



Instituto Federal de Educação, Ciência e Tecnologia de Brasília – Câmpus Taguatinga  
Ciência da Computação – Análise de Algoritmos  
Lista de Exercícios – Programação Dinâmica  
Prof. Daniel Saad Nogueira Nunes

Aluno: \_\_\_\_\_

Matrícula: \_\_\_\_\_

## Exercício 1

(Maior subsequência comum)

Implemente um algoritmo que, dado duas sequências  $A$  e  $B$  como entrada, compute a maior subsequência comum destas duas subsequências (LCS).

- Entrada: Strings  $X$  e  $Y$ .
- Saída: uma maior subsequência comum entre  $X$  e  $Y$ .

Exemplo:

- $A = aaabraaacaadaabraa$
- $B = xxxarstbrxaxcxxxaxxdxxxaxxbxxra$
- $LCS(A, B) = abracadabra$ .

## Exercício 2

(Maior subsequência crescente)

Seja  $V = (v_0, v_1, \dots, v_{n-1})$  uma sequência de elementos. Uma subsequência de  $V$  de tamanho  $k$ , pode ser definida como  $S = (v_{i_0}, v_{i_1}, v_{i_2}, \dots, v_{i_k})$  em que cada  $0 \leq i_0 < i_1 < i_2 < \dots < i_{k-1} \leq n - 1$ . Em outras palavras, uma subsequência de  $V$  pode ser formada ao ler  $V$  da esquerda para a direita e escolher elementos, não necessariamente consecutivos, de  $V$ .

Desta forma, dada uma sequência de elementos  $V$ , deseja-se saber qual o tamanho da maior subsequência crescente, isto é, o tamanho da maior subsequência de  $V$  em que os seus elementos estejam em ordem crescente.

- Entrada:  $V$ , uma sequência de elementos.
- Saída: o tamanho da maior subsequência de  $V$  em que seus elementos estejam em ordem crescente.

Exemplo:

- 
- Entrada:  $V = (-7, 2, 10, 9, 3, 1, 8)$
  - Resposta: 4, pois temos a subsequência  $S = (-7, 2, 3, 8)$ .

### Exercício 3

Como recuperar a subsequência crescente mais longa do problema anterior?

### Exercício 4

(Maior subsequência alternante)

Utilizando a definição de subsequência da questão anterior, compute a maior soma alternante dentre as subsequências de  $V$ . Uma soma alternante de uma sequência  $S = (s_0, s_1, \dots, s_{k-1})$  é aquela em que elementos em índices pares são somados e elementos em índices ímpares são subtraídos, isto é, a soma alternante de  $S$  neste caso é  $s_0 - s_1 + s_2 - \dots \pm s_{k-1}$ , em que o último sinal é “+”, caso  $k$  seja ímpar, e “-” caso contrário.

- Entrada:  $V$  o vetor de inteiros positivos.
- Saída: o maior valor possível de uma soma alternante dentre todas as subsequências de  $V$ .

Exemplo:

- Entrada:  $V = (9, 2, 4, 2, 1)$
- Saída: 7, que corresponde a soma alternante da subsequência  $9 - 2 + 4 = 11$

### Exercício 5

Como recuperar a subsequência de maior soma alternante do exercício anterior?

### Exercício 6

(Distância de Edição)

A distância de edição entre duas strings  $X$  e  $Y$  é o menor número de operações que você deve realizar em  $X$  para transformá-la em  $Y$ . Normalmente as operações consideradas são as de:

- Modificar um símbolo de  $X$ .
- Inserir um novo símbolo em  $X$  em qualquer posição.
- Remover um símbolo de  $X$

Assim, dadas strings  $X$  e  $Y$ , forneça o menor número de operações para transformar  $X$  em  $Y$ .

- Entrada: strings  $X$  e  $Y$ .
- Saída: o menor número de operações para transformar  $X$  em  $Y$ .

Exemplo:

- Entrada:  $X = \text{sunday}, Y = \text{saturday}$ .
- Saída: 3 operações: trocar  $n$  por  $r$ , inserir o caractere  $a$  e inserir o caractere  $t$ :

---

s \_ \_ u n d a y  
s a t u r d a y

### Exercício 7

Como modificar o algoritmo da distância de edição para ele indicar quais operações foram feitas em cada posição da string  $X$ ?

### Exercício 8

#### (Parentetização ótima de matrizes)

Dado as dimensões de  $n$  matrizes, implemente um algoritmo que calcule a parentetização ótima de modo a minimizar o número de operações feitas na multiplicação dessas matrizes.

- Entrada: um vetor  $p[]$  de tamanho  $n + 1$  de modo que  $p[i]$  e  $p[i + 1]$  indicam respectivamente o número de linhas e colunas da  $i$ -ésima matriz.
- Saída: a parentetização de modo a realizar o mínimo possível de operações para multiplicar as  $n$  matrizes.

Exemplo:

- Entrada:  $p = (10, 100, 5, 50)$ .
- Saída:  $(A^0 \cdot A^1) \cdot A^2$

### Exercício 9

#### (O problema da mochila booleano)

Roberval está planejando assaltar uma galeria de artes. Ele dispõe de um saco que aguenta até  $W$  kgs. Nesta galeria existem  $n$  peças distintas que podem ser roubadas. A  $i$ -ésima peça tem peso  $w[i]$  e fornece  $v[i]$  de lucro na feira da cidade. Escreva um algoritmo que forneça o lucro máximo de Roberval de acordo com as restrições do problema.

- Entrada:  $W \in \mathbb{N}^*$ ,  $v[]$  e  $w[]$ .
- Saída: a maior quantidade de lucro que Roberval pode ter.

Exemplo:

- Entrada:  $W = 5$ ,  $v = (100, 20, 60, 40)$  e  $w = (3, 2, 4, 1)$ .
- Saída: 140 reais.

### Exercício 10

#### (O problema da mochila com repetições)

Carlinhos está jogando o seu novo RPG chamado “Contos de um Paladino Solitário” e está buscando otimizar a sua estratégia para matar o chefe que se chama “Epaminond o terrível”. O paladino de Carlinhos possui  $W$  pontos de mana e possui  $n$  habilidades. Cada habilidade  $i$  consome  $w[i]$  pontos de mana e retira  $v[i]$  pontos de vida de Epaminond e a mesma habilidade pode ser utilizada diversas vezes. Escreva um algoritmo que indique qual o máximo de dano que o Paladino de Carlinhos pode infligir no demônio Epaminond.

- Entrada:  $W \in \mathbb{N}^*$ ,  $v[]$  e  $w[]$ .

- 
- Saída: a maior quantidade de pontos que o paladino de Carlinhos pode infligir em Epaminond.

Exemplo:

- Entrada:  $W = 8$ ,  $v = (10, 40, 50, 70)$  e  $w = (1, 3, 4, 5)$ .
- Saída: 110 pontos de dano.

### **Exercício 11**

Como modificar a solução do problema da mochila com repetições e booleano para fornecer as escolhas feitas em vez de apenas o resultado ótimo?

### **Exercício 12**

Suponha um sistema monetário que possua as moedas de  $\{1, 5, 10, 25, 50, 100\}$  dinheiros e uma quantidade  $W \in \mathbb{N}$  que deve ser paga utilizando essas moedas. Considerando uma quantidade infinita de moedas de cada valor, elabore um algoritmo que diga quantas são as formas de pagar a quantia  $W$ . Seu algoritmo deve possuir complexidade  $O(W)$ .

### **Exercício 13**

Tome um *grid*  $n \times n$ . Elabore um algoritmo que diga quantas são as formas de sair da posição  $(0, 0)$  deste *grid* e chegar a uma posição  $(i, j)$  qualquer andando apenas para baixo ou para a direita.