MO417 - Ata do exercício 15.4-6[1,2]

Enunciado:

$\star$[3] Dê um algoritmo $\in O(nlgn)$ para encontrar a maior sub-sequência monotonicamente crescente de uma sequência de n numeros . (Dica: Observe que o último elemento de uma sub-sequência candidata de comprimento i é pelo menos tão grande quanto o último elemento de uma sub-sequência candidata de comprimento $i-1$. Mantenha sub-sequências candidadas, ligando-as através da sequência de entrada.)

Algumas propostas de soluções foram feitas (em classe e por e-mail) por vários alunos (Ivo Koga, Fábio Nagamine) e pelo professor, mas todas provavelmente todas $\in O(n^2)$. A solução aqui apresentada é de autoria de Renato Hirata. Esta solução proposta para a SCML (Sub-sequência Crescente Mais Longa) baseia-se em programação dinâmica e na dica dada no exercício.

Solução:

O algoritmo aceita, na sequência de entrada, valores repetidos e valores menores que zero. A solução utiliza dois vetores auxiliares:

`Pred[1..n]:` vetor contendo n posições e cada posição tem dois campos:

- `Pred[i].valor` : Valor do predecessor de `Entrada[i]` numa SCML
- `Pred[i].indice` : Índice em entrada desse valor (é necessário para tratar empates e buscas binárias)

Este vetor é utilizado para impressão da solução ótima.

`Menor[k]` : vetor contendo no máximo n posiçoes e cada posição tem dois campos:

- `Menor[k].valor` : Valor do menor último elemento possível de uma SCML de tamanho `k`. (`menor [k]` será pelo menos `menor[k-1]` → dica do livro)
- `Menor[k].indice` : Índice em entrada desse valor

Durante o processamento do algoritmo, o vetor **`Menor`** varia de `0` ao `Tamanho_Maior_SCML` encontrada até o momento. (no máximo n)

1 O exercício é do capítulo 15 do livro *Introduction to Algorithms, Second Edition* de Thomas H. Cormen , Charles E. Leiserson , Ronald L. Rivest e Clifford Stein.

2 Ata foi redigida por Celina d' Ávila Samogin.

3 Segundo os autores, a estrela neste exercício indica que ele é mais adequado para ser resolvido por estudantes já graduados e ele exige mais conhecimentos avançados e mais criatividade para se chegar em uma solução.

```
Busca_SCML(Entrada,n)
01  Tamanho_Maior_SCML:= 0
02  Menor[0].valor:= -∞
03  Menor[0].indice:= 0
04  Para i:= 1 ate n
05     pos := posicao em Menor (entre 0 e Tamanho_Maior_SCML) do
06     maior elemento (de Menor) que seja menor ou igual a Entrada[i]
07     // pode ser usada busca binária, pois o vetor Menor estará
08     // ordenado
09     Menor [pos+1].valor:= Entrada[i]
10     Menor [pos+1].indice:= i
11     Pred[i].valor:= Menor[pos].valor
12     Pred[i].indice:= Menor[pos].indice
13     Se (pos+1) > Tamanho_Maior_SCML
14        Tamanho_Maior_SCML:= pos + 1
15     Fim Se
16  Fim Para
17  // Neste ponto, Pred e Menor estarão preenchidos.
18  // Menor[Tamanho_Maior_SCML].indice guarda a posição do último
19  // elemento da maior SCML
20  //
21  // Imprime SCML a partir da ultima posição
22  Prox:= Menor[Tamanho_Maior_SCML].indice
23  Enquanto prox <> 0
24    Imprime (Entrada[prox])
25    prox:= Pred[prox].indice
26  Fim enquanto
```

Observação: Note que para a impressão, o vetor **Menor** só é utilizado para se obter o último elemento. A partir de então é utilizado o vetor **Pred.** O motivo é que o vetor **Menor** pode perder referências intermediárias da maior SCML enquanto busca outras SCMLs. O vetor **Pred**, ao contrário, guarda para cada elemento de entrada o valor e a posição do predecessor numa SCML que eventualmente pode ser a maior. A saída é impressa em ordem inversa. Não é difícil a implementação de um algoritmo recursivo para imprimir a sub-sequência na ordem correta.

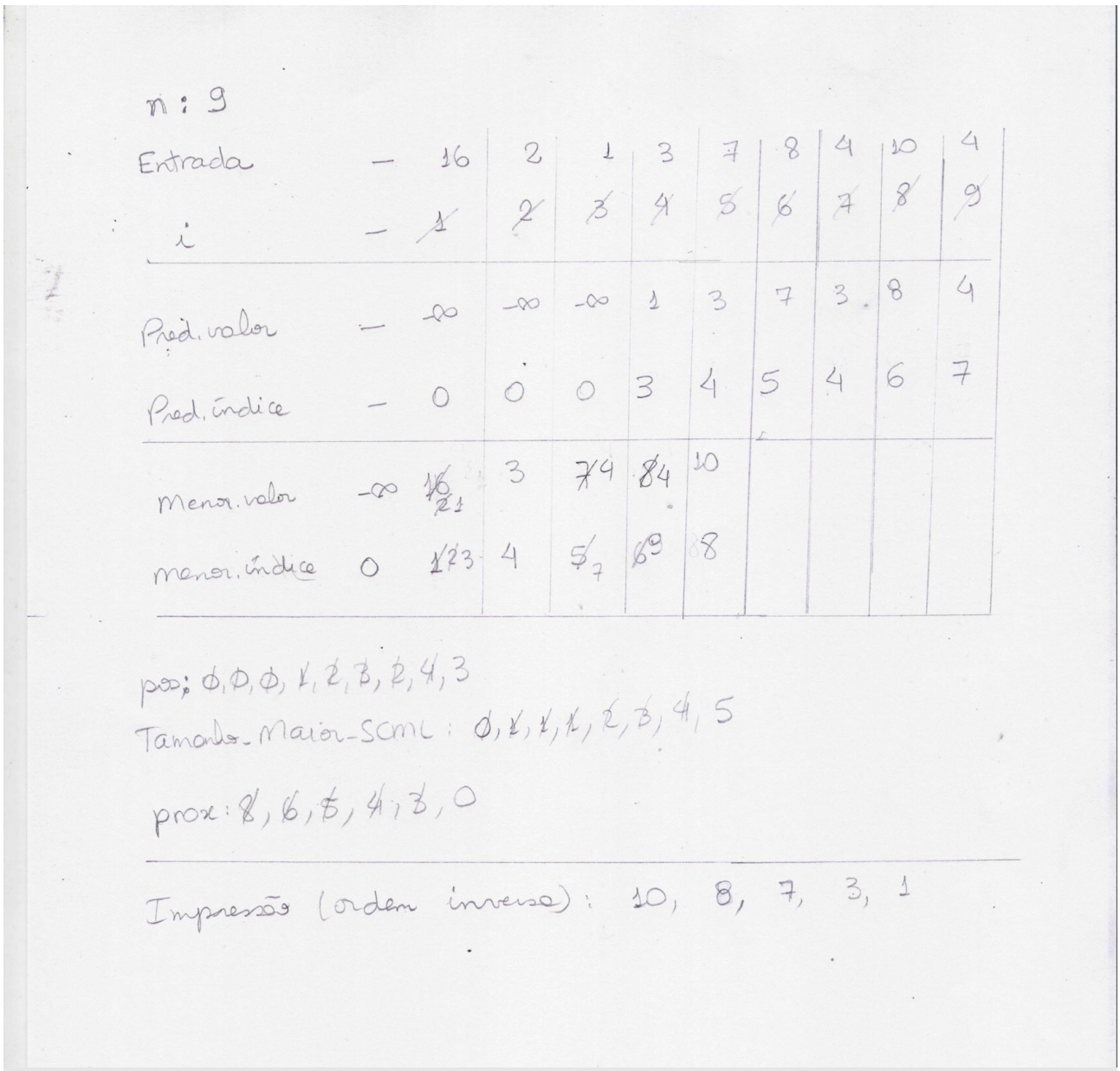

Ilustração 1: Execução do algoritmo Busca_SCML

Analisando as linhas de 04-16 (complexidade ϵ `O(nlogn))` e as linhas de 23-26 (cuja complexidade ϵ `O(n))`, claramente o algoritmo ϵ `O(nlogn)`.