Universidade de São Paulo - USP
Instituto de Ciências Matemáticas e de Computação - ICMC
Departamento de Ciências de Computação
SCC-503 - Algoritmos e Estruturas de Dados II
Prof. Dr. Gustavo E. de Almeida Prado Batista

Projeto - Árvores Geradoras Mínimas

Andrey Gobbo 7152503
Gabriel Fernandes Gonzales 8936565
Marcos Adriano de Carvalho 6791727

São Carlos 2017

Introdução

O seguinte projeto consistiu na implementação de algoritmos de agrupamento de dados utilizando árvores geradoras mínimas.

Dois algoritmos foram usados para o desenvolvimento do trabalho, começando pelo Algoritmo de Prim, que teve sua implementação feita através de uma fila de prioridades com uma heap binária (árvore binária que armazena chaves em seus nós).

O algoritmo percorre os vértices do grafo, faz uma verificação de determinado vértice já foi verificado, percorre os vizinhos e procura o de menor peso. Após finalizar todos os processos, retorna um vetor dos "pais" dos vetores, que depois é usado para reconstruir as arestas, finalizando a árvore geradora mínima.

Algoritmo de Kruskal, onde foi implentada uma estrutura Union-Find, que mantém subconjutos de elementos mutuamente exclusivos.

Para ambos os algoritmos, o usuário entrará com um valor k (inteiro) que é o número de classes que ele deseja que o algoritmo agrupe os dados do arquivo data.txt.

Estrutura de Dados e funções

Neste trabalho utilizamos uma estrutura de dados simples, composta de duas *structs:*

- struct Aresta - composta por dois valores inteiros, sendo eles os vertices de origem e destino, e um valor *float* para o peso (distância euclidiana entre os pontos)

- *struct Grafo* - composta por dois numeros int V e A, sendo os números de vértices e arestas, respectivamente, e um vetor de structs do tipo Aresta.

Como principais funções temos:

- criaGrafo - recebe dois numeros inteiros V e A e aloca memória para o grafo de V vértices e A arestas.

- Kruskal - executa o algoritmo de Kruskal conforme descrito. Recebe um ponteiro para o grafo e um número inteiro (número máximo de classes). A saída do algoritmo é dada em um arquivo .txt.

- Prim - executa o algoritmo de Prim. Retorna a refência de um vetor de pais da heap binária.

- find e Union - funções auxiliares para o algoritmo Kruskal para a criação de subconjuntos de pontos.

- calculaDistanciaEuclidiana - recebe quatro numeros do tipo *float,* sendo eles as coordenadas x e y de dois pontos e retorna um numero *float* dado pela distância Euclidiana dos dois pontos.

Funcionamento do código

A função main do nosso programa lê os dados do arquivo de entrada “data.txt” e salva os valores em dois vetores, representando as coordenadas x e y dos pontos.

Então, executamos a função calculaDistanciaEuclidiana entre todos os pontos e salvamos em uma matriz quadrada para construirmos o grafo em seguida.

Construimos o grafo utilizando os valores da matriz como sendo o peso de uma aresta. Como queremos construir um grafo completo, consideramos que os valores utilizados são apenas da matriz diagonal superior da matriz de distâncias, para que não criemos arestas repetidas.

Após isso executamos o algoritmo de Kruskal e Prim para o grafo montado.

Para compilar e executar o programa:

- compilar: gcc main.c grafo.c -o trabalhoAlg -lm -std=c99
- executar: ./trabalhoAlg
- o arquivo “data.txt” deve estar no mesmo diretório do executável.

Discussão e Resultados Obtidos

Para o Algoritmo de Kruskal, há uma limitação no número de classes geradas para o conjunto de dados inseridos, obtendo-se no máximo seis classes. Apesar disso, o algoritmo limita o numero total de classes de acordo com o numero escolhido no momento de execução do programa.

A partir dos dados obtidos, vimos que cerca de 75% dos pontos pertencem a classes similares às originais, apesar de não conseguirmos dividir os grupo de dados em sete classes, como inicialmente.

Já para o Algoritmo de Prim, não conseguimos chegar a uma solução eficiente e funcional, devido à simplicidade da estrutura de dados escolhida. Essa estrutura facilitou a implementação do algoritmo de Kruskal, uma vez que não precisávamos ter estruturas de dados para os vértices, mas dificultou a implementação de Prim.

Sendo assim, concluimos que o planejamento é importante para que não ocorra divergências entre o resultado esperado e as especificações fornecidas.