Sabrina Araújo da Silva nUSP 12566182

MAC0338 - ANÁLISE DE ALGORITMOS

LISTA 5

exercícios 5 e 16

5. Escreva um algoritmo que, dado n e um vetor $v[1..n]$ de inteiros, determina uma subsequência crescente mais longa de v. Seu algoritmo deve consumir tempo $O(n^2)$. Se puder, dê um algoritmo para este problema que consome tempo $O(n \lg n)$.

```
SSCL ( v )
1    cria vetor aux[1...n]
2    cria vetor resultado[1...n] preenchido com -1
3    aux[1] = 1  // inicialmente a SSLC temporária inicia no primeiro item
4    tamanho = 1
5    para i ← 2 até n faça
6        se v[aux[1]] > v[i] faça  // substitui o primeiro item pelo menor
7            aux[1] ← i
8        senão se v[aux[tamanho]] < v[i] faça  // adiciona um item maior
9            tamanho ← tamanho + 1
10           aux[tamanho] ← i
11           resultado[aux[tamanho]] ← aux[tamanho - 1]
12       senão  // procura onde encaixar o item
13           j ← busca ( v , aux , tamanho , v[i] )
14           aux[j] ← i
15           resultado[aux[j]] ← aux[j-1]
16   i ← 1
17   cria vetor subseq[1... tamanho]
18   j ← aux[tamanho]
19   enquanto j é diferente de -1  // guarda a subsequência crescente mais longa em
                                   um vetor e devolve
20       subseq[i] ← v[j]
21       j ← resultado[j]
22       i ← i + 1
23   retorna subseq
```

```
busca (v, aux, tamanho, item)  // busca a maior posição para o item (busca binária)
1    início ← 1
2    fim ← tamanho
3    enquanto início ≤ fim
4        meio ← (início + fim) / 2
5            se v[aux[meio]] < item e item ≤ v[aux[meio + 1]] faça
6                retorne meio + 1
7            senão se v[aux[meio]] < item faça
8                início ← meio + 1
9            senão fim ← meio - 1
```

O algoritmo consiste em construir o vetor da subsequência com base em:

1. se v[i] é menor que o primeiro valor do vetor aux[], então o substitui por v[i]
2. se v[i] é maior que o último valor do vetor aux[], então adiciona v[i] em aux[].
3. se v[i] está entre o primeiro e o último, então é feita uma busca binária para encontrar a maior posição que v[i] pode estar.

Consumo de tempo

	linha	consumo das execuções das linhas
SSL()	1	$O(1)$
	2	$O(n)$
	3 a 4	$O(1)$
	5 a 15	$O(n)$
	16 a 18	$O(1)$
	19 a 22	$O(n)$
	23	$O(1)$
busca()	1 a 2	$O(1)$
	3 a 9	$T(\lceil n/2 \rceil)$

$$T(n) = T(\lceil n/2 \rceil) + O(n) = O(n \lg n)$$

16. **PC 111105** (Cortes de tora) Você deve cortar uma tora de madeira em vários pedaços. A empresa mais em conta para fazer isso é a *Analog Cutting Machinery (ACM)*, que cobra de acordo com o comprimento da tora a ser cortada. A máquina de corte deles permite que apenas um corte seja feito por vez.

Se queremos fazer vários cortes, é fácil ver que ordens diferentes destes cortes levam a preços diferentes. Por exemplo, considere uma tora com 10 metros de comprimento, que tem que ser cortada a 2, 4 e 7 metros de uma de suas extremidades. Há várias possibilidades. Podemos primeiramente fazer o corte dos 2 metros, depois dos 4 e depois dos 7. Tal ordem custa 10+8+6 = 24, porque a primeira tora tinha comprimento 10, o que restou tinha 8 metros de comprimento e o último pedaço tinha comprimento 6. Se cortássemos na ordem 4, depois 2, depois 7, pagaríamos $10 + 4 + 6 = 20$, que é mais barato.

Seu chefe encomendou um programa que, dado o comprimento l da tora e k pontos $p_1, \ldots, p_k$ de corte da tora, encontre o custo mínimo para executar esses cortes na ACM.

Seja ln o valor mínimo de corte para uma tora de comprimento n. Podemos definir ln recursivamente definindo onde aplicar o primeiro corte. Assim, se o melhor lugar para realizar o primeiro corte é no ponto i (onde $1 \leq i \leq n$), então o valor total é dado por ln = n + li + ln-i. Portanto, temos a recorrência:

$$ln = \begin{cases} 0, \text{ se } n = 0 \\ \min\limits_{1 \leq i \leq n} \{ n + li + ln-i \} \end{cases}$$

O algoritmo consiste em usar uma matriz r[n,n] para salvar os valores de soluções ótimas de subproblemas. Depois verifica se o problema já foi resolvido caso r[i,j] seja diferente de -1.

- ln é a soma do tamanho atual mais os valores mínimos da tora da esquerda e da direita.

Seja n o tamanho da tora de madeira e p um vetor p[1...n] no qual tem valor TRUE nos índices iguais aos pontos de corte.

```
CORTE-TORA (n, p)
1  CRIA VETOR r[n,n]
2  para i←0 até n faça
3      para j←0 até n faça
4          se i=0 ou j=0 faça
5              r[i,j] ← 0
6          senão r[i,j] ← -1
7  retorna CORTE-TORA-AUX (p, 0, n, r)
```

```
CORTE-TORA-AUX (p, start, end, r)
1  se r[start, end] ≥ 0 então
2      retorna r[start, end]
3  minimo ← ∞
4  para i ← start+1 enquanto i é menor que end faça
5      se p[i] = TRUE faça
6          n ← end - start
7          valor ← n + CORTE-TORA(p, start, i, r) + CORTE-TORA(p, i, end, r)
8          se valor < minimo então
9              minimo ← valor
10 se minimo = ∞ então
11     minimo ← 0
12 r[start, end] ← minimo
13 retorna minimo
```