AED2 - Algoritmos e Estr. de Dados II

Aula 18: Árvore Geradora Mínima (MST)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Fevereiro de 2026

Sumário

1. Motivação
2. O que é uma AGM?
3. Algoritmo de Kruskal
4. Algoritmo de Prim
5. Aplicações e Comparação

Problema 1: Rede de Fibra Óptica

Enunciado

Uma operadora precisa conectar **12 cidades** de Minas Gerais com fibra óptica. Cada trecho entre duas cidades tem um **custo diferente** de instalação.

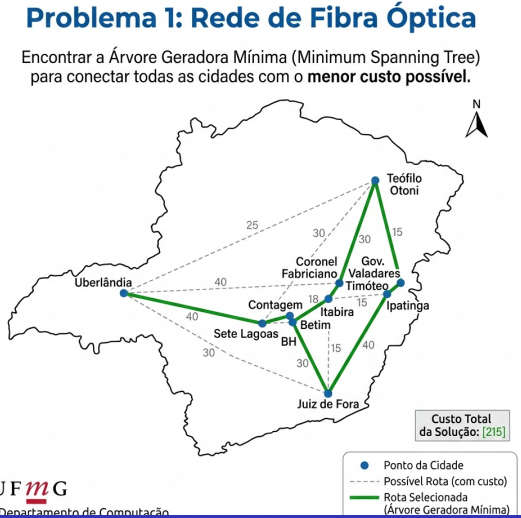
Enunciado

Uma operadora precisa conectar **12 cidades** de Minas Gerais com fibra óptica. Cada trecho entre duas cidades tem um **custo diferente** de instalação.

A Pergunta

Qual conjunto de trechos conecta **todas** as cidades com o **menor custo total**?

Restrição: não podemos ter ciclos – seria desperdício de fibra!



Problema 2: Rede de Irrigação

Enunciado

Uma fazenda tem **8 setores** de plantio e um **reservatório** de água. Cada par de setores pode ser conectado por tubulação, com custos variados.

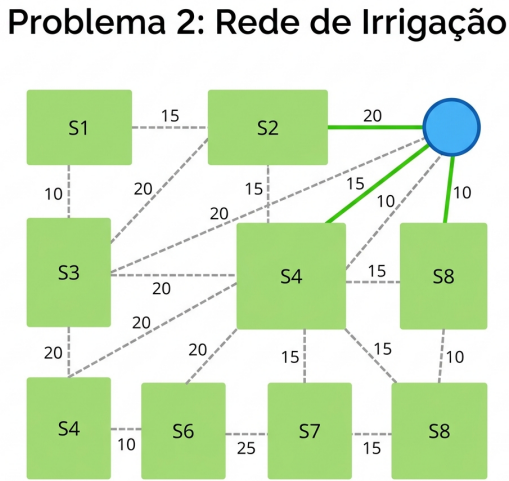
Enunciado

Uma fazenda tem **8 setores** de plantio e um **reservatório** de água. Cada par de setores pode ser conectado por tubulação, com custos variados.

A Pergunta

Como montar a rede de tubos que gaste o **menor material possível** e atinja **todos** os setores a partir do reservatório?

Aqui temos um ponto de partida natural: o reservatório!



Problema 3: Circuito Impresso (PCB)

Enunciado

Um chip tem **20 pinos** que precisam ser interligados por trilhas de cobre numa placa. Cada trilha tem um comprimento proporcional ao cobre gasto.

Problema 3: Circuito Impresso (PCB)

Enunciado

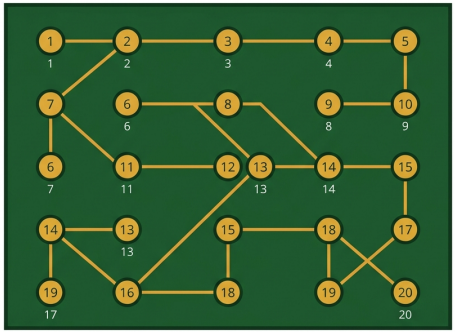
Um chip tem **20 pinos** que precisam ser interligados por trilhas de cobre numa placa. Cada trilha tem um comprimento proporcional ao cobre gasto.

A Pergunta

Como minimizar o **cobre total** usado sem deixar nenhum pino desconectado?

Grafo denso: pinos próximos, muitas conexões possíveis.

Problema 3: Circuito Impresso (PCB)



Problema 4: Clustering de Dados

Enunciado

Você tem **1000 pontos** no espaço e quer agrupá-los em **5 clusters** de forma que pontos próximos fiquem juntos.

Enunciado

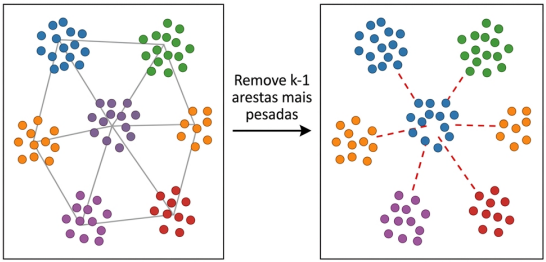
Você tem **1000 pontos** no espaço e quer agrupá-los em **5 clusters** de forma que pontos próximos fiquem juntos.

A Pergunta

Como a estrutura de "árvore de custo mínimo" pode ajudar a **segmentar** os dados?

Dica: remova as $k-1$ arestas mais pesadas da AGM...

Problema 4: Clustering com AGM



Todos esses problemas se resolvem com

AGM

Árvore Geradora Mínima
(Minimum Spanning Tree)

1. Conectando Cidades com Custo Mínimo

Problema: Conectar todas as cidades gastando o mínimo possível.

1. Conectando Cidades com Custo Mínimo

Problema: Conectar todas as cidades gastando o mínimo possível.

MST (Árvore Geradora Mínima):

- Subgrafo **conexo** e **sem ciclos** (é uma árvore)

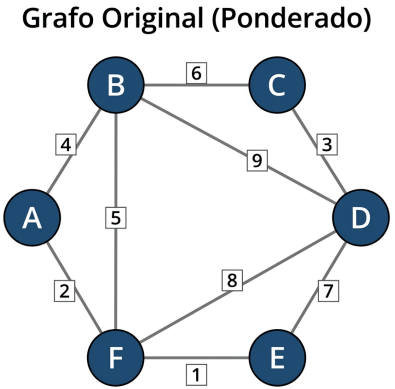


Figure: Grafo ponderado e sua MST (arestas em destaque).

1. Conectando Cidades com Custo Mínimo

Problema: Conectar todas as cidades gastando o mínimo possível.

MST (Árvore Geradora Mínima):

- Subgrafo **conexo** e **sem ciclos** (é uma árvore)
- Contém **todos** os vértices do grafo original

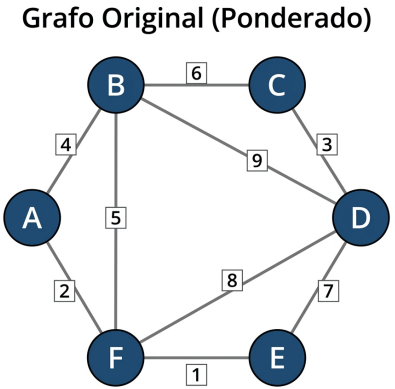


Figure: Grafo ponderado e sua MST (arestas em destaque).

1. Conectando Cidades com Custo Mínimo

Problema: Conectar todas as cidades gastando o mínimo possível.

MST (Árvore Geradora Mínima):

- Subgrafo **conexo** e **sem ciclos** (é uma árvore)
- Contém **todos** os vértices do grafo original
- Soma dos pesos das arestas é **mínima**

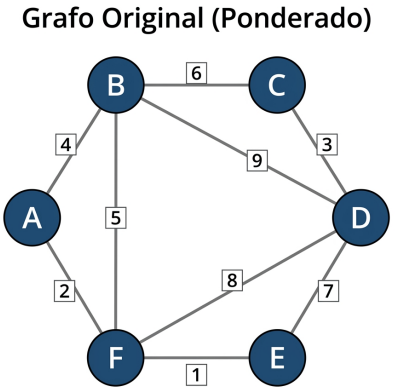


Figure: Grafo ponderado e sua MST (arestas em destaque).

1. Conectando Cidades com Custo Mínimo

Problema: Conectar todas as cidades gastando o mínimo possível.

MST (Árvore Geradora Mínima):

- Subgrafo **conexo** e **sem ciclos** (é uma árvore)
- Contém **todos** os vértices do grafo original
- Soma dos pesos das arestas é **mínima**
- Possui exatamente $V - 1$ arestas

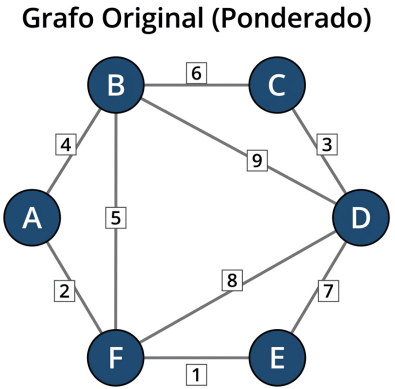


Figure: Grafo ponderado e sua MST (arestas em destaque).

2. Algoritmo de Kruskal (Guloso nas Arestas)

O algoritmo de Joseph Kruskal é intuitivo: "Sempre pegue a aresta mais barata, desde que não forme ciclo".

2. Algoritmo de Kruskal (Guloso nas Arestas)

O algoritmo de Joseph Kruskal é intuitivo: "Sempre pegue a aresta mais barata, desde que não forme ciclo".

Passo a Passo

1. Crie uma lista com **todas** as arestas do grafo.

2. Algoritmo de Kruskal (Guloso nas Arestas)

O algoritmo de Joseph Kruskal é intuitivo: "Sempre pegue a aresta mais barata, desde que não forme ciclo".

Passo a Passo

1. Crie uma lista com **todas** as arestas do grafo.
2. **Ordene** por peso crescente.

2. Algoritmo de Kruskal (Guloso nas Arestas)

O algoritmo de Joseph Kruskal é intuitivo: "Sempre pegue a aresta mais barata, desde que não forme ciclo".

Passo a Passo

1. Crie uma lista com **todas** as arestas do grafo.
2. **Ordene** por peso crescente.
3. Para cada aresta (u, v) na lista ordenada:

2. Algoritmo de Kruskal (Guloso nas Arestas)

O algoritmo de Joseph Kruskal é intuitivo: "Sempre pegue a aresta mais barata, desde que não forme ciclo".

Passo a Passo

1. Crie uma lista com **todas** as arestas do grafo.
2. **Ordene** por peso crescente.
3. Para cada aresta (u, v) na lista ordenada:
   - Se u e v estão em componentes **diferentes** $\Rightarrow$ **adicione** à MST e **una** os conjuntos.

Estrutura Auxiliar: Union-Find (DSU)

`find(x)`: descobre o componente de x. `union(x,y)`: une dois componentes.
Complexidade total: $O(E \log E)$ (dominada pela ordenação).

2. Algoritmo de Kruskal (Guloso nas Arestas)

O algoritmo de Joseph Kruskal é intuitivo: "Sempre pegue a aresta mais barata, desde que não forme ciclo".

Passo a Passo

1. Crie uma lista com **todas** as arestas do grafo.
2. **Ordene** por peso crescente.
3. Para cada aresta (u, v) na lista ordenada:
   - Se u e v estão em componentes **diferentes** $\Rightarrow$ **adicione** à MST e **una** os conjuntos.
   - Se já estão no **mesmo** componente $\Rightarrow$ **descarte** (formaria ciclo).

Estrutura Auxiliar: Union-Find (DSU)

`find(x)`: descobre o componente de x. `union(x,y)`: une dois componentes.
Complexidade total: $O(E \log E)$ (dominada pela ordenação).

2a. Kruskal – Implementação Java

```java
class Edge implements Comparable<Edge> {
    int u, v, weight;
    public Edge(int u, int v, int w) {
        this.u = u; this.v = v; this.weight = w;
    }
    public int compareTo(Edge other) { return this.weight - other.weight; }
}
public class KruskalMST {
    static int[] parent;
    static int find(int i) {
        if (parent[i] == i) return i;
        return parent[i] = find(parent[i]);
    }
    static void union(int i, int j) {
        int rootI = find(i), rootJ = find(j);
        if (rootI != rootJ) parent[rootI] = rootJ;
    }
    public static int kruskal(int V, List<Edge> edges) {
        Collections.sort(edges);
        parent = new int[V];
        for(int i=0; i<V; i++) parent[i] = i;
        int mstWeight = 0, edgesCount = 0;
        for (Edge e : edges) {
            if (find(e.u) != find(e.v)) {
                union(e.u, e.v);
                mstWeight += e.weight;
                edgesCount++;
            }
        }
        if (edgesCount != V - 1) return -1;
```

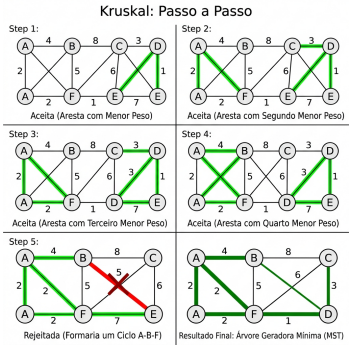
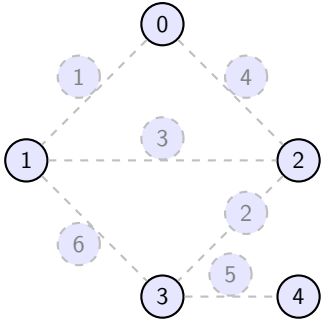


Figure: Representação do grafo.

2b. Kruskal – Trace Animado

Grafo com 5 vértices. Arestas ordenadas: (0,1)=1, (2,3)=2, (1,2)=3, (0,2)=4, (3,4)=5, (1,3)=6

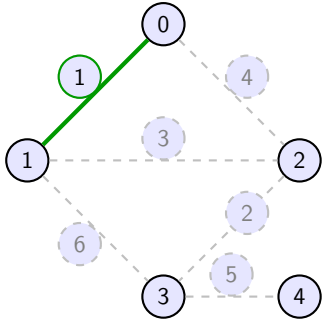
#	Aresta	Peso	Ação	Componentes



2b. Kruskal – Trace Animado

Grafo com 5 vértices. Arestas ordenadas: (0,1)=1, (2,3)=2, (1,2)=3, (0,2)=4, (3,4)=5, (1,3)=6

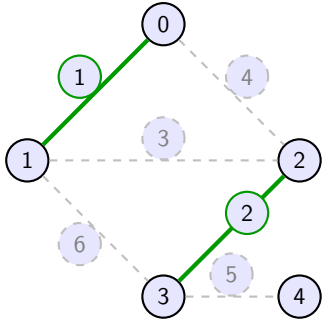
#	Aresta	Peso	Ação	Componentes
1	(0,1)	1	**Aceita**	{0,1} {2} {3} {4}



2b. Kruskal – Trace Animado

Grafo com 5 vértices. Arestas ordenadas: (0,1)=1, (2,3)=2, (1,2)=3, (0,2)=4, (3,4)=5, (1,3)=6

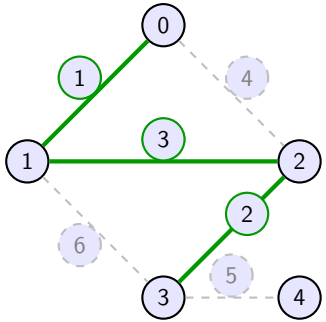
#	Aresta	Peso	Ação	Componentes
1	(0,1)	1	**Aceita**	{0,1} {2} {3} {4}
2	(2,3)	2	**Aceita**	{0,1} {2,3} {4}



2b. Kruskal – Trace Animado

Grafo com 5 vértices. Arestas ordenadas: (0,1)=1, (2,3)=2, (1,2)=3, (0,2)=4, (3,4)=5, (1,3)=6

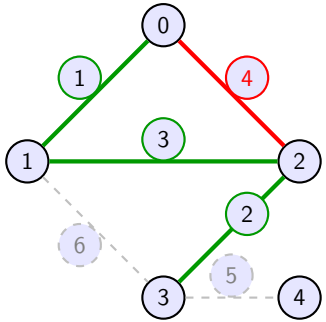
#	Aresta	Peso	Ação	Componentes
1	(0,1)	1	**Aceita**	{0,1} {2} {3} {4}
2	(2,3)	2	**Aceita**	{0,1} {2,3} {4}
3	(1,2)	3	**Aceita**	{0,1,2,3} {4}



2b. Kruskal – Trace Animado

Grafo com 5 vértices. Arestas ordenadas: (0,1)=1, (2,3)=2, (1,2)=3, (0,2)=4, (3,4)=5, (1,3)=6

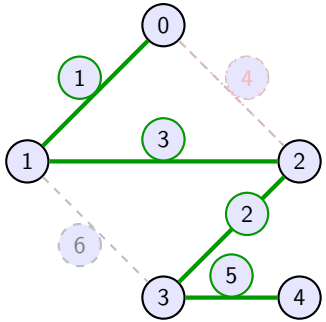
#	Aresta	Peso	Ação	Componentes
1	(0,1)	1	**Aceita**	{0,1} {2} {3} {4}
2	(2,3)	2	**Aceita**	{0,1} {2,3} {4}
3	(1,2)	3	**Aceita**	{0,1,2,3} {4}
4	(0,2)	4	**Rejeitada!**	ciclo na componente



2b. Kruskal – Trace Animado

Grafo com 5 vértices. Arestas ordenadas: (0,1)=1, (2,3)=2, (1,2)=3, (0,2)=4, (3,4)=5, (1,3)=6

#	Aresta	Peso	Ação	Componentes
1	(0,1)	1	**Aceita**	{0,1} {2} {3} {4}
2	(2,3)	2	**Aceita**	{0,1} {2,3} {4}
3	(1,2)	3	**Aceita**	{0,1,2,3} {4}
4	(0,2)	4	**Rejeitada!**	ciclo na componente
5	(3,4)	5	**Aceita**	{0,1,2,3,4}



Resultado

MST = {(0,1), (2,3), (1,2), (3,4)} ⇒ custo = **11**

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

Por que Kruskal aqui?

- **Grafo esparso**: poucas conexões possíveis entre cidades distantes ($E \ll V^2$)

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

Por que Kruskal aqui?

- **Grafo esparso**: poucas conexões possíveis entre cidades distantes ($E \ll V^2$)
- Lista de arestas **já disponível** (orçamentos de fibra por trecho)

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

Por que Kruskal aqui?

- **Grafo esparso**: poucas conexões possíveis entre cidades distantes ($E \ll V^2$)
- Lista de arestas **já disponível** (orçamentos de fibra por trecho)
- Componentes **independentes** podem ser unidos gradualmente

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

Por que Kruskal aqui?

- **Grafo esparso**: poucas conexões possíveis entre cidades distantes ($E \ll V^2$)
- Lista de arestas **já disponível** (orçamentos de fibra por trecho)
- Componentes **independentes** podem ser unidos gradualmente
- Não precisa de ponto de partida – ideal quando não há "centro"

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

Por que Kruskal aqui?

- **Grafo esparso**: poucas conexões possíveis entre cidades distantes ($E \ll V^2$)
- Lista de arestas **já disponível** (orçamentos de fibra por trecho)
- Componentes **independentes** podem ser unidos gradualmente
- Não precisa de ponto de partida – ideal quando não há "centro"

Exemplo Real

10 cidades, apenas **15 trechos** viáveis (esparso!):

- Ordena 15 trechos por custo
- Seleciona 9 arestas ($V - 1$)
- $O(15 \log 15)$ – **instantâneo**

Se fosse denso ($E = 45$), Prim seria mais eficiente.

2c. Quando usar Kruskal? – Grafos Esparsos

Cenário ideal: Rede de fibra óptica entre filiais espalhadas pelo estado.

Por que Kruskal aqui?

- **Grafo esparso**: poucas conexões possíveis entre cidades distantes ($E \ll V^2$)
- Lista de arestas **já disponível** (orçamentos de fibra por trecho)
- Componentes **independentes** podem ser unidos gradualmente
- Não precisa de ponto de partida – ideal quando não há "centro"

Exemplo Real

10 cidades, apenas **15 trechos** viáveis (esparso!):

- Ordena 15 trechos por custo
- Seleciona 9 arestas ($V - 1$)
- $O(15 \log 15)$ – **instantâneo**

Se fosse denso ($E = 45$), Prim seria mais eficiente.

Regra Prática

E próximo de V (esparso) $\Rightarrow$ **Kruskal**. E próximo de V^2 (denso) $\Rightarrow$ **Prim**.

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim “cresce” a MST a partir de um vértice, como uma planta.

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).
2. Insira todas as arestas dele numa **Priority Queue** (Min-Heap).

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).
2. Insira todas as arestas dele numa **Priority Queue** (Min-Heap).
3. Marque o vértice como **visitado**.

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).
2. Insira todas as arestas dele numa **Priority Queue** (Min-Heap).
3. Marque o vértice como **visitado**.
4. Enquanto a PQ não estiver vazia:

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).
2. Insira todas as arestas dele numa **Priority Queue** (Min-Heap).
3. Marque o vértice como **visitado**.
4. Enquanto a PQ não estiver vazia:
   - Remova a aresta de **menor peso** (u, v).

Complexidade: $O(E \log V)$ com Binary Heap. Parece com **Dijkstra**, mas aqui o critério é o peso da aresta, não a distância acumulada.

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).
2. Insira todas as arestas dele numa **Priority Queue** (Min-Heap).
3. Marque o vértice como **visitado**.
4. Enquanto a PQ não estiver vazia:
   - Remova a aresta de **menor peso** (u, v).
   - Se v já foi visitado $\Rightarrow$ **ignore** (lazy deletion).

Complexidade: $O(E \log V)$ com Binary Heap. Parece com **Dijkstra**, mas aqui o critério é o peso da aresta, não a distância acumulada.

3. Algoritmo de Prim (Guloso nos Vértices)

O algoritmo de Robert Prim "cresce" a MST a partir de um vértice, como uma planta.

Passo a Passo

1. Escolha um vértice inicial arbitrário (ex: 0).
2. Insira todas as arestas dele numa **Priority Queue** (Min-Heap).
3. Marque o vértice como **visitado**.
4. Enquanto a PQ não estiver vazia:
   - Remova a aresta de **menor peso** (u, v).
   - Se v já foi visitado $\Rightarrow$ **ignore** (lazy deletion).
   - Senão $\Rightarrow$ **adicione** à MST, marque v, insira arestas de v na PQ.

Complexidade: $O(E \log V)$ com Binary Heap. Parece com **Dijkstra**, mas aqui o critério é o peso da aresta, não a distância acumulada.

3a. Prim – Implementação Java

```java
class Pair implements Comparable<Pair> {
    int v, weight;
    public Pair(int v, int w) { this.v = v; this.weight = w; }
    public int compareTo(Pair o) { return this.weight - o.weight; }
}
public static int prim(int V, List<List<Pair>> adj) {
    boolean[] visited = new boolean[V];
    PriorityQueue<Pair> pq = new PriorityQueue<>();
    pq.add(new Pair(0, 0)); // Comeca do vertice 0
    int mstWeight = 0, nodesCount = 0;
    while (!pq.isEmpty()) {
        Pair current = pq.poll();
        int u = current.v;
        if (visited[u]) continue; // Lazy deletion
        visited[u] = true;
        mstWeight += current.weight;
        nodesCount++;
        for (Pair neighbor : adj.get(u)) {
            if (!visited[neighbor.v])
                pq.add(new Pair(neighbor.v, neighbor.weight));
        }
    }
    if (nodesCount != V) return -1;
    return mstWeight;
}
```

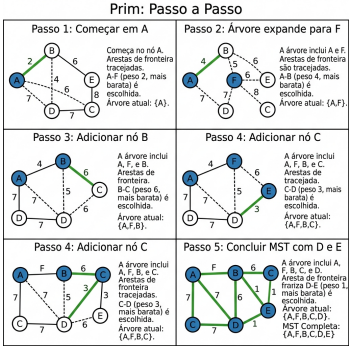
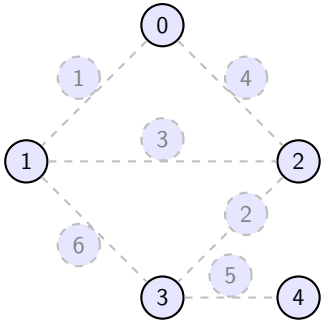


Figure: Árvore a partir do 0.

3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

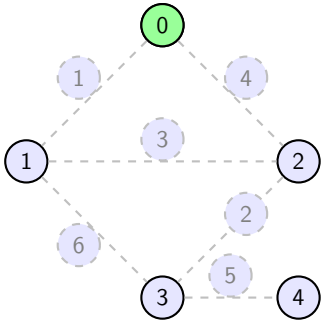
#	PQ.poll()	Peso	Ação	Visitados



3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

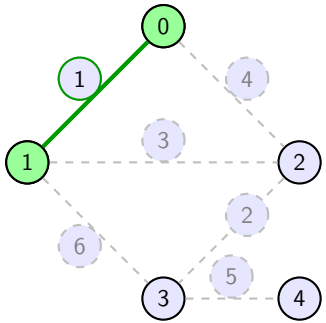
#	PQ.poll()	Peso	Ação	Visitados
1	(0, wt:0)	0	**Visita 0**	{0}



3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

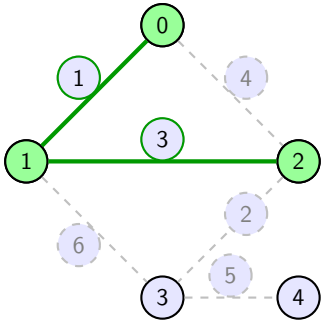
#	PQ.poll()	Peso	Ação	Visitados
1	(0, wt:0)	0	**Visita 0**	{0}
2	(1, wt:1)	1	**Visita 1**	{0,1}



3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

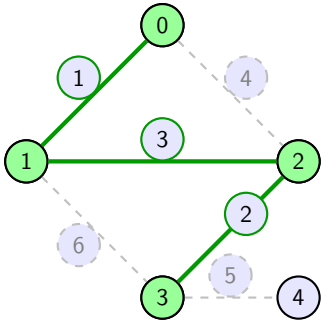
#	PQ.poll()	Peso	Ação	Visitados
1	(0, wt:0)	0	**Visita 0**	{0}
2	(1, wt:1)	1	**Visita 1**	{0,1}
3	(2, wt:3)	3	**Visita 2**	{0,1,2}



3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

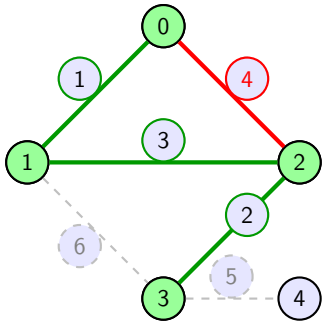
#	PQ.poll()	Peso	Ação	Visitados
1	(0, wt:0)	0	**Visita 0**	{0}
2	(1, wt:1)	1	**Visita 1**	{0,1}
3	(2, wt:3)	3	**Visita 2**	{0,1,2}
4	(3, wt:2)	2	**Visita 3**	{0..3}



3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

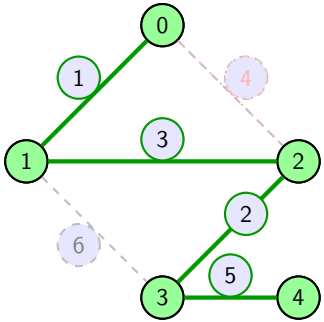
#	PQ.poll()	Peso	Ação	Visitados
1	(0, wt:0)	0	**Visita 0**	{0}
2	(1, wt:1)	1	**Visita 1**	{0,1}
3	(2, wt:3)	3	**Visita 2**	{0,1,2}
4	(3, wt:2)	2	**Visita 3**	{0..3}
5	(2, wt:4)	4	**Ignora!**	2 já visitado



3b. Prim – Trace Animado

Mesmo grafo (5 vértices). Iniciando pelo vértice **0**:

#	PQ.poll()	Peso	Ação	Visitados
1	(0, wt:0)	0	**Visita 0**	{0}
2	(1, wt:1)	1	**Visita 1**	{0,1}
3	(2, wt:3)	3	**Visita 2**	{0,1,2}
4	(3, wt:2)	2	**Visita 3**	{0..3}
5	(2, wt:4)	4	**Ignora!**	2 já visitado
6	(4, wt:5)	5	**Visita 4**	{0..4}



Resultado

MST = mesmas arestas que Kruskal! Custo = **11**.
Passo 5: `if(visited[2]) continue` (*lazy deletion*).

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

Por que Prim aqui?

- **Grafo denso**: quase todo setor conecta com todos ($E \approx V^2$)

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

Por que Prim aqui?

- **Grafo denso**: quase todo setor conecta com todos ($E \approx V^2$)
- **Ponto de partida definido**: o reservatório (vértice 0)

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

Por que Prim aqui?

- **Grafo denso**: quase todo setor conecta com todos ($E \approx V^2$)
- **Ponto de partida definido**: o reservatório (vértice 0)
- A árvore "cresce" naturalmente a partir da fonte de água

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

Por que Prim aqui?

- **Grafo denso**: quase todo setor conecta com todos ($E \approx V^2$)
- **Ponto de partida definido**: o reservatório (vértice 0)
- A árvore "cresce" naturalmente a partir da fonte de água
- Com lista de adjacência densa, $O(E \log V)$ supera $O(E \log E)$

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

Por que Prim aqui?

- **Grafo denso**: quase todo setor conecta com todos ($E \approx V^2$)
- **Ponto de partida definido**: o reservatório (vértice 0)
- A árvore "cresce" naturalmente a partir da fonte de água
- Com lista de adjacência densa, $O(E \log V)$ supera $O(E \log E)$

Exemplo Real

20 setores, quase todos interconectados (**150 arestas**):

- Kruskal: ordena 150 arestas $\Rightarrow O(150 \log 150)$
- Prim: cresce do reservatório $\Rightarrow O(150 \log 20)$
- Prim é **mais eficiente** em grafos densos!

3c. Quando usar Prim? – Grafos Densos

Cenário ideal: Rede de irrigação numa fazenda – todos os setores próximos entre si.

Por que Prim aqui?

- **Grafo denso**: quase todo setor conecta com todos ($E \approx V^2$)
- **Ponto de partida definido**: o reservatório (vértice 0)
- A árvore "cresce" naturalmente a partir da fonte de água
- Com lista de adjacência densa, $O(E \log V)$ supera $O(E \log E)$

Exemplo Real

20 setores, quase todos interconectados (**150 arestas**):

- Kruskal: ordena 150 arestas $\Rightarrow O(150 \log 150)$
- Prim: cresce do reservatório $\Rightarrow O(150 \log 20)$
- Prim é **mais eficiente** em grafos densos!

Outro Cenário Ideal para Prim

Data centers: conectar racks de servidores numa sala. Grafo denso (todos os racks podem se conectar), ponto de partida no switch principal.

4. Aplicações Reais de MST

Redes e Infraestrutura

- Cabeamento de rede (Ethernet/fibra)
- Rede elétrica entre postes
- Encanamento / irrigação
- Estradas entre cidades

4. Aplicações Reais de MST

Redes e Infraestrutura

- Cabeamento de rede (Ethernet/fibra)
- Rede elétrica entre postes
- Encanamento / irrigação
- Estradas entre cidades

Clustering (Agrupamento)

Remova as $k-1$ arestas mais pesadas da MST $\Rightarrow$ k clusters!
Usado em segmentação de imagens e mineração de dados.

4. Aplicações Reais de MST

Redes e Infraestrutura

- Cabeamento de rede (Ethernet/fibra)
- Rede elétrica entre postes
- Encanamento / irrigação
- Estradas entre cidades

Clustering (Agrupamento)

Remova as $k-1$ arestas mais pesadas da MST $\Rightarrow$ k clusters!
Usado em segmentação de imagens e mineração de dados.

Competições de Programação

- "Custo mínimo para conectar tudo" = MST
- Variações: 2ª melhor MST, MST com restrições
- Grafo dirigido $\Rightarrow$ Arborescência (Edmonds)

4. Aplicações Reais de MST

Redes e Infraestrutura

- Cabeamento de rede (Ethernet/fibra)
- Rede elétrica entre postes
- Encanamento / irrigação
- Estradas entre cidades

Clustering (Agrupamento)

Remova as $k-1$ arestas mais pesadas da MST $\Rightarrow$ k clusters!
Usado em segmentação de imagens e mineração de dados.

Competições de Programação

- "Custo mínimo para conectar tudo" = MST
- Variações: 2ª melhor MST, MST com restrições
- Grafo dirigido $\Rightarrow$ Arborescência (Edmonds)

Design de Circuitos (PCB)

Conectar pinos de um chip com trilhas de menor comprimento total – é literalmente uma MST no plano 2D!

5. Kruskal vs Prim: O Veredito

Característica	Kruskal	Prim

5. Kruskal vs Prim: O Veredito

Característica	**Kruskal**	**Prim**
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**

5. Kruskal vs Prim: O Veredito

Característica	Kruskal	Prim
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**
Lógica	Ordena todas as arestas	Cresce árvore com PQ

5. Kruskal vs Prim: O Veredito

Característica	**Kruskal**	**Prim**
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**
Lógica	Ordena todas as arestas	Cresce árvore com PQ
Estrutura auxiliar	Union-Find (DSU)	Min-Heap + visited[]

5. Kruskal vs Prim: O Veredito

Característica	**Kruskal**	**Prim**
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**
Lógica	Ordena todas as arestas	Cresce árvore com PQ
Estrutura auxiliar	Union-Find (DSU)	Min-Heap + visited[]
Complexidade	$O(E \log E)$	$O(E \log V)$

5. Kruskal vs Prim: O Veredito

Característica	**Kruskal**	**Prim**
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**
Lógica	Ordena todas as arestas	Cresce árvore com PQ
Estrutura auxiliar	Union-Find (DSU)	Min-Heap + visited[]
Complexidade	$O(E \log E)$	$O(E \log V)$
Melhor para	Grafos **esparsos**	Grafos **densos**

5. Kruskal vs Prim: O Veredito

Característica	**Kruskal**	**Prim**
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**
Lógica	Ordena todas as arestas	Cresce árvore com PQ
Estrutura auxiliar	Union-Find (DSU)	Min-Heap + visited[]
Complexidade	$O(E \log E)$	$O(E \log V)$
Melhor para	Grafos **esparsos**	Grafos **densos**
Precisa de origem?	**Não**	**Sim**

5. Kruskal vs Prim: O Veredito

Característica	Kruskal	Prim
Abordagem	Guloso nas **arestas**	Guloso nos **vértices**
Lógica	Ordena todas as arestas	Cresce árvore com PQ
Estrutura auxiliar	Union-Find (DSU)	Min-Heap + visited[]
Complexidade	$O(E \log E)$	$O(E \log V)$
Melhor para	Grafos **esparsos**	Grafos **densos**
Precisa de origem?	**Não**	**Sim**

Em Competições

Kruskal é o favorito: DSU tem $\sim$10 linhas e a lógica de ordenar arestas é menos propensa a bugs.

Na Indústria

Prim quando o grafo vem como lista de adjacência (API de rede, banco de dados geográfico) e já há um nó raiz definido.