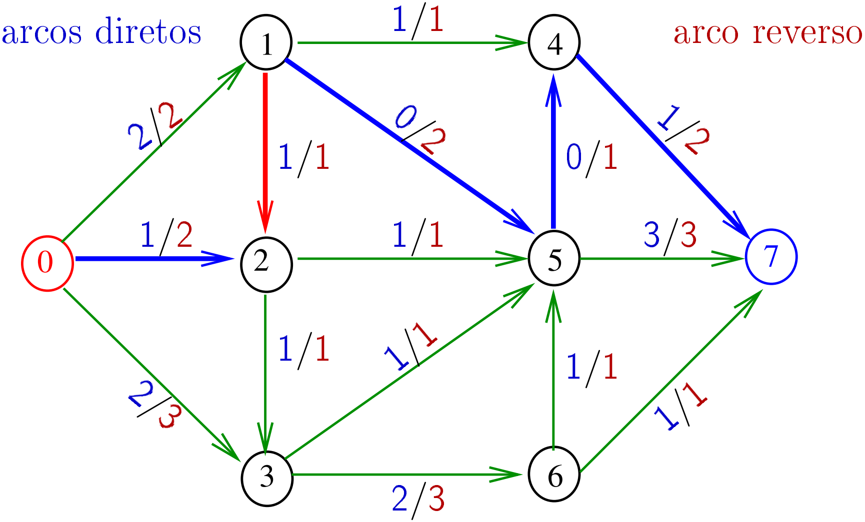
# Caminho de aumento

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 reversos não estão vazios.

# Exemplo



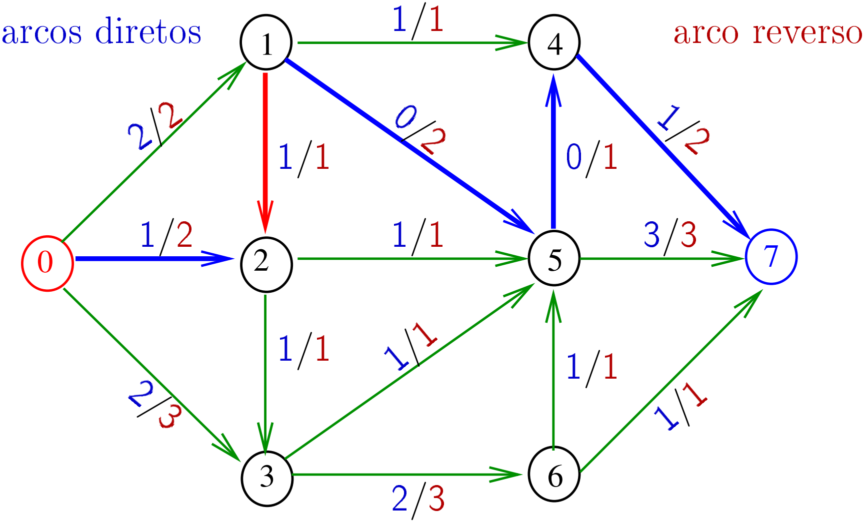
arcos diretos
arco reverso
1/1
2/2
1/1
0/2
0/1
1/2
1/2
1/1
3/3
1/1
1/1
2/3
1/1
1/1
2/3
0
1
2
3
4
5
6
7

# Enviar fluxo através de caminhos de aumento

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

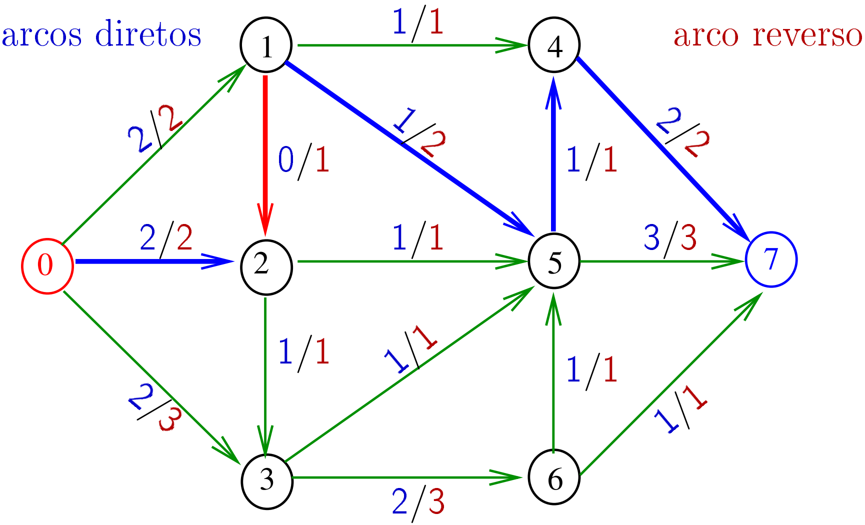
- para cada arco direto, some d ao fluxo
- para cada arco reverso, subtraia d do fluxo.

# Exemplo



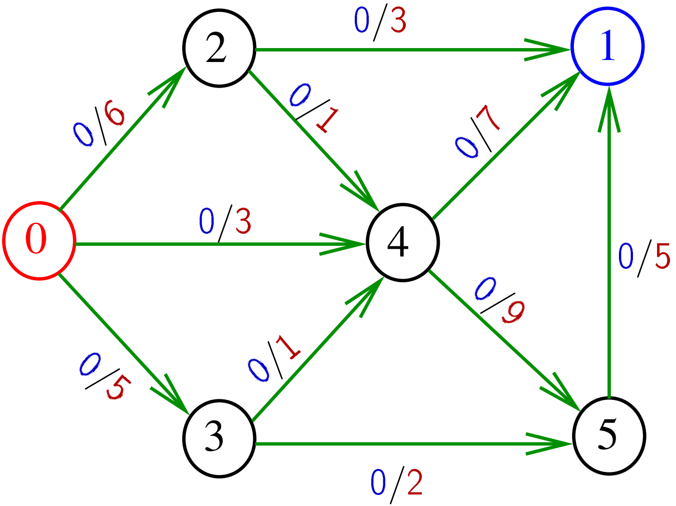
arcos diretos
arco reverso
0
1
2
3
4
5
6
7
1/1
2/2
0/2
1/2
1/1
0/1
1/2
1/1
3/3
1/1
1/1
2/3
1/1
1/1
2/3

# Exemplo


arcos diretos
arco reverso
0
1
2
3
4
5
6
7
2/2
2/2
2/3
1/1
0/1
1/2
1/1
1/1
1/1
1/1
2/3
1/1
2/2
3/3
1/1
1/1

# Método de Ford-Fulkerson

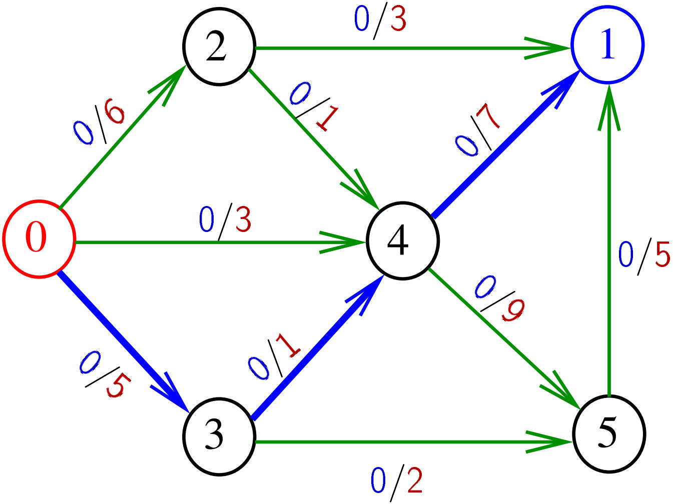
$\text{int}(f) = 0$



# Método de Ford-Fulkerson

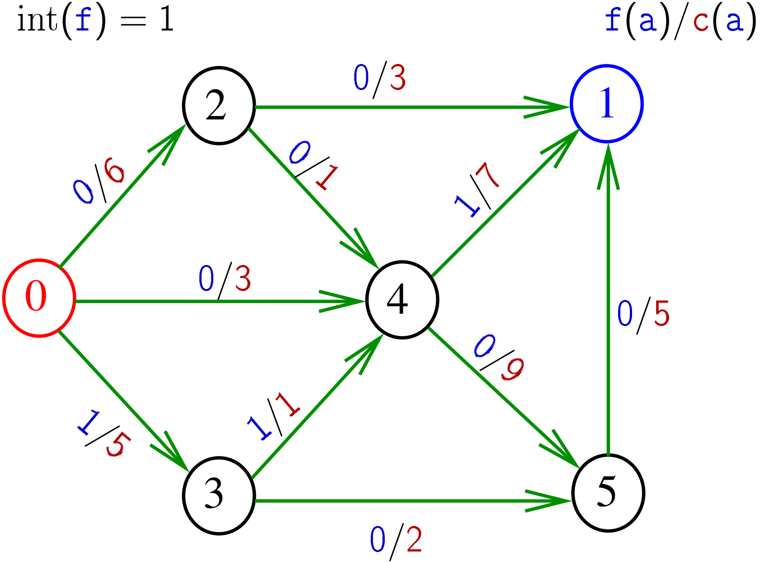
$\text{int}(\texttt{f}) = 0$

$\texttt{f(a)/c(a)}$



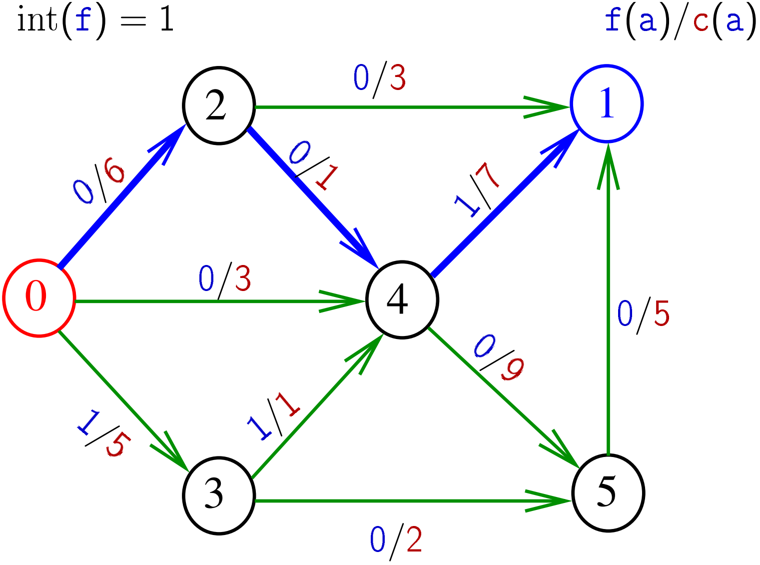
# Método de Ford-Fulkerson

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



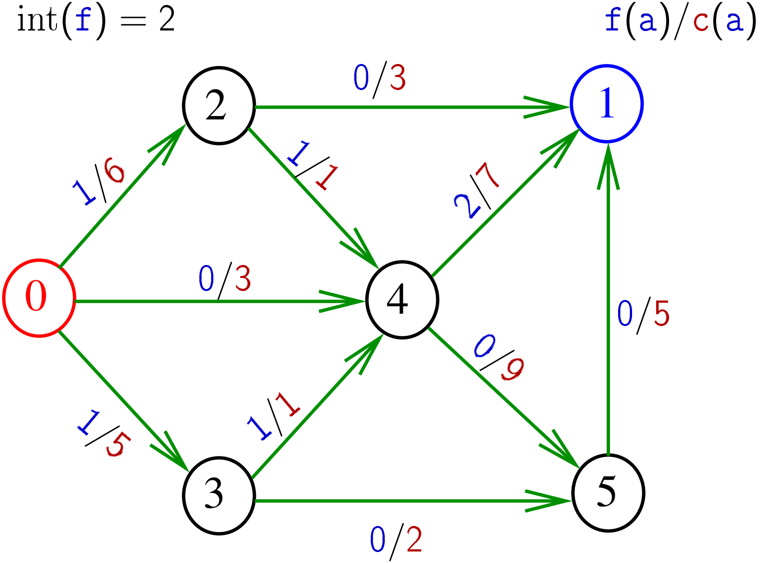
# Método de Ford-Fulkerson

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



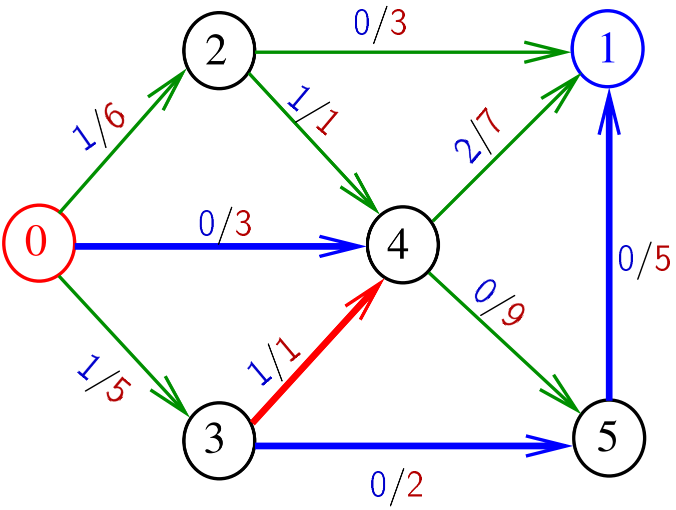
# Método de Ford-Fulkerson

$\mathrm{int}(\mathtt{f}) = 2$



# Método de Ford-Fulkerson

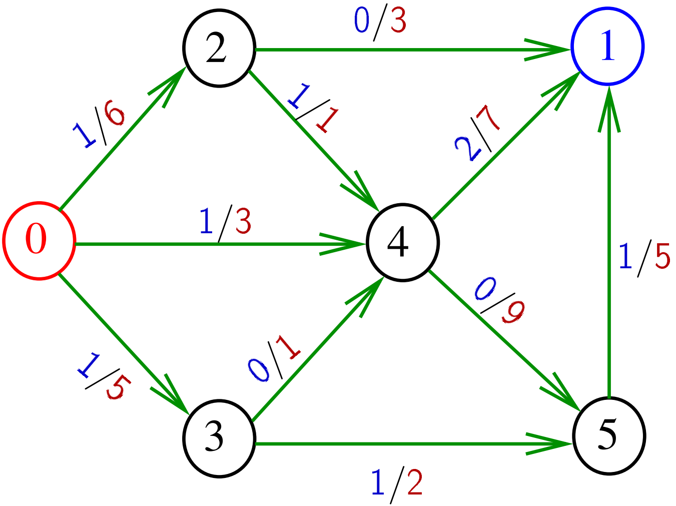
$\mathrm{int}(\mathtt{f}) = 2$ $\mathtt{f(a)/c(a)}$



# Método de Ford-Fulkerson

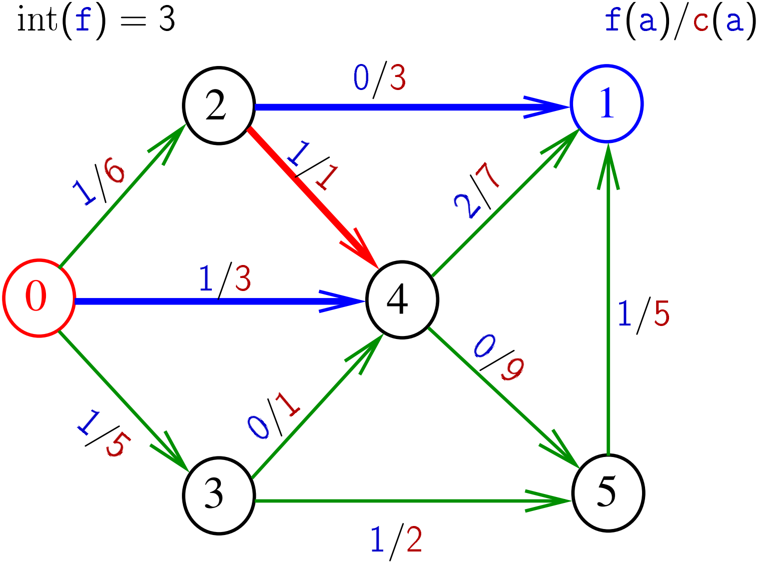
$\text{int}(f) = 3$

$f(a)/c(a)$



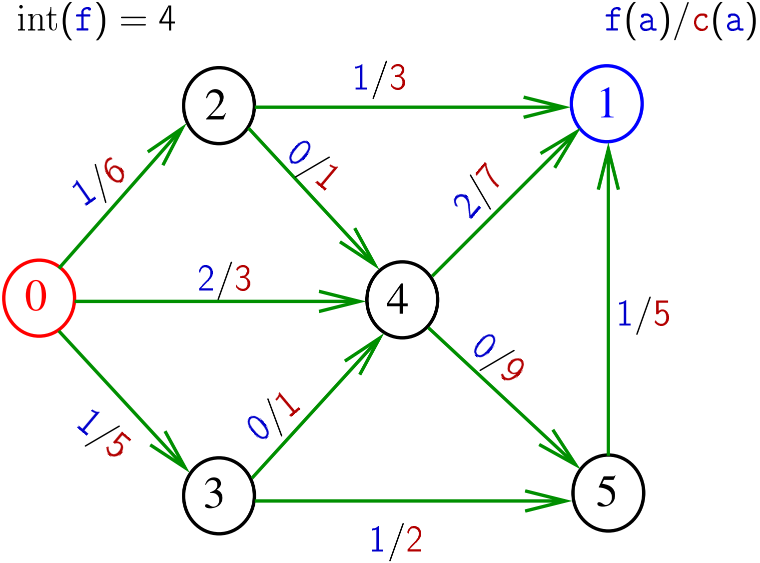
# Método de Ford-Fulkerson

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

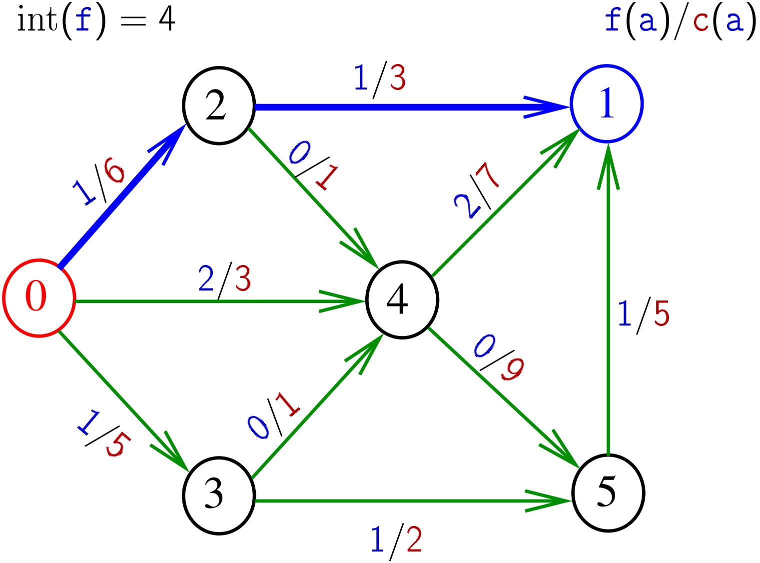


# Método de Ford-Fulkerson

$\mathrm{int}(\mathtt{f}) = 4$

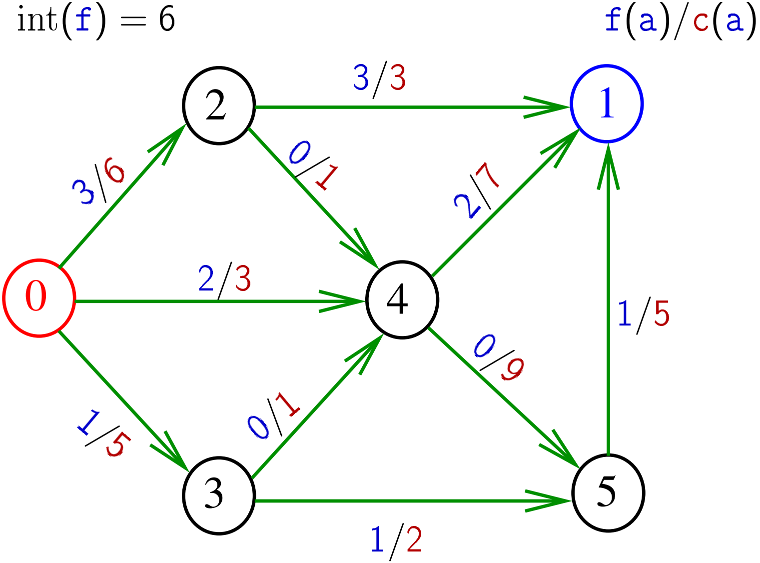


# Método de Ford-Fulkerson



# Método de Ford-Fulkerson

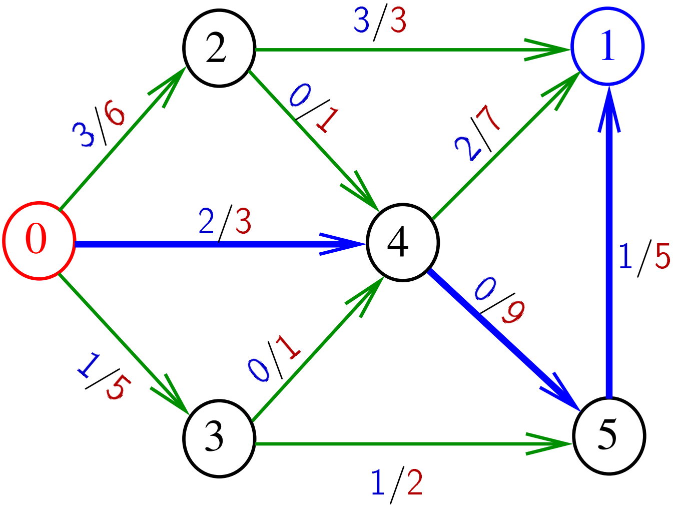
$\text{int}(f) = 6$



# Método de Ford-Fulkerson

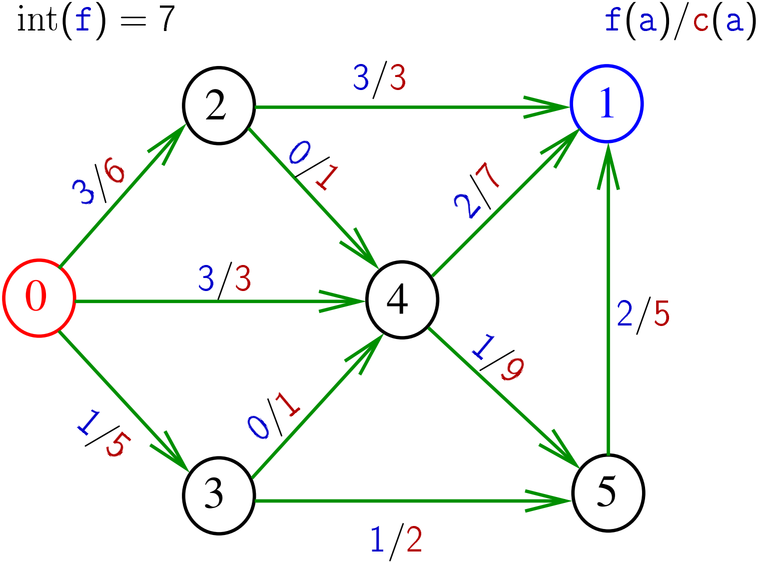
$\text{int}(f) = 6$

$f(a)/c(a)$



# Método de Ford-Fulkerson

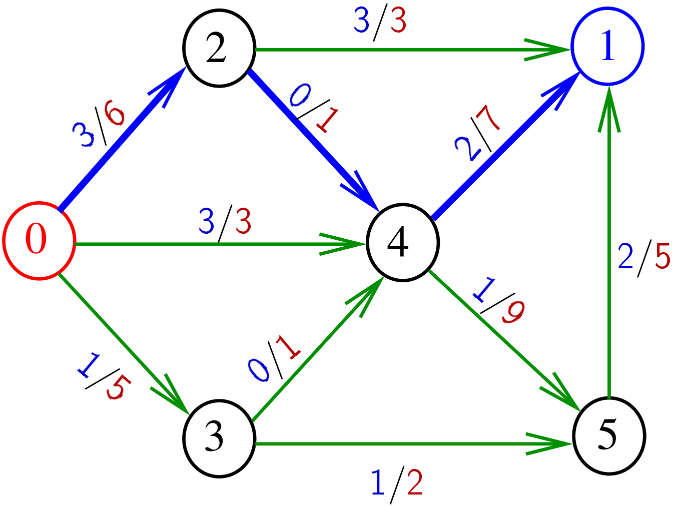
$\text{int}(f) = 7$



# Método de Ford-Fulkerson

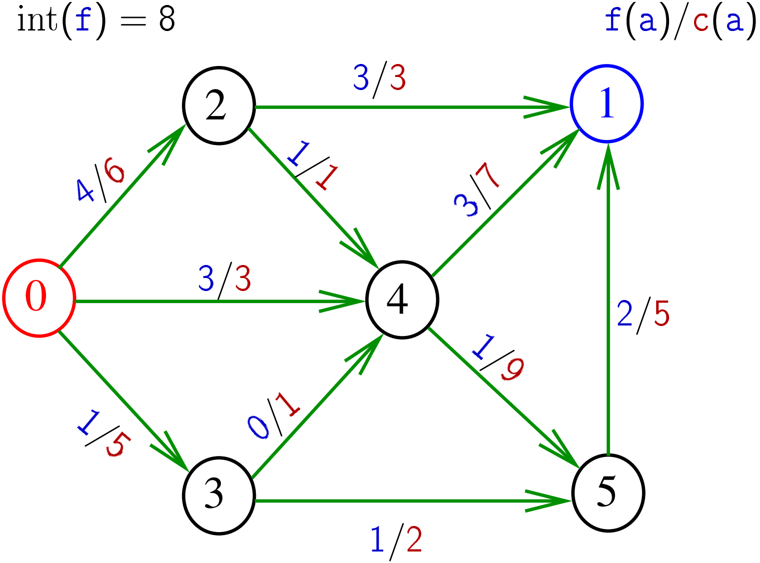
$\text{int}(f) = 7$

$f(a)/c(a)$



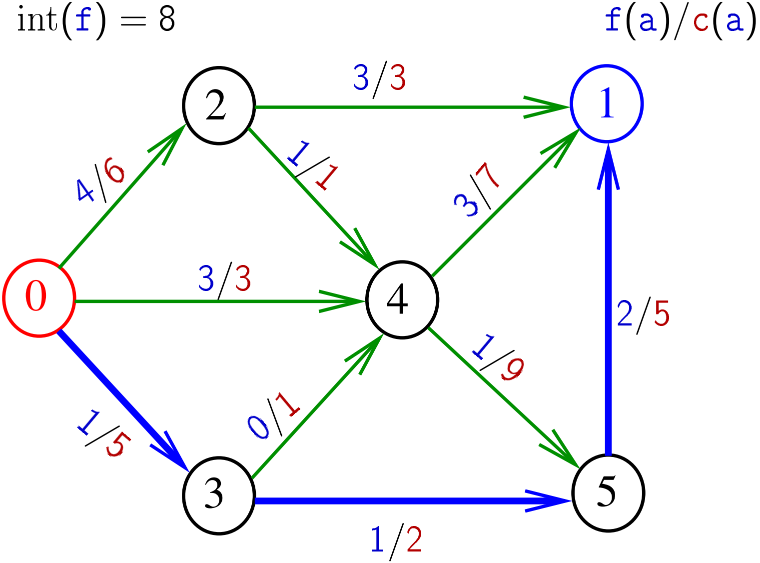
# Método de Ford-Fulkerson

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



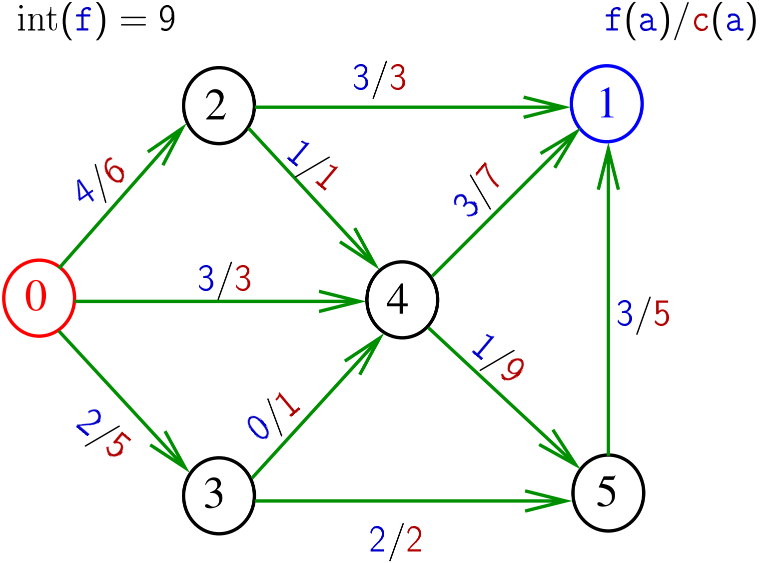
# Método de Ford-Fulkerson

$\mathrm{int}(\mathtt{f}) = 8$



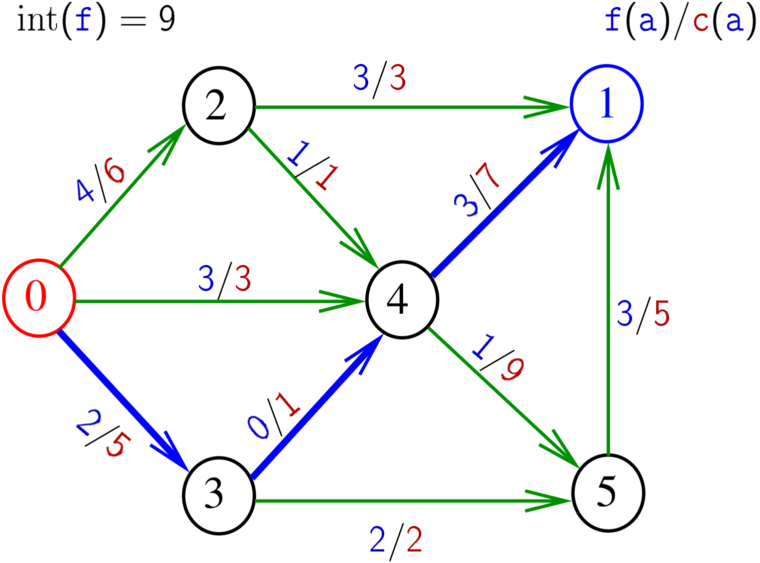
# Método de Ford-Fulkerson

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



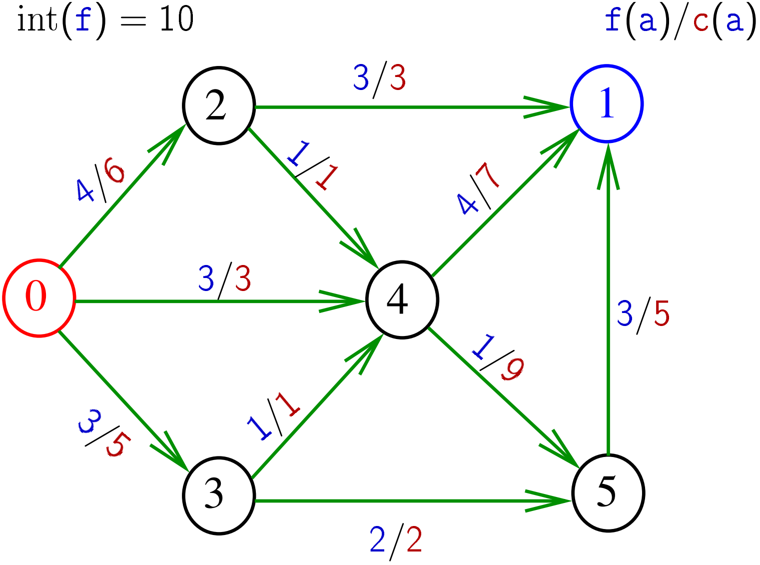
# Método de Ford-Fulkerson

$\mathrm{int}(\mathtt{f}) = 9$



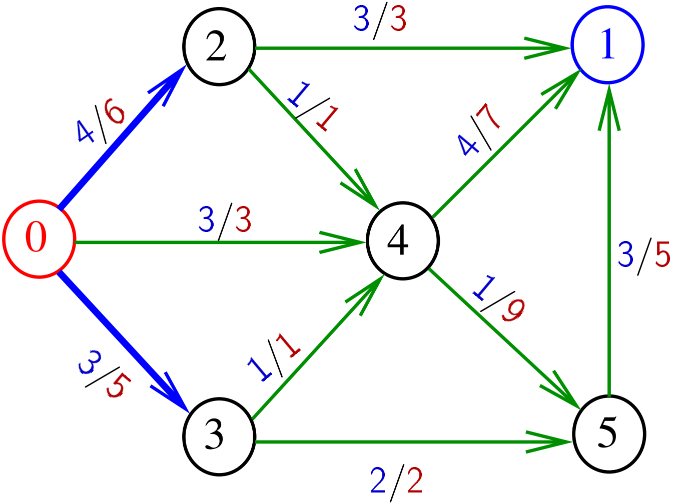
# Método de Ford-Fulkerson

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

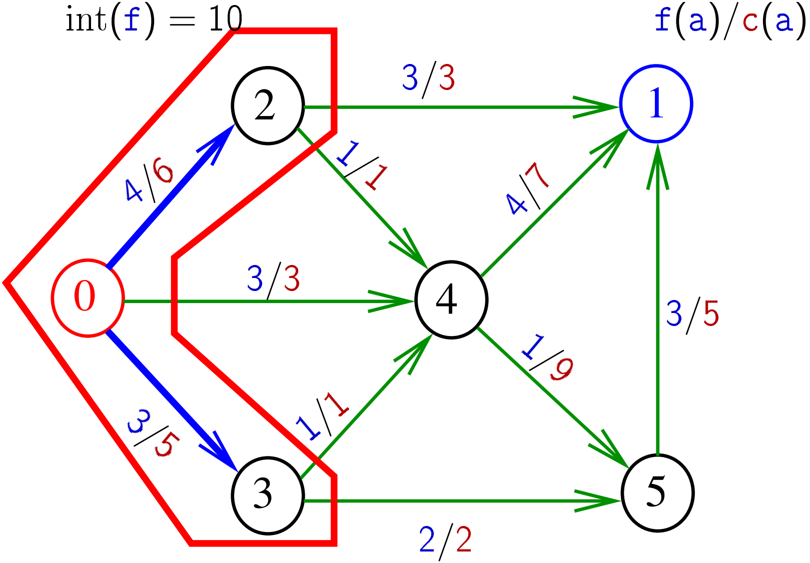


# Método de Ford-Fulkerson

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



# Método de Ford-Fulkerson



# 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:

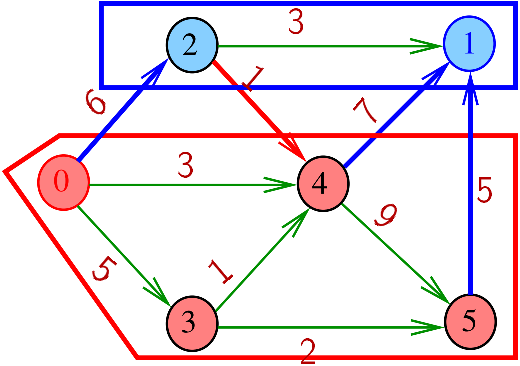
Caso 1: **não existe** um caminho de aumento
Devolva $f$ e pare

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 $P$
Faça $f \leftarrow f'$.

# 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

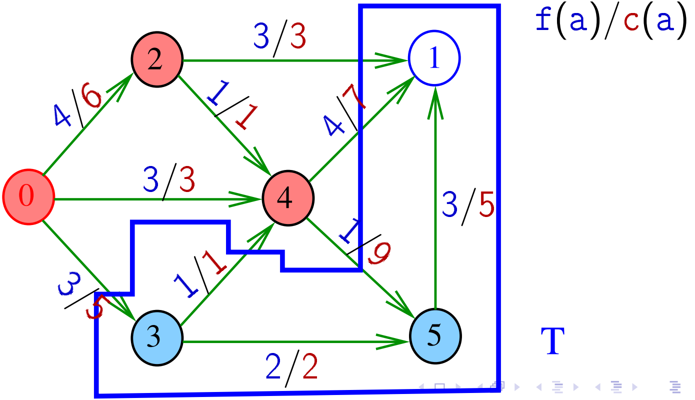


# Lema da dualidade

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

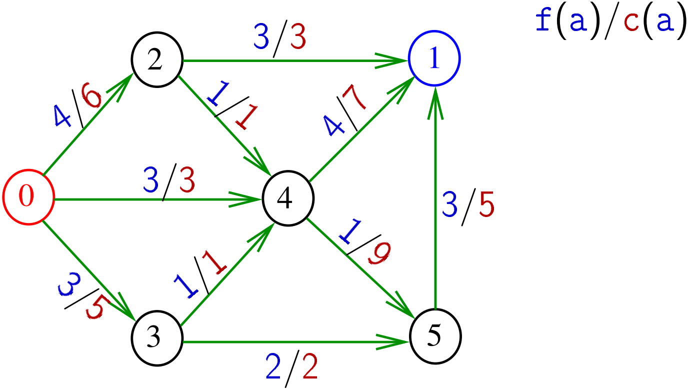
*intensidade de* $f \leq$ *capacidade de* $(S, T)$.

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



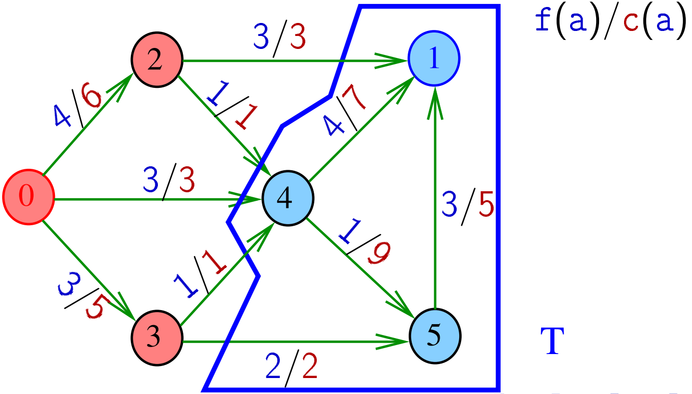
# Conseqüência

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.

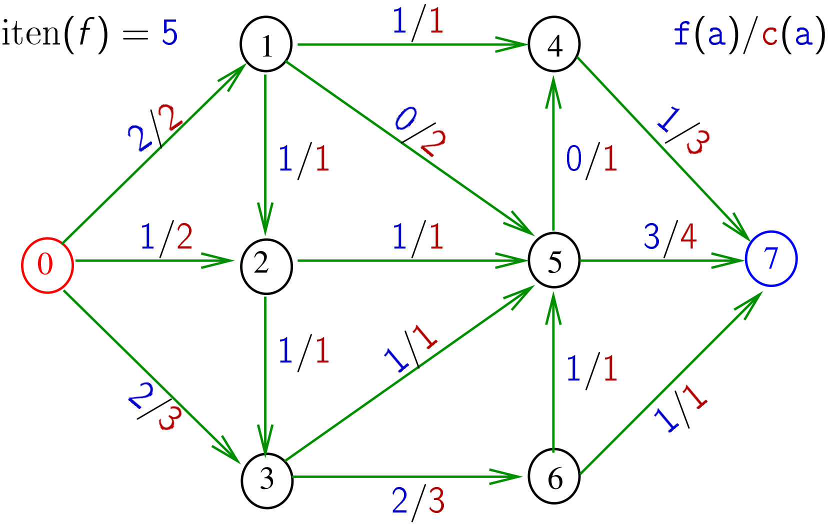


# Conseqüência

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.

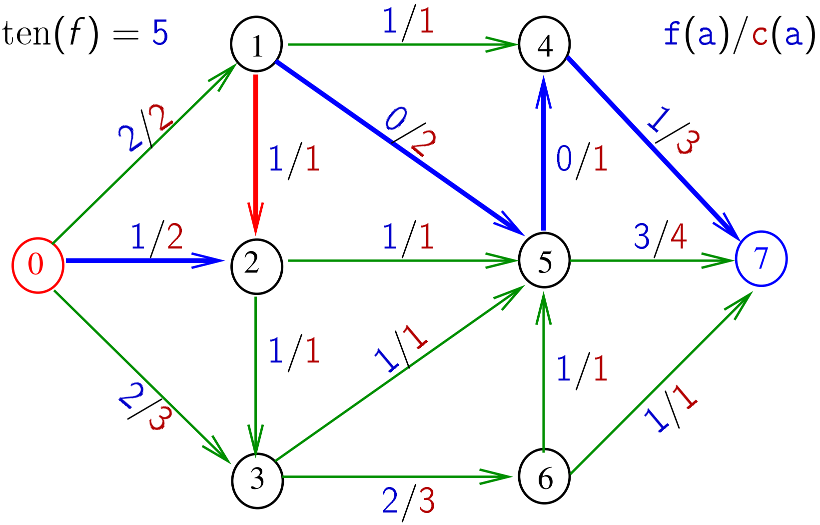


# Fluxo é máximo?

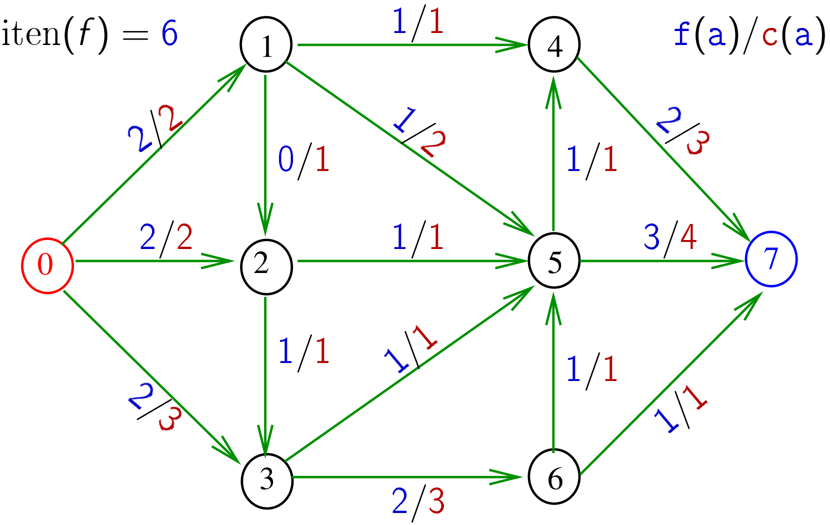


# Caminho de aumento

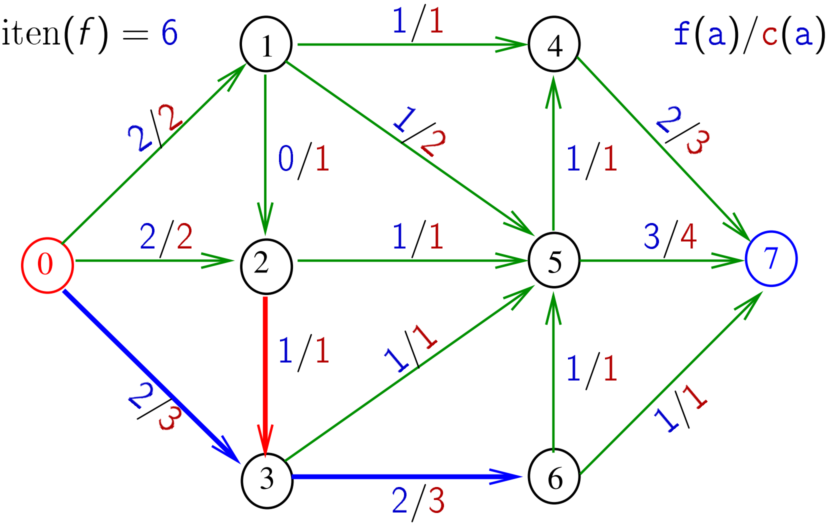
$\text{iten}(f) = 5$



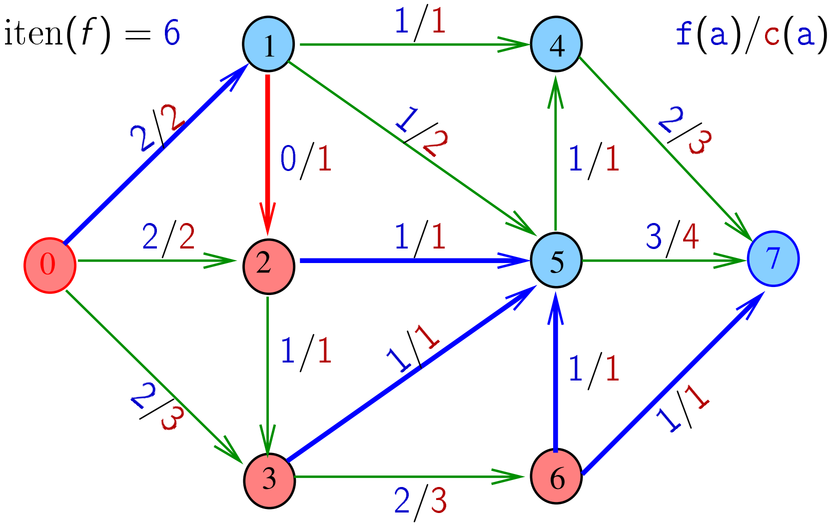
# E agora? Fluxo é máximo?



# Fluxo é máximo!



# Fluxo é máximo!



# 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.

# Estrutura de dados para redes de fluxo

S 22.1

# Listas de adjacência

Redes serão representadas por **listas de adjacência**.
Cada arco `v-u` 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-u` 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;
};
```

# Construtor

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

Insere um arco v-u, de capacidade cap e fluxo nulo na rede G.

```
void
FLOWinsertA (Flownet G, Vertex v, Vertex u, int cap) {
   if (v == u || cap < 0) return;
   G->adj[v] = NEW(u, 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-u`, acrescente à rede um arco `u-v`.

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

A capacidade arco artificial `u-v` será o negativo da capacidade do correspondente arco original `v-u`.

# 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

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

e para cada o arco original teremos

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

# Expand

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

```
void Expand (Flownet G) {
Vertex v,u;
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;
```

# Expand

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

```
void Expand (Flownet G) {
Vertex v,u;
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) {
         u = po->w;
         cap = po->cap;
         flow = po->flow;
         G->adj[u] = pa=
            NEW(v,-cap,-flow,G->adj[u]);
         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) {
   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).

# 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

- $\texttt{cap} \geq 0$ implica em $\texttt{flow} < \texttt{cap}$ e
- $\texttt{cap} < 0$ implica em $\texttt{flow} < 0$

para todo arco do caminho.
A capacidade residual de um arco original da rede expandida é

$$\texttt{cap} - \texttt{flow}$$

e a capacidade residual de um arco artificial é

$$-\texttt{flow}.$$

# 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`-`u` da rede expandida, o código fará

`parnt[u] = p`,

sendo `p` o endereço do nó na lista `adj[v]` para o qual `p->w` vale `u`.

O "pai" `v` de `u` será então `parnt[u]->dup->w`.

# 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;
        }
}
```

# 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 do caminho são menores que M.

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

# 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 do caminho são menores que M.

```
int ShrtstAugmPath(Flownet G,link parnt[]) {
    Vertex s= G->s,t= G->t,v,u;
    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){
        u =p->w;
        if(RC(p)>0 && lbl[u]){
            lbl[u] = 0;
            parnt[u] = p;
            QUEUEput(u);
        }
    }
}
```

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

# Número de iterações

Edmonds e Karp (1972)

| |
|---|
| 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.