TÉCNICAS DE BUSCA E ORDENAÇÃO
TRABALHO 1
PROBLEMA DE AGRUPAMENTO

Erico Guedes

Ezequiel Demetras

INTRODUÇÃO:

O agrupamento, também conhecido como clustering, é uma tarefa importante em aprendizado não supervisionado que visa encontrar grupos naturais em uma base de dados. O objetivo é agrupar objetos semelhantes e separar objetos não semelhantes em diferentes grupos. No entanto, como a similaridade é subjetiva e varia de acordo com o domínio do problema, existem diversos algoritmos de agrupamento na literatura, cada um com suas próprias características. Neste trabalho, focaremos em uma variante específica do problema de agrupamento, que envolve a geração de uma árvore geradora mínima usando o algoritmo de agrupamento de espac¸amento máximo, que utiliza uma MST (Minimum Spanning Tree) para agrupar pontos com distâncias similares. Iremos considerar pontos no espaço real bidimensional, com a distância euclidiana como medida de dissimilaridade entre eles. Este trabalho é relevante por apresentar uma solução para um problema específico de agrupamento, que pode ser útil em diferentes áreas do conhecimento.

METODOLOGIA:

A metodologia escolhida para implementar o algoritmo de agrupamento de espac¸amento máximo, utilizando uma MST, envolveu diversas etapas. Primeiramente, utilizamos a função getline() para ler a entrada de arquivos, juntamente com a função strtok() para quebrar as informações obtidas e descobrir a dimensão

dos pontos no espaço. Para armazenar os pontos do espaço, criamos um vetor com alocação de memória dinâmica.

Em seguida, para o cálculo de distâncias, utilizamos um loop duplo, com foco em eliminar cálculos repetidos. Para organizar o vetor de distâncias, utilizamos a função qsort(), conforme especificado no trabalho. O cálculo das distâncias foi realizado com base na distância euclidiana entre os pontos no espaço, utilizando suas coordenadas.

Após o cálculo das distâncias, utilizamos o algoritmo de Kruskal para gerar a árvore geradora mínima, considerando o inteiro K e utilizando a técnica de weighted quick-union. Finalmente, a função Imprime foi criada para gerar a saída seguindo as especificações do trabalho.

Essa metodologia foi escolhida por ser simples de executar e atender às especificações do trabalho, permitindo uma implementação eficiente do algoritmo de agrupamento de espac¸amento máximo utilizando uma MST. Além disso, as funções utilizadas, como getline(), strtok() e qsort(), são amplamente disponíveis em bibliotecas padrão de C/C++, o que facilita a implementação e a portabilidade do código.

COMPLEXIDADE:

A complexidade de um algoritmo refere-se à quantidade de tempo ou espaço necessários para executar um programa. A seguir, analisaremos 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 for 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.

Calculo de Distancia:

O loop externo tem complexidade O(Contagem), enquanto o loop interno tem complexidade O(i). Dentro do loop interno, Adiciona_Distancia() tem complexidade O(Dimensao). Portanto, a complexidade do cálculo de distância é O(Contagem^2 * Dimensao).

Kruskal:

O loop for 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(Contagem^2 * Dimensao).

Em resumo, a complexidade total do programa é dominada pelo cálculo de distância e é O(Contagem^2 * Dimensao).

ANÁLISE:

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

Leitura: A porcentagem de tempo gasto na leitura dos dados é bastante baixa, apenas 0,07% do tempo total. Isso indica que essa etapa é muito eficiente e não deve ser um gargalo para a execução do programa.

Cálculo e ordenação das distâncias: Essa etapa consome a maior parte do tempo, cerca de 98,3% do tempo total. Isso era esperado, já que o cálculo das distâncias envolve dois loops aninhados e a ordenação dessas distâncias também tem complexidade O(n^2 log n).

Obtenção e identificação dos grupos: Essa etapa consome uma porcentagem muito baixa do tempo total, apenas 0,07%. Isso indica que o algoritmo utilizado para obter a MST é eficiente e não gera gargalos significativos.

Escrita: A etapa de escrita do arquivo de saída consome uma porcentagem baixa do tempo total, cerca de 1,07%. Isso indica que essa etapa é eficiente e não deve ser um gargalo significativo.

Com base nessas porcentagens, podemos concluir que as medições estão de acordo com a análise de complexidade, já que a

etapa mais complexa (cálculo e ordenação das distâncias) consome a maior parte do tempo.