INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO-CAMPUS URUTAÍ

Charley Marra

Daniel Rincon

João Pedro de Queiroz

Renan Nunes

Implementação do Algoritmo de Kruskal para obtenção da Árvore Geradora Mínima

Urutaí - 2025

Resumo: O app utiliza o algoritmo de Kruskal para determinar a Árvore Geradora Mínima (AGM) de um grafo que é ponderado e conectado. Através de uma interface gráfica o usuário é capaz de adicionar vértices e arestas, rodar o algoritmo, ver o resultado e limpar os dados.

O algoritmo opera ao classificar as arestas por peso e escolhendo apenas as que não criam ciclos, empregando a estrutura Union-Find. A saída mostra as arestas selecionadas e o custo total da árvore geradora mínima (AGM).

O app conta com tratamento de exceção que lida com erros frequentes, como entradas inválidas, pesos errados e campos não preenchidos. A ordenação das arestas torna sua complexidade O(E log E).

Em resumo, o software oferece uma solução funcional e intuitiva para o cálculo da (AGM) utilizando Kruskal, complementada com interface gráfica e validação de erros.

Palavras-chave: Algoritmo de Kruskal, Árvore Geradora Mínima (AGM), Arestas, Vértices, Interface gráfica (GUI), Custo total.

RELATÓRIO TÉCNICO — Algoritmo de Kruskal com Interface Gráfica (GUI)

1. Introdução

Este app realiza a implementação do Algoritmo de Kruskal para obtenção da Árvore Geradora Mínima (AGM) de um grafo, permitindo com que usuário interaja por meio de uma interface gráfica (GUI) construída com Java Swing.

O implementação possibilita:

- Criar um grafo informando a quantidade de vértices
- Adicionar arestas com origem, destino e peso
- Executar o algoritmo de Kruskal
- Visualizar todas as arestas adicionadas
- Visualizar a Árvore Geradora Mínima (AGM) resultante
- Ver o peso total (custo mínimo)
- Limpar dados e reiniciar o processo
- Trabalhar com verificação e tratamento de exceções para evitar erros comuns

2. Objetivo

O app tem como objetivo permitir ao usuário:

- Construir um grafo manualmente
- Aplicar Kruskal de forma interativa
- Obter a Árvore Geradora Mínima com segurança
- Visualizar o resultado de forma clara

Pode ser utilizado em diferentes cenários como:

- Aulas de estruturas de dados
- Demonstrações visuais do algoritmo
- Trabalhos acadêmicos de grafos
- Experimentos com (AGM) em redes

3 - Entradas do Usuário: O que informa ao programa

Para começar a usar, você precisa fornecer as informações básicas para construir o grafo:

O Tamanho do Grafo (Vértices)

- Basta colocar a quantidade total de vértices que o seu grafo terá.

As Ligações (Arestas)

- Agora você informa para o programa como deve fazer as ligações.

- Sai de: De onde a aresta começa (um número de vértice).
- Vai para: Para onde a aresta vai (um número de vértice).
- Custa: O "preço" ou "distância" dessa ligação.
- Exemplo: Ligar o ponto 0 no ponto 1 custa 4.
- Os Comandos (Interface)
- Você controla tudo usando botões simples:

Criar Grafo:

- Para iniciar com o número de vértices que você informou.

Adicionar Aresta:

- Para incluir cada ligação.

Rodar Kruskal:

- Ele inicia o cálculo da Árvore Geradora Mínima (MST).

Limpar:

- Para apagar.

4 - Saídas do Programa: O que ele te mostra

Depois de rodar o Kruskal, o programa te mostra o que aconteceu:

- Seu Histórico Completo
- No canto esquerdo, ele te mostra todas as ligações que você adicionou.

O Resultado de Kruskal (MST)

No outro canto, ele te mostra só as ligações essenciais que ele escolheu para formar a Árvore Geradora Mínima (a tal da MST). É o caminho mais econômico!

- O Custo Total

  Um resumo importante! O programa mostrará o Peso total da Árvore Geradora Mínima, permitindo que você saiba qual é a solução de custo mínimo.

- Exemplo: "Peso total: 17"

Mensagens de Alerta

Se algo der errado, você receberá mensagens de erro claras e fáceis de entender (amigáveis), como se fosse um aviso de um assistente:

- "Você tentou adicionar uma aresta sem antes criar o grafo!"
- "Opa! Você deixou algum campo da aresta vazio."
- "Os vértices que você informou não existem no grafo criado."
- "Não aceitamos pesos negativos, por favor, corrija."
- "Não há arestas suficientes para rodar Kruskal. Adicione mais!"

5 - Estrutura Interna: As "Peças" do Programa

Por trás da interface, o programa é organizado em classes que cuidam de tarefas específicas:

- Classe Aresta (O Objeto de Ligação)

  Essa classe é responsável por guardar as informações de cada ligação: origem, destino, e peso.

Ela tem um método especial (compareTo) para que o programa possa ordenar as arestas pelo peso (essencial para o Kruskal!).

Também tem um método para ser exibida como texto (toString).

Classe UnionFind (O Detector de Ciclos)

Essa é a classe "inteligente" que previne que ciclos se formem na sua Árvore Geradora Mínima.

- find(): Ajuda a descobrir a qual conjunto um vértice pertence.
- union(): Junta dois conjuntos.

Ela usa truques de otimização (compressão de caminho e union by rank) para ser super rápida ao lidar com grandes grafos.

Classe Grafo (O Estrutura Central)

A classe que armazena toda a sua rede:

- Guarda o número de vértices e a lista de arestas.
- adicionarAresta(): Onde as ligações são incluídas.
- kruskal(): O coração do programa, onde o algoritmo é executado, usando a lista de arestas e o UnionFind.

Classe KruskalGUI2 (A Interface)

É a classe que você vê e interage. Ela contém:

- Todos os campos de entrada, caixas de texto e botões.
- O código que ouve suas ações (cliques de botão) e chama as funções nas classes Grafo, Aresta, etc., para produzir os resultados.

6.Funcionamento do Algoritmo de Kruskal

O algoritmo Kruskal começa ordenando todas as arestas do grafo que seja em ordem crescente de peso. Após isso, utiliza-se a estrutura de dados Union-Find, sendo ela responsável por identificar se a adição de uma aresta resulta na formação de um ciclo. Depois da ordenação, o algoritmo percorre as arestas de acordo com a forma estabelecida, sendo selecionado apenas aquelas que não produzem ciclos quando inseridas na solução de forma parcial. Essa etapa do processo é repetida de forma contínua até que exatamente $V-1$ arestas

tenham sido escolhidas, onde V esteja representando o número de vértices do grafo. No final desse procedimento, temos uma uma Árvore Geradora Mínima, isto é, a estrutura que conecta todos os vértices com o menor custo total possível

7- Tratamento de Exceções Implementado

O programa conta com um conjunto de tratamento de erros para que assim seja garantido maior robustez, segurança e fluidez. Primeiramente, são capturadas situações de NumberFormatException, que ocorre quando o usuário acaba fornecendo dados que não são numéricos em campos que seria exigido valores inteiros ou reais. Casos que acontecem no IllegalArgumentException também são tratados, sendo acionados no momento que o usuário insere valores que são inválidos, como pesos negativos ou até vértices que são inexistentes. Além disso, temos o IllegalStateException que é lançado quando se tenta executar o algoritmo de kruskal antes da criação adequada do grafo. A verificação de acesso a vértices sendo fora dos limites definidos resulta em IndexOutOfBoundsException, por exemplo quando o grafo possui 5 vértices, mas o usuário acaba tentando utilizar índices além do intervalo permitido. Por fim, situações de NullPointerException que são tratadas para impedir operações sobre estruturas ainda não inicializadas.

8-Complexidade do Algoritmo

A complexidade do algoritmo de kruskal é determinada principalmente pelo processo que busca ordenar as arestas. Primeiramente, todas as E (arestas) do grafo são ordenadas em ordem crescente de acordo com o peso, etapa que possui custo O(E log E). Após a ordenação, as operações relacionadas com a estrutura Union-Find, apresentam custo que é praticamente constante, devido suas otimizações de path compression e union by rank, resultando em um tempo amortizado de O(1) por operação. Sendo assim, a complexidade total do algoritmo permanece sendo dominada pelo custo da ordenação, sendo assim, resultando em um tempo final de execução que seja igual a O(E log E).

Dessa forma, é considerado que E representando o número de arestas do grafo e V o número de vértices. Sendo como em muitos grafos o número de arestas tende a ser significamente maior que o número tido em vértices, a ordenação das arestas se tornam o fator mais difícil do processo, sendo assim justificado sua predominância na complexidade total

9. Limitações de Implementação

- O grafo pelo código não é desenhado visualmente, ele apenas vai mostrar as arestas fazendo conexões, por exemplo: (b,c) 5.
- Não consegue remover as arestas.
- Interface gráfica baseada em Swing, uma tecnologia que pode ser considerada mais antiga.
- O código não impede a criação de grafos desconexos.

10. Possíveis Melhorias Futuras

- Colocar o desenho do grafo na interface gráfica.
- Colorir as arestas na AGM.
- Opção de exportar o resultado em PDF ou CSV.
- Implementar o algoritmo de PRIM para fazer a comparação.
- Permitir carregar os grafos a partir de um arquivo no final.

11. Conclusão

O código mostra como o Algoritmo de Kruskal faz sua operação com a Árvore Geradora Mínima, buscando o menor caminho possível. Através do uso de GUI deixa mais fácil a compreensão do tema para estudantes, fazendo com que o usuário tenha uma boa experiência.

Com seus tratamentos de exceções, deixa o código compatível com o Algoritmo De Kruskal, a aplicação torna-se muito interessante e confiável, pronto para uso acadêmico.