Fluxo em redes

Wladimir Araújo Tavares [1]

[1]Universidade Federal do Ceará - Campus de Quixadá

16 de maio de 2017

1 Fluxo em redes

Fluxo em redes

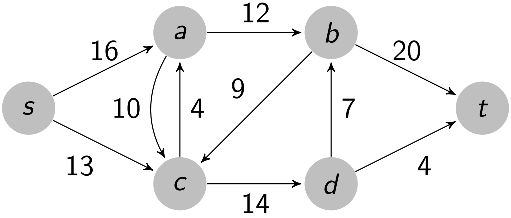
- Diversas aplicações em diferentes contextos:
  - Calcular a maior taxa de material que pode ser enviado desde da fonte até o sorvedouro.
  - Enviar a maior quantidade de caminhões considerando que as estradas possuem um limite de número de caminhões por unidade de tempo.
  - Minimizar o custo de destruir pontes a fim de desconectar duas cidades s e t..

Fluxo em redes

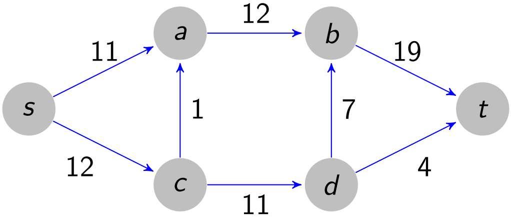
- Uma rede de fluxo $G = (V, E)$ é um grafo dirigido em que cada arco e tem uma capacidade $c(e) > 0$.
- Possui dois vértices especiais fonte s e sorvedouro t.
- Problema: maximizar a quantidade total de fluxo de s para t satisfazendo duas restrições:
  - o fluxo em cada aresta e não excede a capacidade da aresta, ou seja, $f(u, v) \leq c(u, v)$ (Restrição de capacidade).
  - Para cada vértice $v \neq s$ e $v \neq t$, o fluxo de entrada é igual ao fluxo de saída, ou seja, $\sum_{v \in N^+(u)} f(u, v) = \sum_{v \in N^-(u)} f(u, v)$ (Conservação do fluxo).

Fluxo em redes

- Capacidade



- Fluxo Máximo (23 unidades)

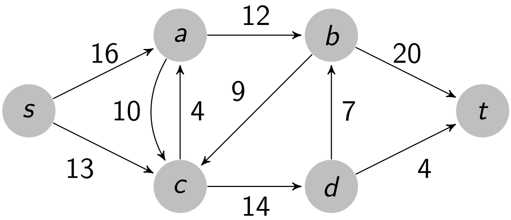


Corte mínimo

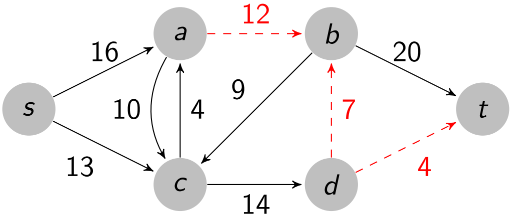
- Queremos remover algumas arestas do grafo tal que depois da remoção das arestas, não exista um caminho de s para t.
- O custo de remover uma aresta e será igual a sua capacidade $c(e)$
- O problema do custo mínimo consiste em encontrar um corte com o custo total mínimo.
- Teorema 26.7 (CLRS, 2° Edição): Fluxo máximo $=$ Custo mínimo

Exemplo

- Capacidade

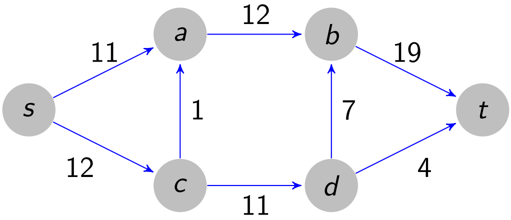


- Corte mínimo



Decomposição em Fluxos

- Um fluxo válido pode ser decomposto em caminhos aumentantes:



  - $s \rightarrow a \rightarrow b \rightarrow t : 11$
  - $s \rightarrow c \rightarrow a \rightarrow b \rightarrow t : 1$
  - $s \rightarrow c \rightarrow d \rightarrow b \rightarrow t : 7$
  - $s \rightarrow c \rightarrow d \rightarrow t : 4$

Algoritmo de Ford-Fulkerson

- Um algoritmo simples e prático para encontrar o fluxo máximo.
- Encontre caminhos em ampliação válidos no grafo residual G_f até que não exista mais nenhum e some o fluxo de todos eles.
- f é um fluxo máximo de G então não existe nenhum caminho de ampliação em G_f.

Algoritmo de Ford-Fulkerson

Algorithm 1 FordFulkerson(G,s,t)

function FordFulkerson(G, s, t)
  $f^* \leftarrow 0$
  for cada $(u, v) \in E(G)$ **do**
    $f[u, v] \leftarrow 0$
    $f[v, u] \leftarrow 0$
  while Enquanto existir um caminho em ampliação p de s até t na rede residual G_f **do**
    $c_f(p) \leftarrow \min\{c[u, v] - f[u, v] : (u, v) \in p\}$
    $f^* \leftarrow f^* + c_f(p)$
    for cada $(u, v) \in p$ **do**
      $f[u, v] \leftarrow f[u, v] + c_f(p)$
      $f[v, u] \leftarrow -f[u, v]$

Análise

- As capacidades são valores inteiros.
- Encontrar o caminho em ampliação deve ter complexidade $O(n+m)$
- A complexidade do algoritmo FordFulkerson é $O((n+m)f^*)$
- A complexidade depende do valor do fluxo máximo.

Representação do grafo

```
typedef struct Edge{
  int from,to, rev, f, cap;
  Edge(int from, int to, int rev, int f, int cap):
    from(from),to(to), rev(rev), f(f), cap(cap) {};
};

vector <Edge> g[MAXN];

void addEdge(int s, int t, int cap)
{
  g[s].push_back( Edge(s,t, g[t].size(), 0, cap)  );
  g[t].push_back( Edge(t,s, g[s].size()-1, 0, 0)  );
}
```

Caminho em ampliação

```
int find_path(int s, int t, int f){
  int v, df;
  if( s == t ) return f;
  vis[s] = true;
  for(int i = 0; i < g[s].size(); i++){
    Edge & e = g[s][i];
    if( e.cap - e.f <= 0) continue;
    v  = e.to;
    if( !vis[v] ){
      df = find_path(v, t, min(f, e.cap-e.f) );
      if( df > 0){
        e.f += df;
        g[v][e.rev].f -= df;
        return df;
      }
    }
  }
  return 0;
}
```

Ford-Fulkerson

```
/*
Algoritmo Ford-Fulkerson
Complexidade O((|V|+|E|)*|f|) = O(V^2*|f|)
*/
int ford_fulkerson(int _src, int _dest)
{
  int totflow, flow;
  src = _src;
  dest = _dest;
  totflow = 0;
  fill( vis, vis + nodes, false);
  while( flow = find_path(src, dest, 0x7fffffff) )
  {
    totflow += flow;
    fill( vis, vis + nodes, false);
  }
  printf("fluxo maximo %d\n", totflow);
  return totflow;
}
```

Algoritmo Edmonds-Karp

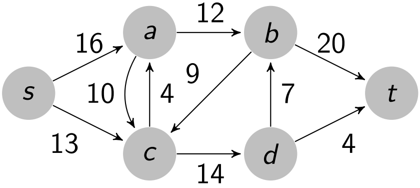
- O algoritmo de Edmonds-Karp é o algoritmo de Ford-Fulkerson utilizando o caminho mínimo para o caminho em ampliação.
- Esta modificação melhorar a complexidade do algoritmo para tempo polinomial.
- A distância do caminho mais curto na rede residual aumenta monotonicamente em cada ampliação do fluxo.
- O algoritmo de Edmonds-Karp executa em $O(VE^2)$

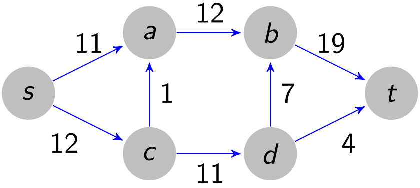
EdmondsKarp

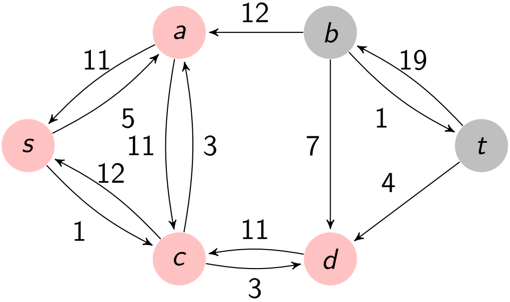
```
int bfs(int src, int dest, int *pred, int *edge){
  int q[nodes];
  int qt;
  bool vis[MAXN];
  qt = 0;
  q[qt++] = src;
  fill( pred, pred + nodes, -1);
  fill( vis, vis + nodes, false);
  for(int qh = 0; qh < qt && pred[dest] == -1; qh++){
    int u = q[qh];
    vis[u] = true;
    for(int i = 0; i < (int)g[u].size(); i++){
      Edge & e = g[u][i];
      if( e.cap - e.f <= 0) continue;
      int v  = e.to;
      if( !vis[v] ){
        pred[v] = u;
        edge[v] = i;
        q[qt++] = v;
      }
    }
  }
  if( pred[dest] == -1) return 0;
  else return 1;
```

Corte Mínimo

- Subtraia o fluxo máximo do grafo original.



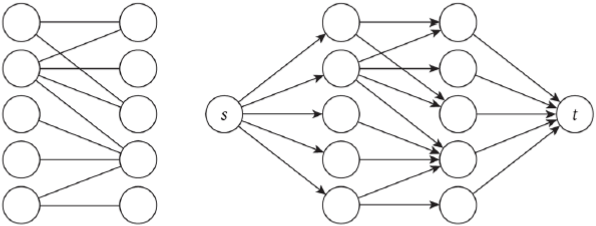




Emparelhamento em grafos bipartidos

- Um grafo bipartido $G(V, E)$ onde $V = X \cup Y$ com $X \cap Y = \emptyset$ e $E \subseteq X \times Y$.
- Grafos bipartidos modelam situações em os objetos são emparelhados com ou atribuídos para outros objetos. (Casamento, residentes/hospitais, tarefas/máquinas).
- Um emparelhamento em um grafo bipartido G é um conjunto $M \subseteq E$ tal que todo vértice de G incide em no máximo um elemento de M.
- Um conjunto de arestas M é um emparelhamento perfeito se todo vértice de V incide em exatamente um aresta de M.

Algoritmo para emparelhamento em grafos bipartidos



FiguraUm grafo bipartido e a correspondente rede de fluxos com todas as capacidades iguais a 1

- Converta o grafo $G = (X \cup Y, E)$ em uma rede de fluxo G': oriente as arestas de X para Y, adicione um vértice s e t, conecte s a cada vértice em X, conecte cada vértice de Y ao vértice t e sete todas as arestas com capacidade igual a 1.

Caminho disjuntos em arestas

- Um conjunto de caminhos em um grafo G é disjuntos em arestas se cada aresta em G aparece no máximo em um caminho.
  CAMINHO DIRECIONADO DISJUNTO EM ARESTAS
  INSTÂNCIA: Grafo direcionado $G = (V, E)$ com dois nós distintos s e t
  SOLUÇÃO: O número máximo de caminhos disjuntos em arestas entre s e t.

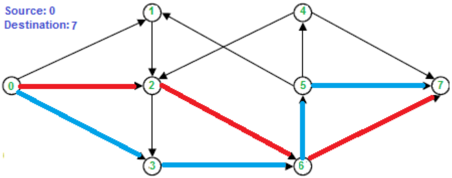
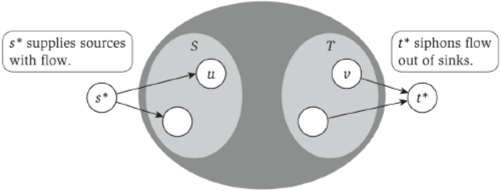


Figura Dois caminhos disjuntos em arestas.

Circulação com demandas

- Dado um grafo direcionado $G = (V, E)$ com a função de capacidade $c : E \rightarrow \mathbb{Z}^+$ e uma função de demanda $d : V \rightarrow \mathbb{Z}$:
  - $d(v) > 0$: nó destino e tem uma demanda de $d(v)$ unidade de fluxo.
  - $d(v) < 0$: nó fonte e tem uma oferta de $-d(v)$ unidades de fluxo.
  - $d(v) = 0$: nó recebe e transmite fluxo.
  - S é o conjunto de vértices com demanda negativa.
  - T é o conjunto de vértices com demanda positiva.
- Uma circulação com demandas é uma função $f : E \rightarrow \mathbb{Z}^+$ que satisfaz
  - Para cada aresta $e \in E$, $0 \leq f(e) \leq c(e)$
  - Para cada vértice v, $\sum_{u \in N^-(V)} f[u, v] - \sum_{u \in N^+(V)} f[v, u] = d(v)$

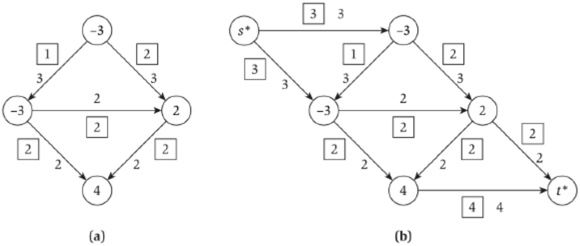
Redução do Problema de Circulação com demandas para o Problema do Fluxo Máximo



FiguraRedução do Problema de Circulação com demandas para o Problema do Fluxo Máximo

- Converta o grafo G em uma rede de fluxos G':
  1. Adicione uma fonte $s*$ e um destino $t*$.
  2. Conecte $s*$ a cada vértice $v \in S$ usando uma aresta com capacidade $-d(v)$.
  3. Conecte cada vértice de $v \in T$ ao vértice $t*$ usando uma aresta com capacidade $d(v)$.

Circulação com demandas



Figura(a) Uma instância do problema de circulação de demandas com a sua solução. Os números dentro do nó representam as demandas; números rotulando as arestas são as capacidades e os números dentro das caixas representam a função de circulação. (b) O resultado da redução desta instância em um problema equivalente de fluxo máximo.