Relatório 1º projecto ASA 2021/2022

Grupo: al012 | Alunos: Valentim Santos (99343), Tiago Santos (99333)

Descrição da solução

PROBLEMA 1:

A solução encontrada para o problema 1 consiste em percorrer o vetor de valores, em que, para cada iteração, o vetor é novamente percorrido de forma a ser possível serem feitas constantes verificações e/ou atualizações no que diz respeito ao comprimento da maior subsequência crescente assim como à contagem das mesmas.

PROBLEMA 2:

A solução para o problema 2 tem uma estrutura semelhante à do primeiro problema, no que diz respeito a percorrer vetores. De resto trata-se de uma implementação de um algorítmo LCS com a simples otimização de apenas incluir, no segundo vetor, valores comuns ao primeiro (visto que estes, pela natureza do problema, seriam ignorados de qualquer das formas).

Análise teórica

LEITURA E PROCESSAMENTO DE INPUT:

A leitura de dados de entrada é feita em tempo linear sendo a sua complexidade $\mathbf{\Theta(n)}$ para o primeiro problema, e $\Theta\mathbf{(n + m)}$ para o segundo (em que ***n*** e ***m*** representam o tamanho das primeira e segunda sequências de input, respetivamente). O processamento do input terá a mesma complexidade que a escrita, visto que acontece ao mesmo tempo que esta.

PREENCHIMENTO DAS TABELAS:

Ambas as tabelas utilizadas nas soluções dos problemas são preenchidas da esquerda para a direita e de cima para baixo, sendo o acesso às mesmas de complexidade $\mathbf{O(1)}$, e assim, estes acessos não são relevantes para a complexidade final das soluções.

PROBLEMA 1:

São mantidos dois vetores (**lengths**: tamanho da LIS que acaba em cada índice, e **counts**: número de subsequências de max len que acabam em cada índice). Percorremos o vetor values (***i***), e para cada iteração voltamos a percorrê-lo (***j***) até ***i***. Quando values[i] > values[j], values[i] será então um novo elemento da LIS, logo:

se o tamanho da LIS a acabar em i (lengths[i]) <= lengths[j], como values[i] é um novo elemento da LIS, lengths[i] será então um incremento de lengths[j] (lengths[j] + 1). Aproveitamos ainda para atualizar o tamanho máximo de todas as LIS, caso lenghts[i] > **max_len** (variável onde guardamos o tamanho máximo das LIS) e reinicializamos o nosso **LIS_counter** (variável onde guardamos o número de LIS de tamanho máximo), pois **max_len** foi alterada.

se lenghts[i]-1 (lengths atual sem values[i]) for igual a lengths[j] então as LIS de ambos os índices são do mesmo tamanho, logo atualizamos counts[i], somando o valor de counts[j] visto que este cálculo é contínuo.

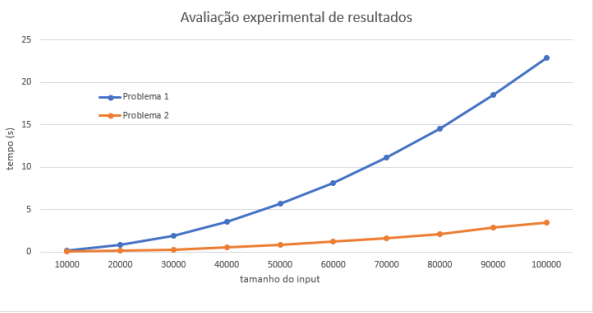
se lengths[i] for igual a max_len aumentamos então o nosso LIS_counter.

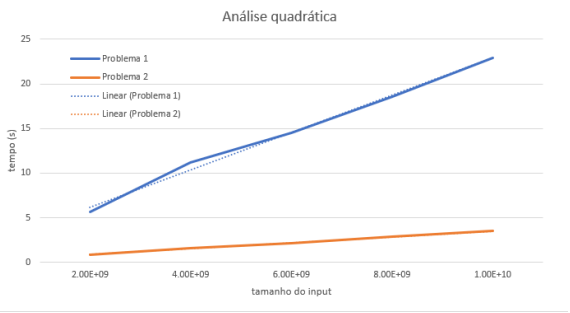
A complexidade final do algoritmo será então **O(n²).**

PROBLEMA 2:

Utilizamos 1 vetor (**lcis**: tamanho da LCIS a acabar em cada índice) e aplicamos um algoritmo de LCS, com a otimização de apenas incluir, no segundo vetor, valores comuns ao primeiro, visto que estes seriam ignorados de qualquer das formas.

A complexidade final do algoritmo será então **O(n × m)** em que ***n*** representa o tamanho do primeiro vetor e ***m*** do segundo.





Sendo ambos os problemas de complexidade **O(n²)**, os gráficos são então os esperados. Para input de tamanho quadrático, o gráfico assume uma curva linear, provando assim a complexidade de ambos os problemas. A razão pela qual o segundo problema apresenta ser muito mais rápido que o primeiro, deve-se ao facto de não calcularmos o número de subsequências crescentes comuns de tamanho máximo.