UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

ERICO GUEDES

EZEQUIEL DEMETRAS SILVA

RELATÓRIO REFERENTE AO TRABALHO 1

Vitória

2023

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

ERICO GUEDES

EZEQUIEL DEMETRAS SILVA

RELATÓRIO REFERENTE AO TRABALHO 1

Relatório apresentado ao curso de Ciência da Computação, na disciplina Técnicas de Busca e Ordenação, como parte dos requisitos para a obtenção de nota.

Vitória

2023

1. INTRODUÇÃO

Neste relatório serão apresentados os seguintes tópicos:

- Processo de desenvolvimento do algoritmo implementado;
- Apresentação da complexidade de cada etapa do algoritmo;
- Exibição das medições de tempo de cada etapa do algoritmo;

2. METODOLOGIA

Neste tópico será tratado como foi desenvolvido o algoritmo de agrupamento. O algoritmo foi dividido em três TADs principais: Arquivo.h, Distancia.h e Ponto.h. Cada TAD possui uma *struct* própria, onde cada uma será abordada logo abaixo.

- Arquivo.h

Esse TAD contém o código necessário para realizar a leitura do arquivo de entrada e a exportação dos dados interpretados. Sua *struct* é:

```
typedef struct {
    int Dimensao;
    int Contagem;
    Ponto* P;
    Distancia* D;
    int QuantD;
} Entrada;
```

1. Dimensao: é armazenado a quantidade de coordenadas que cada ponto irá possuir;
2. Contagem: número total de pontos;
3. P: vetor da *struct* Ponto (será abordada mais à frente);
4. D: vetor da *struct* Distancia (será abordada mais à frente);
5. QuantD: número total de distâncias entre os pontos;

O objetivo dessa estrutura é armazenar todas as informações úteis para o funcionamento do algoritmo.

- Distancia.h

Esse TAD contém o código necessário para realizar as ações referentes às distâncias entre os pontos, assim como aplicar o algoritmo Kruskel. Sua *struct* é:

```
typedef struct {
    int indexP1;
    int indexP2;
    double Distancia;
} Distancia;
```

1. indexP1: indica a posição do ponto 1;
2. indexP2: indica a posição do ponto 2;
3. Distancia: é armazenado a distância entre o ponto 1 e o ponto 2;

O objetivo dessa estrutura foi armazenar as informações relevantes para: calcular a distância entre dois pontos, realizar o algoritmo de kruskel, realizar as conexões utilizando os métodos de *union find*, vistos em aula.

- Ponto.h

Esse TAD contém o código necessário para armazenar as informações referentes a cada ponto individualmente. Sua *struct* é:

```
typedef struct {
    int* ID;
    int* PontoPai;
    char* Nome;
    double* Coordenadas;
    int Tamanho;
    int Exibido;
} Ponto;
```

1. ID: refere-se a posição que o ponto se encontra no vetor de pontos, visto na estrutura do Arquivo.h;
2. PontoPai: armazena o ponteiro do ID do ponto Pai (equivale ao pai de um nó de uma árvore);
3. Nome: armazena o nome do ponto, passado no arquivo de entrada;
4. Coordenadas: lista de valores reais contendo as coordenadas do ponto no espaço;
5. Tamanho: refere-se a quantidade de pontos que são filhos do ponto em questão;

6. Exibido: variável utilizada no momento da geração do arquivo de saída do algoritmo. Essa variável indica se o ponto já foi utilizado no arquivo de saída, evitando assim duplicidade;

O objetivo dessa estrutura foi armazenar as informações individuais de cada ponto, assim como auxiliar na geração do arquivo de saída.

3. ANÁLISE DE COMPLEXIDADE

A complexidade de um algoritmo refere-se à quantidade de tempo ou espaço necessários para executar um programa. A seguir, será analisado a complexidade de cada parte do código fornecido:

- Leitura:

A complexidade da leitura depende do tamanho do arquivo de entrada. A função getline() tem complexidade O(n), onde n é o número de caracteres na linha. O loop while executa enquanto há linhas para ler no arquivo. A função Contar_Dimensao() tem complexidade O(n), onde n é o número de caracteres na linha. A alocação de memória de Coordenadas tem complexidade O(Dimensao), onde Dimensao é o número de dimensões dos pontos. A alocação de memória de P tem complexidade O(Contagem), onde Contagem é o número de pontos lidos. O *loop* que faz o parsing da linha tem complexidade O(n), onde n é o número de caracteres na linha. Portanto, a complexidade da leitura é O(N*M), onde N é o número de linhas no arquivo de entrada e M é o número de caracteres em cada linha.

- Cálculo de Distância:

O *loop* externo vai de 1 até Contagem - 1, enquanto o *loop* interno vai de zero até o *index* atual do loop externo. Dessa forma, considerando que o total de distâncias possíveis entre N pontos é de (N * (N - 1)) / 2, pode-se assumir que a complexidade do cálculo de distâncias é de $O(n^2)$, sendo assim o principal gargalo do algoritmo.

- Kruskal:

O *loop* tem complexidade O(QuantD). A função Conectado() tem complexidade O(Dimensao). A função Uniao() tem complexidade O(1). Portanto, a complexidade do algoritmo de Kruskal é O(QuantD*Dimensao).

- Impressão:

O loop externo tem complexidade O(Contagem). O loop interno tem complexidade O(Contagem). A função Procura() tem complexidade O(Dimensao). A impressão de cada nome de ponto tem complexidade O(1). Portanto, a complexidade da impressão é O(Contagem² * Dimensao). Em resumo, a complexidade total do programa é dominada pelo cálculo de distância, que é O(Contagem²).

4. ANÁLISE EMPÍRICA

Com base na análise de complexidade, podemos fazer algumas considerações em relação à tabela de porcentagens:

Arquivos de entrada	1.txt	2.txt	3.txt	4.txt	5.txt
Ler	0,02	0,021	0,056	0,193	0,649
Cálcular distâncias	0,005	0,02	1,057	17,444	132,647
Organizar distâncias	0,006	0,01	0,999	7,732	29,527
Kruskel	0,003	0,003	0,034	0,195	0,379
Organizar pontos	0,003	0,003	0,007	0,014	0,029
Imprimir resultado	0,004	0,008	0,127	0,704	3,132
TOTAL	0,041	0,065	2,28	26,282	166,363

Pode-se observar que, como foi dito no tópico anterior, o principal causador de demora na execução do algoritmo é a etapa de calcular distâncias, pois se trata de uma complexidade O(n²).