UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

ESTRUTURAS DE DADOS II

Relatório do 1º trabalho

Autores
GUILHERME GOES ZANETTI,
JOÃO PEDRO MILLI BIANCARDI,
LUIZA BATISTA LAQUINI

July 31, 2021

Contents

1 Introdução

O objetivo do presente trabalho é colocar em prática os conhecimentos obtidos na disciplina Estrutura de Dados II na criação de um programa C estruturado a partir de conceitos, algoritmos e métodos aprendidos na disciplina.

O trabalho consiste na aplicação do algoritmo de Krustal, que busca uma MST (Minimal Spanning Tree) para um grafo conexo com pesos, isto é, uma árvore de extensão com peso menor ou igual a cada uma das outras árvores de extensão possíveis.

Dado uma entrada contendo o nome dos pontos e suas respectivas coordenadas, o programa deve gerar agrupamentos, unindo os pontos cuja distância entre eles seja a menor possível até a obtenção de k grupos, onde k é especificado pelo usuário. Este relatório abordará sobre a metodologia do trabalho, assim como a análise de complexidade dos algoritmos utilizados no programa e a análise empírica referente ao tempo de execução da aplicação.

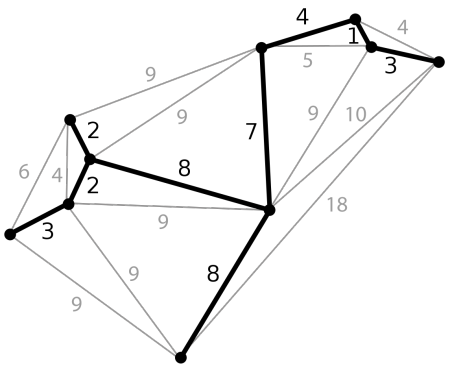

Figure 1: Árvore de extensão mínima de um grafo plano, o número entre os vértices representam os pesos de cada aresta

2 Metodologia

Para melhor estruturação, o trabalho foi modularizado em diversos headers, cada um desempenhando uma função específica:

2.1 main.c

O arquivo do cliente será responsável pela chamada de todos os outros headers e pela execução de algoritmos específicos para o problema. Ele será responsável pela leitura do arquivo de entrada, de onde serão obtidos nomes e coordenadas dos pontos a serem analisados, executará a lógica principal do algoritmo de Kruskal e será responsável pela ordenação lexicográfica dos nomes dos pontos de cada grupo para serem impressos em um arquivo "saida.txt". Para isto, o cliente cria vetores auxiliares da estrutura Ponto, para receber cada ponto dos k grupos existentes e ordená-los lexicograficamente utilizando o algoritmo Merge sort, um algoritmo de ordenação que consiste em dividir os grupos em vários subgrupos e realizar sub-comparações utilizando recursividade.

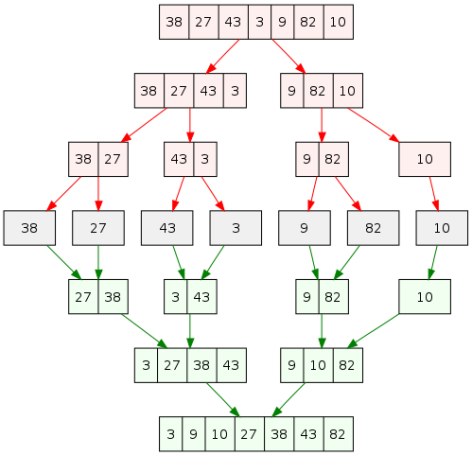

Figure 2: Algoritmo de Merge sort

2.2 ponto.c

O arquivo ponto.c contém a declaração da estrutura ponto e funções relacionadas, os pontos serão os vértices da árvore analisada.

Cada estrutura ponto contém nome, um index relativo a ordem de sua inserção, um

id relativo ao grupo ao qual pertence, um vetor de coordenadas contendo as coordenadas relativas ao ponto e o número total de coordenadas (dimensões) que o ponto tem.

O arquivo contém uma função para calcular a distância euclidiana entre dois pontos, que será útil na aplicação do algoritmo de Krustal, além disso o arquivo contém funções de retorno de variáveis, criação e liberação de memória, além da função para mudar o id do ponto, também útil na aplicação do algoritmo.

2.3 arestas.c

O arquivo aresta.c contém a declaração da estrutura aresta e funções relacionadas, as arestas serão o peso entre pontos representando as suas distâncias.

Cada aresta contém os index dos pontos aos quais ela se refere e a distância entre eles. Há também uma estrutura que contém um vetor com todas as arestas calculadas, permitindo que as arestas sejam ordenadas pelo seu tamanho com um qsort.

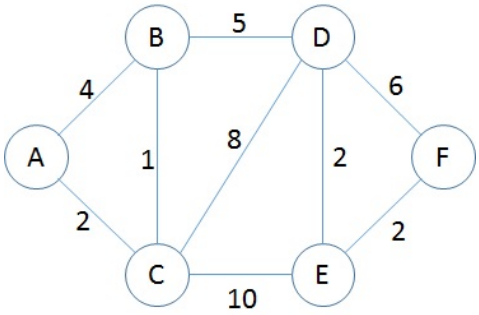

Figure 3: Pontos ou vértices representados por letras e arestas representadas por números

2.4 grupos.c

O arquivo grupos.c contém a declaração da estrutura grupos e funções relacionadas, uma estrutura grupos representa todos os pontos e a que grupos eles pertencem, além de salvar o tamanho atual de cada grupo.

A estrutura grupos contém um vetor com todos os pontos, o número de pontos lidos na entrada e um vetor de inteiros contendo o número de pontos de cada grupo existente. O arquivo também implementa funções de Union e Find, que une dois grupos e retorna a qual grupo um ponto pertence por meio de um algoritmo de *weighted quick-union com path compression*.

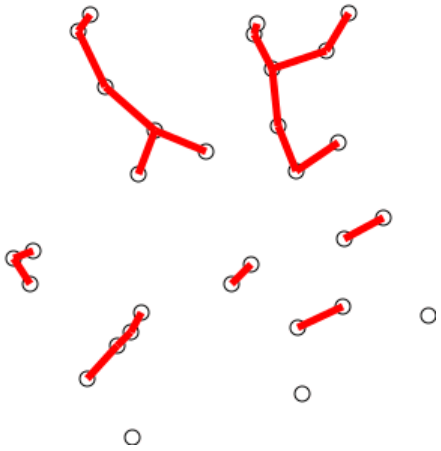

Figure 4: Agrupamento de pontos

3 Análise de Complexidade

Principais partes (passos) da implementação, considerando N como o número de pontos:

Função **calculaTodasAsArestas**: Realiza o cálculo da distância entre todas as arestas existentes e as ordena em um vetor de ponteiros para arestas. A complexidade do cálculo das distâncias seria igual ao número de arestas, já a ordenação por quicksort terá uma complexidade no pior caso:

$$CalculaTodasAsArestas = O(NumDeArestas^2) \quad (1)$$

$$CalculaTodasAsArestas = O(N^4) \quad (2)$$

Sendo igual à $O(N^2.\log^2 N)$ no caso médio.

Merge Sort: Algoritmo de ordenação que divide o vetor atual em dois subvetores, ordena RECURSIVAMENTE esses subvetores e os mescla! Faz, em seu pior caso de execução, no máximo NlogN comparações e 6NlogN acessos à memória.

Kruskal: Algoritmo em teoria dos grafos que busca uma árvore geradora mínima para um grafo conexo com pesos. Ou seja, ele encontra um subconjunto das arestas - que forma uma árvore que inclui todos os vértices - onde o peso total, dado pela soma dos pesos das arestas, é minimizado. Sua complexidade será o número de arestas vezes a complexidade do Find, que no pior caso é logN:

$$Kruskal = O(N^2 log(N)) \quad (3)$$

4 Análise Empírica

Análise do tempo de execução:

Etapa 1: Leitura dos dados.
Etapa 2: Cálculo das arestas.
Etapa 3: Ordenação das distâncias.
Etapa 4: Obtenção da MST.
Etapa 5: Identificação dos grupos.
Etapa 6: Escrita do arquivo de saída.

Observa-se o aumento do tempo de execução de acordo com o aumento de informação no arquivo de entrada:

Arquivo 2.txt = 7,4 KB
Arquivo 4.txt = 278,2 KB
Arquivo 5.txt = 1,0 MB

OBS: Os dados dessa tabela foram obtidos a partir da execução do código em um Notebook com 16gb de memória RAM e utilizando como padrão K=3

ETAPA	ARQUIVO					
	2.txt		4.txt		5.txt	
	Tempo de execução (s)	%	Tempo de execução (s)	%	Tempo de execução (s)	%
1	0,025	27,47%	0,125	0,30%	0,298	0,11%
2	0,032	35,16%	26,156	62,51%	190,067	72,38%
3	0,013	14,29%	12,49	29,85%	59,008	22,47%
4	0,008	8,79%	3,009	7,19%	13,106	4,99%
5	0,006	6,59%	0,049	0,12%	0,093	0,04%
6	0,007	7,69%	0,013	0,03%	0,023	0,01%
Total	0,091		41,842		262,595	

Figure 5: Análise do tempo de execução das etapas do programa