

TRABALHO PRÁTICO 3

Avaliação do evento

Tarcizio Augusto Santos Lafaiete

Universidade Federal de Minas Gerais(UFMG)

Belo Horizonte - MG - Brasil

tarcizio-augusto@hotmail.com

1.Introdução

O problema proposto pelo trabalho, foi organizar na prateleira de uma loja, uma série de rolos, em ordem decrescente, que acabaram de chegar de uma encomenda, seguindo três regras simples. As regras do vendedor são: Colocar o rolo pela esquerda, colocar o rolo pela direita, ou não colocar o rolo. O desafio é descobrir o maior número possível de rolos a serem colocados na prateleira.

2.Interpretando o problema

Recebemos no início M entradas de tamanho variáveis com N elementos (não sendo N uma constante). A partir daí temos que processar esses N elementos de forma a obtermos o resultado final (conjunto R), em quantidade, da melhor sequência de escolhas que o vendedor pode tomar.

Imaginando agora, que o vendedor de fato tomou as melhores escolhas, e obteve o melhor resultado, a nossa resposta ideal (o resultado final da prateleira), pode ser dividido em duas partes: A parte dos rolos postos pela direita (conjunto D), e a parte dos rolos postos pela esquerda (conjunto E). O conjunto D tem seus elementos em ordem decrescente dentro da nossa entrada, enquanto o conjunto E, possui seus elementos em ordem crescente na mesma entrada.

Sabemos que o número de elementos de nossa resposta final é:

$$|R| = |E| + |D| - 1$$

sendo 1 a interseção entre E e D

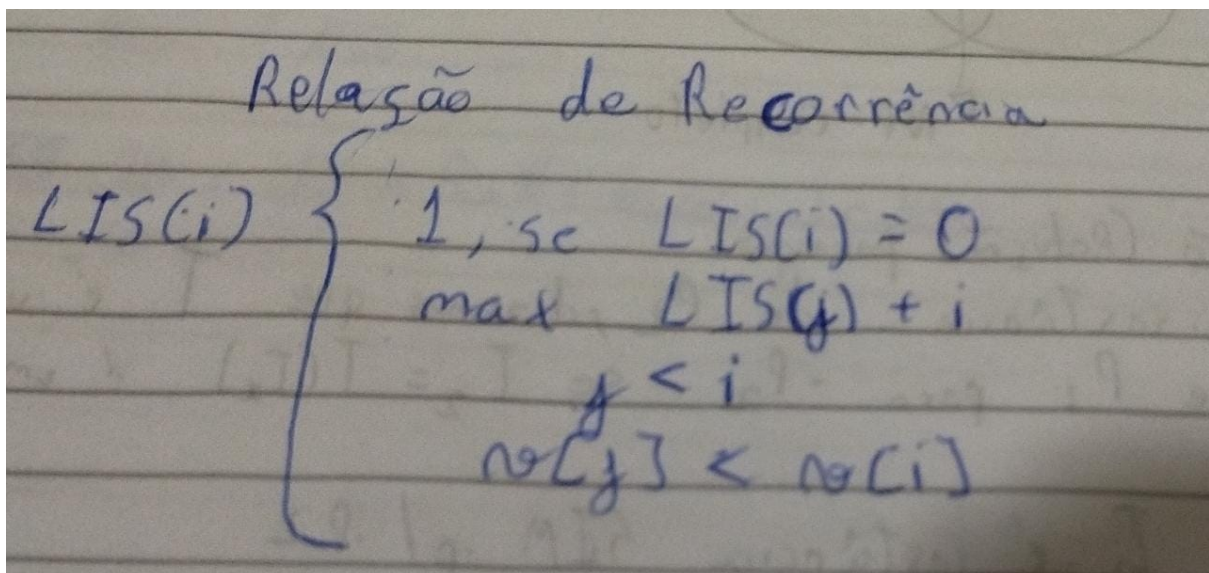
Sendo assim basta que encontremos a maior soma $|E| + |D|$ para otimizarmos a nossa resposta. Para isso usaremos dois algoritmos vistos em sala.

3. Modelagem do problema

Para resolver o desafio, utiliza-se do algoritmo de Maior Subsequência Crescente (LIS), e sua contraparte, Maior Subsequência Decrescente (LDS), para encontrarmos E e D respectivamente, para cada elemento da entrada (LIS e LDS são chamadas apenas uma vez cada), e então encontraremos entre eles, aquele conjunto E + D de maior valor.

A fim de bater o tempo proposto, é necessário utilizar um paradigma chamado de Programação Dinâmica (PD) para cortar gastos desnecessários. A PD tem como principal objetivo, eliminar a necessidade de recalculiar passos já feitos antes, os guardando em uma estrutura de dados dentro do programa. Essa solução foi escolhida tendo em vista a enorme complexidade que o problema exigiria caso não fosse implementada a PD, e o já reconhecido caráter PD da LIS e do LDS, que podem pertencer a uma eficiência de até $O(n \log n)$.

4. O algoritmo



The image shows a handwritten recurrence relation for the Longest Increasing Subsequence (LIS) problem. The title is "Relação de Recorrência". The recurrence is written as:

$$LIS(i) = \begin{cases} 1, & \text{se } LIS(i) = 0 \\ \max_{\substack{j < i \\ no[j] < no[i]}} LIS(j) + 1 \end{cases}$$

O algoritmo de Maior Subsequência (crescente ou decrescente) utiliza-se de um vetor auxiliar (aux), tal que:

$aux[i] =$ maior subsequência (crescente ou decrescente) **terminada** no i-ésimo elemento.

Para facilitar o cálculo e agilizar o processo, foi usada a ordem contrária, e o vetor aux foi populado da seguinte forma:

$aux[i] =$ maior subsequência (crescente ou decrescente) **iniciada** no i-ésimo elemento.

Isso permite ao algoritmo obter o primeiro elemento das duas subsequências sem utilizar computação extra.

Uma vez calculados os vetores aux para a LIS e para a LDS, comparamos eles item a item, para obtermos a maior soma $|E| + |D|$, começada no i-ésimo elemento.

$\text{MAX } i = 0 \text{ até } N (\text{lds-aux}[i] + \text{lis-aux}[i])$

5. Implementação

A implementação completa deste problema foi feita em um arquivo .cpp e para o desenvolvimento do algoritmo a estrutura de dado padrão do c++ utilizada foi apenas o vector. O primeiro passo do programa é realizar a captura da entrada, esta captura é realizada através de dois loops aninhados, no qual, o primeiro loop é referente ao número de casos de teste e o segundo é referente ao número de rolos. O valor dos rolos é guardado em um `std::vector`, que em seguida é passado com sua ordem de forma inversa para outro vetor, isto é feito pelos motivos explicados anteriormente no tópico 4 deste TP.

Após obter o vetor com a ordem de entrada invertida, utilizamos do algoritmo de LDS (Longest Decreasing Subsequence) para calcular a maior subsequência decrescente iniciada com cada um dos i-ésimos termos e salvar estes resultados em um `std::vector`. Em seguida passasse também o vetor invertido pelo algoritmo de LIS (Longest Increasing Subsequence), no qual é calculada a maior subsequência crescente iniciada com cada um dos i-ésimos termos do vetor e salvar estes resultados em um `std::vector`.

Com os cálculos da LDS e da LIS feitos e salvos em vetores, realizamos a soma dos valores da LDS e LIS para cada posição i e salvamos em um novo vetor. A partir deste vetor originado da soma da LDS e da LIS, pode-se encontrar a resposta do problema encontrando o máximo deste vetor. Importante ressaltar que é necessário realizar a subtração de 1 no valor encontrado como resposta, pois quando iniciamos a LDS e a LIS nós contamos o próprio i-ésimo elemento como parte da solução, assim sendo quando somamos LDS e LIS o i-ésimo elemento estará sendo contado duas vezes.

6. Conclusão

Por fim, a complexidade total do algoritmo pertence a $O(n^2)$, sendo N o número de rolos da entrada. Essa complexidade advém dos loops encadeados que são usados dentro da LIS e do LDS, e que se tornam o gargalo do algoritmo.