

AULA 14

Filas



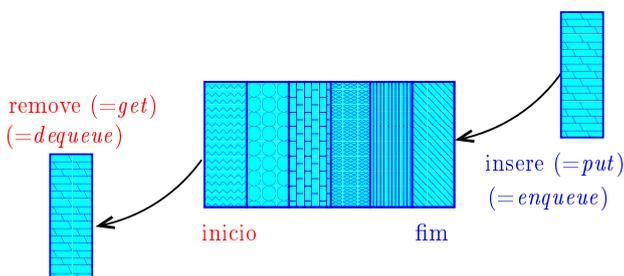
Fonte: <http://www.boreme.com/>

PF 5.1

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

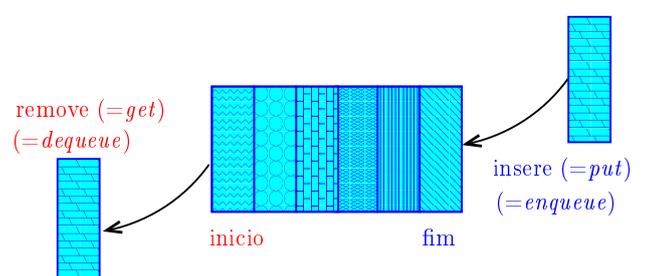
Filas

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**.



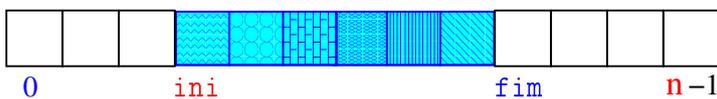
Filas

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*)



Implementação em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



O índice **ini** indica o **primeiro** da fila.

O índice **fim-1** indica o **último** da fila.

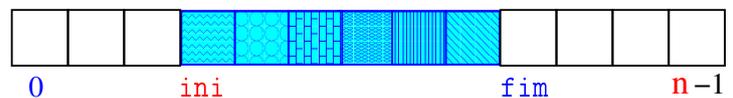
fim é a **primeira posição vaga** da fila.

A fila está **vazia** se "**ini == fim**".

A fila está **cheia** se "**fim == n**".

Implementação em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



Para **remover** (= *dequeue=get*) um elemento faça

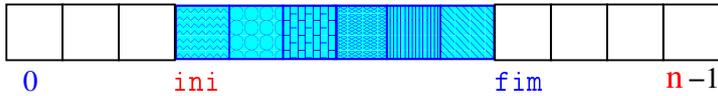
```
x = q[ini++];
```

que é equivalente a

```
x = q[ini];  
ini += 1;
```

Implementação em um vetor

A fila será armazenada em um vetor $q[0 \dots n-1]$.



Para *inserir* (= *queue=put*) um elemento faça

```
q[fim++] = x;
```

que é equivalente a

```
q[fim] = x;
fim += 1;
```

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

Distâncias



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

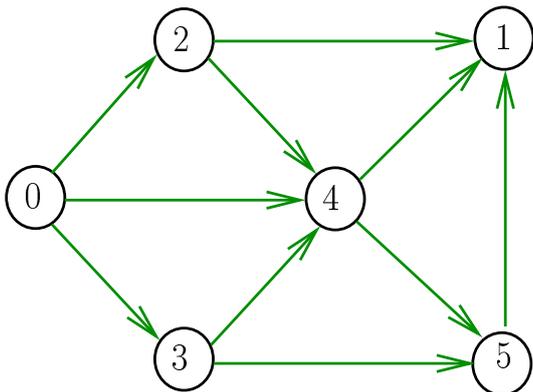
PF 5.2

<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.

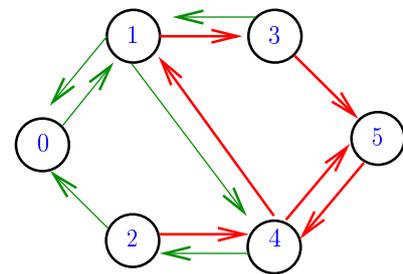


< > < > < > < > < > < >

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

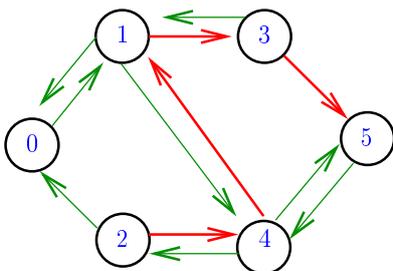


< > < > < > < > < > < >

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

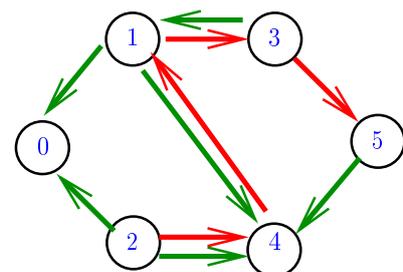


< > < > < > < > < > < >

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

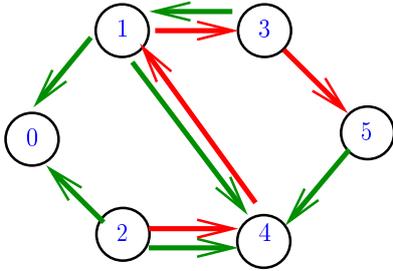


< > < > < > < > < > < >

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 0 a 2 é infinita

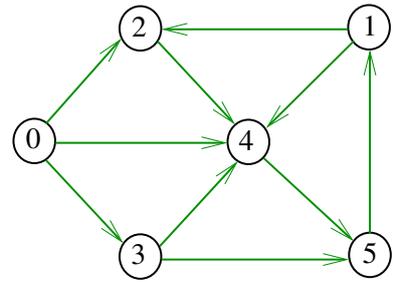


Calculando distâncias

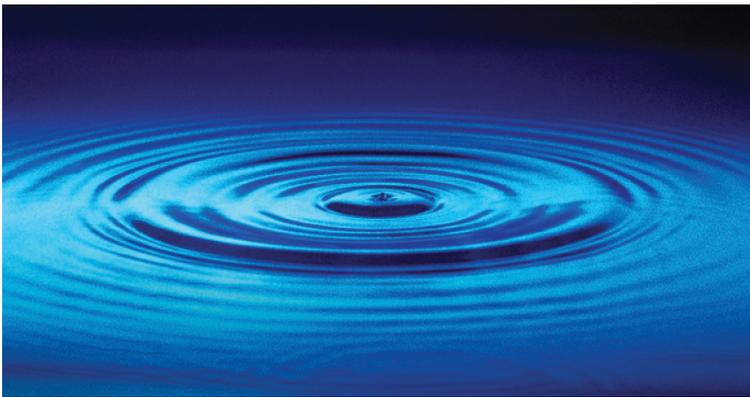
Problema: dada um rede de estradas e uma cidade **c**, determinar a distância de **c** a cada uma das demais cidades

Exemplo: para $c = 0$

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



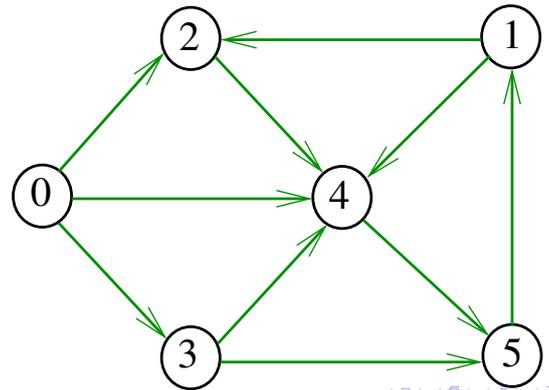
Calculando distâncias



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

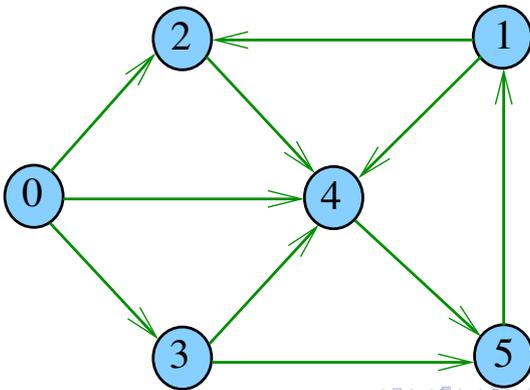
Simulação

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



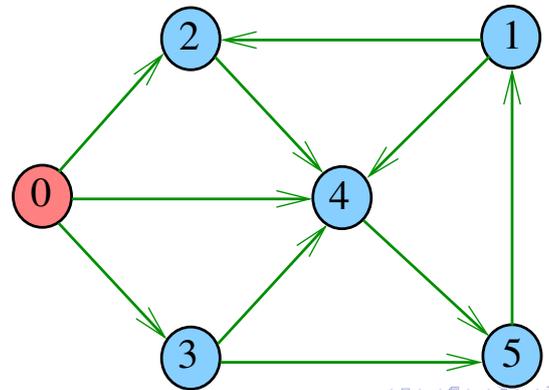
Simulação

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



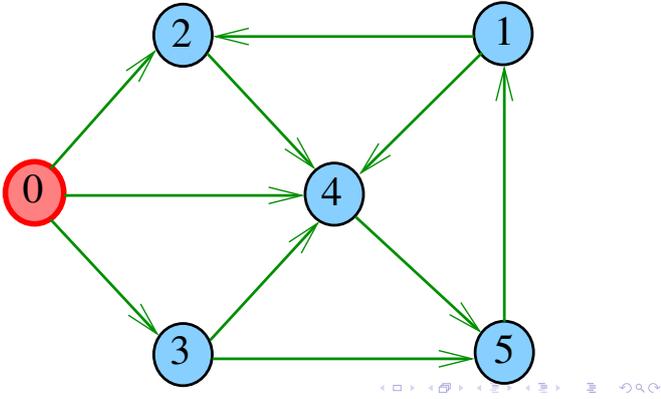
Simulação

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



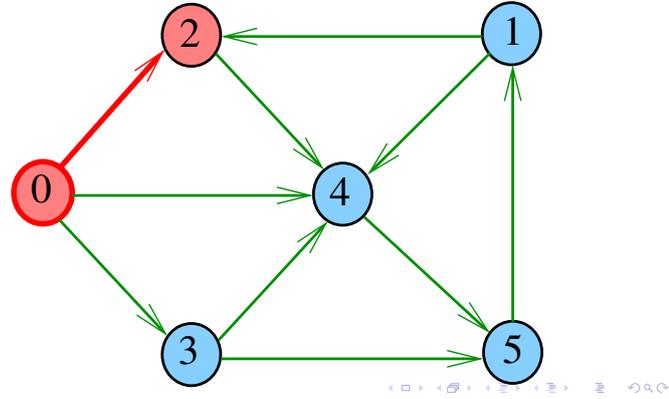
Simulação

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



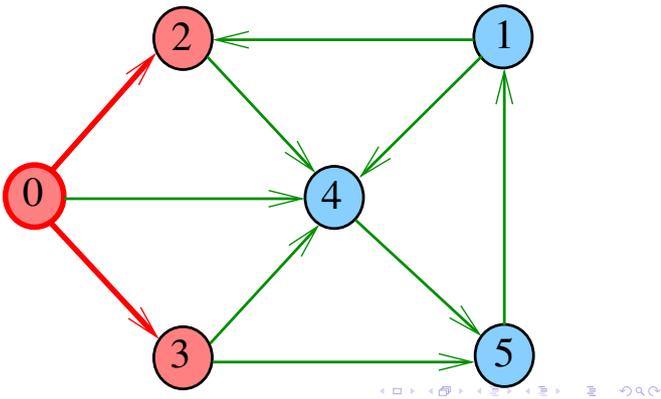
Simulação

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



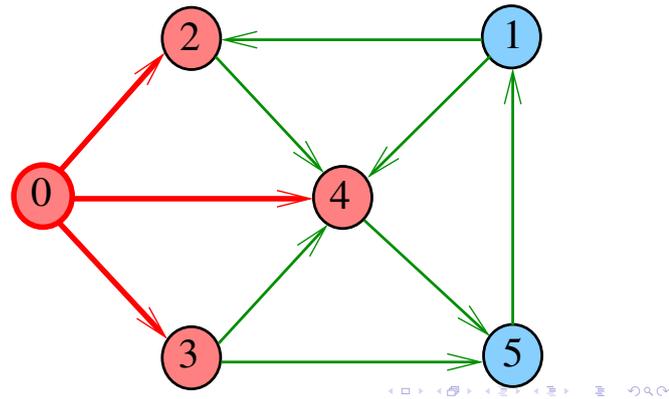
Simulação

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



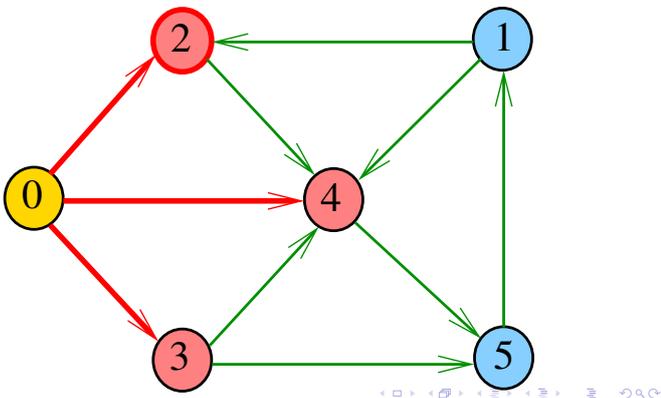
Simulação

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



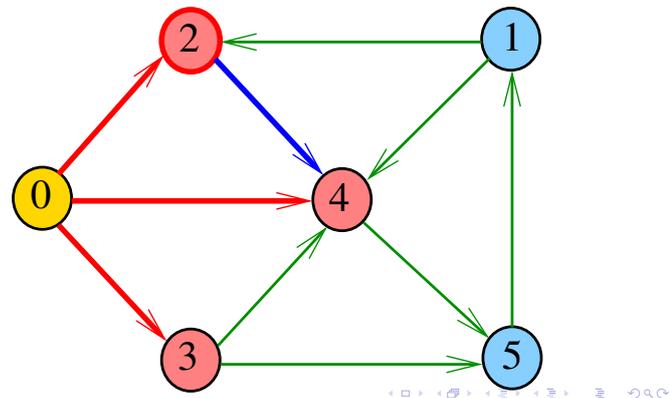
Simulação

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



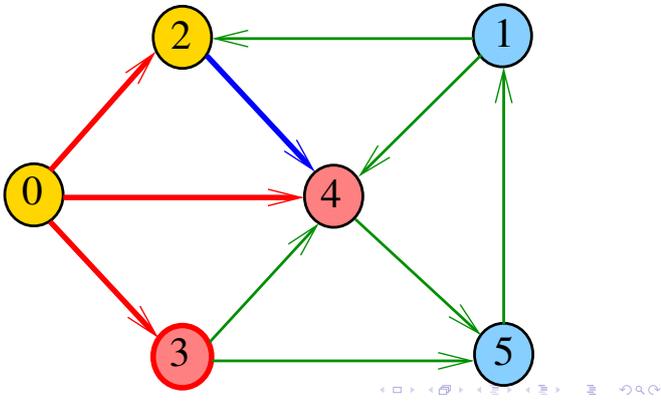
Simulação

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



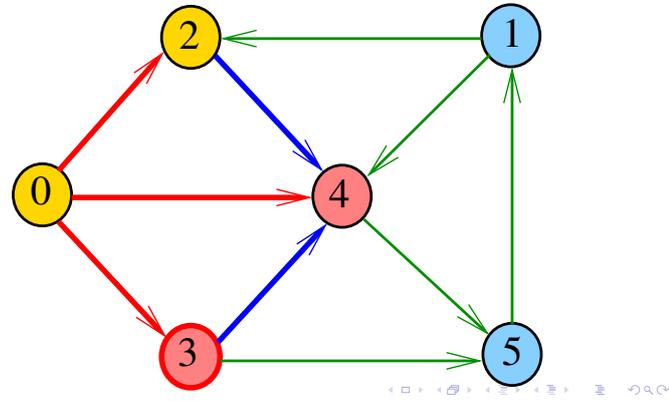
Simulação

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



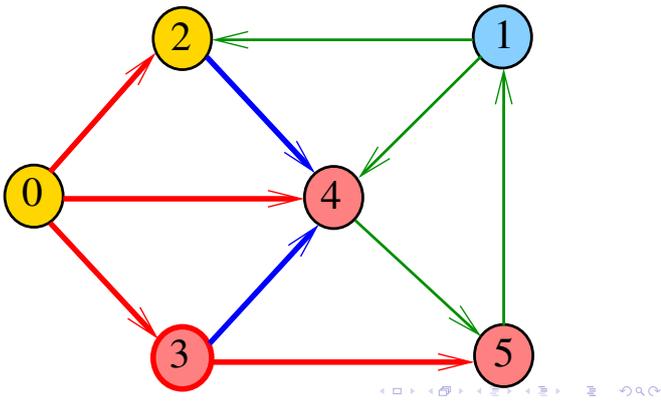
Simulação

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



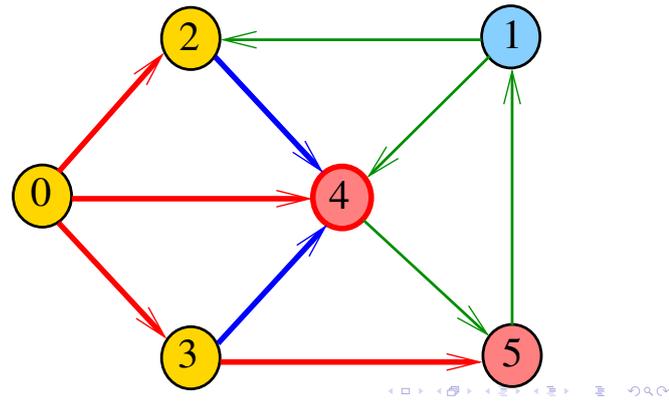
Simulação

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



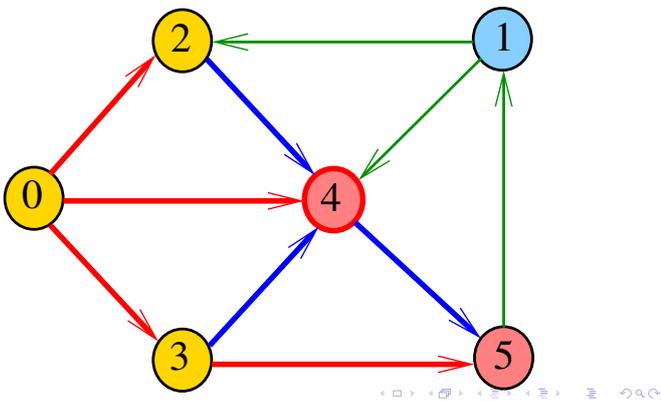
Simulação

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



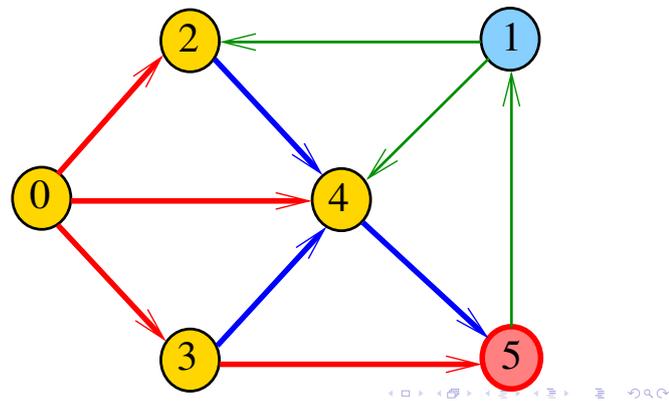
Simulação

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



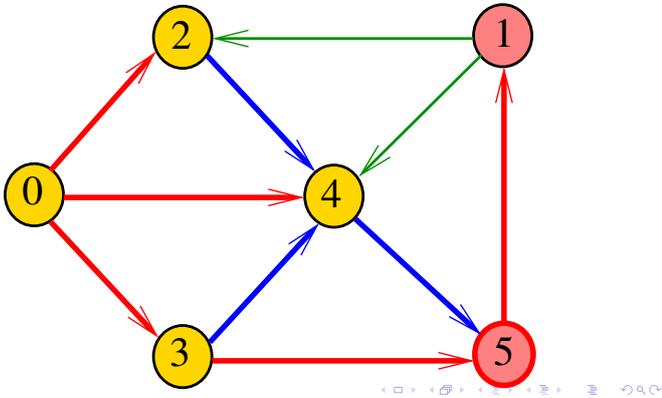
Simulação

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



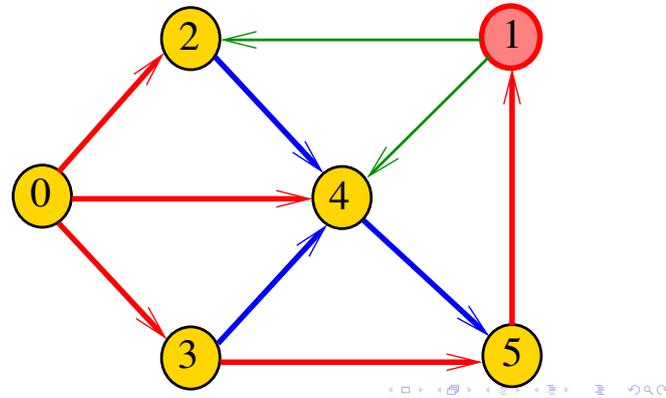
Simulação

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



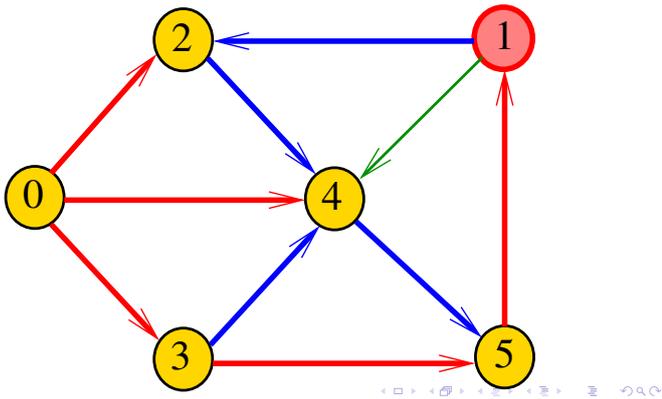
Simulação

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



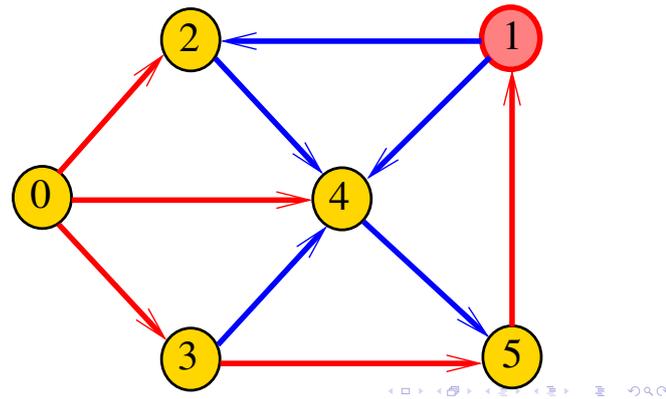
Simulação

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



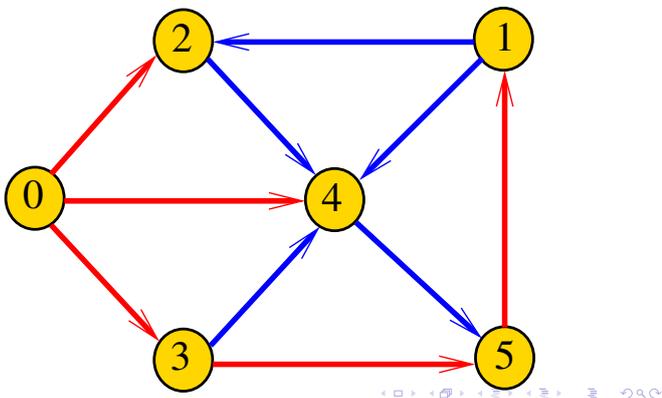
Simulação

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



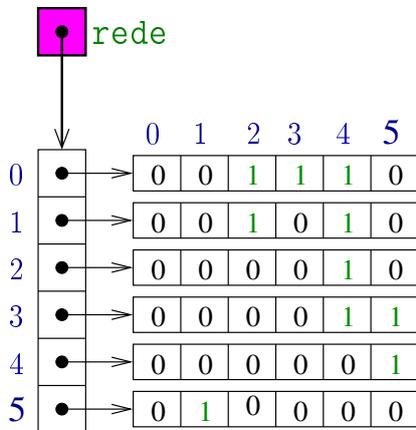
Representação da rede

As ligações entre as cidades são representadas por uma matriz rede.

$rede[i][j] = 1$ se existe estrada da cidade i para a cidade j

$rede[i][j] = 0$ em caso contrário

Representação da rede



distancias

A função `distancias` recebe um inteiro `n`, uma matriz `rede` representando uma rede de estradas entre `n` cidades e uma cidade `c` e devolve um vetor `d` que registra a distância da cidade `c` a cada uma das outras: `d[i]` é a distância de `c` a `i`.

```
int *
distancias (int n, int **rede, int c) {
    int *q; /* guarda a fila */
    int ini; /* q[ini] = 1o. */
    int fim; /* q[fim-1] = ultimo */
    int *d; /* d[i] = distancia de c a i */
    int j;
```

distancias

```
/* queueInit(n): inicialize a fila */
q = mallocSafe(n*sizeof(int));
ini = 0; fim = 0; /* fila vazia */
/* aloque vetor de distancias */
d = mallocSafe(n* sizeof(int));
/* inicialize o vetor de distancias */
for (j = 0; j < n; j++)
    d[j] = n; /* distancia n = infinito */
d[c] = 0;
/* queuePut(c): coloque c na fila */
q[fim++] = c;
```

distancias

```
while (ini != fim) { /*!queueEmpty()*/
    int i = q[ini++]; /* i = queueGet() */
    int di = d[i];
    for (j = 0; j < n; j++)
        if (rede[i][j] == 1 && d[j] > di+1){
            d[j] = di + 1;
            q[fim++] = j; /* queuePut(j) */
        }
    }
free(q); /* queueFree() */
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

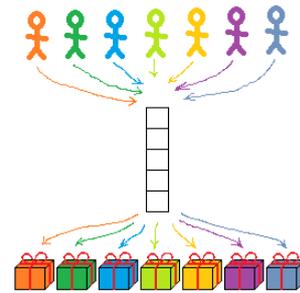
Se $d[i] == n$ para alguma cidade i , então

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

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

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

Interface para filas



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

S 4.6, 4.8

Interface item.h

```
/*
 * item.h
 */
typedef int Item;
```

Interface queue.h

```
/*
 * queue.h
 * INTERFACE: funcoes para manipular uma
 * fila
 */
void queueInit(int);
int queueEmpty();
void queuePut(Item);
Item queueGet();
void queueFree();
```

distancias

A função `distancias` recebe um inteiro n , uma matriz `rede` representando uma rede de estradas entre n cidades e uma cidade c e devolve um vetor d que registra a distância da cidade c a cada uma das outras: $d[i]$ é a distância de c a i .

```
int *
distancias (int n, int **rede, int c) {
    int *d; /* d[i] = distancia de c a i*/
    int j;
```

distancias

```
queueInit(n); /* inicialize a fila */

/* aloque vetor de distancias */
d = mallocSafe(n* sizeof(int));

/* inicialize o vetor de distancias */
for (j = 0; j < n; j++)
    d[j] = n; /* distancia n = infinito */
d[c] = 0;

queuePut(c); /* coloque c na fila */
```

