

* Lista de Exercícios

por Isadora de Oliveira

① 8|0|5 4|3 8|4|5 2

MAXSUBVETORPAR (v[], n)

```
int cnt = 0;  
for i = 0 até n  
    if (v[i] % 2 == 0)  
        cnt ++  
return cnt
```

Complexidade: O(n) / Guloso

MAXSUBVETORPAR (v[], l, r)

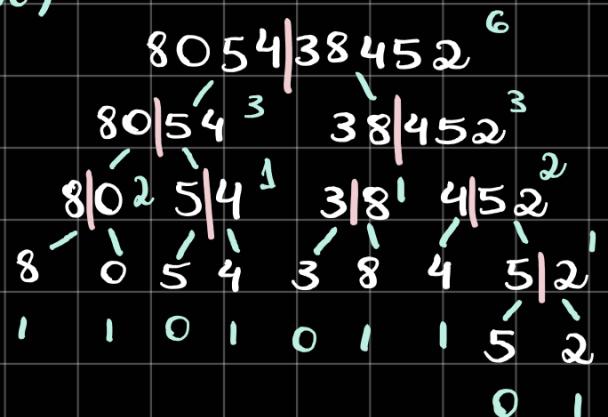
```
if (l == r)  
    if (v[l] % 2 == 0)  
        return 1  
    else  
        return 0
```

int md = (l + r) / 2

int left = MAXSUBVETORPAR (v, l, md)

int right = MAXSUBVETORPAR (v, md + 1, r)

return left + right



Complexidade: $2T(n/2) + O(1) = n^{\lg_2 2} = O(n)$

Fazendo divisão e conquista mas não está aproveitando da forma correta



4 →
3
2
5
8
7
1
10
11

Normal iniciar com dois

maior
subseqüência
alternante

$dp[i][j][dr], v[i]$

$SA(i, j, dr)$

$\{$ if ($i == n$) } contando tam
return 0 } ultrapassou

$\{$ if ($dp[i][j][dr] \neq -1$) } não calculei ainda, -1 da
return $dp[i][j][dr]$ } matriz inicializada não visto

* int ans = $SA(i+1, j, dr)$ } saltar

if ($v[i] > v[j] \&& dr == 0$)

ans = max (ans, 1 + $SA(i+1, i, !dr)$)

if ($v[i] < v[j] \&& dr == 1$)

ans = max (ans, 1 + $SA(i+1, i, !dr)$)

return $dp[i][j][dr] = ans$

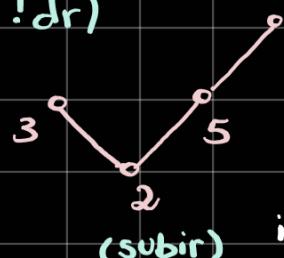
Complexidade : $O(n^2)$ | PD

Relação Recorrência

$SA(i, j, dr)$

$$\begin{cases} 0, & \text{se } i == n \\ SA(i+1, j, dr) \\ \max(SA(i+1, j, dr), 1 + SA(i+1, i, !dr), \text{ se } v[i] > v[j] \text{ e } dr == 0) \\ \max(SA(i+1, j, dr), 1 + SA(i+1, i, !dr), \text{ se } v[i] < v[j] \text{ e } dr == 1) \end{cases}$$

$dp[i][j][dr] =$ maior subseq zigzag que contém $v[i]$ tal



informação

3 2 5
↓ (subir)
3 2 5
↓ i
desci j

subir vai vir
antes, sendo
assim não pre-
cisa guardar 3

que $v[j] - v[i]$ segue a direção do dr esperada.

③ Modificar o número 2

④

$$11001000101 = 3$$

BI(v[], n)

int cntAtual = 0

int cntMax = 0

for i = 0 até n

if (v[i] == 0)

cntAtual ++

else

cntMax = max(cntMax, cntAtual)

cntAtual = 0

return cntMax

Complexidade: O(n) | Guloso

⑤ a) $p[], v[], dp[][]$

```
for (j=0; j < n; j++)
    for (i=0; i < c; i++)
        if (i == 0 || j == 0)
            dp[i][j] = 0
```

continue \rightarrow solução anterior com a capacidade

$$dp[i][j] = dp[i-1][j]$$

$\text{if } (p[i] \leq j)$ \rightarrow não pegar

$$dp[i][j] = \max(dp[i-1][j], v[i] + dp[i-1][j - p[i]])$$

return $dp[n][c]$

\rightarrow resposta anterior pre computada / pegar item anterior aumenta a resp

Complexidade : $O(nC)$

		j							
		0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0
i	1	0	2	2	2	2	2	2	2
2	0	2	4	4	4	4	4	4	4
3	0	2	4	4	4	7	7	7	7
4	0	2	4	4	4	7	9	9	9

$\leftarrow (P_4, V_4)$

$$dp[4][7] = 9$$

$dp[][]$ calculada

itens, $i = n$, $j = C$

while ($i > 0$)

\rightarrow peguei
 $\text{if } (dp[i][j] \neq dp[i-1][j])$

itens = itens $\cup \{i\}$

$j = j - p[i]$

$i--$

return itens

$$[i][j] = [1] - [1] = 0 - 0 = 0$$

sobe na linha de cima e
procura coluna(j) 0 vai ser
o valor do item que quero
coloca + ela, $2 + 0 = 2$

⑥ $p[]$, $v[]$, $dp[][]$

$KNAPSACK(i, C)$ Complexidade : $O(nC)$

```
if ( $i == n+1$ )
    return 0
if ( $dp[i][C] \neq -1$ )
    return  $d[i][C]$ 
int ans =  $KNAPSACK(i+1, C)$ 
if ( $p[i] \leq C$ )
    ans = max (ans,  $KNAPSACK(i+1, C-p[i]) + v[i]$ )
return  $dp[i][C] = ans$ 
```

Computar um item

$p[], v[]$ struct rec {
 $dp[][]$ int value,
 itens[] }

$KNAPSACK(i, C)$

```
if ( $i == n+1$ )
    return {0,  $\emptyset$ }
if ( $dp[i][C] \neq -1$ )
    return  $d[i][C]$ 
rec ans =  $KNAPSACK(i+1, C)$ 
if ( $p[i] \leq C$ )
    rec tmp =  $KNAPSACK(i+1, C-p[i])$ 
        tmp.value +=  $v[i]$ 
        tmp.itens.join ( $v[i]$ )
    ans = max (ans, tmp)
}
return  $dp[i][C] = ans$ 
```

\hookrightarrow sobre carregado

⑧ @Notas: 1, 4, 5 Troco: 8 Seria 5, 1, 1, 1 ao invés de 4, 4
por isso guloso não funciona

(b)

$$m(i, j) = \begin{cases} 0, & j == 0 \text{ forma o troco} \\ \infty, & j < 0 \text{ ou } i > n \text{ e } j \neq 0 \text{ não forma o troco} \\ m(i+1, j) & \text{bem, passou todas moedas} \\ \min(m(i+1, j), 1 + m(i+1, j - c_i), \\ 1 + m(i, j - c_i), c_i \leq j \\ \text{usa a moeda } c_i \\ \text{novamente} \end{cases}$$

(c) $dp[][]$

```
TROCO(i, m)
if(m == 0)
    return 0
if(m < 0)
    return infinity
if(i == n+1 && m != 0)
    return infinity
if(dp[i][m] != -1)
    return dp[i][m]
return dp[i][m] = min(TROCO(i+1, m), 1 + TROCO(i+1, m - c_i),
                      1 + TROCO(i, m - c_i))
```

m = troco tentando formar

⑨ Se 1 2 4 5 6 7 8 9 menor que falta 3

Soga $n = |S|$ Seja V vetor de tamanho $n \forall v[i] = -1$

$$1 \leq i \leq n$$

2 1 9 4 7 6 8 5

1	1	-1	1	1	1	1	1
---	---	----	---	---	---	---	---

1 2 3 4 5 6 7 8

Início o vetor com -1, quando tem o numero marco 1, depois retorno a primeira posição -1

SN(S)

$V[]$ com $|S|$ posições

for ($i=1; i \leq |S|; i++$)

$v[i] = -1$

for ($i=1; i \leq |S|; i++$)

$v[S[i]] = 1$

int id

for ($id=1; id \leq n; id++$)

if ($v[id] == -1 \&& id == -1$)

return

return id

Complexidade: $O(n)$ /Guloso