Tópicos Avançados em Algoritmos

Prof. Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

27 de maio de 2019

Fluxo em Redes

Redes de Transportes

- Redes de transportes que são utilizadas para distribuir produtos dos centros de produção para os mercados, são melhor analisadas quando apresentadas na forma de grafos orientados.
- Uma *rede* $N := N(x, y)$ é um grafo orientado D (o grafo subscrito em N) com dois vértices específicos: a *origem* x e o *destino* y.
- O vértice x representa o centro de produção, e o vértice y o mercado.
- Os demais vértices são chamados de vértices *intermediários* representado pelo conjunto I.

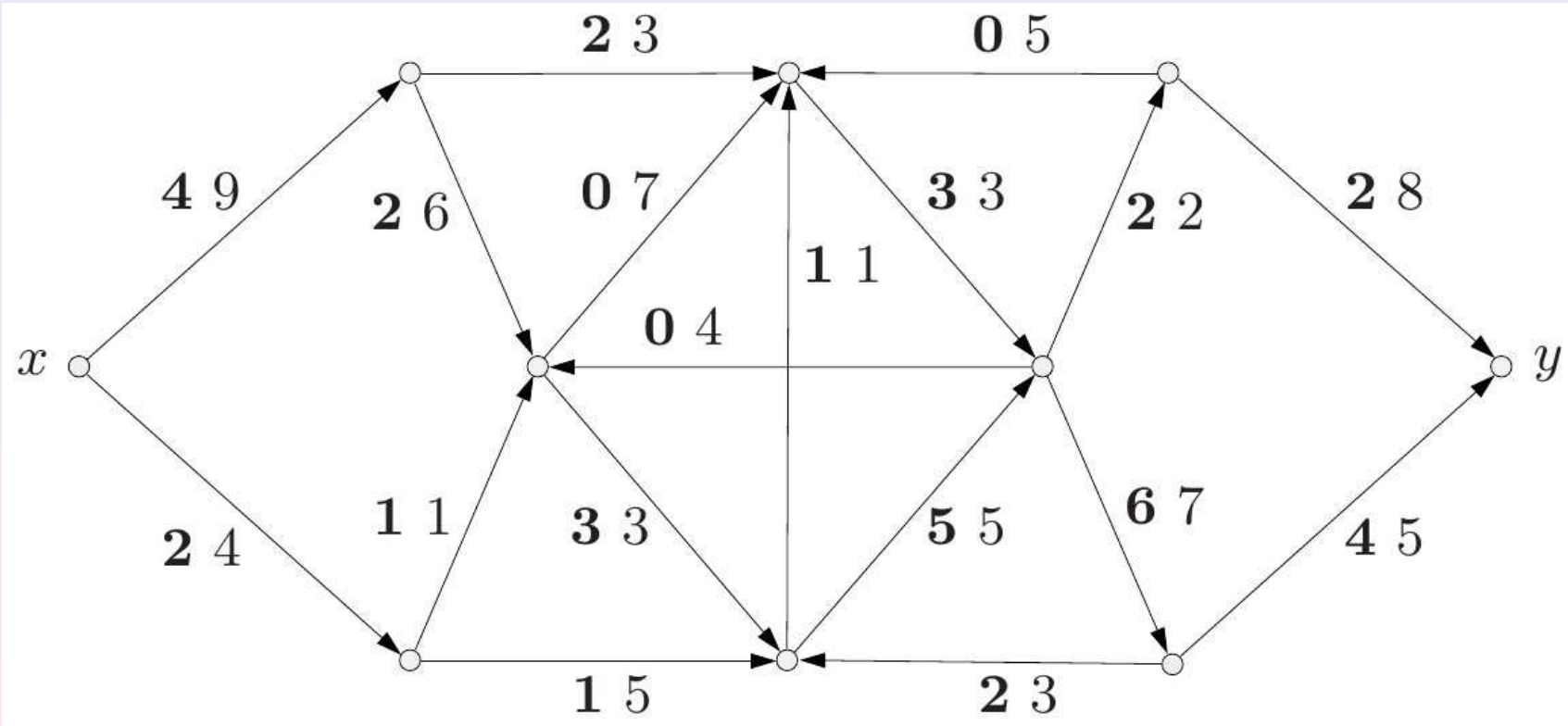
Redes de Transportes

- A função c é chamada de *função de capacidade* de N, e o seu valor em um arco a é a *capacidade* de a.
- A capacidade de um arco é a taxa máxima (ou fluxo máximo) de produtos que aquele arco suporta.
- Vamos definir uma função f no domínio real sobre um conjunto de arcos A. Se $S \subseteq A$, $\sum_{a \in S} f(a) = f(S)$.
- Para $X \subseteq V$:
  $f^+(X) := f(\partial^+(X))$ e $f^-(X) := F(\partial^-(X))$

Fluxos

- Um fluxo de x a y $(x, y)-$fluxo, ou simplemente fluxo em N é uma função f definida em A que satisfaça: $f^+(v) = f^-(v)$, para todo $v \in I$.
- O fluxo representa a taxa com a qual um materia é transportado através de a. Isto requer que em qualquer vértice intermediário, a taxa com que entra material no vértice é a mesma com que sai (não há consumo nem produção no vértice).
- Além disto, um fluxo f é *viável* se estiver restrito à capacidade do arco: $0 \leqslant f(a) \leqslant c(a)$ para todo $a \in A$.
- Para um arco a, se $f(a) = 0$, então ele é $f-$zero, se $0 < f(a) < c(a)$ então ele é $f-$positivo, se $f(a) = c(a)$ então ele é $f-$saturado.

Fluxos em Redes



Fluxo/Capacidade em uma rede: Fluxo em negrito.

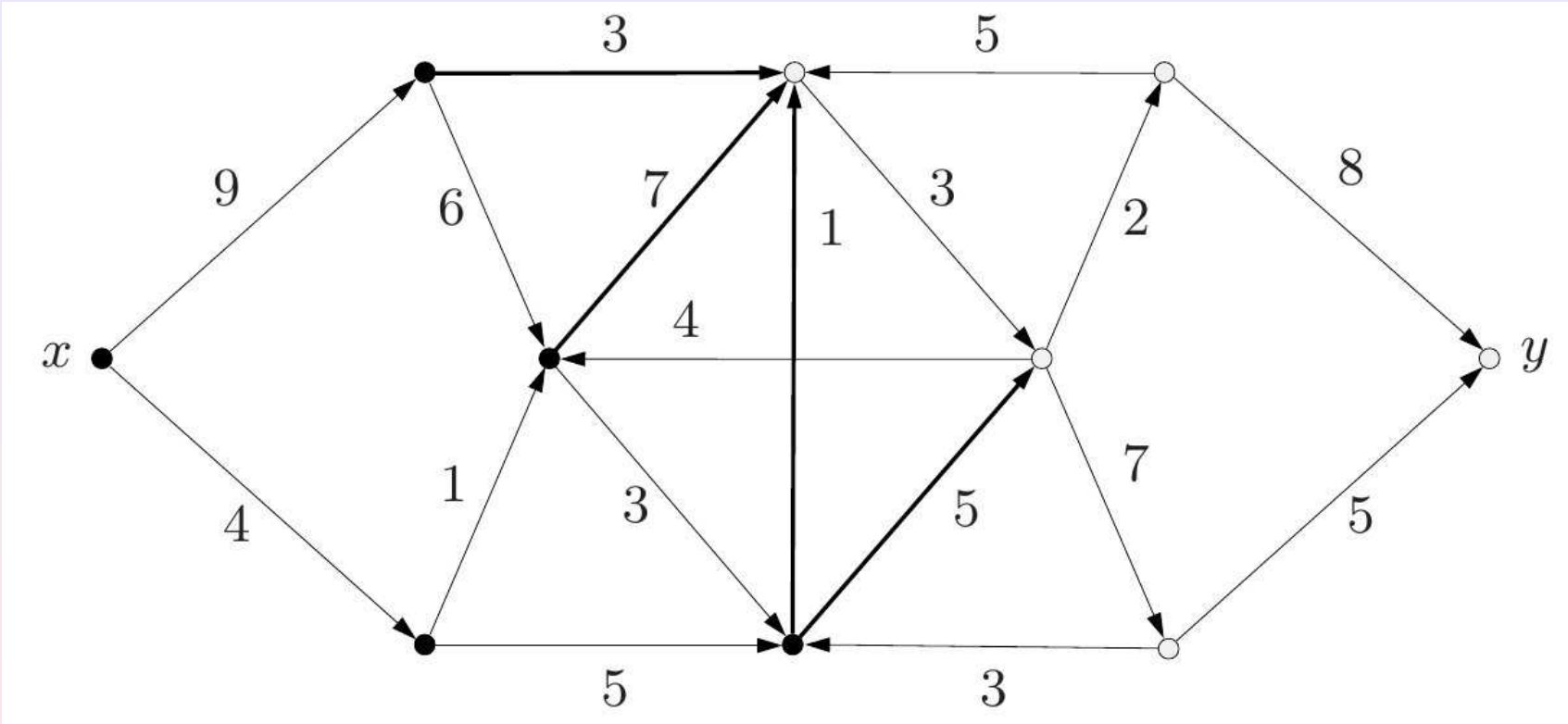
Fluxos em Redes

- Se X é um conjunto de vértices em uma rede N, e f é um fluxo em N, então $f^+(X) - f^-(X)$ é chamado de *fluxo de saída* de X, enquanto que $f^-(X) - f^+(X)$ é chamado de *fluxo de entrada* de X.
- Para todo vértice intermediário $v \in I$, $f^+(v) - f^-(v) = 0$.
- Conclui-se que $f^+(x) - f^-(x) = f^-(y) - f^+(y) = val(f)$ que chamaremos de *valor* de f.
- Para $X \subset V$, tal que $x \in X$ e $y \in V \setminus X$, $val(f) = f^+(X) - f^-(X)$.
- Um fluxo em uma rede N é um *fluxo máximo* se não existir nenhum outro fluxo de valor maior.
- Um problema importante é encontrar o fluxo máximo em uma rede.

Corte

- Para uma grafo orientado D com dois vértices específicos x e y ($D(x, y)$), um $(x, y)-$corte no grafo é um corte de saída $\partial^+(X)$ tal que $x \in X$ e $y \in V \setminus X$.
- Dada a função capacidade c e um corte K, a capacidade de um corte $K := \partial^+(X)$ é a soma das capacidades de seus arcos, $c^+(X)$, que indicaremos como $cap(K)$.
- Fluxos e Cortes estão relacionados da seguinte forma, o valor de qualquer $(x, y)-$fluxo é delimitado superiormente pela capacidade de qualquer corte separando y de x: $val(f) \leqslant cap(K)$
- A igualdade na expressão acima somente vale quando todo arco em $\partial^+(X)$ é $f-$saturado e todo arco em $\partial^-(X)$ é $f-$zero.

Corte



$K(X)$: vértices de X em negrito. $cap(K) = 16$

Fluxo máximo - Corte mínimo

- Dada a desigualdade $val(f) \leqslant cap(K)$, se tivermos f^* e K^* tal que $val(f^*) = cap(K^*)$, então f^* é o fluxo máximo (pois nenhum fluxo pode ser maior que a capacidade K^*, e K^* pois nenhum K pode ser menor que f^*.
- Vamos considerar uma rede $N := N(x, y)$ com fluxo f. Para cada $xy-$caminho P em N (não necessariament orientado), iremos associar um inteiro não negativo $epsilon(P)$ definido como:
  $\epsilon(P) := \min\{\epsilon(a) : a \in A(P)\}$, onde
  $$\epsilon(a) := \begin{cases} c(a) - f(a) & \text{se } a \text{ é um arco direto em } P \\ f(a) & \text{se } a \text{ é um arco reverso em } P \end{cases}$$

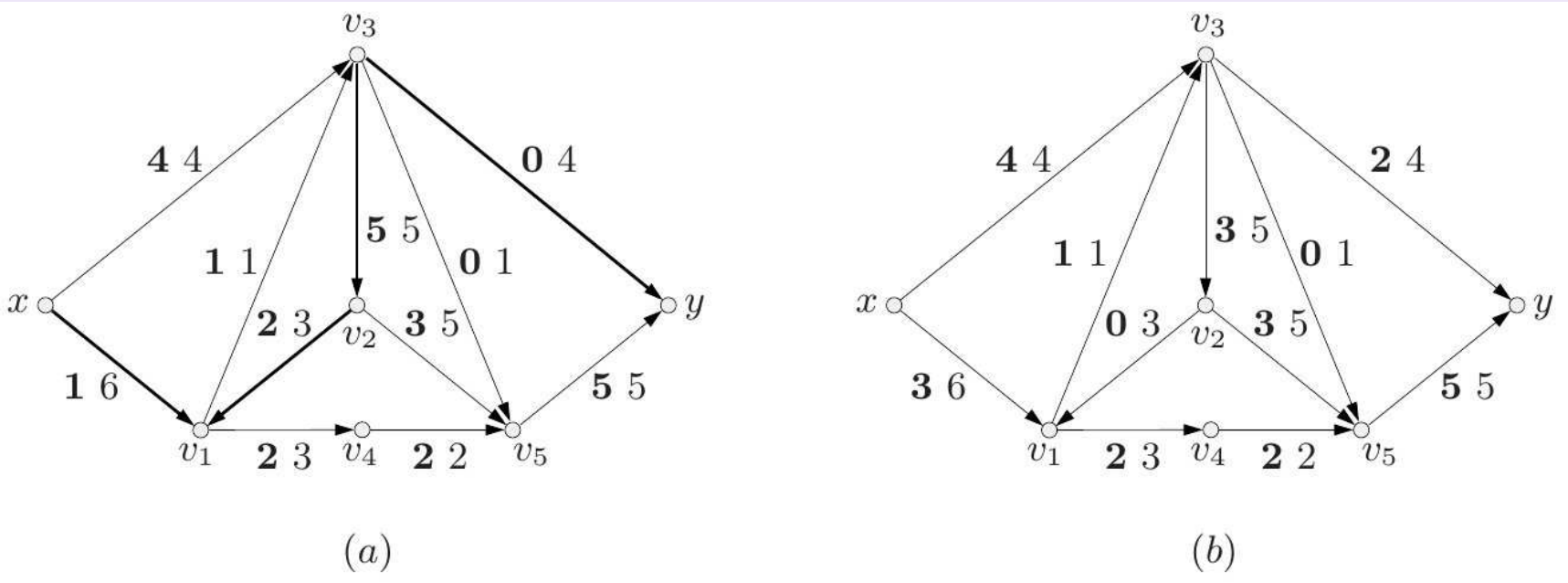
Fluxo máximo - Corte mínimo

- $\epsilon(P)$ é então a quantia máxima de fluxo que pode ser incrementado ao longo do caminho P sem violar as restrições já impostas.
- Um caminho P é dito $f-$saturado, se $\epsilon(P) = 0$.
- Um caminho P é $f-$aumentante se ele não for $f-$saturado. Neste caso, conseguimos um novo fluxo $f'(N)$ onde:

$$f'(a) = \begin{cases} f(a) + \epsilon(P) & \text{se } a \text{ é um arco direto de } P \\ f(a) - \epsilon(P) & \text{se } a \text{ é um arco reverso de } P \\ f(a) & \text{caso contrário} \end{cases}$$

- Um fluxo em uma rede N é máximo, se não houver nenhum caminho $f-$aumentante.

Fluxo máximo - Corte mínimo

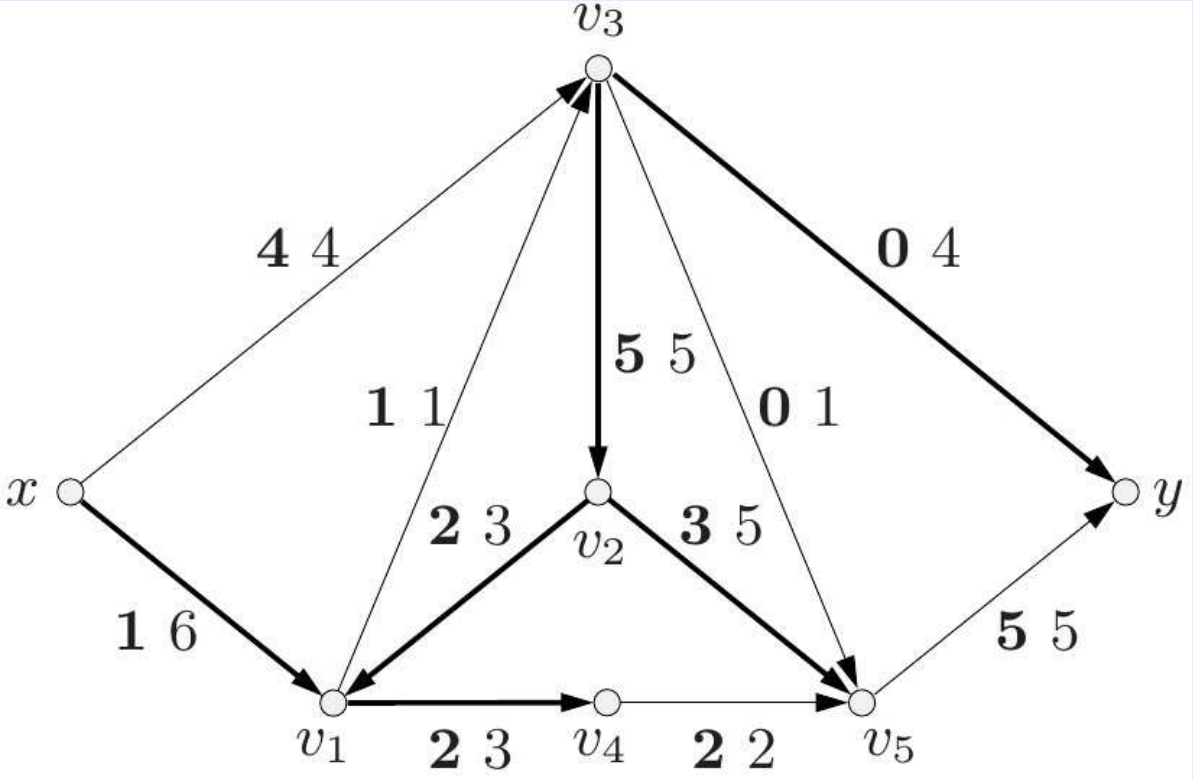


(a) Caminho $f-$aumentante com $\epsilon(P) = 2$; (b) Novo fluxo

Algoritmo Ford-Fulkerson

- O algoritmo é baseado no seguinte fato: *Um fluxo f em uma rede é um fluxo máximo se e somente se não existir nenhum caminho $f-$aumentante*.
- Também baseado no fato de que O fluxo máximo é definido por um corte $\partial^+(X)$ de valor mínimo, um corte onde todos os arcos de saída são $f-$saturados e todos os arcos de entrada são $f-$zero.
- Começamos o algoritmo com um fluxo viável (por exemplo o fluxo 0) e fazemos uma busca em árvore por um caminho $f-$aumentante.
- Uma árvore T com raiz x é dita $f-$aumentante se, para cada vértice v o caminho xTv é $f-$aumentante.

Algoritmo Ford-Fulkerson



Exemplo de uma árvore de busca $f-$aumentante.

Algoritmo Ford-Fulkerson

- Inicialmente a árvore T consiste apenas da origem x.
- Em qualquer etapa, existem duas formas da árvore crescer:
  - Se existir um arco a não $f-$saturado em $\partial^+(X)$, onde $X = V(T)$, ambos a e seu vértice cabeça são incluídos em T.
  - Se existir um arco a $f-$positivo em $\partial^-(X)$, ambos a e seu vértice cauda são incluídos em T.
- Se T atingir o destino y, o caminho xTy é $f-$aumentante, e trocamos f por f'.
- Se falhar, então os arcos de $\partial^+(X)$ são $f-$saturados e de $\partial^-(X)$ são $f-$zero, logo f é um fluxo máximo e $\partial^+(X)$ um corte mínimo.

Ford-Fulkerson

Entrada: uma rede N := N(x,y) e um fluxo viável f em N.
Saida: um fluxo máximo f e um corte mínimo $\partial^+(X)$ em N.

Algoritmo FMCM(N)
enquanto verdade **faça**
$X \leftarrow \{x\}$
para $v \in V$ **faça** $p(v) \leftarrow \emptyset$ estrutura "pai"
enquanto $\exists a := (u, v)$ não $f-$saturado **ou** $\exists a := (v, u) f$-positivo, com $u \in X$ e $v \in V \setminus X$ **faça**
$X \leftarrow X \cup \{v\}$
$p(v) \leftarrow u$
se $y \in X$ **então**
$\epsilon(P) \leftarrow \min\{\epsilon(a) : a \in A(P)\}$ P obtido pela função pai $p(y)$ até x.
para $a \in P$ **faça**
se a direto **então** $f(a) \leftarrow f(a) + \epsilon(P)$
se a reverso **então** $f(a) \leftarrow f(a) - \epsilon(P)$
senão
retorne $(f, \partial^+(X))$

Ford-Fulkerson

- A eficiência do algoritmo depende da forma como é implementada a árvore de busca no grafo.
- Podemos criar a árvore usando DFS e BFS. DFS executa em $O(|f^*|E)$ o que é um pouco ruim, pois f^* sendo o valor do fluxo máximo representa um pseudo-polinomial (ou seja, uma solução exponencial).
- BFS pode ser implementado Em $O(VE^2)$ se aplicarmos a abordagem do algoritmo de Edmonds Karp.
- O algoritmo utiliza uma matriz de adjacência para guardar os fluxos de cada arco, com uma complexidde de espaço de $O(V^2)$

Edmonds Karp

- O algoritmo de Edmonds Karp trabalha de forma um pouco diferente a "Capacidade" e "Fluxo" de um arco.
- Se trabalharmos com o fluxo crescendo no limite da capacidade, então precisamos guardar para cada arco o fluxo e a capacidade.
- Edmonds Karp trabalha com a capacidade simplesmente, cada vez que o fluxo aumenta, a capacidade do arco diminui, até o seu limite zero.
- Assim, fazemos uma busca em largura, observando os arcos que possuem capacidade de fluxo (valor positivo), se encontrarmos vértices até o destino, então teremos, na árvore de busca, um caminho $f-$aumentante.

Edmonds Karp

- Esta abordagem facilita também o tratamento de arcos no caminho reverso.
- Ao aumentar o fluxo de um arco, significa que ao percorrê-lo por um caminho reverso, é possível diminuir seu fluxo (se encontrarmos um caminho $f-$aumentante que passe por este arco no sentido reverso.
- Na abordagem de Edmonds Karp, ao aumentar o fluxo (diminuir sua capacidade no caminho direto), nós aumentaremos sua capacidade no sentido reverso, assim se houver um caminho no sentido reverso, há a possibilidade de diminuir o fluxo do arco.
- Isto facilita uma abordagem por matriz de incidência, e também funciona com grafos não orientados, basta colocar a capacidade em ambos sentidos.
- Vamos nos preocupar apenas com fluxo máximo. Corte mínimo requer uma ligeira modificação.

Edmonds Karp - definição do grafo

```
class rede {
public:
 int e[MAX_V][MAX_V]; // Matriz de Adjacência
 int n; // Número de vértices
 vector<int> pai; // Estrutura pai da árvore de busca
 int o, d; // Vértices de origem e destino da rede

 rede(int _n, int _o, int _d, int _e[MAX_V][MAX_V]): n(_n), o(_o), d(_d) {
  for(int i=0; i<n; i++)
   for(int j=0; j<n; j++)
    e[i][j]=_e[i][j];
 }

 int setflow(int v, int emin); // Atualiza o caminho f-aumentante
 int maxflow(); // Busca em largura e Max Flow
}
```

Edmonds Karp - Atualizando o fluxo

```
int rede::setflow(int v, int emin) {
 int flow = 0; // Se não houver o caminho, o fluxo é zero
 if(v==o) flow = emin; // Tem caminho até a origem
 else if(pai[v]!=-1) { // O caminho ‘‘pai’’ na árvore de busca
  flow = setflow(pai[v],min(e[pai[v]][v],emin)); // busca
recursivamente aresta mínima
  e[pai[v]][v] -=flow; e[v][pai[v]] +=flow; // No caminho todo,
atualiza o fluxo (subtraindo capacidade direta, e aumentando a reversa)
 }
 return flow;
}
```

Edmonds Karp - Busca em Largura de caminhos

```
int maxflow() {
 bool fim = false; int mf = 0; //mf - Fluxo máximo
 while(!fim) { // Termina quando fluxo=0 (sem caminho)
  int fluxo = 0; // cada loop, uma nova BFS
  pai.assign(n,-1);
  vector<int> dist(n,INF); // não visitados
  queue<int> q; // Iniciando BFS
  dist[o] = 0; q.push(o);
  while(!fim && !q.empty()) {
   int u = q.front(); q.pop();
   if(u==d) fim = true; // achei o destino, paro este loop
   else {
    for(int v = 0; v<n; v++) { // vizinhos para a fila
     if(e[u][v] > 0 && dist[v]==INF)
      dist[v] = dist[u] + 1, q.push(v), pai[v] = u;
    }
   }
  }
  fluxo = setflow(d,INF); mf+=fluxo; // fluxo deste loop
  fim = fluxo == 0; // Continuamos achando caminhos f-aumentantes?
 }
 return mf;
}
```

URI 2082 - Viagens no Tempo

Albert Einstein nasceu na Alemanha, mas foi na Suíça, trabalhando como funcionário público, que escreveu em 1905 os trabalhos que revolucionaram a Física moderna e o tornaram famoso. Em 1921 ganhou o prêmio Nobel de Física pela descoberta da lei do efeito fotoelétrico. Muitos acham seus trabalhos sobre a Teoria da Relatividade os mais importantes de sua carreira, entretanto não foram os que renderam o valioso prêmio.

Einstein gostava muito de fazer "experimentos mentais" para avaliar suas teorias. Um desses experimentos é muito famoso e descreve um elevador caindo com um relógio dentro. A ideia de viagens no tempo acabaram surgindo como possíveis, desde que se descobrisse como construir máquinas que pudessem viajar em velocidades maiores do que a velocidade da luz. Certamente, num futuro não muito distante, isso será possível e poderemos viajar livremente entre as eras e ver eventos como o descobrimento do Brasil em 1500, a chegada da Família Real em 1808 ou o Corinthians campeão da Libertadores em 2962 ao vivo.

URI 2082 - Viagens no Tempo

Com as constantes viagens no tempo, será importante regular o serviço. As máquinas do tempo estarão espalhadas por toda a História e os viajantes terão de pegá-las para viajar para o presente ou para o futuro. Devido a restrições técnicas destas máquinas, não será possível viajar para qualquer instante do tempo diretamente, mas apenas para outros momentos históricos, de onde uma nova máquina poderá ser usada para seguir viagem. No entanto, estando em um momento histórico, você consegue ir para qualquer outro momento viajando por uma ou mais máquinas.

Juntamente com os viajantes do tempo, também surgirão os piratas da História, que tentarão roubar tesouros, inverter acontecimentos e mudar a história com os objetivos mais maldosos. Isso acarretará na criação da Polícia do Tempo. No ano de 2850 (antes do Corinthians ganhar sua primeira Libertadores) a Polícia do Tempo resolve isolar acontecimentos históricos, desabilitando ligações entre algumas máquinas. Cada ligação tem um custo associado para ser desabilitado, e sua tarefa é encontrar, dado um conjunto de momentos históricos, um conjunto de ligações – de custo mínimo – que ao serem desconectadas isolam os acontecimentos, ou seja, estando em uma máquina não será possível viajar para algumas das outras máquinas.

URI 2082 - Entrada/Saída

Entrada:
A entrada é composta por diversas instâncias. A primeira linha da entrada contém um inteiro **T** indicando o número de instâncias.

A primeira linha de cada instância contém dois inteiros **N** e **M** ($1 \leqslant \mathbf{N} \leqslant 100$ e $1 \leqslant \mathbf{M} \leqslant \mathbf{N} * (\mathbf{N}-1)/2$) indicando o número de máquinas e o número de ligações, respectivamente. Cada uma das **M** linhas seguinte possui três inteiros **u**, **v** e **c** ($1 \leqslant \mathbf{u}, \mathbf{v} \leqslant N$, $1 \leqslant \mathbf{c} \leqslant 100$) que representam a existência de uma ligação entre a máquina **u** e **v** com custo **c**. Tal ligação pode ser usada para viajar da máquina **u** para máquina **v** e também da máquina **v** para máquina **u**.

Saída:
Para cada instância imprima uma linha contendo a soma dos custos das ligações que devem ser removidas.

URI 2082 - Abordagem

- O que queremos é o menor dos cortes mínimos para o grafo.
- Ou seja, se considerarmos que a rede poderia ter qualquer origem e qualquer destino, então a origem e destino com o menor fluxo máximo e corte mínimo é a nossa solução.
- Infelizmente aplicar o algoritmo para todos os pares é inviável.
- Uma abordagem é escolher como origem, o vértice de menor grau.

```
int main() {
 int t, n, m, min, u, v, c, e[MAX_V][MAX_V], vmin[MAX_V];
 cin >> t;
 while(t--) {
  cin >> n >> m;
  for(int i=0; i<n; i++)
   for(int j=0; j<n; j++)
    e[i][j]=0, vmin[i]=0;
  for(int i=0; i<m; i++) {
   cin >> u >> v >> c; u--; v--;
   e[u][v] = e[v][u] = c; vmin[u]+=c; vmin[v]+=c;
  }
  int o=0;
  for(int i=1; i<n; i++) if(vmin[i] < vmin[o]) o=i;
  for(int i=0; i<n; i++) {
   if(i!=o) {
    rede r(n,o,i,e);
    if(i==0 || (o==0 && i==1)) min = r.maxflow();
    else { int f = r.maxflow(); if(f < min) min = f; }
   }
  }
  cout << min << endl;
 }
 return 0;
}
```