

# 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

30 de novembro de 2025

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

# O Problema



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

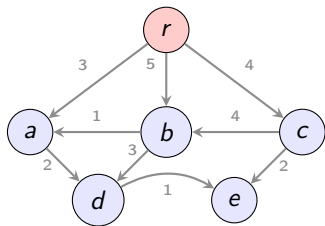
Dado um  $r$ -digrafo ponderado  $(D, w, r)$ :

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

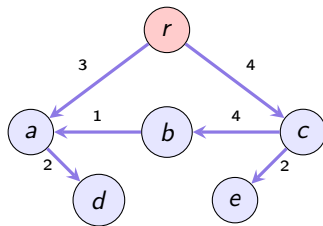
## Algoritmos estudados:

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

# Exemplo: $r$ -Arborescência Geradora Mínima



Digrafo Original

 $r$ -Arborescência Geradora

Custo: 16



Geradora Mínima

Custo: 13

# Algoritmo de Chu-Liu-Edmonds

# Chu-Liu-Edmonds



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

$\text{chu-liu-edmonds}((D, w, r))$ :

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

# Exemplo: Escolha Gulosa

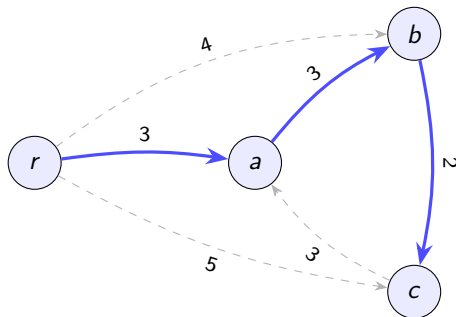
## Definição:

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 arcs 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  $v \in V \setminus \{r\}$ :

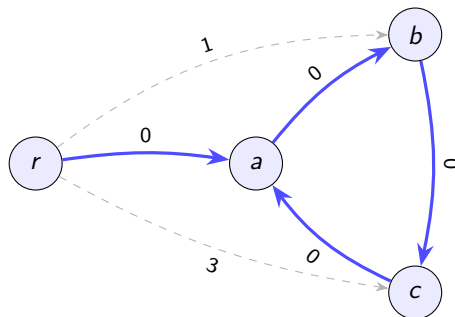
$$\lambda(v) := \min\{w(a) : a \in \delta^-(v)\}$$

Custo  $\lambda$ -reduzido:

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

## Valores de $\lambda$ :

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



Arcos do ciclo têm custo zero!

Arcos com custo zero são candidatos para  $D_0$

## Passo 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:**

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



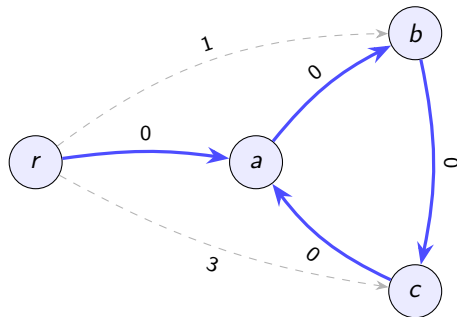
Passo 3: Verificação de  $D_0$ **Verificar:**

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 4: Contração de Ciclos

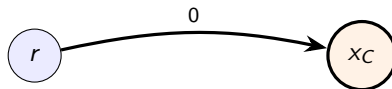
### Operação:

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  $C$



Digrafo contraído  $D'$  - *podem ter arcos saindo de  $x_C$  em  $D'$ .*

### Propriedade

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

## Passo 5: Chamada Recursiva

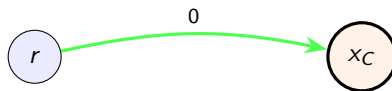


**Novo problema:**  $(D', w', r)$

**Chamada recursiva:**

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

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



$r$ -arborescência ótima em  $D'$

# 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 arcs de  $C$  exceto aquele que entra em  $v$



**Resultado:**  $T$  é uma  $r$ -arborescência de custo mínimo

$r$ -arborescência final no digrafo original

# Complexidade do Algoritmo



## Análise de Complexidade:

- Cada chamada recursiva reduz o número de vértices em pelo menos 1
- No pior caso, pode haver até  $O(n)$  chamadas recursivas
- Cada chamada envolve operações de redução de custos, construção de  $D_0$ , detecção de ciclos e contração, cada uma com complexidade  $O(m)$

## Complexidade Total:

$$O(n \cdot m)$$

onde  $n$  é o número de vértices e  $m$  é o número de arcos no digrafo original.

# Intermissão



## Algoritmo de András Frank



# Algoritmo de András Frank



## Abordagem em Duas Fases

**Fase I (Fulkerson):** Construir cobertura de  $r$ -conjuntos via redução de custos

**Fase II (Frank):** Extrair  $r$ -arborescência geradora da cobertura

### Objetivo da Fase I:

Construir uma sequência  $\sigma = ((f_i, R_i, \lambda_i))_{i \in [k]}$  tal que:

- $F := \{f_i : i \in [k]\}$  é uma **cobertura de  $r$ -conjuntos**
- A sequência  $(R_i, \lambda_i)_{i \in [k]}$  é  **$w$ -disjunta**

### Objetivo da Fase II:

Extrair  $r$ -arborescência geradora mínima usando  $F$  e a propriedade  $w$ -disjunta.

# Fase I: Conceitos Fundamentais



## Definições:

### $r$ -conjunto minimal

Um  $r$ -conjunto  $R$  é **minimal não coberto por  $F$**  se:

- $F$  não entra em  $R$  (i.e.,  $F \cap \delta^-(R) = \emptyset$ )
- Para todo  $\emptyset \subset R' \subset R$ , existe arco de  $F$  que entra em  $R'$

### Sequência $w$ -disjunta

Uma sequência  $((R_i, \lambda_i))_{i \in [k]}$  é  **$w$ -disjunta** se:

$$\sum_{i \in [k]} \lambda_i [a \in \delta^-(R_i)] \leq w(a) \quad \text{para cada } a \in A(D)$$

A soma dos  $\lambda_i$  sobre os conjuntos que  $a$  entra não excede o peso de  $a$ .

# Fase I: Algoritmo de Fulkerson



## Processo iterativo

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

### 1 Inicializar:

- $c := w$  (custos correntes)
- $\sigma := \epsilon$  (sequência vazia)
- $F := \emptyset$  (conjunto de arcos selecionados)

### 2 Enquanto existir fonte $R_k$ em $\mathcal{C}(D_0)$ com $r \notin R_k$ :

- Calcular  $\lambda_k := \min\{c(a) : a \in \delta^-(R_k)\}$
- Selecionar  $f_k \in \delta^-(R_k)$  com  $c(f_k) = \lambda_k$
- $\sigma := \sigma \cdot (f_k, R_k, \lambda_k)$
- $F := F \cup \{f_k\}$
- $c := c - \lambda_k 1_{\delta^-(R_k)}$  (reduzir custos)
- $D_0 := (V, F)$  (atualizar digrafo auxiliar)

### 3 Devolver: $\sigma$

# Fase I: Encontrando $r$ -conjuntos Minimais



**Como encontrar um  $r$ -conjunto minimal não coberto?**

Seja  $D_0 := (V, F)$  onde  $F = \{f_i : i \in [k]\}$ .

- 1 Calcular a condensação  $\mathcal{C}(D_0)$
- 2 Identificar componentes fortemente conexas (CFCs)
- 3 Encontrar uma fonte  $S$  em  $\mathcal{C}(D_0)$  tal que  $r \notin S$

## Proposição

Toda fonte  $S$  de  $\mathcal{C}(D_0)$  com  $r \notin S$  é um  $r$ -conjunto minimal não coberto por  $F$ .

**Complexidade:** identificação de CFCs em  $O(|A|)$  usando Kosaraju.

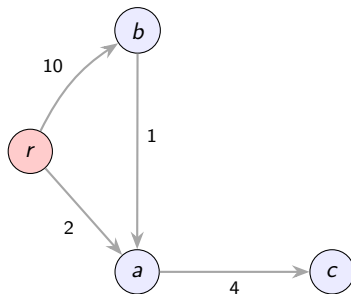
# Exemplo: Digrafo Inicial

**Digrafo ponderado  $(D, w, r)$ :**

- Vértices:  $\{r, a, b, c\}$
- Raiz:  $r$

**Pesos dos arcos:**

- $(r, a) : 2$ ,  $(r, b) : 10$
- $(b, a) : 1$ ,  $(a, c) : 4$



## Problema

Aplicar o algoritmo de András Frank para encontrar a  $r$ -arborescência geradora de custo mínimo.

# Fase I: Iteração 1 - Encontrar $r$ -conjunto Minimal

## Estado inicial:

- $F = \emptyset$  (nenhum arco selecionado)
- $D_0 = (V, \emptyset)$
- $\mathcal{C}(D_0)$  tem 4 fontes

## Fontes que são $r$ -conjuntos:

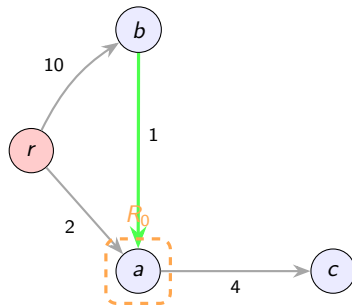
- $\{a\}, \{b\}, \{c\}$

## Escolha:

$R_0 = \{a\}$  (minimal)

## Arcos que entram em $\{a\}$ :

- $(r, a) : 2$
- $(b, a) : 1 \leftarrow$  **mínimo**



$$\lambda_0 = 1, f_0 = (b, a)$$

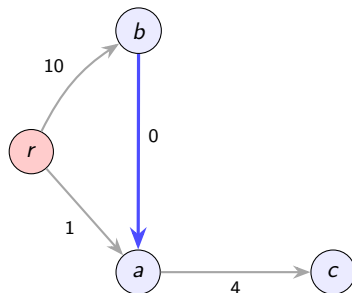
## Fase I: Iteração 2 - Redução de Custos

## Atualização:

- $\sigma = [(f_0, R_0, \lambda_0)]$
- $F = \{(b, a)\}$
- $D_0 = (V, \{(b, a)\})$

## Redução de custos:

- $c(r, a) = 2 - 1 = 1$
- $c(b, a) = 1 - 1 = 0 \checkmark$



Arcos azuis têm custo zero

## Observação

Custos são reduzidos para garantir que  $\sigma$  seja  $w$ -disjunta.

## Fase I: Iteração 3 - Redução de Custos

## Atualização:

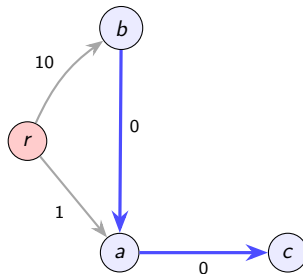
- $F = \{(b, a), (a, c)\}$
- $D_0 = (V, F)$

## Redução:

- $c(a, c) = 4 - 4 = 0 \checkmark$

$\mathcal{C}(D_0)$  com  $F = \{(b, a), (a, c)\}$ :

- CFCs:  $\{r\}, \{b\}, \{a\}, \{c\}$
- Fontes em  $D_0$ :  $\{r\}, \{b\}$



Arcos com custo zero em azul



## Fase I: Iteração 4 - Redução de Custos

**Estado:**  $F = \{(b, a), (a, c)\}$

$\mathcal{C}(D_0)$  ainda tem:

- Fonte  $\{b\}$  não contém  $r$

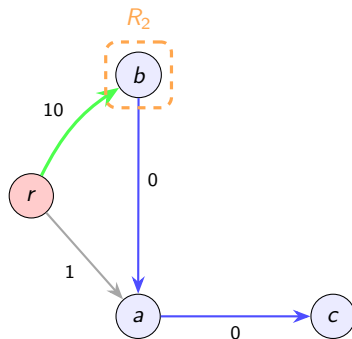
**Escolha:**  $R_2 = \{b\}$

Arcos que entram em  $\{b\}$ :

- $(r, b) : 10 \leftarrow$  único arco

**Seleção:**

- $\lambda_2 = 10$
- $f_2 = (r, b)$



# Fase I: Iteração 5 - Estado Final

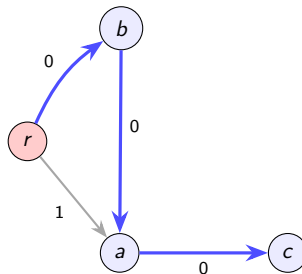
## Atualização:

- $F = \{(b, a), (a, c), (r, b)\}$
- $D_0 = (V, F)$

## Redução:

- $c(r, b) = 10 - 10 = 0$

$\mathcal{C}(D_0)$  com  $F = \{(b, a), (a, c), (r, b)\}$ :



**Condição de parada satisfeita!**

Todos arcos de  $F$  têm custo zero!

## Sequência devolvida

$\sigma = [(f_0 = (b, a), R_0 = \{a\}, \lambda_0 = 1), (f_1 = (a, c), R_1 = \{c\}, \lambda_1 = 4), (f_2 = (r, b), R_2 = \{b\}, \lambda_2 = 10)]$

# Intermissão



## Fase II: Duas Abordagens

# Fase II: Construção da Arborescência



**Entrada:** Sequência  $(f_i)_{i \in [k]}$  da Fase I

**Objetivo:** Extrair  $J \subseteq \{f_i : i \in [k]\}$  que é uma  $r$ -arborescência geradora

**Algoritmo guloso:**

- 1 Iniciar com  $U := \{r\}$  e  $J := \emptyset$
- 2 Para  $t = 1$  até  $|V| - 1$ :
  - Para cada  $f_i = (u_i, v_i)$  na sequência:
  - Se  $u_i \in U$  e  $v_i \notin U$ :
    - $U := U \cup \{v_i\}$
    - $J := J \cup \{f_i\}$
    - Passar para próxima iteração

## Invariante

Em cada iteração,  $\varrho_J(R_i) \leq 1$  para todo  $i \in [k]$

# Fase II: Exemplo - Extração da Arborescência

## Entrada da Fase II:

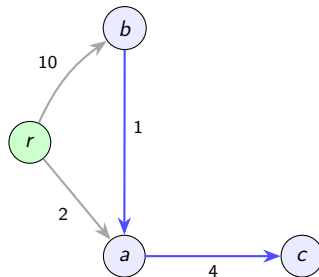
- $F = [f_0, f_1, f_2] = [(b, a), (a, c), (r, b)]$

## Estado inicial:

- $U = \{r\}$  (alcançados)
- $J = \emptyset$  (arborescência)

## Objetivo:

- Selecionar  $|V| - 1 = 3$  arcos
- Manter propriedade de arborescência



Verde = vértices em  $U$

## Observação

Fase II precisa considerar **todos** arcos de  $D$ , não apenas  $F$ .

## Fase II: Iteração 1

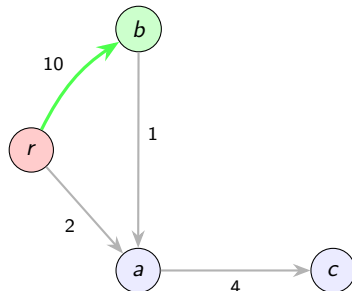
Estado:  $U = \{r\}$ ,  $J = \emptyset$

Procurar em  $F$ :

- $f_0 = (b, a)$ :  $b \notin U \times$
- $f_1 = (a, c)$ :  $a \notin U \times$
- $f_2 = (r, b)$ :  $r \in U$ ,  $b \notin U \checkmark$

Ação:

- Adicionar  $(r, b)$  a  $J$
- $U := U \cup \{b\} = \{r, b\}$



$J = \{(r, b)\}$ , custo = 10

## Fase II: Iteração 2

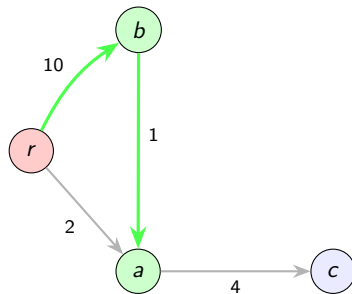
**Estado:**  $U = \{r, b\}$ ,  $J = \{(r, b)\}$

**Procurar em  $F$ :**

- $f_0 = (b, a)$ :  $b \in U$ ,  $a \notin U$  ✓

**Ação:**

- Adicionar  $(b, a)$  a  $J$
- $U := U \cup \{a\} = \{r, b, a\}$



$J = \{(r, b), (b, a)\}$ , custo = 11

## Fase II: Iteração 3 - Estado Final

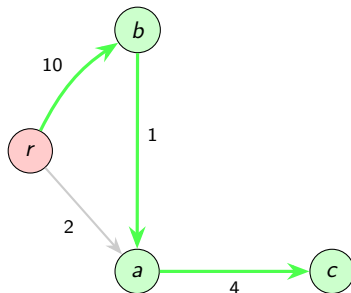
**Estado:**  $U = \{r, b, a\}$ ,  $|J| = 2$

**Procurar em  $F$ :**

- $f_1 = (a, c)$ :  $a \in U$ ,  $c \notin U$  ✓

**Ação:**

- Adicionar  $(a, c)$  a  $J$
- $U := U \cup \{c\} = V$
- $|J| = 3 = |V| - 1$  ✓



$J = r$ -arborescência geradora!

## Resultado Final

$J = \{(r, b), (b, a), (a, c)\}$  com custo  $w(J) = 10 + 1 + 4 = 15$  (custo mínimo!)



# Intermissão



## Chu-Liu-Edmonds vs András Frank

# 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)  | <b>0,016 s</b> | <b>0,020 s</b> |

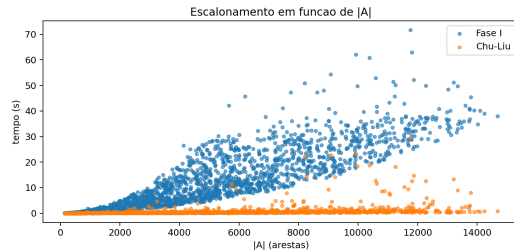
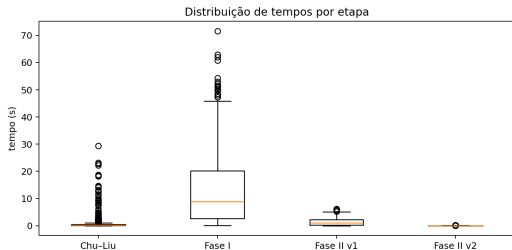
## Speedup Fase II

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

# Escalonamento e Consumo de Memória

## Escalonamento temporal:

- Tempo cresce linearmente com o número de arestas
- Fase I de Frank domina o tempo total
- Fase II (heap) é residual e muito rápida



# 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

# Conclusões dos Experimentos



- Equivalência teórica e prática dos algoritmos confirmada
- Chu-Liu/Edmonds é mais eficiente
- Fase I de Frank é o gargalo computacional
- Heap na Fase II traz ganhos práticos expressivos
- Algoritmos são escaláveis e viáveis para grandes digrafos

# Intermissão



## Didática do Abstrato

# Fundamentos Cognitivos e Didáticos



## Desafios do Ensino de Matemática Abstrata

- Conhecimento abstrato exige transitar entre registros: intuitivo, visual, simbólico e formal.
- **Carga cognitiva:**
  - Intrínseca: complexidade dos conceitos e pré-requisitos.
  - Extrínseca: forma de apresentação e coordenação entre texto, fórmulas e figuras.
  - Pertinente: esforço dedicado à organização dos esquemas mentais.
- Combinar representações verbais e visuais reduz sobrecarga e favorece integração semântica.

# Desafios na Ensino de Algoritmos de Grafos



## Três Eixos de Dificuldade

- ❶ **Decisões locais vs. coerência global:** Escolhas localmente ótimas podem gerar ciclos, dificultando a compreensão da solução global.
- ❷ **Contração e expansão:** Transitar entre grafo original, condensado e reexpansão exige rastreabilidade e clareza sobre o que muda e o que permanece.
- ❸ **Relação com a teoria primal-dual:** Dificuldade em conectar ações operacionais do algoritmo com fundamentos teóricos e certificados de otimalidade.

**Solução:** Visualização e interação bem projetadas facilitam a integração entre prática e teoria.



# O Ecossistema de Ferramentas para Ensino de Grafos



## Categorias de Ferramentas Digitais

- **Diagramas programáveis:** Visualização estável e integrada ao texto matemático (*Graphviz*, *TikZ*).
- **Exploração e edição de grafos:** Manipulação gráfica e análise estrutural (*Gephi*, *yEd*, *Cytoscape*).
- **Visualização de algoritmos:** Animações e explicações dinâmicas (*VisuAlgo*).
- **Ambientes programáveis:** Integração de código, texto e visualização para exemplos reprodutíveis (*Jupyter*, *NetworkX*).

Nenhuma ferramenta cobre todos os aspectos didáticos de forma integrada. A aplicação proposta busca preencher essa lacuna.

# Objetivos da Ferramenta Didática



- Facilitar a compreensão dos algoritmos Chu-Liu-Edmonds e András Frank
- Permitir aos usuários interagir com grafos e observar o funcionamento dos algoritmos
- Fornecer feedback imediato sobre as operações realizadas
- Ser acessível via navegador web, sem necessidade de instalação

# Intermissão



## Aplicação Web

# Aplicação web



Ferramenta web interativa para ensino de algoritmos de arborescências dirigidas, permitindo visualização passo a passo, edição livre de grafos e exportação de resultados, com arquitetura modular e foco didático.

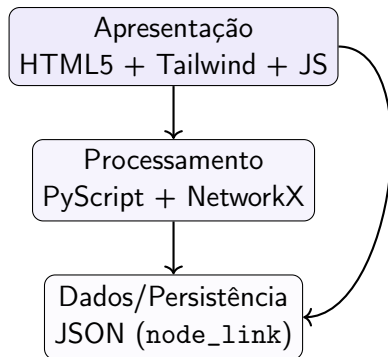
## Módulos principais:

- **Visualização Algorítmica:** Páginas para execução passo a passo dos algoritmos Chu-Liu-Edmonds e András Frank.
- **Modelagem Livre:** Editor sandbox para desenhar, testar e exportar grafos arbitrários.
- **Disseminação Científica:** Página informativa sobre o projeto e a dissertação.

# Arquitetura da Aplicação

## Estrutura em três camadas:

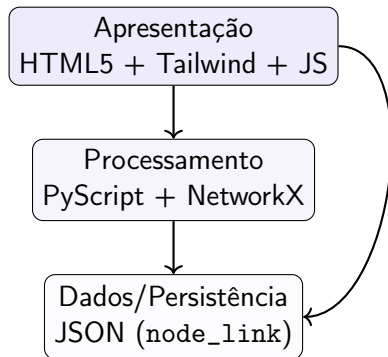
- **Apresentação:** Interface construída em HTML5, estilizada com Tailwind CSS e dinamizada por JavaScript.
- **Processamento (PyScript):** Executa algoritmos em Python (NetworkX) diretamente no navegador, gerando visualizações estáticas com Matplotlib.
- **Dados e Persistência:** Utiliza JSON (node\_link) para serializar grafos, armazenar pesos e permitir exportação/importação entre módulos.



# Arquitetura da Aplicação

## Benefícios:

- Processamento local e rápido
- Facilidade de uso e reprodutibilidade
- Modularidade e extensibilidade



# Princípios de IHC Aplicados



## Fundamentos para o Design da Ferramenta

O desenvolvimento da aplicação web foi guiado por oito princípios de Interação Humano-Computador (IHC), integrando heurísticas de usabilidade e teorias de aprendizagem:

- **Usabilidade:** Interface limpa, controles claros e navegação intuitiva.
- **Eficiência cognitiva:** Redução da carga mental, destaque para informações relevantes.
- **Feedback imediato:** Atualização visual e textual em tempo real a cada ação.
- **Engajamento ativo:** Usuário explora, manipula e prediz resultados.
- **Visão geral com detalhe sob demanda**
- **Consistência semântica:** Terminologia e estilos padronizados em toda a interface.
- **Múltiplos registros de representação:** Grafo visual, log textual e parâmetros simbólicos.
- **Prevenção e recuperação de erros**

# Intermissão



## Conclusões



# 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

# Trabalhos Futuros



## Extensões Possíveis

- 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>