ESCOLA DE PRIMAVERA DA MARATONA SBC DE PROGRAMAÇÃO

PROMOÇÃO:

CEFET-MG

UFMG

UFV

UNIFEI

USP Universidade de São Paulo

UFU

APOIO:

beecrowd

SBC Sociedade Brasileira de Computação

Grupo de Computação Competitiva

REPRESENTAÇÕES DE GRAFOS

Por: *Lucas Andrade*

CONTEÚDOS

01 - INTRODUÇÃO AOS GRAFOS

Um grafo é composto por vértices (nós) e arestas (conexões). Cada vértice representa um objeto, e cada aresta indica uma relação entre dois vértices.

Direcionado: Arestas possuem direção, indicando uma relação unidirecional entre dois vértices

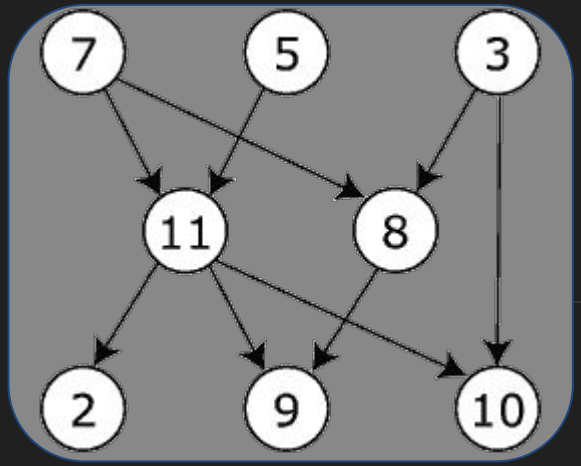
Não Direcionado: Arestas não possuem direção, representando uma relação bidirecional entre dois vértices.

Ponderado: Arestas possuem um peso associado, representando um custo ou distância entre dois vértices.

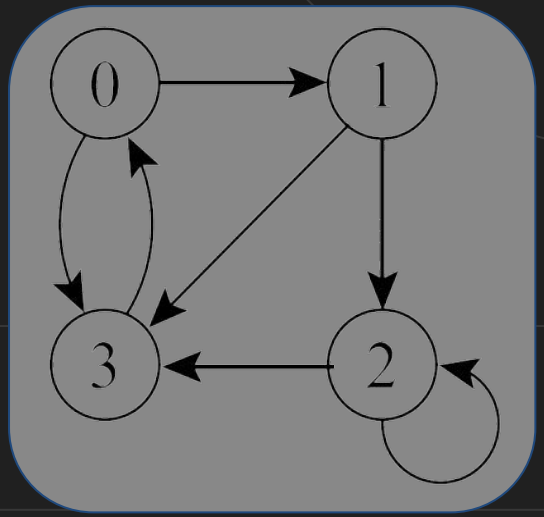
Não Ponderado: Arestas não possuem peso, representando apenas a existência de uma conexão entre dois vértices.

DIRECIONADO

Arestas possuem direção, indicando uma relação unidirecional entre dois vértices



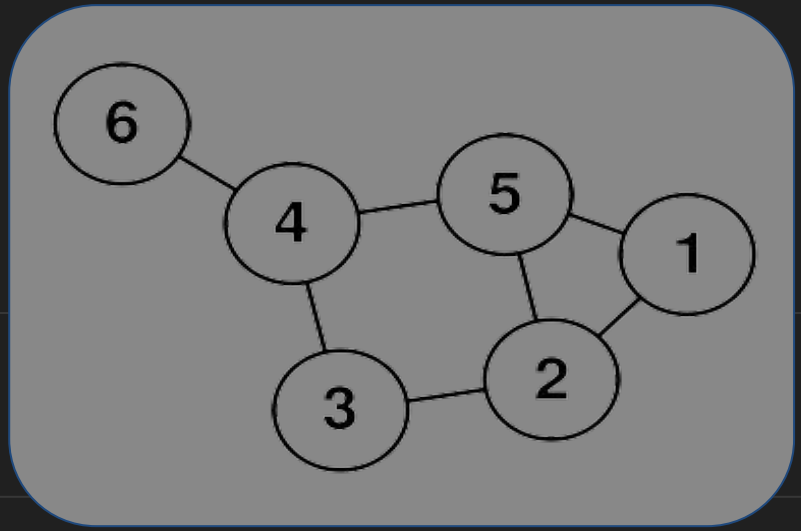
Acíclico



Cíclico

NÃO DIRECIONADO

Arestas não possuem direção, representando uma relação bidirecional entre dois vértices.



01 - INTRODUÇÃO AOS GRAFOS

Um grafo é composto por vértices (nós) e arestas (conexões). Cada vértice representa um objeto, e cada aresta indica uma relação entre dois vértices.

Direcionado: Arestas possuem direção, indicando uma relação unidirecional entre dois vértices

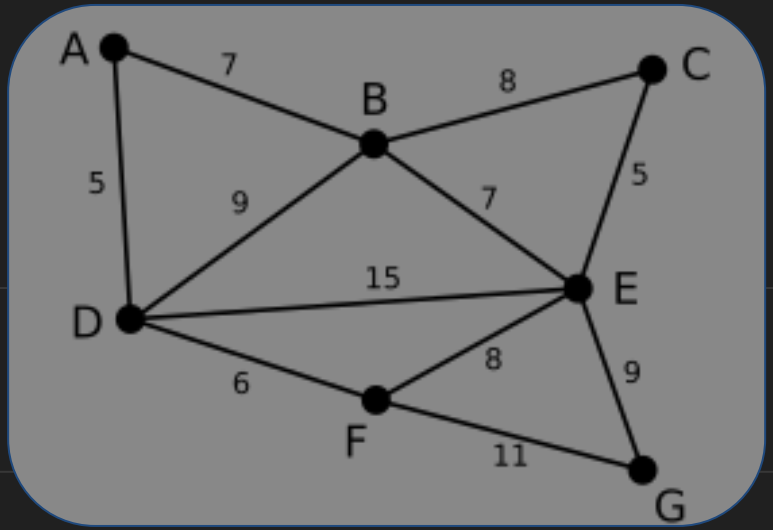
Não Direcionado: Arestas não possuem direção, representando uma relação bidirecional entre dois vértices.

Ponderado: Arestas possuem um peso associado, representando um custo ou distância entre dois vértices.

Não Ponderado: Arestas não possuem peso, representando apenas a existência de uma conexão entre dois vértices.

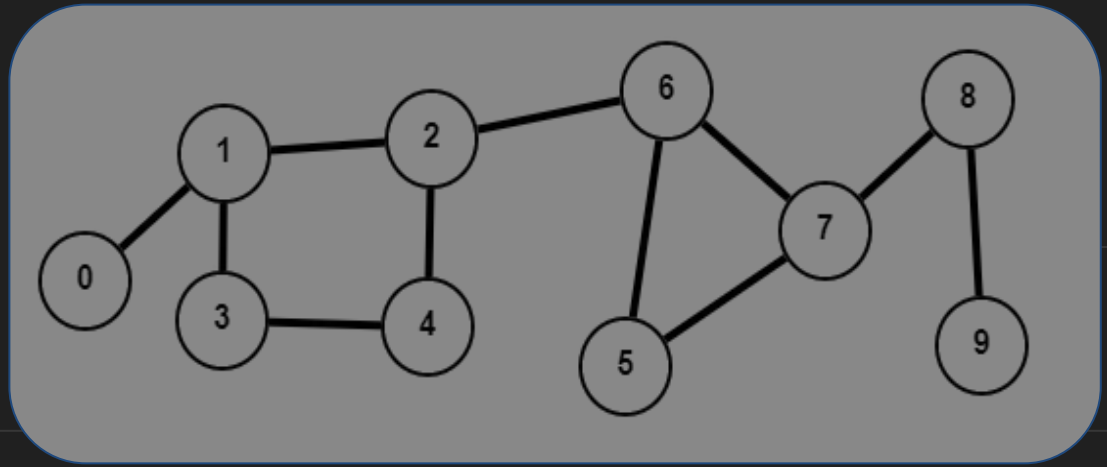
PONDERADO

Arestas possuem um peso associado, representando um custo ou distância entre dois vértices.



NÃO PONDERADO

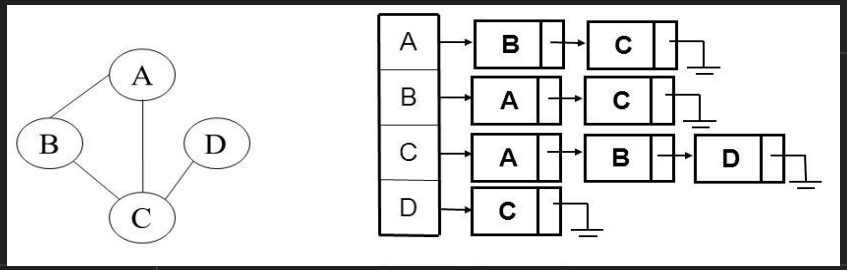
Arestas não possuem peso, representando apenas a existência de uma conexão entre dois vértices.

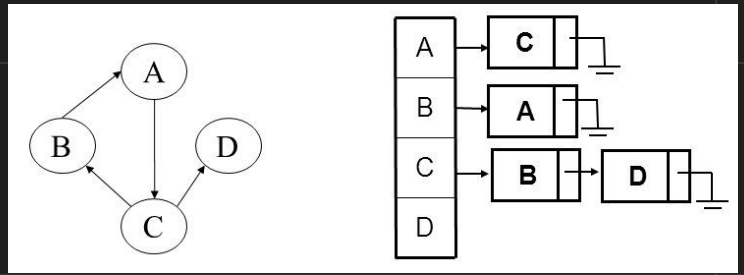


02 - LISTA DE ADJACÊNCIA

A lista de adjacência representa um grafo usando um array de listas. Cada índice do array representa um vértice, e a lista associada contém os vértices adjacentes.

Cada vértice possui uma lista com seus vizinhos. A lista de adjacência é mais eficiente para grafos esparsos.





VANTAGENS E DESVANTAGENS DA LISTA DE ADJACÊNCIA

Um grafo é composto por vértices (nós) e arestas (conexões). Cada vértice representa um objeto, e cada aresta indica uma relação entre dois vértices.

Vantagens:

- **Eficiente em termos de memória**
- **Ideal para grafos esparsos**

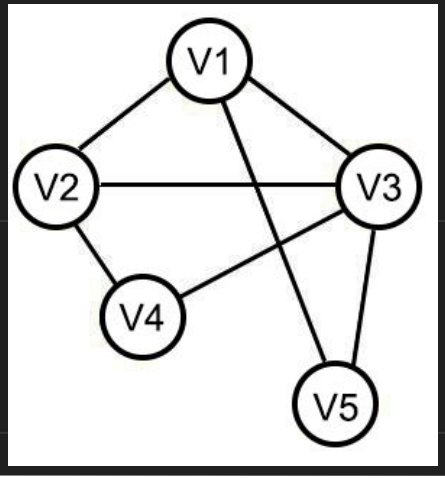
Desvantagens:

- **Mais complexa de implementar**
- **Verificação de conexão entre vértices pode ser mais lenta**

A lista de adjacência é a escolha ideal para grafos com poucos vértices adjacentes.

03 - MATRIZ DE ADJACÊNCIA

Uma matriz de adjacência é uma representação de um grafo onde cada posição da matriz corresponde a um par de vértices.
Se houver uma aresta entre os vértices i e j, a posição M[i][j] da matriz será 1. Caso contrário, será 0.



	V1	V2	V3	V4	V5
V1	0	1	1	0	1
V2	1	0	1	1	0
V3	1	1	0	1	1
V4	0	1	1	0	0
V5	1	0	1	0	0

VANTAGENS E DESVANTAGENS DA MATRIZ DE ADJACÊNCIA

Um grafo é composto por vértices (nós) e arestas (conexões). Cada vértice representa um objeto, e cada aresta indica uma relação entre dois vértices.

Vantagens:

- **Simples de implementar**
- **Verificação rápida de conexão entre vértices**

Desvantagens:

- **Ineficiente para grafos esparsos**
- **Consumo de memória alto**

A matriz de adjacência é mais adequada para grafos densos, com muitas conexões entre os vértices.

04 - COMPARAÇÃO: QUANDO USAR CADA REPRESENTAÇÃO

A escolha entre matriz de adjacência e lista de adjacência depende do tipo de grafo que você está representando.

Matriz de Adjacência: Ideal para grafos densos, onde a maioria dos vértices está conectada.
Lista de Adjacência: Ideal para grafos esparsos, onde a maioria dos vértices tem poucos vizinhos.

Para grafos densos, a matriz de adjacência é mais eficiente. Para grafos esparsos, a lista de adjacência é a melhor opção.

05 - EXERCÍCIO

1. Implemente um grafo não direcionado com 6 vértices, utilizando lista de adjacência. Percorra o grafo e imprima os vizinhos de cada vértice.

2. Implemente um grafo não direcionado com 6 vértices, utilizando matriz de adjacência. Percorra o grafo e imprima os vizinhos de cada vértice.

RESOLVENDO COM LISTA DE ADJACÊNCIA

```cpp
#include <iostream>
#include <vector>
using namespace std;

// Função para adicionar uma aresta entre dois vértices
void adicionarAresta(vector<int> adj[], int u, int v) {
    adj[u].push_back(v); // Adiciona v à lista de adjacência de u
    adj[v].push_back(u); // Adiciona u à lista de adjacência de v (não direcionado)
}

// Função para imprimir os vizinhos de cada vértice
void imprimirGrafo(const vector<int> adj[], int V) {
    for (int i = 0; i < V; ++i) {
        cout << "Vértice " << i << " tem como vizinhos: ";
        for (int vizinho : adj[i]) {
            cout << vizinho << " ";
        }
        cout << endl;
    }
}
```

```cpp
int main() {
    int V = 6; // Número de vértices

    // Lista de adjacência para o grafo com 6 vértices
    vector<int> adj[V];

    // Adicionar arestas ao grafo
    adicionarAresta(adj, 0, 1);
    adicionarAresta(adj, 0, 2);
    adicionarAresta(adj, 1, 3);
    adicionarAresta(adj, 1, 4);
    adicionarAresta(adj, 2, 4);
    adicionarAresta(adj, 3, 5);
    adicionarAresta(adj, 4, 5);

    // Imprimir os vizinhos de cada vértice
    imprimirGrafo(adj, V);

    return 0;
}
```

RESOLVENDO COM MATRIZ DE ADJACÊNCIA

```cpp
#include <iostream>
using namespace std;

// Função para adicionar uma aresta entre dois vértices
void adicionarAresta(int adj[][6], int u, int v) {
    adj[u][v] = 1; // Marca a aresta de u para v
    adj[v][u] = 1; // Marca a aresta de v para u (não direcionado)
}

// Função para imprimir os vizinhos de cada vértice
void imprimirGrafo(int adj[][6], int V) {
    for (int i = 0; i < V; ++i) {
        cout << "Vértice " << i << " tem como vizinhos: ";
        for (int j = 0; j < V; ++j) {
            if (adj[i][j] == 1) {
                cout << j << " ";
            }
        }
        cout << endl;
    }
}
```

```cpp
int main() {
    int V = 6; // Número de vértices

    // Matriz de adjacência para o grafo com 6 vértices
    int adj[6][6] = {0}; // Inicializa a matriz com 0 (sem conexões)

    // Adicionar arestas ao grafo
    adicionarAresta(adj, 0, 1);
    adicionarAresta(adj, 0, 2);
    adicionarAresta(adj, 1, 3);
    adicionarAresta(adj, 1, 4);
    adicionarAresta(adj, 2, 4);
    adicionarAresta(adj, 3, 5);
    adicionarAresta(adj, 4, 5);

    // Imprimir os vizinhos de cada vértice
    imprimirGrafo(adj, V);

    return 0;
}
```

06 - DÚVIDAS ?

OBRIGADO PELA ATENÇÃO

Grupo de Computação Competitiva

GCC.CEFETMG

gcc-cefetmg.github.io/Site-GCC

@gcc.cefetmg