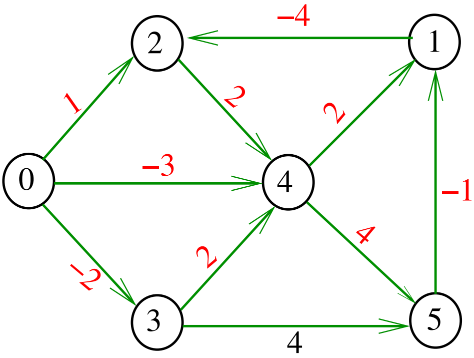
Melhores momentos

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

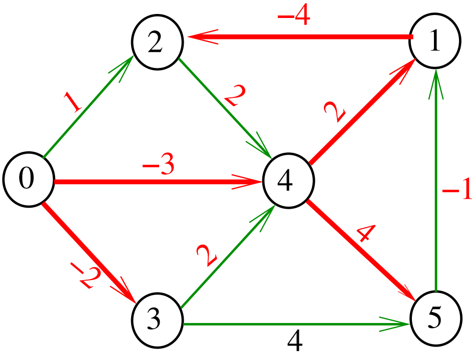
Entra:



Problema da SPT

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

Sai:

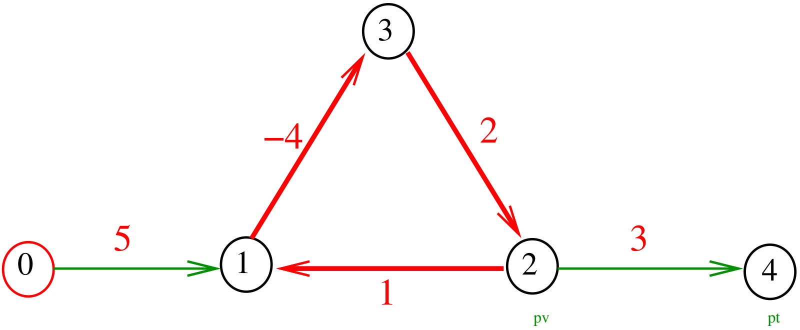


Fato

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

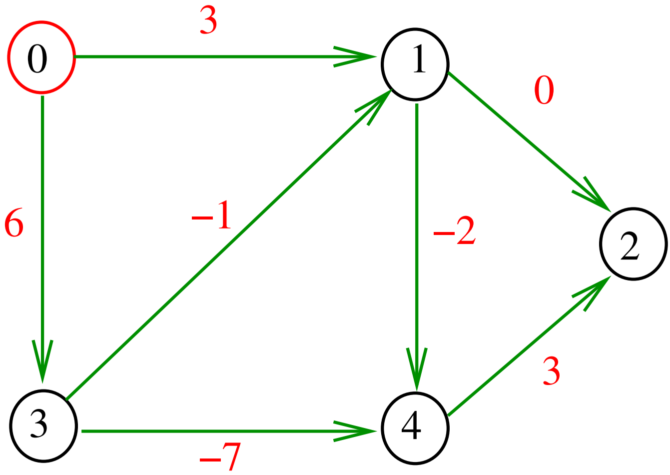
$\texttt{custo}[\texttt{k}][\texttt{w}]$ = menor custo de um caminho de $\texttt{s}$ a $\texttt{w}$ com $\leq$ $\texttt{k}$ arcos.

Recorrência

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

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

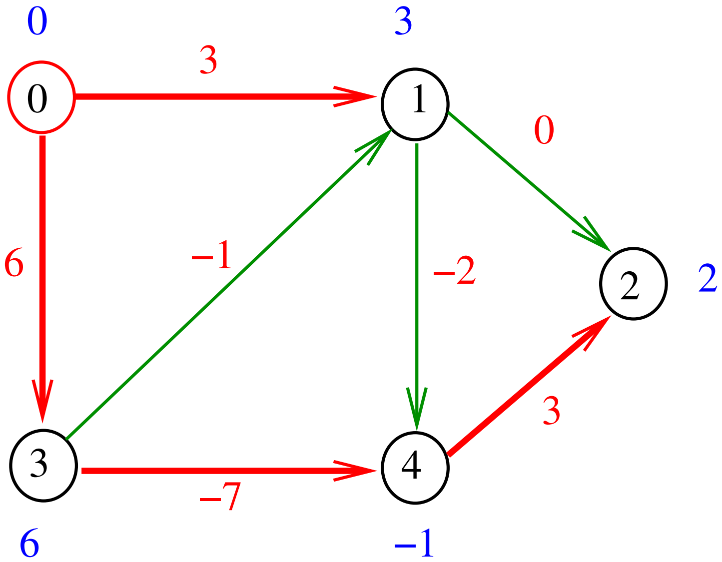
Exemplo



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){
    Vertex v, w; double d;
    for (v=0; v < G->V; v++)
        custo[0][v] = INFINITO;
    custo[0][s] = 0;
    for (k=1; k < G->V; k++)
        for (w=0; w < G->V; w++){
            custo[k][w] = custo[k-1][w];
            for (v=0; v < G->V; v++){
                d=custo[k-1][v]+G->adj[v][w];
                if (custo[k][w] > d)
                    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

	1	2	3	4	5	s	7	8	v
0	*	*	*	*	*	0	*	*	
1						0			
2						0			
3	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$	$\star$	
4				??		0			
5						0			
6						0			
7						0			
k									

Conclusão

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

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

	1	2	3	4	5	s	7	8	v
3	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$	$\star$	
4				??		0			
k									

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

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

```
void bellman_ford2(Digraph G, Vertex s){
    Vertex v, w; double d;
    for (v=0; v < G->V; v++)
        cst[v] = INFINITO;
    cst[s] = 0;
    for (k=1; /* A */ k < G->V; k++)
        for (v=0; v < G->V; v++){
            for (w=0; w < G->V; w++){
                d=cst[v]+G->adj[v][w];
                if (cst[w] > d)
                    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

$$\boxed{\text{O consumo de tempo da função } \texttt{bellman\_ford2} \text{ é } O(V^3).}$$

Bellman-Ford

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

Bellman-Ford

```
for (k=1; k < G->V; k++)
    for (v=0;v < G->V; v++){
        p = G->adj[v];
        while (p != NULL) {
            w= p->w;
            if (cst[w] > cst[v]+p->cst){
                cst[w]=cst[v]+p->cst;
                parnt[w] = v;
            }
            p = p->next;
        }
    }
}
```

Consumo de tempo

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

FIFO Bellman-Ford

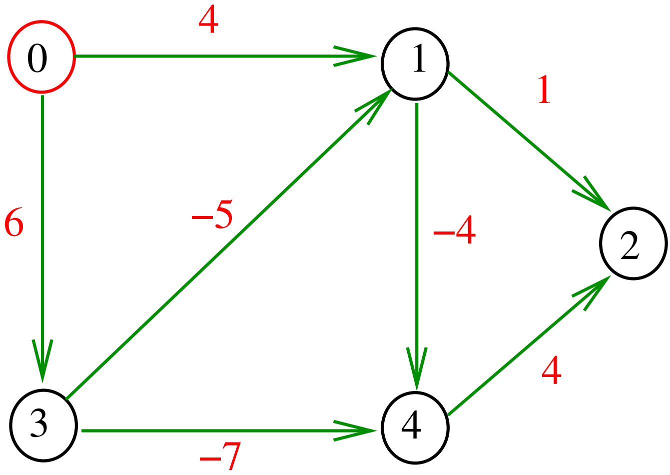
S 21.7

FIFO-Bellman-Ford

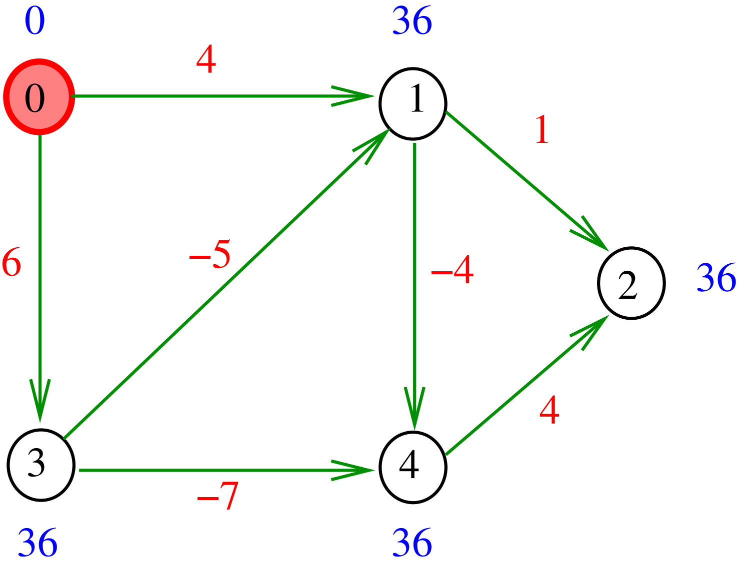
O algoritmo de Bellman e Ford pode ser divido em **passos**

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

FIFO-Bellman-Ford

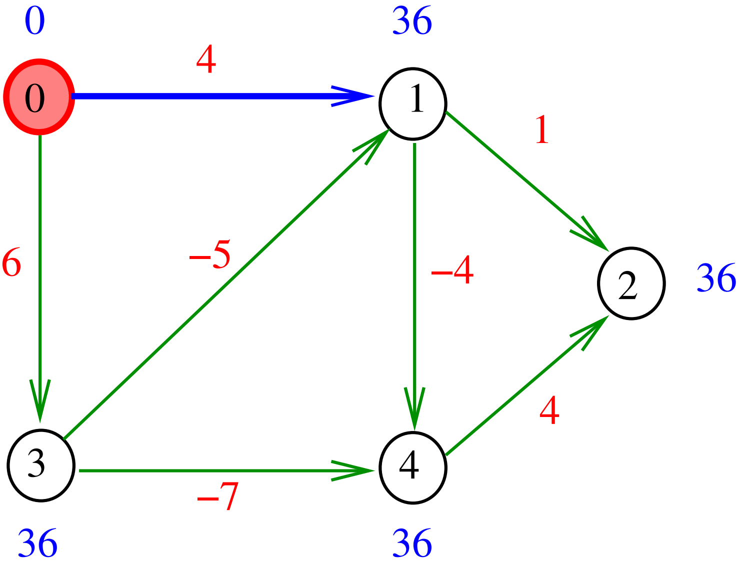


FIFO-Bellman-Ford

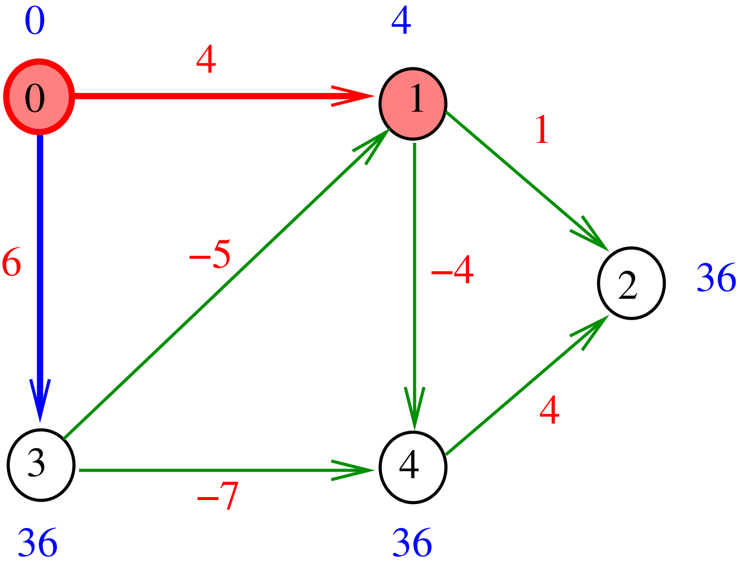


Fim do passo k=0

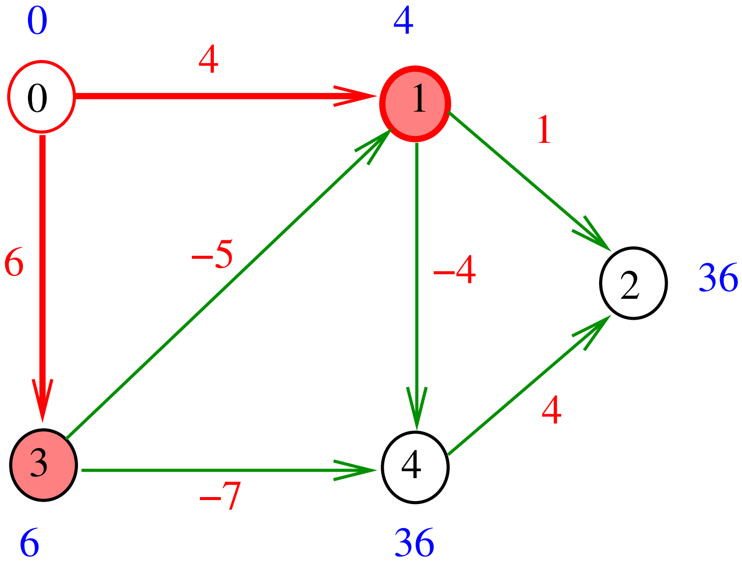
FIFO-Bellman-Ford



FIFO-Bellman-Ford

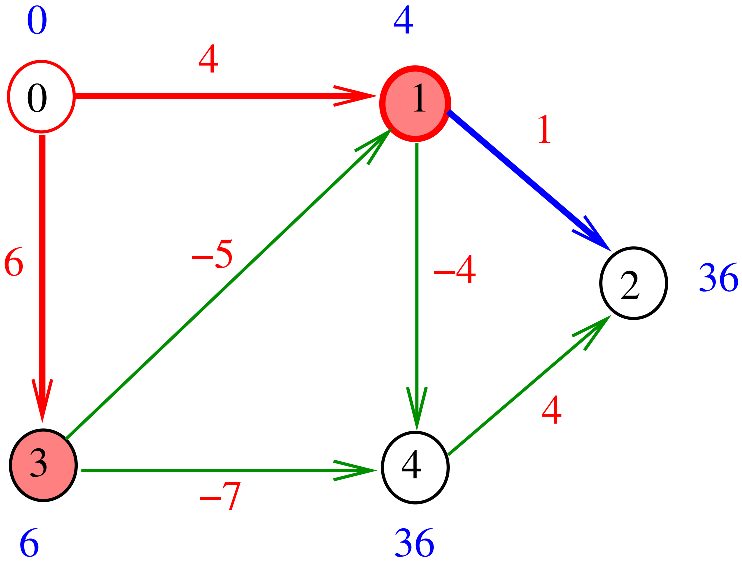


FIFO-Bellman-Ford

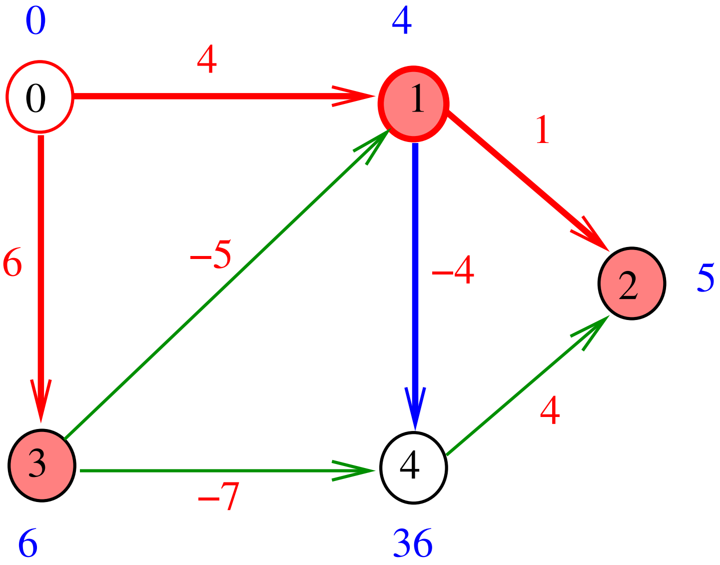


Fim do passo $\texttt{k}$=1

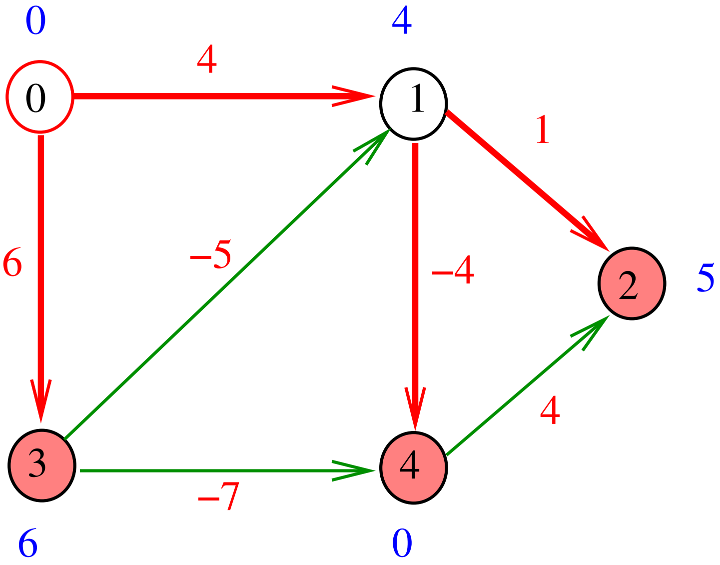
FIFO-Bellman-Ford



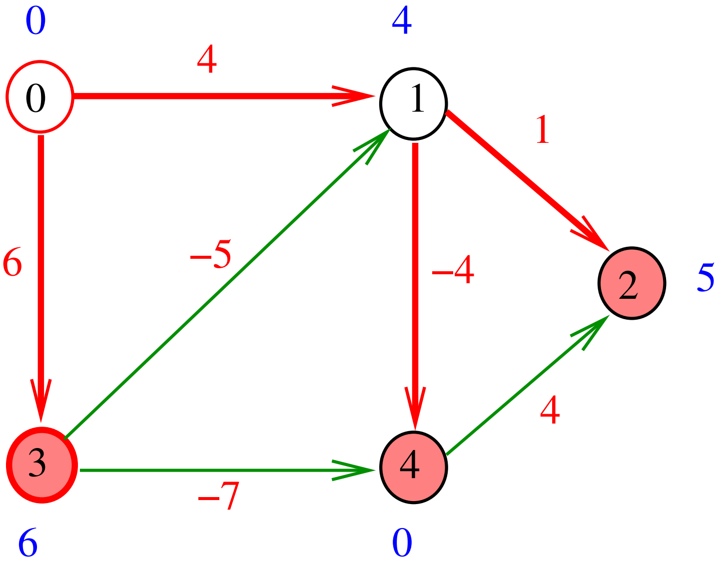
FIFO-Bellman-Ford



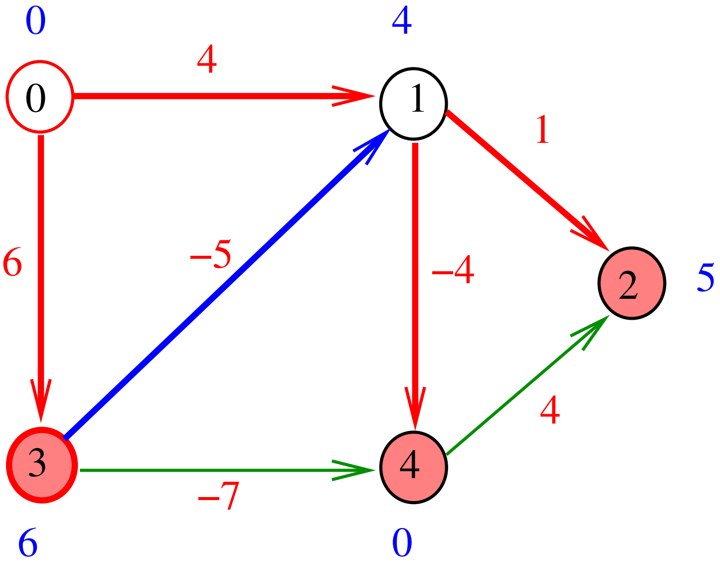
FIFO-Bellman-Ford



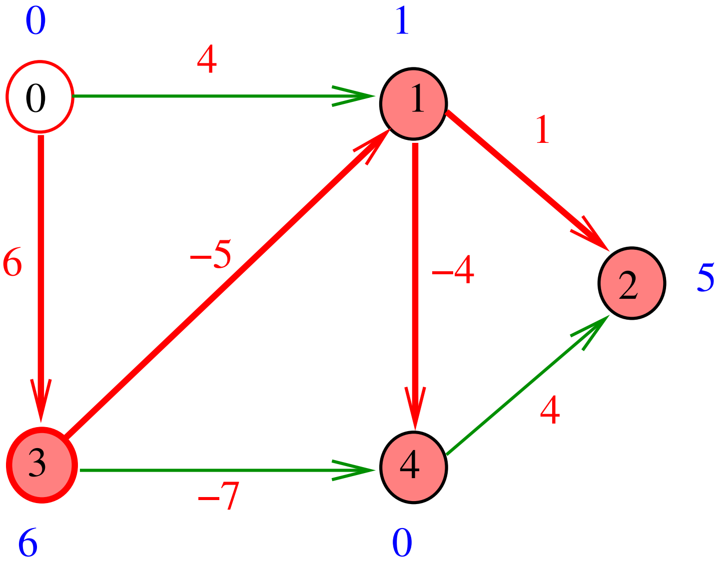
FIFO-Bellman-Ford



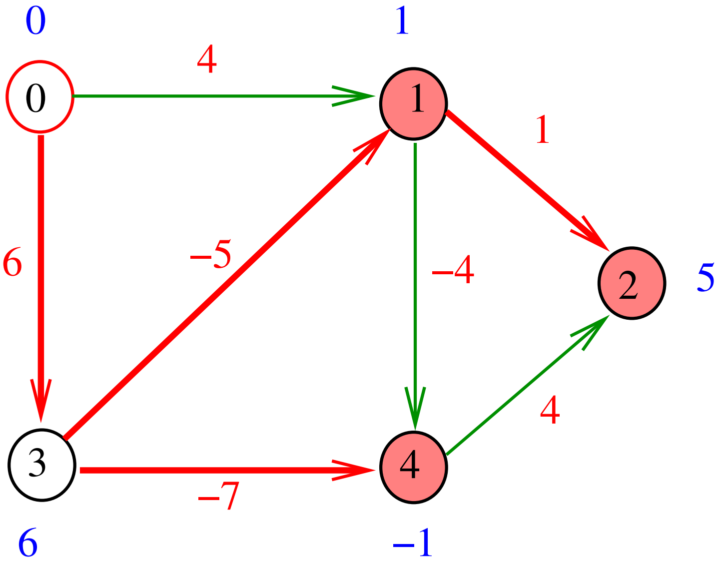
FIFO-Bellman-Ford



FIFO-Bellman-Ford

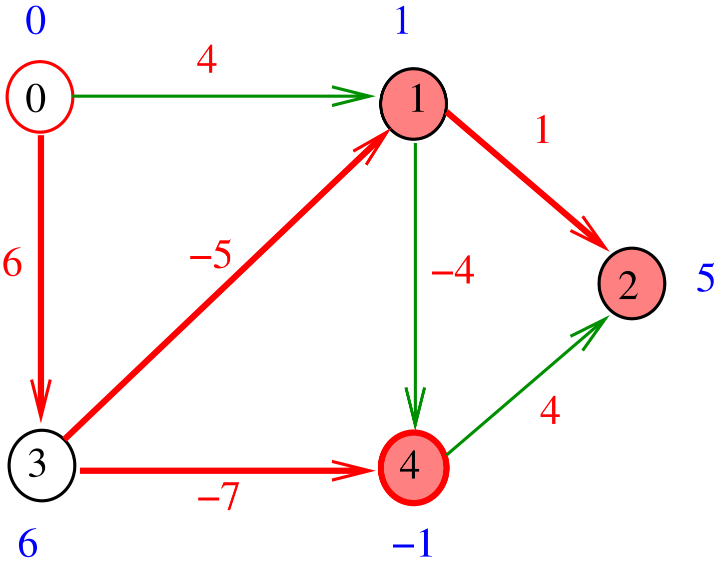


FIFO-Bellman-Ford

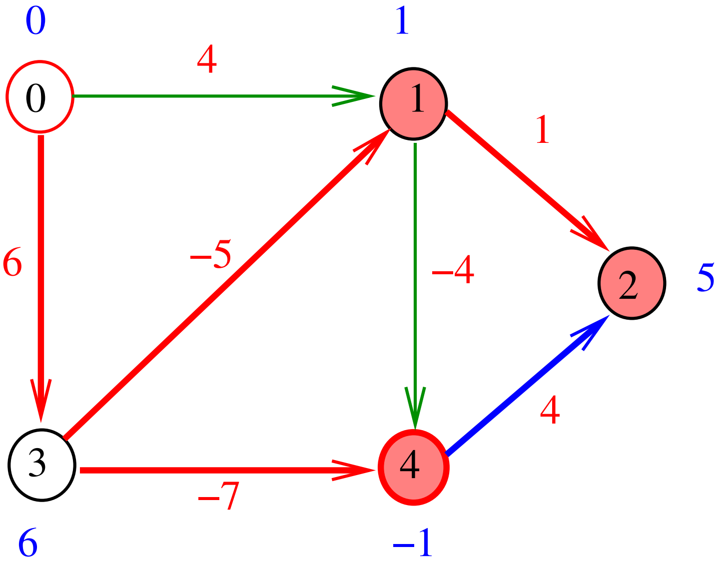


Fim do passo k=2

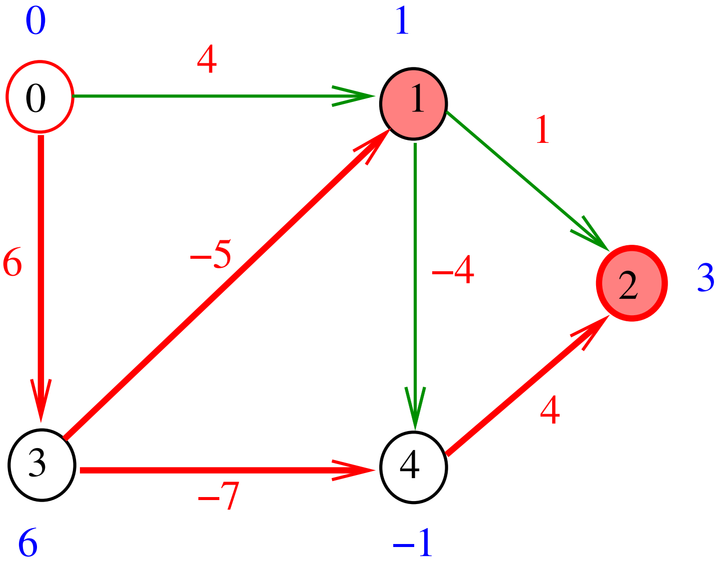
FIFO-Bellman-Ford



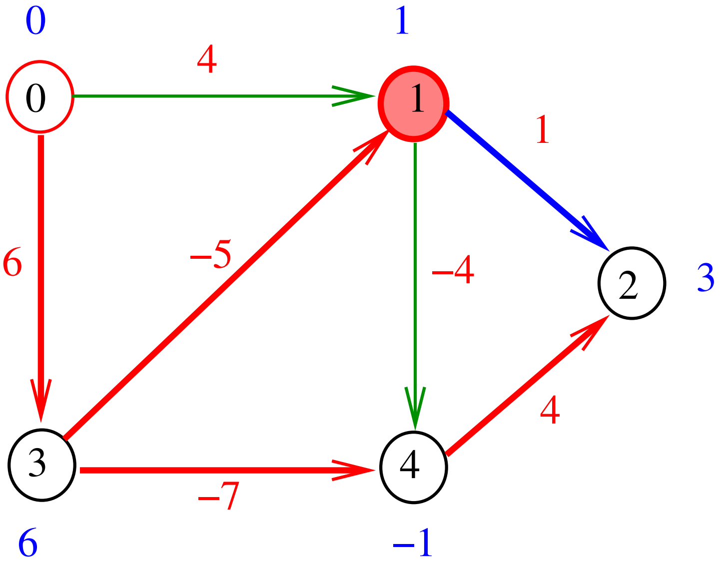
FIFO-Bellman-Ford



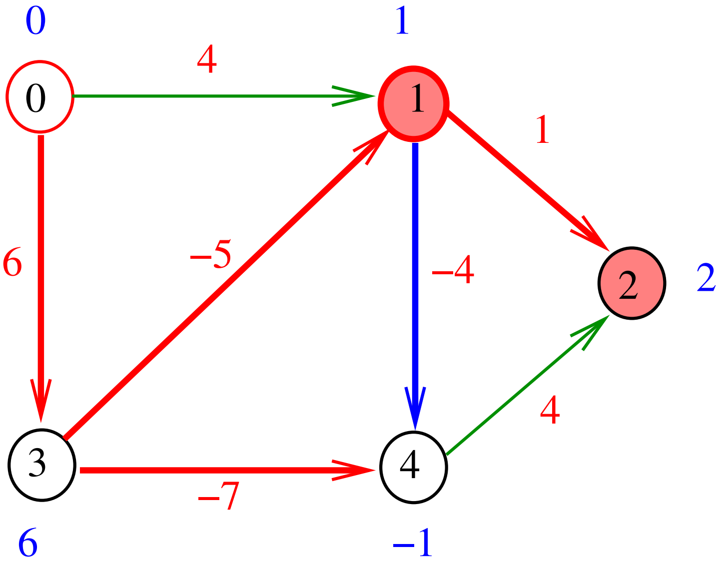
FIFO-Bellman-Ford



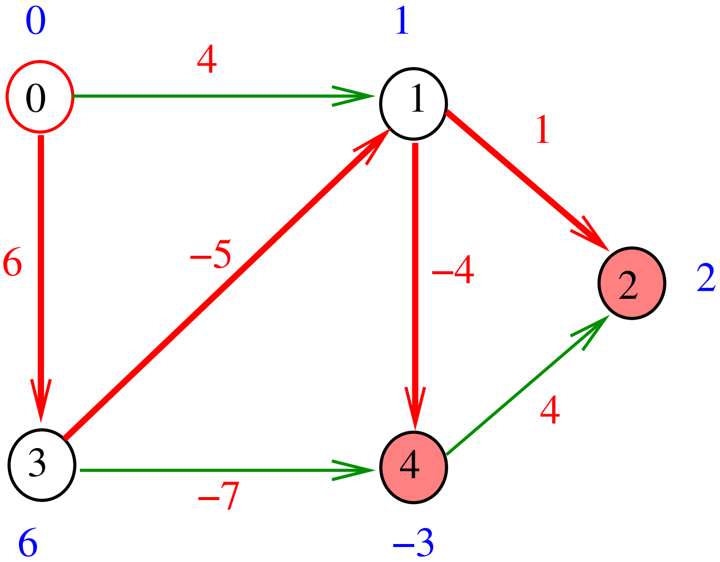
FIFO-Bellman-Ford



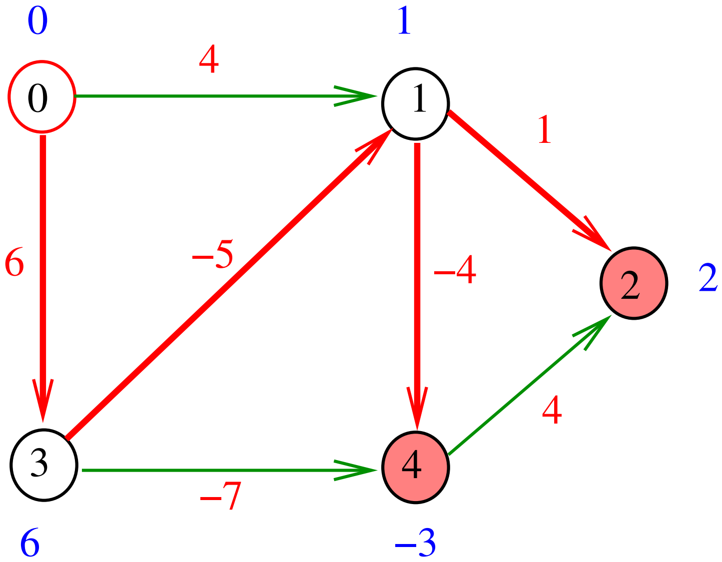
FIFO-Bellman-Ford



FIFO-Bellman-Ford

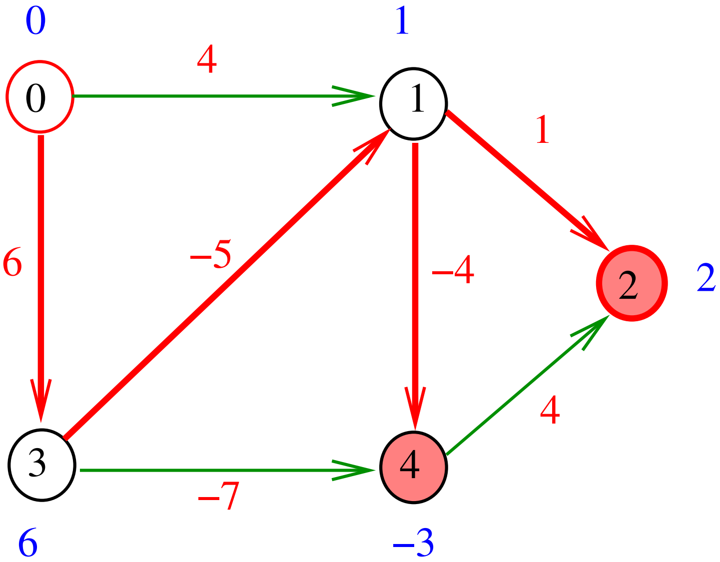


FIFO-Bellman-Ford

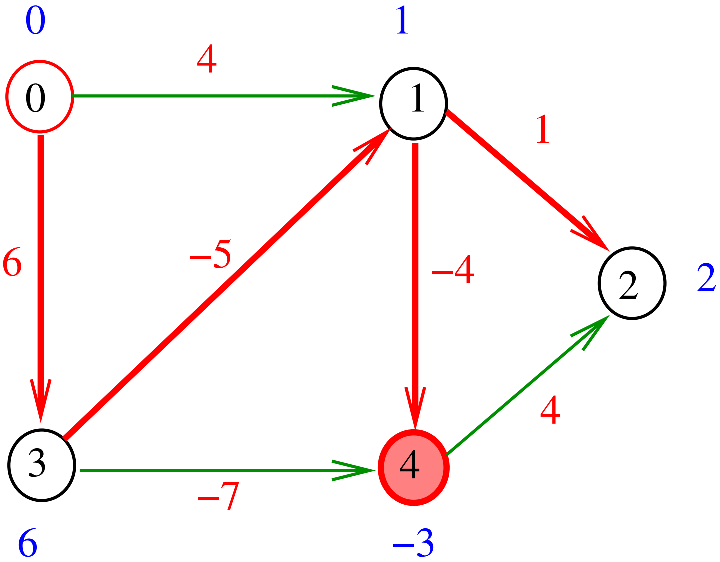


Fim do passo k=3

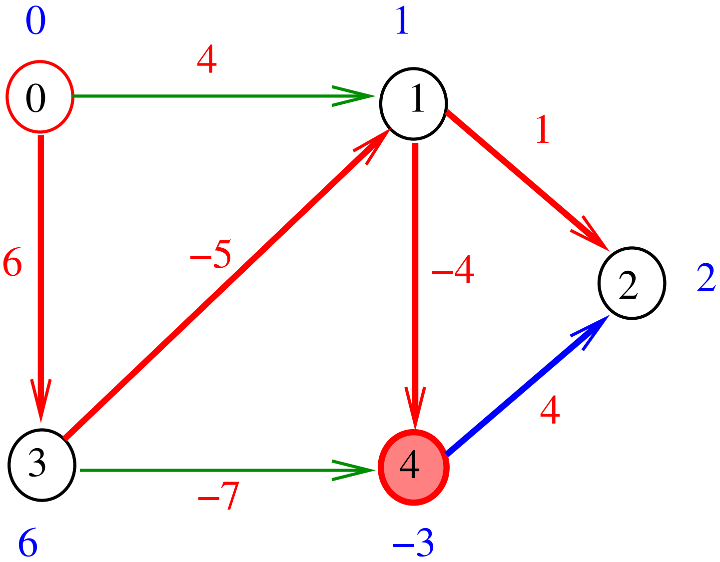
FIFO-Bellman-Ford



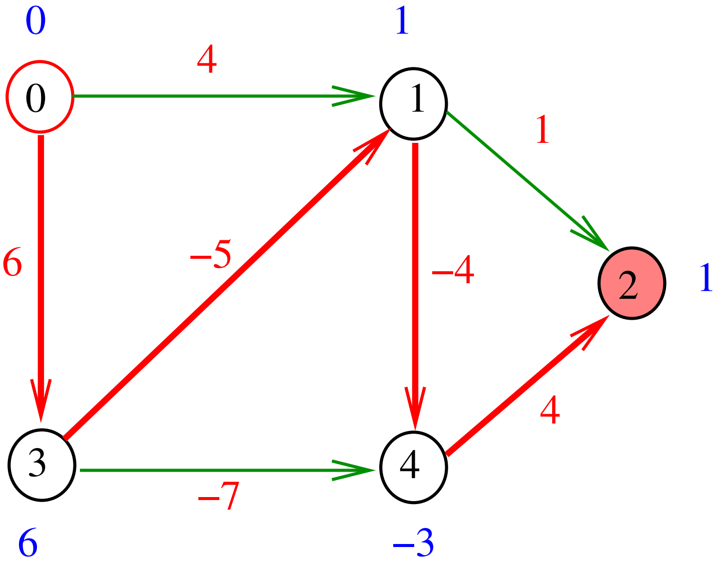
FIFO-Bellman-Ford



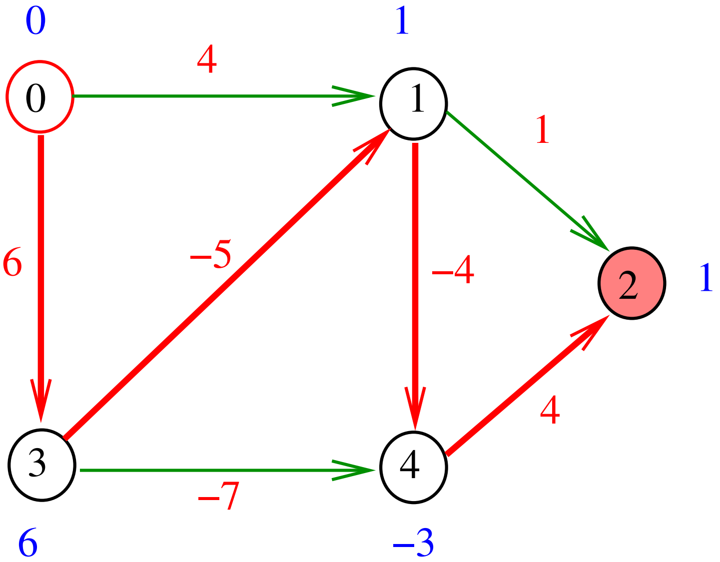
FIFO-Bellman-Ford



FIFO-Bellman-Ford

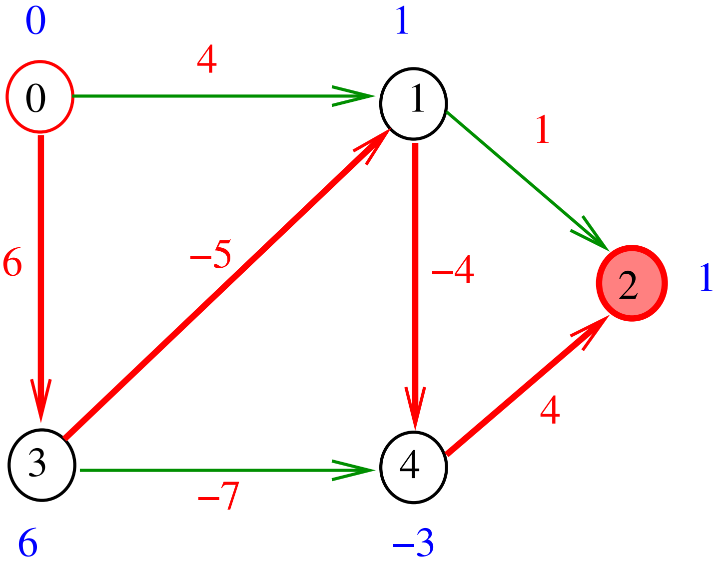


FIFO-Bellman-Ford

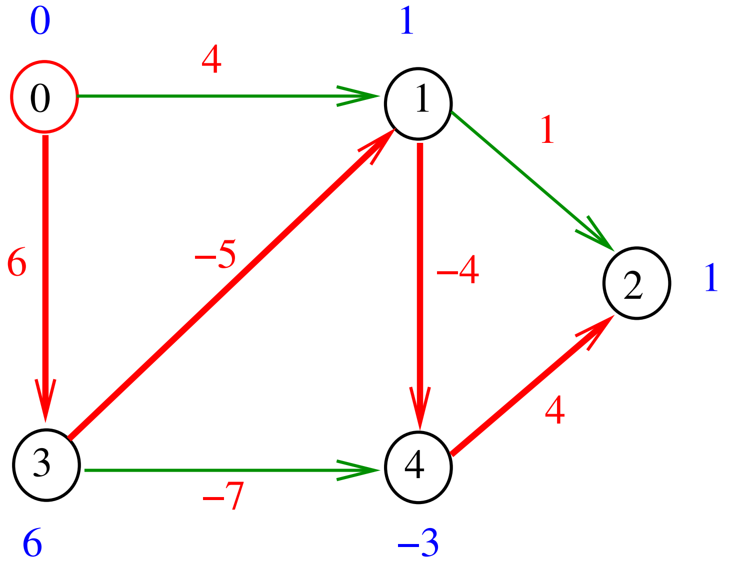


Fim do passo $\mathtt{k}$=4

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)
{
    Vertex v, w; link p; int k=0;
    for (v = 0; v < G->V; v++) {
        cst[v] = maxCST;
        parnt[v] = -1;
    }
    QUEUEinit(G->V);
    cst[s] = 0;
    parnt[s] = s;
    QUEUEput(s); QUEUEput(SENTINELA);
```

```
while (!QUEUEempty()) {
    v = QUEUEget();
    if (v == SENTINELA) {
        if (k++ == G->V) return;
        QUEUEput(SENTINELA);
    } else
    for(p=G->adj[v];p!=NULL;p=p->next)
        if(cst[w=p->w]>cst[v]+p->cst)
            cst[w]=cst[v]+p->cst;
            parnt[w] = v;
            QUEUEput(w);
        }
}
}
```

Relação invariante

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

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

Ciclos negativos

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

$\boxed{\text{O consumo de tempo da função } \texttt{bellman\_ford} \text{ é } O(VA).}$

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 $O(VA)$.