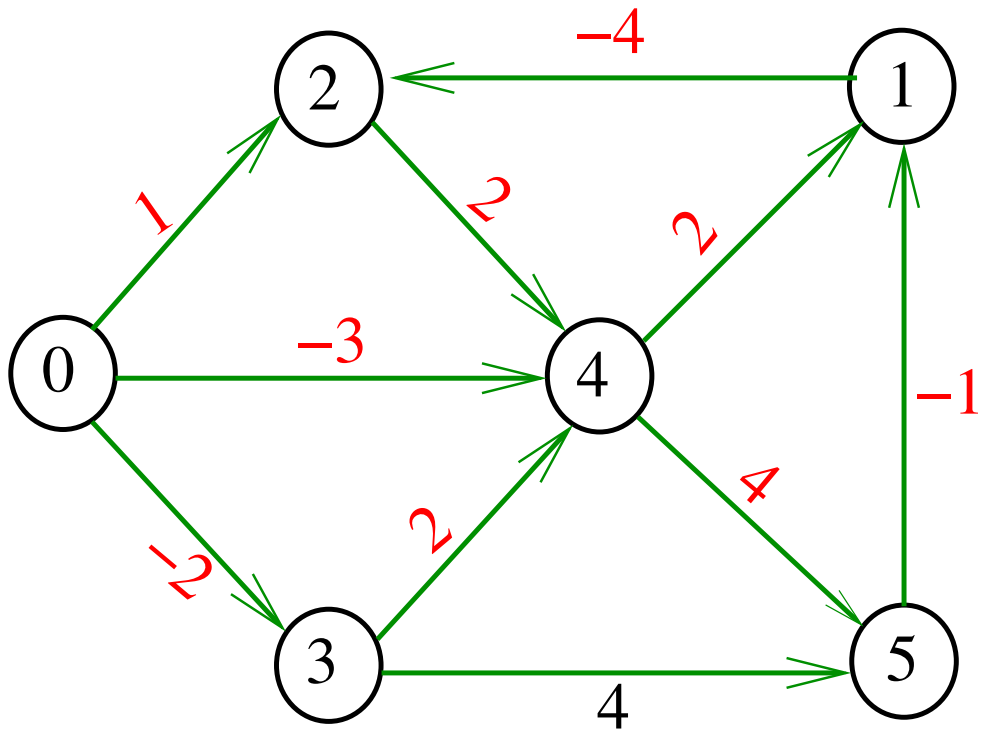
Melhores momentos

AULA 18

Problema da SPT

Problema: Dado um vértice s de um digrafo com custos (possivelmente negativos) nos arcos, encontrar uma SPT com raiz s

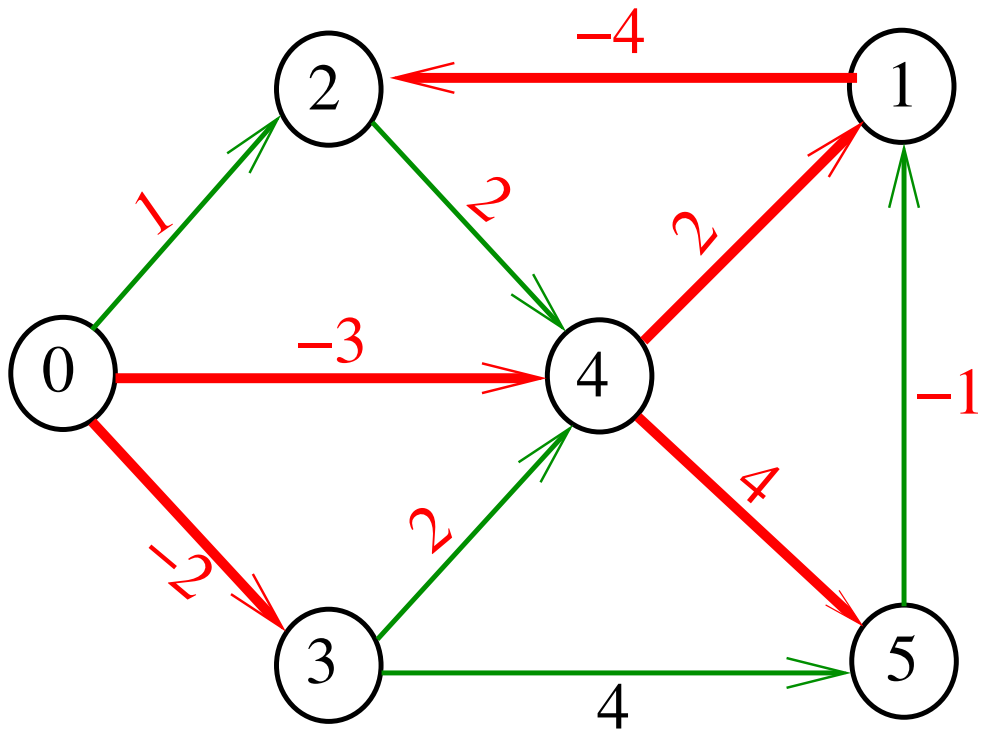
Entra:



Problema da SPT

Problema: Dado um vértice s de um digrafo com custos (possivelmente negativos) nos arcos, encontrar uma SPT com raiz s

Sai:



Fato

O algoritmo de Dijkstra não funciona para digrafos com custos negativos, mesmo que o digrafo seja acíclico.

Programação dinâmica

Propriedade (da subestrutura ótima)
Se G é um digrafo com custo (possivelmente negativos) nos arcos, sem **ciclos negativos** e v_0-v_1-v_2-...-v_k é um caminho mínimo então v_i-v_{i+1}-...-v_j é um caminho mínimo para $0 \leq i \leq j \leq k$

Bellman-Ford

`custo[k][w]` = menor custo de um caminho de `s` a `w` com $\leq$ `k` arcos.

Recorrência:

$$
\begin{aligned}
\texttt{custo[0][s]} &= 0 \\
\texttt{custo[0][w]} &= \texttt{maxCST}, \ \texttt{w} \neq \texttt{s} \\
\texttt{custo[k][w]} &= \min\{\texttt{custo[k-1][w]}, \\
&\qquad \min\{\texttt{custo[k-1][v]} + \texttt{G->adj[v][w]}\}\}
\end{aligned}
$$

Se o digrafo não tem ciclo negativo acessível a partir de `s`, então `custo[V-1][w]` é o menor custo de um **caminho simples** de `s` a `w`

Ciclos negativos

Se `custo[V][v]` $\neq$ `custo[V-1][v]`, então `G` tem um ciclo negativo alcançável a partir de `s`.

Se `G` tem um ciclo negativo alcançável a partir de `s`, então `custo[V][v]` $\neq$ `custo[V-1][v]` para algum vértice `v`.

Consumo de tempo

O consumo de tempo do algoritmo de Bellman-Ford é $O(V^3)$.

Se o grafo for representado através de vetor de listas de adjacência e implementarmos uma fila de tal forma que cada operação consuma tempo constante teremos:

O consumo de tempo do algoritmo de Bellman-Ford é $O(VA)$.

Ciclos negativos

Problema: Dado um digrafo com custos nos arcos, decidir se o digrafo possui algum ciclo negativo.

Uma adaptação da função `bellman_ford` decide se um dado digrafo com custos nos arcos possui algum ciclo negativo. O consumo de tempo dessa função adaptada é $O(VA)$.

Resumo

função	consumo de tempo	observação
`DAGmin`	$O(V + A)$	digrafos acíclicos custos arbitrários
`dijkstra`	$O(A \lg V)$	custos ≥ 0, min-heap
	$O(V^2)$	custos ≥ 0, fila
`bellman-ford`	$O(V^3)$ $O(VA)$	digrafos densos digrafos esparços

O problema SPT em digrafos com ciclos negativos é NP-difícil.

AULA 19

c-Potenciais

Era uma vez um passarinho...

Suponha que um passarinho nos deu

- um digrafo G com custos (possivelmente negativos) nos arcos
- um vetor `cst[ ]` indexado pelos vértices do digrafo
- um parnt `parnt[ ]` indexado pelos vértices do digrafo

e nos disse que

- `cst[v]` é o custo de um caminho mínimo de um vértice `s` a `v`, para cada vértice `v`
- `parnt[ ]` representa uma SPT com raiz `s`

Como podemos verificar essas afirmações?

É fácil . . .

Basta verificar se

- `cst[s]` $= 0$ e `parnt[s]` $=$ `s`
- para cada arco `v-w` vale que

$$\texttt{cst[w]} \leq \texttt{cst[v]} + \texttt{G->adj[v][w]}$$

- para cara arco `v-w` com `v=parnt[w]` vale que

$$\texttt{cst[w]} = \texttt{cst[v]} + \texttt{G->adj[v][w]}$$

```
int eh_SPT (Digraph G, Vertex s,
            double cst[ ], Vertex parnt[ ]) {
   Vertex v, w;link p; double d;
   if (cst[s]!=0 || parnt[s]!=s) return 0;
   for (v = 0; v < G->V; v++)
   for (p=G->adj[v];p!=NULL;p=p->next){
       w = p->w; d= cst[v] + p->cst;
       if (v!=parnt[w] && cst[p->w]>d)
           return 0;
       if (s!=w && v==parnt[w]
               && cst[p->w]!=d)
           return 0;
   }
   return 1;
}
```

Conclusão

A função que verifica a afirmação do passarinho consome tempo $O(V + A)$.

Não sabemos encontrar os vetores `cst[ ]` e `parnt[ ]` em tempo $O(V + A)$.

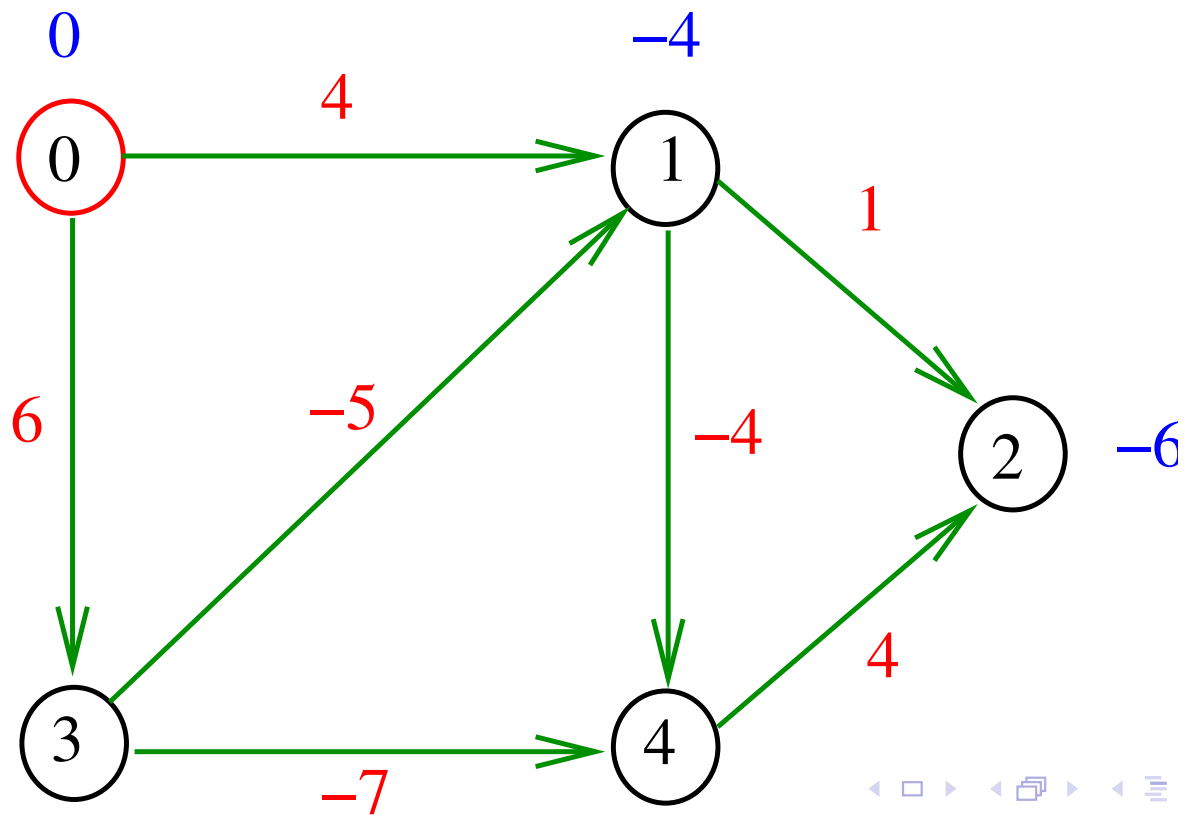
Frequentemente, verificar a resposta é mais 'fácil' do que encontrá-la.

c-potenciais

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

$$y[w] \leq y[v] + \texttt{G->adj[v][w]} \text{ para todo arco } \texttt{v-w}$$

Exemplo:

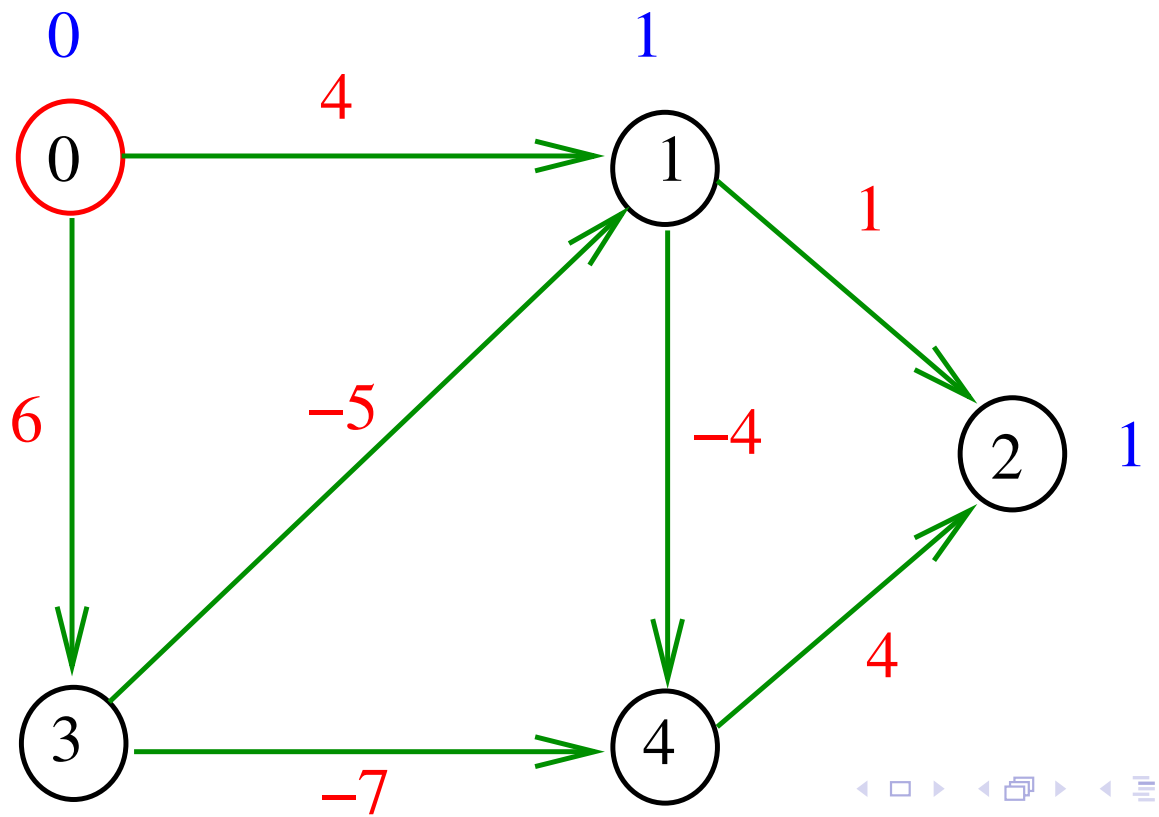


c-potenciais

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

$$y[w] \leq y[v] + \texttt{G->adj[v][w]} \text{ para todo arco } \texttt{v-w}$$

Exemplo:

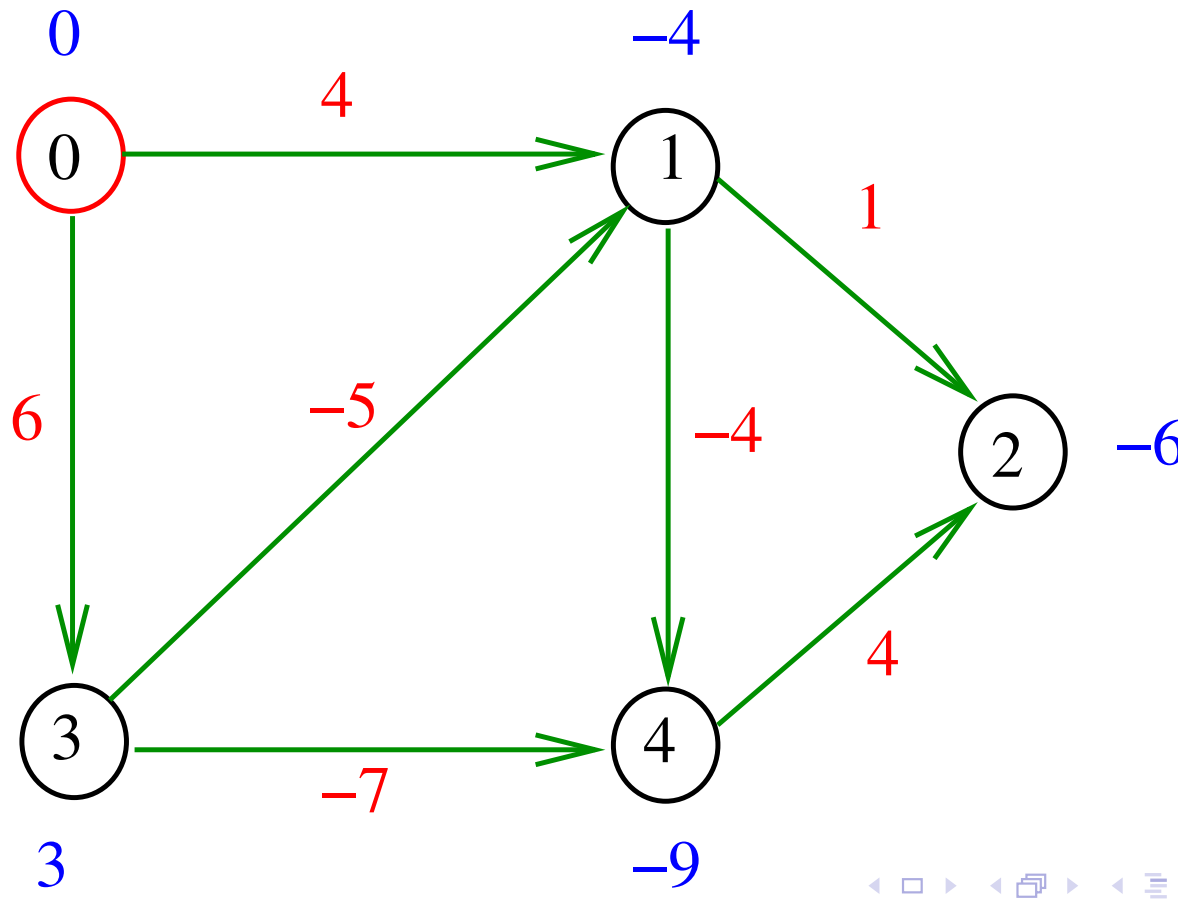


Propriedade dos c-potencias

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

$$\texttt{custo}(\texttt{P}) + \texttt{y}[\texttt{s}] \geq \texttt{y}[\texttt{t}]$$

Exemplo:



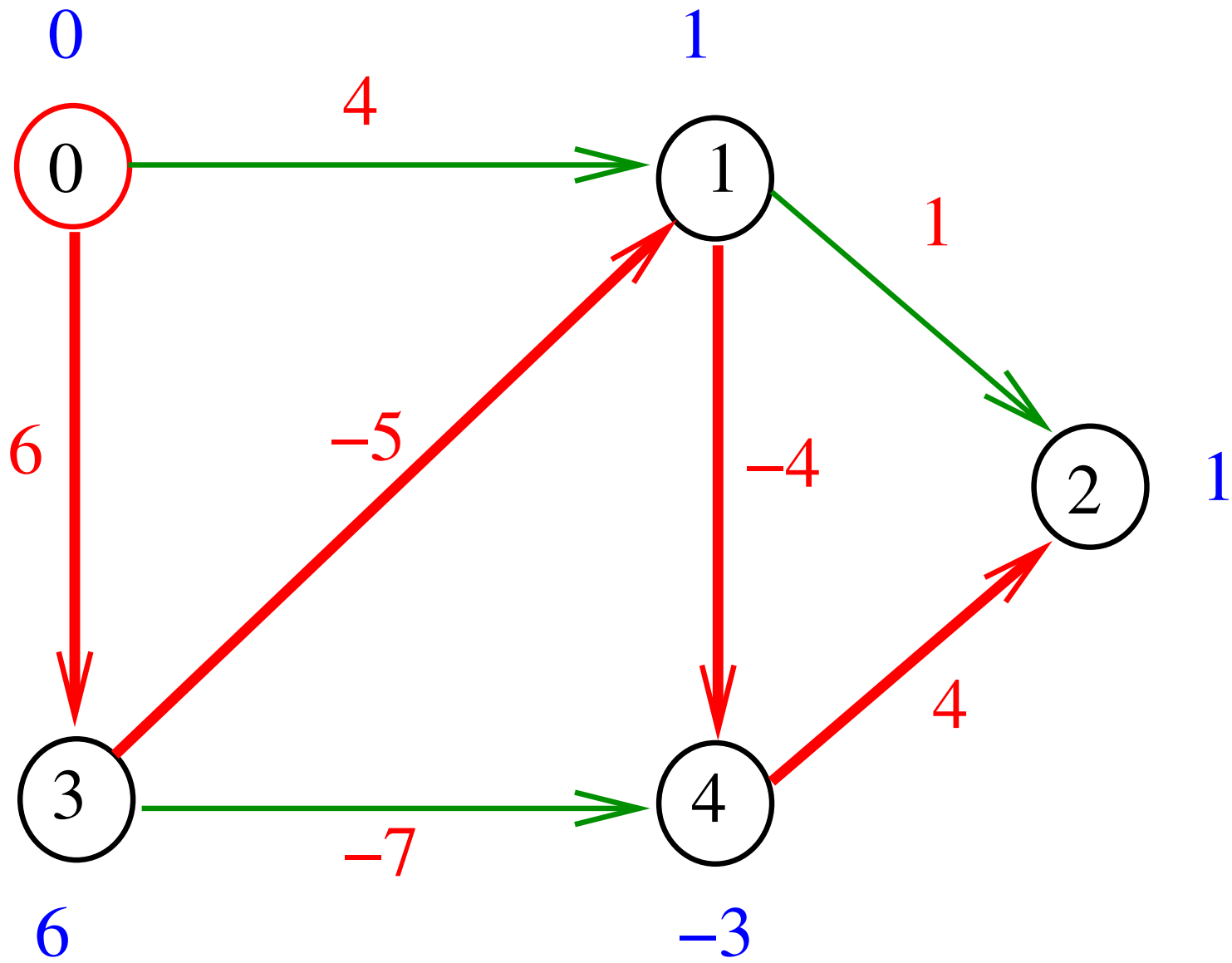
Conseqüência

Se $\mathtt{P}$ é um caminho de $\mathtt{s}$ a $\mathtt{t}$ e $\mathtt{y}$ é um c-potencial tais que

$$\mathtt{custo}(\mathtt{P}) + \mathtt{y}[\mathtt{s}] = \mathtt{y}[\mathtt{t}],$$

então $\mathtt{P}$ é um caminho **mínimo** e $\mathtt{y}$ é um c-potencial tal que $\mathtt{y}[\mathtt{t}] - \mathtt{y}[\mathtt{s}]$ é **máximo**

Exemplo



Invariantes do algoritmo de Bellman-Ford

Abaixo está escrito `y` no papel de `cst` e
Na linha 9 da função `bellman-ford` valem as seguintes invariantes:

(i0) para cada arco `v-w` na arborescência tem-se que $\texttt{y[w]} - \texttt{y[v]} = \texttt{G->adj[v][w]}$;

(i1) $\texttt{parnt[s]} = \texttt{s}$ e $\texttt{y[s]} = 0$;

(i2) para cada vértice `v`,
$\texttt{y[v]} \neq \texttt{maxCST} \Leftrightarrow \texttt{parnt[v]} \neq -1$;

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

Correção de bellman-ford

Início da última iteração:

- y é um c-potencial
- se $\mathtt{y}[\mathtt{t}] \neq \mathtt{maxCST}$, então $\mathtt{parnt}[\mathtt{t}] \neq -1$ [(i2)]. Logo, de (i3), segue que existe um st-caminho P na arborescência. (i0) e (i1) implicam que

$$\mathtt{custo}(\mathtt{P}) = \mathtt{y}[\mathtt{t}] - \mathtt{y}[\mathtt{s}] = \mathtt{y}[\mathtt{t}].$$

  Da propriedade dos c-potencias, concluímos que P é um st-caminho de comprimento mínimo
- se $\mathtt{y}[\mathtt{t}] = \mathtt{maxCST}$, então (i1) implica que $\mathtt{y}[\mathtt{t}] - \mathtt{y}[\mathtt{s}] = \mathtt{maxCST}$ e da propriedade dos c-potencias concluímos que não existe caminho de s a t no digrafo

Conclusão: o algoritmo faz o que promete.

Teorema da dualidade

Da propriedade dos c-potenciais (lema da dualidade) e da correção de `bellman-ford` concluímos o seguinte:

Se `s` e `t` são vértices de um diagrafo, para cada arco `v-w` do digrafo c(`v-w`) é o custo de `v-w`, não existe ciclo negativo ao alcance de `s` e `t` está ao alcance de `s` então

$$\min\{\texttt{custo}(\texttt{P}) : \texttt{P} \text{ é um } \texttt{st}\text{-caminho}\}$$
$$= \max\{\texttt{y}[\texttt{t}] - \texttt{y}[\texttt{s}] : \texttt{y} \text{ é um } c\text{-potencial}\}.$$

Algoritmo de Floyd-Warshall

S 21.3

Problema dos caminhos mínimos entre todos os pares

Problema: Dado um digrafo com custo nos arcos, determinar, para cada par de vértices `s`, `t` o custo de um caminho mínimo de `s` a `t`

Problema dos caminhos mínimos entre todos os pares

Problema: Dado um digrafo com custo nos arcos, determinar, para cada par de vértices `s`, `t` o custo de um caminho mínimo de `s` a `t`

Esse problema pode ser resolvido aplicando-se `V` vezes o algoritmo de Bellman-Ford

O consumo de tempo dessa solução é $O(V^2A)$.

Problema dos caminhos mínimos entre todos os pares

Problema: Dado um digrafo com custo nos arcos, determinar, para cada par de vértices `s`, `t` o custo de um caminho mínimo de `s` a `t`

Esse problema pode ser resolvido aplicando-se `V` vezes o algoritmo de Bellman-Ford

O consumo de tempo dessa solução é $O(V^2A)$.

Um algoritmo mais eficiente foi descrito por Floyd, baseado em uma idéia de Warshall.

O algoritmo supõe que o digrafo não tem ciclo negativo

Programação dinâmica

$0, 1, 2, \ldots, \texttt{V}-1$ = lista dos vértices do digrafo

$\texttt{custo}[\texttt{k}][\texttt{s}][\texttt{t}]$ = menor custo de um caminho de $\texttt{s}$ a $\texttt{t}$ usando vértices em $\{\texttt{s}, \texttt{t}, 0, 1, \ldots, \texttt{k}-1\}$

Programação dinâmica

$0, 1, 2, \ldots,$ V-1 = lista dos vértices do digrafo

custo[k][s][t] = menor custo de um caminho de s a t usando vértices em $\{$s, t, $0, 1, \ldots,$ k$-1\}$

Recorrência:

$$\begin{aligned} \texttt{custo[0][s][t]} &= \texttt{G->adj[s][t]} \\ \texttt{custo[k][s][t]} &= \min\{\texttt{custo[k-1][s][t]}, \\ &\qquad \texttt{custo[k-1][s][k-1]} + \texttt{custo[k-1][k-1][t]}\} \end{aligned}$$

Se o digrafo não tem ciclo negativo acessível a partir de s, então custo[V][s][t] é o menor custo de um **caminho simples** de s a t

```
void floyd_warshall (Digraph G){
 1    Vertex s, t; double d;
 2    for (s=0; s < G->V; s++)
 3         for (t=0; t < G->V; t++)
 4              custo[0][s][t] = G->adj[s][t];
 5    for (k=1; k <=G->V; k++)
 6         for (s=0; s < G->V; s++)
 7              for (t=0; t < G->V; t++){
 8                   custo[k][s][t]=custo[k-1][s][t];
 9                   d=custo[k-1][s][k-1]
                              +custo[k-1][k-1][t];
10                   if (custo[k][s][t] > d)
11                        custo[k][s][t] = d;
                  }
          }
}
```

Consumo de tempo

O consumo de tempo da função `floyd_warshall1` é $O(V^3)$.

```
void floyd_warshall (Digraph G){
 1    Vertex s, t; double d;
 2    for (s=0; s < G->V; s++)
 3         for (t=0; t < G->V; t++)
 4              cst[s][t] = G->adj[s][t];
 5    for (k=1; k <= G->V; k++)
 6         for (s=0; s < G->V; s++)
 7              for (t=0; t < G->V; t++){
 8                   d=cst[s][k-1]+cst[k-1][t];
10                   if (cst[s][t] > d)
11                        cst[s][t] = d;
                  }
          }
}
```

Relação invariante

No início de cada iteração da linha 5 vale que

$\texttt{cst}[\texttt{s}][\texttt{t}] = \texttt{custo}[\texttt{k}][\texttt{s}][\texttt{t}]$ = o menor custo de um caminho de $\texttt{s}$ a $\texttt{t}$ usando vértices em $\{\texttt{s}, \texttt{t}, 0, 1, \ldots, \texttt{k}-1\}$

Novo resumo

função	consumo de tempo	observação
`DAGmin`	$O(V + A)$	digrafos acíclicos custos arbitrários
`dijkstra`	$O(A \lg V)$	custos ≥ 0, min-heap
	$O(V^2)$	custos ≥ 0, fila
`bellman-ford`	$O(V^3)$ $O(VA)$	digrafos densos digrafos esparços
`floyd-warshall`	$O(V^3)$	digrafos sem ciclos negativos

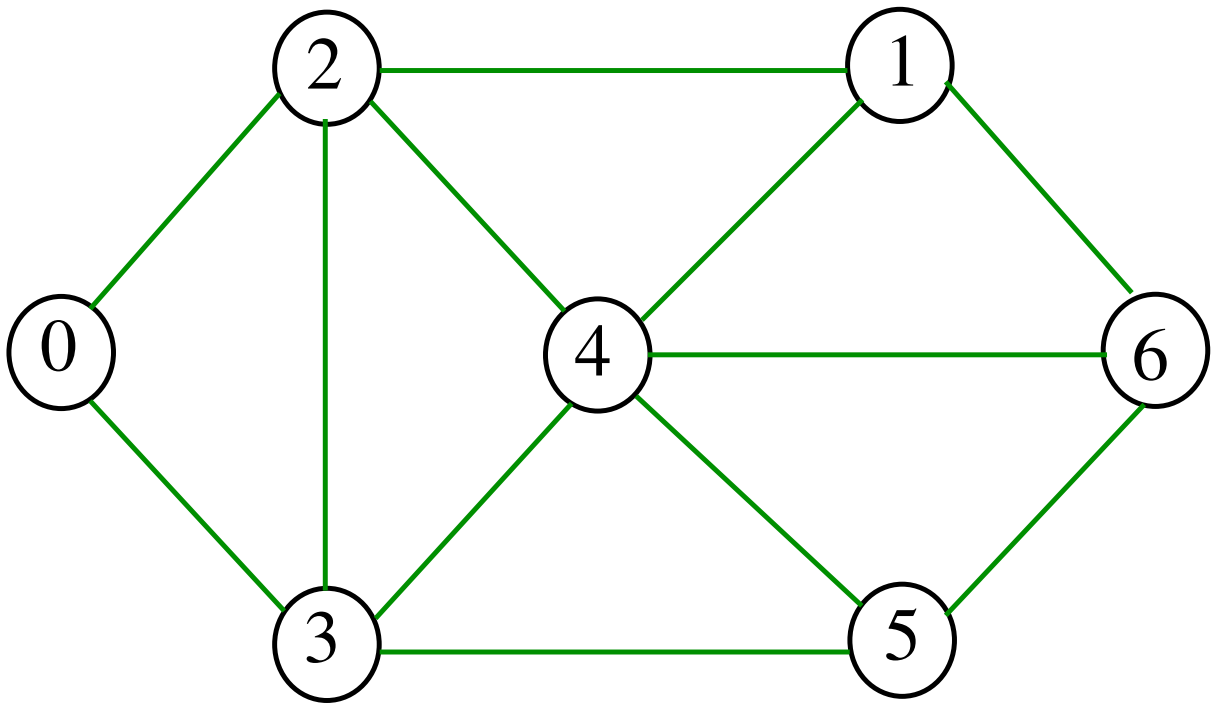
O problema SPT em digrafos com ciclos negativos é NP-difícil.

Árvores geradoras de grafos

Subárvores

Uma **subárvore** de um grafo G é qualquer árvore T que seja subgrafo de G

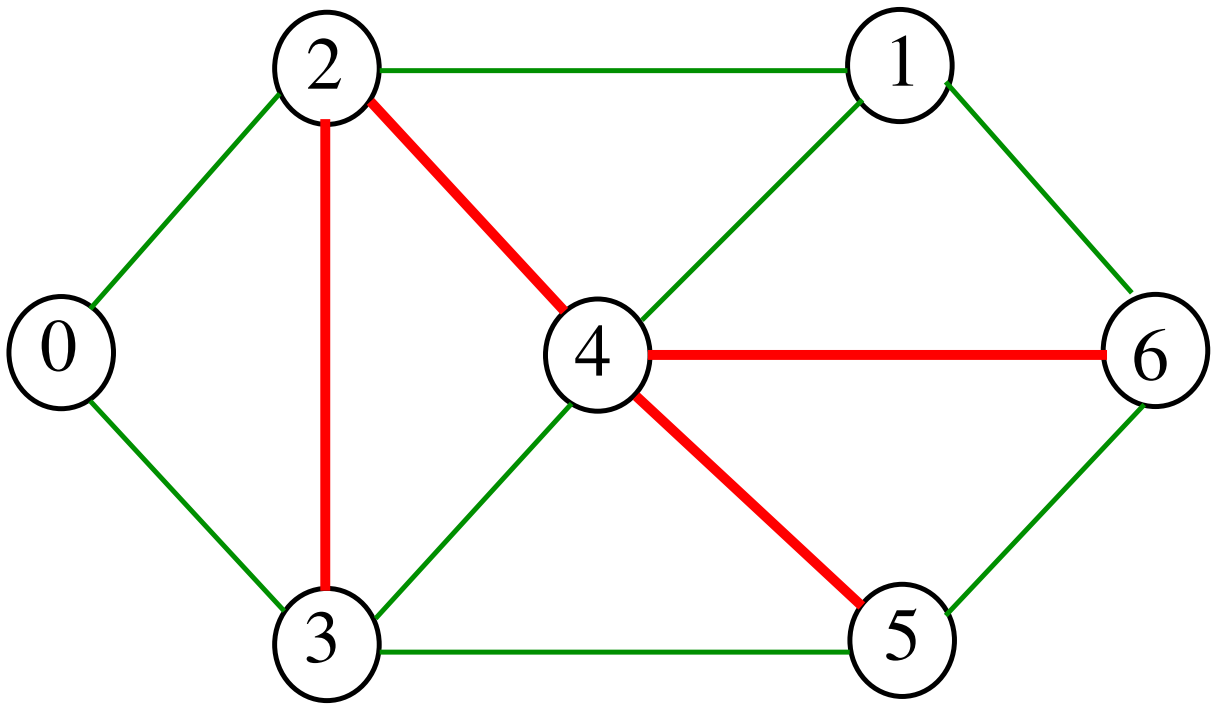
Exemplo:



Subárvores

Uma **subárvore** de um grafo G é qualquer árvore T que seja subgrafo de G

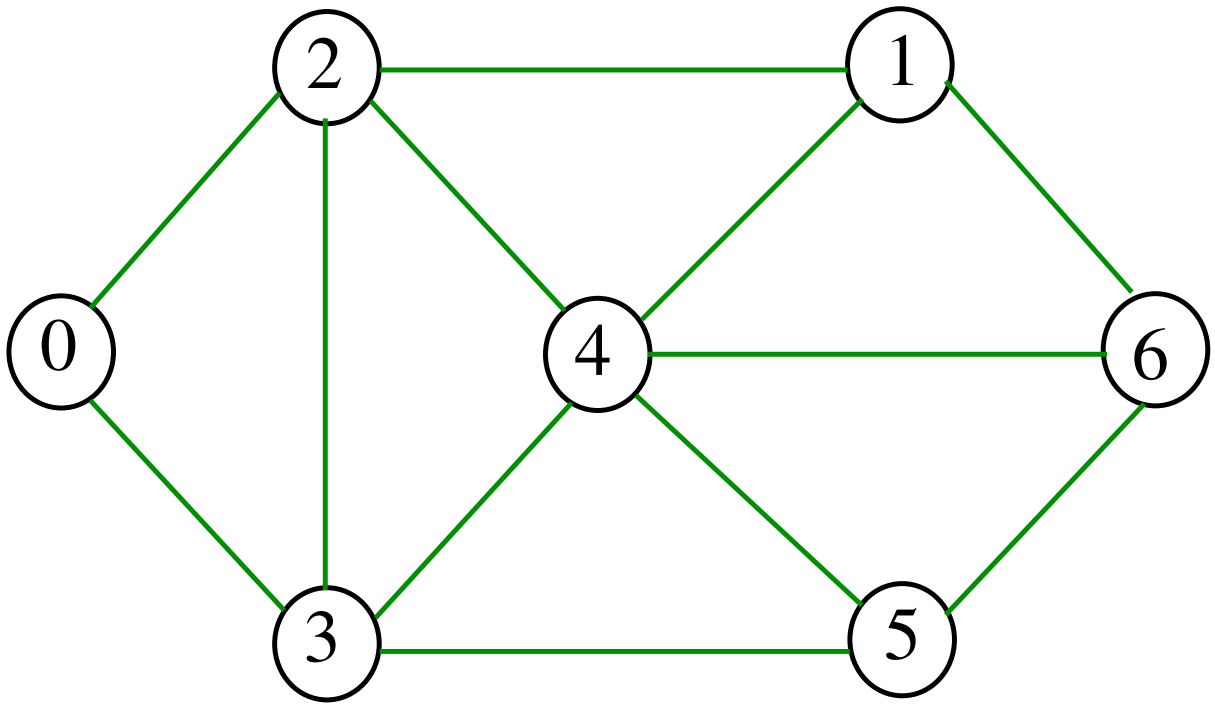
Exemplo: as aretas em vermelho formam uma subárvore



Árvores geradoras

Uma **árvore geradora** (= *spanning tree*) de um grafo é qualquer subárvore que contenha **todos** os vértices

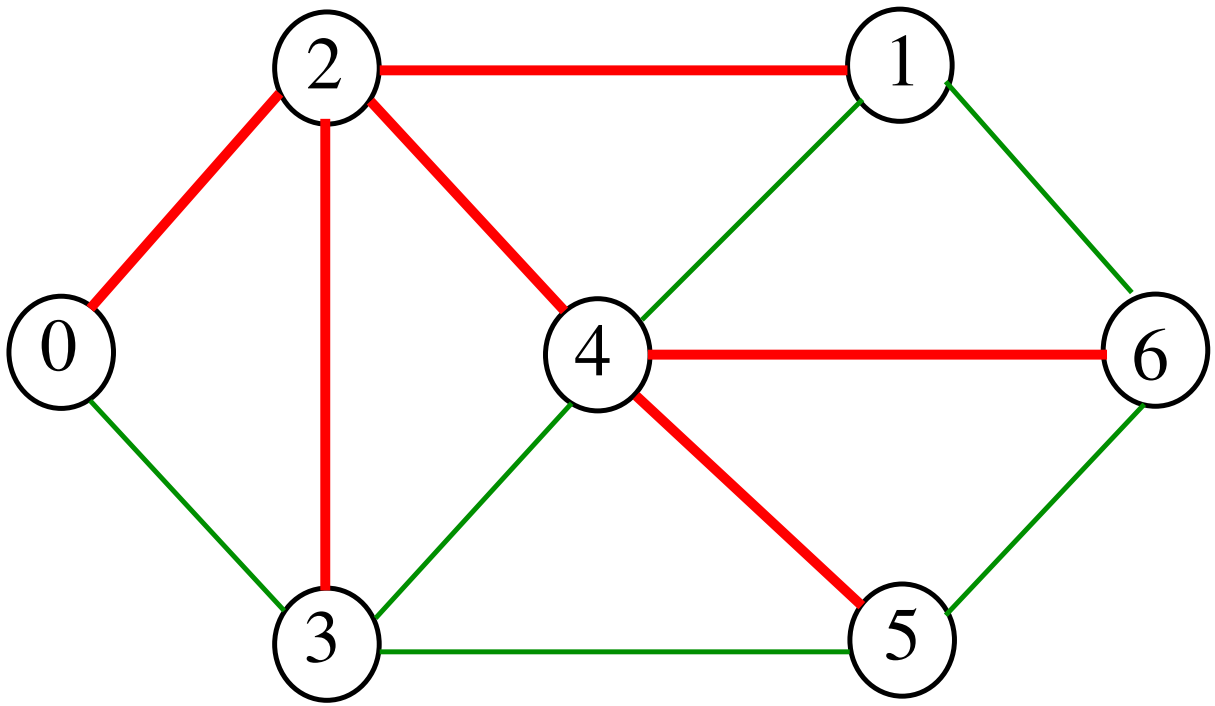
Exemplo:



Árvores geradoras

Uma **árvore geradora** (= *spanning tree*) de um grafo é qualquer subárvore que contenha **todos** os vértices

Exemplo: as aretas em vermelho formam uma árvore geradora

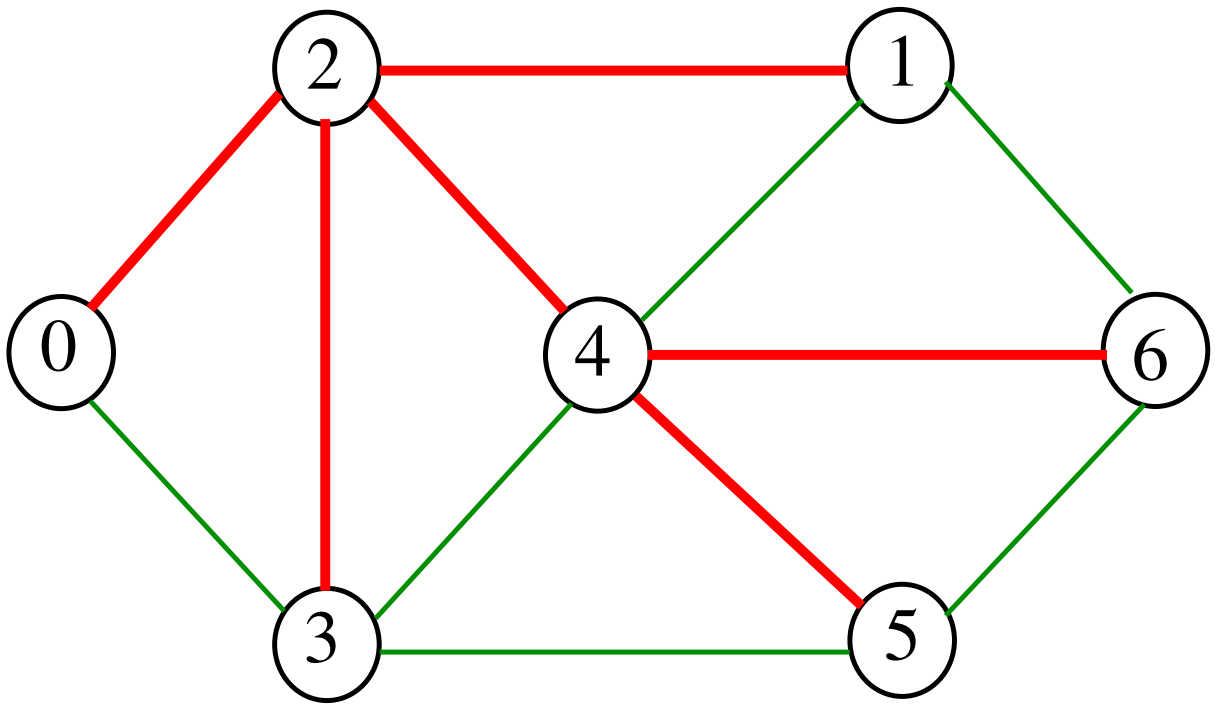


Árvores geradoras

Somente grafos conexos têm árvores geradoras

Todo grafo conexo tem uma árvore geradora

Exemplo:



Algoritmos que calculam árvores geradoras

É fácil calcular uma árvore geradora de um grafo conexo:

- a busca em profundidade e
- a busca em largura

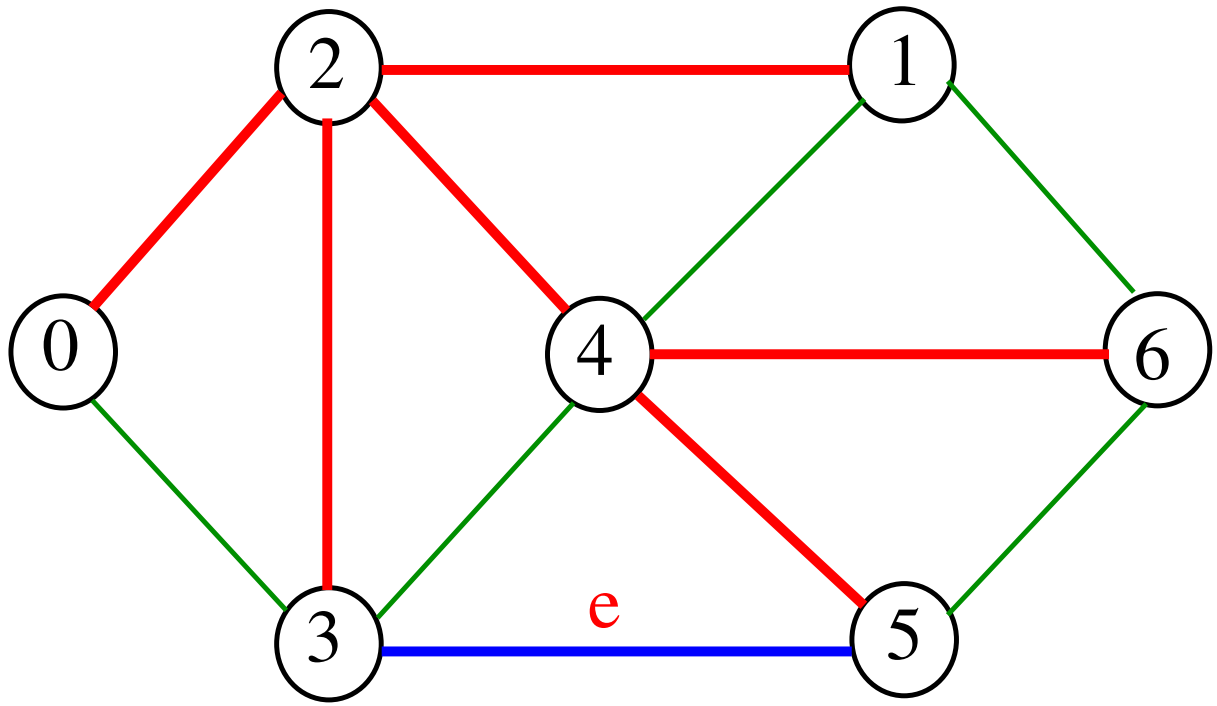
fazem isso.

Qualquer das duas buscas calcula uma arborescência que contém um dos arcos de cada aresta de uma árvore geradora do grafo

Primeira propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G Para qualquer aresta e de G que não esteja em T, T+e tem um **único ciclo** não-trivial

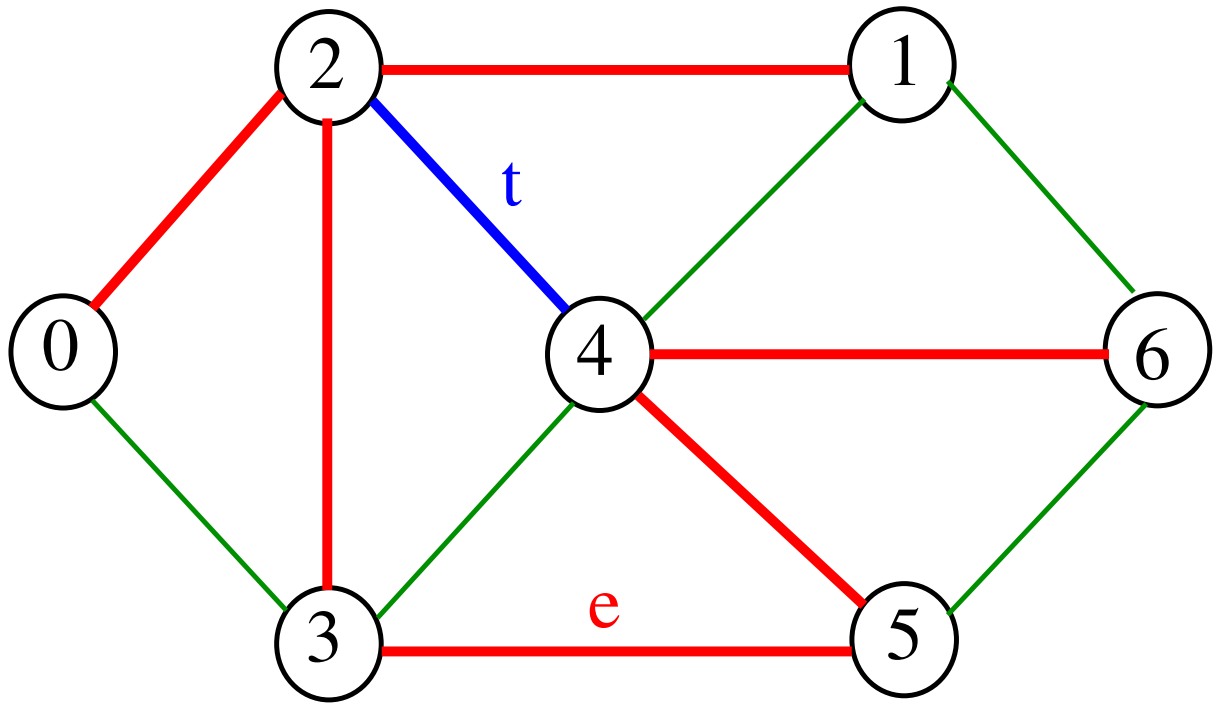
Exemplo: T+e



Primeira propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G Para qualquer aresta t desse ciclo, T+e-t uma **árvore geradora**

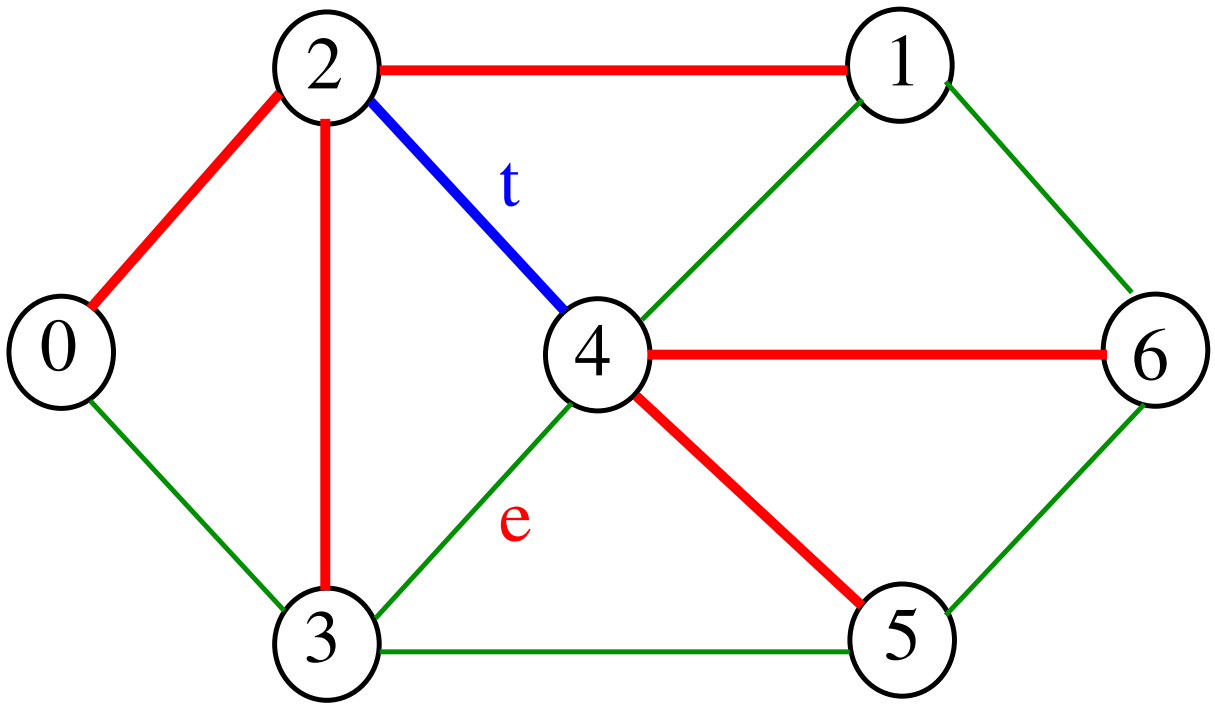
Exemplo: T+e-t



Segunda propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G
Para qualquer aresta t de T e qualquer aresta e que atravesse o corte determinado por T-t, o grafo T-t+e é uma árvore geradora

Exemplo: T-t



Segunda propriedade da troca de arestas

Seja T uma árvore geradora de um grafo G
Para qualquer aresta t de T e qualquer aresta e que atravesse o corte determinado por T-t, o grafo T-t+e é uma árvore geradora

Exemplo: T-t+e