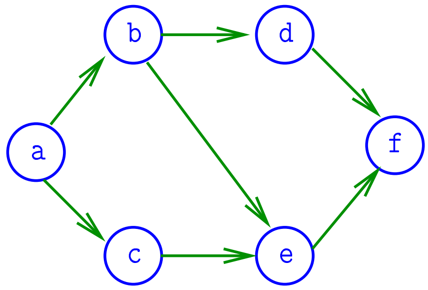
Melhores momentos

AULA 1

Digrafos

digrafo = de vértices e conjunto de arcos
arco = par ordenado de vértices

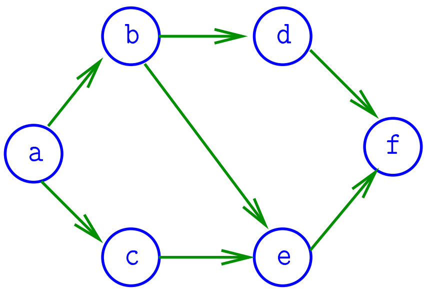
Exemplo: `v` e `w` são vértices e `v-w` é um arco



Especificação

Digrafos podem ser especificados através de sua **lista de arcos**

Exemplo:

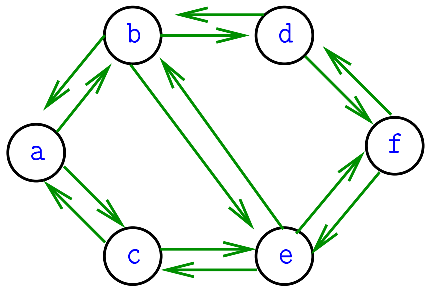



```
d-f
b-d
a-c
b-e
e-f
a-b
```

Grafos

grafo = digrafo **simétrico**
aresta = par de arcos anti-paralelos

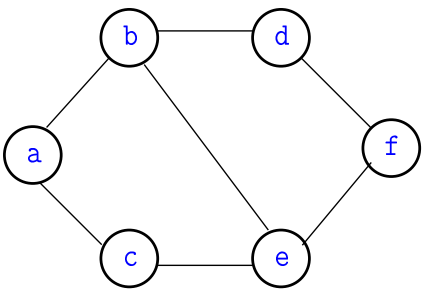
Exemplo: `b-a` e `a-b` formam uma aresta



Grafos

Um **grafo** é um digrafo **simétrico**

Exemplo: representação usual



Estrutura de dados

Vértices são representados por objetos do tipo `Vertex`.
Arcos sao representados por por objetos do tipo `Arc`

```
#define Vertex int

typedef struct {
    Vertex v;
    Vertex w;
} Arc;
```

Grafos no computador

Usaremos duas representações clássicas:

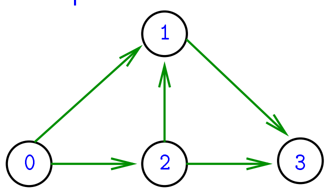
- **matriz de adjacência** (**agora**)
- vetor de listas de adjacência (**próximas aulas**)

Matriz de adjacência de digrafo

Matriz de adjacência de um digrafo tem linhas e colunas indexadas por vértices:

`adj[v][w]` = 1 se `v-w` é um arco
`adj[v][w]` = 0 em caso contrário

Exemplo:



	0	1	2	3
0	0	1	1	0
1	0	0	0	1
2	0	1	0	1
3	0	0	0	0

Consumo de espaço: $\Theta(V^2)$ fácil de implementar

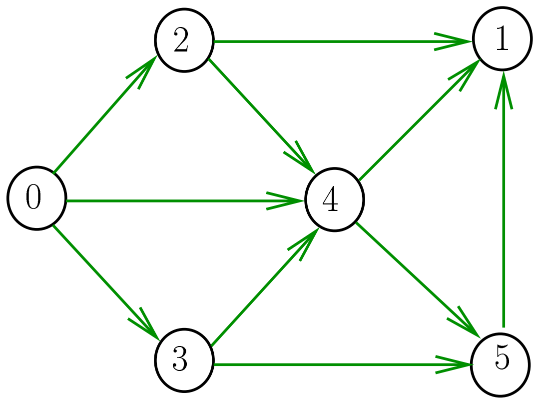
Estrutura digraph

V = número de vértices
A = número de arcos
adj = ponteiro para a matriz de adjacência

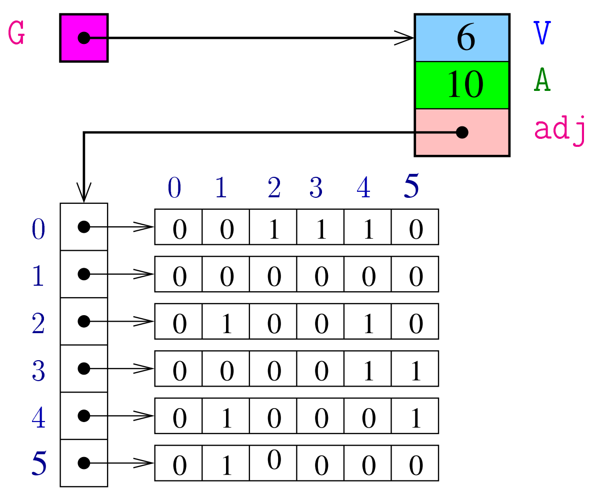
```
struct digraph {
    int V;
    int A;
    int **adj;
};

typedef struct digraph *Digraph;
```

Digrafo

Digraph G



Estruturas de dados



MATRIXint

Aloca uma matriz com linhas `0..r-1` e colunas `0..c-1`, cada elemento da matriz recebe valor `val`

```
int **MATRIXint (int r, int c, int val) {
    Vertex i, j;
    int **m = malloc(r * sizeof(int *));
    for (i = 0; i < r; i++)
        m[i] = malloc(c * sizeof(int));
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            m[i][j] = val;
    return m;
}
```

Consumo de tempo

linha	número de execuções da linha	
1	$= 1$	$= \Theta(1)$
2	$= r + 1$	$= \Theta(r)$
3	$= r$	$= \Theta(r)$
4	$= r + 1$	$= \Theta(r)$
5	$= r \times (c + 1)$	$= \Theta(r\,c)$
6	$= r \times c$	$= \Theta(r\,c)$

total $\Theta(1) + 3\,\Theta(r) + 2\,\Theta(r\,c)$
$= \Theta(r\,c)$

Conclusão

Supondo que o consumo de tempo da função `malloc` é constante

> O consumo de tempo da função `MATRIXint` é $\Theta(r\,c)$.

DIGRAPHinit

Devolve (o endereço de) um novo digrafo com vértices `0,..,V-1` e nenhum arco.

```
Digraph DIGRAPHinit (int V) {
    Digraph G = malloc(sizeof *G);
    G->V = V;
    G->A = 0;
    G->adj = MATRIXint(V,V,0);
    return G;
}
```

AULA 2

Funções básicas (continuação)

S 17.3

DIGRAPHinsertA

Insere um arco `v-w` no digrafo G.
Se `v == w` ou o digrafo já tem arco `v-w`, não faz nada

```
void
DIGRAPHinsertA(Digraph G,Vertex v,Vertex w)
```

DIGRAPHinsertA

Insere um arco v-w no digrafo G.
Se v == w ou o digrafo já tem arco v-w, não faz nada

```
void
DIGRAPHinsertA(Digraph G,Vertex v,Vertex w)
{
   if (v != w && G->adj[v][w] == 0) {
        G->adj[v][w] = 1;
        G->A++;
   }
}
```

DIGRAPHremoveA

Remove do digrafo G o arco v-w
Se não existe tal arco, a função nada faz.

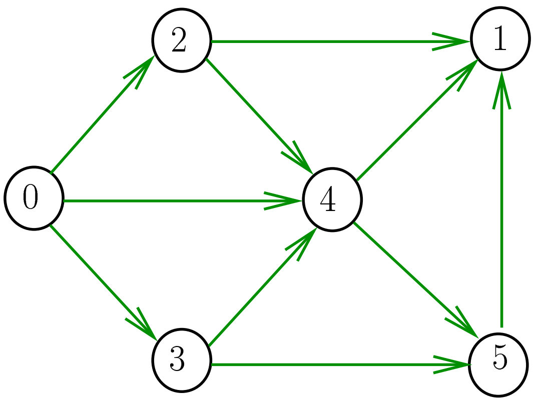
```
void
DIGRAPHremoveA(Digraph G,Vertex v,Vertex w)
```

DIGRAPHremoveA

Remove do digrafo G o arco v-w
Se não existe tal arco, a função nada faz.

```
void
DIGRAPHremoveA(Digraph G,Vertex v,Vertex w)
{
   if (G->adj[v][w] == 1) {
        G->adj[v][w] = 0;
        G->A--;
   }
}
```

DIGRAPHshow




```
0:   2   3   4
1:
2:   1   4
3:   4   5
4:   1   5
5:   1
```

DIGRAPHshow

Para cada vértice v de G, imprime, em uma linha, os vértices adjacentes a v

```
void DIGRAPHshow (Digraph G) {
```

DIGRAPHshow

Para cada vértice v de G, imprime, em uma linha, os vértices adjacentes a v

```
void DIGRAPHshow (Digraph G) {
     Vertex v, w;
     for (v = 0; v < G->V; v++) {
          printf("%2d:", v);
          for (w = 0; w < G->V; w++)
               if (G->adj[v][w] == 1)
                    printf("%2d", w);
          printf("\n");
     }
}
```

Consumo de tempo

linha	número de execuções da linha	
1	$= V + 1$	$= \Theta(V)$
2	$= V$	$= \Theta(V)$
3	$= V \times (V + 1)$	$= \Theta(V^2)$
4	$= V \times V$	$= \Theta(V^2)$
5	$\leq V \times V$	$= O(V^2)$
6	$= V$	$= \Theta(V)$
total	$3\,\Theta(V) + O(V^2) + 3\,\Theta(V^2)$ $= \Theta(V^2)$	

Conclusão

O consumo de tempo da função DIGRAPHShow é $\Theta(V^2)$.

Funções básicas para grafos

Funções básicas para grafos

```
#define GRAPHinit DIGRAPHinit
#define GRAPHshow DIGRAPHshow
```

Função que insere uma aresta v-w no grafo G

```
void
GRAPHinsertE (Graph G, Vertex v, Vertex w)
```

Funções básicas para grafos

```
#define GRAPHinit DIGRAPHinit
#define GRAPHshow DIGRAPHshow
```

Função que insere uma aresta v-w no grafo G

```
void
GRAPHinsertE (Graph G, Vertex v, Vertex w)
{
   DIGRAPHinsertA(G, v, w);
   DIGRAPHinsertA(G, w, v);
}
```

Exercício. Escrever a função GRAPHremoveE

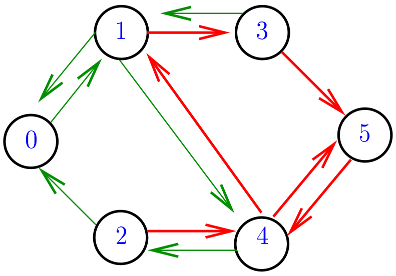
Caminhos em digrafos

S 17.1

Caminhos

Um **caminho** num digrafo é qualquer seqüência da forma $v_0 - v_1 - v_2 - \ldots - v_{k-1} - v_p$, onde v_{k-1}-v_k é um arco para $k = 1, \ldots, p$.

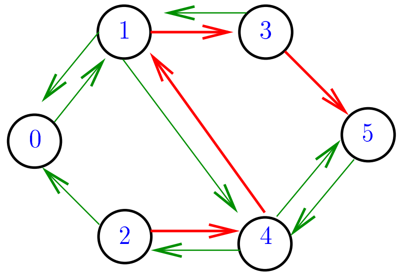
Exemplo: 2-4-1-3-5-4-5 é um caminho com **origem** 2 é **término** 5



Caminhos simples

Um caminho é **simples** se não tem vértices repetidos

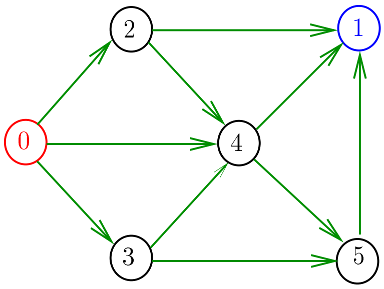
Exemplo: 2-4-1-3-5 é um caminho simples de 2 a 5



Procurando um caminho

Problema: dados um digrafo `G` e dois vértices `s` e `t` decidir se existe um caminho de `s` a `t`

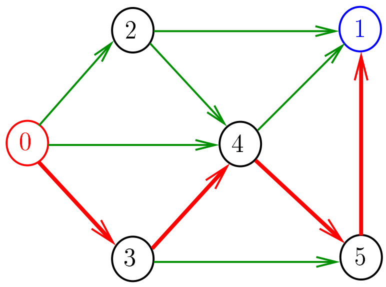
Exemplo: para `s` $= 0$ e `t` $= 1$ a resposta é SIM



Procurando um caminho

Problema: dados um digrafo `G` e dois vértices `s` e `t` decidir se existe um caminho de `s` a `t`

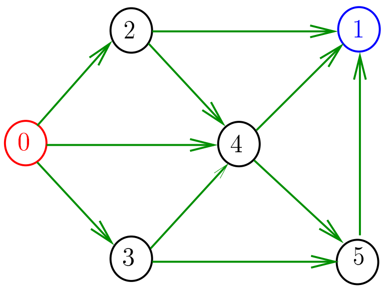
Exemplo: para `s` $= 0$ e `t` $= 1$ a resposta é SIM



Procurando um caminho

Problema: dados um digrafo `G` e dois vértices `s` e `t` decidir se existe um caminho de `s` a `t`

Exemplo: para `s` $= 5$ e `t` $= 4$ a resposta é NÃO



DIGRAPHpath

Recebe um digrafo `G` e vértices `s` e `t` e devolve 1 se existe um caminho de `s` a `t` ou devolve 0 em caso contrário
Supõe que o digrafo tem no máximo `maxV` vértices.

```
int DIGRAPHpath (Digraph G, Vertex s, Vertex t)
```

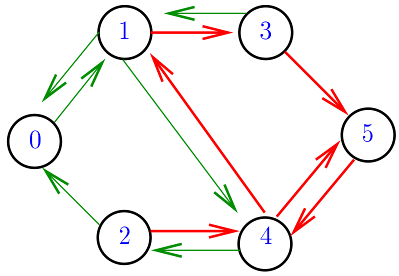
Caminhos em digrafos

S 17.1

Caminhos

Um **caminho** num digrafo é qualquer seqüência da forma v_0–v_1–v_2–…–v_{k-1}–v_p, onde v_{k-1}-v_k é um arco para $k = 1, \ldots, p$.

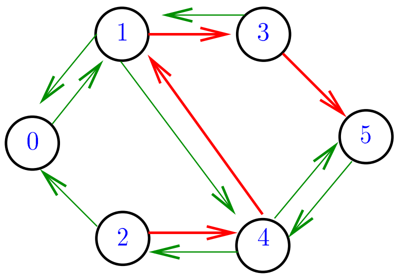
Exemplo: 2-4-1-3-5-4-5 é um caminho com **origem** 2 é **término** 5



Caminhos simples

Um caminho é **simples** se não tem vértices repetidos

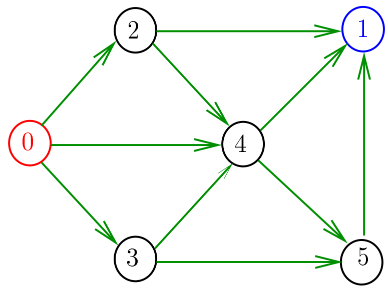
Exemplo: 2-4-1-3-5 é um caminho simples de 2 a 5



Procurando um caminho

Problema: dados um digrafo G e dois vértices s e t decidir se existe um caminho de s a t

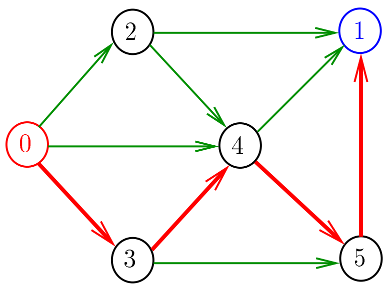
Exemplo: para s = 0 e t = 1 a resposta é SIM



Procurando um caminho

Problema: dados um digrafo G e dois vértices s e t decidir se existe um caminho de s a t

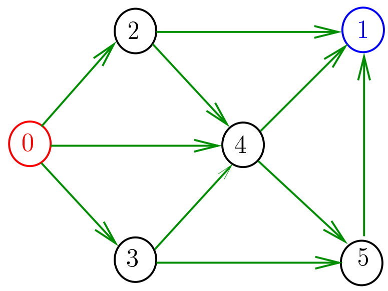
Exemplo: para s = 0 e t = 1 a resposta é SIM



Procurando um caminho

Problema: dados um digrafo G e dois vértices s e t decidir se existe um caminho de s a t

Exemplo: para s = 5 e t = 4 a resposta é NÃO

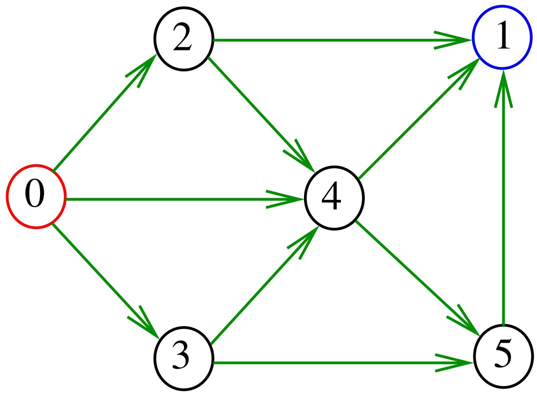


DIGRAPHpath

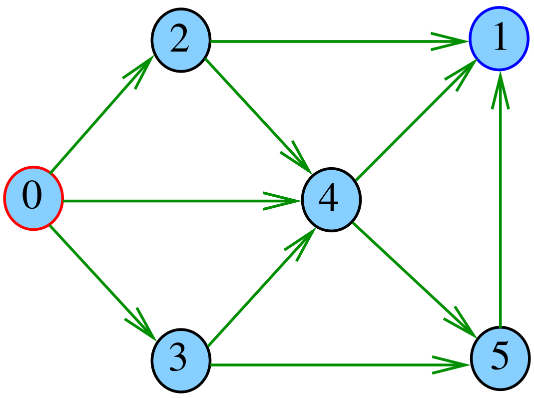
Recebe um digrafo G e vértices s e t e devolve **1** se existe um caminho de s a t ou devolve **0** em caso contrário
Supõe que o digrafo tem no máximo maxV vértices.

int `DIGRAPHpath (Digraph G, Vertex s, Vertex t)`

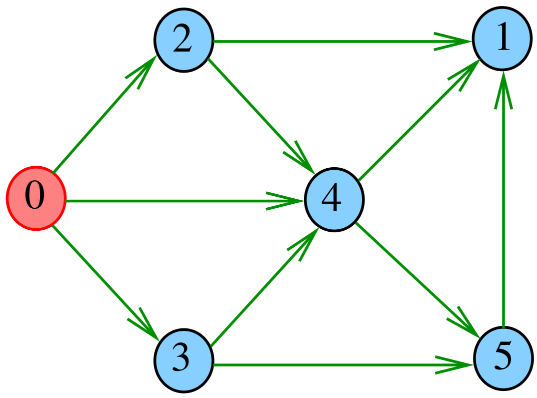
DIGRAPHpath(G,0,1)



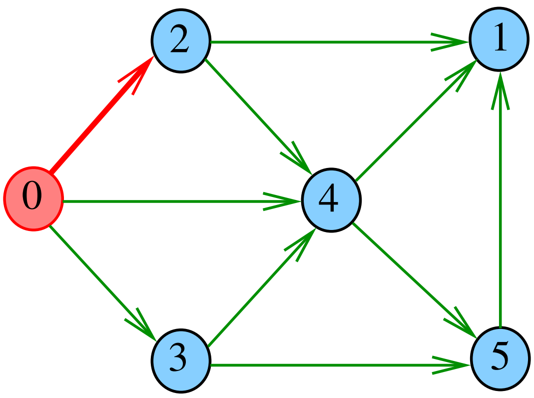
DIGRAPHpath(G,0,1)



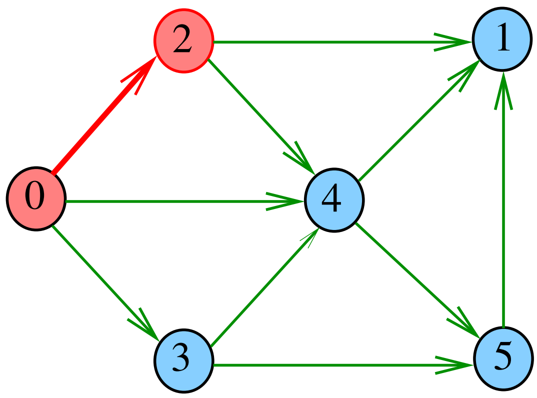
pathR(G,0)



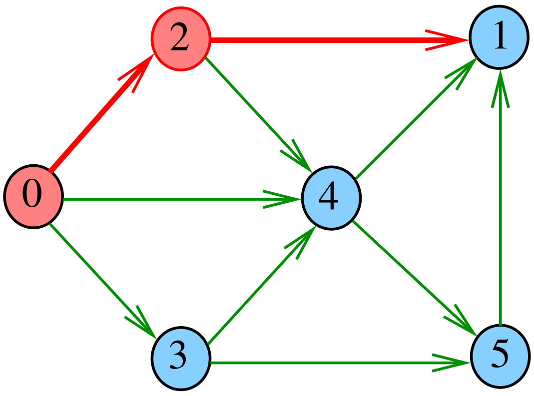
pathR(G,0)



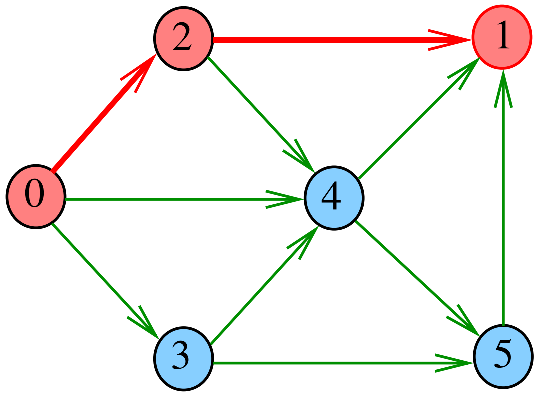
pathR(G,2)



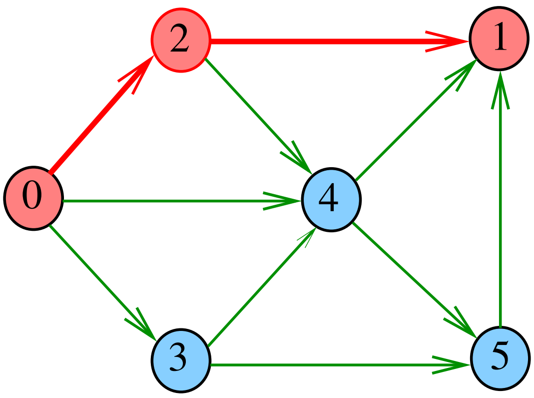
pathR(G,2)



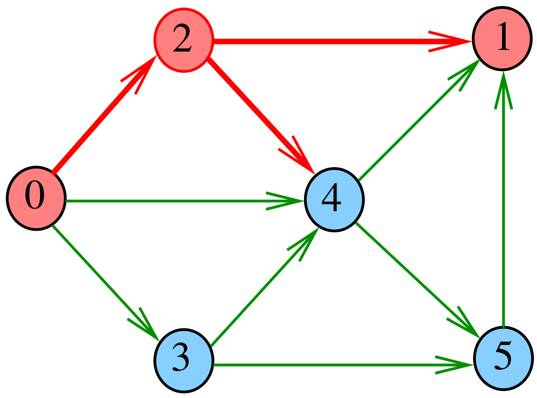
pathR(G,1)



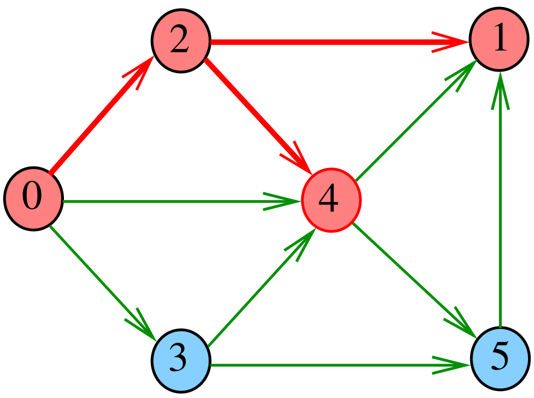
pathR(G,2)



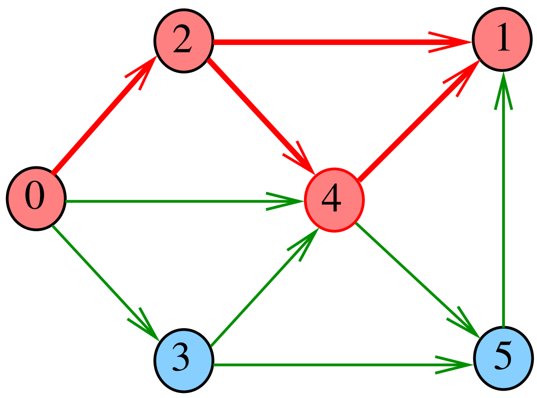
pathR(G,2)



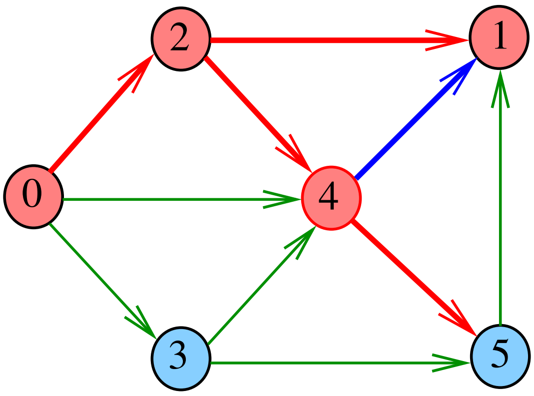
pathR(G,4)



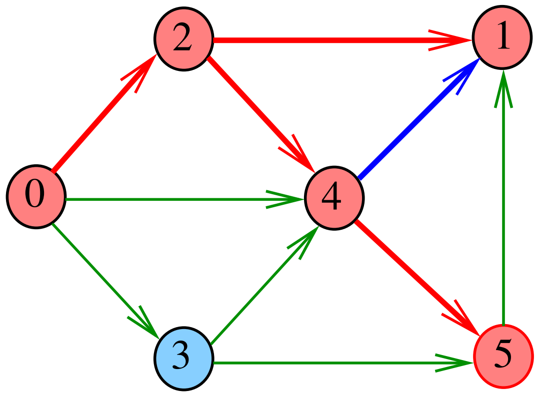
pathR(G,4)



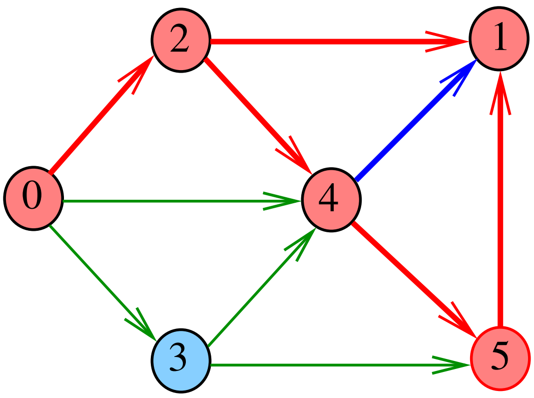
pathR(G,4)



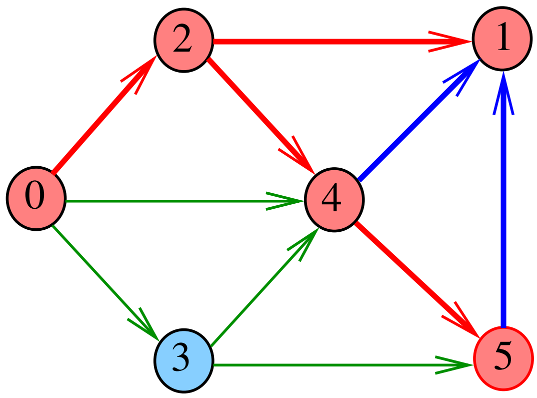
pathR(G,5)



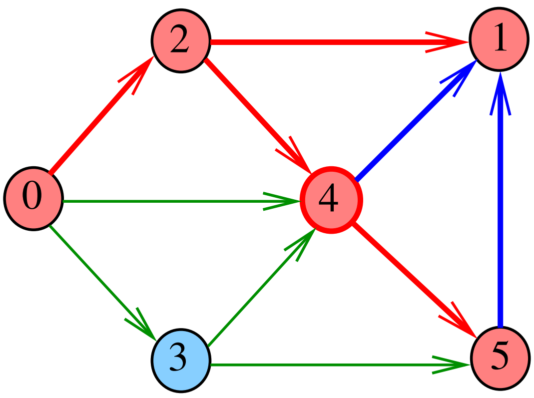
pathR(G,5)



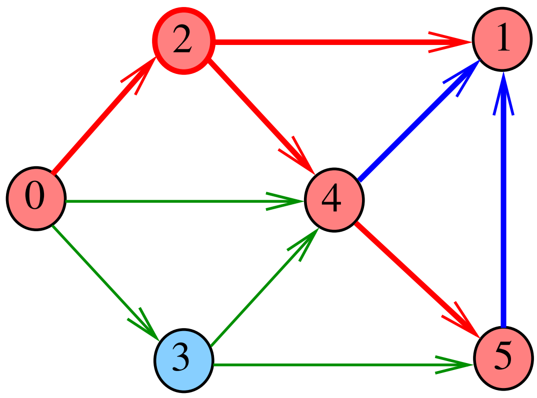
pathR(G,5)



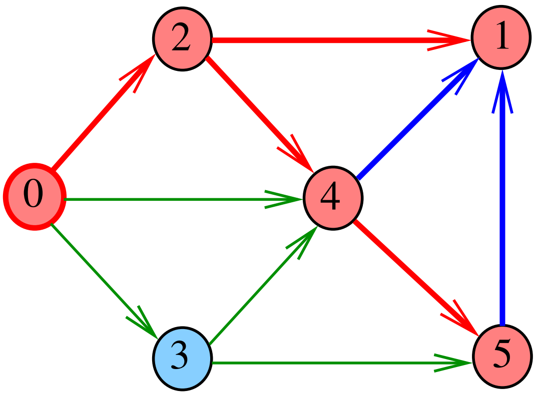
pathR(G,4)



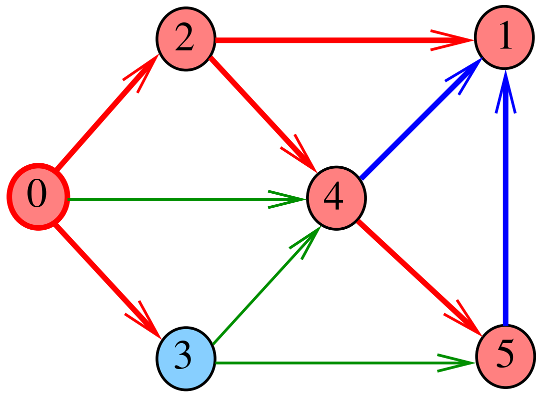
pathR(G,2)



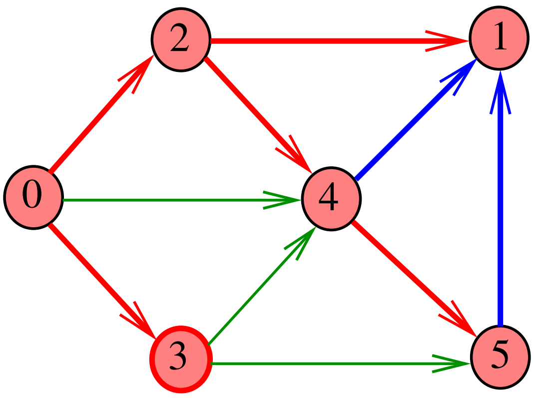
pathR(G,0)



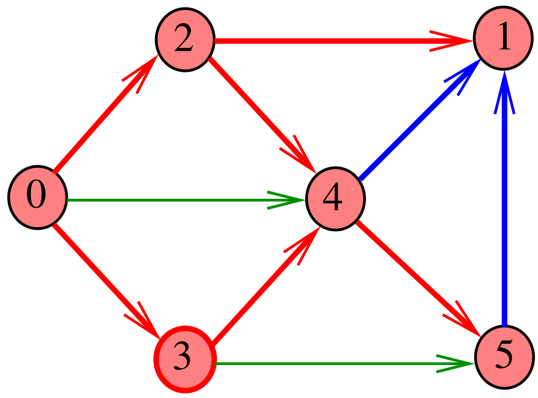
pathR(G,0)



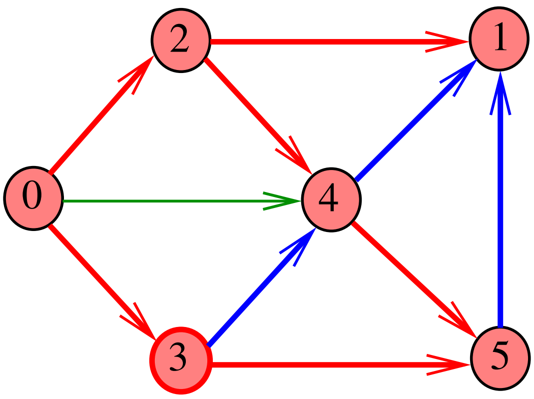
pathR(G,3)



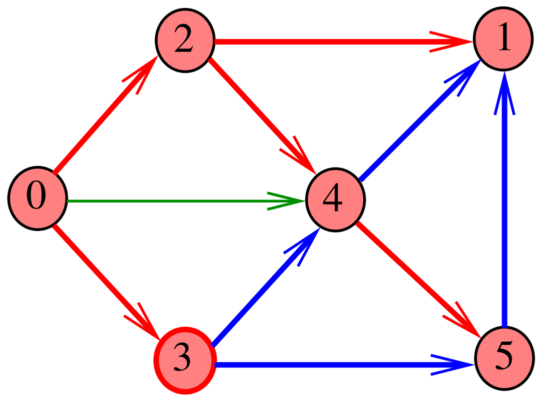
pathR(G,3)



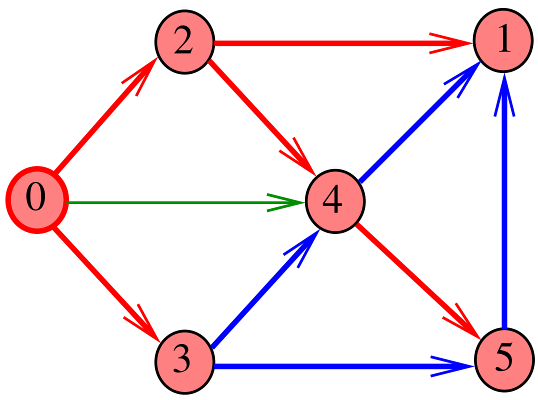
pathR(G,3)



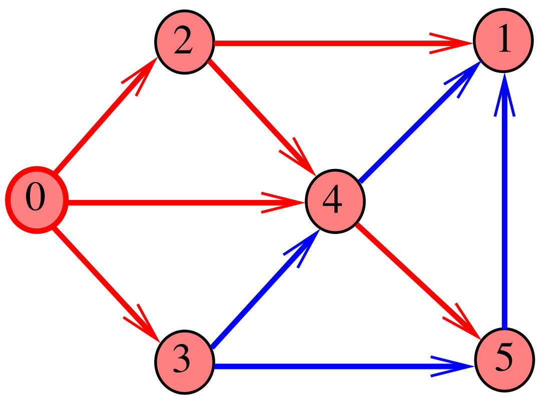
pathR(G,3)



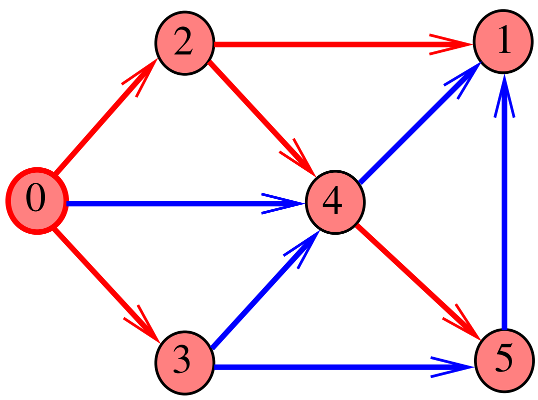
pathR(G,0)



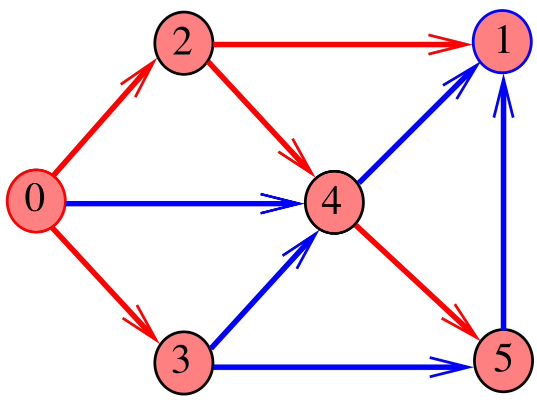
pathR(G,0)



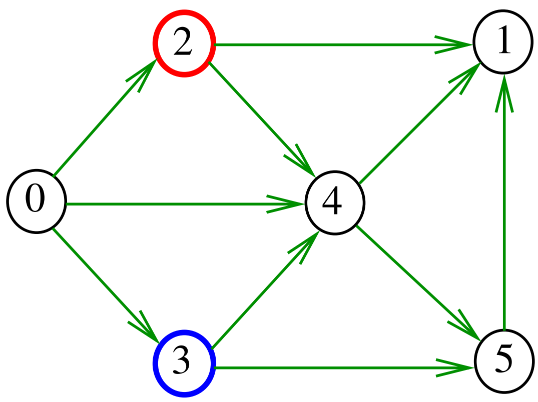
pathR(G,0)



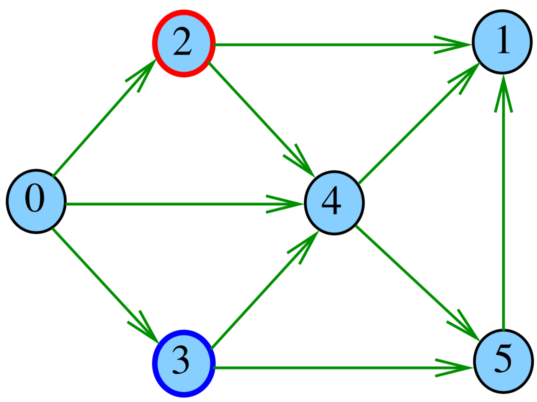
DIGRAPHpath(G,0,1)



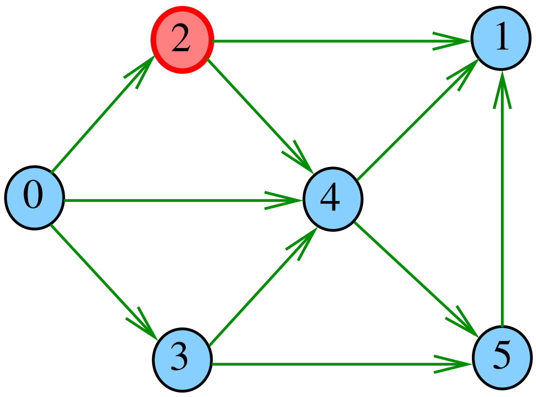
DIGRAPHpath(G,2,3)



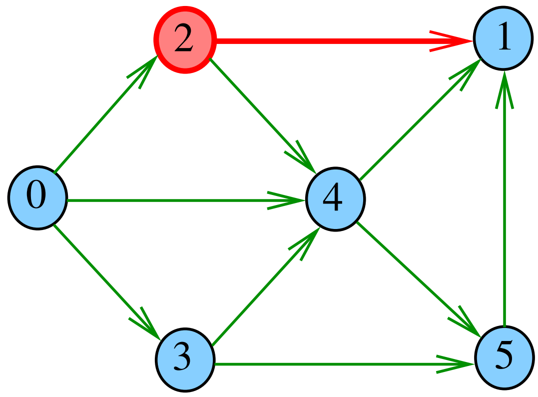
DIGRAPHpath(G,2,3)



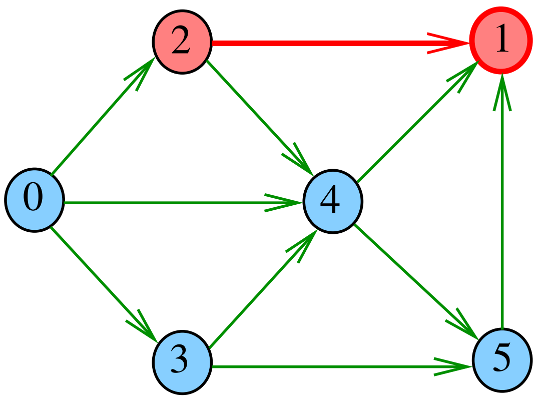
pathR(G,2)



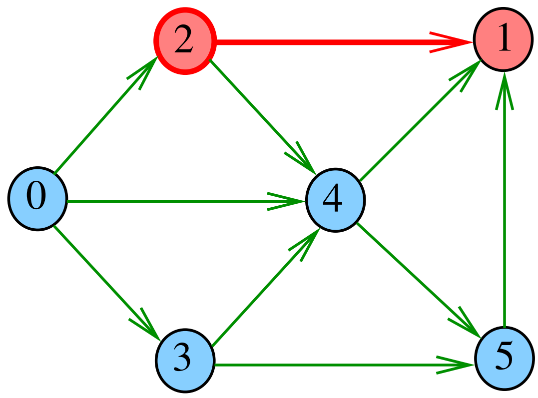
pathR(G,2)



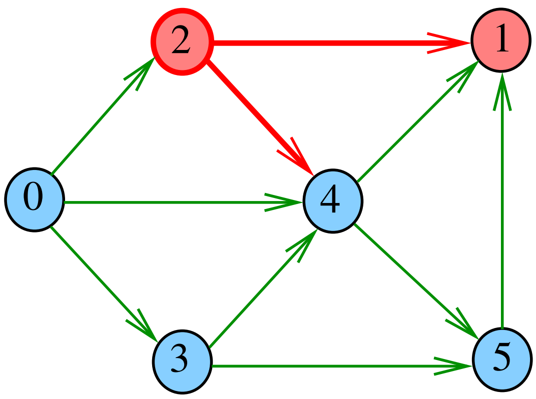
pathR(G,1)



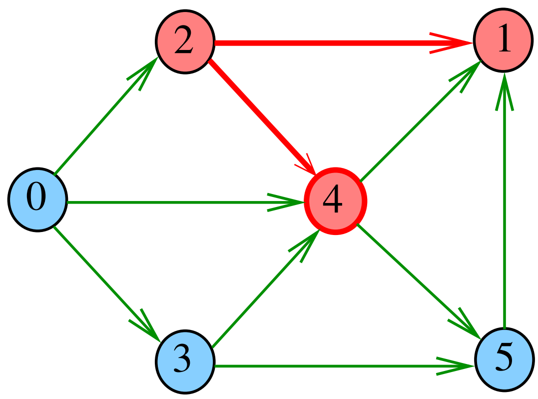
pathR(G,2)



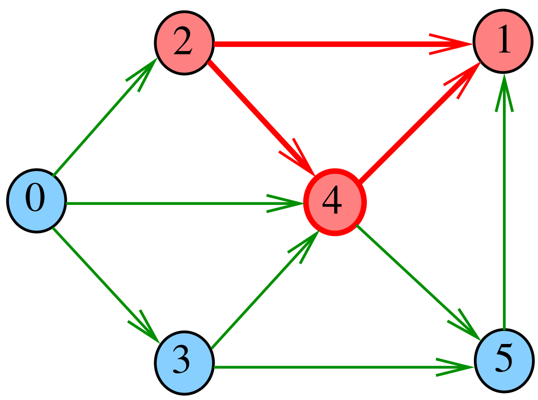
pathR(G,2)



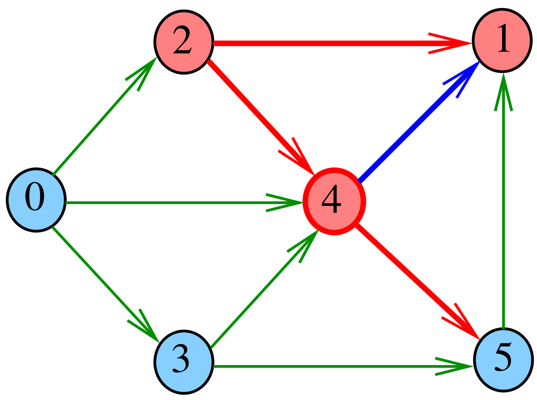
pathR(G,4)



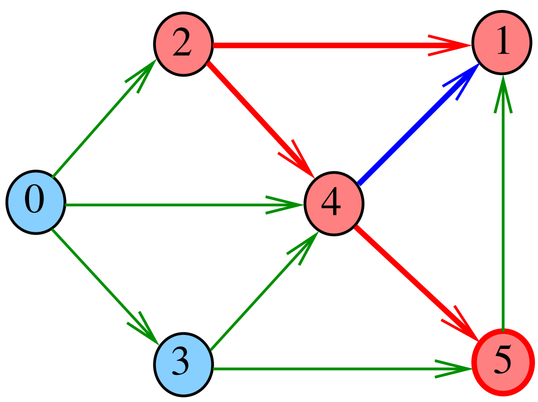
pathR(G,4)



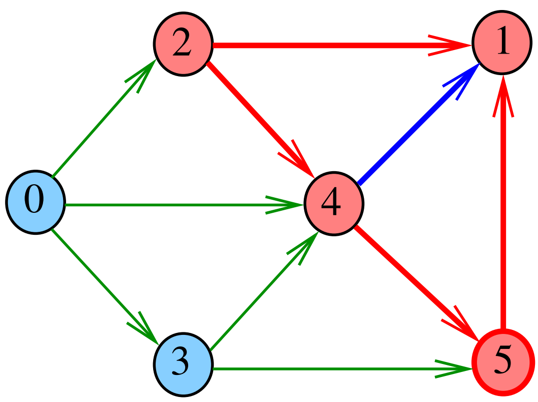
pathR(G,4)



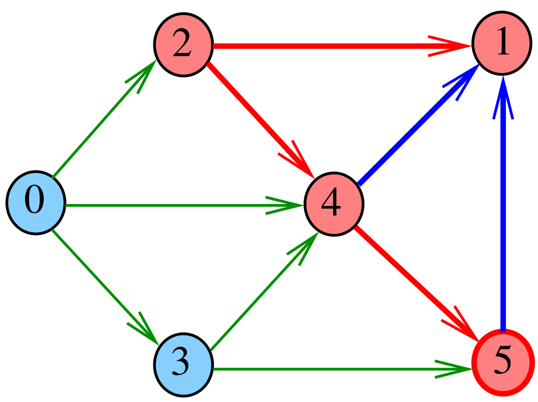
pathR(G,5)



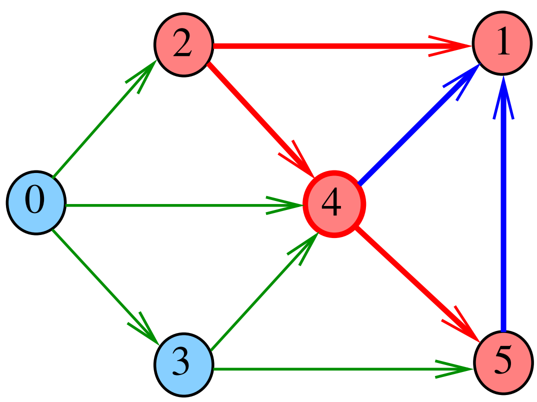
pathR(G,5)



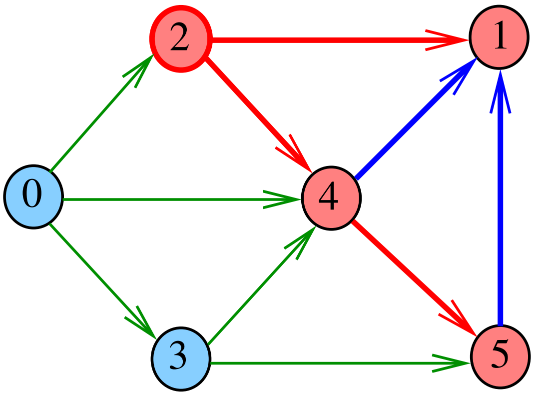
pathR(G,5)



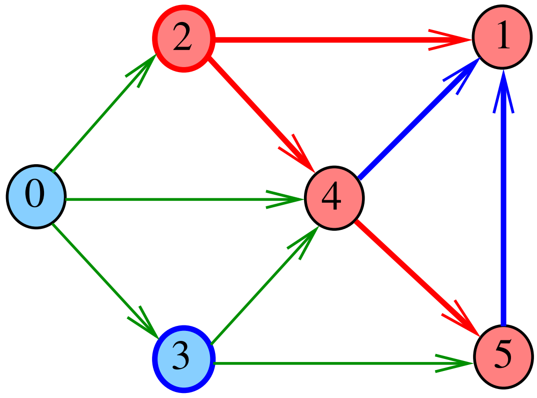
pathR(G,4)



pathR(G,2)



DIGRAPHpath(G,2,3)



DIGRAPHpath

```
static int lbl[maxV];
int DIGRAPHpath (Digraph G, Vertex s, Vertex t)
{
    Vertex v;
    for (v = 0; v < G->V; v++)
        lbl[v] = −1;
    pathR(G,s);
    if (lbl[t] == −1) return 0;
    else return 1;
}
```

pathR

Visita todos os vértices que podem ser atingidos a partir de v

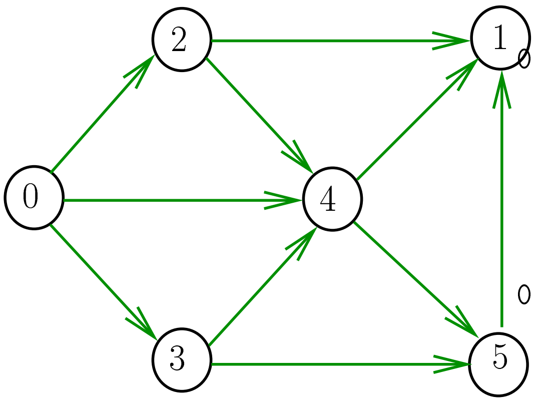
```
void pathR (Digraph G, Vertex v)
```

pathR

Visita todos os vértices que podem ser atingidos a partir de v

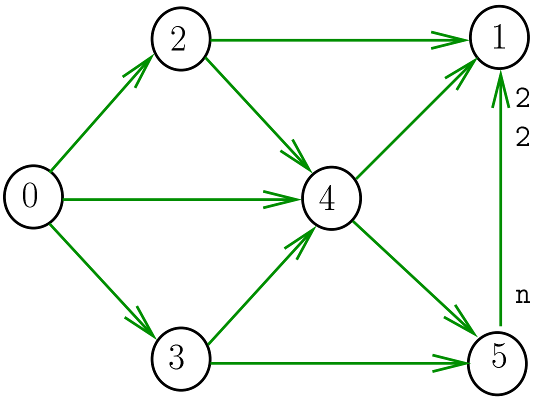
```
void pathR (Digraph G, Vertex v)
{
    Vertex w;
    lbl[v] = 0;
    for (w = 0; w < G->V; w++)
        if (G->adj[v][w] == 1)
            if (lbl[w] == −1)
                pathR(G, w);
}
```

DIGRAPHpath(G,0,1)




```
0-2 pathR(G,2)
  2-1 pathR(G,1)
  2-4 pathR(G,4)
    4-1
    4-5 pathR(G,5
      5-1
0-3 pathR(G,3)
  3-4
  3-5
0-4
existe caminho
```

DIGRAPHpath(G,2,3)



```
2-1 pathR(G,1)
2-4 pathR(G,4)
  4-1
  4-5 pathR(G,5)
    5-1
nao existe caminh
```

Consumo de tempo

Qual é o consumo de tempo da função DIGRAPHpath?

Consumo de tempo

Qual é o consumo de tempo da função DIGRAPHpath?

linha	número de execuções da linha	
1	$= V + 1$	$= \Theta(V)$
2	$= V$	$= \Theta(V)$
3	$= 1$	$=$????
4	$= 1$	$= \Theta(1)$
5	$= 1$	$= \Theta(1)$

total $= 2\,\Theta(1) + 2\,\Theta(V) +$???

$= \Theta(V) +$????

Conclusão

O consumo de tempo da função DIGRAPHpath é $\Theta(V)$ mais o consumo de tempo da função PathR.

Consumo de tempo

Qual é o consumo de tempo da função PathR?

Consumo de tempo

Qual é o consumo de tempo da função PathR?

linha	número de execuções da linha	
1	$\leq V$	$= O(V)$
2	$\leq V \times (V + 1)$	$= O(V^2)$
3	$\leq V \times V$	$= O(V^2)$
4	$\leq V \times V$	$= O(V^2)$
5	$\leq V - 1$	$= O(V)$

total $= 2\,O(V) + 3\,O(V^2)$

$= O(V^2)$

Conclusão

O consumo de tempo da função `PathR` para matriz de adjacência é $O(V^2)$.

O consumo de tempo da função `DIGRAPHpath` para matriz de adjacência é $O(V^2)$.