Introdução

O presente relatório tem por finalidade realizar uma breve descrição dos métodos e estruturas utilizados na implementação do algoritmo de agrupamento utilizando uma MST, além de uma análise de suas consequências para o desempenho do programa.
Dessa forma, o documento conta com uma análise do gasto de memória das estruturas, além da complexibilidade das partes principais do código e do tempo gasto, de acordo com a entrada, para cada parte dessas.
Ademais, com o uso de um grupo de entradas de teste, e saídas para comparação, foi possível verificar a eficácia do algoritmo, e que este funciona adequadamente de acordo com o esperado, solucionando o problema de agrupamento até certo ponto, de acordo com o proposto que é uma análise de uma variante específica e bem definidas do problema.

Metodologia

Foram utilizadas apenas duas estruturas na composição do trabalho, sendo a MST imprescindível para o algoritmo, a outra única estrutura chamada Vetor foi criada para facilitar a leitura e armazenamento dos pontos, além da sua utilização na MST, já que o id de cada ponto poderia ser o mesmo para ambas as estruturas. As outras estruturas utilizadas foram básicas, como a matriz para armazenar as distâncias e o id dos vetores aos quais as distâncias pertencem, para facilitar no momento de fazer a união.
A MST foi feita utilizando o quick union com compressão, tornando a estrutura mais rápida e eficaz nos momentos de executar a união, além de comprimida facilitando o encontro do pai para o print dos grupos.
As decisões de algoritmo foram feitas seguindo os algoritmos de kruskal e o disponibilizado nas especificações, sendo poucas as decisões tomadas sem o amparo de um destes dois, como por exemplo, a forma escolhida para mostrar os resultados.
Dessa forma, os vetores foram organizados lexicograficamente logo após a leitura, e para a amostragem dos resultados os grupos foram todos salvos na primeira passagem ao vetor, no momento da amostragem, facilitando a identificação destes para o acesso futuro.

Análise de complexibilidade

Para a análise de complexibilidade vamos dividir o código em cinco partes principais, e analisar separadamente cada uma, sendo elas a leitura dos dados, o cálculo das distâncias, a ordenação das distâncias, a obtenção da MST e a escrita dos resultados.

- **leitura dos dados:**

Na leitura dos dados temos que considerar a quantidade de iterações feitas, uma vez que a alocação de memória dinâmica tem complexibilidade O(n), e a criação do Vetor também, o principal fator de análise serão os loops .
Dessa forma, o loop externo será executado de acordo com a quantidade de pontos, enquanto o loop interno de acordo com a quantidade de coordenadas, nos dando uma complexibilidade O(pontos * coordenadas).

- **cálculo das distâncias:**
  Com relação ao cálculo das distâncias, levamos em consideração a quantidade de distâncias a serem calculadas, que será dada sempre pela (quantidade de vetores * (quantidade de vetores -1))/2. Essa fórmula foi utilizada pois ela nos dá a quantidade de elementos na diagonal da matriz quadrada formada pela quantidade de pontos, que é justamente a quantidade de distâncias.
  No trecho do cálculo de distância, temos dois loops que iteram sobre todos os pontos, entretanto, a condição só permite que sejam calculadas as distâncias da diagonal inferior, dessa forma não levaremos as informações dos loop na computação da complexibilidade.
  Assim, levando em consideração apenas quando a função de cálculo é chamada, teremos também um loop nesta, que ocorrerá de acordo com a quantidade de coordenadas. Como as outras funções usadas no cálculo não influenciam significativamente, pois tem complexibilidade O(1), ficamos com uma complexibilidade O(distancias * quantidade de coordenadas).

- **Ordenação das distâncias:**
  A ordenação das distâncias foi feita utilizando apenas o qsort da biblioteca stdlib.h, e que possui complexibilidade O(N^2) no pior caso.

- **Obtenção da mst:**
  Para a obtenção da mst e criação dos grupos, que são feitas ao mesmo tempo, vamos olhar primeiro para a função de união do quick union. Esta função por sua vez tem sua complexibilidade ditada pela função de encontrar o pai dos pontos dados, que depende da profundidade de cada árvore, que como já visto em sala, possui tamanho máximo de log N. Graças a isso, no pior caso, encontrar o pai tem complexibilidade O(log N), e dessa forma a função de união também.
  Por fim, temos o loop que dita quantas vezes a união é feita de acordo com a quantidade de ligações mínimas necessárias para se ter a quantidade de grupos desejada, e por isso a condição de parada é a quantidade de pontos - a quantidade de grupos desejada, que será a quantidade de aresta. Temos então que a obtenção da MST e dos grupos possuem complexibilidade O(arestas * log N).

- **escrita dos resultados:**

A escrita dos resultados é feita pensando na quantidade de grupo e dos elementos de cada grupo, por isso o for mais externo cobre os grupos enquanto o interno cobre todos os pontos, deixando uma complexibilidade O(grupos * pontos). Entretanto como para printar o vetor é necessário procurar o pai, e esta operação também exige um loop, é necessário levá-la em consideração, e como visto no tópico anterior ela é O(log n), ficamos então com O(grupos * pontos * log n).
Existem outras operações de loop durante a escrita, mas como elas acontecem apenas uma vez, não é tão relevante numa análise mais geral.

Análise empírica:

Após a medição de tempo de execução das principais partes do algoritmo citadas no tópico anterior, podemos analisar não só o tempo que o programa leva para ser executado, mas quais as partes que estão comprometendo o seu desempenho, de forma mais concreta.
Foram usadas 4 entradas para as medições, estas possuem as seguintes especificações:

- Entrada 1: 100 pontos, 4 grupos - execução total: 0,001360 s.
- Entrada 2: 1000 pontos, 5 grupos - execução total: 0,275764 s.
- Entrada 3: 2500 pontos, 5 grupos - execução total: 1,973686 s.
- Entrada 4: 5000 pontos, 10 grupos - execução total: 9,496213 s.

Na tabela abaixo podemos ver a porcentagem de tempo ocupada pelas principais partes do código levando em conta o tempo total.

partes do código / Entradas	leitura dos dados	cálculo das distâncias	Ordenação das distâncias	Obtenção da MST e grupos	escrita
entrada 1	7,5%	8,6%	49,4%	0,66%	0,95%
entrada 2	0,19%	1,61%	51,6%	1,66%	0,09%
entrada 3	0,12%	1,31%	58,21%	0,9%	0,03%
entrada 4	0,1%	2,44%	59%	0,44%	0,017%

Analisando a tabela vemos que a análise empírica bate com a de complexibilidade, já que as partes que foram analisadas como tendo maior complexibilidade, como a ordenação de distâncias(O(N^2)), estão crescendo mais percentualmente de acordo com a entrada, enquanto as outras vão perdendo espaço por terem um complexibilidade bem menor.