1- **Explique o que vem a ser uma Árvore em Estrutura de Dados? Em sua resposta, reflita sobre as terminologias: root (raiz), nó-dado, valor-chave, aresta, níveis hierárquicos, regra de organização de dados, nó-pai, nó-filho, nó-folha, subárvore, caminho, visitar nó, tipos de percurso.**

A árvore é uma estrutura de dados não linear e hierárquica, ou seja, não há um caminho único para percorrê-la. Conceptualmente diferente das listas, em que os dados se encontram numa sequência, nas árvores os dados estão dispostos de forma hierárquica, seus elementos se encontram "acima" ou "abaixo" de outros elementos da árvore.

Uma árvore é formada por um conjunto de elementos que armazenam informações chamados **nodos** ou **nós-dado**, todo nó de uma árvore binária contém, ao menos, um **valor-chave** (que poderá ser um int, char, char *, etc) e dois outros ponteiros: um para a subárvore esquerda e outro para a subárvore direita, esses ponteiros são chamados de **arestas**.

O primeiro nó de uma árvore é chamado de raiz (root), que possui ligações para outros elementos denominados **nós-filhos**. Eventualmente, quando novos nós saem dos nós-filhos, eles se tornam **nós-pai**. O elemento que não possui ramos,ou seja, os ponteiros apontam para NULL, é conhecido como **nó-folha**.

O **nível hierárquico** de um nó é a distância dele até a raiz, e a **altura** de uma árvore é a distância do nó-folha com o caminho mais longo até a raiz. **Subárvore** é um conjunto isolado de nós da árvore para realizar algum tipo de operação.

As árvores, como dito anteriormente, são estruturas que podem ser percorridas de diferentes formas, cada percurso pode produzir um resultado específico, sendo assim, algumas formas de percorrer uma árvore são:

Pré-Ordem

Esse algoritmo segue a seguinte lógica:

1. Visita o primeiro nó, que é a raiz;
2. Vai buscar o valor na subárvore esquerda da raiz;
3. Caso não encontre o valor na subárvore esquerda, percorre a subárvore direita.

Em Ordem

O algoritmo em ordem segue a seguinte lógica:

1. Percorre a subárvore à esquerda;
2. Visita o nó raiz;
3. Percorre a subárvore à direita

Pós-Ordem

O algoritmo pós-ordem segue a seguinte lógica:

- Percorre a subárvore à esquerda;
- Percorre a subárvore à direita;
- Visita o nó raiz.

Em Nível:O percurso em nível de uma árvore binária consiste em realizar operações em todos os nós, um nível por vez. Para que isso seja possível é necessária a utilização de outra estrutura de dados, sendo ela a **fila. (Raiz, Filhos, Netos, etc…).**

2.Como funciona a inserção de um novo nó-dado na árvore binária (regra básica)? Obs.: Cuidado, existem técnicas que reorganizam a árvore já na inserção de modo a mantê-la balanceada, p. ex., as árvores binárias alinhadas, que não são alvo desta questão.

Caso não haja nenhum nó-dado na árvore e este será o primeiro a ser inserido, então ele assumirá o papel de raiz da árvore. Para todo novo nó-dado a ser inserido, entrará na árvore como um nó-folha. Se o elemento for menor do que o nó-pai será inserido no ramo da esquerda. Caso seja maior ou igual, será inserido no ramo da direita.

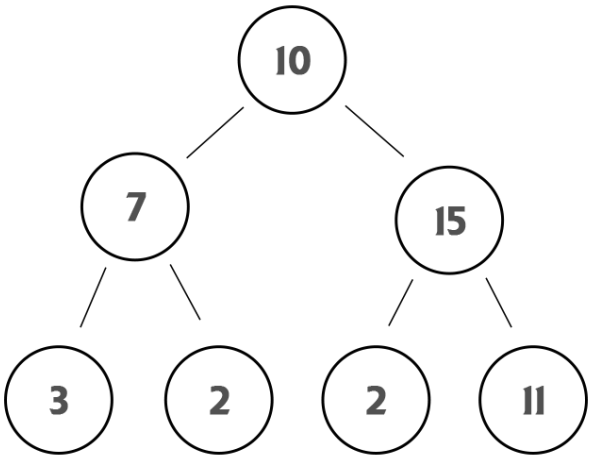
3- Desenvolva uma síntese explicativa sobre os métodos de travessia em profundidade aplicadas em árvores binárias: inorder (em ordem), preorder (pré-ordem), postorder (pós-ordem) e inlevel (em nível). Deve-se explicar estratégias recursivas em um exemplo de árvore.

InOrder: essa travessia primeiro percorre a subárvore da esquerda do nó raiz, acessa o nó atual, seguido da subárvore da direita do nó atual.

```
void EMORDEM (tipo_nodo *PR) {
        if (PR != NULL) {  //Verifica se a arvore não esta vazia
                // Percorre primeiro a subárvore da esquerda.
```

```
                EMORDEM (PR->pont_esq );
                escreva (PR->info );
                EMORDEM (PR->pont_dir );
                };
}
```

Exemplo:



Travessia:

1. Root
   1.1 Recursão Filho a Esquerda

2. No: 7
   2.1 Recursão Filho a Esquerda

3. No: 3
   3.1 Recursão Filho a Esquerda ok
   3.2 Printa na tela o valor 3. ok
   3.3 Recursão Filho a Direita ok

Como uma parte do lado esquerdo foi finalizado, o processo volta completando as outras pendências da recursividade.
Voltando para o Passo (2)

2. No : 7.
    2.1 Recursão Filho a Esquerda ok
    2.2 Printa na tela o valor 7 ok
    2.3 Recursão Filho a Direita

4. No : 2.
    4.1 Recursão Filho a Esquerda ok
    4.2 Printa na tela o valor 2. ok
    4.3 Recursão Filho a Direita ok

Com isso, o lado esquerdo já foi atravessado, falta voltar com a recursividade até a raiz (Passo 1) e seguir para o lado direito.

1. Root
    1.1 Recursão Filho a Esquerda ok
    1.2 Printa na tela o valor 10. ok
    1.2 Recursão Filho a Direita ok

5. No :15
    5.1 Recursão Filho a Esquerda

6. No: 2
    6.1 Recursão Filho a Esquerda ok
    6.2 Printa na tela o valor 2. ok
    6.3 Recursão Filho a Direita ok

Agora, o processo volta terminando de executar a recursividade do Passo 5:

5. No: 15
    5.1 Recursão Filho a Esquerda ok
    5.2 Printa na tela o valor 15. ok
    5.3 Recursão Filho a Direita
7. No: 11
    7.1 Recursão Filho a Esquerda ok
    7.2 Printa na tela o valor 11. ok

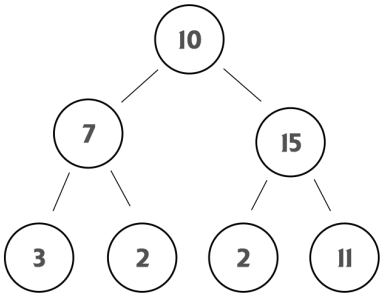
7.3 Recursão Filho a Direita ok

Por fim, travessia concluída, o resultado printado na tela será:
3 - 7 - 2 - 10 - 2 - 15 - 11

Pre-Order: Essa travessia primeiro acessa o valor do nó atual, faz a travessia pela subárvore da esquerda e, depois, pela subárvore da direita, respectivamente.

```
void PREORDEM (tipo_nodo *PR) {
        if (PR != NULL) {
                escreva (PR->info );
                PREORDEM (PR->pont_esq );
                PREORDEM (PR->pont_dir );
        };
}
```

Travessia



1. Root
   1.1 Escreva (10) ok
   1.2 Recursão Filho a Esquerda

2. Nó: 7
   2.1 Escreva (7) ok
   2.1 Recursão Filho a Esquerda

3. Nó: 3
    3.1 Escreva (3) ok
    3.2 Recursão Filho a Esquerda ok
    3.2 Recursão Filho a Direita ok

Voltando para o Passo 2

2. Nó: 7
    2.1 Escreva (7) ok
    2.2 Recursão Filho a Esquerda ok
    2.3 Recursão Filho a Direita
4. Nó: 2
    4.1 Escreva (2) OK
    4.2 Recursão Filho a Esquerda ok
    4.3 Recursão Filho a Direita ok

Lado esquerdo finalizado, voltando para a raiz (Passo 1)

1. Root
    1.1 Escreva (10) ok
    1.2 Recursão Filho a Esquerda ok
    1.3 Recursão Filho a Direita

5. Nó: 15
    5.1 Escreva (15) ok
    5.2 Recursão Filho a Esquerda

6. Nó: 2
    6.1 Escreva (2) ok
    6.2 Recursão Filho a Esquerda ok
    6.3 Recursão Filho a Direita ok

Voltando para o Passo 5

5. Nó: 15
    5.1 Escreva (15) ok
    5.2 Recursão Filho a Esquerda ok

5.3 Recursão Filho a Direita ok

7. Nó : 11

7.1 Escreva (11) OK

7.2 Recursão Filho a Esquerda ok

7.3 Recursão Filho a Direita ok

Resultado em tela:
10 7 3 2 15 2 11

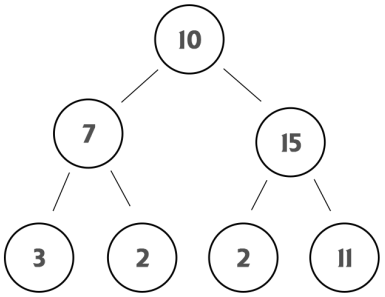
Post-Order: Essa travessia coloca o valor do nó raiz no fim, percorrendo primeiro a subárvore da esquerda e depois a da direita. A ordem relativa das subárvores da esquerda e da direita permanece a mesma. Somente a posição do nó raiz muda em relação às travessias anteriores.

```
void postOrder(struct node* root) {
	if (root == null) {
		return;
	}
	// Percorre primeiro a subárvore da esquerda.
	postOrder(root.left);

	// Percorre em seguida a subárvore da direita.
	postOrder(root.right);

	    // Imprime o valor do nó atual.
	printf("%d ", root.data);

}
```



1. Root
    1.1 Recursão Filho a Esquerda
2. Nó: 7
    2.1 Recursão Filho a Esquerda
3. Nó: 3
    3.1 Recursão Filho a Esquerda Ok
    3.2 Recursão Filho a Direita Ok
    3.3 Escreva (3) Ok

Voltando para a recursividade do Passo 2

2. Nó: 7
    2.1 Recursão Filho a Esquerda Ok
    2.2 Recursão Filho a Direita
4. Nó: 2
    4.1 Recursão Filho a Esquerda Ok
    4.2 Recursão Filho a Direita Ok
    4.3 Escreva (2) Ok

Voltando para a recursividade do Passo 2

2. Nó: 7
    2.1 Recursão Filho a Esquerda Ok
    2.2 Recursão Filho a Direita Ok
    2.3 Escreva (7) Ok

Lado Esquerdo processado, voltando para o Root para continuar a recursividade

1. Root
    1.1 Recursão Filho a Esquerda Ok
    1.2 Recursão Filho a Direita

5. Nó: 15

    5.1 Recursão Filho a Esquerda

6. Nó: 2

    6.1 Recursão Filho a Esquerda Ok

    6.2 Recursão Filho a Direita Ok

    6.3 Escreva (2)

Voltando para o Passo 5 para continuar a recursividade:

5. Nó: 15

    5.1 Recursão Filho a Esquerda Ok

    5.2 Recursão Filho a Direita

7. Nó: 11

    7.1 Recursão Filho a Esquerda Ok

    7.2 Recursão Filho a Direita Ok

    7.3 Escreva (11)

Voltando para o Passo 5 para finalizar a recursividade:

5. Nó: 15

    5.1 Recursão Filho a Esquerda Ok

    5.2 Recursão Filho a Direita Ok

    5.3 Escreva (15) Ok

Por fim, terminar a recursividade do nó Root:

1. Root

    1.1 Recursão Filho a Esquerda Ok

    1.2 Recursão Filho a Direita Ok

    1.3 Escreva (10) Ok

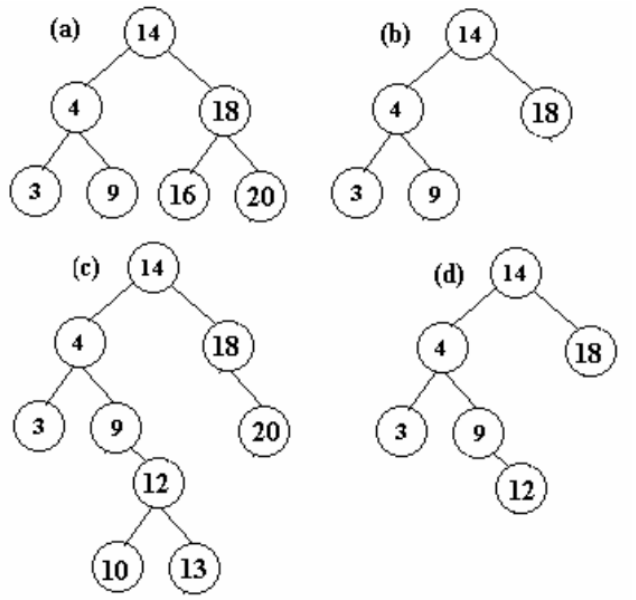
Resultado da Travessia:

3 - 2 - 7 - 2 - 11 - 15 -10

Em Nível:O percurso em nível de uma árvore binária consiste em realizar operações em todos os nós, um nível por vez. Para que isso seja possível é necessária a utilização de outra estrutura de dados, sendo ela a **fila. (Raiz, Filhos, Netos, etc…).**

4- Compare a árvore desbalanceada x balanceada. Logo, quais são as vantagens/desvantagens de cada caso?

Uma árvore é considerada balanceada se e somente se para qualquer nó, a altura de suas duas subárvores diferem de no máximo uma unidade. Caso essa diferença seja maior que um, a árvore será considerada desbalanceada.
Usando como exemplo a imagem a seguir:



Analisando as quatro árvores acima, podemos notar que, para a raiz, a diferença da altura das subárvores direita e esquerda é igual a 0 na árvore (a),podendo ser chamada de perfeitamente balanceada. Já na (b) a diferença é 1. Mesmo com essa diferença, ambas são consideradas árvores balanceadas.
Já na árvore (c) , para a raiz, a altura da subárvore esquerda é 4 e a altura da subárvore direita é 2. Portanto, a diferença entre elas é 2 e sendo assim ela é uma

árvore desbalanceada. O mesmo ocorre com a árvore (d), a subárvore esquerda tem altura 3, enquanto a direita tem 1 de altura, resultando numa diferença de 2.

Árvore Balanceada:

Vantagens: as operações de inserção, remoção e pesquisa são sempre de ordem O(log N). O balanceamento da altura adiciona apenas um fator constante à velocidade de inserção.

Desvantagens: difícil de programar e depurar, mais espaço para o fator de balanceamento.

Já na árvore desbalanceada, a principal desvantagem é que possui uma complexidade O(n) para realizar as operações de inserção, remoção e pesquisa, sendo maior do que O(log N) da balanceada.

5. Em linhas gerais, remover (1) um nó-dado tipo folha ou (2) que deriva apenas 1 filho é fácil. Já remover (3) um nó-pai de 2 filhos que podem possuir subárvore pode ser uma tarefa mais complicada, exigindo técnicas como "remoção por fusão" ou "remoção por cópia". Logo, explique como funcionam as soluções para os 3 casos citados.

No caso 1, para remover um nó-dado tipo folha basta realizar um free e o ponteiro do pai recebe NULL para o nó-dado a ser removido.

No caso 2, para remover um nó-dado que deriva apenas um filho basta fazer um by-pass, ou seja, fazer o ponteiro do pai receber o ponteiro do filho que está apontando para um “nó-neto”. Com isso, o nó-pai estará apontando para o “nó-neto”, podendo fazer a remoção do nó-filho com um free. Ao final, o nó-neto sobe de posição, se tornando nó-filho.

No caso 3, para remover um nó-pai de dois filhos, pode-se operar de duas maneiras diferentes. Pode-se substituir o valor do nó a ser retirado pelo valor sucessor (o nó mais à esquerda da subárvore direita) ou pelo valor antecessor (o nó mais à direita da subárvore esquerda), removendo-se aí o nó sucessor (ou antecessor).