Relatório - LAB 5 a LAB 8

Identificação: Pedro Mesquita Maia – 2312664

Objetivo:
O objetivo desses laboratórios foi implementar e testar diferentes formas de representar grafos, além de aplicar algoritmos para encontrar a árvore geradora mínima (MST) de um grafo. Ao longo dos LAB 5 a LAB 8, fomos desafiados a trabalhar com representações de grafos, implementar o algoritmo de busca em largura (BFS) e testar os algoritmos de Prim e Kruskal para obter a MST.

No LAB 5, o desafio foi representar um grafo de duas maneiras: como lista de adjacências e como matriz de adjacências. A lista de adjacências é mais eficiente em termos de espaço, pois apenas armazena os vértices vizinhos, sendo útil quando o grafo é esparso. Já a matriz de adjacências utiliza uma estrutura 2D para indicar se existe uma aresta entre dois vértices, o que pode ser mais eficiente quando o grafo é denso. Ambas as representações foram implementadas e testadas com um grafo simples de 5 vértices e 7 arestas, e funcionaram bem em termos de clareza e facilidade de manipulação.

No LAB 6, implementamos a busca em largura (BFS), um algoritmo clássico para explorar grafos. O objetivo era percorrer o grafo começando pelo vértice 0 e visitar todos os vértices vizinhos de maneira nível a nível. Utilizamos uma fila para garantir que os vértices fossem processados na ordem em que foram descobertos. A BFS foi eficiente e conseguiu explorar todo o grafo corretamente.

No LAB 7, usamos o algoritmo de Prim para calcular a árvore geradora mínima (MST). O algoritmo começa com um vértice e, a cada passo, adiciona a aresta de menor peso que conecta um vértice da MST a um vértice fora dela. Para garantir que a aresta de menor peso fosse escolhida, usamos uma fila de prioridade. A árvore geradora mínima foi exibida corretamente, mostrando as arestas de menor custo que conectam todos os vértices, e conseguimos calcular o custo total da MST.

No LAB 8, implementamos o algoritmo de Kruskal, que é outra maneira de encontrar a árvore geradora mínima. Diferente do Prim, o Kruskal trabalha diretamente com as arestas do grafo. Ele ordena as arestas por peso e, em seguida, vai adicionando as mais leves à MST, desde que não formem ciclos. Para isso, usamos uma estrutura de dados chamada conjunto disjunto, que facilita verificar se a adição de uma aresta vai formar um ciclo. A árvore geradora mínima foi gerada corretamente e exibimos o custo total das arestas que formam a MST.

Ao longo de todos os laboratórios, tive a oportunidade de entender melhor como os grafos funcionam e como diferentes algoritmos podem ser aplicados para resolver problemas práticos, como encontrar a árvore geradora mínima. O uso de estruturas de dados como filas de prioridade e conjuntos disjuntos foi essencial para que os algoritmos funcionassem corretamente e de maneira eficiente. Além disso, a implementação de cada algoritmo foi uma ótima oportunidade para melhorar minhas habilidades de programação e entender melhor as diferenças entre os métodos de Prim e Kruskal.

O código foi compilado com o comando gcc -o kruskal kruskal.c e gcc -o buscaAmplitude buscaAmplitude.c, os outros labs estão nas imagens. Após os testes, todas as funcionalidades foram confirmadas como corretas. Este trabalho eu achei mais tranquilo que o último (de árvore b+), e também ajudou a revisar os conceitos para a prova.