Trabalho Prático 1
Ruas e Áreas Verdes de Somatório

Davi Santos Rodrigues - 2022043752

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

davisrodrigues@ufmg.br

1. Introdução

Este trabalho aborda a verificação de uma malha de ruas da infraestrutura de uma cidade fictícia. Para resolver os 3 problemas propostos é necessário implementar algoritmos robustos e eficientes, pois o volume de dados pode aumentar a complexidade rapidamente.

Assim, busca aprofundar o conhecimento de grafos, desde a implementação até os seus algoritmos mais famosos e clássicos, como Dijkstra e Tarjan.

2. Modelagem

O programa foi desenvolvido na linguagem C++(11), compilada pelo compilador G++ da GNU Compiler Collection. Eu modelei as regiões como os vértices e coloquei atributos em cada vértice visando futuramente facilitar a implementação dos algoritmos. As arestas são não direcionadas, mas ponderadas e representam as ruas do cenário.

2.1. Estrutura de Dados

. As estruturas de dados utilizadas no projeto são:

- Vector: Um TAD nativo da STL do C++, muito prático, pois possui métodos já implementados e gerencia sua própria alocação de memória.

- Min-Heap: Uma fila de prioridade que seleciona sempre a menor estimativa de distância dos vértices, se auto-balanceia nos processos de inserção e remoção.

2.2. Algoritmos

Os algoritmos utilizados foram Dijkstra, menor caminho em um grafo ponderado não negativamente e Tarjan para encontrar pontes no grafo.

3. Solução

3.1. Problema 1

A solução do problema 1 era uma aplicação clássica do algoritmo de Dijkstra: encontrar o menor caminho entre dois vértices em um grafo ponderado não negativamente. Dijkstra é um algoritmo guloso, o que quer dizer que ele sempre toma a melhor decisão em cada passo. A melhor decisão, nesse caso, é escolher o próximo vértice com menor estimativa de distância no momento e adicionar ele ao conjunto dos

vértices já visitados, por isso a fila de prioridade é imprescindível para o funcionamento do algoritmo, e dessa forma ele consegue achar a distância de um vértice para qualquer outro do grafo.

3.2. Problema 2

Para resolver o problema 2 utilizei uma propriedade matemática dos grafos. Se uma aresta (u, v, w), participa de algum caminho mínimo entre o vértice X e o vértice Y, a distância de X para u acrescida da distância de Y para v e de w deve ser a distância de X para Y. Logicamente, pois se fosse maior, não seria um caminho mínimo e por contradição não poderia ser menor.

3.3. Problema 3

Uma vez possuindo a lista de arestas que participam de algum caminho mínimo, é evidente que as críticas, que participam de todos os caminhos mínimos, estão no conjunto. Desenhando os grafos dos exemplos no paint, percebi que poderia apagar as arestas que não estão em nenhum caminho mínimo e notei que a diferença entre uma aresta crítica e uma não era que a aresta crítica, quando removida, desconectava o grafo, e que se encaixavam na definição de ponte.

Modelei então o subgrafo com as arestas mínimas e procurei um algoritmo que encontrasse pontes em grafos e me deparei com o algoritmo de Tarjan. O algoritmo de Tarjan funciona com a execução de uma DFS a partir de qualquer ponto, ele computa o tempo de início de cada vértice e sempre que ocorre o backtrack ele testa se essa aresta é uma ponte, para fazer isso ele usa outra medida de tempo para cada vértice que funciona como o tempo de término, porém, ao chegar em um nó já visitado, pode-se pegar o tempo de término dele caso seja menor que o atual, durante o teste, ele compara se o tempo de término do que chamou é menor que o do que foi chamado, se for a aresta é uma ponte.

4. Análise de Complexidade

Problema 1:

Para resolver o problema 1, foi usado uma execução do algoritmo de Dijkstra, portanto, segundo os slides e o livro, Dijkstra nesse caso é $O((N + M) \log N)$ em complexidade de tempo. Em memória, o Grafo em lista de adjacência é $O(M+N)$, os vetores usados: dist, visitados, Min-Heap; são todos $O(N)$, logo a complexidade de espaço final é $O(M+N)$.

Problema 2:

Para resolver o problema 2, foi usado mais uma execução do algoritmo de Dijkstra e depois, para cada aresta processei a equação matemática para descobrir se eram mínimas ou não. Logo, $O((N+M) \log N) + O(M) = O((N+M) \log N)$. Em espaço, foram criados vetores $O(M)$, portanto o custo do grafo domina: $O(N+M)$.

Problema 3:

Para resolver o problema 3, criei um subgrafo, que no pior dos casos é $O(M + N)$ para preencher, e o algoritmo de Tarjan acha todas as pontes em uma única DFS, que é $O(V + A)$. Dessa forma, o custo em tempo do problema 3 é $O(N+M)$. Em espaço, criar o subgrafo, no pior caso, é $O(M+N)$, e os vetores auxiliares para a solução são $O(N)$, porém é importante apontar que a DFS ocupa espaço na pilha de recursão que é de acordo com seu nó mais profundo, delimitado por $O(N)$, portanto o custo de memória é $O(M+N)$, mas $O(N)$ na pilha de recursão.

5. Considerações Finais

O desenvolvimento deste trabalho prático possibilitou o projeto, a implementação e a solução de problemas clássicos sobre grafos. Foi muito interessante revisitar a implementação dessa estrutura de dados, agora com um pouco mais de liberdade, ficou mais fácil construir as listas de adjacência. O mais desafiador foi pensar na solução dos problemas, mas nada impossível, pois muitos problemas de grafos possuem soluções elegantes propostas por pessoas muito inteligentes. Acredito que foi importante para o desenvolvimento do conhecimento em grafos e seus algoritmos.

6. Referências

Ziviani, N. (2006). *Projetos de Algoritmos com Implementações em Java e C++: Capítulo 3: Estruturas de Dados Básicas*. Editora Cengage.

SEDGEWICK, Robert; WAYNE, Kevin. Algorithms. 4. ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2011.

https://www.youtube.com/watch?v=qrAub5z8FeA, canal “take U forward”, vídeo: “G-55. Bridges in Graph - Using Tarjan's Algorithm of time in and low time”.