Teoria dos Grafos
Aula 26

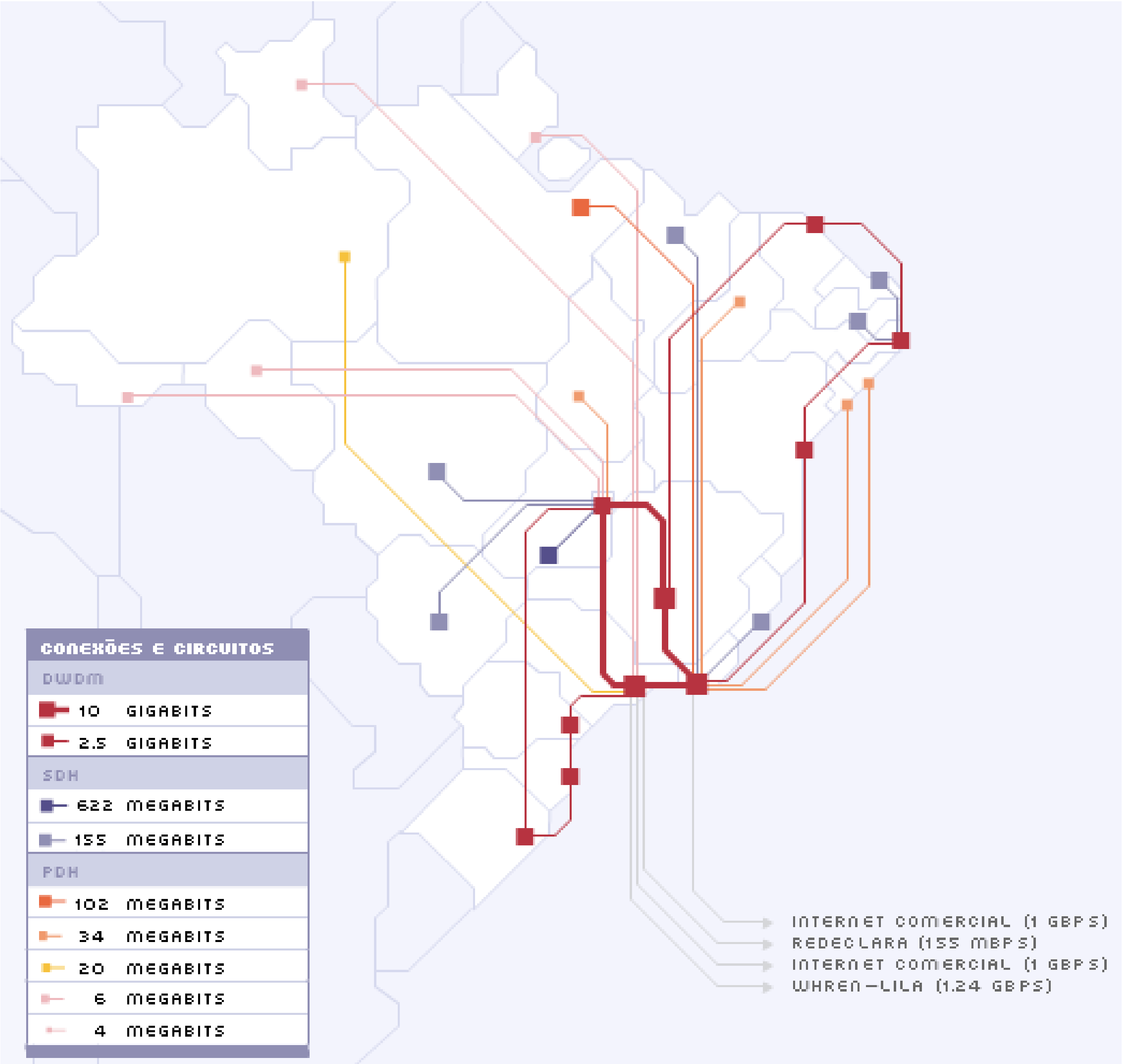
Aula passada

- Redes de fluxo
- Problema do fluxo máximo
- Problema do corte mínimo
- Dualidade

Aula de hoje

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

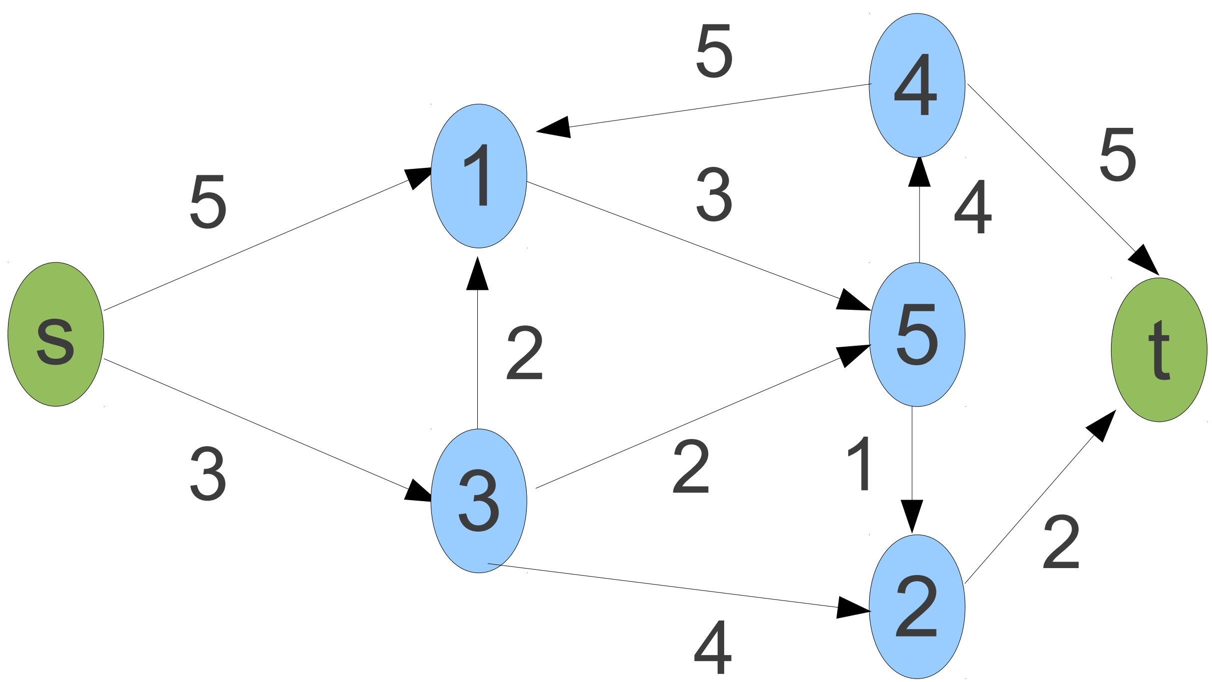
Backbone da RNP



- RNP: Rede Nacional de Pesquisa em Ensino
  - ligação entre instutições nacionais
  - Capacidade dos enlaces (“bits por segundo”)

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*



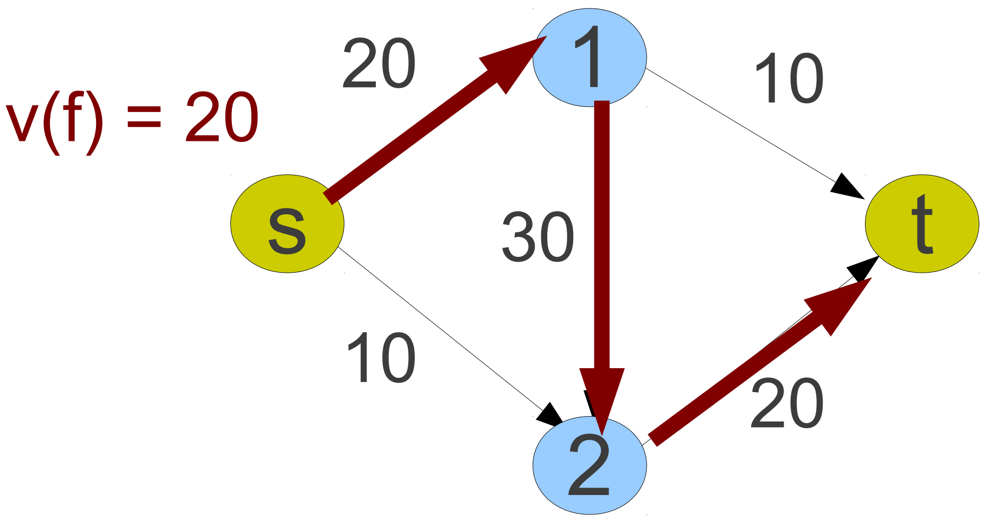
- Fluxo máximo?
- Limitante para fluxo máximo?

Fluxo na Rede

- Como definir um *fluxo* na rede?
- Determinar o fluxo de cada aresta da rede
- Função f : E → R , com restrições
- 1) Capacidade
  - Fluxo em uma aresta menor que capacidade
- 2) Conservação
  - Fluxo que entra igual ao fluxo que sai
  - Fluxo que sai da origem igual fluxo que entra no destino
- Valor do fluxo *f*
  - quantidade de fluxo saindo da origem

Buscando um Algoritmo

- Algoritmo guloso
- Começar com f(e) = 0, para todo e
- Procurar caminho P entre s-t com f(e) < c(e), para todo e em P
- Aumentar fluxo em P
- Repetir até não conseguir mais



- Problema: fluxo não volta atrás!

Grafo Residual

- **Idéia**: dar chance do fluxo voltar atrás!
- Construir um grafo onde isto é possivel
  - Grafo residual
- Arestas (direcionadas) originais: e = (u, v)
  - capacidade c(e), fluxo f(e)
- Arestas residuais (do grafo residual)
  - dois tipos: originais e reversas
  - $e = (u,v)$, $e^R = (v,u)$
  - manter os dois tipos no grafo residual

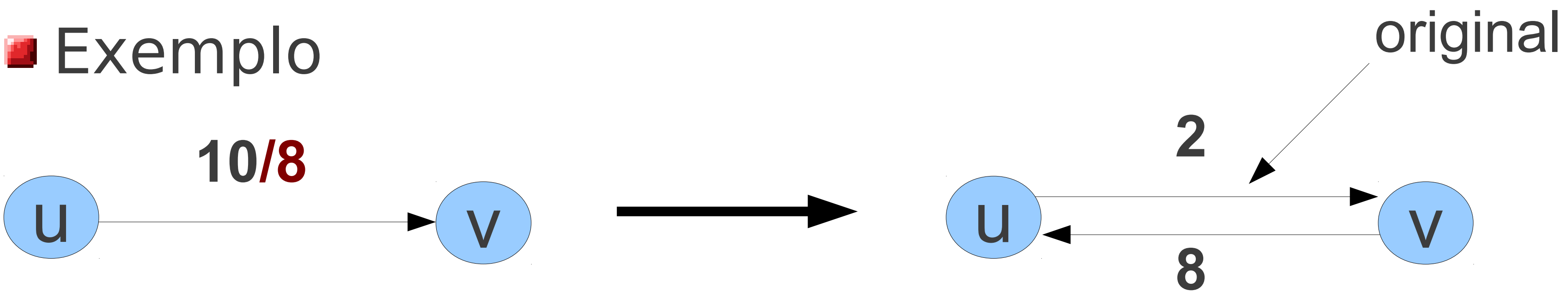
Arestas do Grafo Residual

- Capacidade das arestas
  - dado grafo original e um fluxo f

$$c_f(e) = c(e) - f(e) \quad , \text{ quando e for original}$$

$$c_f(e) = f(e) \quad , \text{ quando e for reversa}$$

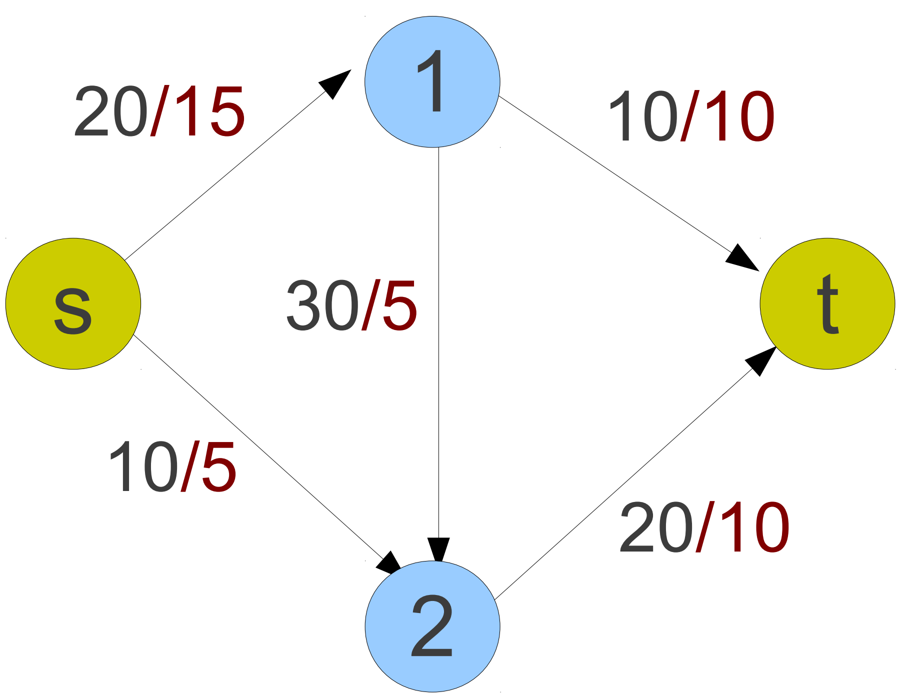
- Exemplo



- Grafo residual
  - Todos os vértices, mais arestas (originais e reversas) com capacidade

Construindo Grafo Residual

- Vértices iguais ao original
- Arestas originais e reversas com capacidade
  - apenas quando capacidade > 0
- Exemplo



Aumentando Fluxo

- **Idéia**: aumentar o fluxo de um caminnho
  - “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
  - $f(e) = f(e) + b$, se e for aresta original
  - $f(e^R) = f(e^R) – b$, se e for aresta reversa

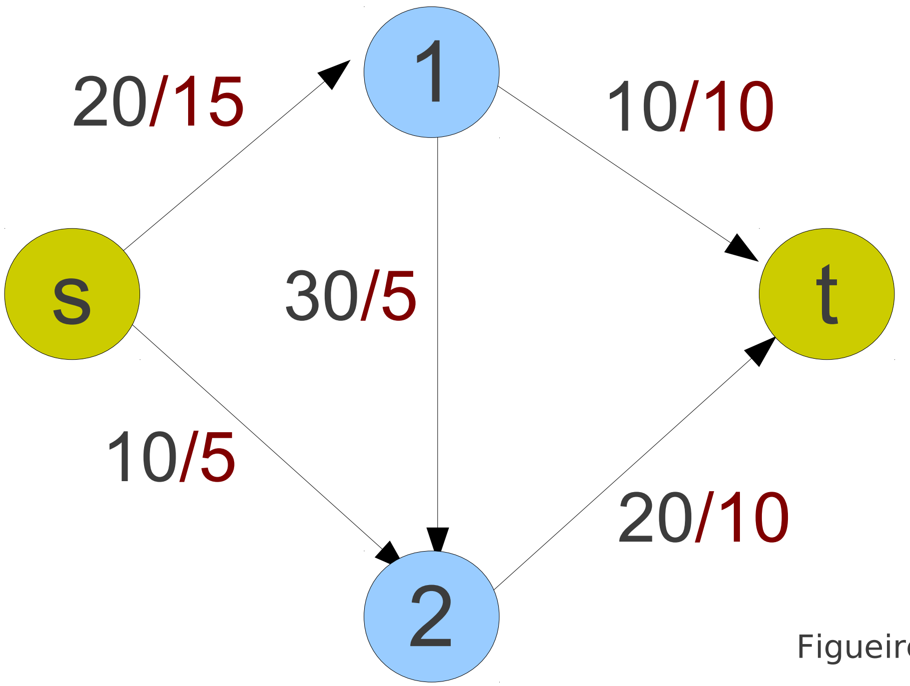
Aumentando Fluxo

```
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, 1, t}

P não possui
aresta reversa



Ford-Fulkerson

- **Idéia**: aumentar o fluxo dos caminhos, enquanto for possivel
- 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

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

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

- Exemplo?

Análise do Término

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

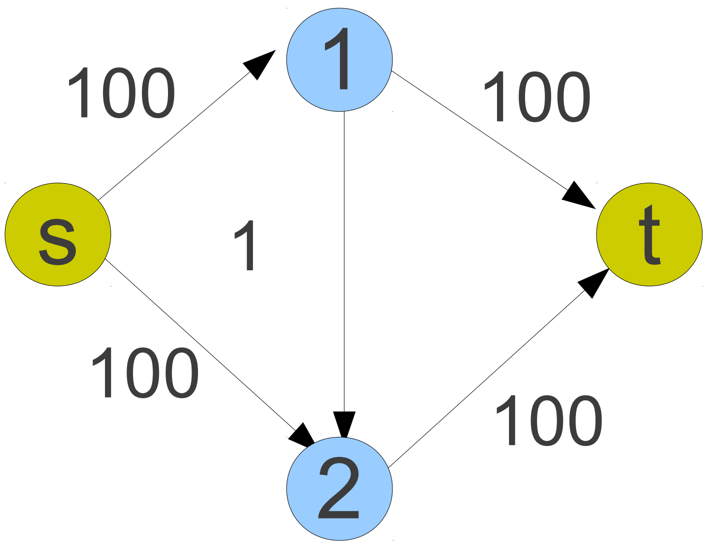
- Complexidade de cada passo: O(m)
- Total de passos: O(C)
- Complexidade: O(mC)
- Algoritmo Polinomial?
- *Pseudo-Polinomial*
  - C é um número com $\log_2 C$ bits

Capacidade Reais

- Assumimos capacidade inteira
  - Onde? Para que?
- Algoritmo pode não terminar com capacidades reais
  - Incremento a cada passo pode ser arbitrariamente pequeno
  - Número de iterações pode divergir
- 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...
  - 200 iterações, pois C = 200



Melhorando Algoritmo

- **Idéia**: encontrar caminhos de maior capacidade, maiores gargalos
- Problema: caminho com maior gargalo (entre todos) pode ser difícil encontrar
  - 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)
- Custo total: $O(m^2 \log C)$
- Polinomial?
- Outras variações que não dependem de C
  - Custo O(mn)