Fluxo máximo

Laboratório de Programação Competitiva - 2020

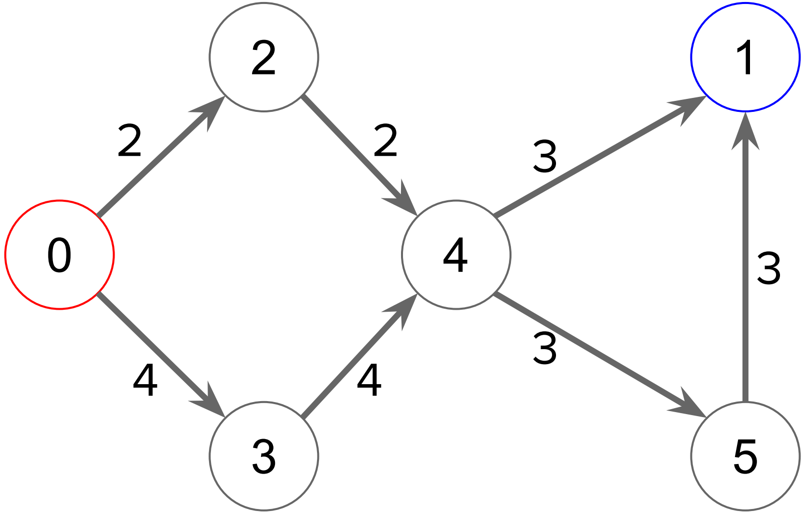
Pedro Henrique Paiola

Introdução

- O problema do fluxo máximo é uma poderosa ferramenta de modelagem, capaz de representar uma grande variedade de problemas.
    - Grande parte da dificuldade em exercícios de Programação Competitiva envolvendo fluxo máximo não está na aplicação do algoritmo, mas sim na modelagem!

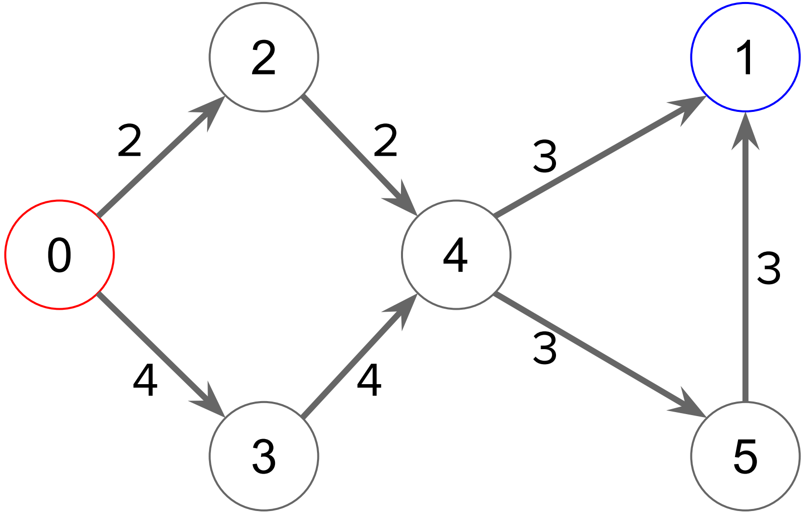
Fluxo

- Considere um grafo direcionado ponderado com os “pesos” das arestas representam fluxos.



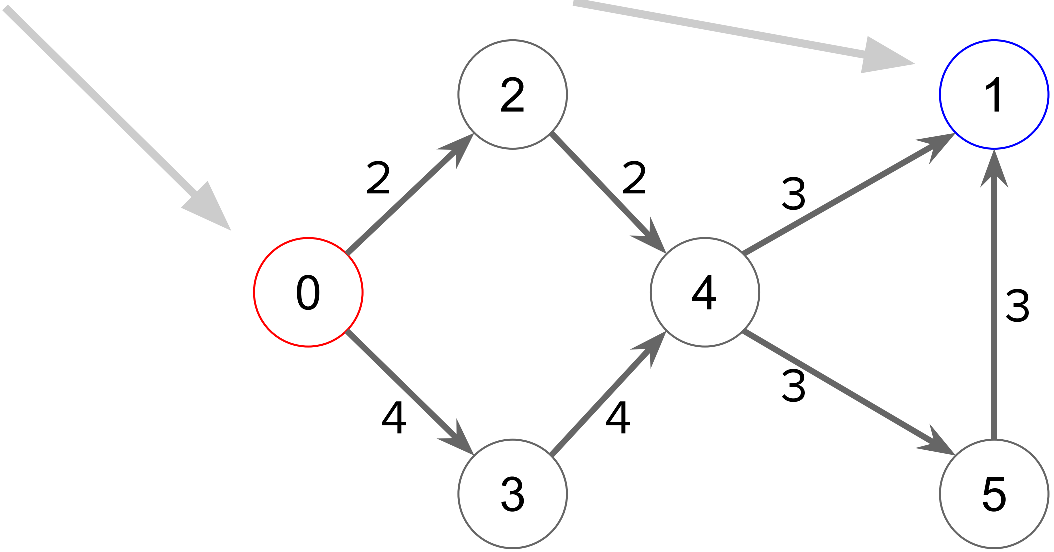
Fluxo

- O peso de cada aresta deve ser positivo, e denotaremos o peso de uma aresta (v,w) como $f_{v,w}$ = fluxo na aresta (v,w)
- Ex: $f_{2,4} = 2$



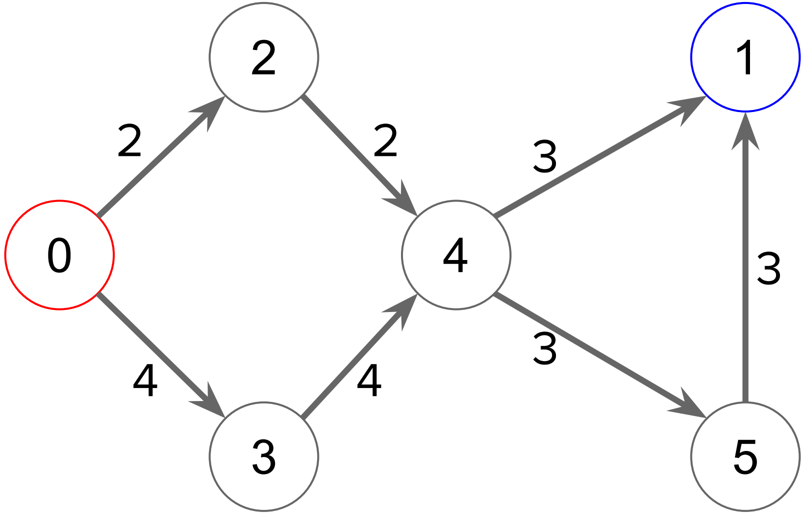
Fluxo

- Em um grafo de fluxo, temos dois vértices que se distinguem dos outros: o vértice inicial (fonte) e o vértice final (sumidouro ou sorvedouro)



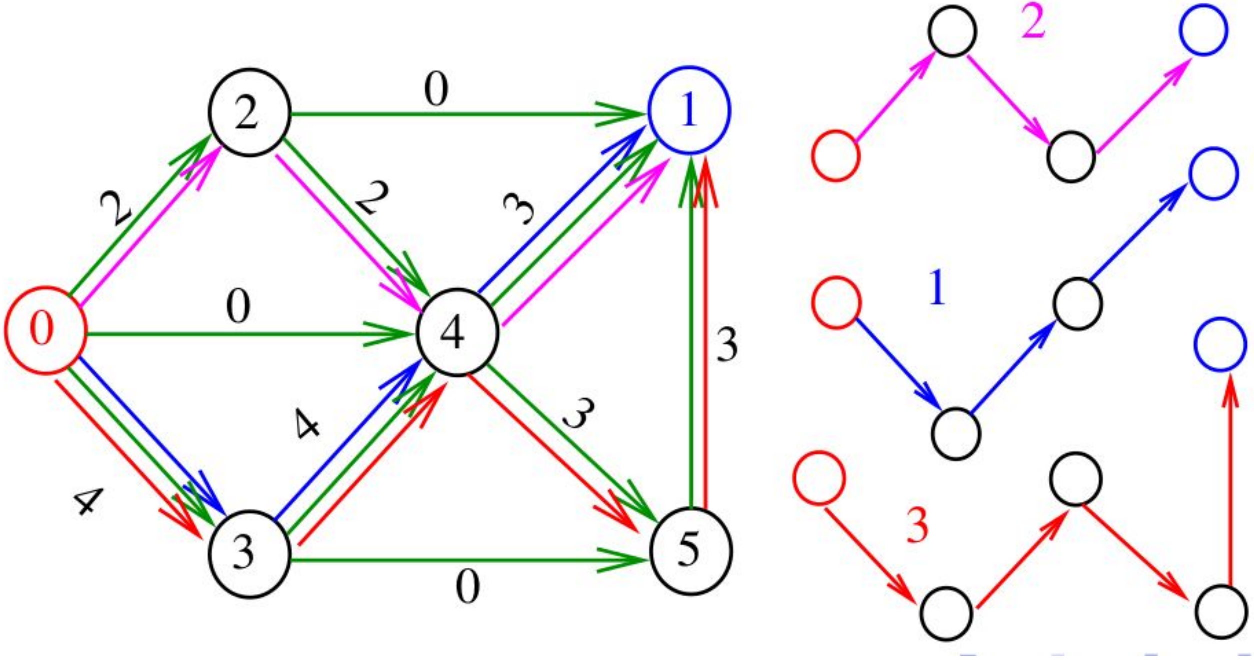
Fluxo

- Os outros vértices podem ser chamados de junções. Para as junções, podemos observar que a soma dos fluxos dos arcos que ENTRAM é igual a soma dos fluxos dos arcos que SAEM.



Fluxo como coleção de caminhos

- Todo fluxo pode ser visto como uma coleção de caminhos, todos com origem na fonte e término no sumidouro.



Problema do fluxo máximo

- Até o momento, consideramos um grafo de fluxo, onde cada aresta já apresenta o fluxo de um vértice para o outro.
- Porém, em um problema de fluxo máximo, recebemos como entrada um **grafo capacitado**, onde o peso de cada aresta representa a capacidade de um arco ($c_{v,w}$)
- Problema do fluxo máximo: dado um grafo capacitado com fonte s e sumidouro t, encontrar um fluxo de intensidade máxima dentre os que respeitam as capacidades dos arcos ($f_{v,w} <= c_{v,w}$)

Aplicações

- Maximizar o fluxo de uma rede de distribuição de uma companhia a partir de suas fábricas para os seus clientes.
- Maximizar o fluxo de óleo/água através de um sistema de oleodutos/aquedutos.
- Maximizar o fluxo de veículos através de uma rede de transporte
- Na resolução de outros problemas de Grafos:
    - Corte mínimo
    - Emparelhamento em grafos bipartidos

Formalizando

$$max \sum_j f_{s,j}$$

$$s.a$$

$$0 \leq f_{i,j} \leq c_{i,j}, \forall (i,j)$$

$$\sum_j f_{i,j} = \sum_k f_{k,i}, \forall i \neq s, t$$

Formalizando

- Uma vez que o problema de fluxo máximo pode ser formulado como um problema de programação linear, o algoritmo Simplex pode ser utilizado para a obtenção da solução ótima.
- Porém, existem algoritmos mais eficientes, baseado em Caminhos de Aumento ou Caminhos Aumentadores (*Augmenting Path*)
- Estes algoritmos são baseados em dois conceitos intuitivos: o conceito de rede residual e o de caminho de aumento (propriamente dito)

Rede residual

- Suponha que uma dada aresta (v,w) possui um fluxo $f_{v,w}$ e uma capacidade $c_{v,w}$
- Podemos definir a **capacidade residual** $c^R_{v,w} = c_{v,w} - f_{v,w}$. Ou seja, o quanto resta da capacidade da aresta.
- Uma rede residual G^R de uma rede G é formado por arestas anotadas com suas devidas capacidades residuais.

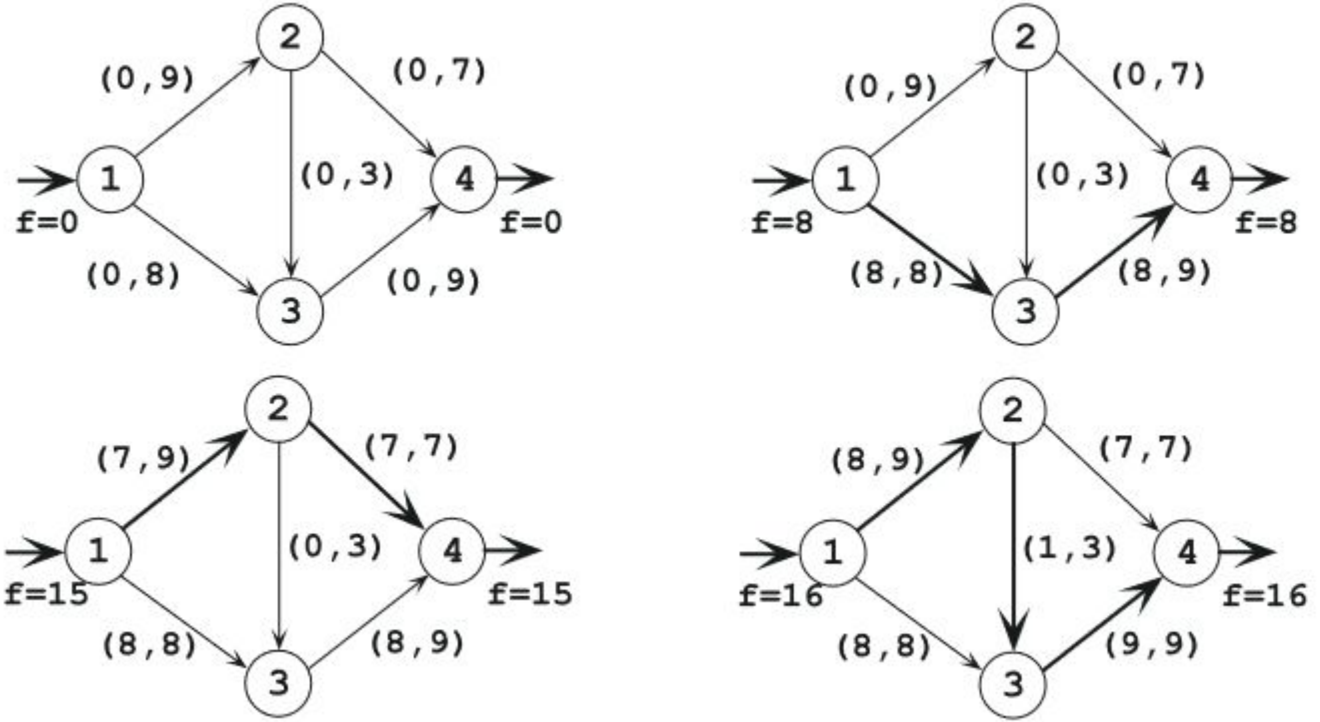
Caminhos de aumento

- Um caminho de aumento é um caminho orientado a partir da fonte para o sumidouro na rede residual, tal que todo arco sobre este caminho possui capacidade residual estritamente positiva (maior do que 0)
- O mínimo destes resíduos é chamado de **capacidade residual de aumento**, uma vez que este representa a quantidade de fluxo que pode viavelmente ser adicionada ao caminho todo

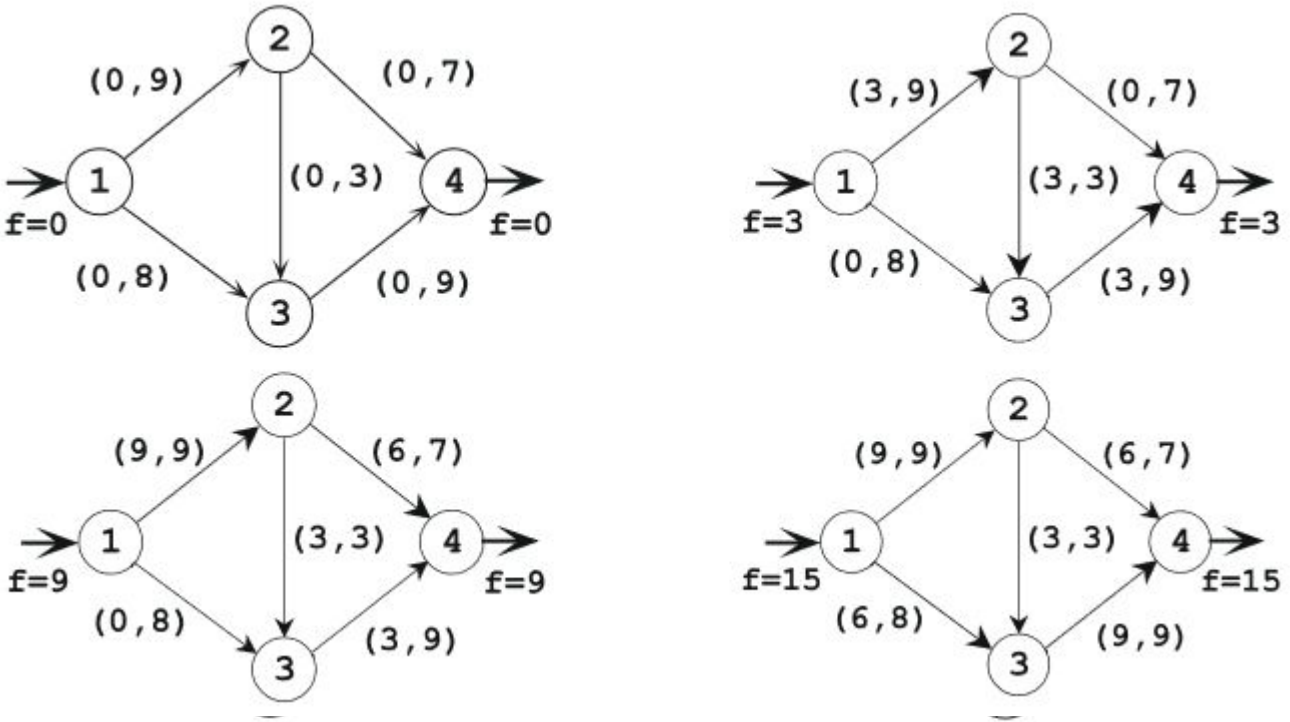
Algoritmo base

1. Injetar um fluxo nulo na fonte
2. Capacidade residual dos arcos = capacidade total dos arcos
3. Buscar um caminho de aumento. Se não existir, foi encontrada uma solução
4. Somar ao fluxo de entrada a capacidade residual de aumento do caminho selecionado
5. Alterar as capacidades residuais dos ramos do caminho selecionado, diminuindo o fluxo injetado
6. Voltar ao passo 3

Algoritmo base



Problema com algoritmo base



Fluxos “negativos”

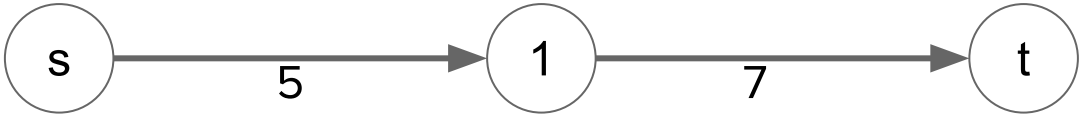
- O algoritmo apresentado não nos leva sempre a uma solução ótima.
- Para isso, temos que dotar o algoritmo da capacidade de “se arrepender”, de deixar de enviar uma certa quantidade de fluxo por uma certa aresta.
- Para isso, vamos inserir “fluxos negativos” na rede, que basicamente são fluxos que atravessam as arestas no sentido contrário a sua orientação.
- Vamos chamar os arcos de fluxo negativo de **arcos reversos**.

Fluxos “negativos”

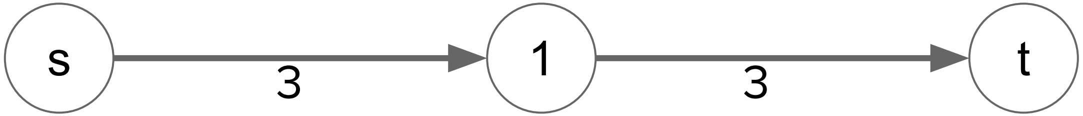
- Com isso, uma rede residual G^R irá conter duas arestas para cada aresta (v, w) de G:
  - (v, w) com capacidade $c^R_{v,w} = c_{v,w} - f_{v,w}$
  - (w, v) com capacidade $c^R_{w,v} = f_{v,w}$

Fluxos “negativos”

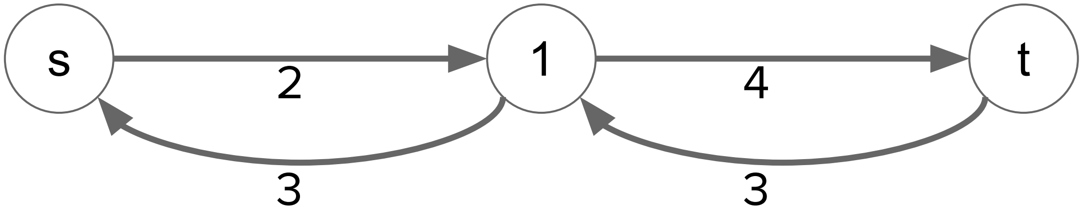
- Grafo capacitado G



- Grafo de fluxo G^F



- Rede residual G^R



Algoritmo de Ford-Fulkerson

Enquanto existirem caminhos entre s e t em G^R

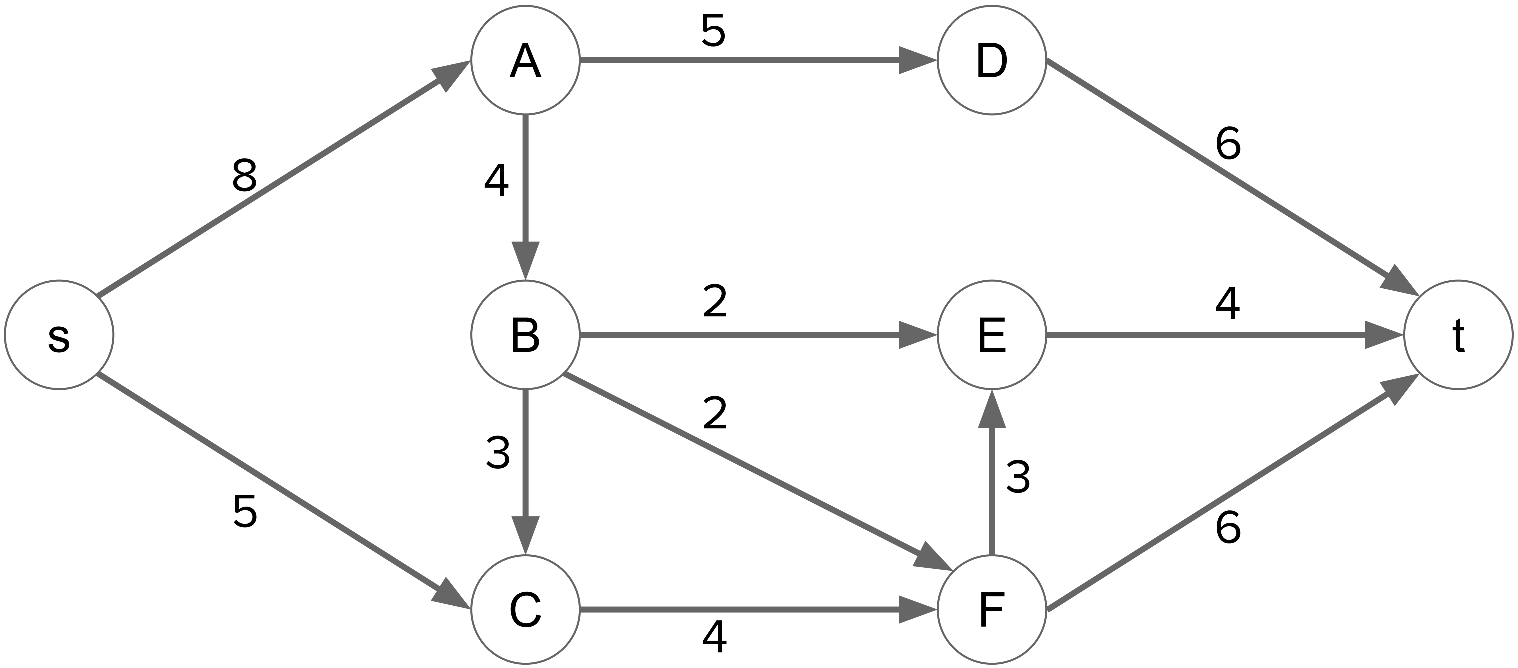
Selecionar um caminho de aumento qualquer em G^R

Determinar o valor mínimo (f) nos arcos desse caminho

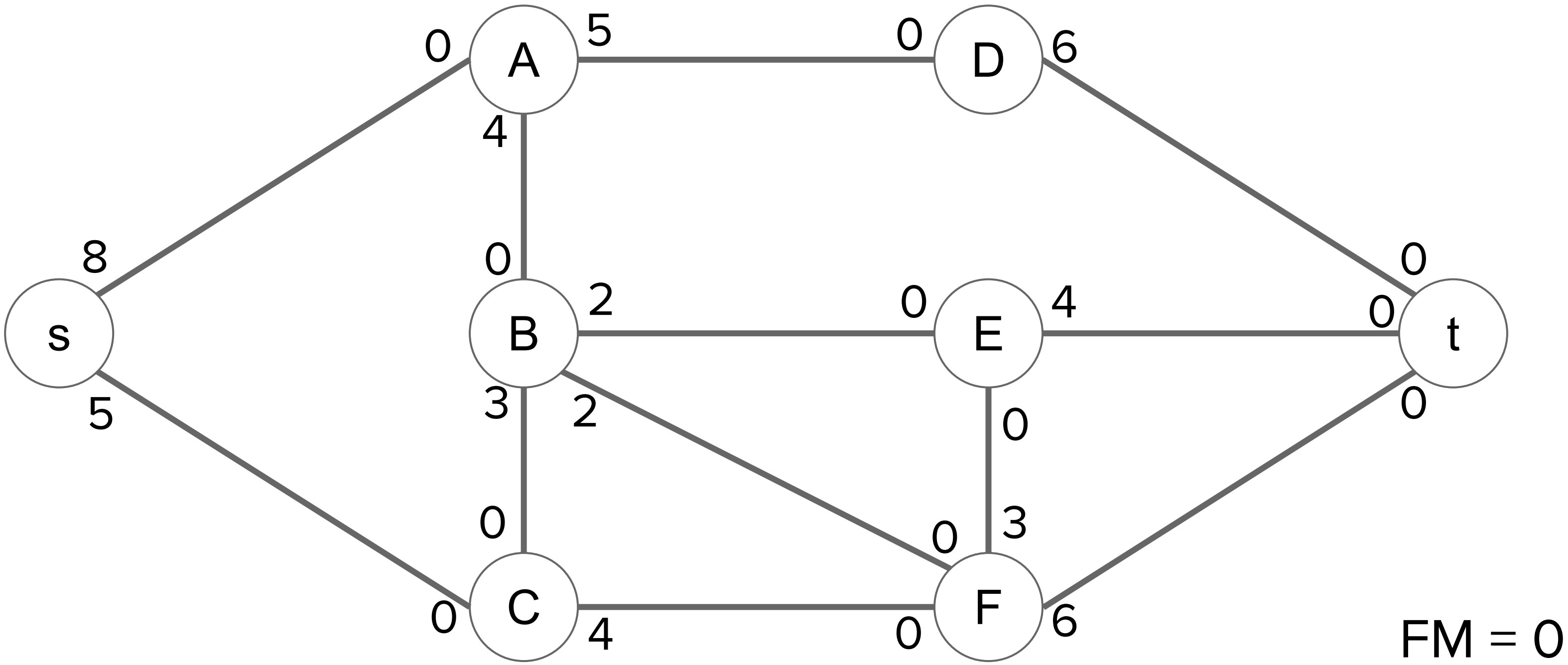
Aumentar esse valor de fluxo (f) a cada um dos arcos respectivos em G^F

Recalcular G^R

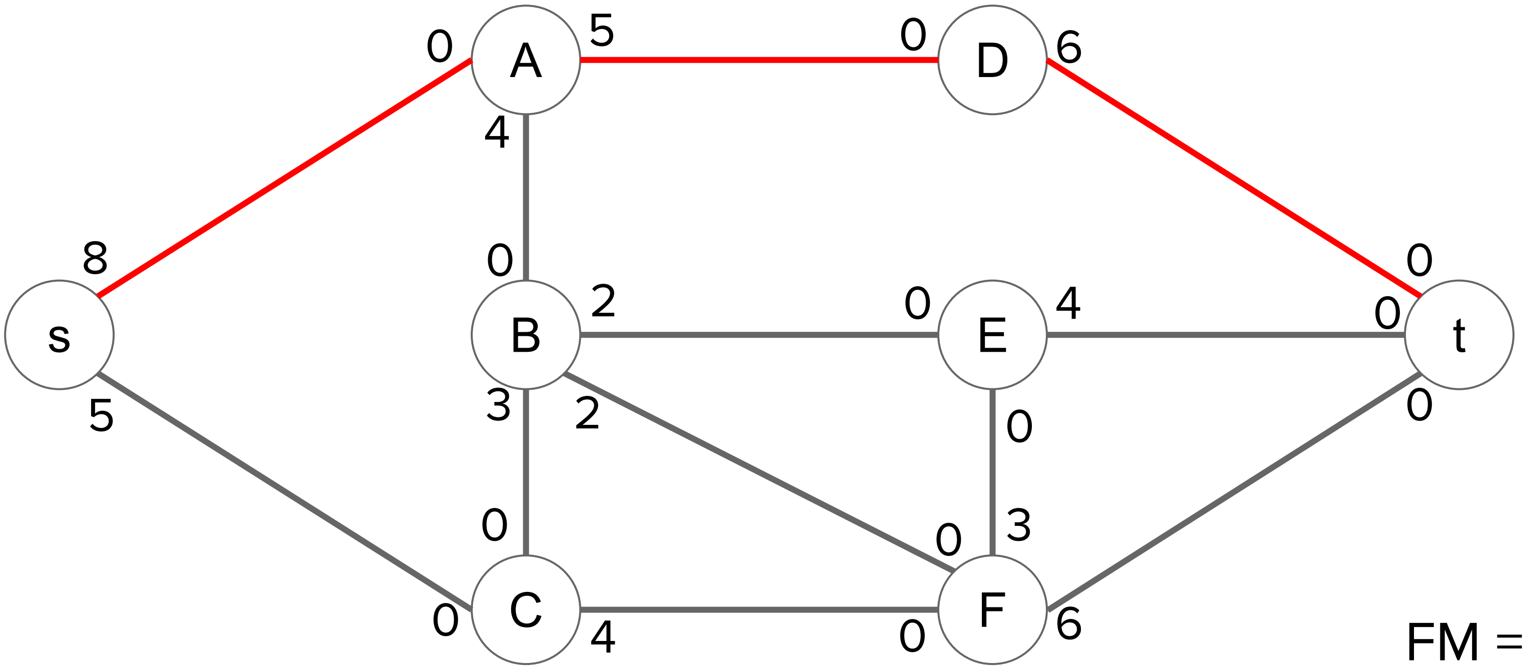
Algoritmo de Ford-Fulkerson



Algoritmo de Ford-Fulkerson

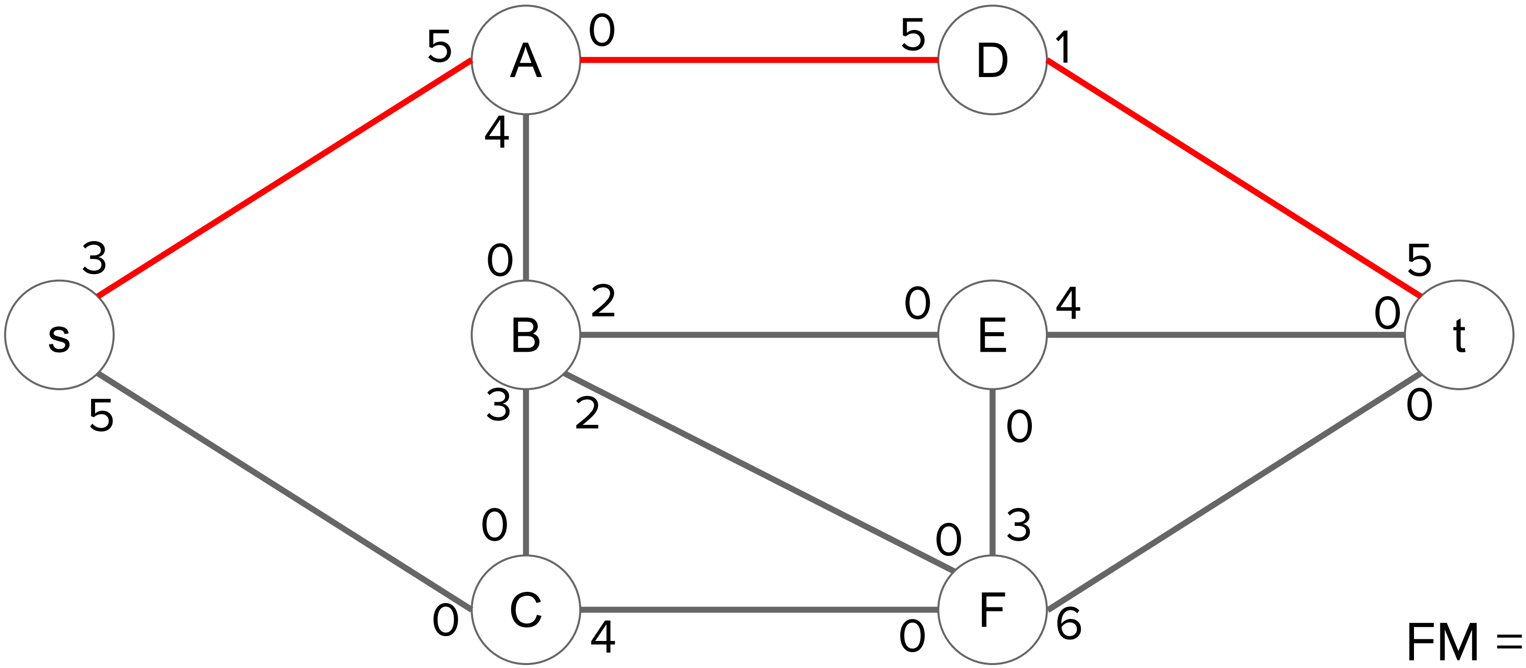


Algoritmo de Ford-Fulkerson



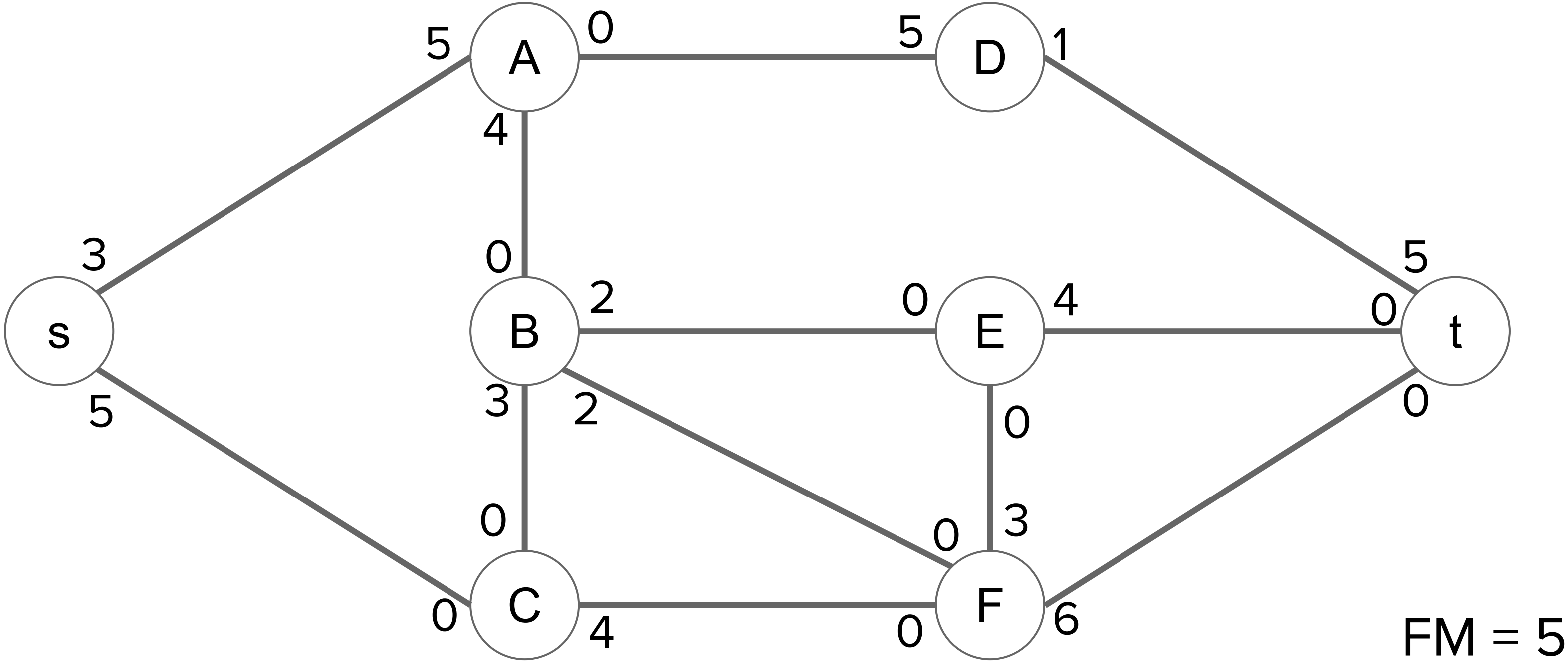
FM = 0

Algoritmo de Ford-Fulkerson

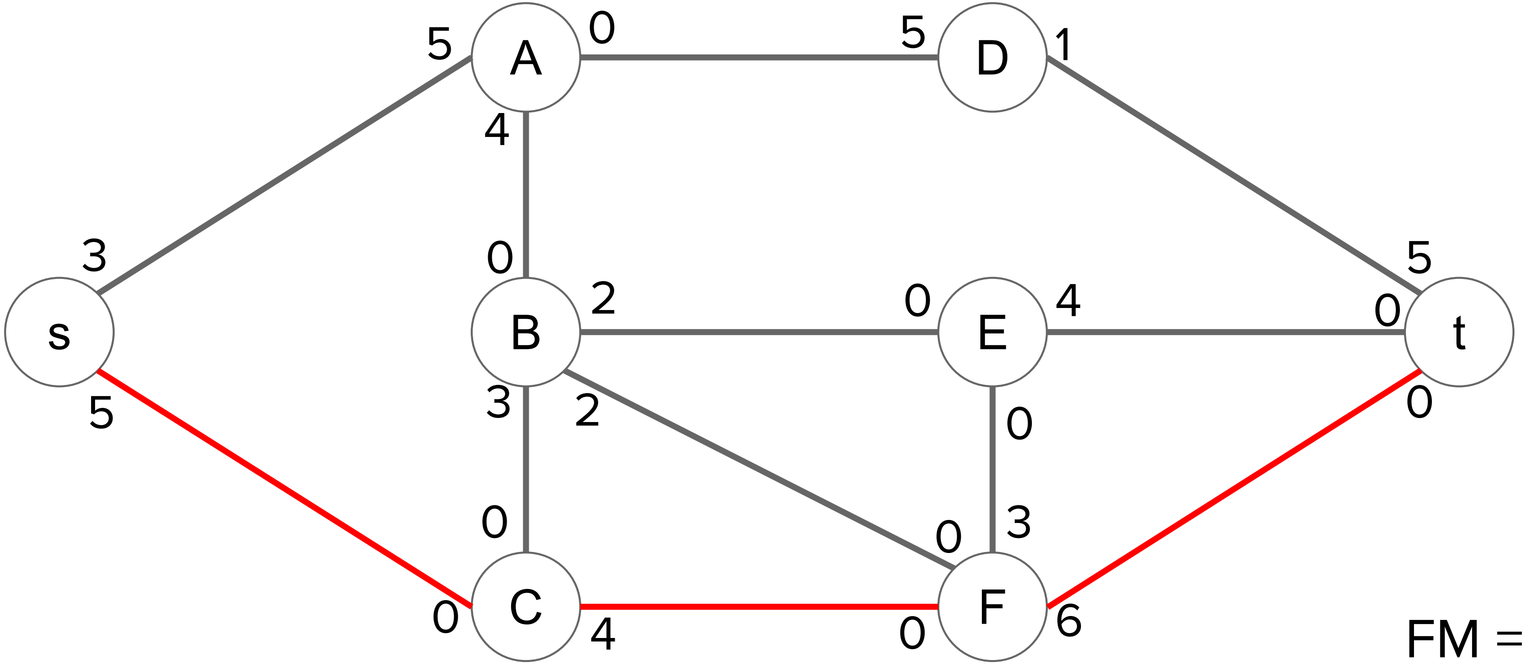


FM = 5

Algoritmo de Ford-Fulkerson

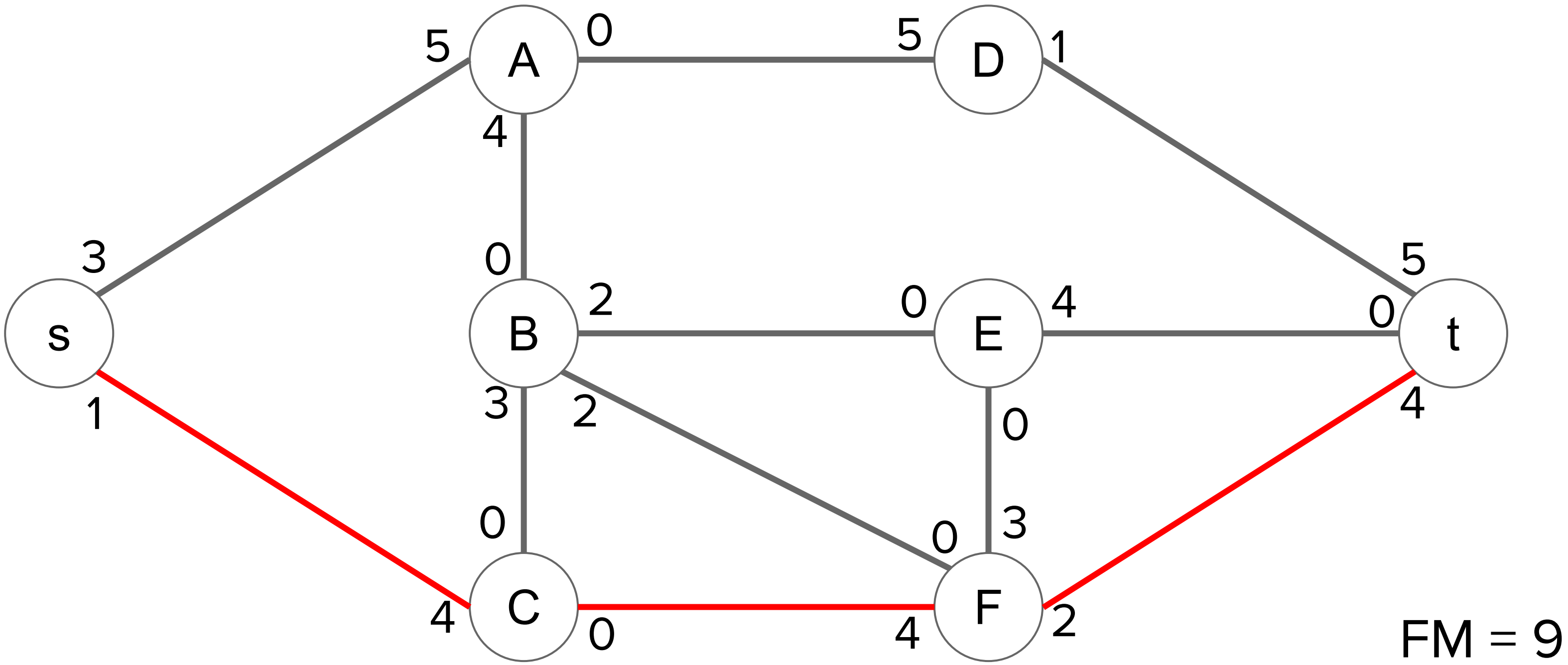


Algoritmo de Ford-Fulkerson

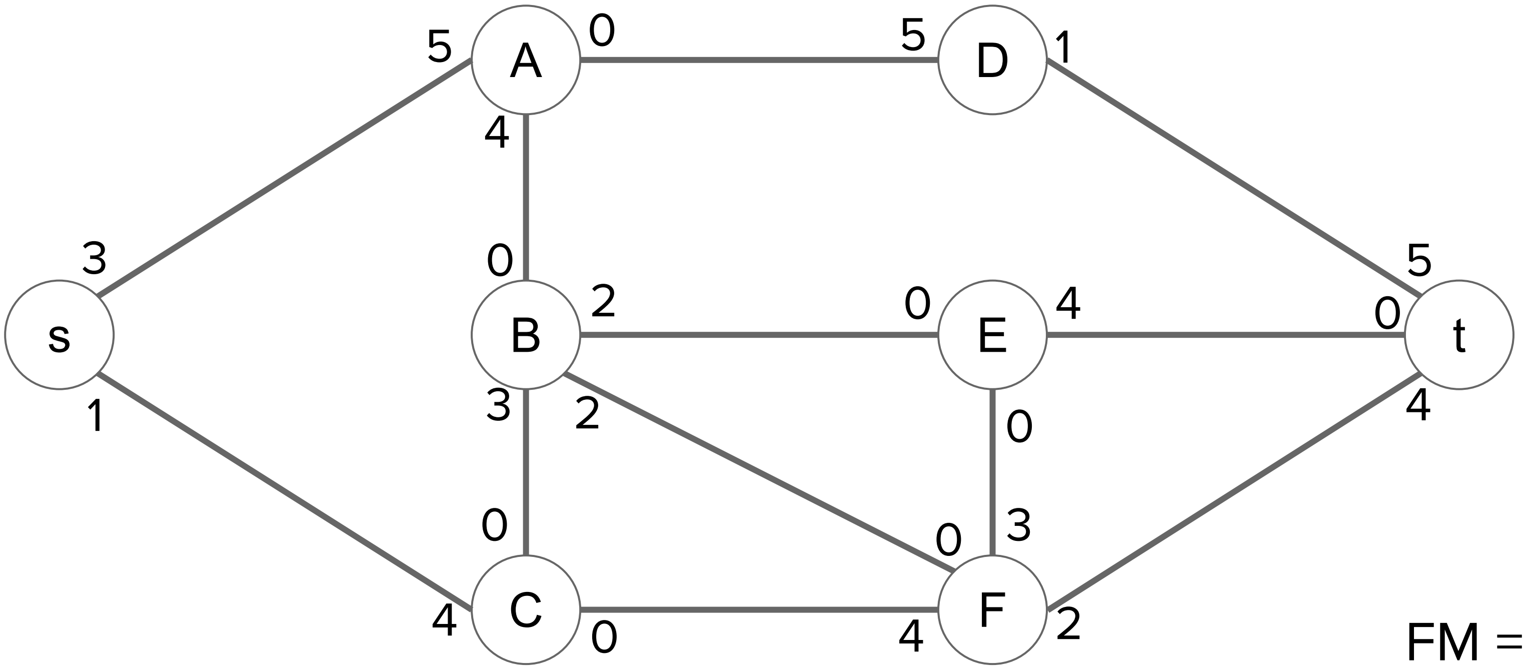


FM = 5

Algoritmo de Ford-Fulkerson

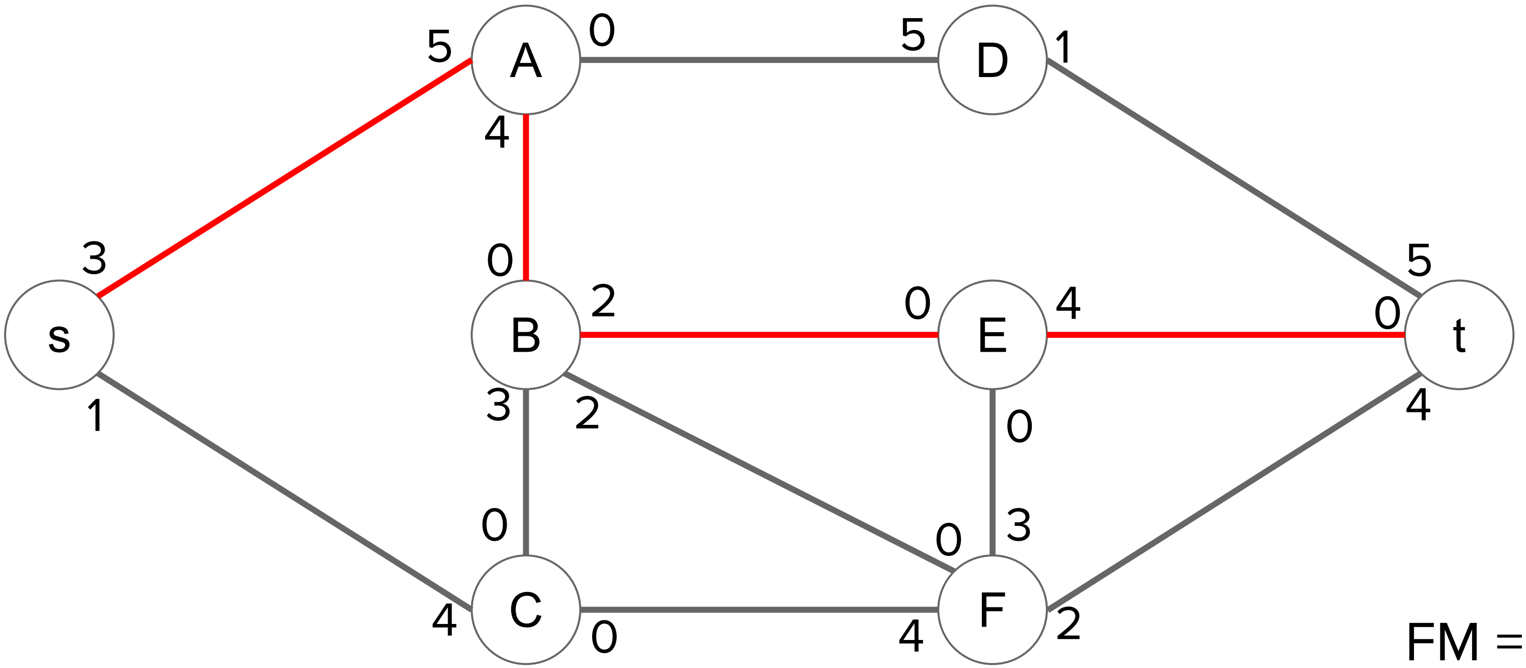


Algoritmo de Ford-Fulkerson



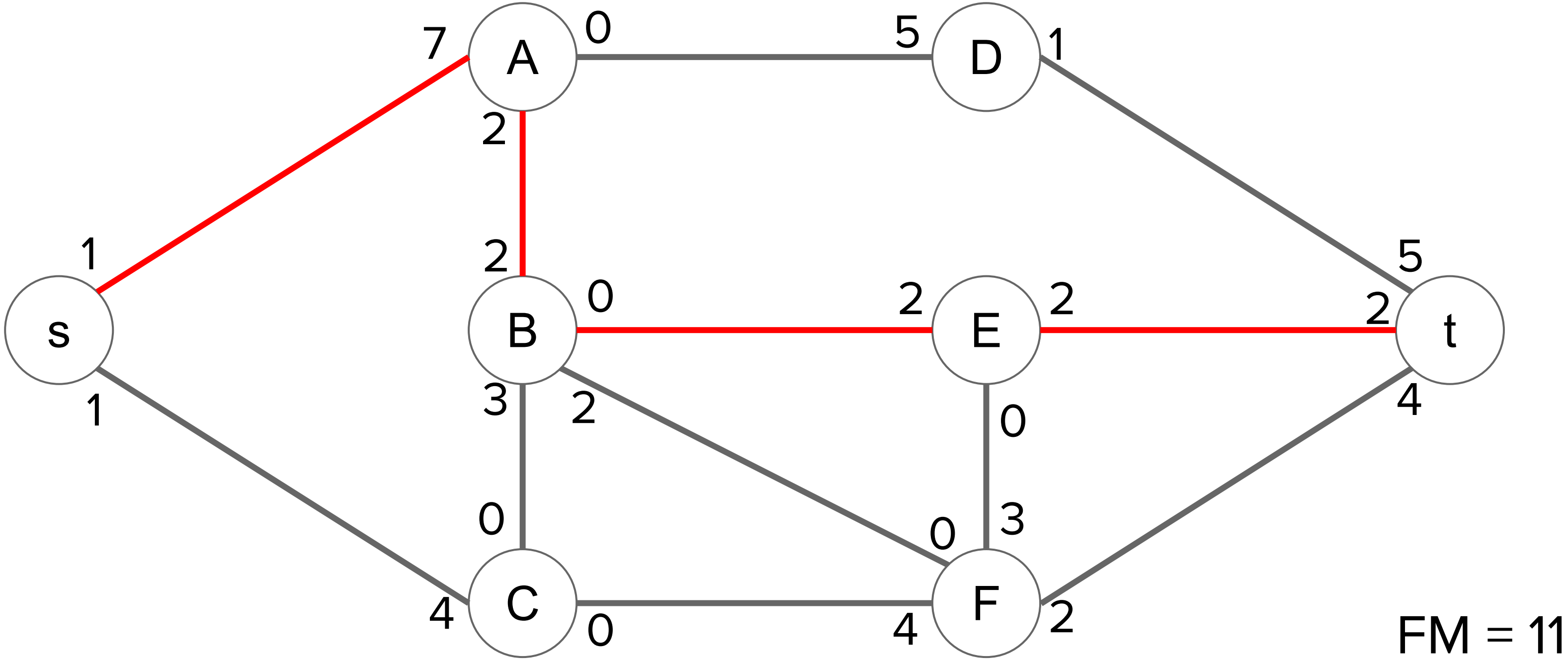
FM = 9

Algoritmo de Ford-Fulkerson

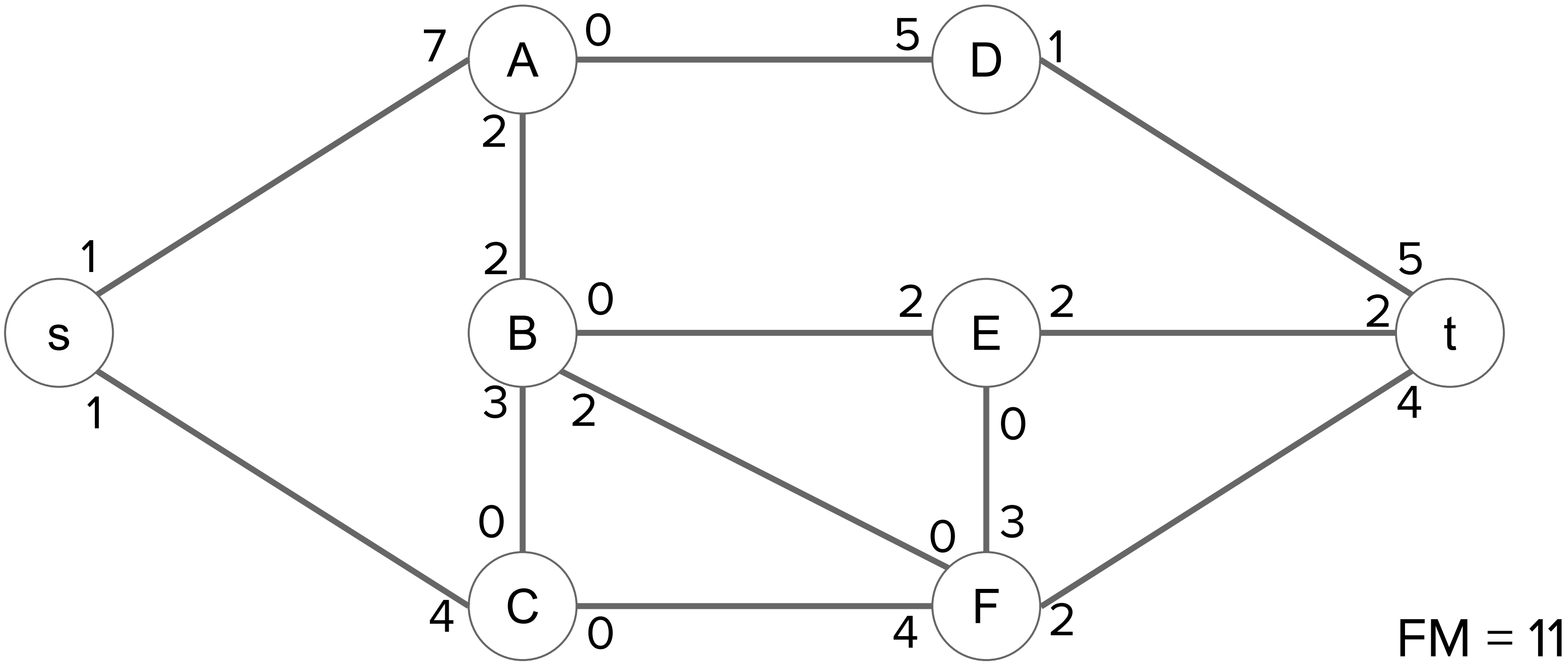


FM = 9

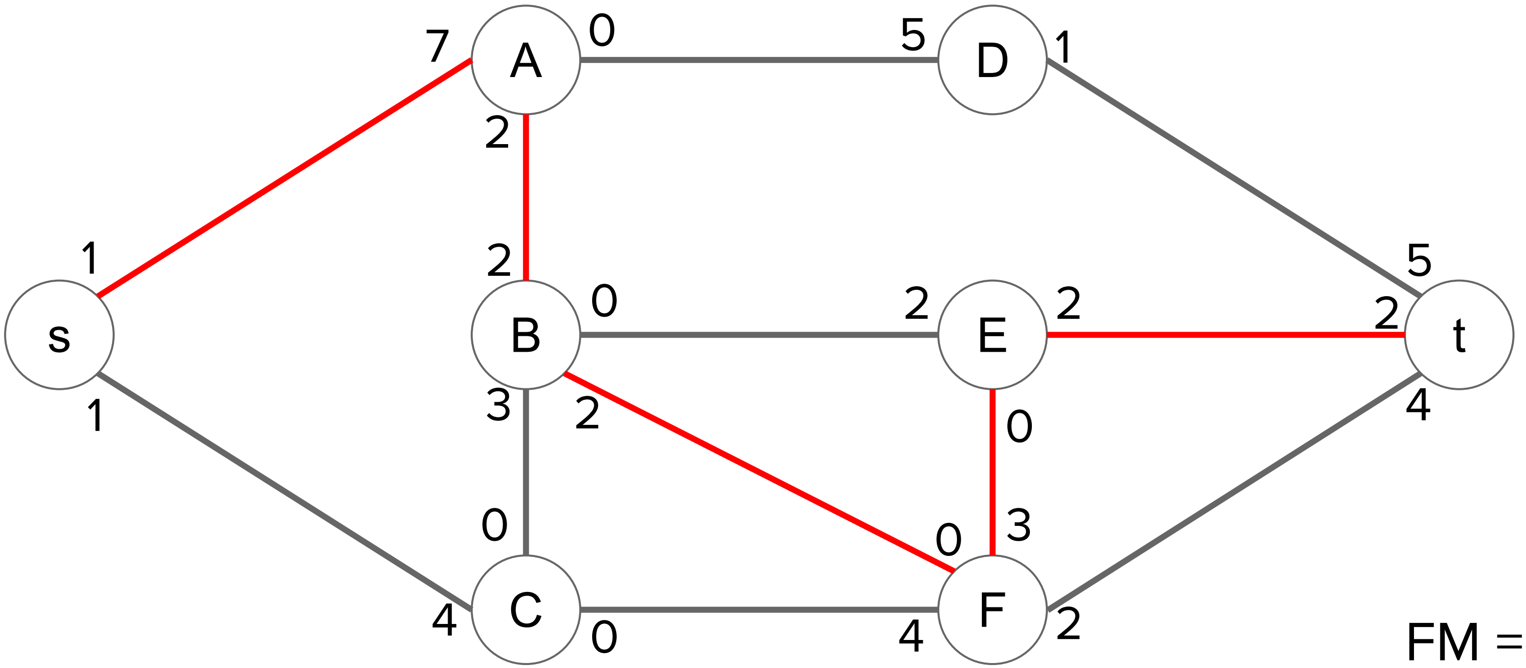
Algoritmo de Ford-Fulkerson



Algoritmo de Ford-Fulkerson

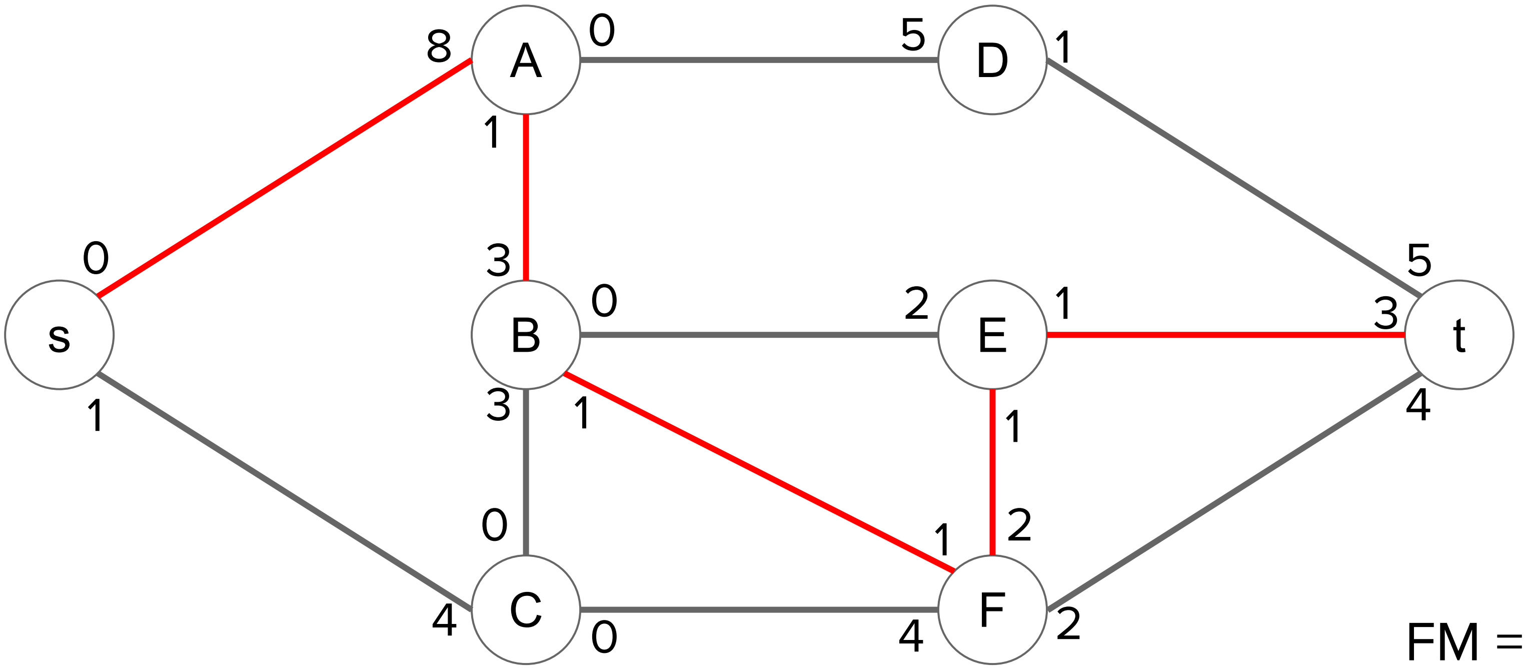


Algoritmo de Ford-Fulkerson



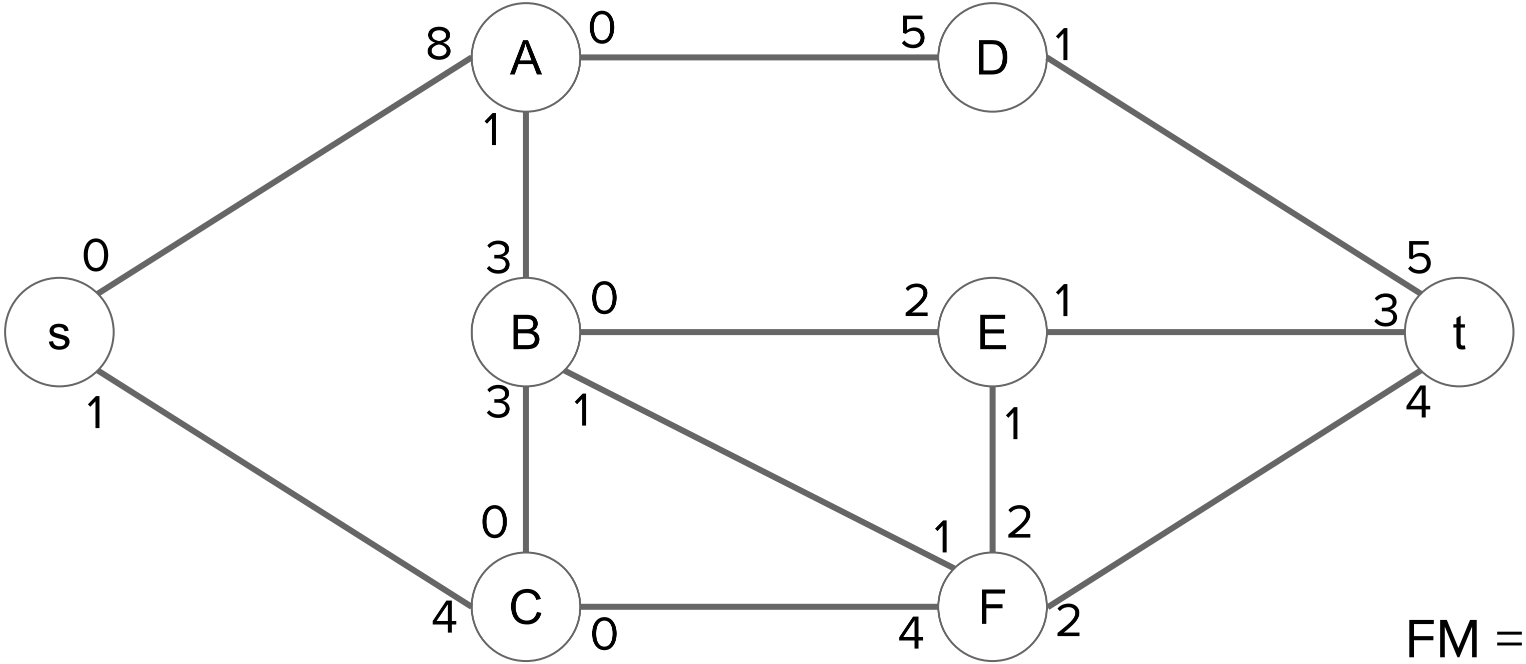
FM = 11

Algoritmo de Ford-Fulkerson



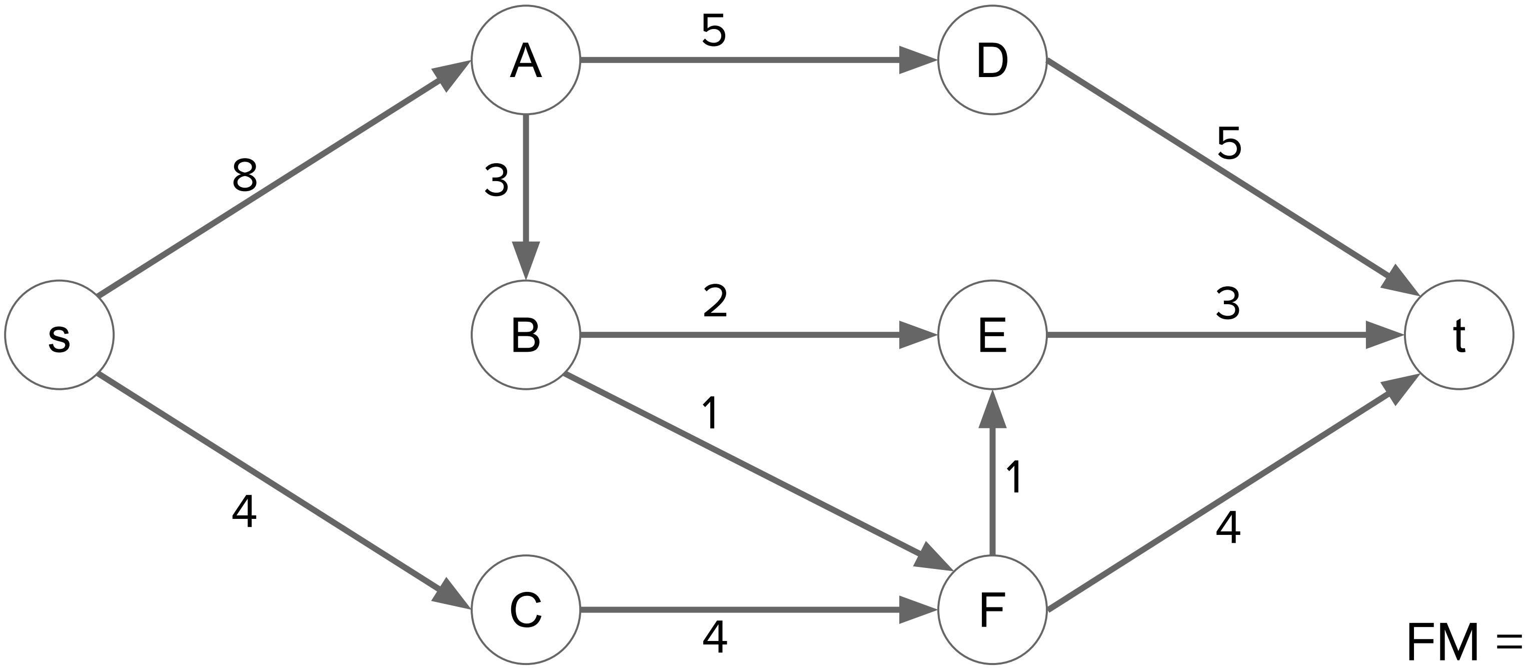
FM = 12

Algoritmo de Ford-Fulkerson



FM = 12

Algoritmo de Ford-Fulkerson



FM = 12

Algoritmo de Ford-Fulkerson

- Se as capacidades forem inteiros e o fluxo máximo M
  - O algoritmo tem a propriedade de integralidade: os fluxos finais são também inteiros
  - O algoritmo pode levar até M iterações (fluxo aumenta pelo menos 1 unidade por iteração)
  - Cada iteração pode ser feita em tempo O(|E|)
  - Complexidade: O(M.|E|)

Algoritmo de Edmonds-Karp

- Versão melhorada do Ford-Fulkerson.
- Em cada iteração do algoritmo de Ford-Fulkerson escolhe-se um caminho de aumento de comprimento mínimo (obtido através de um BFS)
- Complexidade: $O(|E|^2|V|)$
  - Demonstração: https://linux.ime.usp.br/~marcosk/mac0499/files/monografia.pdf

Algoritmo de Dinic

- Versão melhorada do Edmonds-Karp.
- Ao realizar repetidas buscas em largura na rede residual, partindo sempre da fonte até o sorvedouro, ocorre um certo desperdício. A rede residual não se altera muito de uma iteração para a seguinte, e isto significa que podemos estar recalculando a mesma coisa várias vezes quando reiniciamos a busca.
- O algoritmo de Dinic procura aproveitar informações de buscas anteriores.

Algoritmo de Dinic

- A ideia é, ao aplicar a BFS, consideramos a árvore encontrada.
- A partir desta árvore, podemos rotular cada vértice a partir do seu nível (distância a partir da fonte): level[v].
- A partir disso, construímos uma **rede em camadas** (*level graph*) de G^R, onde são mantidas apenas as arestas (v, u) para as quais vale level[u] = level[v] + 1. Esta rede será acíclica

Algoritmo de Dinic

- Através de uma DFS, podemos obter caminhos de aumento nesta rede em camadas. Isso será feito até esgotar todos os caminhos de aumento possíveis.
- Quando os caminhos se esgotam, iniciamos uma nova fase, aplicando uma BFS novamente
- Implementação: https://cp-algorithms-brasil.com/grafos/fluxo4.html

Algoritmo de Dinic

- Através de uma DFS, podemos obter caminhos de aumento nesta rede em camadas. Isso será feito até esgotar todos os caminhos de aumento possíveis.
- Quando os caminhos se esgotam, iniciamos uma nova fase, aplicando uma BFS novamente
- Implementação: https://cp-algorithms-brasil.com/grafos/fluxo4.html
- Complexidade: $O(|V|^2|E|)$
  - Demonstração: https://linux.ime.usp.br/~marcosk/mac0499/files/monografia.pdf

Fluxo de custo mínimo

- Objetivo: transportar **uma certa quantidade F de fluxo** (<= fluxo máximo) da fonte para o sumidouro, com **custo total mínimo**.
    - Neste caso, além da capacidade, cada aresta têm um custo associado (custo de transportar uma unidade de fluxo)
    - Um caso específico do problema é quando **F = fluxo máximo**, ou seja, queremos obter o fluxo máximo, mas com o menor custo possível (caso em que temos mais de uma solução ótima para o fluxo máximo)

Fluxo de custo mínimo

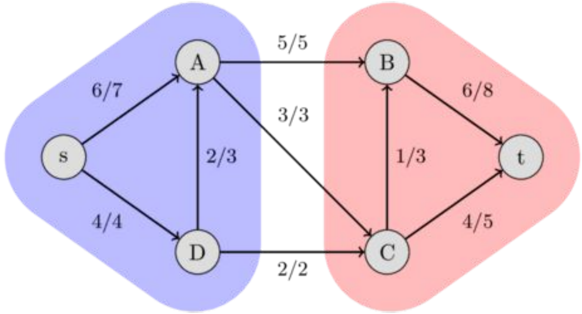
- A solução é muito parecida com o algoritmo de Edmonds-Karp.
- A diferença é que ao invés de procurar o caminho mais curto (em número de arestas) com uma BFS, vamos procurar o caminho mínimo (considerando o custo).
- Para a rede residual, as arestas de retorno (w, v) terão custo inverso as arestas diretas (v,w). Ou seja, $custo_{w,v} = -custo_{v.w}$
- Sendo assim, teremos custos negativos no nosso grafo, o que torna dificil o uso do algoritmo de Dijkstra. Vamos dar preferência ao Bellman-Ford
- Implementação com SPFA: https://cp-algorithms-brasil.com/grafos/fluxo7.html

Fluxo de custo mínimo

```
Para cada aresta e de E[G]
    fluxo[e] = 0
redeResidual = G
custo = 0
repita para sempre
    c = menor caminho de f até s (pela redeResidual)
    se não existir caminho c
        encerrar ciclo
    atualizar custo
    aumentar fluxo ao longo do caminho c
    gerar redeResidual
return fluxo, custo
```

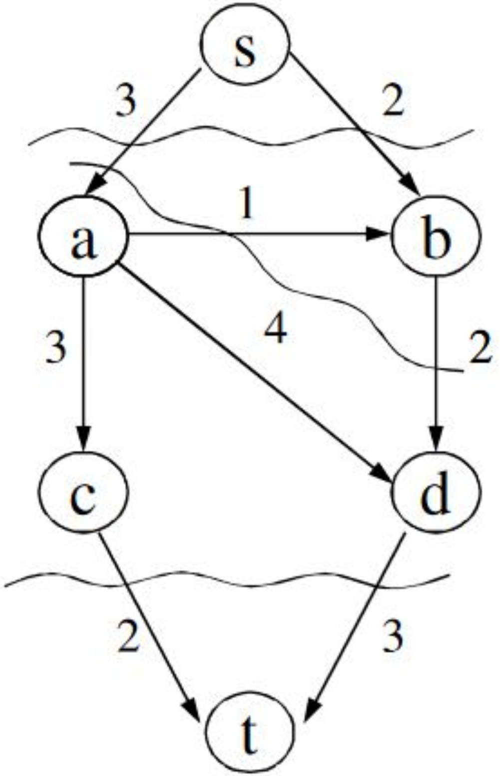
Teorema do Fluxo Máximo - Corte Mínimo

- 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, com fonte em s e sumidouro t, é uma partição de V em conjuntos S e T = V - S, tal que s ∈ S e t ∈ T.
- Ou seja, elimina-se um conjunto de arestas de forma que separe o grafo em dois, sendo que s e t ficam em partes diferentes.
- Um corte mínimo é um corte cuja a soma das capacidades das arestas cortadas é mínima.



Teorema do Fluxo Máximo - Corte Mínimo

- Se quisermos só saber o valor do corte mínimo, podemos usar qualquer algoritmo de fluxo máximo
- Mas se precisarmos saber quais arestas pertencem a este corte? Podemos utilizar o algoritmo de Stoer-Wagner
  - https://linux.ime.usp.br/~marcosk/mac0499/files/monografia.pdf
  - https://basics.sjtu.edu.cn/~dominik/teaching/2016-cs214/presentation-slides/2016-12-06-StoerWagner-BigNews.pdf
  - Implementação: Notebook da UFPE



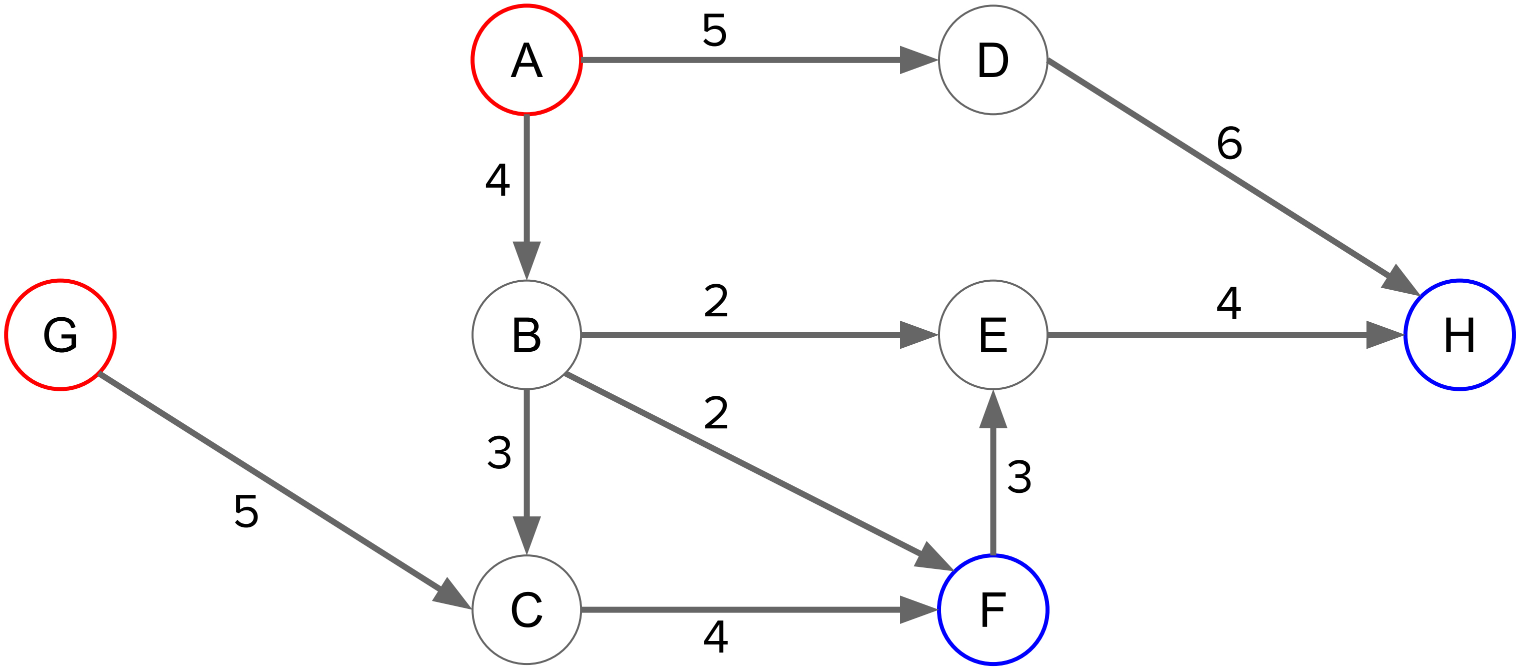
Casos especiais

- Diversos problemas podem ser resolvidos como sendo um problema de fluxo máximo.
- Alguns deles, porém, parecem não se adequar a todas as restrições colocadas até então.
- No entanto, em muitos destes casos conseguimos nos adequar ao problema de fluxo máximo apenas representando o grafo de forma conveniente

Múltiplas fontes e sumidouros

- Um caso especial é quando temos múltiplas fontes e/ou sumidouros.

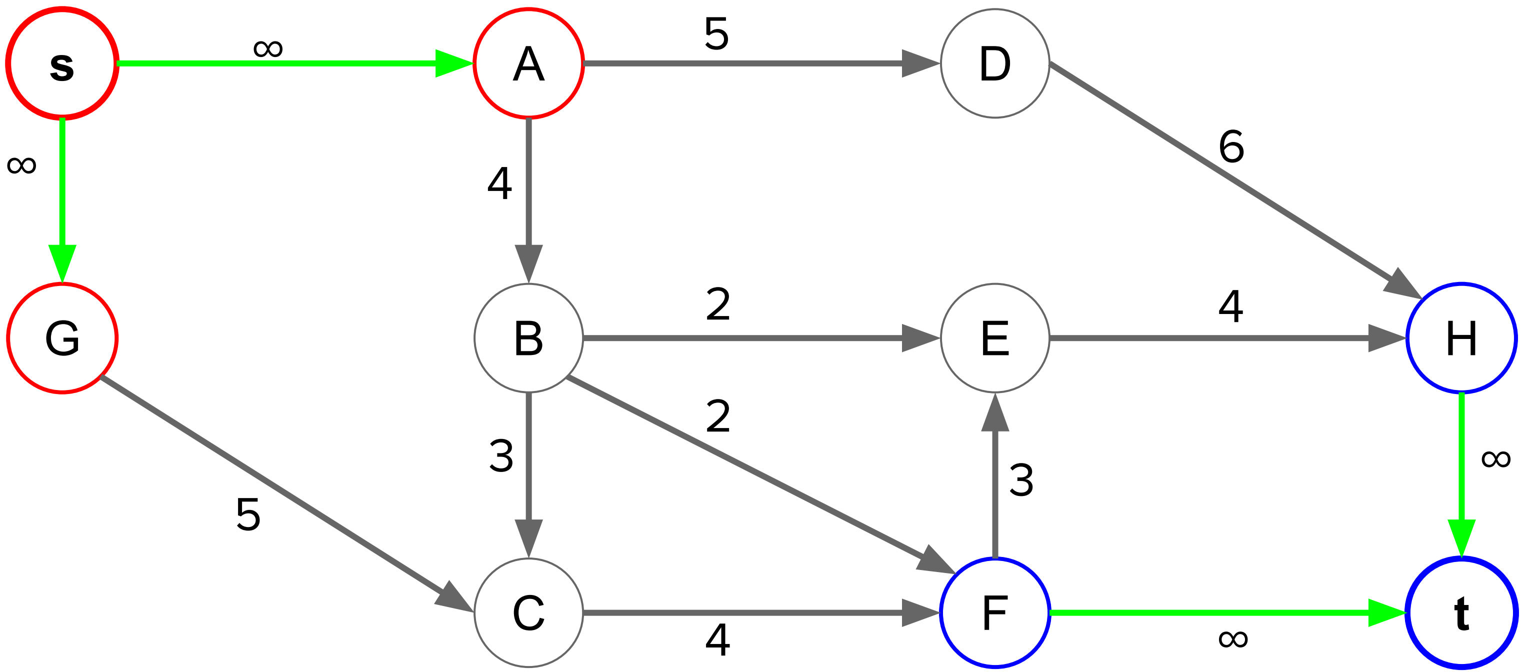
Múltiplas fontes e sumidouros



Múltiplas fontes e sumidouros

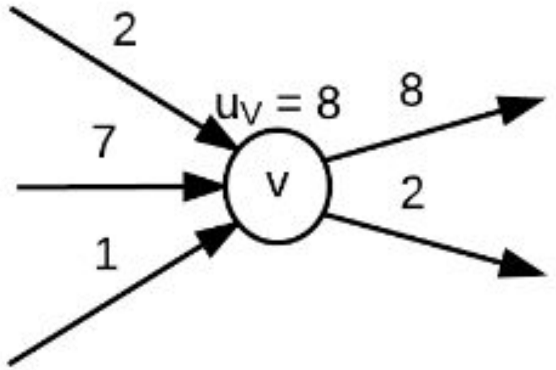
- Um caso especial é quando temos múltiplas fontes e/ou sumidouros.
- Neste podemos adicionar um nó artificial que servirá como fonte, e ligá-lo a todos as nossas fontes originais, com capacidade infinita. De forma análoga, adicionamos um sumidouro artificial.

Múltiplas fontes e sumidouros



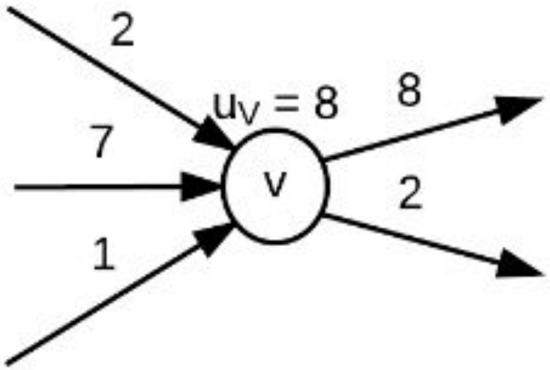
Vértices capacitados

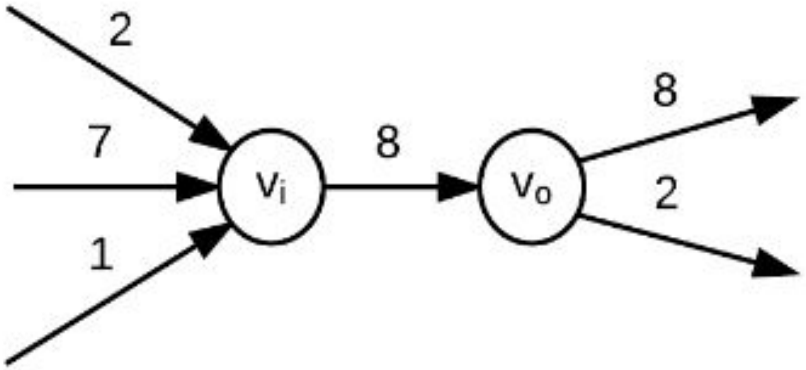
- Outra situação possível é a em que os vértices também são capacitados. Ou seja, além da capacidade das arestas, cada vértice possui um limite de fluxo que pode correr por ele



Vértices capacitados

- Novamente, podemos adequar nosso grafo para não alterar nosso algoritmo original. No caso, da vértice se dividirá em dois. Um vértice de entrada v_i e um vértice de saída v_o, ligados pela aresta (v_i, v_o) com $c_{vi,vo} = u_v$





Referências

Notas de aula de Pesquisa Operacional II com a Profª. Drª. Andréa Carla Gonçalves Vianna

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/flow.html

https://www.ufjf.br/epd015/files/2010/06/fluxo_maximo.pdf

https://web.fe.up.pt/~jfo/ensino/io/docs/IOT_fluxomaxcaminhomin.pdf

http://www.dsc.ufcg.edu.br/~pet/ciclo_seminarios/tecnicos/2010/FluxoMaximoCustoMinimoGrafos.pdf

https://paginas.fe.up.pt/~rossetti/rrwiki/lib/exe/fetch.php?media=teaching:1011:cal:07_2.08_1.grafos5.pdf

Referências

https://cp-algorithms-brasil.com/grafos/fluxo.html

https://cp-algorithms-brasil.com/grafos/fluxo4.html

https://cp-algorithms-brasil.com/grafos/fluxo7.html

https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/flow-FF.html

https://linux.ime.usp.br/~marcosk/mac0499/files/monografia.pdf

https://www.geeksforgeeks.org/dinics-algorithm-maximum-flow/