

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

26 de novembro de 2025



Sumário

- 1 Introdução
- 2 Algoritmo de Chu-Liu-Edmonds
- 3 Algoritmo de András Frank
- 4 Resultados Experimentais
- 5 Aplicação Web
- 6 Conclusões

O Problema

Encontrar uma r -Arborescência Geradora de Custo Mínimo

Dado um r -digafo ponderado (D, w, r) :

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

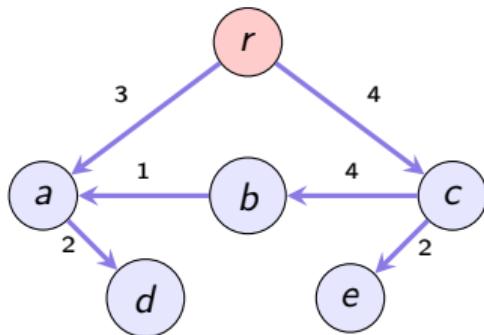
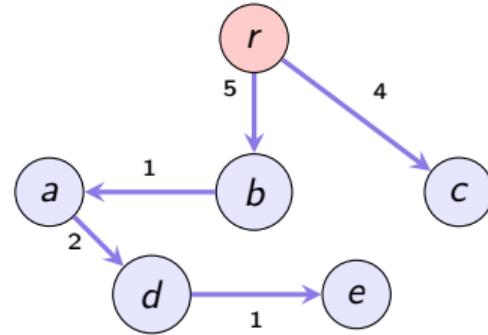
Algoritmos estudados:

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

Exemplo: r -Arborescência Geradora Mínima



Digrafo Original

 r -Arborescência Geradora

Geradora Mínima

Custo: 16

Custo: 13

Chu-Liu-Edmonds

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

chu-liu-edmonds((D, w, r)):

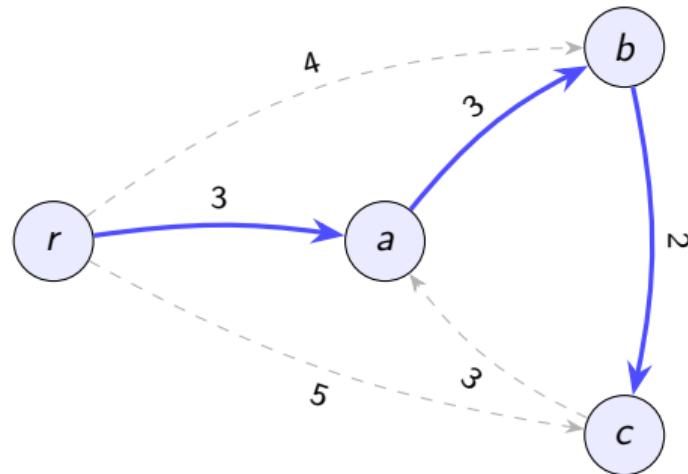
- ① **Reduzir custos:** para cada vértice $v \neq r$, subtrair $\lambda(v) = \min\{w(a) : a \in \delta^-(v)\}$
- ② **Construir D_0 :** escolhendo um arco a_v de custo reduzido zero para cada $v \neq r$
- ③ **Verificar:** se D_0 é uma r -arborescência \Rightarrow **devolver D_0**
Caso contrário:
- ④ **Contração:** 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')

Escolha Gulosa

Definição:

Para cada $v \neq r$, escolher um arco a_v de custo mínimo que entra em v :

$$T := \{a_v : v \in V \setminus \{r\}\}$$



Propriedade:

Se T é uma r -arborescência, então T tem custo mínimo.

Resultado

$T = \{(r, a), (a, b), (b, c)\}$ é uma r -arborescência de custo mínimo!

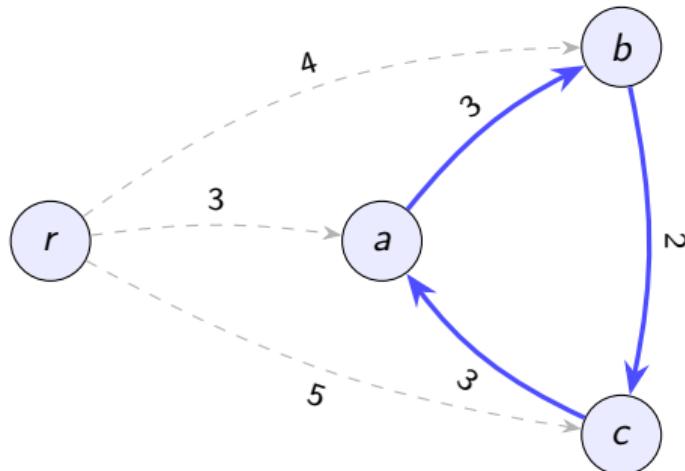
E quando a escolha gulosa falha?

Problema:

A escolha gulosa pode produzir um conjunto T que *não* é uma r -arborescência.

Exemplo:

Os arcos de custo mínimo formam um ciclo (a, b, c, a) sem alcançar r .



Arcos azuis formam um **ciclo!**

Passo 1: Redução de Custos

Definição:

Para cada $v \in V \setminus \{r\}$:

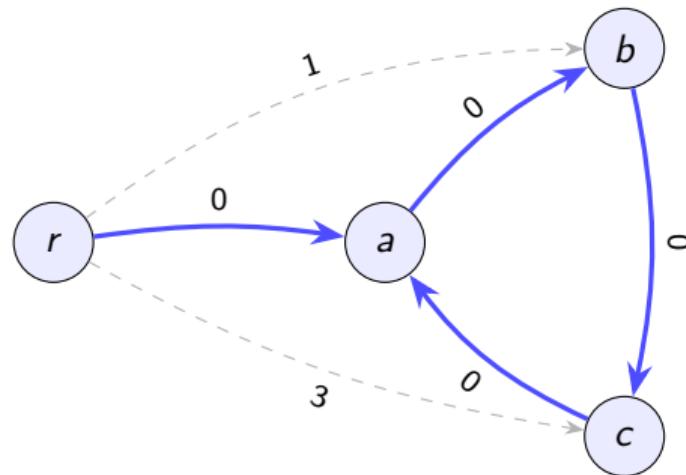
$$\lambda(v) := \min\{w(a) : a \in \delta^-(v)\}$$

Custo λ -reduzido:

$$w_\lambda(uv) := w(uv) - \lambda(v)$$

Valores de λ :

- $\lambda(a) = 3, \lambda(b) = 3, \lambda(c) = 2$



Arcos do ciclo têm custo zero!

Arcos com custo zero são candidatos para D_0

Passo 1: Implementação da Redução de Custos em Python

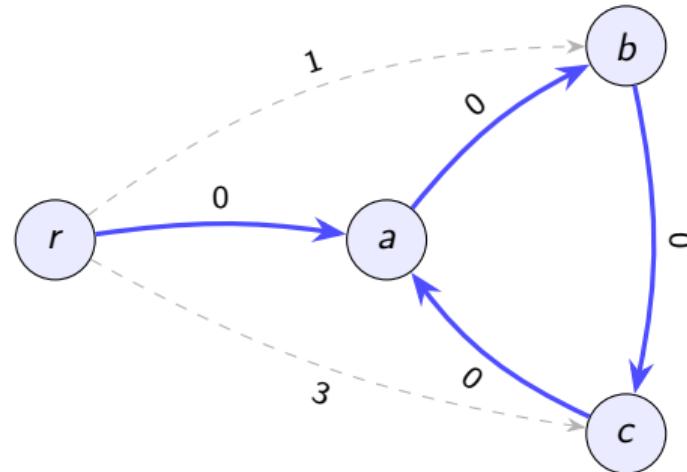
Passo 2: Construção de D_0

Formação de D_0 : Para cada $v \neq r$, escolher um arco $a_v \in \delta^-(v)$ com $w_\lambda(a_v) = 0$ formar:

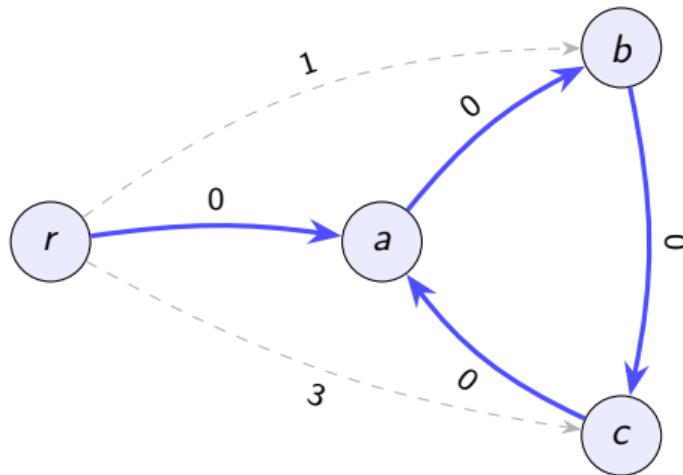
$$D_0 := (V, \{a_v : v \in V \setminus \{r\}\})$$

Arcos escolhidos:

- (r, a)
- $(a, b), (c, a)$



Passo 2: Implementação da Construção de D0 em Python

Passo 3: Verificação de D_0 **Verificar:**Se D_0 é uma r -arborescência \Rightarrow **devolver** D_0 **Caso contrário:** D_0 contém algum ciclo C . \Rightarrow **prosseguir** para os passos 4 e 5. D_0 não é uma r -arborescência!Neste exemplo, $D_0 = \{(r, a), (a, b), (c, a)\}$ não forma uma r -arborescência pois contém o ciclo (a, b, c, a) .

Passo 3: Implementação da Verificação de D0 em Python

Passo 4: Contração de Ciclos

Operação:

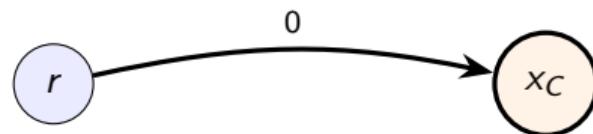
Contrair ciclo C em supervértice x_C .

Novo problema: (D', w', r) onde:

- $D' := D/C \mapsto x_C$
- $w' := w_\lambda/C \mapsto x_C$

O arco de D' que entra em x_C deve corresponder ao arco de D que entra em algum vértice de C

Podem ter arcos saindo de x_C em D' .



Digrafo contraído D'

Propriedade

Uma solução ótima em D' pode ser expandida para uma solução ótima em D .

Passo 4: Implementação da Contração de Ciclos em Python

Passo 5: Chamada Recursiva

Novo problema: (D', w', r)

Chamada recursiva:

$$T' := \text{chu-liu-edmonds}(D', w', r)$$



r-arborescência ótima em D'

Resultado: T' é uma r -arborescência de custo mínimo em (D', w')

Passo 5: Implementação da Chamada Recursiva em Python

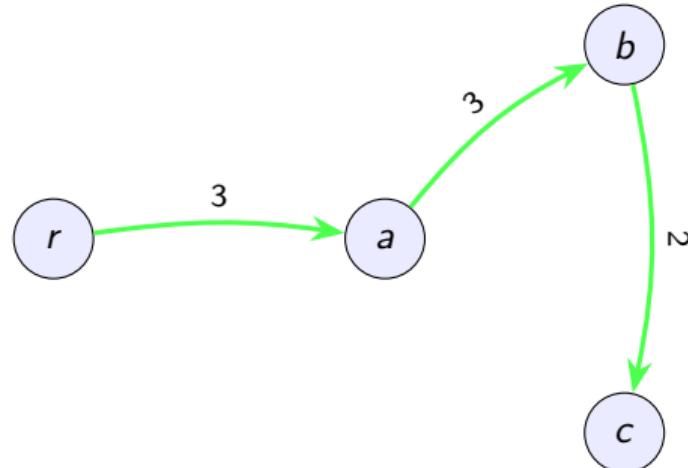
Passo 6: Reexpansão da Solução

Dado: T' ótima em (D', w')

Construir: T ótima em (D, w)

Procedimento:

- ① Seja uv o arco de D correspondente ao arco ux_C de T'
- ② Incluir uv em T
- ③ Incluir todos os arcos de C exceto aquele que entra em v



Resultado: T é uma r -arborescência de custo mínimo

r -arborescência final no digrafo original

Passo 6: Implementação da Reexpansão da Solução em Python

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:

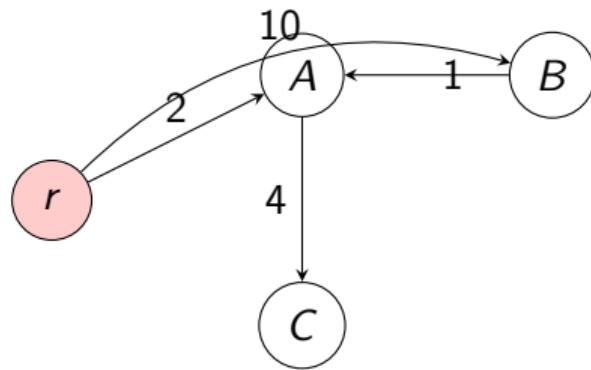
- Trabalha com múltiplos vértices simultaneamente
- Usa componentes fortemente conexas
- Redução sistemática de custos

Complexidade:

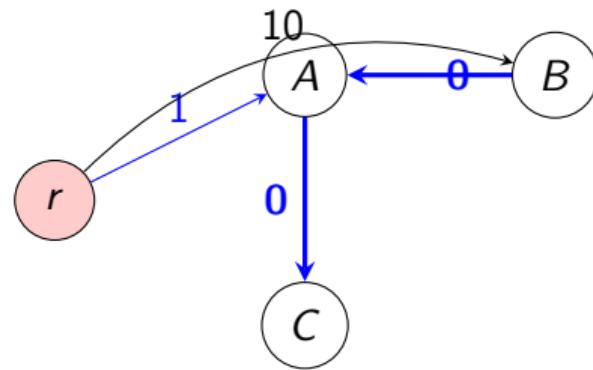
- Fase I: $O(nm)$
- Fase II v1 (lista): $O(n^2)$
- Fase II v2 (heap): $O(n \log n)$

Fase I: Redução de Custos

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



Original



Após Redução

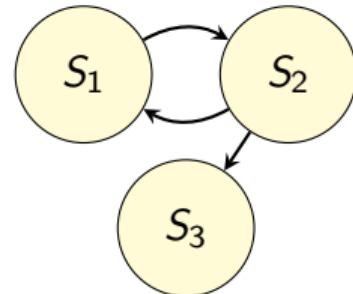
Arcos com custo zero formam o digrafo D_0

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 Optimalidade

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

Fase II: Construção da Arborescência

Objetivo: Extrair arborescência de D_0 respeitando λ

- 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

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)

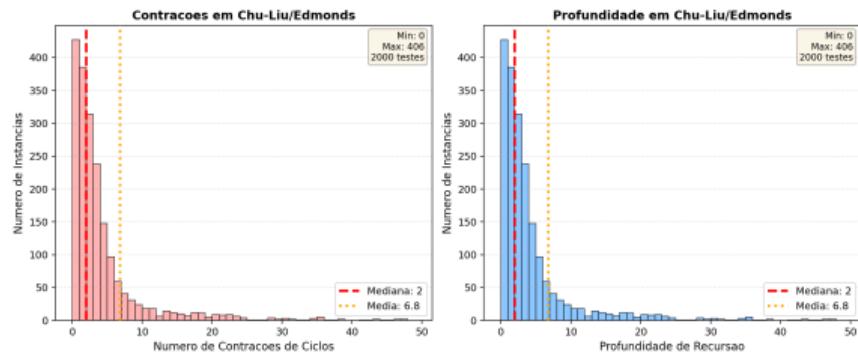
Características Estruturais

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**

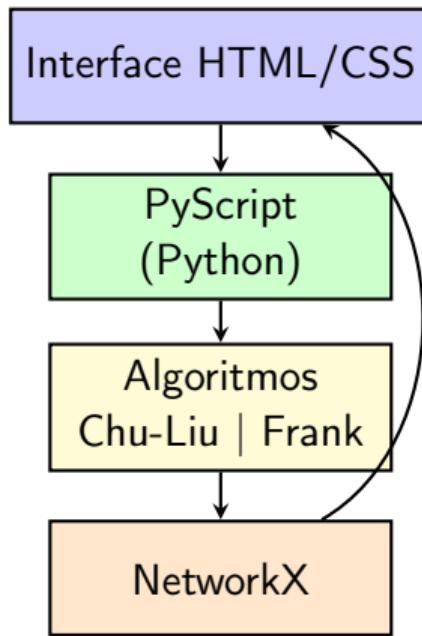
Solução Proposta:

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

Tecnologias:

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

Arquitetura da Aplicação



Interface: Página Principal

 ArboGraph

Algoritmos para o problema da arborescência geradora mínima: uma aplicação didática interativa

- Home
- Chu-Liu-Edmonds
- András Frank (V1)
- András Frank (V2)
- Desenhe um digrafo
- Nossa dissertação

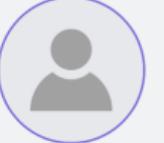
Dúvidas ?
Quer aprender mais sobre esses algoritmos, leia nossa tese :)

[Link](#)

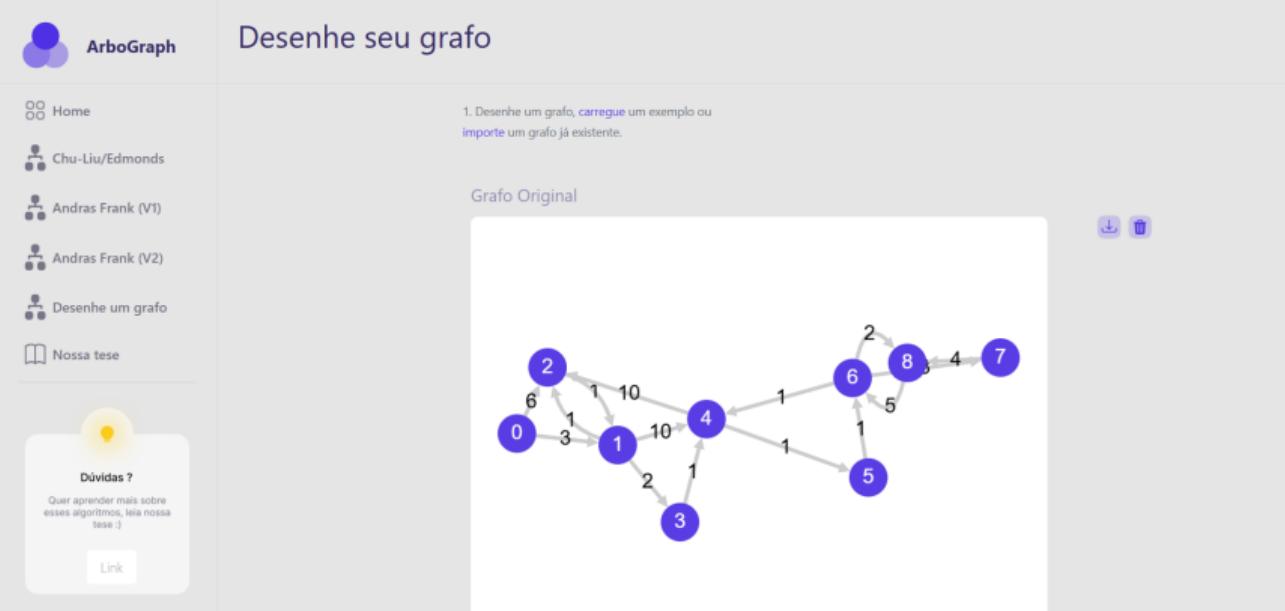
Resumo

Este trabalho investiga e implementa algoritmos de busca de uma r -arborescência geradora mínima em digrafos. A partir da formulação clássica e da literatura de Chu-Liu-Edmonds e também da formulação de András Frank, desenvolvemos uma aplicação web que permite: (i) desenhar ou importar um digrafo ponderado, (ii) escolher o nó raiz r , (iii) executar o algoritmo passo a passo com visualização das contrações, seleção de arcos de custo mínimo e reconstrução da arborescência, e (iv) exportar resultados e logs. A solução combina PyScript e NetworkX para a lógica algorítmica, Cytoscape para edição e visualização interativa, e Tailwind/Flowbite na interface. Como contribuição, o sistema oferece um ambiente didático que torna transparentes as decisões do algoritmo e facilita a análise e comparação de soluções em diferentes instâncias, apoiando ensino, experimentação e validação.

Integrantes do Projeto



Interface: Desenho de Grafos



The screenshot shows the ArboGraph web application interface. On the left, there is a sidebar with the following menu items:

- Home
- Chu-Liu/Edmonds
- Andras Frank (V1)
- Andras Frank (V2)
- Desenhe um grafo
- Nossa tese

Below the menu is a yellow button labeled "Dúvidas ?" with a tooltip: "Quer aprender mais sobre esses algoritmos, leia nossa tese :)" and a "Link" button.

The main area is titled "Desenhe seu grafo" and contains instructions: "1. Desenhe um grafo, carregue um exemplo ou importe um grafo já existente." Below this is a section titled "Grafo Original" showing a directed graph with 9 nodes (0-8) and edges with weights:

```

graph LR
    0((0)) -- 3 --> 1((1))
    0((0)) -- 6 --> 2((2))
    1((1)) -- 1 --> 0((0))
    1((1)) -- 10 --> 4((4))
    1((1)) -- 2 --> 3((3))
    2((2)) -- 1 --> 1((1))
    3((3)) -- 1 --> 4((4))
    4((4)) -- 10 --> 1((1))
    4((4)) -- 1 --> 5((5))
    4((4)) -- 1 --> 6((6))
    5((5)) -- 1 --> 4((4))
    5((5)) -- 1 --> 6((6))
    5((5)) -- 1 --> 7((7))
    6((6)) -- 2 --> 8((8))
    6((6)) -- 5 --> 8((8))
    6((6)) -- 4 --> 7((7))
    7((7)) -- 4 --> 6((6))
    8((8)) -- 2 --> 6((6))
  
```

On the right side of the graph area are two small icons: a downward arrow and a trash can.

Funcionalidades:

- Adicionar vértices e arestas
- Definir pesos

Interface: Chu-Liu-Edmonds



ArboGraph

- Home
- Chu-Liu/Edmonds
- Andras Frank (V1)
- Andras Frank (V2)
- Desenhe um grafo
- Nossa tese

Dúvidas ?

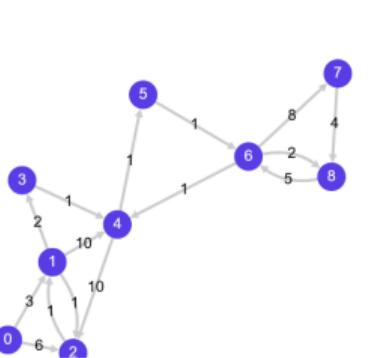
Quer aprender mais sobre esses algoritmos, leia nossa tese :)

[Link](#)

Chu-Liu / Edmonds

- 1 Crie um grafo
Desenhe um grafo, [carregue](#) um exemplo ou [importe](#) um grafo já existente.
- 2 Escolha o nó raiz
- 3 Execute o algoritmo 

Grafo Original



» Passo A Passo

Execute o algoritmo para visualizar o passo-a-passo

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

Interface: András Frank

ArboGraph

Andras Frank v1

Passo A Passo

1 Crie um grafo
Desenhe um grafo, [carregue](#) um exemplo ou [importe](#) um grafo já existente.

2 Escolha o nó raiz
node

3 Execute o algoritmo

Execute o algoritmo para visualizar o passo-a-passo

Grafo Original

Dúvidas ?
Quer aprender mais sobre esses algoritmos, leia nossa tese :)

Link

- 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



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



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× (mediana), 61× (média)
- **Comportamento prático** muito melhor que limites teóricos
 - Contrações: mediana 2 (limite $O(n)$)
 - Memória modesta: 11,5 MB



UFABC

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>