# Algoritmos para r-Arborescências Geradoras Mínimas em Digrafos:
## Uma Aplicação Web Interativa

Lorena Sampaio, Samira Haddad
Orientador: Prof. Dr. Mário Leston Rey

Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

25 de novembro de 2025

# Sumário

# O Problema

## Encontrar uma $r$-Arborescência Geradora de Custo Mínimo

Dado um $r$-digrafo ponderado $(D, w, r)$:

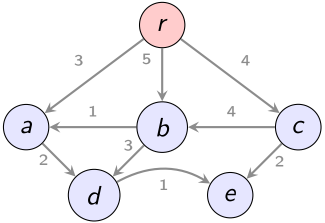
- Encontrar uma $r$-arborescência geradora de custo mínimo de $D$

**Algoritmos estudados:**

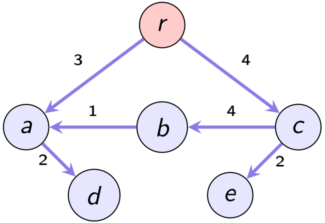
1. Chu-Liu-Edmonds (1965-67)
2. András Frank (1981-2014)

UFABC

# Exemplo: $r$-Arborescência Geradora Mínima

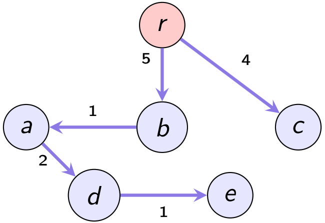



Digrafo Original




r-Arborescência Geradora

Custo: 16




Geradora Mínima

Custo: 13

# Chu-Liu-Edmonds: Ideia Principal

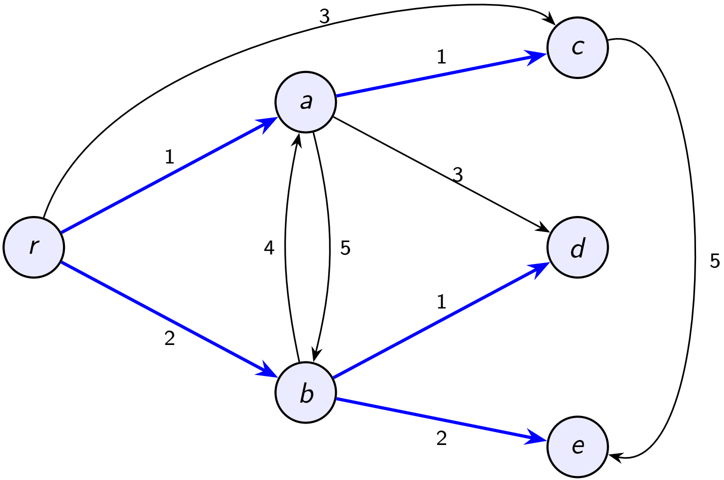
## Algoritmo Recursivo: dado um r-digrafo ponderado $(D, w, r)$

chu–liu–edmonds($(D, w, r)$):

1. **Reduzir custos**: para cada vértice $v \neq r$, subtrair $\lambda(v) = \min\{w(a) : a \in \delta^-(v)\}$
2. **Construir** $D_0$: escolhendo um arco $a_v$ de custo reduzido zero para cada $v \neq r$
3. **Verificar**: se $D_0$ é uma $r$-arborescência $\Rightarrow$ **devolver** $D_0$
   Caso contrário:
4. **Contração**: encontrar ciclo $C$ em $D_0$ e contrair
5. **Chamada recursiva**: Seja $D' = D/C$ e $w' = w_\lambda/C$. Calcular $T' =$ chu-liu-edmonds($D', w', r$)
6. **Devolver**: expandir($T'$)

UFABC

# Digrafo de Exemplo

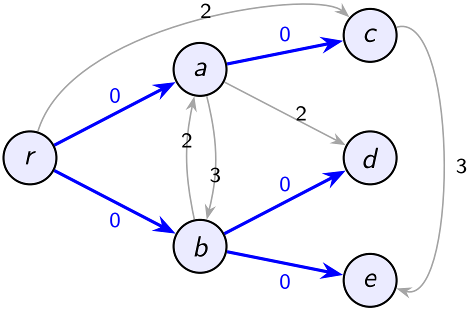


# Passo 1: Redução de Custos e Construção de $D_0$

**Redução de custos $\lambda$:**

- $\lambda(a) = 1$ (arco $(r, a)$)
- $\lambda(b) = 2$ (arco $(r, b)$)
- $\lambda(c) = 1$ (arco $(a, c)$)
- $\lambda(d) = 1$ (arcos $(b, d)$)
- $\lambda(e) = 2$ (arco $(b, e)$)

*Escolhemos arcos de custo zero para formar $D_0$*




## $D_0$ é uma $r$-arborescência!

Neste exemplo, $D_0 = \{(r, a), (r, b), (a, c), (b, d), (b, e)\}$ forma uma $r$-arborescência.
*Em outras palavras:* todos os vértices são alcançáveis a partir de $r$ e cada vértice tem exatamente um arco entrando

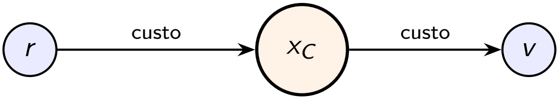
# Caso com Ciclo: Outro Exemplo

**Quando $D_0$ tem ciclo:**

Se após redução de custos, $D_0$ contém um ciclo $C$, contraímos $C$ em supervértice $x_C$

*Em outras palavras:* todos os vértices de $C$ viram um único vértice

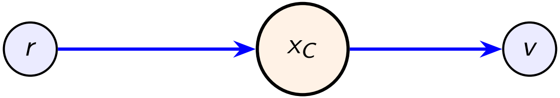
Custos são ajustados usando os custos $\lambda$-reduzidos




## Chamada Recursiva

Resolvemos chu-liu-edmonds($D/C \mapsto x_C, w_\lambda/C \mapsto x_C, r$)
*Em outras palavras:* aplicamos o mesmo algoritmo no digrafo menor

# Passo 3: Solução Recursiva (quando há ciclo)




O algoritmo devolve $T'$ (arborescência no digrafo contraído)

*Em outras palavras:* encontramos a $r$-arborescência ótima no grafo contraído

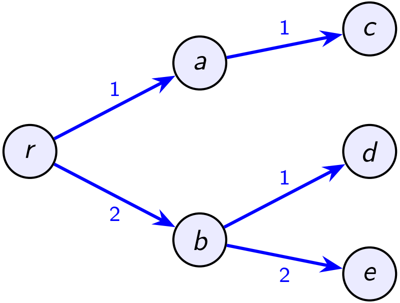
**Próximo passo:** expandir $T'$ de volta para obter $T$ no digrafo original

# Solução Final

**Para nosso exemplo:**
Como $D_0$ já é uma $r$-arborescência, o algoritmo termina diretamente!

*Em outras palavras:*

- Não há ciclo
- Todos vértices alcançáveis de $r$
- Cada vértice tem grau de entrada 1
- Solução ótima encontrada!




Solução Ótima

Custo total: $1 + 2 + 1 + 1 + 2 = 7$

# András Frank: Visão Geral

## Abordagem em Duas Fases

**Fase I:** Construir cobertura de subconjuntos minimais via redução de custos
**Fase II:** Extrair arborescência da cobertura

**Diferencial:**

- Trabalha com múltiplos vértices simultaneamente
- Usa componentes fortemente conexas
- Redução sistemática de custos

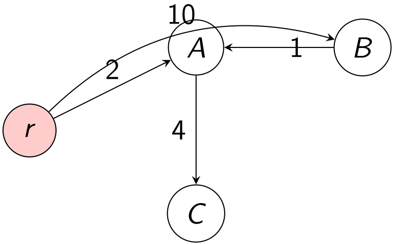
**Complexidade:**

- Fase I: $O(nm)$
- Fase II v1 (lista): $O(n^2)$
- Fase II v2 (heap): $O(n \log n)$

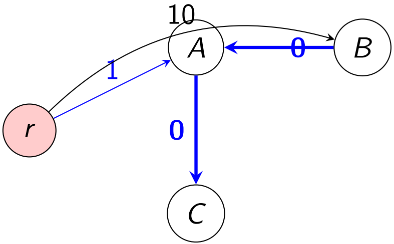
UFABC

# Fase I: Redução de Custos

**Para cada vértice** $v \neq r$: subtrair o mínimo de entrada




**Original**




**Após Redução**

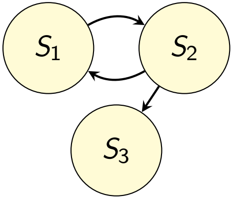
Arcos com custo **zero** formam o digrafo $D_0$

# Fase I: Componentes Fortemente Conexas

Identificar componentes fortemente conexas (CFCs) em $D_0$

Cada CFC forma um **subconjunto minimal**

Construir sequência laminar de subconjuntos




## Condição de Otimalidade

Sequência $\lambda$ satisfaz: $|\delta^-(X)| = 1$ para cada $X$ em $\lambda$

# Fase II: Construção da Arborescência

**Objetivo:** Extrair arborescência de $D_0$ respeitando $\lambda$

1. Iniciar com conjunto $R = \{r\}$
2. Para cada $v$ fora de $R$:
   - Selecionar arco $(u, v)$ com $u \in R$ e custo reduzido zero
   - Adicionar $v$ a $R$
3. Repetir até incluir todos os vértices

## Resultado

Arborescência ótima com mesma solução: custo 14

UFABC

# Comparação de Desempenho

**Experimentos:** 2000 digrafos aleatórios, $|V| \in [101, 4996]$

| **Algoritmo** | **Tempo Mediano** | **Tempo Médio** |
|---|---|---|
| Chu-Liu-Edmonds | 0,25 s | 0,58 s |
| Frank Fase I | 8,93 s | 12,40 s |
| Frank Fase II (lista) | 0,98 s | 1,34 s |
| Frank Fase II (heap) | **0,016 s** | **0,020 s** |

## Speedup Fase II

Heap vs Lista: aceleração de **58,12 vezes** (mediana)

UFABC

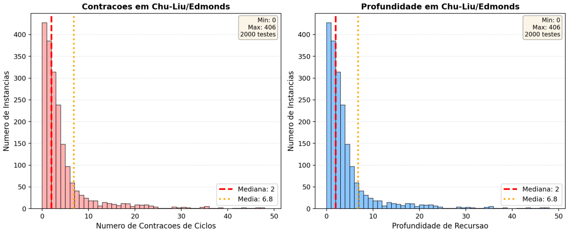
# Características Estruturais

**Contrações (Chu-Liu):**

- Mediana: 2 contrações
- Média: 6,82
- Máximo: 406
- 93,8% com $< 20$

Muito abaixo do limite teórico $O(n)$

**Consumo de memória:** mediana 11,5 MB (Fase I)



# Motivação Didática

**Desafio**

Algoritmos de grafos são **abstratos** e **difíceis de visualizar**

**Solução Proposta:**

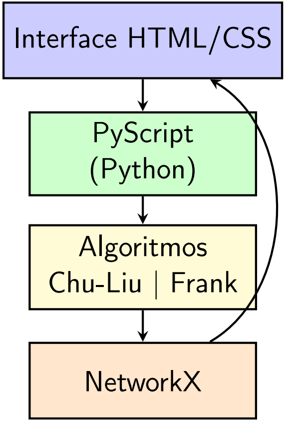
- Visualização interativa
- Execução passo a passo
- Feedback imediato
- Acessível via navegador

**Tecnologias:**

- PyScript (Python no browser)
- JavaScript
- HTML5/CSS3
- NetworkX

UFABC

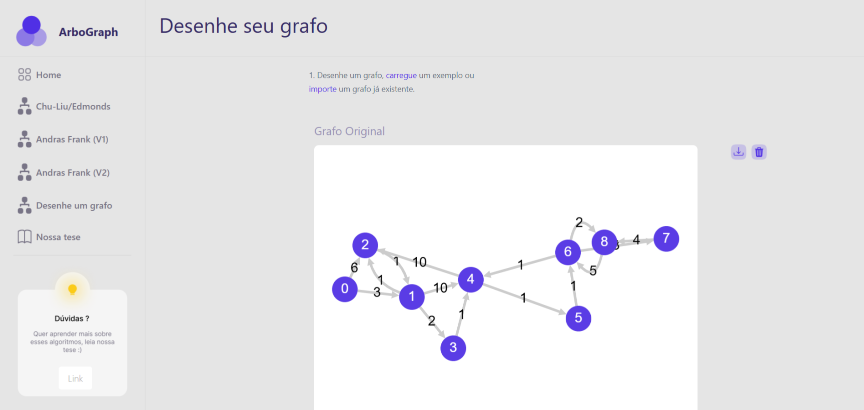
# Arquitetura da Aplicação



# Interface: Página Principal


UFABC

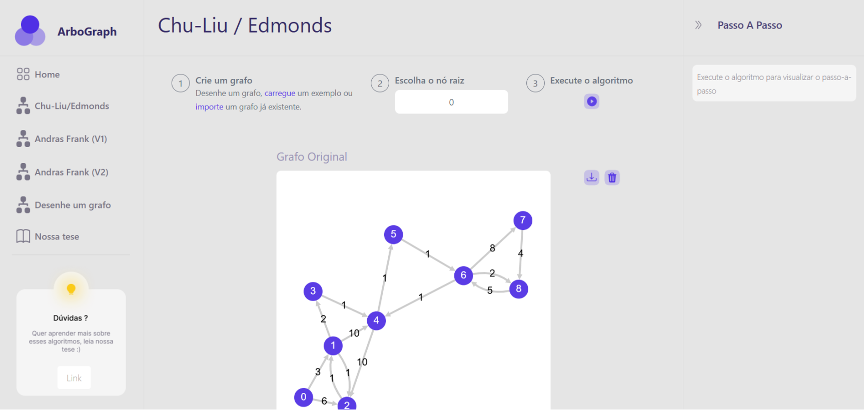
# Interface: Desenho de Grafos




Funcionalidades:

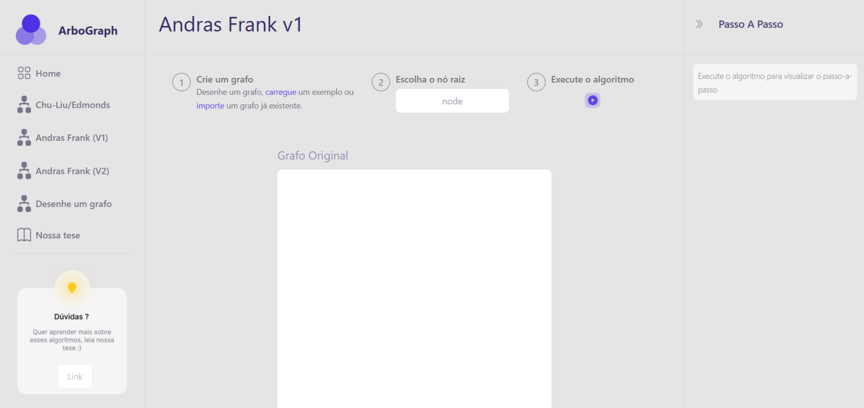
- Adicionar vértices e arestas
- Definir pesos

# Interface: Chu-Liu-Edmonds




- Visualização passo a passo
- Destacamento de ciclos detectados
- Log detalhado das operações

# Interface: András Frank




- Exibição das duas fases
- Visualização de CFCs
- Comparação entre versões (lista vs heap)

# Princípios de Design

## Teoria dos Registros de Representação (Duval)

Transitar entre diferentes representações:

- **Visual:** diagramas do grafo
- **Simbólico:** código Python
- **Textual:** log das operações

## Feedback Imediato

Validação em tempo real das operações do usuário

UFABC

# Contribuições do Trabalho

1. **Implementação completa** de dois algoritmos clássicos
   - Chu-Liu-Edmonds: recursivo com contração
   - András Frank: duas fases com otimização heap
2. **Análise experimental** detalhada
   - 2000 instâncias aleatórias
   - Comparação de desempenho e características estruturais
3. **Aplicação web interativa**
   - Ferramenta didática para visualização
   - Execução passo a passo dos algoritmos
   - Design centrado no usuário

UFABC

# Principais Resultados

- **Corretude validada:** custos idênticos em todas as instâncias
- **Chu-Liu-Edmonds** mais rápido para construção direta
  - Mediana: 0,25 s vs 8,93 s (Fase I Frank)
- **Otimização heap** fundamental na Fase II
  - Speedup: 58$\times$ (mediana), 61$\times$ (média)
- **Comportamento prático** muito melhor que limites teóricos
  - Contrações: mediana 2 (limite $O(n)$)
  - Memória modesta: 11,5 MB

# Trabalhos Futuros

## Extensões Possíveis

- Implementar outras variantes (Tarjan, Gabow)
- Análise em grafos com estruturas especiais
- Paralelização dos algoritmos
- Extensão para grafos dinâmicos

## Melhorias na Aplicação

- Modo de edição visual de grafos
- Geração automática de casos de teste
- Exercícios interativos com correção automática
- Integração com plataformas de ensino (Moodle, Jupyter)

# Obrigado!

Perguntas?

https://github.com/lorenypsum/graph-visualizer