

Melhores momentos

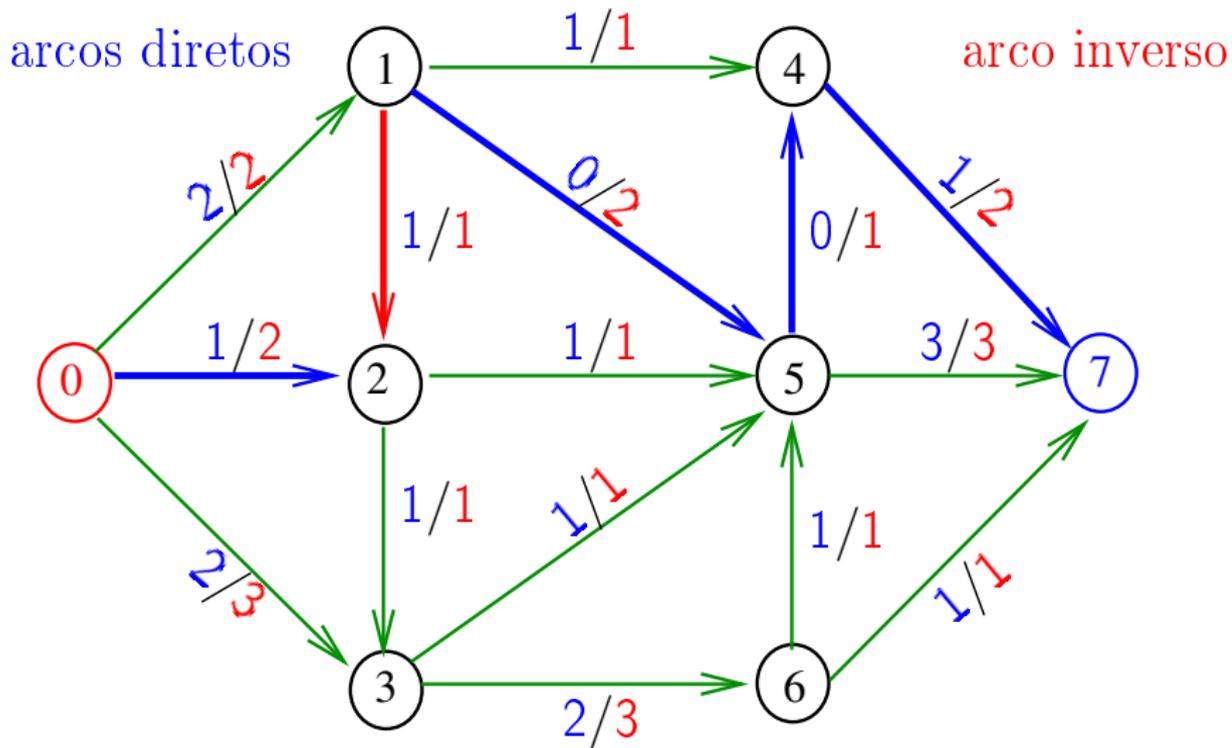
AULA 24

Caminho de aumento

Um **caminho de aumento** (= *augmenting path*) é um pseudo-caminho do vértice inicial ao final onde:

- ▶ os **arcos diretos** não estão cheios e
- ▶ os **arcos inversos** não estão vazios.

Exemplo

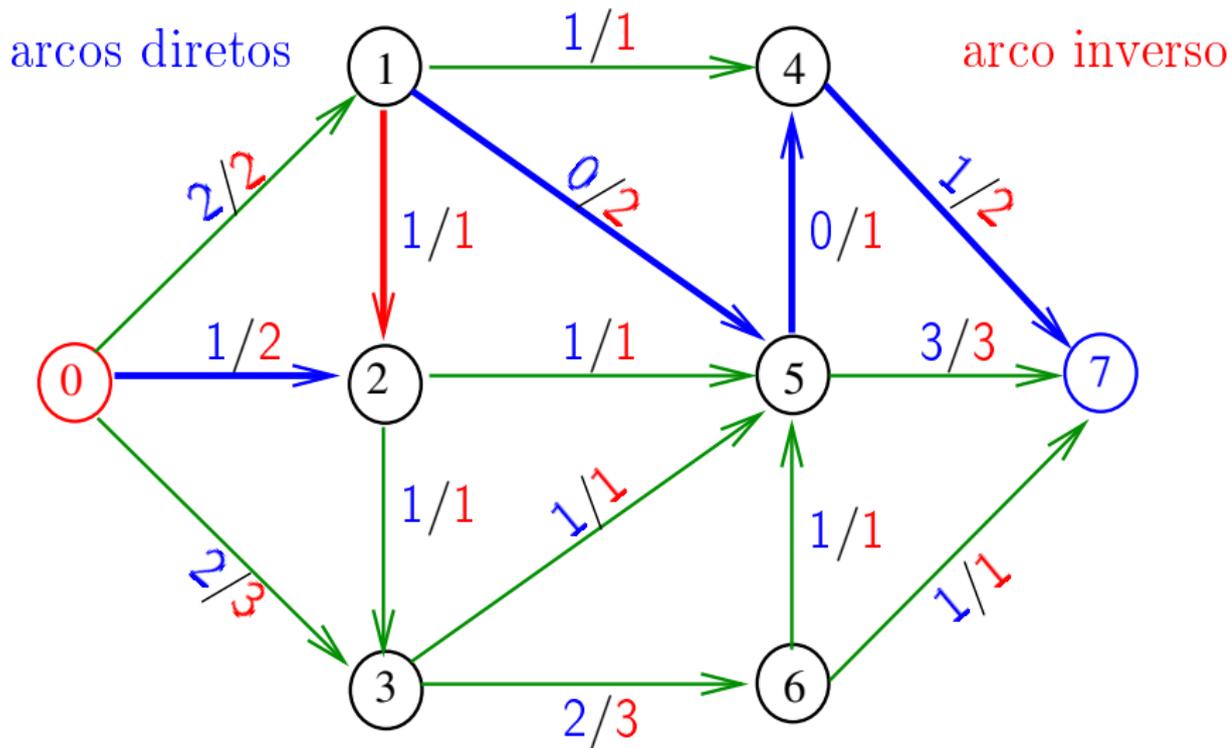


Enviar fluxo através de caminhos de aumento

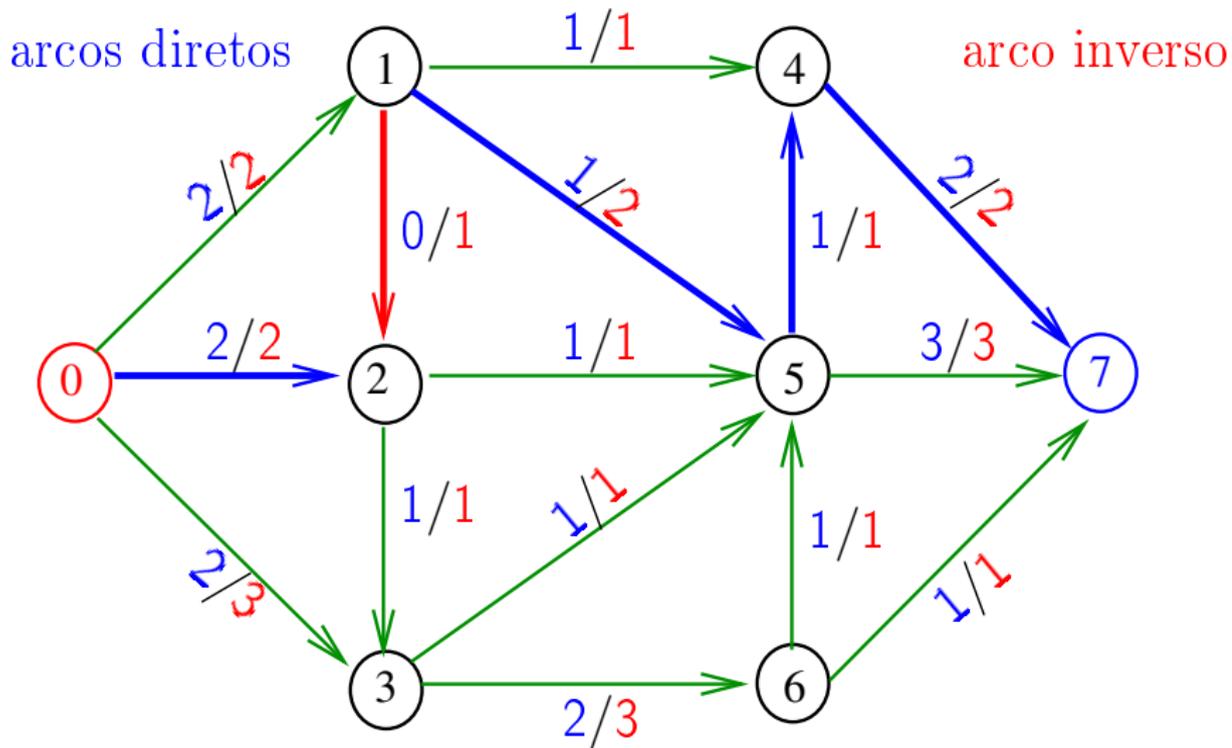
A operação de **enviar** d unidades de fluxo ao longo de um caminho de aumento consiste de:

- ▶ para cada **arco direto**, some d ao fluxo
- ▶ para cada **arco inverso**, subtraia d do fluxo.

Exemplo



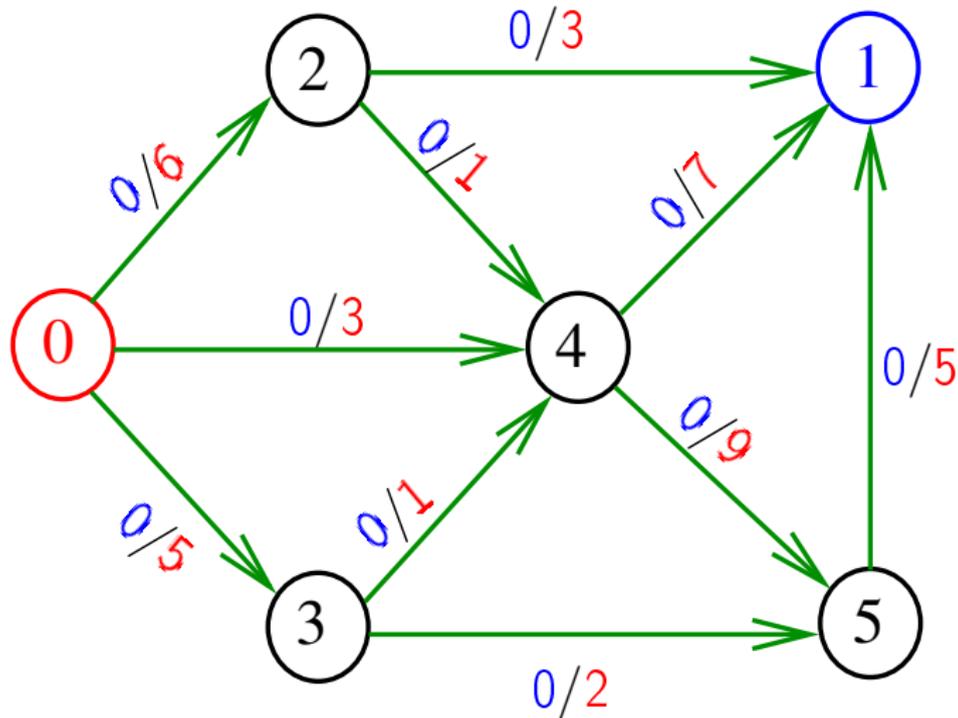
Exemplo



Método de Ford-Fulkerson

$\text{int}(f) = 0$

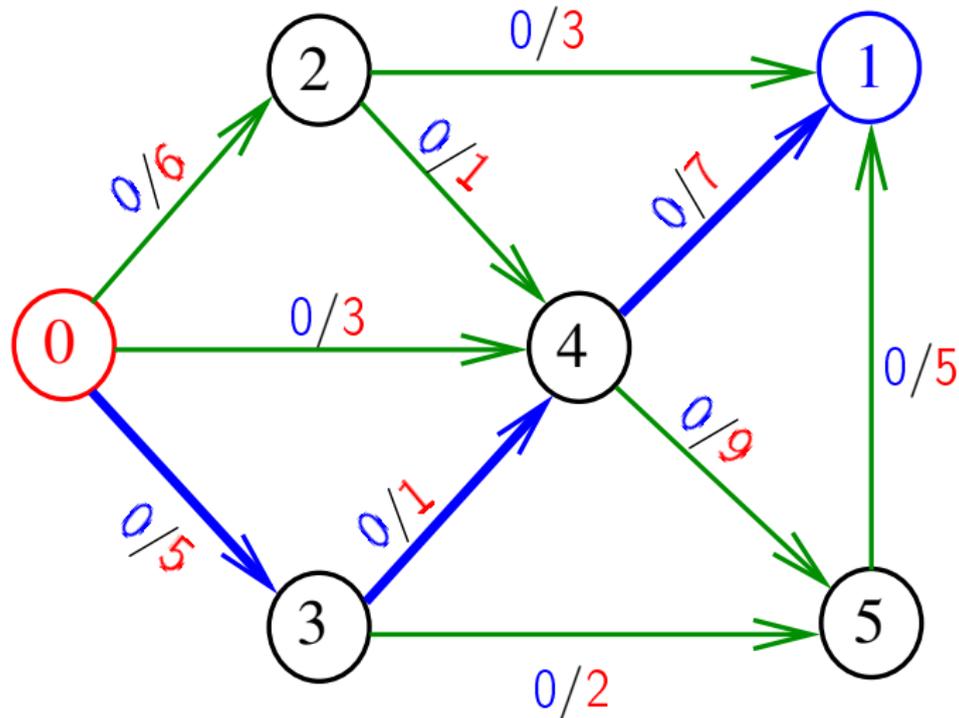
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 0$

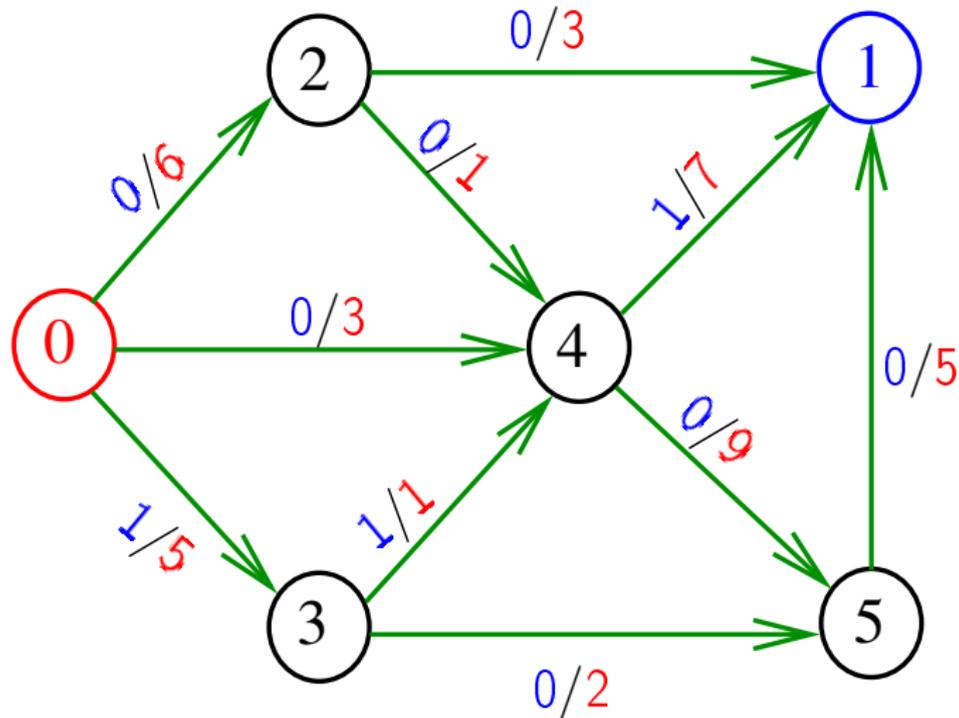
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 1$

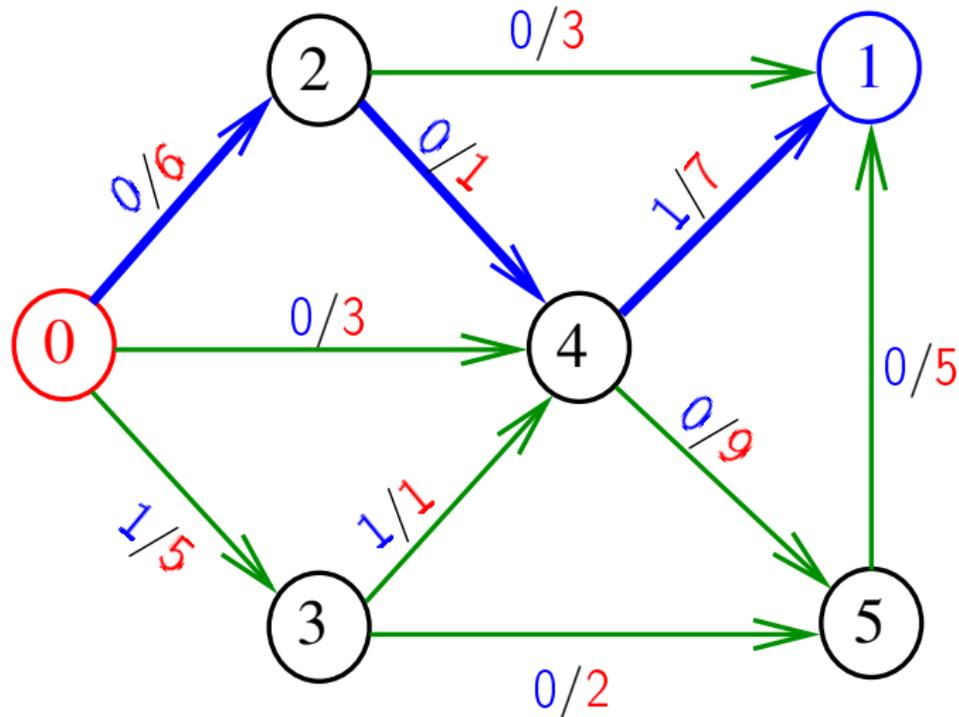
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 1$

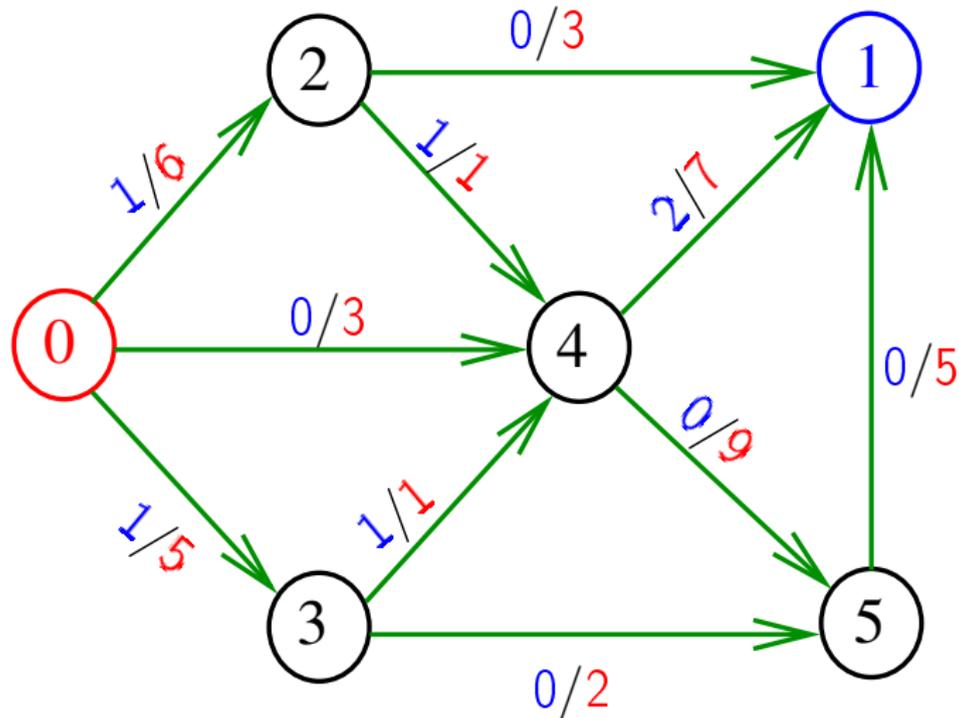
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 2$

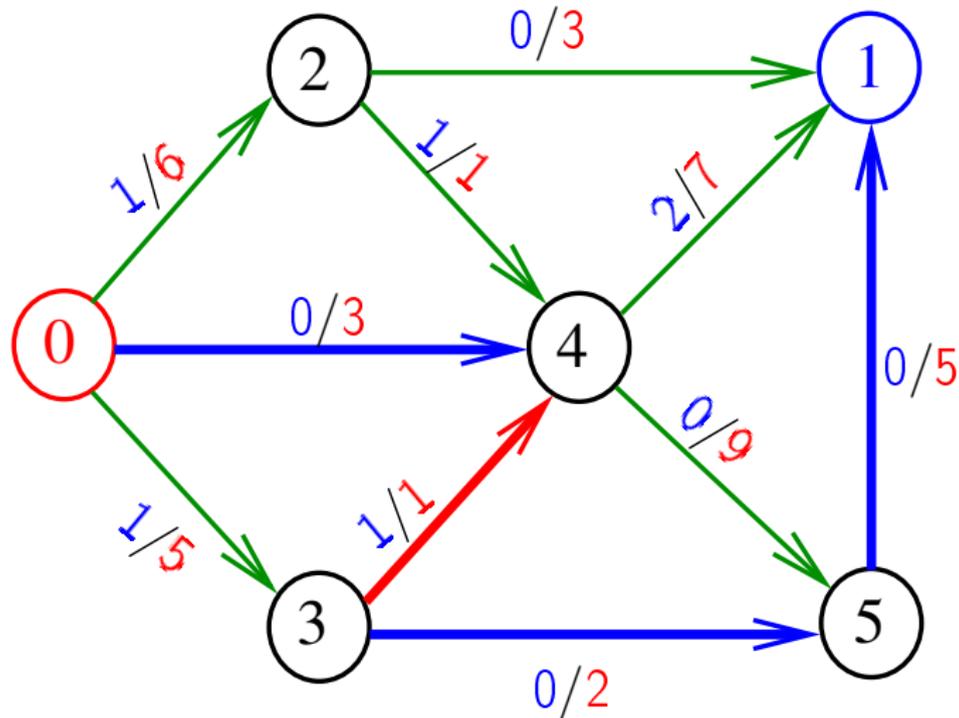
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 2$

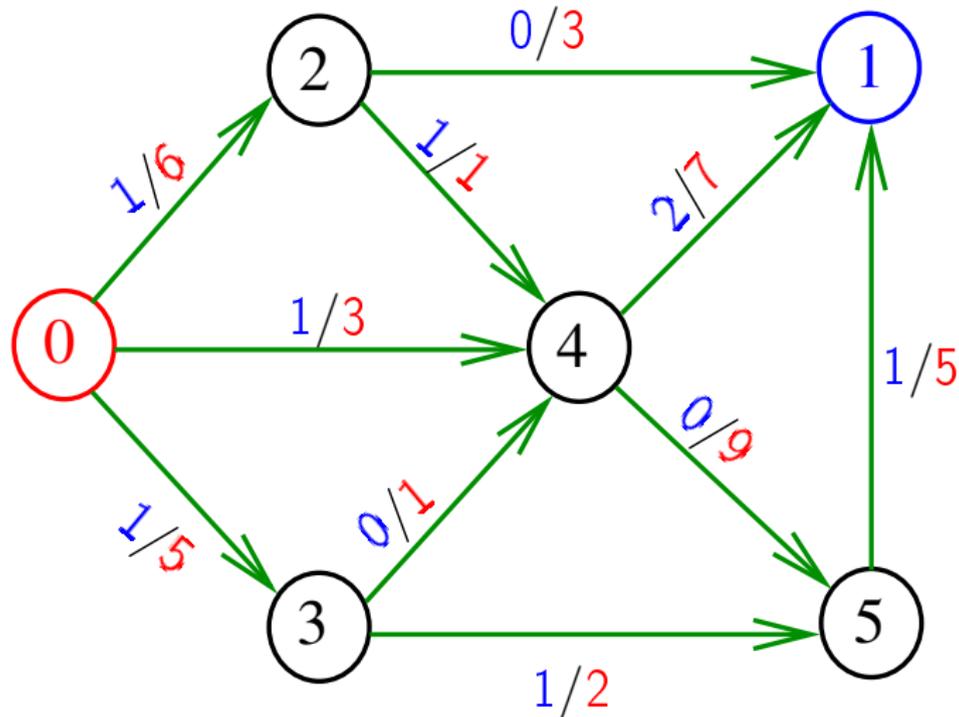
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 3$

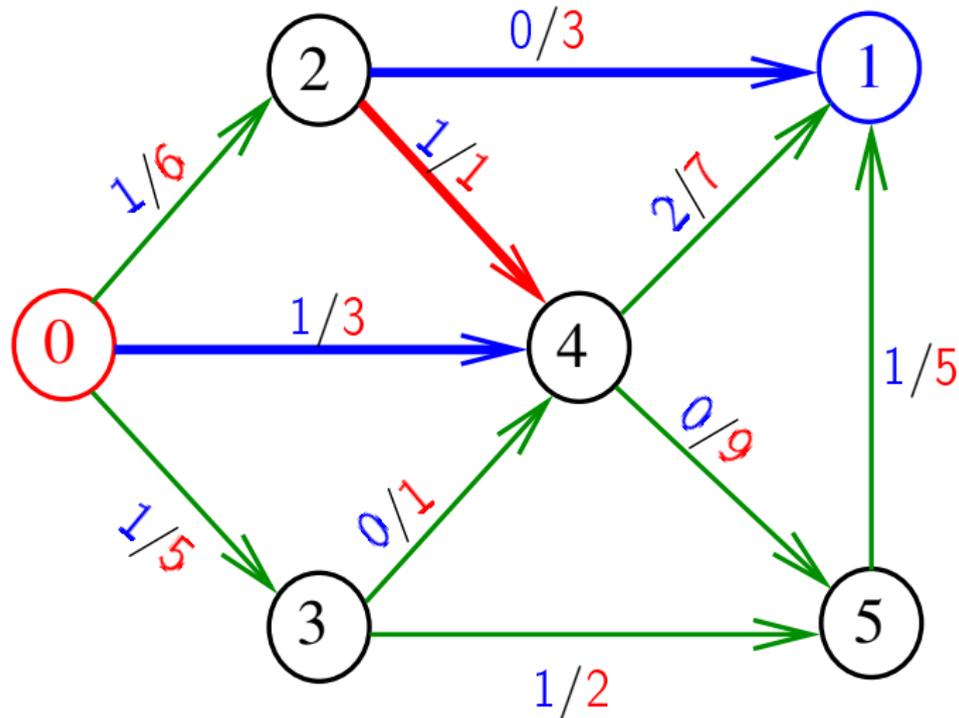
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 3$

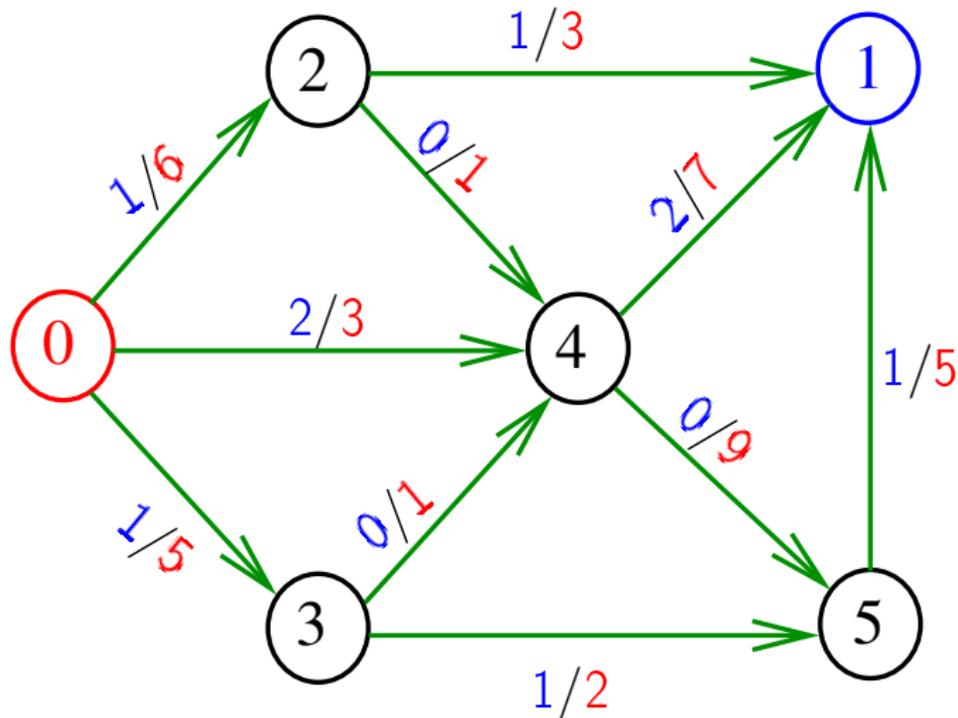
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 4$

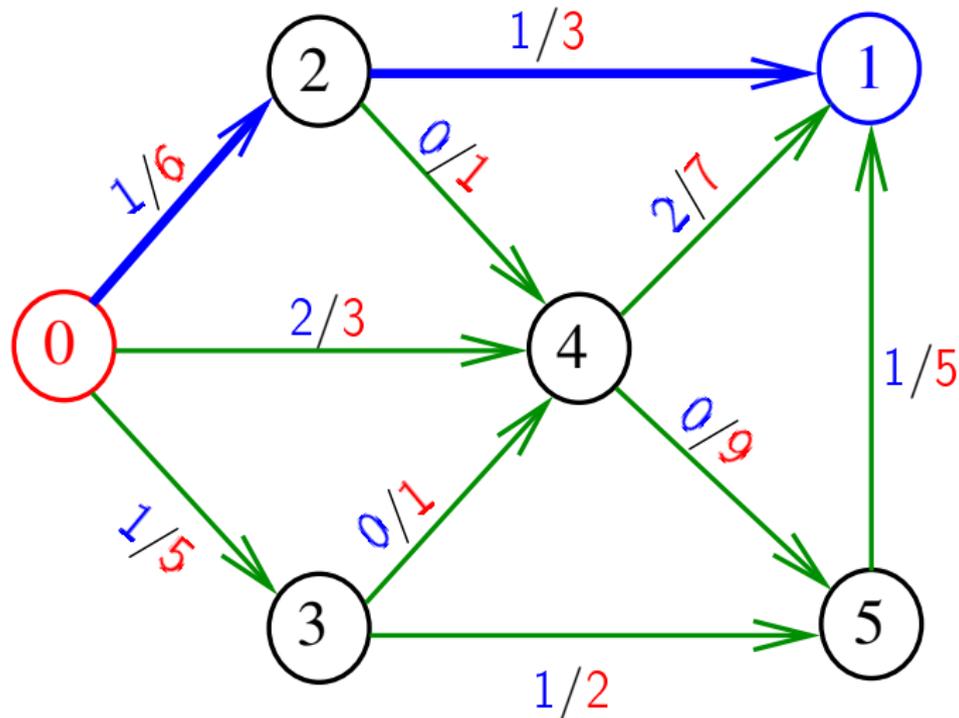
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 4$

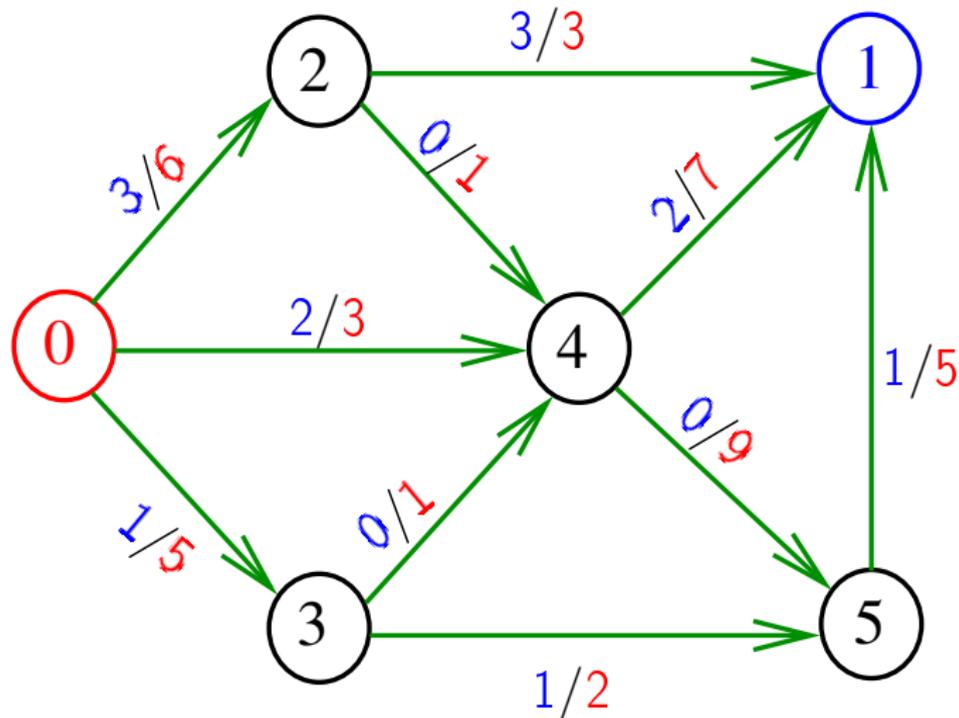
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 6$

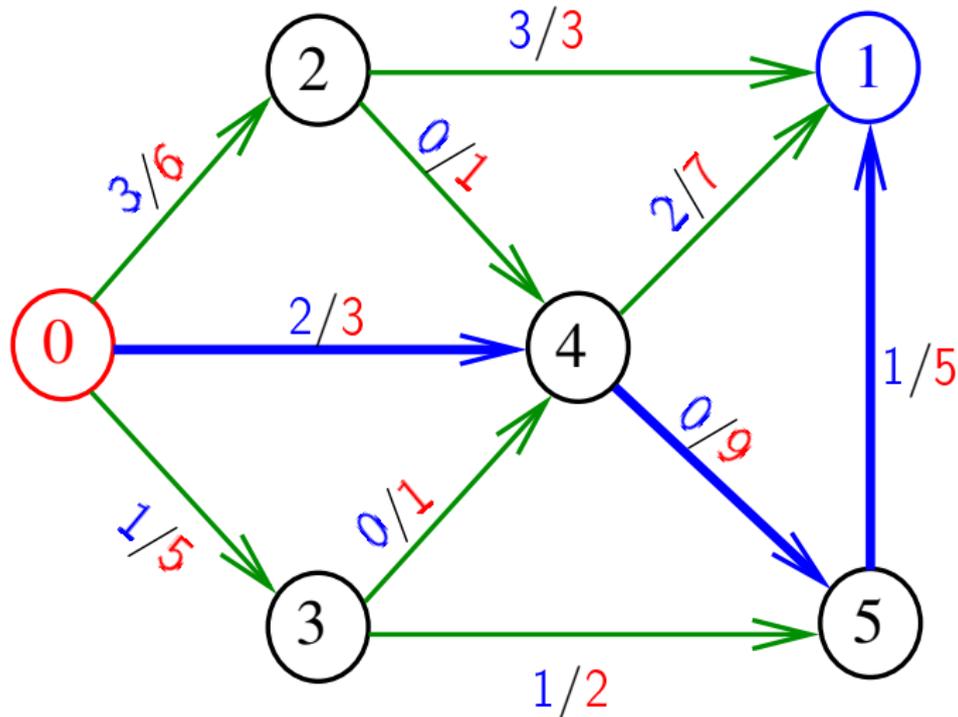
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 6$

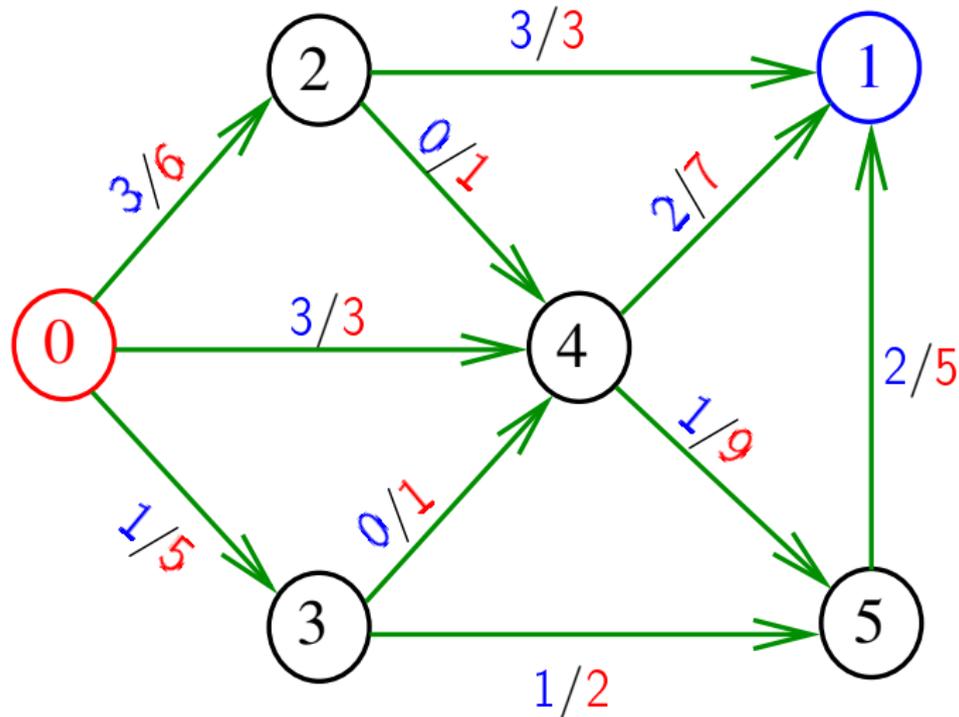
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 7$

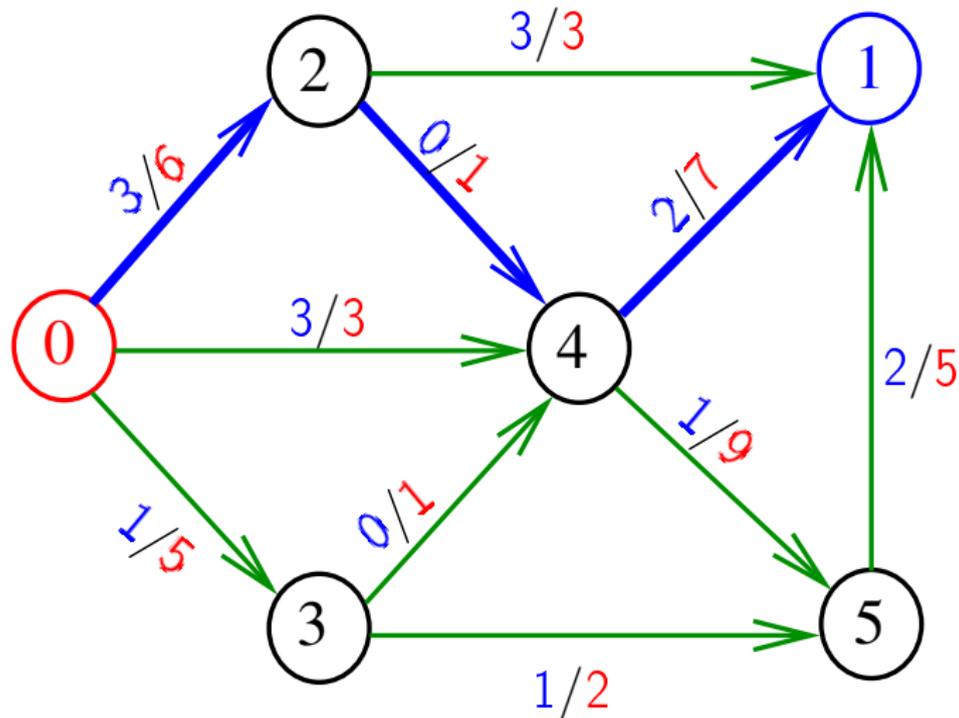
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 7$

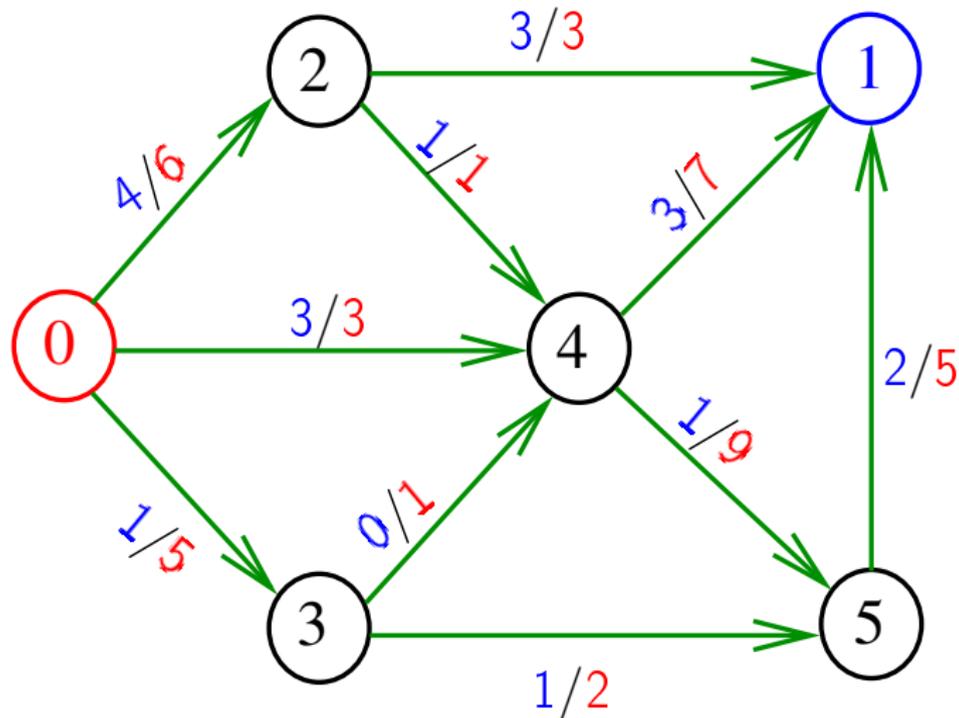
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 8$

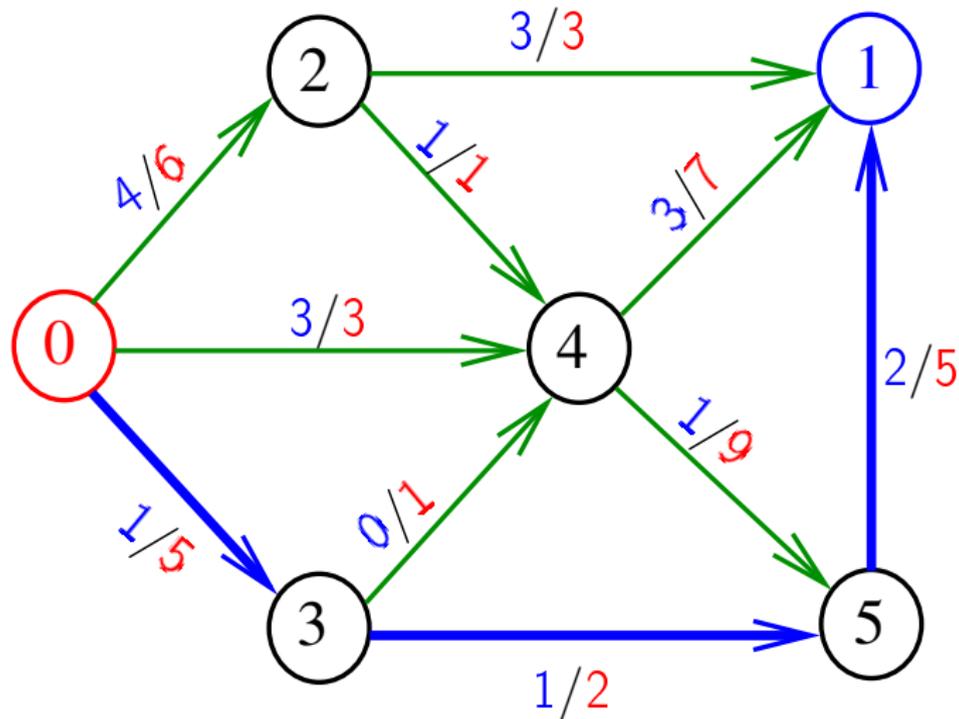
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 8$

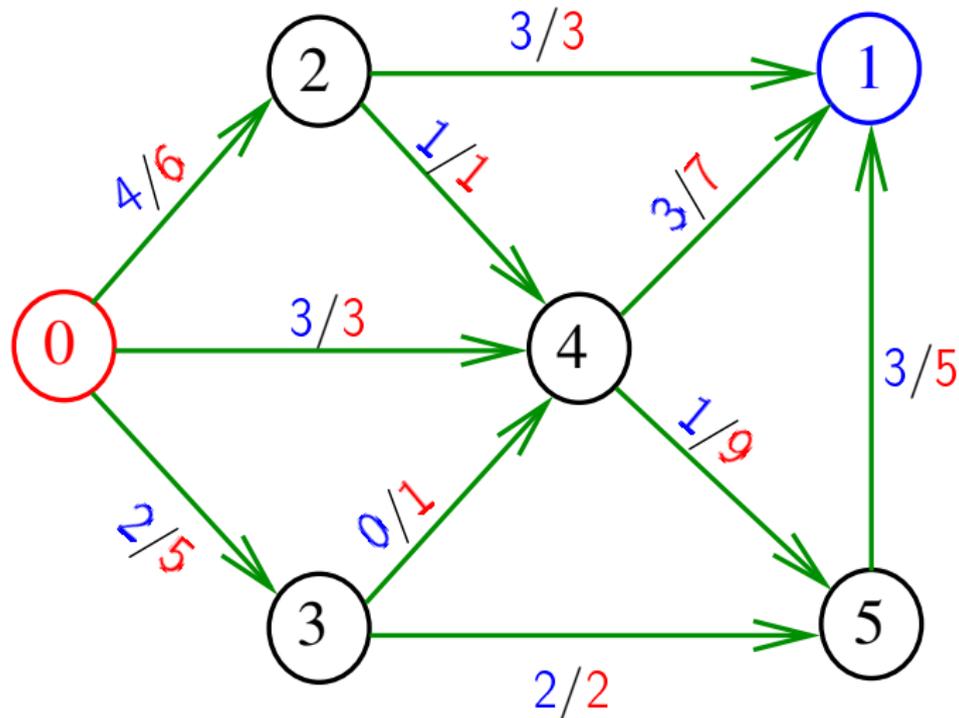
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 9$

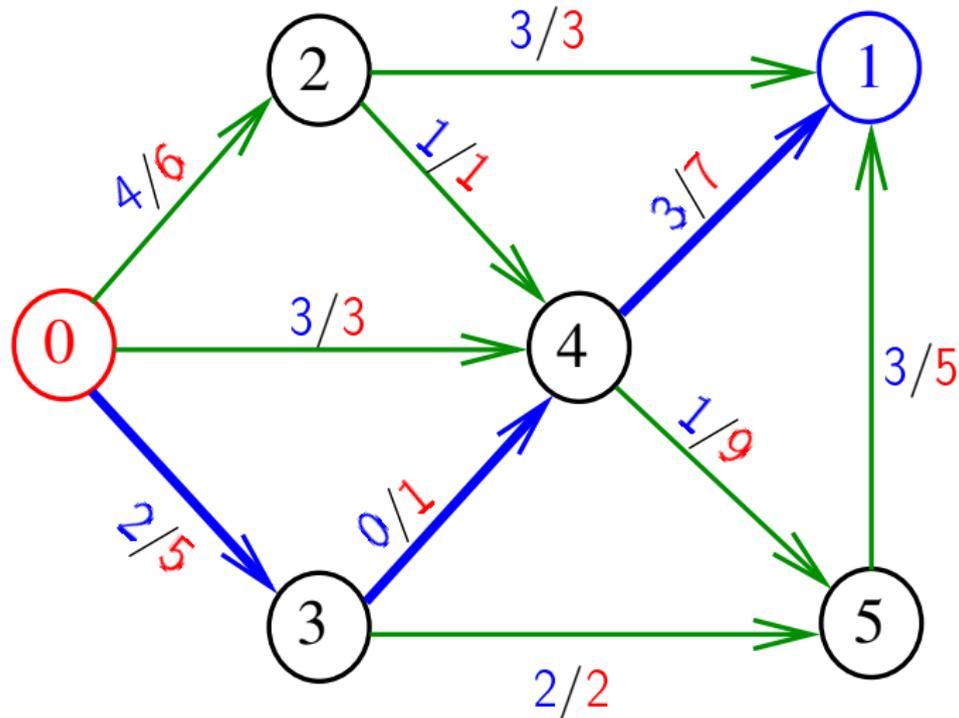
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 9$

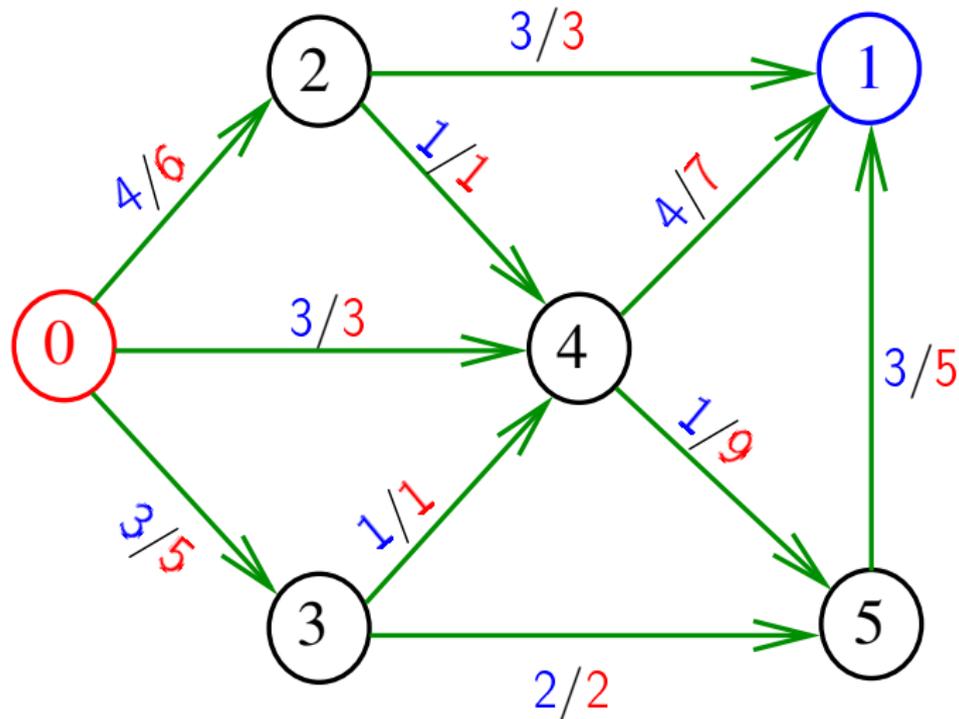
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 10$

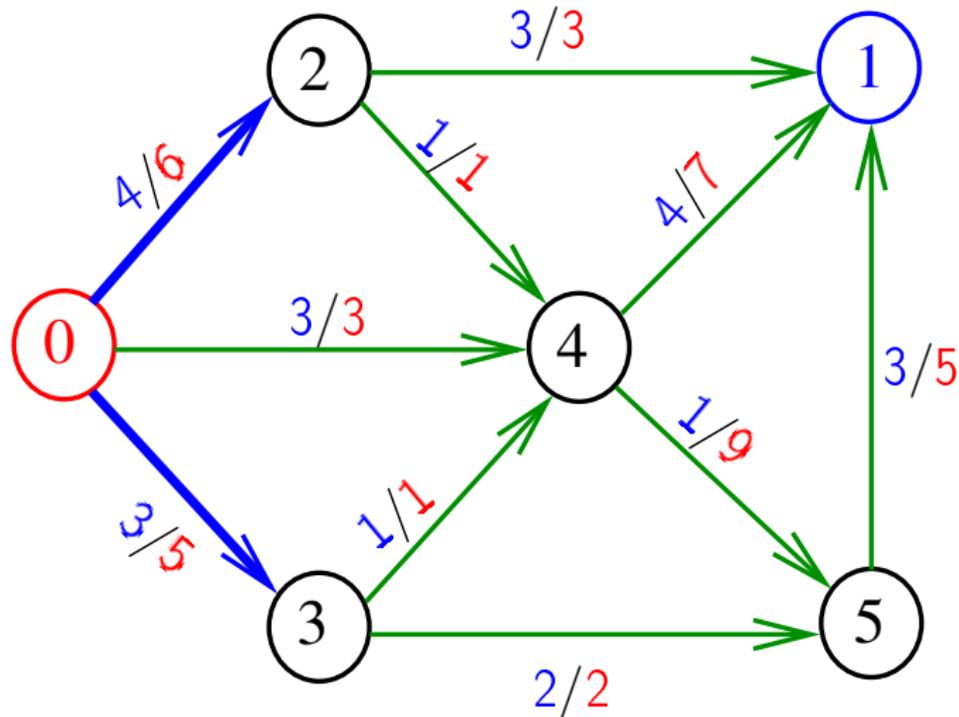
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 10$

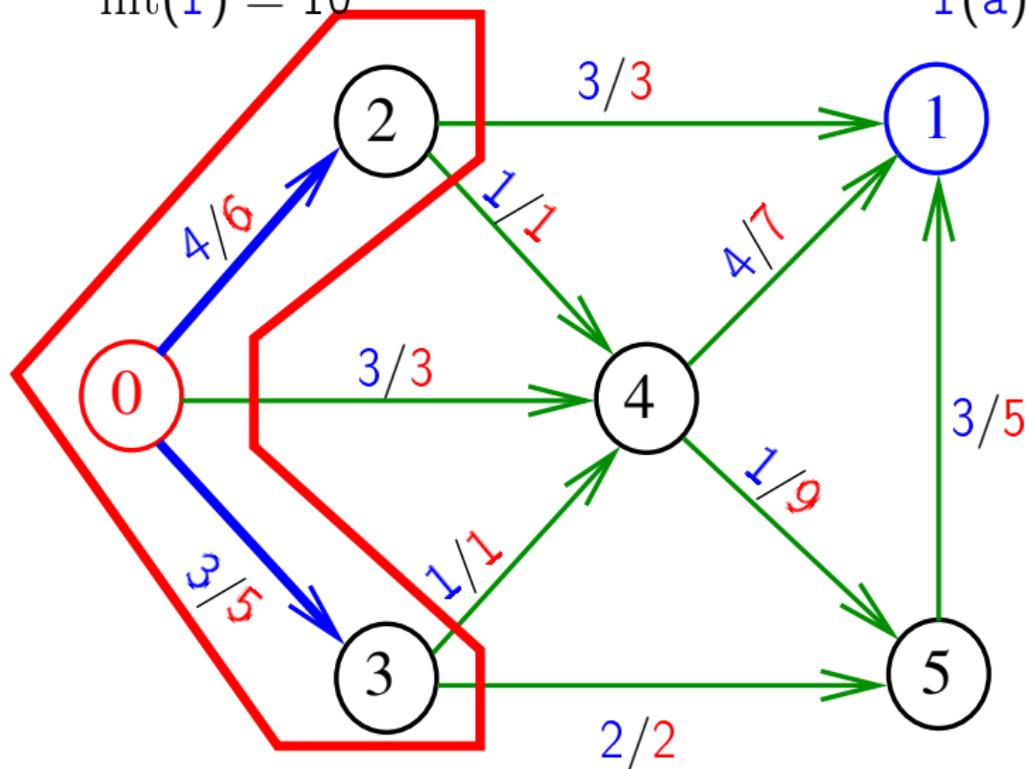
$f(a)/c(a)$



Método de Ford-Fulkerson

$\text{int}(f) = 10$

$f(a)/c(a)$



Método dos caminhos de aumento

O método é iterativo. Cada iteração começa com uma fluxo f que respeita as capacidades.

No início da primeira iteração f é o fluxo nulo.

Cada iteração consiste em:

Caso 1: **não existe** um caminho de aumento

Devolva f e pare

Caso 2: **existe** uma caminho de aumento

Seja d a capacidade residual de um caminho de aumento P

Seja f' o fluxo obtido ao enviarmos d unidades de fluxo ao longo de P

Comece nova iteração com f' no papel de f

Relações invariantes

No início de cada iteração temos que:

(i0) f é inteiro;

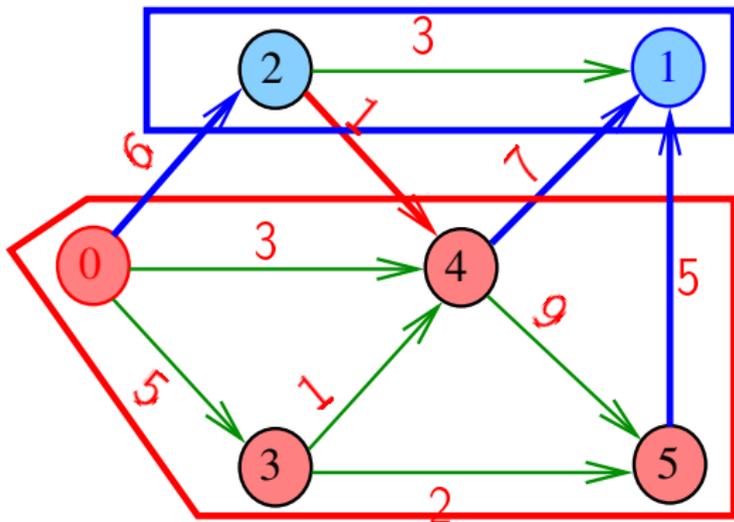
(i1) f é um fluxo;

(i2) f respeita c .

Capacidade de um corte

Numa rede capacitada, a capacidade de um corte (S,T) é a soma das capacidades dos **arcos diretos** do corte.

Exemplo: corte de capacidade 18

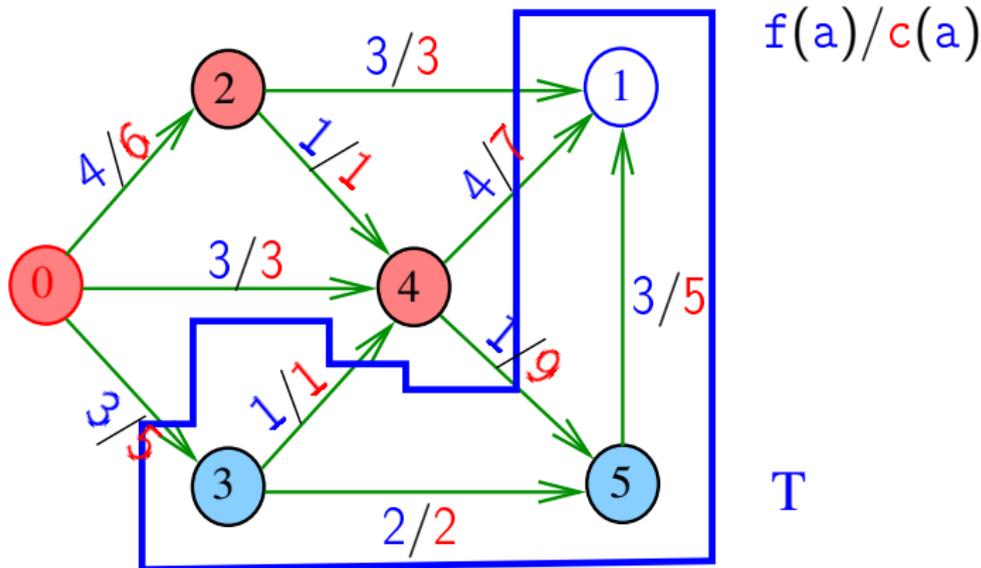


Lema da dualidade

Se f é um fluxo que respeita c e (S, T) é um corte então

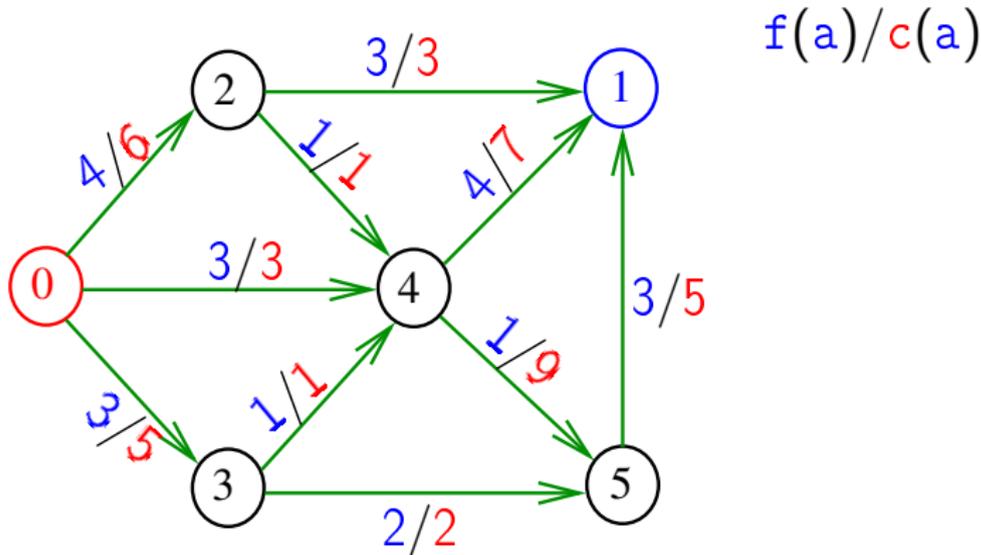
intensidade de $f \leq$ capacidade de (S, T) .

Exemplo: $\text{int}(f) = 10 \leq 24 = c(S, T)$.



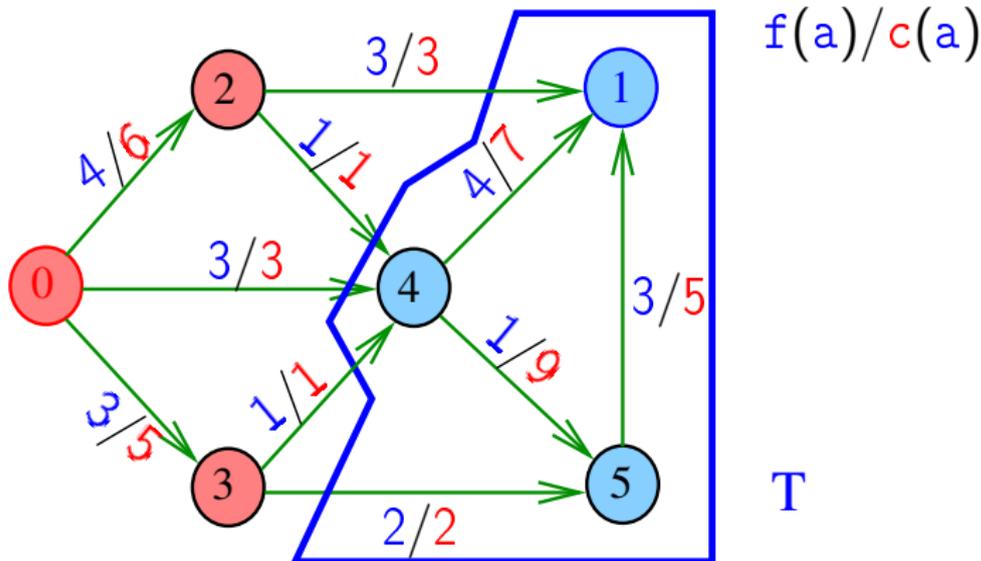
Consequência

Se f é um fluxo que respeita c e (S, T) é um corte tais que intensidade de $f =$ capacidade de (S, T) .
então f é um fluxo de **máximo** e (S, T) é um corte de **capacidade mínima**.



Consequência

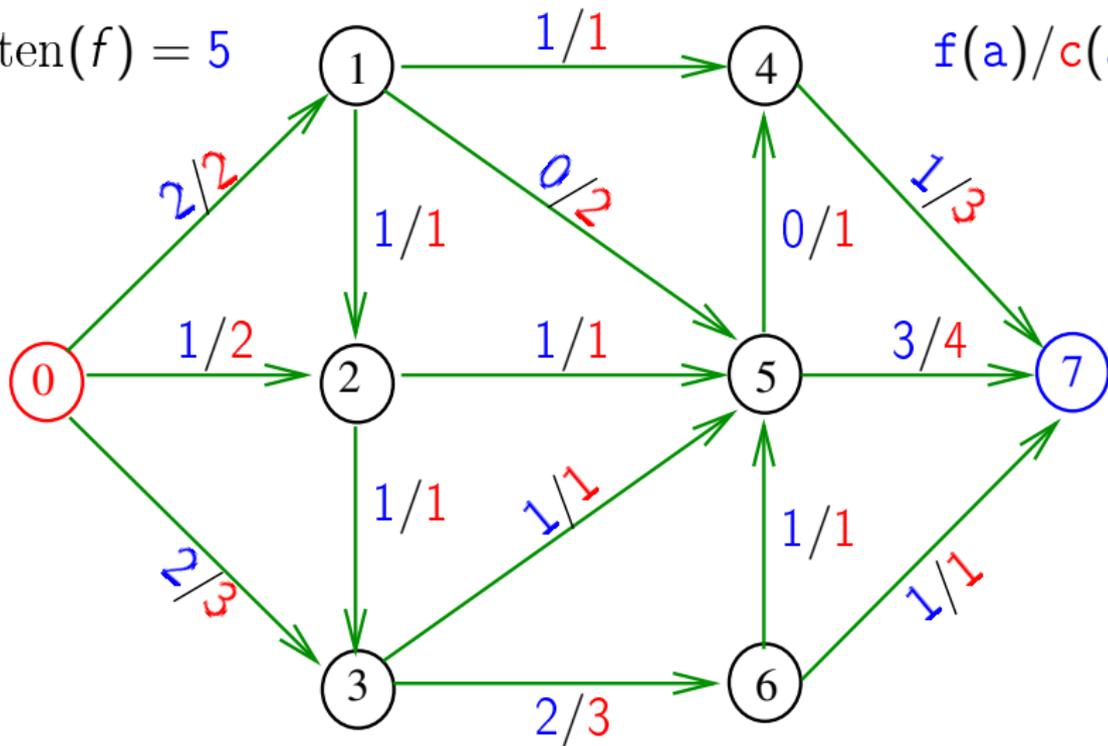
Se f é um fluxo que respeita c e (S, T) é um corte tais que intensidade de $f =$ capacidade de (S, T) .
então f é um fluxo de **máximo** e (S, T) é um corte de **capacidade mínima**.



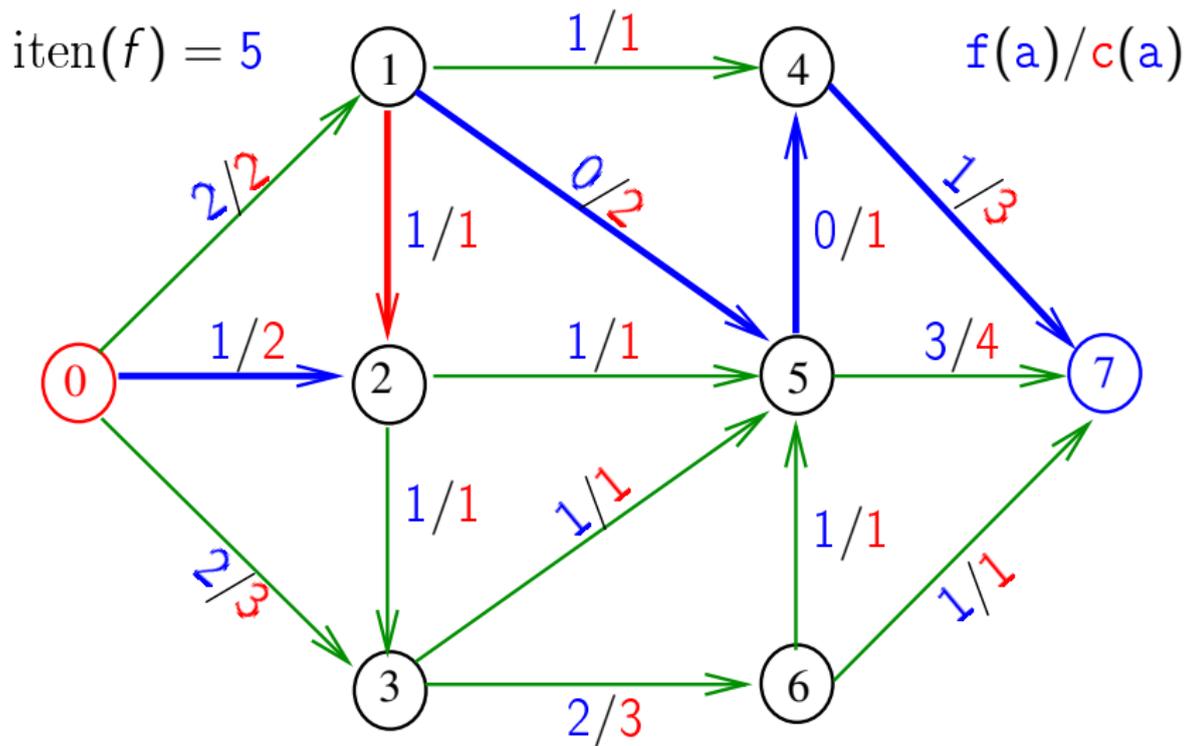
Fluxo é máximo?

$\text{iten}(f) = 5$

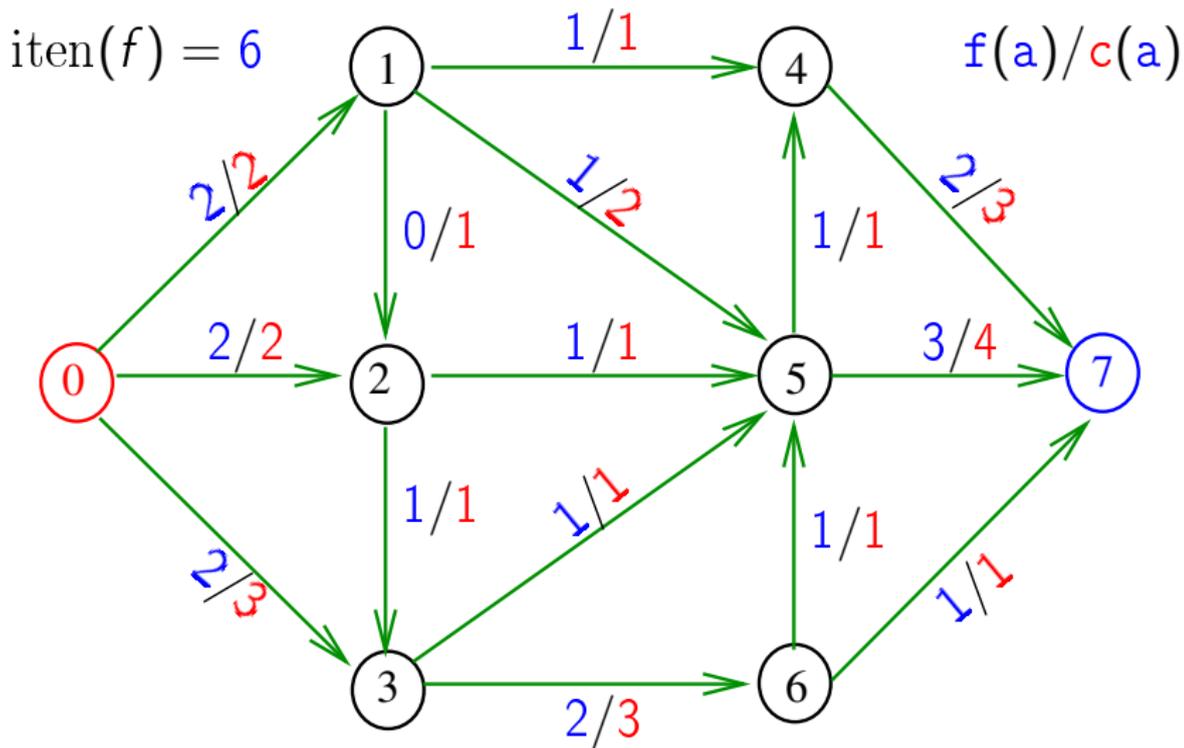
$f(a)/c(a)$



Caminho de aumento



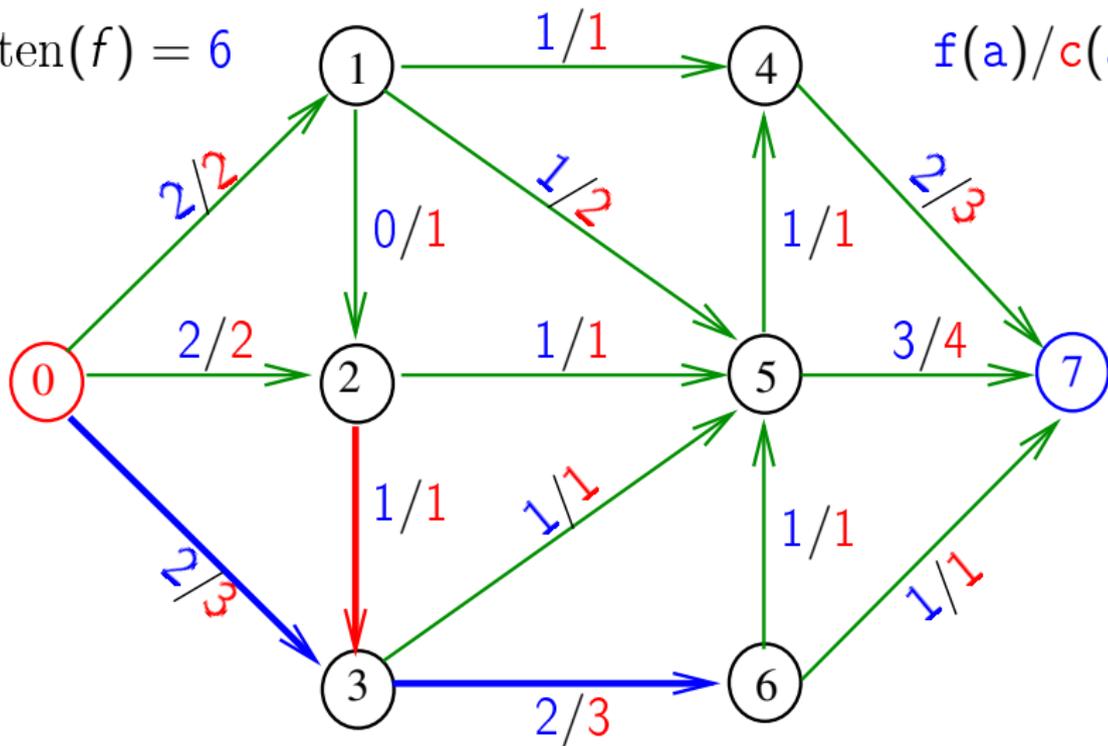
E agora? Fluxo é máximo?



Fluxo é máximo!

$\text{iten}(f) = 6$

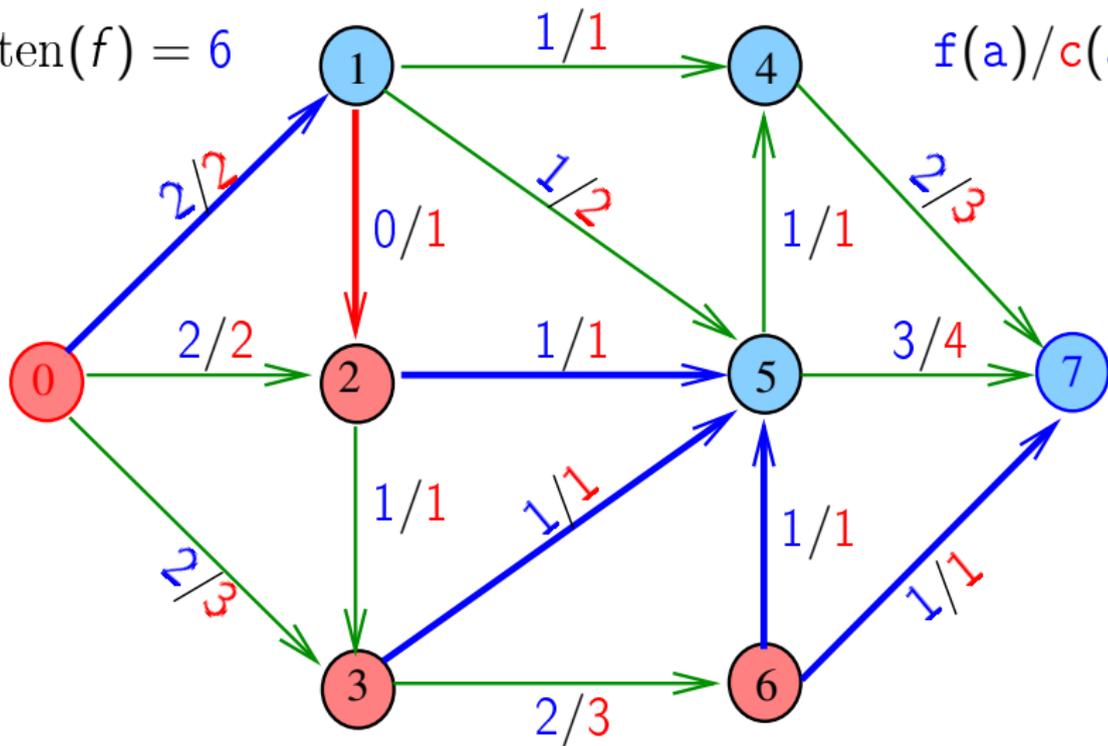
$f(a)/c(a)$



Fluxo é máximo!

$\text{iten}(f) = 6$

$f(a)/c(a)$



Teorema do fluxo máximo e corte mínimo

O teorema foi demonstrado por Ford e Fulkerson e, independentemente, por Kotzig.

Para quaisquer dois vértices s e t em uma rede capacidade com função-capacidade c tem-se que

$$\begin{aligned} & \max\{\text{int}(f) : f \text{ é fluxo que respeita } c\} \\ &= \min\{c(S, T) : (S, T) \text{ é um corte}\}. \end{aligned}$$

Teorema do fluxo máximo e corte mínimo

O teorema foi demonstrado por Ford e Fulkerson e, independentemente, por Kotzig.

Em qualquer rede capacitada, a intensidade de um **fluxo máximo** é igual à capacidade de um **corte mínimo**.

AULA 25

Estrutura de dados para redes de fluxo

S 22.1

Listas de adjacência

Redes serão representadas por **listas de adjacência**.

Cada arco $v-w$ será representado por um nó na lista encadeada $adj[v]$.

Além do campo w , esse nó terá os campos

- ▶ cap para armazenar a **capacidade** do arco $v-w$ e
- ▶ $flow$ para armazenar o valor do **fluxo** no arco.
- ▶ dup para armazenar ...

Estrutura node

```
typedef struct node *link;
```

```
struct node {  
    Vertex w;  
    link next;  
    int cap;  
    int flow;  
    link dup;  
};
```

Constructor

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

Flownet

```
struct flownet {  
    int V,A;  
    link *adj;  
    Vertex s,t;  
};  
  
typedef struct flownet *Flownet;
```

Flowinit

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

FLOWinsert

Inserir um arco $v-w$, de capacidade cap e fluxo nulo na rede G .

void

```
FLOWinsertA (Flownet G, Vertex v, Vertex w, int
cap) {
    if (v == w || cap < 0) return;
    G->adj[v] = NEW(w, cap, 0, G->adj[v]);
    G->adj[v]->dup = NULL;
    G->A++;
}
```

Redes de fluxo expandidas

É difícil procurar **caminhos de aumento** numa rede de fluxo porque esses caminhos podem ter arcos inversos.

Para contornar essa dificuldade, vamos introduzir o conceito de **rede de fluxo expandida**.

Para cada arco $v-w$, acrescente à rede um arco $w-v$.

Diremos que os novos arcos são **artificiais** e os antigos são **originais**

A **capacidade** arco artificial $w-v$ será o **negativo** da capacidade do correspondente **arco original** $v-w$.

Redes expandidas

O **fluxo** em cada **arco artificial** será o negativo do fluxo no correspondente **arco original**.

O campo `dup` nos nós será usado para apontar de um **arco original** para o correspondente **arco artificial** e vice-versa.

Para cada o **arco artificial** teremos

$$\text{cap} \leq \text{flow} \leq 0$$

e para cada o **arco original** teremos

$$0 \leq \text{flow} \leq \text{cap}$$

Expand

Função que transforma uma rede de fluxo na correspondente rede de fluxo expandida:

```
void Expand (Flownet G) {
```

Expand

Função que transforma uma rede de fluxo na correspondente rede de fluxo expandida:

```
void Expand (Flownet G) {  
  Vertex v, w;  
  int cap, flow;  
  link po, pa;  
  for (v = 0; v < G->V; v++)  
    for(po=G->adj[v]; po!=NULL; po=po->next)  
      po->dup = NULL;
```

```

for (v = 0; v < G->V; v++)
    for(po=G->adj[v]; po!=NULL; po=po->next)
        if (po->dup== NULL) {
            w = po->w;
            cap = po->cap;
            flow = po->flow;
            G->adj[w] = pa=
                NEW(v, -cap, -flow, G->adj[w]);
            po->dup= pa;
            pa->dup= po;
        }
}

```

flowV

`flowV` calcula o saldo de fluxo no vértice `v` de uma rede de fluxo expandida `G`.

```
int flowV (Flownet G, Vertex v) {
```

flowV

`flowV` calcula o saldo de fluxo no vértice `v` de uma rede de fluxo expandida `G`.

```
int flowV (Flownet G, Vertex v) {  
    link p;  
    int x = 0;  
    for (p = G->adj[v]; p != NULL; p = p->next)  
        x += p->flow;  
    return x;  
}
```

A intensidade do fluxo é `flowV(G, G->s)`.

Rede expandida e capacidades residuais

Um caminho de **s** a **t** na rede de fluxo expandida corresponde a um caminho de aumento na rede de fluxo original se

- ▶ $cap \geq 0$ implica em $flow < cap$ e
- ▶ $cap < 0$ implica em $flow < 0$

para todo arco do caminho.

A capacidade residual de um **arco original** da rede expandida é

$$cap - flow$$

e a capacidade residual de um **arco artificial** é

$$-flow.$$

Algoritmo de fluxo máximo: versão shortest augmenting paths

S 22.2

Camada externa da implementação

Um **caminho de aumento** pode ser representado por um caminho de capacidade residual positiva na rede expandida.

Para encontrar um tal caminho, podemos usar o algoritmo de **busca em largura** como modelo.

Na implementação a seguir, o vetor `parnt` será usado de maneira um pouco diferente: ao percorrer um arco $v-w$ da rede expandida, o código fará

$$\text{parnt}[w] = p,$$

sendo p o endereço do nó na lista `adj[v]` para o qual $p \rightarrow w$ vale w .

O "pai" v de w será então `parnt[w]`.

MaxFlow

Recebe uma rede capacitada (não-expandida) G e calcula um fluxo máximo.

```
void MaxFlow (Flownet G) {
```

MaxFlow

Recebe uma rede capacitada (não-expandida) G e calcula um fluxo máximo.

```
void MaxFlow (Flownet G) {  
    Vertex s = G->s, t = G->t, x;  
    int d; link parnt[maxV];  
    Expand(G);  
    while (1) {  
        d = AugmentingPath(G, parnt);  
        if (d == 0) break;  
        for(x=t; x!=s; x=parnt[x]->dup->w){  
            parnt[x]->flow += d;  
            parnt[x]->dup->flow -= d;  
        }  
    }  
}
```

Shortest augmenting paths

Para encontrar um caminho de aumento que tenha número mínimo de arcos, basta aplicar o algoritmo de **busca em largura** à rede de fluxo expandida.

Esta é uma implementação shortest-augmenting-path da função AugmentingPath.

```
#define ShrtstAugmPath AugmentingPath
```

A macro RC recebe um link p e calcula a capacidade residual do arco da rede de fluxo expandida que vai do vértice p->dup->w ao vértice p->w.

```
#define RC(p) (p->cap >= 0 ? p->cap - p->flow : -p->flow)
```

ShrtstAugmPath

A função `ShrtstAugmPath` devolve 0 se não há caminho de aumento.

Caso contrário, devolve a capacidade residual `d` de um caminho de aumento na rede expandida e armazena o caminho no vetor `parnt`.

A função supõe que todas as capacidades são menores que `M`.

```
int ShrtstAugmPath(Flownet G, link parnt[]) {
```

ShrtstAugmPath

A função `ShrtstAugmPath` devolve 0 se não há caminho de aumento.

Caso contrário, devolve a capacidade residual `d` de um caminho de aumento na rede expandida e armazena o caminho no vetor `parnt`.

A função supõe que todas as capacidades são menores que `M`.

```
int ShrtstAugmPath(Flownet G, link parnt[]) {  
    Vertex s = G->s, t = G->t, v, w;  
    int lbl[maxV], d; link p;  
    for (v = 0; v < G->V; v++) lbl[v] = -1;  
    QUEUEinit(G->V);
```

```

lbl[s] = 0;
QUEUEput(s);
while (!QUEUEempty()) {
    v = QUEUEget();
    for(p=G->adj[v]; p!=NULL; p=p->next){
        w = p->w;
        if(RC(p)>0 && lbl[w]==-1){
            lbl[w] = 0;
            parnt[w] = p;
            QUEUEput(w);
        }
    }
}

```

```
if (lbl[t] == -1) return 0;
d = M;
for (w = t; w != s; w = p->dup->w){
    p = parnt[w];
    if (d > RC(p)) d = RC(p);
}
return d;
}
```

Número de iterações

O número de caminhos de aumento usados pela combinação de `MaxFlow` com `ShrtstAugmPath` nunca é maior que $VA/2$, sendo V o número de vértices e A o número de arcos *originais*.

Consumo de tempo

O consumo de tempo de `MaxFlow` com `ShrtstAugmPath` é $O(VA(V + A))$, sendo V o número de vértices e A o número de arcos originais.

Considerações finais

MAC0328

MAC0328 Algoritmos em grafos **foi**:

- ▶ uma disciplina introdutória em projeto e análise de algoritmos sobre grafos
- ▶ um **laboratório de algoritmos** sobre grafos

MAC0328

MAC0328 **combinou** técnicas de

- ▶ programação
- ▶ estruturas de dados
- ▶ análise de algoritmos
- ▶ teoria dos grafos

para resolver problemas sobre **grafos**.

Pré-requisitos

O pré-requisito oficial de [MAC0328](#) **era**

- ▶ [MAC0122](#) Princípios de Desenvolvimento de Algoritmos.

No entanto, **era** recomendável que já tivessem cursado

- ▶ [MAC0211](#) Laboratório de programação; e
- ▶ [MAC0323](#) Estruturas de dados

Costuma ser conveniente cursar [MAC0328](#) simultaneamente com

- ▶ [MAC0338](#) Análise de algoritmos.

Principais tópicos foram

- ▶ digrafos e grafos
- ▶ estruturas de dados para digrafos e grafos
- ▶ busca em profundidade
- ▶ caminhos e ciclos
- ▶ florestas e árvores
- ▶ grafos bipartidos
- ▶ digrafos acíclicos
- ▶ ordenação topológica
- ▶ pontes e ciclos
- ▶ grafos conexos e componentes
- ▶ articulações e grafos biconexos
- ▶ digrafos fortemente conexos
- ▶ busca em largura
- ▶ caminhos mínimos
- ▶ árvores geradoras mínimas
- ▶ fluxo em redes

FIM