Trabalho Prático 1 - Algoritmos 1

João Henrique Alves Martins

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

jalvesmartins16@ufmg.br

1 Introdução

Grande parte dos problemas da vida real pode ser modelada através de soluções computacionais, e os grafos são uma das estruturas que abrangem o maior número de problemas, o que evidencia sua importância. Para este trabalho, foi proposta a modelagem de três problemas de mobilidade urbana por meio de grafos e seus algoritmos, envolvendo análise de rotas mais curtas e identificação de potenciais pontos críticos no trânsito da cidade fictícia de Somatório. A solução resultaria na implementação de áreas verdes com menor impacto no trânsito.

2 Modelagem

O programa foi implementado na linguagem Python. Essa linguagem foi escolhida, ao invés de C ou C++, pois já havia implementado essas estruturas nessas linguagens, e quis aproveitar a oportunidade para aprender e desenvolver minhas habilidades em Python. Além disso, as estruturas de dados nativas do Python e a ausência de necessidade de gerir a memória manulamente facilitaram muito a execução do projeto.

Cada problema apresentou sua individualidade na solução, mas aproveitaram em comum a modelagem da classe Grafo e a implementação do algoritmo de Dijkstra. O grafo foi modelado de forma que as regiões fossem os vértices e as ruas conectando essas regiões fossem as arestas. Essa modelagem é comum em aplicativos como o Waze ou em problemas que envolvem caminhos ou labirintos.

O grafo foi implementado com uma lista de adjacências, estrutura escolhida por sua eficiência em grafos esparsos, que é o caso típico de malhas viárias urbanas. Cada vértice mantém uma lista de arestas incidentes, e cada aresta é representada pela classe Edge, que armazena seu "id"(Incomum, porém necessário para a solução do problema), o vértice de destino e o peso (distância).

Para representar o grafo não-direcionado das ruas, cada aresta é adicionada bidirecionalmente na lista de adjacências. Além da lista de adjacências, foi mantido um dicionário de arestas que mapeia cada identificador para seus vértices extremos e peso, facilitando operações de busca de arestas específicas.

O algoritmo de Dijkstra foi implementado utilizando o min-heap da biblioteca heapq do Python, garantindo complexidade O(E log V), onde V é o número de vértices e E é o número de arestas. Esse algoritmo calcula as distâncias mínimas de um vértice origem para todos os outros vértices do grafo.

3 Solução

A solução implementada baseia-se em três algoritmos que trabalham sobre a estrutura de grafo modelada. Todos compartilham o uso do algoritmo de Dijkstra como base para calcular distâncias mínimas,

mas cada um adiciona técnicas específicas para resolver seu problema particular. A seguir, são apresentadas as ideias gerais de cada algoritmo.

3.1 Problema 1: Distância Mínima

O primeiro problema consiste em encontrar a menor distância entre os vértices 1 e N, representando o caminho mais curto entre a praça central e o parque ecológico.

A solução utiliza diretamente o algoritmo de Dijkstra, que opera da seguinte forma: inicialmente, todas as distâncias são marcadas como infinito, exceto a do vértice de origem (vértice 1), que recebe distância zero. Um min-heap mantém os vértices a serem processados, sempre escolhendo aquele com menor distância acumulada.

Para cada vértice retirado do heap, o algoritmo relaxa suas arestas, ou seja, verifica se é possível chegar aos vizinhos com uma distância menor passando por ele. Se sim, a distância do vizinho é atualizada e ele é inserido no heap. O processo continua até que todos os vértices alcançáveis sejam processados.

Ao final, a distância armazenada para o vértice N representa o caminho mais curto entre 1 e N. Essa é a resposta final do primeiro problema. Esse problema foi bem simples de se resolver.

3.2 Problema 2: Arestas em Caminhos Mínimos

O segundo problema busca identificar todas as arestas que participam de pelo menos um caminho mínimo entre os vértices 1 e N.

A solução executa o algoritmo de Dijkstra duas vezes: uma partindo do vértice 1 e outra partindo do vértice N. Com isso, obtém-se a distância mínima de 1 para todos os vértices e de N para todos os vértices.

Para verificar se uma aresta (u, v) pertence a algum caminho mínimo, verifica-se a seguinte condição: a distância de 1 até u, somada ao peso da aresta, somada à distância de v até N deve ser igual à distância mínima entre 1 e N. Como o grafo é não-direcionado, essa verificação precisa ser feita nas duas direções, ou seja, tanto para u $\rightarrow$ v quanto para v $\rightarrow$ u.

Todas as arestas que satisfazem essa condição em pelo menos uma direção são candidatas a receber áreas verdes sem aumentar a distância mínima. A função que resolve esse problema retorna uma lista com esses elementos, que correspondem a solução do problema 2.

3.3 Problema 3: Arestas Críticas

O último e mais complexo problema identifica as arestas críticas, isto é, aquelas que aparecem em todos os caminhos mínimos entre 1 e N. Se uma aresta crítica for removida, necessariamente a distância mínima entre 1 e N aumentará.

Esse problema foi consideravelvemente desafiador, visto que não vimos um algoritmo exato que soluciona esse problema nas aulas. Além disso, foi imposto um limite de complexidade $O(V^2)$.

A solução implementada começa identificando as arestas candidatas através do algoritmo do Problema 2, para otimizar o número de vértices verificados. Em seguida, é necessário contar quantos caminhos mínimos existem entre 1 e N e quantos desses caminhos passam por cada aresta candidata.

A contagem de caminhos é feita após executar o Dijkstra, os vértices são processados em ordem crescente de distância. Para cada vértice, o número de caminhos mínimos até ele é calculado somando os caminhos dos vértices predecessores que estão a uma aresta de distância mínima dele. O vértice de origem inicia com 1 caminho.

Para verificar se uma aresta é crítica, calcula-se quantos caminhos mínimos passam por ela. Isso é feito multiplicando o número de caminhos de 1 até uma extremidade da aresta pelo número de

caminhos da outra extremidade até N. Se esse produto for igual ao número total de caminhos mínimos entre 1 e N, então todos os caminhos obrigatoriamente passam por essa aresta, tornando-a crítica.

A solução é bem interessante e foi fruto de discussão e pesquisa com alguns colegas para alcançar o a complexidade exigida nas instruções do trabalho. A solução usa conceitos avançados, encontrados no tema de Centralidade de Intermediação (Betweenness Centrality), especialmente no trabalho de Ulrik Brandes. Sem a pesquisa sobre essa aplicação, seria complicado encontrar essa solução ótima.

4 Análise de Complexidade

Nesta seção, é apresentada a análise detalhada da complexidade de tempo dos algoritmos implementados. A notação assintótica utilizada considera V como o número de vértices e E como o número de arestas do grafo.

4.1 Dijkstra

O algoritmo de Dijkstra é a base de todas as soluções implementadas. Sua complexidade depende da estrutura de dados utilizada para a fila de prioridades. Neste trabalho, foi utilizada a implementação de min-heap da biblioteca heapq do Python.

Cada vértice é inserido e removido da fila no máximo uma vez, resultando em O(V log V) operações. Além disso, cada aresta pode causar uma operação de atualização na fila, resultando em O(E log V) operações. Portanto, a complexidade total do Dijkstra é O((V + E) log V).

Em grafos esparsos como o do trabalho, onde E é dominante em relação a V, essa complexidade pode ser simplificada para O(E log V).

O problema 1 apenas roda Dijkstra, portanto, sua complexidade é a mesma do algoritmo.

4.2 Arestas em Caminhos Mínimos

A solução executa o Dijkstra duas vezes: uma partindo do vértice 1 e outra partindo do vértice N. Em seguida, itera sobre todas as E arestas do grafo para verificar a condição de pertencimento a caminhos mínimos.

Portanto, a complexidade do problema 2 é O(E log V + E) = O(E log V).

4.3 Arestas Críticas

A solução executa a solução 2 para separar as arestas em caminhos mínimos, depois roda Dijkstra duas vezes, depois realiza a contagem de caminhos mínimos para a ida e volta, que também roda Dijkstra e itera no pior caso por todas as arestas. Por fim, itera sobre as arestas candidatas para verificar criticidade. No total, roda Dijkstra 6 vezes e itera sobre as arestas 4 vezes. Para grafos grandes com poucos caminhos mínimos, a otimização que utilizei diminui consideravelmente o número de arestas visitadas nas iterações.

Portanto, a complexidade do problema 3 é O(E log V + E) = O(E log V), correspondendo ao pedido nas instruções.

5 Considerações Finais

Este trabalho proporcionou uma experiência prática valiosa na modelagem e resolução de problemas através de grafos. A implementação dos três problemas de mobilidade urbana demonstrou como conceitos teóricos podem ser aplicados em situações cotidianas, como análise de tráfego e planejamento

urbano. É fascinante enxergar como o que aprendemos em sala de aula pode ser usado para solucionar os nossos problemas da vida real.

Além disso, a oportunidade de aprender e implementar um trabalho inteiro em Python foi muito interessante, visto que não temos muitas disciplinas que usam essa linguagem. A linguagem é simples e consideravelmente menos verbosa que o C.

Por fim, a experiência de aplicar e adaptar os algoritmos estudados em sala de aula para obter resultados além do proposto inicialmente foi bastante enriquecedora. O principal desafio do trabalho consistiu em modelar o problema adequadamente e criar variações dos algoritmos clássicos para atender aos requisitos específicos de cada questão. A necessidade de ir além da simples implementação do Dijkstra, adicionando camadas de análise como contagem de caminhos e identificação de criticidade, exigiu compreensão profunda dos conceitos envolvidos. De modo geral, a realização deste trabalho foi uma experiência gratificante que contribuiu significativamente para o aprendizado de algoritmos em grafos e suas aplicações práticas.

6 Referências

Vimieiro, R. (2025). Instruções para a realização do Trabalho Prático 1 de Algoritmos 1. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Vimieiro, R. (2025). Slides virtuais da disciplina de Algoritmos 1. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Brandes, Ulrik. "A Faster Algorithm for Betweenness Centrality."Journal of Mathematical Sociology, vol. 25, no. 2, pp. 163–177, 2001.