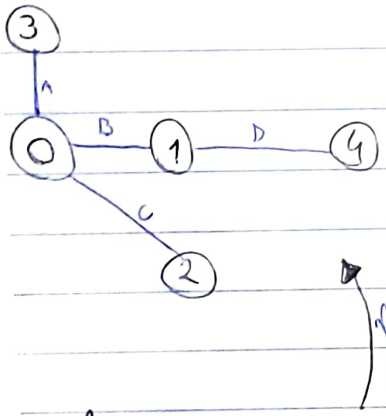
Grafo

Grafo: conjunto de vértices e arestas
Vértice: objeto com atributos
Aresta: conexão entre dois vértices

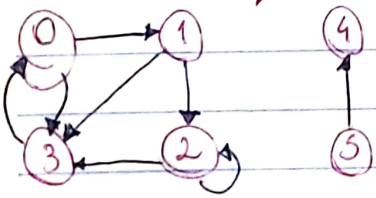
Notação



Grafo (G): G = (V, A)
Vértices (V): [0; 1; 2; 3; 4]
Arestas (A): A (3,0) ou (0,3)
B (0,1) ou (1,0)
C (0,2) ou (2,0)
D (1,4) ou (4,1)

Não direcionado

Grafo direcionado: uma aresta (u, v) sai de u e entra em v. u é adjacente a v.



grau → in-degree
out-degree

Grau de vértice: quantidade de arestas que nele incidem (ou dele saem também, se for direcionado)

Caminho

Sequência de vértices de tamanho K existente entre um vértice A e um B. Comprimento = nº arestas.

Ciclo

Um caminho que começa e termina no mesmo vértice

Ciclo simples ⇒ vértices todos distintos

Subgrafo

$G' = (V', A')$ é subgrafo de $G = (A, V)$ se $V' \subseteq V$ e $A' \subseteq A$

subgrafo próprio ⇒ $V' \neq V$ ou $A' \neq A$

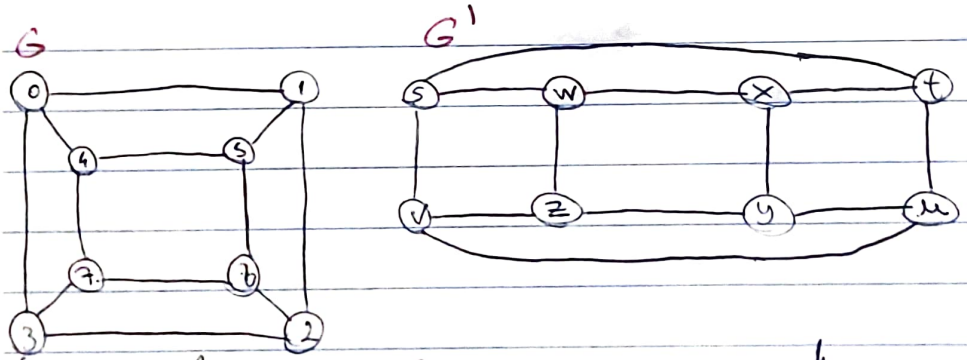
Grafo conexo → existe um caminho para todo par de vértices.

Componentes conexas → subgrafos conexos de um grafo

Fortemente Conexos (direcionados) → com um par de vértices um é alcançável a partir da outra e vice-versa.

Um grafo direcionado fortemente conectado tem apenas um componente fortemente conectado.

Isomorfo

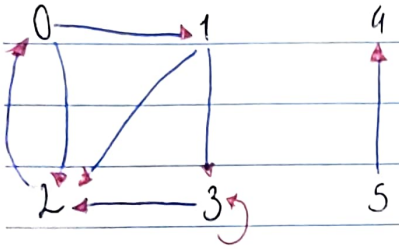


é possível re-rotular os vértices de G para rótulos de G', mantendo as arestas.

Vizinho (vz) e Adjacente (adj)

- para grafos direcionados
- dado V_w, um vizinho é qualquer adjacente a V_w em sua versão não-dir

- ligados por uma aresta.



- não dir
- dir

1 adj 3 ? S, N
1 vz 3 ? S, S
3 adj 1 ? S, S
3 vz 1 ? S, S

Grafo completo → toda par de vértice é adjacente!
Grafo ponderado → arestas tem peso
Grafo transposto → dado G, G' tem a direção das arestas inversas
Grafo bipartido → parte-se V de G em V_1 e V_2 e todas as arestas ligam os dois conjuntos.

Árvores

Livre → grafo não-dir, acíclo e conexo
Floresta → não-dir, acíclo
Geradora → subgrafo de um conexo que contém V e forma uma árvore
Floresta geradora → subgrafo que contém V e forma uma floresta

Busca

Busca em Profundidade (DFS)

- Partindo de um vértice inicial, ela explora o máximo possível cada um dos seus ramos antes de retroceder ("backtracking")

Usos:

- Encontrar componentes (fortemente) conectados
- Ordenação topológica
- Resolver outros - caminhos (labirinto)

```
void DFS (Grafo* grafo, int ini, int visitado[]) {
    visitado[ini] = 1;
    printf("%d ", ini);

    Aresta* arestaAtual = grafo->vertices[ini].arestas;
    while(arestaAtual != NULL) {
        if(!visitado[arestaAtual->dest]) {
            DFS(grafo, arestAtual->dest, visitado);
        }
        arestaAtual = arestaAtual->prox;
    }
}
```

Busca em Largura

- Partindo de um vértice inicial, ela explora todos os vértices vizinhos. Em seguida, para cada vértice vizinho ela repete esse processo, visitando os vértices ainda inexplorados

Pode ser usada para:

- Achar componentes conectados
- Achar todos os vértices conectados a apenas 1 componente
- Achar o menor caminho entre dois vertices
- Testar bipartição entre grafo.

```
void BFS (Grafo* grafo, int ini) {
    int visitado[MAX_VERT] = {0};
    int fila[MAX_VERT];
    int frent = 0, tras = -1;

    visitado[ini] = 1;
    fila[++tras] = ini;

    while (frent <= tras) {
        int atual = fila[frent++];
        printf("%d", atual);
        Aresta* arestaAtual = grafo->vertices[atual].arestas;
        while (arestaAtual != NULL) {
            if(!visitado[arestaAtual->dest]){
                visitado[arestaAtual->dest] = 1;
                fila[++tras] = arestaAtual->dest;
            }
            arestaAtual = arestaAtual->prox;
        }
    }
}
```