```
EXTENDED-BOTTOM-UP-CUT-ROD(p; n)
1     sejam r[n] e s[n] vetores
2     r[0] = 0
3     para j = 1 até n
4         q = -∞
5         para i = 1 até j
6             if q < p[i] + r[j-i]
7                 q = p[i] + r[j - i ]
8                 s[j] = i
9         r[ j ] = q
10 retorne r and s
```

tamanhos	1	2	3	4	5	6	7	8	9	10
preços(p)	2	5	8	8	10	13	14	15	16	21

1) Determine o preço de revenda máximo para uma barra de tamanho 5.

2) Escreve o algoritmo que descreve a solução (os cortes).

```
MATRIX-CHAIN-ORDER(p)
1     n = p.length - 1
2     sejam m[1 . . n; 1 . . n] e s[1 . . n - 1; 2 . . n] matrizes
3     para i = 1 até n
4         m[i , i] = 0
5     para L = 2 até n          // l é o tamanho da cadeia
6         para i = 1 até n - L + 1
7             j = i + L - 1
8             m[i , j] = ∞
9             para k = i até  j - 1
10                q = m[i , k] + m[k + 1, j] + p[i-1]*p[k]*p[j]
11                se q < m[ i , j]
12                    m[i , j] = q
13                    s[i , j] = k
14 retorne m and s
```

matriz	A1	A2	A3	A4
dimensão	20 x 10	10x30	30x5	5x20

p	20	10	30	5	20

1) Determine o mínimo número de multiplicação de escalares necessário para efetuar a multiplicação da cadeia de matrizes.

2) Escreva o algoritmo que descreve a solução (Parentização Ótima) para o problema acima.

```
LCS-LENGTH(X; Y)
1     m = X.length
2     n = Y.length
3     sejam b[1 .. m, 1 . . n] e c[0 . . m, 0 . . n] matrizes
4     para i = 1 até m
5         c[i, 0] = 0
6     para j = 0 até n
7         c[0, j] = 0
8     para i = 1 até m
9         para j = 1 até n
10            se x[i] == y[j]
11                c[i, j] = c[i-1, j-1] + 1
12                b[i, j] = “d”
13            senão_se c[i-1, j]  ≥ c[i , j-1]
14                c[i, j] = c[i-1, j ]
15                b[i, j] = “c”
16            senão c[i, j] = c[i, j-1]
17                 b[i, j] = “ e ”
18 retorne c and b
```

1) Calcule o tamanho da LCS para as Cadeias:
X= ABRACADABRA e;
Y= DACABRA.

2) Escreva o algoritmo que descreva a solução (a mais longa subcadeia comum) para o problema acima.

```
RECURSIVE-ACTIVITY-SELECTOR(s; f; k; n)
1     m = k + 1
2     enquanto m ≤ n e s[m] < f[k] //encontra a primeira
3         m = m + 1                    //atividade em Sk para
terminar
4     se m ≤ n
5       retorne {am} U RECURSIVE-ACTIVITY-ELECTOR(s,f,m,n)
6     senão retorne { };
```

i	1	2	3	4	5	6	7	8	9	10
s_i	1	4	8	5	8	10	10	12	11	15
f_i	6	7	10	12	12	15	16	16	18	20

1) Seleção o máximo de atividades que podem ser alocadas para a tabela de atividades e horários de inicio e fim acima.

```
HUFFMAN(C)
1     n = |C|
2     Q = C
3     para i = 1 até n-1
4         z=New(node)
5         z.left = x = EXTRACT-MIN(Q)
6         z.right = y = EXTRACT-MIN(Q)
7         z.freq = x.freq + y.freq
8         INSERT(Q,z)
9     retorne EXTRACT-MIN(Q) // retorna a raiz da árvore
```

caracter		**a**	**b**	**c**	**d**	**e**	**f**	**g**
frequência	30	20	25	15	35	10	40	

C = { 40, 35, 30, 25, 20, 15, 10}

1) Determine os códigos binários (codeword) para a compactação do texto com a frequência de caracteres dada acima.

2) Quantos bytes foram economizados pelo codeword definidos no exercício anterior ?