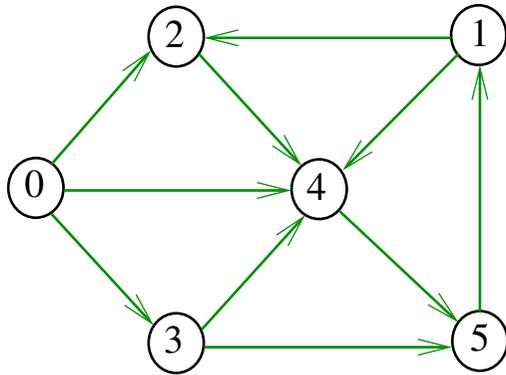


## Calculando distâncias

**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

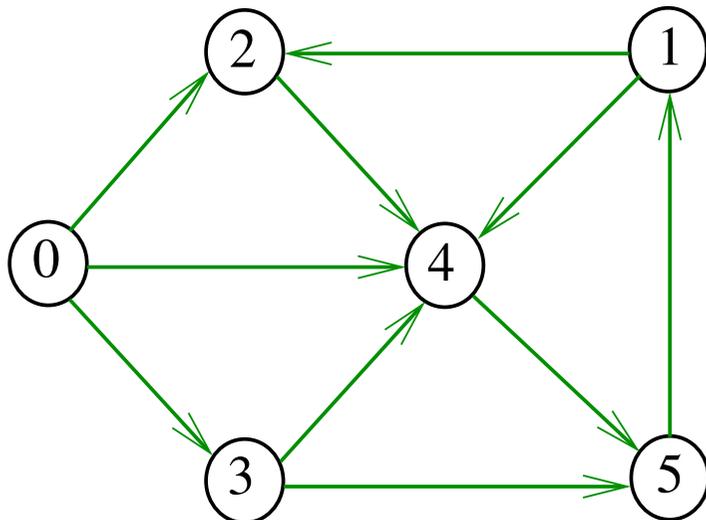
A **busca em largura** (= *breadth-first search* = *BFS*) começa por um vértice, digamos  $s$ , especificado pelo usuário.

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



## DIGRAPHdist

```
#define INFINITO G->V /* ou maxV */
static int dist[maxV];
static Vertex parnt[maxV];
void DIGRAPHdist (Digraph G, Vertex s) {
1  Vertex v, w ; link p ;
2  for (v = 0; v < G->V ; v++)
3      dist[v] = INFINITO;
4      parnt[v] = -1;
5  }
6  QUEUEinit(G->V);
7  dist[s] = 0;
8  parnt[s] = s ;
}
```

## DIGRAPHdist

```
8  QUEUEput(s);
9  while (!QUEUEempty()) {
10     v = QUEUEget();
11     for(p = G->adj[v]; p; p = p->next)
12         if (dist[w = p->w] == INFINITO){
13             dist[w] = dist[v] + 1;
14             parnt[w] = v ;
15             QUEUEput(w);
16         }
17     }
18 }
19 QUEUEfree();
20 }
```

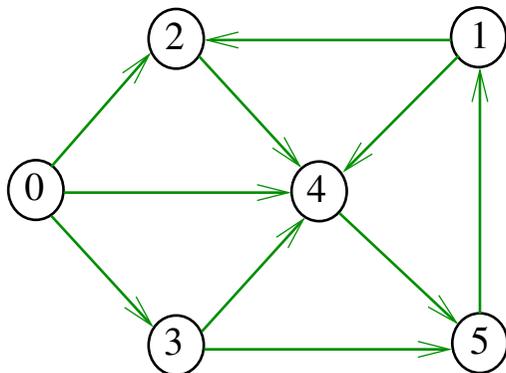
## 1-potenciais

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

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

Exemplo: 

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



## 1-Potenciais

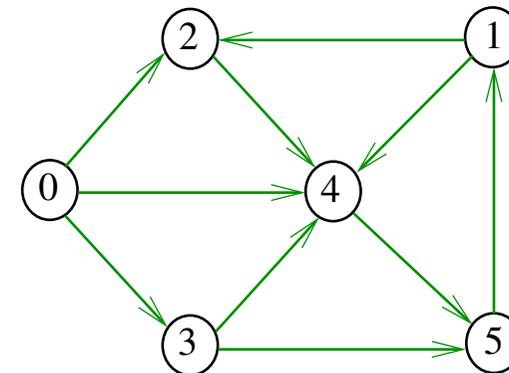
### Propriedade dos 1-potenciais

**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



## Consequência

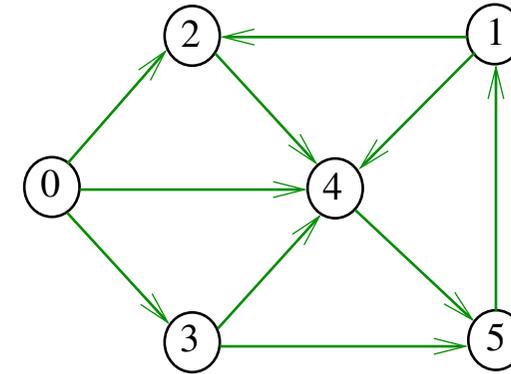
Se  $P$  é um caminho de  $s$  a  $t$  e  $y$  é um 1-potencial tais que

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

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	3	1	1	1	2



## Invariantes

Abaixo está escrito  $y$  no papel de  $dist$   
Na linha 9 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  $v$ ,  
 $y[v] \neq G \rightarrow V \Leftrightarrow parnt[v] \neq -1$ ;
- (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 9 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  $y[t] \neq G \rightarrow V$ , então  $\text{parnt}[t] \neq -1$  [(i2)]. Logo, de (i3), segue que existe um  $s$ - $t$ -caminho  $P$  na arborescência BFS. (i0) e (i1) implicam que

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

Da propriedade dos 1-potenciais, concluímos que

$P$  é um  $s$ - $t$ -caminho de comprimento mínimo

- se  $y[t] = G \rightarrow V$ , então (i1) implica que  $y[t] - y[s] = G \rightarrow V$  e da propriedade dos 1-potenciais concluímos que não existe caminho de  $s$  a  $t$  no grafo

**Conclusão:** o algoritmo faz o que promete.

## Custos nos arcos

S 20.1

## 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

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

## 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**

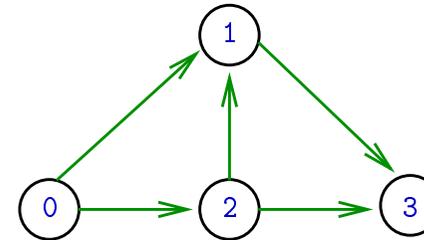
```
Arc ARC (Vertex v , Vertex w , double cst)
{
1   Arc e;
2   e.v = v ;   e.w = w ;
3   e.cst = cst;
4   return e;
}
```

## 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,  $adj[v][w] = \text{maxCST}$

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;
};
```

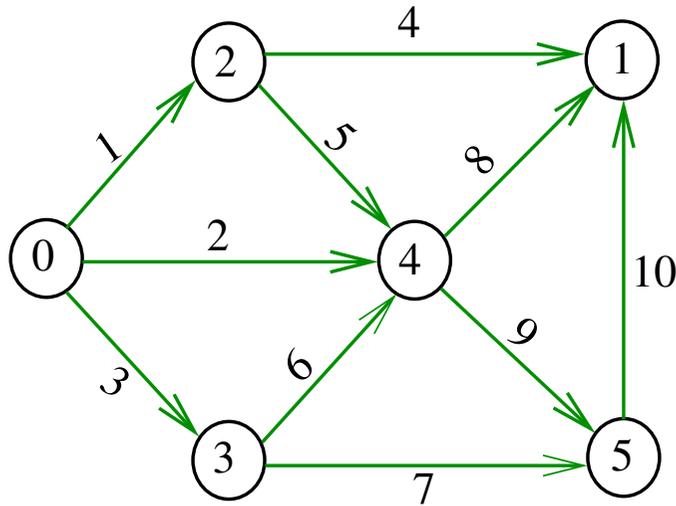
## Estrutura Digraph

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

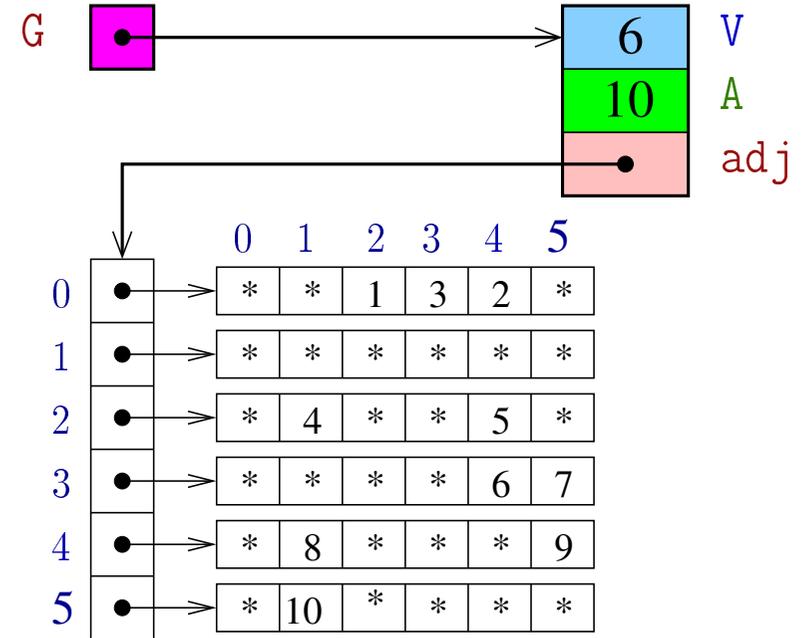
```
typedef struct digraph *Digraph;
```

## Digrafo

Digraph G



## Estruturas de dados



## Com 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

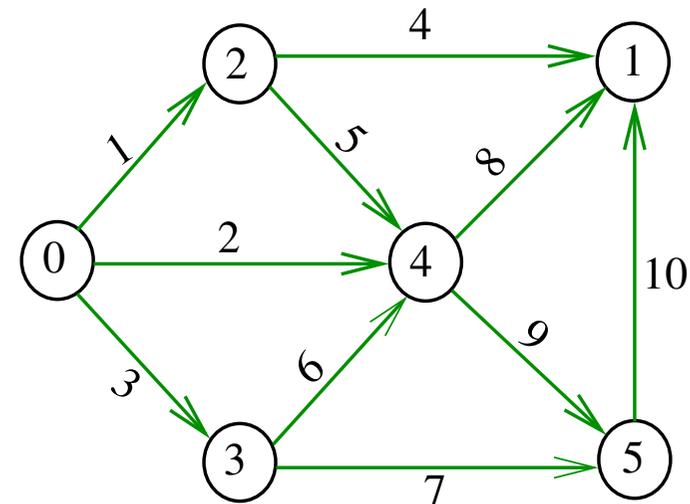
```

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

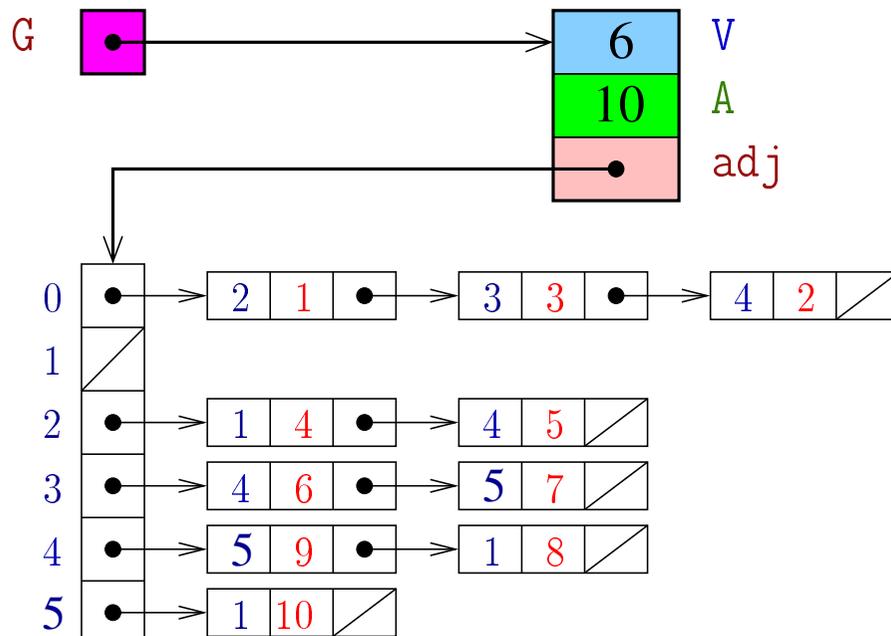
```

## Digrafo

Digraph G



## Estruturas de dados



## Estrutura Digraph

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

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

## 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;
};
```

## Construtor de nós

**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  $x \rightarrow w = w$ , e  $x \rightarrow cst = cst$  e  $x \rightarrow next = next$

```
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  $0, \dots, V-1$  e nenhum arco

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

Algoritmos em Grafos — 1ª sem 2014

## Caminhos de custo mínimo

S 21.0 e 21.1

Algoritmos em Grafos — 1ª sem 2014

## 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 ; p = p->next)
        if (p->w == w) return;
    G->adj[v] = NEW(w, cst, G->adj[v]);
    G->A++;
}
```

29 / 1

Algoritmos em Grafos — 1ª sem 2014

30 / 1

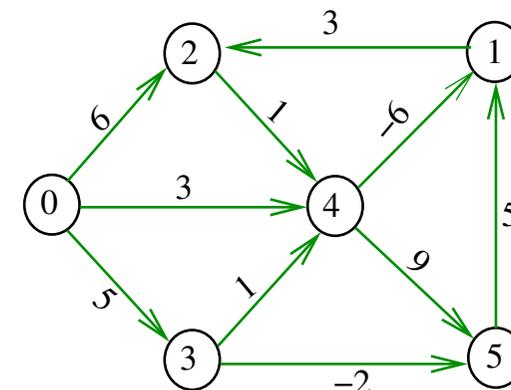
## Custo de um caminho

**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.



31 / 1

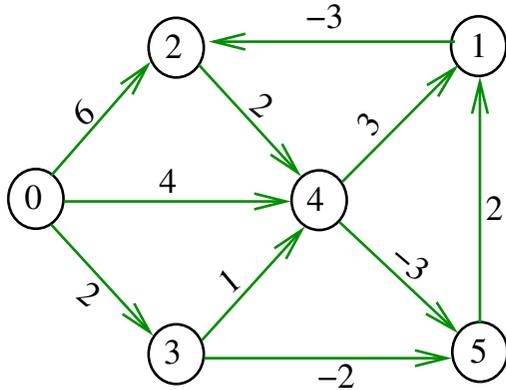
Algoritmos em Grafos — 1ª sem 2014

32 / 1

## Caminho mínimo

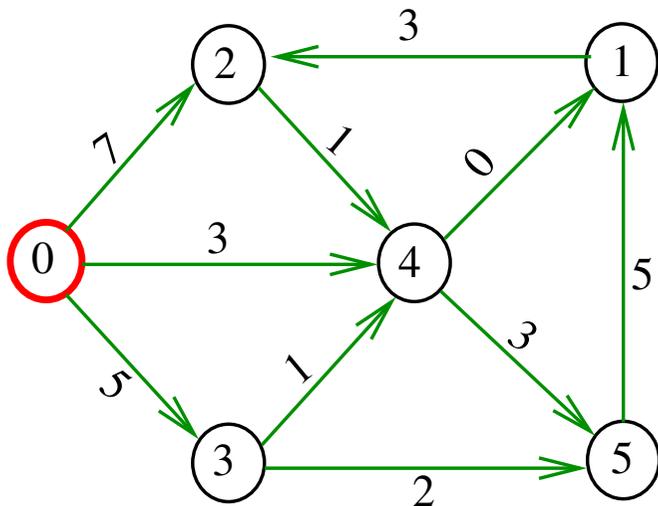
Um caminho  $P$  tem **custo mínimo** se o custo de  $P$  é menor ou igual ao custo de todo caminho com a mesma origem e término

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



## Exemplo

Entra:



Sai:



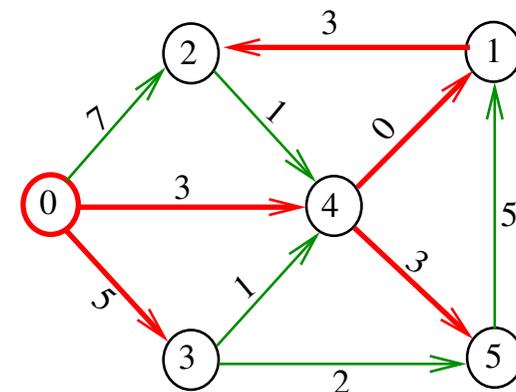
## Problema

**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  $t$ .

## Arborescência de caminhos mínimos

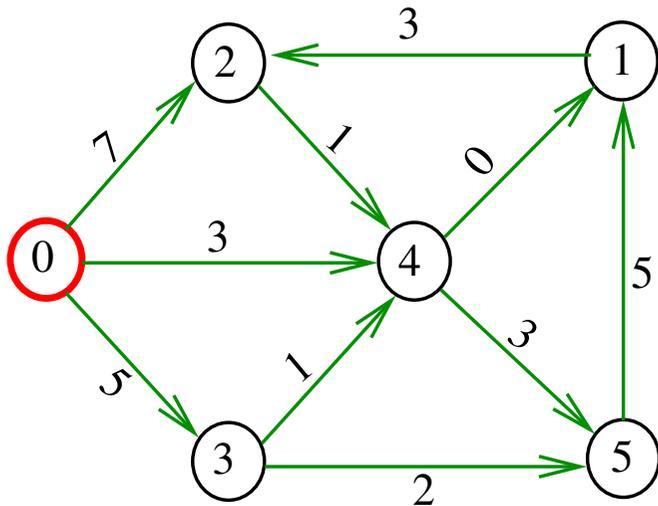
Uma arborescência com raiz  $s$  é de **caminhos mínimos** (= *shortest-paths tree* = *SPT*) se para todo vértice  $t$  que pode ser alcançado a partir de  $s$ , o único caminho de  $s$  a  $t$  na arborescência é um caminho mínimo



## Problema da SPT

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

Entra:



Sai:

