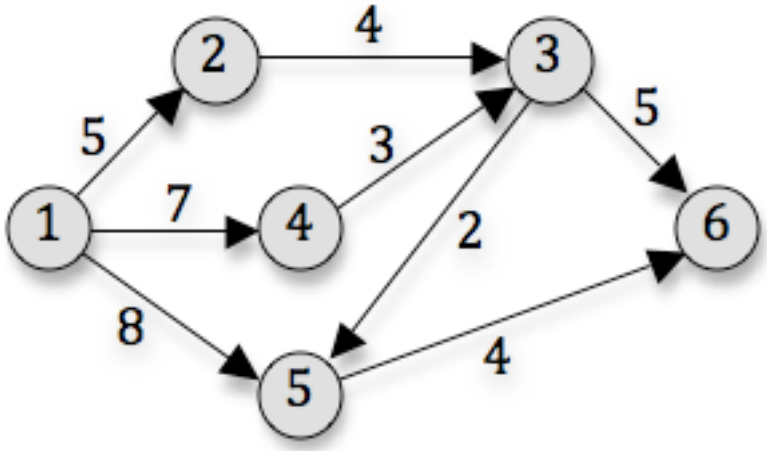
Redes

- Uma rede é um caso especial de digrafo, em que existe um vértice especial (**fonte**), no qual não incidem arcos, e um vértice especial (**destino**), do qual não saem arcos. Numa rede, o peso de um arco é chamado **capacidade** do arco.

Exemplo:



Nesta rede, o nó **1** é a fonte e nó **6** é o destino.

- Considere que os nós são estações de bombeamento de água e os arcos são tubulações por onde a água deve escorrer. Um problema clássico em redes é o **problema de fluxo máximo**: qual é a máxima quantidade de água que pode ser transferida da fonte para o destino nesta rede?

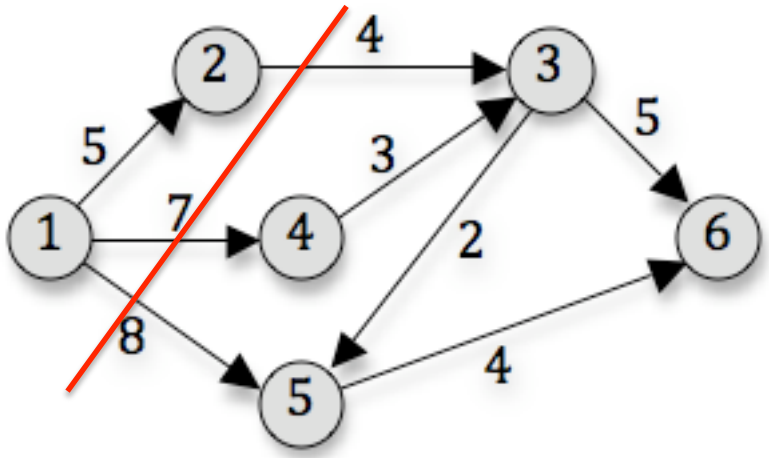
Fluxo máximo em redes

- O **fluxo** em uma rede R = (V, A) pode ser definido como uma função f : A → R, tal que:
  - O fluxo em qualquer arco a ∈ A não pode ser maior do que a capacidade deste arco;
  - O fluxo total que chega em um vértice v ∈ V é igual ao fluxo total que sai deste vértice, onde v não é nem a fonte nem o destino.
- Sejam **f** a fonte e **d** o destino de uma rede R = (V, A). Seja $V = V_f \cup V_d$, tal que $f \in V_f$ e $d \in V_d$. Define-se um **corte** em R, separando **f** de **d**, como o conjunto de arcos (uv) de R tais que $u \in V_f$ e $v \in V_d$.
- **Exemplo**: Sejam $V_f = \{1, 2\}$ e $V_d = \{3, 4, 5, 6\}$. Então o conjunto C = { (23), (14), (15) } é um **corte** desta rede. Isto significa que se as arestas de C forem cortadas, não há como escoar fluxo de **f** para **d**.

Fluxo máximo em redes

- Seja C um corte. Define-se a **capacidade do corte** C como a soma das capacidades de suas arestas.

Exemplo:



$V_f = \{1, 2\}$

$V_d = \{3, 4, 5, 6\}$

$C = \{ (23), (14), (15) \}$

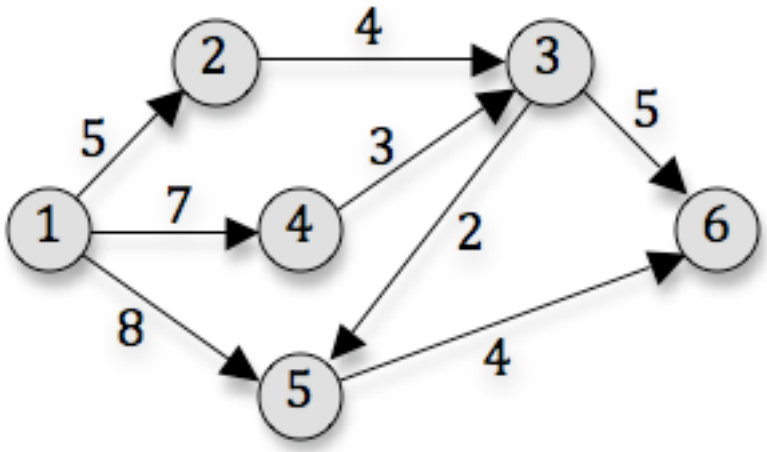
$cap(C) = 4 + 7 + 8 = 19$

- Evidentemente, o fluxo que escoa da fonte **f** para o destino **d** de uma rede R não pode ser maior do que a capacidade de qualquer corte de R.
- **Teorema (Ford e Fulkerson)**. Seja R uma rede com fonte **f** e destino **d**. Então o fluxo máximo de **f** para **d** em R é igual à capacidade do corte de menor capacidade em R.

Fluxo máximo em redes

- Portanto, dada uma rede, o corte de menor capacidade determina o fluxo máximo entre a fonte e o destino desta rede. No entanto, determinar o corte de capacidade mínima não é fácil, pois podem existir muitos cortes em uma rede.
- Seja R uma rede e **a** = (uv) um arco de R. Desconsiderando o sentido da seta, vamos chamar **a** de **aresta** de R.
- Seja R uma rede e **C** = (v_1, v_2, ..., v_n) uma sequência de vértices de R tal que (v_1v_2), (v_2v_3), ..., ($v_{n-1}v_n$) são **arestas** de R. Então, **C** é um **caminho** de v_1 a v_n em R.

Exemplo:



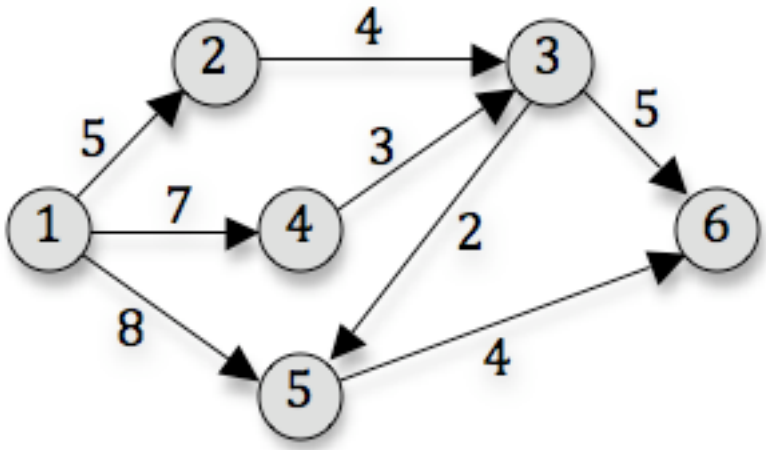
As sequências **(1, 2, 3, 6)** e **(1, 5, 3, 6)** são caminhos de 1 a 6 nesta rede.

Fluxo máximo em redes

- Seja R = (V, A) uma rede e **C** = (v_1, v_2, ..., v_n) um **caminho** de v_1 a v_n em R. Seja **a** = ($v_i v_j$) uma aresta do caminho **C**. A aresta **a** é denominada de **arco direto** de R em relação ao caminho C, se ($v_i v_j$) $\in$ A (**a** é um arco que sai de v_i e incide em v_j). Caso contrário, se ($v_j v_i$) $\in$ A, a aresta **a** é um **arco contrário** de R em relação ao caminho C.
- Seja R uma rede e **a** uma aresta de R. Define-se a **folga** da aresta **a** como: folga(a) = cap(a) - fluxo(a).
- Seja R uma rede e **C** um caminho de **f** (fonte) para **d** (destino) em R. O caminho C é denominado de **caminho de aumento de fluxo** se, para cada aresta **a** nesse caminho, tem-se:
  - folga(**a**) > 0, se **a** for um arco direto em C;
  - fluxo(**a**) > 0, se **a** for um arco contrário em C.

Fluxo máximo em redes

Exemplo:



Sejam os caminhos:

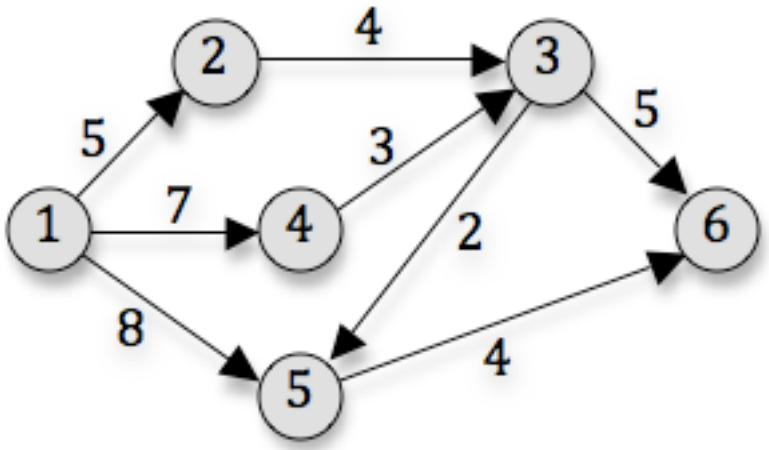
C_1 = (1, 2, 3, 6)

C_2 = (1, 5, 3, 6)

- No caminho C_1, seja fluxo = 3. Neste caso, como todas as arestas são diretas, temos:
  - folga(12) = 5 - 3 = 2 > 0
  - folga(23) = 4 - 3 = 1 > 0
  - folga(36) = 5 - 3 = 2 > 0
- Portanto, C_1 é um caminho de aumento de fluxo.

Intuitivamente: é possível aumentar o fluxo neste caminho, pois a **capacidade** das arestas **não está** sendo **totalmente utilizada**. Observe também que se o fluxo aumentar para 4, este caminho deixa de ser um caminho de aumento de fluxo, pois o fluxo na aresta (23) terá atingido sua capacidade máxima (**folga = 0**).

Fluxo máximo em redes



Caminho C_2 = (1, 5, 3, 6)

- No caminho C_2, seja fluxo = 1. Neste caso, temos:
  - folga(15) = 8 - 1 = 7 > 0
  - fluxo(53) = 1 > 0 (notar que (53) é arco contrário em C_2)
  - folga(36) = 5 - 1 = 4 > 0
- Portanto, C_2 também é um caminho de aumento de fluxo.
- O **algoritmo de Ford-Fulkerson** utiliza os caminhos de aumento de fluxo para resolver o problema de fluxo máximo em uma rede.

Algoritmo de Ford-Fulkerson

- O algoritmo atribui, inicialmente, o valor zero para o fluxo em cada aresta da rede e atribui a cada vértice **v** da rede um rótulo da forma: (predecessor(v), qf(v)), em que **qf(v)** é a quantidade de fluxo que pode ser transferida de **f** para **v**.
- A ideia do algoritmo é ir aumentando o fluxo que escoa pelas arestas até que isso não seja possível.
- Na rotulação dos vértices, os arcos diretos são tratados diferentemente dos arcos contrários.
- Se o vértice **u** é acessado do vértice **v** por meio de um **arco direto**, temos:

  rótulo(u) = (v+, min(qf(v), folga(vu))
- Se o vértice **u** é acessado do vértice **v** por meio de um **arco contrário**, temos:

  rótulo(u) = (v-, min(qf(v), fluxo(uv))

Algoritmo de Ford-Fulkerson

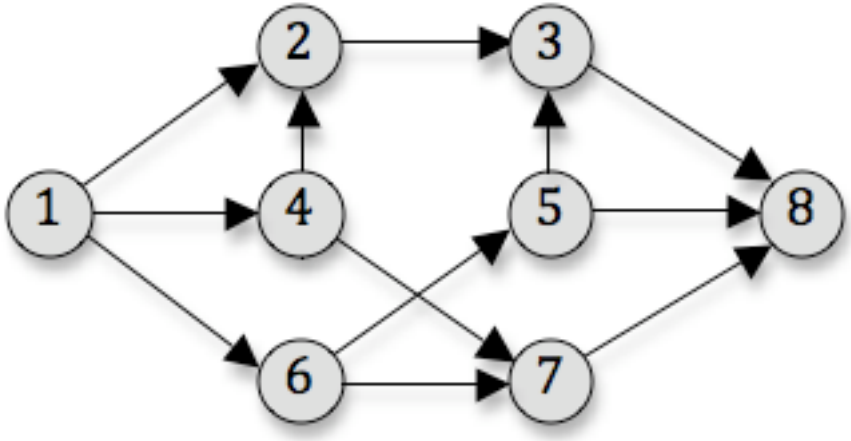
```
AlgoritmoFordFulkerson(rede r, vertice f, vertice d)
{
  ajustar o fluxo de todas as arestas e vértices para 0;
  rotulo(f) = (vazio,INFINITO);
  rotulados = {f};
  while (rotulados != ∅)
  {
    retire um vértice v de rotulados;
    for (todos vértices u não rotulados adjacentes a v)
    {
      if (direto(vu) && folga(vu) > 0)
        rotulo(u) = (v+, min(qf(v),folga(vu)));
      else
      if (contrario(vu) && fluxo(uv) > 0)
        rotulo(u) = (v-, min(qf(v),fluxo(uv)));

      if (u recebeu um rótulo)
      {
        if (u == d)
        {
          if (qf(d) > 0)
            AumentarFluxo(caminho de f a d);
          rotulados = {f};
        }
        else
          rotulados = rotulados ∪ {u};
      }
    }
  }
}
```

Observe que algoritmo não estabelece a forma com que a rede deve ser **percorrida**, ou seja, em que ordem os vértices são incluídos e retirados do conjunto **rotulados**.

```
AumentarFluxo(caminho C)
{
  for (cada aresta a ∈ C)
  {
    if (direto(a))
      f(a) = f(a) + qf(d);
    else
      f(a) = f(a) - qf(d);
  }
}
```

Algoritmo de Ford-Fulkerson

Exemplo:



Vamos considerar que a rede será percorrida em **profundidade**, ou seja, que o conjunto **rotulados** é organizado como uma **pilha**.

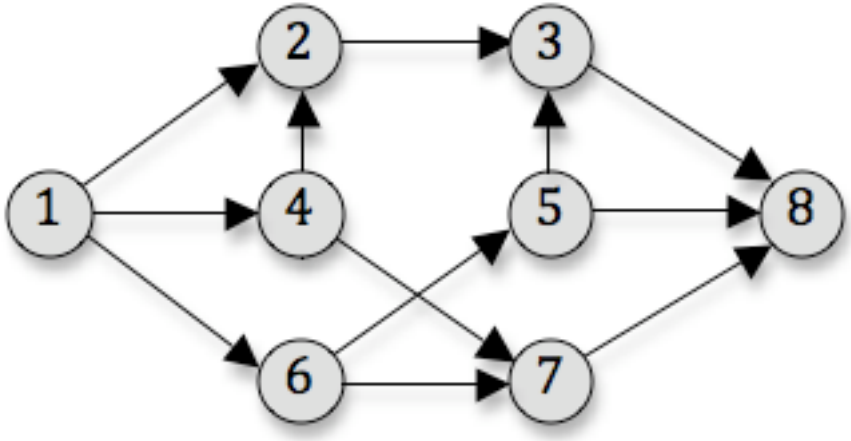
iteração		0	1	2	3	4	5	6	7	8	...
arestas	(12)	2,0									
	(14)	4,0								4,1	
	(16)	1,0			1,1						
	(23)	5,0									
	(38)	3,0									
	(42)	2,0									
	(47)	3,0								3,1	
	(53)	5,0									
	(58)	2,0								2,1	
	(65)	2,0								2,1	
	(67)	3,0			3,1					3,0	
	(78)	1,0			1,1						
vértices	1	-,∞									
	2		1,2			1,2					
	3									5,1	
	4		1,4			1,4					
	5			6,1					7,1		
	6		1,1					7,1			
	7			6,1			4,3				
	8				7,1					5,1	
rotulados		**1**	2,4,**6**	2,4,5,**7**	**1**	2,**4**	2,**7**	2,**6**	2,**5**	**1**	

A tabela mostra as **capacidades** dos arcos, assim como o **fluxo** em cada arco e as **rotulações dos vértices** ao longo da execução do algoritmo.

aresta: (cap, fluxo)

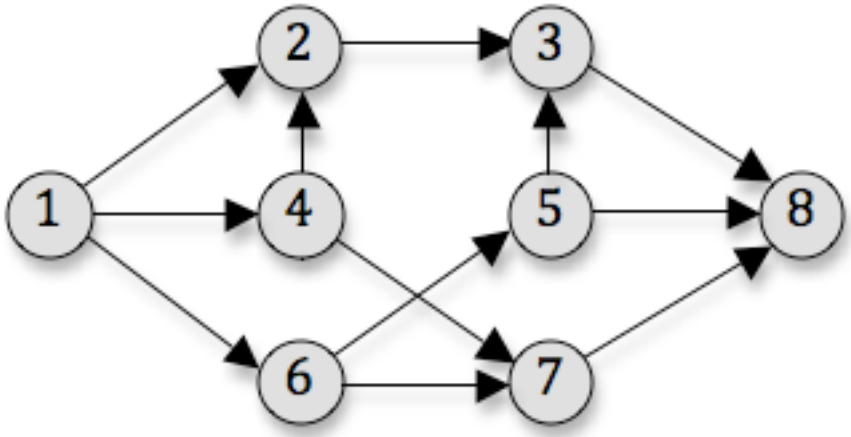
vértice: (predecessor, qf)

Algoritmo de Ford-Fulkerson



- **Iteração 0**: fluxo = 0 em todas as arestas, o vértice 1 (fonte) recebe o rótulo (-, ∞) e o conjunto rotulados = {1}.
- **Iteração 1**: O vértice 1 é retirado de rotulados e os vértices adjacentes a 1 (2, 4, 6) são examinados:
  - nó 2: qf(1) = ∞, folga(12) = 2, rótulo(2) = (1,2), rotulados = {2}
  - nó 4: qf(1) = ∞, folga(14) = 4, rótulo(4) = (1,4), rotulados = {2,4}
  - nó 6: qf(1) = ∞, folga(16) = 1, rótulo(6) = (1,1), rotulados = {2,4,6}

Algoritmo de Ford-Fulkerson



- **Iteração 2**: O vértice 6 é retirado e seus nós adjacentes ainda não rotulados (5, 7) são examinados:
  - nó 5: qf(6) = 1, folga(65) = 2, rótulo(5) = (6,1), rotulados = {2,4,5}
  - nó 7: qf(6) = 1, folga(67) = 3, rótulo(7) = (6,1), rotulados = {2,4,5,7}
- **Iteração 3**: O vértice 7 é retirado e seu único nó adjacente ainda não rotulado (8) é examinado (notar que 4 e 6 também são adjacentes a 7, mas já foram rotulados):
  - nó 8: qf(7) = 1, folga(78) = 1, rótulo(8) = (7,1), rotulados = {1}

Como 8 é o **destino** e qf(8) = 1, o caminho que leva de 1 a 8, ou seja, o caminho {1, 6, 7, 8} é um **caminho de aumento de fluxo**.

Algoritmo de Ford-Fulkerson

- A rotina **AumentarFluxo** é chamada para este caminho. Neste caso, o fluxo nas arestas (16), (67) e (78) é aumentado em 1. Em seguida, o processo é reiniciado, procurando por outro caminho de aumento de fluxo.
- **Iteração 4**: Os vértices adjacentes ao vértice 1 são examinados novamente:
  - nó 2: qf(1) = ∞, folga(12) = 2, rótulo(2) = (1,2), rotulados = {2}
  - nó 4: qf(1) = ∞, folga(14) = 4, rótulo(4) = (1,4), rotulados = {2,4}
  - nó 6: qf(1) = ∞, folga(16) = 0.
- **Iteração 5**: O vértice 4 é selecionado. O único vértice adjacente a 4 que ainda não está rotulado é o vértice 7 (os outros vértices adjacentes, 1 e 2, já estão rotulados). O nó 7 é então examinado:
  - nó 7: qf(4) = 4, folga(47) = 3, rótulo(7) = (4,3), rotulados = {2,7}

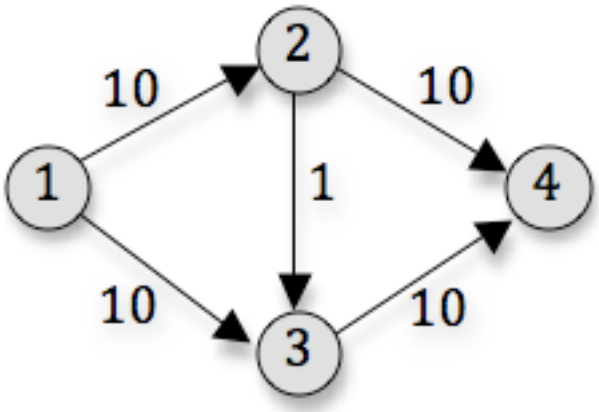
Algoritmo de Ford-Fulkerson

- **Iteração 6**: O vértice 7 é selecionado. Os vértices adjacentes a 7 ainda não rotulados são 6 e 8:
  - nó 6: qf(7) = 3, fluxo(76) = 1, rótulo(6) = (7,1), rotulados = {2,6}
  - nó 8: qf(7) = 3, folga(78) = 0.
- **Iteração 7**: O vértice 6 é selecionado. Seu único vizinho ainda não rotulado é o vértice 5:
  - nó 5: qf(6) = 1, folga(65) = 2, rótulo(5) = (7,1), rotulados = {2,5}
- **Iteração 8**: O vértice 5 é selecionado. Seus vizinhos são: 3, 6 (já rotulado) e 8. Portanto:
  - nó 3: qf(5) = 1, folga(53) = 5, rótulo(3) = (5,1), rotulados = {2,3}
  - nó 8: qf(5) = 1, folga(58) = 2, rótulo(8) = (5,1), rotulados = {1}

Neste caso, o caminho de aumento de fluxo que leva de 1 a 8 é {1, 4, 7, 6, 5, 8} (basta verificar os vértices selecionados em cada iteração). Neste caminho, o arco (76) é contrário. Assim, a rotina **AumentarFluxo** aumenta de 1, o fluxo nas arestas (14), (47), (65) e (58), e diminui de 1, o fluxo na aresta (67).

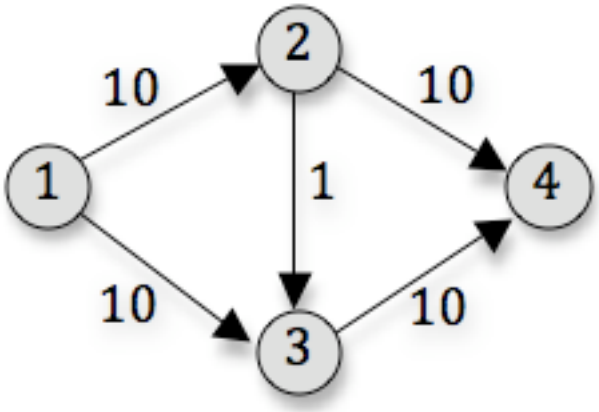
Algoritmo de Ford-Fulkerson

- **Exercício**. Completar a tabela de execução do algoritmo de Ford-Fulkerson. Qual é o fluxo máximo no vértice 8?
- **Exercício**. Ao final da execução do algoritmo de Ford-Fulkerson será encontrado o fluxo máximo nas arestas da rede e, consequentemente, um corte de capacidade mínima. Que corte é este?
- A complexidade do algoritmo de Ford-Fulkerson não é necessariamente uma função do número de vértices ou de arestas da rede. Considere, por exemplo:



Com a busca em profundidade, será encontrado o **caminho de aumento de fluxo**: {1, 2, 3, 4}, com o fluxo nas arestas (12), (23) e (34) aumentado para 1. O próximo caminho será: {1, 3, 2, 4}, com o fluxo nas arestas (13) e (24) aumentado para 1 e na aresta contrária (32) diminuído para 0.

Algoritmo de Ford-Fulkerson



O próximo caminho será igual ao primeiro e o seguinte, igual ao segundo e assim sucessivamente. Ao final teremos encontrado 20 caminhos de aumento de fluxo apesar da rede ter apenas 4 vértices. Isto ocorre porque cada caminho encontrado aumenta (ou diminui) o fluxo nas arestas de apenas 1 unidade.

- Com a busca em profundidade tenta-se atingir o destino o mais rapidamente possível, para aplicar a rotina de aumento de fluxo nos arcos.
- Edmond e Karp mostraram que o caminho de aumento de fluxo mais curto leva a melhores resultados. Para isto, deve-se usar a busca em largura.
- O **algoritmo de Edmond-Karp** é o de Ford-Fulkerson, mas usando busca em largura (**rotulados** é uma **fila**). O algoritmo de Edmond-Karp tem complexidade $O(|V||A|^2)$.

Algoritmo de Ford-Fulkerson

- No algoritmo de Edmond-Karp evitam-se os pequenos aumentos de fluxo nos arcos da rede, como ocorrem no algoritmo de Ford-Fulkerson.
- No entanto, um grande número de vértices precisa ser rotulado para encontrar o menor caminho de aumento de fluxo e todos esses rótulos serão descartados quando o processo é reiniciado para a busca de outro caminho.
- Uma abordagem que combina as duas buscas foi proposta por Dinic. A busca em largura evita a ocorrência de pequenos incrementos de fluxo e a busca em profundidade é feita tomando o caminho mais curto.
- O **algoritmo de Dinic** tem complexidade $O(|V|^2|A|)$, o que, em geral, é melhor do que o algoritmo de Edmond-Karp (pois, em geral, $|V| < |A|$).

FIM

Muito obrigado.