1.A estrutura

É uma estrutura de dados não linear onde cada nó pode ter até ***dois nós filhos***, referidos como **nó esquerda** e **nó direita**. O primeiro nó da árvore é denominado de **nó raiz** e os nós mais abaixo são denominados de **folhas**. Uma estrutura de árvore binária pode ser visualizada como uma estrutura hierárquica onde o nó raiz é o topo da árvore e as folhas são o fundo.

Cada nó da árvore possui os seguintes componentes:

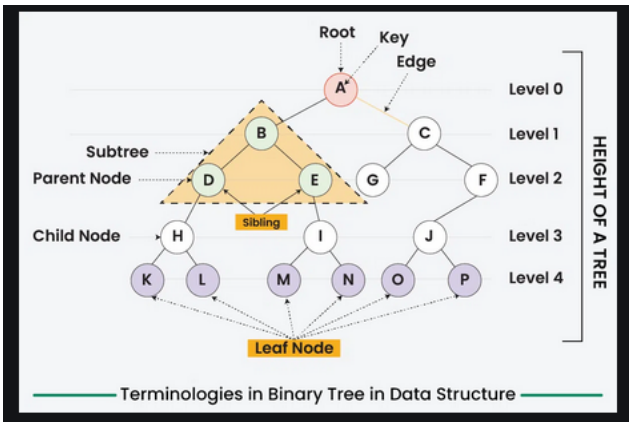
- Data (elemento)
- Ponteiro para o nó direita
- Ponteiro para o nó esquerda

Exemplo de estrutura de árvore binária:

```
typedef struct Arvore
{
   int data;
   struct Arvore *esquerda_ptr;
   struct Arvore *direita_ptr;
}arvore;
```

Terminologias utilizadas em árvores binárias:

- **Nós:** A parte mais fundamental de uma árvore binária, onde cada nó contém *data* e ponteiros para a esquerda e direita
- **Raíz:** O nó mais acima da árvore. Não possui pais e serve como o ponto de início de uma árvore
- **Nó pai:** Um nó que possui um ou mais nós filhos (máx: 2)
- **Nó filho:** Um nó descendente de um nó pai
- **Nó folha:** Nó que não possui nenhum nó filho ou ambos são nulos
- **Nó interno:** Um nó que possui pelo menos um nó filho. Isso inclui todos os nós, exceto o nó raiz e os nós folha
- **Profundidade de um nó:** O número de bordas de um nó específico até o nó raiz. A profundidade do nó raiz é 0
- **Altura de uma árvore binária:** O número de nós do nó folha mais profundo até o nó raiz



Importantes detalhes de árvores binárias:

- O número máximo de nós em nível *N* vale 2^n
- O número máximo de nós em uma árvore binária de altura H vale $2^H - 1$
- O número total de nós folha em uma árvore binária vale o total de nós com 2 filhos + 1
- Em uma árvore binária de *N* nós, a altura mínima ou o número mínimo de níveis vale $Log_2(N + 1)$

As árvores binárias podem ser classificadas em diferentes fatores baseados em:

Número de filhos:

- Full binary tree
- Degenerate binary tree
- Skewed binary tree

Nível:

- Complete binary tree
- Perfect binary tree
- Balanced binary tree

Valor do nó:

- Binary search tree
- AVL tree
- Red Black tree
- B tree
- B+ tree

- Segment tree

1.1 Travessia em árvores binárias

A busca em árvore binária envolve visitar todos os nós da árvore. Estes algoritmos podem ser classificados em duas categorias:

- **Depth-First Search (busca em profundidade):** DFS explora um ramo/galho até seu máximo antes de retroceder. Isto é implementado utilizando recursão. Os principais métodos para percorrer são **pré-ordem**, visita-se o nó primeiro e em seguida o da esquerda e após o da direita, **em-ordem**, visita-se o nó da esquerda primeiro e em seguida o nó principal e por fim o nó da direita, e, **pós-ordem**, visitando-se primeiro o nó da esquerda, em seguida o nó da direita e então o próprio nó.
- **Breadth-First Search (busca em largura):** BFS explora todos os nós no atual nível de profundidade antes de mover para nós no próximo nível. Geralmente é implementado com filas. Também é referido como busca em ordem de nível.

1.2 Inserção em árvores binárias

Não existe ordem de inserção de elementos em uma árvore binária, logo não é necessário se preocupar com esta ordem. Primeiramente cria-se o nó raiz no caso de uma árvore vazia. Então, inserções subsequentes requer a busca de nós vazios em um mesmo nível. Quando um nó, seja esquerda ou direita, estiver vazio, ali o novo elemento é inserido. Estas inserções são sempre primeiro no nó esquerdo.

1.3 Busca em árvores binárias

A busca por um valor em uma árvore binária significa procurar pela árvore para encontrar um nó com determinado valor. Como árvores binárias são diferentes de árvores com busca binária, tipicamente é utilizado um método de travessia. O mais comum é DFS e BFS. O processo continua até encontrar o valor desejado ou chegar no fim da árvore.

1.4 Remoção de uma árvore binária

Deletar um elemento de uma árvore binária significa remover um nó específico da árvore, mantendo a estrutura da árvore. Primeiro, é preciso encontrar o elemento a ser removido utilizando um algoritmo de travessia, então, substituir o valor do nó com o valor do último nó da árvore (geralmente o nó direito) e por fim, deletar o nó desejado.

Operação	Complexidade de tempo	Espaço auxiliar
Travessia em ordem	O(n)	O(n)
Travessia pré-ordem	O(n)	O(n)
Travessia pós-ordem	O(n)	O(n)

Inserção (não balanceada)	O(n)	O(n)
Busca (não balanceada)	O(n)	O(n)
Remoção (não balanceada)	O(n)	O(n)

É possível utilizar a **travessia de Morris** para percorrer todos os nós da árvore binária em O(n), porém, com espaço auxiliar de O(1)

2.Aplicações

- Árvores binárias podem ser usadas para representar dados hierárquicos
- Aŕvores de Huffman podem ser utilizadas em algoritmos de compressão de dados
- Outra aplicação é **fila de prioridade**, utilizada para busca máxima ou mínima em O(1)
- Útil para indexação segmentada em banco de dados no armazenamento cache no sistema
- Podem ser utilizadas para implementar algoritmos de decisão, um algoritmo de *machine learning* de análise de classificação e regressão

3.Vantagens

- **Busca eficiente:** Árvores de busca binária são eficientes em buscar por elementos específicos. Como cada nó tem pelo menos 2 nós filhos comparado com listas encadeadas e *arrays*
- **Memória eficiente:** Árvores binárias requerem menos memória comparado com outras estrutura de dados
- Árvores binárias são relativamente fáceis de se implementar e entender como cada nó tem dois nós filhos

4.Desvantagens

- **Estrutura limitada:** Estruturas de árvores binárias são limitadas a dois nós filhos por nó pai, o que pode limitar a sua usabilidade em certas aplicações
- **Estrutura não balanceada:** Árvores não balanceadas são árvores onde sub-árvores são maiores do que outras sub-árvores, o que pode levar a operações ineficientes
- **Pouca eficiência em espaço:** Árvores binárias podem ser ineficientes em espaço quando comparados com listas encadeadas e *arrays*

- **Performance devagar em piores casos:** No pior dos casos, uma árvore binária se torna degenerada ou *skewed*, o que quer dizer que cada nó possui apenas um nó filho.