**Descrição do Problema e da Solução**

O 1º projeto de ASA 2021/2022 é composto por dois problemas. Segundo o enunciado, o problema 1 consiste, dada uma sequência de inteiros, em determinar o tamanho da sua maior subsequência estritamente crescente (LIS), bem como o número de subsequências estritamente crescentes de tamanho máximo. Já o problema 2 resume-se, dadas duas sequências de inteiros, apenas a calcular o tamanho da maior subsequência comum estritamente crescente entre as duas (LCIS). Ambas as nossas soluções, implementadas em linguagem C++, recorrem a uma estrutura personalizada, composta por um vetor de inteiros e um inteiro que representa o tamanho desse vetor, para a representação das sequências e resolução dos problemas.

A solução do problema 1 é facilmente determinada com recurso a programação dinâmica e a dois vetores/tabelas auxiliares (com o tamanho da sequência inicial e inicializadas a 1) que resolvem os seus subproblemas (referentes ao tamanho e ao número de LIS até, e inclusive, cada índice da sequência inicial). Dadas duas variáveis (atualizadas aquando a descoberta de uma LIS com um novo tamanho máximo, ou, de uma LIS de tamanho igual à então descoberta) que representam a solução do problema, a ideia é iterar sobre todos os elementos da sequência inicial (ciclo exterior) e, para cada índice, percorrer todas as posições que estão para trás do mesmo (ciclo interior). (...)

A resposta à primeira pergunta é facilmente determinada, uma vez que, se partirmos do princípio que é preciso deitar abaixo mais do que um dominó para toda a sequência cair, significa que o DAG tem mais do que uma fonte (nó com grau de entrada igual a zero), dado que são os únicos tipos de nós impossíveis de aceder a partir de qualquer outro. Logo, o número de fontes é o número mínimo de dominós que se tem de deitar abaixo. No código, inicialmente, todos os nós são considerados fontes, mas à medida que os dados de entrada são lidos, as fontes são atualizadas.

A segunda pergunta já não é tão prontamente computada, visto que o problema do maior caminho num grafo não é tão simples quanto o do mais curto. Isto deve-se ao facto de o caminho mais longo não gozar da propriedade da subestrutura ótima (ou seja, os subcaminhos do caminho mais longo não são necessariamente os mais longos entre os nós desse subcaminho). Mas, é fácil perceber que o tamanho do caminho mais longo vai ser a maior distância de um nó a uma das fontes. No código, o foco da implementação é, após identificadas as fontes, a ordenação topológica dos nós do grafo. Em vez de se aplicar uma DFS para cada fonte, são apenas dadas as corretas distâncias aos seus nós adjacentes, que depois propagam estas distâncias para as suas adjacências, e assim consecutivamente. Esta propagação é efetuada pela ordem topológica, de modo a se saber quais nós visitar e “atribuir distância” primeiro (para não se propagar uma distância de um nó que podia ser maior).

**Análise Teórica**

Análise teórica da complexidade total e das várias etapas da solução proposta.

Inserir aqui um pseudo código de muito alto nível a indicar a complexidade de cada etapa.

Exemplo:

* Leitura dos dados de entrada: simples leitura do input, com ciclo(s) a depender de linearmente/quadraticamente/… de V/E/V+E/… Logo, Θ(V)
* Processamento da instância para fazer alguma coisa. Logo, O(??)
* Aplicação do algoritmo X para fazer algo. Logo, O(?X?X)
* Transformação dos dados com uma dada finalidade. O(?Y?Y?)
* Apresentação dos dados. O(???)

Complexidade global da solução: O(!??!)

**Avaliação Experimental dos Resultados**

Descrição do tipo experiências feitas e gráfico demonstrativo da avaliação de tempos associados.

Gerar pelo menos 10 instâncias (e indicar quais) de tamanho incremental e cálculo dos tempos para cada instância.

Gerar o gráfico do tempo (eixo do YYs) em função do tamanho da instância de entrada (eixo dos XXs) como exemplificado abaixo. Indicar a informação dos eixos.



Concluir se o gráfico gerado está concordante com a análise teórica prevista.