Relatório 1º projecto ASA 2021/2022

Grupo: al088

Aluno(s): Francisco Lopes (99220) e Nuno Martins (99292)

Descrição do Problema e da Solução

Problema 1:

O problema consiste numa sequência de números inteiros onde se pretende calcular o tamanho da maior subsequência estritamente crescente (LIS - "Longest Increasing Subsequence") juntamente com o número de subsequências estritamente crescentes de tamanho máximo.

Após a obtenção do vetor de inteiros a partir do input e da obtenção do seu tamanho, entramos na função 'solve1'. Nesta inicializamos 2 vetores de inteiros: um 'vec' que irá contar o tamanho da subsequência estritamente crescente de tamanho máximo até ao index atual e 'count' que é o número de subsequências até ao index atual, e inicializamos também o contador para o tamanho da subsequência estritamente crescente mais longa a 1. Começamos a iterar e se por acaso entre j e i a sequência for estritamente crescente e o tamanho da LIS até j for igual ao tamanho da LIS até i até agora, então o tamanho da LIS que termina em i é igual ao tamanho da LIS que termina em j mais 1 unidade e o número de LIS até ao index i é igual ao número de LIS até ao index j. Caso haja mais LIS com o mesmo tamanho que a LIS a terminar em i, então incrementa o contador de número de LIS até esse index e no final atualizamos o tamanho da LIS máxima com o tamanho da LIS a terminar no index i. No final, uma vez que temos o tamanho máximo da LIS, basta iterarmos pelo vetor 'count' e contar todas as LIS de tamanho igual à LIS de tamanho máximo, e mostramos ao utilizador o tamanho máximo da LIS da sequência original de inteiros, assim como o número total de LIS com esse tamanho.

Problema 2:

O problema consiste em duas sequências de números inteiros onde se pretende calcular o tamanho da maior subsequência estritamente crescente comum a ambas.

Inicialmente obtemos a partir do input o primeiro vetor de inteiros, assim como criamos um set com os valores não repetidos desse vetor, para que quando formos obter os valores do segundo vetor de inteiros a partir do input, possamos apenas escolher os valores que estejam presentes no set, com complexidade temporal de O(1) (na verificação da existência do valor no set). Assim, obtemos o segundo vetor de inteiros apenas com os elementos comuns ao primeiro vetor, para evitar computação desnecessária na função 'solve2'. São também registados os tamanhos do primeiro e segundos vetores e passados esses tamanhos e os próprios vetores à função 'solve2'. Dentro da função 'solve2', é inicializada uma tabela de tamanho igual ao segundo vetor com os valores iniciais a zero. Essa tabela irá ser responsável por guardar todos os tamanhos da maior subsequência estritamente crescente comum entre os dois vetores de inteiros (LCIS = 'longest common increasing subsequence') que terminam em cada index do segundo vetor. De seguida itera-se por todos os elementos do primeiro vetor, inicializamos um contador a zero que indica o tamanho da atual LCIS e iteramos por todos os elementos do segundo vetor.

Se ambos os elementos que estamos a iterar entre os dois vetores forem iguais então, verifica-se se o tamanho da atual LCIS mais 1 é maior que a LCIS que terminava no index atual do segundo vetor, se sim, atualiza-se na tabela o novo maior tamanho da LCIS a terminar nesse index. Posteriormente procuramos o próximo menor elemento comum entre os dois vetores para o elemento do primeiro vetor que estamos atualmente a percorrer e atualizamos a variável de tamanho conformemente. No final, procuramos pelo valor máximo da nossa tabela que indica a LCIS de tamanho máximo e mostramos ao utilizador esse mesmo tamanho.

Análise Teórica

Problema 1:

- Leitura dos dados de entrada: simples leitura do input, com um ciclo a depender linearmente do tamanho do input, logo O(n), em que n=tamanho vetor inteiros inicial.
- Iteração com dois ciclos for (um dentro do outro) com complexidade temporal total igual a O(n^2), n=tamanho vetor inteiros inicial.
- Ciclo for final para contar o número de LIS de tamanho máximo com complexidade temporal O(n), n=tamanho vetor inteiros inicial.
- Complexidade espacial é O(n), n=tamanho vetor inteiros inicial, uma vez que temos de reservar memória para um vetor inicial para guardar o input, assim como 2 vetores auxiliares, 'vec' e 'count' ambos de tamanho n=tamanho vetor inteiros inicial.

Complexidade Temporal final do algoritmo = O(n^2)

Complexidade Espacial final do algoritmo = O(n)

Problema 2:

- Leitura dos dados de entrada: simples leitura do input, com um ciclo a depender linearmente do tamanho do input do primeiro vetor e outro a depender do tamanho do input do segundo vetor, logo complexidade temporal O(n+m), n=tamanho 1º vetor, m=tamanho 2º vetor.
- Iteração por todos os elementos da tabela para a inicializar a 0, logo complexidade temporal é O(m), m=tamanho do 2º vetor.
- Iteração de todos os elementos do primeiro vetor e em cada elemento itera-se por todos os elementos do segundo vetor, logo complexidade temporal O(n*m), n=tamanho 1º vetor, m=tamanho 2º vetor.
- No final, iterar pelos valores da tabela e retornar o maior valor da tabela, com complexidade temporal O(m), m=tamanho do 2º vetor.

Complexidade Temporal final do algoritmo = O(n*m)

Complexidade Espacial final do algoritmo = O(n+m)

Avaliação Experimental dos Resultados: foram testados em ambos os programas 13 inputs de valor entre 100 e 204800 (multiplicando de 2 em 2) e no final verificámos que de facto o gráfico está em concordância com a análise teórica relativa à complexidade temporal prevista.