Algoritmo de Dijkstra

Caminho mínimo

Em muitas situações é útil determinar o caminho mais curto entre duas localizações, quando esse problema trata de arestas de mesmo peso ou sem ponderação, esse problema se resume a buscar um caminho que percorra a menor quantidade de arestas entre a origem e o destino.

Quando as arestas do grafo são ponderadas, a determinação do caminho mínimo precisa levar em conta o "custo" desses pesos na determinação do percurso desejado.

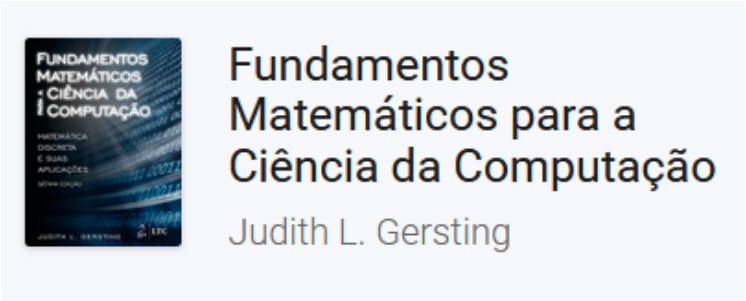
Uma solução conhecida para esse problema é implementada pelo algoritmo de Dijkstra o qual trata um grafo cujas arestas ponderadas (pesos positivos) e considera o cálculo do caminho mínimo entre dois vértices pré-determinados.

É importante destacar que Dijkstra não é a única opção para a determinação do caminho mínimo, um outro algoritmo muito citado na literatura e que lida com o mesmo tipo de problema é o algoritmo de Bellman-Ford.

Com uma boa implementação, o tempo de execução do algoritmo de Dijkstra é inferior ao do algoritmo de Bellman-Ford (Cormen).

Caminho mínimo

Vamos agora ao algoritmo de Dijkstra apresentado na seção CAMINHO MÍNIMO E ÁRVORE GERADORA MÍNIMA do livro da Judith Gersting (disponível no formato ebook e fisicamente no acervo da biblioteca do CCT).



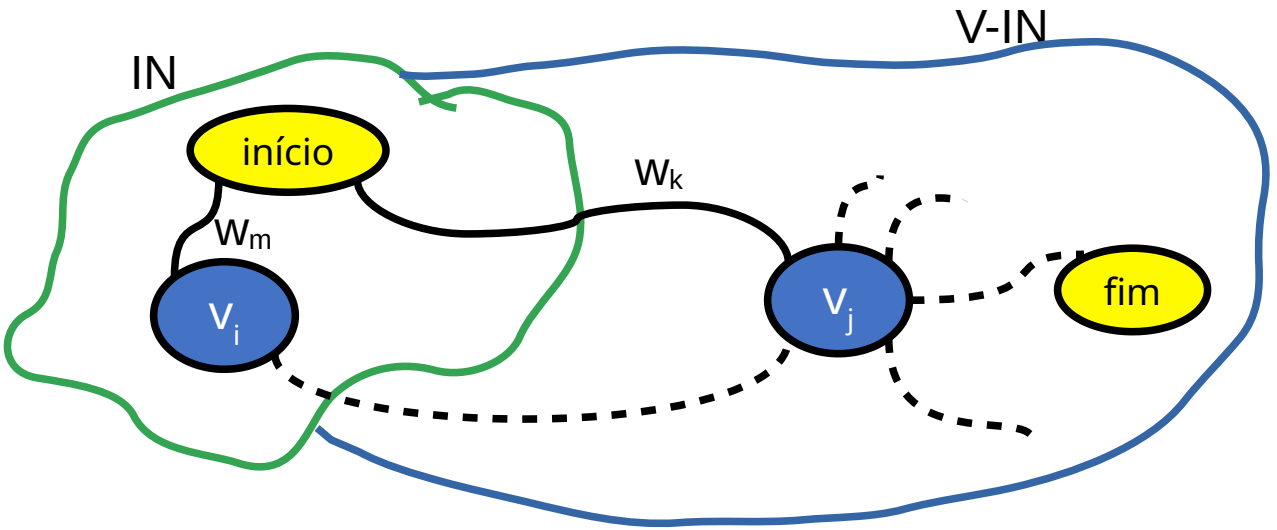
Demonstração (fonte Wikipedia)

Dijkstra

Seja G(V,E) um grafo ponderado, Dijkstra mantém um conjunto IN de vértices já processados (vértices para os quais já foram determinados os pesos finais de caminhos mínimos que partem do vértice de saída).

O algoritmo seleciona repetidamente o vértice ainda não processado pertencente a {V – IN} que tem a mínima estimativa do caminho mínimo.

A ideia é avaliar o custo para alcançar esse vértice via algum caminho mínimo atual ou por outro caminho mais promissor (em termos do custo total). Esse processo é chamado de relaxação.

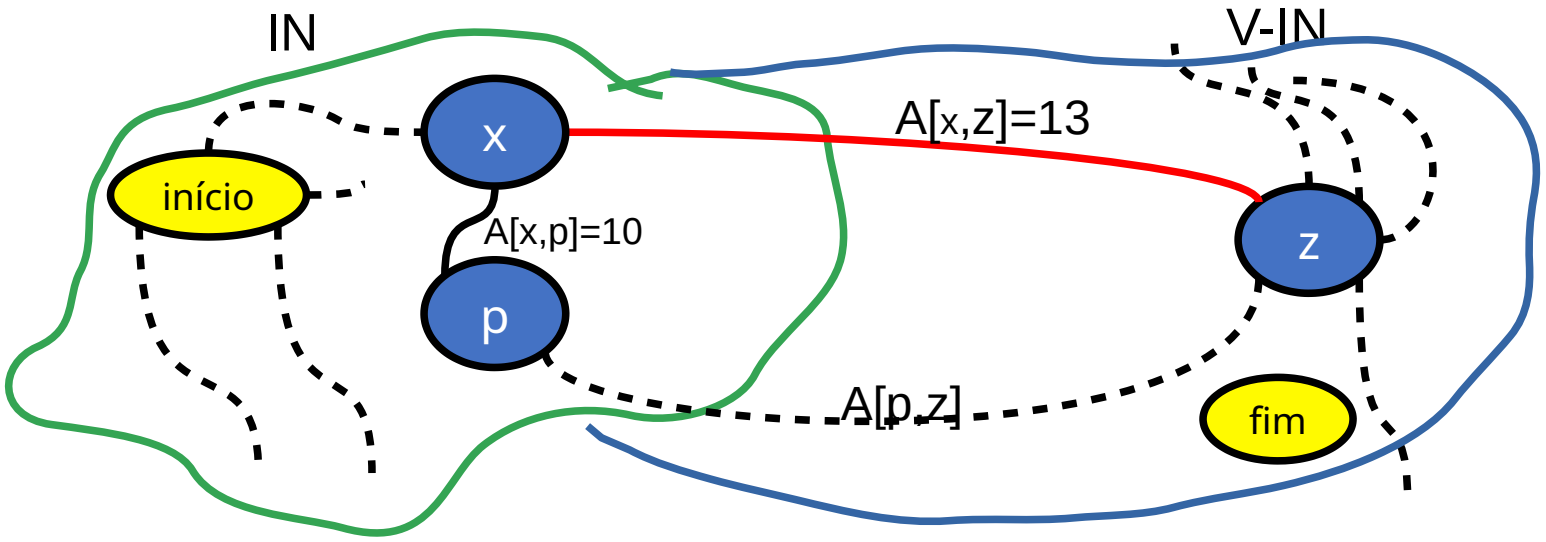


Dijkstra

Seja a Matriz de adjacências *A* de G(V,E) e um caminho mínimo que vem sendo promissor e chega em *p* a partir de *x e outros vértices não exibidos*.
Deseja-se avaliar se a próxima configuração do caminho será {x, p, z} ou {x, z}, ou seja, o caminho até "z" passa por "p" ou vem direto de x?

Para tanto se avalia uma expressão chamada "relaxação":

Relaxação: d[z] = min(d[z],d[p]+A[p,z])



Dijkstra

ALGORITMO Caminho Mínimo do vértice **x** ao **y** por Dijkstra
Inicializações:
A: matriz de adjacências;
IN: conjunto de vértices; {menor caminho conhecido a partir de x}
d: vetor de inteiros; {distância a partir de x usando os vértices de IN}
s: vetor de vértices; {vértice anterior no caminho mínimo}
begin
(inicia o conjunto IN e os vetores d e s}
IN:= {x}; // vértice de partida
d[x] : = 0; //distância inicial do vértice de partida até ele mesmo

//complementado a inicialização
for todos os vértices z que não pertençam a IN do
    begin
      d[z] := A[x, z]; //distâncias do vértice de partida conforme matriz adj
      s[z] := x; // referencial inicial sempre é o vértice de saída
end;

Dijkstra

Complemento do algoritmo de Dijkstra

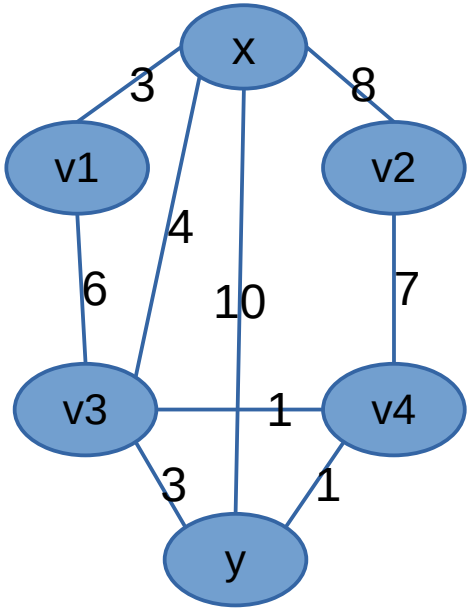
```
enquanto y não pertence a IN faça
    p = nó z não pertencente a IN com d[z] mínima
    IN = IN ⋃ {p}
    para todos os nós não pertencentes a IN faça
        DistânciaAnterior = d[z]
        d[z] = min(d[z], d[p] + A[p, z])
        se d[z] < DistânciaAnterior então
            s[z] = p
escreva (“Em ordem inversa, os nós do caminho são”)
escreva (y)
z = y
repita
    escreva (s[z])
    z = s[z]
até z = x
escreva (“A distância percorrida é”, d[y])
```

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Inicialização*:*

x é o vértice de partida *(s=x)*

Ao final da fase de inicialização teremos IN={x} e *d* contendo todas as distâncias de *x* até os demais vértices em {*V-IN}*.



Matriz de adjacências (A)						
	x	v1	v2	v3	v4	y
x	∞	3	8	4	∞	10
v1	3	∞	∞	6	∞	∞
v2	8	∞	∞	∞	7	∞
v3	4	6	∞	∞	1	3
v4	∞	∞	7	1	∞	1
y	10	∞	∞	3	1	∞

IN V-IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10
s	–	x	x	x	x	x

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (1ª iteração):

ENQUANTO o destino *y* não pertencer a *IN*:

// 1) determine *p*, o vértice com menor valor d[ ] que pertença a V-IN

No caso: p=v1

IN | V-IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10

// 2) atualize *IN = IN U p*

No caso: IN={x,v1} → V-IN={v2,v3,v4,y}

IN | V-IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10

// 3) Agora verifique se há um caminho mínimo chegando em *z em V-IN*

PARA todo *z* em *V-IN*

DistânciaAnterior = d[z]

calcule a relaxação: d[z] = min(d[z],d[p]+A[p,z])

SE d[z] < DistânciaAnterior:

ENTÃO: s[z]=p

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (1ª iteração):

ENQUANTO o destino *y* não pertencer a *IN*:

No caso: p=v1

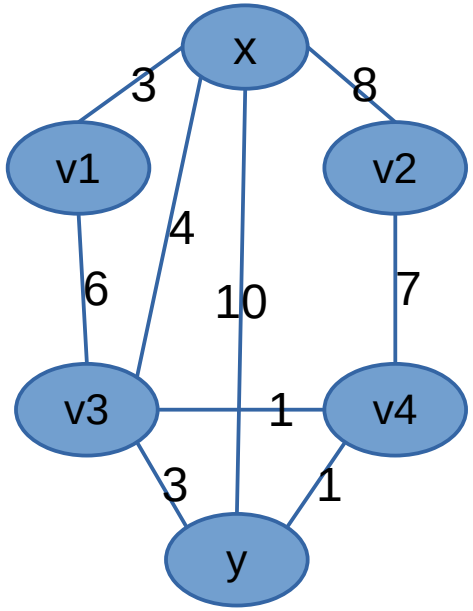
PARA todo *z* em V-IN={v2,v3,v4,y}, d[z] = min(d[z],d[p]+A[p,z])

z=v2: d[v2] = min{d[v2],d[v1]+A[v1,v2]}= min{8,3+∞}=8

z=v3: d[v3] = min{d[v3],d[v1]+A[v1,v3]}= min{4,3+6}=4

z=v4: d[v4] = min{d[v4],d[v1]+A[v1,v4]}= min{∞,3+∞}=∞

z=y: d[y] = min{d[y],d[v1]+A[v1,y]}= min{10,3+∞}=10



Matriz de adjacências (A)						
	x	v1	v2	v3	v4	y
x	∞	3	8	4	∞	10
v1	3	∞	∞	6	∞	∞
v2	8	∞	∞	∞	7	∞
v3	4	6	∞	∞	1	3
v4	∞	∞	7	1	∞	1
y	10	∞	∞	3	1	∞

IN V-IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10
s	–	x	x	x	x	x

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (1ª iteração):

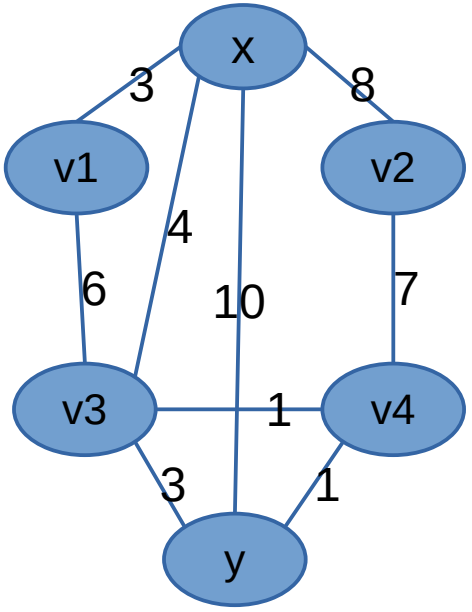
No caso: p=v1 e *z* em V-IN={v2,v3,v4,y}, d[z] = min(d[z],d[p]+A[p,z])

d[v2] = 8
→ manteve a mesma *DistânciaAnterior* → não altere a tabela (d,s)

d[v3] = 4
→ manteve a mesma *DistânciaAnterior* → não altere a tabela (d,s)

d[v4] = ∞
→ manteve a mesma *DistânciaAnterior* → não altere a tabela (d,s)

d[y] = 10
→ manteve a mesma *DistânciaAnterior* → não altere a tabela (d,s)



Os valores na tabela se mantêm inalterados:

IN | V-IN

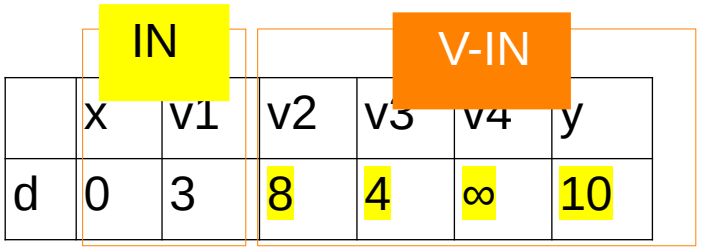
	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10
s	–	x	x	x	x	x

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (2ª iteração):

// 1) determine *p*, o vértice com menor valor d[ ] que pertença a V-IN

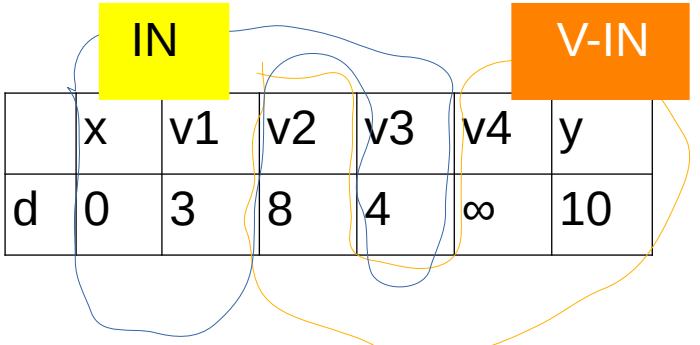
No caso: p=v3



	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10

// 2) atualize *IN = IN U p*

No caso: IN={x,v1,v3} → V-IN={v2,v4,y}



	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10

// 3) verifique se há um caminho mínimo chegando em *z em V-IN*

PARA todo *z* em *V-IN*

DistânciaAnterior = d[z]

calcule a relaxação: d[z] = min(d[z],d[p]+A[p,z])

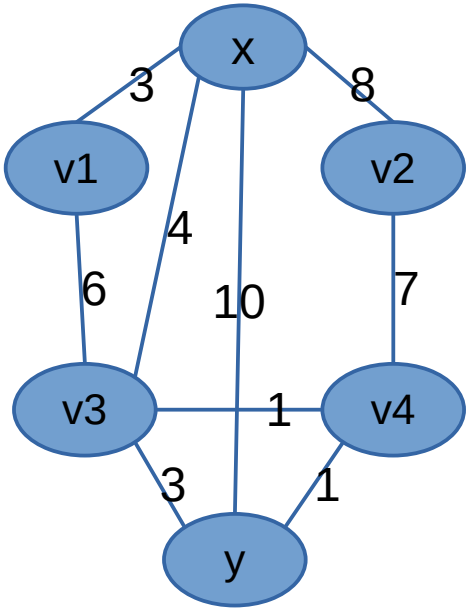
SE d[z] < DistânciaAnterior:

ENTÃO: s[z]=p

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (2ª iteração):

ENQUANTO o destino *y* não pertencer a *IN*:
No caso: p=v3
PARA todo *z* em V-IN={v2,v4,y}, d[z] = min(d[z],d[p]+A[p,z])
z=v2: d[v2] = min{d[v2],d[v3]+A[v3,v2]}= min{8,4+∞}=8
z=v4: d[v4] = min{d[v4],d[v3]+A[v3,v4]}= min{∞,4+1}=5
z=y: d[y] = min{d[y],d[v3]+A[v3,y]}= min{10,4+3}=7



Matriz de adjacências (A)						
	x	v1	v2	v3	v4	y
x	∞	3	8	4	∞	10
v1	3	∞	∞	6	∞	∞
v2	8	∞	∞	∞	7	∞
v3	4	6	∞	∞	1	3
v4	∞	∞	7	1	∞	1
y	10	∞	∞	3	1	∞

IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10
s	–	x	x	x	x	x

V-IN

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (2ª iteração):

No caso: p=v3 e *z* em V-IN={v2,v4,y}, d[z] = min(d[z],d[p]+A[p,z])

d[v2] = 8

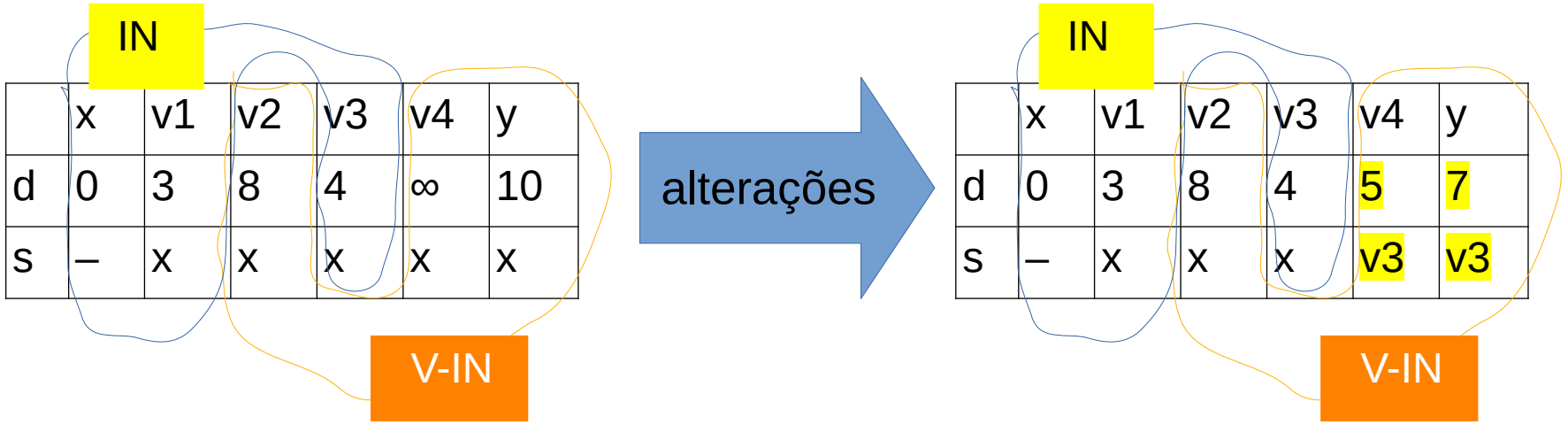
→ manteve a mesma *DistânciaAnterior* → não altere a tabela (d,s)

d[v4] = 5

→ menor do que *DistânciaAnterior* = ∞ → altera a tabela (d,s)

d[y] = 7

→ menor do que a *DistânciaAnterior* = *10* → altere a tabela (d,s)



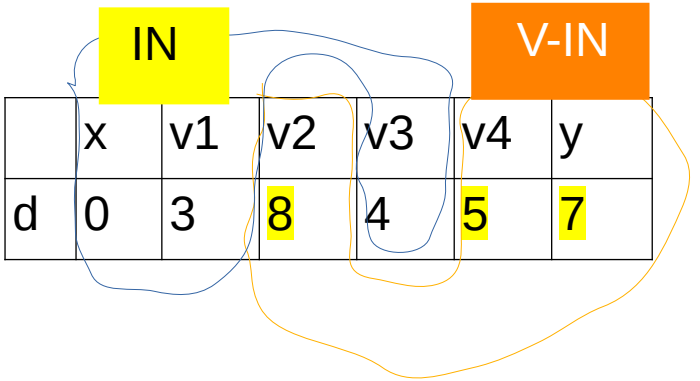
	x	v1	v2	v3	v4	y
d	0	3	8	4	∞	10
s	–	x	x	x	x	x

	x	v1	v2	v3	v4	y
d	0	3	8	4	5	7
s	–	x	x	x	v3	v3

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

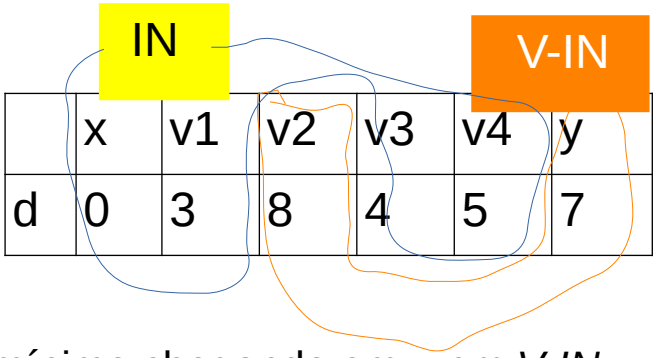
Laço principal (3ª iteração):

// 1) determine *p*, o vértice com menor valor d[ ] que pertença a V-IN
No caso: p=v4



	x	v1	v2	v3	v4	y
d	0	3	8	4	5	7

// 2) atualize *IN = IN U p*
No caso: IN={x,v1,v3,v4} → V-IN={v2,y}



	x	v1	v2	v3	v4	y
d	0	3	8	4	5	7

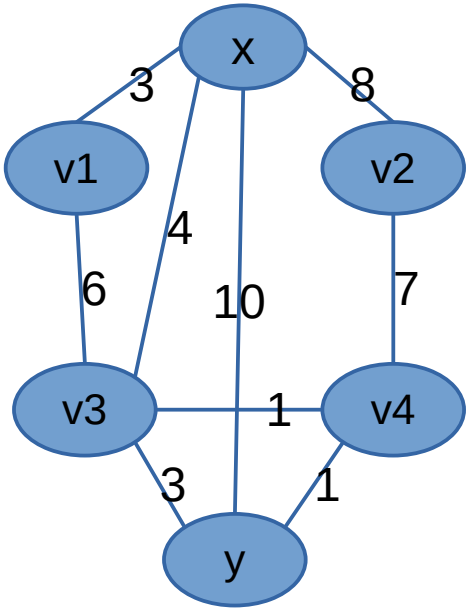
// 3) verifique se há um caminho mínimo chegando em *z em V-IN*
PARA todo *z* em *V-IN*

DistânciaAnterior = d[z]
calcule a relaxação: d[z] = min(d[z],d[p]+A[p,z])
SE d[z] < DistânciaAnterior:
ENTÃO: s[z]=p

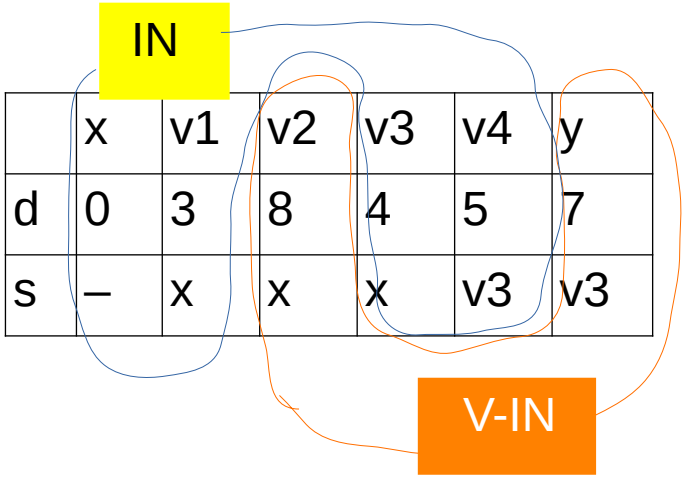
Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (3ª iteração):

ENQUANTO o destino *y* não pertencer a *IN*:
No caso: p=v4
PARA todo *z* em V-IN={v4,y}, d[z] = min(d[z],d[p]+A[p,z])
z=v2: d[v2] = min{d[v2],d[v4]+A[v4,v2]}= min{8,5+7}=8
z=y: d[y] = min{d[y],d[v4]+A[v4,y]}= min{10,5+1}=6



Matriz de adjacências (A)						
	x	v1	v2	v3	v4	y
x	∞	3	8	4	∞	10
v1	3	∞	∞	6	∞	∞
v2	8	∞	∞	∞	7	∞
v3	4	6	∞	∞	1	3
v4	∞	∞	7	1	∞	1
y	10	∞	∞	3	1	∞



	x	v1	v2	v3	v4	y
d	0	3	8	4	5	7
s	–	x	x	x	v3	v3

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (3ª iteração):

No caso: p=v4 e *z* em V-IN={v2,y}, d[z] = min(d[z],d[p]+A[p,z])

d[v2] = 8

→ manteve a *DistânciaAnterior* → não altera a tabela (d,s)

d[y] = 6

→ menor do que *DistânciaAnterior* = *7*→ altere a tabela (d,s)

IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	5	7
s	–	x	x	x	v3	v3

V-IN

alterações

IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	5	6
s	–	x	x	x	v3	v4

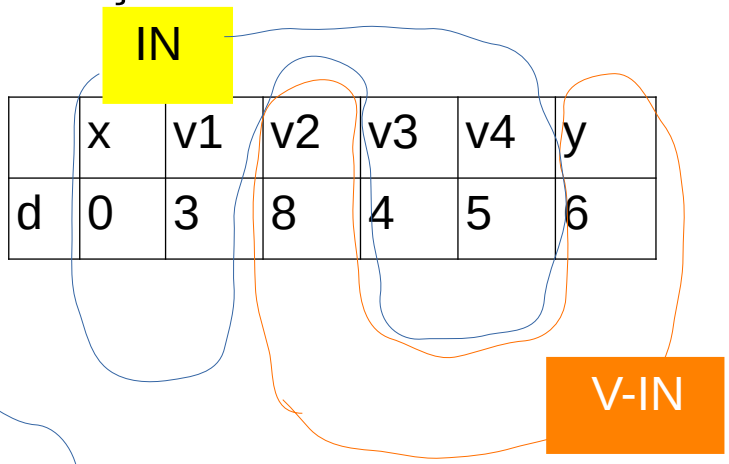
V-IN

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (4ª iteração):

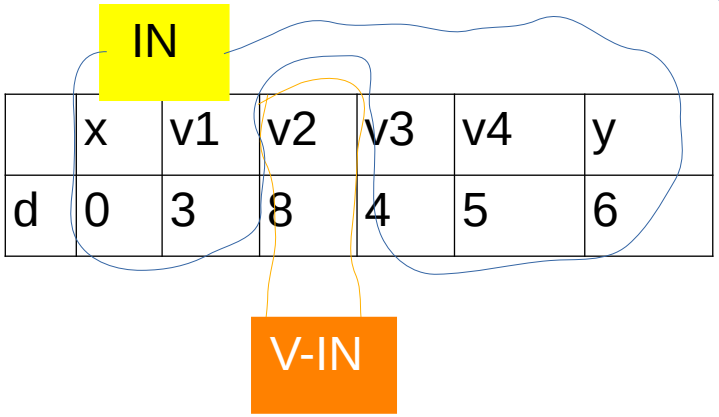
// 1) determine *p*, o vértice com menor valor d[ ] que pertença a V-IN

No caso: p=y



// 2) atualize *IN = IN U p*

No caso: IN={x,v1,v4,v3,y} → V-IN={v2}



// 3) verifique se há um caminho mínimo chegando em *z em V-IN*

PARA todo *z* em *V-IN*

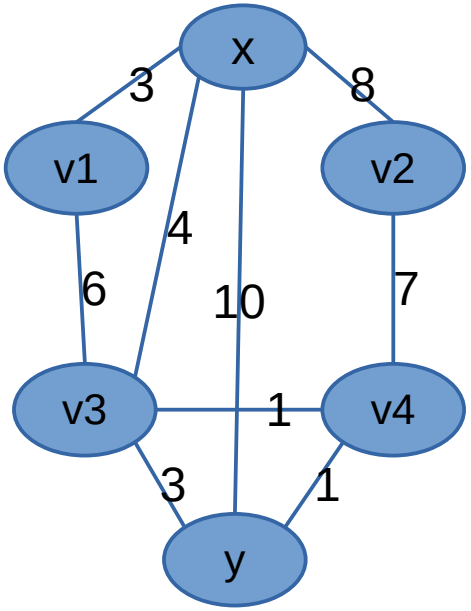
DistânciaAnterior = d[z]
calcule a relaxação: d[z] = min(d[z],d[p]+A[p,z])
SE d[z] < DistânciaAnterior:
ENTÃO: s[z]=p

Simulação do algoritmo de Dijkstra (*CaminhoMinimo em* Judith Gersting)

Laço principal (5ª iteração):

Não ocorre pois y ∈ IN

ENQUANTO o destino *y* não pertencer a *IN*:
é FALSA a condição que garantiria a iteração



IN

	x	v1	v2	v3	v4	y
d	0	3	8	4	5	6
s	–	x	x	x	v3	v4

V-IN

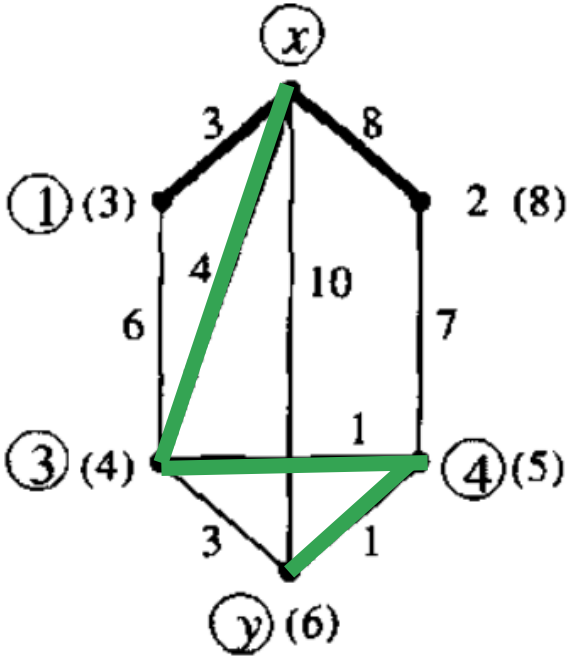
Agora que y pertence a IN, o laço while termina.

O caminho passa por y, s[y] = 4, s[4] = 3 e .v[3] = x.

Assim, o caminho usa os vértices x, 3, 4 e y. (O algoritmo nos fornece estes vértices na ordem inversa.)

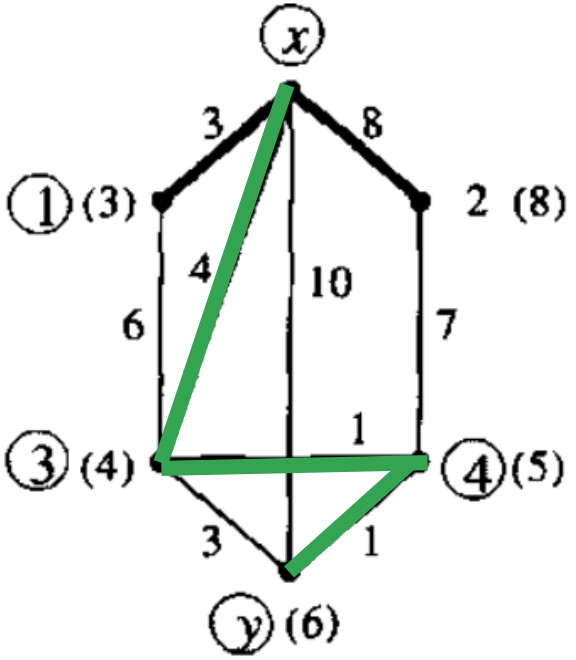
A distância do caminho é d[y] = 6

	x	1	2	3	4	y
d	0	3	8	4	5	6
s	-	x	x	x	3	4



Experimente esse mesmo grafo no app online https://algo-dijkstra.vercel.app/

	x	1	2	3	4	y
d	0	3	8	4	5	6
s	-	x	x	x	3	4



Caminho mínimo

Utilizando matriz de adjacências

n = |V|, a complexidade do Dijkstra $O(n^2)$

Podendo ser reduzida para O((V+E)*logV) usando lista de adjacências e uma fila de prioridade min-heap-priority para armazenar vértices ainda não visitados ordenados pelos pesos das arestas.

Tempo para visitar cada vértice = O(V+E)

Tempo para processar cada vizinho de um vértice = O(logV).

total=O(V+E)∗O(logV)=O((V+E)logV)

https://www.scaler.com/topics/data-structures/dijkstra-algorithm/

CORMEN, 2002

Caminho mínimo

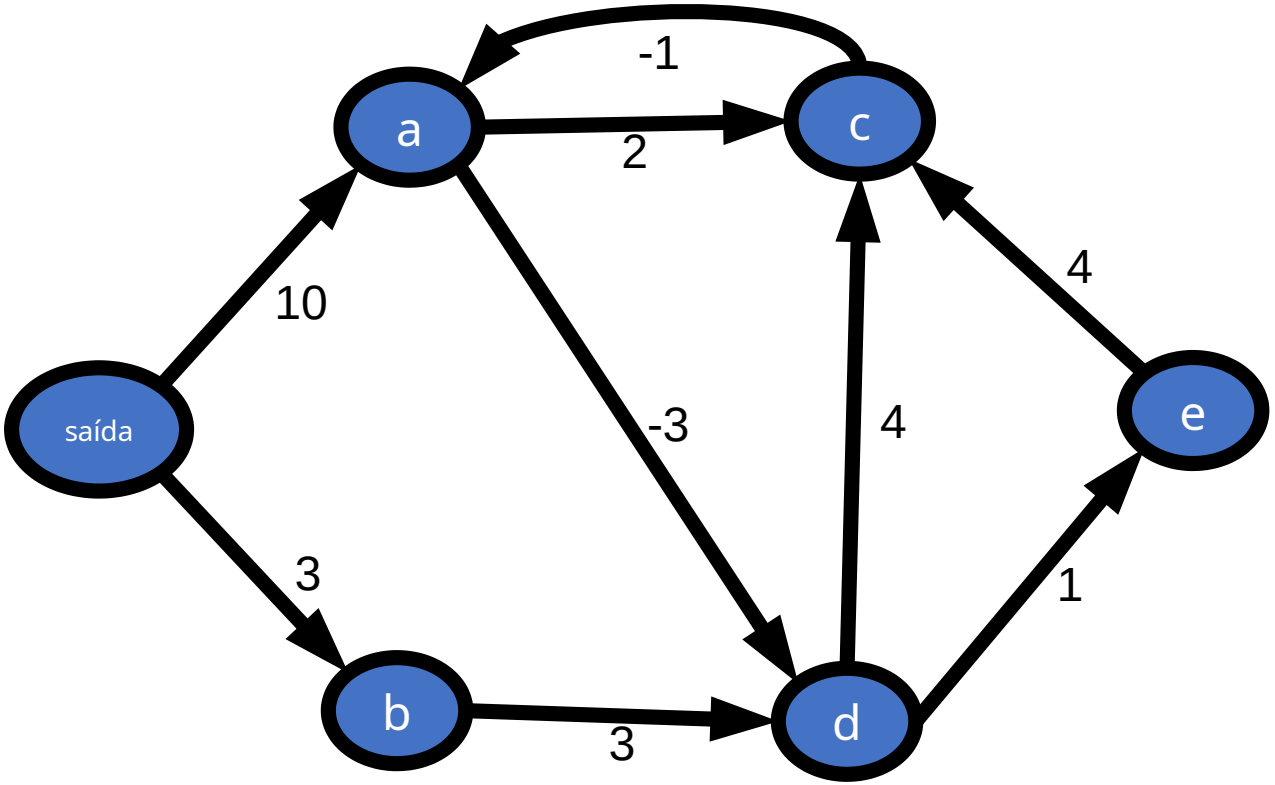
Conforme já fora mencionado, o algoritmo de Bellman-Ford também resolve o problema de caminho mínimo

Algumas diferenças entre Bellman-Ford e algoritmo de Dijkstra:

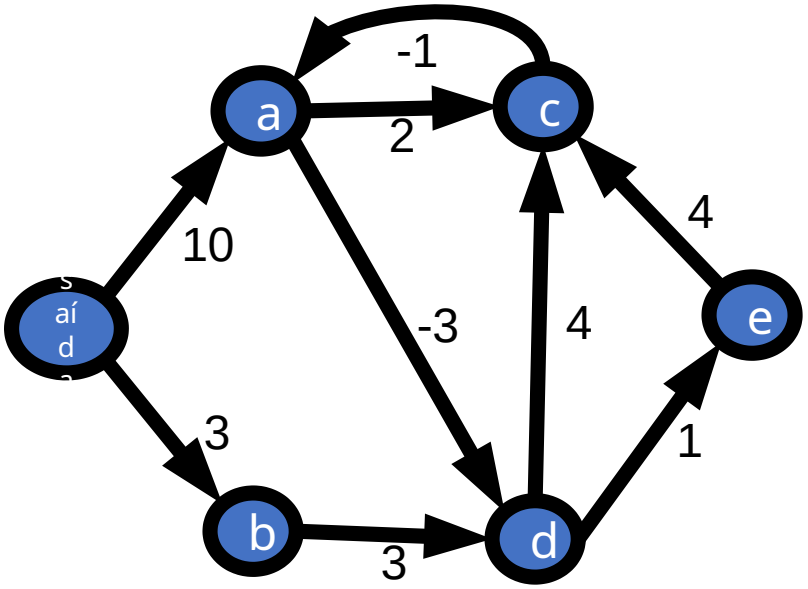
1. Dijkstra trata apenas arestas com pesos positivos. Calculando o caminho de uma fonte a um destino;
2. O algoritmo de Bellman–Ford consegue tratar arestas com pesos negativos.
3. Bellman–Ford indica se existe ou não um ciclo de peso negativo (ida e volta totaliza peso negativo) que pode ser alcançado da fonte. Se tal ciclo existe, o algoritmo sinaliza a ausência de solução. Se tal ciclo não existe, o algoritmo produz os caminhos mínimos e seus pesos de uma fonte para todos os outros vértices;

   Se num grafo existe um ciclo cuja soma dos custos das suas arestas é negativa, repetindo este ciclo, pode reduzir-se o comprimento de um caminho tanto quanto se queira [Cardoso et al.].

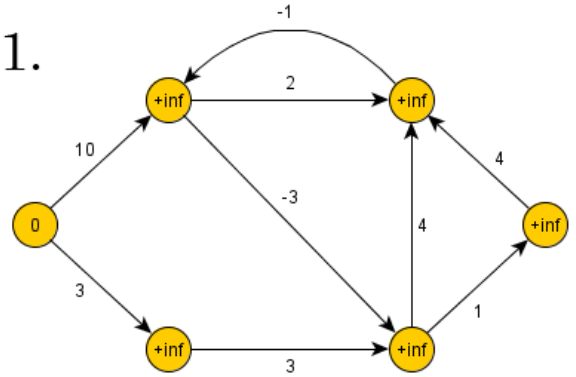
Estude o algoritmo Bellman-Ford (sugestão: pg. 538 do livro da Judith Gersting) e o aplique para encontrar o caminho mínimo entre o vértice “saída” no grafo abaixo:



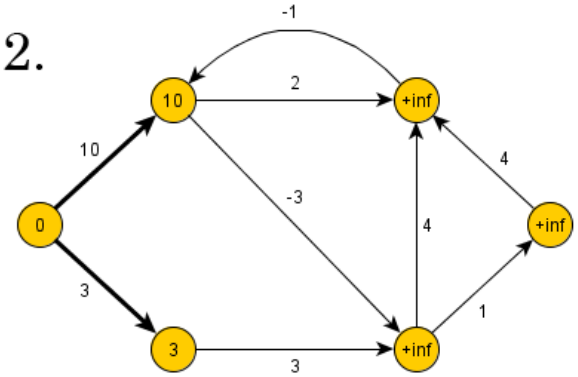
http://www.programming-algorithms.net/article/47389/Bellman-Ford-algorithm



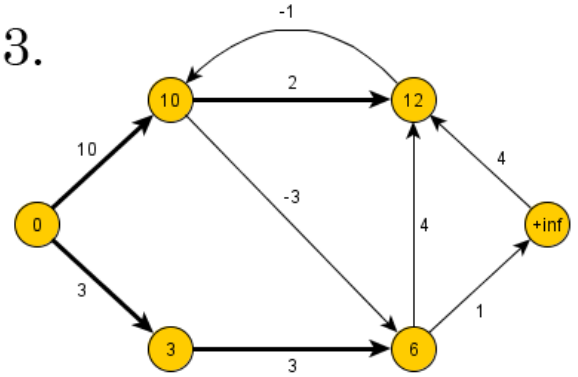
1.



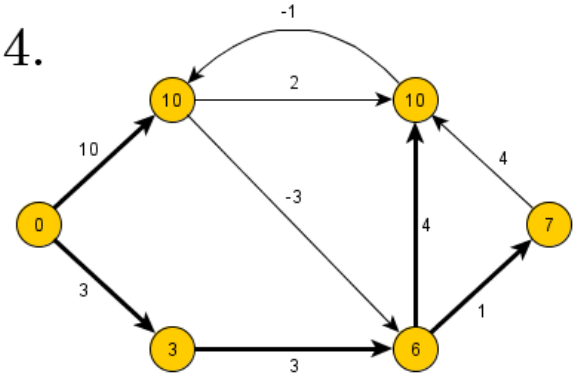
2.



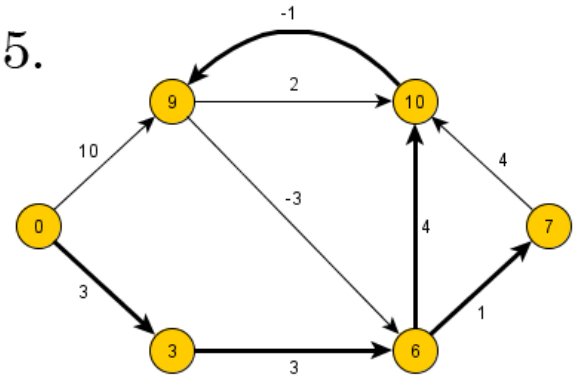
3.



4.



5.



http://www.programming-algorithms.net/article/47389/Bellman-Ford-algorithm

TEG

Bibliografia

Básica

LUCCHESI, C. L. et alli. Aspectos Teóricos da Computação, Parte C: Teoria dos Grafos, projeto Euclides, 1979.

SANTOS, J. P. O. et alli. Introdução à Análise Combinatória. UNICAMP; 1995.

SZWARCFITER, J. L. Grafos e Algoritmos Computacionais. Campus, 1986.

GERSTING, Judith L. Fundamentos Matemáticos para a Ciência da Computação. Rio de Janeiro. 3a Ed. Editora.

Complementar:

1.) CORMEN, T. Introduction to Algorithms, third edition, MIT press, 2009

2.) ROSEN, K. Discrete Mathematics and its applications, seventh edition, McGraw Hill, 2011.

3.) WEST, Douglas, B. Introduction to Graph Theory, second edition, Pearson, 2001.

4.) BONDY, J.A., MURTY, U.S.R., Graph Theory with applications , Springer, 1984.

5.) SEDGEWICK, R. Algorithms in C - part 5 - Graph Algorithms, third edition, 2002, Addison-Wesley.

6.) GOLDBARG, M., GOLDBARG E., Grafos: Conceitos, algoritmos e aplicações. Editora Elsevier, 2012.

7.) BONDY, J.A., MURTY, U.S.R., Graph Theory with applications , Springer, 1984

8.) FEOFILOFF, P., KOHAYAKAWA, Y., WAKABAYASHI, Y., uma introdução sucinta à teoria dos grafos. 2011. (www.ime.usp.br/~pf/teoriadosgrafos)

9.) DIESTEL, R. Graph Theory, second edition, springer, 2000

10.) FURTADO, A. L. Teoria de grafos. Rio de janeiro. Editora LTC. 1973.

11.) WILSON, R.J. Introduction to Graph Theory. John Wiley & Sons Inc., 1985

12.) BOAVENTURA NETTO , P. O. Grafos: Teoria, Modelos, Algoritmos. Edgard Blucher, SP, quinta edição

Tutoriais, artigos, notas de aula...

Vários livros podem ser acessados np formato eletrônico (e-book) via

https://www.udesc.br/bu/acervos/ebook

Exemplos:

Teoria Computacional de Grafos - Os Algoritmos

Jayme Luiz Szwarcfiter

Fundamentos Matemáticos para a Ciência da Computação

Judith L. Gersting

Grafos

Marco Goldbarg

Algoritmos - Teoria e Prática

Thomas Cormen