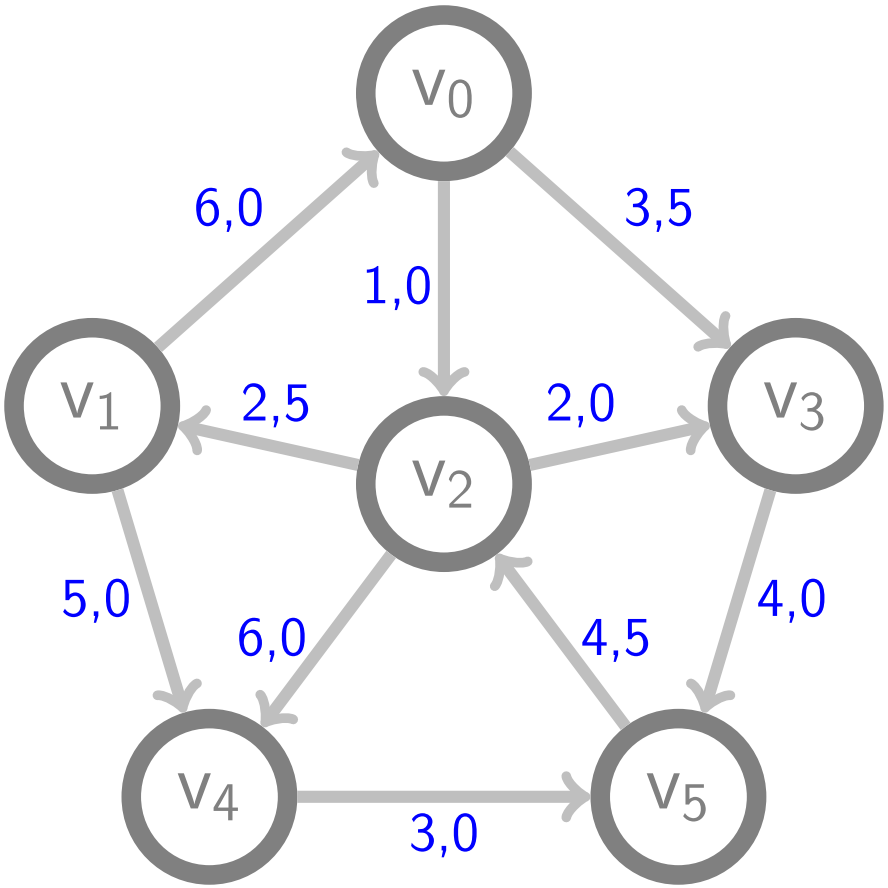
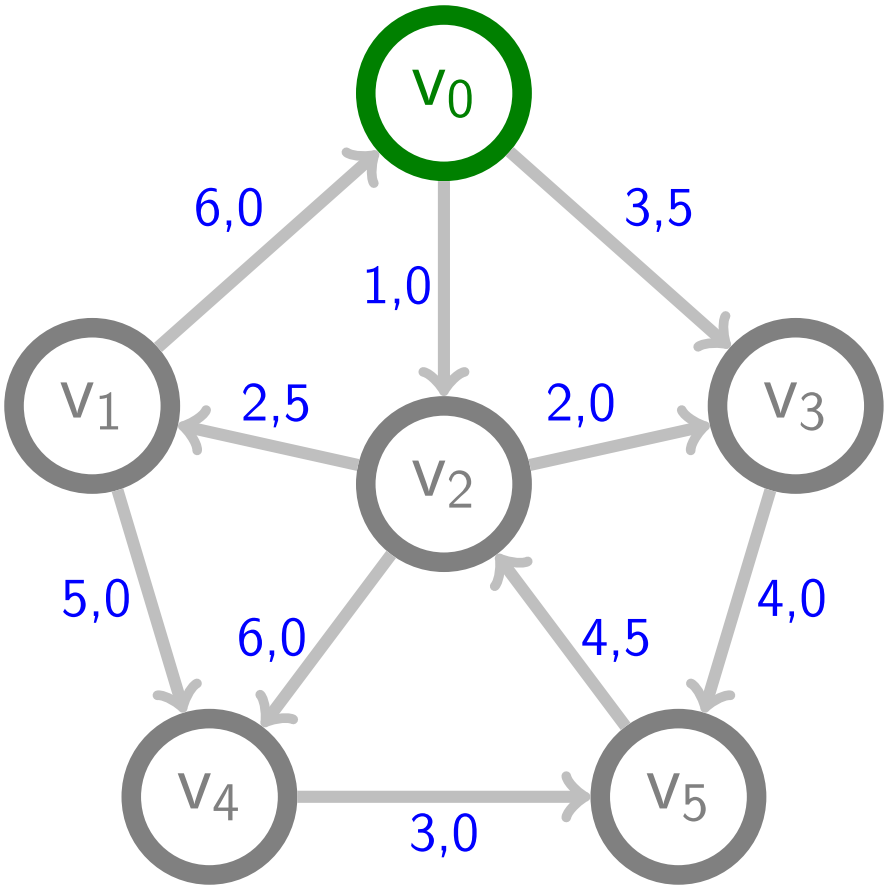
Algoritmos e Estruturas de Dados II Aula 14 - Caminhos de Custo Mínimo - Dijkstra

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

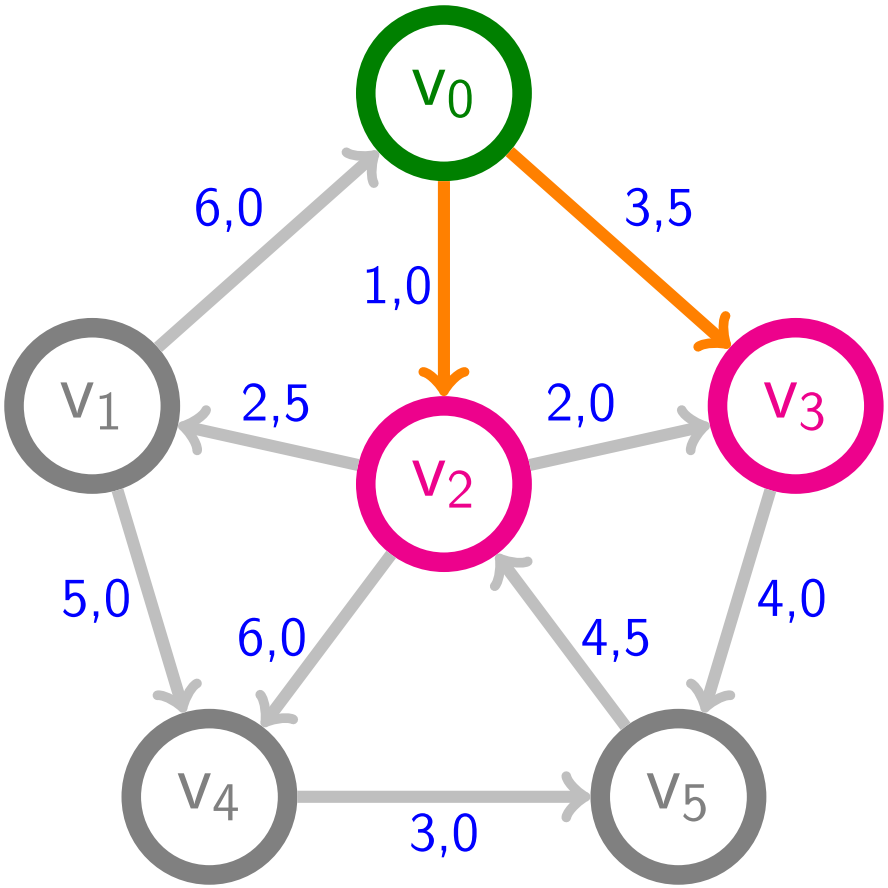
Árvore de Caminhos de Peso Mínimo



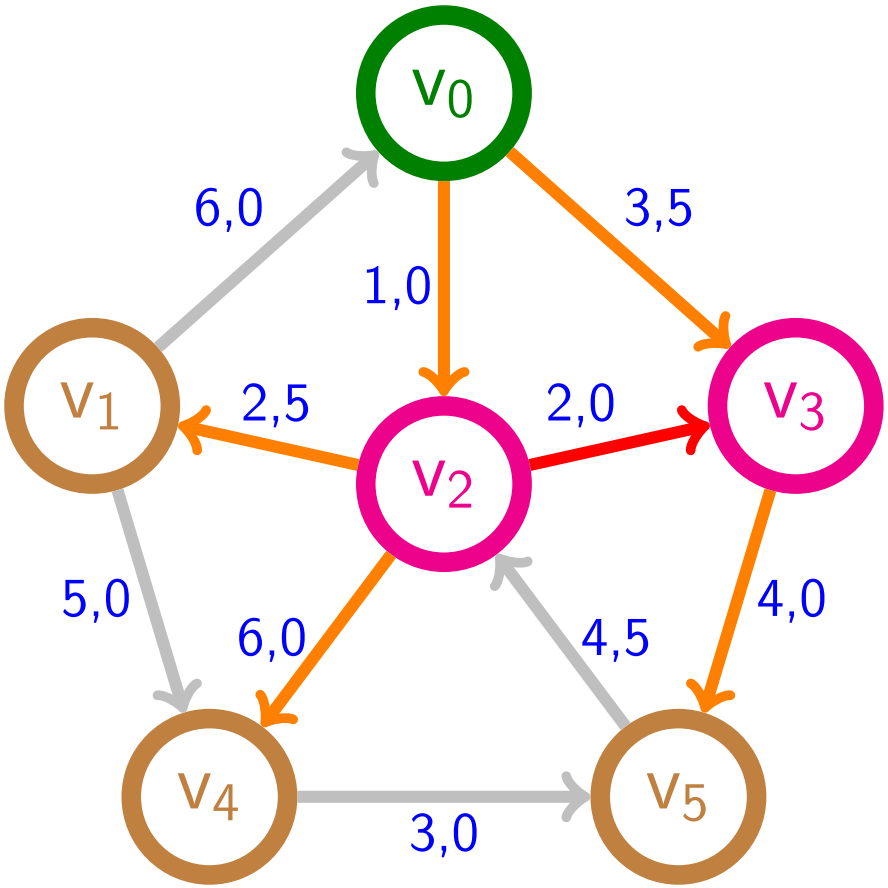
Árvore de Caminhos de Peso Mínimo



Árvore de Caminhos de Peso Mínimo

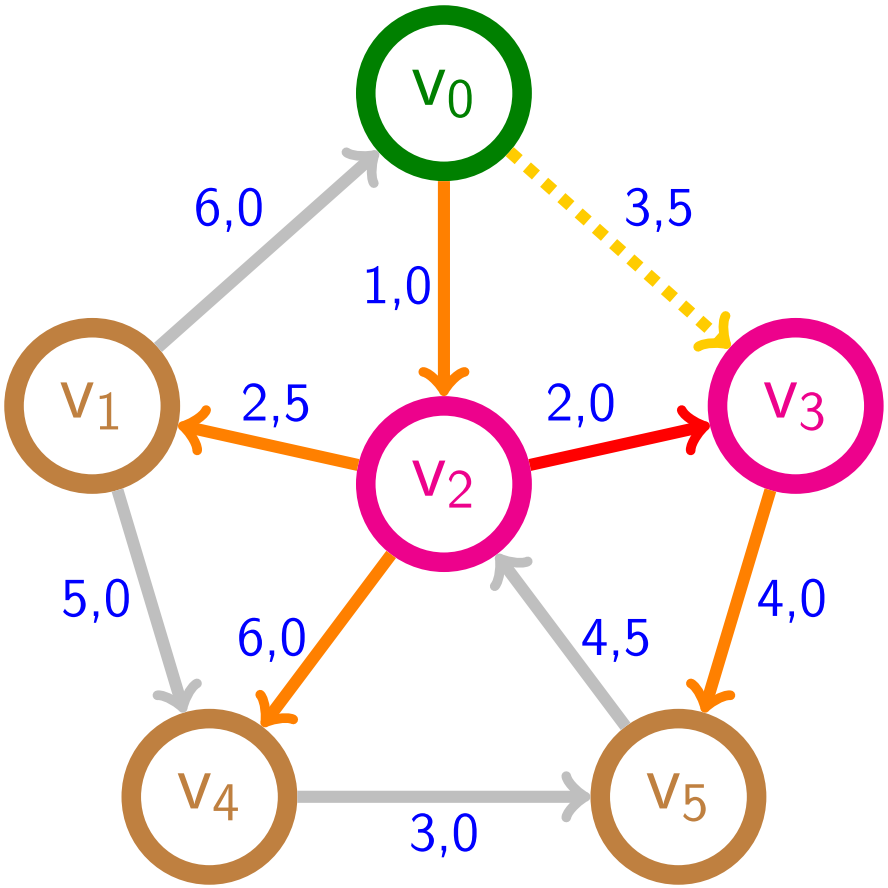


Árvore de Caminhos de Peso Mínimo



Existe **um caminho de menor custo** ('atalho') de v0 para v3, passando por v2.

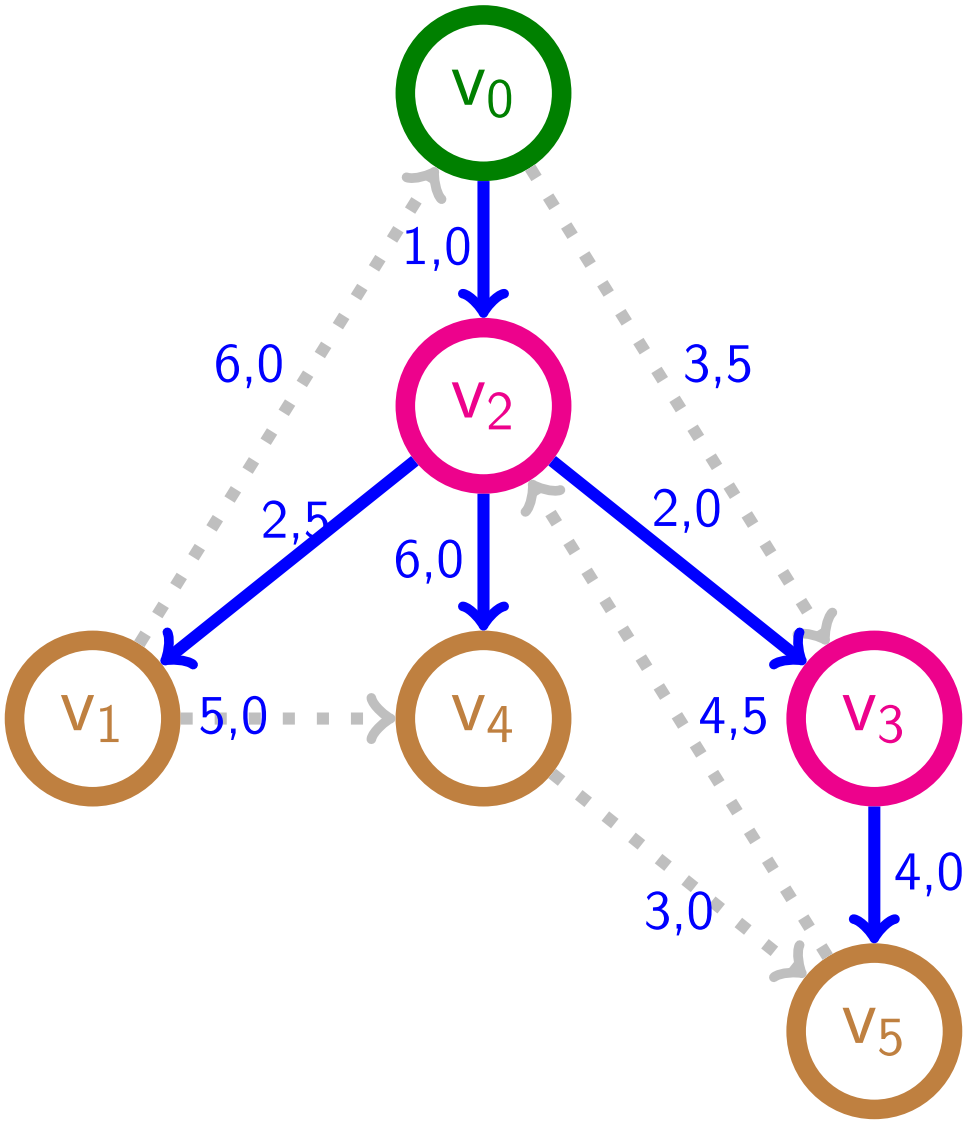
Árvore de Caminhos de Peso Mínimo



Existe **um caminho de menor custo** ('atalho') de v0 para v3, passando por v2.

A operação de atualizar a distância entre dois vértices, passando por outro(s) vértice(s) é chamada de **relaxamento**.

Árvore de Caminhos de Peso Mínimo



Caminhos de Peso Mínimo

Mas como encontramos os caminhos de custo (ou peso) mínimo?

- Existem diversas abordagens, aprenderemos três que usam o relaxamento.
  - **Algoritmo de Floyd-Warshall:** usado para encontrar os caminhos de peso mínimo entre todos os pares de vértices.
  - **Bellman-Ford:** usado para encontrar os caminhos de peso mínimo entre um vértice de origem e os demais.
  - **Dijkstra:** usado para encontrar os caminhos de peso mínimo entre um vértice de origem e os demais.

Algoritmo de Dijkstra

Concebido em 1956 e publicado em 1959, este algoritmo tem as seguintes características:

[i]Prioridade alcançada utilizando fila de prioridade com Heap de Fibonacci. A complexidade original é de $O(V^2)$.

Algoritmo de Dijkstra

Concebido em 1956 e publicado em 1959, este algoritmo tem as seguintes características:

- **Objetivo:** Encontrar os caminhos de custo mínimo entre um vértice de origem e os demais vértices;

[i] Prioridade alcançada utilizando fila de prioridade com Heap de Fibonacci. A complexidade original é de $O(V^2)$.

Algoritmo de Dijkstra

Concebido em 1956 e publicado em 1959, este algoritmo tem as seguintes características:

- **Objetivo:** Encontrar os caminhos de custo mínimo entre um vértice de origem e os demais vértices;
- **Aplicação:** Grafos direcionados e ponderados (arestas com pesos não negativos);

[i] Prioridade alcançada utilizando fila de prioridade com Heap de Fibonacci. A complexidade original é de $O(V^2)$.

Algoritmo de Dijkstra

Concebido em 1956 e publicado em 1959, este algoritmo tem as seguintes características:

- **Objetivo:** Encontrar os caminhos de custo mínimo entre um vértice de origem e os demais vértices;
- **Aplicação:** Grafos direcionados e ponderados (arestas com pesos não negativos);
- **Complexidade de tempo:** $O(E + V * log(V))$[i];

[i] Prioridade alcançada utilizando fila de prioridade com Heap de Fibonacci. A complexidade original é de $O(V^2)$.

Algoritmo de Dijkstra

Concebido em 1956 e publicado em 1959, este algoritmo tem as seguintes características:

- **Objetivo:** Encontrar os caminhos de custo mínimo entre um vértice de origem e os demais vértices;
- **Aplicação:** Grafos direcionados e ponderados (arestas com pesos não negativos);
- **Complexidade de tempo:** $O(E + V * log(V))$[i];
- Abordagem Gulosa e que usa Programação Dinâmica.

[i] Prioridade alcançada utilizando fila de prioridade com Heap de Fibonacci. A complexidade original é de $O(V^2)$.

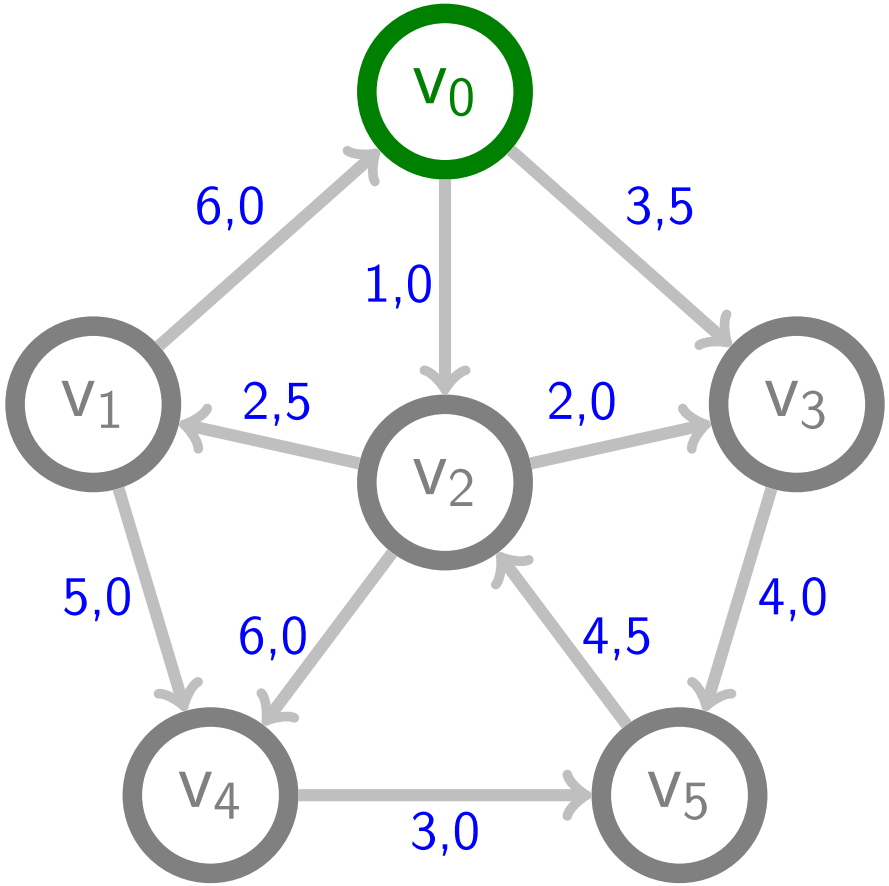
Algoritmo de Dijkstra

Pseudocódigo:

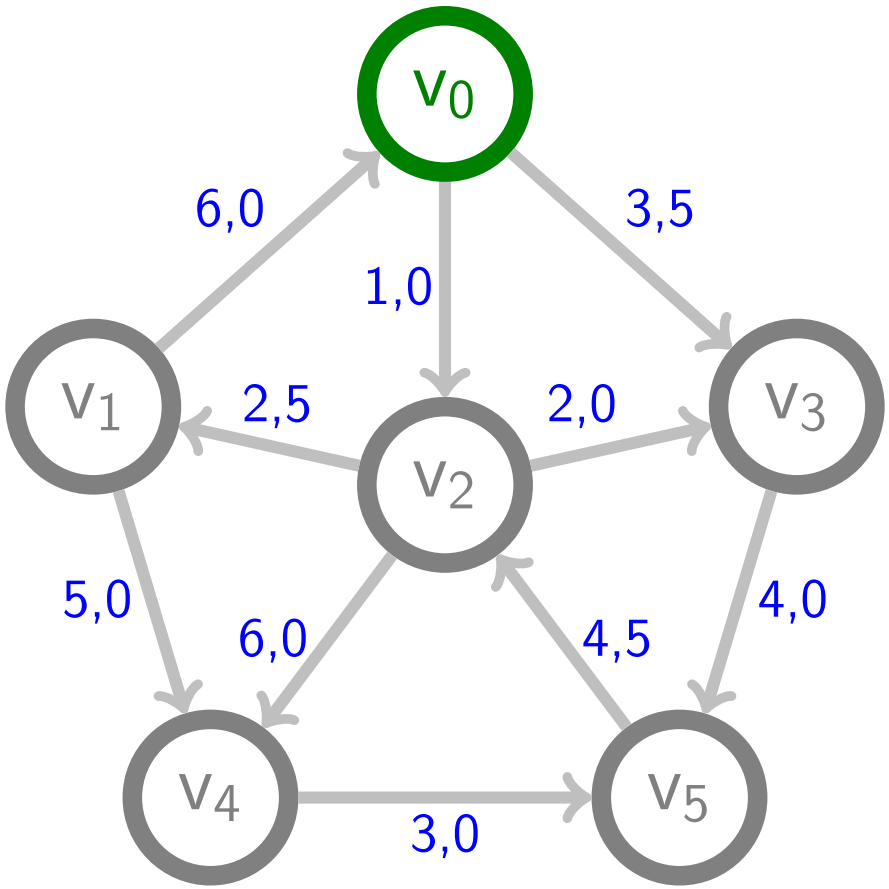
```
ENQUANTO houver vértices não processados:
    SELECIONE u: vértice não processado com menor
                                        distância à origem:
    MARQUE u como processado
    PARA CADA v, adjacente de u, ainda não processado
        SE dist[v] > dist[u] + peso(u-v) ENTÃO
            dist[v] = dist[u] + peso(u-v)
            pred[v] = u
```

Sendo *dist* o arranjo de distância de cada vértice ao vértice de origem, iniciado com zero para o vértice de origem e 'infinito' para os demais, *pred* o arranjo de predecessores e peso(u-v) o peso da aresta de u para v.

Algoritmo de Dijkstra



Algoritmo de Dijkstra



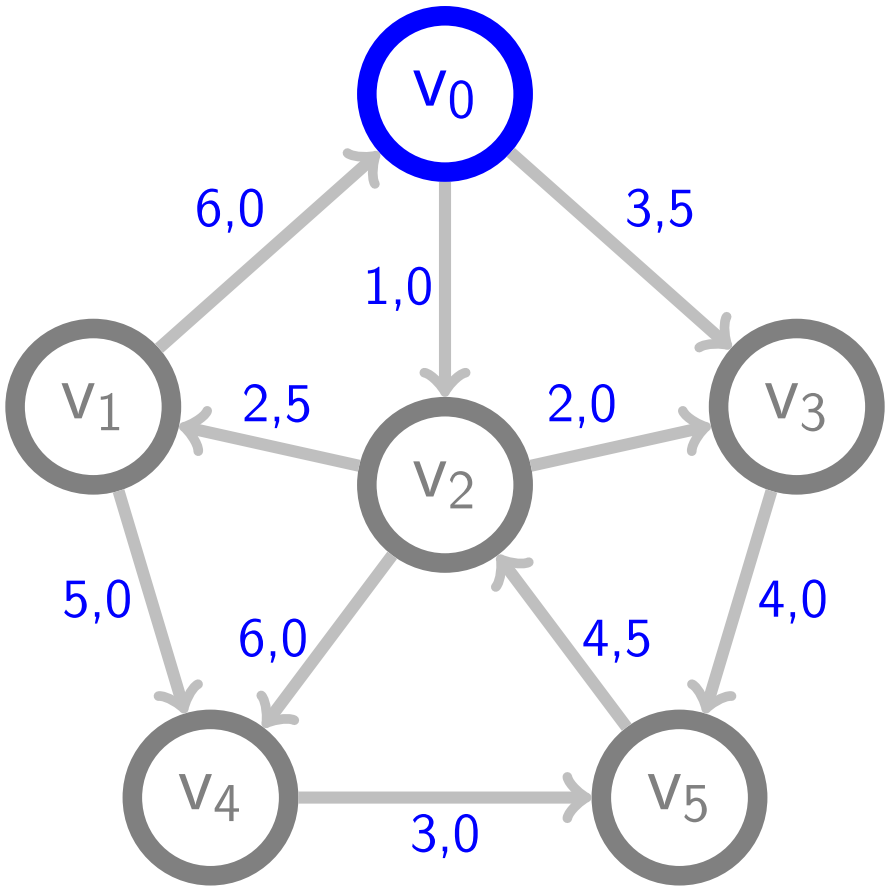
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5

Algoritmo de Dijkstra



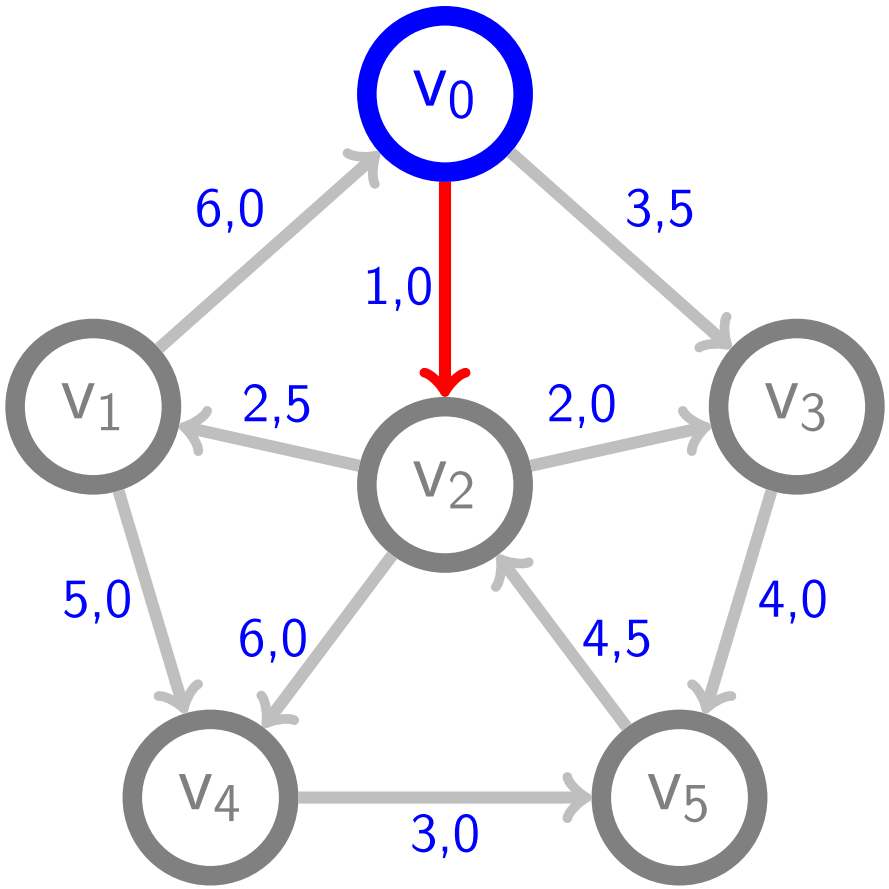
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	-	-	-	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	-1	-1	-1	-1	-1

Algoritmo de Dijkstra



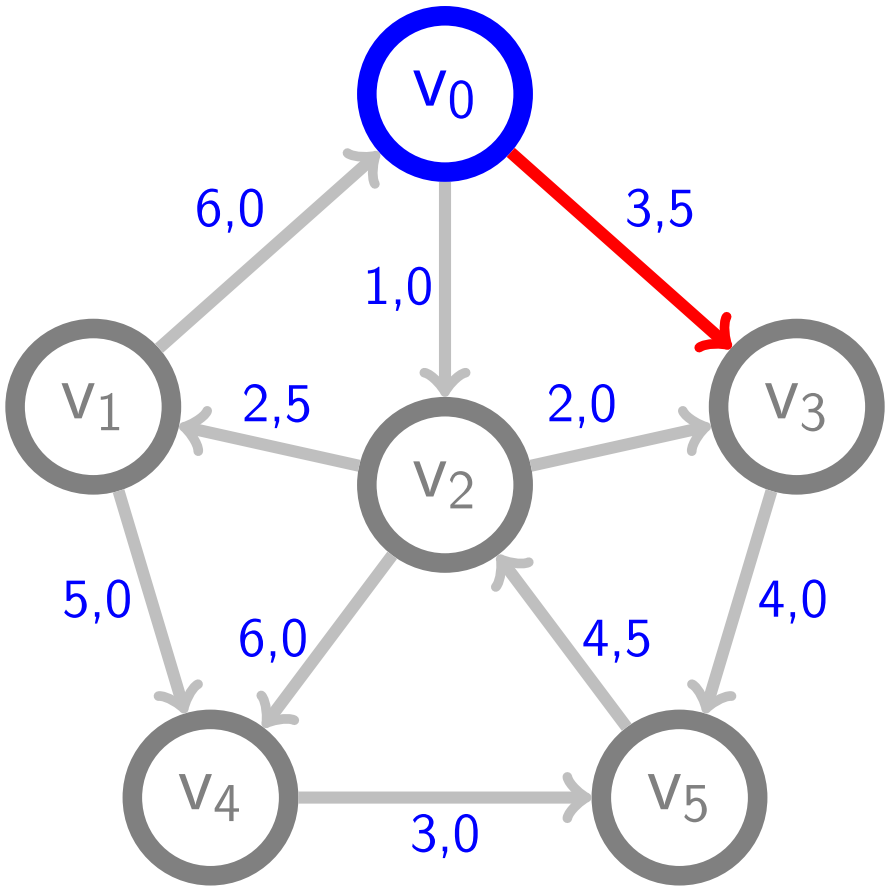
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	-	1,0	-	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	-1	0	-1	-1	-1

Algoritmo de Dijkstra



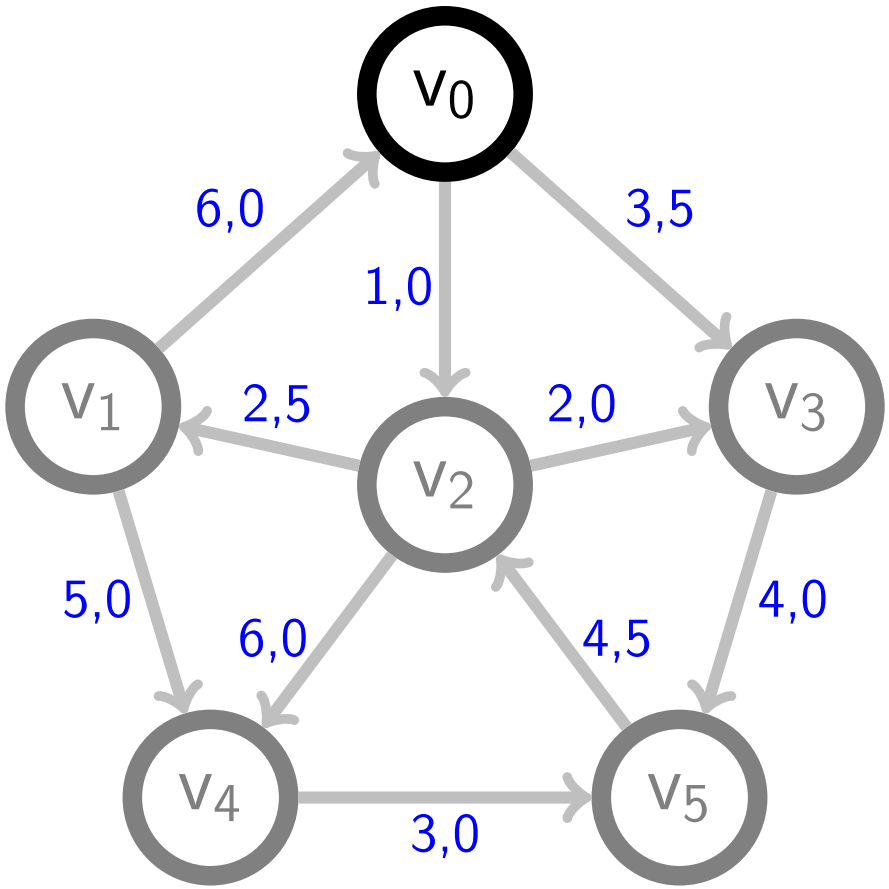
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	-	1,0	3,5	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	-1	0	0	-1	-1

Algoritmo de Dijkstra



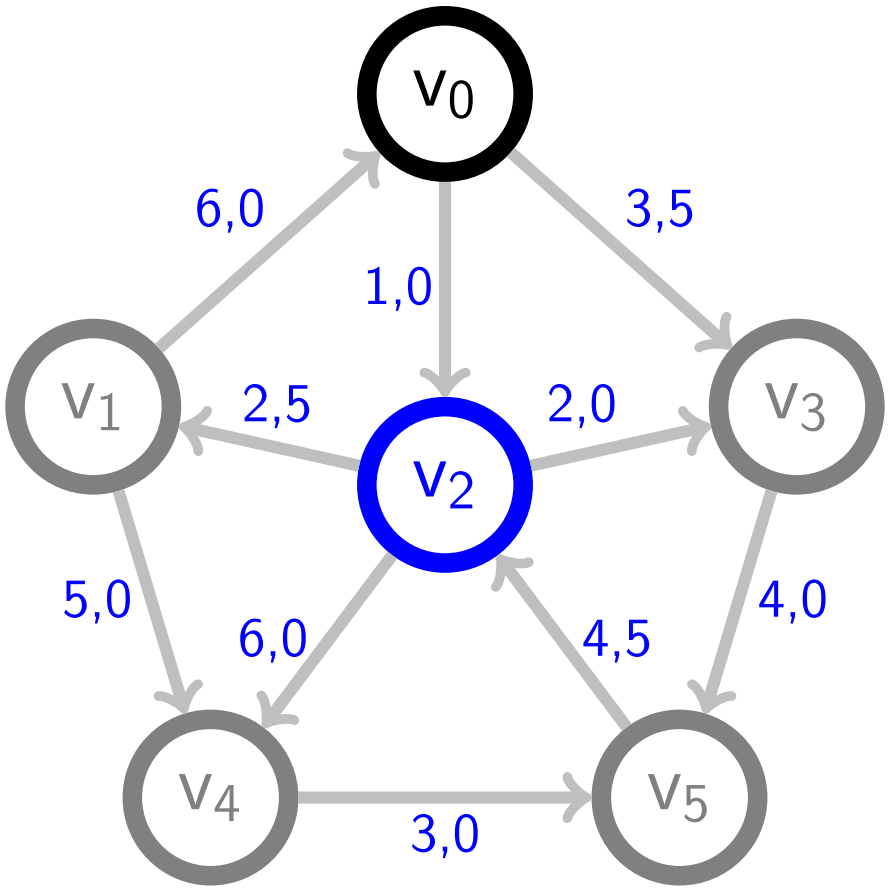
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	-	1,0	3,5	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	-1	0	0	-1	-1

Algoritmo de Dijkstra



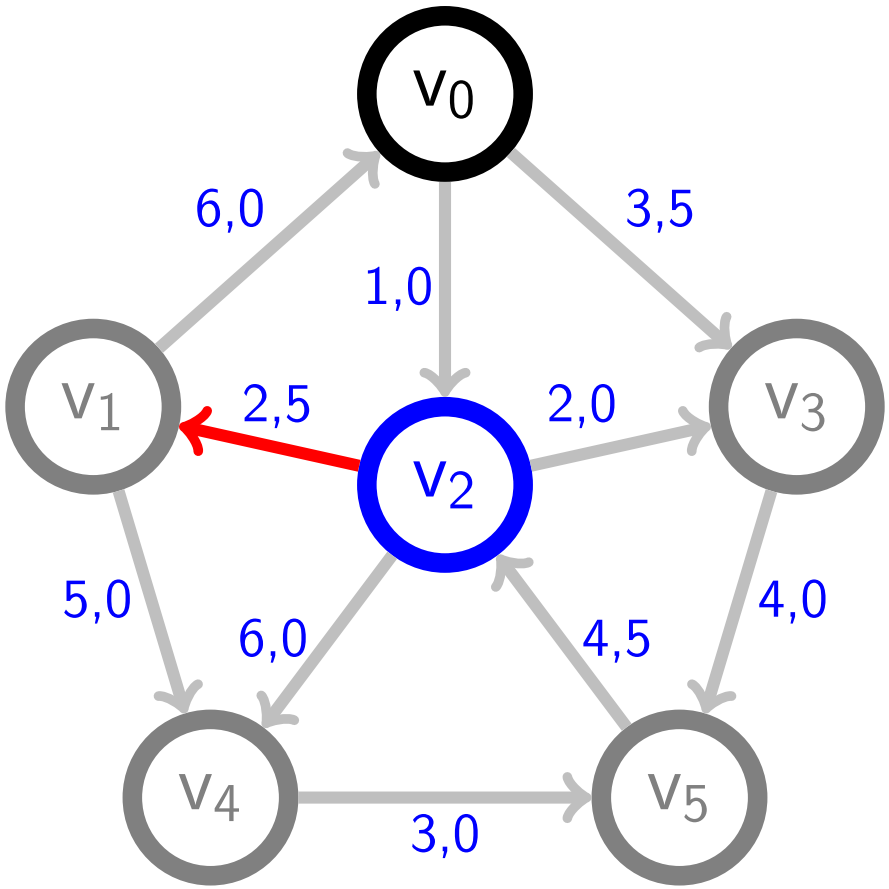
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	-	1,0	3,5	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	-1	0	0	-1	-1

Algoritmo de Dijkstra



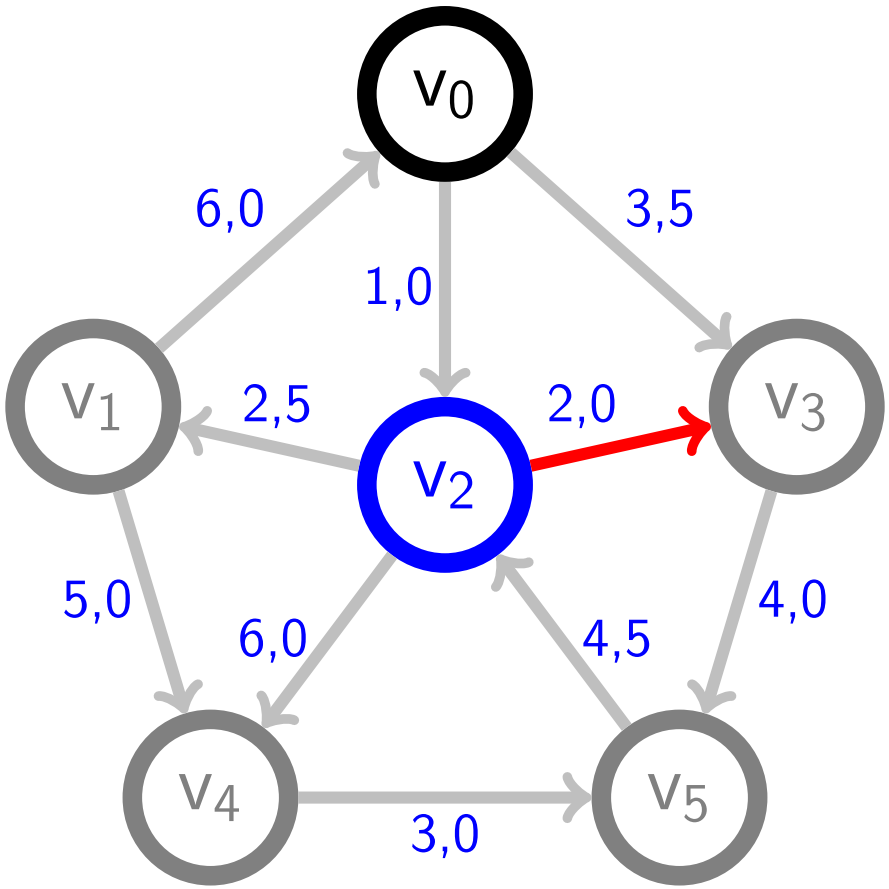
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,5	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	0	-1	-1

Algoritmo de Dijkstra



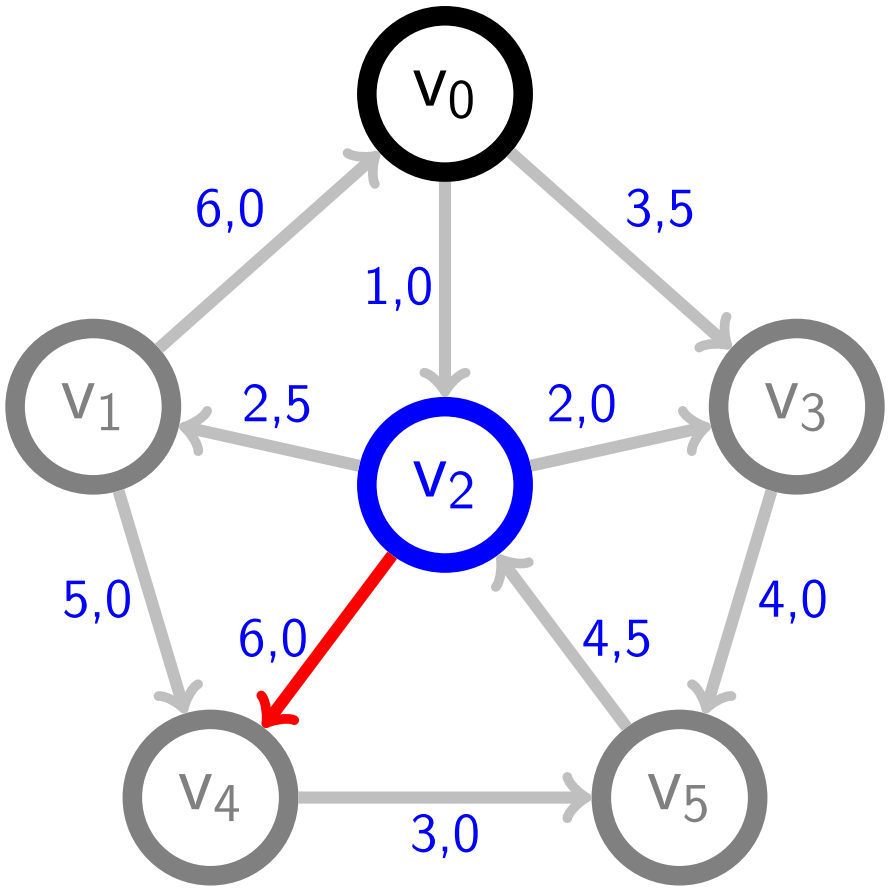
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	-	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	-1	-1

Algoritmo de Dijkstra



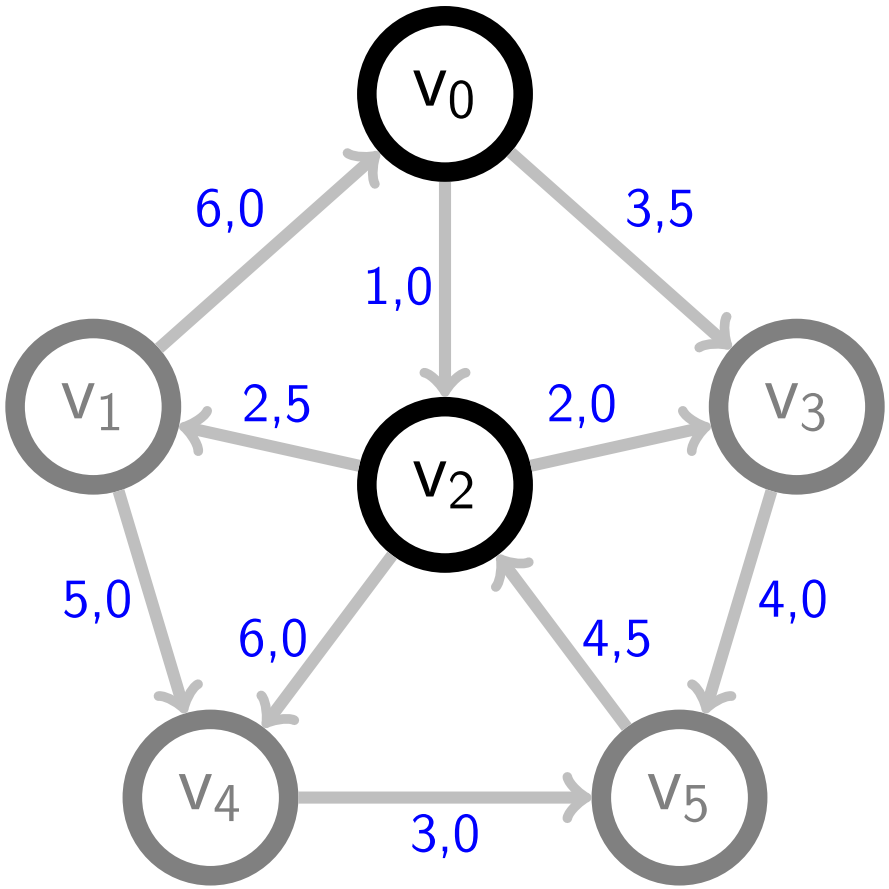
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	-1

Algoritmo de Dijkstra



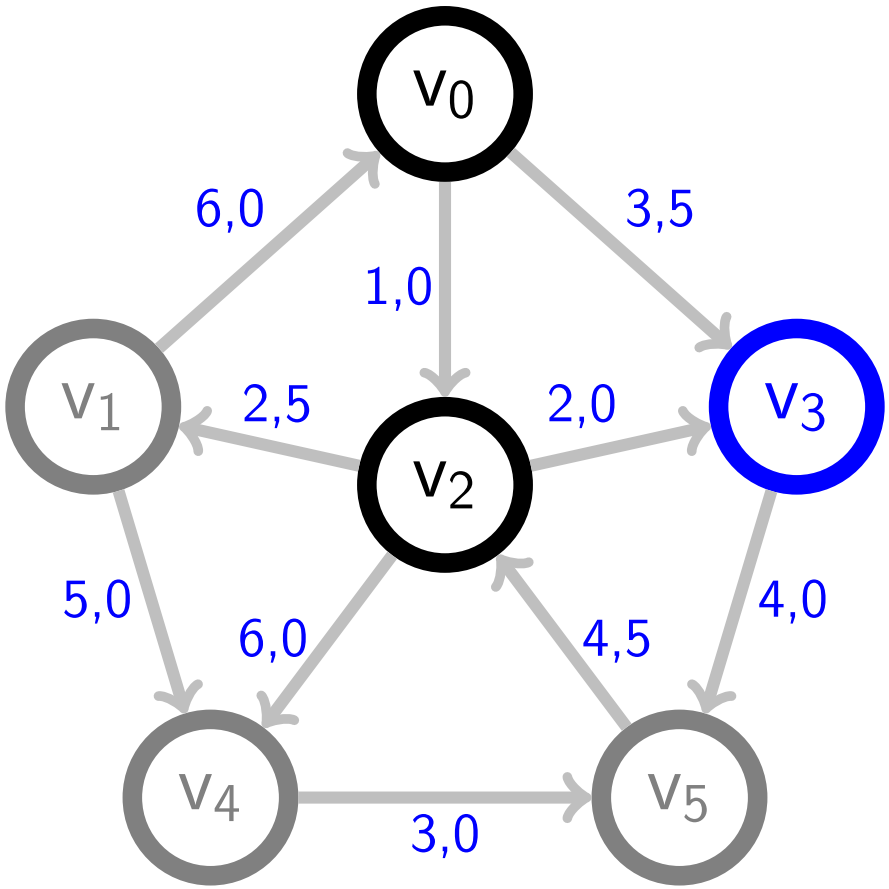
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	-1

Algoritmo de Dijkstra



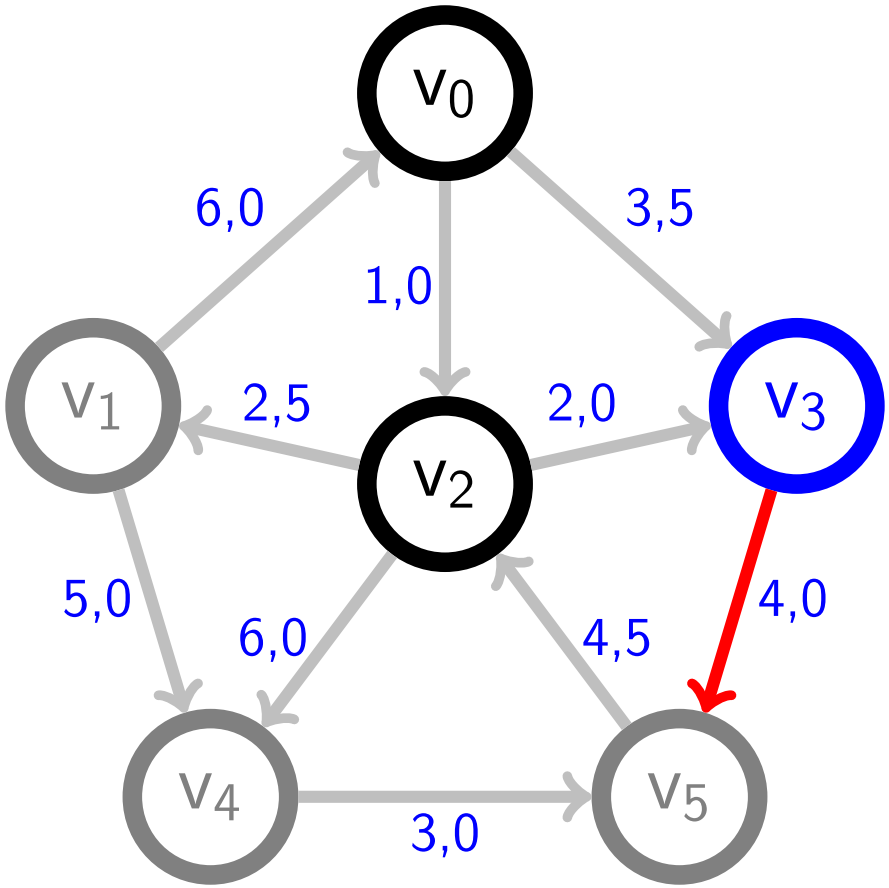
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	-

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	-1

Algoritmo de Dijkstra



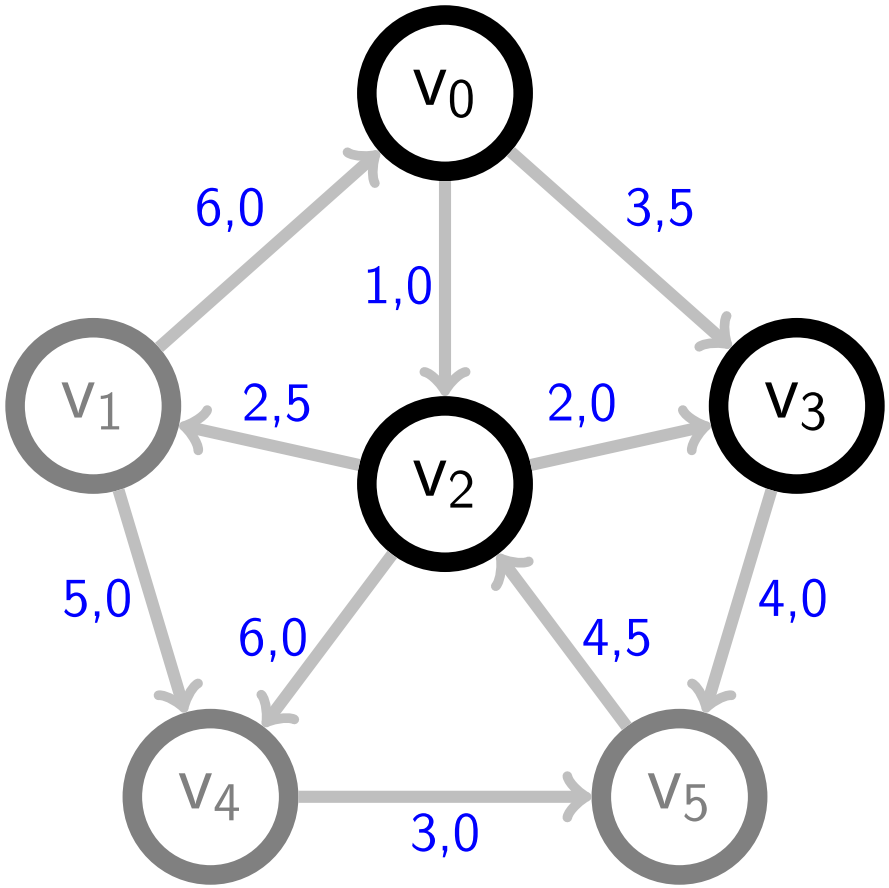
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



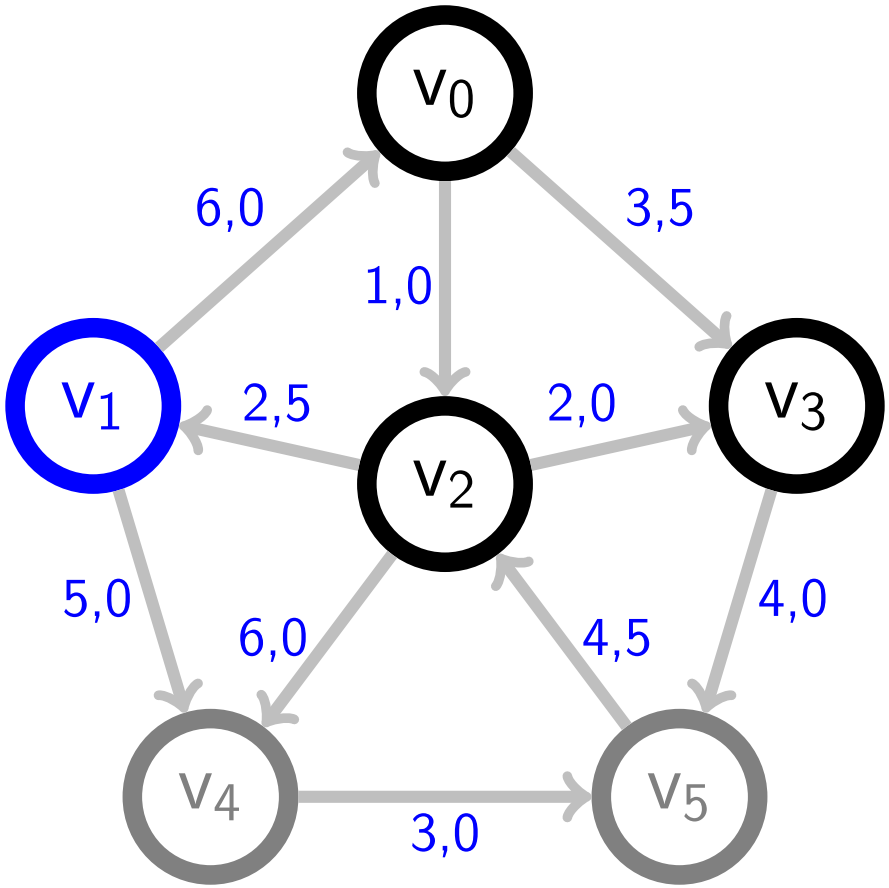
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



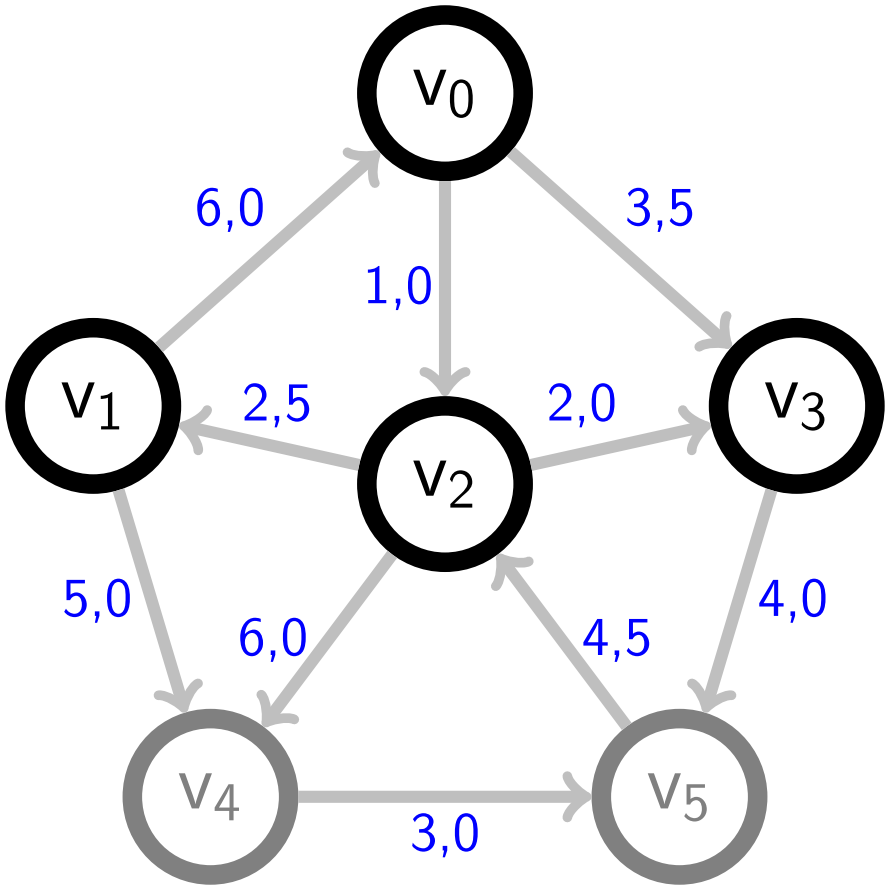
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



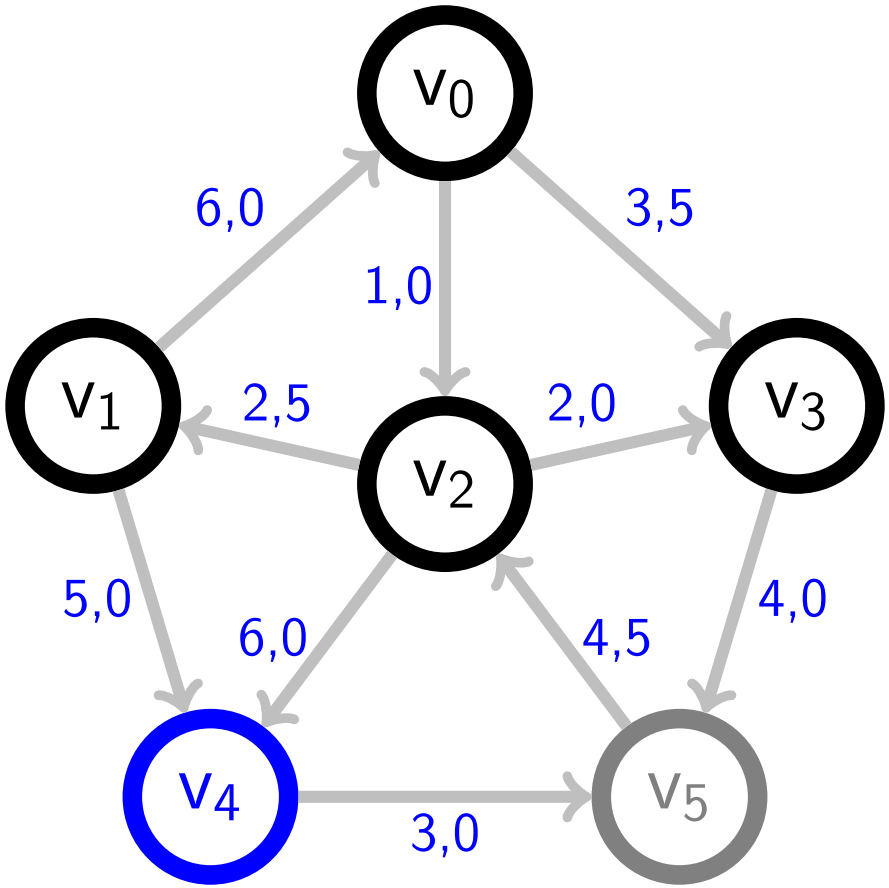
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



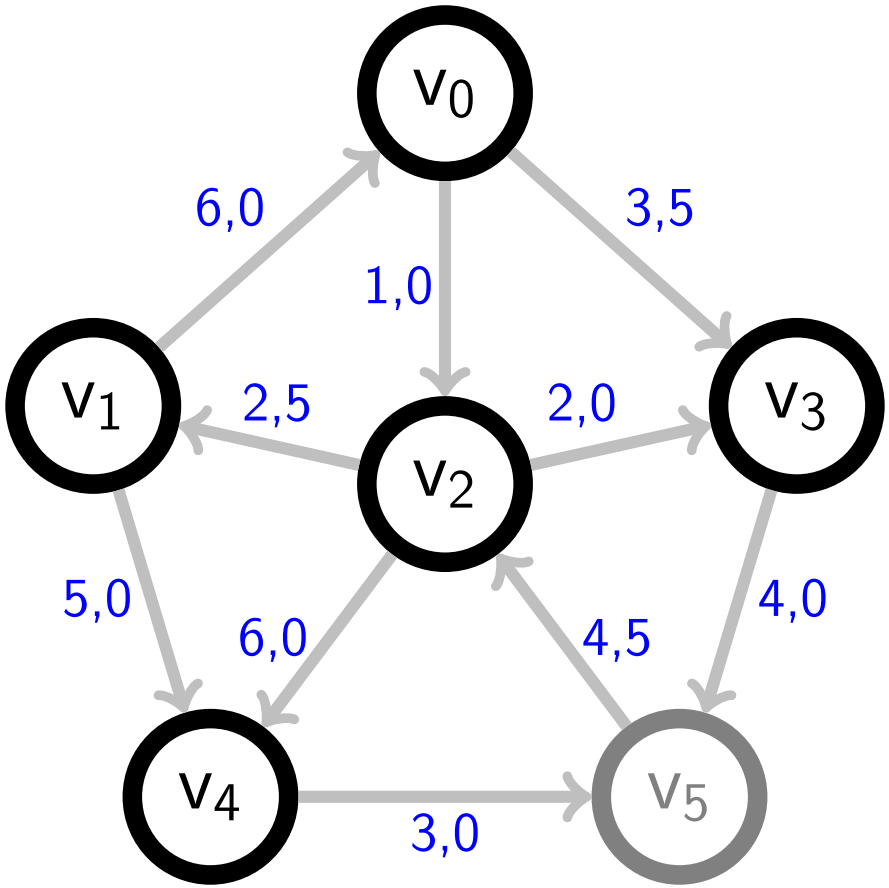
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



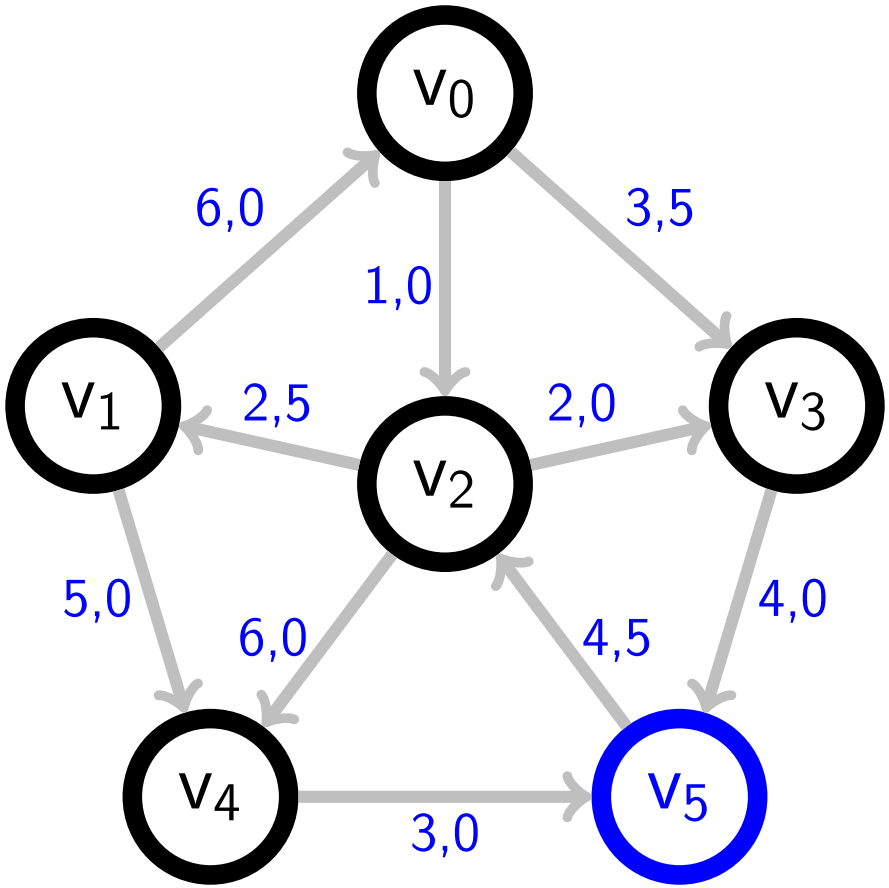
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



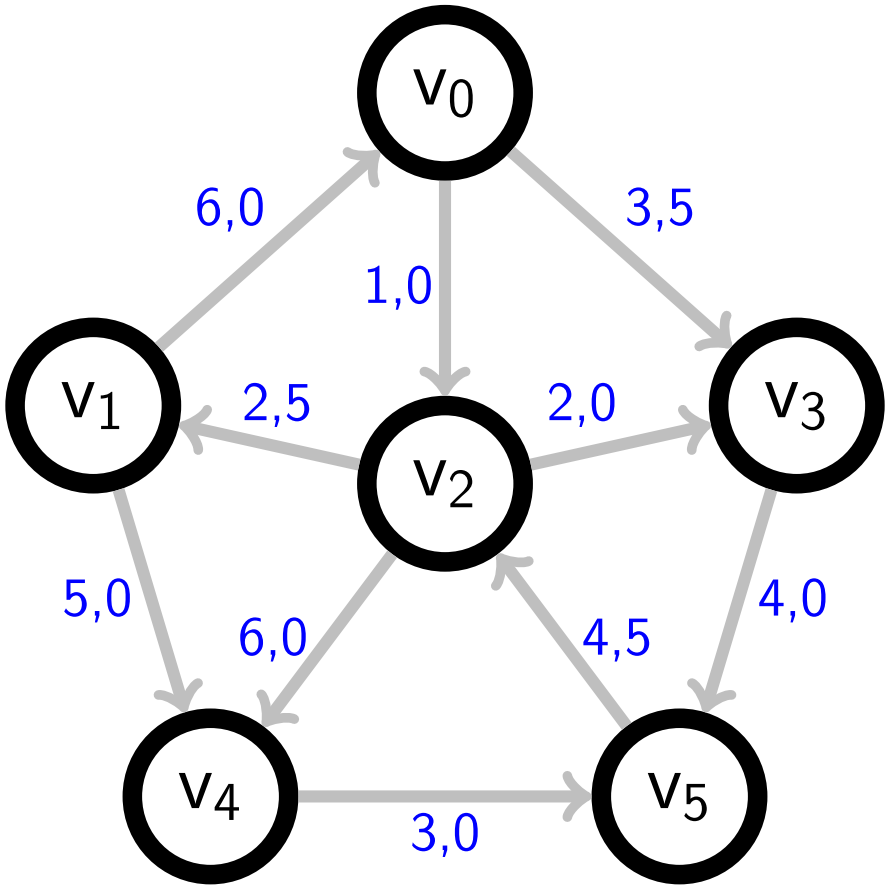
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



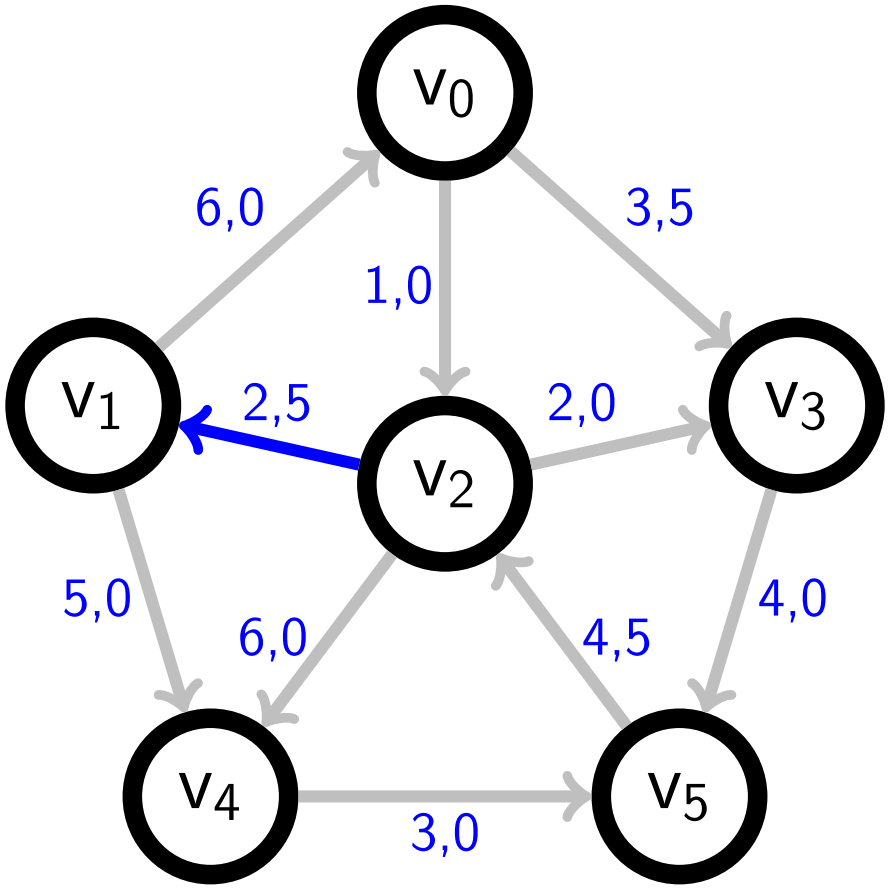
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



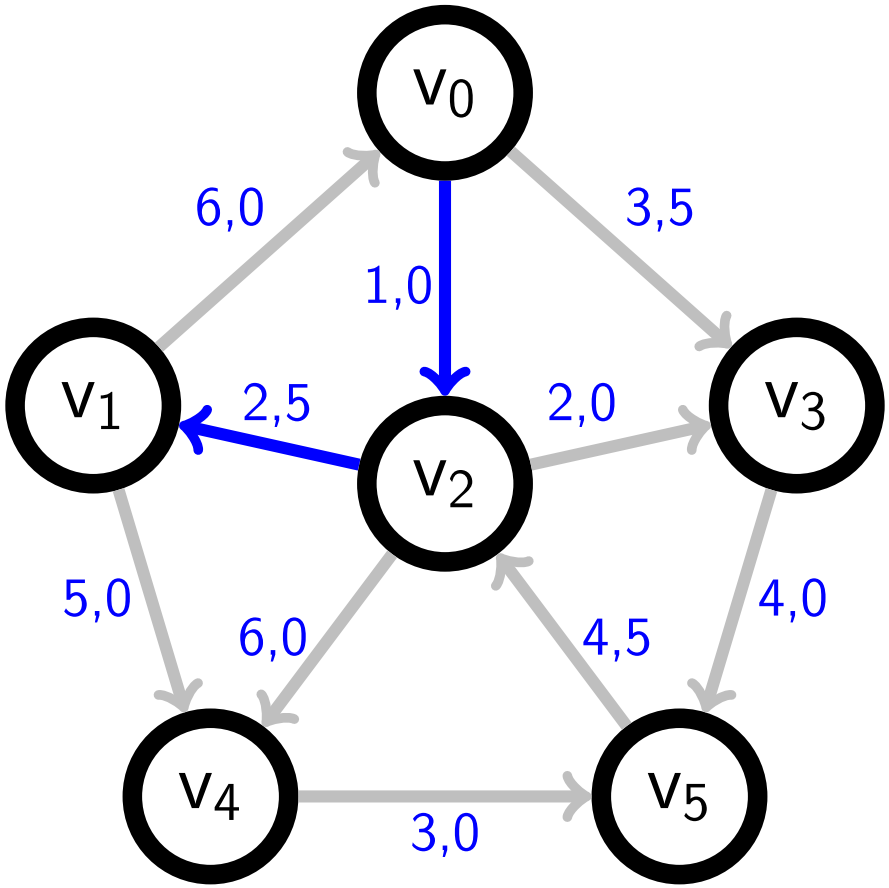
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



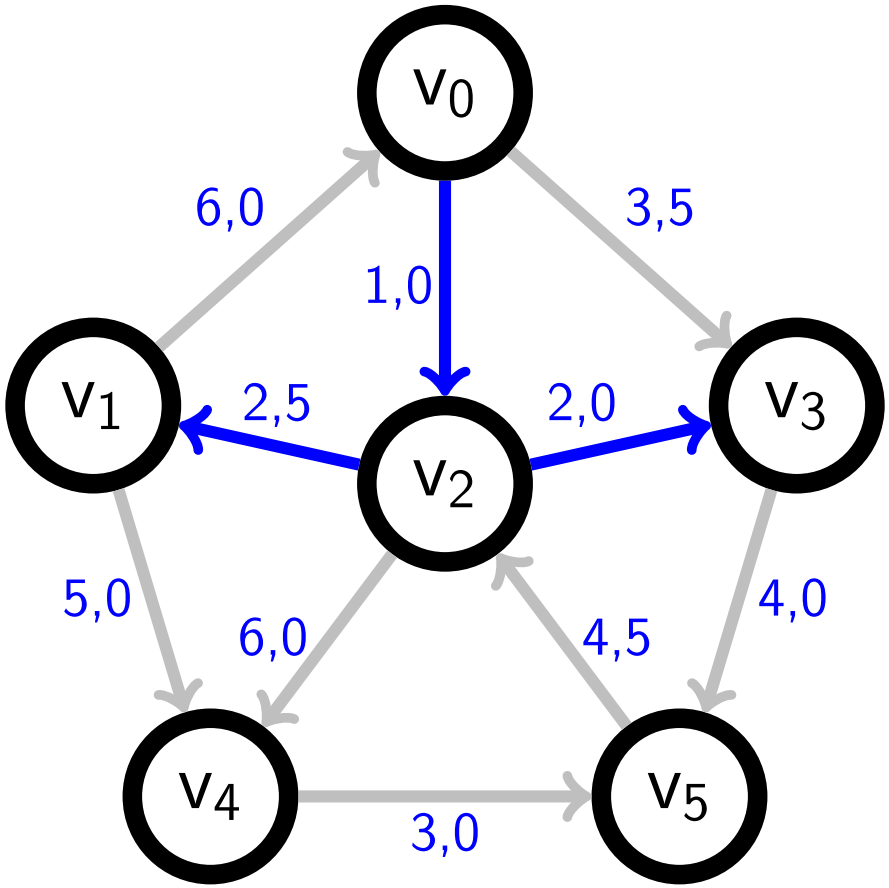
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



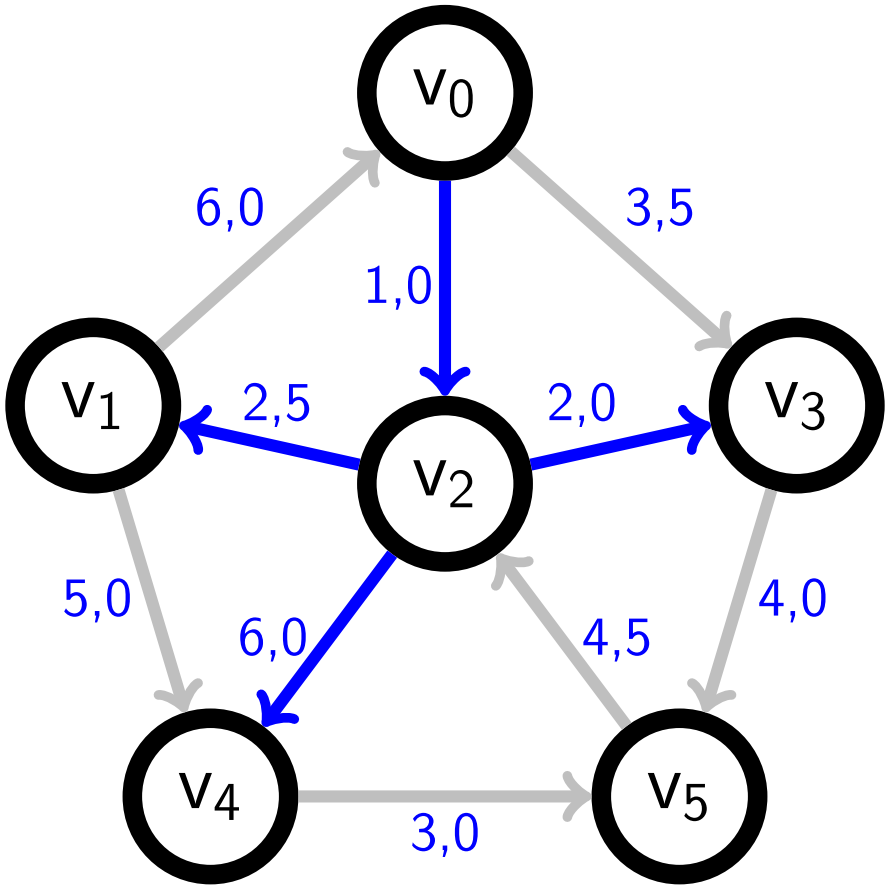
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



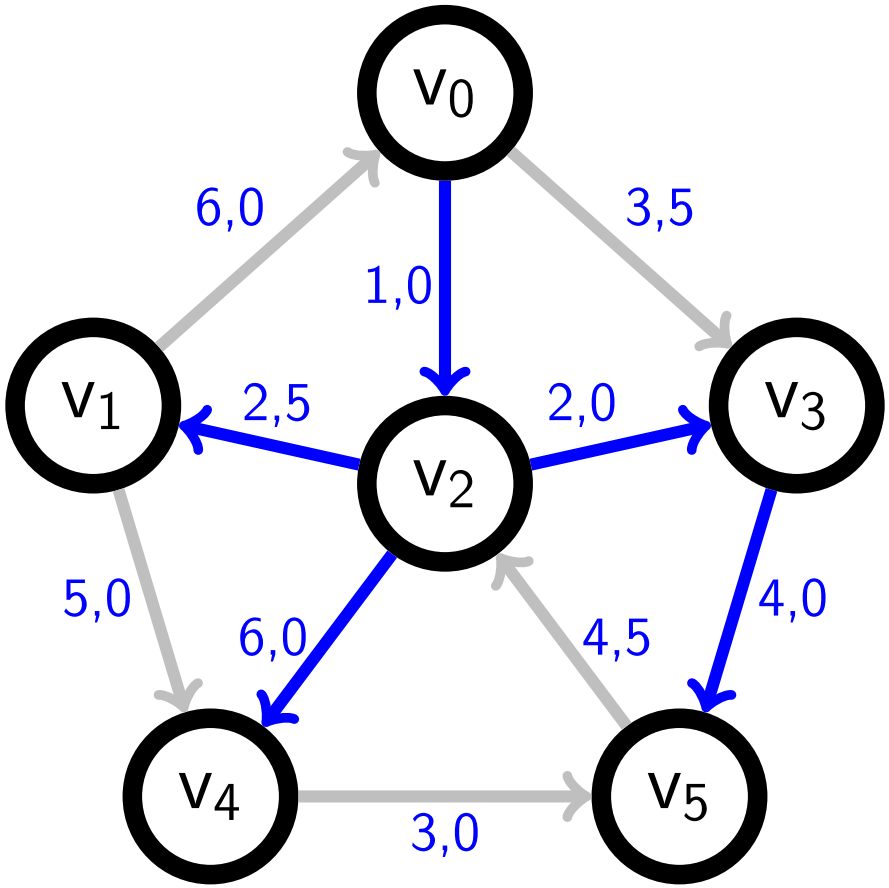
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



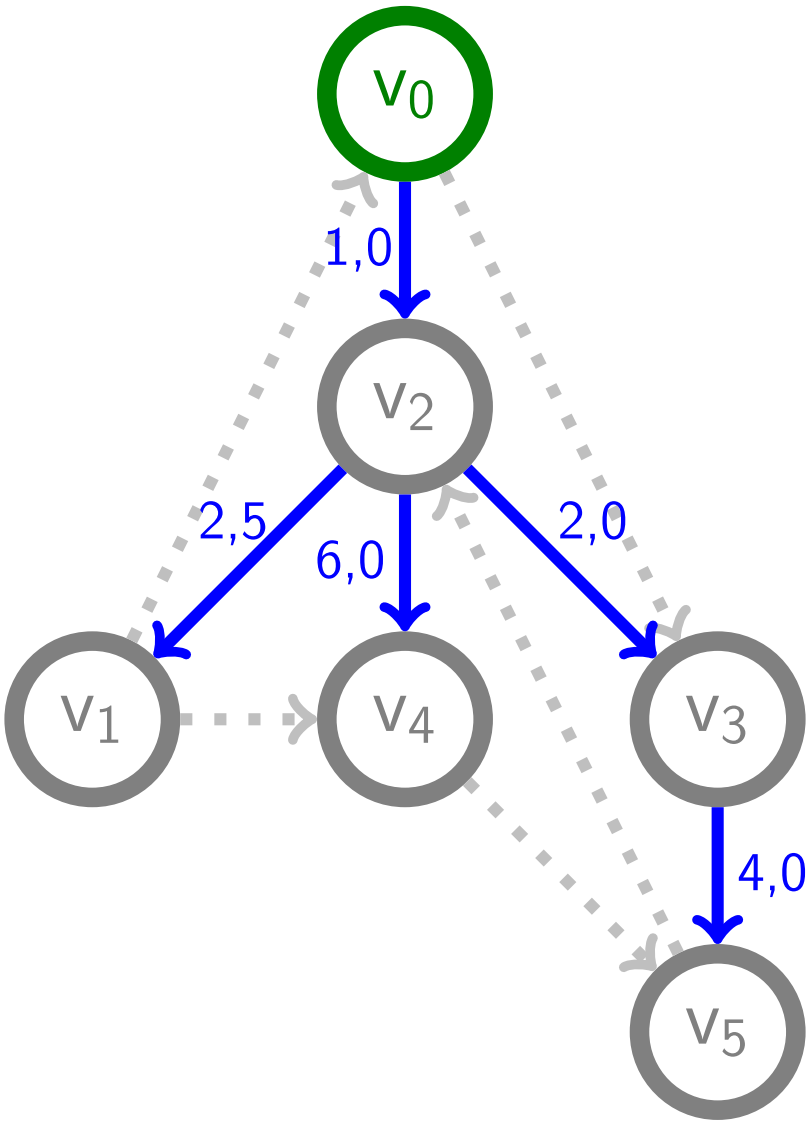
Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Algoritmo de Dijkstra



Arranjo de Distâncias

v0	v1	v2	v3	v4	v5
0,0	3,5	1,0	3,0	7,0	7,0

Arranjo de Predecessores

v0	v1	v2	v3	v4	v5
0	2	0	2	2	3

Caminhos de Custo Mínimo - Dijkstra

```
void AlgoritmoDeDijkstra(Grafo* g, int origem,
                                   Peso* dist, int* pred){
```

...

Caminhos de Custo Mínimo - Dijkstra

```
void AlgoritmoDeDijkstra(Grafo* g, int origem,
                                   Peso* dist, int* pred){
  int i, u, v, n;
  n = g->numVertices;
  for(i=0;i<n;i++) {
    dist[i] = INFINITO;
    pred[i] = -1;
  }
```

...

Caminhos de Custo Mínimo - Dijkstra

```
void AlgoritmoDeDijkstra(Grafo* g, int origem,
                                   Peso* dist, int* pred){
  int i, u, v, n;
  n = g->numVertices;
  for(i=0;i<n;i++) {
    dist[i] = INFINITO;
    pred[i] = -1;
  }
  dist[origem] = 0;
  pred[origem] = origem;
  PONT vizinhoAtual;

  ...
```

Caminhos de Custo Mínimo - Dijkstra

```
void AlgoritmoDeDijkstra(Grafo* g, int origem,
                                   Peso* dist, int* pred){
  int i, u, v, n;
  n = g->numVertices;
  for(i=0;i<n;i++) {
    dist[i] = INFINITO;
    pred[i] = -1;
  }
  dist[origem] = 0;
  pred[origem] = origem;
  PONT vizinhoAtual;

  FilaDePrioridade fp;
  inicializaFilaDePrioridadeComPesos(&fp, n, INFINITO);
  diminuiPrioridade(&fp, origem, 0.0);
  ...
```

Caminhos de Custo Mínimo - Dijkstra

```
    ...
    while(! filaDePrioridadeEstaVazia(&fp) ){









    }

}
```

Caminhos de Custo Mínimo - Dijkstra

```
...
while(! filaDePrioridadeEstaVazia(&fp) ){
  u = excluiElemento(&fp) ;
  vizinhoAtual = listaDeVizinhos(g, u) ;









  }

}
```

Caminhos de Custo Mínimo - Dijkstra

```
...
while(! filaDePrioridadeEstaVazia(&fp) ){
  u = excluiElemento(&fp) ;
  vizinhoAtual = listaDeVizinhos(g, u) ;
  while (vizinhoAtual){
    v = vizinhoAtual->vertice;




    }
    vizinhoAtual = vizinhoAtual->prox;
  }
}

}
```

Caminhos de Custo Mínimo - Dijkstra

```
...
while(! filaDePrioridadeEstaVazia(&fp) ){
  u = excluiElemento(&fp) ;
  vizinhoAtual = listaDeVizinhos(g, u) ;
  while (vizinhoAtual){
    v = vizinhoAtual->vertice;
    if (dist[v] > dist[u] + vizinhoAtual->peso){
       dist[v] = dist[u] + vizinhoAtual->peso;
       pred[v] = u;
       diminuiPrioridade(&fp, v, dist[v]) ;
    }
    vizinhoAtual = vizinhoAtual->prox;
  }
}

}
```

Caminhos de Custo Mínimo - Dijkstra

```
...
while(! filaDePrioridadeEstaVazia(&fp) ){
  u = excluiElemento(&fp) ;
  vizinhoAtual = listaDeVizinhos(g, u) ;
  while (vizinhoAtual){
    v = vizinhoAtual->vertice;
    if (dist[v] > dist[u] + vizinhoAtual->peso){
       dist[v] = dist[u] + vizinhoAtual->peso;
       pred[v] = u;
       diminuiPrioridade(&fp, v, dist[v]) ;
    }
    vizinhoAtual = vizinhoAtual->prox;
  }
}
liberaFilaDePrioridade(&fp) ;
}
```

Caminhos de Custo Mínimo - Dijkstra

```
...
while(! filaDePrioridadeEstaVazia(&fp) ){
  u = excluiElemento(&fp) ;
  vizinhoAtual = listaDeVizinhos(g, u) ;
  while (vizinhoAtual){
    v = vizinhoAtual->vertice;
    if (dist[v] > dist[u] + vizinhoAtual->peso){
       dist[v] = dist[u] + vizinhoAtual->peso;
       pred[v] = u;
       diminuiPrioridade(&fp, v, dist[v]) ;
    }
    vizinhoAtual = vizinhoAtual->prox;
  }
}
liberaFilaDePrioridade(&fp) ;
}
```

$$O((E + V) * log(V))$$

Caminhos de Custo Mínimo - Dijkstra

```
...
while(! filaDePrioridadeEstaVazia(&fp) ){
  u = excluiElemento(&fp) ;
  vizinhoAtual = listaDeVizinhos(g, u) ;
  while (vizinhoAtual){
    v = vizinhoAtual->vertice;
    if (dist[v] > dist[u] + vizinhoAtual->peso){
       dist[v] = dist[u] + vizinhoAtual->peso;
       pred[v] = u;
       diminuiPrioridade(&fp, v, dist[v]) ;
    }
    vizinhoAtual = vizinhoAtual->prox;
  }
}
liberaFilaDePrioridade(&fp) ;
}
```

$O((E + V) * log(V))$

com Heap de Fibonacci:

$O(E + V * log(V))$

Algoritmos e Estruturas de Dados II Aula 14 - Caminhos de Custo Mínimo - Dijkstra

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri