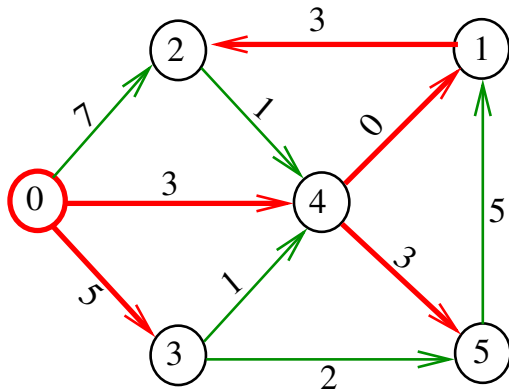


Arborescência de caminhos mínimos

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 simples mínimo



dijkstra

Recebe digrafo **G** com custos **não-negativos** nos arcos e um vértice **s**

Calcula uma arborescência de caminhos mínimos com raiz **s**.

A arborescência é armazenada no vetor `parnt`

As distâncias em relação a **s** são armazenadas no vetor `cst`

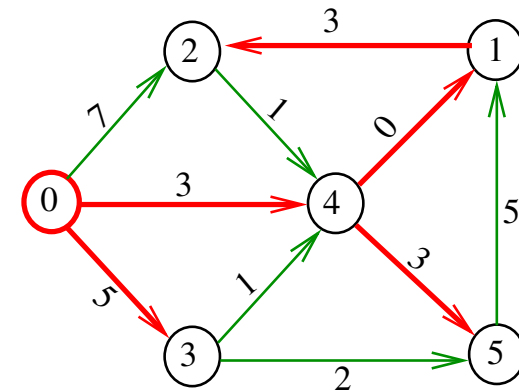
void

```
dijkstra(Digraph G, Vertex s,  
        Vertex parnt[], double cst[]);
```

Problema

O algoritmo de Dijkstra resolve o problema da SPT:

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



Conclusão

O consumo de tempo da função `dijkstra` é $O(V + A)$ mais o consumo de tempo de

- 1 execução de `PQinit` e `PQfree`,
- $O(V)$ execuções de `PQinsert`,
- $O(V)$ execuções de `PQempty`,
- $O(V)$ execuções de `PQdelmin`, e
- $O(A)$ execuções de `PQdec`.

Consumo de tempo Min-Heap

	heap	d -heap	fibonacci heap
PQinsert	$O(\lg V)$	$O(\log_D V)$	$O(1)$
PQdelmin	$O(\lg V)$	$O(\log_D V)$	$O(\lg V)$
PQdec	$O(\lg V)$	$O(\log_D V)$	$O(1)$
dijkstra	$O(A \lg V)$	$O(A \log_D V)$	$O(A + V \lg V)$

Conclusão

O consumo de tempo da função `dijkstra` implementada com um min-heap é $O(A \lg V)$.

Mais algoritmo de Dijkstra

S 21.1 e 21.2

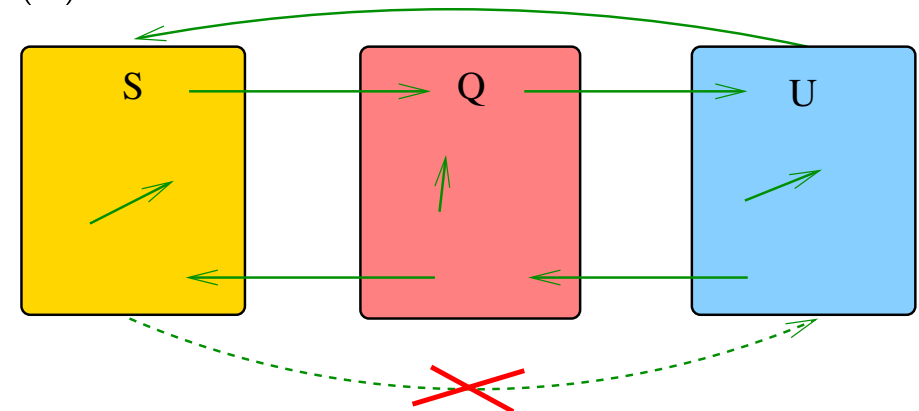
Relações invariantes

S = vértices examinados

Q = vértices visitados = vértices na fila

U = vértices ainda não visitados

(i0) não existe arco $v-w$ com v em S e w em U



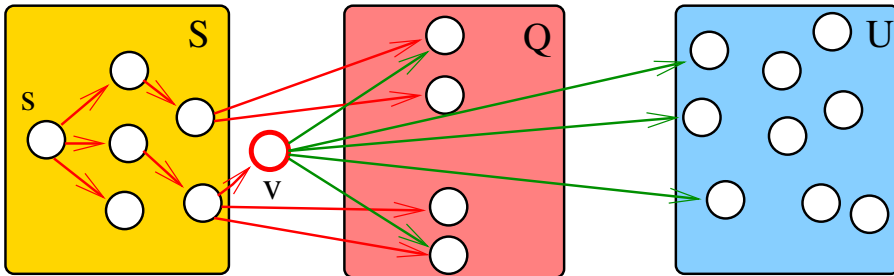
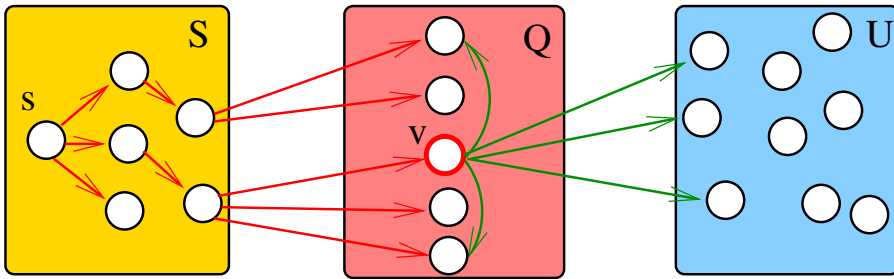
Outra implementação para digrafos densos

```
#define INFINITO maxCST
```

```
void
```

```
DIGRAPHsptD1 (Digraph G, Vertex s,
              Vertex parnt[], double cst[]) {
1  Vertex w, w 0, fr[maxV];
2  for (w = 0; w < G->V ; w++) {
3      parnt[w] = -1;
4      cst[w] = INFINITO;
5  }
6  fr[s] = s ;
7  cst[s] = 0;
```

Iteração



9 / 1

Algoritmos em Grafos — 1º sem 2014

10 / 1

```
8 while (1) {
9     double mincst = INFINITO;
10    for (w = 0; w < G->V ; w++)
11        if (parnt[w]==-1 && mincst>cst[w])
12            mincst = cst[u = w];
13    if (mincst == INFINITO) break;
14    parnt[u] = fr[u];
15    for (w = 0; w < G->V ; w++)
16        if (cst[w]>cst[u]+G->adj[u][w]) {
17            cst[w] = cst[u]+G->adj[u][w];
18            fr[w] = u ;
19        }
20    }
21 }
```

Caminhos mínimos em DAGs

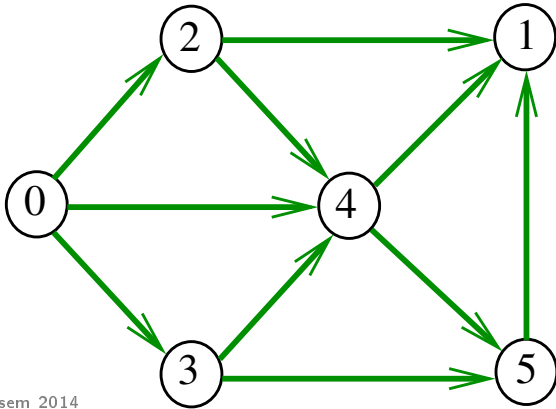
S 19.6

DAGs

Relembrando: Um digrafo é **acíclico** se não tem ciclos

Digrafos acíclicos também são conhecidos como DAGs (= *directed acyclic graphs*)

Exemplo: um digrafo acíclico



Ordenação topológica

Uma **permutação** dos vértices de um digrafo é uma seqüência em que cada vértice aparece uma e uma só vez

Uma **ordenação topológica** (= *topological sorting*) de um digrafo é uma permutação

$ts[0], ts[1], \dots, ts[V-1]$

dos seus vértices tal que todo arco tem a forma

$ts[i] \rightarrow ts[j]$ com $i < j$

$ts[0]$ é necessariamente uma **fonte**
 $ts[V-1]$ é necessariamente um **sorvedouro**

Problema

Problema:

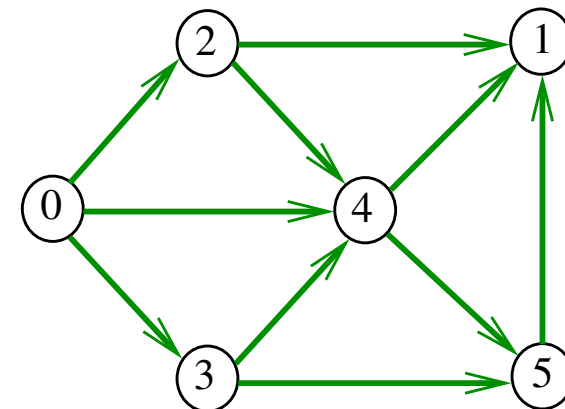
Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar, para cada vértice t que pode ser alcançado a partir de s , um **caminho simples mínimo** de s a t

Problema:

Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar uma **SPT** com raiz s

Exemplo

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



Fato

Para todo digrafo G , vale uma e apenas uma das seguintes afirmações:

- G possui um **ciclo**
- G é um DAG e, portanto, admite uma **ordenação topológica**

Existem algoritmos $O(V + A)$ que encontram ordenação topológica ou ciclo.

Problema

Problema:

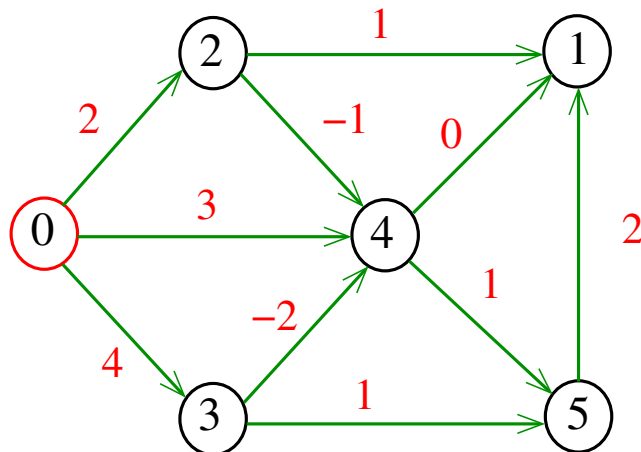
Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar, para cada vértice t que pode ser alcançado a partir de s , um **caminho simples mínimo** de s a t

Problema:

Dado um vértice s de um DAG com custos **possivelmente negativos** nos arcos, encontrar uma **SPT** com raiz s

Simulação

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



DAGmin

A função **DAGmin** recebe um DAG G com custos **possivelmente negativos** e uma ordenação topológica ts de G . Recebe também um vértice s .

Para cada vértice t , a função calcula o custo de um caminho de custo mínimo de s a t . Esse número é depositado em $cst[t]$.

void

DAGmin (Digraph G , Vertex $ts[]$, Vertex s ,
double $cst[]$)

DAGmin

```
void DAGmin (Digraph G, Vertex ts[], Vertex s,
             double cst[]) {
1  int i ; Vertex v ; link p ;
2  for (v = 0; v < G->V ; v++)
3      cst[v] = INFINITO;
4  cst[s] = 0;
5  for (v = ts[i = 0]; i < G->V ; v = ts[i++]){
6      if (cst[v] == INFINITO) continue;
7      for (p =G->adj[v]; p ; p = p ->next)
8          if (cst[p ->w] > cst[v] + p ->cst)
9              cst[p ->w] = cst[v] + p ->cst;
        }
    }
```

Caminhos máximos em DAGs

Do ponto de vista computacional, o problema de encontrar um caminho simples de custo máximo num digrafos com custos nos arcos é difícil.

Mais precisamente, problema é **NP-difícil** como vocês verão no final de *Análise de Algoritmos*.

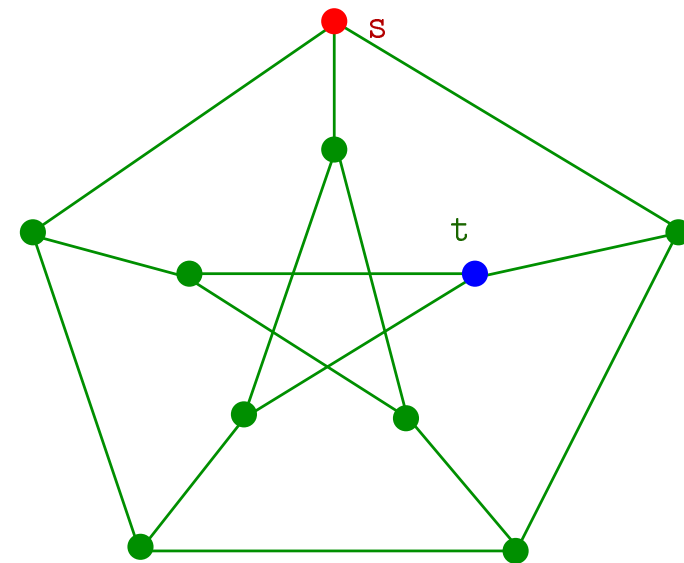
O problema torna-se fácil, entretanto, quando restrito a DAGs.

Consumo de tempo

O consumo de tempo da função `DAGmin` é $O(V + A)$.

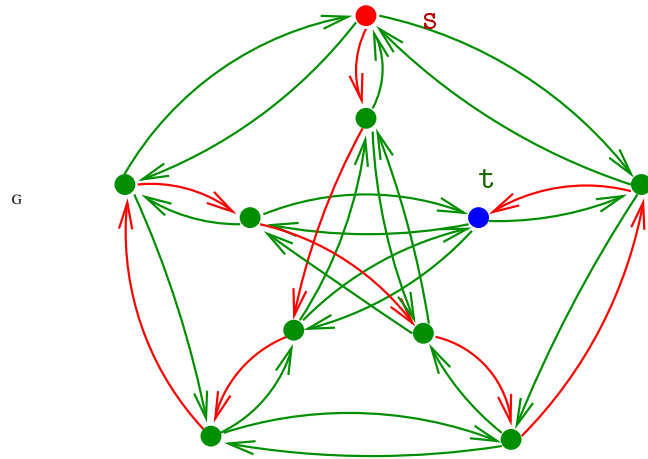
Caminhos hamiltonianos

Problema: Dados vértices `s` e `t` de um grafo encontrar um caminho hamiltoniano de `s` e `t`



Redução polinomial

todos custos = -1



G possui um s t -caminho hamiltoniano \Leftrightarrow
 G possui um s t -caminho **simples** de custo

Complexidade computacional

O problema do **caminho simples** de custo mínimo é NP-difícil.

NP-difícil = **não se conhece** algoritmo de consumo de 'tempo polinomial'

Em outras palavras: **ninguém conhece** um algoritmo eficiente para o problema ...

Se alguém conhece, não contou para ninguém ...

Conclusão

É sabido que:

O problema do caminho hamiltoniano é NP-difícil.

Assim, como problema do caminho simples mínimo com custos negativos é tão difícil quanto o problema do caminho hamiltoniano.

O problema do caminho simples de custo mínimo é NP-difícil.

DAGmax

```

void DAGmax (Digraph G, Vertex ts[], Vertex s,
             double cst[]) {
1  int i ; Vertex v ; link p ;
2  for (v = 0; v < G->V ; v++)
3      cst[v] = -INFINITO;
4  cst[s] = 0;
5  for (v = ts[i = 0]; i < G->V ; v = ts[++i]){
6      if (cst[v] == -INFINITO) continue;
7      for (p = G->adj[v]; p != NULL; p = p
->next)
8          if (cst[p->w] < cst[v] + p->cst)
9              cst[p->w] = cst[v] + p->cst;
}

```

Consumo de tempo

O consumo de tempo da função `DAGmax` é
 $O(V + A)$.