Relatório 1º Projeto ASA 2020/2021

Grupo: al034

Aluno(s): David Belchior (95550) e Diogo Santos (95562)

Descrição do Problema e da Solução

Considerámos o conjunto de dominós como um **grafo dirigido acíclico** (DAG), em que cada dominó é um vértice e uma ligação entre um dominó que pode derrubar outro é uma aresta.

Para encontrar o número mínimo de dominós a derrubar para assegurar que todos caem, inferimos que esse valor corresponderia ao número de dominós que não podem ser derrubados por outros, o que equivale ao **número de *sources* do grafo**.

Além disso, concluímos que **a maior sequência de dominós derrubados teria de começar obrigatoriamente num desses dominós** (no grafo, as *sources*), pois, começando noutro qualquer, existiria sempre um caminho maior, começando num dominó que pudesse derrubar aquele em que começámos. Ao percorrer o grafo por **ordem topológica** (recorrendo a uma modificação do **algoritmo de Kahn**), asseguramos que, ao visitar um dado vértice, todos os subcaminhos até ele já foram percorridos, e, como tal, já possui o valor da sequência máxima que permite chegar a ele, propagando, de forma correta, esse valor aos seus vizinhos.

Algoritmo de Kahn: https://en.wikipedia.org/wiki/Topological_sorting#Kahn.27s_algorithm

Análise Teórica

O nosso programa efetua os seguintes passos:

- **Leitura dos dados de entrada**: leitura do input e criação do grafo por via de uma lista de adjacências. Este processo depende apenas do número de arestas do grafo (E), logo a sua complexidade é **O(E)**.
- **Procura pelas *sources* do grafo** (verificando apenas o seu *in-degree*). Logo, **O(V)**.
- **Aplicação de uma versão modificada do algoritmo de Kahn** para percorrer, por ordem topológica, os vértices do grafo, de modo a garantir a propagação correta dos caminhos mais longos.
    - A colocação das *sources* na *queue* é, no pior caso, O(V);
    - A análise aos caminhos mais longos dos vértices adjacentes totaliza O(E);
    - Cada vértice é colocado na *queue* no máximo uma vez, logo o ciclo que verifica se a *queue* não está vazia é realizado O(V) vezes.

  Assim, por análise agregada, o algoritmo em causa tem complexidade O(V) + O(V) + O(E) = **O(V+E)**.
- **Apresentação dos dados**: **O(1)**.

Complexidade global da solução: O(E) + O(V) + O(V+E) + O(1) = **O(V+E)**.

Relatório 1º Projeto ASA 2020/2021

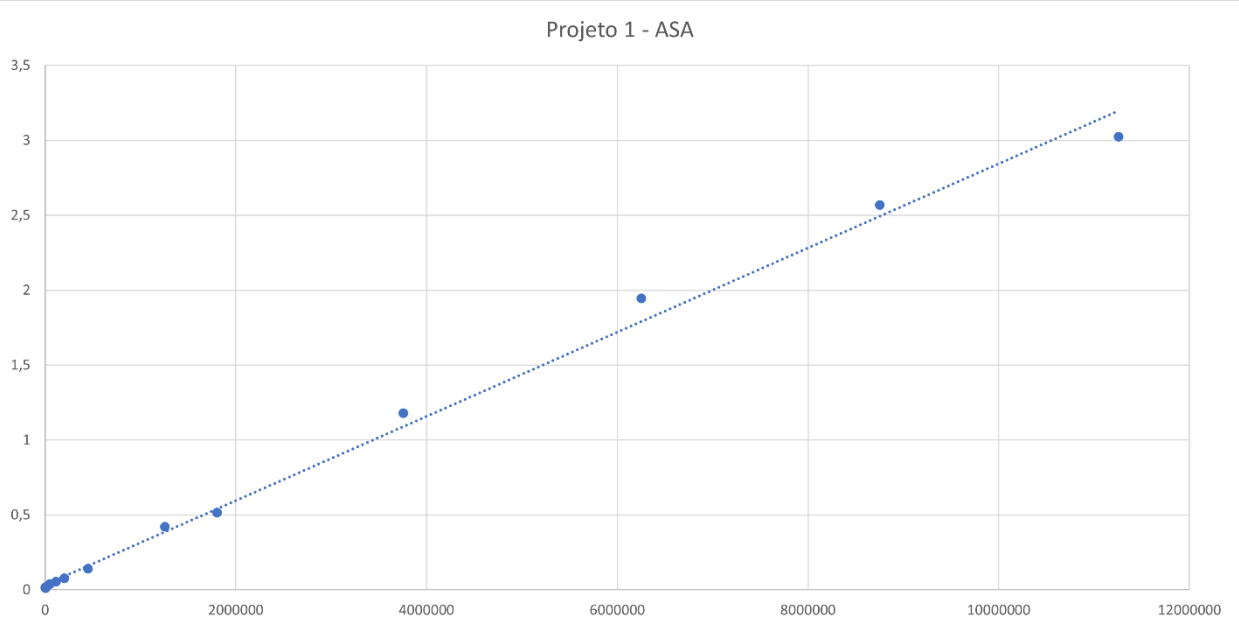
Grupo: al034

Aluno(s): David Belchior (95550) e Diogo Santos (95562)

Avaliação Experimental dos Resultados

Usando o programa **randomDAG.cpp**, fornecido previamente, gerámos 2 grafos com 100, 200, 500, 1000 e 2000 vértices, usando, para a probabilidade de criação de uma aresta, 0.1 e 0.9; além disso, gerámos 5 grafos com 5000 vértices, com probabilidades 0.1, 0.3, 0.5, 0.7 e 0.9.

Após correr o nosso projeto com os 15 grafos criados, obtivemos o gráfico seguinte, onde o eixo das abcissas é a soma dos vértices com as arestas (V+E) e o eixo das ordenadas é o tempo de execução (em segundos).



Com recurso a uma aproximação linear, concluímos que o tempo de execução confirma as nossas previsões teóricas, possuindo uma complexidade de **O(V+E)**.