INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

Ciência da
Computação

ESTRUTURA DE DADOS II

Prof. Adilso Nunes de Souza

MENOR CAMINHO

- O menor caminho entre dois vértices é a aresta que os conecta. No entanto, é muito comum em um grafo não existir uma aresta conectando dois vértices, isto é, eles não são adjacentes.
- Apesar disso, dois vértices podem ser conectados por uma sequência de arestas.
- Caso essa seja a menor sequência de arestas dizemos que é o **menor caminho**, **caminho mais curto** ou **caminho geodésico** entre eles.

MENOR CAMINHO

- Segundo BACKES, 2016 o menor caminho entre dois vértices é o caminho que apresenta o menor comprimento dentre todos os possíveis que conectam esses vértices.
- Geralmente o comprimento se refere ao número de arestas que conectam os dois vértices.
- Considerando um grafo ponderado, podemos calcular o menor caminho com a soma dos pesos das arestas que compõem o caminho.

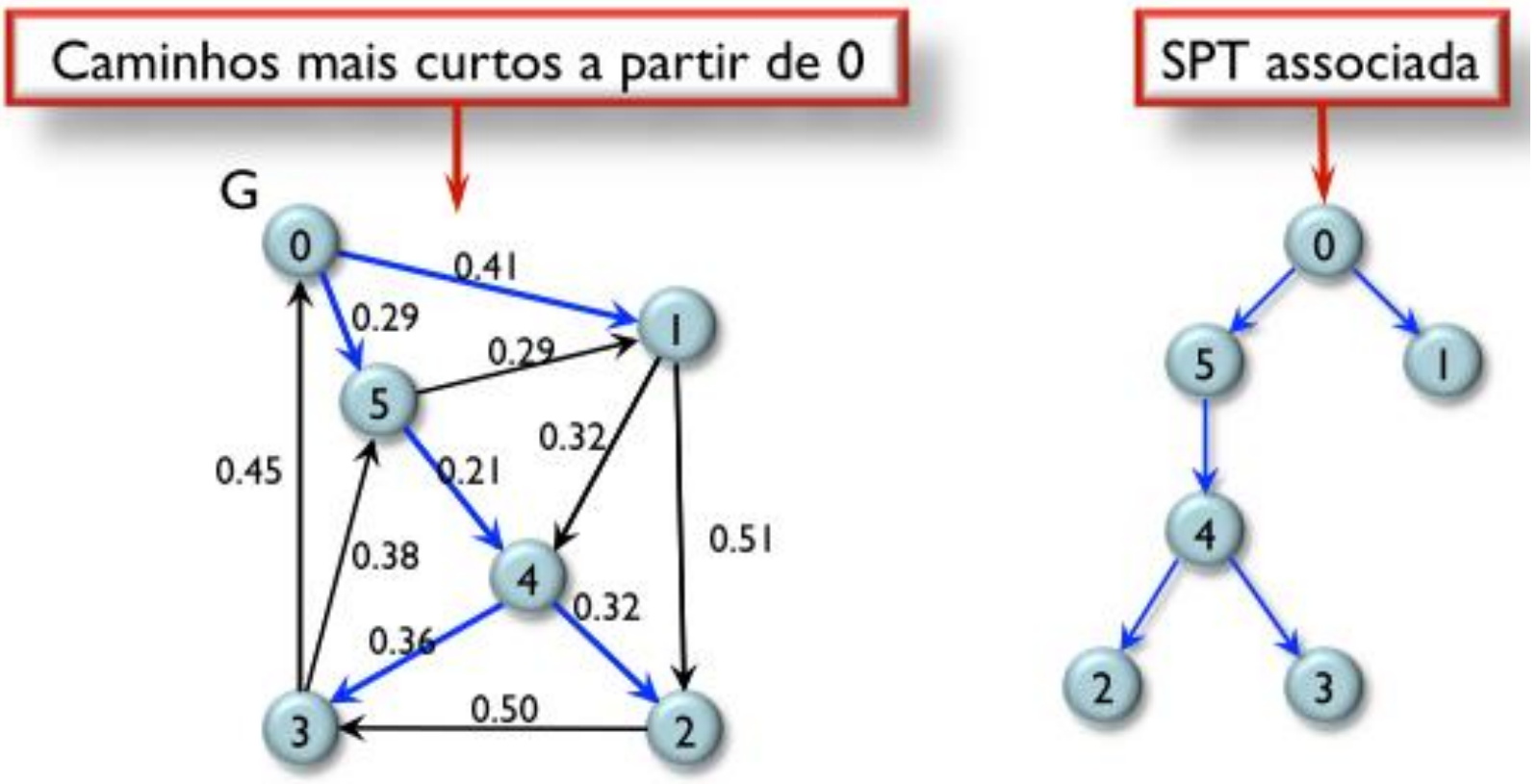
MENOR CAMINHO

- Em grafos não ponderados (ou arestas de mesmo peso) a busca em largura soluciona o problema do caminho mais curto entre dois vértices.
- São conhecidos como problemas de caminhos mais curtos - “shortest path problems”.
- Para estes algoritmos é necessário utilizar uma estrutura do tipo árvore, denominada SPT (Shortest Path Tree) árvore de caminho mais curto.

SPT

- É um subgrafo contendo s e todos os vértices alcançáveis a partir de s que forma uma árvore direcionada com raiz em s tal que todo caminho da árvore é um caminho mínimo no grafo.

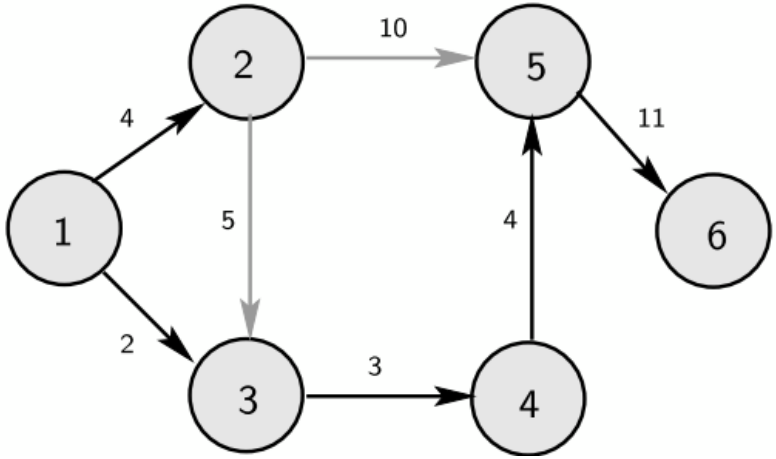
SPT


Caminhos mais curtos a partir de 0
G
0
1
2
3
4
5
0.41
0.29
0.29
0.32
0.45
0.21
0.38
0.51
0.32
0.36
0.50
SPT associada
0
5
1
4
2
3

SPT

- A árvore de caminhos mínimos (SPT) pode ser representada por dois vetores indexados pelos vértices v
  - dist[v]: armazena o comprimento do caminho mínimo entre s e v.
  - prev[v]: armazena a última aresta do caminho mínimo entre s e v.

SPT

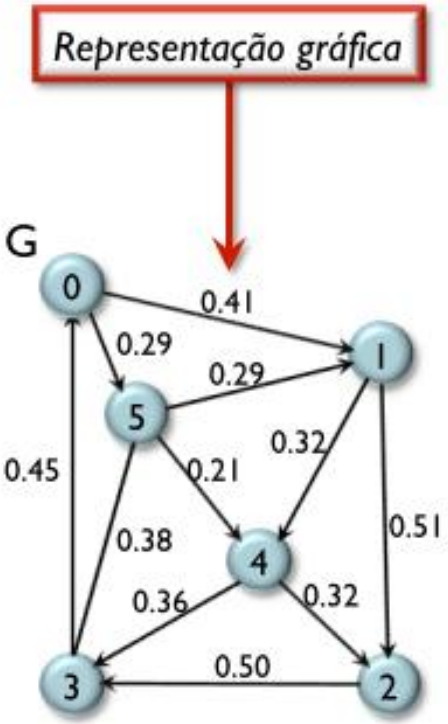


v	prev[v]	dist[v]
1	–	0
2	(1,2)	4
3	(1,3)	2
4	(3,4)	5
5	(4,5)	9
6	(5,6)	20

- OBS: Árvore de caminhos mínimos representado pelas setas em preto.

TIPOS DE BUSCA

- Problema 1 - Caminho mais curto fonte-destino:
  - Dado um vértice inicial s e um vértice destino t, qual o caminho mais curto no grafo de s para t



S = 0

T = 2

Caminhos possíveis:

0 – 1 – 2: 0.92

0 – 1 – 4 – 2: 1.05

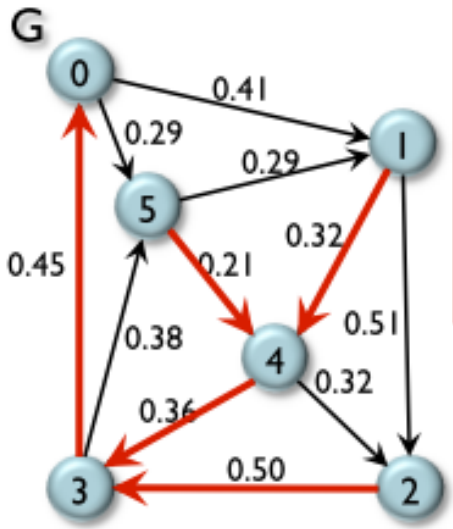
0 – 5 – 4 – 2: 0.82

Caminho mais curto:

0 – 5 – 4 – 2: 0.82

TIPOS DE BUSCA

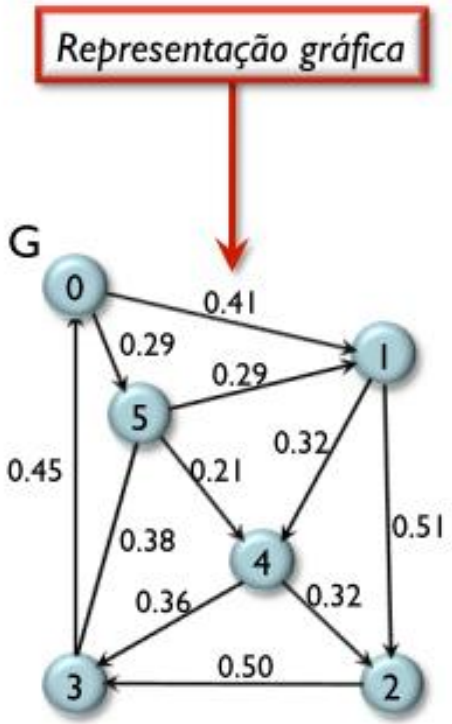
- Problema 2 - Caminhos mais curtos de fonte única:
  - Dado um vértice inicial s, quais os caminhos mais curtos que ligam todos os outros vértices à s?
  - S = 0



	0
1.13	1-4-3-0
0.95	2-3-0
0.45	3-0
0.81	4-3-0
1.02	5-4-3-0

TIPOS DE BUSCA

- Encontre os caminhos mais curtos para os diferentes casos:
  - S = 1
  - S = 4
  - S = 5



TIPOS DE BUSCA

- Problema 3 - Caminhos mais curtos entre todos:
  - Quais os caminhos mais curtos ligando todos os vértices de um grafo?
  - Pode ser resolvido aplicando o algoritmo V vezes, uma vez para cada vértice origem.

APLICAÇÕES

- Rotas de veículos
- Planejamento de tráfego urbano
- Navegação robótica
- Roteamento em telecomunicações
- Mapa de conexões de voo
- Redes elétricas e telefônicas

PESOS DAS ARESTAS

- Intuitivamente as arestas denotam distâncias, mas pesos de arestas podem representar outras medidas que não sejam distâncias, como tempo, custo, multas, prejuízos ou qualquer outra quantidade que se acumule linearmente ao longo de um caminho e que seria interessante minimizar.

ALGORITMOS

- Exemplos de algoritmos para calcular o menor caminho:
  - Prim
  - Kruskal
  - Bellman-Ford
  - Dijkstra
  - Floyd-Warshall

PRIM

- O algoritmo de PRIM é um algoritmo clássico capaz de obter uma solução ótima para o problema da árvore geradora mínima.
- Uma árvore geradora mínima (spanning tree) é um subgrafo que contém todos os vértices do grafo original e um conjunto de arestas que permite conectar todos esses vértices na forma de uma árvore.

PRIM

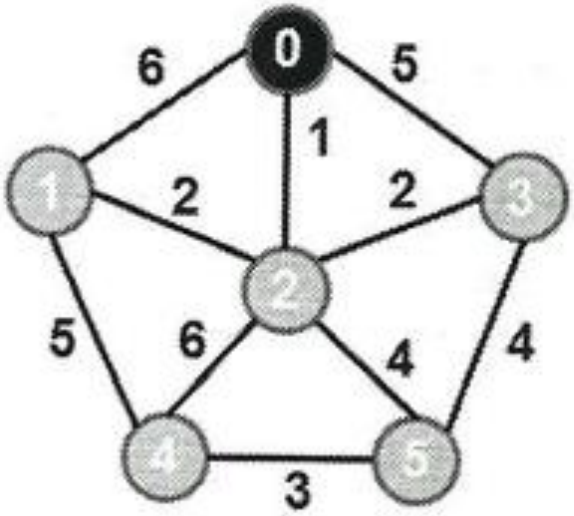
- Para um grafo possuir uma árvore geradora mínima ele deve satisfazer as seguintes propriedades: não direcionado, conexo e ponderado.
- O algoritmo de PRIM inicia a árvore com um vértice qualquer e adiciona um novo vértice à árvore a cada iteração. Esse processo continua até que todos os vértices do grafo façam parte da árvore, ou não seja possível achar uma aresta de menor peso conectando os vértices.

PRIM

- O algoritmo de PRIM pode ser implementado com uma função que recebe o grafo, o vértice que será o ponto de partida para crescer a árvore e um array para marcar quem é o pai de cada vértice na árvore resultante.

PRIM - Exemplo

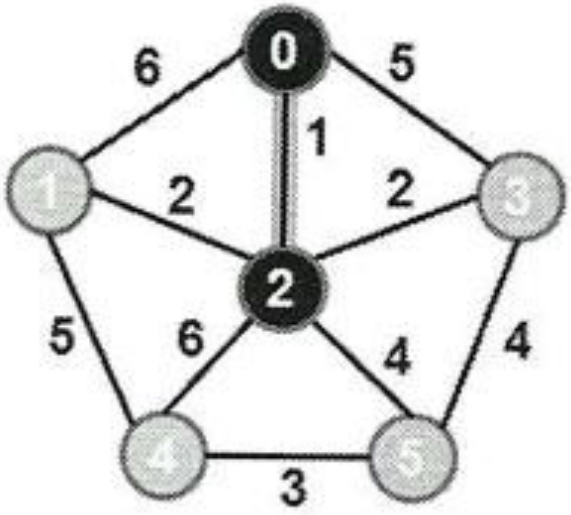
- Inicia o cálculo com o vértice 0.
- Atribui seu próprio índice como pai e o restante dos vértice recebem pai igual a -1 (sem pai)



pai
0
-1
-1
-1
-1
-1

PRIM - Exemplo

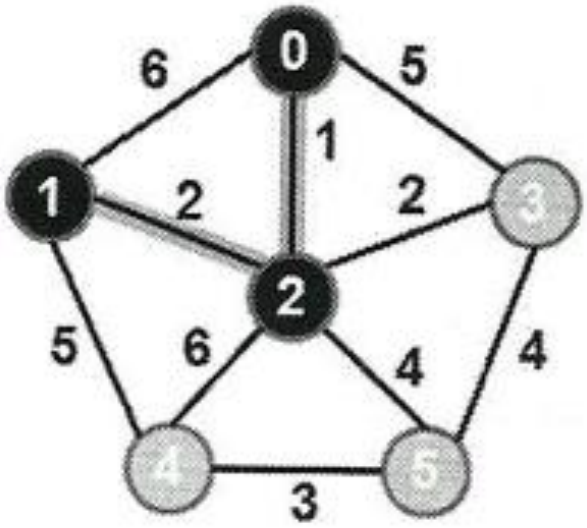
- Procura nos vértices com pai por um vértice sem pai e com menor peso: vértice 2
- Atribui vértice 0 como pai do vértice 2.



pai
0
-1
0
-1
-1
-1

PRIM - Exemplo

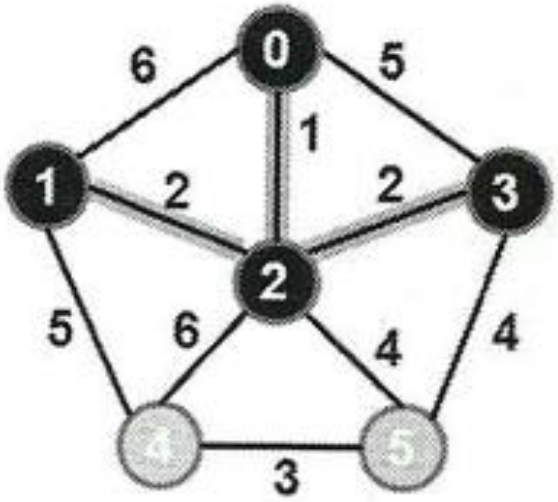
- Procura nos vértices com pai por um vértice sem pai e com menor peso: vértice 1.
- Atribui vértice 2 como sendo pai do vértice 1.



pai
0
2
0
-1
-1
-1

PRIM - Exemplo

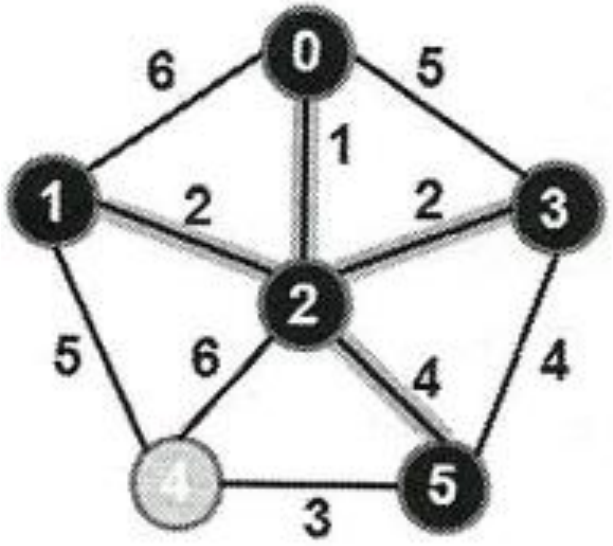
- Procura nos vértices com pai por um vértice sem pai e com menor peso: vértice 3.
- Atribui vértice 2 como pai do vértice 3



pai
0
2
0
2
-1
-1

PRIM - Exemplo

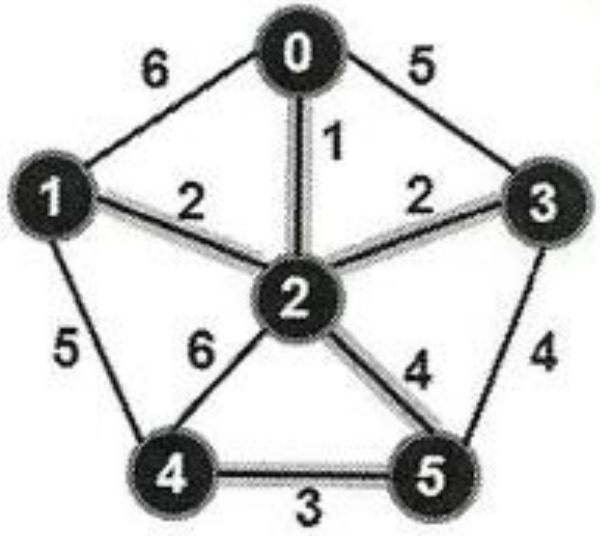
- Procura nos vértices com pai por um vértice sem pai e com menor peso: vértice 5.
- Atribui vértice 2 como pai do vértice 5



pai
0
2
0
2
-1
2

PRIM - Exemplo

- Procura nos vértices com pai por um vértice sem pai e com menor peso: vértice 4.
- Atribui vértice 5 como pai do vértice 4



pai
0
2
0
2
5
2

PRIM - Exemplo

```
01  void algoritmoPRIM_Grafo(Grafo *gr,int orig,int *pai){
02    int i, j, dest, NV, primeiro;
03    double menorPeso;
04    NV = gr->nro_vertices;
05    for(i=0; i < NV; i++)
06        pai[i] = -1;// sem pai
07
08    pai[orig] = orig;
09    while(1){
10        primeiro = 1;
11        for(i=0; i < NV; i++){
12            if(pai[i] != -1){
13                for(j=0; j<gr->grau[i]; j++){
14                    if(pai[gr->arestas[i][j]] == -1){
15                        if(primeiro){
16                           menorPeso = gr->pesos[i][j];
17                           orig = i;
18                           dest = gr->arestas[i][j];
19                           primeiro = 0;
20                        }else{
21                          if(menorPeso > gr->pesos[i][j]){
22                             menorPeso = gr->pesos[i][j];
23                             orig = i;
24                             dest = gr->arestas[i][j];
25                          }
26                        }
27                    }
28                }
29            }
30        }
31        if(primeiro == 1)
32            break;
33
34        pai[dest] = orig;
35    }
36  }
```

PRIM

- A eficiência do algoritmo de PRIM depende da forma usada para encontrar a aresta de menor peso. Usando uma fila de prioridades para achar a aresta, o custo computacional pode ser reduzido.

REFERÊNCIAS

- PEREIRA, Silvio do Lago. Estrutura de Dados Fundamentais: Conceitos e Aplicações, 12. Ed. São Paulo, Érica, 2008.
- BACKES, André Ricardo, Estrutura de dados descomplicada: em linguagem C, 1 Ed. – Rio de Janeiro: Elsevier, 2016.
- ROCHA, Anderson, Grafos – Representações, buscas e Aplicações.
- SENGER, H., Notas de Aula, Universidade de São Judas Tadeu, 1999.
- WALDEMAR Celes, Renato Cerqueira, José Lucas Rangel, Introdução a Estruturas de Dados, Editora Campus (2004).
- VELOSO, Paulo. SANTOS, Celso dos. AZEVEDO, Paulo. FURTADO, Antonio. Estrutura de dados. Rio de Janeiro: Ed. Elsevier, 1983 27ª reimpressão.
- https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
- ZIVIANI, Nivio. Projeto de Algoritmos com implementações em Java e C++, 2007.