Relatório 1º Projeto ASA 2020/2021

Grupo: al024
Alunos: António Coelho (ist195535) e Gustavo Aguiar (ist195587)

1 Descrição do Problema e da Solução

O problema foi interpretado como um de grafos (em específico um DAG), em que as peças de dominós correspondem a vértices e as dependências entre os mesmos a arcos. Concluiu-se que o número de intervenções necessárias para garantir que todas as peças caiam é dado pelo menor número de vértices a partir dos quais se pode percorrer todo o grafo, i.e., a quantidade de fontes[1] no DAG. Do mesmo modo, determinou-se que o tamanho da maior sequência de peças a caírem representa o número de vértices no caminho mais longo do DAG. Assim, como auxílio de resolução dos problemas encontrados utilizaram-se as seguintes referências:

- Algoritmo de Kahn para ordenação topológica num DAG
- Artigo de Mumit Khan, CSE 211: Descobrir o caminho mais longo num DAG

2 Análise Teórica

A representação do DAG em memória foi feita com recurso a uma lista de adjacências - complexidade de espaço $\Theta(V+E)$ - assumindo que o grafo não apresentava uma densidade elevada, fruto da natureza real do problema. Assim, dada a representação em lista de adjacências e com vista ao desempenho, procurou-se dar uso a algoritmos que corressem em tempo linear de acordo com o tamanho do grafo[2] - $O(V+E)$.

A solução visada para a resolução dos problemas apresentados consiste em 3 etapas: ler o DAG de *input* (1), ordenar topologicamente o DAG (2) e para cada vértice em ordem topologicamente linear percorrer a sua lista de adjacências a fim de encontrar o maior caminho no DAG (3).

Para (1) - leitura de *input* - usando `scanf`, ler os dados de entrada dentro de um ciclo a depender linearmente de E, i.e., $O(E)$.

Para (2) - ordenar topologicamente o DAG - percorrer os vértices do DAG adicionando a uma *stack* os de grau de entrada zero. Para cada um desses adicioná-lo

[1] Uma fonte é um vértice com grau de entrada zero.
[2] Tamanho do grafo é dado por $|V|+|E|$.

à ordenação topológica e visitar os seus adjacentes; decrementar em 1 valor o grau de entrada dos adjacentes (remover o arco do DAG) e adicioná-los à *stack* se ficarem com grau de entrada 0. Repetir o processo para cada vértice na stack. Logo, $O(V) + O(V + E) = O(2V + E) = O(V + E)$.

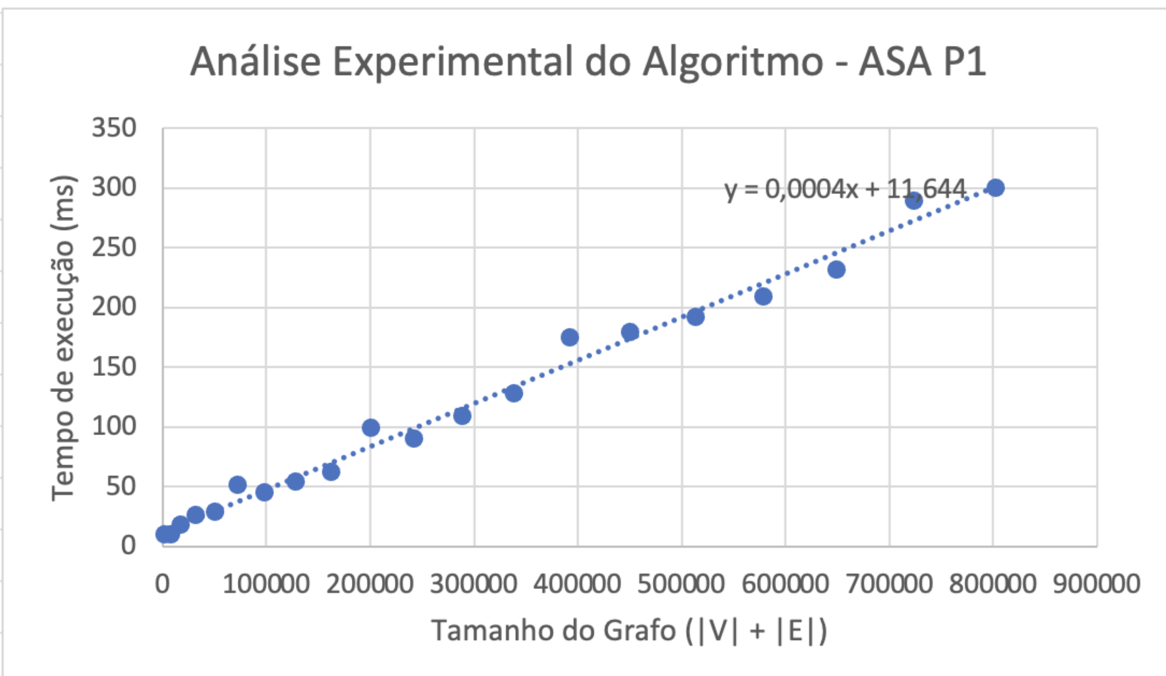
Para (3) - encontrar o maior caminho do DAG - para cada vértice $v \in V$ em ordenação linear calcular dist(v)=$max_{(u,v)\in E}$\{dist(u)+1\} e devolver $max_{(v)\in V}$\{dist(v)\}, onde dist(v) representa a maior distância entre uma fonte do DAG ao vértice v. Logo, $O(V + E) + O(V + E) = O(2V + 2E) = O(V + E)$.

Conclui-se que a complexidade geral da solução aos problemas é dada por $O(E) + O(V) + O(V + E) + O(V + E) = O(3V + 3E) = \mathbf{O(V{+}E)}$.

3 Avaliação Experimental dos Resultados

Para a devida análise do algoritmo implementado, utilizou-se um computador com processador *Intel Core i5 Quad-Core* a 1.4 GHz, com 8 GB de memória RAM, com o sistema operativo *macOS Big Sur*.

Utilizou-se a ferramenta `randomDAG` fornecida pelo corpo docente para gerar *DAGs* de tamanho variável até à ordem de grandeza de 10^6, com probabilidade de densidade de arco $p = 0.4$, bem como a chamada de sistema `time` com o intuito de cronometrar o desempenho do algoritmo implementado dado *DAGs* de *input* gerados pelo `randomDAG`.



Por fim, registaram-se os valores obtidos da testagem e ajustou-se uma regressão linear que demonstra, experimentalmente, a veracidade da complexidade do algoritmo esperada teoricamente, vendo que o tempo de execução tem uma relação linear com o tamanho do grafo - $\mathbf{O(V{+}E)}$.