Programação Dinâmica – I

1. **Tabelação / "memoizing"**
   Algoritmo simplista, tempo exponencial

```
def fibm(n,m):
    if n==0: return 0
    if n==1: return 1
    return (fibm(n-2,m)+fibm(n-1,m)) % m
```

   Algoritmo com tabelação; linear para `m` fixo

```
maximo=1000
f = ["NAO_DEF"]*maximo

def fibm(n,m):
    if n==0: return 0
    if n==1: return 1
    if f[n] != "NAO_DEF":
        return f[n]
    res = (fib(n-2,m)+fib(n-1,m)) % m
    f[n]=res
    return res
```

2. **Custo mínimo do produto de matrizes**

```
 def matmin(d):
1    infinito = 1E20
2    m=len(d)-1                          # num de matrizes
3    c     = [[0]*m  for i in range(m)]  # definir mat. bidimensional com 0's
4    inter = [[-1]*m for i in range(m)]  # definir mat. bidimensional com -1's
5    for k in range(2,m+1):              # num de matrizes do sub-produto
6        for i in range(m-k+1):
7            j=i+k-1
8            cmin=infinito
9            for p in range(i,j):
10               custo=c[i][p] + c[p+1][j] + d[i]*d[p+1]*d[j+1]
11               if custo<cmin:
12                   cmin=custo
13                   inter[i][j]=p
14           c[i][j]=cmin
     return (c[0][m-1],inter)
```

3. **Produto óptimo das matrizes**

```
0   # Imprime  Ma ... Mb com a parentização óptima
1   def expr(inter,a,b):
2       p=inter[a][b]
3       if a==b:
4           printm(a),
5           return
6       print "(",
7       expr(inter,a,p)
8       print "*",
9       expr(inter,p+1,b)
10      print ")",
```

4. **Com tabelação – estudar nos apontamentos**

5. **Sequência máxima (não contígua) comum a 2 strings**
Tamanho da sequência óptima (esq.) — obtenção da sequência óptima (dir.)

`s` = "`bacbace`" (horizontal) →

		b	a	c	b	a	c	e
		0	0	0	0	0	0	0
a		0	*1*	1	1	1	1	1
b		1	1	1	*2*	2	2	2
a		1	*2*	2	2	*3*	3	3
e		1	2	2	2	3	3	*4*
c		1	2	*3*	3	3	*4*	4
c		1	2	3	3	3	*4*	4

`t` = "`abaecc`" (vertical) ↓

		b	a	c	b	a	c	e
		0	0	0	0	0	0	0
a		0	1	1	1	1	1	1
b		1	1	1	2	2	2	2
a		1	2	2	2	3	3	3
e		1	2	2	2	3	3	3
c		1	2	3	3	3	4	4
c		1	2	3	3	3	4	4

Notas. (i) Assuma a existência de uma coluna inicial (não visível em cima) de 0's. (ii) Explique como se pode obter a sequência máxima "`abae`".

6. **Função para o comprimento da sequência máxima comum**

```
 0  # Retorna o comprimento da maior sub-seq comum
 1  # strings 1..len-1  (índice 0 ignorado)
 2  def maxsubseq(s,t):
 3      m=len(s)   # caracteres 0 1 ... m-1
 4      n=len(t)   # caracteres 0 1 ... n-1
 5      ms = [[0]*(n+1) for i in range(m+1)]  # criar uma matriz [1..m][1..n]
 6      for i in range(1,m):
 7          for j in range(1,n):
 8              if s[i] != t[j]:
 9                  ms[i][j] = max(ms[i-1][j],ms[i][j-1])
11              else:
12                  ms[i][j] = 1+ms[i-1][j-1]
13      return ms[m-1][n-1]
```

7. **Função para a sequência máxima comum**

```
 0  # Retorna o comprimento da maior sub-seq. comum e essa sub-seq.
 1  def maxsub(s,t):
 2      m=len(s)
 3      n=len(t)
 4      ms  = [[0]*(n+1) for i in range(m+1)]
 5      seq = [[""]*(n+1) for i in range(m+1)]
 6      for i in range(1,m):
 7          for j in range(1,n):
 8            | if s[i] != t[j]:
 9            |     if ms[i-1][j] >= ms[i][j-1]:
10            |         ms[i][j]  = ms[i-1][j]
11            |         seq[i][j] = seq[i-1][j]
12            |     else:
13            |         ms[i][j]  = ms[i][j-1]
14            |         seq[i][j] = seq[i][j-1]
15            | else:
16            |     ms[i][j]  = 1+ms[i-1][j-1]
17            |     seq[i][j] = seq[i-1][j-1]+s[i] # concat. de strings
18          print ms[i]
19      return seq[m-1][n-1]
```

8. **Complexidade: solução naïve e Programação Dinâmica**