1 Modelagem

Para resolver o problema de encontrar o diâmetro de um grafo G com n vértices e m arestas, foi implementado um programa em C++ que consiste de duas etapas principais: determinar a distância entre todos os pares de vértices do grafo; e, a partir das distâncias computadas, obter o valor do diâmetro de G.

A distância entre os pares de vértices de G foi calculada usando o algoritmo de Floyd-Warshall, apresentado no capítulo 25 de [1]. No programa, são criadas duas matrizes 3D com dimensões $(n + 1)$ x n x n: D e P. A matriz D irá armazenar as distâncias calculadas pelo algoritmo Floyd-Warshall e a matriz P irá armazenar os caminhos computados, sendo que as distâncias e caminhos mínimos estarão representados em $D[n]$ e $P[n]$ após a execução do algoritmo. A matriz $D[0]$ contém a representação inicial do grafo como uma matriz de adjacência, onde cada posição d_{ij} corresponde ao peso da aresta $i \rightarrow j$, se ela existir, a 0, se $i = j$, ou a INF, se ela não existir (onde INF é uma constante bem grande).

Após a leitura da entrada e inicialização das matrizes D e P, a função **calculateDiameter** é chamada, onde o diâmetro será de fato computado. Nesta função, uma outra função **floydWarshall** é invocada para o cálculo das distâncias mínimas entre todos os pares de vértices. **floydWarshall** é uma implementação direta de uma versão modificada do algoritmo Floyd-Warshall, onde além da matriz D, a matriz P, que guarda os caminhos mínimos, também é atualizada ao longo da execução (versão proposta no exercício 25.2-3 de [1]). Após o cálculo das distâncias, a matriz $D[n]$ é percorrida a fim de se obter o valor da distância máxima, que corresponde ao diâmetro do grafo, e dois vértices diametrais, que correspondem aos índices da posição da matriz que contém o diâmetro. Por fim, um caminho mínimo entre esses vértices diametrais é obtido usando a função **getPath**, que também é uma implementação direta do procedimento Print-All-Pairs-Shortest-Path (apresentado no capítulo 25 de [1]), e os resultados são exibidos na saída padrão.

2 Análise de Complexidade

A função **floydWarshall** tem complexidade $O(n^3)$, visto que há três laços *for* encadeados que serão executados n vezes cada um e cada iteração tem custo de $O(1)$ resultando em complexidade $O(n^3)$. A função **getPath** terá complexidade de $O(n)$, visto que um caminho mínimo entre dois vértices terá no máximo $n - 1$ vértices. Assim, a função **calculateDiameter** terá complexidade $O(n^3)$, uma vez que chama **floydWarshall** e as outras etapas têm complexidades $O(n^2)$ (obter a distância máxima) e $O(n)$ (imprimir o caminho mínimo entre os vértices diametrais). As outras funções do programa têm complexidade $O(n^3)$ (alocação de memória) e $O(n^2)$ (liberação de memória, leitura das arestas).

Assim, a complexidade total do programa é $O(n^3)$, onde n é o número de vértices do grafo.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.