Estruturas de Dados

Aula 16 — Redes de Fluxo

Prof Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia — Campus Feira de Santana

2026

Agenda

Redes de Fluxo

Algoritmo de Ford-Fulkerson

Edmonds-Karp

Teorema Max-Flow Min-Cut

Aplicações

Implementação em C

Exercícios

Redes de Fluxo

Capacidade, fluxo e conservação

O que é uma Rede de Fluxo?

Definição formal

$G = (V, E)$ dirigido, com:

Capacidade $c(u, v) \geq 0$: máximo que pode fluir pela aresta (u, v)

Fonte $s \in V$: produz fluxo

Sumidouro $t \in V$: consome fluxo

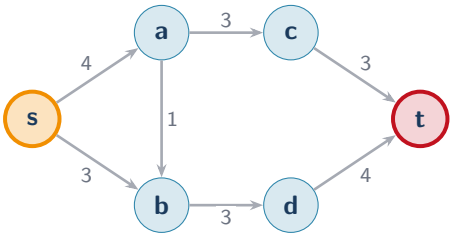
Fluxo $f(u, v)$: satisfaz:

- *Restrição de capacidade*: $0 \leq f(u, v) \leq c(u, v)$
- *Conservação*: para $v \neq s, t$:
  $$\sum_u f(u, v) = \sum_w f(v, w)$$

Valor do fluxo

$|f| = \sum_v f(s, v) - \sum_v f(v, s)$, **objetivo**: maximizar $|f|$.

Exemplo de rede de fluxo



- **s** = fonte, **t** = sumidouro
- Rótulos nas arestas = **capacidades**
- **Pergunta**: qual o fluxo máximo de s a t?

Grafo Residual

Ideia

Dado um fluxo f, o **grafo residual** $G_f = (V, E_f)$ representa quanto fluxo adicional pode ser enviado. Para cada aresta (u, v) com fluxo $f(u, v)$ e capacidade $c(u, v)$:

Aresta direta: capacidade residual
$c_f(u, v) = c(u, v) - f(u, v)$

Aresta reversa: permite "desfazer" fluxo
$c_f(v, u) = f(u, v)$

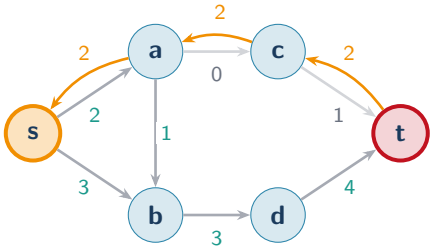
Caminho aumentante

Qualquer caminho $s \rightsquigarrow t$ no grafo residual G_f com $c_f(u, v) > 0$ em todas as arestas.

O **gargalo** (bottleneck) do caminho é $\delta = \min_{(u,v)\in P} c_f(u, v)$.

Aumentar δ unidades de fluxo ao longo de P gera um novo fluxo $f' = f \uparrow P$ com $|f'| > |f|$.

Grafo residual após f(s,a)=2, f(a,c)=2, f(c,t)=2



Arestas laranja = reversas (permitem cancelar fluxo)

Algoritmo de Ford-Fulkerson

Caminhos aumentantes iterativos

Ford-Fulkerson — Algoritmo

Pseudocódigo

Entrada: $G = (V, E, c)$, s, t

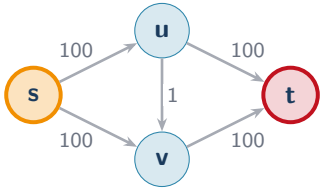
1. Para toda aresta (u, v): $f(u, v) \leftarrow 0$
2. **Enquanto** existe caminho $P : s \rightsquigarrow t$ em G_f:
   a. $\delta \leftarrow \min_{(u,v) \in P} c_f(u, v)$
   b. Para cada $(u, v) \in P$:
      $f(u, v) \leftarrow f(u, v) + \delta$
      $f(v, u) \leftarrow f(v, u) - \delta$
3. Retornar f

Complexidade

Se capacidades são inteiras: termina em $\mathcal{O}(|f^*|)$ iterações
Cada iteração: $\mathcal{O}(E)$ para busca de caminho
Total: $\mathcal{O}(E \cdot |f^*|)$
Pode ser lento se $|f^*|$ é grande!

Problema: escolha de caminho

Ford-Fulkerson **não especifica** como encontrar o caminho aumentante. Com DFS e capacidades irracionais, pode não convergir. Com inteiros, convergência é garantida, mas pode ser lenta:



DFS poderia usar $s \to u \to v \to t$ e $s \to v \to u \to t$ alternadamente:

200 iterações para fluxo máximo = 200!

Ford-Fulkerson — Trace no Exemplo

Rede:

$s \to a$:4, $s \to b$:3, $a \to c$:3, $a \to b$:1 $b \to d$:3, $c \to t$:3, $d \to t$:4

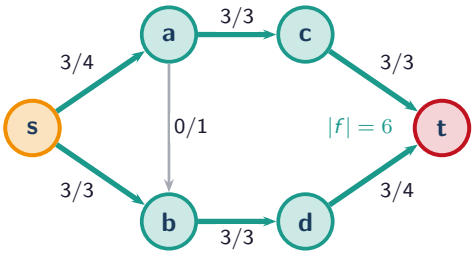
It.	Caminho P	δ	$\lvert f \rvert$
1	$s \to a \to c \to t$	3	3
2	$s \to b \to d \to t$	3	6

Nenhum caminho em G_f → fluxo máximo = 6

Verificação

Saída de s: $f(s,a) + f(s,b) = 3 + 3 = 6$
Entrada em t: $f(c,t) + f(d,t) = 3 + 3 = 6$ ✓
Conservação em a: $3 = 3 + 0$ ✓
Conservação em b: $3 = 3 + 0$ ✓

Fluxo final (formato fluxo/capacidade)



verde = aresta saturada ou com fluxo

Edmonds-Karp

Ford-Fulkerson com BFS

Edmonds-Karp — Correção de Ford-Fulkerson

A ideia

Substituir a busca de caminho por **BFS** no grafo residual. BFS sempre encontra o caminho **mais curto** (menor número de arestas) de s a t. Esta escolha garante complexidade **polinomial independente do valor do fluxo**.

Complexidade

Número de aumentações: $\mathcal{O}(VE)$
Cada BFS: $\mathcal{O}(E)$
Total: $\mathcal{O}(VE^2)$
Independe do valor do fluxo!

Comparação: DFS vs BFS em Ford-Fulkerson

Ford-Fulkerson (DFS)		Edmonds-Karp (BFS)
$\mathcal{O}(E \cdot \lvert f^* \rvert)$	Complexidade	$\mathcal{O}(VE^2)$
Pode ser exponencial	Pior caso	Sempre polinomial
Não converge com irracional	Irracionais	Converge sempre

Por que BFS?

Com BFS, a distância de s a cada vértice em G_f **nunca decresce** ao longo das iterações. Isso limita o número de vezes em que cada aresta pode ser um gargalo: no máximo $\mathcal{O}(V)$ vezes.

Teorema Max-Flow Min-Cut

O resultado fundamental

Definição de Corte

Um (s,t)-**corte** é uma partição (S,T) de V tal que $s \in S$ e $t \in T$. **Capacidade do corte:**

$$c(S,T) = \sum_{\substack{u \in S \\ v \in T}} c(u,v)$$

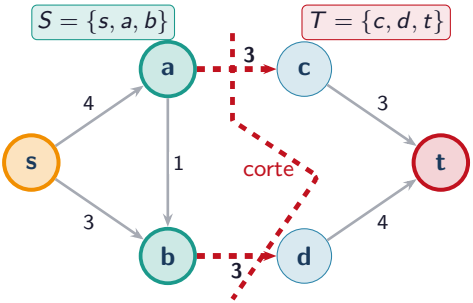
(soma das capacidades das arestas de S para T; arestas de T para S **não** contam)

Teorema Max-Flow Min-Cut

Para qualquer rede de fluxo G, fonte s e sumidouro t, as seguintes afirmações são equivalentes:

f é um fluxo **máximo**
O grafo residual G_f **não possui** caminho $s \rightsquigarrow t$
$|f| = c(S,T)$ para algum **corte mínimo** (S,T)

Corte mínimo no exemplo (custo = 6)



$$c(S,T) = c(a,c) + c(b,d) = 3 + 3 = 6 = |f^*|$$

Interpretação do Teorema

Intuição física

Imagine a rede como um sistema de canos. O fluxo máximo que pode fluir de s a t é limitado pela "garganta" mais estreita da rede — o corte de capacidade mínima.

É impossível enviar mais do que a capacidade total de qualquer corte que separa s de t.

Corolário

Para encontrar o corte mínimo após rodar Edmonds-Karp:

1. No grafo residual G_f final, faça BFS de s.
2. $S =$ vértices alcançáveis a partir de s.
3. $T = V \setminus S$.
4. Arestas do corte = arestas de S para T no grafo original.

Aplicações do teorema

Segurança de redes: identificar enlaces críticos (corte mínimo)
Logística: gargalo de uma cadeia de suprimentos
Bipartite matching: o teorema implica o Teorema de König
Separação de vértices: encontrar conjunto mínimo de vértices que desconecta s de t
Análise de vulnerabilidade: quais k arestas remover para desconectar a rede?

Teorema de König

Em grafos bipartidos:

máximo matching = mínimo vertex cover

Consequência direta do Max-Flow Min-Cut.

Aplicações

Matching bipartido e além

Matching Bipartido via Fluxo Máximo

Problema

Dado grafo bipartido $G = (X \cup Y, E)$, encontrar o **maior conjunto de arestas** sem vértices em comum (*maximum matching*).

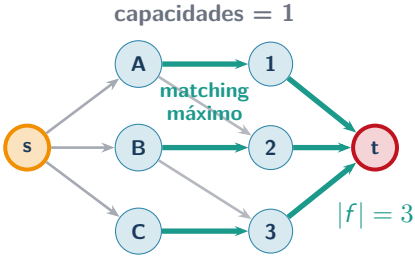
Exemplo: atribuir k tarefas a k trabalhadores, cada um com aptidões diferentes.

Redução ao fluxo

1. Adicionar **fonte** s com arestas $s \to x_i$, $c = 1$, para todo $x_i \in X$
2. Adicionar **sumidouro** t com arestas $y_j \to t$, $c = 1$, para todo $y_j \in Y$
3. Todas as arestas bipartidas: $c = 1$
4. Calcular **fluxo máximo** de s a t

Valor do fluxo = tamanho do matching máximo.

Grafo bipartido e sua rede de fluxo



A→1, B→2, C→3: matching de tamanho 3 = $|f^*|$

Outras Aplicações de Fluxo

Roteamento em redes

Determinar a largura de banda máxima entre dois pontos de uma rede de computadores ou de transporte, respeitando as capacidades de cada enlace. O corte mínimo identifica os gargalos críticos.

Seleção de projetos

Cada projeto tem lucro e custo de recursos. Modelar como fluxo encontra quais projetos maximizam o lucro líquido. Usado em finanças e engenharia.

Segmentação de imagem

Pixels são vértices; bordas são arestas com peso (diferença de intensidade). Min-cut separa objeto do fundo. Base de algoritmos como GrabCut.

Fluxo com custo mínimo

Extensão: cada aresta tem **capacidade** e **custo por unidade**. Objetivo: enviar exatamente f unidades de s a t com custo total mínimo.
Algoritmos: Bellman-Ford sobre o grafo residual (MCMF), algoritmo de Hungarian, SPFA.

Fluxo em redes bipartidas

Alocação de tarefas: trabalhadores $\times$ tarefas
Casamento estável: Gale-Shapley
Escalonamento: turnos $\times$ funcionários
Transporte: origens $\times$ destinos

Implementação em C

Edmonds-Karp: BFS + Ford-Fulkerson

BFS para Encontrar Caminho Aumentante

```c
#include <stdio.h>
#include <string.h>
#include <limits.h>
#define MAX 200

int cap[MAX][MAX];//capacidade residual
int pai[MAX];//vetor de predecessores
int visitado[MAX];

//retorna 1 se existe caminho s->t;
//preenche pai[]
int bfs(int s,int t,int V){
  memset(visitado,0,sizeof(visitado));
  memset(pai, -1, sizeof(pai));
  int fila[MAX], ini = 0, fim = 0;
  visitado[s] = 1; fila[fim++] = s;

//continua ao lado ->
```

```c
  while(ini < fim){
    int u = fila[ini++];
    for(int v = 0; v < V; v++){
      if(!visitado[v] && cap[u][v] > 0){
        visitado[v] = 1; pai[v] = u;
        if (v == t) return 1;//achou caminho
        fila[fim++] = v;
      }
    }
  }
  return 0;//sem caminho aumentante
}
```

Edmonds-Karp — Algoritmo Principal

```c
/* Edmonds-Karp: retorna o fluxo máximo de s a t */
int edmondsKarp(int s, int t, int V) {
  int fluxo_total = 0;
  /* bfs encontra caminho aumentante e preenche pai[] */
  while (bfs(s, t, V)) {
    /* calcula o gargalo (bottleneck) do caminho */
    int delta = INT_MAX;
    for (int v = t; v != s; v = pai[v]) {
      int u = pai[v];
      if (cap[u][v] < delta) delta = cap[u][v];
    }
    /* atualiza capacidades residuais */
    for (int v = t; v != s; v = pai[v]) {
      int u = pai[v];
      cap[u][v] -= delta;//consome capacidade direta
      cap[v][u] += delta;//adiciona capacidade reversa
    }
    fluxo_total += delta;
    printf("Aumentação: delta=%d,fluxo_total=%d\n",delta,fluxo_total);
  }
  return fluxo_total;
}
```

Exercícios

Pratique fluxo em redes

Exercícios

1. **Trace manual (Edmonds-Karp):** Aplique o algoritmo de Edmonds-Karp (use BFS para cada caminho aumentante) na rede abaixo. Mostre o grafo residual após cada iteração e identifique o corte mínimo ao final.

   Arestas: $s \to u{:}5$, $s \to v{:}4$, $u \to v{:}3$, $u \to t{:}3$, $v \to t{:}5$

2. **Corte mínimo:** Dado o grafo do exercício 1, identifique **todos** os cortes (s, t) e suas capacidades. Qual é o corte mínimo? Verifique que seu valor coincide com o fluxo máximo encontrado.

3. **Matching bipartido:** Monte a rede de fluxo equivalente ao grafo bipartido $X = \{P, Q, R\}$, $Y = \{1, 2, 3, 4\}$, arestas: $P \to 1$, $P \to 3$, $Q \to 2$, $Q \to 3$, $R \to 2$, $R \to 4$. Execute Edmonds-Karp e determine o matching máximo.

4. **Implementação:** Adapte o código de Edmonds-Karp para também imprimir as arestas do corte mínimo ao final. *Dica*: após a última BFS, vértices alcançáveis a partir de s formam S.

Fim da Aula 16

Dúvidas e próximos passos

Encerrando a Aula 16

Resumo da aula

- **Rede de fluxo**: capacidades, conservação, valor $|f|$
- **Grafo residual**: arestas diretas $(c - f)$ e reversas (f)
- **Ford-Fulkerson**: caminhos aumentantes, $\mathcal{O}(E|f^*|)$
- **Edmonds-Karp**: BFS garante $\mathcal{O}(VE^2)$, independente do fluxo
- **Max-Flow Min-Cut**: $|f^*| = c(S^*, T^*)$, três condições equivalentes
- **Bipartite matching**: redução em $\mathcal{O}(1)$, fluxo resolve em $\mathcal{O}(VE)$

Próxima aula — Programação Dinâmica

- Subestrutura ótima e sobreposição de subproblemas
- Memoização vs tabulação (top-down/bottom-up)
- Longest Common Subsequence (LCS)
- 0/1 Knapsack
- Edição de distância (Levenshtein)

carolsoko@ifba.edu.br

Dúvidas? Mande uma mensagem!