

Algoritmos em Grafos: Fluxo Máximo em Redes de Transporte

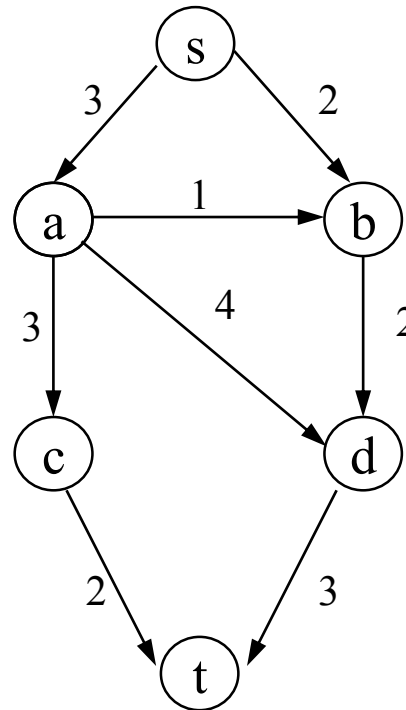
J. Pascoal Faria, R. Rossetti, L. Ferreira

FEUP, MIEIC

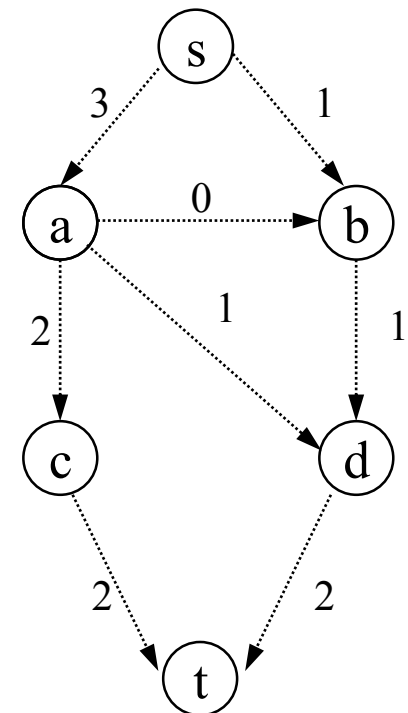
Rede de transporte

- Modelar fluxos conservativos entre dois pontos através de canais com capacidade limitada
 - s: fonte (produtor)
 - t: poço (consumidor)
 - fluxo não pode ultrapassar a capacidade da aresta
 - soma dos fluxos de entrada num vértice intermédio igual à soma dos fluxos de saída
- Por vezes as arestas têm custos associados (custo de transportar uma unidade de fluxo)

Rede e capacidades

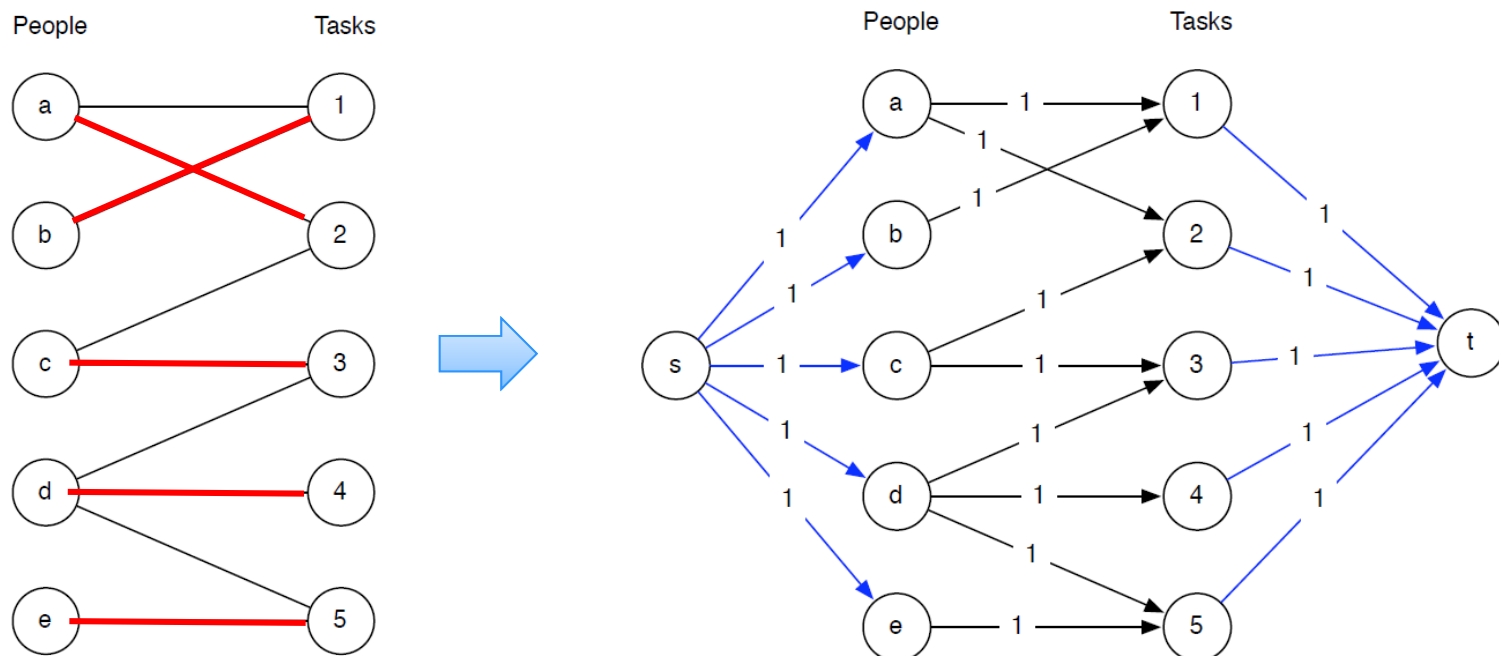


Rede e fluxos



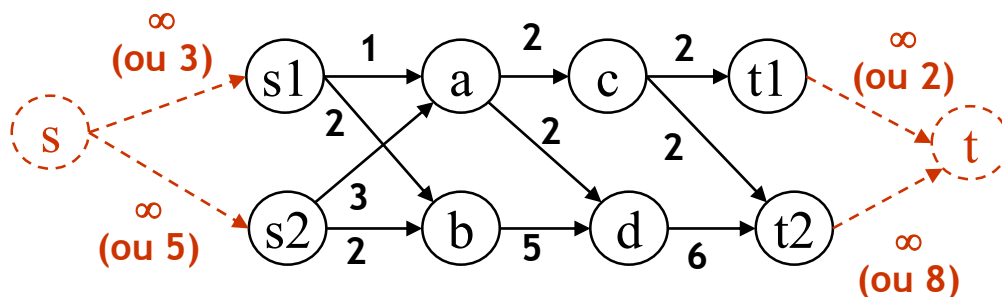
Exemplos de aplicação

- Rede de abastecimento de líquido ponto a ponto
- Tráfego entre dois pontos
- Emparelhamento máximo em grafos bipartidos (*maximum bipartite matching*)



Redes com múltiplas fontes e poços

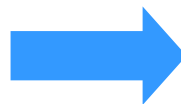
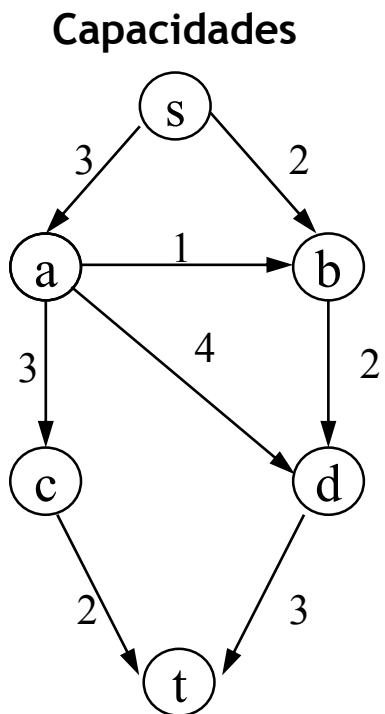
- Caso de múltiplas fontes e poços (ou mesmo de vértices que podem ser simultaneamente fontes, poços e vértices intermédios) é facilmente redutível ao caso base (uma fonte e um poço)



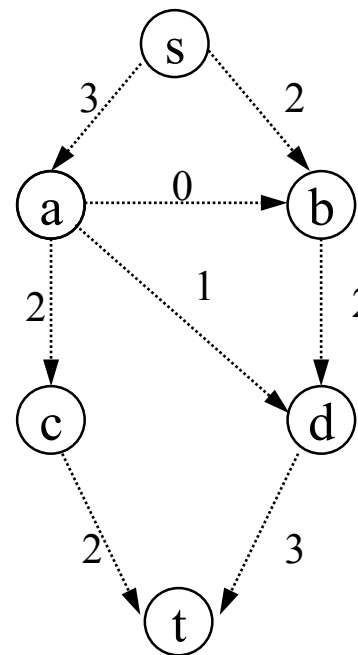
- Se a rede tiver custos nas arestas, as arestas adicionadas têm custo 0

Problema do fluxo máximo

- Encontrar um fluxo de valor máximo (fluxo total que parte de s / chega a t)



Fluxo máximo (valor 5)



Formalização

Dados de entrada :

c_{ij} - capacidade da aresta que vai do nó i a j (0 se não existir)

Dados de saída (variáveis a calcular) :

f_{ij} - fluxo que atravessa a aresta que vai do nó i para o nó j (0 se não existir)

Restrições :

$$0 \leq f_{ij} \leq c_{ij}, \forall_{ij}$$

$$\sum_j f_{ij} = \sum_j f_{ji}, \forall_{i \neq s, t}$$

Objectivo :

$$\max \sum_j f_{sj}$$

Algoritmo de Ford-Fulkerson (1955)

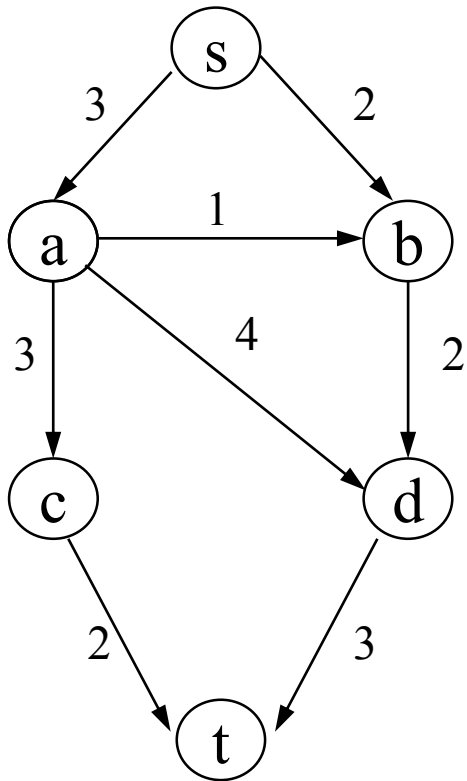
■ Estruturas de dados:

- G - grafo base de capacidades $c(v,w)$
- G_f - grafo de fluxos $f(v,w)$
 - inicialmente fluxos iguais a 0
 - no fim, tem o fluxo máximo
- G_r - grafo de resíduos (auxiliar)
 - para cada arco (v, w) em G com $c(v,w) > f(v,w)$, cria-se um arco no mesmo sentido em G_r de resíduo igual a $c(v,w) - f(v,w)$ (capacidade disponível)
 - para cada arco (v, w) em G com $f(v,w) > 0$, cria-se um arco em sentido inverso em G_r de resíduo igual a $f(v,w)$
 - necessário para garantir que se encontra a solução óptima (ver exemplo)!

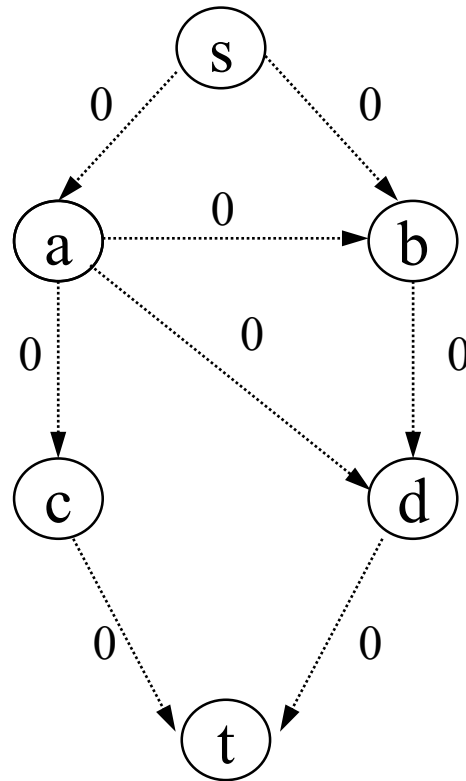
Algoritmo de Ford-Fulkerson (1955)

- Método (dos caminhos de aumento):
 - Enquanto existirem caminhos entre s e t em G_r
 - Seleccionar um caminho qualquer em G_r entre s e t (caminho de aumento)
 - Determinar o valor mínimo (f) nos arcos desse caminho
 - Aumentar esse valor de fluxo (f) a cada um dos arcos correspondentes em G_f
 - Recalcular G_r

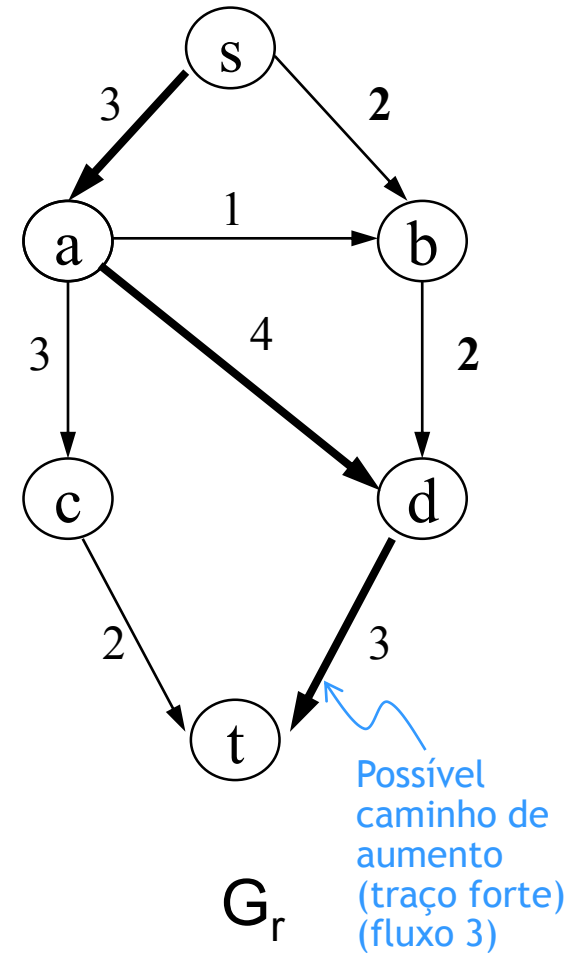
Exemplo: inicialização



G

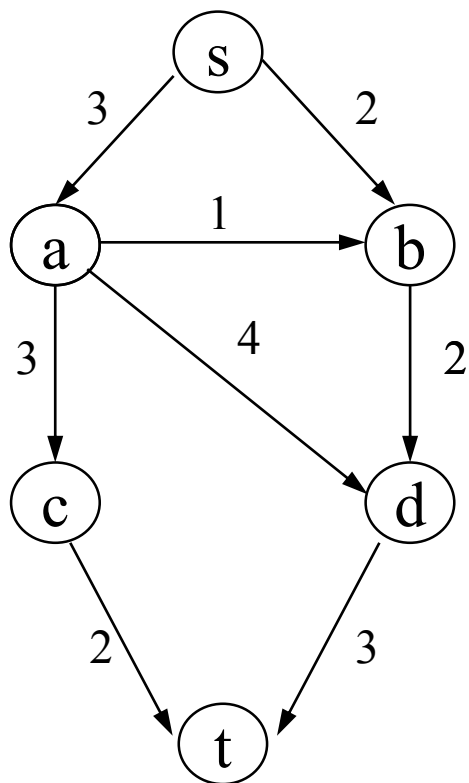


G_f

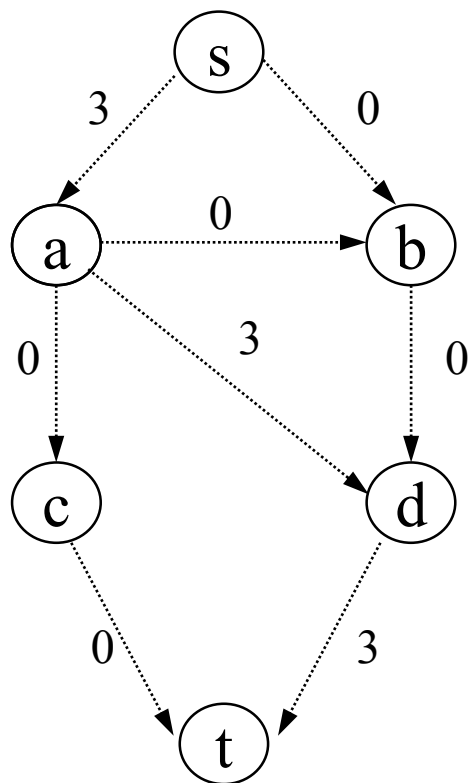


G_r

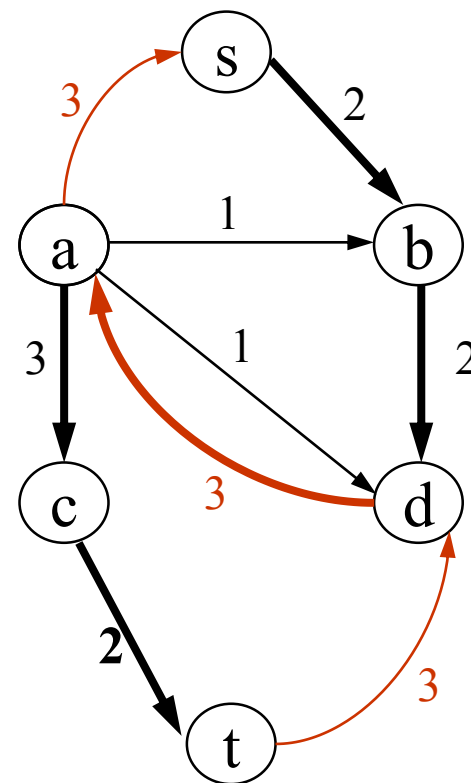
Exemplo: 1ª iteração



G



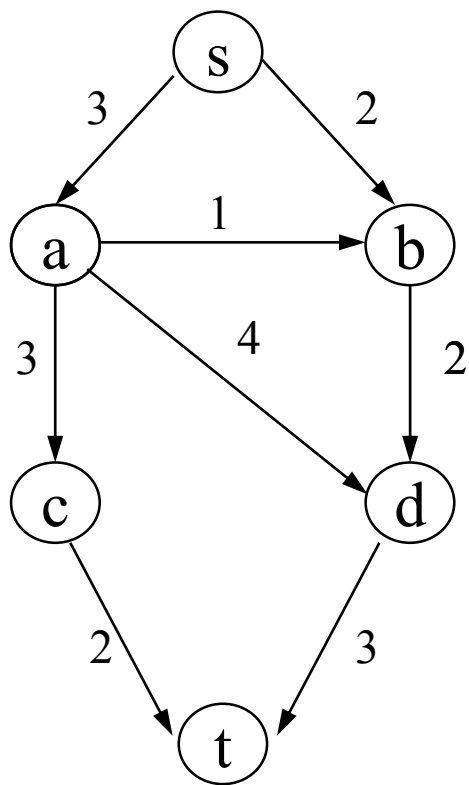
G_f



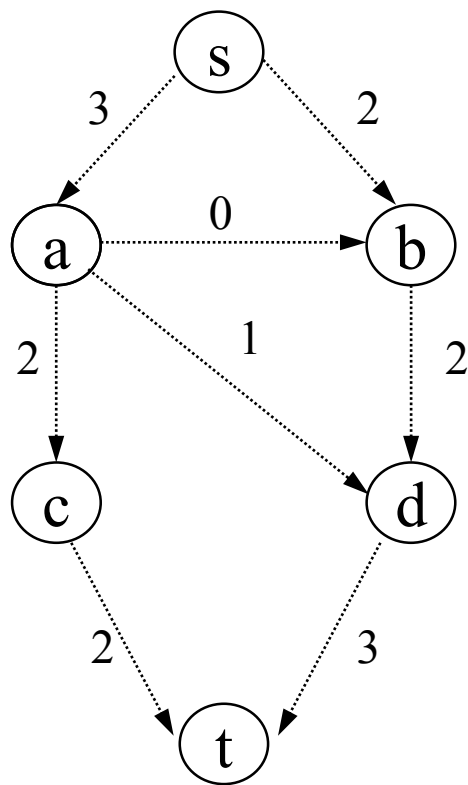
G_r

Se não tivesse as arestas em sentido inverso, parava aqui com solução não óptima!
(Caminho de aumento com fluxo 2)

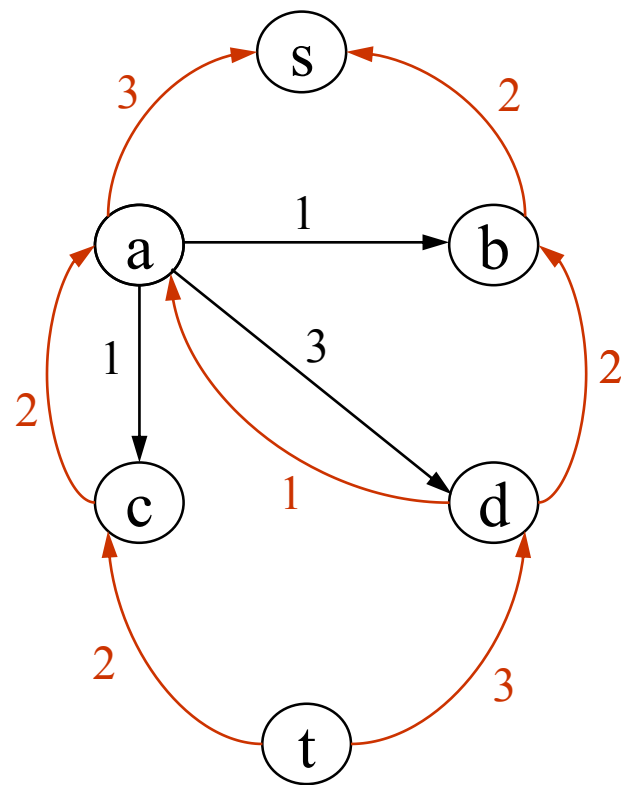
Exemplo: 2ª iteração



G



G_f



G_r

FIM

(valor do fluxo máximo = 5)

Análise do algoritmo de Ford-Fulkerson

- Se as capacidades forem números racionais, o algoritmo termina com o fluxo máximo
- Se as capacidades forem inteiros e o fluxo máximo M
 - Algoritmo tem a propriedade de integralidade: os fluxos finais são também inteiros
 - Bastam M iterações (fluxo aumenta pelo menos 1 por iteração)
 - Cada iteração pode ser feita em tempo $O(|E|)$
 - Tempo de execução total: $O(M |E|)$ - mau

Algoritmo de Edmonds-Karp (1969)

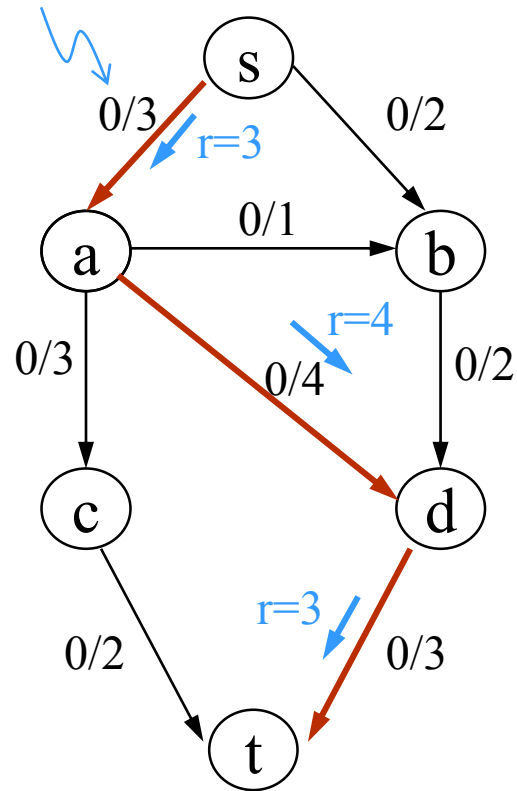
- Em cada iteração do algoritmo de Ford-Fulkerson escolhe-se um caminho de aumento de comprimento mínimo
 - O exemplo apresentado anteriormente já obedece a este critério!
 - Um caminho de aumento mais curto pode ser encontrado em tempo $O(|E|)$ através de pesquisa em largura
 - N° máximo de aumentos é $|E| \cdot |V|$ (*ver referências*)
 - Tempo de execução: $O(|V| |E|^2)$

Implementação

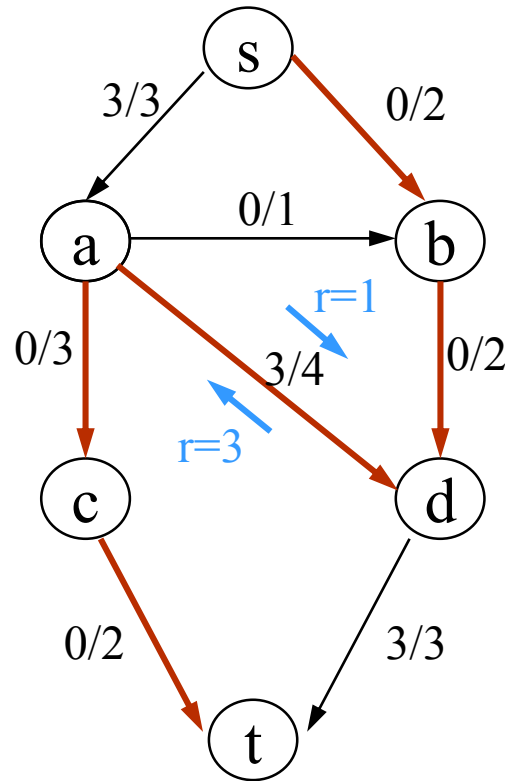
- Para efetuar os cálculos num único grafo, guardam-se:
 - Em cada aresta:
 - **orig**: apontador para vértice de origem
 - **dest**: apontador para vértice de destino
 - **capacity**: capacidade da aresta
 - **flow**: fluxo na aresta
 - Em cada vértice:
 - **outgoing**: vetor de apontadores para arestas que saem do vértice
 - **incoming**: vetor de apontadores para arestas dirigidas ao vértice
 - **visited**: campo booleano usado na procura do caminho de aumento
 - **path**: apontador para aresta anterior no caminho de aumento
 - No grafo:
 - **vertexSet**: vetor de apontadores para vértices
- O grafo de resíduos é determinado “*on the fly*”
 - Arestas percorridas no sentido normal têm resíduo = capacidade - fluxo
 - Arestas percorridas no sentido inverso têm resíduo = fluxo

Exemplo

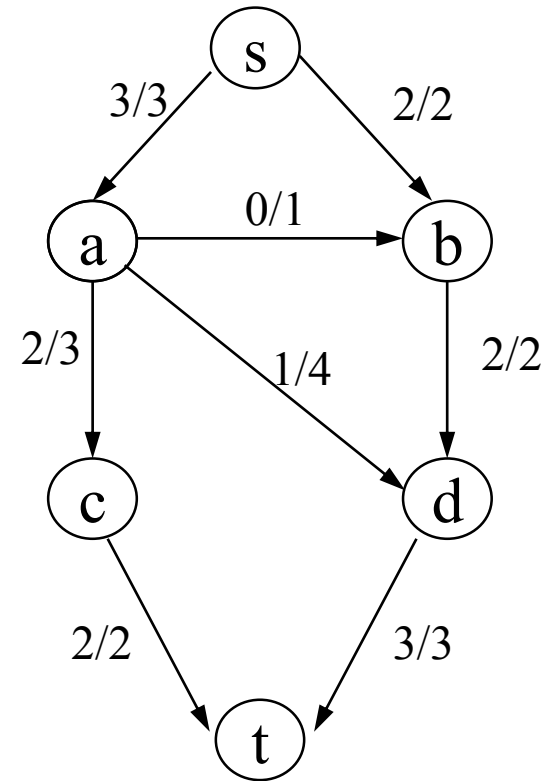
fluxo/capacidade



(f = 3)



(f = 2)



FIM

Pseudo-código

FordFulkerson(g, s, t):

1. ResetFlows(g)

2. **while** FindAugmentationPath(g, s, t) **do**

3. $f \leftarrow$ FindMinResidualAlongPath(g, s, t)

4. AugmentFlowAlongPath(g, s, t, f)

ResetFlows(g):

1. **for** $v \in \text{vertexSet}(g)$ **do**

2. $\text{flow}(v) \leftarrow 0$


```

FindAugmentationPath(g,s,t): // Edmonds-Karp (breadth-first)
1.  for v ∈ vertexSet(g) do visited(v) ← false
2.  visited(s) ← true
3.  Q ← ∅
4.  ENQUEUE(Q, s)
5.  while Q ≠ ∅ ∧ ¬ visited(t) do
6.      v ← DEQUEUE(Q)
7.      for e ∈ outgoing(v) do // direct residual edges
8.          TestAndVisit(Q, e, dest(e), capacity(e) - flow(e))
9.      for e ∈ incoming(v) do // reverse residual edges
10.         TestAndVisit(Q, e, orig(e), flow(e))
11. return visited(t)

```

```

TestAndVisit(Q, e, w, residual):
1. if ¬ visited(w) ∧ residual > 0 then
2.     visited(w) ← true
3.     path(w) ← e // previous edge in shortest path
4.     ENQUEUE(Q, w)

```

FindMinResidualAlongPath(g, s, t):

```
1.  f ← ∞
2.  v ← t
3.  while v ≠ s do
4.      e ← path(v)
5.      if dest(e) = v then // direct residual edge
6.          f ← min(f, capacity(e) - flow(e))
7.          v ← orig(e)
8.      else // reverse residual edge
9.          f ← min(f, flow(e))
10.     v ← dest(e)
```

AugmentFlowAlongPath(g, s, t, f):

```
1.  v ← t
2.  while v ≠ s do
3.      e ← path(v)
4.      if dest(e) = v then // direct residual edge
5.          flow(e) ← flow(e) + f
6.          v ← orig(e)
7.      else // reverse residual edge
8.          flow(e) ← flow(e) - f
9.          v ← dest(e)
```

Algoritmos mais eficientes

| year | authors | complexity |
|------|-------------------------------|--|
| 1955 | Ford-Fulkerson [19] | $O(mnU)$ |
| 1970 | Dinic [15] | $O(mn^2)$ |
| 1969 | Edmonds-Karp [17] | $O(m^2n)$ |
| 1972 | Dinic [15], Edmonds-Karp [17] | $O(m^2 \log U)$ |
| 1973 | Dinic [16], Gabow [20] | $O(mn \log U)$ |
| 1974 | Karzanov [37] | $O(n^3)$ |
| 1977 | Cherkassky [11] | $O(n^2 m^{1/2})$ |
| 1980 | Galil-Naamad [21] | $O(mn(\log n)^2)$ |
| 1983 | Sleator-Tarjan [44] | $O(mn \log n)$ |
| 1986 | Goldberg-Tarjan [26] | $O(mn \log(n^2/m))$ |
| 1987 | Ahuja-Orlin [3] | $O(mn + n^2 \log U)$ |
| 1987 | Ahuja-Orlin-Tarjan [4] | $O(mn \log(2 + n\sqrt{\log U/m}))$ |
| 1990 | Cheriyon-Hagerup-Mehlhorn [9] | $O(n^3 / \log n)$ |
| 1990 | Alon [5] | $O(mn + n^{8/3} \log n)$ |
| 1992 | King-Rao-Tarjan [38] | $O(mn + n^{2+\epsilon})$ |
| 1993 | Phillips-Westbrook [42] | $O(mn \log_{m/n} n + n^2 (\log n)^{2+\epsilon})$ |
| 1994 | King-Rao-Tarjan [39] | $O(mn \log_{m/(n \log n)} n)$ |
| 1997 | Goldberg-Rao [23] | $O(\min\{m^{1/2}, n^{2/3}\} m \log(n^2/m) \log U)$ |

($m = |E|$, $n = |V|$, $U = \text{capacidade máxima}$)

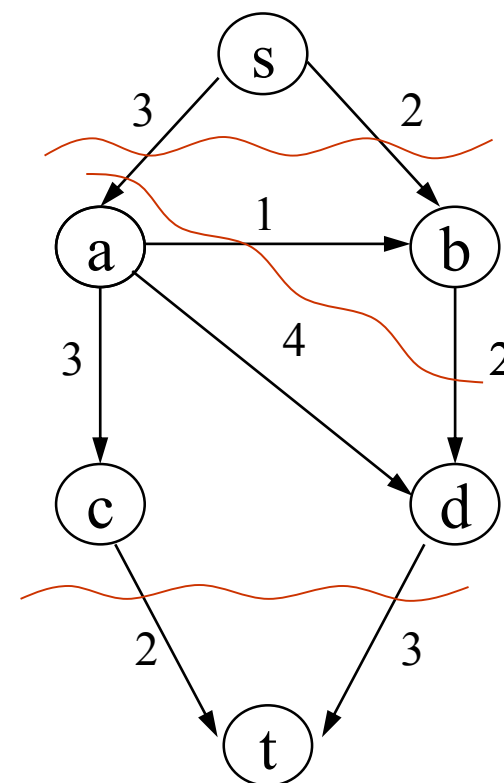
T. Asano and Y. Asano: recent developments in Maximum Flow Algorithms. Journal of the Operations Research, 1 (2000).



Dualidade entre fluxo máximo e corte mínimo

- Teorema: O valor do fluxo máximo numa rede de transporte é igual à capacidade do corte mínimo
 - Um corte (S, T) numa rede de transporte $G=(V, E)$ com fonte s e poço t é uma partição de V em conjuntos S e $T=V-S$ tal que $s \in S$ e $t \in T$
 - A capacidade de um corte (S, T) é a soma das capacidades das arestas cortadas dirigidas de S para T
 - Um corte mínimo é um corte cuja capacidade é mínima

Cortes mínimos na rede do exemplo:



Referências e informação adicional

- “Introduction to Algorithms”, 3rd Edition, T.H. Cormen, C. E. Leiserson, R. L. Rivest , C. Stein., MIT Press, 2009
 - Chapter 26 Maximum Flow
- “Data Structures and Algorithm Analysis in Java”, Second Edition, Mark Allen Weiss, Addison Wesley, 2006
- “The Algorithm Design Manual”, Steven S. Skiena, Springer-Verlag, 1998