Trabalho prático de Grafos: Diâmetro

Exercise 1. Modelagem

Answer 1. Para resolver o problema do diâmetro de um grafo deve-se encontrar os caminhos mínimos considerando todas as fontes. Vários métodos podem ser aplicados, foi escolhido o método de Johnson: Trata-se da execução do algoritmo de Dijkstra para cada vértice do grafo. Ao final de cada execução do algoritmo de Dijkstra identifica-se a maior distância entre vértices, e com a execução para todos encontra-se o diâmetro do grafo.

Escolheu-se como representação do grafo uma matriz de adjacências, pois é simples de implementar, e diminui a variação do tempo de execução e memória. Ou seja, ao identificar que o código executa em menos que $1s$ para um grafo denso com 300 vértices, a execução para um outro grafo qualquer com essa dimensão também terá um tempo similar.

Exercise 2. Análise de complexidade

Answer 2. Toda a implementação pode ser dividida em três partes: inicialização, aplicação do algoritmo de Dijkstra para cada um dos vértices, e finalização (onde é retornado a saída).

Em relação a inicialização da implementação um laço percorre cada linha da entrada. Essa entrada é utilizada para formar a representação do grafo em questão. Como a representação é uma matriz de adjacência, pode-se dizer sem perda de generalidade que a inicialização tem complexidade $\mathcal{O}(|V|^2)$, onde $|V|$ é cardinalidade do número de vértices do grafo.

A complexidade do algoritmo de Dijkstra está diretamente relacionada a forma de representação do grafo e a estrutura de dados utilizada na implementação da fila de prioridades. A inicialização de cada execução do algoritmo de Dijkstra toma tempo $\mathcal{O}(|V|)$, pois é executado uma atribuição para cada vértice da distância mínima ao vértice de origem, e definido o predecessor como nulo.

A representação do grafo foi feita através de uma matriz de adjacência. Isso significa que, para identificar se um vértice u é vizinho de um vértice v, devemos acessar a $matriz[u][v]$ e verificar se a distância é diferente de infinito. Para percorrer todos os vizinhos de u devemos percorrer $matriz[u][1, \ldots, |V|]$. Portanto a complexidade do laço que faz o relaxamento das arestas de u é $\Theta(|V|)$.

Para simular a fila de prioridades foram implementadas as seguintes funções: *menorCustoAtual* com complexidade $\Theta(|V|)$ e *estaVazio* também com complexidade $\Theta(|V|)$. É fácil verificar a complexidade dessas funções, pois são constituídas de um laço percorrendo cada vértice e fazendo algumas operações condicionais e de atribuição cujo tempo é $\mathcal{O}(1)$.

O algoritmo de Dijkstra executa o laço principal uma vez para cada vértice, pois um vértice não volta a entrar na fila de prioridades após ter saído, e a cada iteração apenas um vértice é removido. Portanto para uma execução do algoritmo de Dijkstra toma-se tempo $\mathcal{O}(|V|\,[\theta|V| + \theta|V|])$, desconsiderando constantes $\mathcal{O}(|V|^2)$.

Sabendo a complexidade da execução do método de Dijkstra para um vértice, podemos calcular a complexidade total da implementação assim:

$$\mathcal{O}(|V|^2 + |V| * [complexidade(Dijkstra) + |V|]) \tag{1}$$

Onde $|V|^2$ é devido a inicialização do Grafo, $|V| * complexidade(Dijkstra)$ é devido a execução para vértice do método de Dijkstra, e ao final de cada chamada a Dijkstra verifica-se as distâncias do caminho retornado para encontrar o máximo (por isso adiciona-se $|V|$). Em resumo, a complexidade da implementação pode ser descrita como $\mathcal{O}(|V|^3)$.

A finalização toma tempo $\mathcal{O}(|V|)$ pois um diâmetro (que é um caminho mínimo) pode ter no máximo $|V| - 1$ arestas. Como $\mathcal{O}(|V|) << \mathcal{O}(|V|^3)$ a finalização não impacta na complexidade geral da implementação.