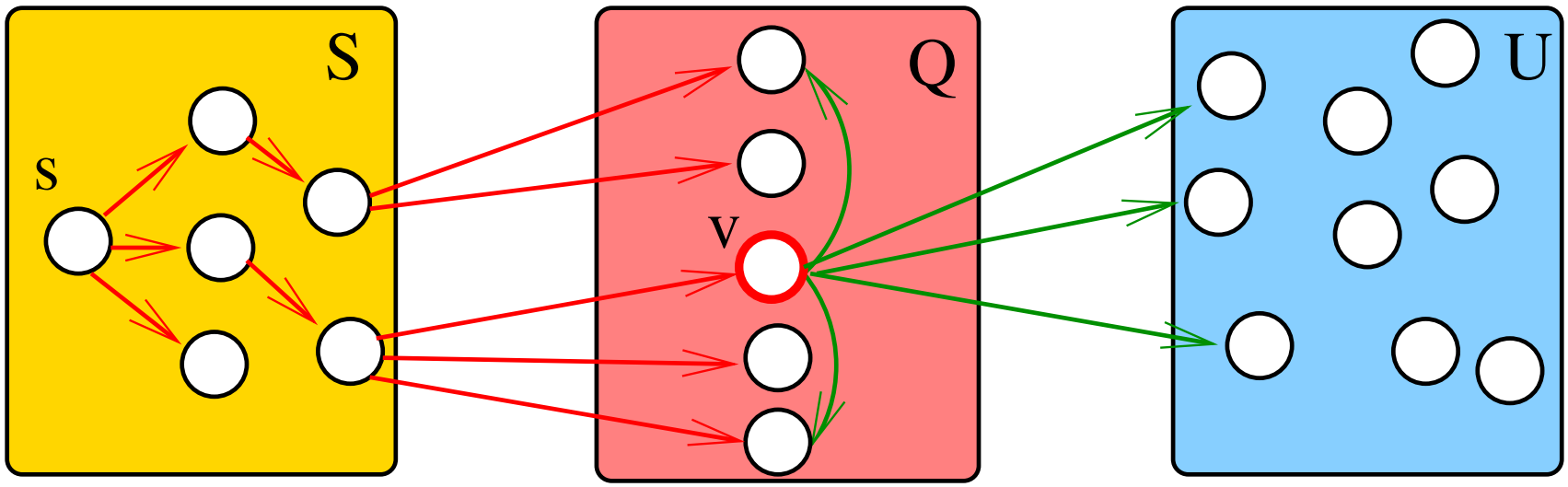
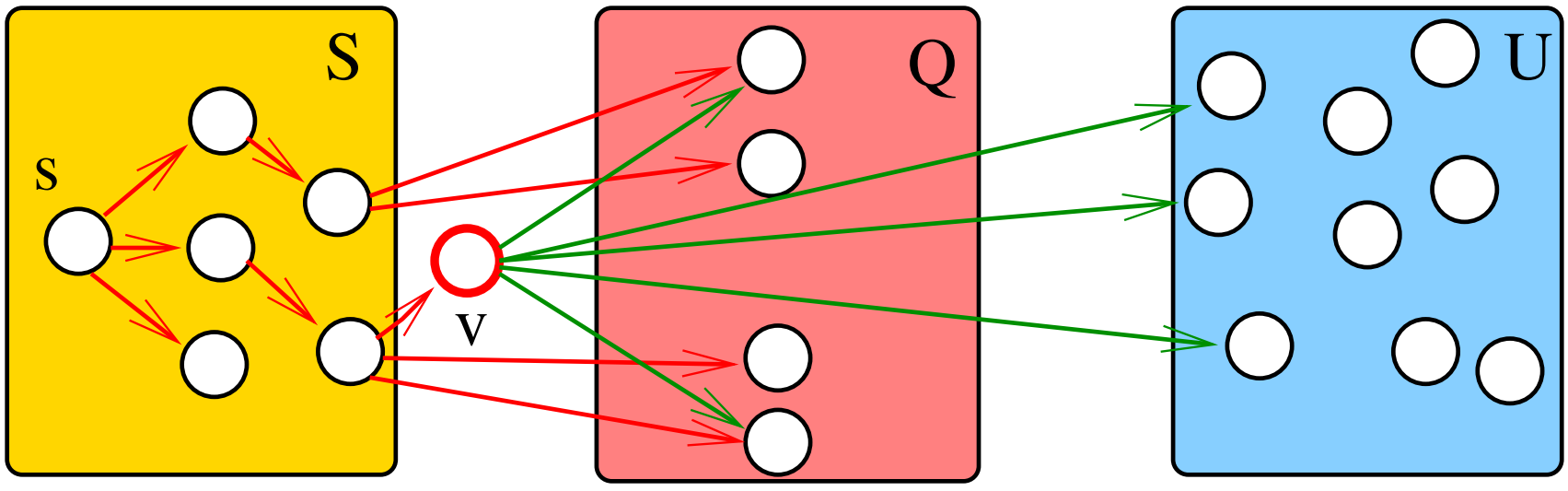
Compacto dos melhores momentos

AULA 22

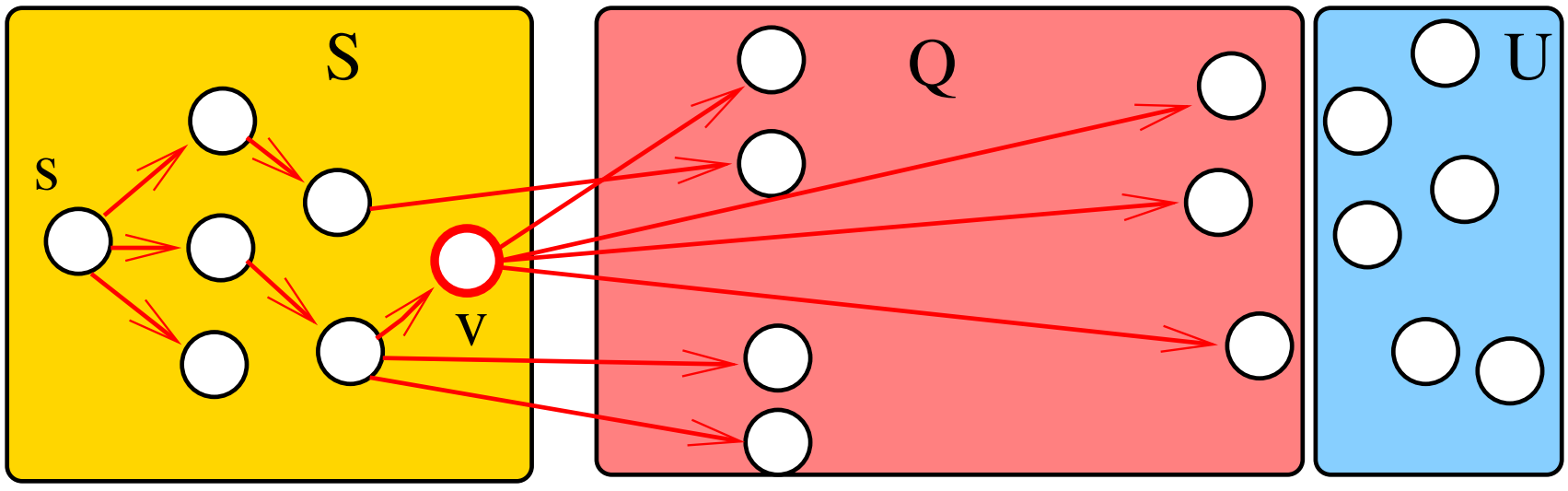
Dijkstra: iteração



Dijkstra: iteração



Dijkstra: iteração



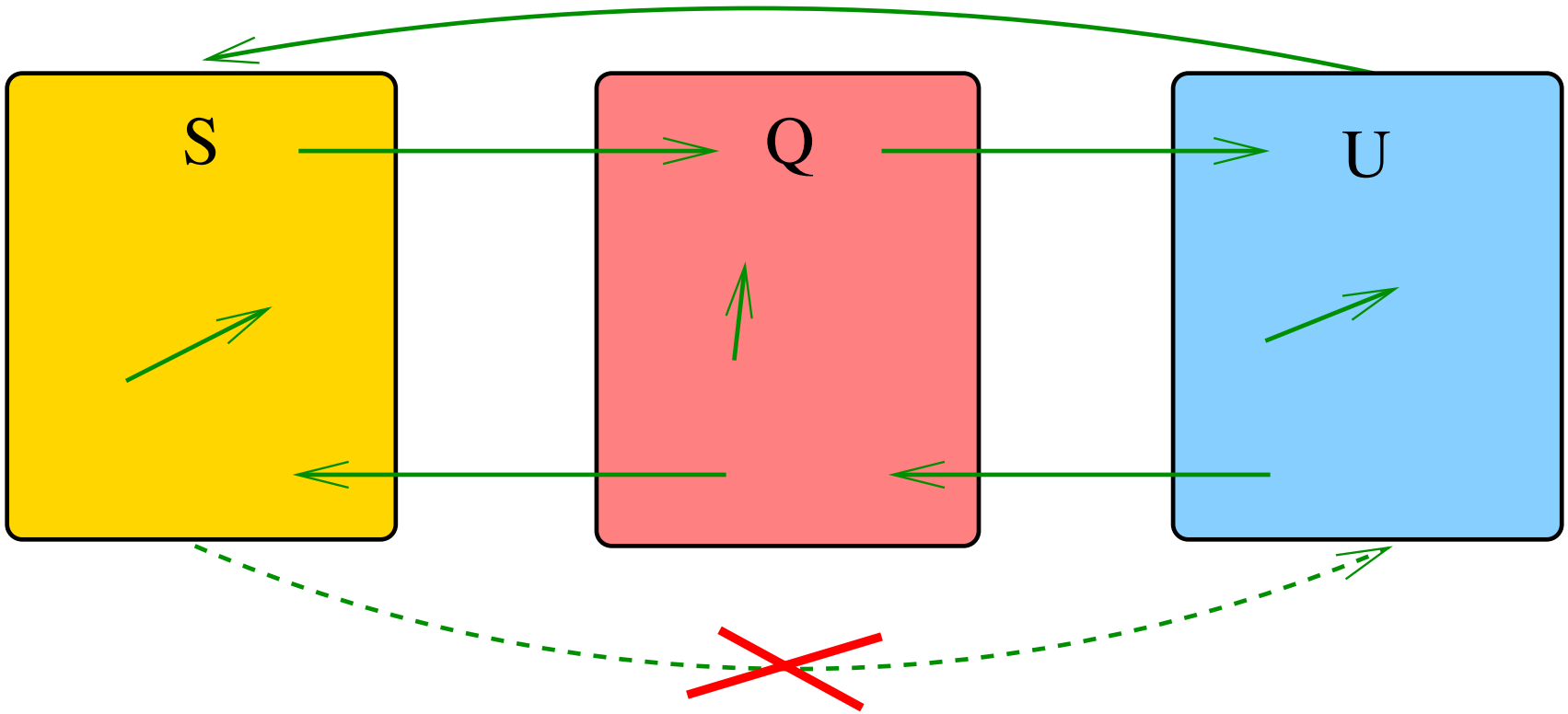
Dijkstra: relações invariantes

S = vértices examinados

Q = vértices visitados = vértices na fila

U = vértices ainda não visitados

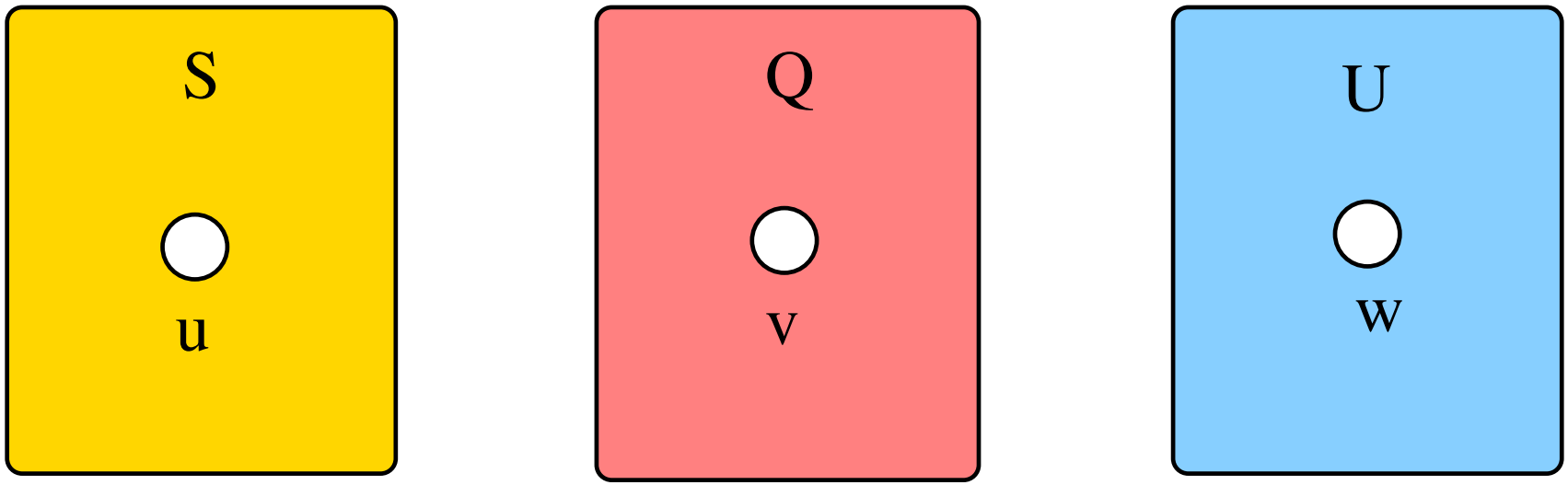
(i0) não existe arco v-w com v em S e w em U



Dijkstra: relações invariantes

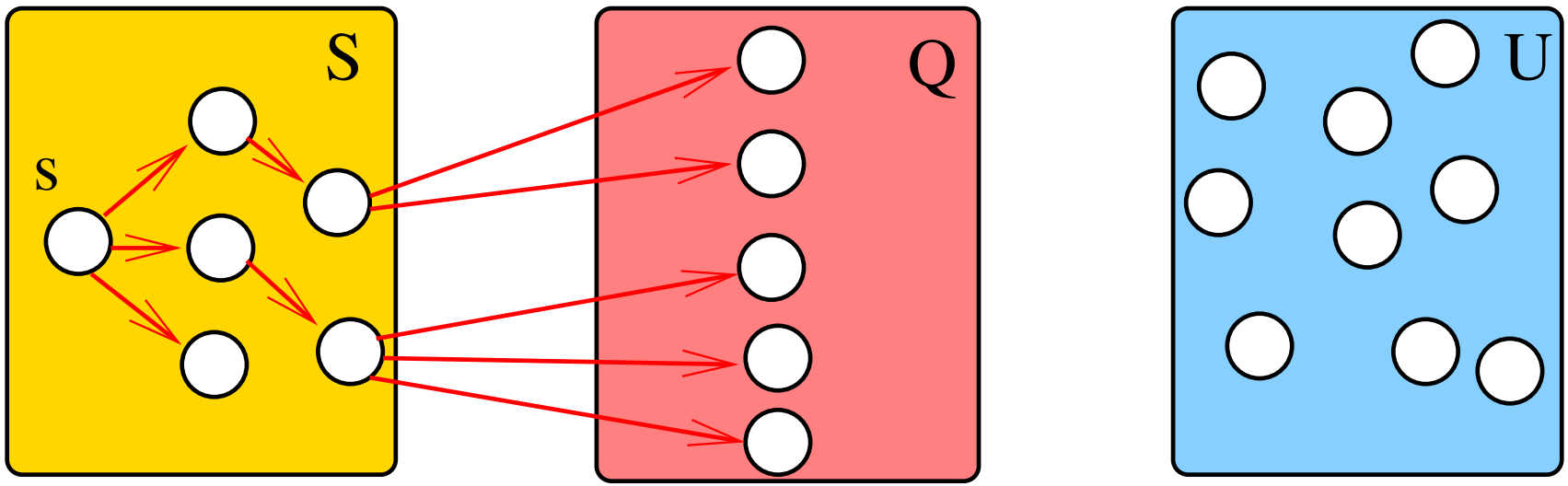
(i1) para cada u em S, v em Q e w em U

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



Dijkstra: relações invariantes

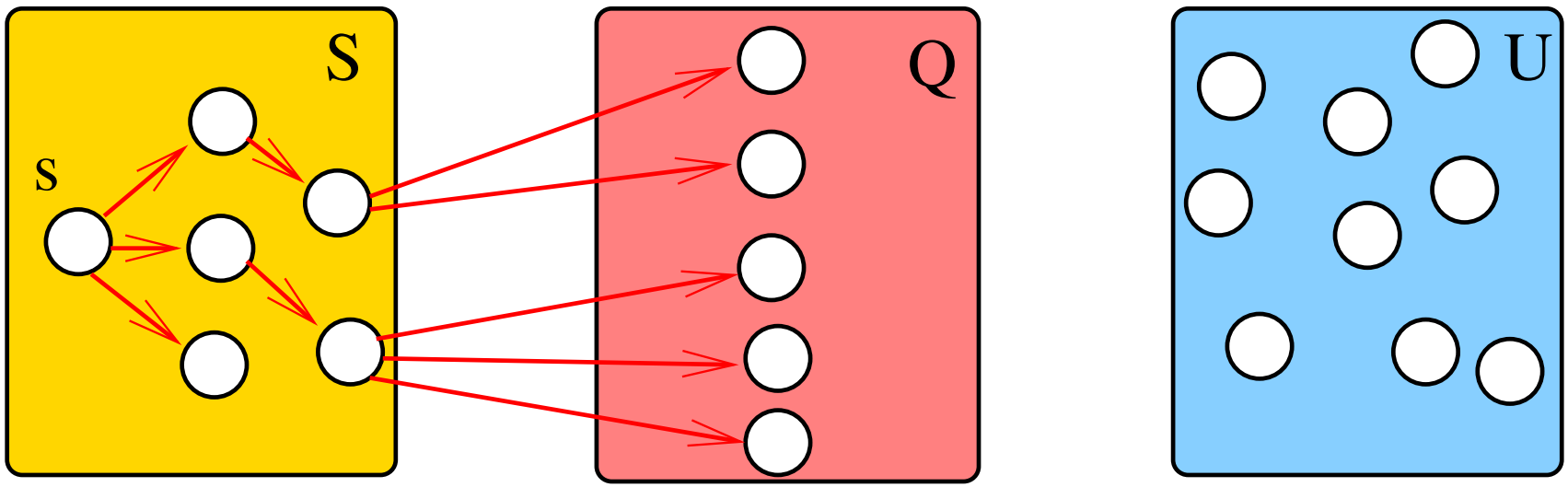
(i2) O vetor edgeTo restrito aos vértices de S e Q determina um árborescência com raiz s



Dijkstra: relações invariantes

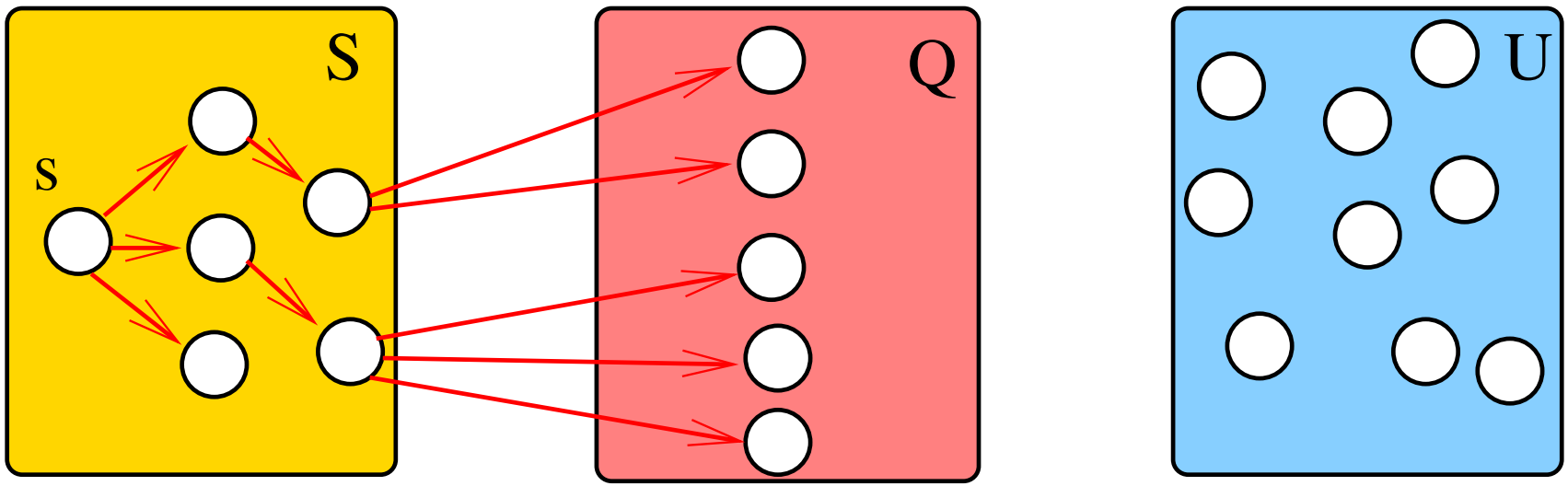
(i3) Para arco v-w na arborescência vale que

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



Dijkstra: relações invariantes

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



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>`,
$\leq V$ execuções de `insert()`,
$\leq V$ execuções de `isEmpty()`,
$\leq V$ execuções de `delMin()`, e
$\leq E$ execuções de `contains()`,
$\leq E$ execuções de `decreaseKey()`.

Consumo de tempo Min-Heap

`IndexMinPQ`	$\Theta(V)$
`isEmpty`	$\Theta(1)$
`insert`	$\Theta(\lg\ V)$
`delMin`	$\mathrm{O}(\lg\ V)$
`decreaseKey`	$\Theta(\lg\ V)$
`contains`	$\Theta(1)$

Conclusão

O consumo de `Dijkstra` é $\mathrm{O}(\mathtt{E} \lg \mathtt{V})$.

Para **grafos densos** podemos alcançar consumo de tempo ótimo . . . detalhes **MAC0328** Algoritmos em Grafos.

Consumo de tempo Min-Heap

	heap	d-heap	fibonacci heap
`insert`	$\mathrm{O}(\lg \mathtt{V})$	$\mathrm{O}(\log_D \mathtt{V})$	$\mathrm{O}(1)$
`delMin`	$\mathrm{O}(\lg \mathtt{V})$	$\mathrm{O}(\log_D \mathtt{V})$	$\mathrm{O}(\lg \mathtt{V})$
`decreaseKey`	$\mathrm{O}(\lg \mathtt{V})$	$\mathrm{O}(\log_D \mathtt{V})$	$\mathrm{O}(1)$
`Dijkstra`	$\mathrm{O}(\mathtt{E} \lg \mathtt{V})$	$\mathrm{O}(\mathtt{E} \log_D \mathtt{V})$	$\mathrm{O}(\mathtt{E} + \mathtt{V} \lg \mathtt{V})$

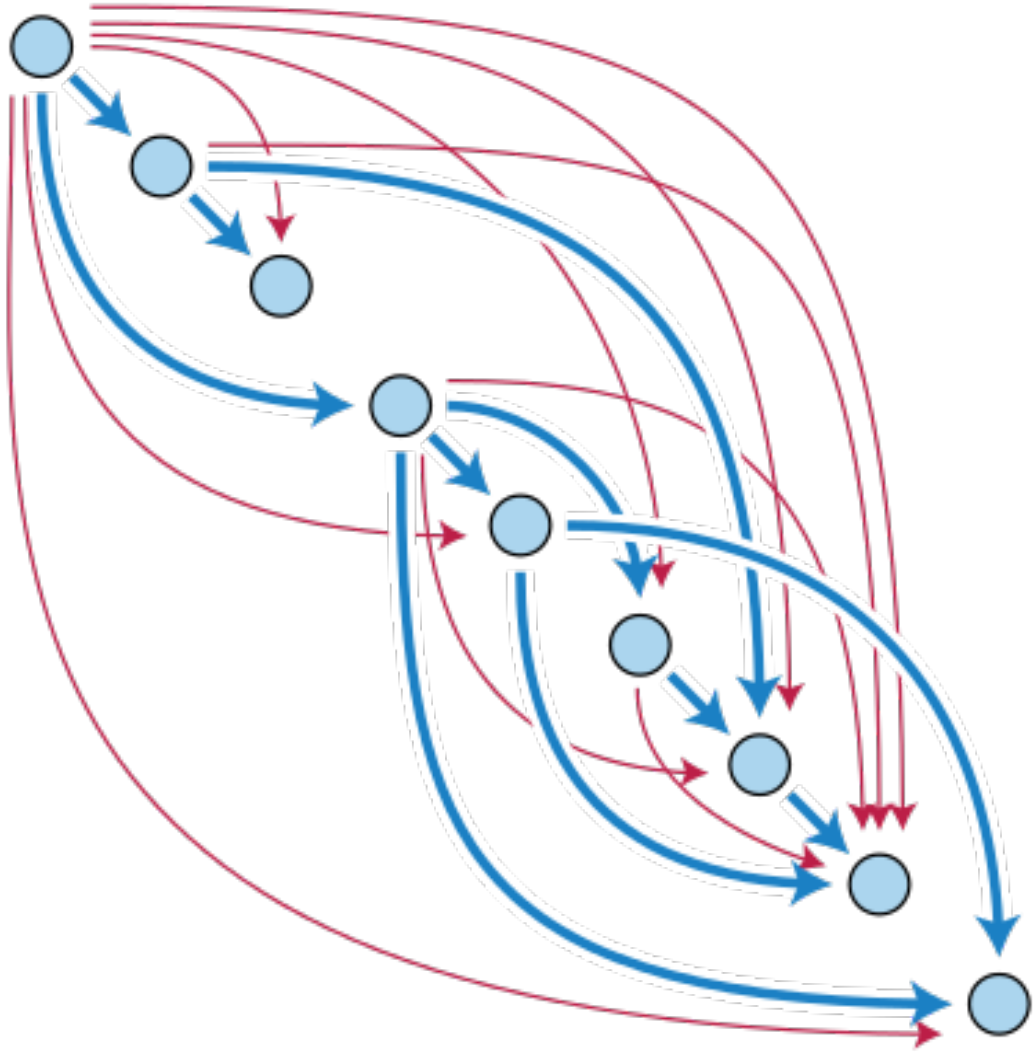
Consumo de tempo Min-heap

	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.

AULA 24

Caminhos mínimos em DAGs

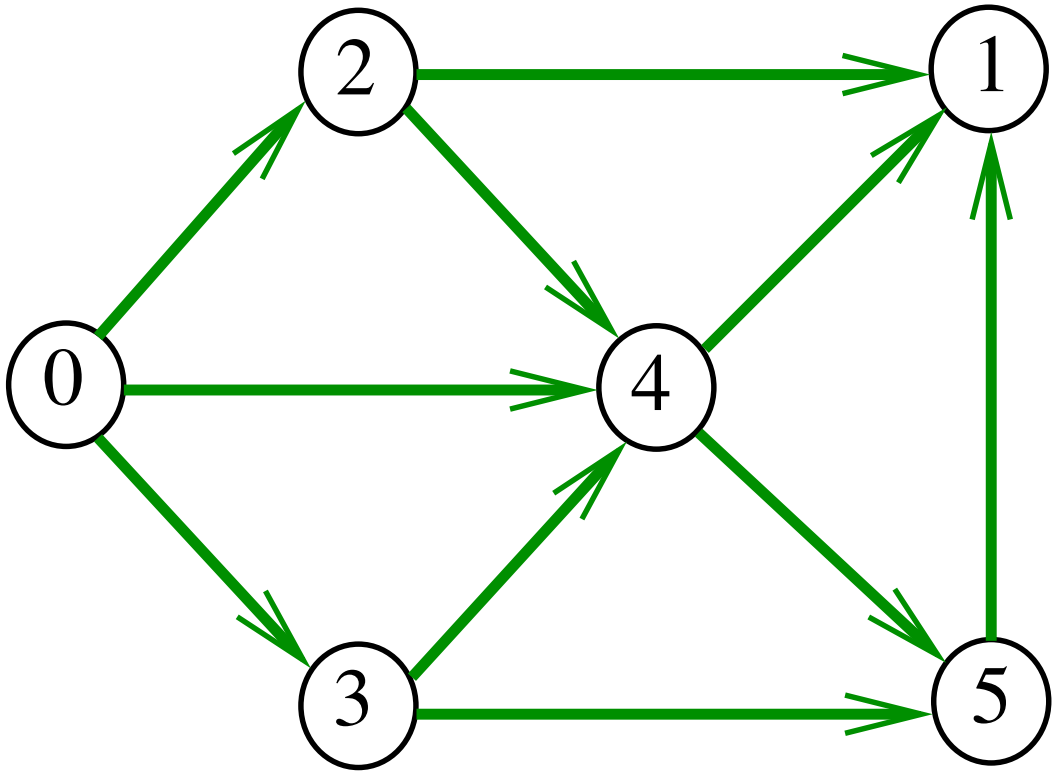


Fonte: Directed acyclic graph

DAGs

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

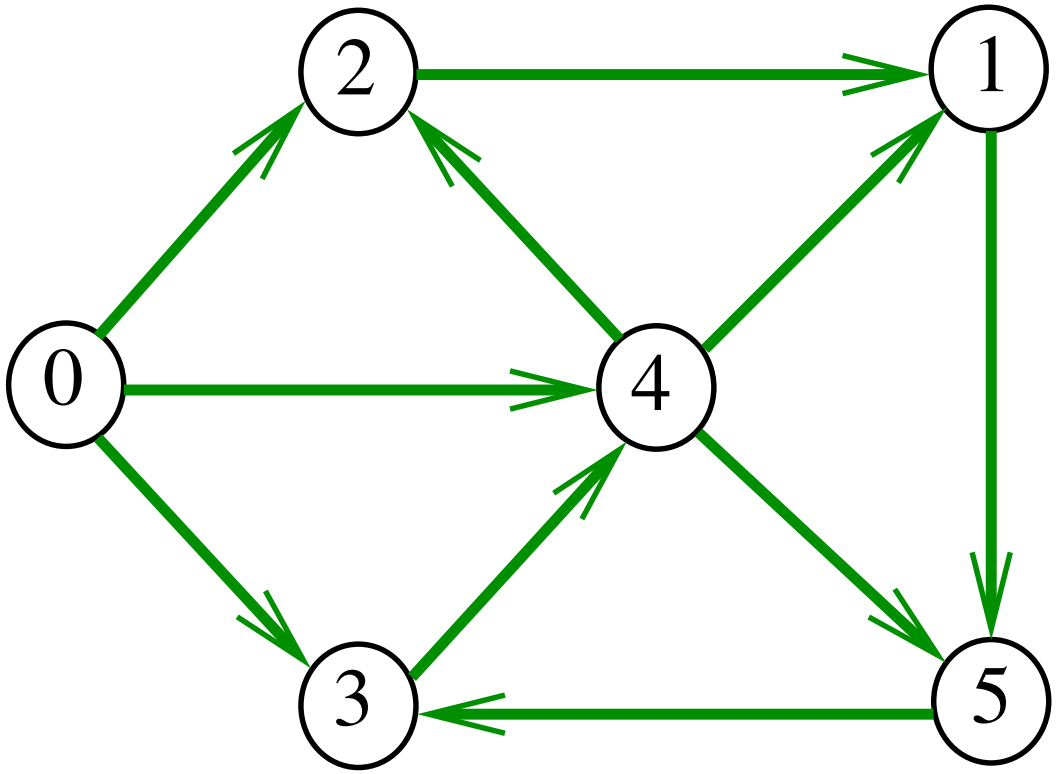
Exemplo: um digrafo acíclico



DAGs

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

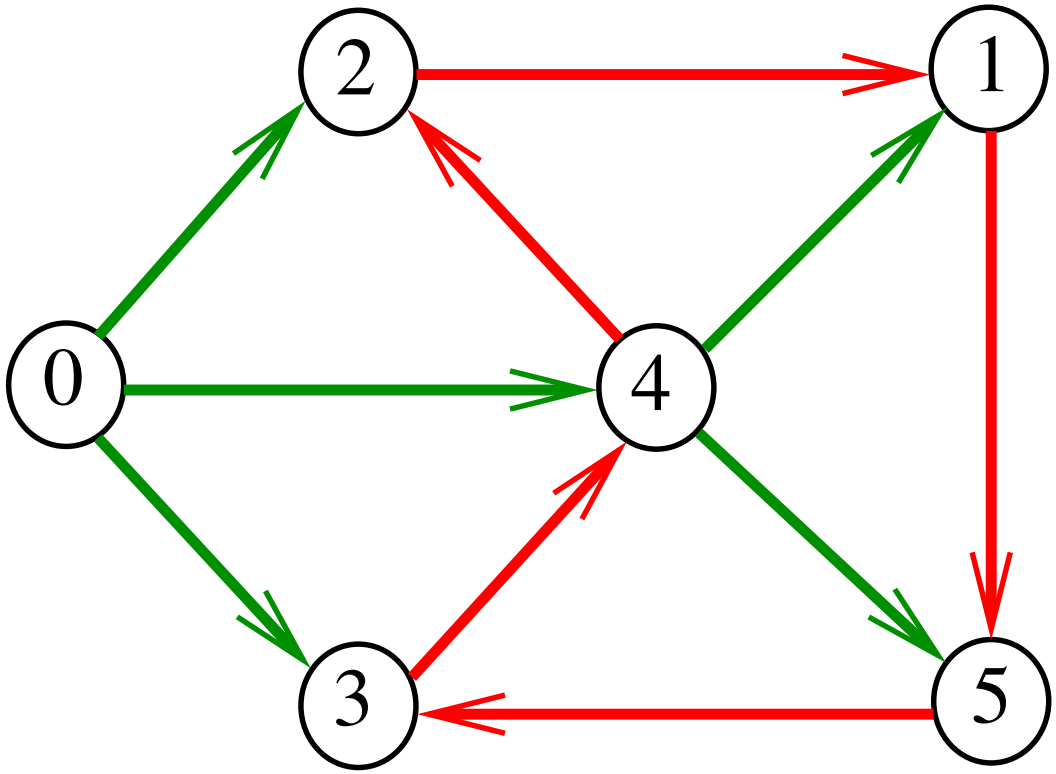
Exemplo: um digrafo que **não** é acíclico



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]`

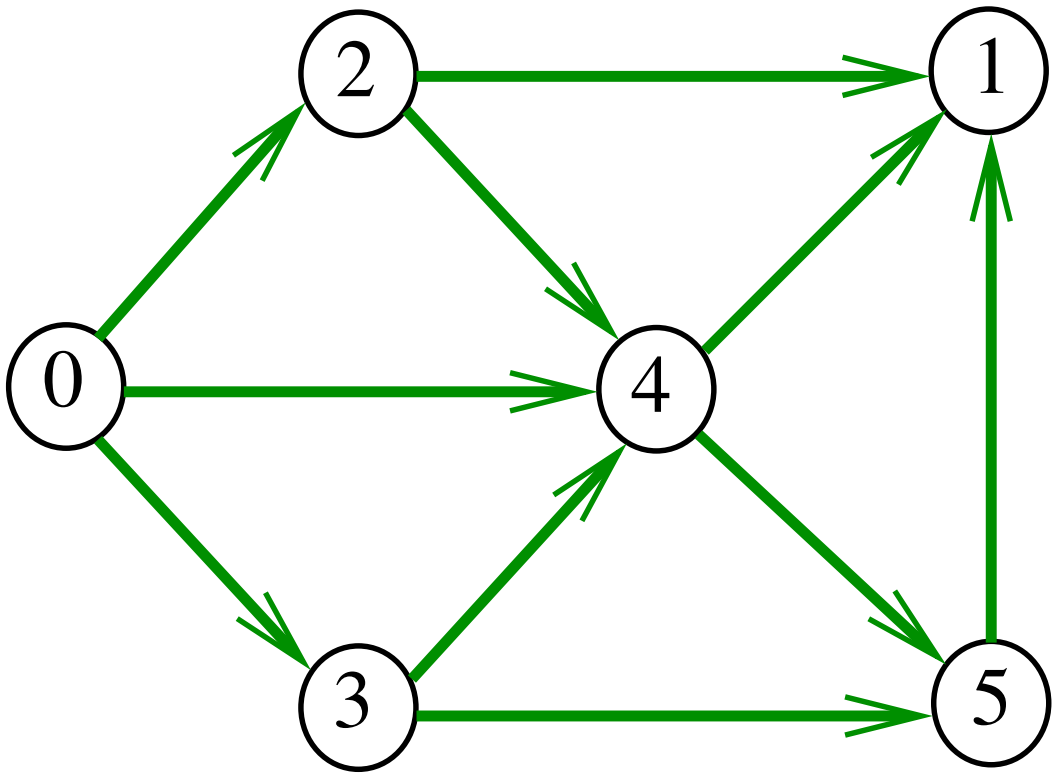
dos seus vértices tal que todo arco tem a forma

`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

Fonte: Well-Known Powerful Yin Yang Symbol Dates Back To Ancient China

Problema

Problema:

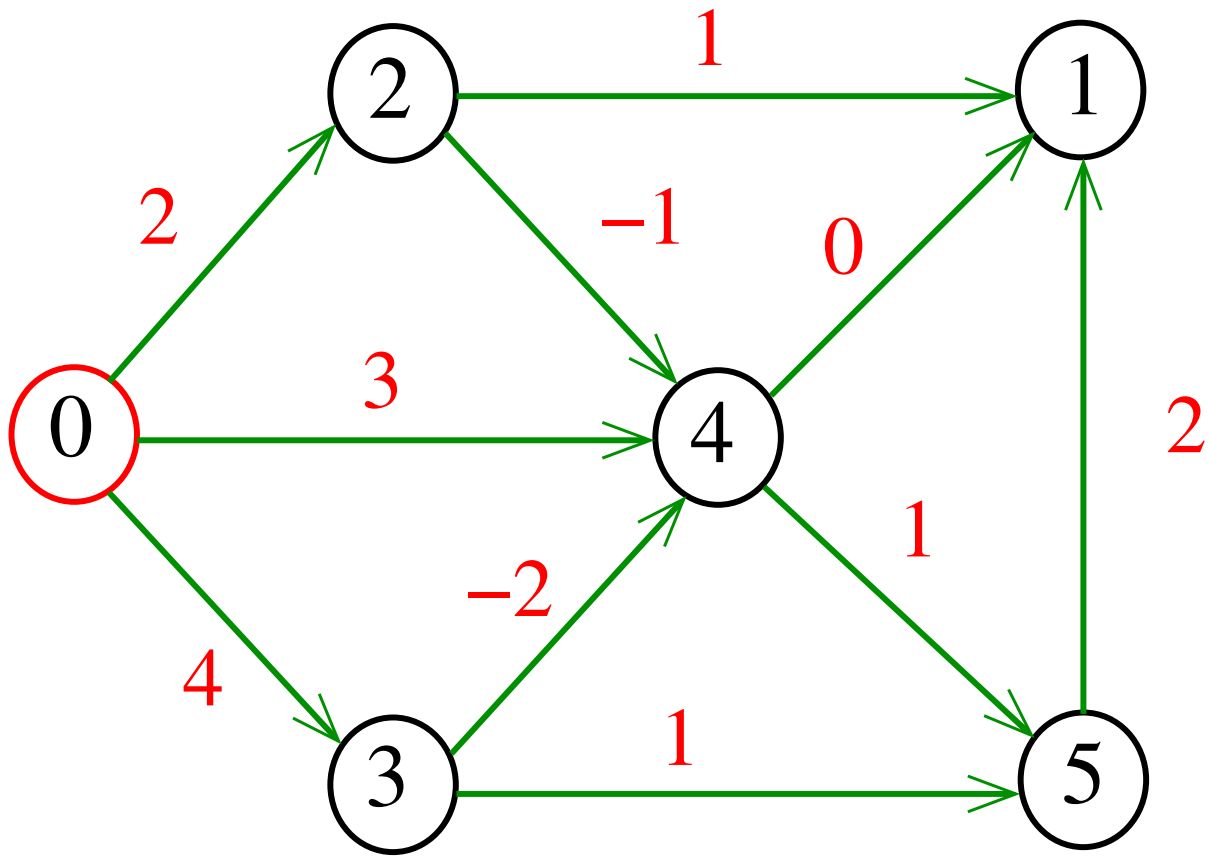
Dado um vértice s de um DAG com custos ***possivelmente negativos*** *nos arcos, encontrar, para cada vértice t que pode ser alcançado a partir de s, um caminho simples mínimo de s a t*

Problema:

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

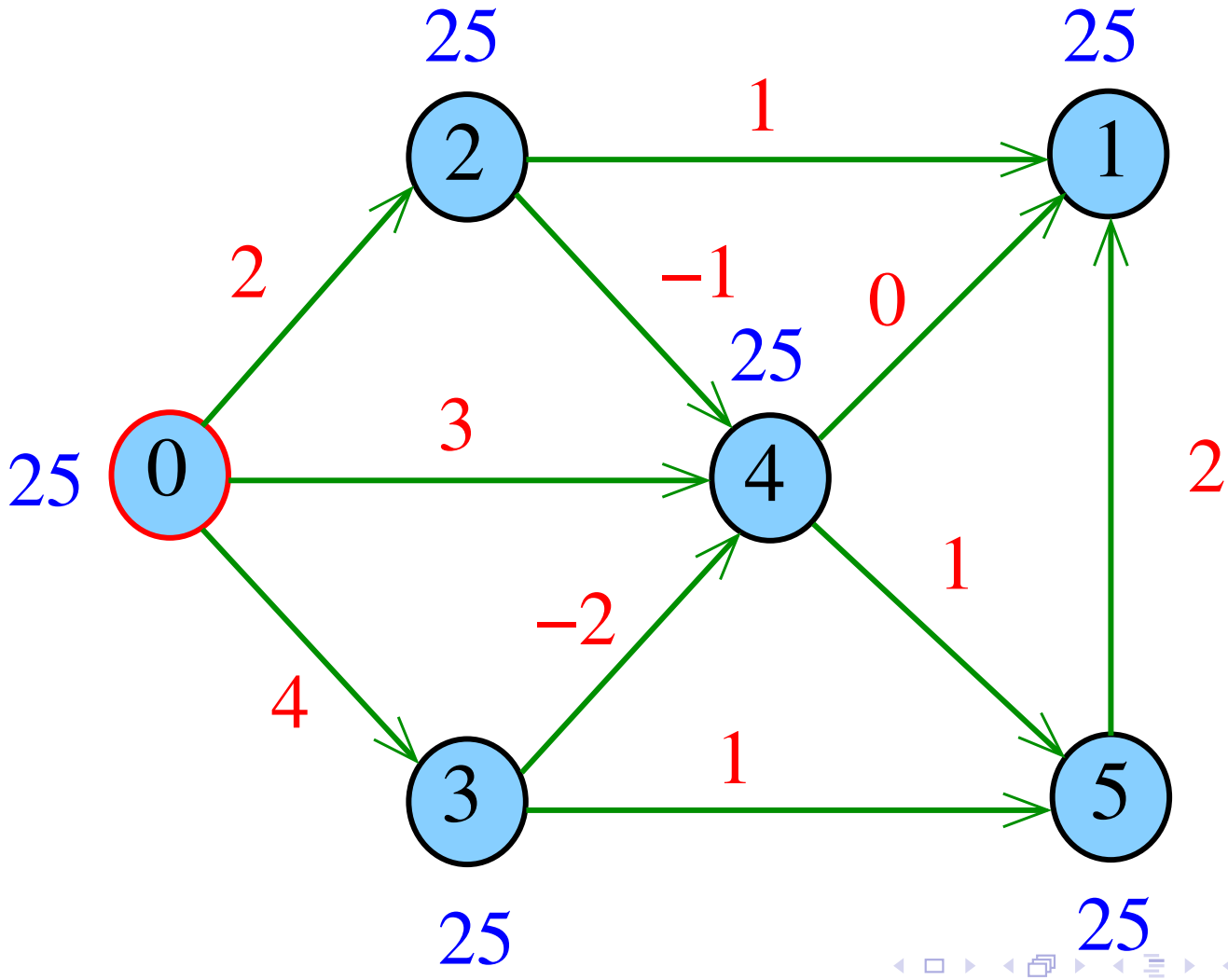
Simulação

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



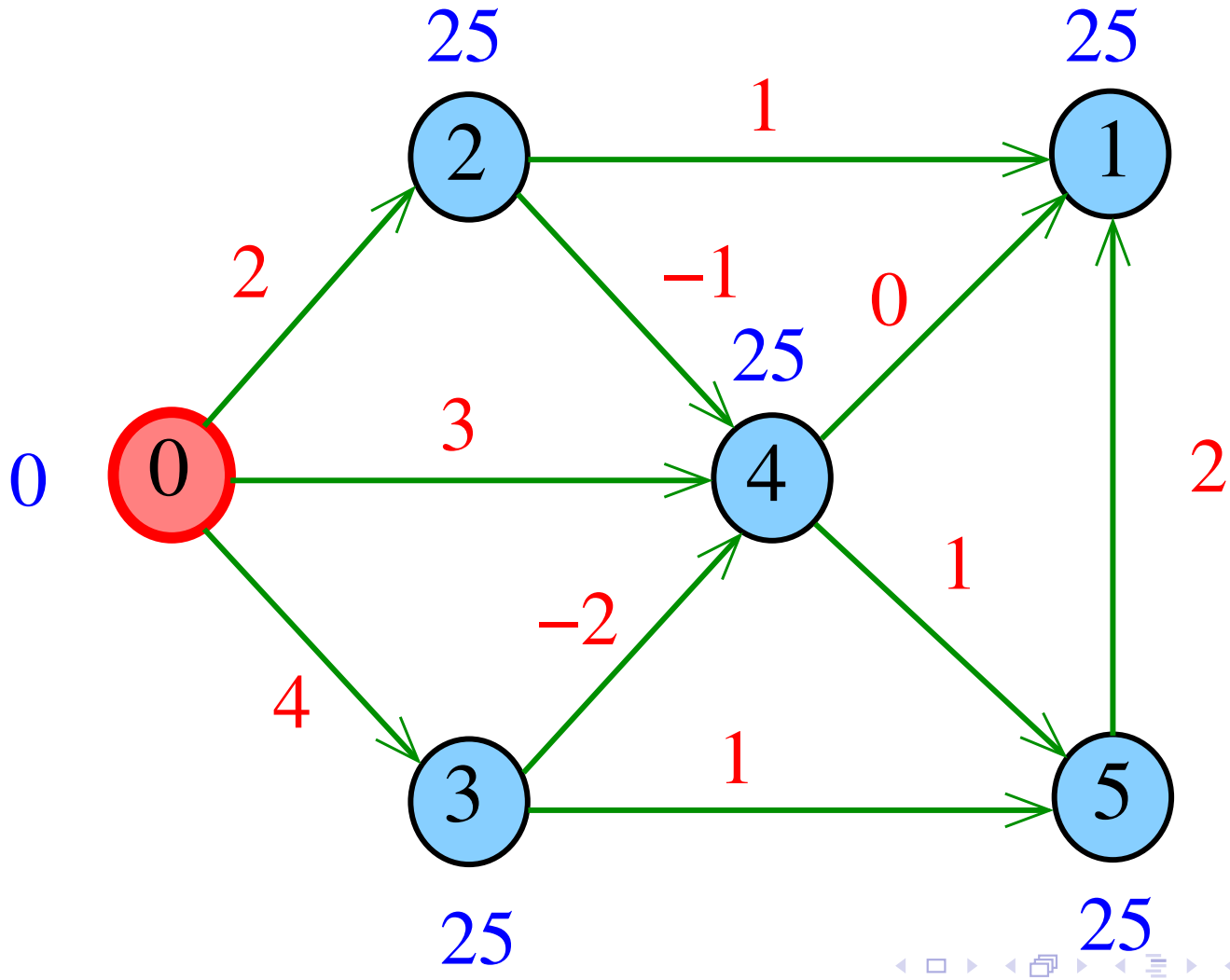
Simulação

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



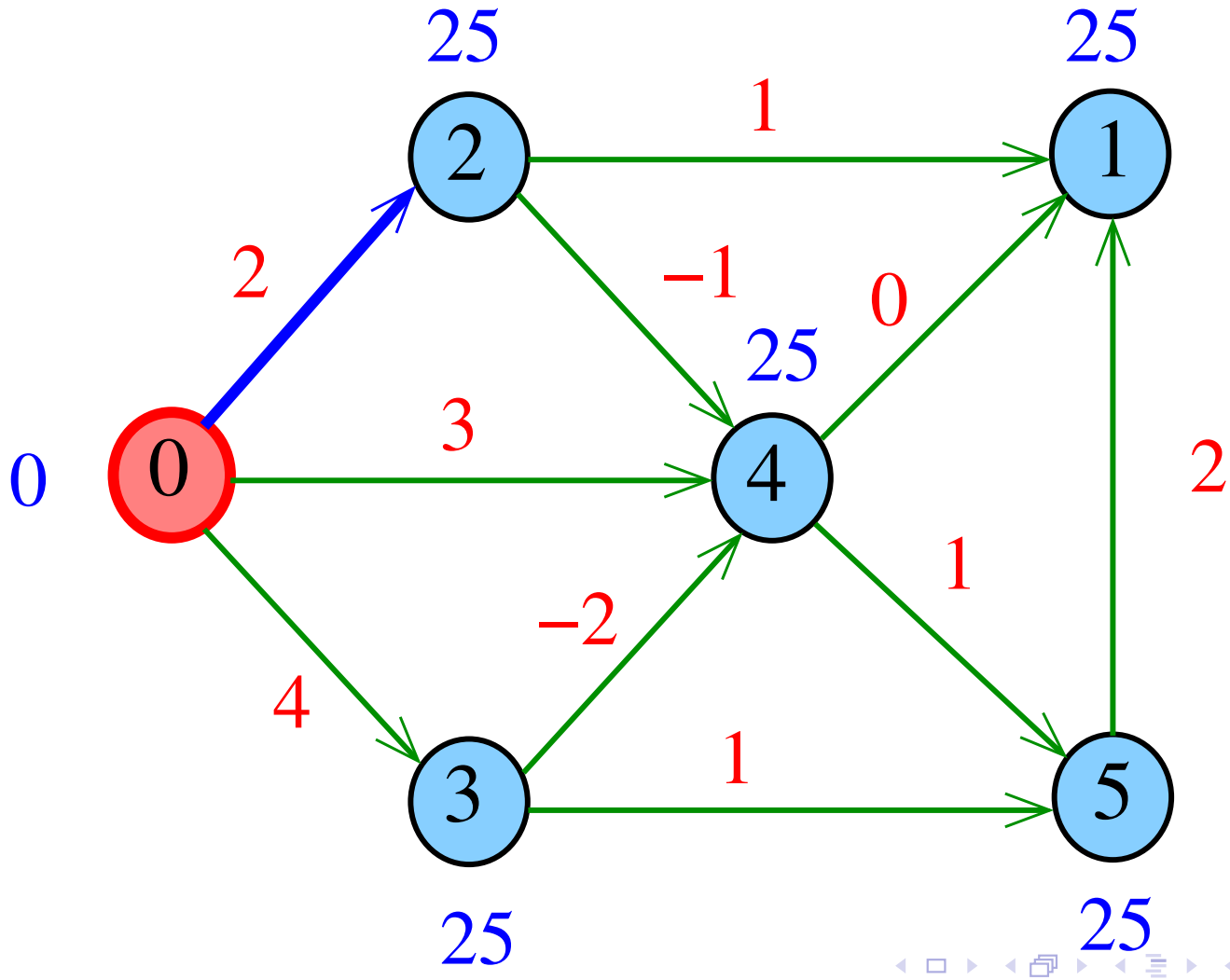
Simulação

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



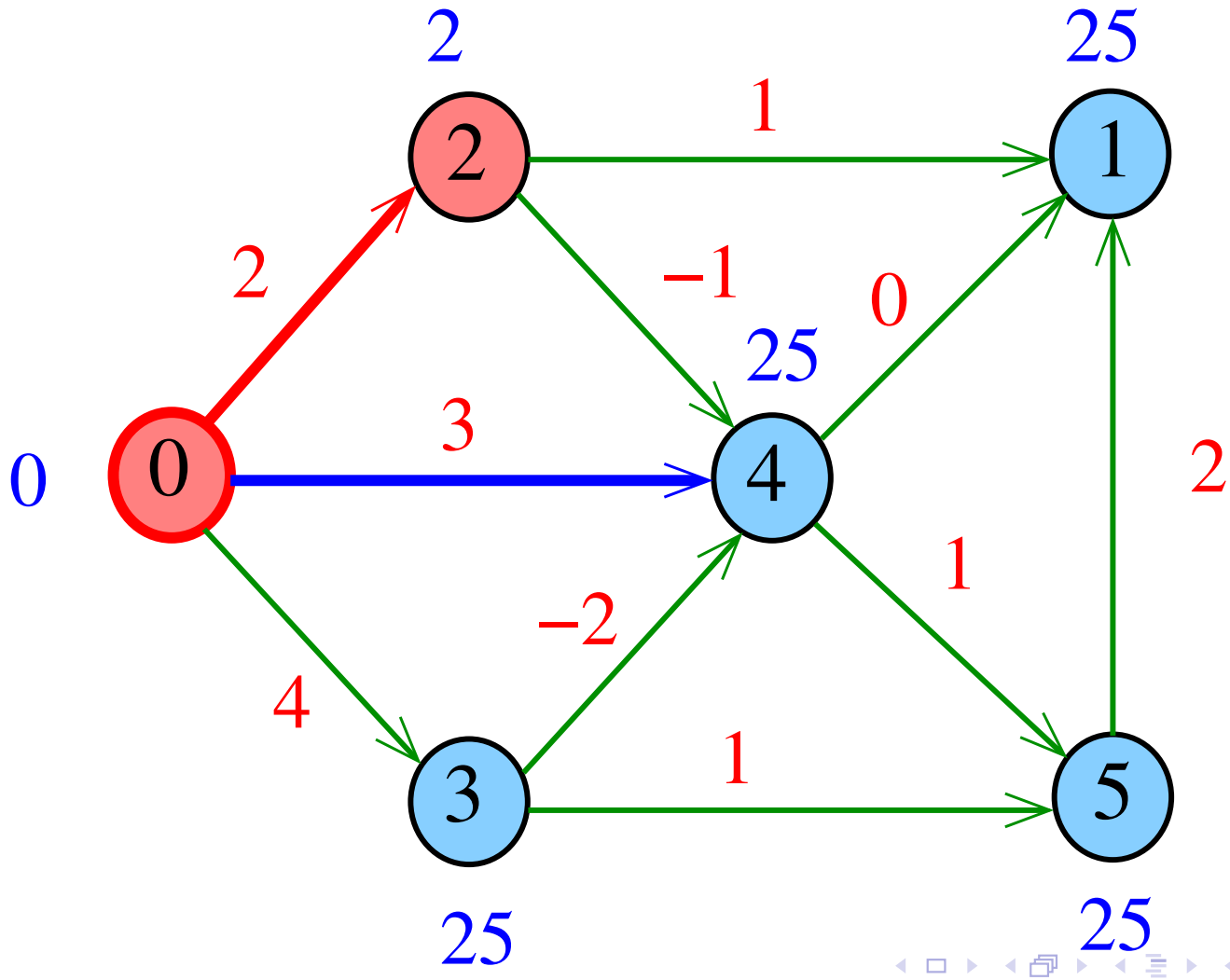
Simulação

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



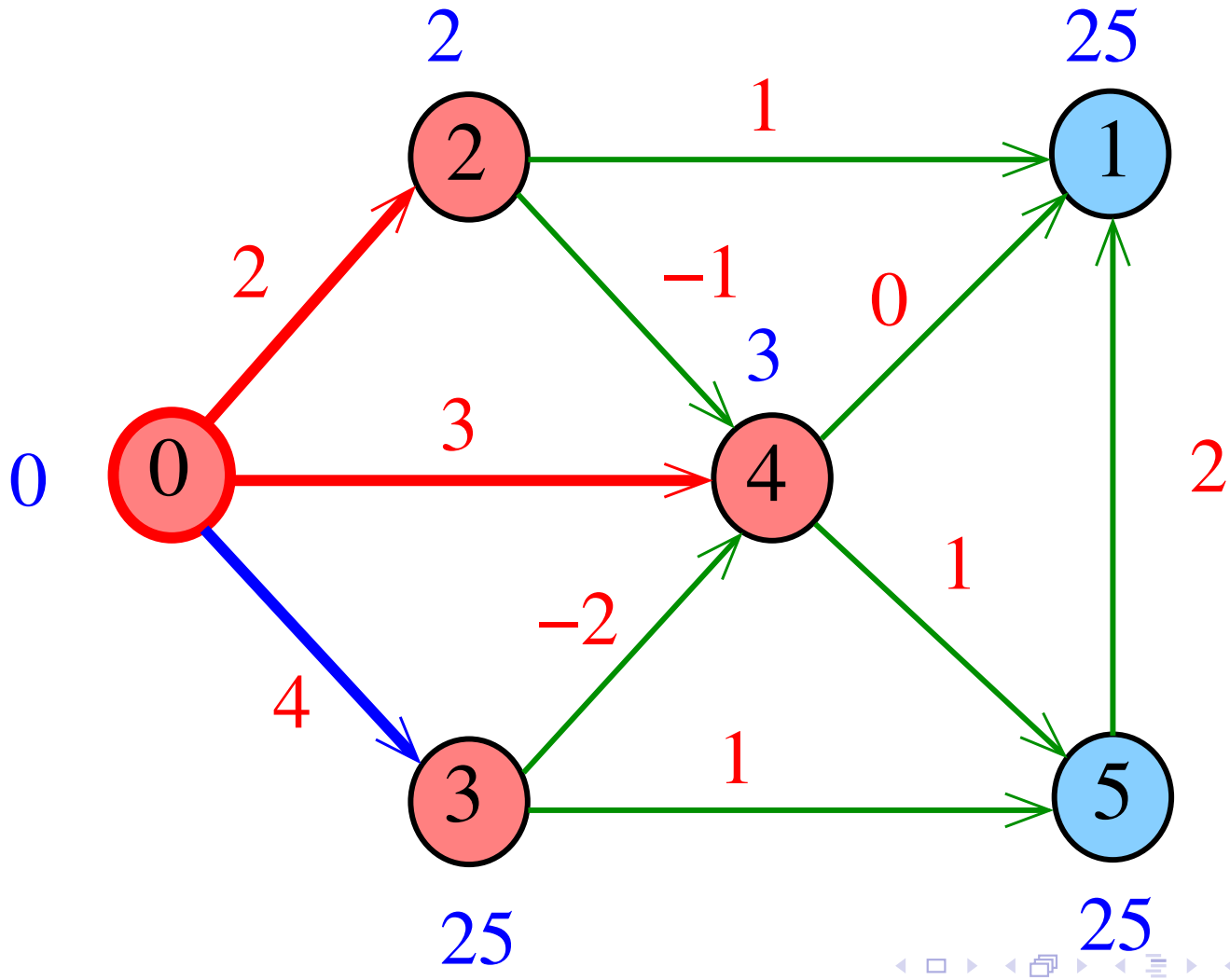
Simulação

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



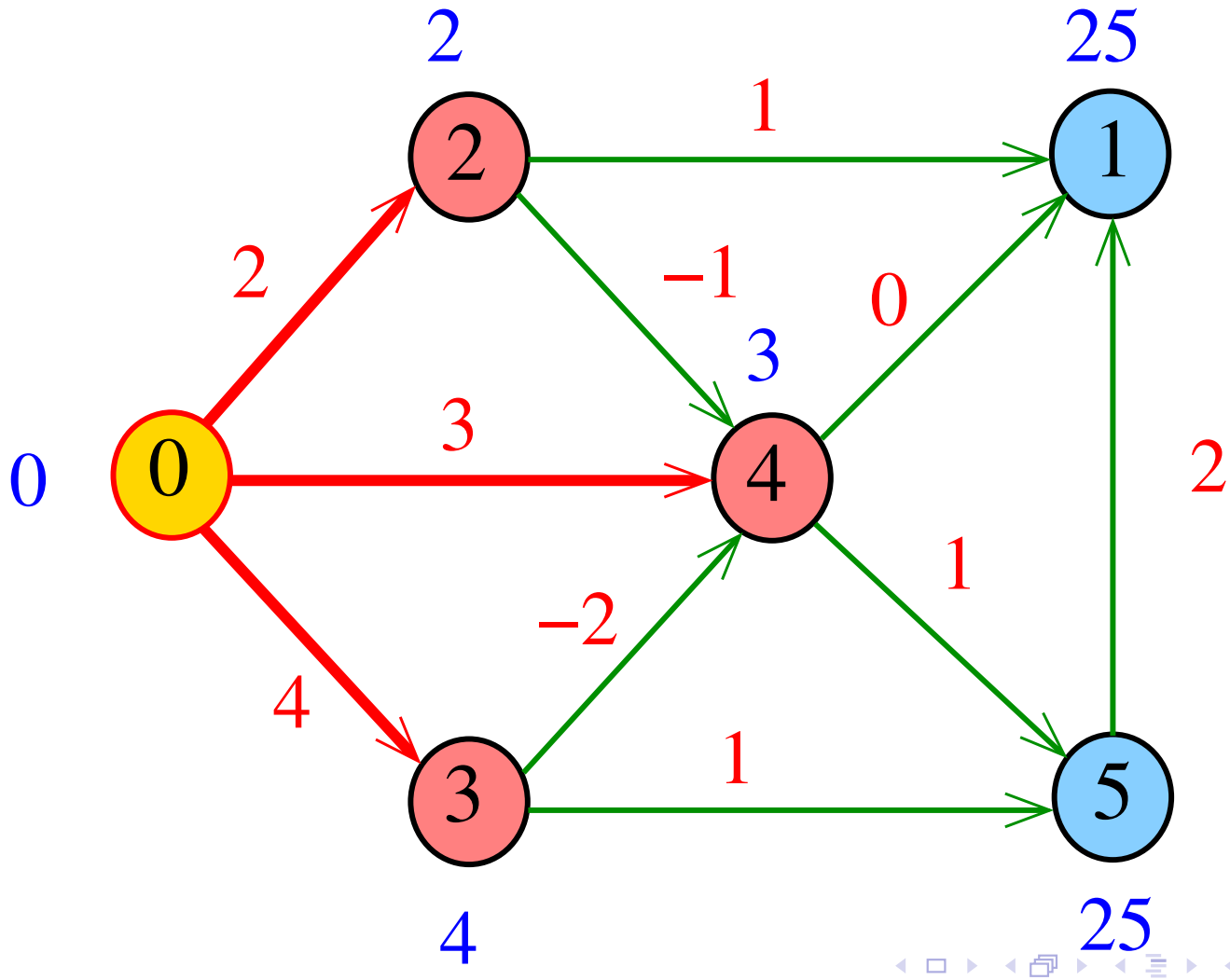
Simulação

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



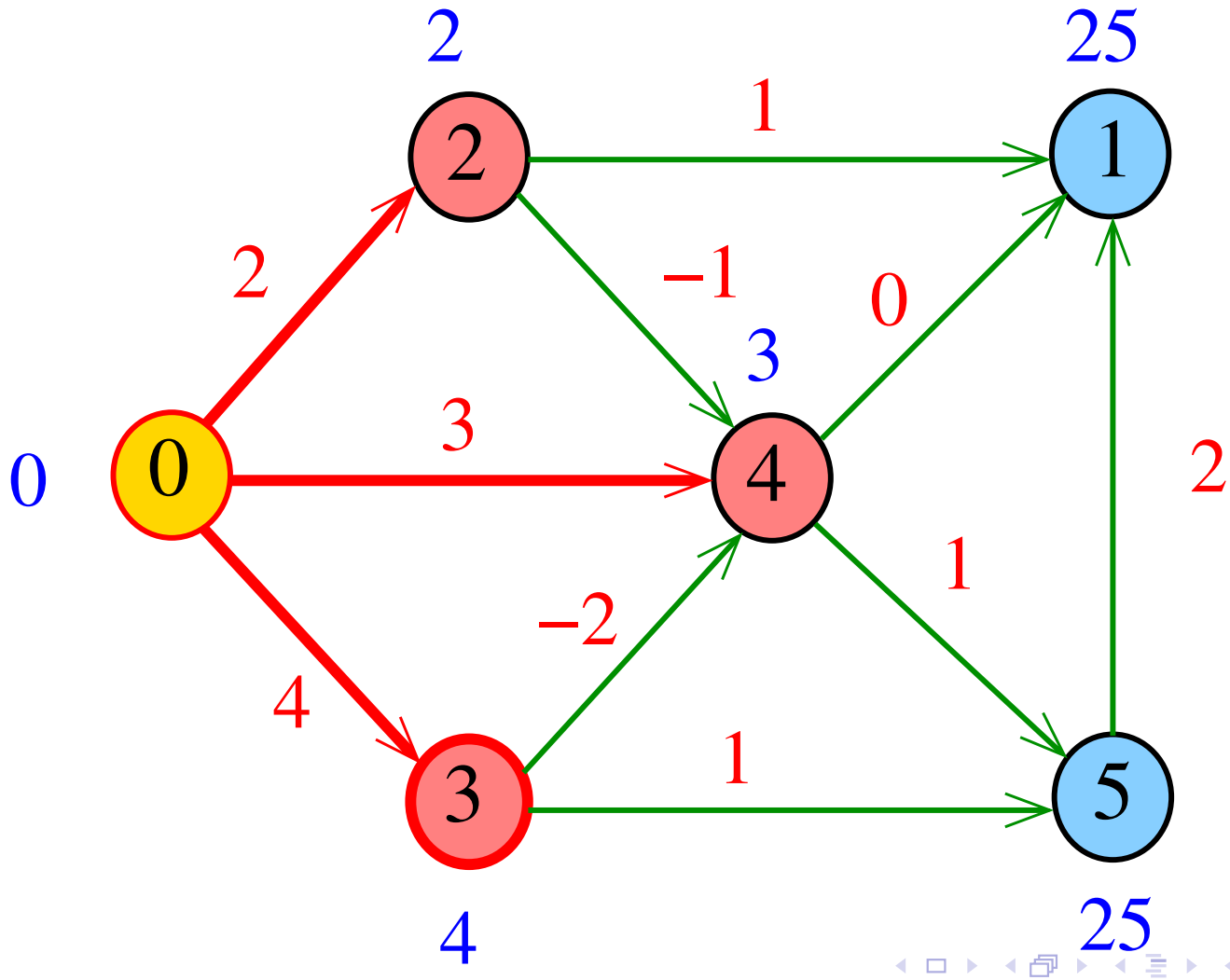
Simulação

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



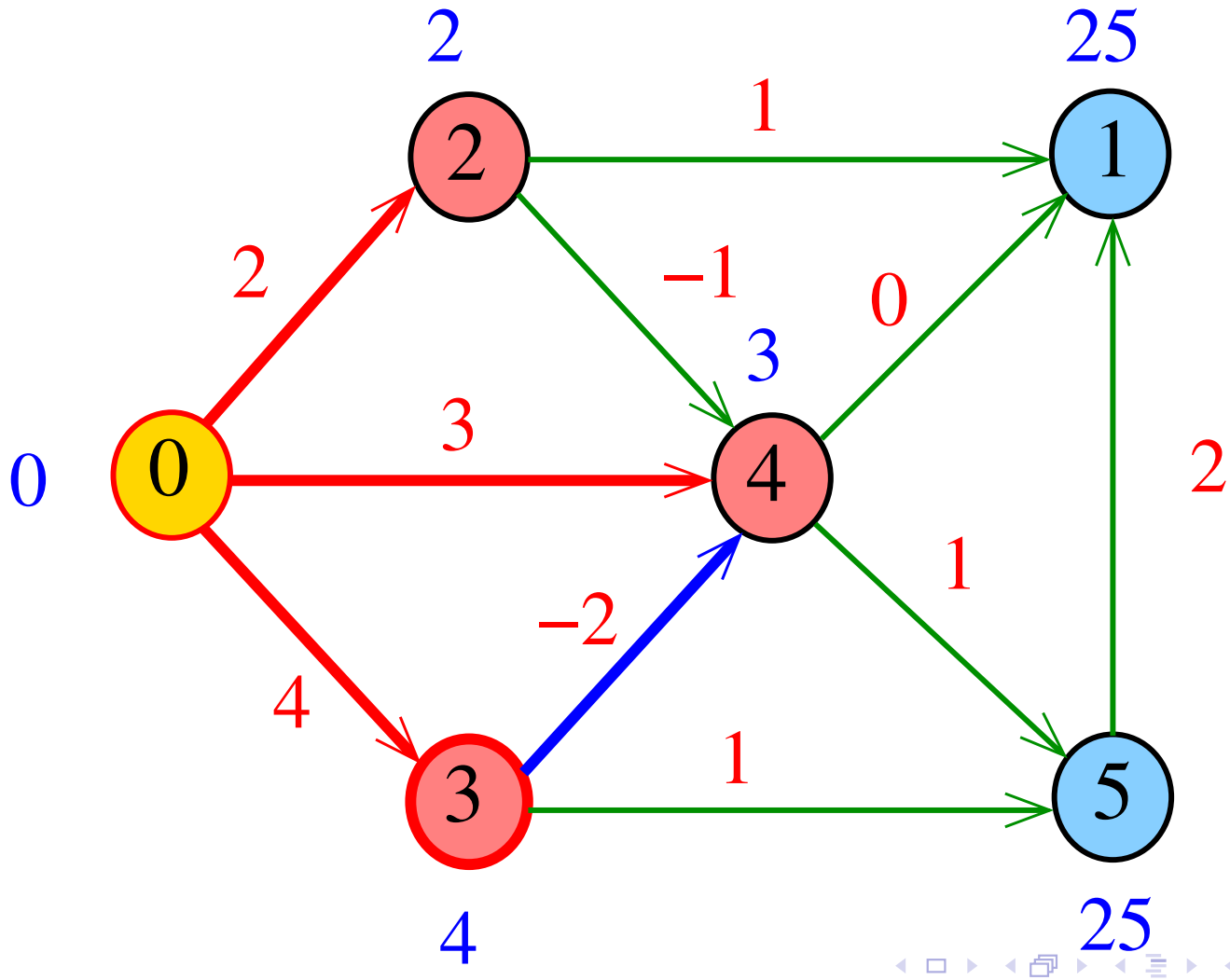
Simulação

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



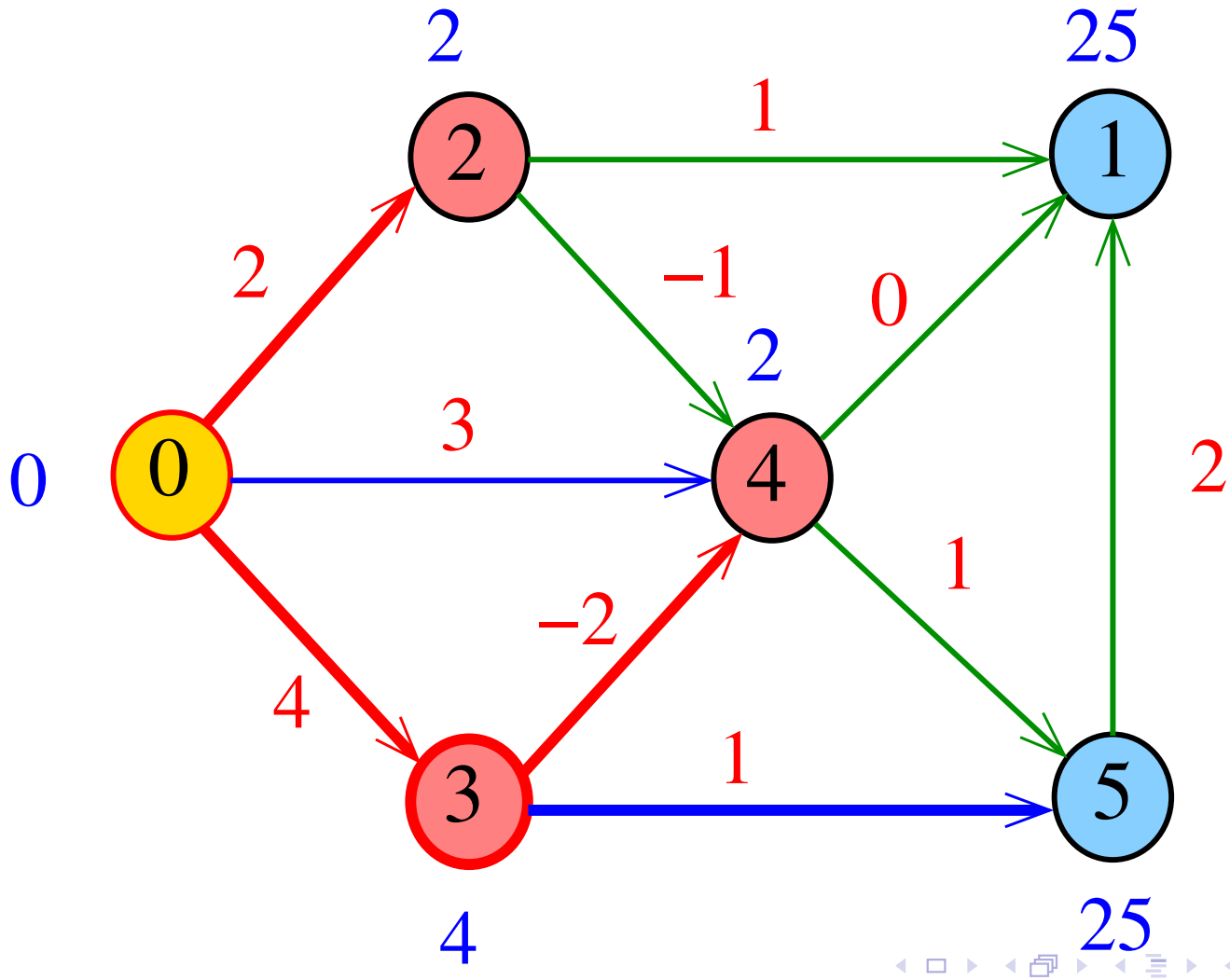
Simulação

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



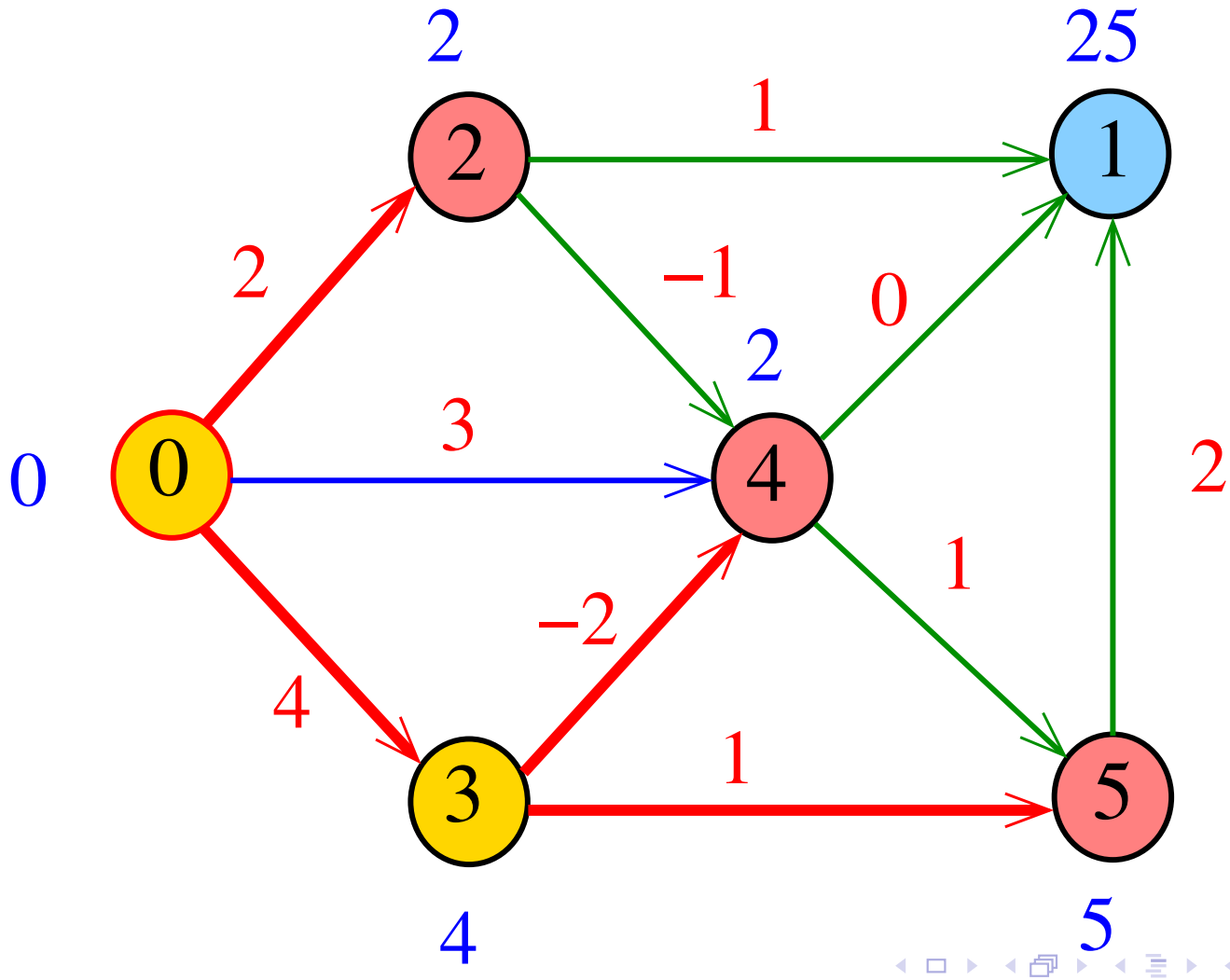
Simulação

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



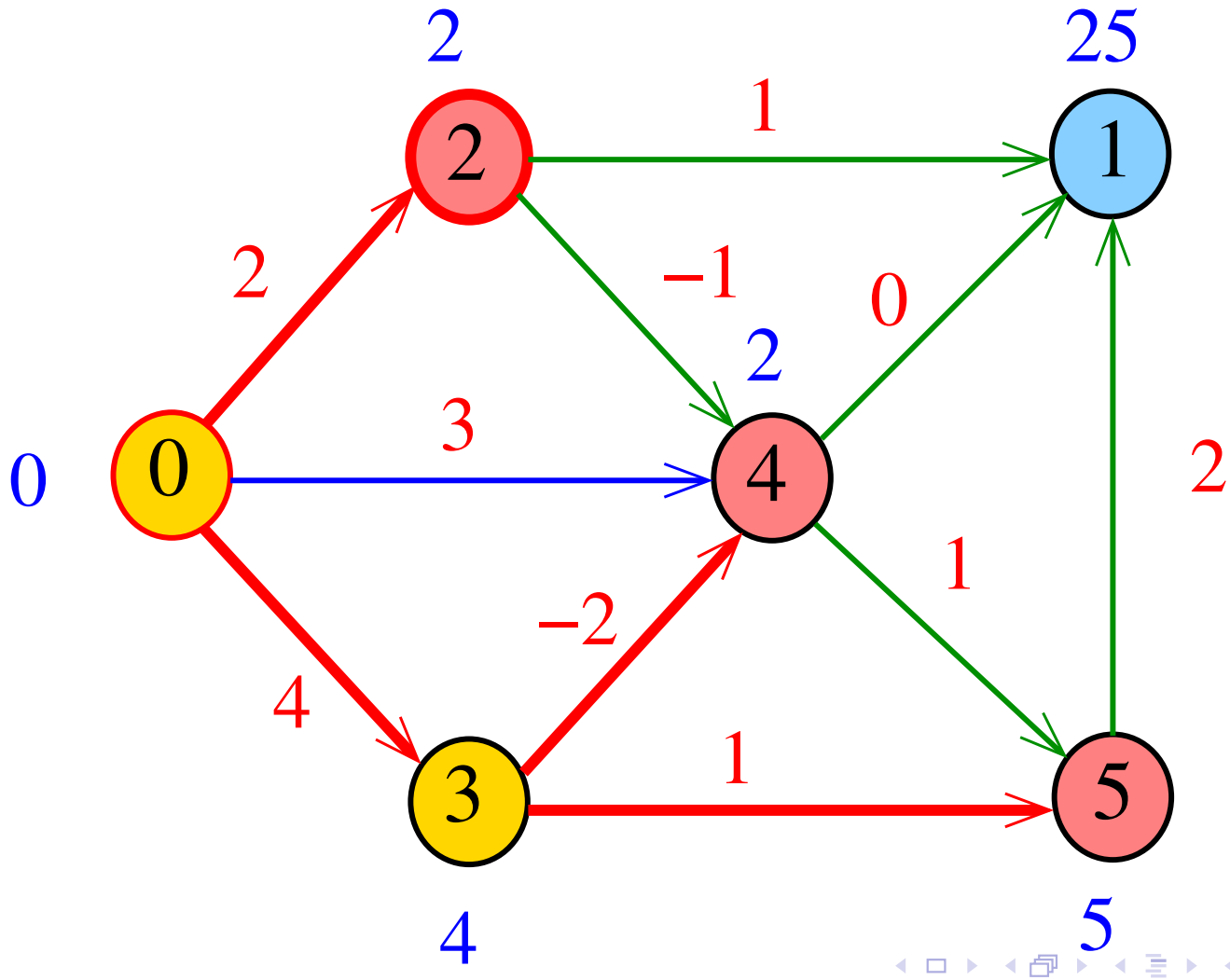
Simulação

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



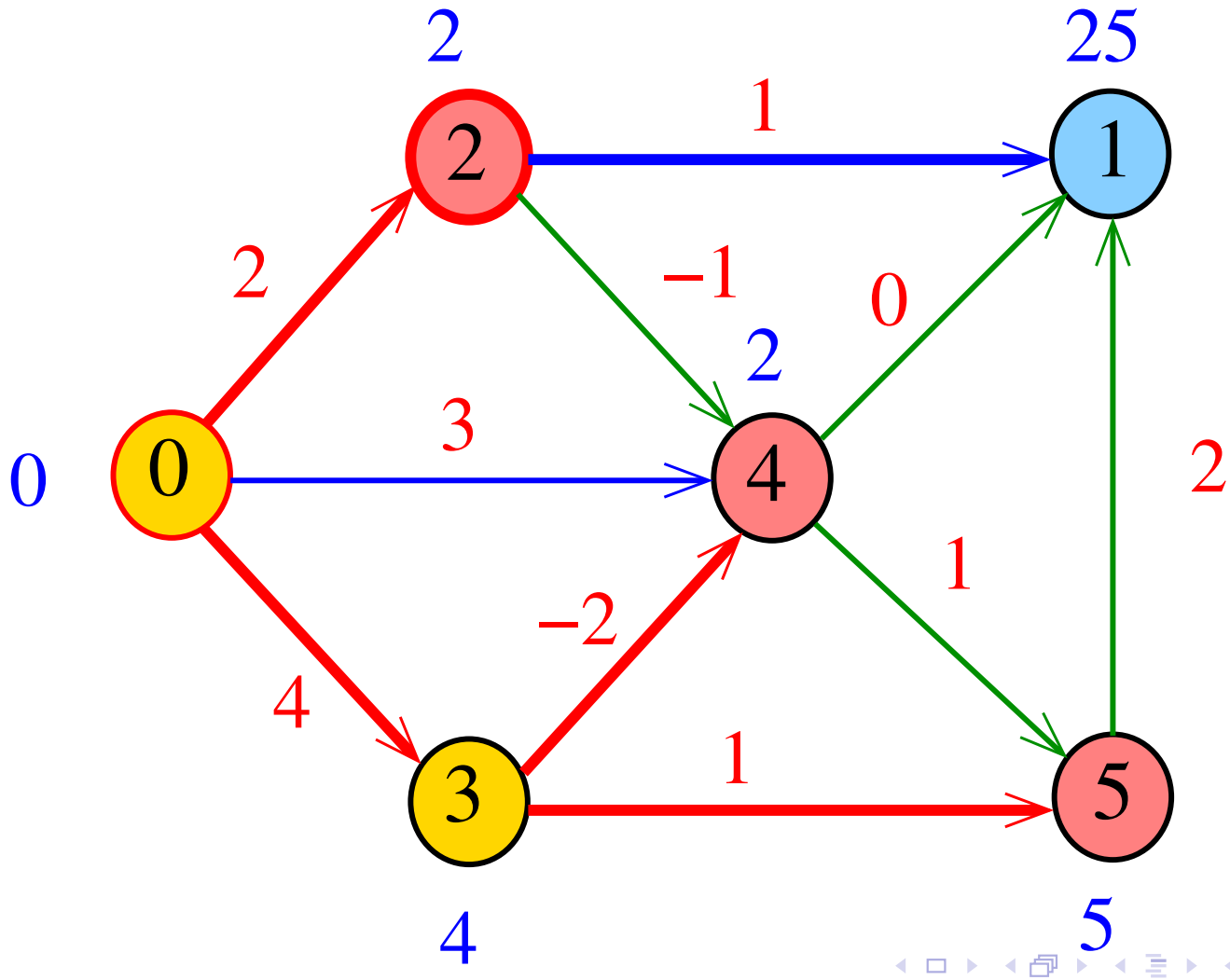
Simulação

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



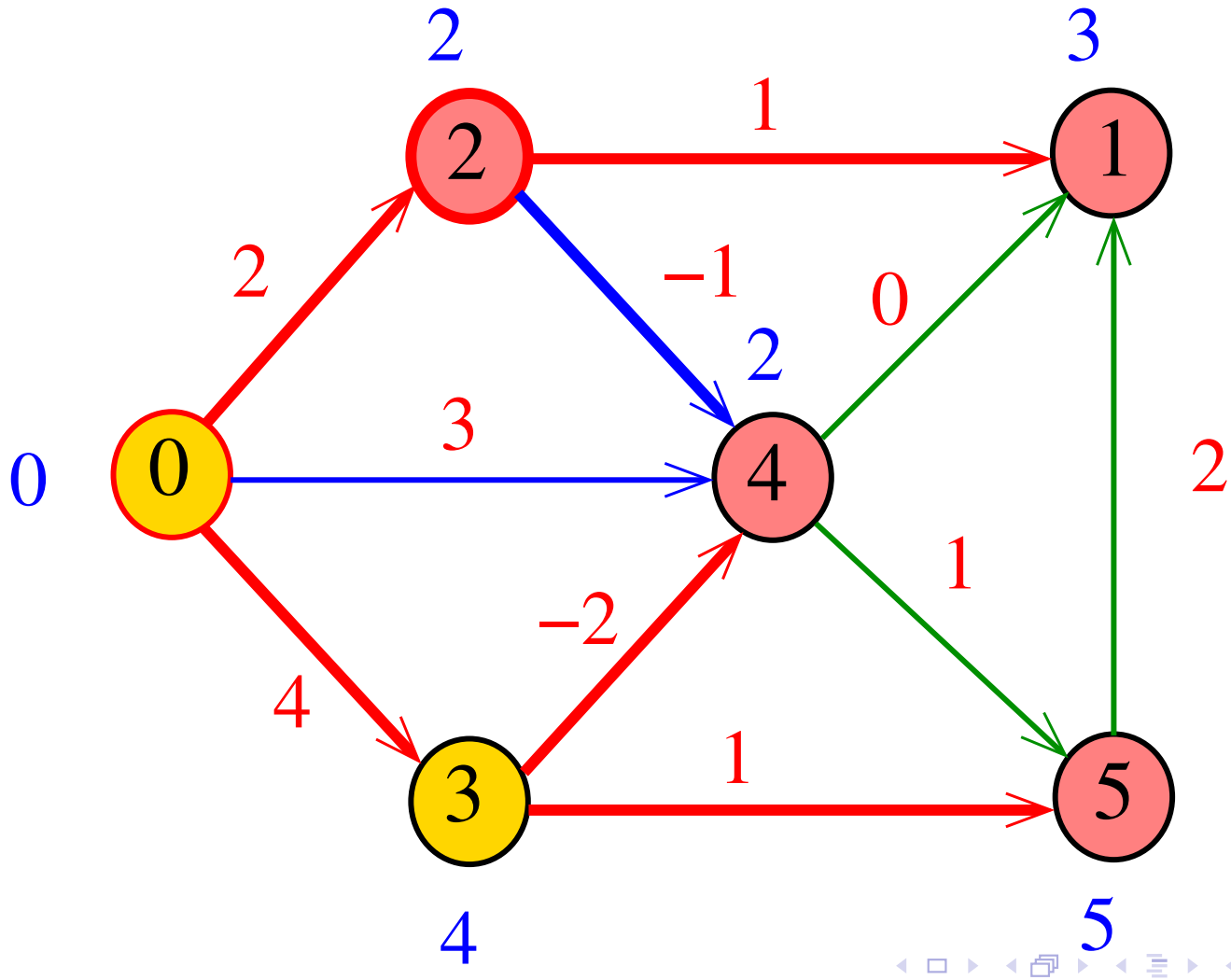
Simulação

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



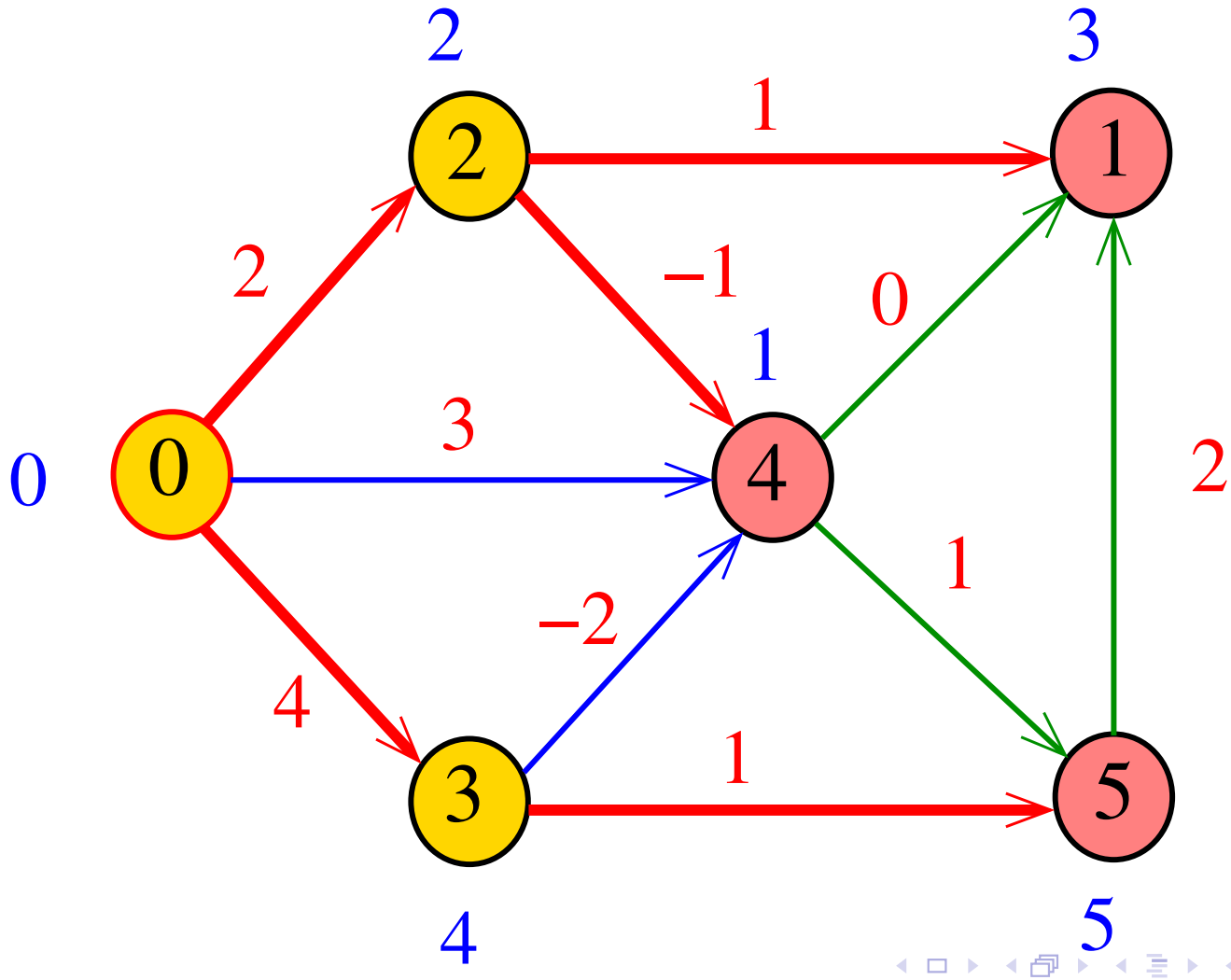
Simulação

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



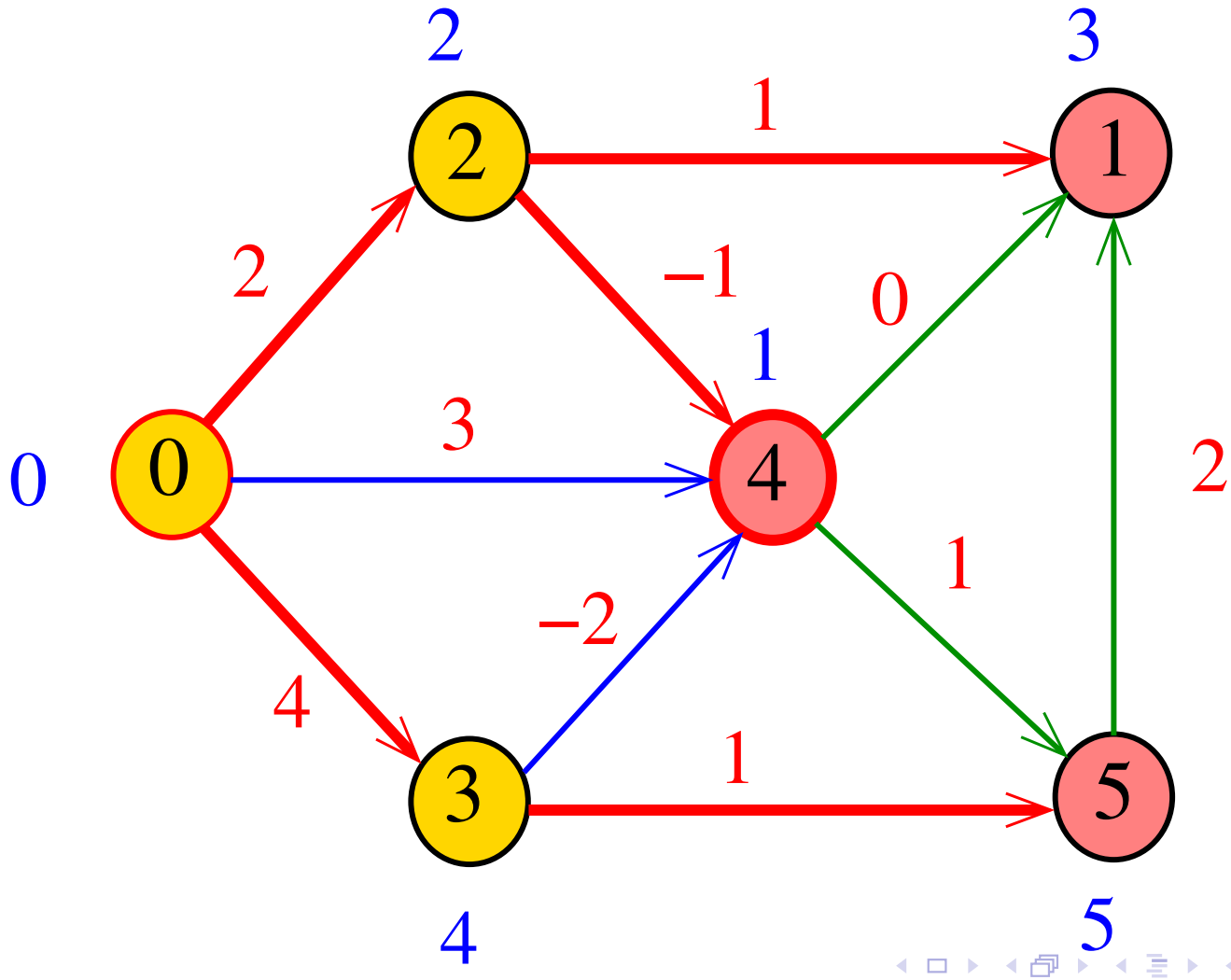
Simulação

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



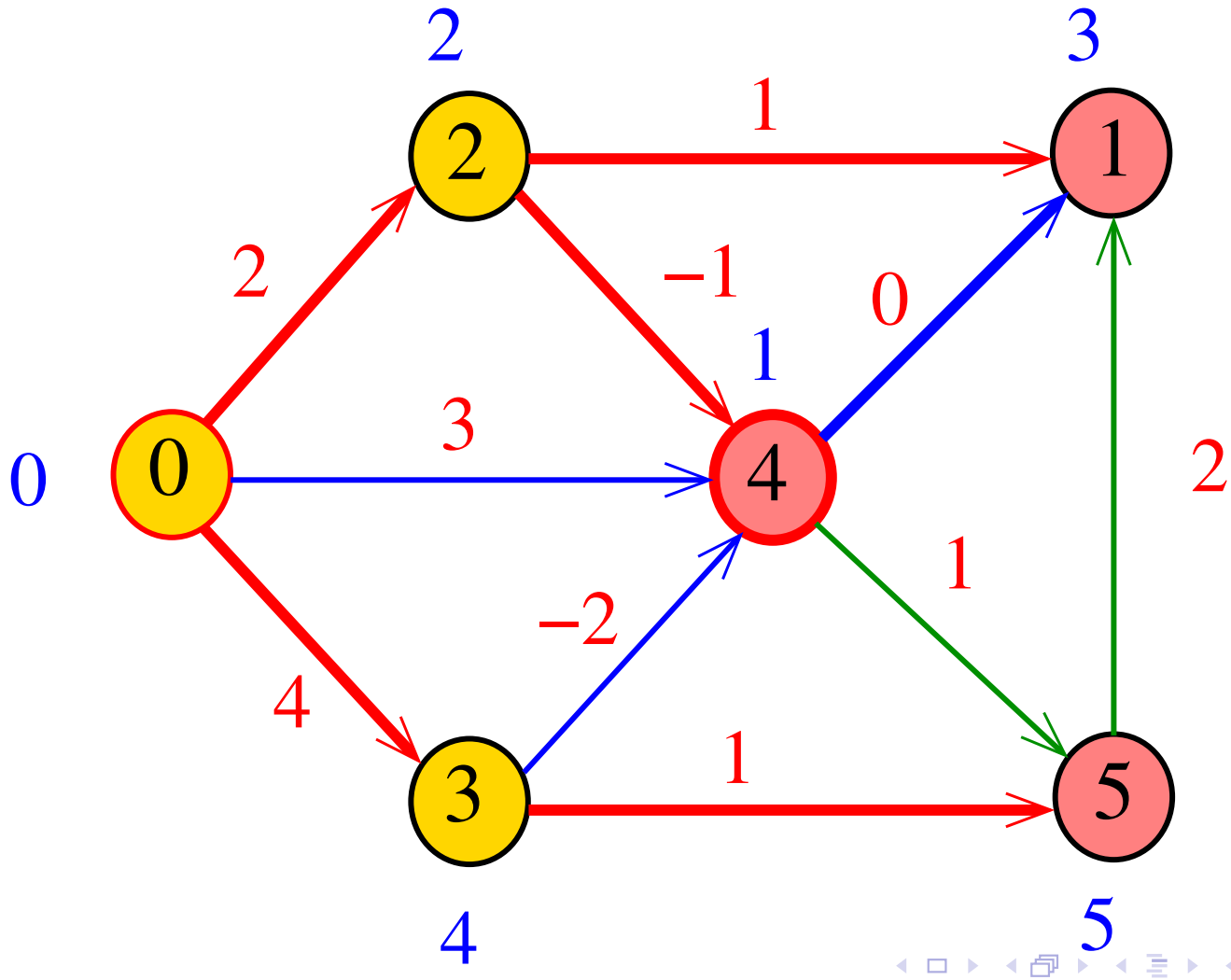
Simulação

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



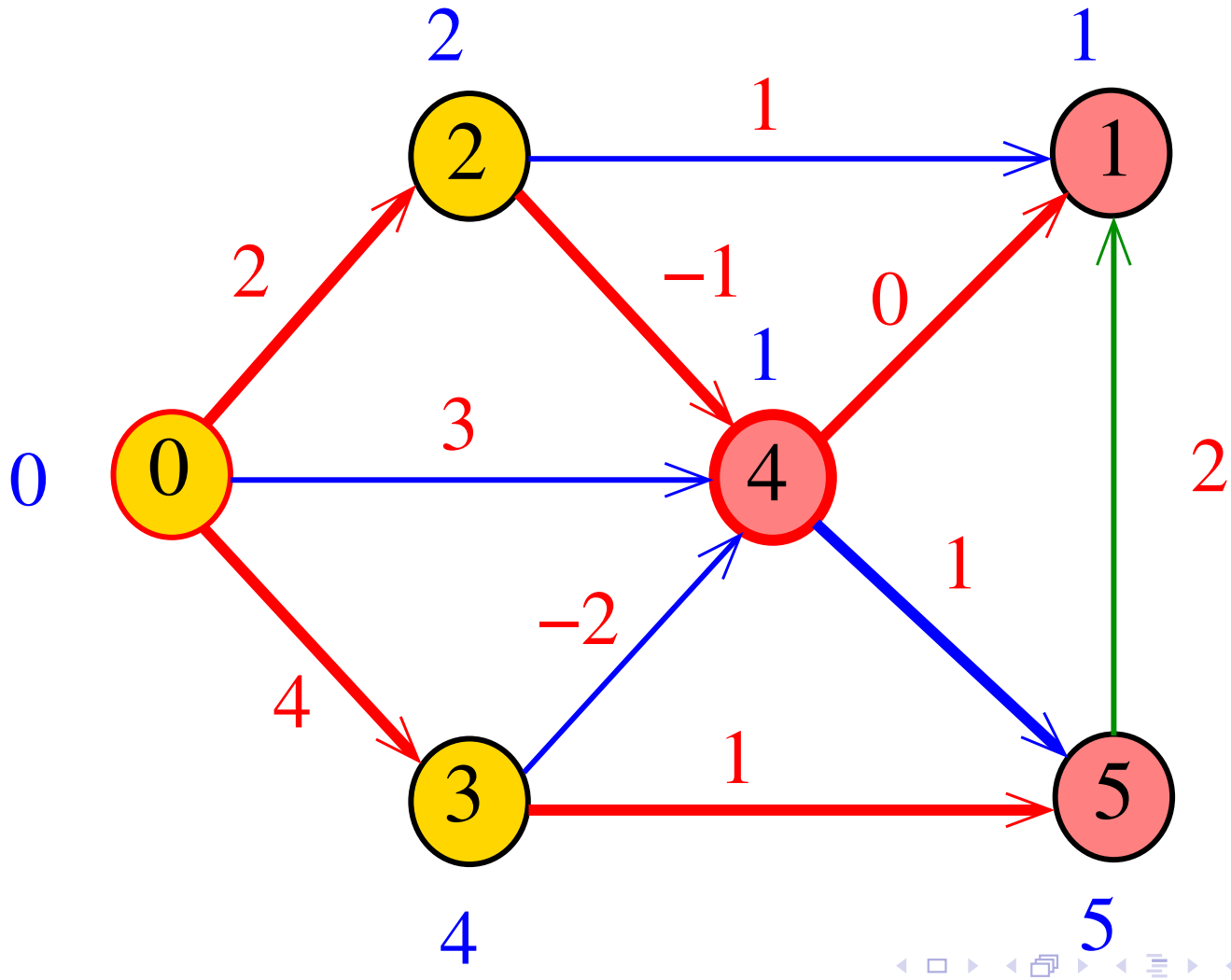
Simulação

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



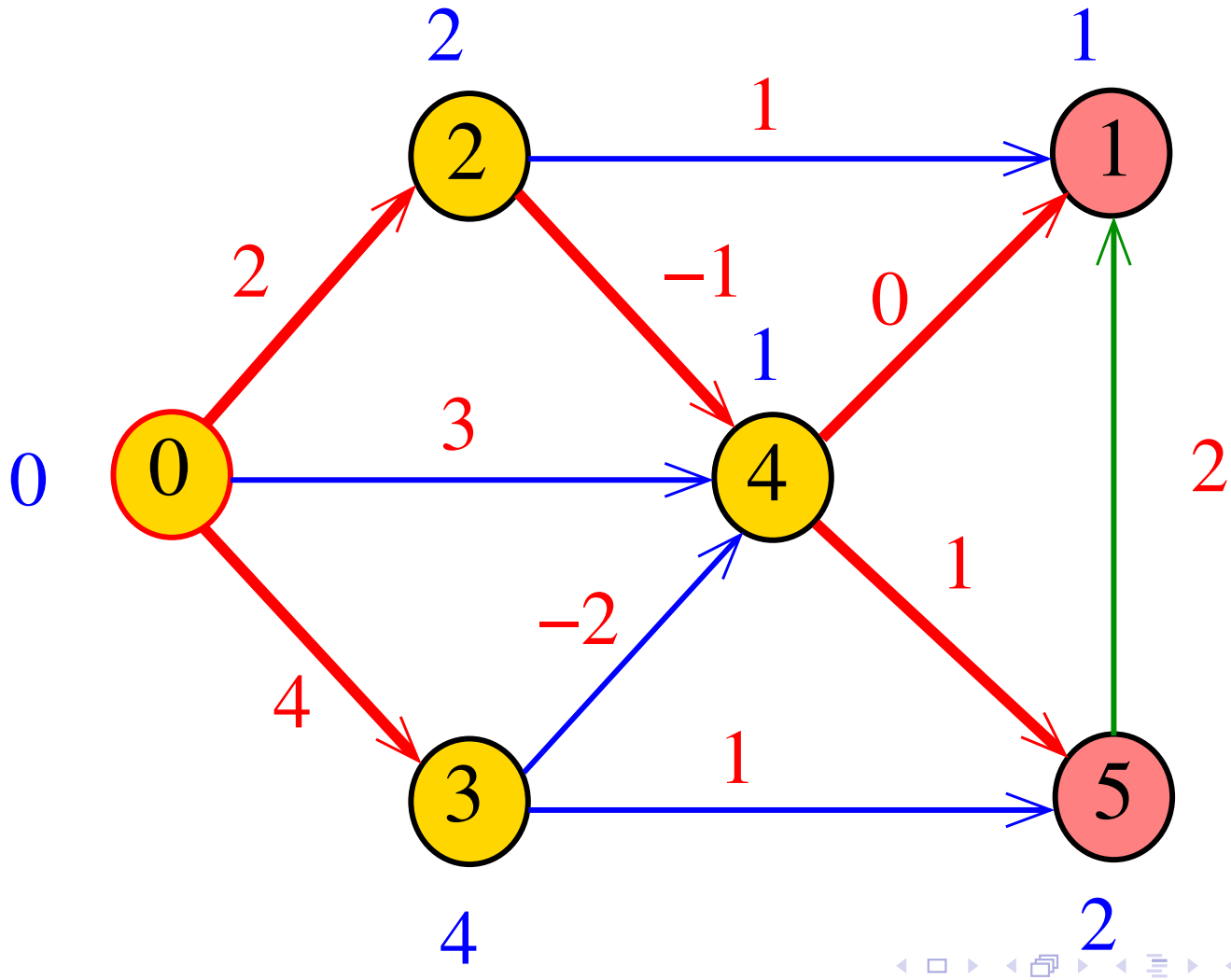
Simulação

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



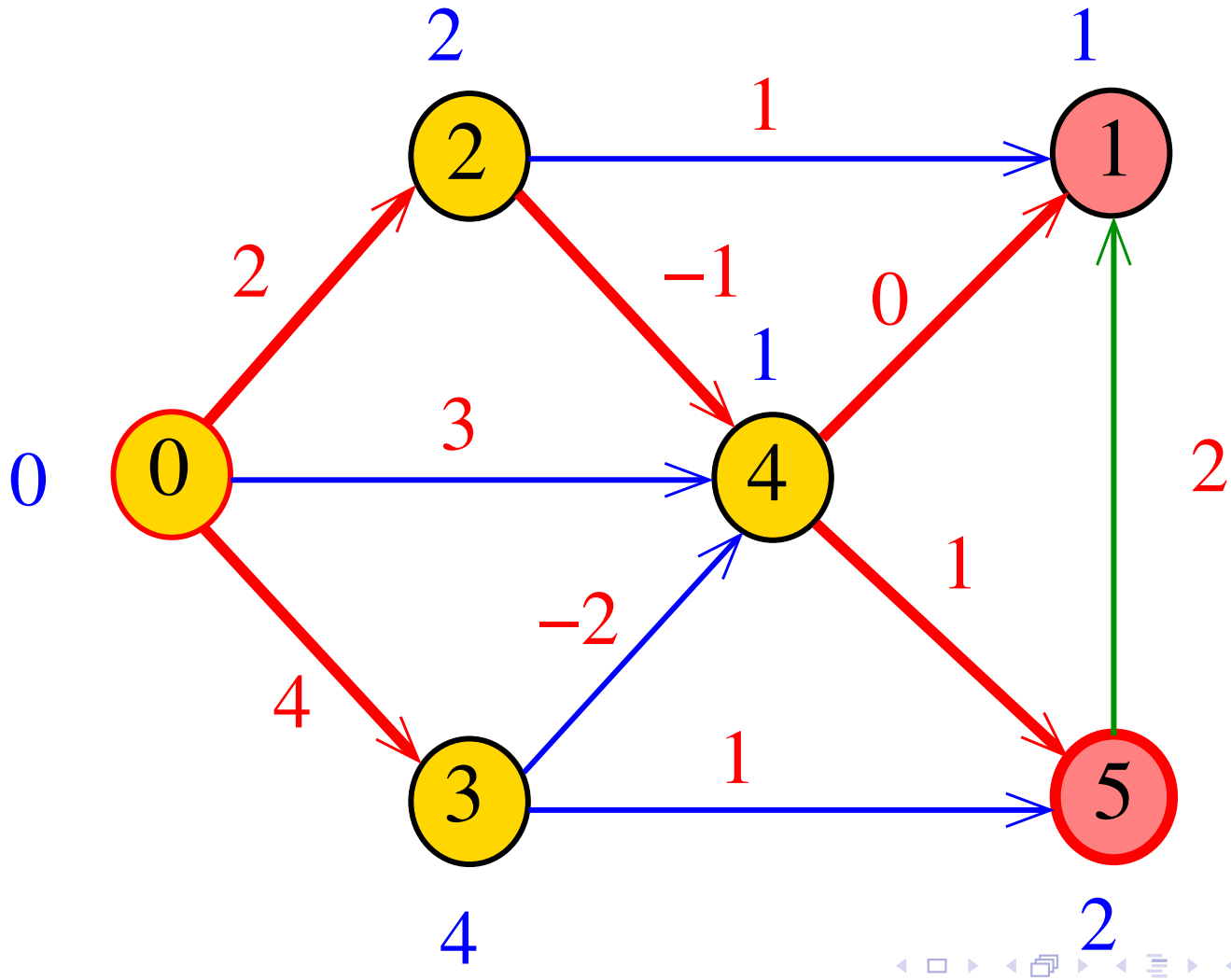
Simulação

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



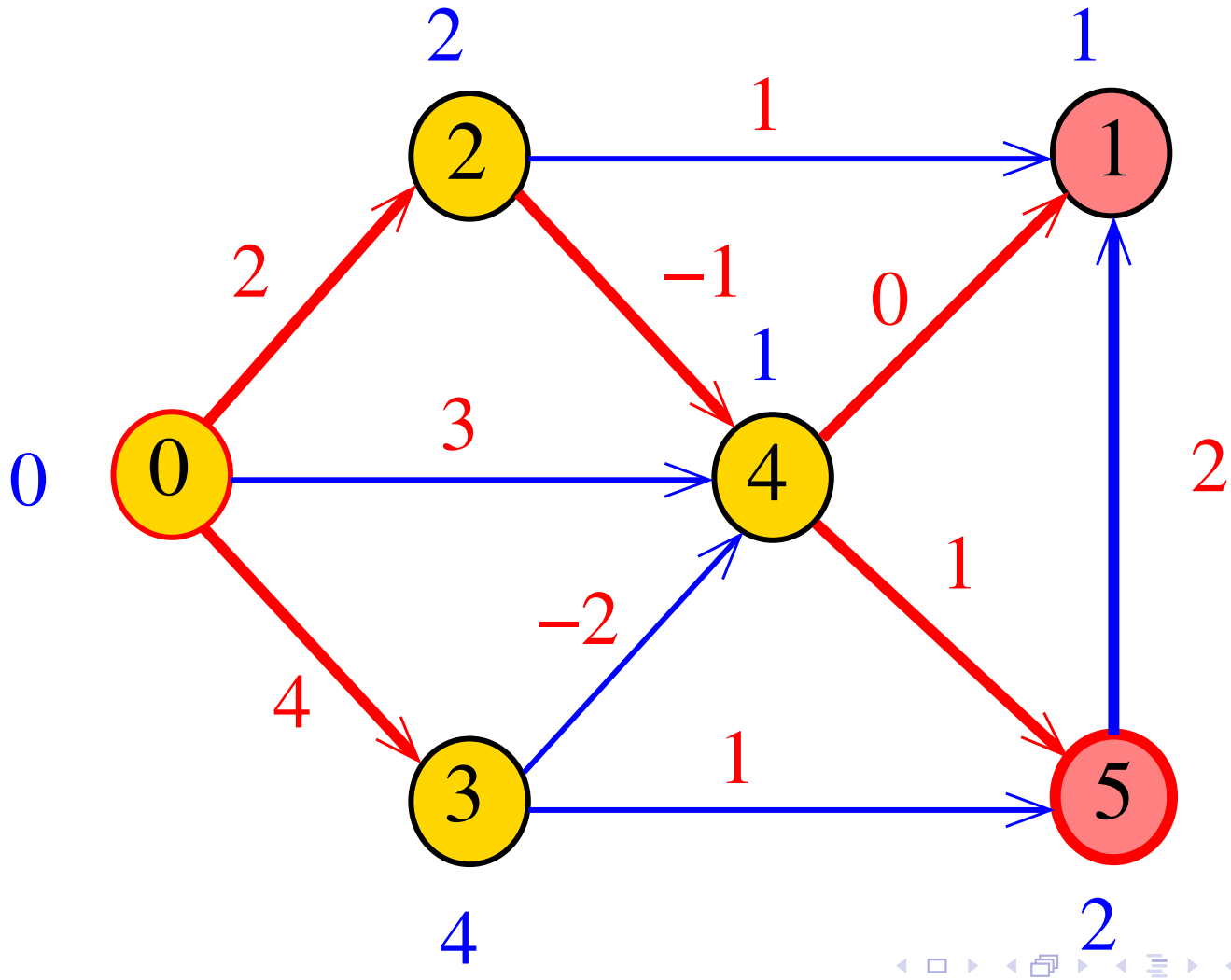
Simulação

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



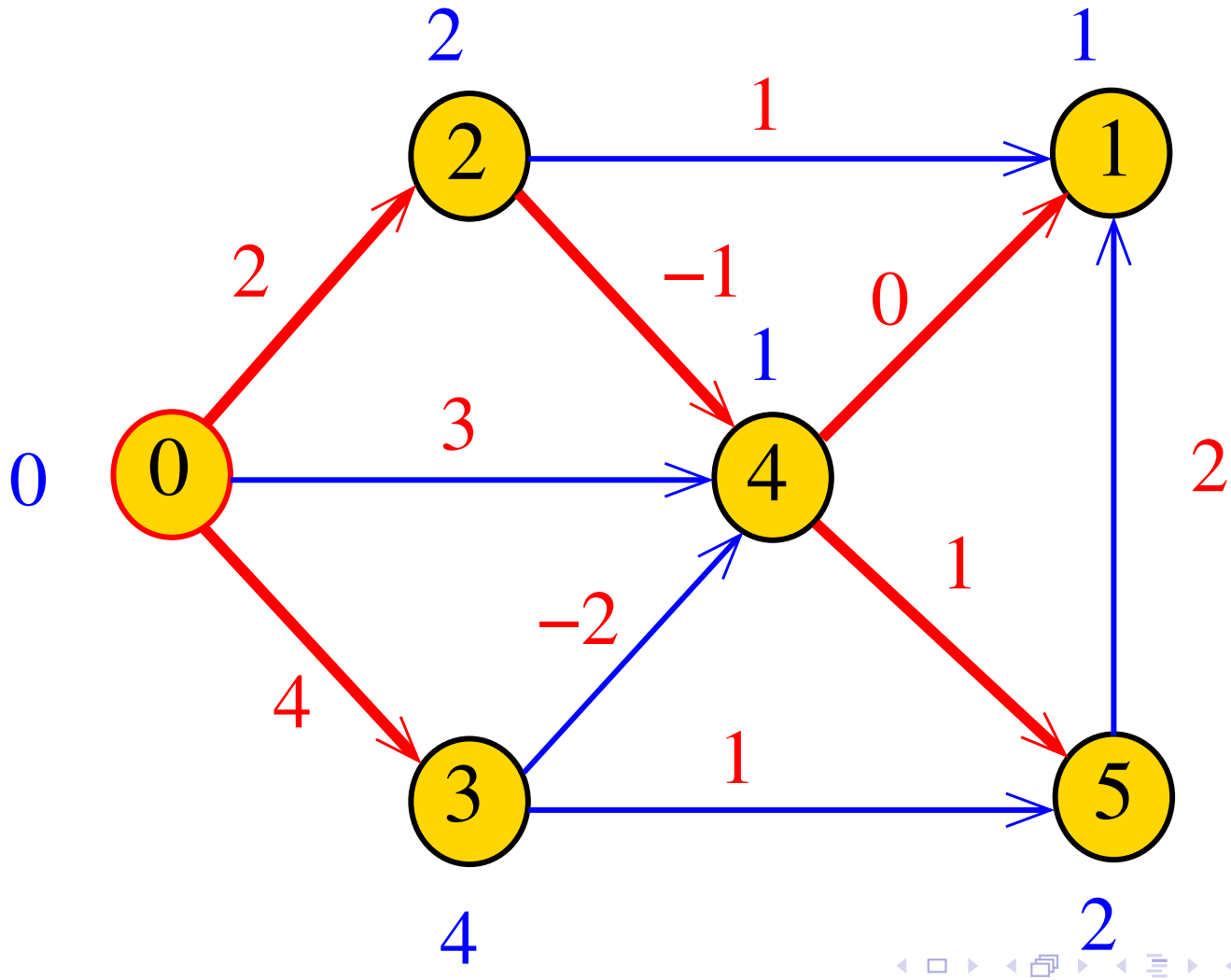
Simulação

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



AcyclicSP

A classe AcyclicSP recebe um DAG G com custos possivelmente negativos. Recebe também um vértice s.

A classe determina uma ordenação topológica dos vértices de Gatravés da classe DFStopological modificada para trabalhar com EWDigraphs.

Para cada vértice t, a função calcula o custo de um caminho de custo mínimo de sa t. Esse número é depositado em distTo[t].

```
public class AcyclicSP(EWDigraph G,
                       int s);
```

Classe AcyclicSP: esqueleto

```
public class AcyclicSP {
   private static final int INFINITY;
   private final int s;
   private Arc[] edgeTo;
   private double[] distTo;
   public AcyclicSP(EWDigraph G, int s) {}
   private void acyclic(EWDigraph G,
                        int s) {}
   public boolean hasPath(int v) {}
   public boolean distTo(int v) {}
   public Iterable<Arc> pathTo(int v)
}
```

AcyclicSP

Encontra um caminho de s a todo vértice alcançável a partir de s.

```
public AcyclicSP(EWDigraph G, int s) {
   INFINITY = Double.POSITIVE_INFINITY;
   edgeTo = new Arc[G.V()];
   distTo = new double[G.V()];
   this.s = s;
   for (int v = 0; v < G.V(); v++)
       distTo[v] = INFINITY;
   acyclic(G, s);
}
```

acyclic()

```
private void acyclic(EWDigraph G, int s){
   DFStopological ts= new DFStopological(G);
   distTo[s] = 0;
   for(int v:  ts.order()) {
       for (Arc e :  G.adj(v)) {
           int v = e.from(), w = e.to();
           Double d = distTo[v]+e.weight();
           if (distTo[w] > d) {
              edgeTo[w] = e;
              distTo[w] = d;
           }
       }
   }
```

AcyclicSP

Há um caminho de s a v?

```
// Método copiado de BFSpaths.
public boolean hasPath(int v) {
   return distTo[v] < INFINITY;
}

// retorna o comprimento de um
// caminho mínimo de s a t
public int distTo(int v) {
   return distTo[v];
}
```

AcyclicSP

Retorna um caminho de s a v ou null se um tal caminho não existe.

```
// Método adaptado de DFSpaths.
public Iterable<Arc> pathTo(int v) {
   if (!hasPath(v)) return null;
   Stack<Arc> path = new Stack<Arc>();
   for (Arc e = edgeTo[v]; e != null;
                  e = edgeTo[e.from()])
      path.push(e);
   return path;
}
```

Consumo de tempo

O consumo de tempo de `AcyclicSP` para vetor de listas de adjacência é $O(V + E)$.

O consumo de tempo de `AcyclicSP` para matriz de adjacências é $O(V^2)$.

Busca de Substrings

Substring search

Referências: Busca de substring (PF), Substring Searching (SW), slides (SW), vídeo (SW).

Introdução

Problema: Dada uma string `pat` e uma string `txt`, encontrar uma ocorrência de `pat` em `txt`.

Exemplo: encontre `ATTGG` em:

```
TGGTAAGCGGTTCCTGCCCCGGCTCAGGGCCAAGAACAGATGAGACAGCTGAGTGATGGGCCAAACAGGATATCTGTGG
TAAGCAGTTCCTGCCCCGGCTCGGGGCCAAGAACAGATGGTCCCCAGATGCGGTCCAGCCCTCAGCAGTTTCTAGTGAA
TCATCAGATGTTTCCAGGGTGCCCCAAGGACCTGAAAATGACCCTGTACCTTATTTGAACTAACCAATCAGTTCGCTTC
TCGCTTCTGTTCGCGCGCTTCCGCTCTCCGAGCTCAATAAAAGAGCCCACAACCCCTCACTCGGCGCGCCAGTCTTCCG
ATAGACTGCGTCGCCCGGGTACCCGTATTCCCAATAAAGCCTCTTGCTGTTTGCATCCGAATCGTGGTCTCGCTGTTCC
TTGGGAGGGTCTCCTCTGAGTGATTGACTACCCACGACGGGGGTCTTTCATTTGGGGGCTCGTCCGGGATTTGGAGACC
CCTGCCCAGGGACCACCGACCCACCACCGGGAGGTAAGCTGGCCAGCAACTTATCTGTGTCTGTCCGATTGTCTAGTGT
CTATGTTTGATGTTATGCGCCTGCGTCTGTACTAGTTAGCTAACTAGCTCTGTATCTGGCGGACCCGTGGTGGAACTGA
CGAGTTCTGAACACCCGGCCGCAACCCTGGGAGACGTCCCAGGGACTTTGGGGGCCGTTTTTGTGGCCCGACCTGAGGA
AGGGAGTCGATGTGGAATCCGACCCCGTCAGGATATGTGGTTCTGGTAGGAGACGAGAACCTAAAACAGTTCCCGCCTC
CGTCTGAATTTTTGCTTTCGGTTTGGAACCGAAGCCGCGCGTCTTGTCTGCTGCAGCATCGTTCTGTGTTGTCTCTGTC
TGACTGTGTTTCTGTATTTGTCTGAAAATTAGGGCCAGACTGTTACCACTCCCTTAAGTTTGACCTTAGGTCACTGGAA
AGATGTCGAGCGGATCGCTCACAACCAGTCGGTAGATGTCAAGAAGAGACGTTGGGTTACCTTCTGCTCTGCAGAATGG
CCAACCTTTAACGTCGGATGGCCGCGAGACGGCACCTTTAACCGAGACCTCATCACCCAGGTTAAGATCAAGGTCTTTT
CACCTGGCCCGCATGGACACCCAGACCAGGTCCCCTACATCGTGACCTGGGAAGCCTTGGCTTTTGACCCCCCTCCCTG
GGTCAAGCCCTTTGTACACCCTAAGCCTCCGCCTCCTCTTCCTCCATCCGCCCCGTCTCTCCCCCTTGAACCTCCTCGT
TCGACCCCGCCTCGATCCTCCCTTTATCCAGCCCTCACTCCTTCTCTAGGCGCCGGAATTCGTTAACTCGAGGATCCGG
CTGTGGAATGTGTGTCAGTTAGGGTGTGGAAAGTCCCCAGGCTCCCCAGCAGGCAGAAGTATGCAAAGCATGCATCTCA
ATTAGTCAGCAACCAGGTGTGGAAAGTCCCCAGGCTCCCCAGCAGGCAGAAGTATGCAAAGCATGCATCTCAATTAGTC
AGCAACCATAGTCCCGCCCCTAACTCCGCCCATCCCGCCCCTAACTCCGCCCAGTTCCGCCCATTCTCCGCCCCATGGC
TGACTAATTTTTTTTATTTATGCAGAGGCCGAGGCCGCCTCGGCCTCTGAGCTATTCCAGAAGTAGTGAGGAGGCTTTT
TTGGAGGCCTAGGCTTTTGCAAAAAGCTGCCCAAGCTGATCCCCGGGGGCAATGAGATATGAAAAAGCCTGAACTCACC
GCGACGTCTGTCGAGAAGTTTCTGATCGAAAAGTTCGACAGCGTCTCCGACCTGATGCAGCTCTCGGAGGGCGAAGAAT
CTCGTGCTTTCAGCTTCGATGTAGGAGGGCGTGGATATGTCCTGCGGGTAAATAGCTGCGCCGATGGTTTCTACAAAGA
TCGTTATGTTTATCGGCACTTTGCATCGGCCGCGCTCCCGATTCCGGAAGTGCTTGACATTGGGGAATTCAGCGAGAGC
```

Introdução

Dizemos que um vetor $\texttt{pat}[0\,..\,\texttt{m}-1]$ **casa com** $\texttt{txt}[0\,..\,\texttt{n}-1]$ **a partir de** $\texttt{i}$ se

$$\texttt{pat}[0\,..\,\texttt{m}-1] = \texttt{txt}[\texttt{i}\,..\,\texttt{i}+\texttt{m}-1]$$

para algum $\texttt{i}$ em $[0\,..\,\texttt{n}-\texttt{m}]$.

Exemplo:

	0	1	2	3	4	5	6	7	8	9
txt	x	c	b	a	b	b	c	b	a	x

	0	1	2	3
pat	b	c	b	a

$\texttt{pat}[0\,..\,3]$ casa com $\texttt{txt}[0\,..\,9]$ a partir de 5.

Busca de substrings

Problema alternativo: Dados $\texttt{pat}[0\,..\,\texttt{m}-1]$ e $\texttt{txt}[0\,..\,\texttt{n}-1]$, encontrar o número de ocorrências de pat em txt.

Exemplo: Para $\texttt{n} = 10$, $\texttt{m} = 4$, e

	0	1	2	3	4	5	6	7	8	9
txt	b	b	a	b	a	b	a	c	b	a

	0	1	2	3
pat	b	a	b	a

pat ocorre 2 vezes em txt.

Algoritmo de força bruta

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													

Algoritmo de força bruta

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												

Algoritmo de força bruta

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											

Algoritmo de força bruta

pat = a b a b b a b a b b a

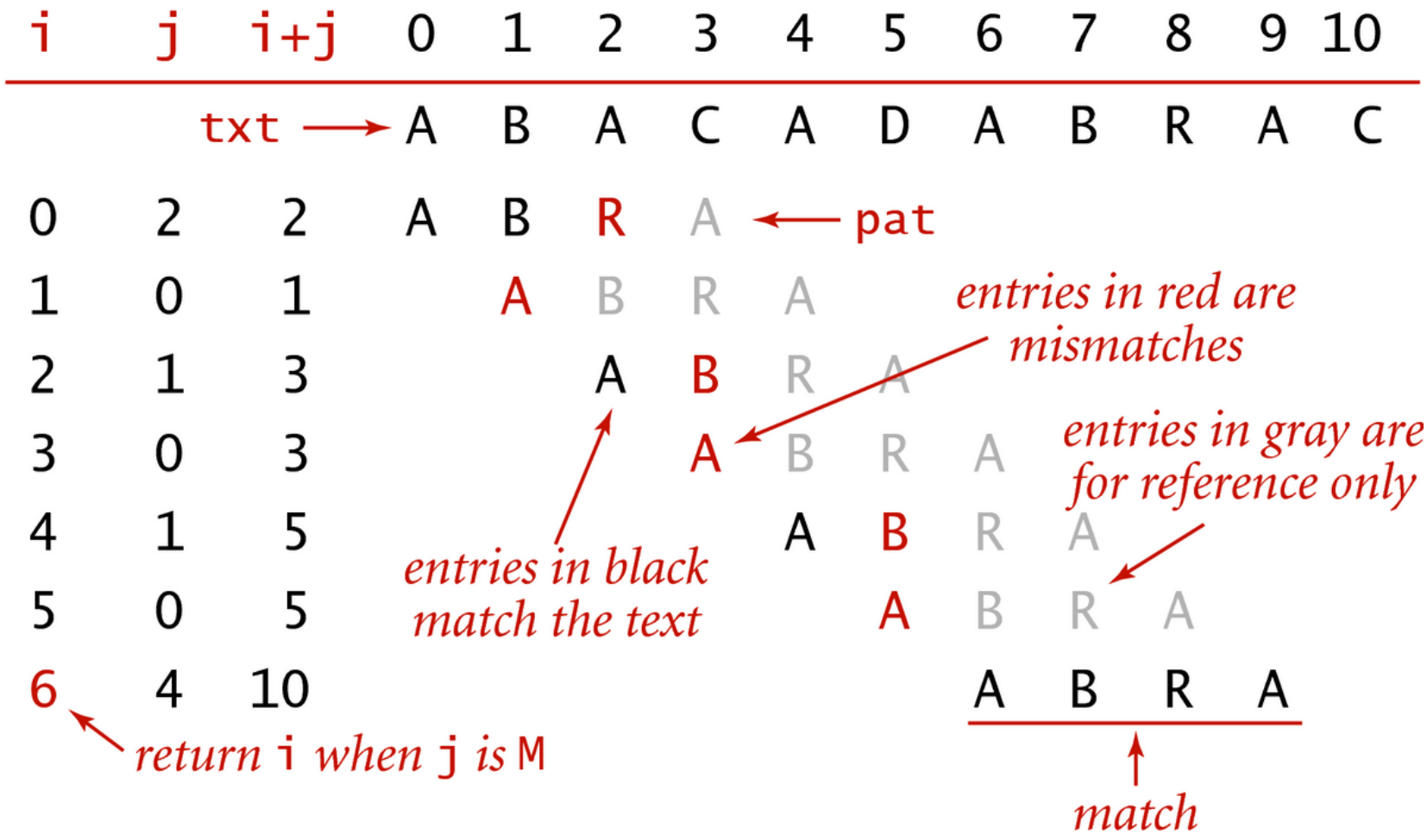
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										

Algoritmo de força bruta

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										
4					a	b	a	b	b	a	b	a	b	b	a									
5						a	b	a	b	b	a	b	a	b	b	a								
6							a	b	a	b	b	a	b	a	b	b	a							
7								a	b	a	b	b	a	b	a	b	b	a						
8									a	b	a	b	b	a	b	a	b	b	a					
9										a	b	a	b	b	a	b	a	b	b	a				
10											a	b	a	b	b	a	b	a	b	b	a			
11												a	b	a	b	b	a	b	a	b	b	a		
12													a	b	a	b	b	a	b	a	b	b	a	

Algoritmo de força bruta



Brute-force substring search

Algoritmo de força bruta

Devolve a primeira de ocorrências de pat em txt.

```
public static
int search(String pat, String txt) {
   int i, n = txt.length();
   int j, m = pat.length();
   for (i = 0; i <= n-m; i++) {
       for (j = 0; j < m; j++)
           if(txt.charAt(i+j)!=pat.charAt(j))
               break;
       if (j == m) return i;
   }
   return n;
}
```

Algoritmo de força bruta

Relação invariante: no início de cada iteração do "`for (j= 0; ...)`" vale que

(i0) $\texttt{pat}[0\,..\,\texttt{j}-1] = \texttt{txt}[\texttt{i}\,..\,\texttt{i}+\texttt{j}-1]$

Consumo de tempo

Consumo de tempo da função `search`().

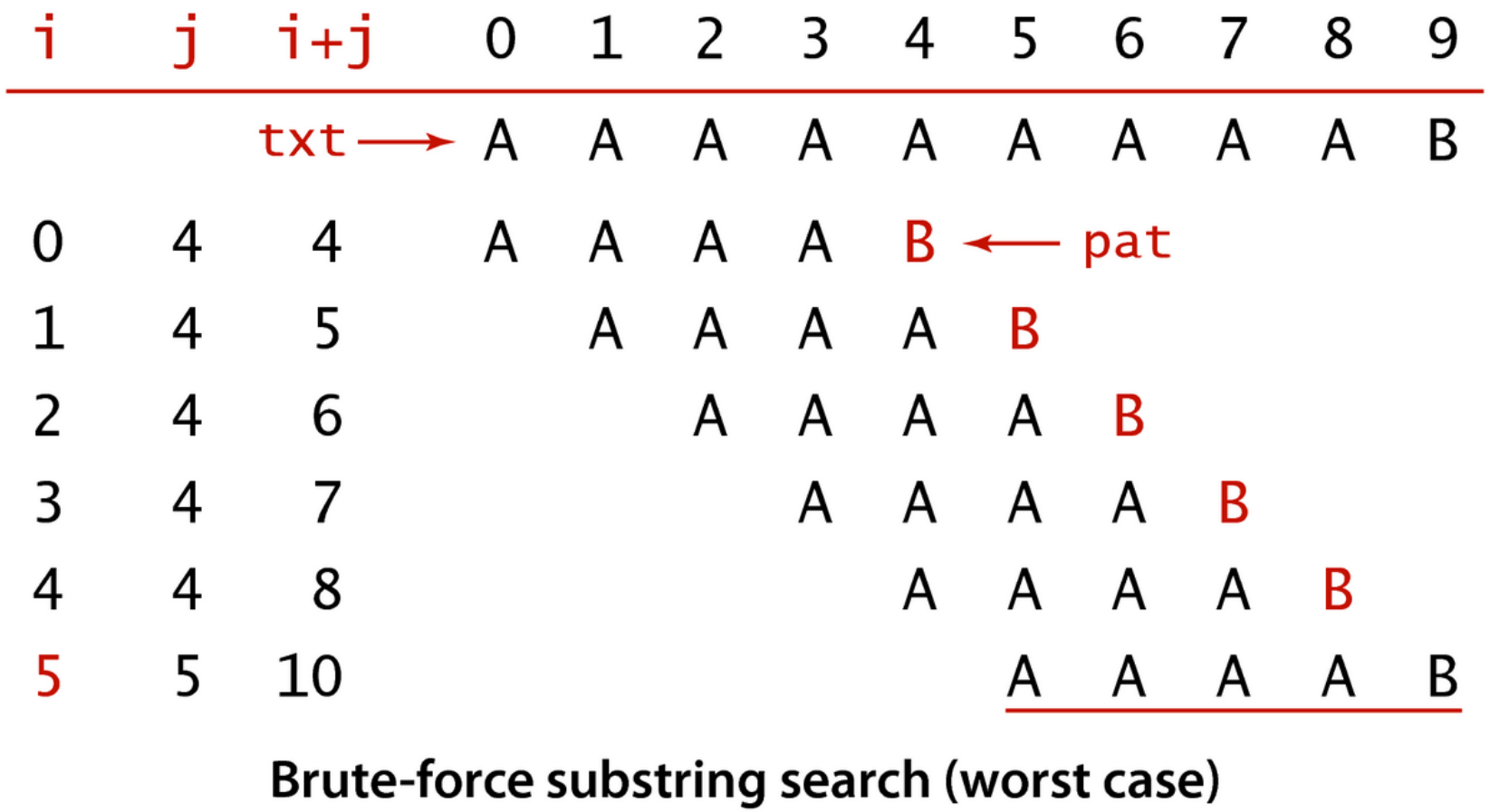
linha	todas as execuções da linha
1-2	$= 1$
3	$= \mathtt{n} - \mathtt{m} + 1$
4	$\leq (\mathtt{n} - \mathtt{m} + 1)(\mathtt{m} + 1)$
5	$\leq (\mathtt{n} - \mathtt{m} + 1)\,\mathtt{m}$
6	$\leq (\mathtt{n} - \mathtt{m} + 1)$
7	$= \mathtt{n} - \mathtt{m}$
8-9	$= 1$
total	$< 3\,(\mathtt{n} - \mathtt{m} + 2) + 2\,(\mathtt{n} - \mathtt{m} + 1)(\mathtt{m} + 1)$
	$= \mathrm{O}((\mathtt{n} - \mathtt{m} + 1)\mathtt{m})$

Pior caso

pat = a a a a a a a a a a b

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	txt
0	a	a	a	a	a	a	a	a	a	a	b													
1		a	a	a	a	a	a	a	a	a	a	b												
2			a	a	a	a	a	a	a	a	a	a	b											
3				a	a	a	a	a	a	a	a	a	a	b										
4					a	a	a	a	a	a	a	a	a	a	b									
5						a	a	a	a	a	a	a	a	a	a	b								
6							a	a	a	a	a	a	a	a	a	a	b							
7								a	a	a	a	a	a	a	a	a	a	b						
8									a	a	a	a	a	a	a	a	a	a	b					
9										a	a	a	a	a	a	a	a	a	a	b				
10											a	a	a	a	a	a	a	a	a	a	b			
11												a	a	a	a	a	a	a	a	a	a	b		
12													a	a	a	a	a	a	a	a	a	a	b	

Pior caso



Brute-force substring search (worst case)

Melhor caso

pat = b a a a a a a a a a a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	txt
0	b	a	a	a	a	a	a	a	a	a	a													
1		b	a	a	a	a	a	a	a	a	a	a												
2			b	a	a	a	a	a	a	a	a	a	a											
3				b	a	a	a	a	a	a	a	a	a	a										
4					b	a	a	a	a	a	a	a	a	a	a									
5						b	a	a	a	a	a	a	a	a	a	a								
6							b	a	a	a	a	a	a	a	a	a	a							
7								b	a	a	a	a	a	a	a	a	a	a						
8									b	a	a	a	a	a	a	a	a	a	a					
9										b	a	a	a	a	a	a	a	a	a	a				
10											b	a	a	a	a	a	a	a	a	a	a			
11												b	a	a	a	a	a	a	a	a	a	a		
12													b	a	a	a	a	a	a	a	a	a	a	a

Conclusões

O consumo de tempo de `search`() no pior caso é $O((n - m + 1)m)$.

O consumo de tempo de `search`() no melhor caso é $O(n - m + 1)$.

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a mn.

Em geral o algoritmo é rápido e faz não mais que $1.1 \times n$ comparações.

Algoritmo de força bruta: versão alternativa

```
public static
int search(String pat, String txt) {
   int i, n = txt.length();
   int j, m = pat.length();
   for (i=0, j=0; i < n && j < m; i++) {
       if(txt.charAt(i)==pat.charAt(j))j++;
       else {
           i -= j; // retrocesso
           j = 0;
       }
   }
   if (j == m) return i - m;
   return n; }
```

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										
4					a	b	a	b	b	a	b	a	b	b	a									

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										
4					a	b	a	b	b	a	b	a	b	b	a									
5						a	b	a	b	b	a	b	a	b	b	a								

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										
4					a	b	a	b	b	a	b	a	b	b	a									
5						a	b	a	b	b	a	b	a	b	b	a								
6							a	b	a	b	b	a	b	a	b	b	a							

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												
2			a	b	a	b	b	a	b	a	b	b	a											
3				a	b	a	b	b	a	b	a	b	b	a										
4					a	b	a	b	b	a	b	a	b	b	a									
5						a	b	a	b	b	a	b	a	b	b	a								
6							a	b	a	b	b	a	b	a	b	b	a							
7								a	b	a	b	b	a	b	a	b	b	a						
8									a	b	a	b	b	a	b	a	b	b	a					
9										a	b	a	b	b	a	b	a	b	b	a				
10											a	b	a	b	b	a	b	a	b	b	a			
11												a	b	a	b	b	a	b	a	b	b	a		
12													a	b	a	b	b	a	b	a	b	b	a	

Força bruta: direita para esquerda

Devolve a primeira de ocorrências de pat em txt.

```
public static
int search(String pat, String txt) {
   int i, n = txt.length();
   int j, m = pat.length();
   for (i = 0; i <= n-m; i+=1 /*skip*/) {
       for (j = m-1; j >= 0; j--)
           if(txt.charAt(i+j)!=pat.charAt(j))
              break;
       if (j == -1) return i;
   }
   return n;
}
```

Próximos passos

Existe algoritmo mais rápido que o força bruta?

Existe algoritmo que faz apenas `n` comparações entre caracteres?

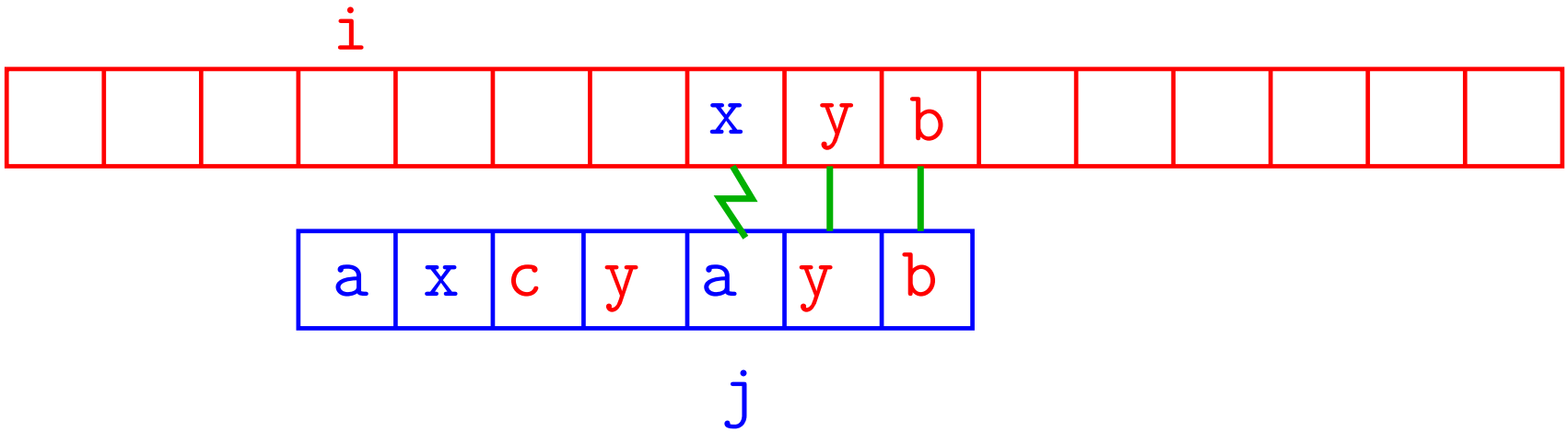
Existe algoritmo que faz menos que `n` comparações?

Boyer-Moore

Fonte: ADS: Boyer Moore String Search

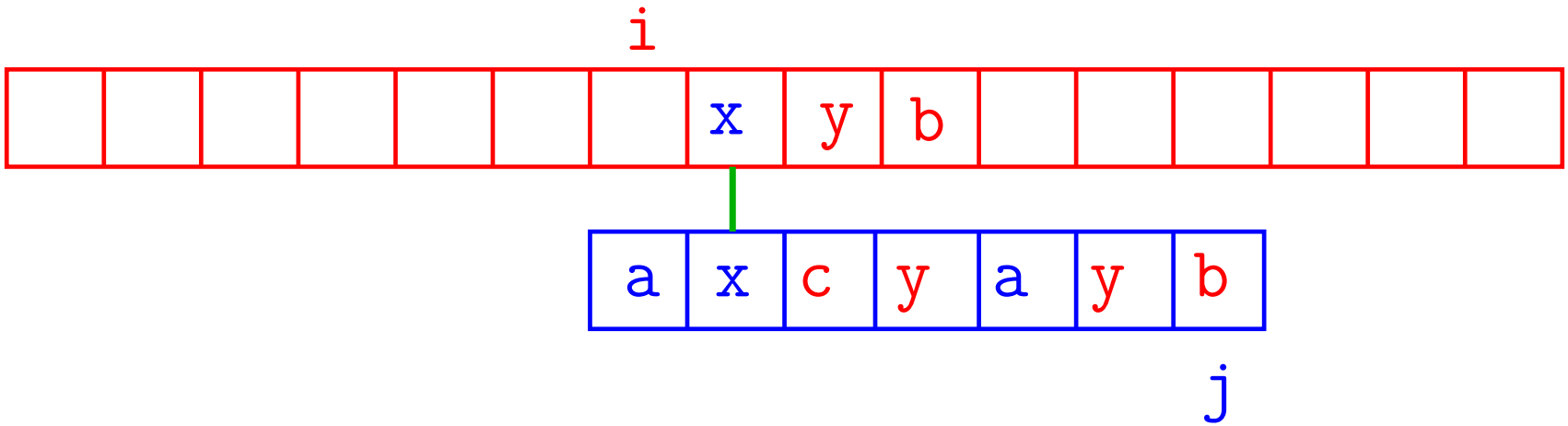
Primeiro algoritmo de Boyer-Moore

O primeiro algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Primeiro algoritmo de Boyer-Moore

O primeiro algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Boyer-Moore

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
1 a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
1 a n d a n d o
2       a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
1 a n d a n d o
2     a n d a n d o
3       a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
-------------------------------------------------------------------
1 a n d a n d o
2       a n d a n d o
3           a n d a n d o
4               a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
-------------------------------------------------------------------
1 a n d a n d o
2     a n d a n d o
3         a n d a n d o
4             a n d a n d o
5                 a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
   a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
  ---------------------------------------------------------------------------------------
1 a n d a n d o
2       a n d a n d o
3           a n d a n d o
4               a n d a n d o
5                   a n d a n d o
6                       a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
   a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
  ---------------------------------------------------------------------------------------
1 a n d a n d o
2       a n d a n d o
3           a n d a n d o
4               a n d a n d o
5                   a n d a n d o
6                       a n d a n d o
7                           a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
-------------------------------------------------------------------
1 a n d a n d o
2       a n d a n d o
3           a n d a n d o
4               a n d a n d o
5                   a n d a n d o
6                       a n d a n d o
7                           a n d a n d o
8                               a n d a n d o
```

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						
9									a	b	a	b	b	a	b	a	b	b	a					

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						
9									a	b	a	b	b	a	b	a	b	b	a					
10										a	b	a	b	b	a	b	a	b	b	a				

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						
9									a	b	a	b	b	a	b	a	b	b	a					
10										a	b	a	b	b	a	b	a	b	b	a				
11											a	b	a	b	b	a	b	a	b	b	a			

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						
9									a	b	a	b	b	a	b	a	b	b	a					
10										a	b	a	b	b	a	b	a	b	b	a				
11											a	b	a	b	b	a	b	a	b	b	a			
12												a	b	a	b	b	a	b	a	b	b	a		

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						
9									a	b	a	b	b	a	b	a	b	b	a					
10										a	b	a	b	b	a	b	a	b	b	a				
11											a	b	a	b	b	a	b	a	b	b	a			
12												a	b	a	b	b	a	b	a	b	b	a		
13													a	b	a	b	b	a	b	a	b	b	a	

Bad-character heuristic

Ideia ("*bad-character heuristic*"): calcular um deslocamento de modo que `txt[j]` fique emparelhado com a última ocorrência do caractere `txt[j]` em `pat`.

Suponha que o conjunto a que pertencem todos os elementos de `pat` e de `txt` é conhecido de antemão. Este conjunto é o **alfabeto** do problema.

Suponha que o alfabeto é o conjunto de todos os 256 caracteres.

Bad-character heuristic

Para implementar essa ideia fazemos um pré-processamento de `pat`, determinando para cada símbolo `x` do alfabeto a posição de sua última ocorrência em `pat`.

`pat`

0	1	2	3	4	5	6
a	n	d	a	n	d	o

`right`

0	...	'a'	'b'	'c'	'd'	...	...	'n'	'o'	'p'	...	255
-1	$\ldots$	3	-1	-1	5	$\ldots$	$\ldots$	4	6	-1	$\ldots$	$\ldots$

Classe BoyerMoore: esqueleto

```
public class BoyerMoore {
   private final int R; // tam.  alfabeto
   // pulo bad-character
   private int[] right;
   // padrão como array ou String
   private char[] pattern;
   private String pat;
   public BoyerMoore(String pat) {...}
   public BoyerMoore(char[] pattern,
                     int R) {...}
   public int search(String txt) {...}
   public int search(char[] text) {...}
}
```

BoyerMoore: construtor

```
public BoyerMoore(String pat) {
   this.R = 256;
   this.pat = pat;

   // última ocorrência de c em pat
   right = new int[R];
   for (int c = 0; c < R; c++)
       right[c] = -1;

   for (int j = 0; j < pat.length(); j++)
       right[pat.charAt(j)] = j;
}
```

BoyerMoore: search()

Recebe strings pat e txt com m $\geq$ 1 e n $\geq$ 0, e retorna o índice da primeira ocorrências de pat em txt. Se pat não ocorre em txt, retorna n.

```
public int search(String txt) {
   int n = txt.length();
   int m = pat.length();
   int skip;
```

BoyerMoore: search()

```
    for (int i = 0; i <= n-m; i += skip) {
        skip = 0;
        for (int j = m-1; j >= 0; j--) {
            if(pat.charAt(j)!=txt.charAt(i+j)){
                int r= right[txt.charAt(i+j)]
                skip = Math.max(1,j-r);
                break;
            }
        }
        if (skip == 0) return i; // achou
    }
    return n; // não achou
}
```

Pior caso

pat = b a a a a a a a a a a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	txt
1	b	a	a	a	a	a	a	a	a	a	a													
2		b	a	a	a	a	a	a	a	a	a	a												
3			b	a	a	a	a	a	a	a	a	a	a											
4				b	a	a	a	a	a	a	a	a	a	a										
5					b	a	a	a	a	a	a	a	a	a	a									
6						b	a	a	a	a	a	a	a	a	a	a								
7							b	a	a	a	a	a	a	a	a	a	a							
8								b	a	a	a	a	a	a	a	a	a	a						
9									b	a	a	a	a	a	a	a	a	a	a					
10										b	a	a	a	a	a	a	a	a	a	a				
11											b	a	a	a	a	a	a	a	a	a	a			
12												b	a	a	a	a	a	a	a	a	a	a		
13													a	b	a	a	a	a	a	a	a	a	a	a

Melhor caso

pat = a b c d e

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	txt
1	a	b	c	d	e																			

Melhor caso

pat = a b c d e

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	txt
1	a	b	c	d	e																			
2						a	b	c	d	e														

Melhor caso

pat = a b c d e

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	txt
1	a	b	c	d	e																			
2						a	b	c	d	e														
3											a	b	c	d	e									

Melhor caso

pat = a b c d e

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	txt
1	a	b	c	d	e																			
2						a	b	c	d	e														
3											a	b	c	d	e									
4																a	b	c	d	e				

Melhor caso

pat = a b c d e

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	?	x	?	?	?	txt
1	a	b	c	d	e																			
2						a	b	c	d	e														
3											a	b	c	d	e									
4																a	b	c	d	e				
5																					a	b	c	...

Conclusões

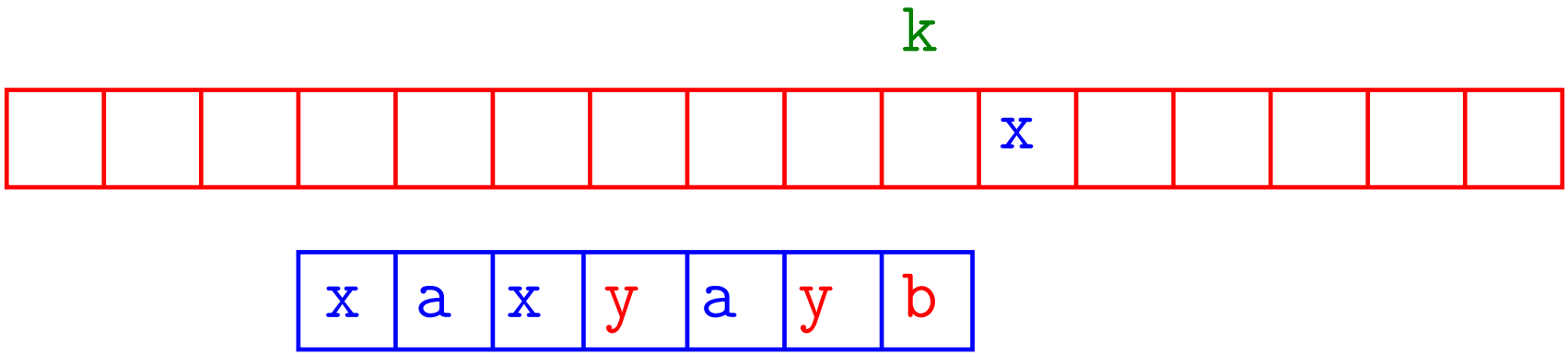
O consumo de tempo do algoritmo BoyerMoore no pior caso é $O((n - m + 1)m)$.

O consumo de tempo do algoritmo BoyerMoore no melhor caso é $O(n/m)$.

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a mn e no melhor caso o algoritmo é **sublinear**.

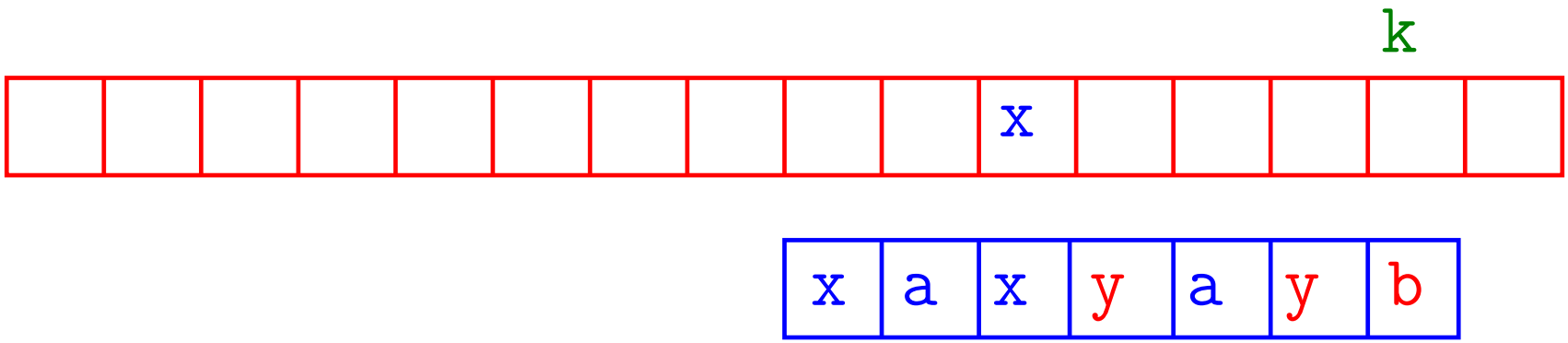
Bad-character heuristic: variante

O primeiro algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Bad-character heuristic: variante

O primeiro algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Bad-character heuristic: variante

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
1 a n d a n d o
```

Bad-character heuristic: variante

pat = a n d a n d o

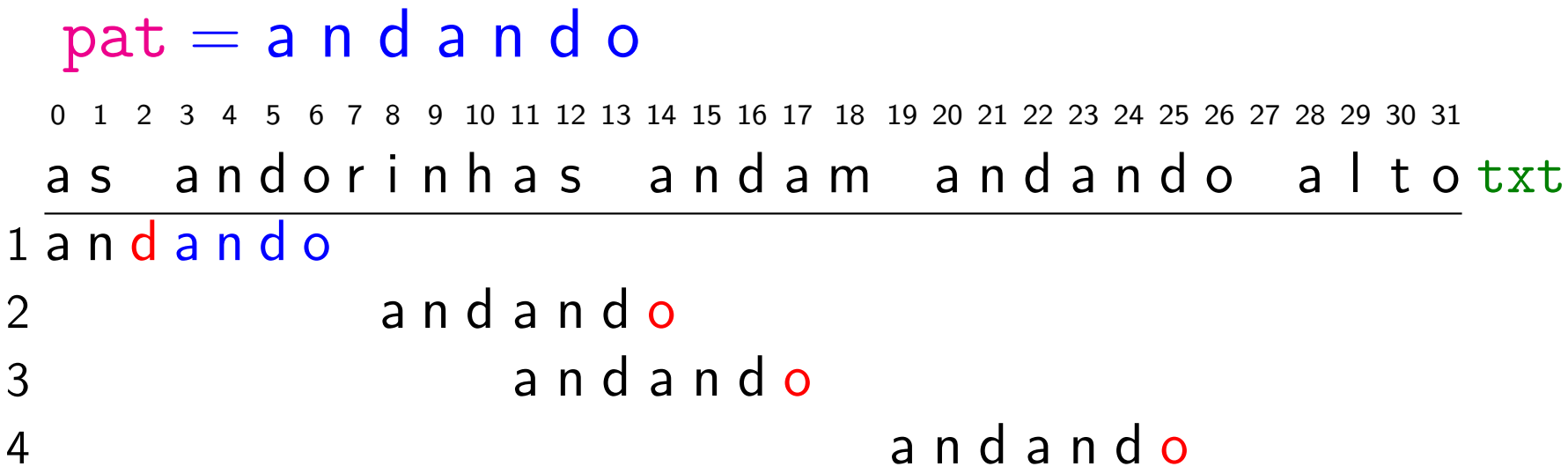
```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
1 a n d a n d o
2             a n d a n d o
```

Bad-character heuristic: variante

pat = a n d a n d o

```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
1 a n d a n d o
2             a n d a n d o
3                 a n d a n d o
```

Bad-character heuristic: variante



Bad-character heuristic: variante

pat = a n d a n d o

```
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
   a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o  txt
  ---------------------------------------------------------------------------------------
1 a n d a n d o
2             a n d a n d o
3                 a n d a n d o
4                                     a n d a n d o
5                                       a n d a n d o
```

Bad-character heuristic: variante

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											
3					a	b	a	b	b	a	b	a	b	b	a									

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											
3					a	b	a	b	b	a	b	a	b	b	a									
4							a	b	a	b	b	a	b	a	b	b	a							

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											
3					a	b	a	b	b	a	b	a	b	b	a									
4							a	b	a	b	b	a	b	a	b	b	a							
5								a	b	a	b	b	a	b	a	b	b	a						

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											
3					a	b	a	b	b	a	b	a	b	b	a									
4							a	b	a	b	b	a	b	a	b	b	a							
5								a	b	a	b	b	a	b	a	b	b	a						
6										a	b	a	b	b	a	b	a	b	b	a				

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											
3					a	b	a	b	b	a	b	a	b	b	a									
4							a	b	a	b	b	a	b	a	b	b	a							
5								a	b	a	b	b	a	b	a	b	b	a						
6										a	b	a	b	b	a	b	a	b	b	a				
7												a	b	a	b	b	a	b	a	b	b	a		

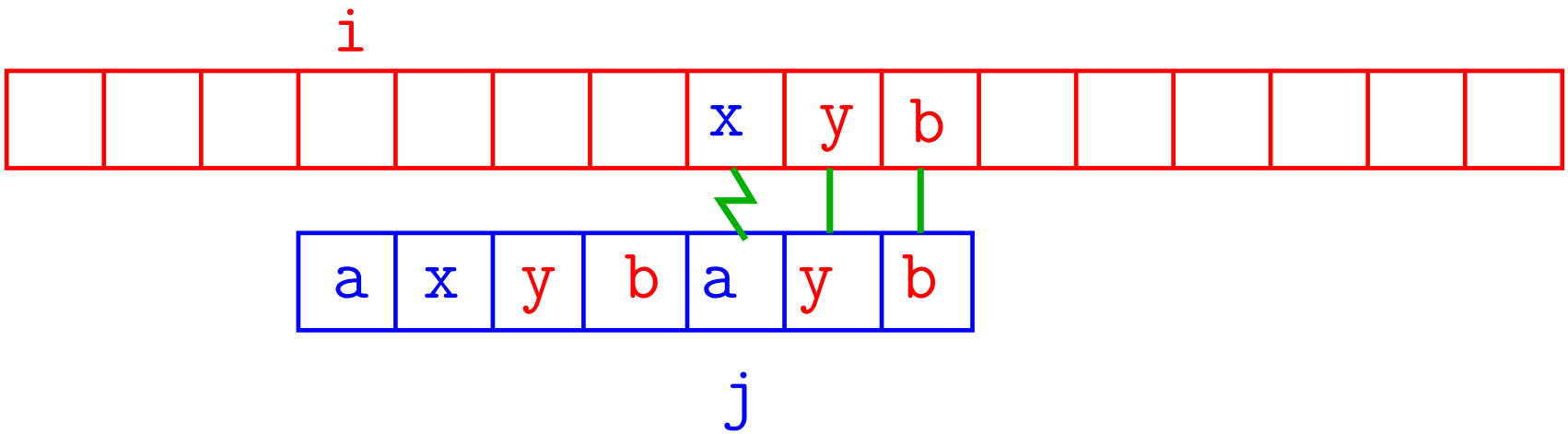
Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											
3					a	b	a	b	b	a	b	a	b	b	a									
4							a	b	a	b	b	a	b	a	b	b	a							
5								a	b	a	b	b	a	b	a	b	b	a						
6										a	b	a	b	b	a	b	a	b	b	a				
7												a	b	a	b	b	a	b	a	b	b	a		
8													a	b	a	b	b	a	b	a	b	b	a	

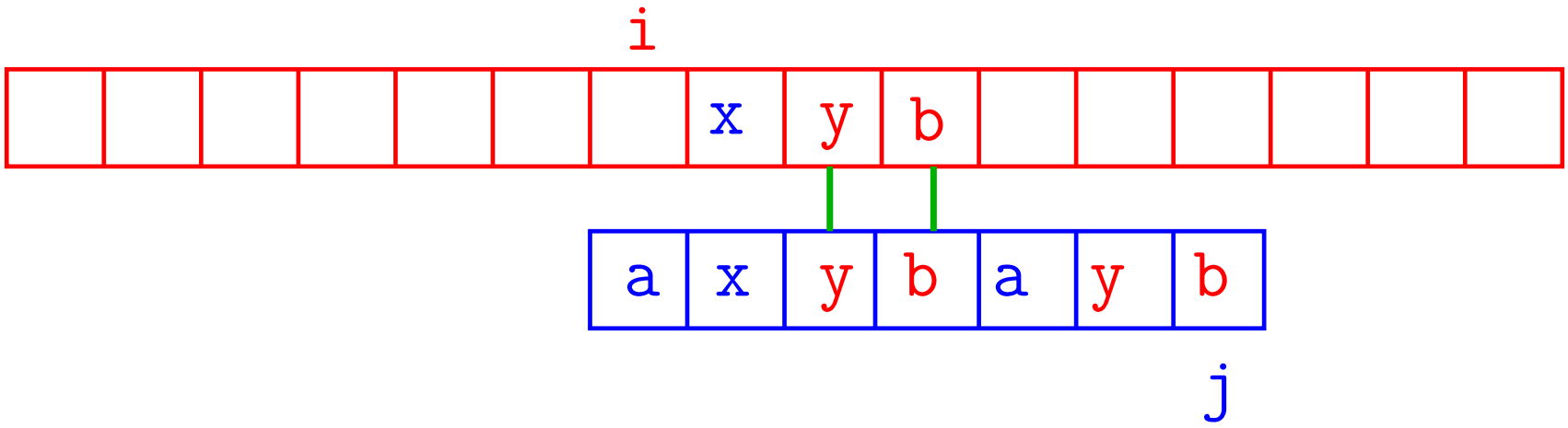
Segundo algoritmo de Boyer-Moore

O segundo algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Segundo algoritmo de Boyer-Moore

O segundo algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Good suffix heuristic

Não precisa conhecer o alfabeto explicitamente.

A implementação deve começar com um pré-processamento de `pat`: para cada `j` em $0, 1, \ldots,$ `m` $- 1$ devemos calcular o maior `k` em $0, 1, \ldots,$ `m` $- 2$ tal que:

- `pat[j..m-1]` é sufixo de `pat[0..k]` ou
- `pat[0..k]` é sufixo de `pat[j..m-1]`

Chamemos de `bm[j]` esse valor `k`.

Good suffix heuristic

Exemplo 1:

	0	1	2	3	4	5
pat	c	a	a	b	a	a

	0	1	2	3	4	5
bm	−1	−1	−1	−1	2	4

Exemplo 2:

	0	1	2	3	4	5	6	7
pat	b	a	–	b	a	∗	b	a

	0	1	2	3	4	5	6	7
bm	1	1	1	1	1	1	4	4

Good suffix heuristic

O vetor `bm[]` pode ser calculado facilmente em tempo $O(m^3)$.

Com mais trabalho, o vetor `bm[]` pode ser determinado em tempo $O(m)$. Veja CLRS, seção 34.4.

Veja também a página do professor Paulo Feofiloff: Busca de palavras em um texto