Utilize uma fila de prioridade para ordenar os elementos do vetor.

Quando executado localmente, o ficheiro **`ex2.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
smallestPosElementVector(): ERRO

smallestPosElementVector(): OK
   Vetor: 4 5 6 6 10 9 7 2 6 1
   Elemento (k = 4): 5

smallestPosElementVector(): OK
   Vetor: -35 95 63 58 63 3
   Elemento (k = 2): 3

smallestPosElementVector(): OK
   Vetor: 87 -58 -21
   Elemento (k = 1): -58
```

b. Implemente a função **`largestPosElementString()`** que retorna o k-ésimo maior elemento (palavra) de uma *string* (frase).

```
string largestPosElementString(string text, int k);
```

A função recebe a *string* a analisar (`text`) e a posição do elemento a encontrar (`k`). Deverá verificar se os argumentos são válidos (`k` deve ser um número positivo inferior ou igual ao número de palavras). Em caso de erro, a função retorna uma *string* vazia.

Poderá recorrer ao uso de uma *stringstream* para separar as palavras na frase e utilize uma fila de prioridade para ordenar essas mesmas palavras.

A função de comparação para ordenação dos elementos na fila de prioridade, **`compareMax`**, já se encontra implementada.

Quando executado localmente, o ficheiro **`ex2.cpp`** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
largestPosElementString(): ERRO
```

```
largestPosElementString(): OK
   String original: when there is smoke there is fire
   String: (k = 4): fire

largestPosElementString(): OK
   String original: better safe than sorry
   String: (k = 3): safe

largestPosElementString(): OK
   String original: blood is thicker than water
   String: (k = 1): thicker
```

MT2 de 2023

3. Implemente a função **`maxCollisions()`** que, dado uma tabela de dispersão e um vetor de strings, identifica a string, que teve o maior numero de colisões quando foi inserido. A função deverá retornar uma string, se o vetor estiver vazio ou nenhum dos elementos estiver na tabela deve devolver uma string vazia. A tabela de dispersão é do tipo endereçamento aberto, ou seja, existe a função hash(***`hashFunction`***) e uma função de sondagem(***`probingFunction`***) .

Nota: Não é para inserir nenhuma string na tabela, é só para ir verificar como foi feita a inserção.

```
string maxCollisions(HashTable &ht, vector<string> &v);
```

Quando executado localmente, o ficheiro **ex3.cpp** deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
resultado: Bernardo
```

MT2 de 2023

4. Tenha em consideração a estrutura de grafo implementada.

a. Complete a função **`BFS(int s)`** que recebe um inteiro **s**. A função deverá implementar o algoritmo **`Breadth First Transversal`** e devolver uma **`queue`** com os nós ordenados, de acordo com a aplicação do algoritmo BST.

```
queue BFS(int s)
```

BFS é um algoritmo de pesquisa em estruturas de dados. O algoritmo faz uma pesquisa por níveis de profundidade, ou seja o algoritmo começa no nó dado (int s) do grafo, passa por todos os nós da mesma profundidade e só depois visita os nós do próximo nível de profundidade
Não se esqueça que os grafos podem ter ciclos pelo que deve ter o cuidado de saber que nós já foram visitados.

Sugestão: utilize a biblioteca STL **queue** para auxílio.

O resultado para o primeiro teste deverá ser:

```
Breadth First Tranversal from vertix 1
1 0 3 2 4
```

b. Complete a função **`isReachable(`**`int s, int d`**`)`** que recebe dois inteiros, o identificador (s) do nó inicial e o identificador (d) do nó final. A função deverá verificar se existe um caminho do nó **s** para o nó **d**. A função deverá devolver **true** se o caminho existir ou **false** se o caminho não existir.

`bool isReachable(int s,int d)`

Sugestão: Utilize a função BFS, previamente implementada, para percorrer o grafo a partir do nó **s**.
SPS: se não resolveu a a) assuma que a função BFS está disponível nesta alínea.

O resultado para o primeiro teste deverá ser:

```
A path exists from 1 to 3
No path exists from 3 to 1
```

MT2 de 2023

5 . Tendo por base o que aprendeu sobre árvores binárias, implemente as seguintes alíneas.

NOTA: Os nós das árvores binárias utilizadas nos exercícios correspondem à estrutura TreeNode, cuja representação se encontra a seguir.

```
struct TreeNode {
  int val;
  TreeNode *left;
  TreeNode *right;
  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```

val é um inteiro que representa o valor do nó a guardar na árvore, e left e right correspondem aos apontadores para os filhos da esquerda e direita, respetivamente.

a) Implemente a função void inorderTraversal(TreeNode *root) que deverá percorrer a árvore em ordem, imprimindo o valor dos seus nós.
b) Implemente a função int maxDepth(TreeNode *root) que deverá calcular a profundidade máxima da árvore, retornando esse valor. root corresponde ao apontador para a raiz da árvore e deverá verificar se o argumento é válido.
c) Implemente a função `TreeNode*` **`lowestCommonAncestor`**`(TreeNode *root, TreeNode *node1, TreeNode *node2)` que deverá encontrar (e retornar) o nó

antecessor mais baixo comum aos nós apontados por `node1` e `node2`. Em caso de erro, retorna `NULL`. Deve verificar se os argumentos são válidos.

`root` corresponde ao apontador para a raiz da árvore.

MT2 de 2024

6. Tendo por base o que aprendeu sobre grafos não dirigidos, implemente as seguintes alíneas.

NOTA: Os vértices dos grafos utilizados nos exercícios correspondem a objetos da classe Graph, cujos atributos correspondem a v (número de vértices do grafo) e edges (lista de adjacências).

```
class Graph {
 private:
  int v;
  vector<list<int>> edges;
};
```

root corresponde ao apontador para a raiz da árvore e deverá verificar se o argumento é válido.

a) Implemente a função int addEdge(int v1, int v2) que adiciona uma aresta ao grafo (não direcionado) entre os nós v1 e v2.

A função retorna 0 se a aresta for inserida com sucesso, 1 caso a aresta já exista ou -1 em caso de erro. Deve verificar se os argumentos são válidos, ou seja, dentro dos intervalos esperados.

b) Implemente a função int countNeighbours(int node) que determina (e retorna) o número de vizinhos de determinado vértice (node).

A função retorna -1 em caso de erro e deve verificar se o argumento é válido, ou seja, dentro do intervalo esperado.

MT2 de 2024

7. Considere a representação de àrvores binárias BST.

Implemente uma função booleana **isBST()** que recebe uma árvore binária com a aestrutura de uma BST e verifica se é uma árvore binária de pesquisa (devolve true) ou não (devolve false).

bool isBST(Node * root)

Quando executado localmente, o ficheiro **ex.cpp** deve apresentar o seguinte resultado ao testar

a implementação da função mencionada.

```
++ Exercício 5a) ++
Tree 1 false
Tree 2 true
Tree 3 true
```

8. Tendo por base a biblioteca STL, implemente as seguintes alíneas associadas à manipulação de filas de prioridade.

Implemente a função firstToLast() que passa o elemento com mais prioridade para último(menor prioridade) e retorna 0.

int firstToLast(priority_queue<int> &pq);

A função recebe um max-heap (pq) e deverá verificar se o argumento é válido (pq não deve ser vazio). O elemento deve ficar com o valor do elemento que estava em último menos 1. Em caso de erro, a função retorna o valor -1.

Quando executado localmente, o ficheiro ex9.cpp deve apresentar o seguinte resultado ao testar a implementação da função mencionada.

```
Before Queue: 44 34 24 14 4
Return: 0
After Queue: 34 24 14 4 3
Before Queue: 4
Return: 0
After Queue: 4
```