Relatório 1º projecto ASA 2021/2022

Grupo: tp037

Aluno(s): Joana Mendonça (83597)

Descrição do Problema e da Solução

Este projeto apresenta uma abordagem de **programação dinâmica** para resolver o problema da maior subsequência crescente e de maior subsequência crescente comum entre dois vetores, dado que surgia uma sobreposição de subproblemas que resolvidos com programação dinâmica apresentava melhores resultados em termos de complexidade de tempo, comparativamente com uma solução *naive* ou recursiva.

No problema 1 da **maior subsequência crescente** (função `LIS`), usei dois vetores: `length` (que guarda o tamanho da maior subsequência crescente) e `count` (que guarda a contagem de maiores subsequências crescentes).

Neste algoritmo iteramos sobre a sequência `(i = 0 to n)` e depois iteramos sobre os índices à esquerda de `i, (j = 0 to i)`. O objetivo é incrementar o valor de `length[i]` cada vez que encontra um valor menor à sua esquerda (`X[i] > X[j]`), que tenha um tamanho de `length` igual ao seu, ignorando os valores maiores ou iguais, uma vez que pretendemos uma subsequência que seja crescente.

No entanto, caso exista um valor à esquerda menor que `X[i]`, cujo tamanho de `length[j] < length[i] e length[j]+1 == length[i]`, significa que esse valor origina uma nova subsequência crescente, e como tal, o vetor `count` é incrementado.

No problema 2 da **maior subsequência crescente comum entre dois vetores** (função `LCIS`), usei um vetor `length`, que guarda o tamanho da maior subsequência crescente comum entre dois vetores, que retorna no final o tamanho da mesma.

Neste algoritmo, iteramos sobre o vetor X e de seguida sobre o vetor Y e se o número for igual, incrementamos o vetor `length`.

Relatório 1º projecto ASA 2021/2022

Grupo: tp037

Aluno(s): Joana Mendonça (83597)

Análise Teórica

Problema 1

A leitura de input é efetuada com uma leitura simples a depender linearmente do tamanho de entrada do input. O(len(input)).

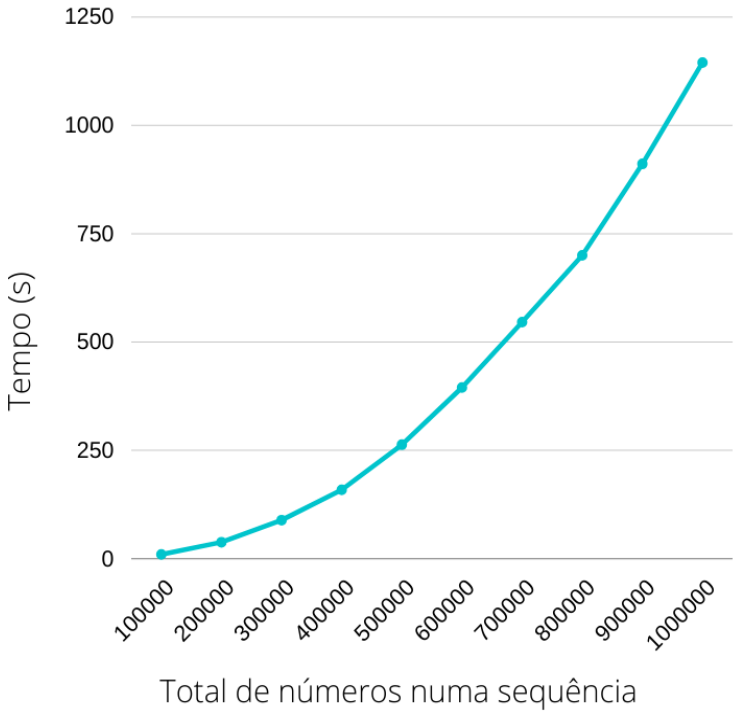
No primeiro ciclo da função `LIS`: `for i = 0 to n`, iteramos *n* vezes sobre o loop sendo a sua complexidade de tempo O(n). Já no segundo ciclo (ciclo interior): `for j = 0 to i`, iteramos também *n* vezes, no pior caso (quando `i = n`), sendo a complexidade temporal deste loop O(n). Perfazendo na totalidade, $O(n^2)$.

A iteração para encontrar o valor do tamanho da maior subsequência crescente tem no pior cenário $O(n)$.

O ciclo para determinar o número de maiores subsequências crescentes é também O(n).

Complexidade global da solução: $O(n^2)$

O gráfico seguinte foi gerado para uma probabilidade fixa de 0.999.



Como era expectável com base na análise teórica, o valor de tempo vai aumentando quadraticamente à medida que o tamanho das sequências também aumenta, o que apesar de ser melhor que uma implementação de brute-force, não é ideal, pois para sequências superiores a 300000, já atingimos tempos na casa dos minutos. Uma possível melhoria do algoritmo para evitar que a sua complexidade de tempo seja quadrática, poderia ser recorrer a filas de prioridade, com complexidade de $O(n \log n)$.

Relatório 1º projecto ASA 2021/2022

Grupo: tp037

Aluno(s): Joana Mendonça (83597)

Problema 2

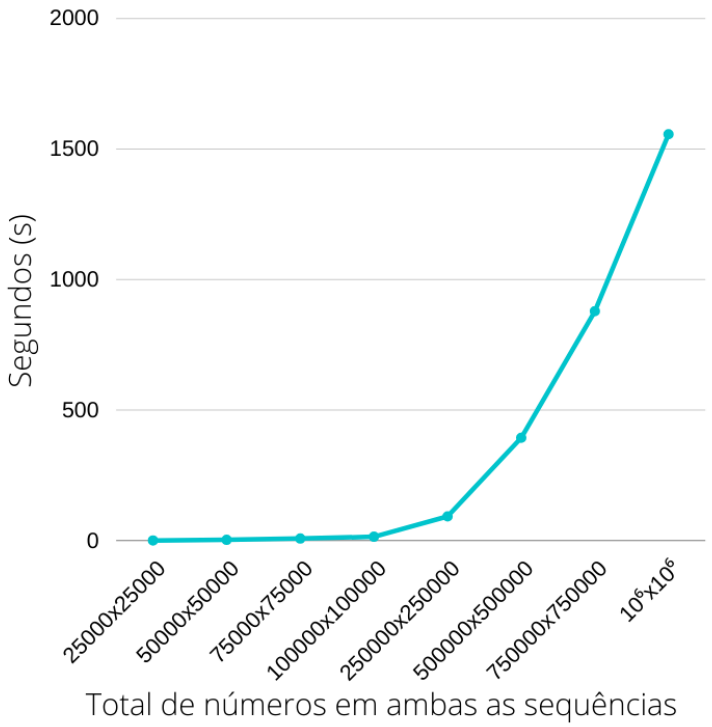
A leitura de input é efetuada com duas leitura simples a depender linearmente do tamanho de entrada do input: O(len(input)), cada.

No primeiro ciclo da função LCIS (`for i = 0 to n`), iteramos n vezes sobre o vetor X: O(n). E no ciclo interior (`for j = 0 to m`), iteramos m vezes sobre o vetor Y: O(m). Sendo a complexidade de tempo global destes ciclos, O(nm).

A iteração para encontrar o valor do tamanho da maior subsequência crescente comum tem complexidade *O(m)*.

Complexidade global da solução: *O(nm)*

O gráfico seguinte foi gerado para uma probabilidade fixa de 0.999.



Como era expectável com base na análise teórica, o valor de tempo vai aumentando quadraticamente (quando o tamanho das sequências é igual) à medida que o tamanho das sequências também aumenta.