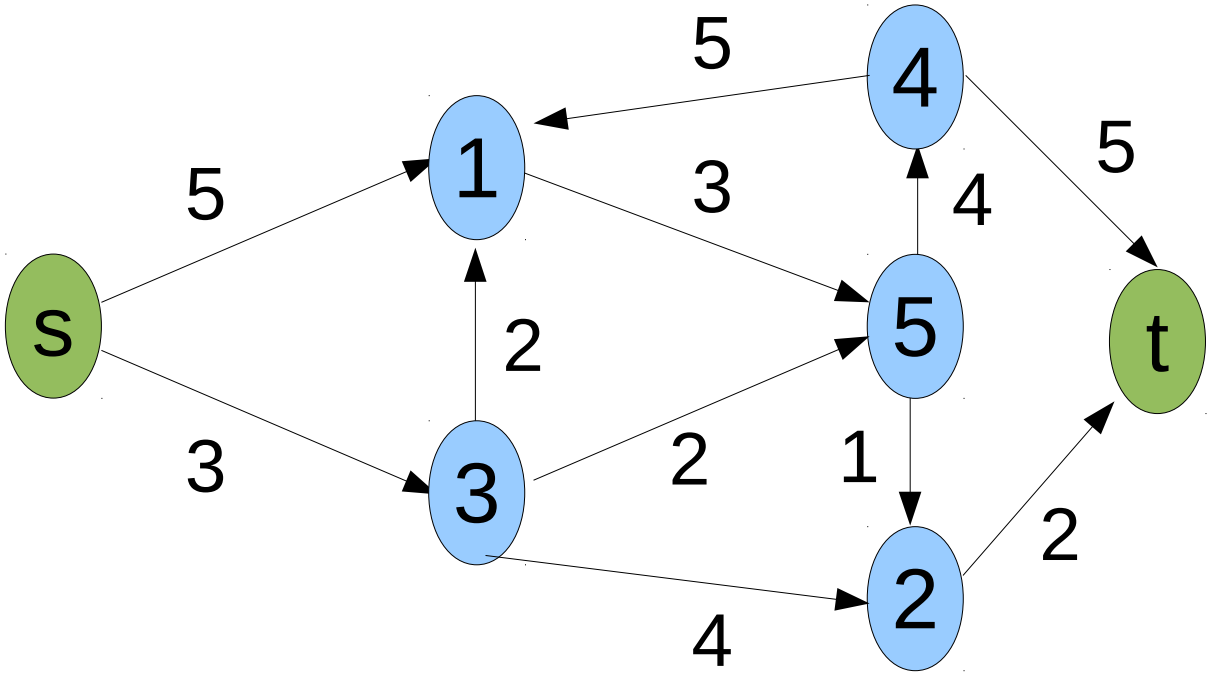
Grafos – Aula 17

Roteiro

- Algoritmo de Ford-Fulkerson
- Análise do algoritmo
- Melhorando algoritmo inicial

Problema do Fluxo Máximo

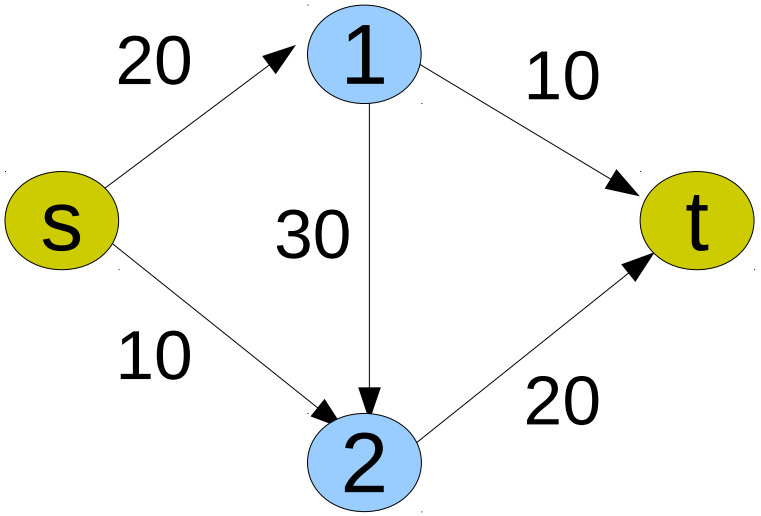
- Dado G=(V,E) com capacidade nas arestas
  - e dois vértices *s* e *t*
- **Problema**: Determinar fluxo máximo entre *s* e *t*



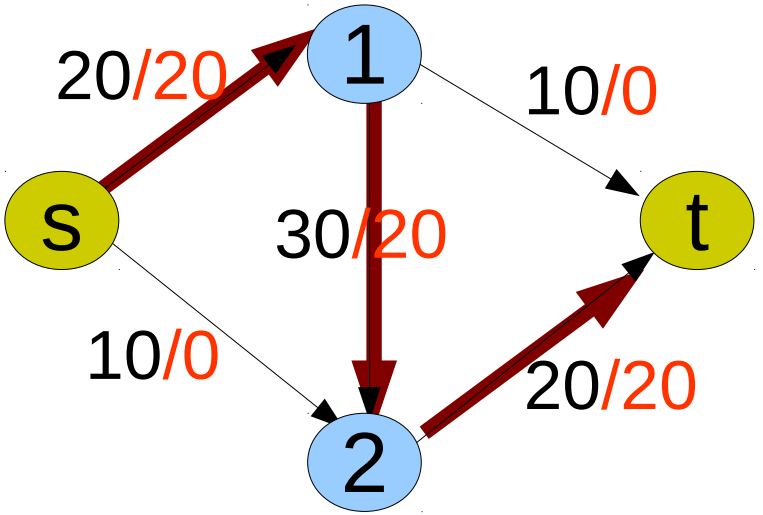
Buscando um Algoritmo

- Ideia para um algoritmo guloso
- Começar com f(e) = 0, para todo e
- Procurar caminho P entre s-t com f(e) < c(e), para toda aresta em P
- Aumentar fluxo em P, saturando o caminho
  - saturar = adicionar fluxo tal que ao menos uma aresta tenha f(e) = c(e)
- Repetir até não encontrar mais caminho

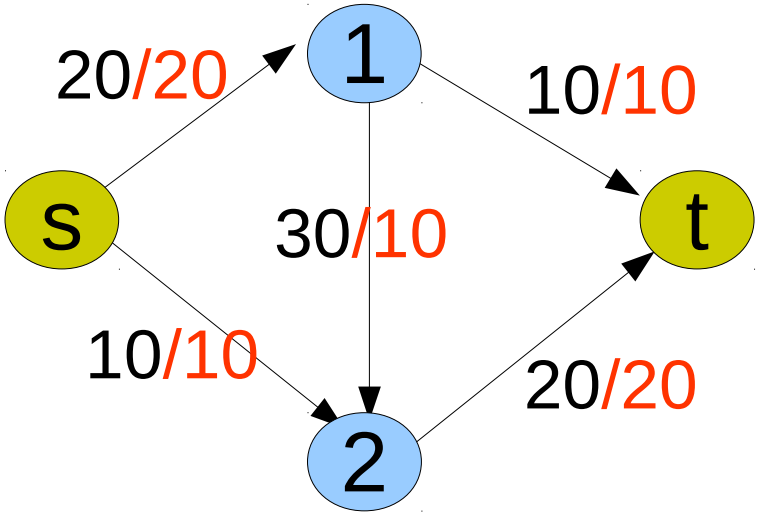
Guloso em Ação



- P = {s,1,2,t}, saturação = 20



- Não há mais caminho, v(f) = 20



- Fluxo máximo = 30
- Guloso não permite redistribuir o fluxo

Grafo Residual

- **Idéia**: dar chance do fluxo voltar atrás!
- Construir um grafo onde isto é possível
  - Grafo Residual
- Arestas originais, do grafo original
  - capacidade c(e), fluxo f(e)
- Arestas residuais, do grafo residual
  - cada aresta original dá origem a duas arestas: original e reversa
  - $e = (u,v)$, $e^R = (v,u)$
  - arestas tem apenas capacidade

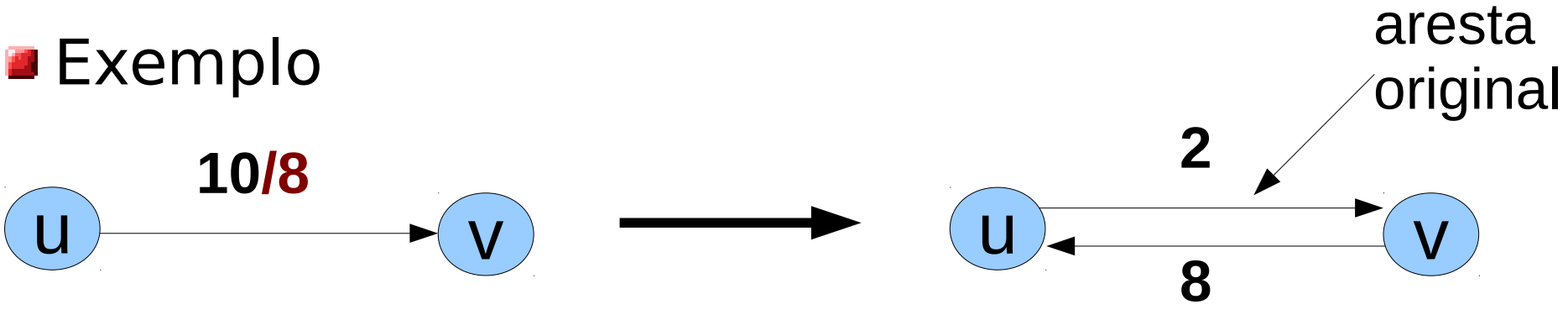
Arestas do Grafo Residual

- Capacidade das arestas
  - capacidade e fluxo na aresta original

$c_R(e) = c(e) - f(e)$, quando e for original

$c_R(e) = f(e)$, quando e for reversa

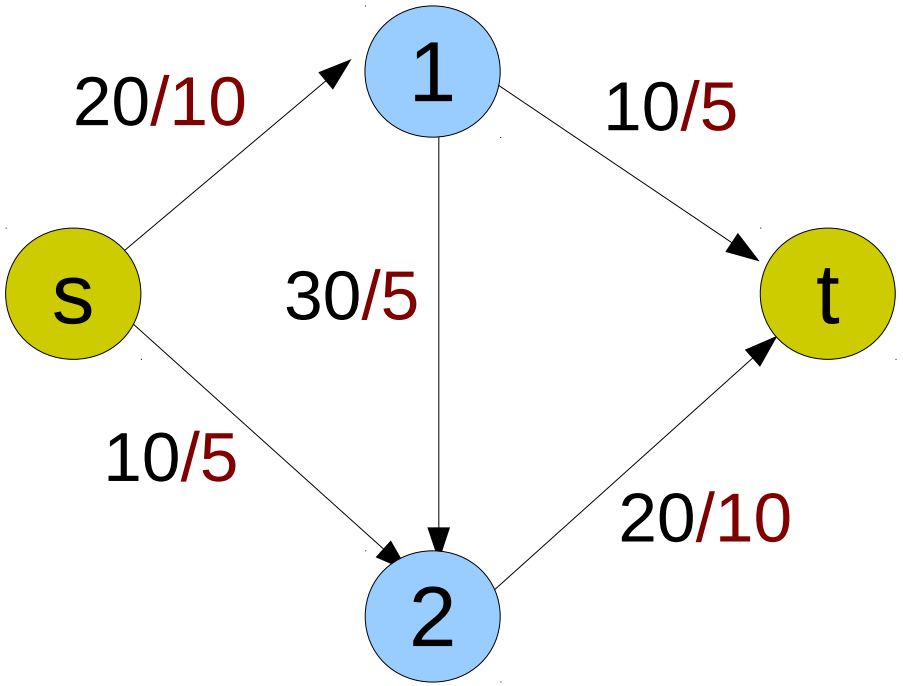
- Exemplo



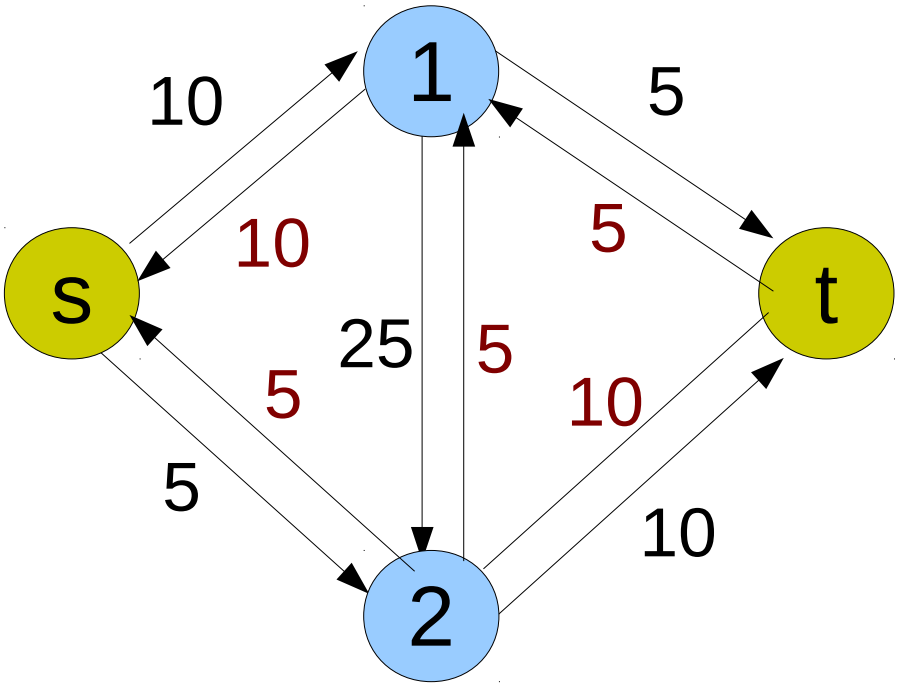
- Grafo residual
  - Vértices e arestas (originais e reversas) com capacidade

Construindo Grafo Residual

- Arestas originais e reversas com capacidade
- Exemplo



Grafo Residual



Aumentando Fluxo

- **Idéia**: aumentar o fluxo de um caminho
  - "saturar" o caminho
- Dado um caminho P, entre s e t no *grafo residual*
  - atualizar fluxo do caminho
- Encontrar gargalo do caminho, b
  - capacidade da aresta de menor capacidade
- Atualizar fluxos (no grafo original)
  - f(e) = f(e) + b, se e for aresta original
  - $f(e^R) = f(e^R) - b$, se e for aresta reversa
  - e^R: aresta original correspondente a aresta reversa

Aumentando Fluxo

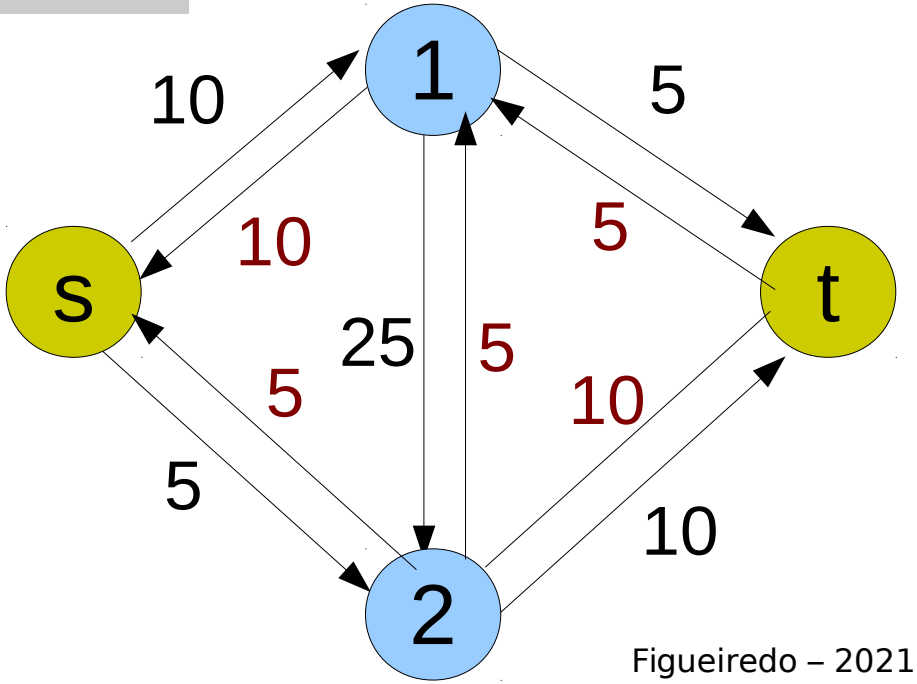
```
Augment(f, c, P) {
    b ← bottleneck(P)
    foreach e ∈ P {
        if (e ∈ E) f(e) ← f(e) + b
        else       f(e^R) ← f(e^R) - b
    }
    return f
}
```

f : fluxo
c : capacidade
P : caminho

- Exemplo:
  P = {s, 2, 1, t}

P possui
aresta reversa

Grafo Residual



Ford-Fulkerson

- **Idéia**: aumentar o fluxo dos caminhos, enquanto for possível
- 1) Inicializar com fluxo 0
- 2) Descobrir um caminho P (no grafo residual)
  - aumentar fluxo deste caminho, via gargalo
- 3) Atualizar grafo residual
  - capacidade das arestas
- 4) Parar quando não houver mais caminho P

Ford-Fulkerson

- Algoritmo: pseudo-código de mais alto nível

```
Ford-Fulkerson(G, s, t, c) {
    foreach e ∈ E   f(e) ← 0
    G_f ← residual graph

    while (there exists augmenting path P) {
        f ← Augment(f, c, P)
        update G_f
    }
    return f
}
```

Análise do Término

- Assumir capacidades inteiras
- Então fluxos e capacidades residuais inteiras
  - Fluxo máximo é inteiro
- C : limitante superior para fluxo máximo
  - C = soma das capacidades de saída de *s*
- **Teorema:** algoritmo termina em no máximo $v(f^*) <= C$ iterações
- **Prova:** cada iteração aumenta valor do fluxo em ao menos uma unidade
  - gargalo é sempre no mínimo, b = 1

Complexidade

- Número de iterações = O(C)
- Complexidade de cada iteração?
- Encontrar caminho P
  - BFS = O(n + m)
- Descobrir gargalo e atualizar fluxos do caminho
  - Caminho mais comprido = O(n)
- Atualizar grafo residual
  - Iterar por arestas (originais + residuais) = O(n + m)
- Assumir grafo conexo: m = Ω(n)
- Cada passo = O(m)

Complexidade

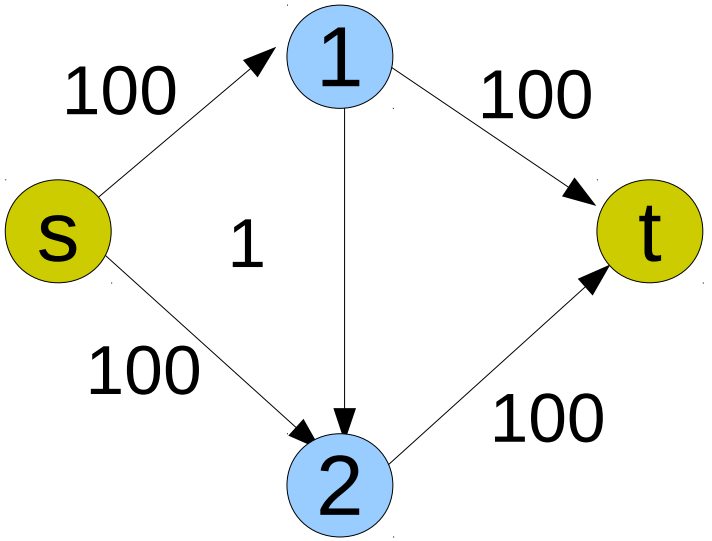
- Complexidade de cada passo: O(m)
- Total de passos: O(C)
- Complexidade: O(mC)
- Algoritmo é *Pseudo-Polinomial*
  - C é o valor do fluxo máximo (e não um número de elementos)
  - C é número com $\log_2 C$ bits

Capacidade Reais

- Assumimos capacidade inteira
- Algoritmo pode não terminar com capacidades reais
  - incremento a cada passo pode ser arbitrariamente pequeno
  - número de iterações pode divergir (salvo precisão numérica)
- Modelos de problemas reais trabalham com capacidade inteiras
  - ex. bps, carros por hora, etc.

Casos Patológicos

- Número de iterações pode ser muito alto
- Escolha patológica dos caminhos para aumento de fluxo
- Exemplo
- Escolher P1 = {s, 1, 2, t}
- Escolher P2 = {s, 2, 1, t}
- Alternar entre eles, cada um com gargalo b = 1
  - 200 iterações, pois C = 200



Melhorando Algoritmo

- **Idéia**: encontrar caminhos de maior capacidade, maiores gargalos
- **Problema:** encontrar caminho de maior gargalo tem complexidade maior que linear
  - aumento do tempo de cada iteração
- **Idéia**: caminhos com gargalos suficientemente grandes
  - reduzir restrição ao longo do algoritmo
- Restringir caminhos do grafo residual

Melhorando Algoritmo

- Δ : parâmetro de escala
- $G_f(\Delta)$: grafo residual restringido
  - arestas com capacidade residual de ao menos Δ
- Algortimo modificado
  - 1) Δ = maior potência de 2, que não seja maior do que C (capacidade de saída de s)
  - 2) Trabalhar com $G_f(\Delta)$ até que não haja mais P
  - 3) Fazer $\Delta = \Delta/2$
  - 4) $G_f(1)$ = Grafo residual convencional

Complexidade

- Número de iterações que reduzem Δ: $\log_2 C$
- Número máximo de caminhos P para um dado valor de Δ
  - primeira iteração: 1
  - em geral, no máximo 2m (pode-se mostrar)
  - encontrar cada caminho: O(m)
- Custo total: $O(m^2 \log C)$
- Outros algoritmos que não dependem de C
  - Edmond-Karp, Ford-Fulkerson usando BFS, $O(m^2n)$
  - King-Rao-Tarjan – O(mn log n)