TP3 – Exposição de Tecidos

Algoritmos 1

Departamento de Ciência da Computação - Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brazil
chbmleao@ufmg.br

1. Identificação

Carlos Henrique Brito Malta Leão

Matrícula: 2021039794

2. Introdução

Dado um arranjo de números inteiros, o problema requisita a construção de um novo arranjo, de forma que os números inseridos estejam ordenados em ordem decrescente. Porém, essa construção apresenta algumas restrições. Nesse sentido, primeiramente, os números só podem ser manipulados na ordem imposta pelo vetor original, por exemplo, dado o elemento de índice i do vetor original, ao inseri-lo no novo vetor, não será mais possível inserir o elemento de índice $i - 1$, por outro lado, a inserção do elemento $i + 1$ pode ser considerada. Nesse aspecto, ao considerar a inserção de um elemento no novo vetor, podemos tomar três diferentes decisões:

- Colocar o elemento ao início do novo arranjo.
- Colocar o elemento ao final do novo arranjo.
- Não colocar o elemento no novo arranjo.

Considerando estas opções de decisão para cada elemento do vetor original, a questão do problema é descobrir qual a maior quantidade de elementos que é possível inserir no novo vetor de forma que fique ordenado decrescentemente, isto considerando todas as restrições supracitadas. Além disso, outro fator importante a ser considerado é que os valores dos números do vetor original não se repetem, o que facilita a resolução do problema. Por fim, para realizar uma implementação eficiente que resolvesse esse problema, foi necessário a utilização de conceitos de Programação Dinâmica estudados durante o curso de Algoritmos 1.

3. Modelagem

Certamente, a escolha do primeiro elemento do novo arranjo é extremamente importante para a resposta final, já que, dado o primeiro elemento escolhido, à sua esquerda só poderão ser inseridos elementos maiores que este, e, à sua direita apenas elementos menores, para formar um vetor de números decrescente. Nesse sentido, existem dois problemas que tratam essas duas situações, o problema da Maior Subsequência Crescente (Longest Increasing Subsequence, LIS) e, de forma complementar, Maior Subsequência

Decrescente (Longest Decreasing Subsequence, LDS). Nesse sentido, temos as seguintes definições para os dois problemas:

- $LIS(i)$: tamanho da maior subsequência crescente, que **termina** com o i-ésimo elemento do arranjo.
- $LDS(i)$: tamanho da maior subsequência decrescente, que **termina** com o i-ésimo elemento do arranjo.

Porém, a nossa relação de recorrência desejada, para resolver o problema supracitado na seção 2, o chamaremos de $LIDS$, apresenta o seguinte formato:

- $LIDS(i)$: tamanho da maior subsequência crescente somada ao tamanho da maior subsequência decrescente, que **comecem** com o i-ésimo elemento do arranjo.

Claramente, existe uma diferença importante entre as relações de recorrência, em que $LIDS(i)$ **começa** com o i-ésimo termo, enquanto $LIS(i)$ e $LDS(i)$ **terminam**. Contudo, existe uma solução simples para este problema. Nesse aspecto, antes de utilizar os algoritmos LIS e LDS é preciso inverter o vetor. Dessa forma, realizando o mesmo algoritmo, com o vetor invertido, teremos as seguintes ressignificações para os conceitos de LIS e LDS:

- $LIS(i)$: tamanho da maior subsequência decrescente, que **começa** com o i-ésimo elemento do arranjo.
- $LDS(i)$: tamanho da maior subsequência crescente, que **começa** com o i-ésimo elemento do arranjo.

É importante reforçar que, estas afirmações acima, representam o resultado da execução dos algoritmos LIS e LDS no vetor inverso, considerando o vetor original. Em outras palavras, o resultado desses algoritmos para o vetor inverso continua sendo o mesmo já citado anteriormente, porém, esse resultado também gera conclusões distintas para o vetor original, estas são as ressignificações supracitadas.

Por fim, ao realizar essa simples alteração, é trivial encontrar o valor de $LIDS(i)$, em que basta somar os resultados de $LIS(i)$ e $LDS(i)$, executados com o vetor inverso. Assim, é preciso encontrar o $LIDS(i)$ para todos os elementos i do vetor, dessa forma, teremos o tamanho da maior subsequências crescente somada ao tamanho da maior subsequência decrescente, que começa com o i-ésimo elemento do arranjo para todos os elementos. Nesse aspecto, o elemento que apresentar o maior $LIDS$, corresponderá, também, à maior quantidade de elementos que é possível inserir no novo vetor.

Para exemplificar este processo que envolve a inversão do vetor, as execuções dos algoritmos LIS e LDS neste vetor invertido, e, por fim, a definição de $LIDS$, observe o seguinte diagrama, que apresenta os valores de um arranjo de 10 elementos e os índices de cada elemento ao realizar a inversão do arranjo:

Utilizaremos o índice do vetor invertido 9, que representa o valor 6, para calcular os valores de $LIS(9)$ e $LDS(9)$, que apresenta um caso mais interessante do problema. Nesse sentido, $LIS(9) = 4$, com os possíveis números $[1, 2, 3, 6]$, representados em verde no diagrama acima. Estes números, tirando o 6, representam os números que ficarão à direita de 6 no vetor final, ou seja, representam uma sequência decrescente a partir de 6. Por outro lado, $LDS(9) = 4$, com os possíveis números $[10, 9, 8, 6]$, representados em vermelho no diagrama acima. Estes números, tirando o 6, representam os números que ficarão à esquerda de 6 no vetor final, ou seja, representam uma sequência crescente a partir de 6.

Por fim, é possível, então, calcular o $LIDS(i)$. Sabemos que $LIDS(i) = LIS(i) + LDS(i) - 1 = 4 + 4 - 1 = 7$. É preciso subtrair 1, porque o número 6 aparece em ambas subsequências de números, o que não pode ocorrer no vetor final. Assim, teremos um possível vetor final com o seguinte conjunto de valores: $[10, 9, 8, 6, 3, 2, 1]$ e, além disso, concluímos que 7 é o tamanho da maior subsequência crescente somada ao tamanho da maior subsequência decrescente, que **comecem** com o nono elemento do arranjo invertido. Por fim, ao determinar os valores de $LIDS$ para todos os elementos do vetor, descobriremos que o $LIDS$ do vetor será realmente 7.

3.1 Relação de recorrência

Partindo do pressuposto que o método descrito na seção 3 foi entendido, temos as seguintes definições (ao executar o algoritmo no vetor invertido):

- $LIS(i)$: tamanho da maior subsequência decrescente, que **começa** com o i-ésimo elemento do arranjo. Ao realizar o tratamento pelo vetor invertido teremos a seguinte relação de recorrência:

$$LIS(i) = \begin{cases} 1, & i = 0 \\ \max\limits_{\substack{j<i \\ v[j]<v[i]}} LIS(j) + 1, & c.c \end{cases}$$

- $LDS(i)$: tamanho da maior subsequência crescente, que **começa** com o i-ésimo elemento do arranjo. Ao realizar o tratamento pelo vetor invertido teremos a seguinte relação de recorrência:

$$LDS(i) = \begin{cases} 1, & i = 0 \\ \max\limits_{\substack{j<i \\ v[j]>v[i]}} LDS(j) + 1, & c.c \end{cases}$$

- $LIDS(i)$: tamanho da maior subsequência crescente somada ao tamanho da maior subsequência decrescente, que **comecem** com o i-ésimo elemento do arranjo.

$$LIDS(i) = \begin{cases} 1, & i = 0 \\ LIS(i) + LDS(i) - 1 & c.c \end{cases}$$

De forma complementar, ao considerar apenas o vetor invertido de forma isolada, ou seja, sem estabelecer uma relação dos resultados com o vetor original, é possível considerar a seguinte definição para LIS e LDS:

- $LIS(i)$: tamanho da maior subsequência crescente, que **termina** com o i-ésimo elemento do arranjo invertido.
- $LDS(i)$: tamanho da maior subsequência decrescente, que **termina** com o i-ésimo elemento do arranjo invertido.

3.1 Detalhamento do código

Após a análise da relação de recorrência do algoritmo, partimos para a explicação do código desenvolvido para a resolução do problema. Todo o código já é tratado com o vetor invertido. Essa inversão do vetor ocorre durante a leitura do arquivo de entrada. Após esse processo, partimos para a execução dos algoritmos LIS e LDS, que serão melhor explicados a seguir.

Ambos estes algoritmos precisam de um vetor auxiliar cada, que apresentam tamanho n igual ao número de elementos do vetor, em que todos os elementos são inicializados com 1. Este vetor armazenará os valores da maior subsequência crescente ou decrescente. Dessa forma, os algoritmos apresentam dois laços aninhados, o mais externo itera o vetor usando i e o interno utiliza j. O laço externo itera todos os elementos do vetor original, já o interno, itera apenas até i. Isso ocorre, porque a definição de $LIS(i)$ depende, exclusivamente, dos elementos do vetor que são anteriores a i, ou seja, apenas os que aparecem antes na ordem. Nesse sentido, o loop interno itera do elemento 0 até $i - 1$. Considerando $values$ o vetor que armazena todos elementos do vetor invertido, no caso do algoritmo LIS, se $values[i] > values[j]$ e $lis[j] + 1 > lis[i]$, $lis[i]$ assume o valor de $lis[j] + 1$. Isso ocorre porque se o elemento i é maior que j, significa que i pode entrar na sequência a frente de j, assumindo o seu número de elementos da maior subsequência que termina em j mais um. Um processo semelhante ocorre com o algoritmo LDS, mas neste é verificado se $values[i] < values[j]$, já que busca-se uma subsequência decrescente.

Por fim, ao executar estes dois algoritmos, temos o tamanho das maiores subsequências, crescentes e decrescentes, que começam com cada um dos elementos do

vetor. Em seguida, para encontrar $LIDS$, é preciso realizar uma soma dos dois vetores encontrados, ou seja, $LIDS[\ \ ] = LIS[\ \ ] + LDS[\ \ ]$. Após realizar esta soma vetorial, basta encontrar o maior valor existente no vetor $LIDS$ e subtrair 1, já que o mesmo valor é somado duas vezes, como já foi supracitado. Por fim, este valor é impresso na tela e é possível partir para o próximo caso de teste.

3.2 Análise de Complexidade

Por fim, após esclarecer a lógica por trás da resolução do problema, a relação de recorrência e o detalhamento do código, podemos realizar uma análise sucinta da complexidade assintótica do programa como um todo. Para isso, nesta seção, consideraremos n como o número de elementos do arranjo, ou o número de rolos recebidos pela loja de tecidos em um dia. Além disso, apenas um caso de teste será considerado.

Existem cinco principais procedimentos executados no programa. Primeiramente, a inversão do arranjo original de entrada demanda tempo linear, $O(n)$, e é realizado durante a própria leitura do arquivo de entrada. Além disso, ao final do programa, é preciso realizar a soma vetorial de dois vetores e encontrar o maior elemento de um vetor, em que, ambos apresentam tempo linear, complexidade assintótica de $O(n)$.

Ao final, restam os algoritmos LIS e LDS, que apresentam um comportamento muito semelhante, apenas com a mudança de uma comparação. Esses algoritmos executam dois loops aninhados. O loop mais externo itera, sempre, os n elementos, enquanto o loop mais interno itera, no máximo, n elementos. Na maioria das execuções, este laço interno executa um número de vezes menor que n, já que é delimitado pelo iterador i do loop mais externo. Assim, a complexidade assintótica de ambas funções é de $O(n^2)$.

Por fim, podemos somar todas estas complexidades, e encontrar uma complexidade quadrática, $O(n^2)$, para todo o programa.

$$O(n) + 2O(n) + 2O(n^2)$$

$$\boldsymbol{O(n^2)}$$

Porém, essa complexidade assintótica pode ser melhorada. Para realizar esta melhora, é preciso utilizar mais um vetor auxiliar m. Nesse vetor, $m[i]$ representa o menor elemento terminando uma subsequência de comprimento i. Nesse sentido, esse vetor ordenado pode ser utilizado para realizar uma busca binária e encontrar o maior ou menor j. Essa implementação, diminuiria a complexidade assintótica para $O(n \log n)$.