

Filas

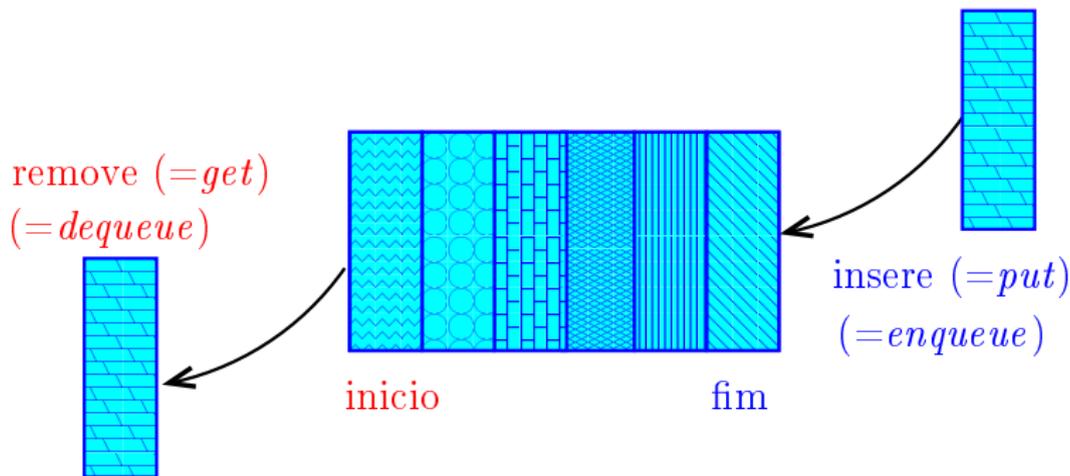


Fonte: <http://www.boreme.com/>
PF 5.1

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

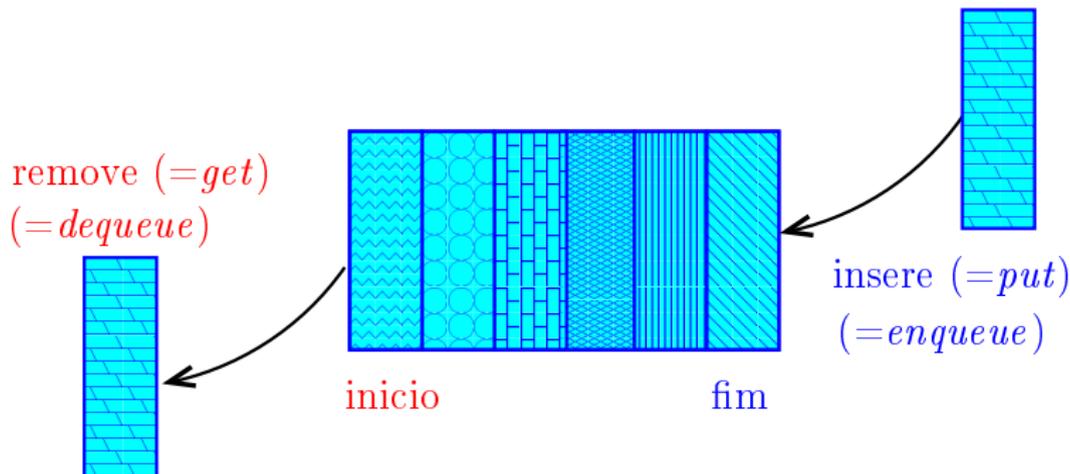
Filas

Uma **fila** (= *queue*) é uma lista dinâmica em que todas as **inserções** são feitas em uma extremidade chamada de **fim** e todas as **remoções** são feitas na outra extremidade chamada de **início**.

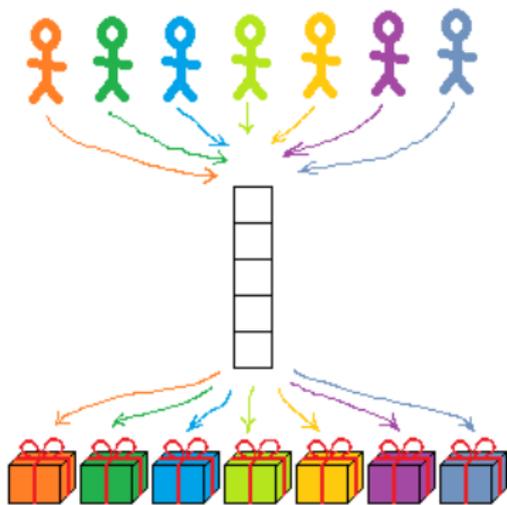


Filas

Assim, o primeiro objeto a ser removido de uma fila é o primeiro que foi inserido. Esta política de manipulação é conhecida pela sigla **FIFO** (= *First In First Out*)



Interface para filas



Fonte: <http://yosefk.com/blog>

S 4.6, 4.8

Implementação

```
class Fila:
    def __init__(self):
        self.itens = [ ]
    def __str__(self):
        return str(self.itens)
    def vazia(self):
        return self.itens == [ ]
    def insira(self, item):
        self.itens.append(item)
    def remove(self):
        return self.itens.pop(0)
    def __len__(self):
        return len(self.itens)
```

Distâncias

page4angels.blogspot.com



**“Dear Andy: How have you been?
Your mother and I are fine. We miss you.
Please sign off your computer and come
downstairs for something to eat. Love, Dad.”**

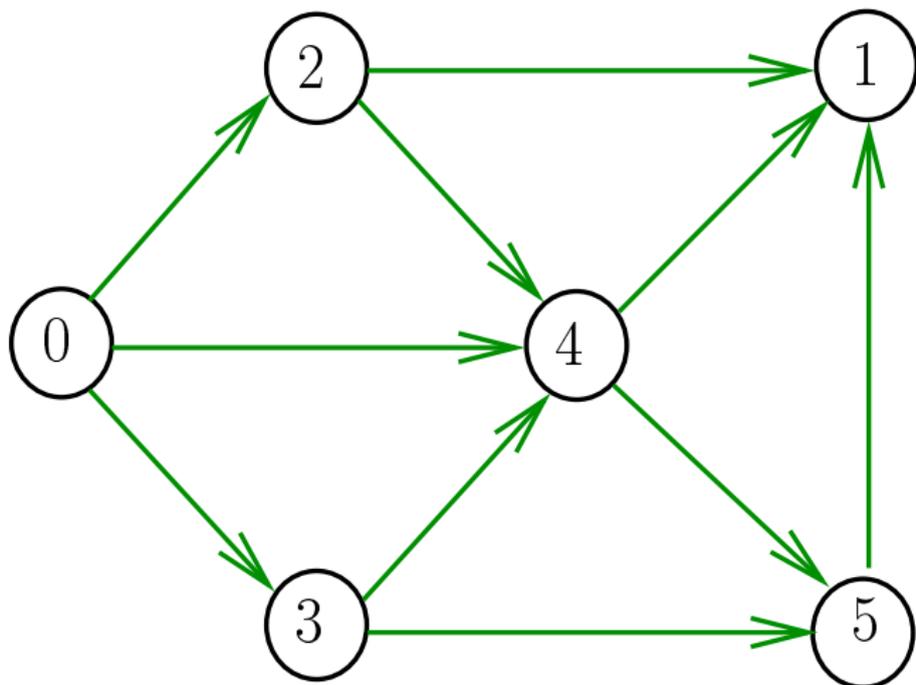
Fonte: <http://vandanasanju.blogspot.com.br/>

PF 5.2

<http://www.ime.usp.br/~pf/algoritmos/aulas/fila.html>

Rede de estradas

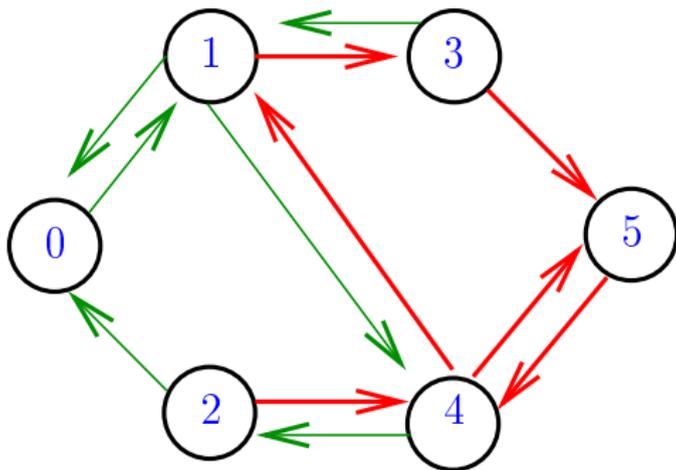
Considere n cidades numeradas de 0 a $n-1$ interligadas por estradas de mão única.



Comprimento

O **comprimento** de um caminho é o número de estradas no caminho, contando-se as repetições

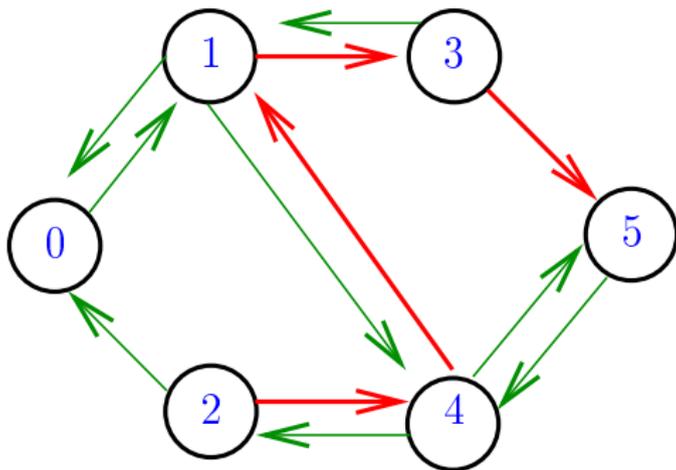
Exemplo: 2-4-1-3-5-4-5 tem comprimento 6



Comprimento

O **comprimento** de um caminho é o número de estradas no caminho, contando-se as repetições.

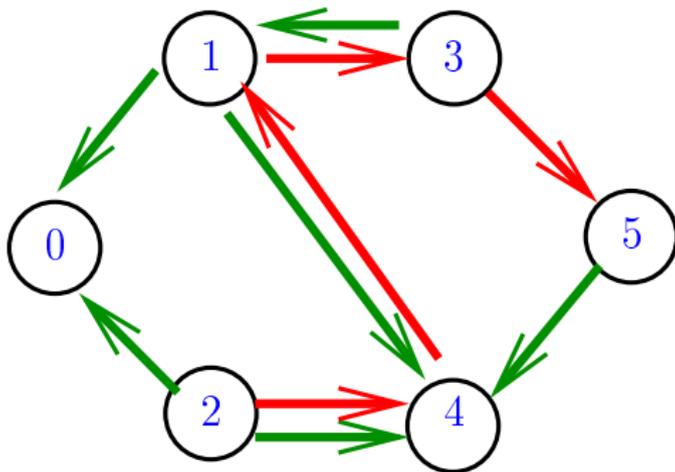
Exemplo: 2-4-1-3-5 tem comprimento 4



Distância

A **distância** de uma cidade c a uma cidade i é o menor comprimento de um caminho de c a i . Se não existe caminho de c a i a distância é “**infinita**”

Exemplo: a distância de 2 a 5 é 4

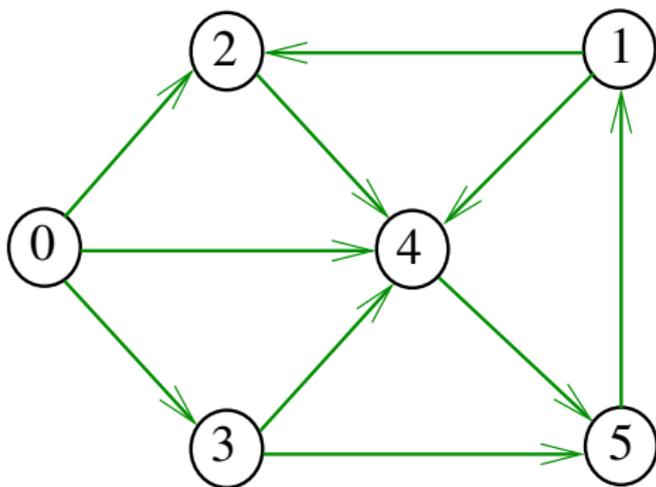


Calculando distâncias

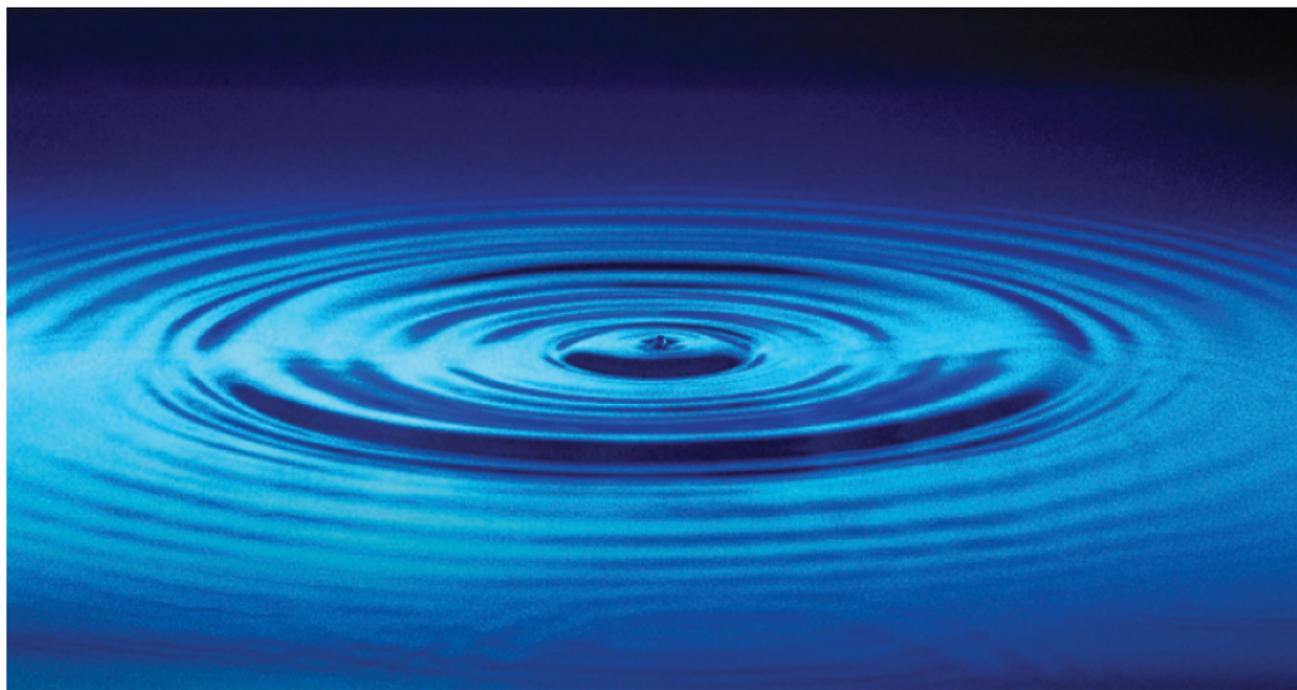
Problema: dada um rede de estradas e uma cidade c , determinar a distância de c a cada uma das demais cidades

Exemplo: para $c = 0$

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



Calculando distâncias

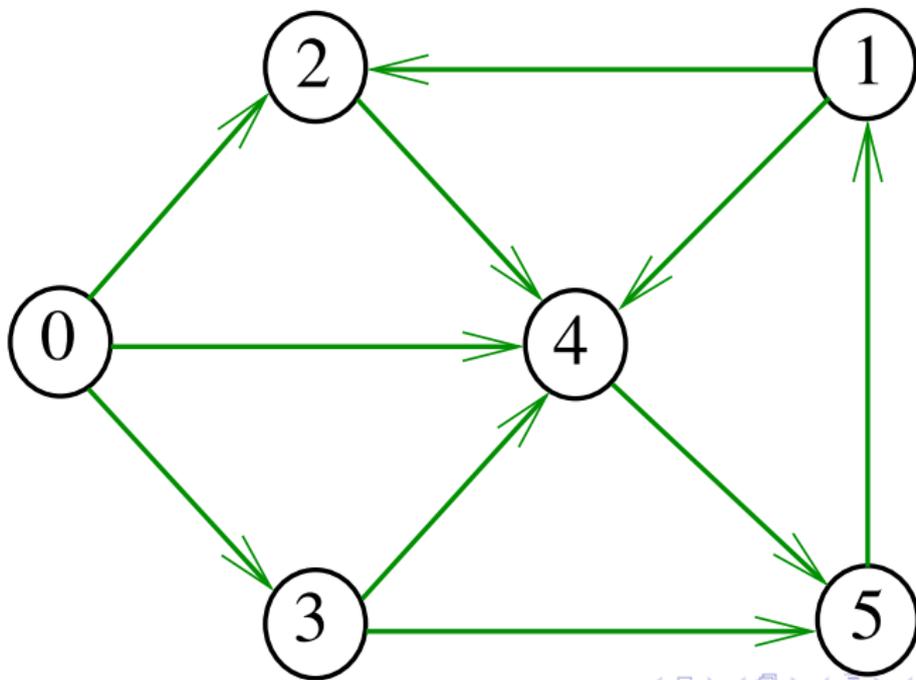


Fonte: <http://catalog.flatworldknowledge.com/bookhub/>

Simulação

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

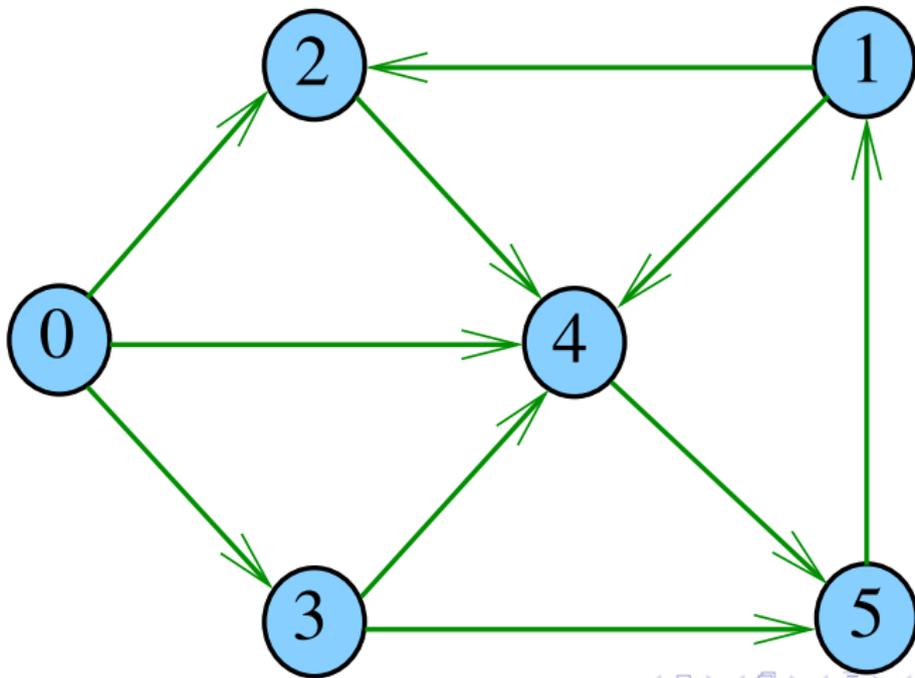
i	0	1	2	3	4	5
$d[i]$						



Simulação

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

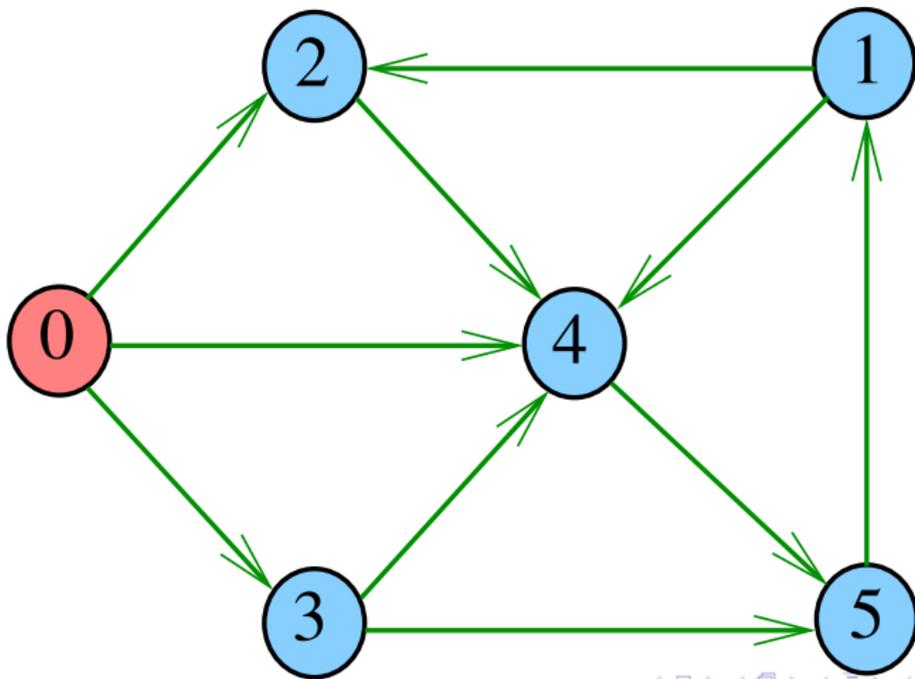
i	0	1	2	3	4	5
$d[i]$	6	6	6	6	6	6



Simulação

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

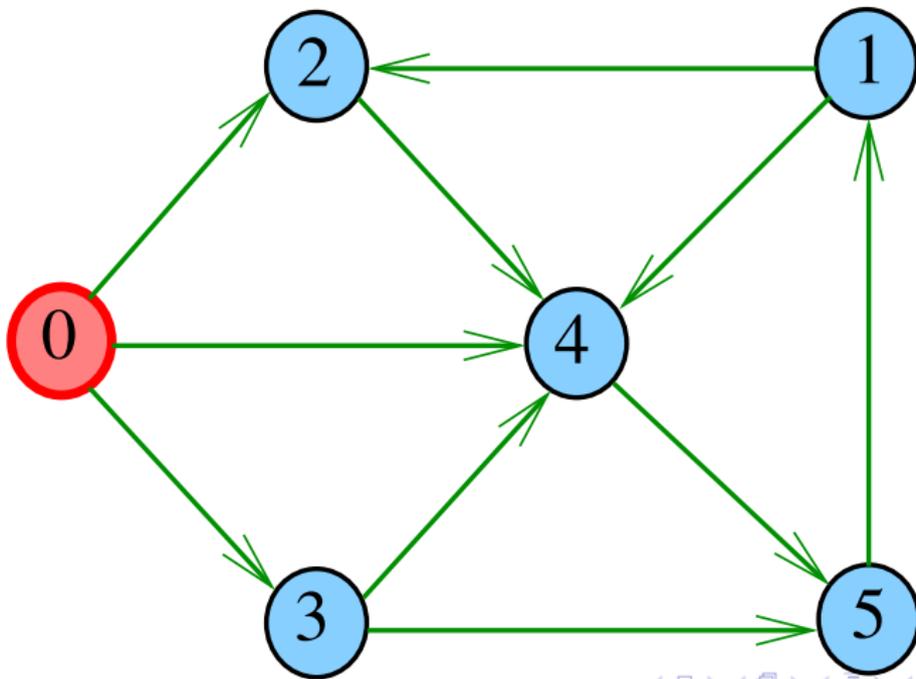
i	0	1	2	3	4	5
$d[i]$	6	6	6	6	6	6



Simulação

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

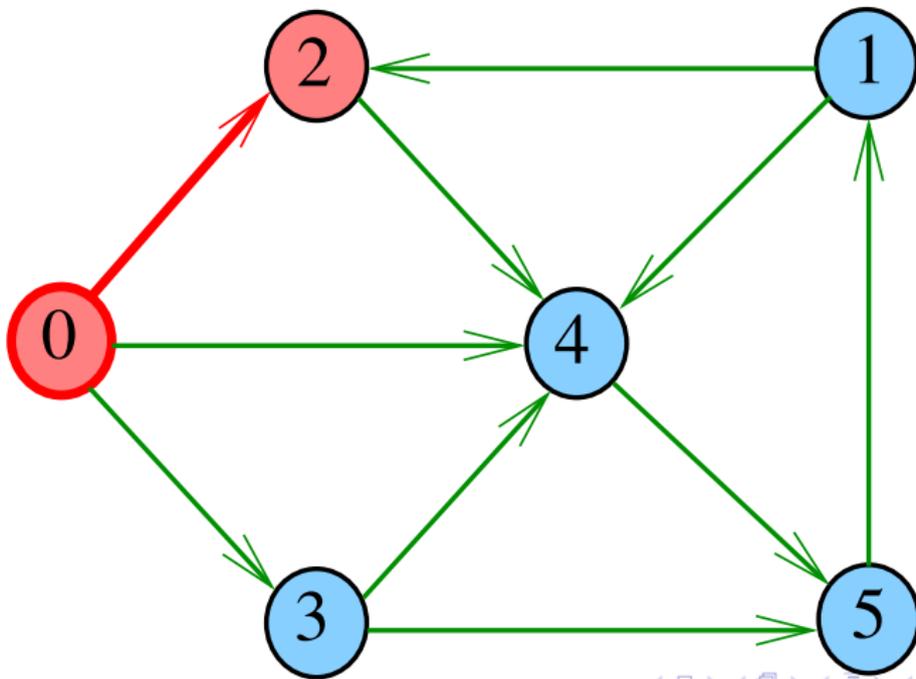
i	0	1	2	3	4	5
$d[i]$	0	6	6	6	6	6



Simulação

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

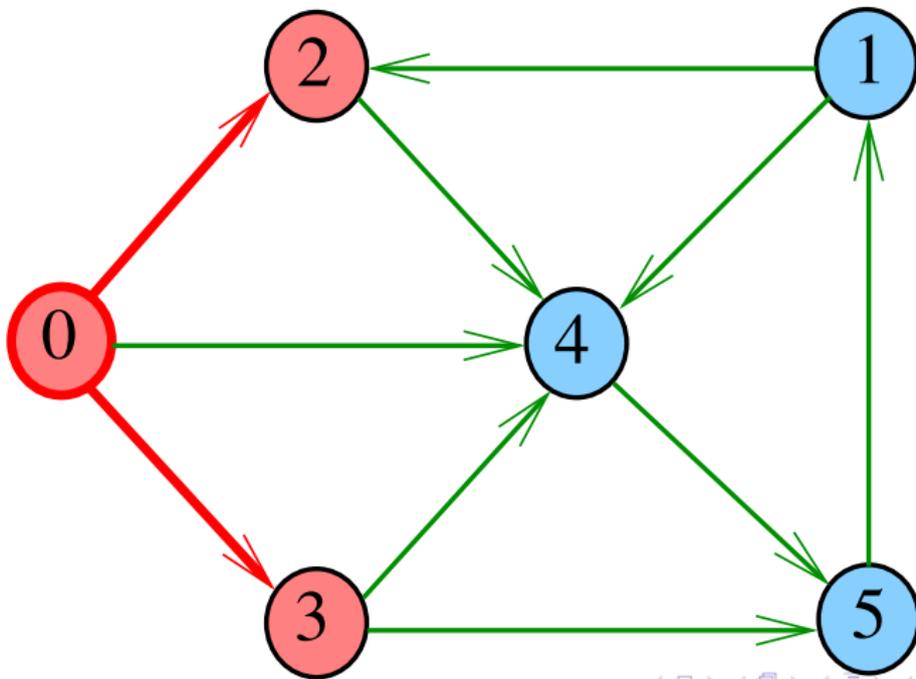
i	0	1	2	3	4	5
$d[i]$	0	6	1	6	6	6



Simulação

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

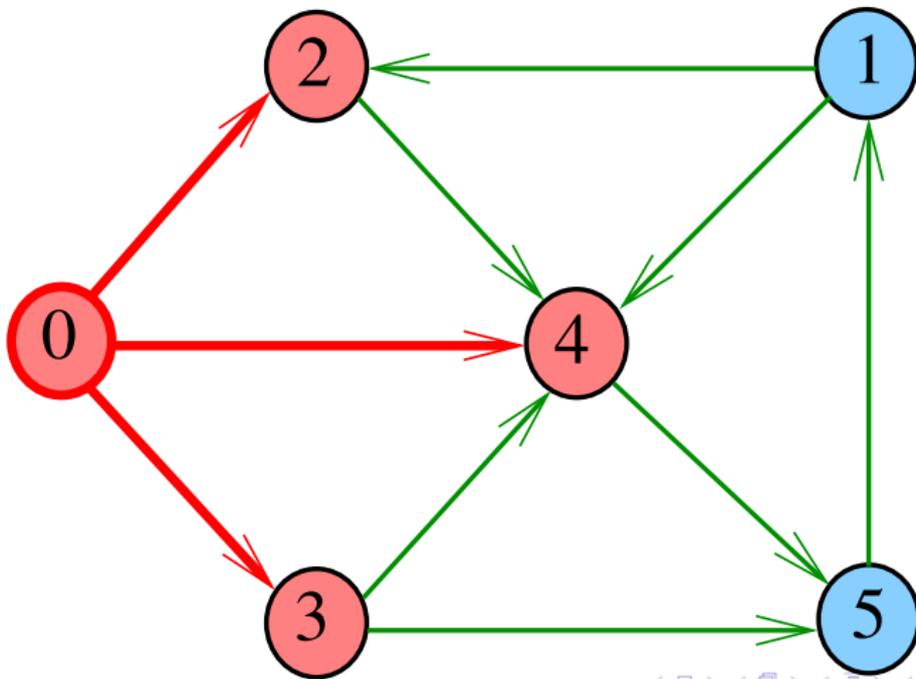
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	6	6



Simulação

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

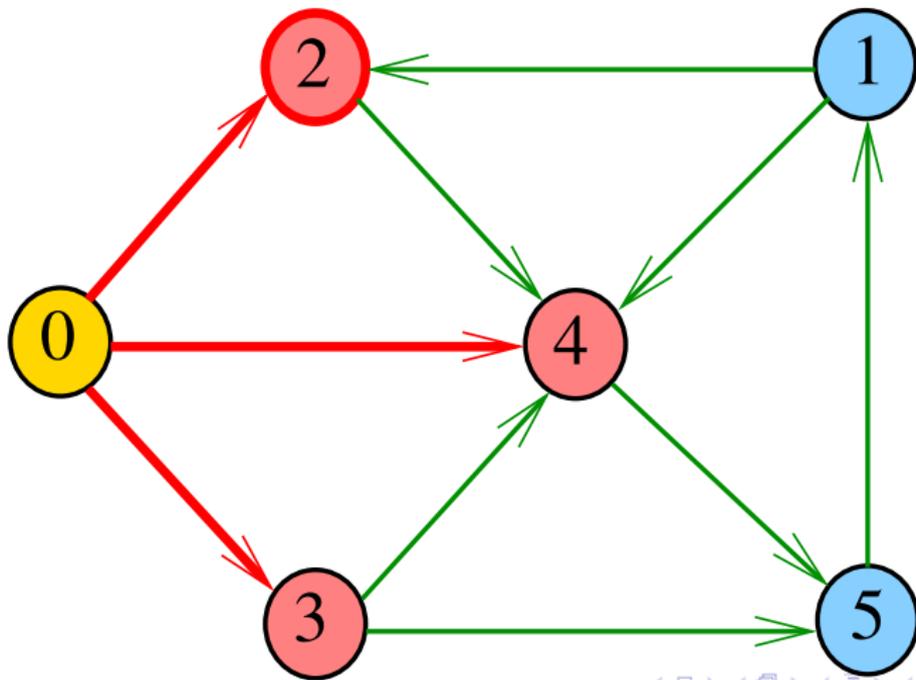
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	6



Simulação

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

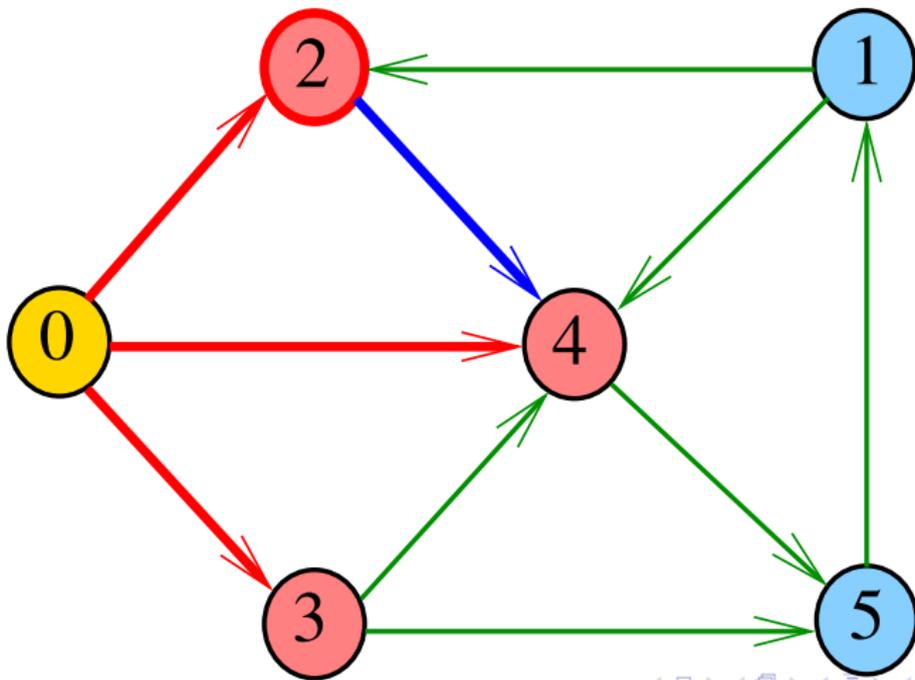
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	6



Simulação

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

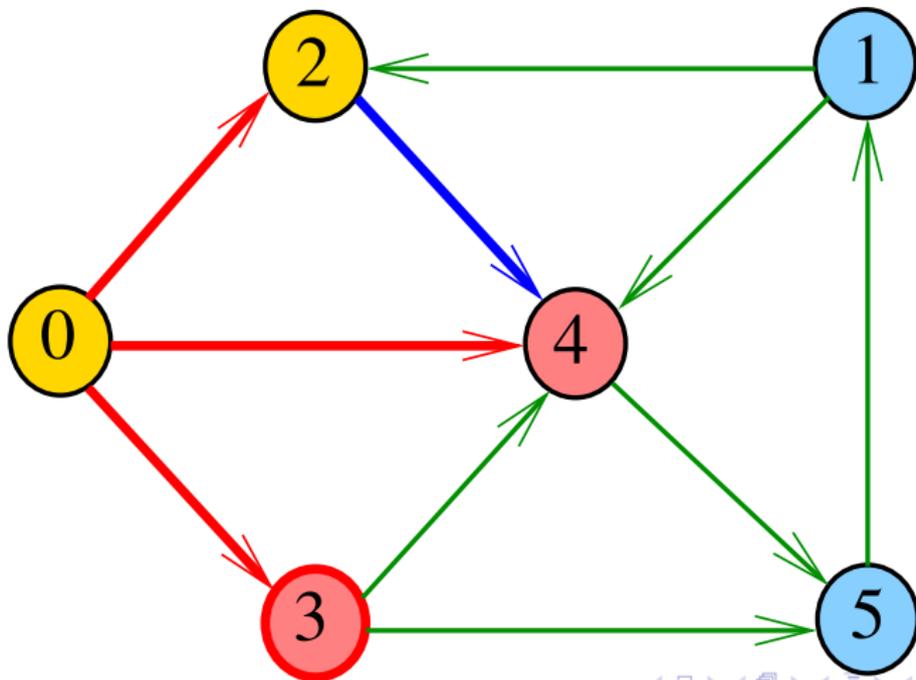
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	6



Simulação

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

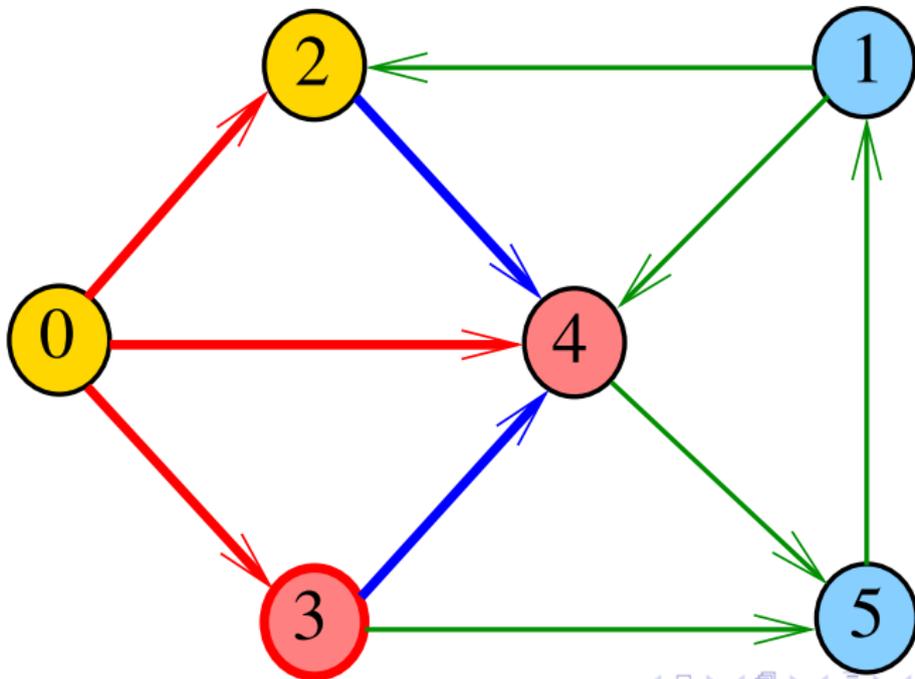
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	6



Simulação

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

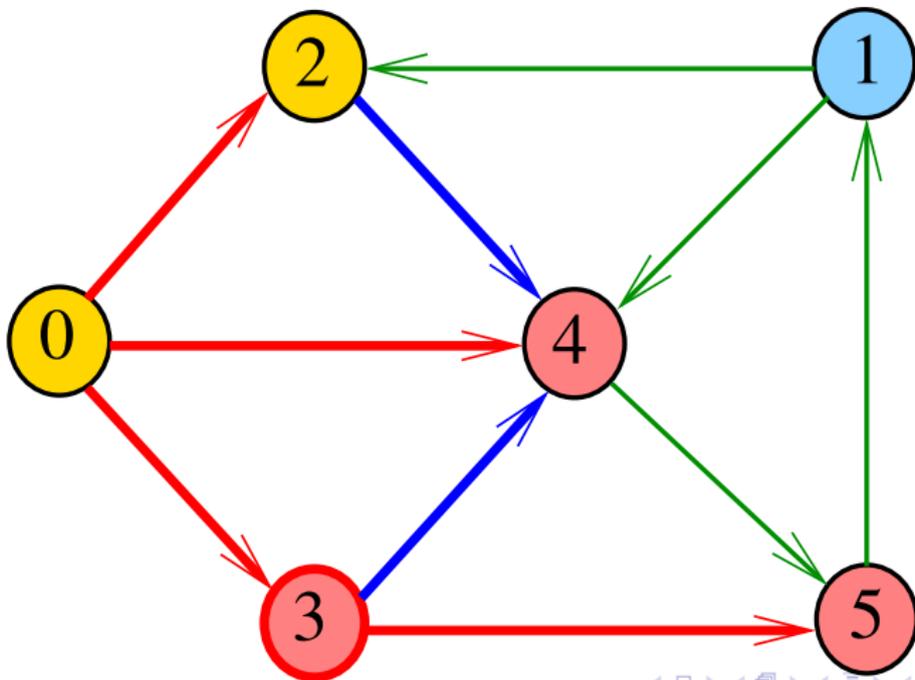
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	6



Simulação

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

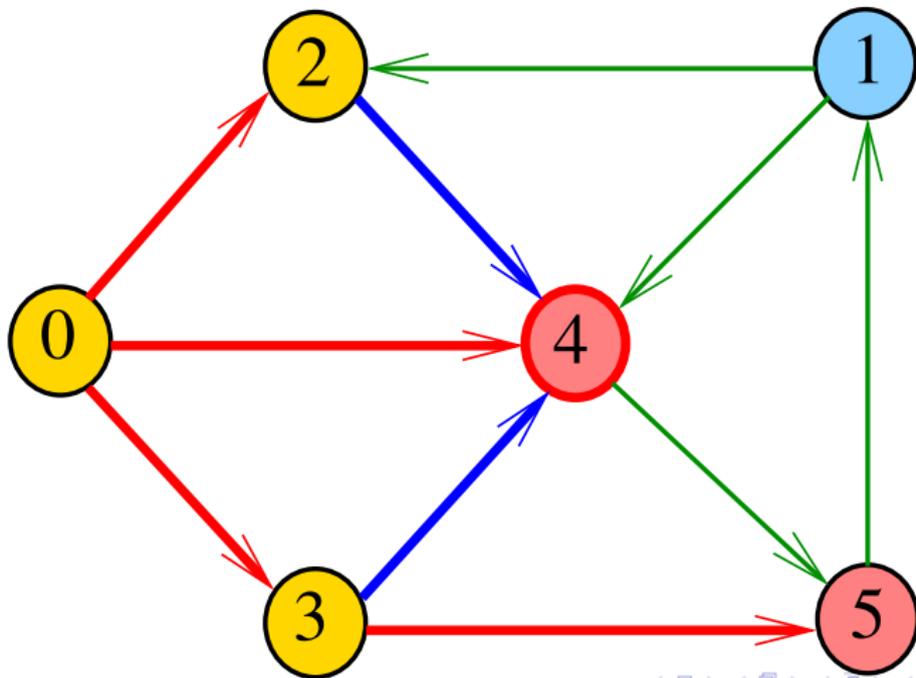
i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	2



Simulação

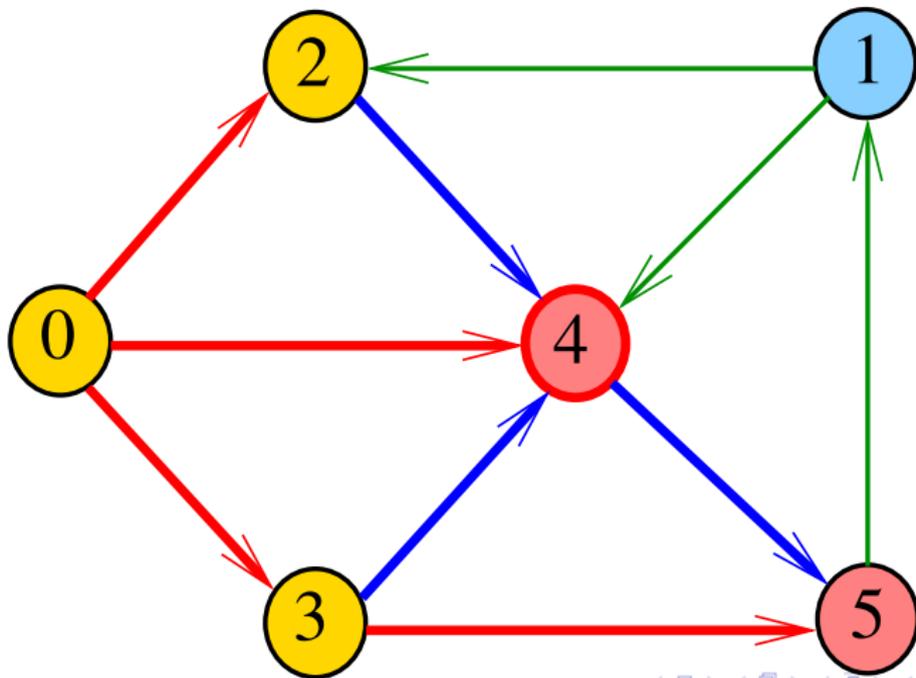
i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5	

i	0	1	2	3	4	5
$d[i]$	0	6	1	1	1	2



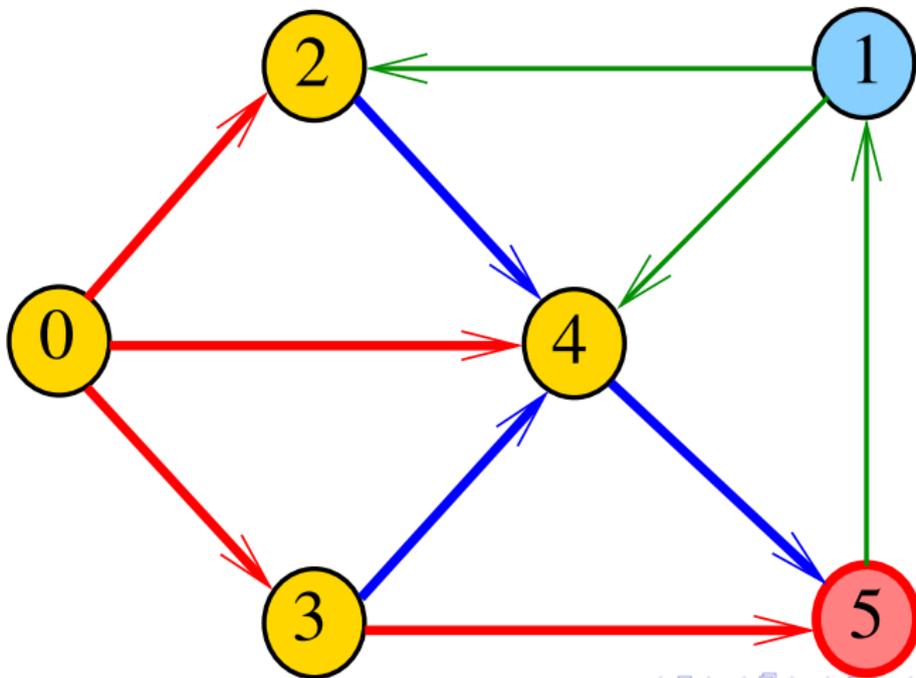
Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5		$d[i]$	0	6	1	1	1	2



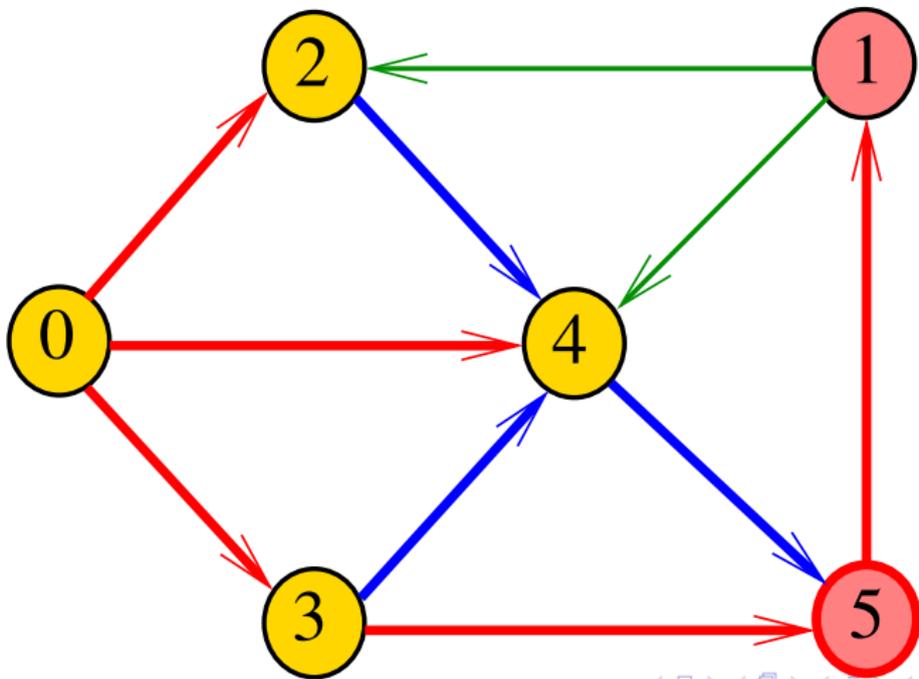
Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5		$d[i]$	0	6	1	1	1	2



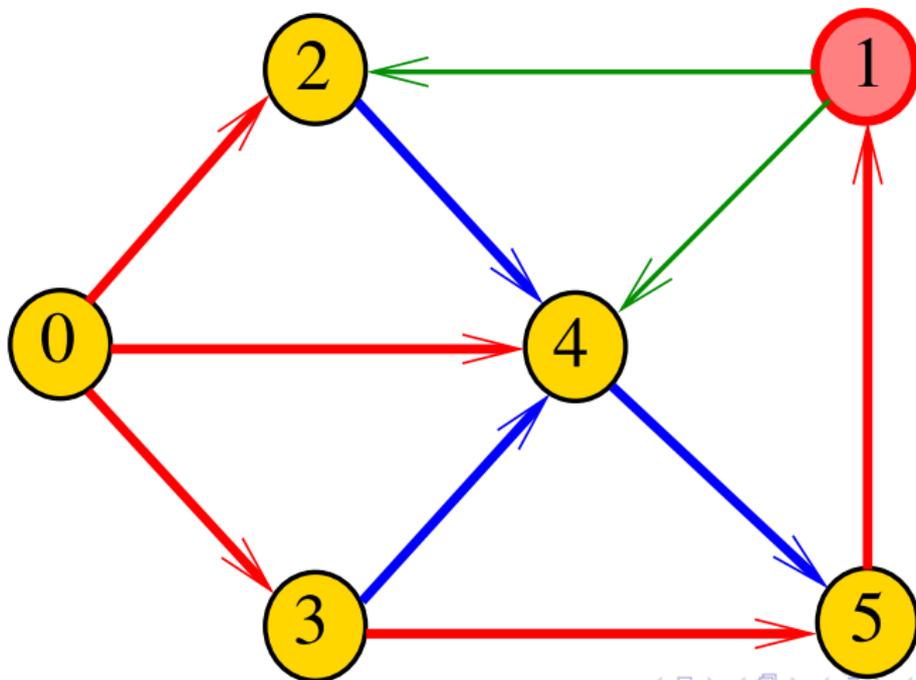
Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5	1	$d[i]$	0	3	1	1	1	2



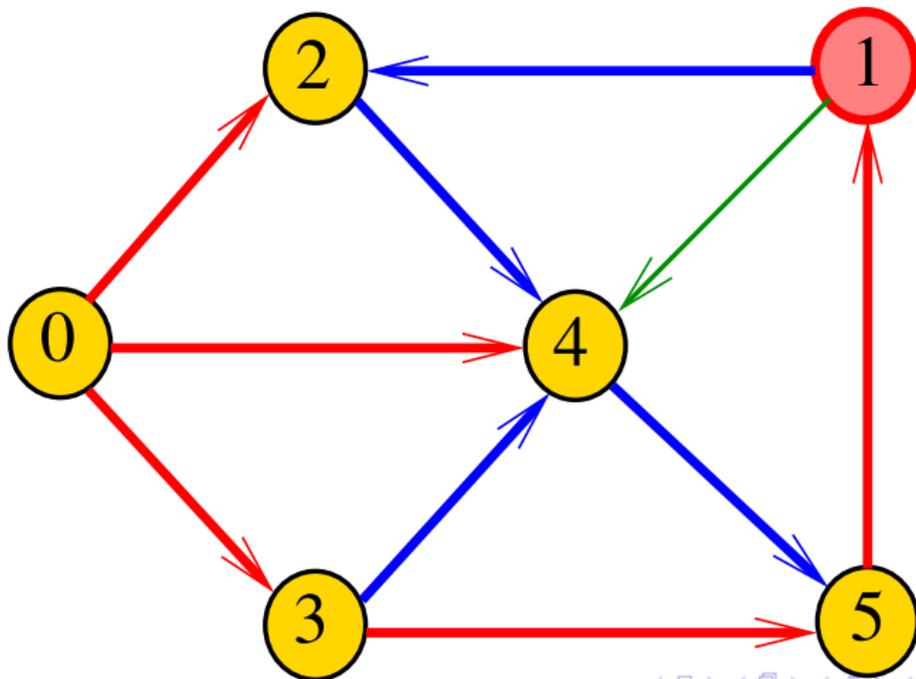
Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5	1	$d[i]$	0	3	1	1	1	2



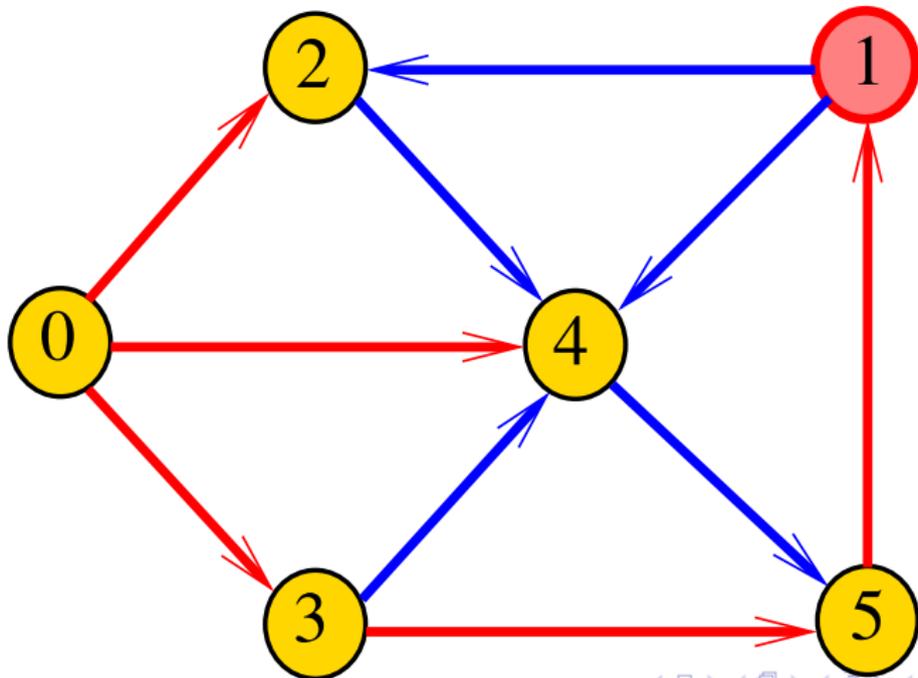
Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5	1	$d[i]$	0	3	1	1	1	2



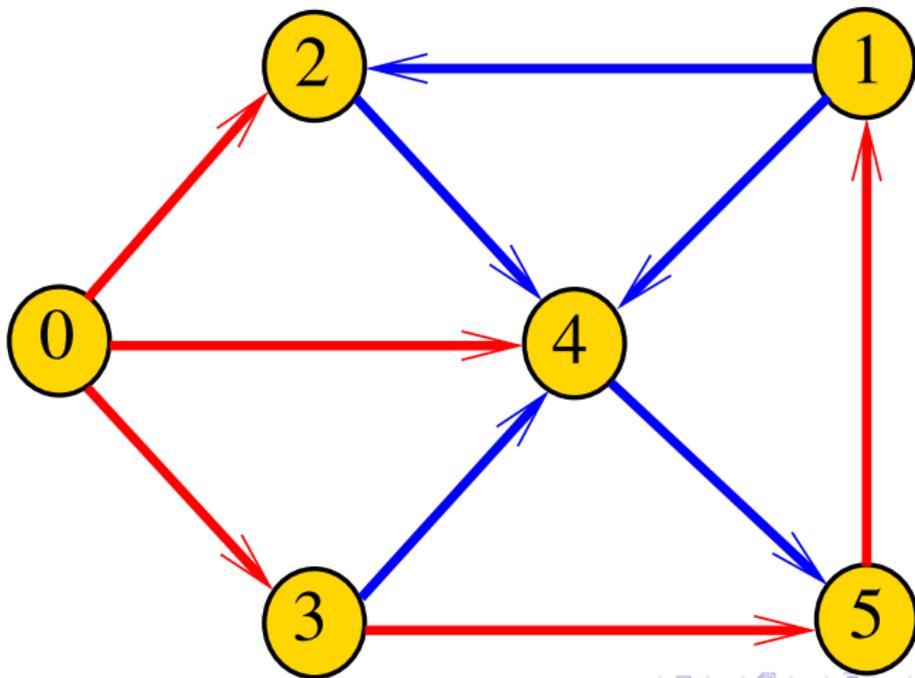
Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5	1	$d[i]$	0	3	1	1	1	2



Simulação

i	0	1	2	3	4	5	i	0	1	2	3	4	5
$q[i]$	0	2	3	4	5	1	$d[i]$	0	3	1	1	1	2



Representação da rede

A rede de estradas será representada por um objeto `rede` de uma classe `Rede` tal que:

`rede.existe_estrada(i, j) = True` se existe
estrada da cidade `i` para a cidade `j`

`rede.existe_estrada(i, j) = False` em caso
contrário

distancias

```
def distancias(c, rede):  
    '''(int, Rede) -> list  
    Recebe o índice  $c$  de uma cidade e uma  
    rede de estradas com  $n$  cidades.  
    A função cria e retorna uma lista  
     $d[0:n]$  tal que para  $i = 0, \dots, n-1$ ,  
     $d[i]$  é a distância da cidade  $c$  a  
    cidade  $i$ .  
    Se não existe caminho da cidade  $c$  a  
    cidade  $i$  então  $d[i]=n$ .  
    '''
```

distancias

```
# pegue o número de cidades da rede
n = len(rede)
# crie o vetor de distância com
# 'infinito' em cada posição
d = n * [n]
# a distância até origem é zero
d[c] = 0
# crie a fila de cidades
q = Fila()
# coloque a cidade origem na fila
q.insira(c)
```

distancias

```
while not q.vazia():
    # i é a cidade no início da fila
    i = q.remove()
    # examine as cidades vizinhas de i
    for j in range(n):
        if rede.existe_estrada(i,j) \
            and d[j] > d[i]+1:
            d[j] = d[i] + 1
            q.insira(j)
return d
```

Relações invariantes

No início de cada iteração do `while` a fila consiste em

*zero ou mais cidades à distância k de c ,
seguidos de zero ou mais cidades à
distância $k+1$ de c ,*

para algum k

Isto permite concluir que, no início de cada iteração, para toda cidade i , se $d[i] \neq n$ então $d[i]$ é a distância de c a i

Consumo de tempo

O consumo de tempo da função `distancias` é proporcional a n^2

O consumo de tempo da função `distancias` é proporcional a $O(n^2)$

Condição de inexistência

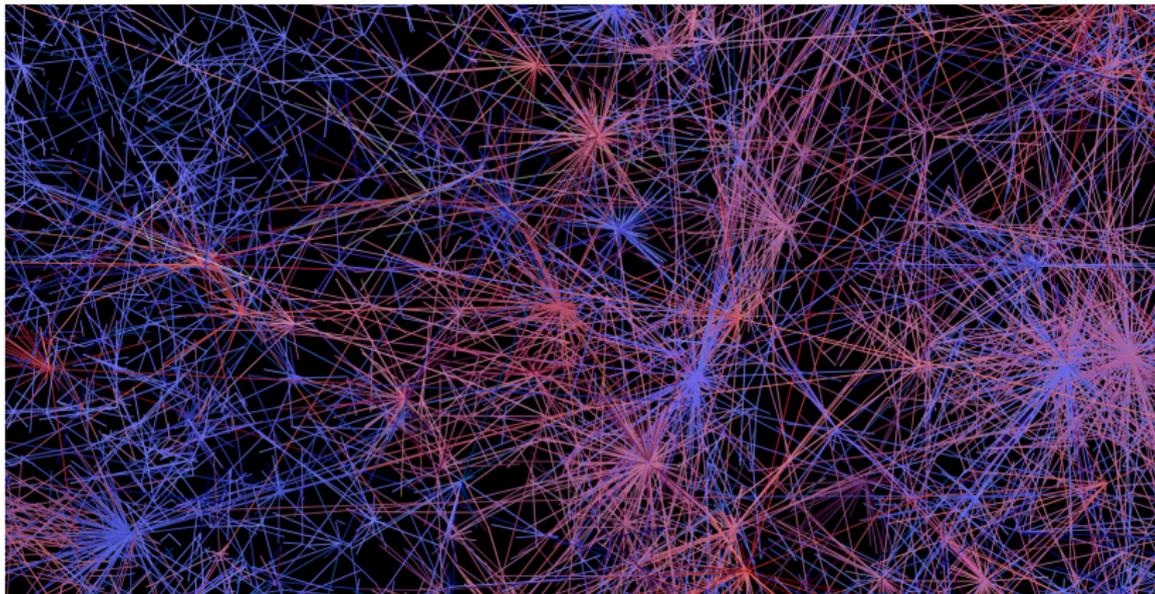
Se $d[i] == n$ para alguma cidade i , então

$$S = \{v : \text{dist}[v] < n\}$$

$$T = \{v : \text{dist}[v] == n\}$$

são tais que toda estrada entre cidades em S e cidades em T tem seu início em T e fim em S

Redes



Fonte:

<https://dhs.stanford.edu/gephi-workshop/>

Agora, representação da rede ...

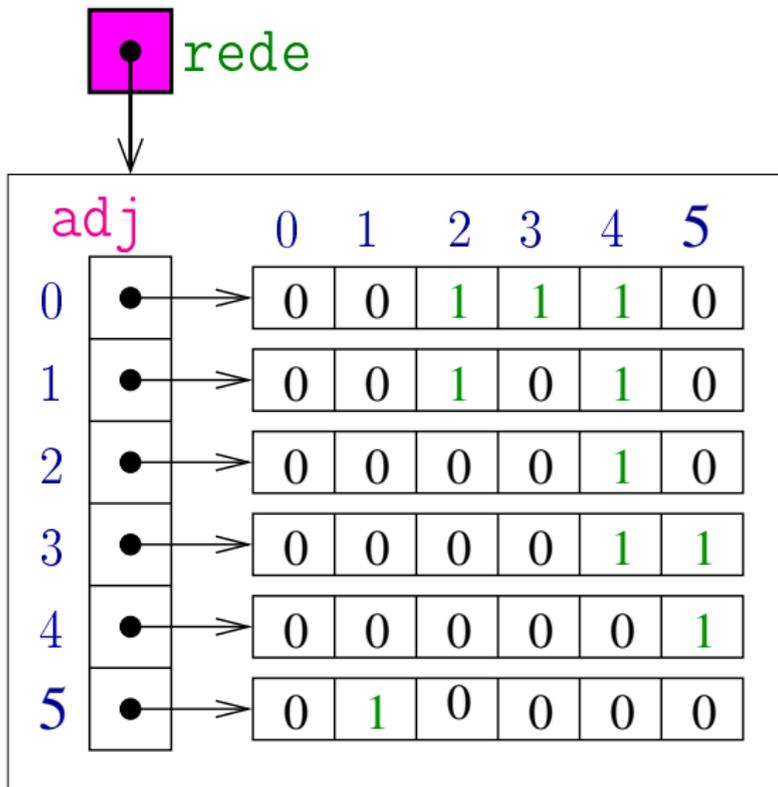
A rede de estradas será representada por um objeto `rede` de uma classe `Rede` tal que:

`rede.existe_estrada(i, j)` = `True` se existe
estrada da cidade `i` para a cidade `j`

`rede.existe_estrada(i, j)` = `False` em caso
contrário

Representaremos a rede através de uma estrutura de dados conhecida como **matriz de adjacência**.

Agora, representação da rede ...



Classe Rede

```
class Rede:
    def __init__(self, n):
        '''(Rede, intC) -> None
        Recebe um inteiro positivo n e retorna
        uma Rede com n cidades e sem estradas.
        As cidades são números entre 0 e n-1.
        self.adj[i][j] == 1, existe estrada
        self.adj[i][j] == 0, não existe
        estrada cd i a j.
        '''
        self.adj = crie_matriz(n,n,0)
```

Classe Rede

```
def __str__(self):  
    '''(Rede) -> str  
    Recebe uma Rede self e cria  
    e retorna um string que representa  
    a rede.  
    '''  
    s = ''  
    adj = self.adj # apelido  
    n = self.__len__()  
    for i in range(n):  
        for j in range(n):  
            s += str(adj[i][j]) + ' '  
        s += '\n'  
    return s
```

Classe Rede

```
def insira_estrada(self, i, j):  
    '''(Rede, int, int) -> None  
    Recebe uma Rede self e um par de  
    inteiros representando cidades e  
    insere na rede a estrada de i a j.  
    '''  
    self.adj[i][j] = 1
```

Classe Rede

```
def existe_estrada(self, i, j):  
    '''(Rede, int, int) -> bool  
    Recebe uma Rede self e um par de  
    inteiros representando cidades e  
    retorna True se existe estrada de i  
    a j e False em caso contrário.  
    '''  
    return self.adj[i][j] == 1
```

Classe Rede

```
def __len__(self):  
    '''(Rede) -> int  
    Recebe uma Rede self e retorna o  
    número de cidades na rede  
    '''  
    return len(self.adj)
```

```
def funcao_crie_matriz():
def crie_matriz(n_lin, n_col, valor):
    '''(int,int,obj) -> list of list
    Recebe inteiros não negativos n_lin e
    n_col e cria e retorna uma matriz com
    n_lin linhas e n_col colunas em que
    toda posição é inicializada com valor.
    '''
    matriz = []
    for i in range(n_lin):
        linha = []
        for j in range(n_col):
            linha.append(valor)
        matriz.append(linha)
    return matriz
```