Aluno: Edwino Alberto Lopes Stein

Matricula: 1201324411;

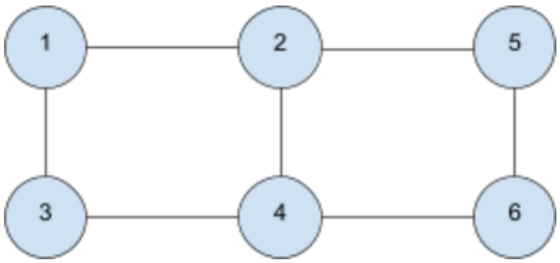
Disciplina: Analise de Algoritmos;

Professor: Dr. Herbert Oliveira Rocha;

[QUESTÃO – 03] Defina e dê exemplos:

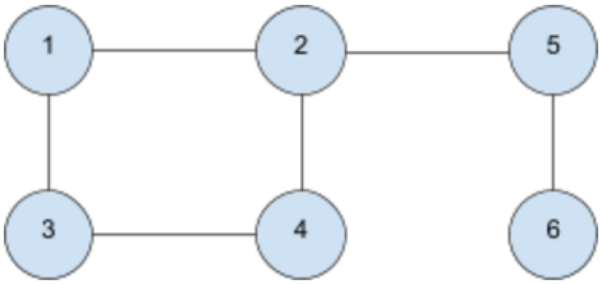
a) Grafos**:**

Um grafo G, é formado por dois conjuntos de elementos, onde o primeiro conjunto V são os vértices e o segundo conjunto E são as arestas, sendo definido como G = (V,E). Cada aresta liga dois vértices, e cada vértice pode está ligado a nenhum ou a vários outros vértices.

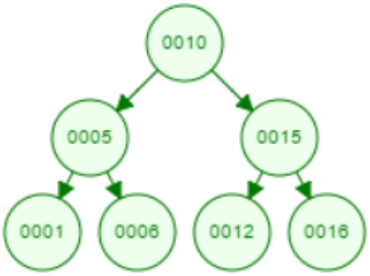


b) Grafo conexo, acíclico e direcionado:

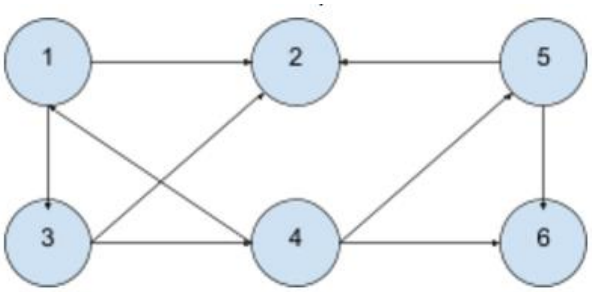
- **Grafo conexo** é quando existe pelo menos um caminho entre qualquer vértice para todos os outros vértices.



- **Grafo acíclico** é o tipo de grafo em que os caminhos entre os vértices não contenha vértices repetidos.



- **Grafo direcionado** é um tipo de grafo onde as arestas possuem sentido, ou seja, não é permitido percorrer pelas arestas no sentido contrário.

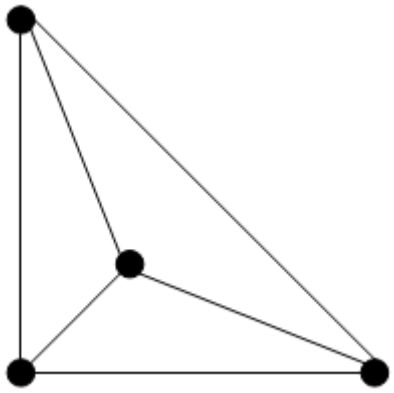


c) Adjacência x Vizinhança em grafos:

É o subgrafo formado por todos os vértices ligados diretamente a um vértice do grafo. Um exemplo para vizinhança em grafos é um mapa rodoviário, onde os vértices são cidade e as arestas são as estradas, considerando uma cidade A, as cidades vizinhas serão as cidades onde exista uma estrada ligada diretamente a A.

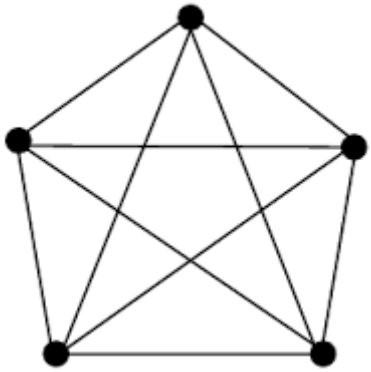
d) Grafo planar

É um tipo de grafo onde pode ser imerso no plano de tal forma que suas arestas não se cruzem.

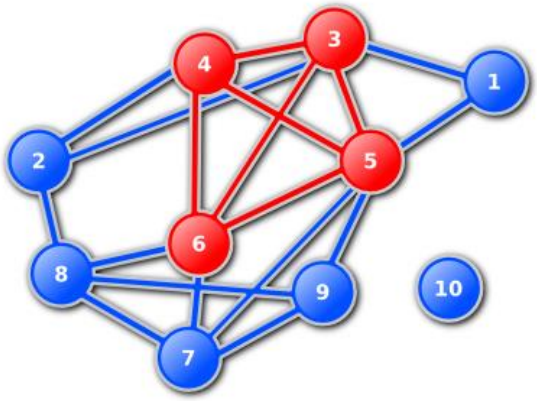


e) Grafo completo, clique e grafo bipartido

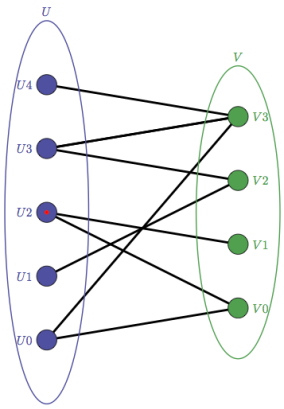
- **Grafo completo** é um tipo de grafo simples em que todos os vértices são adjacentes a todos os outros vértices.



- **Clique** é um subgrafo completo;

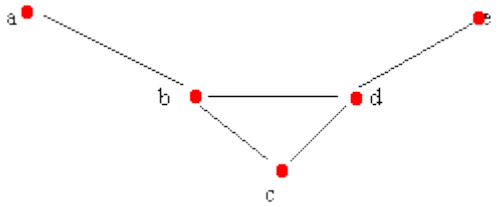


- **Grafo bipartido** é um tipo de grafo onde existem dois subgrafos onde os vértices só se ligam entre os vértices do outro subgrafo.

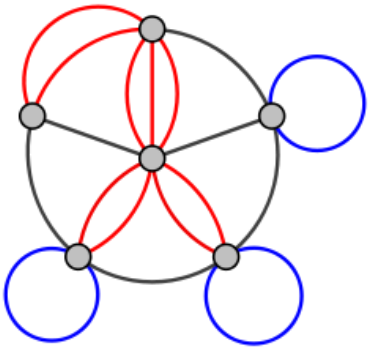


f) Grafos simples x multigrafo x dígrafo

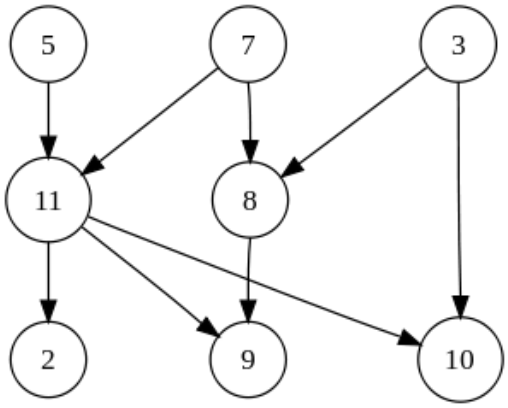
- **Grafo simples** é um tipo de grafo que não contém nem laços nem arestas múltiplas.



- **Multigrafo** é um tipo de grafo os vértices possuem arestas múltiplas.



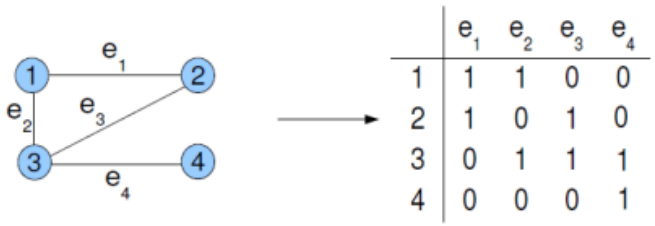
- **Dígrafo** é um tipo de grafo direcionando e simples.



[QUESTÃO – 04] Defina e apresente exemplos de matriz de incidência, matriz de adjacência e lista de adjacência. Adicionalmente, descreva o impacto (vantagens e desvantagens) da utilização de matriz de adjacência e lista de adjacência.

- **Matriz de incidência**

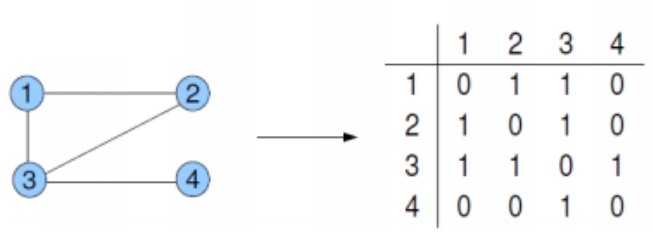
A matriz de incidência utiliza uma matriz m x n, onde n é o numero de vértices e m é o número de arestas. Para essas matrizes, os vértices são as linhas e as arestas são as colunas e cada elemento da matriz indica se aresta incide sobre o vértice. Consegue proporcionar um acesso constante, porém a utilização de matrizes demanda muito espaço.



	e_1	e_2	e_3	e_4
1	1	1	0	0
2	1	0	1	0
3	0	1	1	1
4	0	0	0	1

- **Matriz de adjacência:**

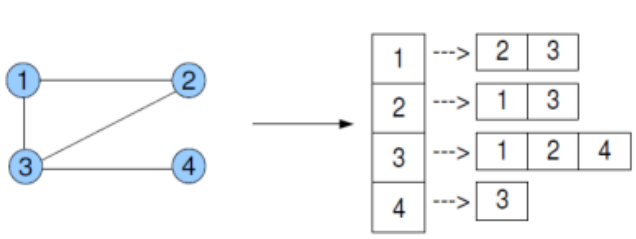
A matriz de adjacência de um grafo com |V| vértices é uma matriz |V|x|V| de 0s e 1s, na qual a entrada na linha i e coluna j é 1 se e somente se a aresta (i,j)(i,j) estiver no grafo. Uma vantagem da utilização de matriz de adjacência está no tempo constante de acesso (O(n)), e uma das desvantagens é a grande necessidade de espaço (Θ(V^2)).



	1	2	3	4
1	0	1	1	0
2	1	0	1	0
3	1	1	0	1
4	0	0	1	0

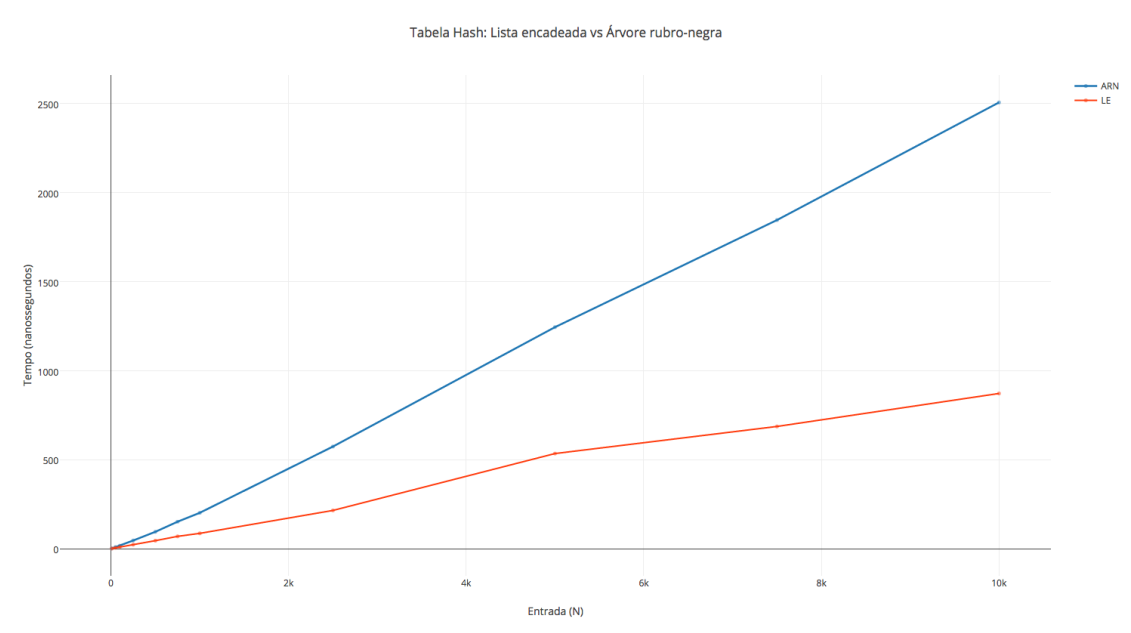
- **Lista de adjacência:**

A representação de um grafo com listas de adjacência combina as matrizes de adjacência com as listas de arestas. Para cada vértice do grafo, existe uma lista encadeada que armazena os vértices adjacentes. Uma vantagem é a economia de espaço, porém existe um aumento no tempo de acesso.



[QUESTÃO – 05] Comente sobre tabelas hash, apresentando a complexidade para as operações realizadas. Adicionalmente, implemente um tabela hash com encadeamento separado usando: lista encadeada e árvore vermelho e preto. Apresente um estudo empírico para obter custos de inserção na medida em que o número de chaves aumenta. Gere gráficos para mostrar o custo de inserção para tamanhos distintos de N (exemplo: de 10 a 10000). Descreva uma análise de comparação em relação ao tempo de execução.

É uma estrutura de dados derivada da tabela de endereçamento direto, porém sacrifica um pouco o desempenho com a finalidade de uma redução significativa do espaço de armazenamento. Esta estrutura tenta mapear o universo de chaves para um espaço menor através da função hash ou função de espalhamento.
Para o caso médio, o tempo de acesso é de O(1) chegando a O(n) para o pior caso, apresentando uma pequena perca de desempenho quando comparada com a tabela de endereçamento direto, que o acesso é sempre O(1).
Um dos grandes problemas da table hash são as colisões, que corre quando duas ou mais chaves são associadas ao mesmo índice. Por esse motivo é necessário a uma ótima função hash, e a aplicação de algumas técnicas, como a utilização de listas encadeadas ou arvores binarias de busca.



Compilar e executar o programa:

Os códigos do programa se encontram nos arquivos:

- questao5.c: Arquivo do programa principal;
- structs/src/hashTable.c: Implementação genérica da tabela hash;
- structs/src/hashTableDLL.c: Implementação da tabela hash com lista encadeada;
- structs/src/hashTableRBT.c: Implementação da tabela hash com árvore rubro-negra;
- structs/src/linkedList.c: Implementação da lista encadeada;
- structs/src/RBTree.c: Implementação da árvore rubro-negra;

Para compilar e executar, basta utilizar o script run.sh e passar como parâmetro o

arquivo questao5.c, seguidos dos parâmetros esperados pelo programa:

```
$ ./run.sh questao5.c <estrutura> <arquivo_de_entrada> <tamanho_do_buffer>
```

Onde:

- **<estrutura>**: Estrutura para ser utilizada pela tabela hash:
  - **dll**: para lista encadeada;
  - **rbt**: para árvore rubro-negra;
- **<arquivo_de_entrada>**: Arquivo de entrada contendo os valores que serão ordenados. Este arquivo pode ser gerado utilizando o programa generate.c;
- **<tamanho_do_buffer>**: Quantidade máxima de valores esperados pelo programa;

Exemplo:

```
$ ./run.sh questao5.c rbt input-1000-random.txt 1000
```

OBS.: Os resultados do programa serão gravados em arquivos no diretório output.

[QUESTÃO – 06] Defina, explicando as principais características e exemplifique:

a) Enumeração explícita x implícita:

- **Enumeração explicita:** Tenta encontrar todas as possibilidades.
- **Enumeração implícita:** Com base em algum conhecimento do problema, elimina algumas possibilidades e enumera o resto.

b) Programação Dinâmica:

Utiliza um esquema de enumeração de soluções que visa, através de uma abordagem de divisão-e-conquista (decomposição), minimizar o montante de computação a ser feito. A sua aplicação é indicada quando há casos em que um mesmo subproblema aparece diversas vezes ao longo do processo, onde a decomposição pura e simples é incapaz de reconhecer este fato, pois os subproblemas são resolvidos uma vez só e reutiliza a solução toda vez que o mesmo aparecer novamente.

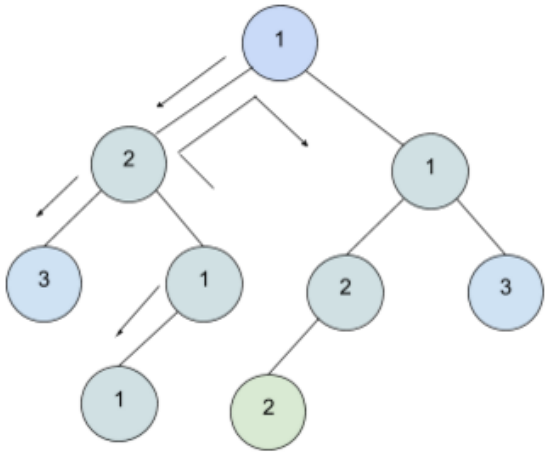
1º. Método: usando decomposição pura	2º. Método: aplicando PD
fibo (*n*) se *n*<= 2 então retornar *n*-1; senão retornar fibo(*n*-1) + fibo(*n*-2) ;	fib[1] := 0; fib[2] := 1; para *j* := 3 a *n* faça fib[*j*] := fib[*j*-1] + fib[*j*-2] ;

c) Algoritmo Guloso:

Tipo de algoritmos que tenta sempre fazer a melhor escolha local para ter uma melhor escolha global, geralmente atingindo soluções bem próximas da ótima, porém em alguns casos não encontra soluções tão ótimas. Um exemplo de utilização de algoritmos gulosos é encontrar uma solução do problema da mochila fracionária.

d) Backtracking:

É um tipo de algoritmo que representa um refinamento da busca por força bruta, em que múltiplas soluções podem ser eliminadas sem serem explicitamente examinadas.



[QUESTÃO – 09] Defina e exemplifique:

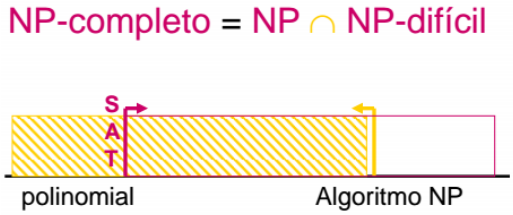
a) Problema SAT x Teoria da NP-Completude:

- **Problema SAT:** Seja uma expressão booleana na forma conjuntiva, deve existir uma valoração das variáveis que torne a expressão verdadeira.

$$\alpha = (x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

A resposta para α é “SIM” (x=1, y=1, z=0)

- **Teoria da NP-Completude:** Dentro da classe dos problemas NP-Difíceis, existe um conjunto de problemas maior ou igual ao problema do SAT.



b) Classes P, NP, NP-Difícil e NP-Completo:

- **Classe P:** Existe solução em tempo polinomial conhecida. Um exemplo é o cálculo do Máximo divisor comum;

- **Classe NP:** Existe uma solução não polinomial conhecida, porém toda solução pode ser verificada em tempo polinomial. Um exemplo é o problema do isomorfismo de grafos: determinar se dois grafos podem ser desenhados de forma idêntica.

- **Classe NP-Difícil:** Não existe solução. Um exemplo é problema de decisão da soma de subconjuntos.

- **Classe NP-Completo:** Existem soluções conhecidas, porém até hoje ninguém encontrou um algoritmo polinomial e também até hoje ninguém provou que tal algoritmo não existe. Um exemplo é o problema do caixeiro viajante.