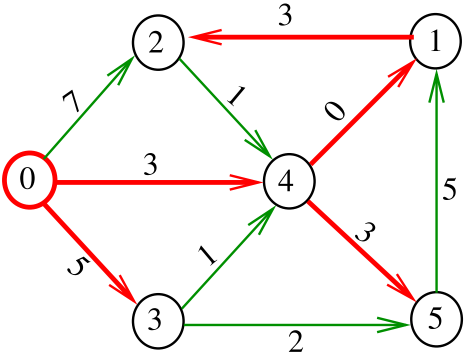
Melhores momentos

AULA 15

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,

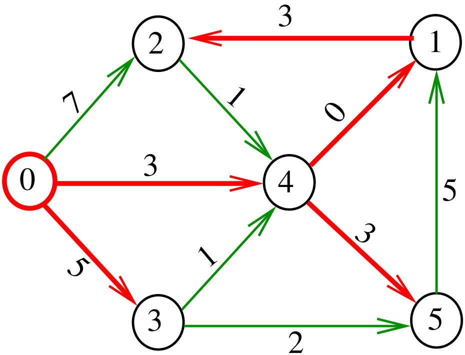
> *o único caminho de s a t na arborescência é um caminho simples mínimo*



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



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[]);
```

Conclusão

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

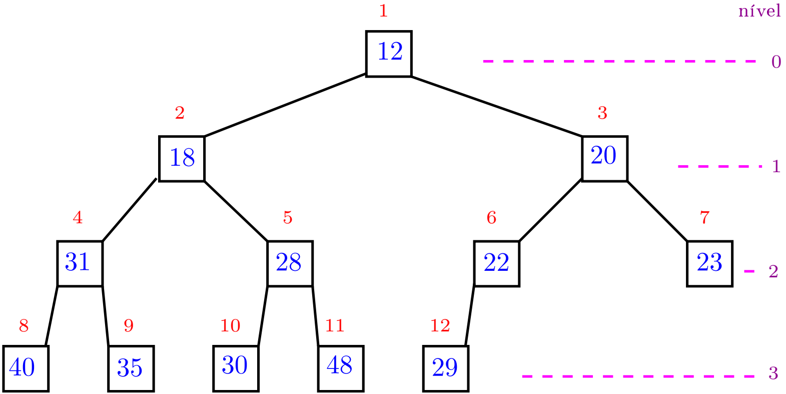
1	execução de	`PQinit` e `PQfree`,
$O(V)$	execuções de	`PQinsert`,
$O(V)$	execuções de	`PQempty`,
$O(V)$	execuções de	`PQdelmin`, e
$O(A)$	execuções de	`PQdec`.

AULA 16

Dijkstra para grafos esparços

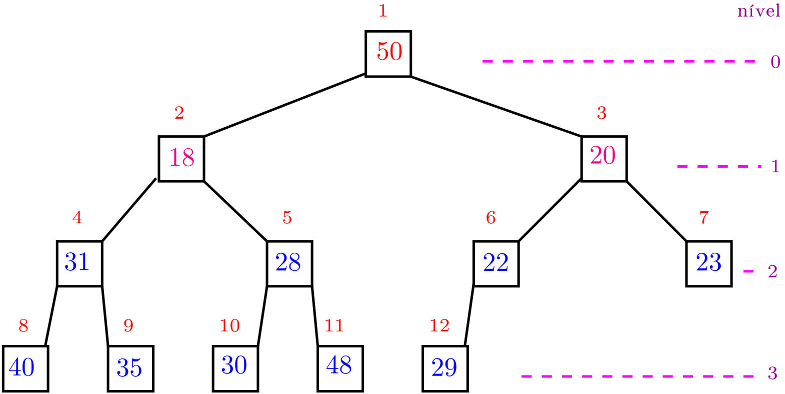
S 21.1 e 21.2

Min-heap



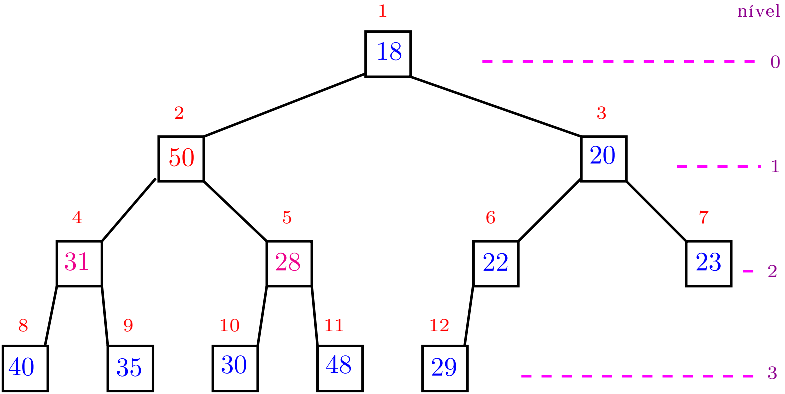
1	2	3	4	5	6	7	8	9	10	11	12
12	18	20	31	28	22	23	40	35	30	48	29

Rotina básica de manipulação de min-heap



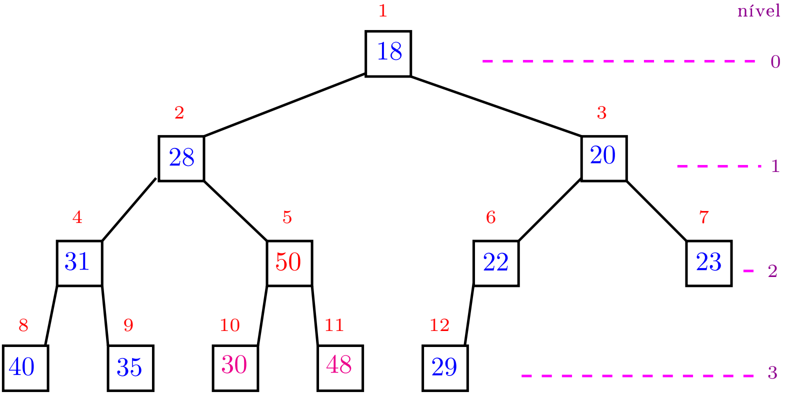
1	2	3	4	5	6	7	8	9	10	11	12
50	18	20	31	28	22	23	40	35	30	48	29

Rotina básica de manipulação de min-heap



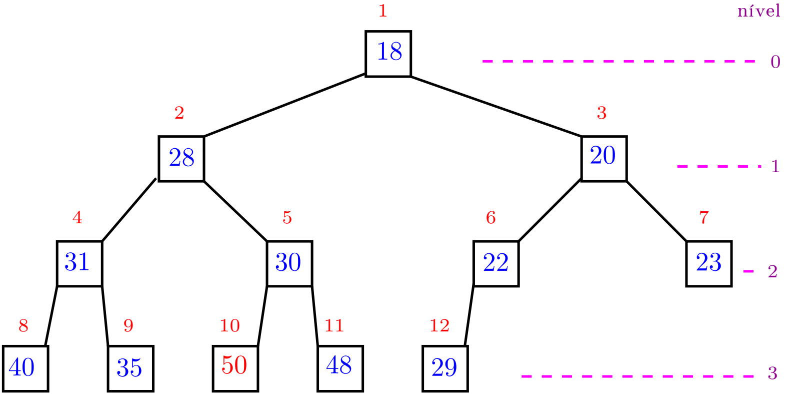
1	2	3	4	5	6	7	8	9	10	11	12
18	50	20	31	28	22	23	40	35	30	48	29

Rotina básica de manipulação de min-heap



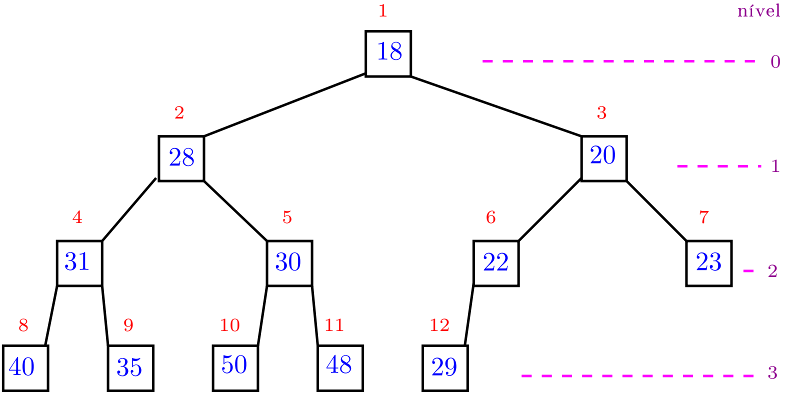
1	2	3	4	5	6	7	8	9	10	11	12
18	28	20	31	50	22	23	40	35	30	48	29

Rotina básica de manipulação de min-heap



1	2	3	4	5	6	7	8	9	10	11	12
18	28	20	31	30	22	23	40	35	50	48	29

Rotina básica de manipulação de min-heap



1	2	3	4	5	6	7	8	9	10	11	12
18	28	20	31	30	22	23	40	35	50	48	29

Implementação clássica Min-Heap

O vetor qp é o "inverso"de pq:

para cada vértice v, qp[v] é o único índice tal que pq[qp[v]] == v.

É claro que qp[pq[i]] == i para todo i.

```
static Vertex pq[maxV+1];
static int N;
static int qp[maxV];
```

PQinit, PQempty, PQinsert

```
void PQinit(void) {
    N = 0;
}
int PQempty(void) {
    return N == 0;
}
void PQinsert(Vertex v) {
    qp[v] = ++N;
    pq[N] = v;
    fixUp(N);
}
```

PQdelmin e PQdec

```
Vertex PQdelmin(void) {
   exch(1, N);
   --N;
   fixDown(1);
   return pq[N+1];
}
void PQdec(Vertex w) {
   fixUp(qp[w]);
}
```

exch e fixUp

```
static void exch(int i, int j) {
    Vertex t;
    t = pq[i]; pq[i] = pq[j]; pq[j] = t;
    qp[pq[i]] = i;
    qp[pq[j]] = j;
}
static void fixUp(int i) {
    while (i>1 && cst[pq[i/2]]>cst[pq[i]]){
        exch(i/2, i);
        i = i/2;
    }
}
```

fixDown

```
static void fixDown(int i) {
    int j;
    while (2*i <= N) {
        j = 2*i;
        if (j< N && cst[pq[j]] > cst[pq[j+1]])
            j++;
        if (cst[pq[i]] <= cst[pq[j]]) break;
        exch(i, j);
        i = j;
    }
}
```

Conclusão

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

Este consumo de tempo é ótimo para **digrafos esparsos**.

Outra implementação para digrafos esparsos

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

Outra implementação para digrafos esparsos

```
while (!PQempty()) {
    v = PQdelmin();
    if (cst[v] == INFINITO) break;
    for (p=G->adj[v];p!=NULL;p=p->next){
        w = p->w;
        if (cst[w] > cst[v] + p->cst){
            cst[w] = cst[v] + p->cst;
            PQdec(w);
            parnt[w] = v;
        }
    }
}
}
```

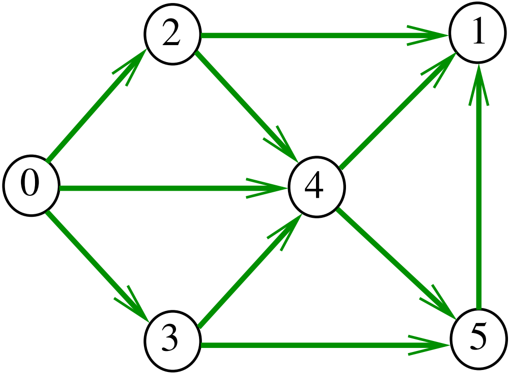
Caminhos mínimos em DAGs

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

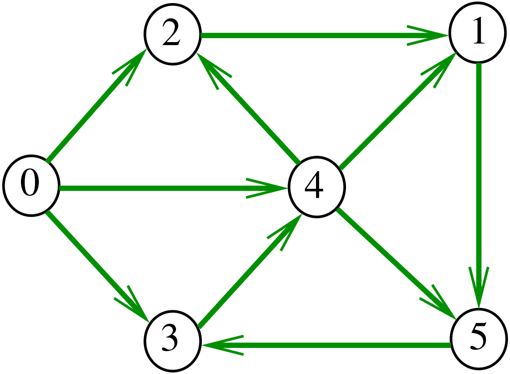


DAGs

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

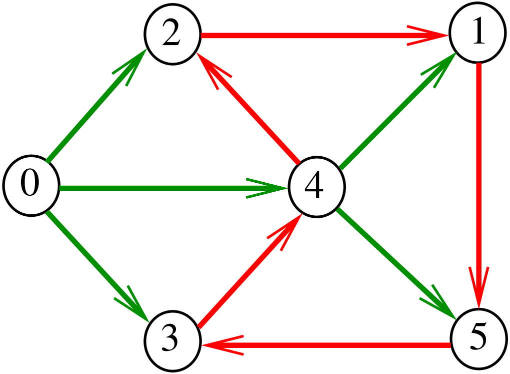


DAGs

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



Ordenaçao 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], ..., ts[V-1]
```

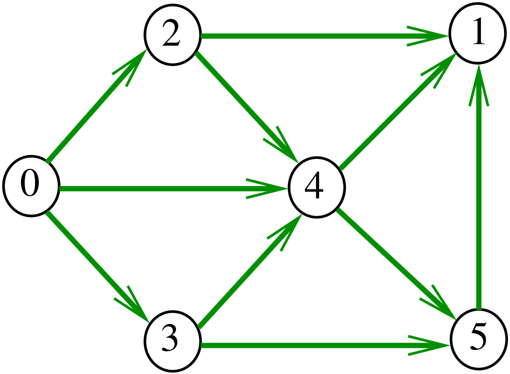
dos seus vértices tal que todo arco tem a forma

```
ts[i]-ts[j] 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 umas das seguintes afirmações:

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

Problema

Problema:

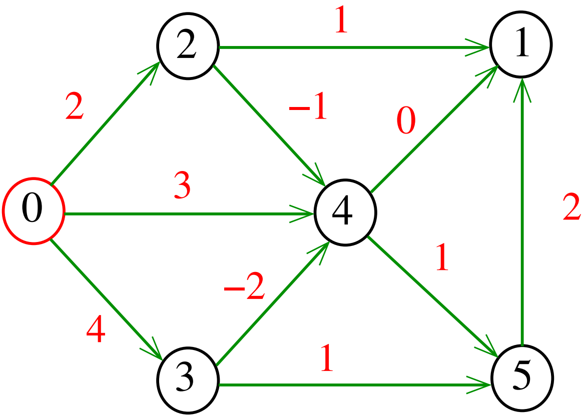
Dado um vértice s de um DAG com custos possivelmente negativos nos arcos, encontrar, para cada vértice t que pode ser alcançado a partir de s, um caminho mínimo simples de s a t

Problema:

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

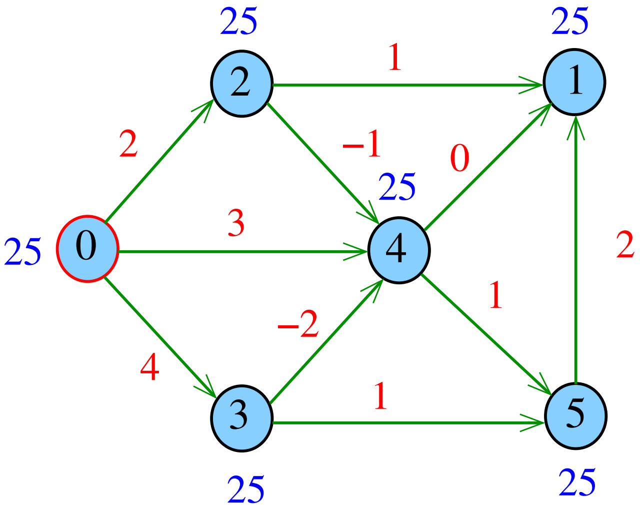
Simulação

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



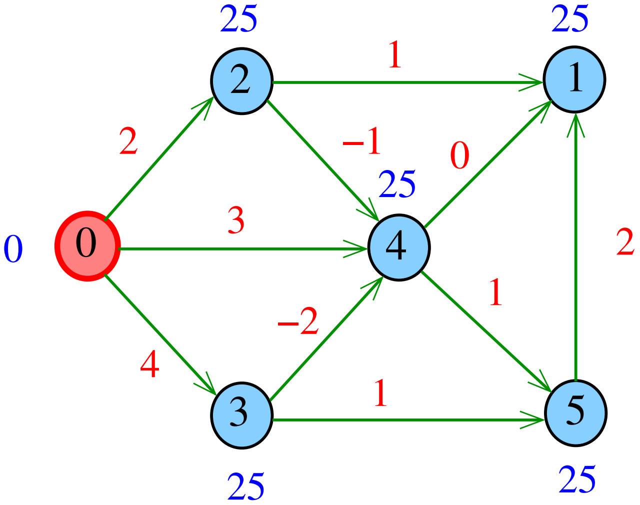
Simulação

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



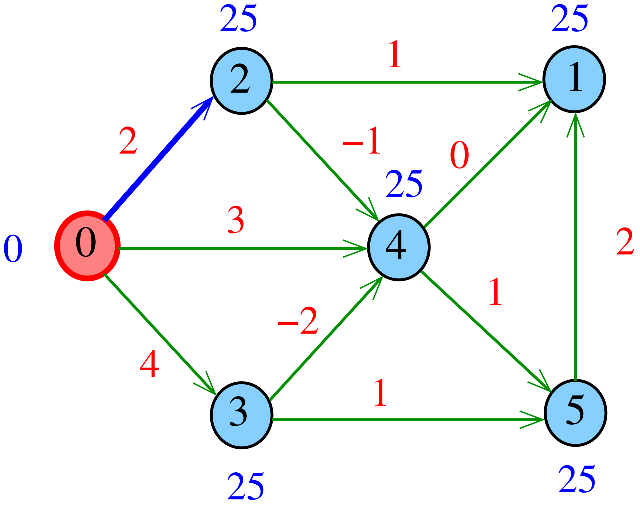
Simulação

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



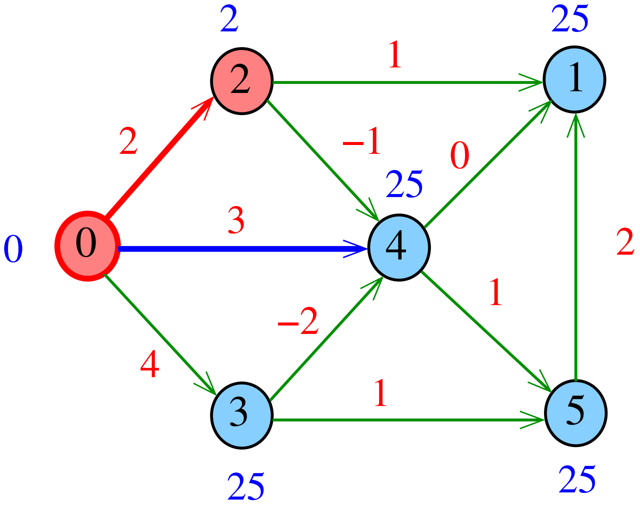
Simulação

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



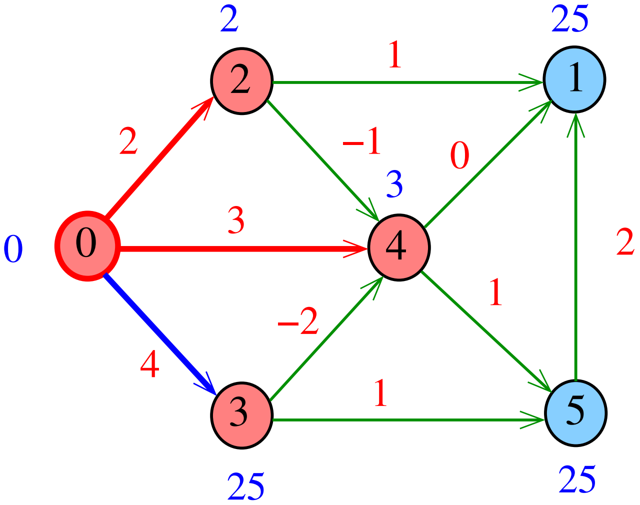
Simulação

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



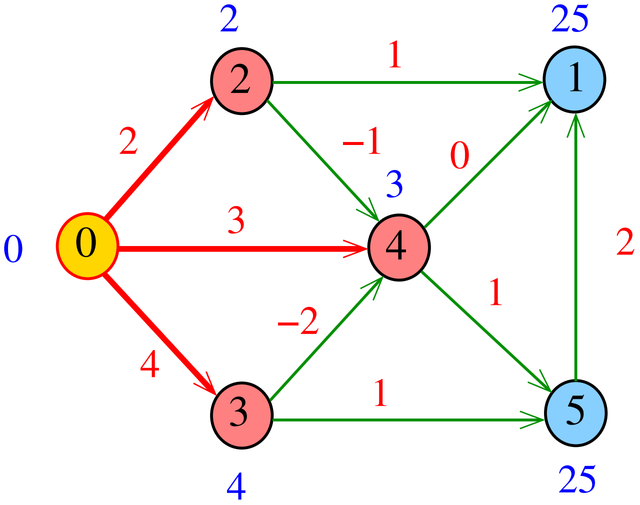
Simulação

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



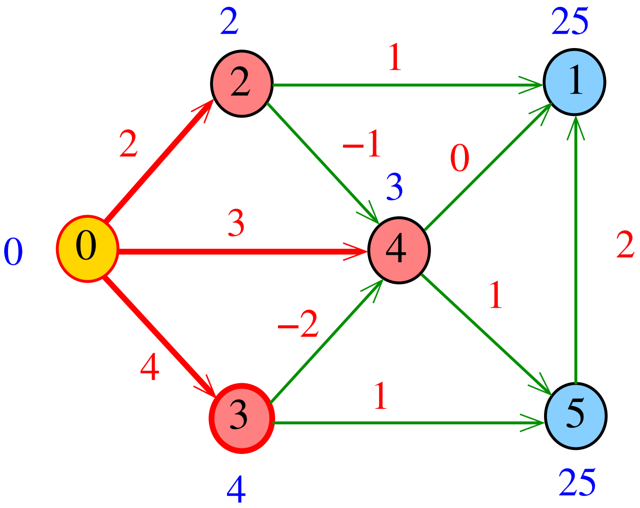
Simulação

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



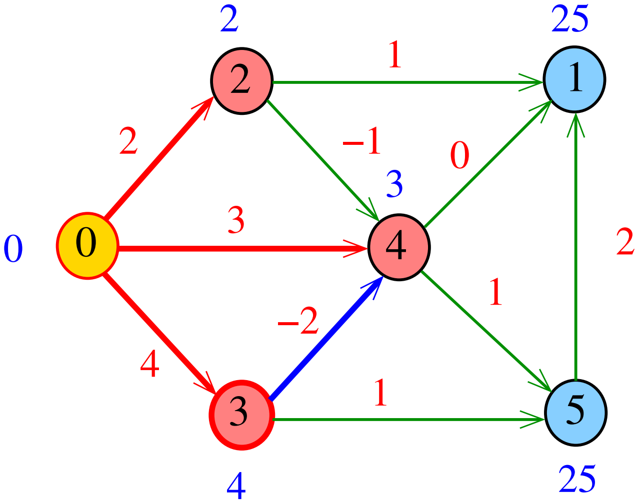
Simulação

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



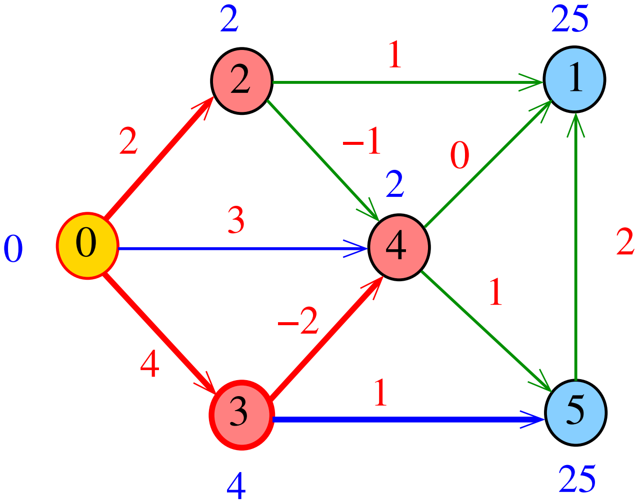
Simulação

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



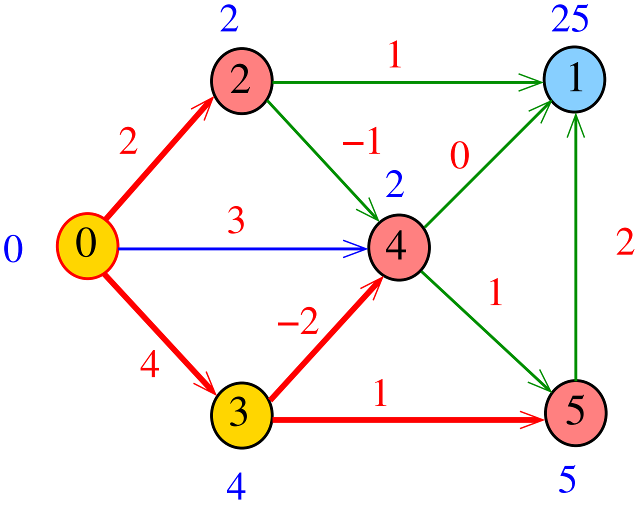
Simulação

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



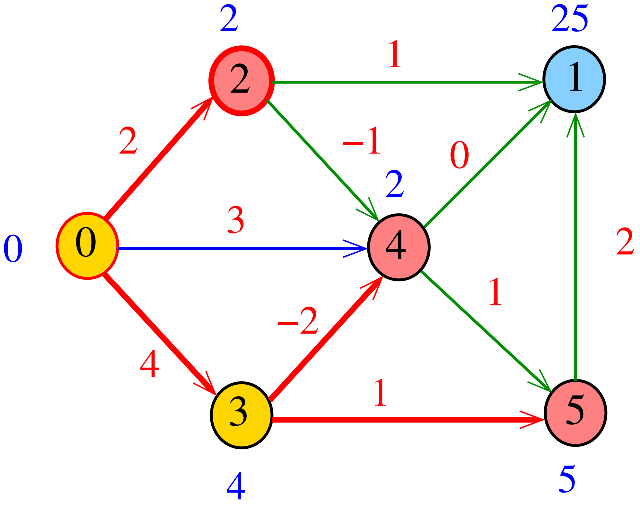
Simulação

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



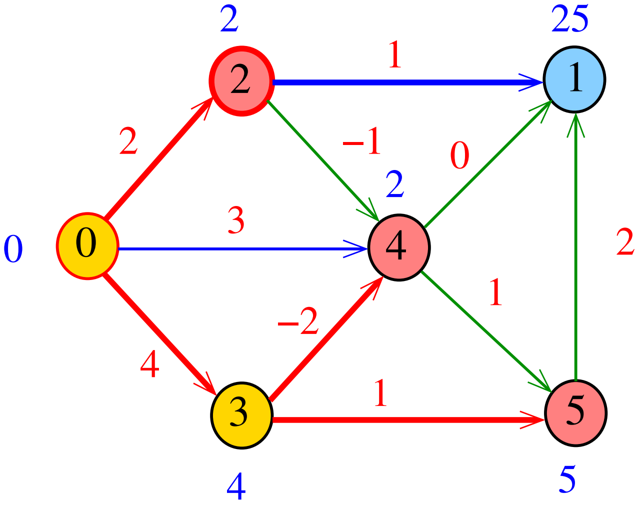
Simulação

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



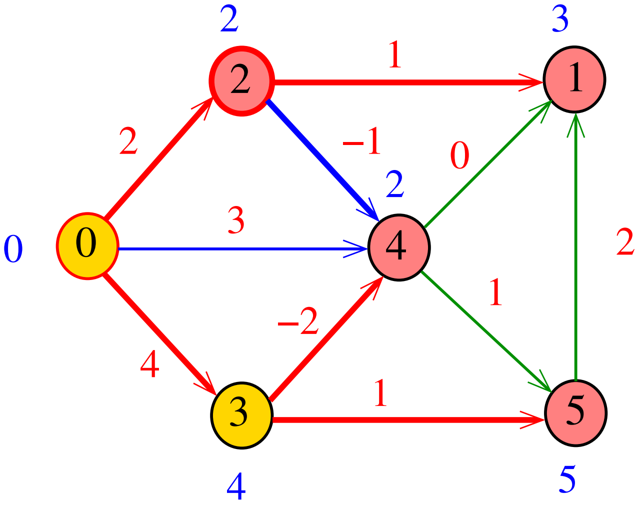
Simulação

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



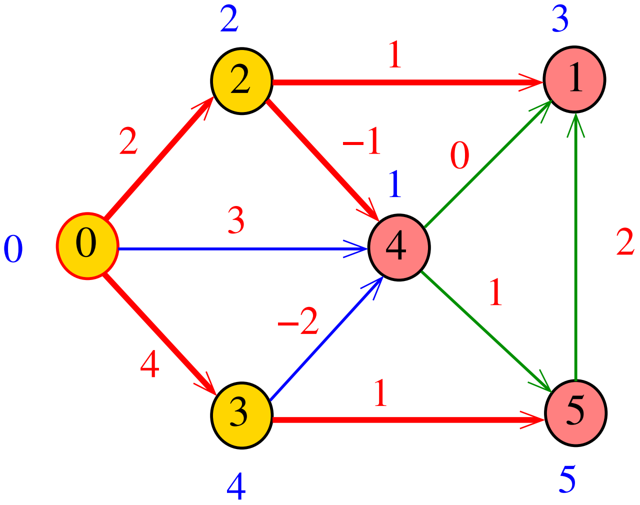
Simulação

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



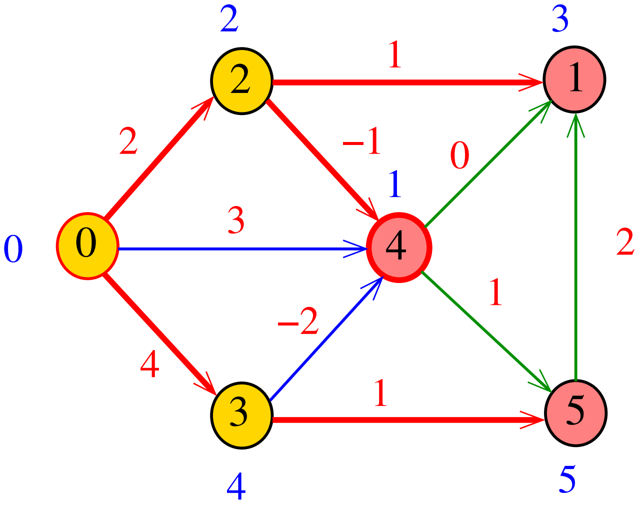
Simulação

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



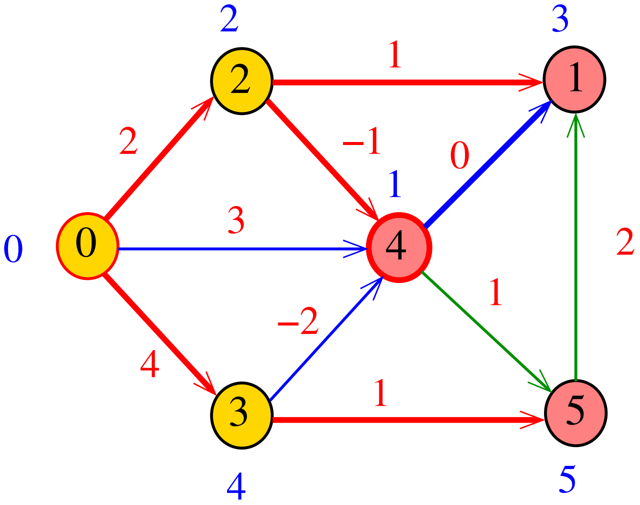
Simulação

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



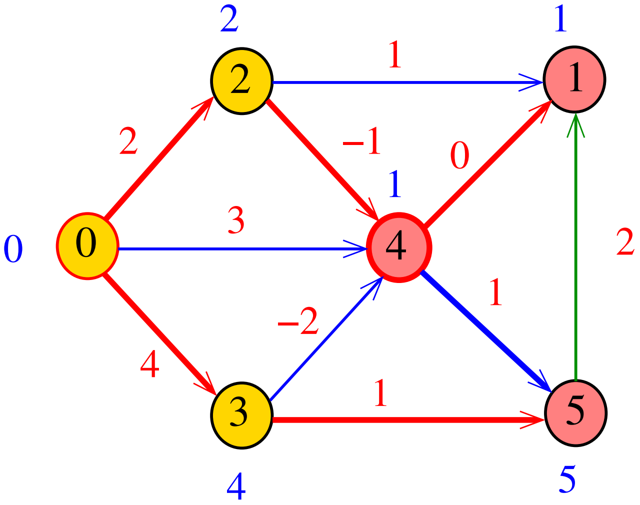
Simulação

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



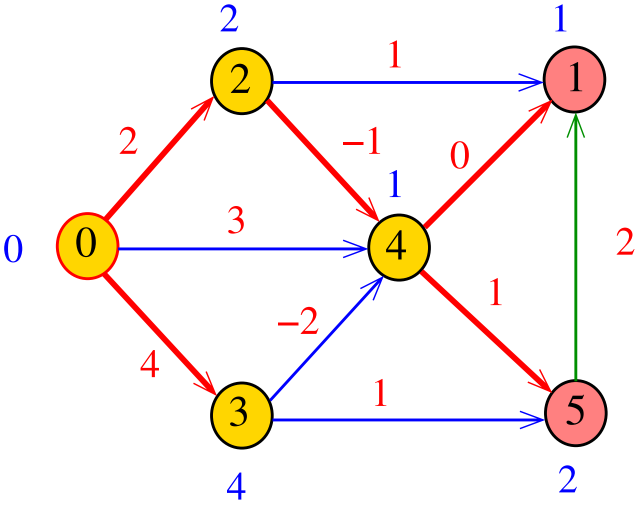
Simulação

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



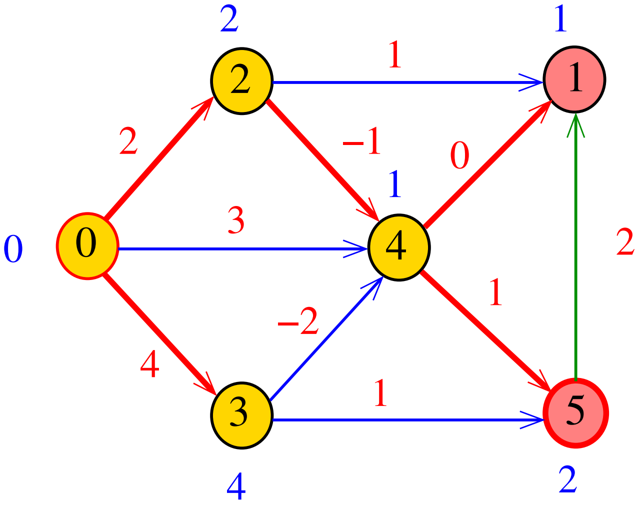
Simulação

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



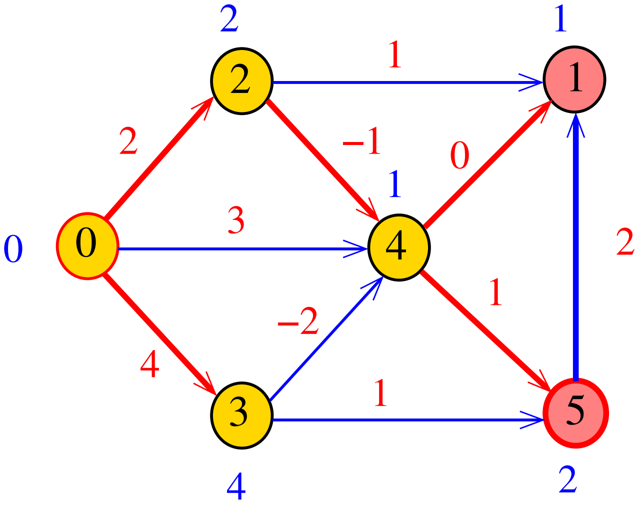
Simulação

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



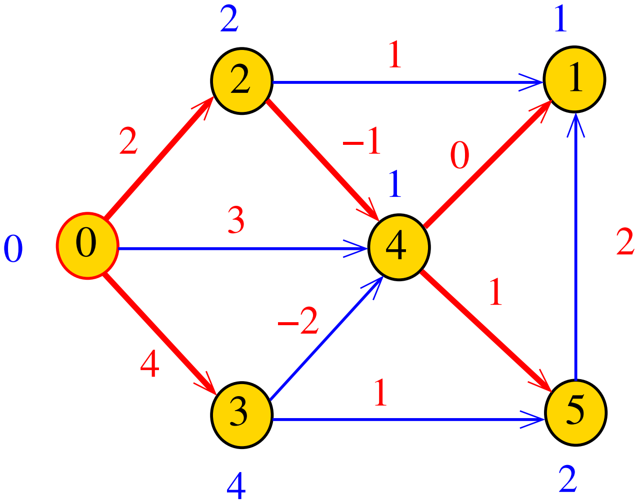
Simulação

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[]) {
    int i; Vertex v; link p;
    for (v= 0; v < G->V; v++)
        cst[v] = INFINITO;
    cst[s] = 0;
    for (v = ts[i = 0]; i < G->V; v = ts[i++])
    for (p=G->adj[v];p!=NULL;p=p->next)
        if (cst[p->w] > cst[v] + p->cst)
            cst[p->w] = cst[v] + p->cst;
}
```

Consumo de tempo

O consumo de tempo da função `DAGmin` é
$O(\mathtt{V} + \mathtt{A})$.

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.

DAGmax

```
void
DAGmax (Digraph G, Vertex ts[], Vertex s,
                double cst[]) {
    int i; Vertex v; link p;
    for (v= 0; v < G->V; v++)
        cst[v] = -INFINITO;
    cst[s] = 0;
    for (v= ts[i = 0]; i < G->V; v = ts[i++])
    for (p=G->adj[v];p!=NULL;p=p->next)
        if (cst[p->w] < cst[v] + p->cst)
            cst[p->w] = cst[v] + p->cst;
}
```

Consumo de tempo

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

Programação dinâmica

Programação dinâmica

Propriedade (da subestrutura ótima)
Se G é um digrafo com custo não-negativos nos arcos e v_0-v_1-v_2-...-v_k é um caminho mínimo então v_i-v_{i+1}-...-v_j é um caminho mínimo para $0 \leq i \leq j \leq k$

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

Propriedade 1
O valor de $\texttt{custo}[s][t]$ é

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

Programação dinâmica

Propriedade (da subestrutura ótima)

Se $\texttt{G}$ é um digrafo com custo não-negativos nos arcos e $\texttt{v}_0\text{-}\texttt{v}_1\text{-}\texttt{v}_2\text{-}\ldots\text{-}\texttt{v}_\texttt{k}$ é um caminho mínimo então $\texttt{v}_\texttt{i}\text{-}\texttt{v}_{\texttt{i}+1}\text{-}\ldots\text{-}\texttt{v}_\texttt{j}$ é um caminho mínimo para $0 \leq \texttt{i} \leq \texttt{j} \leq \texttt{k}$

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

Propriedade 2

O valor de $\texttt{custo}[\texttt{s}][\texttt{t}]$ é

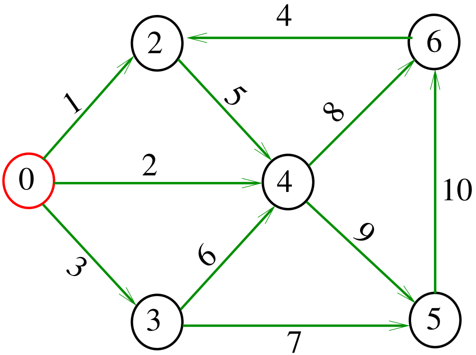
$$\min\{\texttt{custo}[\texttt{s}][\texttt{v}] + \texttt{custo}[\texttt{v}][\texttt{t}] : \texttt{v-t} \text{ é arco}\}$$

Dijkstra em digrafos com custos negativos

Problema da SPT

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

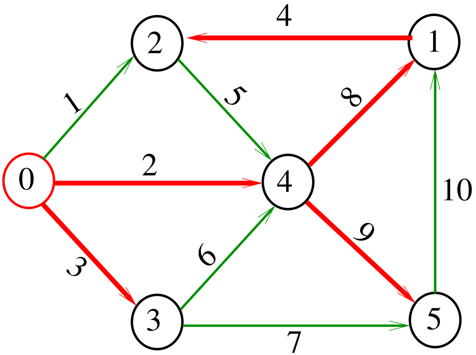
Entra:



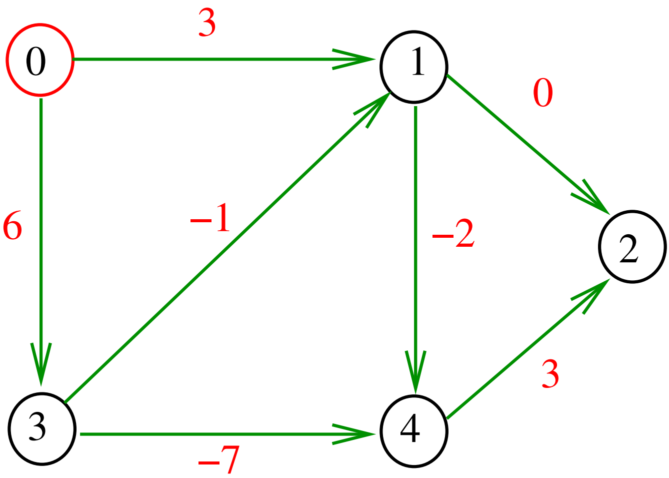
Problema da SPT

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

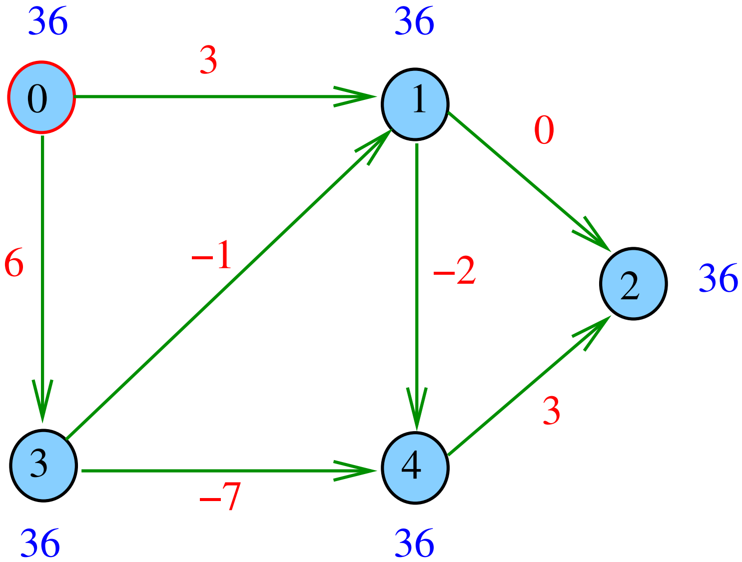
Sai:



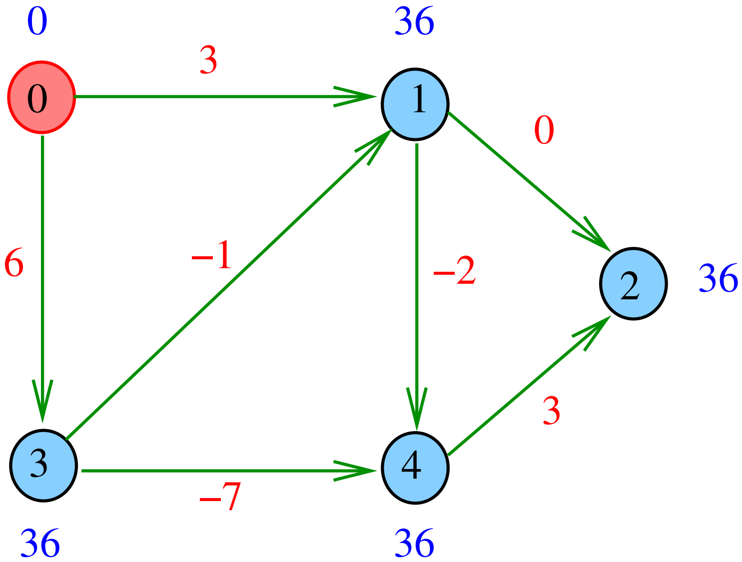
Apelemos para Dijkstra



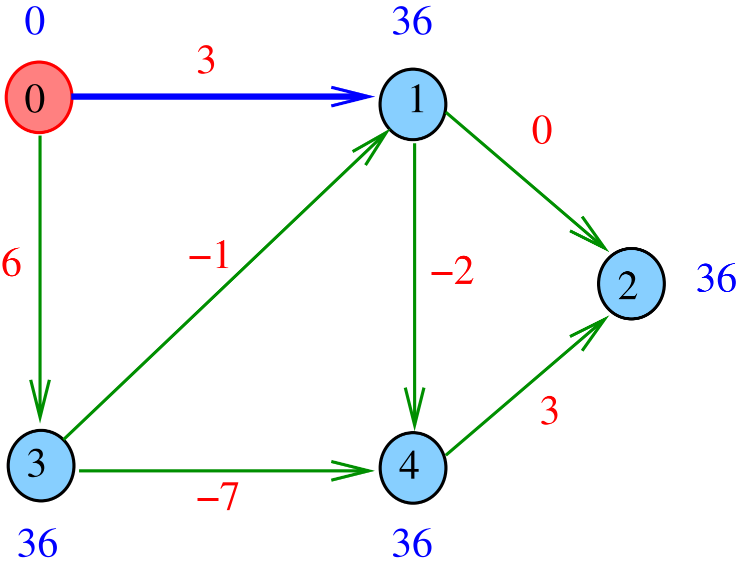
Apelemos para Dijkstra



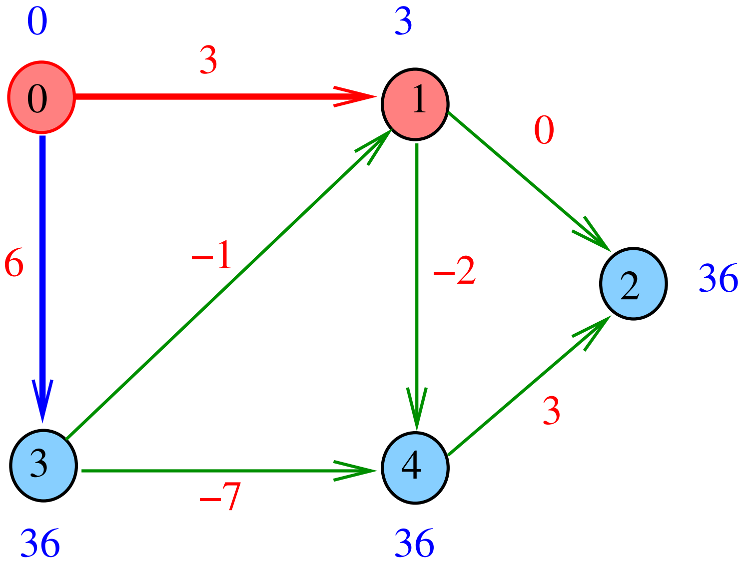
Apelemos para Dijkstra



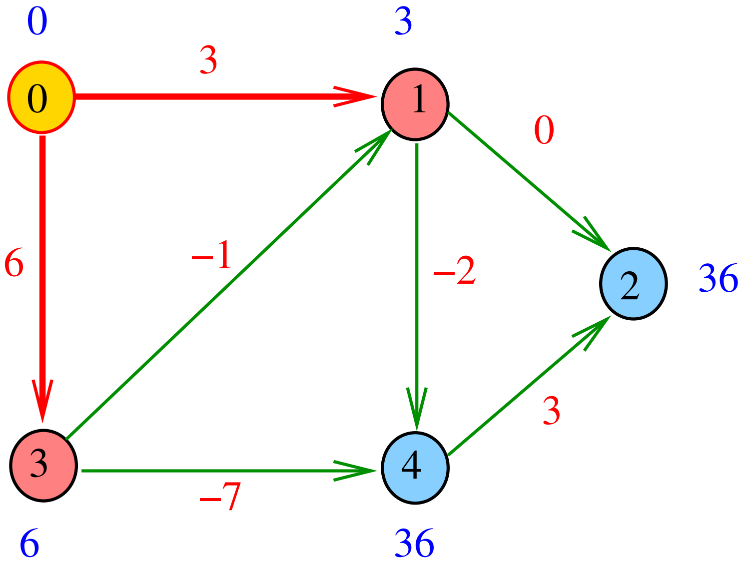
Apelemos para Dijkstra



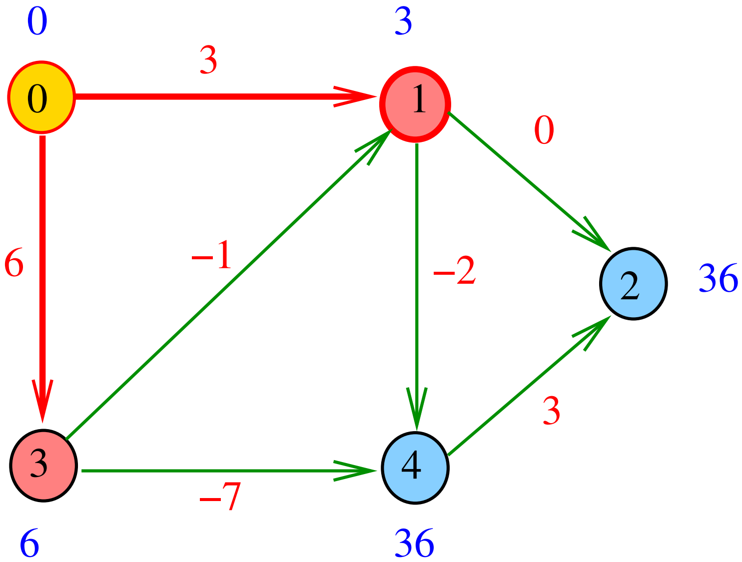
Apelemos para Dijkstra



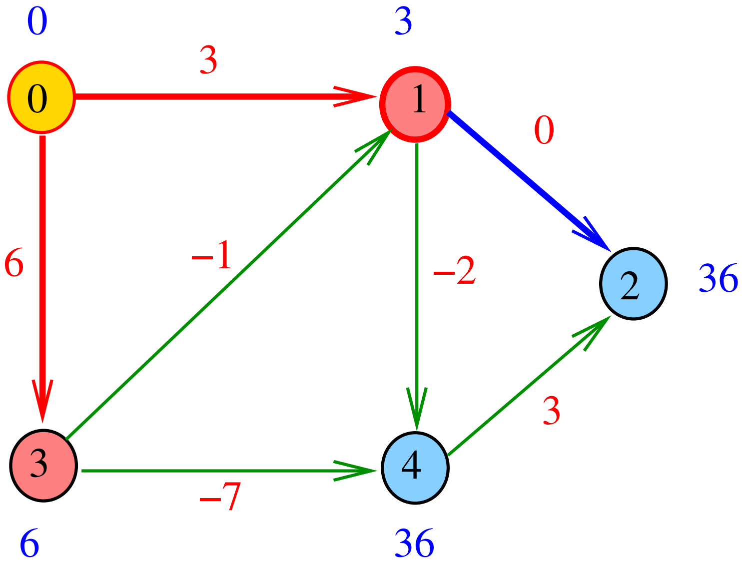
Apelemos para Dijkstra



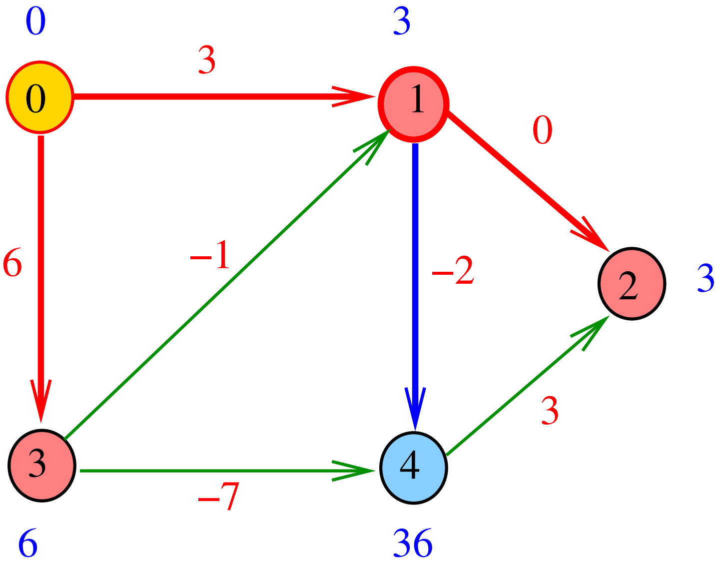
Apelemos para Dijkstra



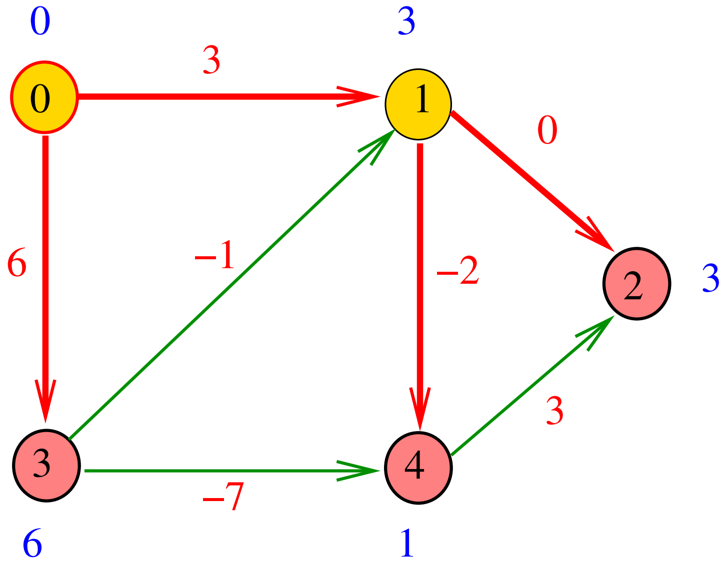
Apelemos para Dijkstra



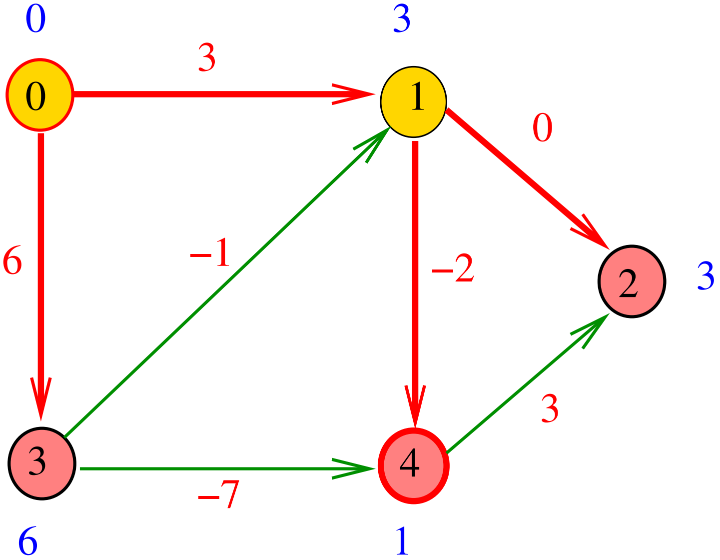
Apelemos para Dijkstra



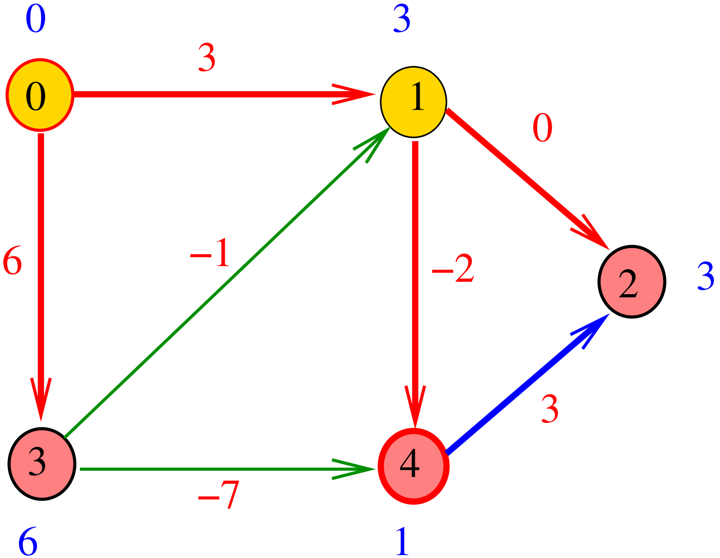
Apelemos para Dijkstra



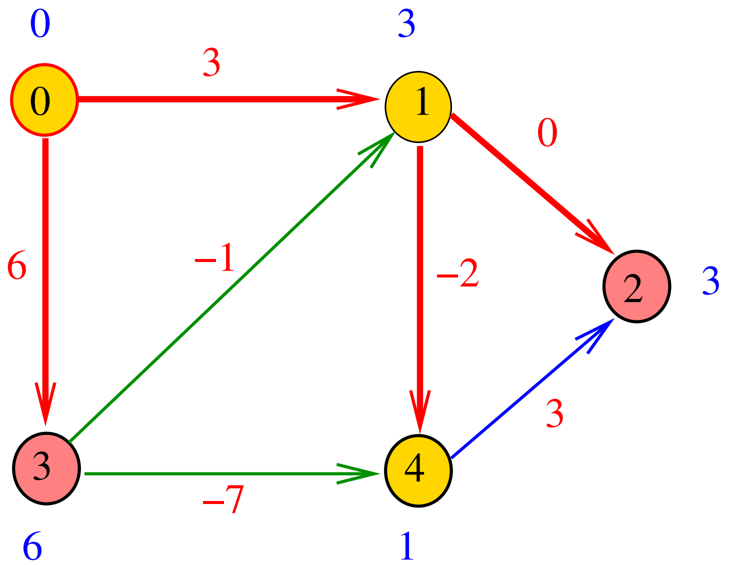
Apelemos para Dijkstra



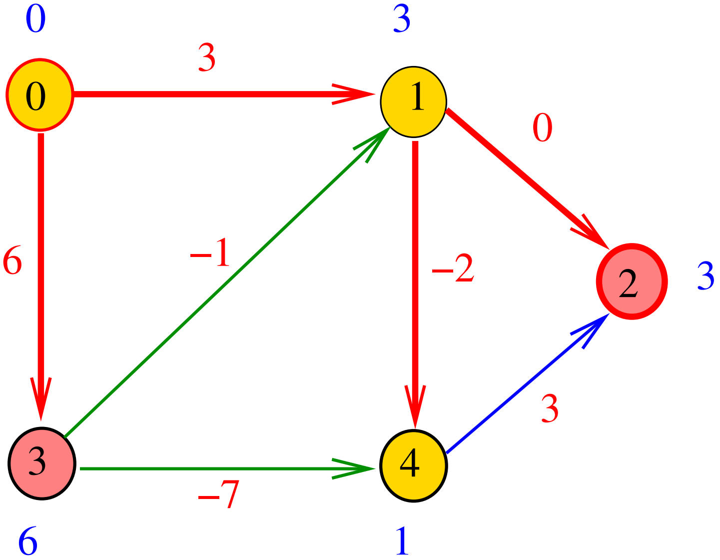
Apelemos para Dijkstra



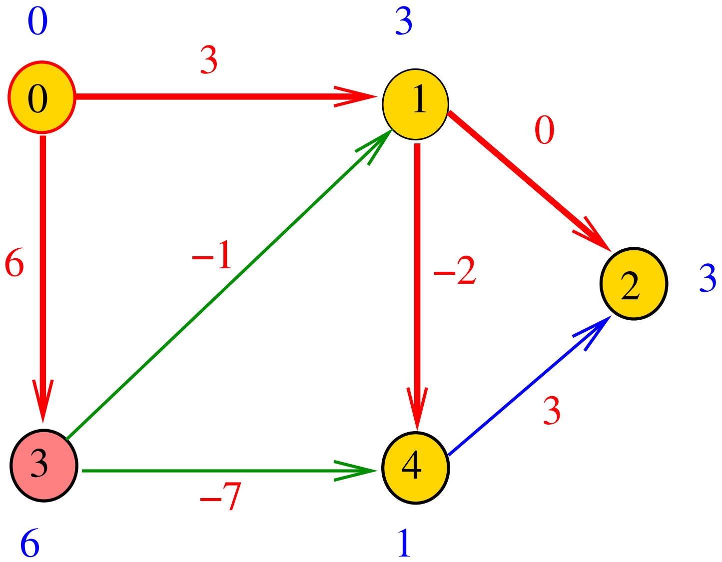
Apelemos para Dijkstra



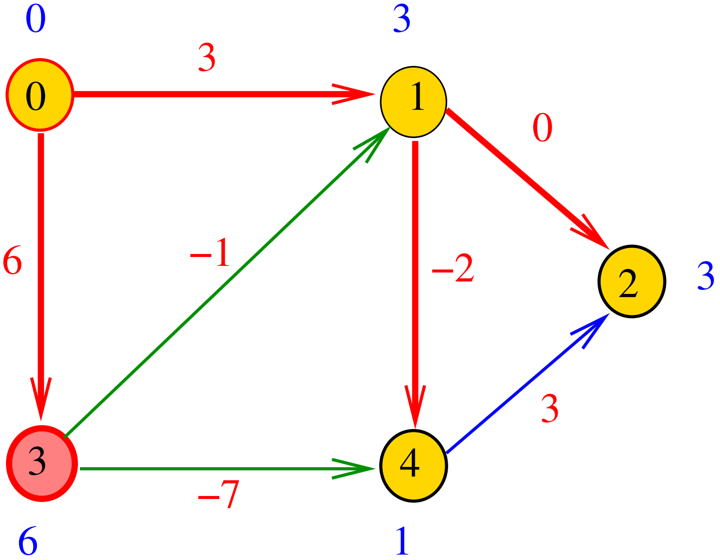
Apelemos para Dijkstra



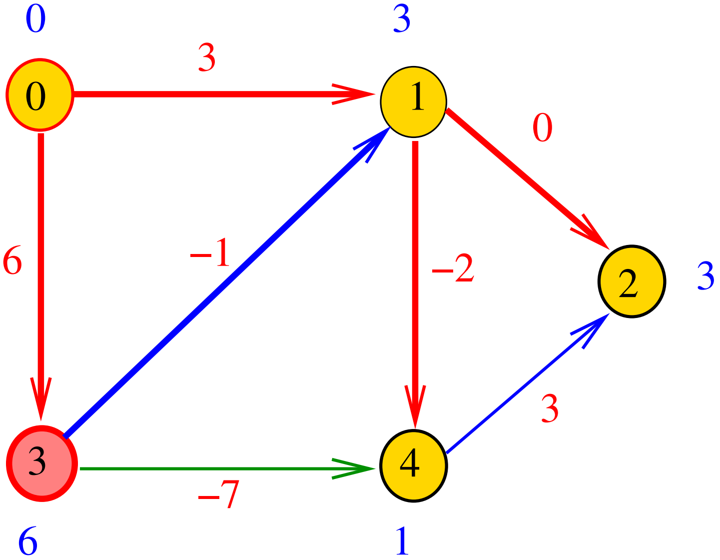
Apelemos para Dijkstra



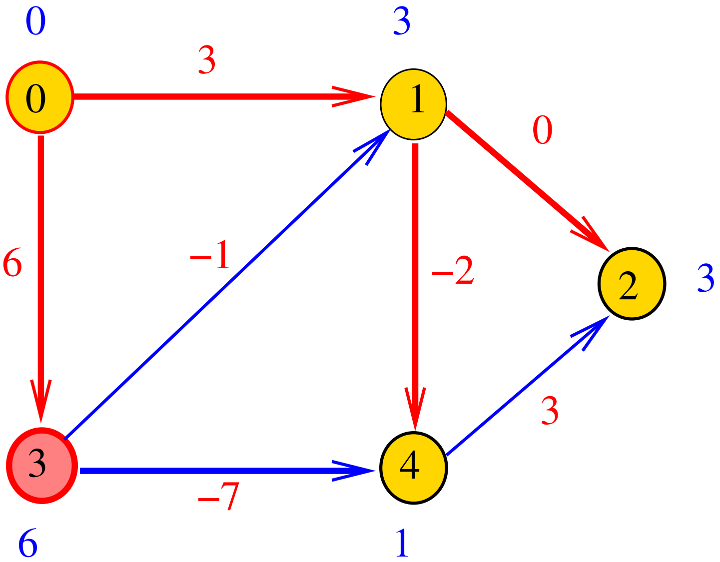
Apelemos para Dijkstra



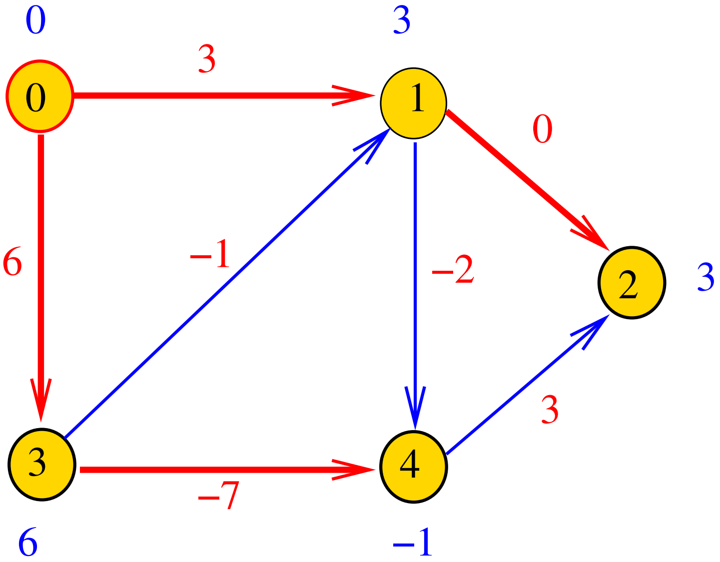
Apelemos para Dijkstra



Apelemos para Dijkstra



Opsss



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