

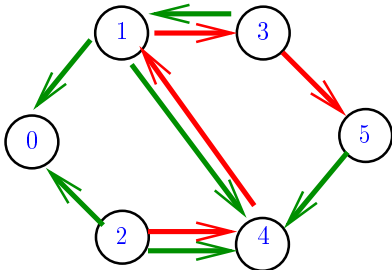
Melhores momentos

AULA 18

Distância

A **distância** de um vértice s a um vértice t é o menor comprimento de um caminho de s a t . Se não existe caminho de s a t a distância é **infinita**

Exemplo: a distância de 0 a 2 é **infinita**



Busca em largura

A **busca em largura** (=breadth-first search search = BFS) começa por um vértice, digamos s , especificado pelo usuário.

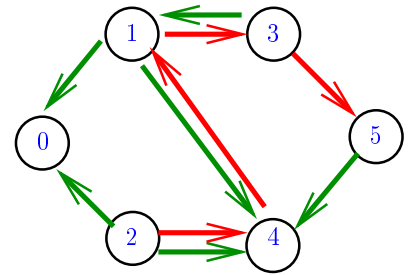
O algoritmo

- visita s ,
- depois visita vértices à **distância 1** de s ,
- depois visita vértices à **distância 2** de s ,
- depois visita vértices à **distância 3** de s ,
- e assim por diante

Distância

A **distância** de um vértice s a um vértice t é o menor comprimento de um caminho de s a t . Se não existe caminho de s a t a distância é **infinita**

Exemplo: a distância de 2 a 5 é 4

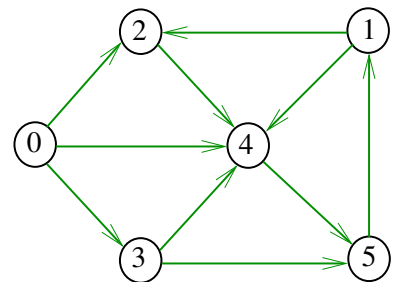


Calculando distâncias

Problema: dados um digrafo G e um vértice s , determinar a distância de s aos demais vértices do digrafo

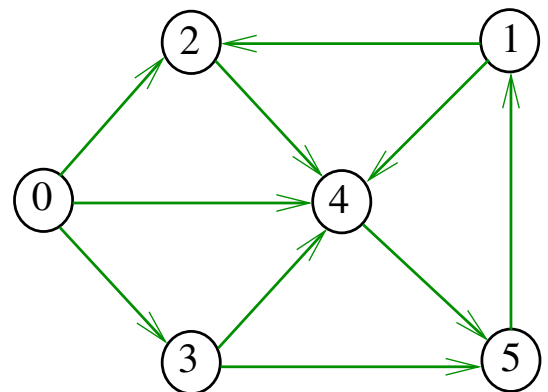
Exemplo: para $s = 0$

| | | | | | | |
|--------------------|---|---|---|---|---|---|
| v | 0 | 1 | 2 | 3 | 4 | 5 |
| $\text{distTo}[v]$ | 0 | 3 | 1 | 1 | 1 | 2 |



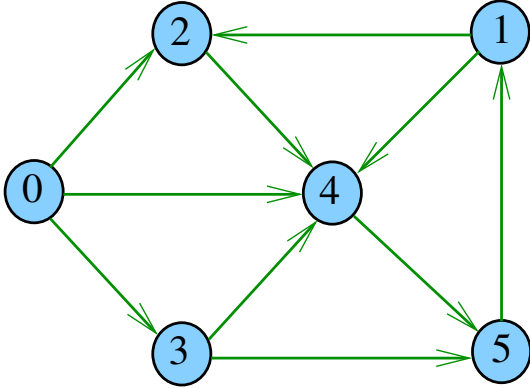
Simulação

| | | | | | | |
|--------------------|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 |
| $q[i]$ | | | | | | |
| v | 0 | 1 | 2 | 3 | 4 | 5 |
| $\text{distTo}[v]$ | | | | | | |



Simulação

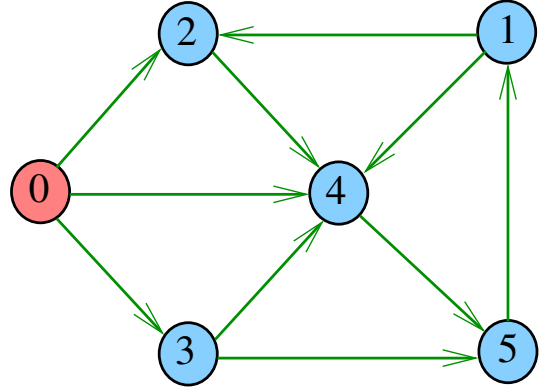
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | | | | | | | distTo[v] | 6 | 6 | 6 | 6 | 6 | 6 |



Navigation icons: back, forward, search, etc.

Simulação

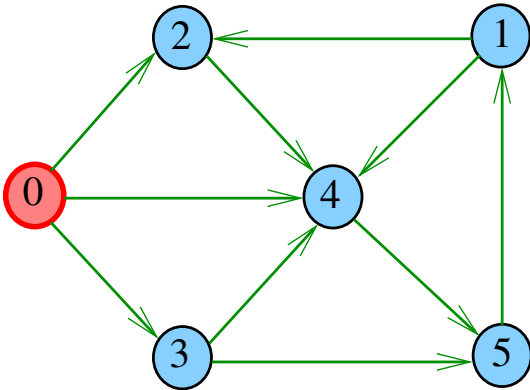
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | | | | | | distTo[v] | 6 | 6 | 6 | 6 | 6 | 6 |



Navigation icons: back, forward, search, etc.

Simulação

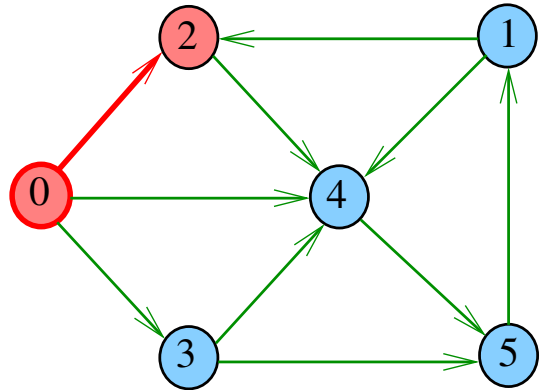
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | | | | | | distTo[v] | 0 | 6 | 6 | 6 | 6 | 6 |



Navigation icons: back, forward, search, etc.

Simulação

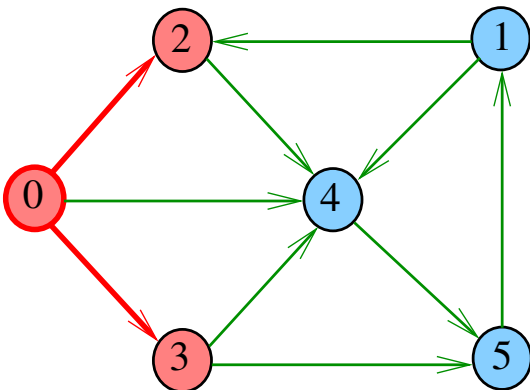
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | | | | | distTo[v] | 0 | 6 | 1 | 6 | 6 | 6 |



Navigation icons: back, forward, search, etc.

Simulação

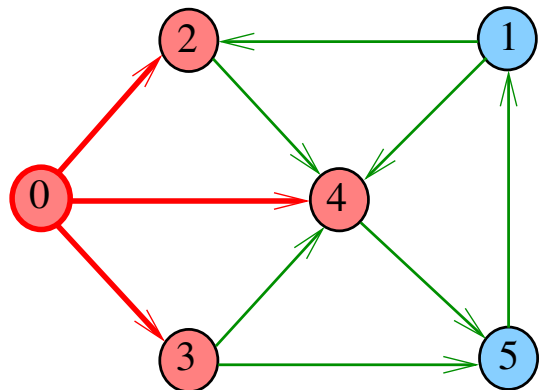
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | | | | distTo[v] | 0 | 6 | 1 | 1 | 6 | 6 |



Navigation icons: back, forward, search, etc.

Simulação

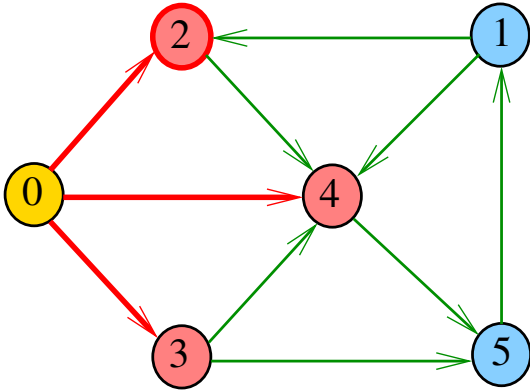
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 6 |



Navigation icons: back, forward, search, etc.

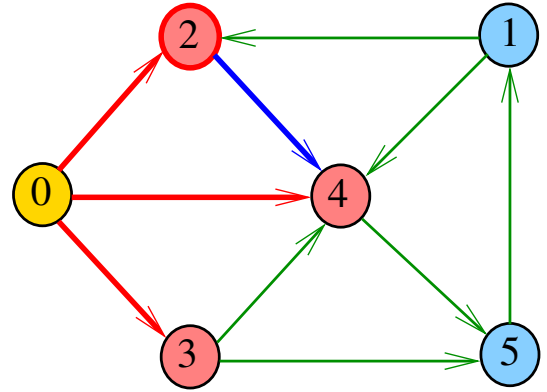
Simulação

| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| q[i] | 0 | 2 | 3 | 4 | | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 6 |



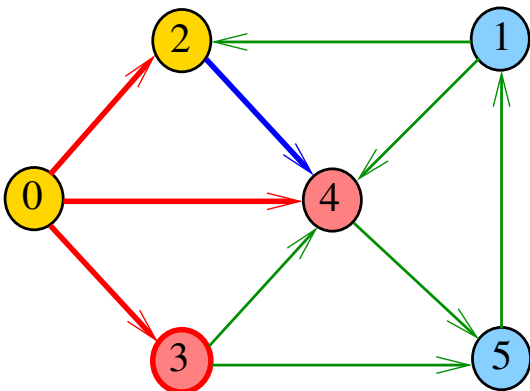
Simulação

| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| q[i] | 0 | 2 | 3 | 4 | | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 6 |



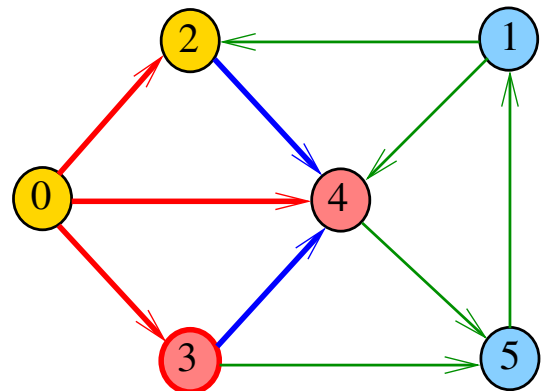
Simulação

| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| q[i] | 0 | 2 | 3 | 4 | | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 6 |



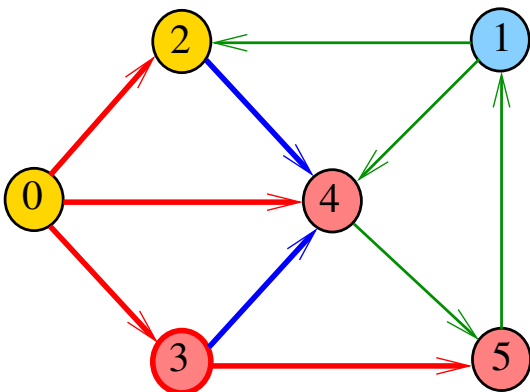
Simulação

| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| q[i] | 0 | 2 | 3 | 4 | | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 6 |



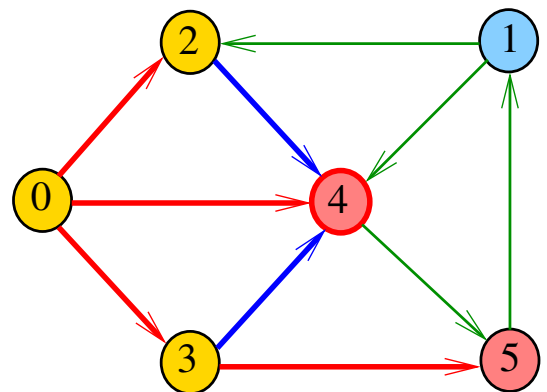
Simulação

| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| q[i] | 0 | 2 | 3 | 4 | 5 | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 2 |



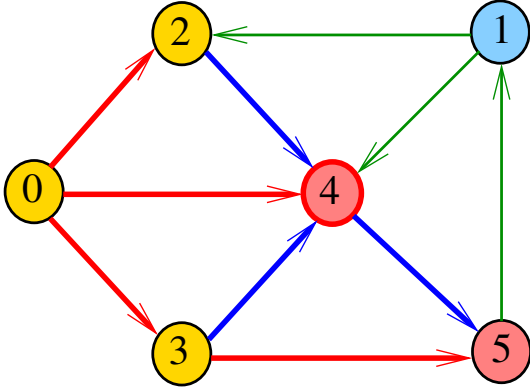
Simulação

| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| q[i] | 0 | 2 | 3 | 4 | 5 | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 2 |



Simulação

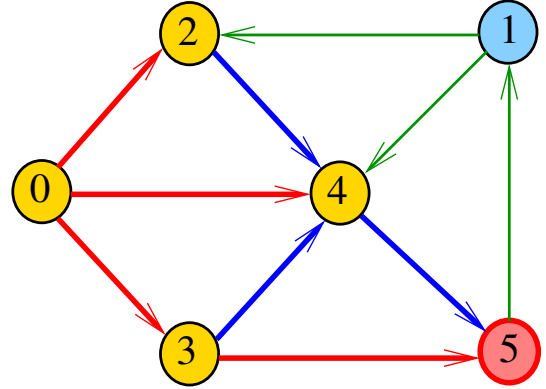
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 2 |



Navigation icons: back, forward, search, etc.

Simulação

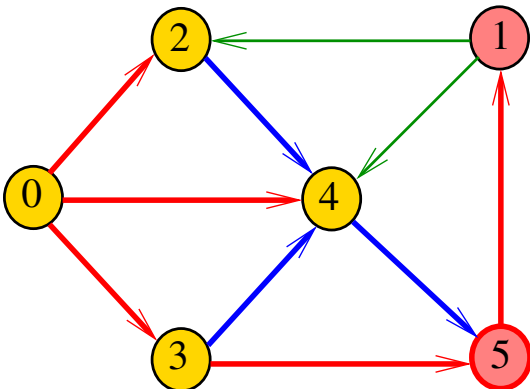
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | | distTo[v] | 0 | 6 | 1 | 1 | 1 | 2 |



Navigation icons: back, forward, search, etc.

Simulação

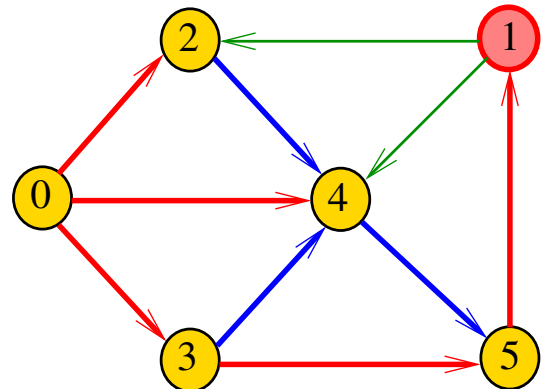
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | 1 | distTo[v] | 0 | 3 | 1 | 1 | 1 | 2 |



Navigation icons: back, forward, search, etc.

Simulação

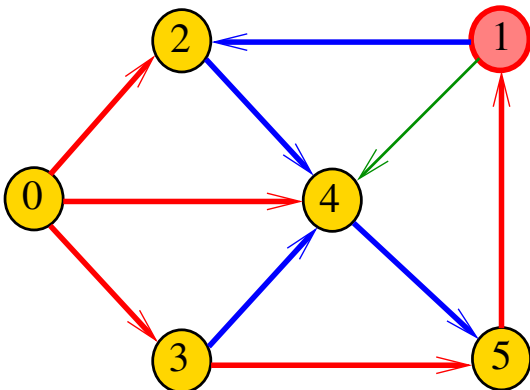
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | 1 | distTo[v] | 0 | 3 | 1 | 1 | 1 | 2 |



Navigation icons: back, forward, search, etc.

Simulação

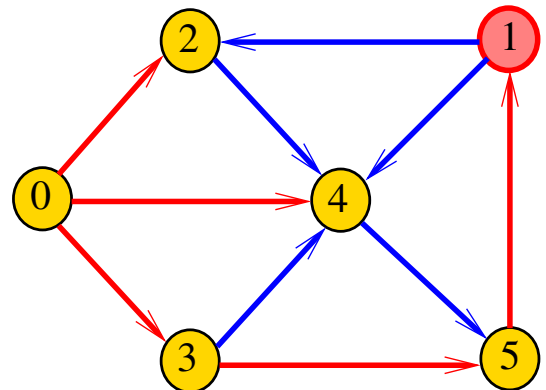
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | 1 | distTo[v] | 0 | 3 | 1 | 1 | 1 | 2 |



Navigation icons: back, forward, search, etc.

Simulação

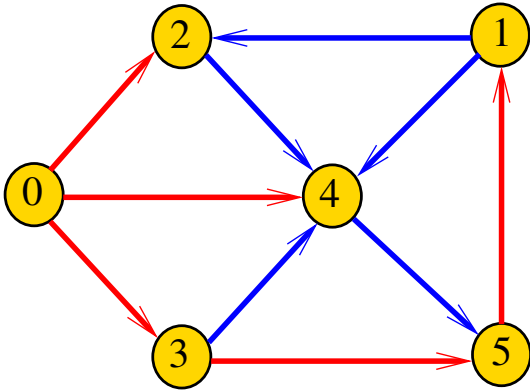
| | | | | | | | | | | | | | |
|----------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | <i>v</i> | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | 1 | distTo[v] | 0 | 3 | 1 | 1 | 1 | 2 |



Navigation icons: back, forward, search, etc.

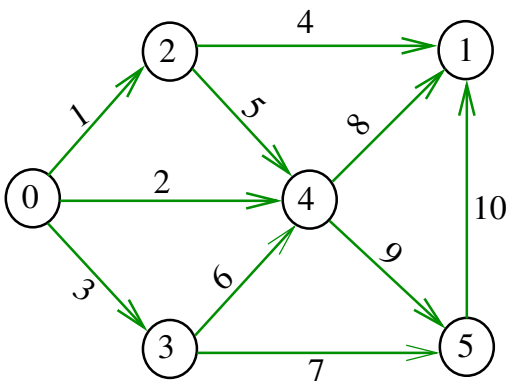
Simulação

| | | | | | | | | | | | | | |
|------|---|---|---|---|---|---|-----------|---|---|---|---|---|---|
| i | 0 | 1 | 2 | 3 | 4 | 5 | v | 0 | 1 | 2 | 3 | 4 | 5 |
| q[i] | 0 | 2 | 3 | 4 | 5 | 1 | distTo[v] | 0 | 3 | 1 | 1 | 1 | 2 |



Digrafo

EdgeWeightedDigraph G



Classe EdgeWeightedDigraph

A estrutura **digraph** representa um digrafo
V contém o número de vértices
E contém o número de arcos do digrafo
adj é uma referência para vetor de listas de adjacência

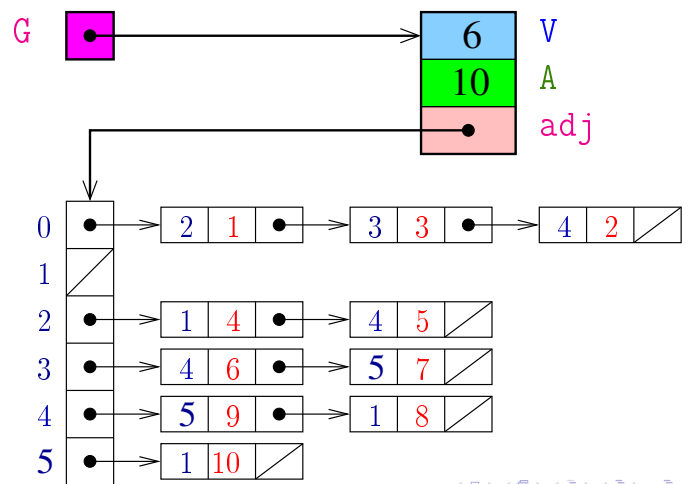
```
public EdgeWeightDigraph (int V) {
    this.V = V;
    this.E = 0;
    adj=(Bag<DirectedEdge>[])new Bag[V];
    for (int v = 0; v < V; v++)
        adj[v] = new Bag<DirectedEdge>();
}
```

Digrafos com custos nos arcos

Muitas aplicações associam um número a cada arco de um digrafo
 Diremos que esse número é o **custo** ou **peso** do arco
 Vamos supor que esses números são do tipo **double** na classe **DirectedEdge**.

```
DirectedEdge(int v, int w, double weight)
    private final int v;
    private final int w;
    private final double weight;
    double weight()...
    int from()...
    int to()...
```

Estruturas de dados



AULA 19

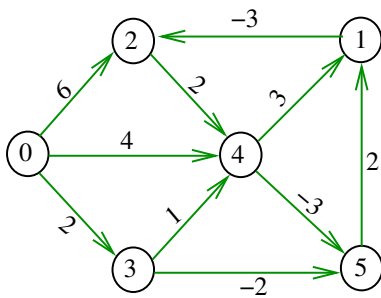
Caminhos de custo mínimo

S 21.0 e 21.1

Caminho mínimo

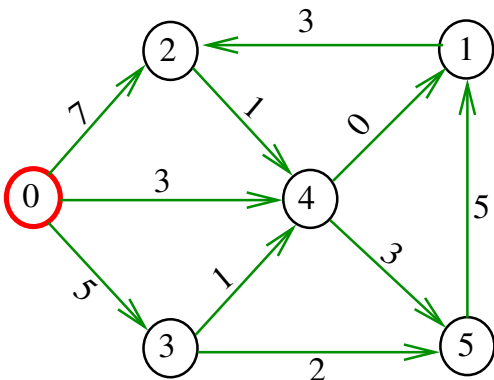
Um caminho P tem **custo mínimo** se o custo de P é menor ou igual ao custo de todo caminho com a mesma origem e término

O caminho 0-3-4-5-1-2 é mínimo, tem custo -1



Exemplo

Entra:



Navigation icons

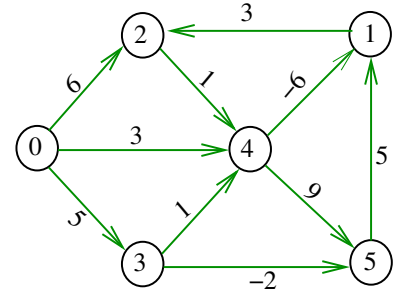
Custo de um caminho

Custo de um caminho é soma dos custos de seus arcos

Custo do caminho 0-2-4-5 é 16.

Custo do caminho 0-2-4-1-2-4-5 é 14.

Custo do caminho 0-2-4-1-2-4-1-2-4-5 é 12.



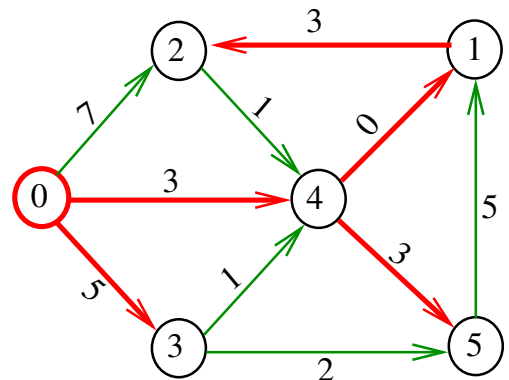
Problema

Problema dos Caminhos Mínimos com Origem Fixa (*Single-source Shortest Paths Problem*):

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

Exemplo

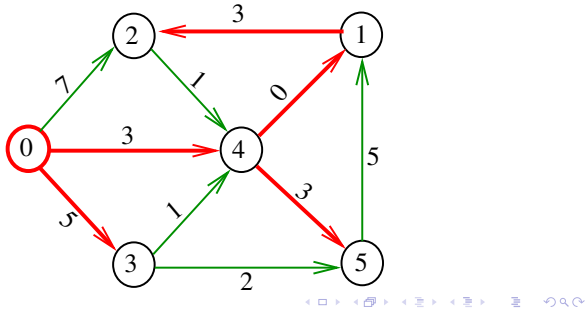
Sai:



Navigation icons

Arborescência de caminhos mínimos

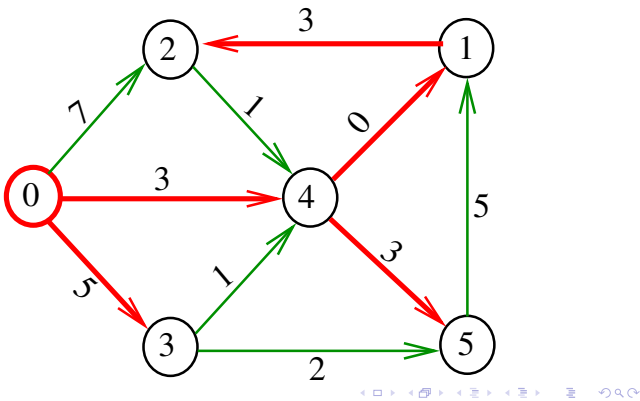
Uma arborescência com raiz **s** é de **caminhos mínimos** (= *shortest-paths tree* = SPT) se para todo vértice **t** que pode ser alcançado a partir de **s**, o único caminho de **s** a **t** na arborescência é um caminho mínimo



Problema da SPT

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

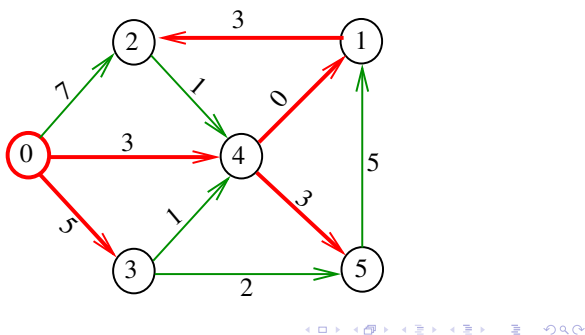
Sai:



Problema

O algoritmo de Dijkstra resolve o problema da SPT:

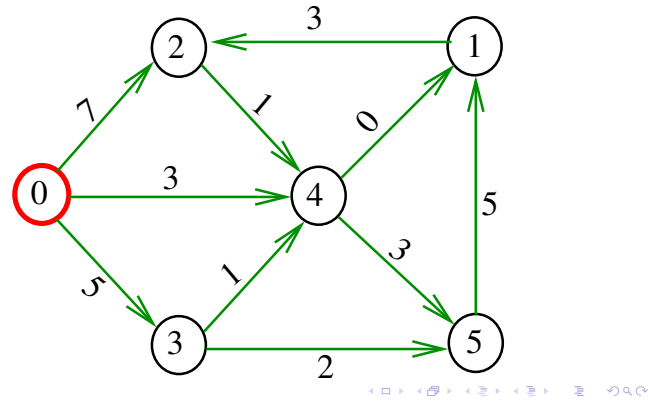
Dado um vértice **s** de um digrafo com custos **não-negativos** nos arcos, encontrar uma SPT com raiz **s**



Problema da SPT

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

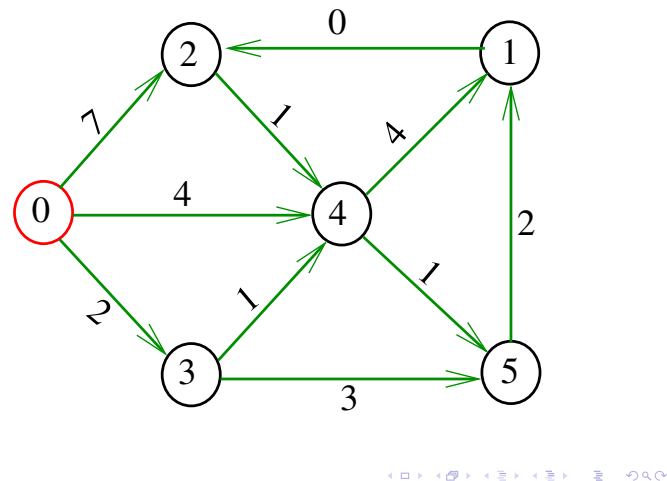
Entra:



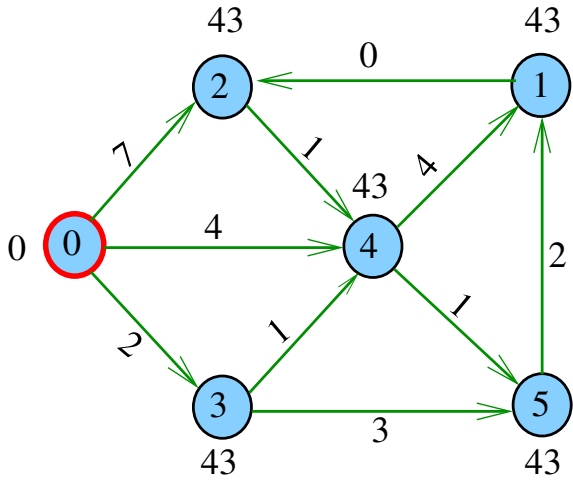
Algoritmo de Dijkstra

S 21.1 e 21.2

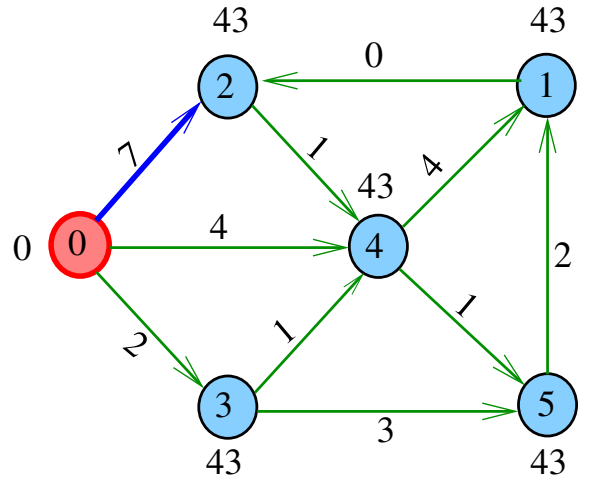
Simulação



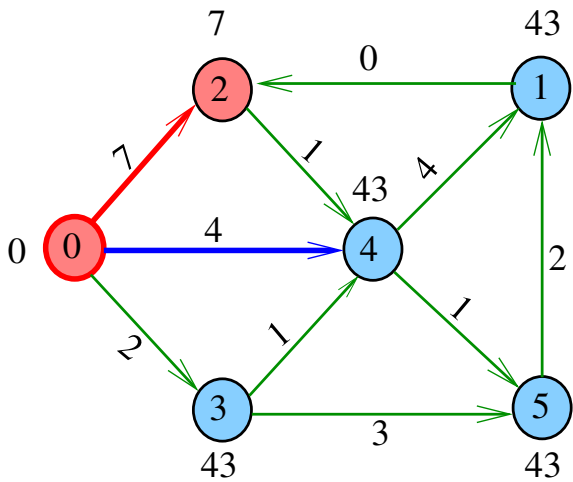
Simulação



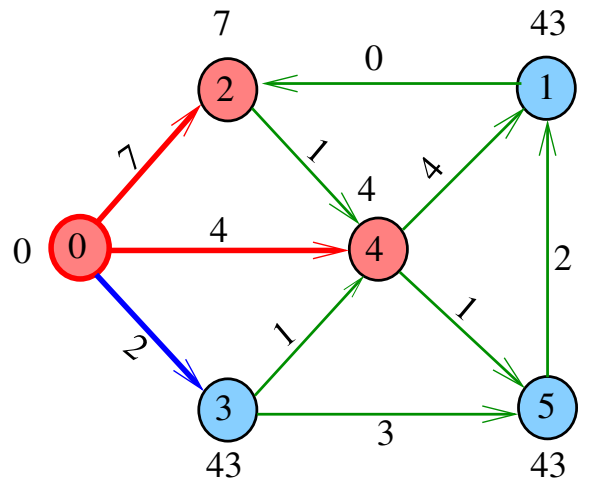
Simulação



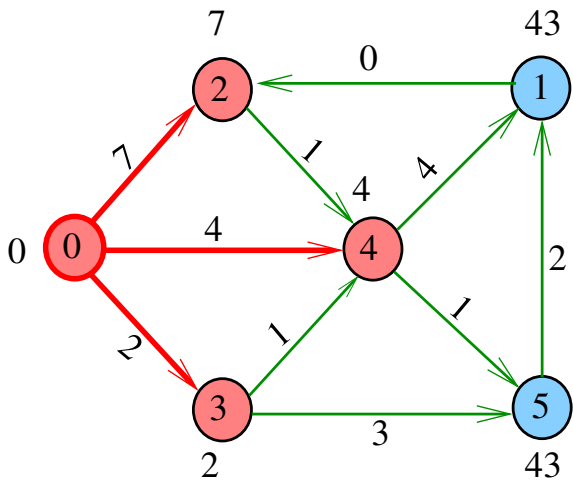
Simulação



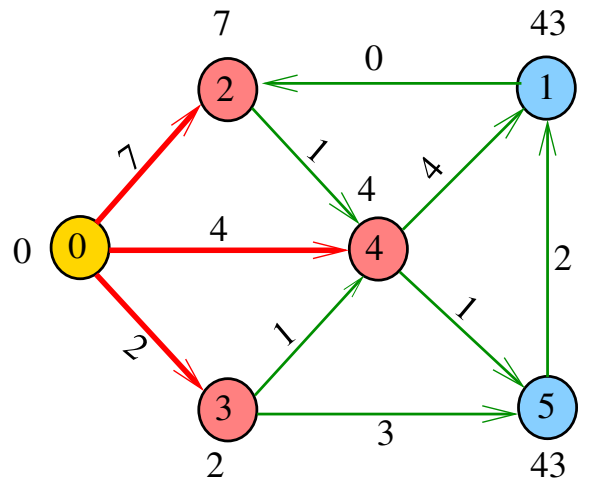
Simulação



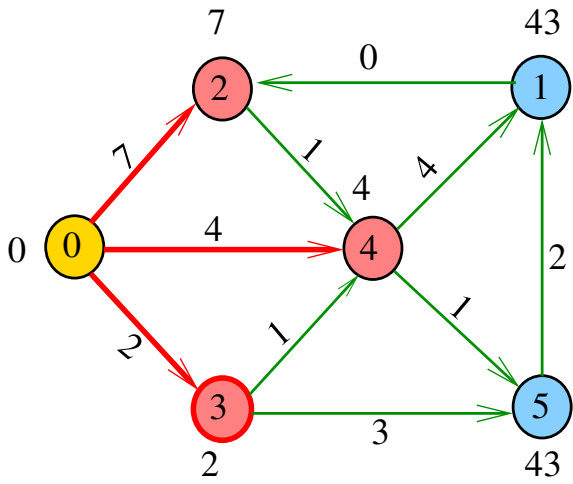
Simulação



Simulação

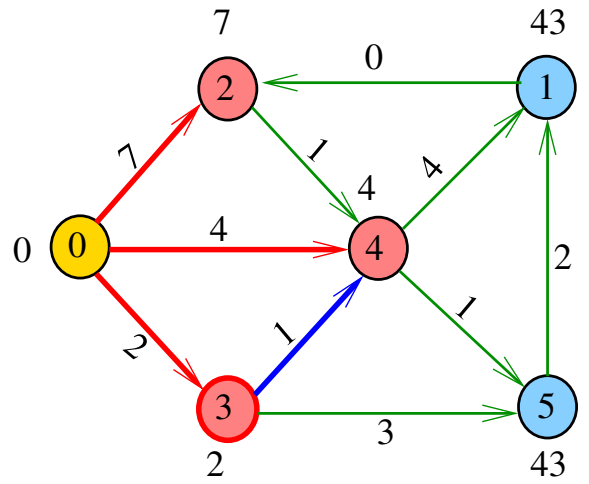


Simulação



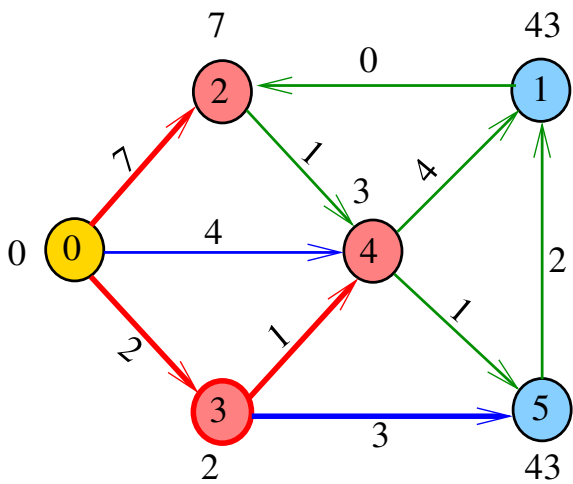
Navigation icons

Simulação



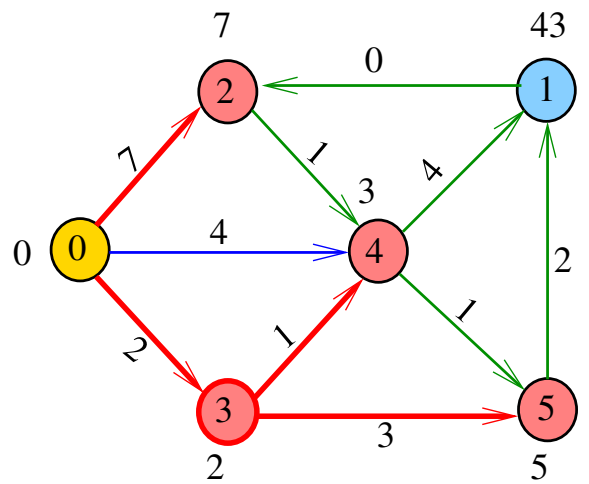
Navigation icons

Simulação



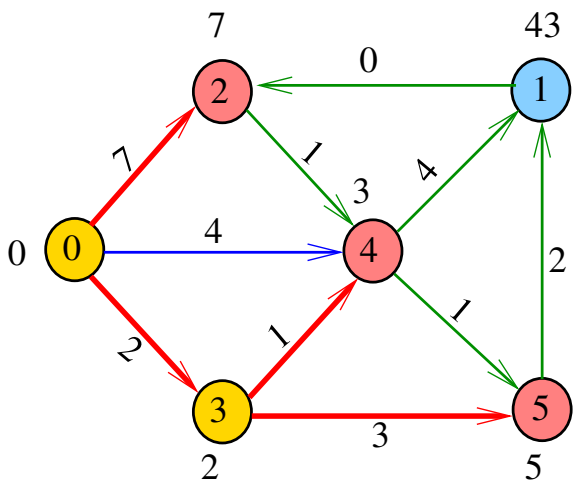
Navigation icons

Simulação



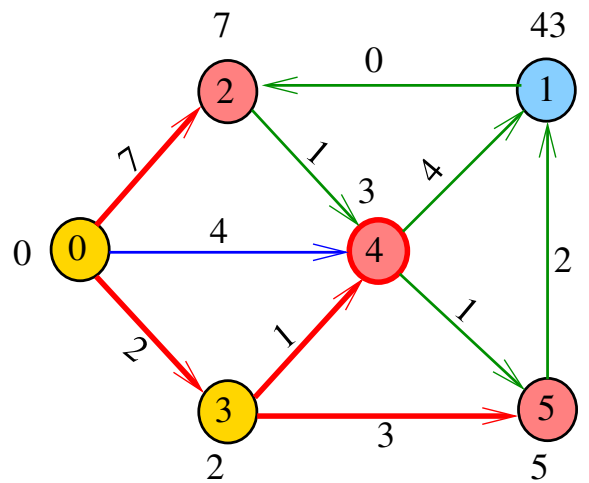
Navigation icons

Simulação



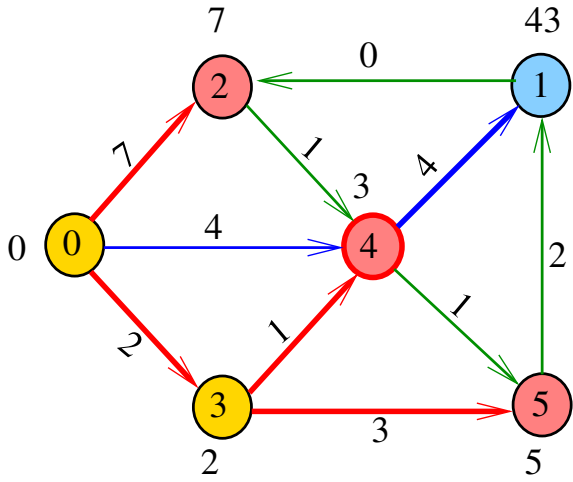
Navigation icons

Simulação

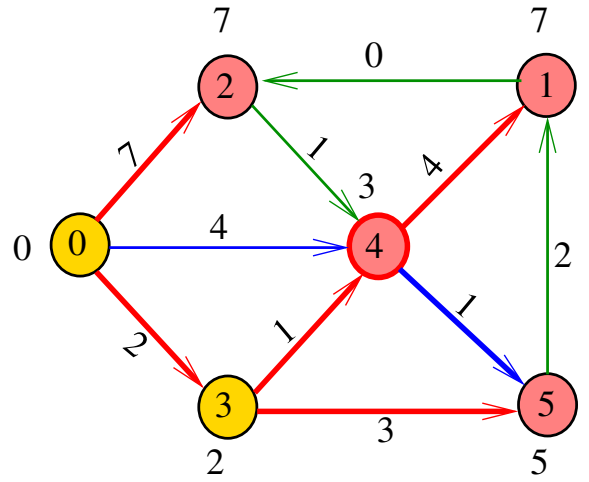


Navigation icons

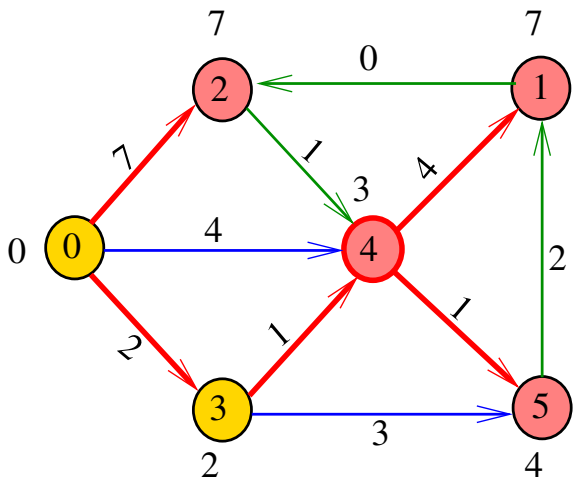
Simulação



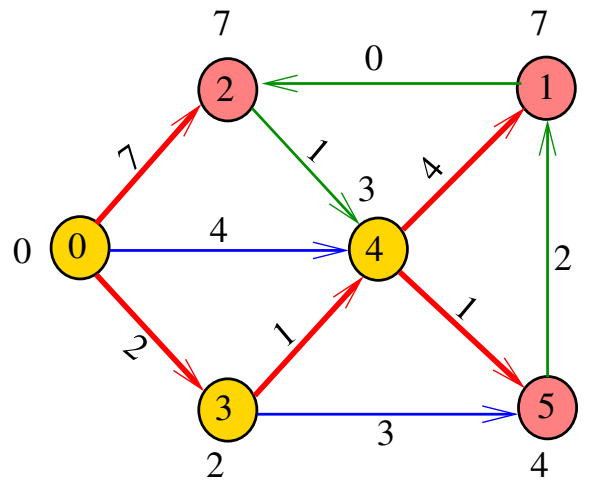
Simulação



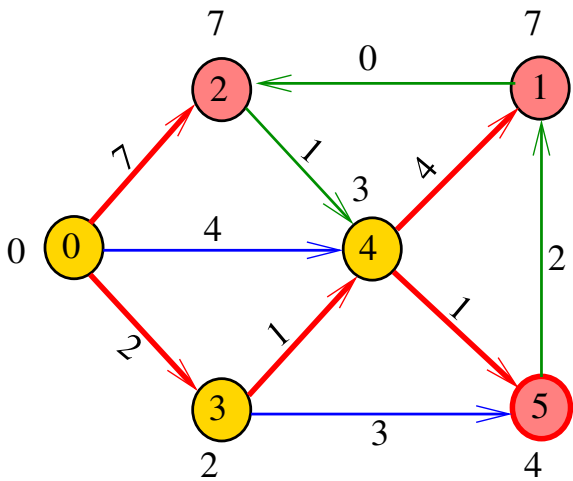
Simulação



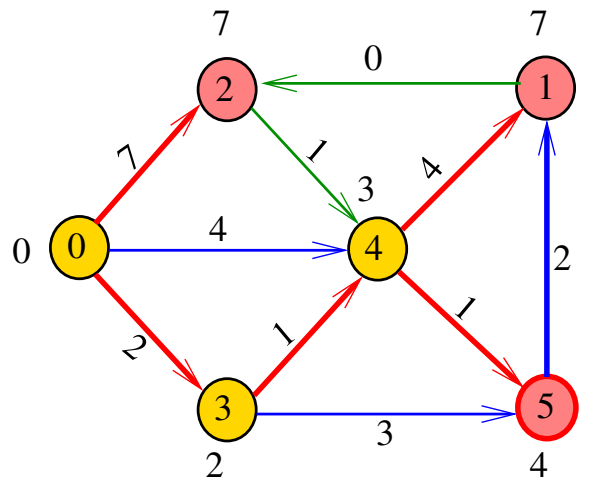
Simulação



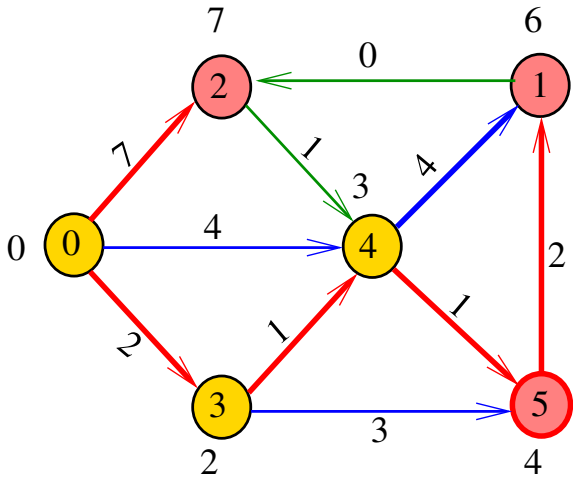
Simulação



Simulação



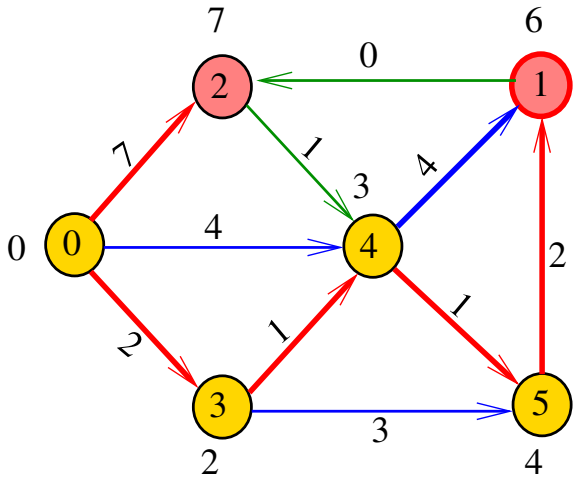
Simulação



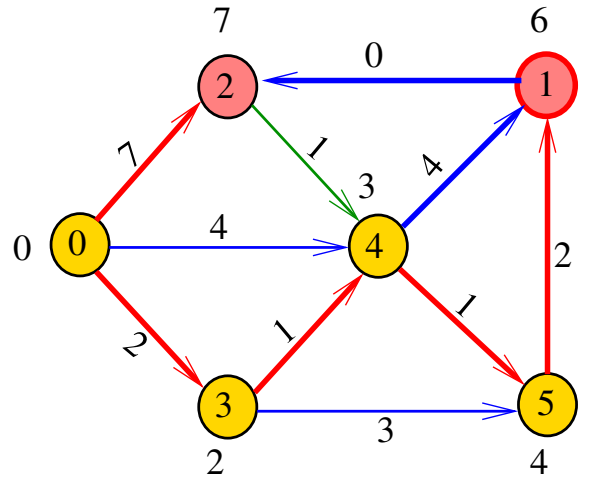
Simulação



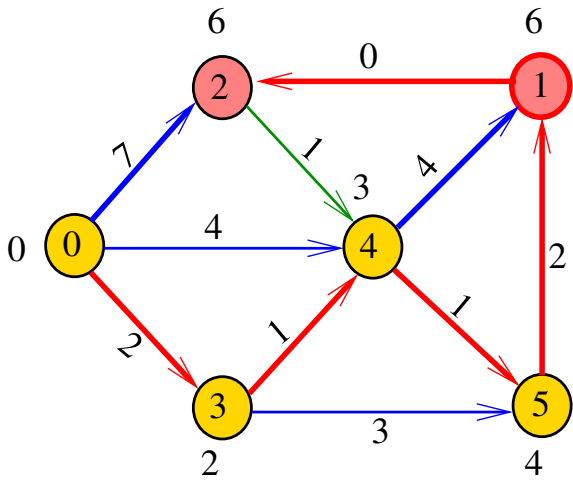
Simulação



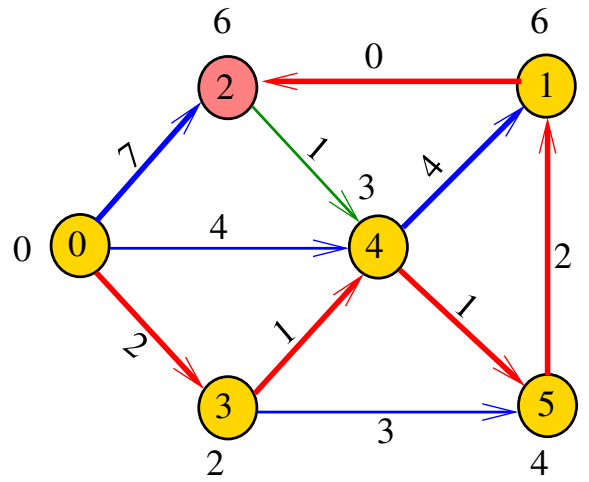
Simulação



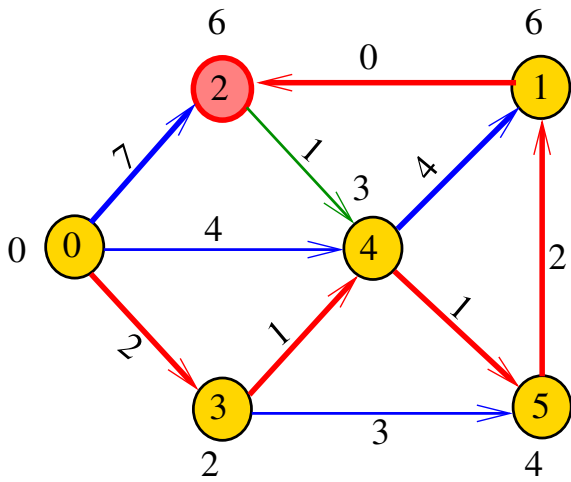
Simulação



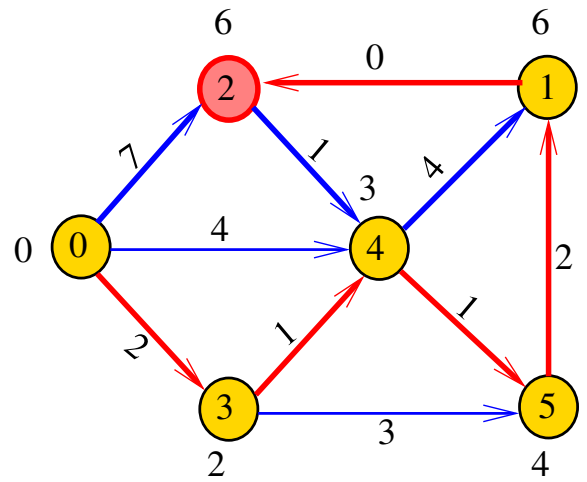
Simulação



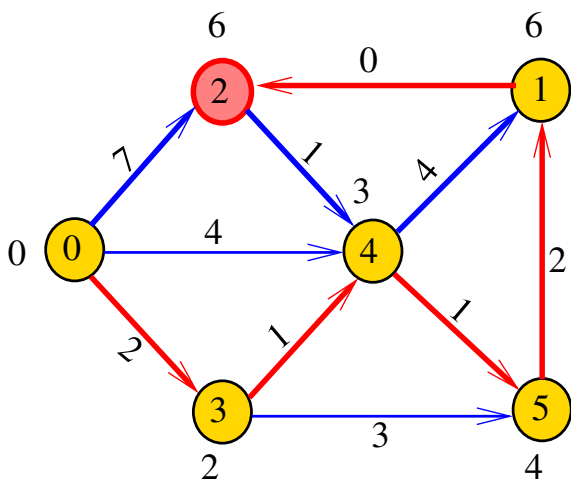
Simulação



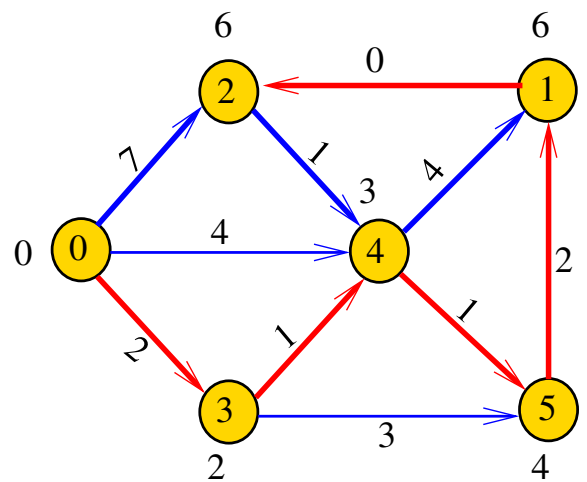
Simulação



Simulação



Simulação



DijkstraSP

Fila com prioridades

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 **edgeTo**
 As distâncias em relação a **s** são armazenadas no vetor **distTo**

```
private double[] distTo;
private DirectedEdge[] edgeTo;
```

A classe DijkstraSP usa uma fila com prioridades
`private IndexMinPQ<Double> pq;`
 A fila é manipulada pelos métodos:

- ▶ `IndexMinPQ<Double>()`: crie uma fila de vértices em que cada vértice **v** tem prioridade `distTo[v]`
- ▶ `isEmpty()`: a fila está vazia?
- ▶ `contains(v)`: **v** está na fila?
- ▶ `insert(v,valor)`: insere **v** com prior. **valor**
- ▶ `delMin()`: retorna o um vértice de prioridade mínima.
- ▶ `decreaseKey(w,valor)`: reorganiza a fila depois que o valor de `distTo[w]` foi decrementado.

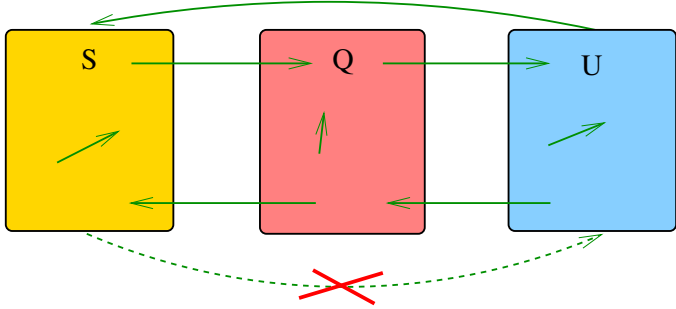
Relações invariantes

S = vértices examinados

Q = vértices visitados = vértices na fila

U = vértices ainda não visitados

(i0) não existe arco v-w com v em S e w em U

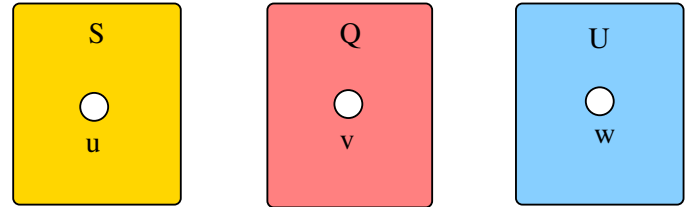


Navigation icons

Relações invariantes

(i1) para cada u em S, v em Q e w em U

$$\text{distTo}[u] \leq \text{distTo}[v] \leq \text{distTo}[w]$$

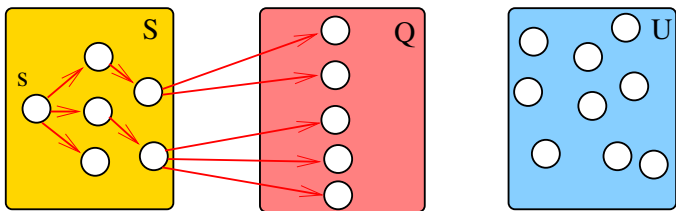


Navigation icons

Relações invariantes

(i2) O vetor edgeTo restrito aos vértices de S e Q

determina um **árborescência com raiz s**

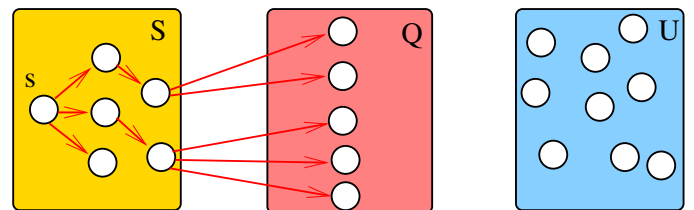


Navigation icons

Relações invariantes

(i3) Para arco v-w na arborescência vale que

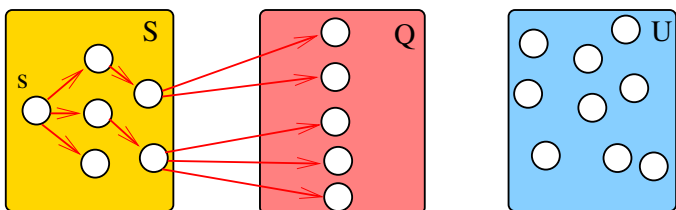
$$\text{distTo}[w] = \text{distTo}[v] + \text{custo do arco } vw$$



Navigation icons

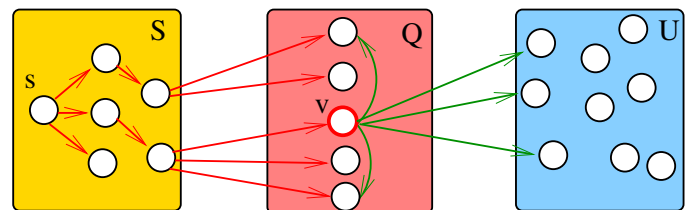
Relações invariantes

(i3) Para cada vértice v em S vale que $\text{distTo}[v]$ é o custo de um caminho mínimo de s a v.



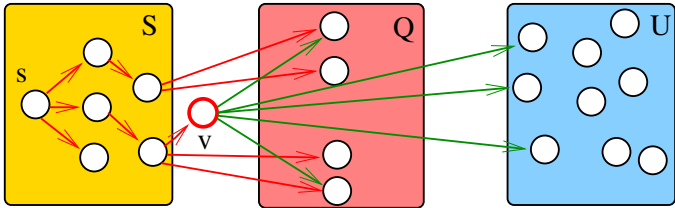
Navigation icons

Iteração



Navigation icons

Iteração



Navigation icons

Consumo de tempo

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

- 1 execução de `IndexMinPQ<Double>`,
- $O(V)$ execuções de `insert()`,
- $O(V)$ execuções de `isEmpty()`,
- $O(V)$ execuções de `contains()`,
- $O(V)$ execuções de `delMin()`, e
- $O(E)$ execuções de `decreaseKey()`.

Navigation icons

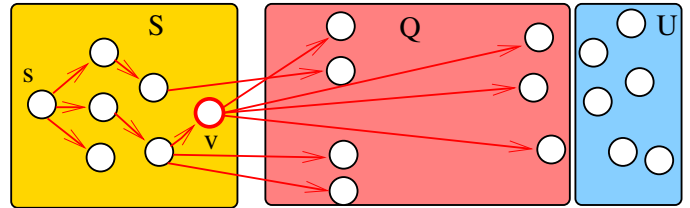
Conclusão

O consumo de `DijkstraSP` é $O(E \lg V)$.

Para **grafos densos** podemos alcançar consumo de tempo é ótimo ... detalhes MAC0328.

Navigation icons

Iteração



Navigation icons

Consumo de tempo Min-Heap

| | |
|---------------------------------------|-----------------|
| <code>IndexMinPQ<Double></code> | $\Theta(1)$ |
| <code>isEmpty</code> | $\Theta(1)$ |
| <code>insert</code> | $\Theta(\lg V)$ |
| <code>delMin</code> | $O(\lg V)$ |
| <code>decreaseKey</code> | $\Theta(\lg V)$ |
| <code>contains</code> | $\Theta(1)$ |

Navigation icons

Consumo de tempo Min-Heap

| | heap | d -heap | fibonacci heap |
|--------------------------|--------------|-----------------|------------------|
| <code>insert</code> | $O(\lg V)$ | $O(\log_D V)$ | $O(1)$ |
| <code>delMin</code> | $O(\lg V)$ | $O(\log_D V)$ | $O(\lg V)$ |
| <code>decreaseKey</code> | $O(\lg V)$ | $O(\log_D V)$ | $O(1)$ |
| <code>dijkstra</code> | $O(E \lg V)$ | $O(E \log_D V)$ | $O(E + V \lg V)$ |

Navigation icons

Consumo de tempo Min-Heap

| | bucket heap | radix heap |
|-------------|-------------|--------------------|
| insert | $O(1)$ | $O(\lg(VC))R$ |
| delMin | $O(C)$ | $O(\lg(VC))$ |
| decreaseKey | $O(1)$ | $O(E + V \lg(VC))$ |
| dijkstra | $O(E + VC)$ | $O(E + V \lg(VC))$ |

C = maior custo de um arco.

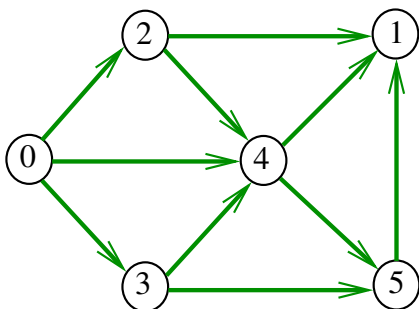
Caminhos mínimos em DAGs

S 19.6

DAGs

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

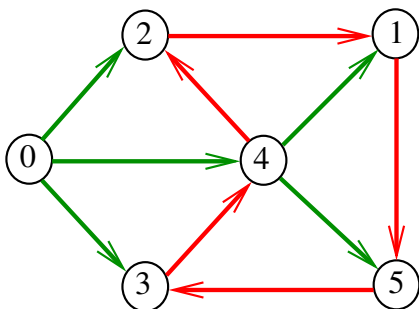
Exemplo: um digrafo acíclico



DAGs

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

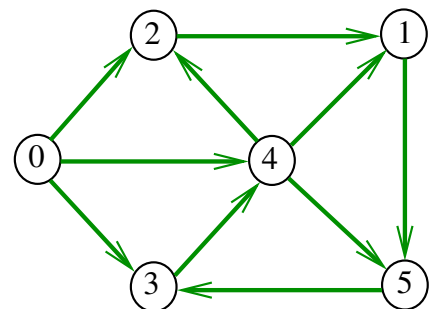
Exemplo: um digrafo que **não** é acíclico



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

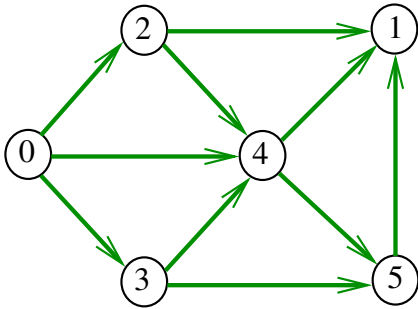
$$ts[i] \rightarrow ts[j] \text{ com } i < j$$

$ts[0]$ 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:

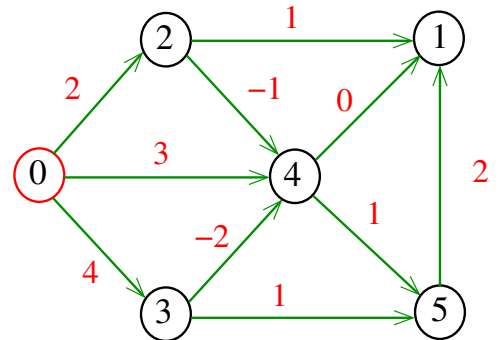
Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar, para cada vértice t que pode ser alcançado a partir de s , um **caminho simples mínimo** de s a t

Problema:

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

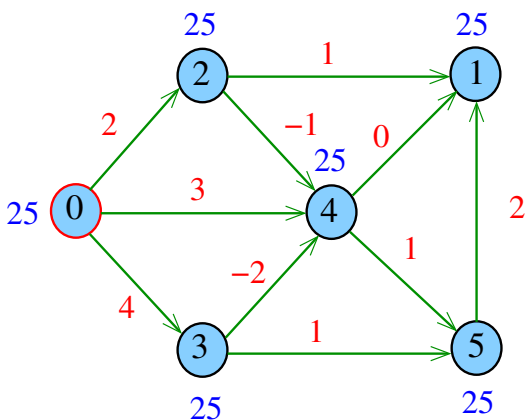
Simulação

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



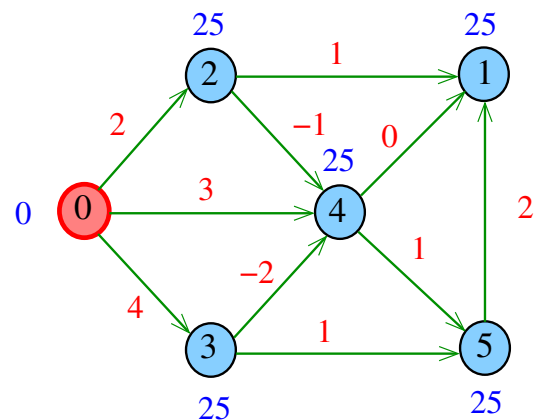
Simulação

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



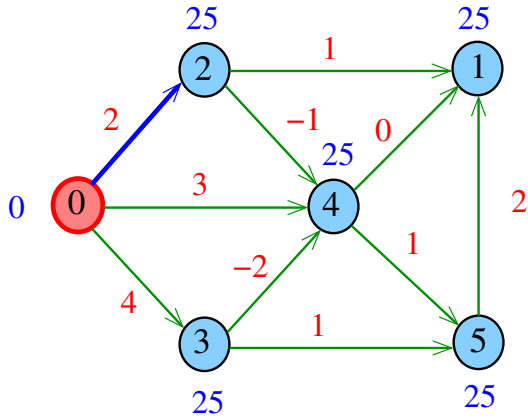
Simulação

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



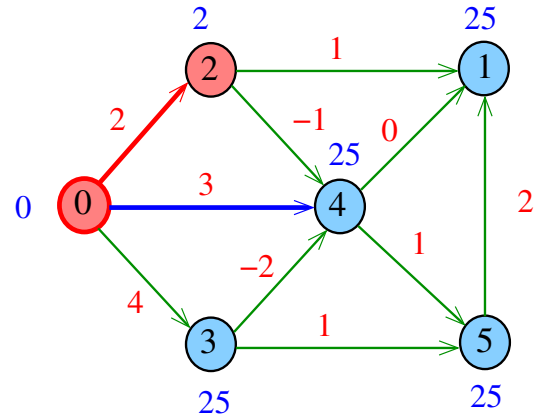
Simulação

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



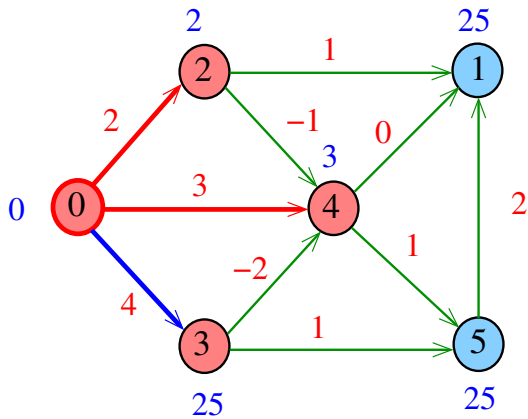
Simulação

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



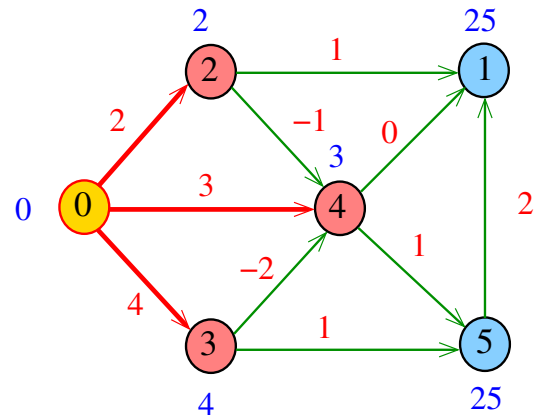
Simulação

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



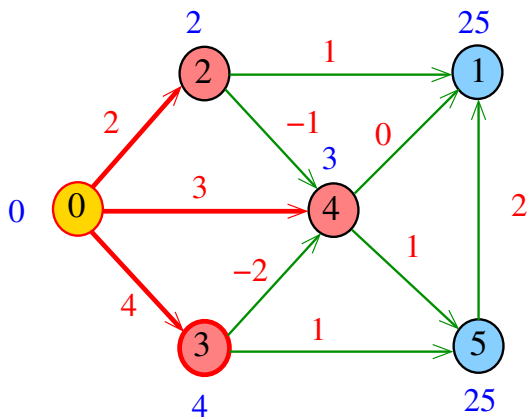
Simulação

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



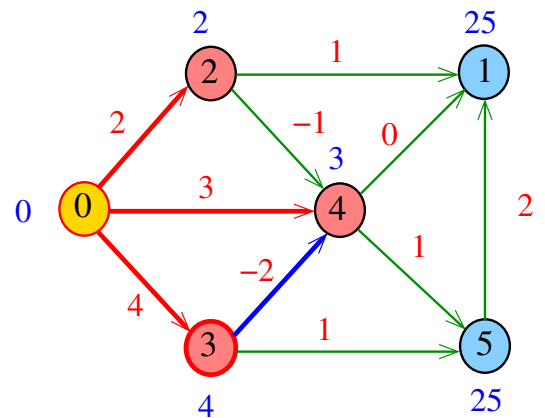
Simulação

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



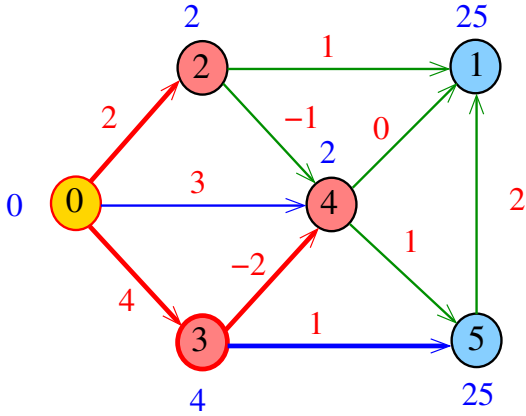
Simulação

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



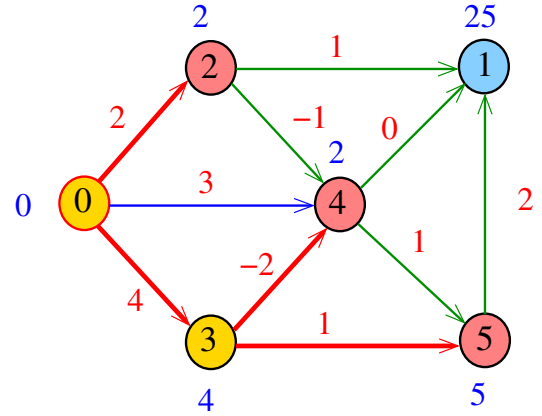
Simulação

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



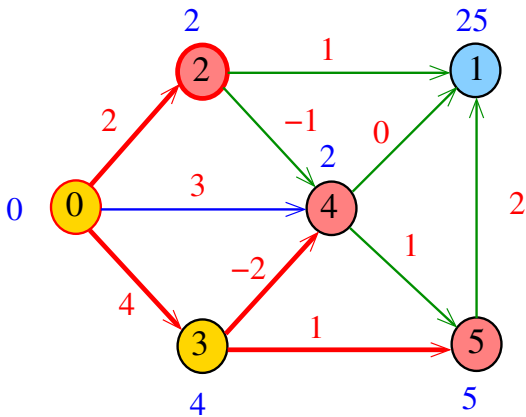
Simulação

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



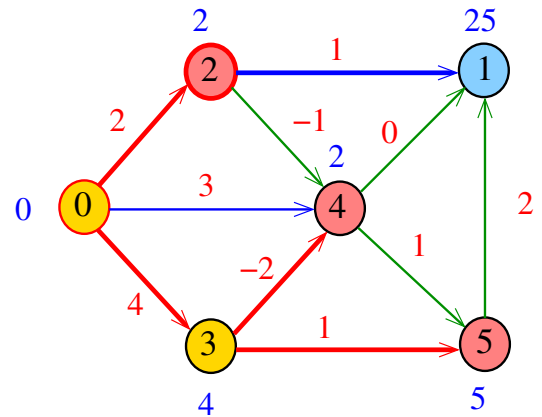
Simulação

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



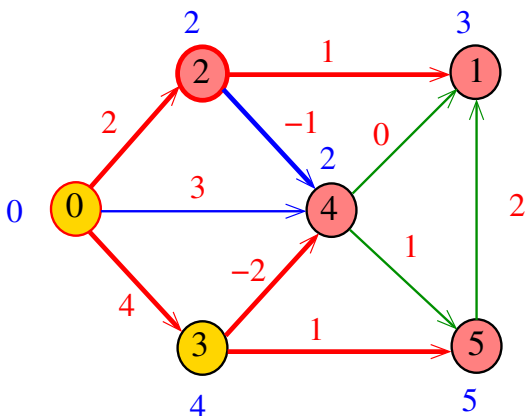
Simulação

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



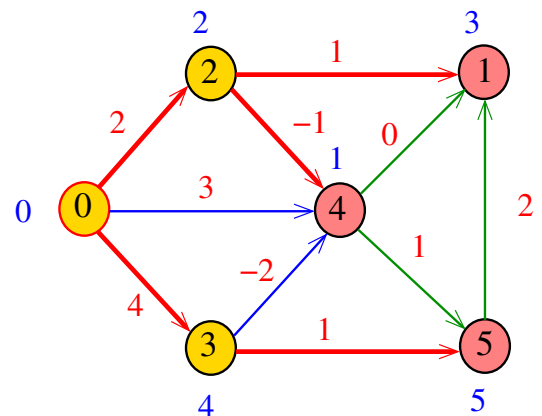
Simulação

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



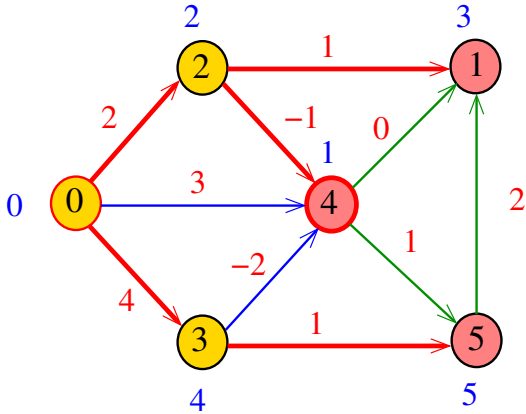
Simulação

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



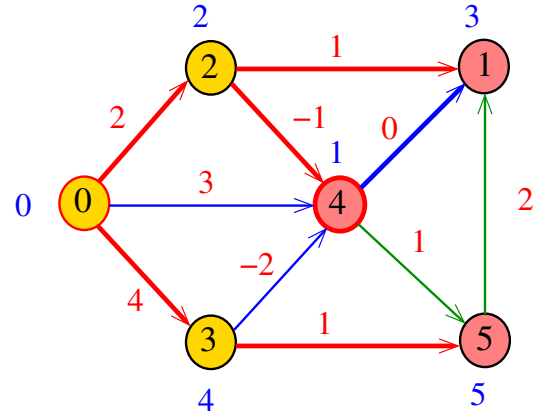
Simulação

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



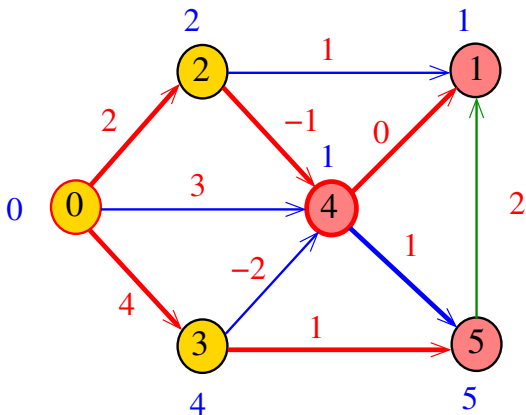
Simulação

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



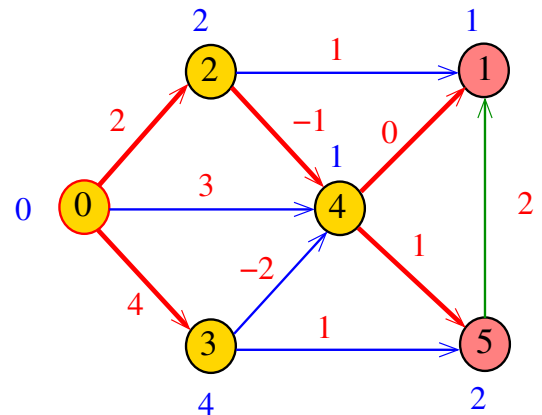
Simulação

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



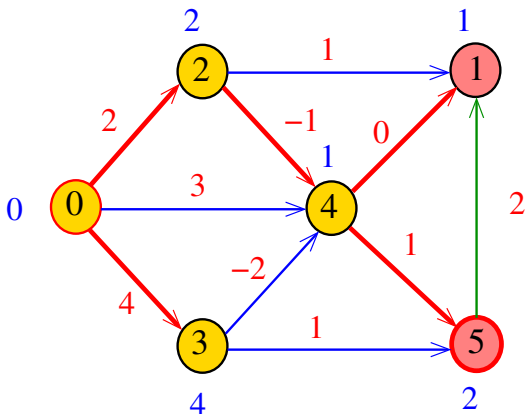
Simulação

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



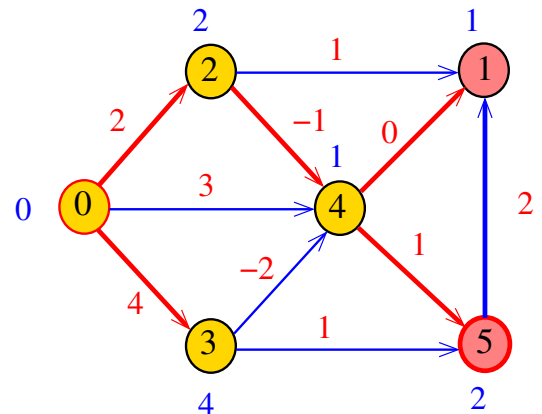
Simulação

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



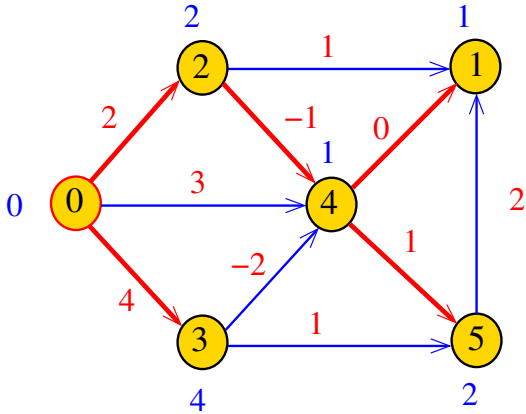
Simulação

| | | | | | | |
|-------|---|---|---|---|---|---|
| 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 |



Consumo de tempo

Navigation icons

AcyclicSP

A classe **AcyclicSP** 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 **distTo[t]**.

```
void
AcyclicSP (EdgeWeightedDigraph G,
ints);
```

Navigation icons

Caminhos máximos em DAGs

Do ponto de vista computacional, o problema de encontrar um caminho simples de **custo máximo** num digrafo 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.

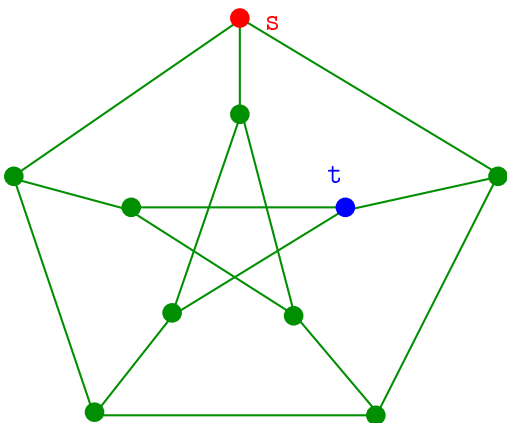
Navigation icons

Navigation icons

O consumo de tempo de **AcyclicSP** é $O(V + E)$.

Caminhos hamiltonianos

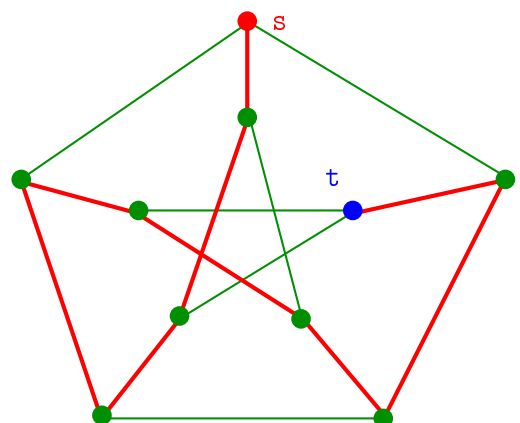
Problema: Dados vértices **s** e **t** de um grafo encontrar um **caminho hamiltoniano** de **s** e **t**



Navigation icons

Caminhos hamiltonianos

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



Navigation icons