**Estruturas Lineares**

* **Listas Encadeadas:**
  + Uma lista encadeada é uma estrutura de dados onde cada elemento (nó) contém um valor e um ponteiro para o próximo nó da lista.
  + **Lista Simplesmente Encadeada:** Cada nó possui apenas um ponteiro, que aponta para o próximo nó.
  + **Lista Duplamente Encadeada:** Cada nó possui dois ponteiros, um para o próximo nó e outro para o anterior, permitindo uma navegação bidirecional.
* **Fila:**
  + Uma fila é uma estrutura de dados do tipo FIFO (First In, First Out), onde os elementos são inseridos no final (enqueue) e removidos do início (dequeue).
  + Uma fila pode ser implementada usando arrays ou listas encadeadas.

**Árvores e Árvores Binárias de Pesquisa (BST)**

* **Árvore Binária de Pesquisa (BST):**
  + Uma árvore onde cada nó possui no máximo dois filhos. O filho à esquerda tem um valor menor que o do nó pai, e o filho à direita tem um valor maior.
  + As operações básicas como inserção, busca e remoção podem ser feitas de forma eficiente em O(log n) se a árvore estiver balanceada.
* **Árvores AVL:**
  + Uma árvore binária de pesquisa que é balanceada de acordo com a altura. Em uma árvore AVL, a diferença de altura entre as subárvores de qualquer nó não pode ser maior que 1.
  + O balanceamento é mantido através de rotações após operações de inserção e remoção, garantindo que a altura da árvore seja logarítmica, mantendo a eficiência das operações.

**Fila de Prioridade e Heap**

* **Fila de Prioridade:**
  + É uma estrutura de dados onde cada elemento tem uma prioridade associada, e os elementos com maior prioridade são removidos antes dos de menor prioridade, independentemente da ordem de inserção.
  + Pode ser implementada de maneira eficiente usando um **heap**.
* **Heap:**
  + Um heap é uma árvore binária completa onde o valor de cada nó é maior (max heap) ou menor (min heap) que os valores de seus filhos.
  + Em um **max heap**, o maior elemento está na raiz, enquanto em um **min heap**, o menor elemento está na raiz.
  + A estrutura de heap é usada em algoritmos como o de ordenação por heap (heapsort) e na implementação de filas de prioridade, onde a operação de inserção e remoção ocorre em tempo O(log n).

**Hashing**

* **Função de Hash:**
  + Uma função de hash transforma dados de entrada de tamanho arbitrário (como uma chave) em um índice fixo que é usado para armazenar o valor em uma tabela hash.
  + A função de hash visa distribuir uniformemente os dados para evitar colisões (quando duas chaves diferentes resultam no mesmo índice).
* **Colisão e Encadeamento:**
  + Colisões ocorrem quando diferentes entradas resultam no mesmo índice na tabela hash. O **encadeamento (chaining)** resolve colisões armazenando os elementos que colidem em uma lista ligada dentro da tabela hash.
  + Outras técnicas de resolução de colisão incluem **sondagem linear** e **sondagem quadrática**.

**Grafos**

* **Grafo:**
  + Um grafo é uma estrutura composta por um conjunto de vértices (ou nós) e arestas (ou conexões) entre eles.
  + **Grafos dirigidos** possuem arestas com direção, enquanto **grafos não dirigidos** possuem arestas bidirecionais.
* **Busca em Largura (BFS) e Busca em Profundidade (DFS):**
  + **BFS (Busca em Largura):** Explora os nós em camadas, visitando todos os vizinhos de um nó antes de avançar para os próximos níveis. BFS é ideal para encontrar o caminho mais curto em grafos não ponderados.
  + **DFS (Busca em Profundidade):** Explora um caminho até o fim antes de retroceder. DFS é útil em problemas de conectividade, como encontrar componentes conexos em um grafo.

**Grafos Eulerianos**

* **Grafo Euleriano:**
  + Um grafo é **Euleriano** se existe um circuito (um caminho fechado) que percorre cada aresta exatamente uma vez.
  + **Condições para um grafo ser Euleriano:** Em um grafo não dirigido, todos os vértices devem ter grau par, e o grafo deve ser conexo. Em grafos dirigidos, o grau de entrada e saída de cada vértice deve ser igual.