Aula 5

## Estrutura de Dados

Prof. Vinicius Pozzobon Borin

## Conversa Inicial

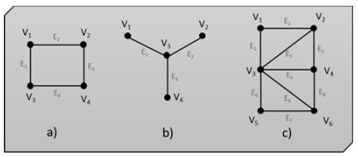
- O objetivo desta aula é apresentar os conceitos que envolvem a estrutura de dados do tipo grafo
- Será mostrado como realizar as manipulações dos dados dentro de uma estrutura de grafos, como construir o grafo e como buscar e andar pelos vértices de um grafo

- Representação de grafos:
  - Matriz de Incidências
  - Matriz de Adjacências
  - Lista de Adjacências
- Descoberta do grafo:
  - Algoritmo de busca em profundidade
  - Algoritmo de busca em largura
- Caminho mínimo do grafo:
  - Algoritmo de *Dijkstra*

## Grafos: Definições

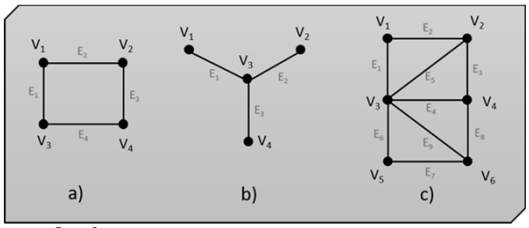
## Grafos

- Um Grafo G é um conjunto de vértices conectados por meio de arestas sem uma distribuição fixa ou padronizada



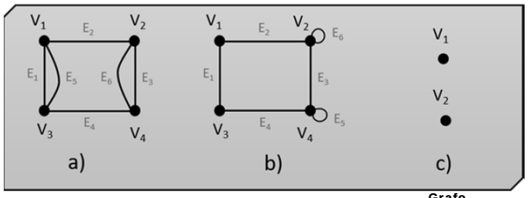

Grafo completo Grafo completo

## Grau de um grafo



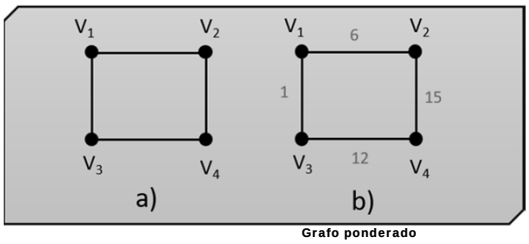

Grau 2 | Graus 1 e 3 | GRAUS 2, 3 e 5

## Particularidades em grafos




Arestas múltiplas | Laços | Grafo Desconexo

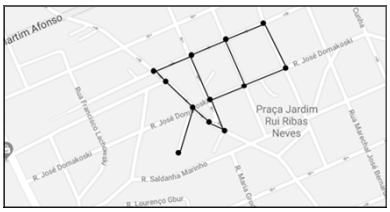
## Grafo ponderado




Grafo ponderado

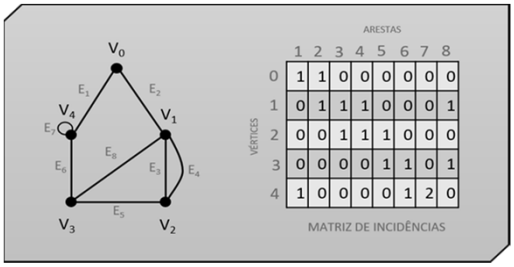
## Aplicação

- Mapa rodoviário

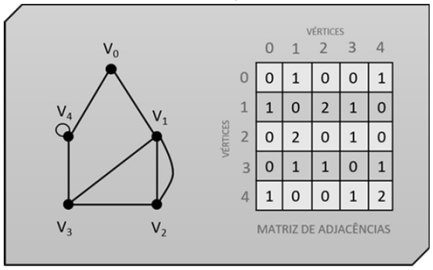



## Representação de Grafos

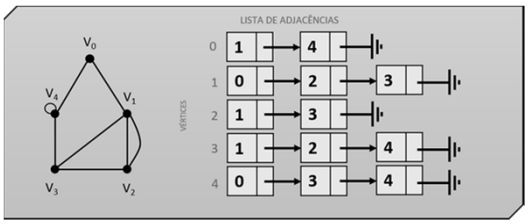
## Matriz de incidências



## Matriz de adjacências




## Lista de adjacências




- **Com vetor:**

```
1  //Vetor do tamanho do número de vértices do grafo
2  //O vetor conterá o endereço o primeiro vizinho (variável ponteiro)
3  Vertices[NUMERO_DE_VERTICES]: ListaDeVizinhos[->]
4
5  //Lista encadeada de vizinhos de cada vértice
6  registro ListaDeVizinhos
7     prox: ListaDeVizinhos[->]
8  fimregistro
```

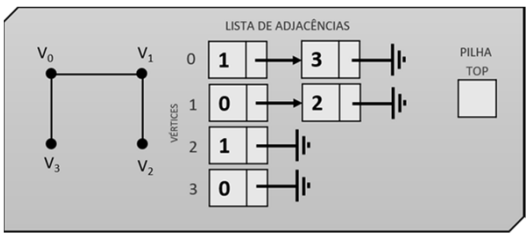
- **Com listas:**

```
1   //Lista encadeada com os vértices e o primeiro vizinho
2   registro Vertices
3      numero: inteiro
4      prox: Vertices[->]
5   fimregistro
6
7   //Lista encadeada de vizinhos de cada vértice
8   registro ListaDeVizinhos
9      prox: Vertices[->]
10  fimregistro
```

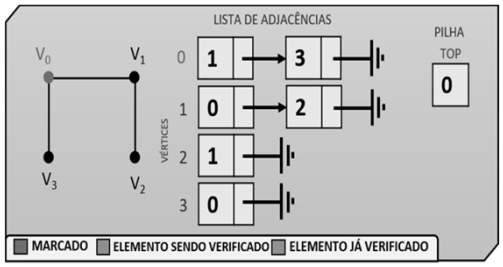
## Algoritmo de Busca em Profundidade no Grafo

## Busca em profundidade

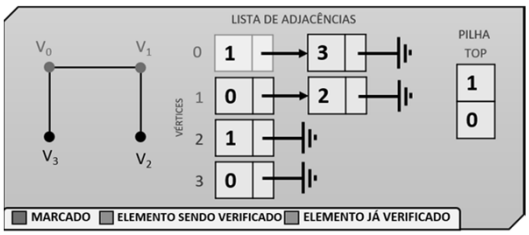
- **Estado inicial**




- **Etapa 1**



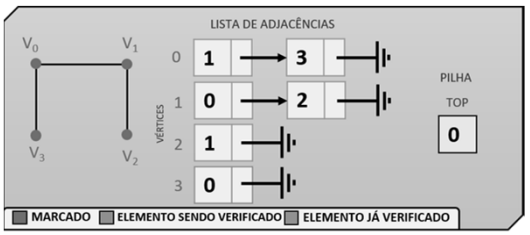
- **Etapa 2**




- **Etapa 3**


- **Etapa 4**


- **Etapa 5**




- **Etapa 6**


- **Pseudocódigo**

```
1  função BuscaProfundidade (Vizinhos: ListaDeVizinhos, tam: inteiro,
2  v: inteiro, marcado[NUMERO_VERTICES]: inteiro, Top: Pilha[->])
3  var
4        vert: vertice[->] //Variável de varredura
5        w, i: inteiro
6  inicio
7     marcado[v] = 1 //Marca o vértice como visitado
8     Empilhar(Top, v) //Inserindo o vértice na pilha
9     para i de 1 até tam faça
10       //Localiza os vizinhos do vértice 'v', iniciando no 1º
11       vert = Vizinhos[v].Vertices
12       enquanto (vert <> NULO) faça
13             w = vert->numero
14             se (marcado[w] <> 1) então
15                //Acessa recursivamente o próximo vértice não visitado
16                BuscaProfundidade(Vizinhos, tam, w, marcado, Top)
17             fimse
18             vert = vert->prox  //Próximo vizinho de 'v'
19       fimenquanto
20    fimpara
21    Desempilhar(Top) //Remove o vértice na pilha
22 fimfunção
```

**Algoritmo de Busca**
**em Largura no Grafo**

## Busca em largura

- **Estado inicial**


- **Etapa 1**


- **Etapa 2**


- **Etapa 3**


- **Etapa 4**

- **Etapa 5**


- **Etapa 6**


- **Etapa 7**


- **Pseudocódigo**

```
1  função BuscaLargura (Vizinhos: ListaDeVizinhos, tam: inteiro, v: inteiro,
2  marcado[NUMERO_VERTICES]: inteiro, distancias[NUMERO_VERTICES]: inteiro,
3  Head: Fila[->])
4  var
5        vert: vertice[->] //Variável de varredura
6        w, i, v_uso: inteiro
7  inicio
8     marcado[v] = 1 //Marca o vértice como visitado
9     distancias[v] = 0 //Distância de 'v' para ele mesmo é zero
10    InserirNaFila(Head, v) //Insere o vértice na fila
```

- **Continuação...**

```
11    enquanto (Head <> NULO) faça
12       v_uso = RemoverDaFila(Head) //Remove o vértice da fila
13       para i de 1 até tam faça
14          //Varre a lista de vizinhos do vértice
15          vert = Vizinhos[v_uso].Vertices
16          enquanto (vert <> NULO)
17             w = vert->numero
18             //Se vertice não estiver, marcado calcula a distância
19             //em relação a origem
20             se (marcado[w] == 0) então
21                marcado[w] = 1
22                distancias[w] = distancias[v_uso] + 1
23                //Insere ele na fila para visitar seus vizinhos depois
24                InserirNaFila (Head, w)
25             fimse
26             //Proximo vértice adjacente a 'v'
27             vert = vert->prox
28          fimenquanto
29       fimpara
30    fimenquanto
31 fimfunção
```

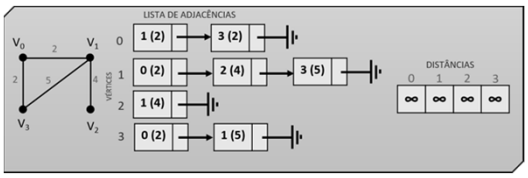
**Algoritmo do Caminho Mínimo em Grafo: *Dijkstra***

## Dijkstra

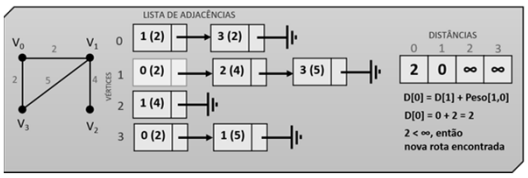
- **Pseudocódigo do grafo ponderado**

```
1   //Lista encadeada com os vértices e o primeiro vizinho
2   registro Vertices
3       numero: inteiro
4       peso: inteiro
5       prox: Vertices[->)
6   fimregistro
7
8   //Lista encadeada de vizinhos de cada vértice
9   registro ListaDeVizinhos
10      prox: Vertices[->)
11  fimregistro
```

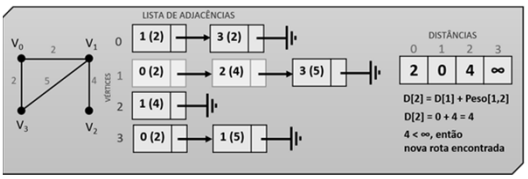
- **Estado inicial**



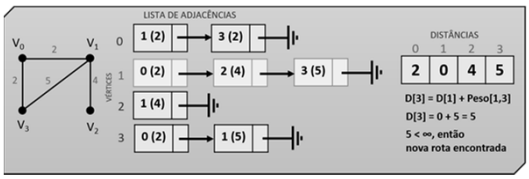

- **Etapa 1**



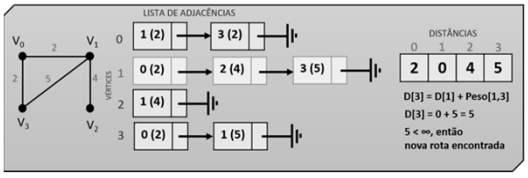

- **Etapa 2**



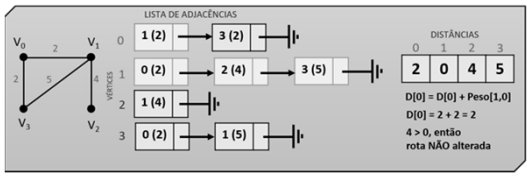

- **Etapa 3**



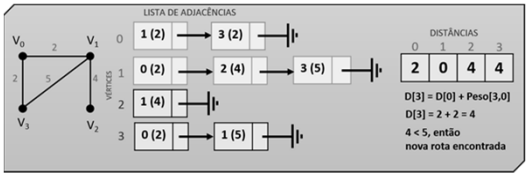

- **Etapa 4**



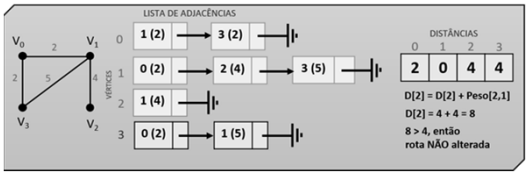
- **Etapa 5**



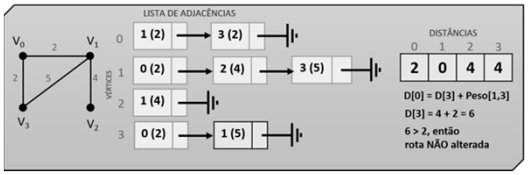

- **Etapa 6**



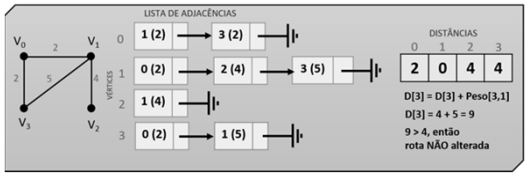

- **Etapa 7**



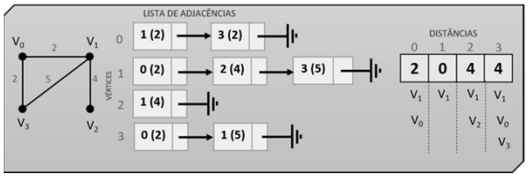

- **Etapa 8**




- **Etapa 9**




- **Etapa 10**



- **Pseudocódigo**

```
1   função Dijkstra (Origem: inteiro)
2   var
3       distancias[NUMERO_VERTICES]: inteiro
4       predecessor[NUMERO_VERTICES]: inteiro
5       visitado[NUMERO_VERTICES]: inteiro
6       cont: inteiro //Numero de nós já visitados
7       D_min, prox, i, j: inteiro
8       vert: vertice[->)
9   inicio
10      para i de 0 até (NUMERO_VERTICES - 1) faça
11          distancias[i] = INFINITO
12          predecessor[i] = Origem
13          visitado[i] = 0
14      fimpara
15      distancias[Origem] = 0
16      cont = 0
```

- **Continuando...**

```
18      enquanto (cont < NUMERO_VERTICES - 1) faça
19          D_min = INFINITO
20          //Próximo vértice é aquele com a menor distância
21          para i de 0 até (NUMERO_VERTICES - 1) faça
22              se ((distancias[i] < D_min) E (visitado[i]) == 0) então
23                  D_min = distancias[i]
24                  prox = i
25              fimse
26          fimpara
27          visitado[prox] = 1
28          enquanto (vert <> NULO) faça
29              se (visitado[vert->numero] == 0) então
30                  se ((D_min + vert->peso) < distancias[vert->numero]) então
31                      distancias[vert->numero] = D_min + vert->peso
32                      predecessor[vert->numero] = prox
33                  fimse
34              fimse
35              cont = cont + 1
36              vert = vert->prox
37          fimenquanto
38      fimenquanto
39  fimfunção
```

## Referências


- **ASCENCIO, A. F. G. Estrutura de dados: algoritmos, análise da complexidade e implementações em JAVA e C/C++. São Paulo: Pearson, 2011._____. Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ (padrão ANSI) JAVA. 3. ed. São Paulo: Pearson, 2012.**
- **CORMEN, T. H. Algoritmos. Teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.**
- **LAUREANO, M. Estrutura de dados com algoritmos e C. Rio de Janeiro: Brasport, 2008.**
- **MIZRAHI, V. V. Treinamento em linguagem C. 2. ed. São Paulo: Peason, 2008.**
- **PUGA, S.; RISSETI, G. Lógica de programação e estrutura de dados. 3. ed. São Paulo: Pearson, 2016.**