Seminário de Teoria dos Grafos

Otávio Ferreira Silva

03/06/2025

Introdução

- Esse Slide terá como intuito explicar o algorítmo de Kruskal feito em C++, onde irei explicar o mesmo.

O que é?

É uma solução eficiente para encontrar a árvore geradora mínima (AGM) de um grafo ponderado. Ele funciona adicionando arestas de menor peso ao grafo, garantindo que não se formem ciclos, até que todos os vértices estejam conectados. Esse algoritmo é um exemplo de algoritmo guloso, pois sempre escolhe a solução localmente ótima, esperando encontrar a solução globalmente ótima. O algorítmo de Kruskal é muito utilizado em projetos de rede, planejamentos de infraestrutura, protocolos de autoconfiguração, etc.

Vem aí

O próximo slide mostrará o resultado da compilação e execução de um código de Algorítmo de Kruskal (feito no Visual Studio Code) em C++, em que o código lê um arquivo (presente no arquivo zip GrafosCC) chamado inp.txt. Nele, há um exemplo de grafo onde há 5 arestas e 6 vértices, mostra as adjacências do grafo e, no final, mostra as arestas e o custo total da Árvore Geradora Mínima (AGM).

Execução do algorítmo

```
Microsoft Windows [versão 10.0.26100.1742]
(c) Microsoft Corporation. Todos os direitos reservados.

D:\Seminário do Andre>g++ Kruskal.cpp -o kruskal_app

D:\Seminário do Andre>kruskal_app < inp.txt
Lista de Adjacencia do Grafo Lido:
0: [-> 1, w: 1.3] [-> 4, w: 2.3]
1: [-> 0, w: 1.3] [-> 2, w: 3.2] [-> 4, w: 4.4]
2: [-> 1, w: 3.2] [-> 3, w: 0.5]
3: [-> 2, w: 0.5] [-> 4, w: 2.6]
4: [-> 0, w: 2.3] [-> 1, w: 4.4] [-> 3, w: 2.6]

Total Vertices: 5
Total Arestas: 6

--- Executando Algoritmo de Kruskal ---
Arestas da Arvore Geradora Minima (AGM):
2 -- 3  (Peso: 0.5)
0 -- 1  (Peso: 1.3)
0 -- 4  (Peso: 2.3)
3 -- 4  (Peso: 2.6)
Custo total da AGM: 6.70

D:\Seminário do Andre>
```

Figure: Resultado da leitura do inp.txt

Atenção

Os próximos slides irão apresentar o código usado para o seminário no bloco de notas. O motivo foi, devido o número alto de linhas do código, foi necessário dividir o código em 4 imagens.

Código usado Parte 1

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>

// Usando o typedef do seu arquivo GrafoListaPeso.cpp
typedef std::vector<std::vector<std::pair<int, double>>> LP;

// Estrutura para representar uma aresta, necessária para o algoritmo de Kruskal
struct Aresta {
    int origem, destino;
    double peso;
};

// Estrutura auxiliar para Disjoint Set Union (DSU) ou Union-Find
struct DisjointSet {
    int *pai;

    // Construtor
    DisjointSet(int n) {
        pai = new int[n];
        for (int i = 0; i < n; i++) {
            pai[i] = i; // Cada vértice é seu próprio pai inicialmente
        }
    }
```

Ln 12, Col 17 | 4.042 caracteres | 100% | Windows (CRLF) | UTF-8

Figure: Parte 1 do código

Código usado Parte 2

```
    // Encontra o representante do conjunto de 'i' (com compressão de caminho)
    int encontrar(int i) {
        if (pai[i] == i) {
            return i;
        }
        return pai[i] = encontrar(pai[i]);
    }

    // Une os conjuntos de 'x' e 'y'
    void unir(int x, int y) {
        int raiz_x = encontrar(x);
        int raiz_y = encontrar(y);
        if (raiz_x != raiz_y) {
            pai[raiz_x] = raiz_y;
        }
    }

    // Destrutor para liberar a memória
    ~DisjointSet() {
        delete[] pai;
    }
};

// Função para comparar duas arestas pelo peso. Usada para ordenação.
bool compararArestas(const Aresta& a, const Aresta& b) {
    return a.peso < b.peso;
```

Figure: Parte 2 do código

Código usado Parte 3

```
}

// Implementação do Algoritmo de Kruskal
void kruskalAGM(int V, std::vector<Aresta>& arestas) {
    std::cout << "\n--- Executando Algoritmo de Kruskal ---\n";
    std::vector<Aresta> resultado;
    double custo_total = 0.0;

    // 1. Ordena todas as arestas em ordem crescente de peso.
    std::sort(arestas.begin(), arestas.end(), compararArestas);

    // Cria os subconjuntos para a estrutura Union-Find.
    DisjointSet ds(V);

    // 2. Itera sobre as arestas ordenadas.
    for (const auto& aresta : arestas) {
        int u = aresta.origem;
        int v = aresta.destino;

        int set_u = ds.encontrar(u);
        int set_v = ds.encontrar(v);

        // 3. Verifica se a aresta forma um ciclo.
        if (set_u != set_v) {
            resultado.push_back(aresta);
            custo_total += aresta.peso;
```

Figure: Parte 3 do código

Código usado Parte 4

```
            ds.unir(u, v);
        }
    }

    // Imprime o resultado
    std::cout << "Arestas da Arvore Geradora Minima (AGM):" << std::endl;
    for (const auto& aresta : resultado) {
        std::cout << aresta.origem << " -- " << aresta.destino << "  (Peso: " << aresta.peso << ")"
<< std::endl;
    }
    std::cout << "Custo total da AGM: " << std::fixed << std::setprecision(2) << custo_total << std::endl;
}

// Função de leitura do arquivo GrafoListaPeso.cpp
void leituraGrafo(LP &G, int m) {
    int a, b;
    double c;
    while (m--) {
        std::cin >> a >> b >> c;
        G[a].push_back(std::make_pair(b, c));
        G[b].push_back(std::make_pair(a, c));
    }
}

// Função de escrita do arquivo GrafoListaPeso.cpp - COM CORREÇÃO
void escritaGrafo(LP &G) {
```

Figure: Parte 4 do código

Código usado Parte 5

```
    int n = G.size();
    std::cout << "Lista de Adjacencia do Grafo Lido:" << std::endl;
    for (int u = 0; u < n; u++) {
        std::cout << u << ": ";
        // Alterado para ser compatível com C++11/14
        for (const auto& par : G[u]) {
            int v = par.first;
            double peso = par.second;
            std::cout << "[-> " << v << ", w: " << peso << "] ";
        }
        std::cout << std::endl;
    }
}

int main() {
    int n, m;
    std::cin >> n >> m;

    LP Grafo;
    Grafo.assign(n, std::vector<std::pair<int, double>>());
    leituraGrafo(Grafo, m);
    escritaGrafo(Grafo);

    std::cout << "\nTotal Vertices: " << n << std::endl;
    std::cout << "Total Arestas: " << m << std::endl;
```

Figure: Parte 5 do código

Código usado Parte Final

```
    std::vector<Aresta> todasArestas;
    for (int u = 0; u < n; u++) {
        // Alterado para ser compatível com C++11/14
        for (const auto& par : Grafo[u]) {
            int v = par.first;
            double peso = par.second;
            if (u < v) {
                todasArestas.push_back({u, v, peso});
            }
        }
    }

    kruskalAGM(n, todasArestas);

    return 0;
}
```

Figure: Parte final do código

Obrigado!

Até um outro dia.