

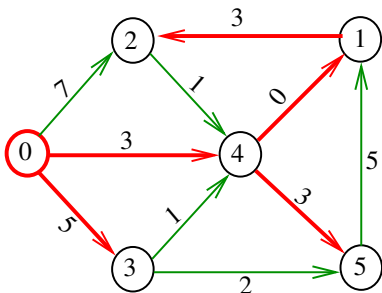
Melhores momentos

AULA 15

Problema

O algoritmo de Dijkstra resolve o problema da SPT:

Dado um vértice s de um digrafo com custos *não-negativos* nos arcos, encontrar uma SPT com raiz s



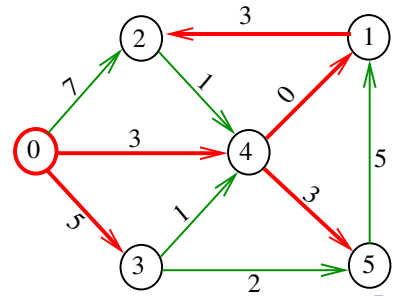
Conclusão

O consumo de tempo da função `dijkstra` é $O(V + A)$ mais o consumo de tempo de

- 1 execução de `PQinit` e `PQfree`,
- $O(V)$ execuções de `PQinsert`,
- $O(V)$ execuções de `PQempty`,
- $O(V)$ execuções de `PQdelmin`, e
- $O(A)$ execuções de `PQdec`.

Arborescência de caminhos mínimos

Uma arborescência com raiz s é de **caminhos mínimos** (= *shortest-paths tree* = SPT) se para todo vértice t que pode ser alcançado a partir de s , o único caminho de s a t na arborescência é um caminho simples mínimo



dijkstra

Recebe digrafo G com custos *não-negativos* nos arcos e um vértice 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`

void

```
dijkstra(Digraph G, Vertex s,  
Vertex parnt[], double cst[]);
```

Consumo de tempo Min-Heap

	heap	d -heap	fibonacci heap
<code>PQinsert</code>	$O(\lg V)$	$O(\log_D V)$	$O(1)$
<code>PQdelmin</code>	$O(\lg V)$	$O(\log_D V)$	$O(\lg V)$
<code>PQdec</code>	$O(\lg V)$	$O(\log_D V)$	$O(1)$
<code>dijkstra</code>	$O(A \lg V)$	$O(A \log_D V)$	$O(A + V \lg V)$

Conclusão

O consumo de tempo da função `dijkstra` implementada com um min-heap é $O(\lg V)$.

AULA 16

Dijkstra para digrafos esparsos

S 21.1 e 21.2

Outra implementação para digrafos esparsos

```
void
DIGRAPHsptD3 (Digraph G, Vertex s, Vertex
              parnt[], double cst[]) {
1  Vertex v, w; link p;
2  PQinit();
3  for (v = 0; v < G->V; v++) {
4      parnt[v] = -1;
5      cst[v] = INFINITO;
6      PQinsert(v);
7  }
7  cst[s] = 0;
8  parnt[s] = s;
9  PQdec(s);
```

Outra implementação para digrafos esparsos

```
10 while (!PQempty()) {
11     v = PQdelmin();
12     if (cst[v] == INFINITO) break;
13     for (p=G->adj[v]; p!=NULL; p=p->next){
14         w = p->w;
15         if (cst[w] > cst[v] + p->cst){
16             cst[w] = cst[v] + p->cst;
17             parnt[w] = v;
18             PQdec(w);
19         }
20     }
21 }
```

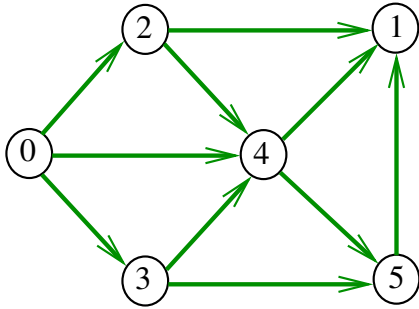
Caminhos mínimos em DAGs

S 19.6

DAGs

Um digrafo é **acíclico** se não tem ciclos
Digrafos acíclicos também são conhecidos como DAGs (= *directed acyclic graphs*)

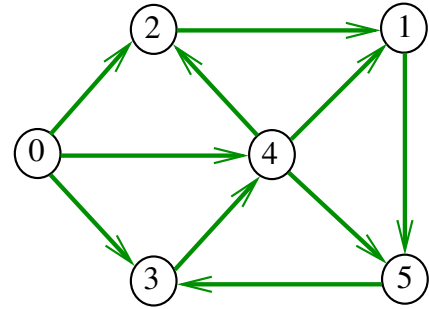
Exemplo: um digrafo acíclico



DAGs

Um digrafo é **acíclico** se não tem ciclos
Digrafos acíclicos também são conhecidos como DAGs (= *directed acyclic graphs*)

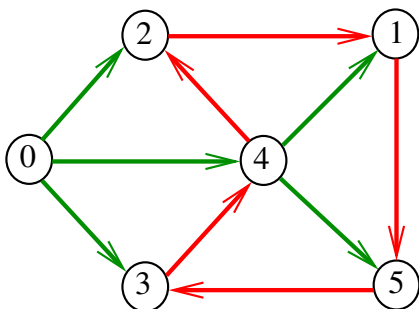
Exemplo: um digrafo que **não** é acíclico



DAGs

Um digrafo é **acíclico** se não tem ciclos
Digrafos acíclicos também são conhecidos como DAGs (= *directed acyclic graphs*)

Exemplo: um digrafo que **não** é acíclico



Ordenação topológica

Uma **permutação** dos vértices de um digrafo é uma seqüência em que cada vértice aparece uma e uma só vez

Uma **ordenação topológica** (= *topological sorting*) de um digrafo é uma permutação

$$ts[0], ts[1], \dots, ts[V-1]$$

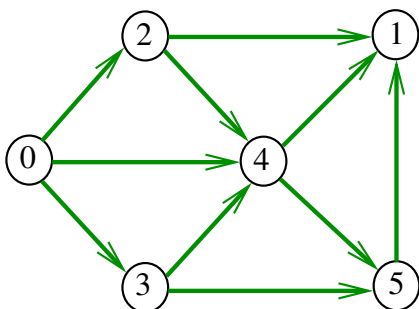
dos seus vértices tal que todo arco tem a forma

$$ts[i] \rightarrow ts[j] \text{ com } i < j$$

$ts[0]$ é necessariamente uma **fonte**
 $ts[V-1]$ é necessariamente um **sorvedouro**

Exemplo

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



Fato

Para todo digrafo G , vale uma e apenas uma das seguintes afirmações:

- ▶ G possui um **ciclo**
- ▶ G é um DAG e, portanto, admite uma **ordenação topológica**

Problema

Problema:

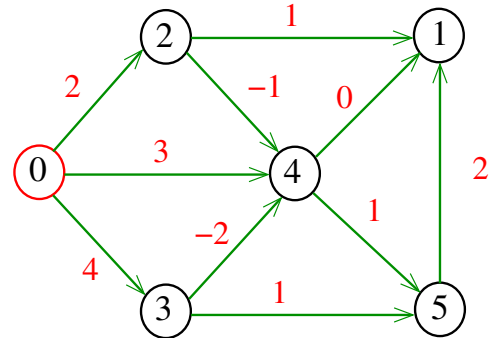
Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar, para cada vértice t que pode ser alcançado a partir de s , um **caminho simples mínimo** de s a t

Problema:

Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar uma **SPT** com raiz s

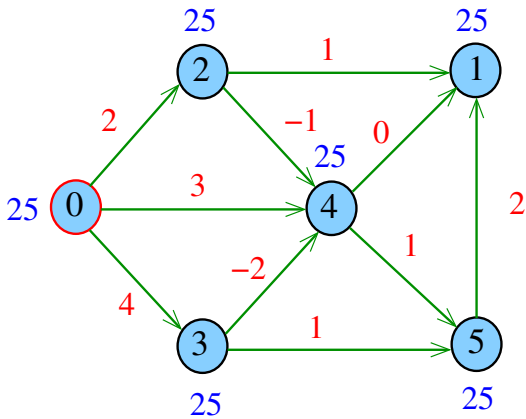
Simulação

i	0	1	2	3	4	5
$ts[i]$	0	3	2	4	5	1



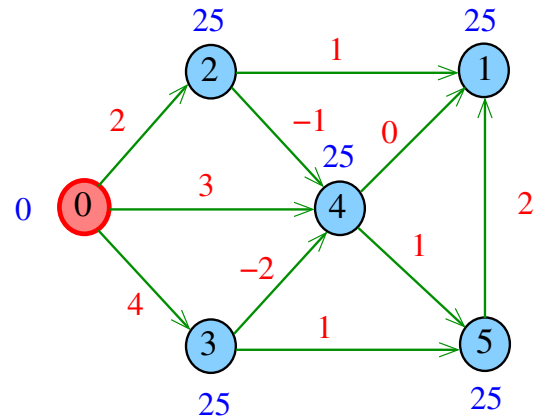
Simulação

i	0	1	2	3	4	5
$ts[i]$	0	3	2	4	5	1



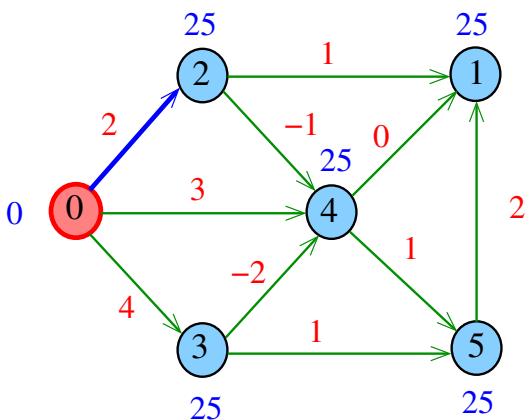
Simulação

i	0	1	2	3	4	5
$ts[i]$	0	3	2	4	5	1



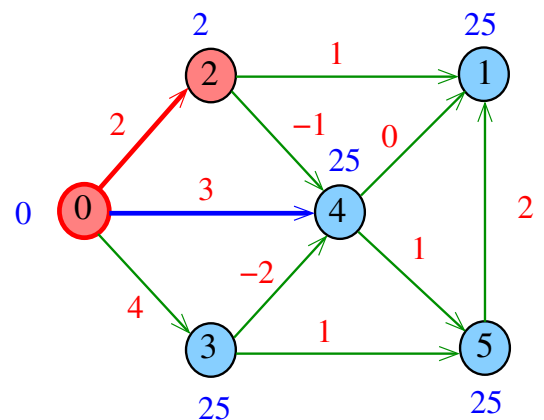
Simulação

i	0	1	2	3	4	5
$ts[i]$	0	3	2	4	5	1



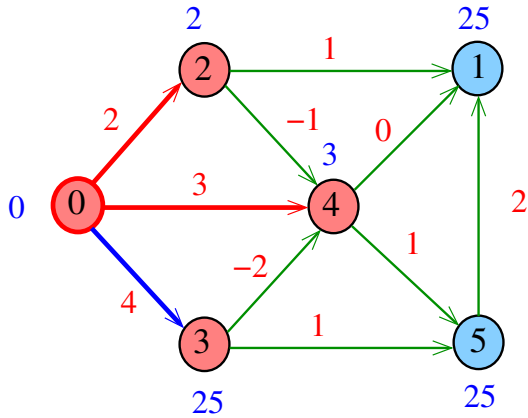
Simulação

i	0	1	2	3	4	5
$ts[i]$	0	3	2	4	5	1



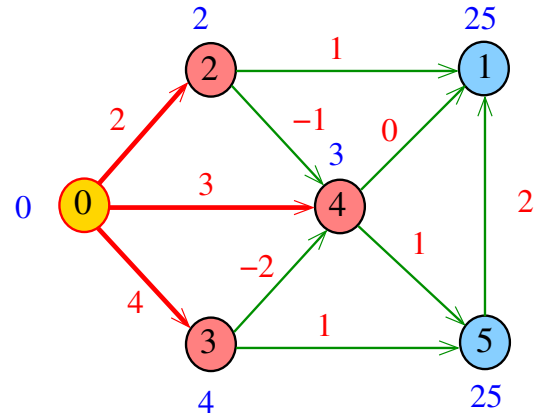
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



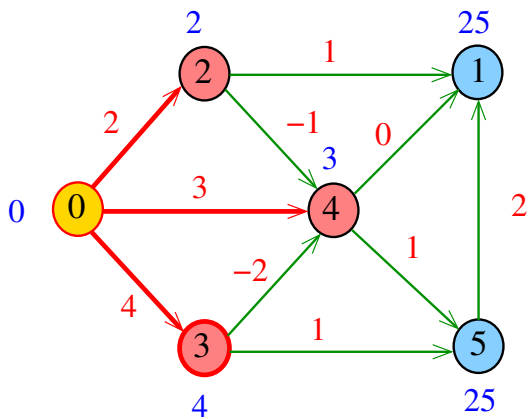
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



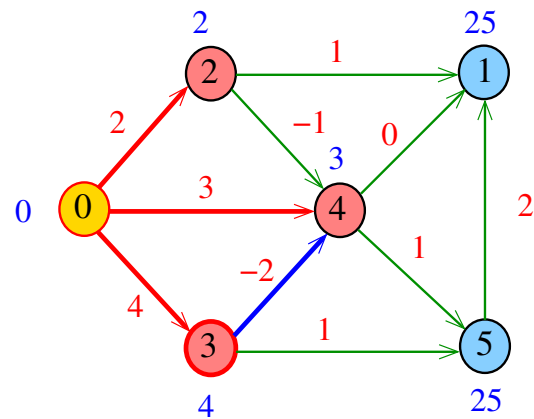
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



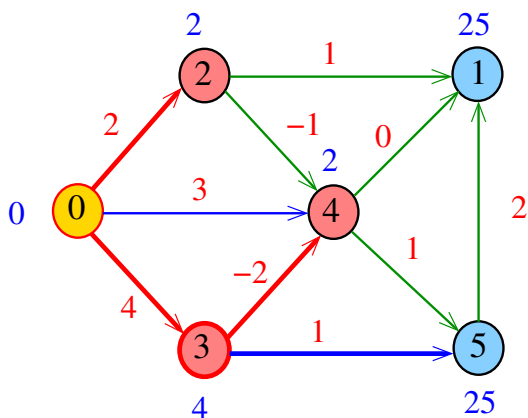
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



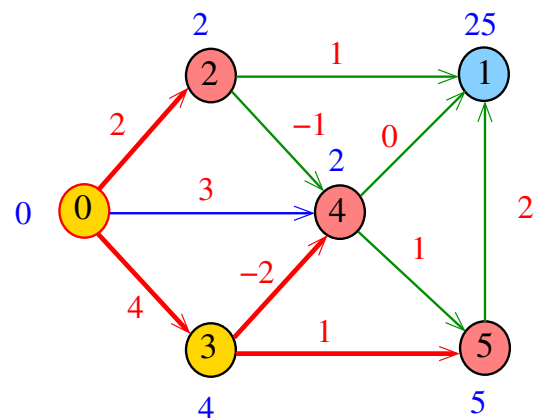
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



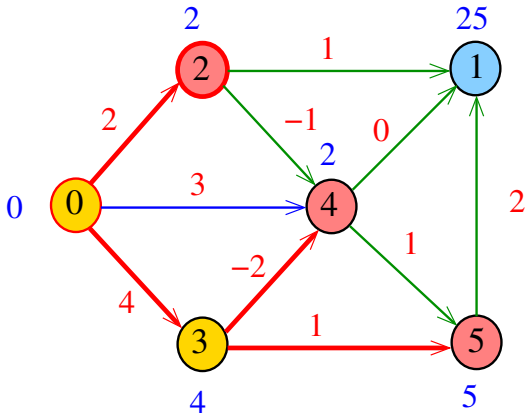
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



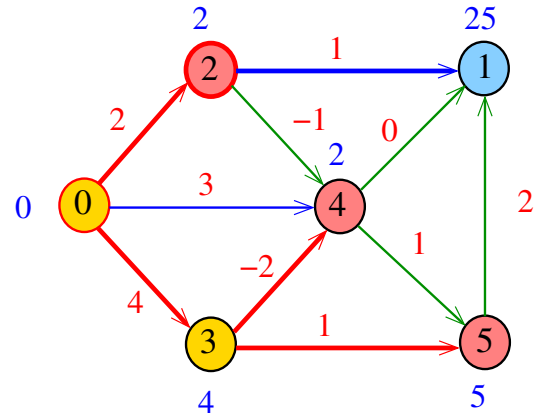
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



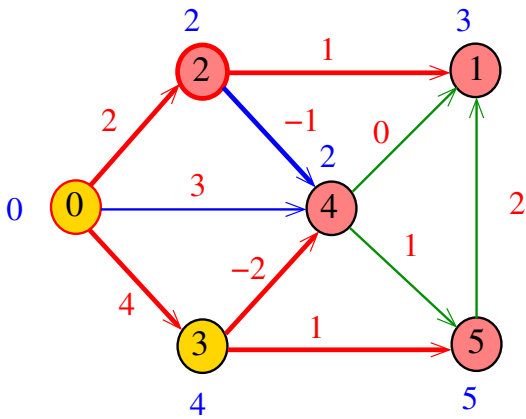
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



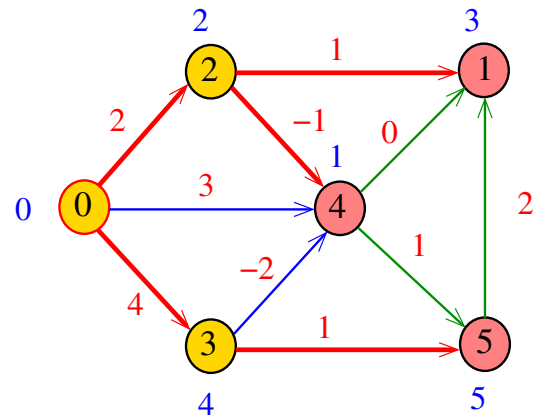
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



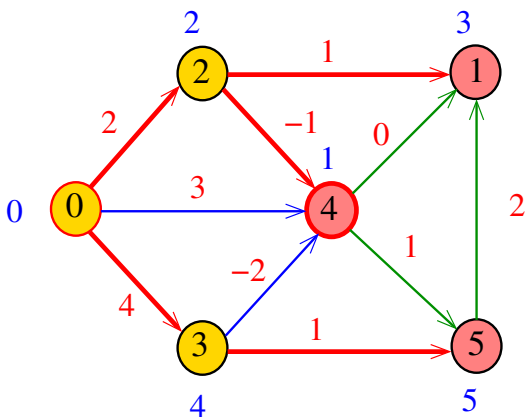
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



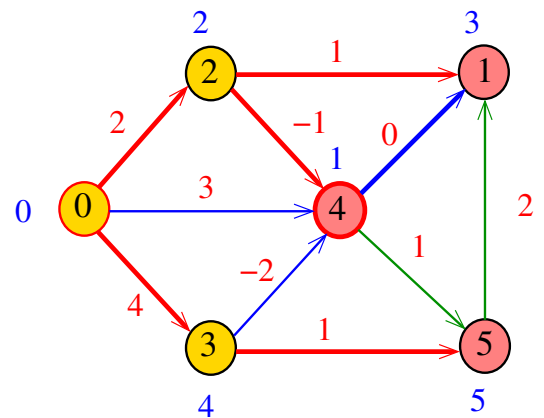
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



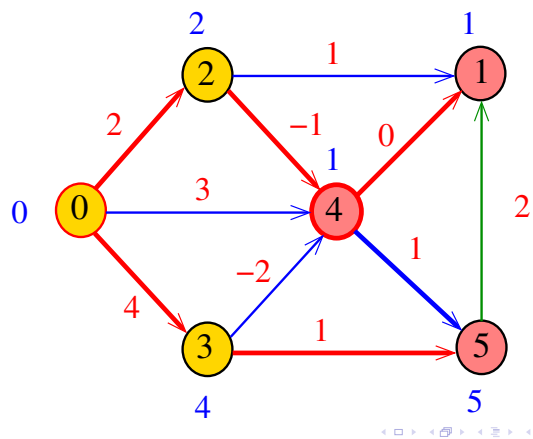
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



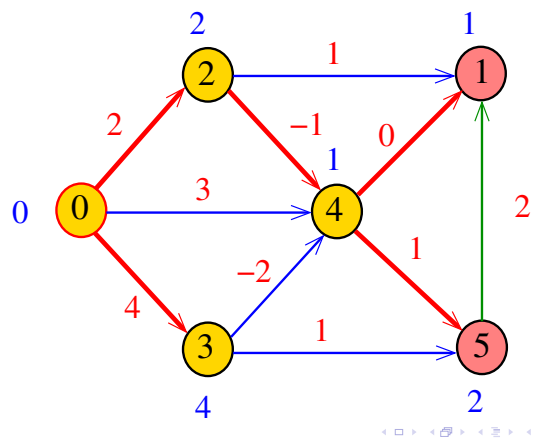
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



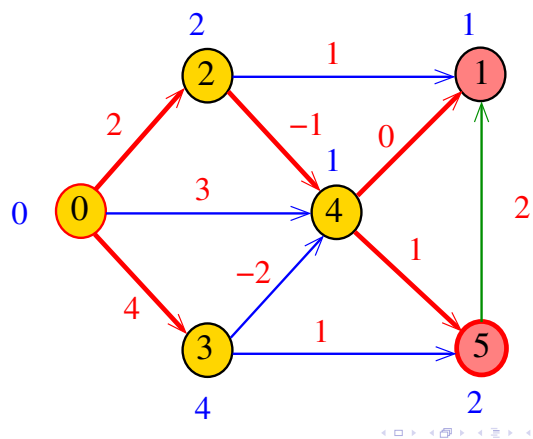
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



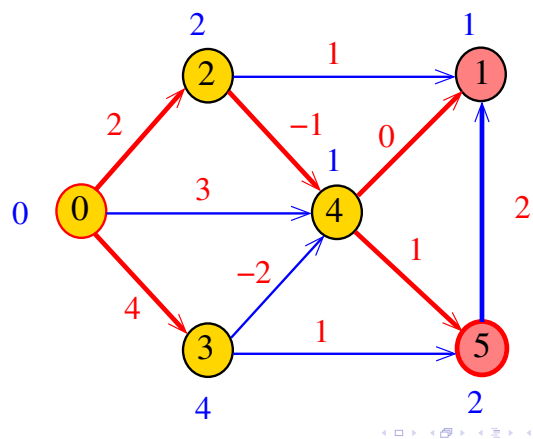
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



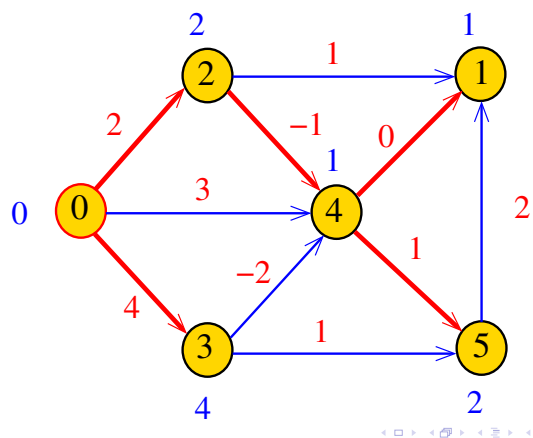
Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



Simulação

i	0	1	2	3	4	5
ts[i]	0	3	2	4	5	1



DAGmin

A função **DAGmin** recebe um DAG **G** com custos **possivelmente negativos** e uma ordenação topológica **ts** de **G**. Recebe também um vértice **s**. Para cada vértice **t**, a função calcula o custo de um caminho de custo mínimo de **s** a **t**. Esse número é depositado em **cst[t]**.

```
void
DAGmin (Digraph G, Vertex ts[], Vertex s,
double cst[])
```

DAGmin

```
void DAGmin (Digraph G, Vertex ts[], Vertex s,
             double cst[]) {
1  int i; Vertex v; link p;
2  for (v= 0; v < G->V; v++)
3      cst[v] = INFINITO;
4  cst[s] = 0;
5  for (v = ts[i = 0]; i < G->V; v = ts[i++] ){
6      if (cst[v] == INFINITO) continue;
7      for (p=G->adj[v]; p!=NULL; p=p->next)
8          if (cst[p->w] > cst[v] + p->cst)
9              cst[p->w] = cst[v] + p->cst;
}
}
```

Consumo de tempo

O consumo de tempo da função **DAGmin** é $O(V + A)$.

Caminhos máximos em DAGs

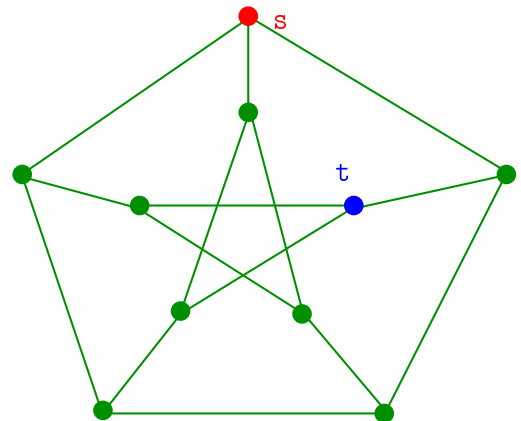
Do ponto de vista computacional, o problema de encontrar um caminho simples de **custo máximo** num digrafos com custos nos arcos é difícil.

Mais precisamente, problema é **NP-difícil** como vocês verão no final de **Análise de Algoritmos**.

O problema torna-se fácil, entretanto, quando restrito a DAGs.

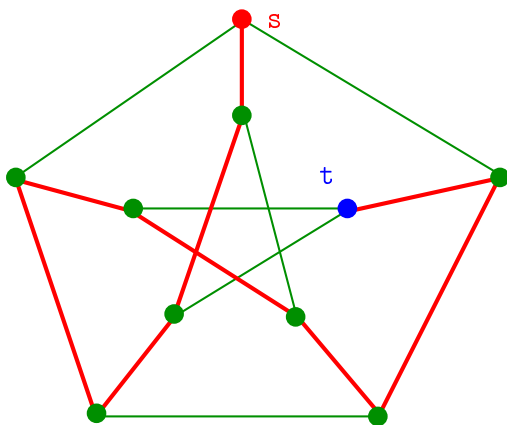
Caminhos hamiltonianos

Problema: Dados vértices **s** e **t** de um grafo encontrar um **caminho** hamiltoniano de **s** e **t**



Caminhos hamiltonianos

Problema: Dados vértices **s** e **t** de um grafo encontrar um **caminho** hamiltoniano de **s** e **t**



DAGmax

```
void DAGmax (Digraph G, Vertex ts[], Vertex s,
             double cst[]) {
1  int i; Vertex v; link p;
2  for (v= 0; v < G->V; v++)
3      cst[v] = -INFINITO;
4  cst[s] = 0;
5  for (v = ts[i = 0]; i < G->V; v = ts[i++] ){
6      if (cst[v] == -INFINITO) continue;
7      for (p=G->adj[v]; p!=NULL; p=p->next)
8          if (cst[p->w] < cst[v] + p->cst)
9              cst[p->w] = cst[v] + p->cst;
}
}
```


Consumo de tempo

O consumo de tempo da função `DAGmax` é $O(V + A)$.

Programação dinâmica

Programação dinâmica

Propriedade (da subestrutura ótima)

Se G é um digrafo com custos não-negativos nos arcos e $v_0-v_1-v_2-\dots-v_k$ é um caminho mínimo então $v_i-v_{i+1}-\dots-v_j$ é um caminho mínimo para $0 \leq i \leq j \leq k$

$\text{custo}[v][w]$ = menor custo de uma caminho de v a w

Propriedade 1

O valor de $\text{custo}[s][t]$ é

$$\min\{\text{custo}[s][v] + \text{custo}[v][t] : v \text{ é vértice}\}$$

Dijkstra em digrafos com custos negativos

Programação dinâmica

Propriedade (da subestrutura ótima)

Se G é um digrafo com custos não-negativos nos arcos e $v_0-v_1-v_2-\dots-v_k$ é um caminho mínimo então $v_i-v_{i+1}-\dots-v_j$ é um caminho mínimo para $0 \leq i \leq j \leq k$

$\text{custo}[v][w]$ = menor custo de uma caminho de v a w

Propriedade 2

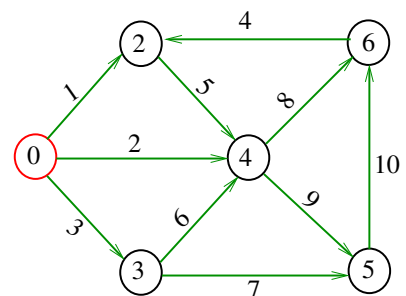
O valor de $\text{custo}[s][t]$ é

$$\min\{\text{custo}[s][v] + G \rightarrow \text{adj}[v][t] : v-t \text{ é arco}\}$$

Problema da SPT

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

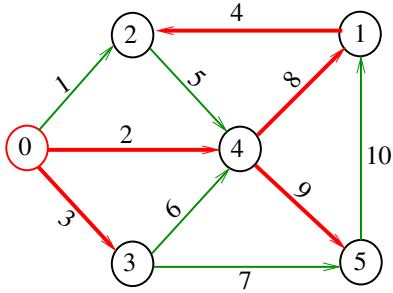
Entra:



Problema da SPT

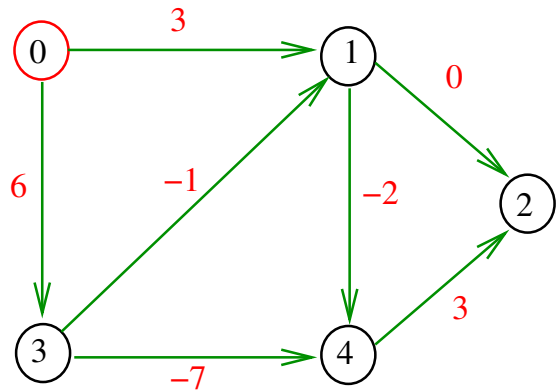
Problema: Dado um vértice s de um digrafo com custos **possivelmente negativos** nos arcos, encontrar uma SPT com raiz s

Sai:



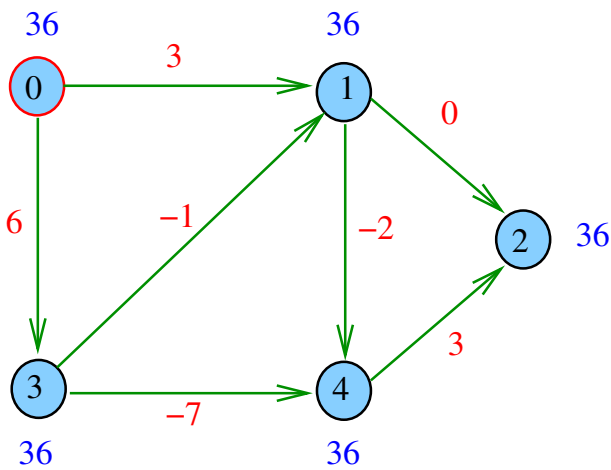
< > < > < > < > < > < >

Apelemos para Dijkstra



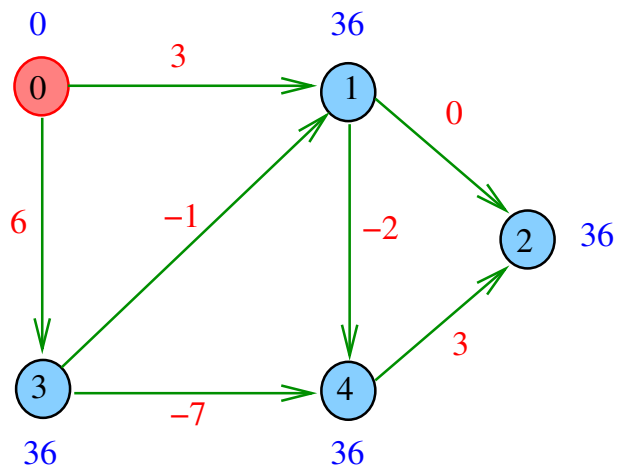
< > < > < > < > < > < >

Apelemos para Dijkstra



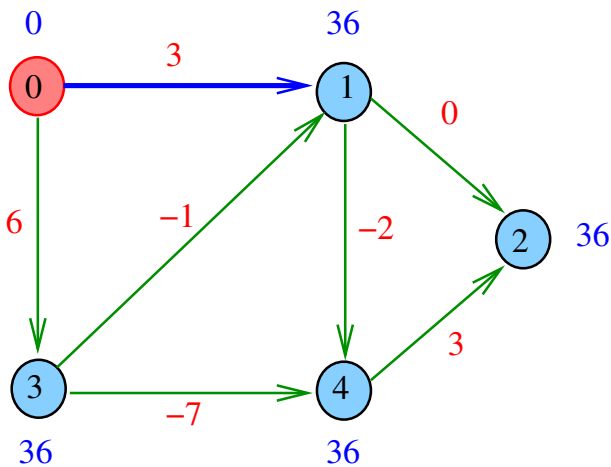
< > < > < > < > < > < >

Apelemos para Dijkstra



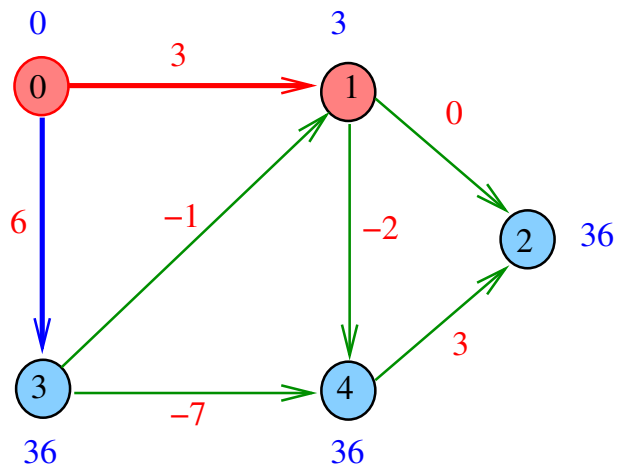
< > < > < > < > < > < >

Apelemos para Dijkstra



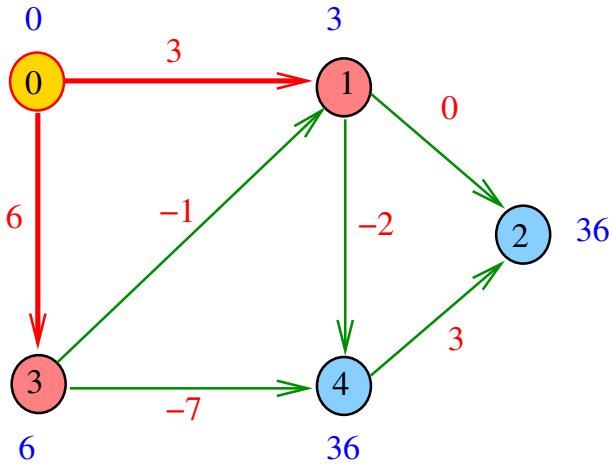
< > < > < > < > < > < >

Apelemos para Dijkstra



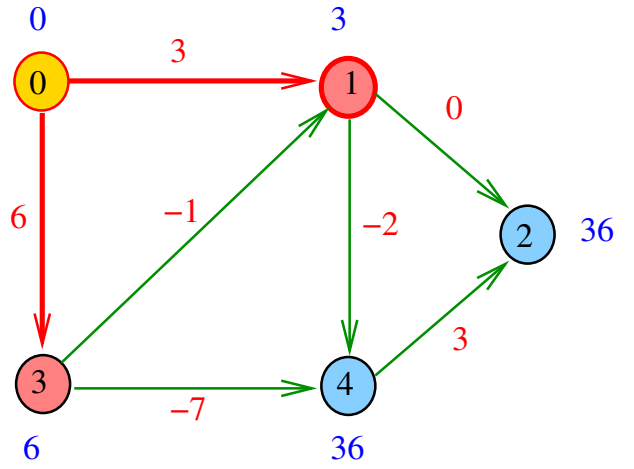
< > < > < > < > < > < >

Apelemos para Dijkstra



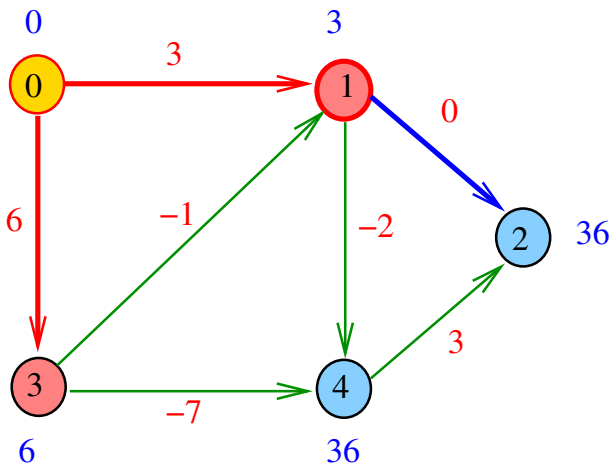
Navigation icons

Apelemos para Dijkstra



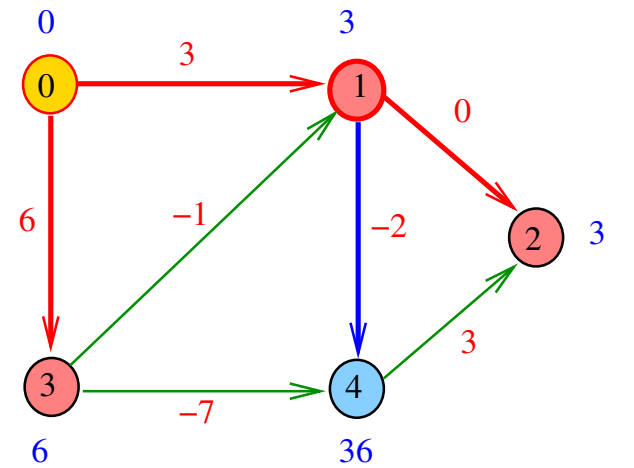
Navigation icons

Apelemos para Dijkstra



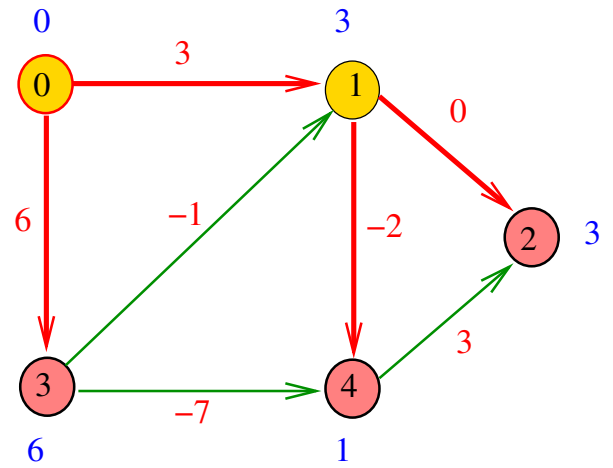
Navigation icons

Apelemos para Dijkstra



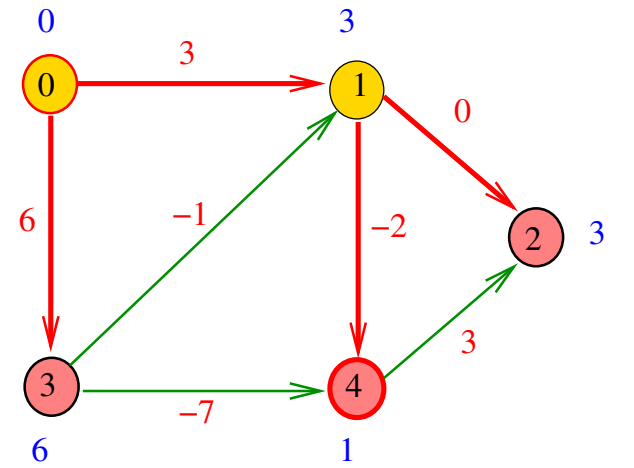
Navigation icons

Apelemos para Dijkstra



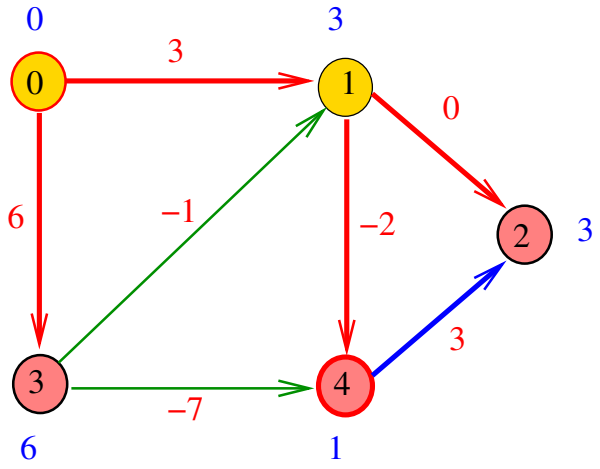
Navigation icons

Apelemos para Dijkstra

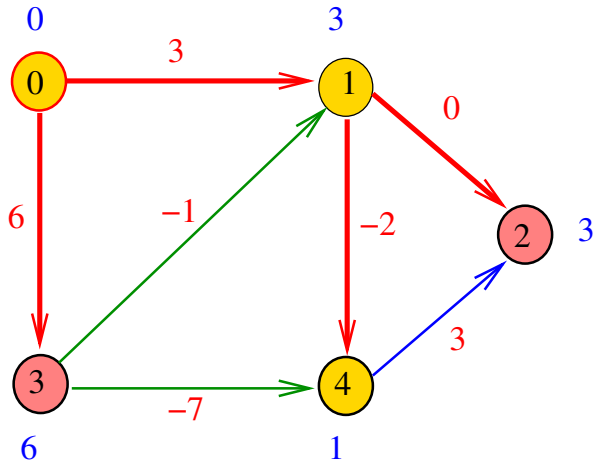


Navigation icons

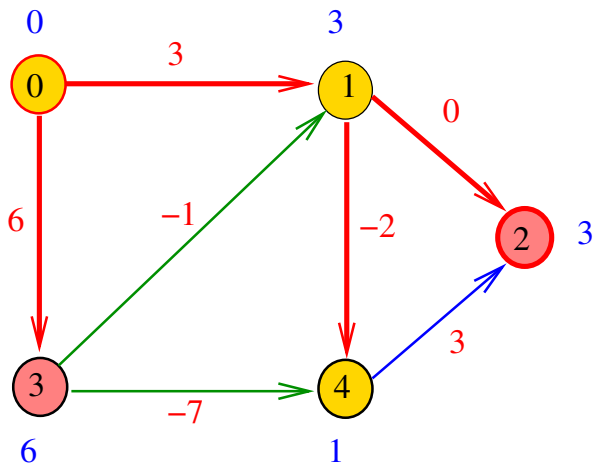
Apelemos para Dijkstra



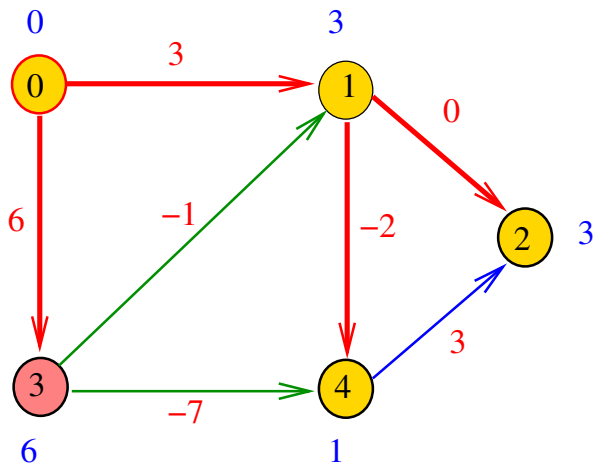
Apelemos para Dijkstra



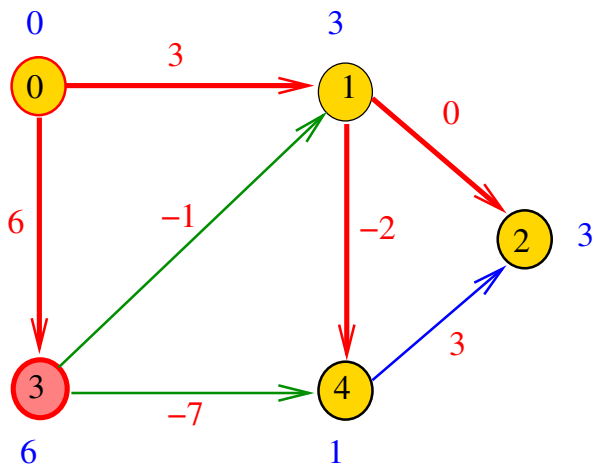
Apelemos para Dijkstra



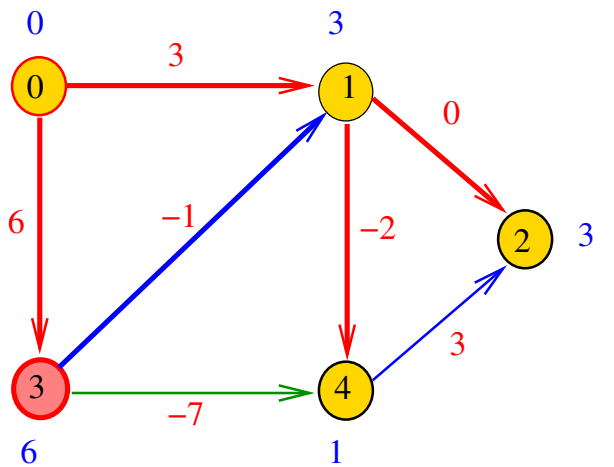
Apelemos para Dijkstra



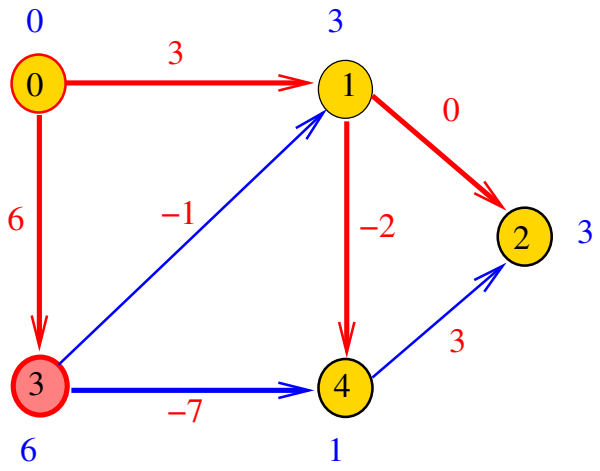
Apelemos para Dijkstra



Apelemos para Dijkstra

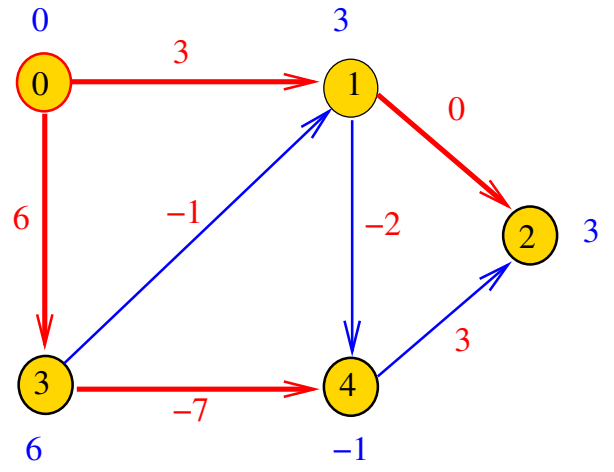


Apelemos para Dijkstra



Navigation icons

Opsss



O caminho mínimo de 0 a 2 tem custo 2 e não 3...