Estrutura de Dados (CCA410)

Aula 12 – Árvores Geradoras Mínimas (Kruskal, Prim)

Prof. Luciano Rossi
Prof. Leonardo Anjoletto Ferreira
Prof. Flavio Tonidandel
Prof. Fabio Suim

Ciência da Computação
Centro Universitário FEI

2º Semestre de 2025

Grafos

Classificação das Estruturas de Dados

Primitivas	Compostas		
	Simples	Lineares	Não-Lineares
Inteiro	Cadeia (string)	Pilha	Árvore
Real	Registro (struct)	Fila	**Grafo**
Lógico	Arranjo (array)	Lista	
Caractere			

centro universitário FEI

Árvores Geradoras Mínimas

centro universitário FEI

Árvores Geradoras Mínimas

Definição

Problema da Árvore Geradora Mínima (MST)

Dado um grafo conexo e ponderado $G = (V, E)$ com função de peso $w : E \to \mathbb{R}$, encontrar um subconjunto de arestas $T \subseteq E$ que:

- Conecta todos os vértices (é uma árvore geradora)
- Minimiza o peso total: $w(T) = \sum_{e \in T} w(e)$

centro universitário FEI

Árvores Geradoras Mínimas

Propriedades

Propriedades Fundamentais

- Uma MST tem exatamente $|V| - 1$ arestas
- É acíclica (não contém ciclos)
- Remove qualquer aresta $\Rightarrow$ grafo desconexo
- Adiciona qualquer aresta $\Rightarrow$ cria um ciclo
- Pode não ser única (se existem pesos iguais)

centro universitário FEI

Árvores Geradoras Mínimas

Aplicações

Aplicações Práticas

- Projeto de redes de telecomunicações
- Distribuição de energia elétrica
- Redes de abastecimento de água
- Design de circuitos integrados (VLSI)
- Clustering e análise de dados
- Redes de transporte e logística

centro universitário FEI

Árvores Geradoras Mínimas

Algoritmos Clássicos

Principais Algoritmos

- **Algoritmo de Prim (1957)**
  - Cresce a árvore a partir de um vértice inicial
  - Complexidade: $O(E \log V)$ com heap binário
- **Algoritmo de Kruskal (1956)**
  - Ordena arestas e adiciona se não formar ciclo
  - Complexidade: $O(E \log E)$ ou $O(E \log V)$
- **Algoritmo de Boruvka (1926)**
  - Mais antigo, útil em grafos paralelos

centro universitário FEI

Teorema do Corte

Seja $(S, V - S)$ um corte qualquer de G e seja e uma aresta de peso mínimo que cruza o corte. Se A é um subconjunto de alguma MST que respeita o corte, então $A \cup \{e\}$ também é subconjunto de alguma MST.

Implicação

Este teorema garante a correção dos algoritmos gulosos (Prim e Kruskal) para encontrar MSTs.

Árvores Geradoras Mínimas

Comparação dos Algoritmos

Aspecto	**Prim**	**Kruskal**
Estrutura de dados	Fila de prioridade	Union-Find
Estratégia	Cresce uma árvore	Adiciona arestas
Melhor para	Grafos densos	Grafos esparsos
Complexidade	$O(E \log V)$	$O(E \log V)$
Paralelização	Difícil	Mais fácil

Algoritmo de Prim

Algoritmo de Prim

Características

Algoritmo de Prim

- **Vantagem:** Eficiente para grafos densos através de fila de prioridade
- **Restrição:** Grafo deve ser conexo e não-direcionado
- **Estratégia:** Algoritmo guloso que sempre escolhe a aresta de menor peso que conecta a árvore a um novo vértice

Algoritmo de Prim

Atributos dos Vértices

Atributos

- $v.chave$: Peso mínimo de qualquer aresta conectando v à árvore
- $v.pai$: Predecessor de v na árvore geradora mínima
- **Estado em** Q: Vértices ainda não incluídos na MST

centro universitário FEI

Algoritmo em Pseudocódigo

centro universitário FEI

Algoritmo de Prim

Pseudocódigo

MST-PRIM(G, w, r)

1 para cada vértice $u \in G.V$ faça
2 $\quad u.chave = \infty$
3 $\quad u.pai = NIL$
4 $r.chave = 0$; $\quad Q = G.V$
5 enquanto $Q \neq \emptyset$ faça
6 $\quad u =$ EXTRAI-MÍNIMO(Q)
7 $\quad$ para cada $v \in G.Adj[u]$ faça
8 $\quad\quad$ se v está em Q e $w(u, v) < v.chave$
9 $\quad\quad\quad$ então $v.chave = w(u, v)$; $\quad v.pai = u$

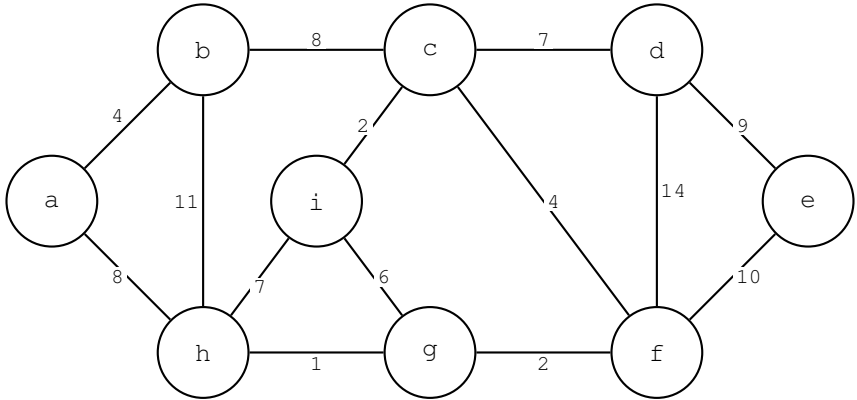
Entrada: Grafo G conexo, função peso w, vértice raiz r
Saída: Árvore geradora mínima (MST) representada pelos atributos pai

centro universitário FEI

Simulação

Algoritmo de Prim

Exemplo



Algoritmo de Prim

Propriedades e Limitações

Propriedades

+ Sempre encontra MST ótima
+ Eficiente para grafos densos
+ Cresce árvore de forma incremental
+ Fácil de implementar com heap
+ Funciona bem com representação de matriz

Limitações

- Requer grafo conexo
- Não funciona para grafos direcionados
- Requer fila de prioridade eficiente
- Menos eficiente para grafos esparsos
- Difícil de paralelizar

centro universitário FEI

Algoritmo de Prim

Complexidade

Análise de Complexidade

- **Com Array Simples:** $O(V^2)$
  - Melhor para grafos densos
- **Com Heap Binário:** $O(E \log V)$
  - Balanceado para grafos gerais
- **Com Heap de Fibonacci:** $O(E + V \log V)$
  - Melhor complexidade teórica

centro universitário FEI

Algoritmo de Prim

Considerações Finais

Quando Usar Prim

Use Prim quando tiver um **grafo conexo não-direcionado** e precisar de uma árvore geradora mínima. É a escolha ideal para **grafos densos** e quando a representação por matriz de adjacência é conveniente. Aplicável em redes de telecomunicações, distribuição elétrica e projetos de circuitos.

Algoritmo de Prim

Implementação e Otimizações

Estruturas de Dados

- **Fila de Prioridade:** Heap binário (min-heap)
- **Representação:** Lista de adjacência para grafos esparsos
- **Array de Chaves:** Para armazenar $chave[v]$
- **Array de Predecessores:** Para reconstruir a MST

Comparação com Kruskal

- **Prim:** Melhor para grafos densos
- **Kruskal:** Melhor para grafos esparsos
- **Prim:** Cresce uma árvore
- **Kruskal:** Une componentes (floresta)

Ambos garantem solução ótima pelo Teorema do Corte

centro universitário FEI

Algoritmo de Kruskal

Árvore Geradora Mínima (MST)

Algoritmo Genérico

Generic-MST(G,w)

```
1 A = ∅
2 enquanto A não é uma MST
3     encontre uma aresta uv que seja "boa" para A
4     A = A ∪ {uv}
5 devolva A
```

Árvore Geradora Mínima (MST)

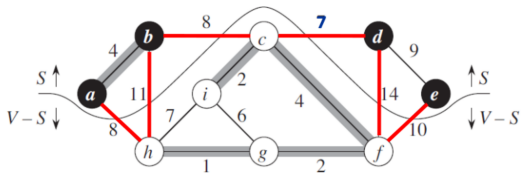
Teorema do Corte

Teorema

- G ponderado e conexo
- A é um subconjunto de uma MST
- Um corte $(S, V - S)$ que respeita A
- Se uv é uma aresta "leve"então uv é uma aresta "boa"para A

Corolário

- C é um componente de $G_A = (V, A)$
- Se uv tem peso mínimo e conecta C e C' (dois componentes distintos) então uv é "boa" para A



Árvore Geradora Mínima (MST)

Invariante dos Algoritmos

Invariante

Em ambos algoritmos, A é um subconjunto de alguma MST

- **Prim:** A é uma árvore
- **Kruskal:** A é uma floresta

Algoritmo de Kruskal

Estratégia Gulosa

Estratégia "Gulosa"

- Escolhe a aresta de menor peso (a mais "barata")

Diferenças entre Prim e Kruskal

Prim:

- Conecta árvore A com algum vértice não visitado

Kruskal:

- Conecta dois componentes distintos da floresta

centro universitário FEI

Algoritmo de Kruskal

Ideia Principal

Funcionamento

- **Inicialmente:** cada vértice = conjunto disjunto
- **A cada passo:** união de dois conjuntos disjuntos
- **Até que:** todos os vértices estejam em um mesmo conjunto

centro universitário FEI

Estrutura de Dados Union-Find

Algoritmo de Kruskal

Conjuntos Disjuntos (Disjoint Sets)

Estrutura de Dados

- Coleção $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ de conjuntos disjuntos
- Cada conjunto é identificado por um **representante**

Operações

- **MAKE-SET(v):** Cria novo conjunto com único elemento v
- **FIND-SET(v):** Devolve representante do conjunto que contém v
- **UNION(u, v):** Une os conjuntos que contêm u e v

centro universitário FEI

Estrutura Union-Find

Operações Fundamentais - Pseudocódigo

MAKE-SET(v)

```
1  v.pai = v
2  v.rank = 0
```

FIND-SET(v)

```
1  se v ≠ v.pai
2     então v.pai = FIND-SET(v.pai)
3  devolva v.pai
```

UNION(u, v)

```
1  raiz_u = FIND-SET(u)
2  raiz_v = FIND-SET(v)
3  se raiz_u = raiz_v
4     retorna // já estão no mesmo conjunto
5  se raiz_u.rank < raiz_v.rank
6     raiz_u.pai = raiz_v
7  senão se raiz_u.rank > raiz_v.rank
8     raiz_v.pai = raiz_u
9  senão
10    raiz_v.pai = raiz_u
11    raiz_u.rank = raiz_u.rank + 1
```

centro universitário FEI

Estrutura Union-Find

Otimizações

Path Compression (Compressão de Caminho)

- Implementada no FIND-SET (linha 2)
- Durante a busca, faz cada nó apontar diretamente para a raiz
- Melhora drasticamente operações futuras
- Complexidade amortizada: $O(\alpha(n))$

Union by Rank (União por Rank)

- Implementada no UNION (linhas 5-11)
- Sempre anexa árvore de menor rank à de maior rank
- Mantém árvores balanceadas
- Evita degeneração em listas lineares

centro universitário FEI

Estrutura Union-Find

Atributos e Complexidade

Atributos

- $v.pai$: Aponta para o pai de v na estrutura
  - Se $v.pai = v$, então v é raiz
- $v.rank$: Limite superior da altura da subárvore de v
  - Usado na heurística union by rank

Complexidade

- **MAKE-SET:** $O(1)$
- **FIND-SET:** $O(\alpha(n))\star$
- **UNION:** $O(\alpha(n))\star$

$\star$Com path compression e union by rank
$\alpha(n)$ é a função inversa de Ackermann
$\alpha(n) \leq 4$ para qualquer n prático
Pode ser considerada constante

centro universitário FEI

Estrutura Union-Find

Exemplo de Execução

Considere as seguintes operações sobre conjuntos $\{1, 2, 3, 4, 5\}$:

1. **Inicialização:** MAKE-SET para cada elemento
   - Cada elemento é seu próprio pai e rank = 0
   - Conjuntos: $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$
2. **UNION(1, 2):** Une conjuntos contendo 1 e 2
   - Mesmos ranks: 2 vira pai de 1, rank de 2 aumenta
   - Conjuntos: $\{1, 2\}, \{3\}, \{4\}, \{5\}$
3. **UNION(3, 4):** Une conjuntos contendo 3 e 4
   - Conjuntos: $\{1, 2\}, \{3, 4\}, \{5\}$
4. **UNION(2, 3):** Une conjuntos contendo 2 e 3
   - Raiz de 2 tem rank maior: 4 aponta para 2
   - Conjuntos: $\{1, 2, 3, 4\}, \{5\}$
5. **FIND-SET(1):** Retorna 2 (representante do conjunto)
   - Com path compression: 1 passa a apontar diretamente para 2

centro universitário FEI

Algoritmo de Kruskal

Detecção de Ciclos

Como Evitar Ciclos

- Uma aresta uv forma ciclo se u e v já estão no mesmo componente
- **Teste:** FIND-SET(u) $\neq$ FIND-SET(v)
- Se os representantes são diferentes, não há ciclo
- Então podemos adicionar uv à MST e fazer UNION(u, v)

centro universitário FEI

Pseudocódigo

Algoritmo de Kruskal

Pseudocódigo Completo

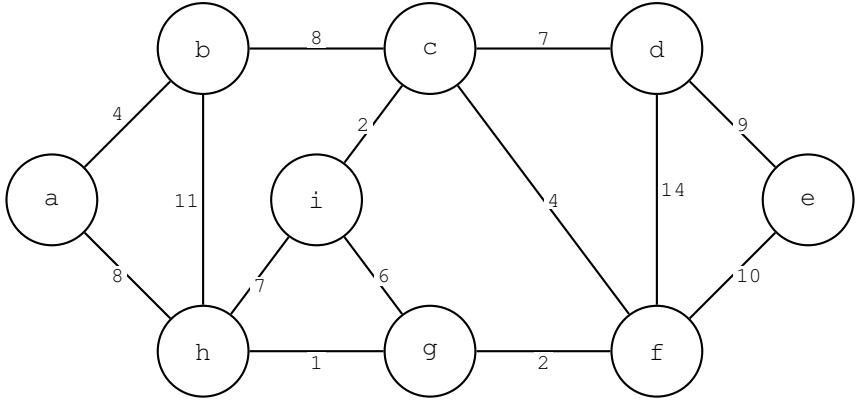
MST-KRUSKAL(G, w**)**

```
1 A = ∅
2 para cada vértice v ∈ G.V faça
3     MAKE-SET(v)
4 ordenar arestas G.E por peso (crescente)
5 para cada aresta uv ∈ G.E (ordenada) faça
6     se FIND-SET(u) ≠ FIND-SET(v)
7        então A = A ∪ {uv}
8           UNION(u, v)
9 devolva A
```

Simulação

Algoritmo de Kruskal

Exemplo - Grafo Inicial



Arestas ordenadas por peso: (g,h):1, (c,i):2, (f,g):2, (a,b):4, (c,f):4, (i,g):6, (c,d):7, (h,i):7, (a,h):8, (b,c):8, (d,e):9, (e,f):10, (b,h):11, (d,f):14

Algoritmo de Kruskal

Execução Passo a Passo

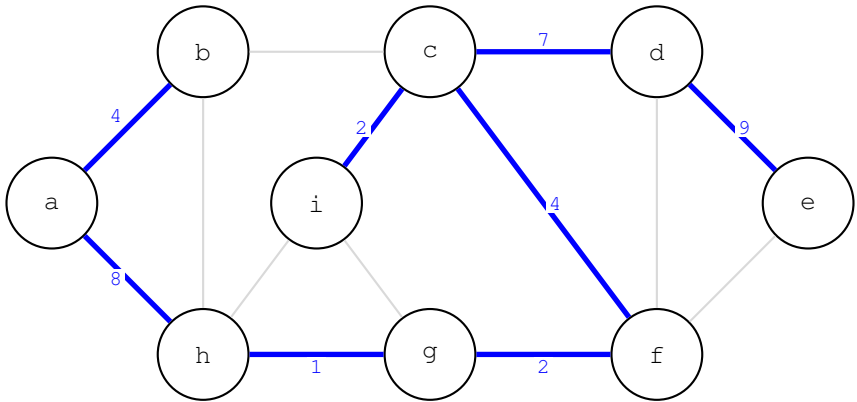
Aresta	Peso	Ação
(g,h)	1	Adiciona – componentes diferentes
(c,i)	2	Adiciona – componentes diferentes
(f,g)	2	Adiciona – componentes diferentes
(a,b)	4	Adiciona – componentes diferentes
(c,f)	4	Adiciona – componentes diferentes
(i,g)	6	Rejeita – forma ciclo
(c,d)	7	Adiciona – componentes diferentes
(h,i)	7	Rejeita – forma ciclo
(a,h)	8	Adiciona – componentes diferentes
(b,c)	8	Rejeita – forma ciclo
(d,e)	9	Adiciona – componentes diferentes

Resultado: MST com 8 arestas e peso total = 37

centro universitário FEI

Algoritmo de Kruskal

Resultado Final



Peso Total da MST: 1 + 2 + 2 + 4 + 4 + 7 + 8 + 9 = 37

Análise de Complexidade

- **Inicialização (linhas 1-3):** $O(V)$
  - V operações MAKE-SET
- **Ordenação das Arestas (linha 4):** $O(E \log E)$
  - Domina a complexidade total
- **Processamento das Arestas (linhas 5-8):** $O(E \cdot \alpha(V))$
  - E operações FIND-SET e até $V-1$ operações UNION
  - $\alpha(V)$ é a função inversa de Ackermann (praticamente constante)
- **Complexidade Total:** $O(E \log E)$ ou equivalentemente $O(E \log V)$
  - Como $E \leq V^2$, temos $\log E = O(\log V)$

Algoritmo de Kruskal

Propriedades e Limitações

Propriedades

+ Sempre encontra MST ótima
+ Eficiente para grafos esparsos
+ Trabalha com arestas (não vértices)
+ Fácil de paralelizar
+ Não requer vértice inicial
+ Detecta componentes conexas

Limitações

- Requer ordenação inicial das arestas
- Necessita estrutura Union-Find
- Menos eficiente para grafos densos
- Não funciona para grafos direcionados
- Complexidade de ordenação domina

Quando Usar Kruskal

Use Kruskal quando tiver um **grafo esparso** (poucas arestas) e precisar de uma árvore geradora mínima. É ideal quando as arestas podem ser ordenadas eficientemente ou quando trabalha com **lista de arestas.** Excelente para problemas de clusterização e detecção de componentes conexas.

Algoritmo de Kruskal

Implementação e Comparações

Estruturas de Dados

- **Union-Find:** Com path compression e union by rank
- **Representação:** Lista de arestas (ideal)
- **Algoritmo de Ordenação:** Merge sort ou heap sort
- **Conjunto A:** Lista de arestas da MST

Kruskal vs Prim

- **Kruskal:** Melhor para grafos esparsos
- **Prim:** Melhor para grafos densos
- **Kruskal:** Processa arestas
- **Prim:** Cresce de um vértice
- **Kruskal:** Mais fácil de paralelizar
- **Prim:** Melhor com matriz de adjacência

Ambos garantem solução ótima pelo Teorema do Corte

centro universitário FEI

Estrutura de Dados (CCA410)

Aula 12 – Árvores Geradoras Mínimas (Kruskal, Prim)

Prof. Luciano Rossi
Prof. Leonardo Anjoletto Ferreira
Prof. Flavio Tonidandel
Prof. Fabio Suim

Ciência da Computação
Centro Universitário FEI

2º Semestre de 2025