### **CTC-12**



# Projeto e Análise de Algoritmos

**Carlos Alberto Alonso Sanches** 

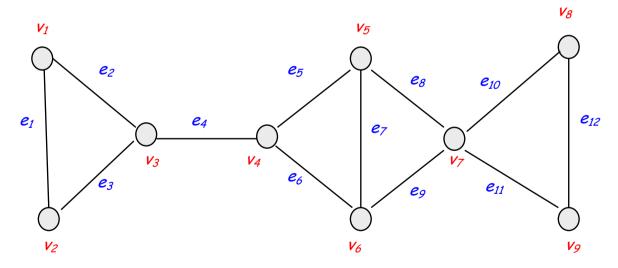
### **CTC-12**

# 7) Algoritmos em grafos

Explorações sistemáticas, Dijkstra, fluxo máximo

### Definição

- Um grafo G=(V,E) é formado pelos vértices  $V=\{v_1, v_2, ..., v_n\}$  e pelas arestas  $E=\{e_1, e_2, ..., e_m\}$ .
- Consideraremos sempre que |V| = n e |E| = m.
- Um exemplo:



$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$$

$$n = 9$$

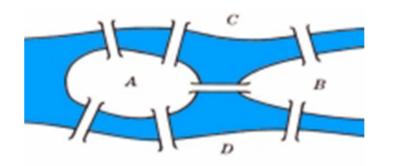
$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$$
 m = 12

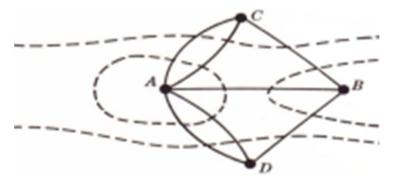
#### Arestas e vértices

- Uma aresta  $e \in E$  é um par não-ordenado (u,v), onde  $u,v \in V$ .
- Neste caso, dizemos que os vértices u e v são adjacentes entre si, e que a aresta e é incidente em u e em v.
- Uma aresta e = (u, v) é chamada de <u>laço</u> quando u = v.
- d(u) é o grau do vértice u, isto é, o número de incidências em u.
- É fácil observar que  $\Sigma_{u \in V} d(u) = 2m$

## Origem histórica

Na cidade de Königsberg (atual Kaliningrado), havia sete pontes sobre o rio Pregel:





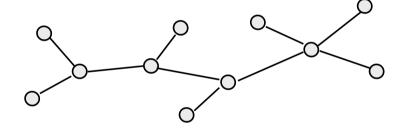
- Será possível fazer um passeio pela cidade, começando e terminando no mesmo local, e passando uma única vez em cada ponte?
- Euler (1736) afirmou que um grafo conexo tem esse passeio se e somente se cada um dos seus vértices tem grau par.
- Todo grafo com essa propriedade é chamado de euleriano.

### Sequências de vértices

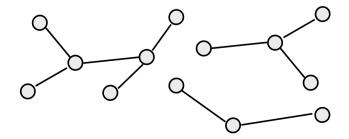
- <u>Caminho</u> é uma sequência alternada de vértices e arestas, onde cada aresta é incidente tanto ao vértice que a antecede como ao que a segue.
- <u>Caminho simples</u> é um caminho no qual cada vértice aparece uma única vez.
- Comprimento de um caminho é o seu número de arestas.
- <u>Ciclo</u> ou <u>circuito</u> é um caminho que começa e termina no mesmo vértice.

### Grafos conexos, árvores e florestas

- Um grafo é <u>conexo</u> se existir um caminho entre qualquer par de vértices, e desconexo em caso contrário.
- Árvore é um grafo conexo sem ciclos.
- Exemplo:



- Floresta é um grafo formado por diversas árvores.
- Exemplo:

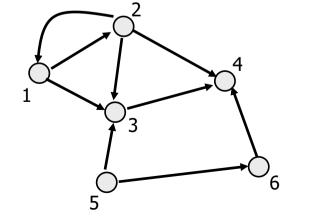


## Digrafos ou grafos orientados

- <u>Digrafos</u> são <u>grafos orientados</u>, isto é, suas arestas possuem direção e são chamadas de arcos.
- Em um digrafo G=(V,E), um arco  $e \in E$  é um par ordenado (u,v), onde  $u,v \in V$ .  $\bigcup_{u} \qquad \bigvee_{v} \qquad \begin{cases} v \text{ \'e sucessor de } u \\ u \text{ \'e predecessor de } v \end{cases}$

■ Cada vértice  $\nu$  tem um grau de saída  $d^{+}(\nu)$  e um grau de entrada  $d^{-}(\nu)$ , que correspondem respectivamente ao total de arcos que saem ou chegam em  $\nu$ .

Exemplo:



$$d^{+}(4) = 0$$

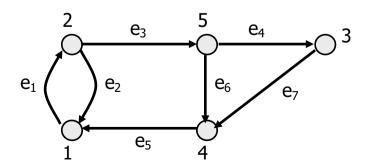
$$d^{-}(4) = 3$$

$$d^+(6) = 1$$

$$d^{-}(1) = 1$$

### Sequências de arcos

- Caminho é uma sequência de arcos  $e_1$ ,  $e_2$ , ...,  $e_q$  tal que a extremidade inicial de  $e_i$  coincide com a final de  $e_{i-1}$ ,  $1 < i \le q$ .
- <u>Ciclo</u> ou <u>circuito</u> é um caminho que começa e termina no mesmo vértice.
- Exemplo:



e<sub>3</sub>, e<sub>4</sub>, e<sub>7</sub>, e<sub>5</sub>: caminho entre os vértices 2 e 1

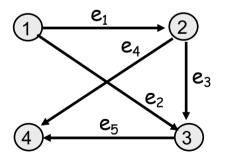
e<sub>3</sub>, e<sub>6</sub>, e<sub>5</sub>, e<sub>1</sub>: ciclo ou circuito

### Representações de grafos

- Há algumas estruturas de dados apropriadas para representar grafos.
- Principais representações:
  - Matriz de incidências
  - Matriz de adjacências
  - Lista de adjacências
    - Com ponteiros
    - Com vetores
  - Lista de arcos

#### Matriz de incidências

- Matriz de incidências é formada por n linhas (uma para cada vértice) e m colunas (uma para cada aresta ou arco).
- lacksquare A posição  $a_{ij}$  dessa matriz indica se a aresta ou o arco  $e_i$ incide sobre o vértice v..
- Exemplo:

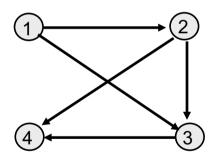


$$A_{\text{n} \times \text{m}} = \begin{bmatrix} +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & +1 & +1 & 0 \\ 0 & -1 & -1 & 0 & +1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$
Grafo não dirigido: haveria somente valores 1 na matriz

Tamanho da estrutura:  $\Theta(n.m)$ 

## Matriz de adjacências

- Matriz de adjacências é formada por n linhas e n colunas.
- A posição  $a_{ij}$  dessa matriz indica se o vértice  $v_j$  é sucessor ou não do vértice  $v_i$ .
- Exemplo:



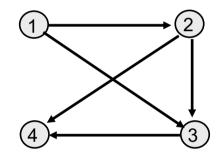
$$A_{n\times n} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Grafo não dirigido: a matriz seria simétrica

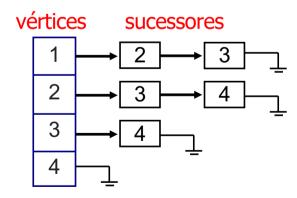
Tamanho da estrutura:  $\Theta(n^2)$  Útil quando grafo é denso:  $m \sim n^2$ 

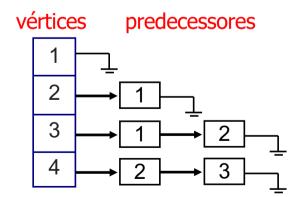
## Lista de adjacências

- <u>Lista de adjacências</u> é formada por um vetor de *n* ponteiros, onde cada vértice aponta para seus sucessores ou predecessores.
- Exemplo:



Grafo não dirigido: haveria o dobro de nós



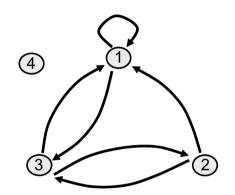


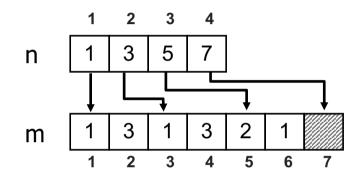
Tamanho da estrutura:  $\Theta(n+m)$ 

Útil quando grafo é esparso: m « n²

### Representações alternativas

Lista de adjacências através de vetores

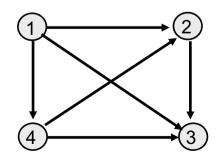




Grafo não dirigido:
o segundo vetor teria
o dobro do tamanho

Tamanho da estrutura: Θ(n+m)

Lista de arcos



Grafo não dirigido: mesma estrutura

Tamanho da estrutura:  $\Theta(m)$ 

### Representação mais adequada

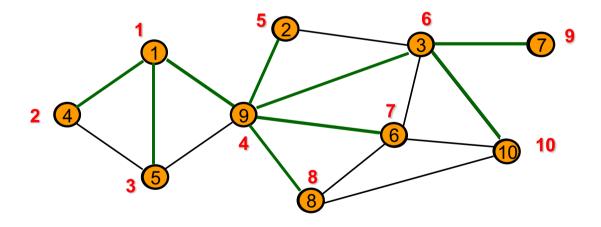
- Alguns critérios para se escolher a melhor representação:
  - Espaço de armazenamento (depende do tamanho do grafo)
  - Teste de pertinência de uma aresta (matriz)
  - Verificação do grau de um vértice (lista)
  - Inserção ou remoção de uma aresta (matriz)
  - Percurso no grafo (lista)
- Geralmente, a lista de adjacências costuma ser mais vantajosa.

## Exploração sistemática de um grafo

- Explorar um grafo é percorrê-lo completamente, visitando todos os vértices e as arestas.
- A ordem dessas visitas depende:
  - do vértice onde a exploração começa;
  - da ordem de armazenamento dos vértices e das arestas na estrutura de dados;
  - do tipo de exploração: em largura ou em profundidade.

### Em largura (breadth-first search)

- Tática: enquanto for possível, examinar todos os vértices à mesma distância do vértice corrente; quando não for mais possível, aprofundar.
- Exemplo (supomos armazenamento em ordem crescente):



 Uma aplicação: a exploração em largura permite encontrar, por exemplo, as distâncias e os menores caminhos entre os vértices.

## Exploração em largura

```
BFS(s) {
   desmarcar todos os vértices;
   int cont = 0;
   queue q;
   marcar s;
   expl[s] = ++cont;
   enqueue(q,s);
   while (!isEmpty(q)) {
         curr = dequeue(q);
         // explorando curr
         for \langle curr, v \rangle \in E  {
             if v está desmarcado {
                marcar v;
                 expl[v] = ++cont;
                 enqueue (q,v);
```

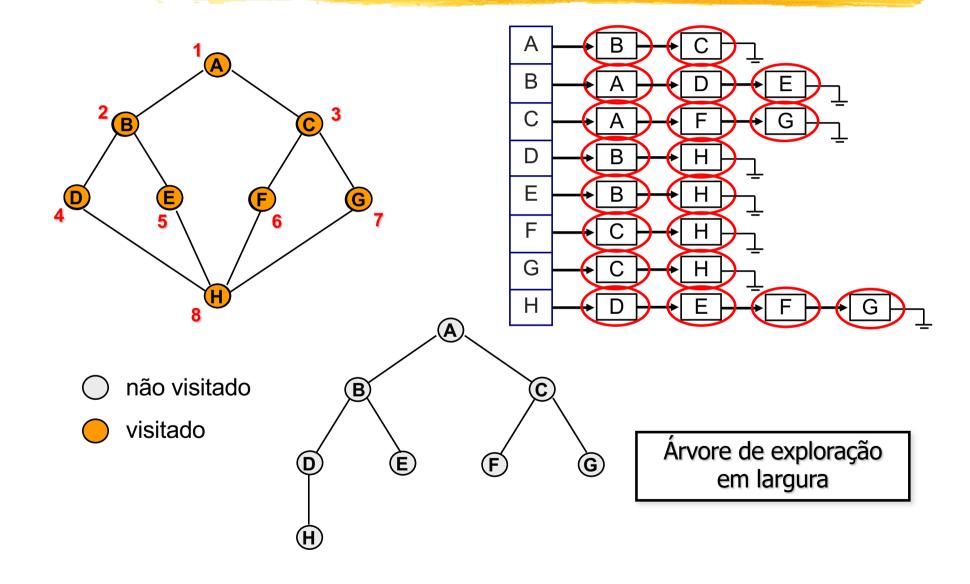
Código aplicável a grafos não orientados e conexos

Cada vértice recebe um número de exploração (equivalente a marcá-lo)

### Exploração em largura

- Durante a execução deste algoritmo, cada vértice pode ficar em três estados:
  - desmarcado (portanto, fora da fila): ainda não foi atingido;
  - marcado e na fila: atingido, mas não completamente explorado;
  - marcado e fora da fila: já explorado.
- Cada vértice entra na fila uma única vez, e cada aresta é visitada duas vezes. Portanto, sua complexidade de tempo é Θ(n+m).

## Exemplo

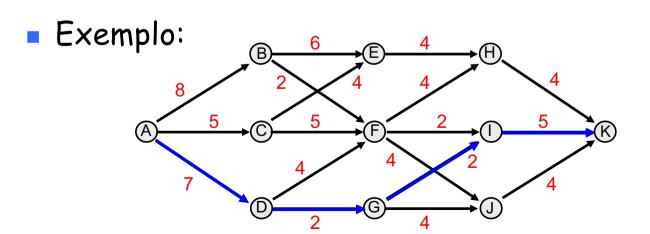


# Exploração em largura (digrafos)

```
int cont;
                                            BFS(v) {
                             Código
queue q;
                                               marcar v;
                            adicional
                                               expl[v] = ++cont;
TravessiaBFS(s)
                                               enqueue (q,v);
   desmarcar todos os vértices:
                                               while (!isEmpty(q)) {
   cont = 0;
                                                   curr = dequeue(q);
   BFS(s);
                                                   // explorando curr
   for v \in V {
                                                   for \langle curr, u \rangle \in E  {
      if v está desmarcado
                                                     if u está desmarcado {
          BFS(v);
                                                        marcar u;
                                                         expl[u] = ++cont;
                                                         enqueue (q,u);
                                                          Tempo: \Theta(n+m)
                                                       Solução análoga para
                                                        grafos desconexos
```

#### Caminhos mais curtos

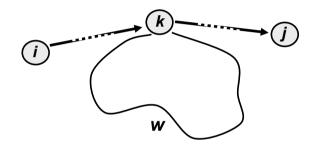
- Um digrafo (ou grafo) G=(V,E), onde  $V=\{v_1, v_2, ..., v_n\}$ , é chamado de ponderado se cada arco (ou aresta)  $(v_i, v_j) \in E$  tem custo  $c_{ij}$ .
- Distância entre dois vértices é a somatória dos custos dos arcos (ou arestas) de um caminho que os une.
- Um problema clássico é encontrar o caminho mais curto ou a distância mínima entre dois vértices.



Distância mínima entre A e K: 7+2+2+5 = 16

### Uma condição de existência

• Considere o caminho abaixo entre  $i \in j$ , e o ciclo w:



- Se o comprimento de w for negativo, qual seria a distância mínima entre i e j?
- Uma condição de existência para o caminho mais curto é que seja elementar, isto é, sem ciclos em seu interior.

### Arcos (ou arestas) de mesmo peso

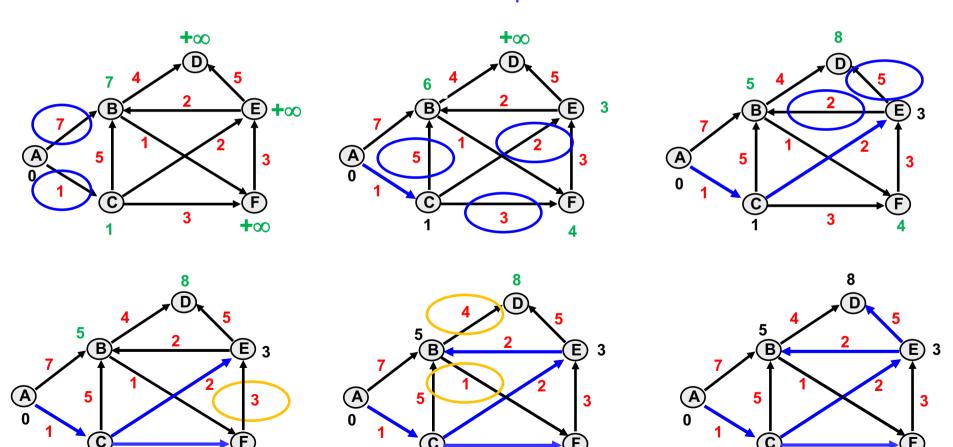
- Quando todas os arcos (ou arestas) têm pesos iguais, basta uma simples alteração na exploração em largura.
- Afinal, os vértices vão sendo enfileirados seguindo a ordem de proximidade: portanto, basta incrementar a distância do vértice antecessor.
- O algoritmo de Dijkstra generaliza essa ideia.

```
BFSMinCam(r) {
   queue q;
   int ce = 0;
   for v \in V - \{r\}
         d[v] = +\infty;
         expl[v] = 0;
   expl[r] = ++ce;
   d[r] = 0;
   enqueue(q,r);
   while (!isEmpty(q)) {
         u = dequeue(q);
         for \langle u, v \rangle \in E  {
              if (expl[v] == 0) {
                 expl[v] = ++ce;
                 d[v] = d[u] + 1;
                 enqueue (q, v);
```

## Exemplo do algoritmo de Dijkstra

Pesos dos arcos, distâncias provisórias, distâncias definitivas

Os arcos indicados são os últimos que atualizaram a distância



## Algoritmo de Dijkstra (1959)

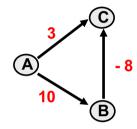
Assemelha-se a uma exploração em largura Encontra todos os caminhos mínimos a partir do vértice u

### Complexidade de tempo

- Grosso modo, o algoritmo de Dijkstra gasta tempo de pior caso  $\Theta(n^2+m) = \Theta(n^2)$ .
- No entanto é possível melhorar essa complexidade de tempo se o conjunto S for implementado com um heap de mínimo.
- Nesse caso, passaria a ser  $\Theta((n+m).log n)$ :
  - O heap possuirá inicialmente n elementos.
  - No total, são realizadas n extrações de mínimo e até m modificações de valor (será preciso manter um vetor auxiliar que armazena a posição corrente que cada vértice ocupa no heap).

## Arcos (ou arestas) com custos negativos

No digrafo abaixo, qual a distância mínima entre os vértices A e C?



- O algoritmo de Dijkstra daria como resposta 3, mas o valor correto é 2... Por que isso acontece?
- No processo de construção do caminho mínimo, o algoritmo de Dijkstra supõe que o custo acumulado sempre cresce. Isso não ocorre se admitirmos arcos (ou arestas) com custos negativos...
- Em um algoritmo mais geral, cada vez que se altera a distância até um vértice, também deve ser recalculada a distância até todos os seus adjacentes.

### Algoritmo de Dijkstra modificado

```
Dijkstra2(u) {
      for v \in V - \{u\}
           d[v] = +\infty;
     d[u] = 0;
      S \leftarrow \{u\};
     while S \neq \emptyset {
           selecionar j \in S;
           S \leftarrow S - \{j\};
           for \langle j, w \rangle \in E
                 if (d[w] > d[j] + c_{iw}) {
                        d[w] = d[j] + c_{jw};
                        pred[w] = j;
                        S \leftarrow S \cup \{w\}; // w \text{ volta para } S
```

 Com uma estrutura adequada para S, a complexidade de tempo desse algoritmo pode ser o(n.m).

### Fluxo máximo

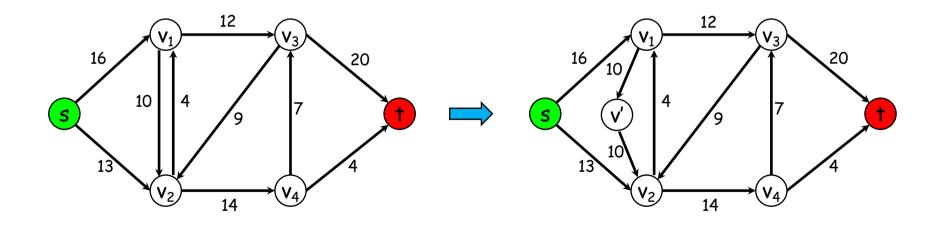
- Podemos interpretar um digrafo como uma rede de fluxo de algum material, com uma fonte, um sorvedouro e vértices intermediários. Cada arco corresponde a um duto com uma capacidade máxima de transmissão.
- Exemplos de aplicação: fabricação e entrega de algum produto, escoamento de líquidos, peças em linhas de montagem, redes de intercomunicação, etc.
- Considerando conservação de fluxo (ou seja, as taxas de entrada e de saída em cada vértice devem ser iguais), deseja-se encontrar a maior taxa de transmissão desde a fonte até o sorvedouro.

### Digrafo da rede de fluxo

- Uma rede de fluxo é um digrafo G(V,E) onde:
  - 1) Cada arco  $(u,v) \in E$  tem capacidade  $c(u,v) \ge 0$
  - 2)  $(u,v) \notin E \Rightarrow c(u,v) = 0$
  - 3) Não há laços: (v,v) ∉ E
  - 4) Não há arcos antiparalelos: (u,v) ∈ E ⇒ (v,u) ∉ E
  - 5) Há somente um vértice fonte s e um vértice sorvedouro t
  - 6) Todo vértice encontra-se em algum caminho entre s e t
  - 7) Com exceção de s, todo vértice possui pelo menos um arco de entrada (portanto, m ≥ n-1)
- Caso seja necessário, as restrições (4) e (5) podem ser contornadas através de redes equivalentes, como veremos a seguir.

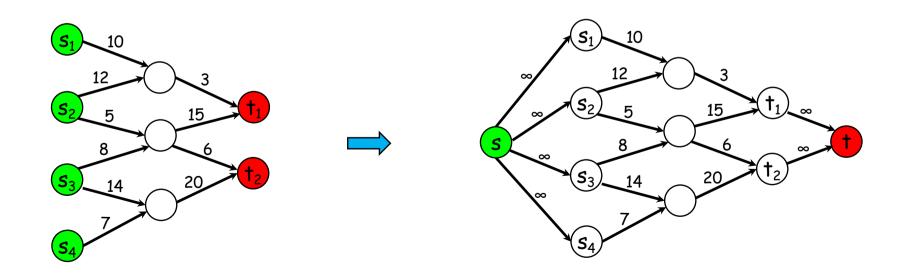
### Redes com arcos antiparalelos

- Uma rede com arcos antiparalelos pode ser transformada em uma rede equivalente sem esses arcos.
- Exemplo:



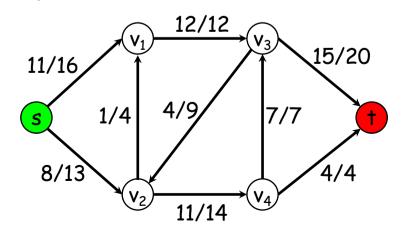
#### Redes com várias fontes ou sorvedouros

- Uma rede com várias fontes ou sorvedouros também pode ser transformada em uma rede equivalente com uma única superfonte e um único supersorvedouro.
- Exemplo:



### Fluxos no digrafo

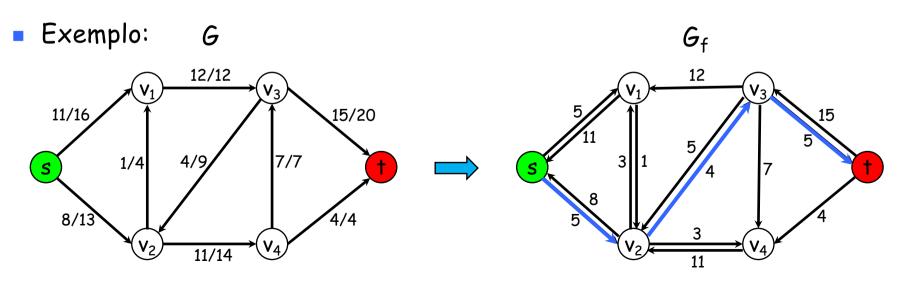
- Um fluxo em G(V,E) é uma função  $f: V \times V \rightarrow R$  tal que:
  - $0 \le f(u,v) \le c(u,v)$  para  $u,v \in V$ : fluxos não ultrapassam as capacidades
  - $(u,v) \notin E \Rightarrow f(u,v) = 0$ : se não houver arco, fluxo será nulo
  - $u \in V \{s,t\} \Rightarrow \Sigma_{v \in V} f(u,v) = \Sigma_{v \in V} f(v,u)$ : conservação de fluxo
  - $\Sigma_{v \in V} f(v,s) = 0$ : não há fluxo de entrada
  - $|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t)$ : fluxo da rede
- Exemplo de rede com fluxo |f| = 19:



Notação no arco (u,v): f(u,v)/c(u,v)

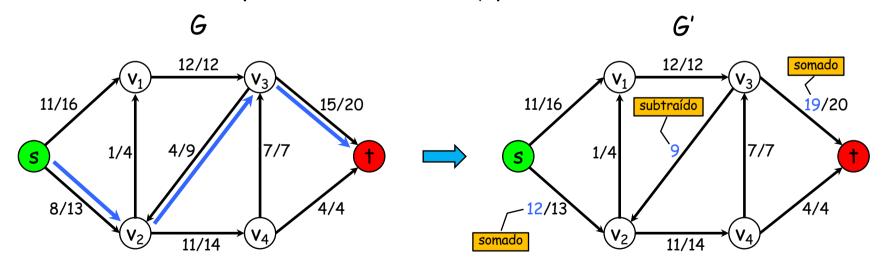
#### Redes residuais

- Dada uma rede G(V,E) com um fluxo f, sua rede residual é composta de arcos com capacidades que indicam como esse fluxo pode ser alterado:
  - Se  $(u,v) \in E$ ,  $c_f(u,v) = c(u,v) f(u,v)$ : é a capacidade residual
  - Se (u,v) ∉ E, c<sub>f</sub>(u,v) = f(v,u): fluxo pode mudar de direção
  - Nos demais casos,  $c_f(u,v) = 0$
- $G_f(V,E_f)$  é a rede residual de G, onde  $E_f = \{(u,v) \in VxV \mid c_f(u,v) > 0\}$ . É uma rede de fluxo com eventuais arcos antiparalelos, ou seja,  $|E_f| \le 2m$ .



#### Caminhos aumentadores

- Dada uma rede G(V,E) com um fluxo f, um caminho aumentador p é um caminho simples de s a t em  $G_f$ .
- Dado um caminho aumentador p em  $G_f$  e a sua capacidade residual  $c_f(p) = \min \{c_f(u,v) \mid (u,v) \in p\}$ , é possível obter um aumento de  $c_f(p)$  no fluxo f.
- Considere o exemplo anterior, onde  $c_f(p) = 4$ :

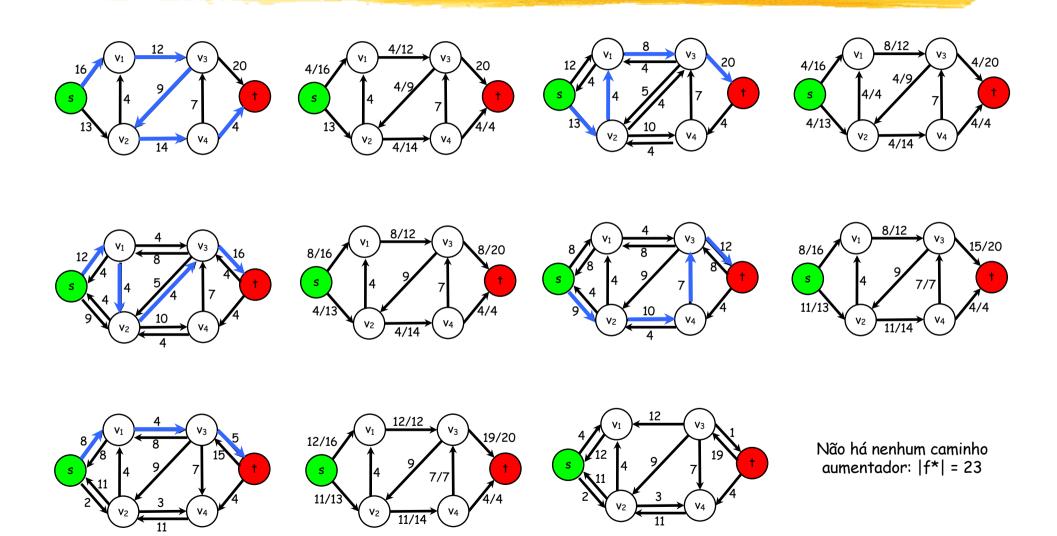


• O fluxo de G' tem valor  $|f| + c_f(p) = 19 + 4 = 23$ .

## Algoritmo de Ford-Fulkerson (1962)

- Se as capacidades dos arcos tiverem valores irracionais, esse algoritmo pode não terminar...
- Supondo capacidades racionais, uma mudança de escala pode transformá-las em valores inteiros. Se f\* for um fluxo máximo nessa rede transformada, então o laço while será executado no máximo |f\*| vezes.
- Como o número de arcos de G<sub>f</sub> é no máximo 2m, um caminho aumentador pode ser encontrado em tempo O(n+m)=O(m) através de uma exploração em largura. Este é também o tempo da atualização posterior da rede de fluxo. Portanto, o tempo total é O(m.|f\*|).

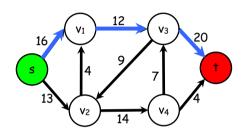
## Exemplo

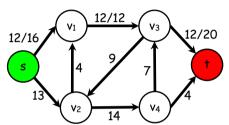


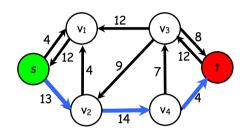
## Algoritmo de Edmonds-Karp (1972)

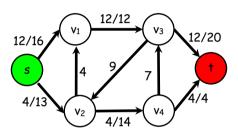
- É igual ao algoritmo de Ford-Fulkerson, mas sempre escolhe o <u>caminho</u> <u>aumentador com um número mínimo de arcos</u>.
- Análise da complexidade de tempo:
  - Uma exploração em largura define as distâncias de s a cada vértice e encontra o menor caminho até t em tempo O(m). No digrafo residual, arco "para frente" é aquele que sai de um vértice com distância i e chega a outro vértice com distância i+1; arco "para trás" é o contrário; e arco "para o lado" une vértices de mesma distância.
  - Em cada iteração do while, pelo menos um arco "para frente" será removido, e alguns arcos "para trás" ou "para o lado" serão acrescentados. Deste modo, a distância de s a cada vértice nunca diminui.
  - Seja d(t) a distância de s a t. No pior caso, serão removidos m-d(t) arcos "para frente", um por iteração do while, sem alterar o valor d(t). Depois disso, ou t ficará desconectado de s, ou d(t) aumentará. Se t ficar desconectado, o algoritmo termina.
  - Como o valor d(t) pode aumentar no máximo n-1 vezes, o tempo total será O(nm²).
- Há ainda outros algoritmos mais eficientes, que não serão apresentados:
   Dinic, Goldberg-Tarjan, King-Rao-Tarjan, Goldberg-Rao, etc.

# Mesmo exemplo anterior

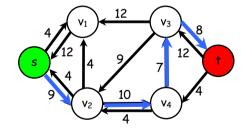


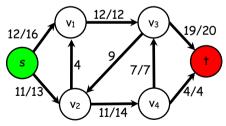


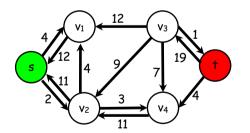




Poderia ter sido escolhido o caminho s-v<sub>2</sub>-v<sub>4</sub>-t







Não há nenhum caminho aumentador: |f\*| = 23