Caminho mais Curto

Caminho mais curto

1. De um nó fonte para todos os demais vértices

2. De todos os nós para um destino único

3. De par único

4. Dentre todos os pares de vértices

Caminho mais curto

1. De um nó fonte para todos os demais vértices

2. De todos os nós para um destino único

3. De par único

4. Dentre todos os pares de vértices

Algoritmos de Caminho Mais Curto

	BFS	Dijkstra	Bellman Ford	Floyd Warshall
Complexidade	O(V+E)	O((V+E)*logV)	O(VE)	$O(V^3)$
Tam Grafo Recomendado	Grande	Grande/Médio	Médio/Pequeno	Pequeno
CMC GRAFO	Melhor Algoritmo	OK	Ruim	Geralmente Ruim
CMC DIGRAFO	Não	Melhor Algoritmo	OK	Geralmente Ruim
Arestas de Peso Negativo	Não	Não	Sim	Sim
Distância de todos para todos	Só Grafos	OK	Ruim	Bom

Adaptado de: Competitive Programing 3, P. 161, Steven & Felix Halim

Programação Dinâmica

A programação dinâmica nos fornece uma maneira de projetar algoritmos que:

Sistematicamente exploram todas as possibilidades (corretude);

Armazenam resultados a fim de evitar computação redundante (eficiência);

Definem a solução para o problema em termos da solução de problemas menores.

Programação Dinâmica

É uma técnica para desenvolvermos algoritmos mais eficientes, armazenando resultados parciais.

Essa técnica evita a computação dos mesmos subproblemas repetidamente pois:

Começa a resolver o problema pelos subproblemas de menor tamanho (estratégia *bottom-up*);

A medida que são computados, armazena as respostas;

Quando o mesmo subproblema precisa ser resolvido novamente, apenas consulta o valor armazenado.

Programação Dinâmica

Aplicável a problemas que possuam as propriedades:

- Subestrutura Ótima

O problema pode ser dividido sucessivamente, e a combinação das soluções ótimas dos subproblemas corresponde à solução ótima do problema original.

- Superposição de Subproblemas

O espaço de subproblemas é pequeno, e eles se repetem durante a solução do problema original.

Programação Dinâmica

Vantagens:

Economiza computação em problemas que possuem superposição de subproblemas

Ganho em desempenho.

Desvantagens:

O número de soluções armazenadas na tabela pode crescer rapidamente caso o espaço de soluções não seja pequeno

Alocação de muita memória.

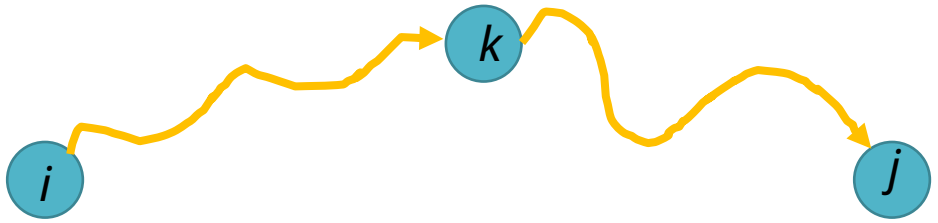
Algoritmo de Floyd-Warshall

Ao invés de considerar arestas, o algoritmo considera os vértices intermediários de um caminho mais curto.

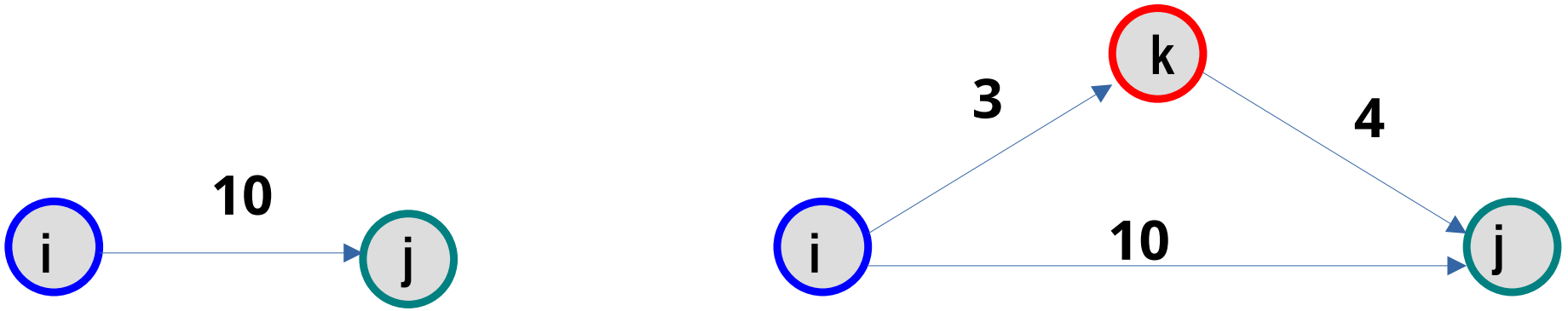
Dado um vértice *k* e um caminho mais curto entre *i* e *j*, podem ocorrer duas situações:

- *k* não pertence ao caminho mais curto entre *i* e *j*. O caminho mais curto que existia antes da consideração de *k* também é um caminho mais curto após a consideração de *k*.

- *k* pertence ao caminho mais curto entre *i* e *j*. Então o caminho mais curto pode ser desmembrado em dois caminhos mais curtos: *i* até *k* e *k* até *j*.



Algoritmo de Floyd-Warshall



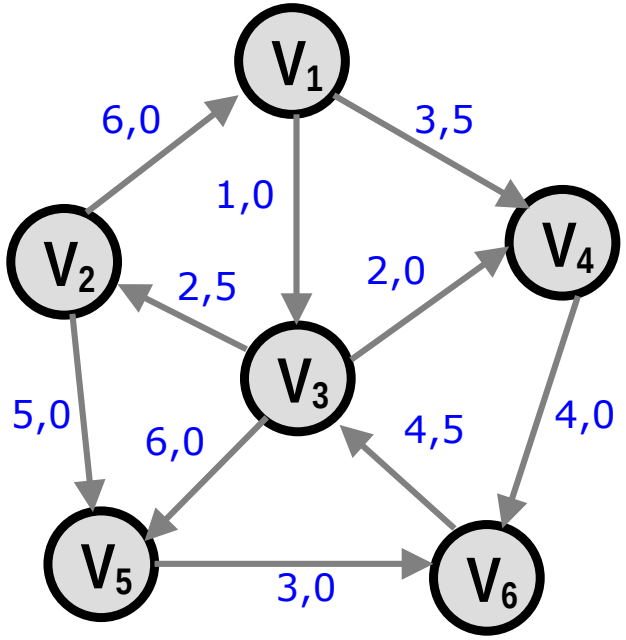
dist[**i**][**j**] > dist[**i**][**k**] + dist[**k**][**j**] ?

Algoritmo de Floyd-Warshall

```
FloydWarshall(Digrafo D, pesos w)
   Para i =1 até (|V|) faça
      Para j =1 até (|V|) faça
         dist[i][j] = w[i][j]
         pred[i][j] = -1
         Se w[i][j] != INF
            pred[i][j] = j
      dist[i][i] = 0
      pred[i][i] = i
   Para k =1 até (|V|) faça
      Para i =1 até (|V|) faça
         Para j =1 até (|V|) faça
            Se dist[i][j] > dist[i][k] + dist[k][j] então
                dist[i][j] = dist[i][k] + dist[k][j]
                pred[i][j] = pred[k][j]
```

Algoritmo de Floyd-Warshall

```
FloydWarshall(Digrafo D, pesos w)
    Para i =1 até (|V|) faça
        Para j =1 até (|V|) faça
            dist[i][j] = w[i][j]
            pred[i][j] = -1
            Se w[i][j] != INF
                pred[i][j] = i
        dist[i][i] = 0
        pred[i][i] = i
```



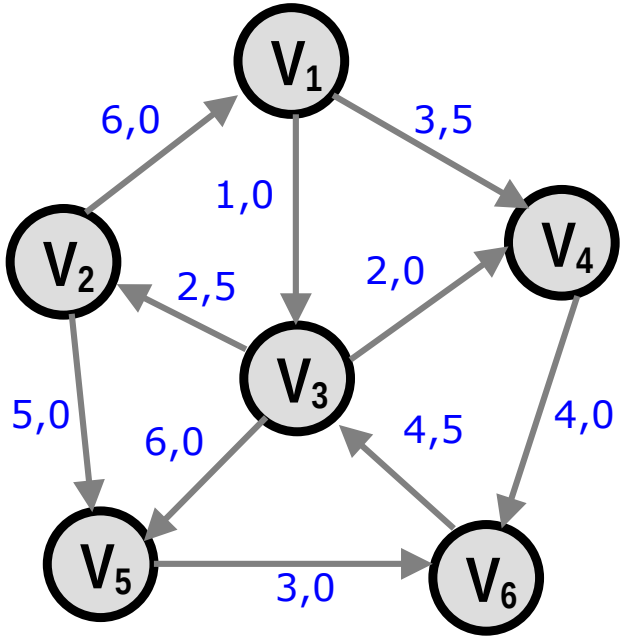
w	V1	V2	V3	V4	V5	V6
V1	-	-	1,0	3,5	-	-
V2	6,0	-	-	-	5,0	-
V3	-	2,5	-	2,0	6,0	-
V4	-	-	-	-	-	4,0
V5	-	-	-	-	-	3,0
V6	-	-	4,5	-	-	-

dist	V1	V2	V3	V4	V5	V6
V1	0	-	1,0	3,5	-	-
V2	6,0	0	-	-	5,0	-
V3	-	2,5	0	2,0	6,0	-
V4	-	-	-	0	-	4,0
V5	-	-	-	-	0	3,0
V6	-	-	4,5	-	-	0

pred	V1	V2	V3	V4	V5	V6
V1	1	-1	1	1	-1	-1
V2	2	2	-1	-1	2	-1
V3	-1	3	3	3	3	-1
V4	-1	-1	-1	4	-1	4
V5	-1	-1	-1	-1	5	5
V6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

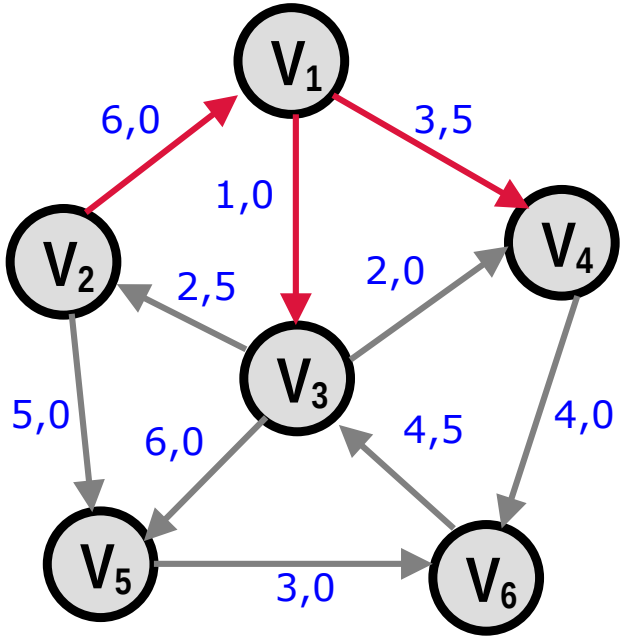
dist[1][1] > dist[1][1]+dist[1][1] 0 > 0 + 0 F
dist[1][2] > dist[1][1]+dist[1][2] I > 0 + I F
dist[1][3] > dist[1][1]+dist[1][3] 1 > 0 + 1 F
dist[1][4] > dist[1][1]+dist[1][4] 3,5> 0 + 3,5 F
dist[1][5] > dist[1][1]+dist[1][5] I > 0 + I F
dist[1][6] > dist[1][1]+dist[1][6] I > 0 + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	-	-	5,0	-
V_3	-	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	-1	-1	2	-1
V_3	-1	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

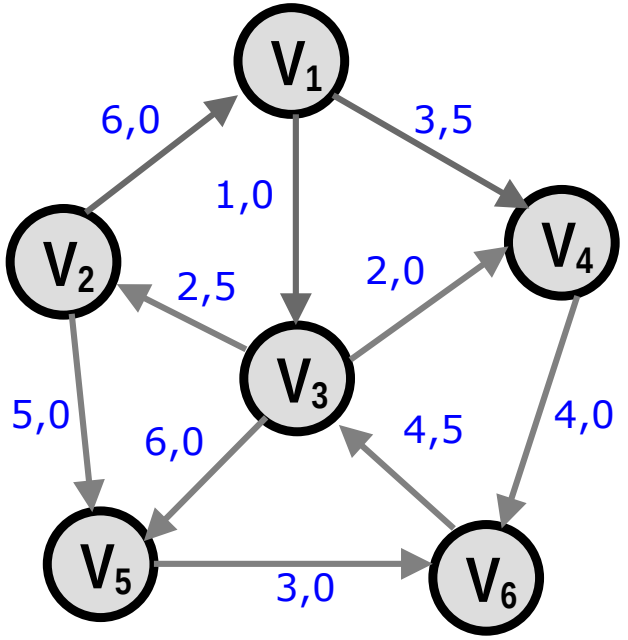
dist[2][1] > dist[2][1]+dist[1][1] 6 > 6 + 0 F
dist[2][2] > dist[2][1]+dist[1][2] 0 > 6 + I F
dist[2][3] > dist[2][1]+dist[1][3] I > 6 + 1 V
dist[2][4] > dist[2][1]+dist[1][4] I > 6 + 3,5 V
dist[2][5] > dist[2][1]+dist[1][5] 5 > 6 + I F
dist[2][6] > dist[2][1]+dist[1][6] I > 6 + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	-	-	5,0	-
V_3	-	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	-1	-1	2	-1
V_3	-1	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

Dist 4→1, 5→ 1 e 6→1 = Inf

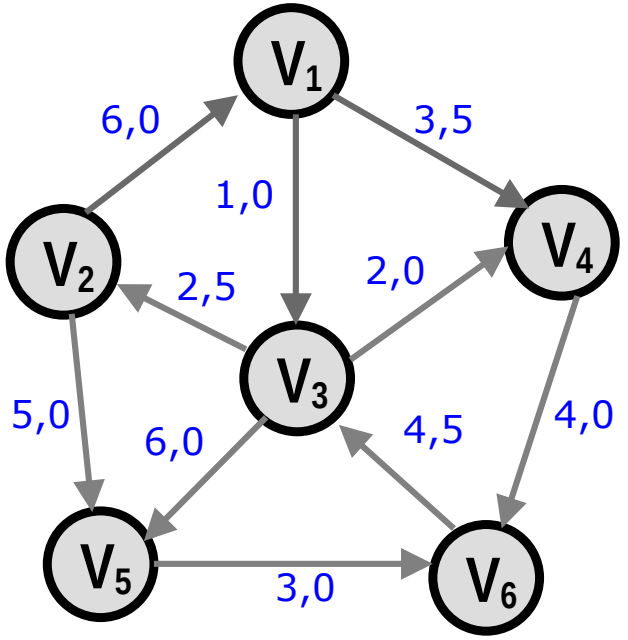
dist[3][1] > dist[3][1]+dist[1][1] I > I + 0 F
dist[3][2] > dist[3][1]+dist[1][2] 2,5 > I + I F
dist[3][3] > dist[3][1]+dist[1][3] 0 > I + 1 F
dist[3][4] > dist[3][1]+dist[1][4] 2 > I + 3,5 F
dist[3][5] > dist[3][1]+dist[1][5] 6 > I + I F
dist[3][6] > dist[3][1]+dist[1][6] I > I + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	7	9,5	5,0	-
V_3	-	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	1	1	2	-1
V_3	-1	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

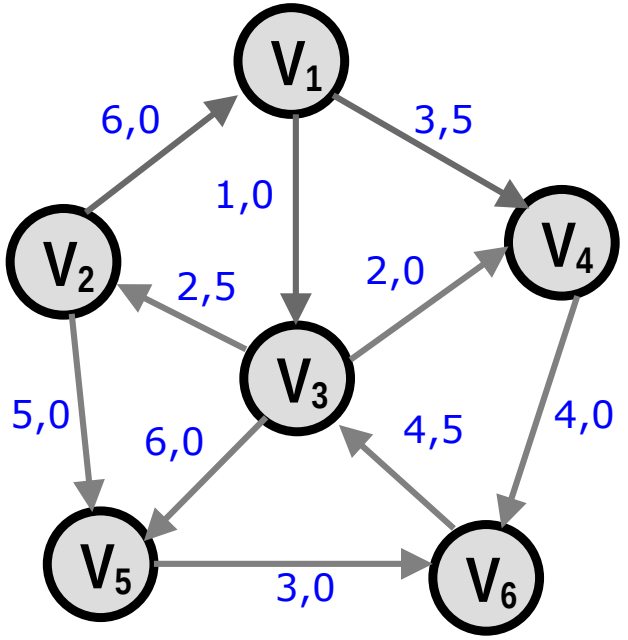
dist[1][1] > dist[1][2]+dist[2][1] 0 > I + 6 F
dist[1][2] > dist[1][2]+dist[2][2] I > I + 0 F
dist[1][3] > dist[1][2]+dist[2][3] 1 > I + 7 F
dist[1][4] > dist[1][2]+dist[2][4] 3,5> I + 9,5 F
dist[1][5] > dist[1][2]+dist[2][5] I > I + 5 F
dist[1][6] > dist[1][2]+dist[2][6] I > I + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	7	9,5	5,0	-
V_3	-	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	1	1	2	-1
V_3	-1	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

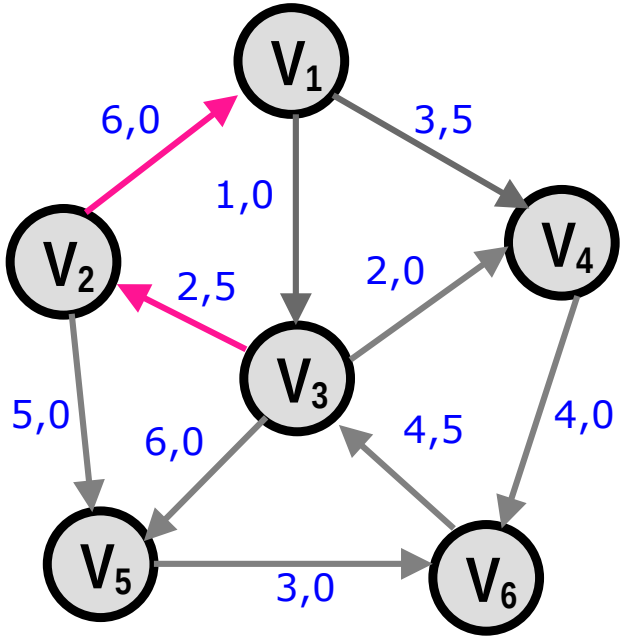
dist[2][1] > dist[2][2]+dist[2][1] 6 > 0 + 6 F
dist[2][2] > dist[2][2]+dist[2][2] 0 > 0 + 0 F
dist[2][3] > dist[2][2]+dist[2][3] 7 > 0 + 7 F
dist[2][4] > dist[2][2]+dist[2][4] 9,5 > 0 + 9,5 F
dist[2][5] > dist[2][2]+dist[2][5] 5 > 0 + 5 F
dist[2][6] > dist[2][2]+dist[2][6] I > 0 + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	7	9,5	5,0	-
V_3	-	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	1	1	2	-1
V_3	-1	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

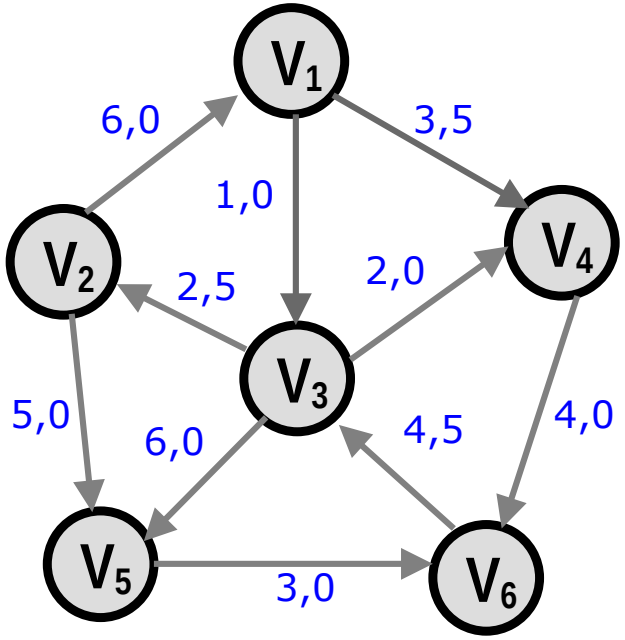
dist[3][1] > dist[3][2]+dist[2][1] I > 2,5 + 6 V
dist[3][2] > dist[3][2]+dist[2][2] 2,5 > 2,5 + 0 F
dist[3][3] > dist[3][2]+dist[2][3] 0 > 2,5 + 7 F
dist[3][4] > dist[3][2]+dist[2][4] 2 > 2,5 + 9,5 F
dist[3][5] > dist[3][2]+dist[2][5] 6 > 2,5 + 5 F
dist[3][6] > dist[3][2]+dist[2][6] I > 2,5 + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	7	9,5	5,0	-
V_3	-	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	1	1	2	-1
V_3	-1	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

Dist 4→2, 5→ 2 e 6→2 = Inf

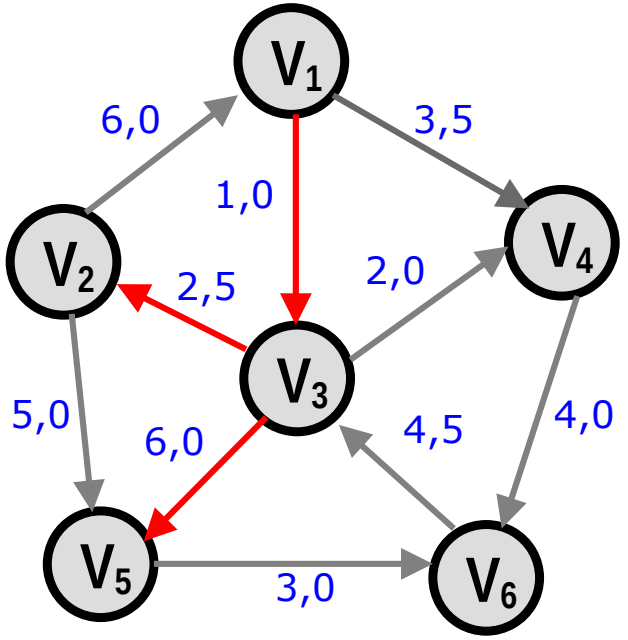
dist[4][1] > dist[4][2]+dist[2][1] I > I + 6 F
dist[4][2] > dist[4][2]+dist[2][2] I > I + 0 F
dist[4][3] > dist[4][2]+dist[2][3] I > I + 7 F
dist[4][4] > dist[4][2]+dist[2][4] 0 > I + 9,5 F
dist[4][5] > dist[4][2]+dist[2][5] I > I + 5 F
dist[4][6] > dist[4][2]+dist[2][6] 4 > I + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	7	9,5	5,0	-
V_3	8,5	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	1	1	2	-1
V_3	2	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

dist[1][1] > dist[1][3]+dist[3][1] 0 > 1 + 8,5 F

dist[1][2] > dist[1][3]+dist[3][2] I > 1 + 2,5 V

dist[1][3] > dist[1][3]+dist[3][3] 1 > 1 + 0 F

dist[1][4] > dist[1][3]+dist[3][4] 3,5> 1 + 9,5 F

dist[1][5] > dist[1][3]+dist[3][5] I > 1 + 5 V

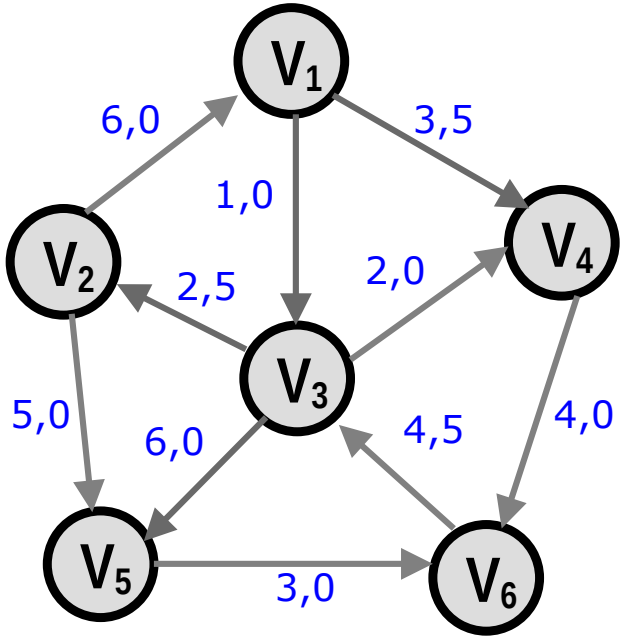
dist[1][6] > dist[1][3]+dist[3][6] I > 1 + I F

dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	-	1,0	3,5	-	-
V_2	6,0	0	7	9,5	5,0	-
V_3	8,5	2,5	0	2,0	6,0	-
V_4	-	-	-	0	-	4,0
V_5	-	-	-	-	0	3,0
V_6	-	-	4,5	-	-	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	-1	1	1	-1	-1
V_2	2	2	1	1	2	-1
V_3	2	3	3	3	3	-1
V_4	-1	-1	-1	4	-1	4
V_5	-1	-1	-1	-1	5	5
V_6	-1	-1	6	-1	-1	6

Algoritmo de Floyd-Warshall

```
Para k =1 até (|V|) faça
  Para i =1 até (|V|) faça
    Para j =1 até (|V|) faça
      Se dist[i][j] > dist[i][k] + dist[k][j] então
          dist[i][j] = dist[i][k] + dist[k][j]
          pred[i][j] = pred[k][j]
```




```
k = 1 2 3 4 5 6
i = 1 2 3 4 5 6
j = 1 2 3 4 5 6
```

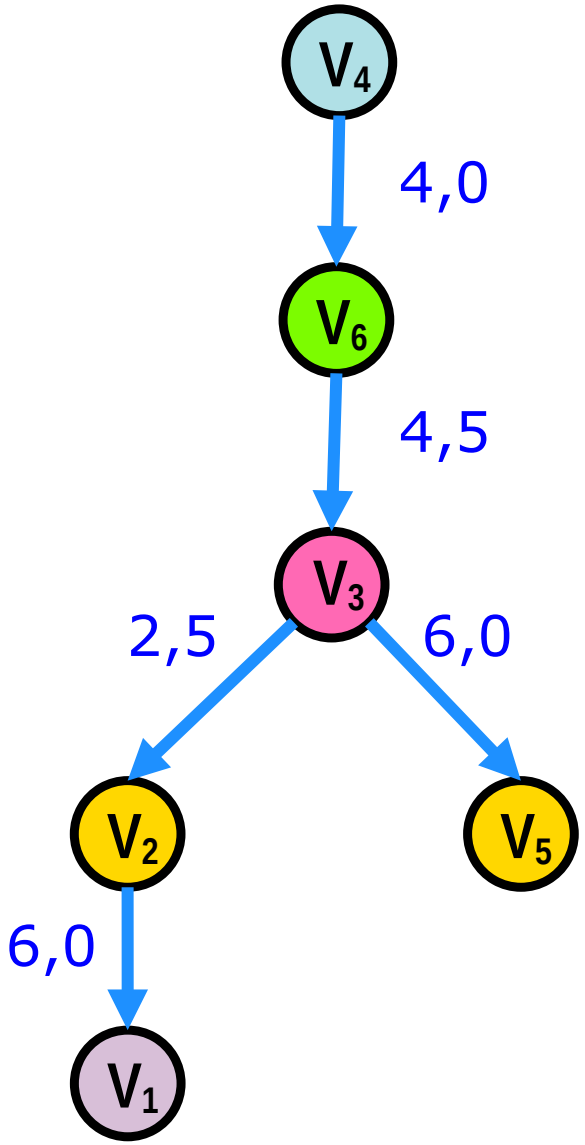
Resultado final:

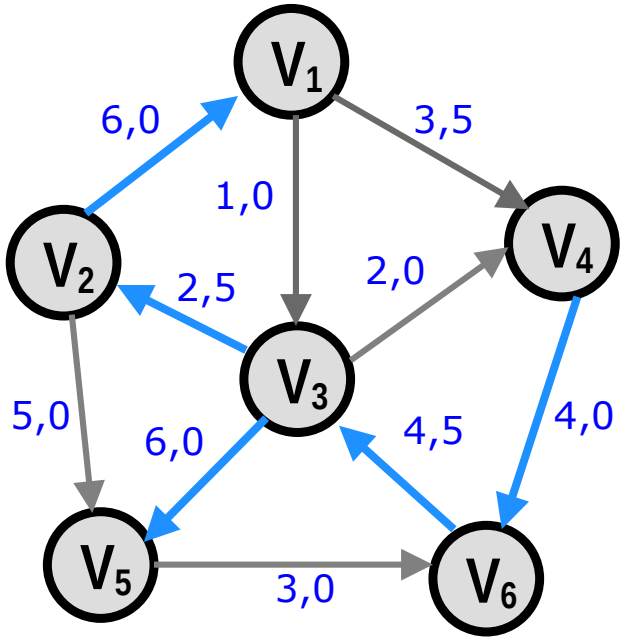
dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	3,5	1	3	7	7
V_2	6	0	7	9	5	8
V_3	8,5	2,5	0	2	6	6
V_4	17	11	8,5	0	14,5	4
V_5	16	10	7,5	9,5	0	3
V_6	13	7	4,5	6,5	10,5	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	3	1	3	3	4
V_2	2	2	1	2	2	5
V_3	2	3	3	3	3	4
V_4	2	3	6	4	3	4
V_5	2	3	6	3	5	5
V_6	2	3	6	3	3	6

E como recuperar os caminhos

Uma **árvore de caminhos mais curtos**, T_s, é uma árvore com raiz r tal que a distância entre r e qualquer vértice u de T_s é exatamente a mesma que o comprimento do menor caminho entre r e u.





dist	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	3,5	1	3	7	7
V_2	6	0	7	9	5	8
V_3	8,5	2,5	0	2	6	6
V_4	17	11	8,5	0	14,5	4
V_5	16	10	7,5	9,5	0	3
V_6	13	7	4,5	6,5	10,5	0

pred	V_1	V_2	V_3	V_4	V_5	V_6
V_1	1	3	1	3	3	4
V_2	2	2	1	2	2	5
V_3	2	3	3	3	3	4
V_4	2	3	6	4	3	4
V_5	2	3	6	3	5	5
V_6	2	3	6	3	3	6

Árvore de Caminhos mais Curtos

Aplicações:

Redes Multicast

VLSI

Instalação de redes de infraestrutura

Referências e outros materiais

Ahuja, R. K., Mehlhorn, K., Orlin, J. B. & Tarjan, R. E. (1990). Faster Algorithms for the Shortest Path Problem. *Journal of the Association for Computing Machinery* **37**(2):213-223.

Bellman, R.E. (1958). On a routing problem, *Quart. Appl. Math.* Vol. 16, pp. 87-90.

Fredman, M. L. & Tarjan, R. E. (1987). Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the Association Computing Machinery* **34**:596-615.

Ford, L. R. Jr. (1956). Network flow theory, The RAND Corp., P-293, Santa Monica, California.

Hagerup, T. (2000). Improved Shortest Paths in the Word RAM. In: *27th International Colloquium on Automata, Languages and Programming*, 61-72.

Moore, E. F. (1967). The shortest path through a maze, *Proceedings of the International Symposium on the Theory of Switching,* Part II, pp. 285. *The Annals of the Computation Laboratory of Harvard University*, vol. 30, Cambridge.

Referências e outros materiais

Raman, R. (1997). Recent Results on Single Source Shortest Paths Problem. SIGACT News 28:81-87.

Raman. R. (1996). Priority Queues: Small, Monotone and TransDichotomous. In: Proceedings 4th Annual European Symposium Algorithms, Lecture Notes in Computational Science 1136:121-137.

Thorup, M. (1999). Undirected SingleSource Shortest Paths with Positive Integer Weights in Linear Time. *Journal of the Association for Computing Machinery* **46**:362-394.

Thorup, M. (2000a). On RAM Priority Queues. *SIAM Journal on Computing* **30**:86-109.