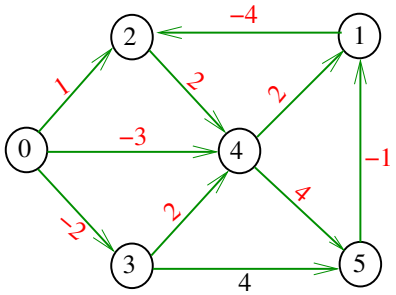
Melhores momentos

AULA 17

Problema da SPT

Problema: Dado um vértice s de um digrafo com custos (possivelmente negativos) nos arcos, encontrar uma SPT com raiz s

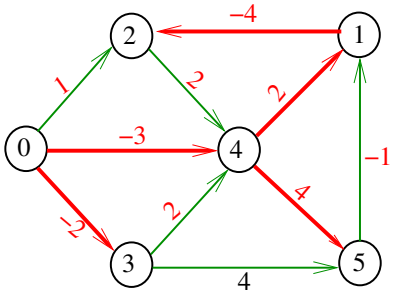
Entra:



Problema da SPT

Problema: Dado um vértice s de um digrafo com custos (possivelmente negativos) nos arcos, encontrar uma SPT com raiz s

Sai:

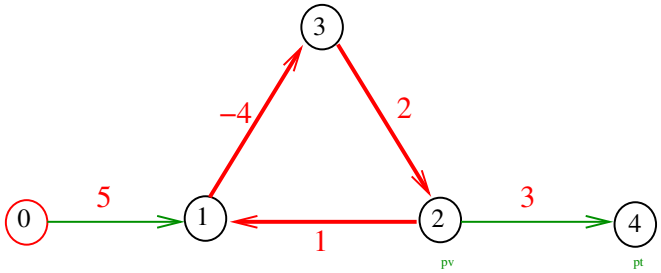


Fato

O algoritmo de Dijkstra não funciona para digrafos com custos negativos, mesmo que o digrafo seja acíclico.

Ciclos negativos

Se o digrafo possui um **ciclo (de custo) negativo** alcançavel a patir de s, então não existe caminho mínimo de s a alguns vértices



Se o digrafo não possui ciclos negativos é possível encontrar caminhos mínimos.

Complexidade computacional

O problema do caminho **simples** de custo mínimo é NP-difícil.

NP-difícil = **não se conhece** algoritmo de consumo de 'tempo polinomial'

Em outras palavras: ninguém conhece um algoritmo eficiente para o problema ...

Se alguém conhece, não contou para ninguém ...

Programação dinâmica

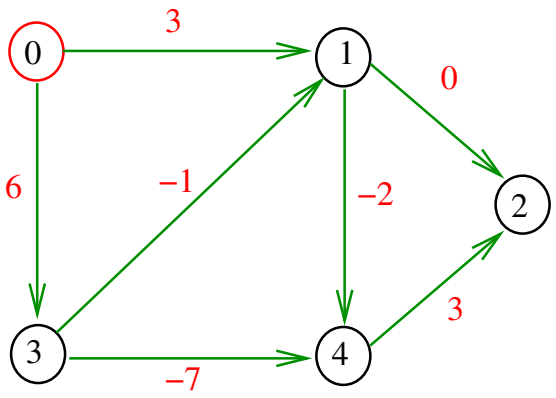
custo[k][w] = menor custo de um caminho de s a w com $\leq$ k arcos.

Recorrência

$$\begin{aligned} \texttt{custo}[0][s] &= 0 \\ \texttt{custo}[0][w] &= \texttt{INFINITO},\ w \neq s \\ \texttt{custo}[k][w] &= \min\{\texttt{custo}[k-1][w], \\ &\qquad \min\{\texttt{custo}[k-1][v] + \texttt{G->adj}[v][w]\}\} \end{aligned}$$

Se o digrafo não tem ciclo negativo acessível a partir de s, então custo[V-1][w] é o menor custo de um caminho de s a w

Exemplo

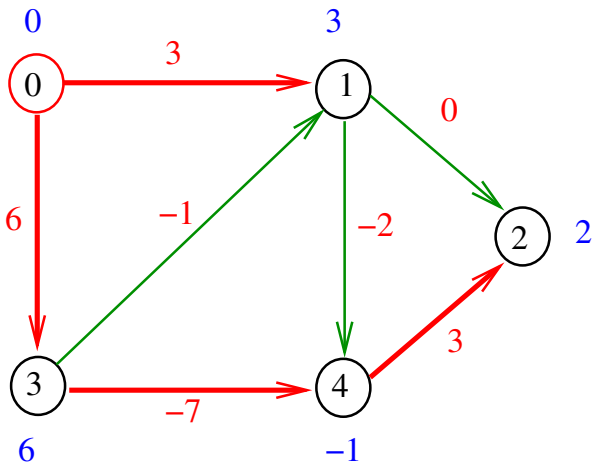


Exemplo

	0	1	2	3	4	v
0	0	*	*	*	*	
1	0	3	*	6	*	
2	0	3	3	6	−1	
3	0	3	2	6	−1	
4	0	3	2	6	−1	

k

Exemplo




```
void bellman_ford1(Digraph G, Vertex s){
 1   Vertex v, w; double d;
 2   for (v=0; v < G->V; v++)
 3       custo[0][v] = INFINITO;
 4   custo[0][s] = 0;
 5   for (k=1; k < G->V; k++)
 6       for (w=0; w < G->V; w++){
 7           custo[k][w] = custo[k-1][w];
 8           for (v=0; v < G->V; v++){
 9               d=custo[k-1][v]+G->adj[v][w];
10               if (custo[k][w] > d)
11                   custo[k][w] = d;
             }
         }
}
```

Consumo de tempo

O consumo de tempo da função bellman_ford1 é $O(V^3)$.

Ciclos negativos

Se custo[k][v] $\neq$ custo[k-1][v], então custo[k][v] é o custo de um caminho de s a v com **exatamente** k arcos.

Se custo[V,v] $\neq$ custo[V-1,v], então G tem um ciclo negativo alcançável a partir de s.

AULA 18

Mais Bellman-Ford

S 21.7

Tabela da programação dinâmica

k	1	2	3	4	5	s	7	8	v
0	*	*	*	*	*	0	*	*	
1						0			
2						0			
3	⋆	⋆	⋆	⋆	⋆	0	⋆	⋆	
4				??		0			
5						0			
6						0			
7						0			

Conclusão

Para implementarmos o algoritmo de Bellman-Ford basta usarmos uma matriz com duas linhas:

*basta usarmos a linha **atual** para calcularmos a **próxima**.*

k	1	2	3	4	5	s	7	8	v
3	⋆	⋆	⋆	⋆	⋆	0	⋆	⋆	
4				??		0			

Na verdade, basta usarmos um **mero vetor**!

```
void bellman_ford2(Digraph G, Vertex s){
 1   Vertex v, w; double d;
 2   for (v=0; v < G->V; v++)
 3       cst[v] = INFINITO;
 4   cst[s] = 0;
 5   for (k=1; /* A */ k < G->V; k++)
 6       for (w=0; w < G->V; w++){
 7           /* cst[w] = cst[w]; */
 8           for (v=0; v < G->V; v++){
 9               d=cst[v]+G->adj[v][w];
10               if (cst[w] > d)
11                   cst[w] = d;
            }
        }
}
```

```
void bellman_ford2(Digraph G, Vertex s){
 1   Vertex v, w; double d;
 2   for (v=0; v < G->V; v++)
 3       cst[v] = INFINITO;
 4   cst[s] = 0;
 5   for (k=1; /* A */ k < G->V; k++)
 6       for (v=0; v < G->V; v++){
 7           for (w=0; w < G->V; w++){
 8               d=cst[v]+G->adj[v][w];
 9               if (cst[w] > d)
10                   cst[w] = d;
            }
        }
}
```

Relação invariante

Em /* A */ na linha vale que

cst[v] $\leq$ custo[k-1][v] = o menor custo de um caminho de s a v com até k-1 arcos

Consumo de tempo

O consumo de tempo da função bellman_ford2 é $O(V^3)$.

Bellman-Ford

```
void bellman_ford3(Digraph G, Vertex s){
1   Vertex v, w;
2   link p;
3   for (v=0; v < G->V; v++){
4       cst[v] = INFINITO;
5       parnt[v] = -1;
    }
6   cst[s] = 0;
```

Bellman-Ford

```
 7   for (k=1; k < G->V; k++)
 8       for (v=0;v < G->V; v++){
 9           p = G->adj[v];
10           while (p != NULL) {
11               w= p->w;
12               if (cst[w] > cst[v]+p->cst){
13                   cst[w]=cst[v]+p->cst;
14                   parnt[w] = v;
                 }
15               p = p->next;
             }
        }
}
```

Consumo de tempo

O consumo de tempo da função bellman_ford3 é $O(VA)$.

FIFO Bellman-Ford

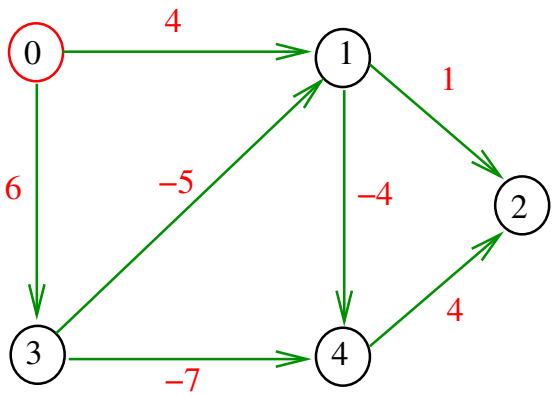
S 21.7

FIFO-Bellman-Ford

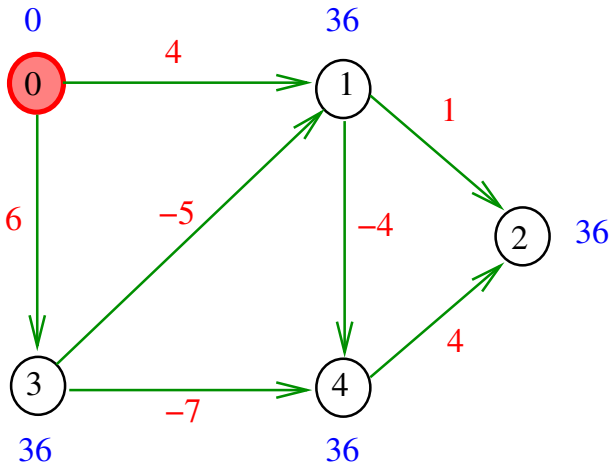
O algoritmo de Bellman e Ford pode ser divido em **passos**

um passo para cada valor de k *(=0,1,2,...).*

FIFO-Bellman-Ford

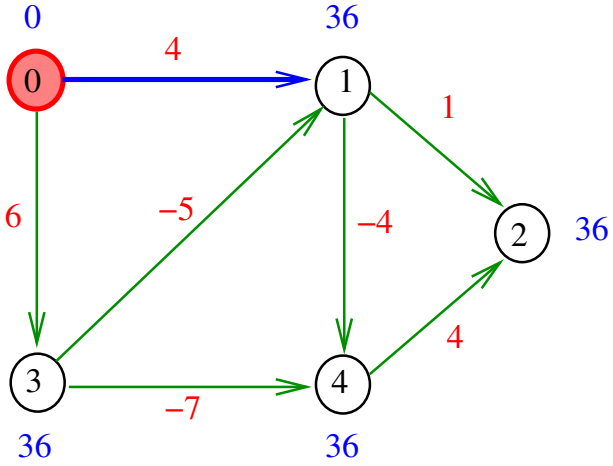


FIFO-Bellman-Ford

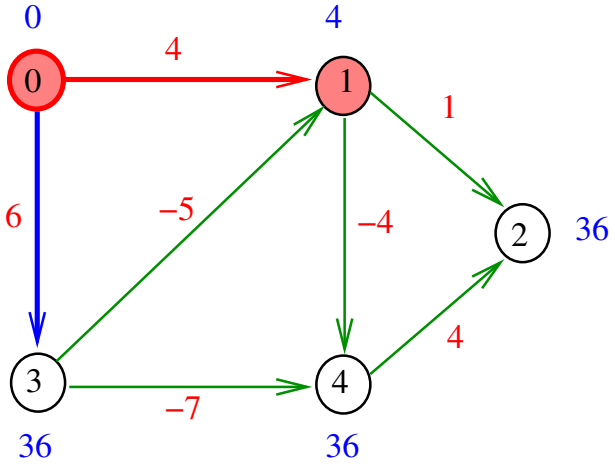


Fim do passo k=0

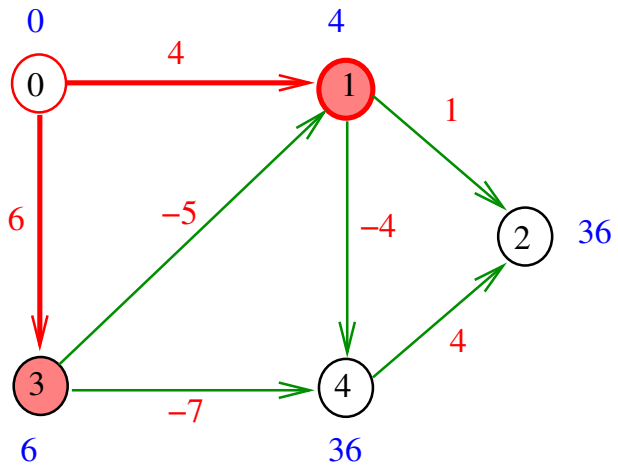
FIFO-Bellman-Ford



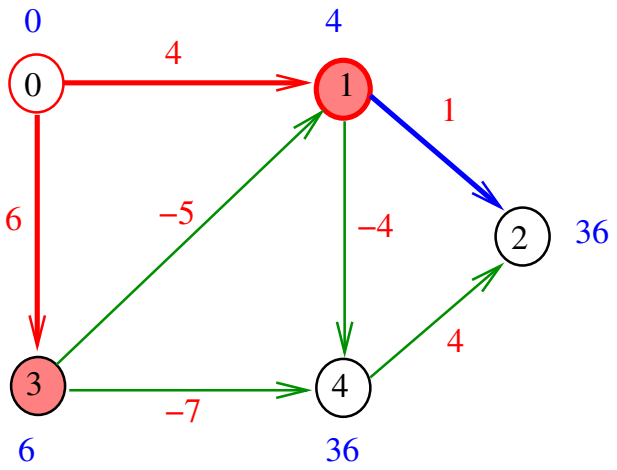
FIFO-Bellman-Ford



FIFO-Bellman-Ford

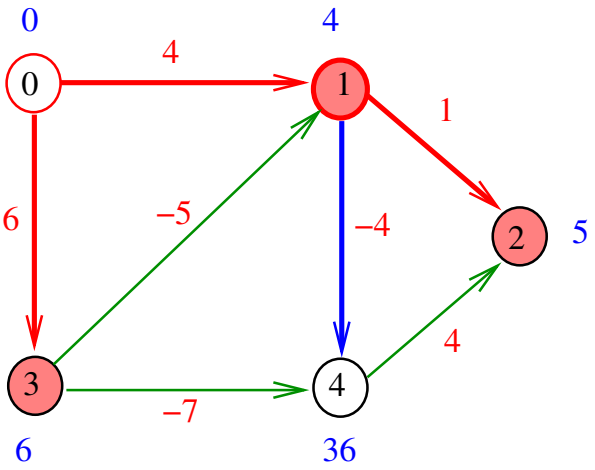


FIFO-Bellman-Ford

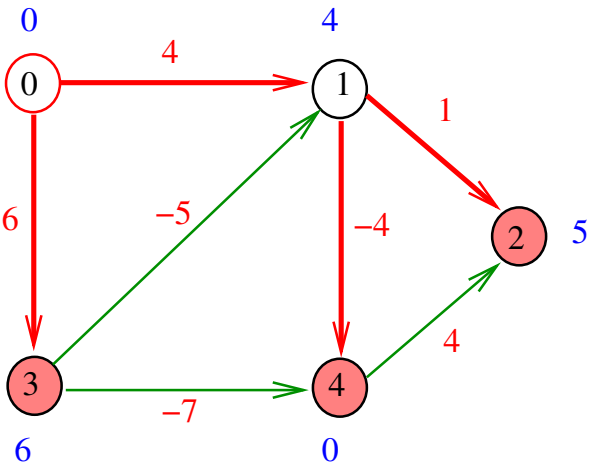


Fim do passo k=1

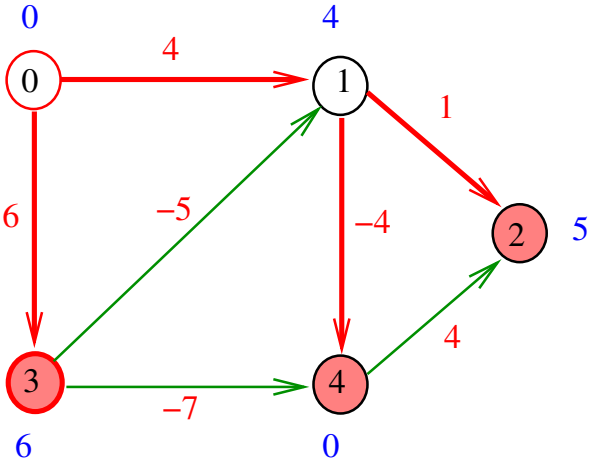
FIFO-Bellman-Ford



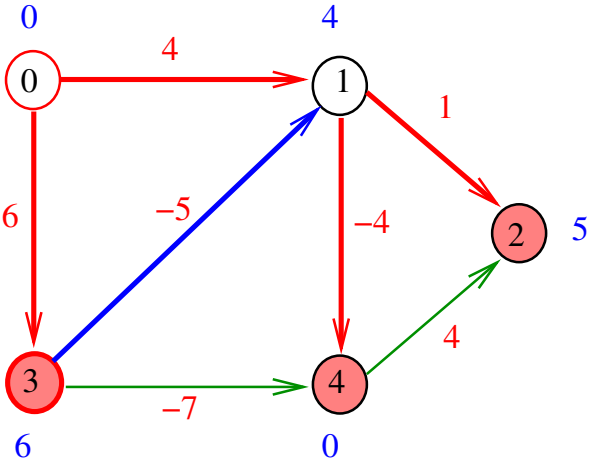
FIFO-Bellman-Ford



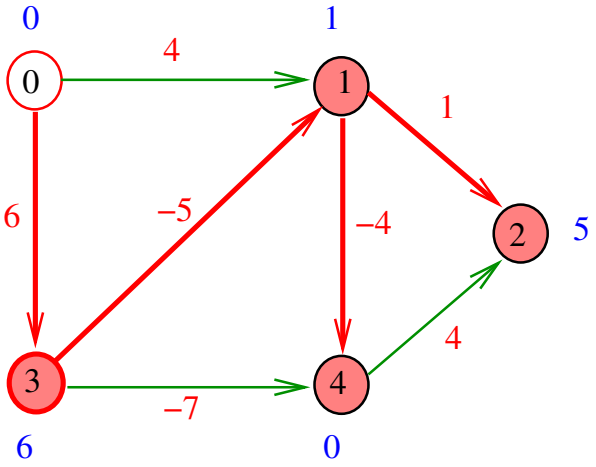
FIFO-Bellman-Ford



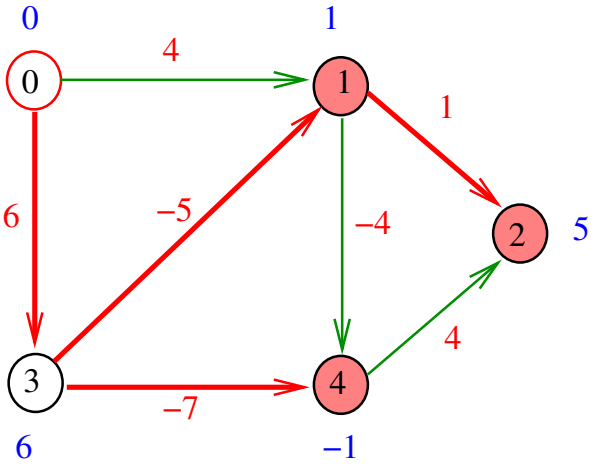
FIFO-Bellman-Ford



FIFO-Bellman-Ford

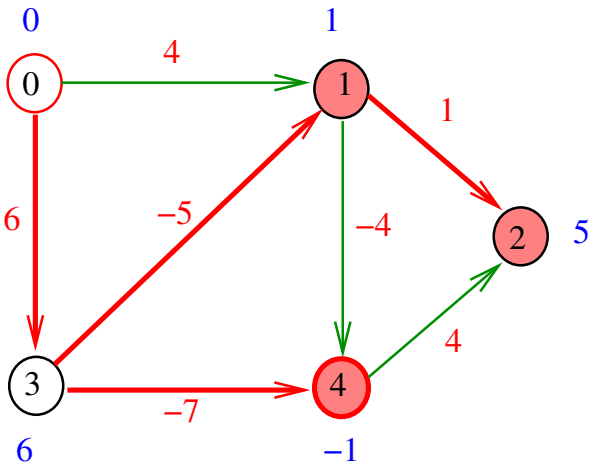


FIFO-Bellman-Ford

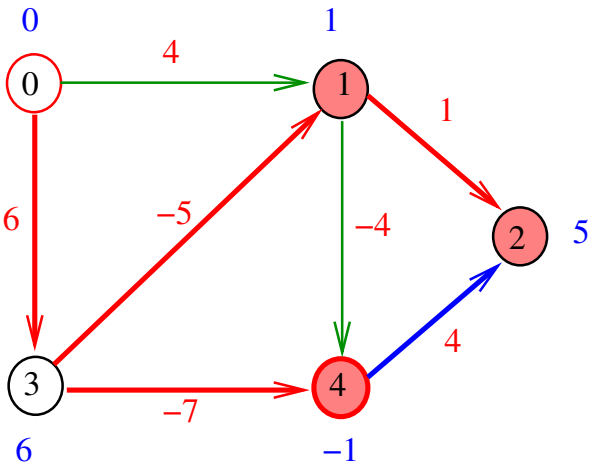


Fim do passo k=2

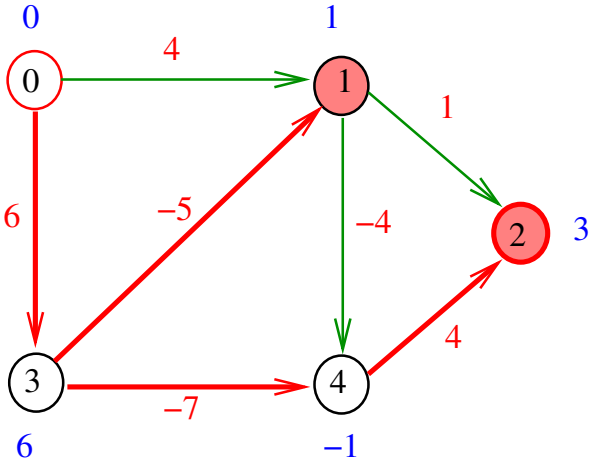
FIFO-Bellman-Ford



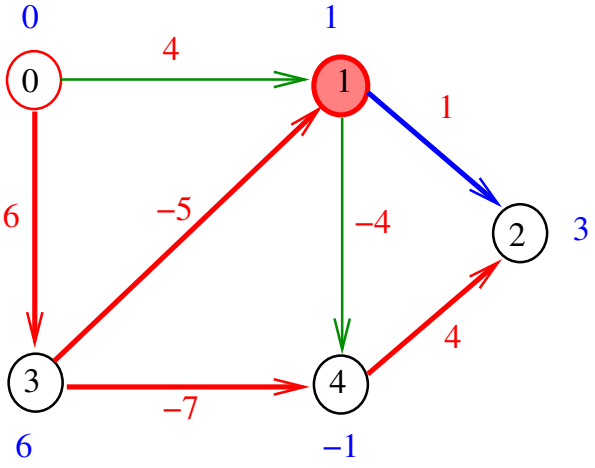
FIFO-Bellman-Ford



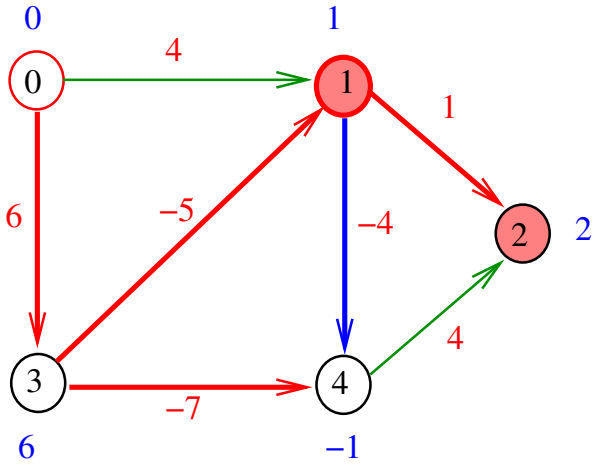
FIFO-Bellman-Ford



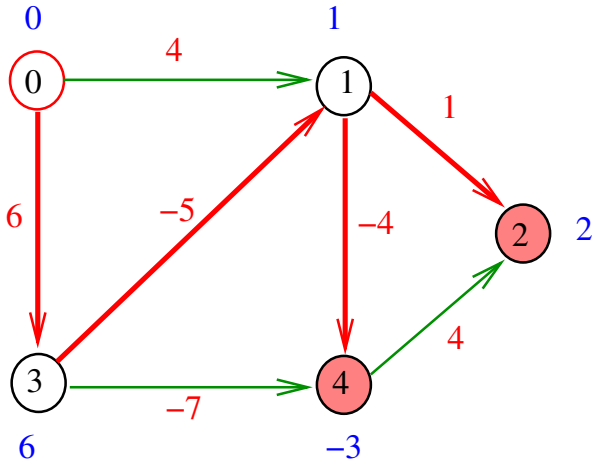
FIFO-Bellman-Ford



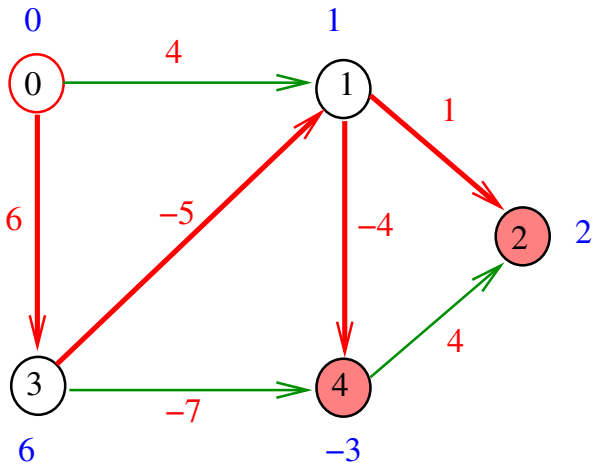
FIFO-Bellman-Ford



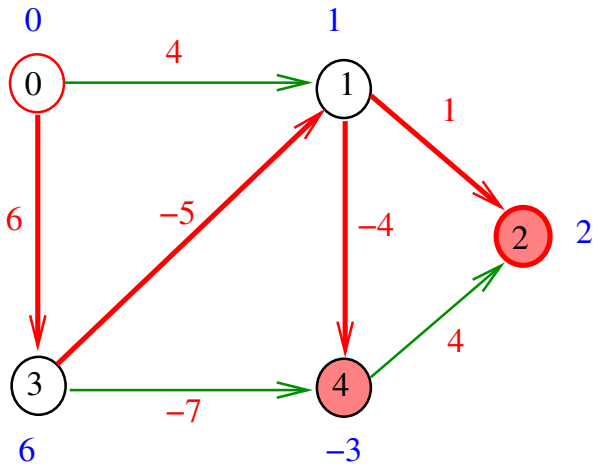
FIFO-Bellman-Ford



FIFO-Bellman-Ford

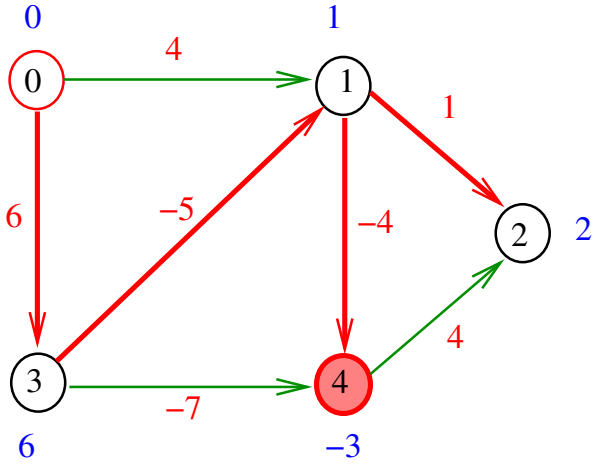


FIFO-Bellman-Ford

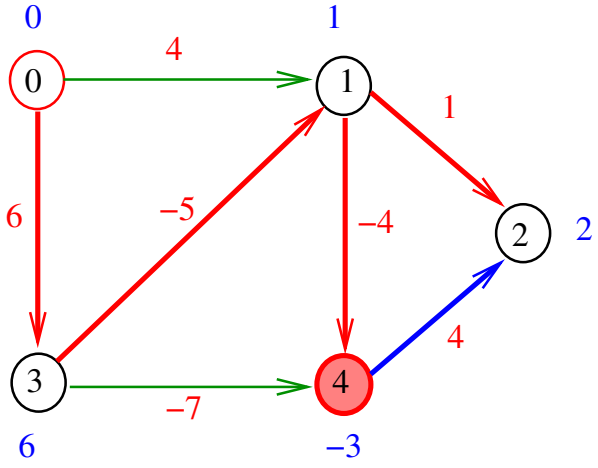


Fim do passo k=3

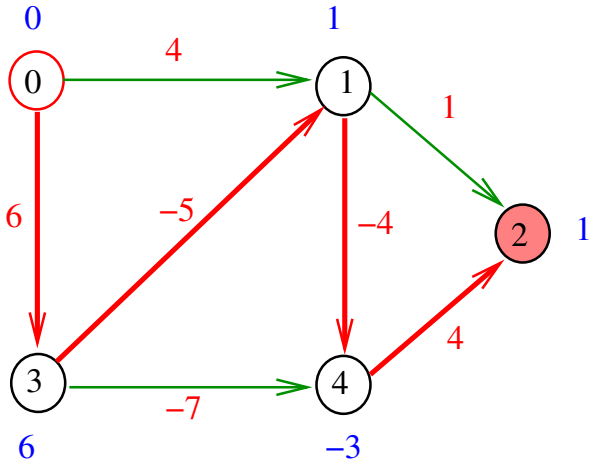
FIFO-Bellman-Ford



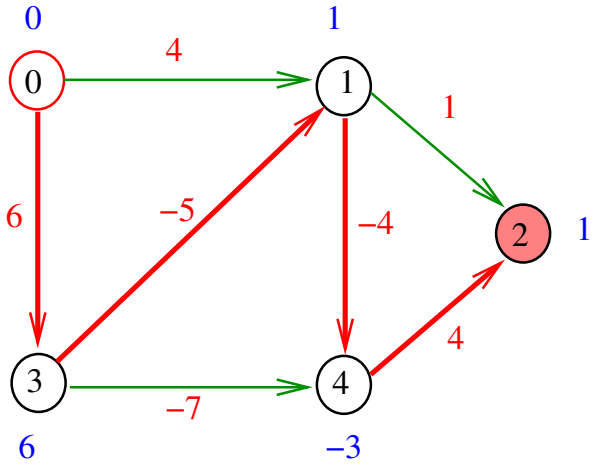
FIFO-Bellman-Ford



FIFO-Bellman-Ford

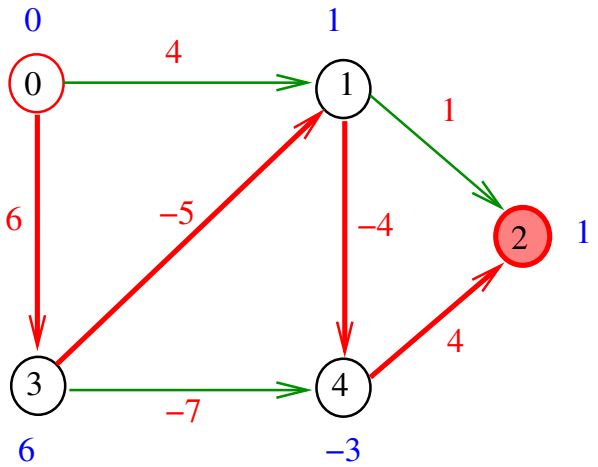


FIFO-Bellman-Ford

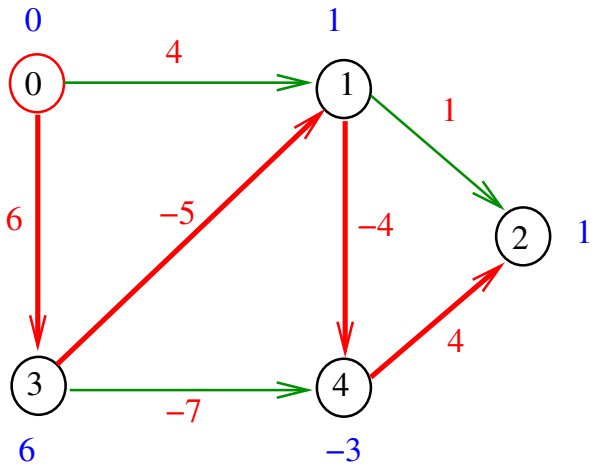


Fim do passo k=4

FIFO-Bellman-Ford



FIFO-Bellman-Ford



Fim do passo k=5

bellman-ford

Recebe digrafo G com custos (possivelmente negativos) nos arcos e um vértice s

Se o digrafo não tem ciclo negativo alcançável a partir de s, calcula uma arborescência de caminhos mínimos com raiz s.

A arborescência é armazenada no vetor parnt

As distâncias em relação a s são armazenadas no vetor cst

bellman-ford

A implementação utiliza uma fila.

Supomos que em cada instante haja no máximo uma cópia de cada vértice na fila:

um vértice deve ser inserido na fila apenas se já não estiver na fila.

```
#define SENTINELA G->V
#define maxV 10000;
double cst[maxV];
Vertex parnt[maxV];
void bellman-ford(Digraph G, Vertex s)
```

bellman-ford

```
void bellman-ford(Digraph G, Vertex s)
{
1   Vertex v, w; link p; int k=0;
2   for (v = 0; v < G->V; v++) {
3       cst[v] = maxCST;
4       parnt[v] = -1;
    }
5   QUEUEinit(G->V);
6   cst[s] = 0;
7   parnt[s] = s;
8   QUEUEput(s); QUEUEput(SENTINELA);
9   while (!QUEUEempty()) {
10      v = QUEUEget();
11      if (v == SENTINELA) {
12          if (k++ == G->V) return;
13          QUEUEput(SENTINELA);
        } else
14      for(p=G->adj[v];p!=NULL;p=p->next)
15          if(cst[w=p->w]>cst[v]+p->cst)
16              cst[w]=cst[v]+p->cst;
17              parnt[w] = v;
18              QUEUEput(w);
        }
}
```

Relação invariante

No início de cada iteração do **while** da linha 9 vale que

$\texttt{cst[v]} \leq \texttt{custo[k][v]} =$ o menor custo de um caminho de s a v com $\leq$ k arcos

Ciclos negativos

```
11      if (v == SENTINELA) {
12          if (k++ == G->V) {
12              if (!QUEUEempty()) {
12              /* tem ciclo negativo */
12              }
12              return;
            }
13          QUEUEput(SENTINELA);
        }
```

O **ciclo negativo** pode ser encontrado no digrafo representado por parnt

Consumo de tempo

linha	número de execuções da linha
2–4	$\Theta(V)$
5	$= 1$ QUEUEinit
6–7	$= 1$
8	$= 2$ QUEUEput
9–10	$O(V^2)$ QUEUEempty e QUEUEget
11	$O(V^2)$
12	$O(V)$
13	$\leq V$ QUEUEput
14–17	$O(VA)$
18	$O(VA)$ QUEUEput
total	$= O(VA) +$???

Conclusão

O consumo de tempo da função bellman-ford é $O(VA)$ mais o consumo de tempo de

1	execução de	QUEUEinit e QUEUEget,
$O(VA)$	execuções de	QUEUEput,
$O(V^2)$	execuções de	QUEUEempty, e
$O(V^2)$	execuções de	QUEUEget

Conclusão

Se implementarmos a fila de tal forma que cada operação consuma tempo constante teremos:

O consumo de tempo da função `bellman_ford` é $O(VA)$.

Conclusão

Para todo grafo digrafo `G` com custo nos arcos e todo par de vértices `s` e `t`, vale uma e apenas umas das seguintes afirmações:

- não existe caminho de `s` a `t`;
- existe um caminho mínimo de `s` a `t`; ou
- existe um caminho de `s` a `t` que contém um ciclo negativo

Ciclos negativos

Problema: Dado um digrafo com custos nos arcos, decidir se o digrafo possui algum ciclo negativo.

Uma adaptação da função `bellman_ford` decide se um dado digrafo com custos nos arcos possui algum ciclo negativo. O consumo de tempo dessa função adaptada é $O(VA)$.