Representação computacional de grafos - Lista de adjacências

TEG0001 - Teoria dos grafos
Prof. Dr. Ricardo José Pfitscher
ricardo.pfitscher@gmail.com

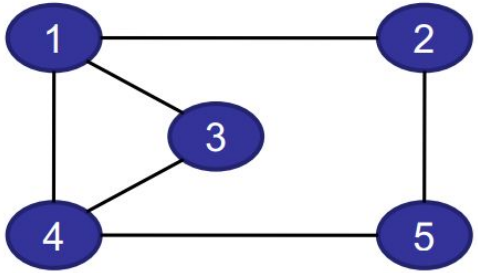
UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

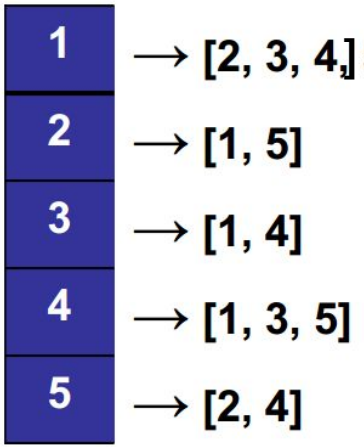
Aula passada - Exercícios

1. Implemente computacionalmente o grafo do exercício de mapeamento de bairro que você fez na aula passada.
2. Implemente métodos para:
    a. verificar se dois vértices são adjacentes e a distância entre eles
    b. retornar o grau de um dado vértice
    c. verificar se um grafo a é subgrafo do grafo b
    d. retornar o número de arestas do grafo
    e. verificar se o grafo é completo

Grafos - lista de adjacências

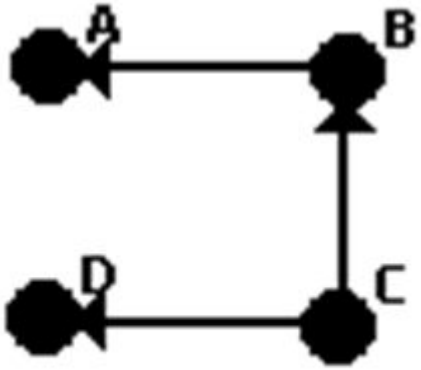
- Representação das adjacências entre vértices é feita através de listas lineares
- Constituição:
  - Formar um índice de vértices:
    - Vetor dinâmico
    - Lista encadeada
  - Para cada elemento do índice:
    - Lista encadeada, descreve os elementos adjacentes conectados





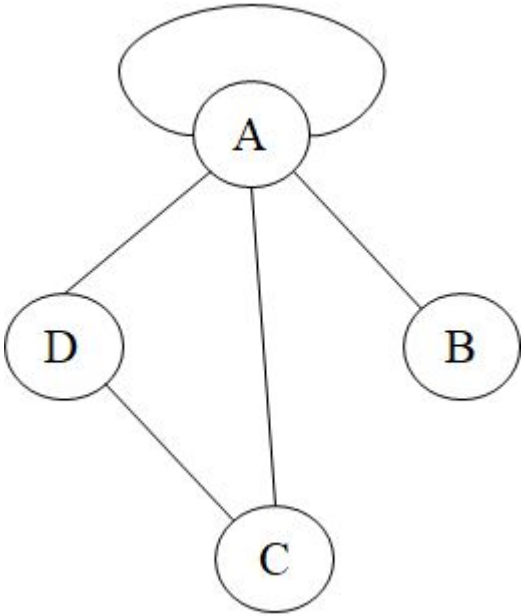
Grafos - lista de adjacências

- Como o seguinte dígrafo pode ser representado com uma lista de adjacências?



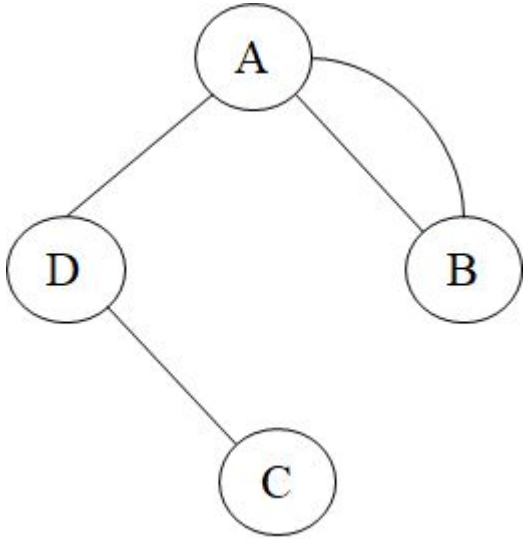
Grafos - lista de adjacências

- Como o seguinte grafo **com laço** pode ser representado com uma lista de adjacências?



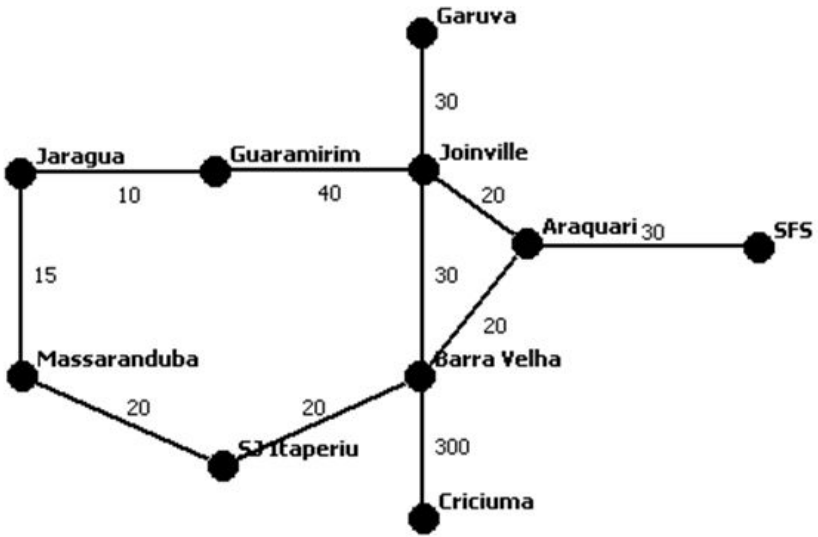
Grafos - lista de adjacências

- Como o seguinte grafo **com arestas paralelas** pode ser representado com uma lista de adjacências?



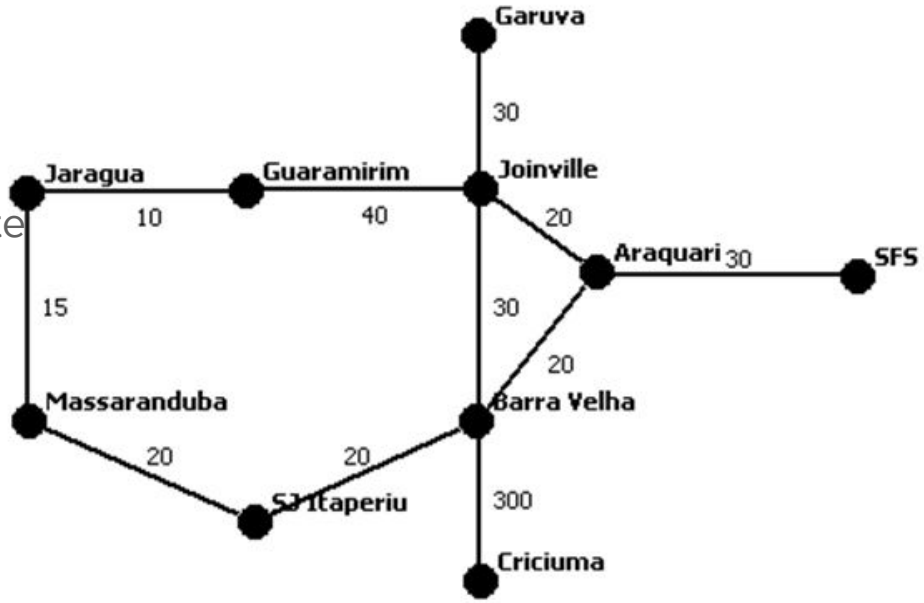
Grafos - lista de adjacências

- Como o seguinte grafo **ponderado** pode ser representado com uma lista de adjacências?



Grafos - lista de adjacências

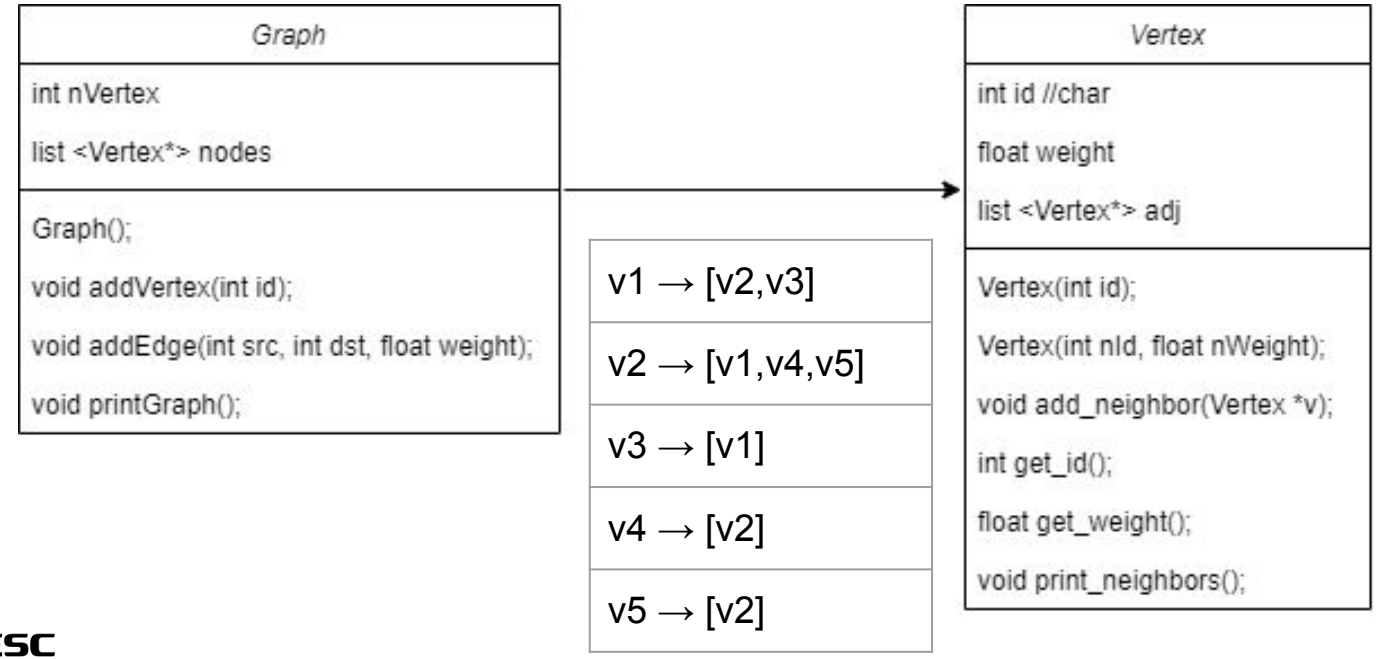
- Como o seguinte grafo **ponderado** pode ser representado com uma lista de adjacências?
  - A lista é formada por nós que contém:
    - o dado do vértice
    - Ponteiro para o vértice adjacente ao indicado no índice
  - Alguns casos obrigam algumas alterações nos nós:
    - Inserção de campos
      - Visita ao vértice
      - Valor de chegada



Grafos - lista de adjacências

- Vantagens:
  - Menor espaço para armazenamento
  - O(n + m) ➔ Isso é linear?
  - Implementação mais dinâmica
    - Mais fácil adicionar um novo nó
- Desvantagens:
  - O acesso é mais lento
  - O(n + m)

Implementação - modelagem



Implementação - C++

```
class Vertex{
    int id;
    float weight;
    list<Vertex*> adj;
    public:
        Vertex(int id);
        Vertex(int nId, float nWeight);
        void add_neighbor(Vertex *v);
        int get_id();
        float get_weight();
        void print_neighbors();

};
```

```
class Graph{
    int nVertex;
    list <Vertex*> nodes;

    public:
        Graph();
        void addVertex(int id);
        void addEdge(int src, int dst, float weight);
        void printGraph();
```

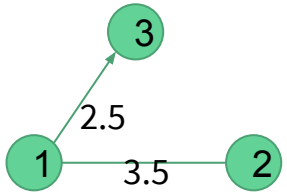
Implementação - C++

```
void Graph::addVertex(int id){
    Vertex *v = new Vertex(id);
    nodes.push_back(v);
    nVertex++;
}
```

```
void Graph::addEdge(int src, int dst, float weight){
    Vertex *tmp = new Vertex(dst, weight);
    Vertex *v;
    list<Vertex*>::iterator it;
    for(it = nodes.begin(); it!=nodes.end() ; it++ ){
        v = *it;
        if(v->get_id() == src){
            v->add_neighbor(tmp);
            break;
        }
    }

}
```

Implementação - C++




```
main(){
     Graph *g = new Graph();
     g->addVertex(1);
     g->addVertex(2);
     g->addVertex(3);
     g->addEdge(1,2,3.5);
     g->addEdge(2,1,3.5);
     g->addEdge(1,3,2.5);
     g->printGraph();

}
```

Python - Utilizando dicionários

- Um dicionário é um array associativo em python, onde você associa uma chave a um valor
  - Exemplo

```
agendaWhats = {}
agendaWhats['Ricardo'] = '+5547988435709'
agendaWhats['Policia'] = '190'
print (agendaWhats)
if 'Policia' in agendaWhats:
    print(agendaWhats['Policia'])
```

  - Mais informações link

Python - Utilizando dicionários - Grafo

- Podemos utilizar um dicionário para armazenar vértices como índices e as arestas como uma lista
- Que grafo é esse?

```
graph = {'A': ['B', 'C'],
             'B': ['C', 'D'],
             'C': ['D'],
             'D': ['C'],
             'E': ['F'],
             'F': ['C']}
```

Python - Grafo ponderado

Ao invés de armazenar uma lista de inteiros para cada vértice, podemos armazenar, para cada vértice, um dicionário dos destinos e custos
Fonte: link

```
class Vertex:
    def __init__(self, node):
        self.id = node
        self.adjacent = {}
    def add_neighbor(self, neighbor, weight=0):
        self.adjacent[neighbor] = weight
```

```
class Graph:
    def __init__(self):
        self.vert_dict = {}
        self.num_vertices = 0

    def add_vertex(self, node):
        self.num_vertices = self.num_vertices + 1
        new_vertex = Vertex(node)
        self.vert_dict[node] = new_vertex
        return new_vertex

    def add_edge(self, frm, to, cost = 0):
        self.vert_dict[frm].add_neighbor(self.vert_dict[to], cost)
        self.vert_dict[to].add_neighbor(self.vert_dict[frm], cost)
```

Implementação - atividade

- Desenvolva um programa para implementar a TDA grafo.
  - Utilizar lista de adjacências
  - O grafo deve ser ponderado
  - Oferecer um menu ao usuário com as seguintes opções:
    i. Adicionar vértices e arestas
    ii. Calcular o grau de um dado vértice
    iii. Mostrar os conjuntos de vértices (V) e arestas (E)
    iv. Responder se um vértice é alcançável **diretamente** a partir de outro