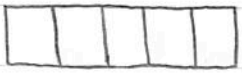
08 / 10 / 18

Módulo 03 : Paradigmas

Vinícius
20 pontos de prova
05 pontos de trabalho
} 25 pontos

* Indução

Base : P(n) é verdadeira, para algum número.
Passo Indutivo : $P(n) \rightarrow P(n+1)$, $n \geq n_0$



Qual maior elemento ?

Se n = 1 ✓ E se tiver n elementos ?

```
MAX(v, n)
    se n = 1                    // funciona para base
        return v[1]
    x = MAX(v, n-1)             // funciona para 1 elemento a menos
    return max(x, v[n])
```

O(n) fazemos n chamadas gastando O(1) em cada.

O(n)

$T(n) \leq c.n$

$T(n) = T(n-1) + d \leq c(n-1) + d \leq cn$

$d < c$

$P(x) = c_0 + c_1 x + c_2 x^2 + \dots + c_n x^n$

$c[\,], x \to P(x)$

Hi: sei calcular $P(x)$ se grau $P(x) \leq n-1$

base: $n=0 \to P(x) = c_0$

1ª forma:

```
calculaPolinomio (c, x, n)
    se n == 0
        ret c0
    v = CP(c, x, n-1)   // recursivo
    ret v + cn x^n
```

$\to T(n) = n + n \cdot f(n)$ é caro!

2ª forma:

$P(x) = c_0 + x(c_1 + c_2 x + c_3 x^2 + \dots + c_n x^{n-1})$

```
calculaPolinomioNovo (c, x, n)
    se n == 0
        ret c0
    v = CPN (c, x, n-1)   // recursivo
    ret cn + x.v
```

$P(x) = c_n + c_{n-1} x^1 + c_{n-2} x^2 + \dots + c_0 x^n$

$c_n + x(c_{n-1} + c_{n-2} x + \dots + c_0 x^{n-1})$

$CPN(c, x, n-1)$

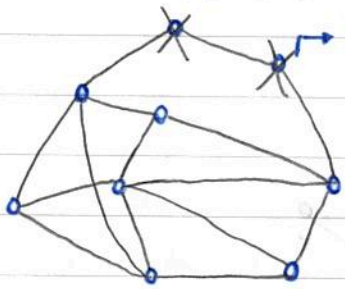
é $O(n)$!

3ª forma:

Sei calcular $P(x)$, se grau $(P(x)) \leq n-1$ e sei calcular x^{n-1}

```
CP (c, x, n)
    se n == 0
        ret (c0, 1)
    (v, g) = CP (c, x, n-1)   // g = x^(n-1)
    ret (v + cn.x.g, g.x)
```

outro exemplo:



Dados G, K

$\exists H$, subgrafo de G tal que grau mínimo de $H \geq K$?

se (n, K) retorna sim ok
senão (n-1, K) retorna sim ok
senão (n-2, K) retorna sim ok
⋮
(K, K)

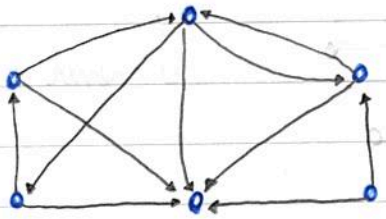
GrauPeloMenosK (G, K)
  Calcula graus // O(|V|+|E|)
  se menor grau ≥ K } O(|V|)
    ret sim
  seja v um vértice com grau < K
    ret GrauPeloMenosK (G-v, K) → // T(n) = T(n-1) + |V| + |E| =
  se |V(G)| ≤ K // O(1) O(|V|.(|V|+|E|)
    ret não Ruim!

O(|V|+|E|) → calcula graus, manter grau atualizado
Cada vértice é apagado uma vez

manter uma fila dos que serão removidos auxilia.
Em cada remoção verifica se algum K virou K-1 e remove também.

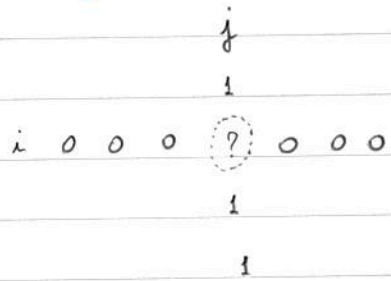
É parecido com busca em largura.

$O(|V| + |E|)$



Qual é o vértice celebridade? Todos o conhecem e ele não conhece ninguém.

matriz



Se $M[i,j] == 0$

i não conhece j, logo j não é celebridade.

Se $M[i,j] == 1$

i conhece j, logo j pode ser celebridade, mas i não pode ser.

vamos diminuindo a cada verificação;
Acha a celebridade de uma matriz menor.
Verifica as interseções (i conhece a celebridade e celebridade não conhece i).

15 / 10 / 18

```
celebridade (M, S)
    se |S| == 1
        ret x ∈ S
    Sejam i, j ∈ S, i ≠ j
    se M[i,j] == 0
        K = celebridade (M, S - {j})
        se K == 0
            ret 0
        se M[K,j] == 0 e M[j,K] == 1
            ret K
    senão
        K = celebridade (M, S - {i})
        se K == 0
            ret 0
        se M[K,i] == 0 e M[i,K] == 1
            ret K
    ret 0
```

* Subarray de soma máxima

um vetor: 2 -3 4 8 -10 18 -15 13 -5 4

Pegar a soma maior possível de um subvetor qualquer.
usando 2 ou 3 for
$O(n^3)$ ou $O(n^2)$

Se o vetor tem n elementos
usaremos um vetor de n-1 elementos

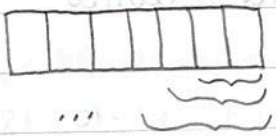
Queremos a maior então substituiremos uma posição negativa por vazia.

[5] → retorna 5

[-8] → retorna ϕ (zero)

Hi: sei resolver somaMáxima para vetores de tamanho n-1

```
SSM (v, n)
    se n == 1
        se v[1] > 0
            ret v[1]
        senão
            ret 0
    t = SSM (v, n-1)          // O(n-1)
    soma = 0
    para i = n até 1          // começa da última posição até o início
                              // O(n)
        soma += v[i]
        se soma > t
            t = soma
```



$T(n) = T(n-1) + C.n$

$T(n-2) + C(n-1) + C.C$

$O(n^2)$

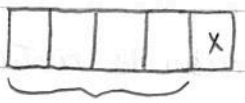
Mudando a indução:

Hi: sei resolver soma Máxima para vetores de tamanho n-1 e sei resolver sub array de soma máxima no final do vetor.

```
SSM2 (v, n)
  se n == 1
    se v[1] > 0
      ret (v[1], v[1])
    senão
      ret (0,0)
```

→ $(T, T') = SSM2(v, n-1)$

ret (max (T, T' + v[n]), max (T' + v[n], 0))



$t = SSM(n-1)$

$t' = SSM$ final vetor $(n-1)$

$max(T, T' + X)$

$max(T' + X, 0)$

Rodando o código:

n (Pos)	1	2	3	4	5	6	7	8	9	10	
SSM	2	2	(4)	12	12	20	20	20	20	(20)	retorno
SSMFV	2	0	4	(12)	2	20	5	18	13	17	
Array	2	-3	4	8	-10	18	-15	13	-5	4	

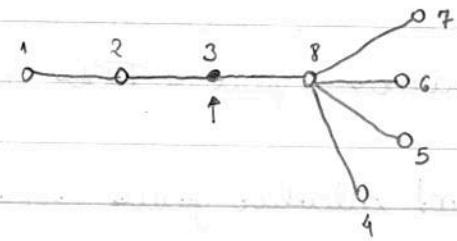
max(2, 4+0) = 4

max(4, 4+8) = 12

Sempre gastaremos O(n)!

Exemplo 2: minimizar a distância máxima

Vamos encontrar o centro de uma árvore



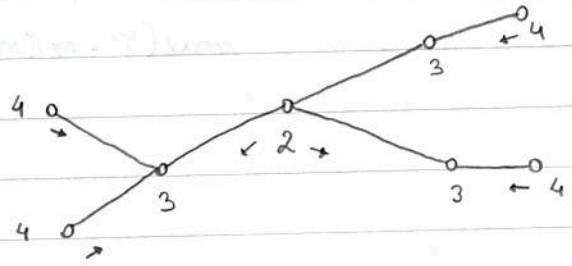
Centralizar os "serviços" em 3.

Todos teriam custo = 2.

Obs. 1: Se a árvore tem pelo menos 3 vértices, a solução não é uma folha.

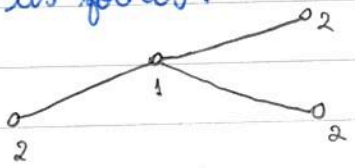
Conjectura: Se T é uma árvore com pelo menos 3 vértices e F é o conjunto de folhas de T, então:

Centro (T) = Centro (T-F)



distâncias do centro a qualquer ponto e de qualquer ponto a outro

Retiramos as folhas:



sempre diminui em 1

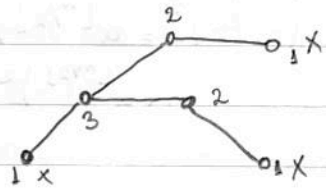
A conjectura virou Teorema!

Hi: Sei resolver Centro (T) para árvores menores que T.

Centro (T)
  se |V(T)| ≤ 2
    ret V(T)
  Determine F, o conjunto de folhas de T
    ret Centro (T-F)

Para verificar quais são folhas, calculamos o grau de todos os vértices.

Folhas = g(1) grau 1 $\Theta(n)$ calcular graus



Ao remover as folhas, colocamos estas em uma fila / lista F

Atualizamos o grau dos vizinhos, que diminuem 1.

$\Theta(n)$

Recursividade até sobrarem 1 ou 2 vértices.

* Dividir para conquistar (Divide and conquer)
- Divide
- Resolve (geralmente usa indução)
- junta as soluções

Exemplos: maior elemento em um vetor de números;
Árvore binária;
Mergesort

- maiorElementoVetor (v, ini, fim)
  se ini == fim
    ret v[ini]
  meio = (ini + fim) / 2
  ret max (maiorElementoVetor (v, ini, fim), maiorElementoVetor (v, meio, fim))

* Exponenciação: a^n → exp (a, n)
  prod = 1
  para i = 1 até n
    prod *= a
  ret prod

$\Theta(n)$

17 / 10 / 18

nova forma de fazer:

$a^n = (a^{n/2})^2 = a^{\lfloor n/2 \rfloor} \times a^{\lceil n/2 \rceil} = (a^{\lfloor n/2 \rfloor})^2 \cdot a^{(n \bmod 2)}$

↑ ímpar

```
exp(a, n)
    se n == 0
        ret 1
    b = exp(a, ⌊n/2⌋)
    b = b x b
    se n é ímpar
        b = b x a
    ret b
```

$O(\log n)$

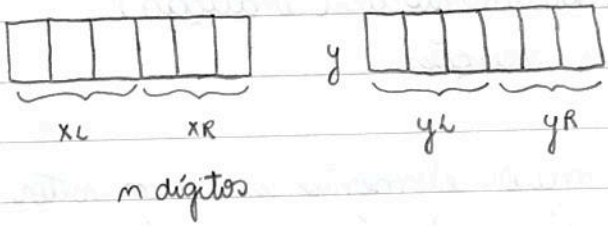
* multiplicação de 2 números com divisão e conquista

Exemplo:

$X = 1001|0000$

$X_L = 1001$

$X_R = 0000$



$$X = 2^{\frac{n}{2}} X_L + X_R$$

$$y = 2^{\frac{n}{2}} y_L + y_R$$

$$X \cdot y = (2^{\frac{n}{2}} X_L + X_R)(2^{\frac{n}{2}} y_L + y_R)$$

$$2^n X_L y_L + 2^{n/2}(X_L y_R + X_R y_L) + X_R y_R$$

3 somas

2 deslocamentos

4 produtos

$$T(n) = 4T\left(\frac{n}{2}\right) + c \cdot n$$

Pelo Teorema mestre:

$n \log 2^4 = n^2$

$O(n^2)$

outra forma de resolver: Algoritmo de Karatsuba

$Z = (X_L + X_R)(y_L + y_R)$ → ignoramos $2^{n/2}$

$\underbrace{X_L y_L}_{Z_1} + \underbrace{X_L y_R + X_R y_L}_{Z_2} + \underbrace{X_R y_R}_{Z_3}$

Sabemos calcular Z_1

$Z_2 = Z - Z_1 - Z_3$

Em Z_1, Z_2 e Z_3 temos:

3 produtos

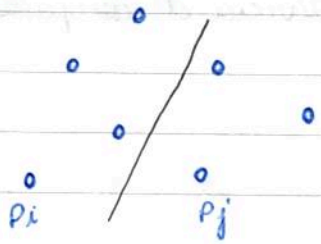
6 somas

2 deslocamentos

$2^n Z_1 + 2^{n/2} Z_2 + Z_3$

$n \log 2^3 \approx n^{1,585}$

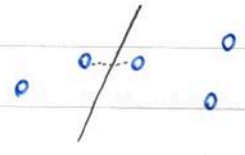
* Menor distância entre 2 pontos

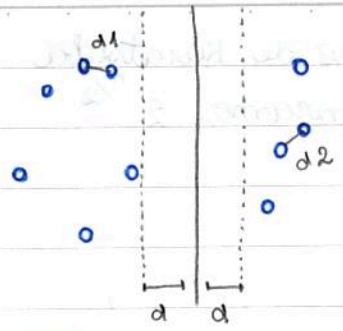


Calcular a distância entre todos os pontos, 2 a 2.

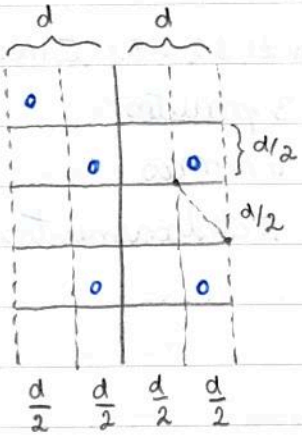
$\text{distância} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} = \Theta(n^2)$

Dividir os pontos em 2 grupos de forma balanceada com uma reta.
Pega as distâncias mínimas de cada lado
E se a distância mínima estiver separada?



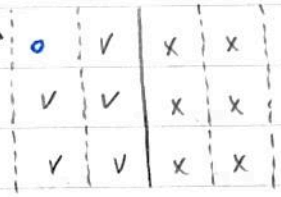


$d = \min(d1, d2)$

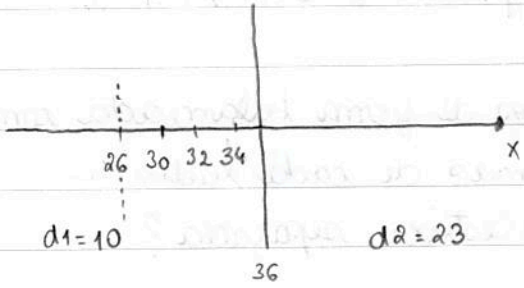


no máximo 1 ponto em cada quadrado.

Procurar todos que estiverem a uma distância "d" do ponto inicial

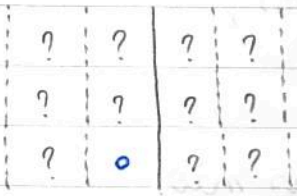


ordena os pontos pelas coordenadas em x:



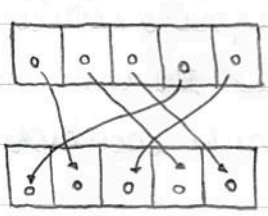
$d = \min(10, 23) = 10$

Ordena em y e pega a distância dos próximos 11 itens próximos ao ponto.

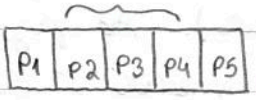


$$T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n \log n$$

não é possível aplicar o Teorema Mestre!
O caso 2 do TM é Θ.



por x

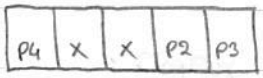


por x

por y

por y

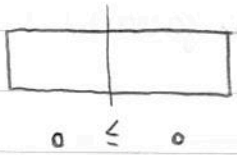
Criaremos um terceiro vetor = P4 P2 P3



Usando recursividade para encontrar os trechos, eliminaremos o log n e fica linear.

Escrever o algoritmo recursivo para ordenar em y. Usar caixa preta para o método que ordena.

* Quicksort
divide ✓
resolve ✓ n log
junta x



* Menor distância (closes Pair)
divide ✓
resolve ✓ n log
junta ✓

* Mergesort
divide x
resolve ✓ n log
junta ✓

* Multiplicação
divide ✓
resolve ✓ $n \log_2 3$
junta ✓

22/10/18

* Algoritmos Gulosos
- Relacionados com otimização (max/min);
- Dijkstra;
- MST (Prim | Kruskal);
- Funciona em alguns casos e em outros não;
- não há uma fórmula definida de quando usar.
- como provar se a solução é ótima? encontrar contradição.

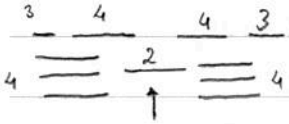
Tenho uma solução ótima. Pego um vértice que não faz parte da solução (o menor). Ao fazer a troca, se o vértice for maior, não pode. Se for menor é contraditório ter uma menor que não faça parte da árvore, que não tenha sido escolhido para a MST.

- outros exemplos fora de grafos:
Temos uma lista de atividades para executar e queremos executar o máximo, maior quantidade de atividades.
1ª estratégia: sempre descartar o item de maior duração.
2ª estratégia: sempre executar o item de menor duração.
3ª estratégia: pegar quem tem menor interseção com outros.

Todas funcionam? Veja contra exemplos:

- menor duração 
só faremos 1 ao invés de 2 ou mais.

- menor interseção 
pegaríamos a do meio e deixaríamos as 2 de baixo



pegamos o de menor interseção e não será a ótima.

algoritmo:

maiorNumeroAtividades(vetor A)

ordene Ai's por fi // os que finalizam primeiro

S = {Ai}

t = t1

para i = 2 até n

se ci > t // começo maior que t

S = S ∪ {Ai}

t = ti

Suponha por absurdo que o algoritmo não encontra o ótimo:

$S = A_{i1}, A_{i2}, \dots, A_{ik}$ $\quad i_1 < i_2 < \dots < i_k$

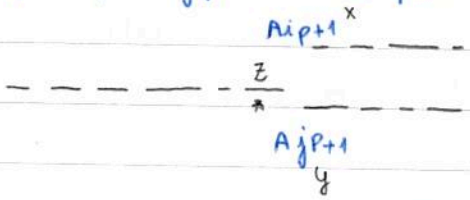
Seja $S^* = A_{j1}, A_{j2}, \dots, A_{jl}$ $\quad j_1 < j_2 < \dots < j_l$ $\quad k < l$

ótimo, maximiza p:

$A_{i1} = A_{j1}, \dots, A_{ip} = A_{jp}$

como S não pode ser um subconjunto de S^* (pelo algoritmo) então:

$A_{ip+1} \neq A_{jp+1}$ $\quad (e\ p+1 \leq k)$

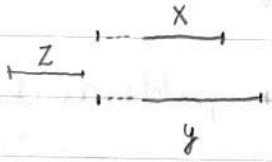


Temos uma solução que em determinado momento possui 2 caminhos

* $f_{ip} = f_{jp} < c_{ip+1}$

$f_{jp} < c_{jp+1}$

$f_{ip+1} \leq f_{jp+1}$



x tem que terminar antes que y.

note que $f_{ip} < c_{ip+1}$, $f_{jp} < c_{jp+1}$ e que $f_{ip} = f_{jp}$

como o algoritmo avalia as atividades em ordem crescente de fi, temos que $f_{ip+1} \leq f_{jp+1}$, seja então $S^{*1} = S^* - \{A_{jp+1}\} \cup \{A_{ip+1}\}$

note que S^{*1} é uma solução válida, pois A_{ip+1} não conflita com A_{jp} (pois ambas pertencem a S) e não conflitam com A_{jp+2}, pois $f_{ip+1} \leq f_{jp+1} < c_{jp+2}$

logo, como $|S^{*1}| = |S^{*}|$, S^{*1} é ótimo, uma contradição, pois contraria a hipótese de que S^* era a solução ótima com o maior número de elementos no início.

* outro exemplo:

Temos um trem que transporta diversos tipos de minério, considerando quantidades diferentes e valores diferentes para cada tipo de minério.

capacidade: 35 toneladas

peso	valor	valor por tonelada	Tipo
20	18	0.90	1
18	16	0.88	2
16	15	0.93	3
14	14	1	4

maximizar o ganho.

35 - 14 tipo 4 ganho 14
21 - 16 tipo 3 ganho 15
5 - 5 tipo 1 ganho 4.5 total: 33.5

Esse é conhecido como problema da mochila ou knapsack.

E quando não pudermos fracionar objetos?

24/10/18

* Programação Dinâmica

Resolvemos problemas através de subproblemas

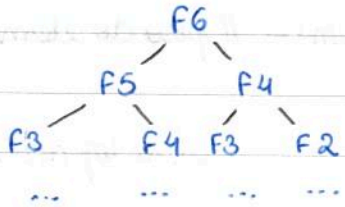
- Sequência de Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, ...

```
fib (n)
    se n = 1 ou n = 2
        ret 1
    senão
        ret fib(n-1) + fib(n-2)
```

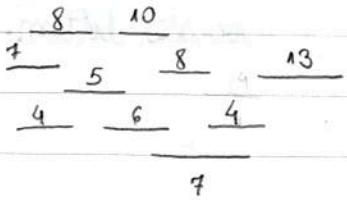
$$\begin{cases} Fn = Fn-1 + Fn-2, & n \geq 3 \\ F1 = F2 = 1 \end{cases}$$

F6 = F5 + F4 ...

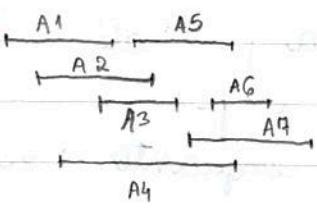


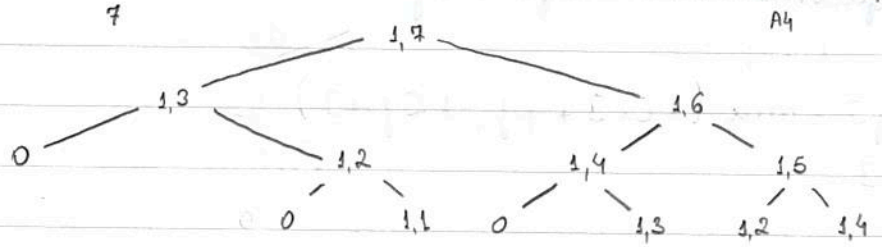
não calcularemos os mesmos resultados mais de uma vez.

- Problema: Estou de férias e quero fazer o máximo de atividades maximizando minha satisfação / alegria (o peso de cada tarefa importa).



outra instância:





O Algoritmo : Recursivo

// mantemos uma tabela para
// gravar os valores. Inicia com -1.

Ai = (ci, fi)
// Atividade com peso

t

-1
-1
-1
-1
...
-1

chamada de memoização ou top-down

```
Atividade (A, n)
    se n == 0
        ret 0
    se t[n] ≠ -1
        ret t[n]   // já foi gravado
    i = n-1
    enquanto i > 0 e fi > cn
        i--;                    → podemos usar busca binária O(log n)
    vi = Atividade (A, i) + pn   // peso do elemento
    v2 = Atividade (A, n-1)
    t[n] = max [v1, v2]
        ret t[n]
```

O(n) n vezes — O(n log n) com busca binária

O(n²) sem busca binária

O algoritmo : Iterativo

versão bottom-up

```
Atividade (A, n)
    t[0] = 0
    para j = 1 até n
        i = j-1
        enquanto i > 0 e fi > cj
            i--;
        t[j] = max (t[i] + pj, t[j-1])
    ret t[n]
```

- outro exemplo: maior subsequência crescente / lis: longest increasing subsequence

Selecionar uma sequência em que os números apareçam em ordem crescente

13, 8, 4, 50, 45, 25, 18, 31, 51, 15

s = 1 1 1 2 2 2 2 3 4 2 → maiores subsequências que terminam no número

s[], n

usando s[n] → s[], n-1 sem usar

Estou procurando o max v[i] em que o último seja menor que n.

s[i] < s[n]

```
para i = 1 até n                    // bottom-up
    v[i] = 1
    para j = 1 até i-1
        se s[j] < s[i] e v[j]+1 > v[i]
            v[i] = v[j]+1
    ret max v[i]   1 ≤ i ≤ n
```

ou:

```
lis(s, n)
    se v[n] ≠ -1
        ret v[n]
    v[n] = 1
    para i = 1 até n-1
        se s[i] < s[n]
        x = lis(s, i)
        se x+1 > v[n]
            v[n] = x+1
    ret v[n]
```

29 / 10 / 18

* Outro exemplo: Envio máximo de mensagens de acordo com o meu orçamento.

Orçamento = 12

m	Valor	mensagens
1	1	3
2	5	8
3	2	4
4	7	10
5	3	4
6	3	5

↑ itens

1, 2, 3 e 6 < 12

1 + 5 + 2 + 3 = 11 < 12 → orçamento

↓ custos

$$\max \begin{cases} S(m-1, c), S(m-1, c-Vm) + Mm & \text{se } m \geq 1, c \geq Vm \\ S(m-1, c) & \text{se } m \geq 1, c < Vm \\ 0 & \text{se } m = 0 \end{cases}$$

Tabela:

m/c	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	(0)	0	0	0	0	0	0	0	0	0	0	0
→ 1	0	3	(3)	3	3	3	3	3	3	3	3	3	3
→ 2	0	3	3	3	3	8	11	(11)	11	11	11	11	11
→ 3	0	3	4	7	7	8	11	12	15	(15)	15	15	15
4	0	3	4	7	7	8	11	12	15	(15)	17	17	18
5	0	3	4	7	7	8	11	12	15	(15)	17	19	19
→ 6	0	3	4	7	8	9	12	12	15	16	17	(20)	(20)

↓ menor custo

Quais escolhi? seguir a seta e ver onde houve aumento.

1, 2, 3 e 6.

Note que temos 2 opções com máximo de mensagens = 20. Qual a mais barata? sempre a que estiver a esquerda.

* Multiplicação de Matrizes:

A1 A2 A3 A4 A5

A1 A2 ... AN

Ai = ai-1 x ai

a0 = 1000

a1 = 1000

a2 = 10

a3 = 100

$P(1,5) = \min_{i \le j \le 4}$

$P(i,k) = \min_{i \le j < k}$

$$\begin{cases} P(1,j) + P(j+1,k) + a_{i-1} a_j a_k, & \text{se } i < k \\ 0, & \text{caso contrário} \end{cases}$$

	1	2	3	4	5
1	0	0 →	0 →	0 →	0
2	x	0	0 →	0 →	0
3	x	x	0	0 →	0
4	x	x	x	0	0
5	x	x	x	x	0

1,4 = A1 A2 A3 A4

A1 x A2 A3 A4

A1 A2 x A3 A4

A1 A2 A3 x A4