Melhores momentos

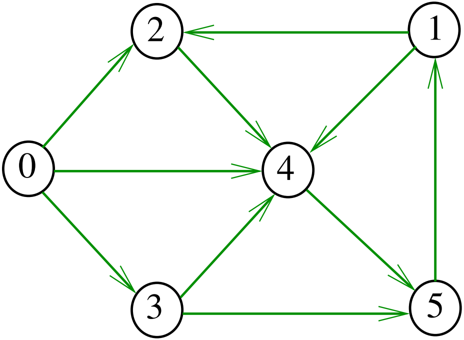
AULA 12

Calculando distâncias

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

Exemplo: para s = 0

v	0	1	2	3	4	5
dist[v]	0	3	1	1	1	2



Busca em largura

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

O algoritmo

visita s,

depois visita vértices à distância 1 de s,

depois visita vértices à distância 2 de s,

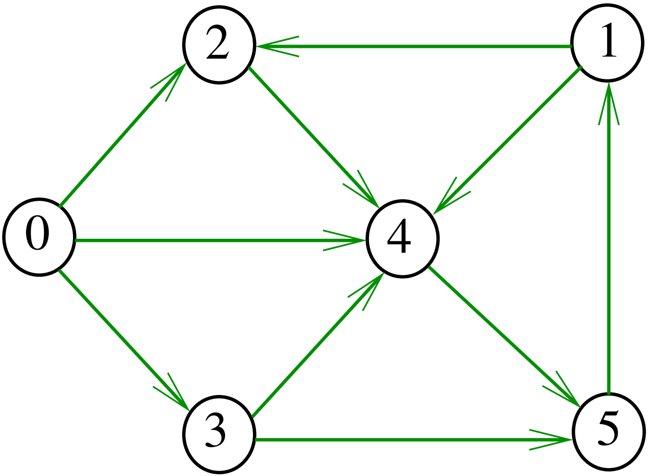
depois visita vértices à distância 3 de s,

e assim por diante

Simulação

i	0	1	2	3	4	5
q[i]						

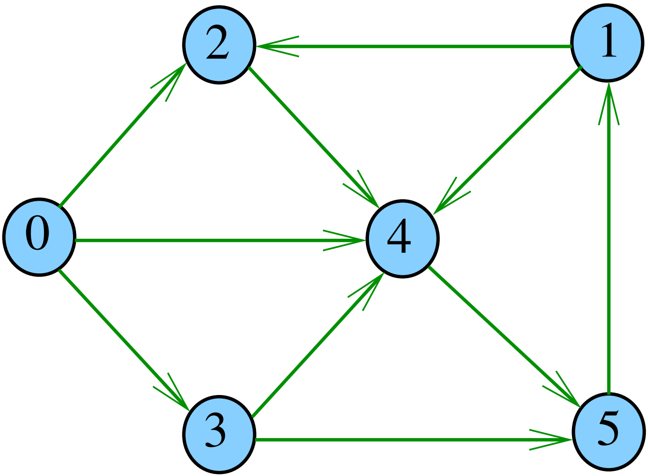
v	0	1	2	3	4	5
dist[v]						



Simulação

i	0	1	2	3	4	5
q[i]						

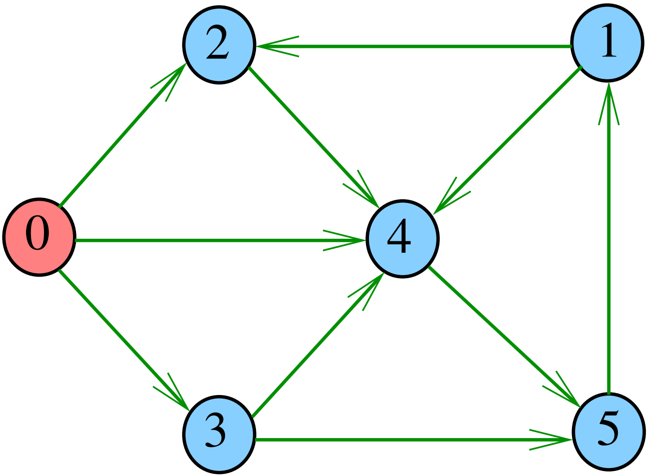
v	0	1	2	3	4	5
dist[v]	6	6	6	6	6	6



Simulação

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

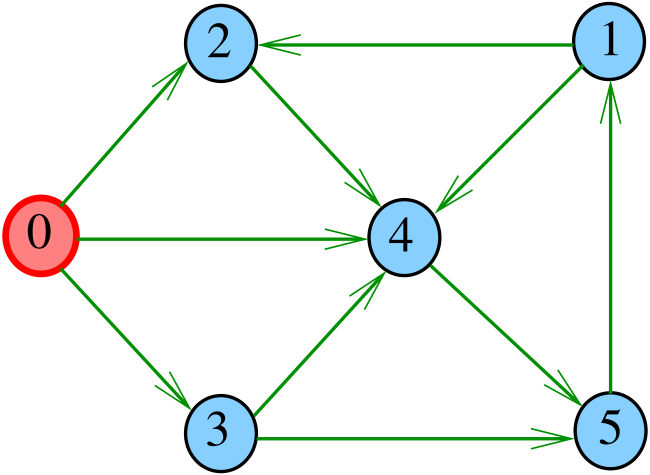
v	0	1	2	3	4	5
dist[v]	6	6	6	6	6	6



Simulação

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

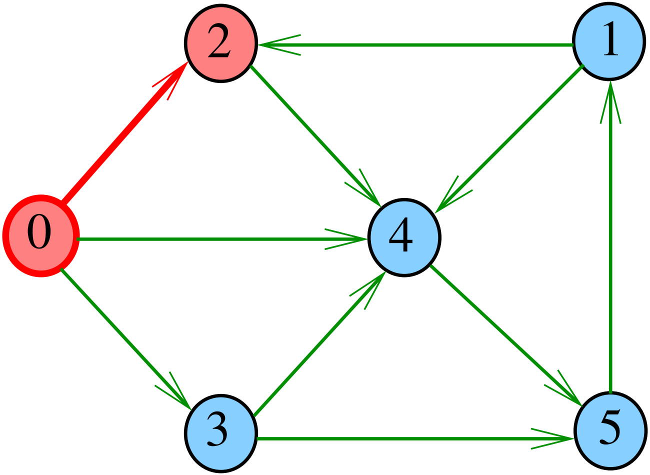
v	0	1	2	3	4	5
dist[v]	**0**	6	6	6	6	6



Simulação

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

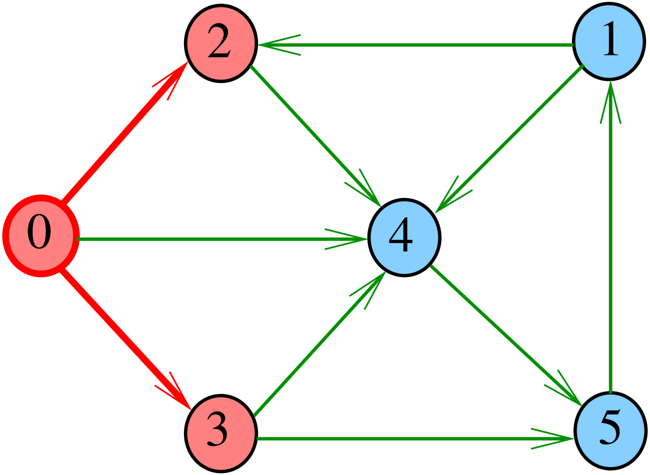
v	0	1	2	3	4	5
dist[v]	0	6	**1**	6	6	6



Simulação

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

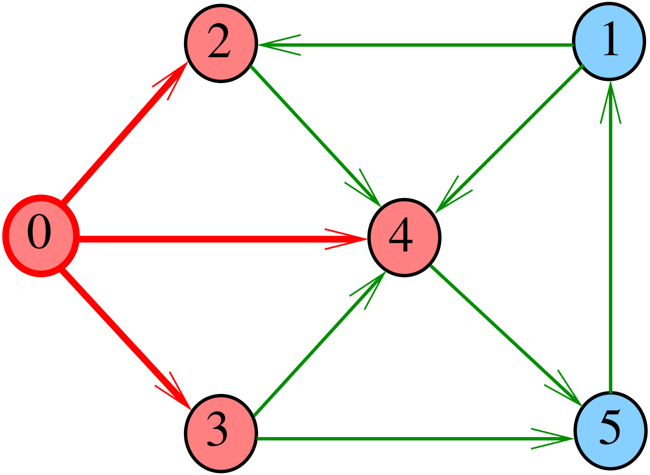
v	0	1	2	3	4	5
dist[v]	0	6	1	1	6	6



Simulação

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

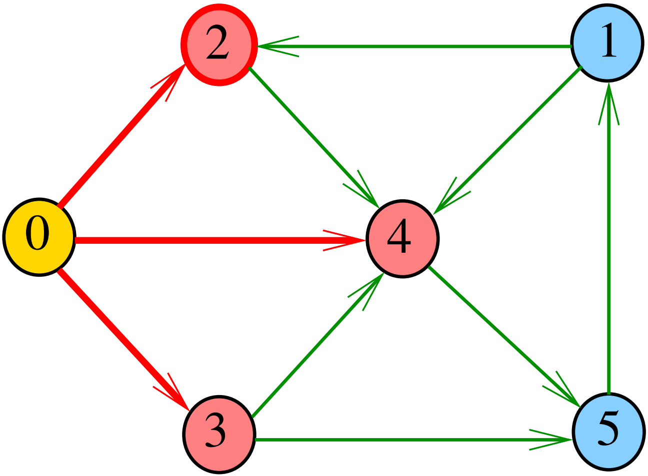
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	6



Simulação

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

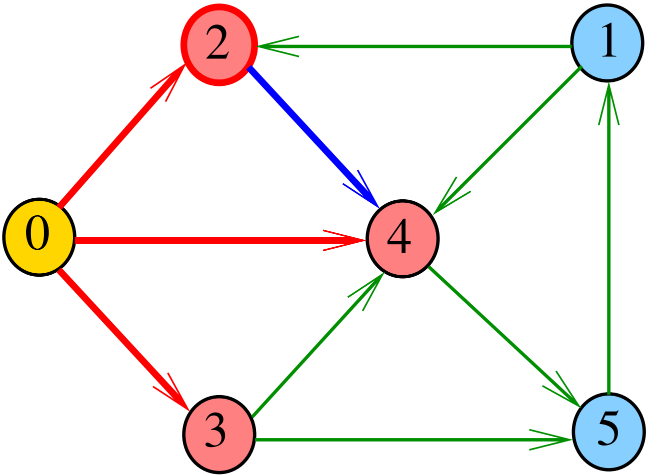
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	6



Simulação

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

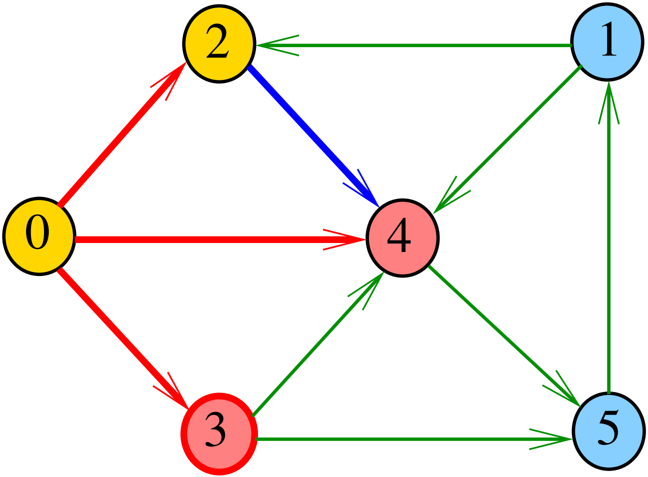
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	6



Simulação

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

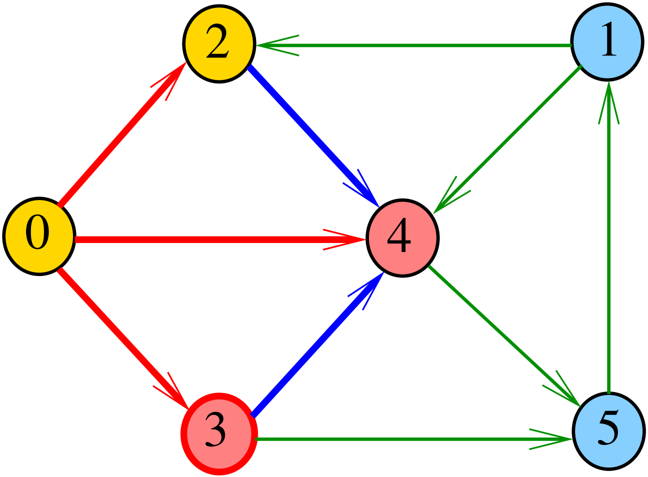
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	6



Simulação

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

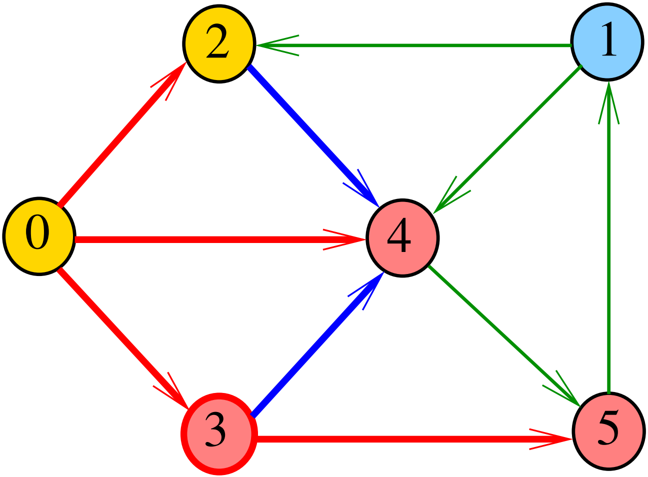
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	6



Simulação

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

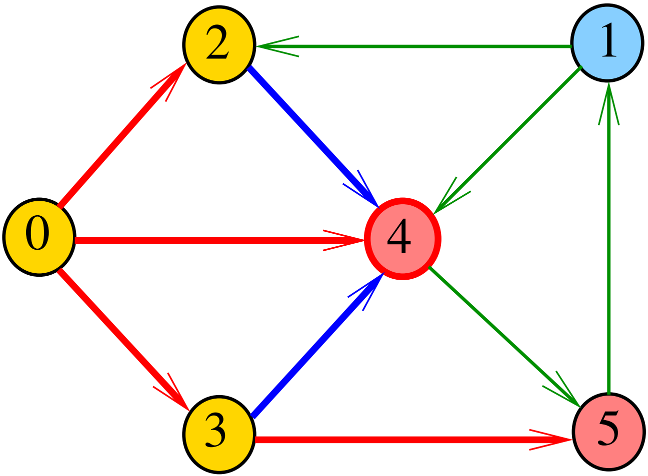
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	2



Simulação

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

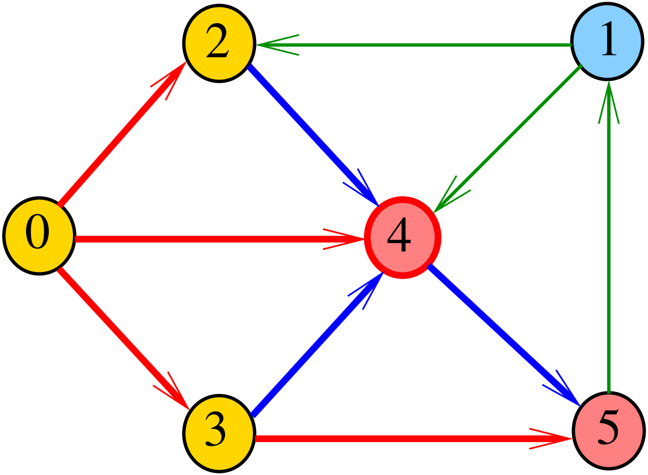
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	2



Simulação

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

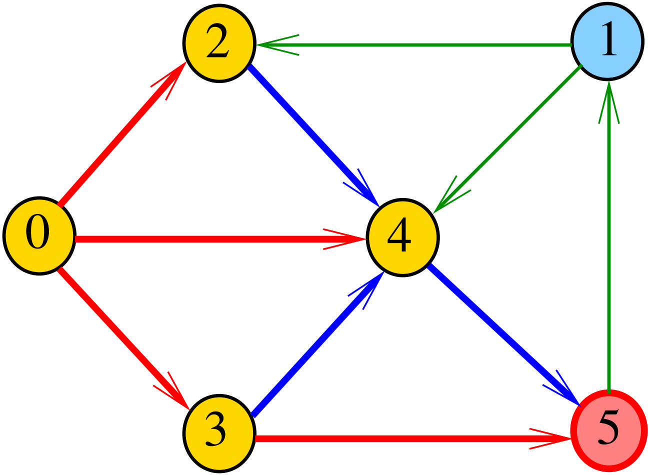
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	2



Simulação

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

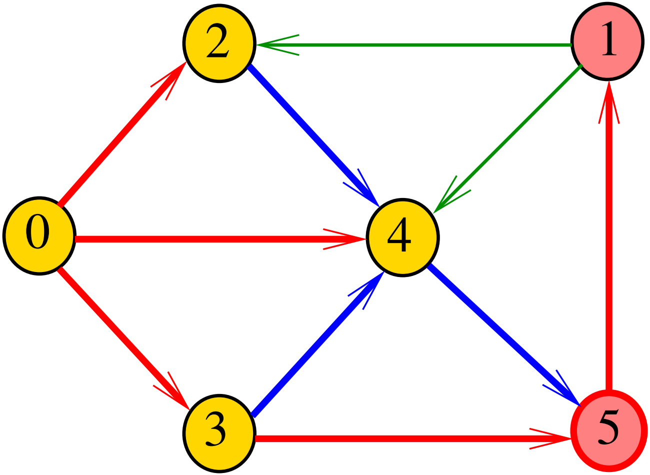
v	0	1	2	3	4	5
dist[v]	0	6	1	1	1	2



Simulação

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

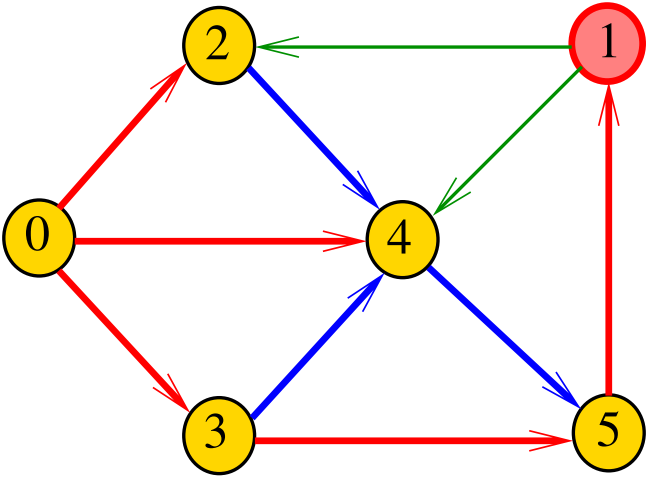
v	0	1	2	3	4	5
dist[v]	0	3	1	1	1	2



Simulação

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

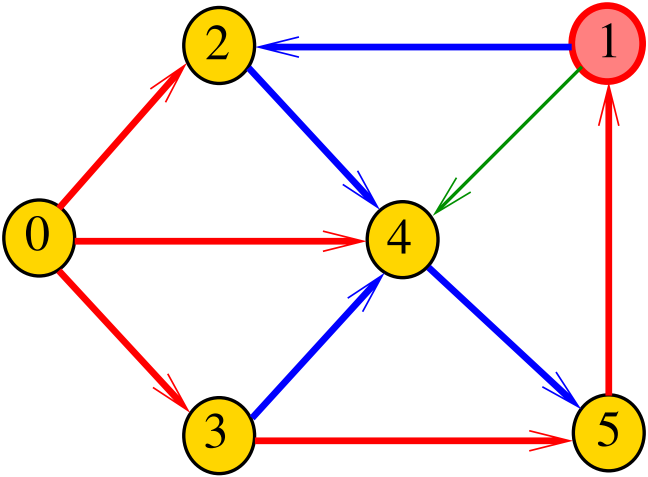
v	0	1	2	3	4	5
dist[v]	0	3	1	1	1	2



Simulação

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

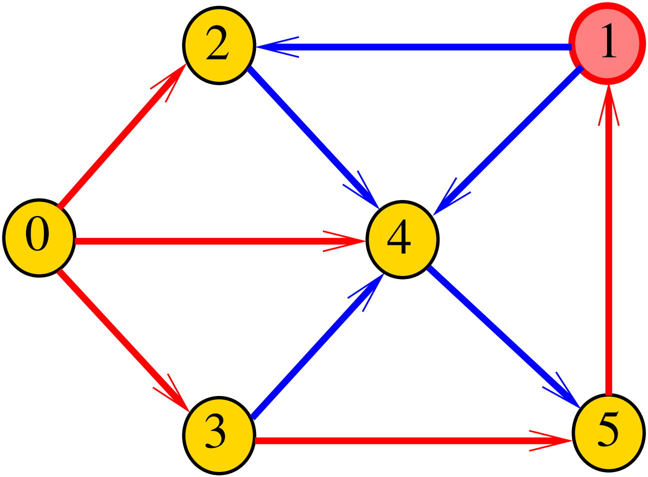
v	0	1	2	3	4	5
dist[v]	0	3	1	1	1	2



Simulação

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

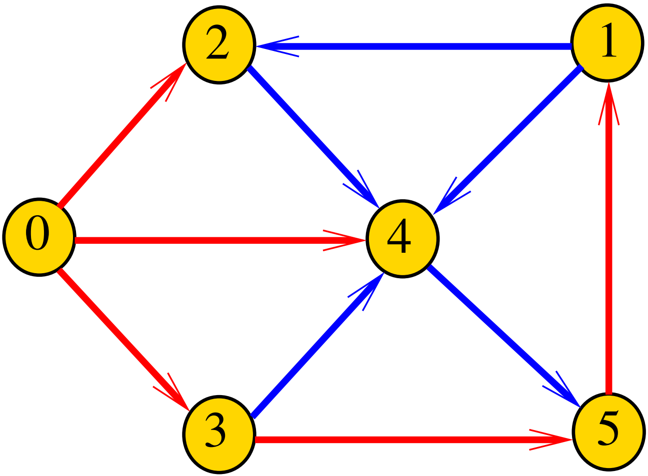
v	0	1	2	3	4	5
dist[v]	0	3	1	1	1	2



Simulação

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

v	0	1	2	3	4	5
dist[v]	0	3	1	1	1	2



DIGRAPHdist

```
#define INFINITO maxV
static int dist[maxV];
static Vertex parnt[maxV];
void DIGRAPHbfs (Digraph G, Vertex s) {
    Vertex v, w; link p;
    for (v = 0; v < G->V; v++)
        dist[v] = INFINITO;
        parnt[v] = -1;
    }
    QUEUEinit(G->V);
    dist[s] = 0;
    parnt[s] = s;
```

DIGRAPHdist

```
QUEUEput(s);
while (!QUEUEempty()) {
    v = QUEUEget();
    for(p=G->adj[v];p!=NULL;p=p->next)
        if (dist[w=p->w] == INFINITO) {
            dist[w] = dist[v] + 1;
            parnt[w] = v;
            QUEUEput(w);
        }
}
QUEUEfree();
}
```

AULA 13

Potenciais

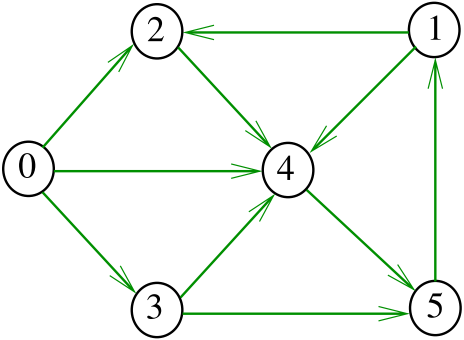
1-potenciais

Um **1-potencial** é um vetor y indexado pelos vértices do digrafo tal que

$$y[w] - y[v] \leq 1 \text{ para todo arco } v\text{-}w$$

Exemplo:

v	0	1	2	3	4	5
y[v]	1	1	1	1	1	1



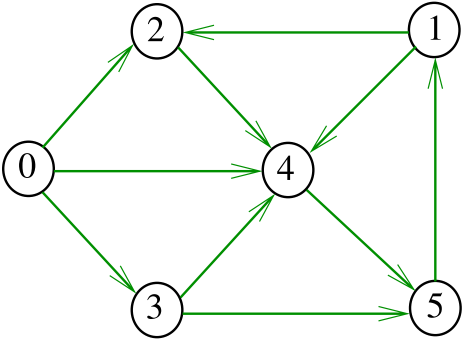
1-potenciais

Um **1-potencial** é um vetor y indexado pelos vértices do digrafo tal que

$$y[\texttt{w}] - y[\texttt{v}] \leq 1 \text{ para todo arco } \texttt{v-w}$$

Exemplo:

v	0	1	2	3	4	5
y[v]	1	2	2	1	1	2



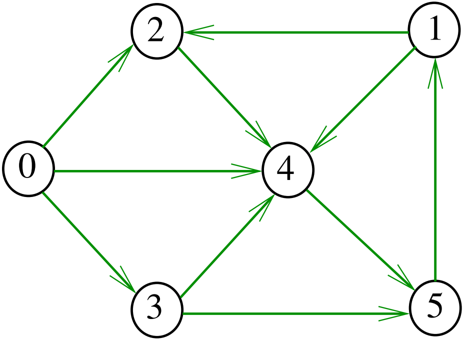
Propriedade dos 1-potencias

Lema da dualidade. Se y é um 1-potencial e P é um caminho de s a t, então

$$y[t] - y[s] \leq |P|$$

Exemplo:

v	0	1	2	3	4	5
y[v]	6	6	6	7	7	7



Conseqüência

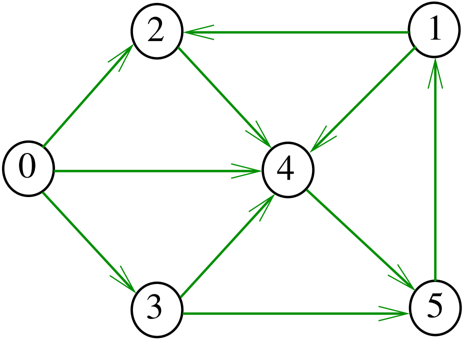
Se P é um caminho de s a t e y é um 1-potencial tais que

$$|P| = y[t] - y[s],$$

então P é um caminho **mínimo** e y é um 1-potencial tal que $y[t] - y[s]$ é **máximo**

Exemplo

v	0	1	2	3	4	5
y[v]	0	2	1	1	1	2



Invariantes

Abaixo está escrito y no papel de `dist`

Na linha 4 da função `DIGRAPHdist` valem as seguintes invariantes:

(i0) para cada arco `v-w` na arborescência BFS tem-se que `y[w]` $-$ `y[v]` $= 1$;

(i1) `parnt[s]` $=$ `s` e `y[s]` $= 0$;

(i2) para cada vértice `v`, `y[v]` $\neq$ `G->V` $\Leftrightarrow$ `parnt[v]` $\neq -1$;

(i3) para cada vértice `v`, se `parnt[v]` $\neq -1$ então existe um caminho de `s` a `v` na arborescência BFS.

Invariantes (continuação)

Abaixo está escrito y no papel de `dist`
Na linha 4 da função `DIGRAPHdist` vale a seguinte relação invariante:

(i4) para cada arco `v-w` se

$$y[w] - y[v] > 1$$

então `v` está na fila.

Correção de DIGRAPHdist

Início da última iteração:

- y é um 1-potencial, por (i4)
- se $\mathtt{y}[\mathtt{t}] \neq \mathtt{G->V}$, então $\mathtt{parnt}[\mathtt{t}] \neq -1$ [(i2)].
  Logo, de (i3), segue que existe um st-caminho P na arborescência BFS. (i0) e (i1) implicam que
  $$|\mathtt{P}| = \mathtt{y}[\mathtt{t}] - \mathtt{y}[\mathtt{s}] = \mathtt{y}[\mathtt{t}].$$
  Da propriedade dos 1-potencias, concluímos que P é um st-caminho de comprimento mínimo
- se $\mathtt{y}[\mathtt{t}] = \mathtt{G->V}$, então (i1) implica que $\mathtt{y}[\mathtt{t}] - \mathtt{y}[\mathtt{s}] = \mathtt{G->V}$ e da propriedade dos 1-potencias concluímos que não existe caminho de s a t no grafo

Conclusão: o algoritmo faz o que promete.

Teorema da dualidade

Da propriedade dos 1-potenciais (lema da dualidade) e da correção de `DIGRAPHdist` concluímos o seguinte:

Se s e t são vértices de um digrafo e t está ao alcance de s então

$$\begin{aligned} &\min\{|\mathtt{P}| : \mathtt{P} \text{ é um } \mathtt{st}\text{-caminho}\} \\ &= \max\{\mathtt{y}[\mathtt{t}] - \mathtt{y}[\mathtt{s}] : \mathtt{y} \text{ é um } 1\text{-potencial}\}. \end{aligned}$$

Custos nos arcos

S 20.1

Digrafos com custos nos arcos

Muitas aplicações associam um número a cada arco de um digrafo

Diremos que esse número é o custo da arco

Vamos supor que esses números são do tipo **double**

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

ARC

A função ARC recebe dois vértices v e w e um valor cst e devolve um arco com ponta inicial v e ponta final w e custo cst

```
Arc ARC (Vertex v, Vertex w, double cst)
{
    Arc e;
    e.v = v;    e.w = w;
    e.cst = cst;
    return e;
}
```

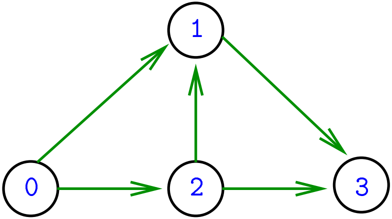
Matriz de adjacência

Matriz de adjacência indica a presença ausência e custo dos arcos:

se `v-w` é um arco, `adj[v][w]` é seu custo

se `v-w` não é arco, `adj[v][w] = maxCST`

Exemplo:



	0	1	2	3
0	$*$	$.3$	2	$*$
1	$*$	$*$	$*$	$.42$
2	$*$	1	$*$	$.12$
3	$*$	$*$	$*$	$*$

$*$ indica `maxCST`

Estrutura digraph

A estrutura **digraph** representa um digrafo

- adj é um ponteiro para a matriz de adjacência
- V contém o número de vértices
- A contém o número de arcos do digrafo.

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

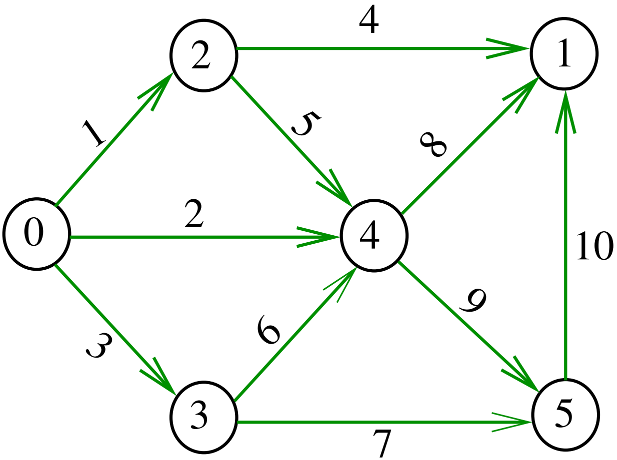
Estrutura `Digraph`

Um objeto do tipo `Digraph` contém o endereço de um `digraph`

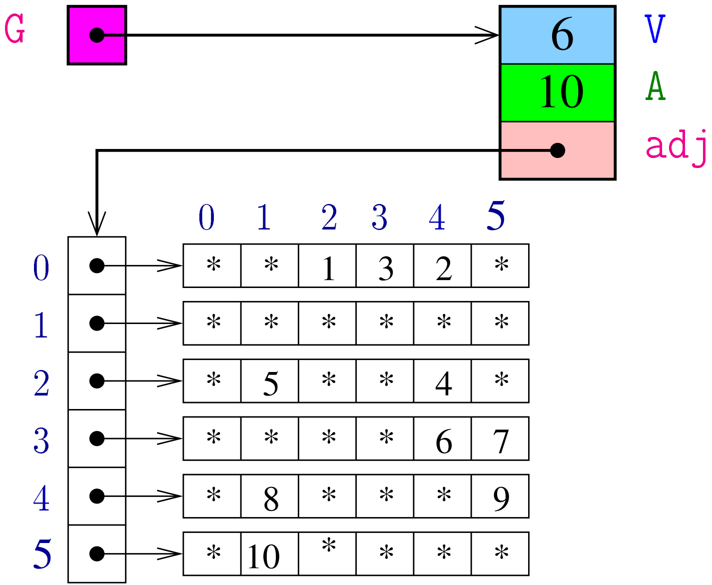
```
typedef struct digraph *Digraph;
```

Digrafo

Digraph G



Estruturas de dados



MATRIXdouble

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

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

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 = MATRIXdouble(V, V, maxCST);
    return G;
}
```

DIGRAPHinsertA

Insere um arco v-w de custo cst 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,
                    double cst)
{
   if (v != w && G->adj[v][w] == maxCST) {
        G->adj[v][w] = cst;
        G->A++;
   }
}
```

Vetor de listas de adjacência

A lista de adjacência de um vértice v é composta por nós do tipo node

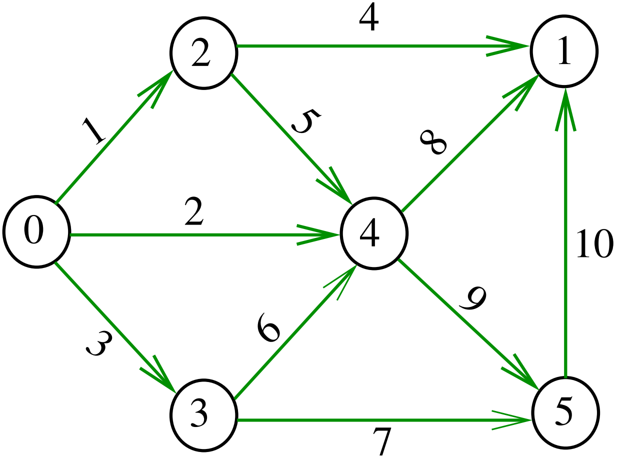
Um link é um ponteiro para um node

Cada nó da lista contém um vizinho w de v, **o custo** do arco v-w e o endereço do nó seguinte da lista

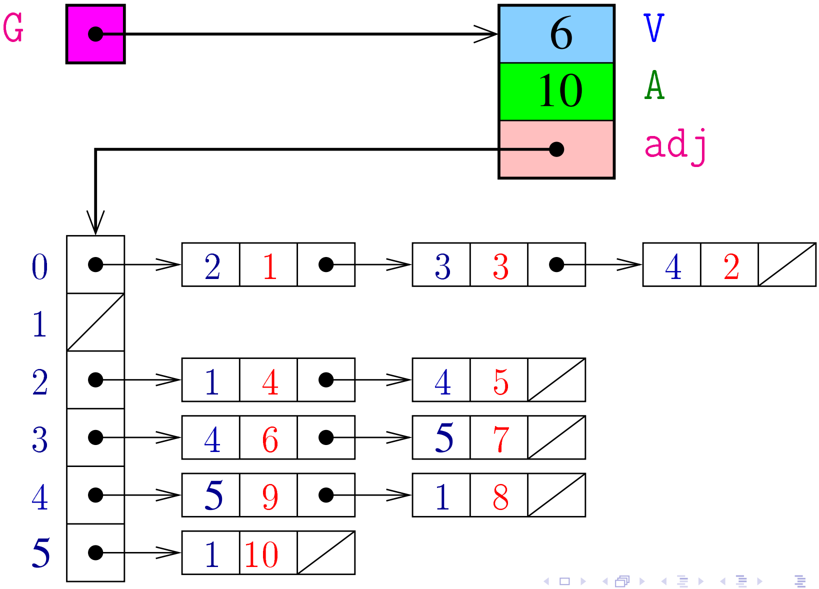
```
typedef struct node *link;
struct node {
    Vertex w;
    double cst;
    link next;
};
```

Digrafo

`Digraph G`



Estruturas de dados



Estrutura digraph

A estrutura **digraph** representa um digrafo

- V contém o número de vértices
- A contém o número de arcos do digrafo
- adj é um ponteiro para vetor de listas de adjacência

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

Estrutura `Digraph`

Um objeto do tipo `Digraph` contém o endereço de um `digraph`

```
typedef struct digraph *Digraph;
```

NEW

NEW recebe um vértice w, um custo cst e o endereço next de um nó e devolve o endereço x de um novo nó com x->w=w, e x->cst=cst e x->next=next

```
link NEW (Vertex w, double cst, link next)
{
    link x = malloc(sizeof *x);
    x->w = w;
    x->cst = cst;
    x->next = next;
    return x;
}
```

DIGRAPHinit

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

```
Digraph DIGRAPHinit (int V) {
    Vertex v;
    Digraph G = malloc(sizeof *G);
    G->V = V;
    G->A = 0;
    G->adj = malloc(V * sizeof(link));
    for (v = 0; v < V; v++)
        G->adj[v] = NULL;
    return G;
}
```

DIGRAPHinsertA

Insere um arco `v-w` de custo `cst` 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,
                    double cst)
{
    link p;
    if (v == w) return;
    for (p = G->adj[v]; p != NULL; p = p->next)
        if (p->w== w) return;
    G->adj[v] = NEW(w, cst, G->adj[v]);
    G->A++;
}
```

DIGRAPHinsertA

O código abaixo transfere a responsabilidade de evitar laços e arcos paralelos ao cliente/usuário

```
void
DIGRAPHinsertA (Digraph G, Vertex v, Vertex w,
                    double cst)
{
    G->adj[v] = NEW(w, cst, G->adj[v]);
    G->A++;
}
```

Caminhos de custo mínimo

S 21.0 e 21.1

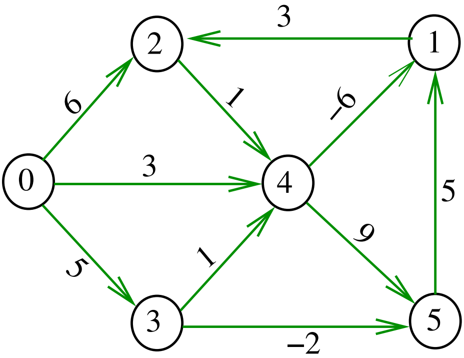
Custo de um caminho

Custo de um caminho é soma dos custos de seus arcos

Custo do caminho 0-2-4-5 é 16.

Custo do caminho 0-2-4-1-2-4-5 é 14.

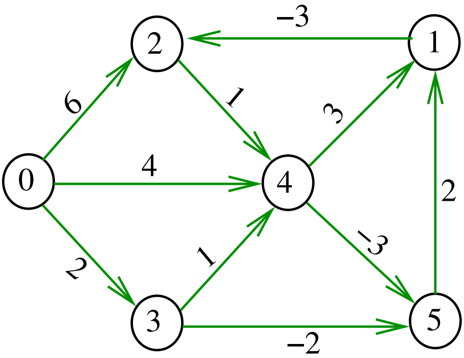
Custo do caminho 0-2-4-1-2-4-1-2-4-5 é 12.



Caminho mínimo

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

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



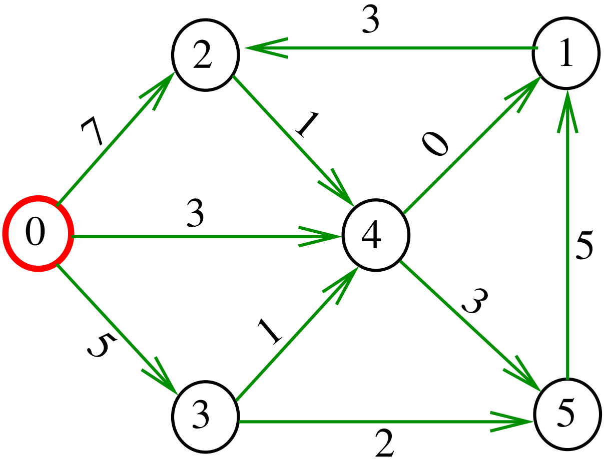
Problema

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

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

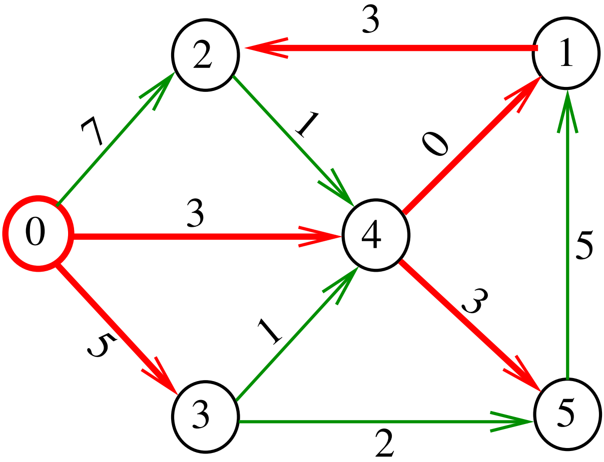
Exemplo

Entra:



Exemplo

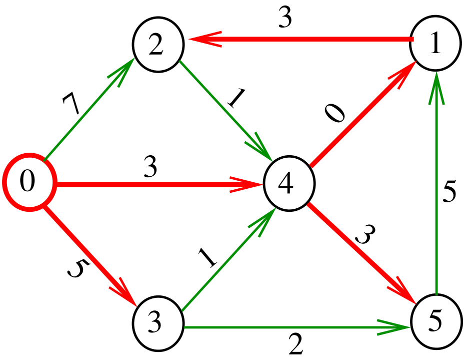
Sai:



Arborescência de caminhos mínimos

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

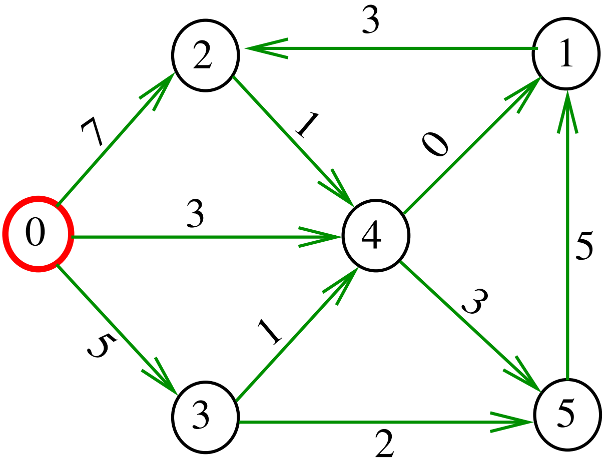
> *o único caminho de s a t na arborescência é um caminho mínimo*



Problema da SPT

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

Entra:



Problema da SPT

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

Sai: