Introdução à Análise Algoritmos

Notas de aula da disciplina IME 04-10820 Estruturas de Dados I

Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

agosto/2019

Ordenação por SELEÇÃO:

Idéia: Dado um vetor com n elementos, realizar n passos de seleção onde, em cada passo, seleciona-se o menor elemento ainda não escolhido.

Exemplo:

	1	2	3	4	5	6	7
0	E	X	E	M	P	L	O
1	E	X	E	M	P	L	O
2	E	E	X	M	P	L	O
3	E	E	L	M	P	X	O
4	E	E	L	M	P	X	O
5	E	E	L	M	O	X	P
6	E	E	L	M	O	P	X

Ordenação por SELEÇÃO:

```
Seleção():    #dados: vetor V, |V|= n
   para i ← 1..n-1 incl.:
      m ← i

      para j ← i+1..n incl.:
         se V[j] < V[m]:
            m ← j

      V[i], V[m]  ←  V[m], V[i]
```


Quantas instruções são executadas na Ordenação por SELEÇÃO, no pior caso (vetor inversamente ordenado)?

```
Seleção():    #dados: vetor V, |V|= n
   para i ← 1..n-1 incl.:                 (n-1)t_p
      m ← i                               (n-1)t_a

      para j ← i+1..n incl.:              n(n-1)t_p/2
         se V[j] < V[m]:                  n(n-1)t_s/2
            m ← j                         n(n-1)t_a/2

      V[i], V[m]  ←  V[m], V[i]           3(n-1)t_a
```

Total: $c_0.n^2+c_1.n+c_2$

Quantas instruções são executadas na Ordenação por SELEÇÃO, no melhor caso (vetor ordenado)?

```
Seleção():    #dados: vetor V, |V|= n
   para i ← 1..n-1 incl.:                 (n-1)t_p
      m ← i                               (n-1)t_a

      para j ← i+1..n incl.:              n(n-1)t_p/2
         se V[j] < V[m]:                  n(n-1)t_s/2
            m ← j                         0

      V[i], V[m]  ←  V[m], V[i]           3(n-1)t_a
```

Total: $c_0.n^2+c_1.n+c_2$

Contagem de instruções - SELEÇÃO:

```
Seleção():    #dados: vetor V, inteiro n
   para i ← 1..n-1 incl.:
      m ← i

      para j ← i+1..n incl.:
         se (V[j] < V[m]):
            m ← j;

      V[i], V[m]  ←  V[m], V[i]
```

Tep(n) = tempo de Seleção, no pior caso =

$(n-1)(t_p+4t_a) + (\Sigma_{1 \le j \le n-1} (t_p+t_s+t_a)j) =$
$n^2(t_p+ t_s+t_a)/2 + n(t_p- t_s+7t_a)/2 - (t_p+4t_a)= a_1.n^2 + b_1.n + c_1.$

Tem(n) = tempo de Seleção, no melhor caso =

$n^2(t_p+ t_s)/2 + n(t_p- t_s+8t_a)/2 - (t_p+4t_a) = a_2.n^2 + b_2.n + c_2.$

Ordenação por INSERÇÃO:

Idéia: Dado um vetor com n elementos, realizar n passos de inserção onde, em cada passo, um novo elemento é inserido a um subconjunto ordenado.

Exemplo:

	1	2	3	4	5	6	7
0	E	X	E	M	P	L	O
1	E	X	E	M	P	L	O
2	E	E	X	M	P	L	O
3	E	E	M	X	P	L	O
4	E	E	M	P	X	L	O
5	E	E	L	M	P	X	O
6	E	E	L	M	O	P	X

Ordenação por INSERÇÃO:

```
Inserção():   #dados: vetor V, |V| = n
   para i ← 2..n incl.:
      j ← i;   V[0] ← V[i];

      enquanto V[j-1] > V[0]:
         V[j] ← V[j-1]
         j ← j-1

      V[j] ← V[0]
```


Quantas instruções são executadas na Ordenação por INSERÇÃO, no pior caso (vetor inversamente ordenado)?

```
Inserção():   #dados: vetor V, |V| = n
   para i ← 2..n incl.:                        (n-1)tp
      j ← i;   V[0] ← V[i];                    2(n-1)ta

      enquanto V[j-1] > V[0]:                  n(n-1)(ta+tsu)
         V[j] ← V[j-1]
         j ← j-1                               n(n-1)/2(ts+tsu)

      V[j] ← V[0]                              (n-1)ta
```

Total: $c_0.n^2+c_1.n+c_2$

Quantas instruções são executadas na Ordenação por INSERÇÃO, no melhor caso (vetor ordenado)?

```
Inserção():   #dados: vetor V, |V| = n
   para i ← 2..n incl.:                        (n-1)tp
      j ← i;   V[0] ← V[i];                    2(n-1)ta

      enquanto V[j-1] > V[0]:                  (n-1)(ta+tsu)
         V[j] ← V[j-1]
         j ← j-1                               0

      V[j] ← V[0]                              (n-1)ta
```

Total: $c_1.n+c_2$ (!!)

Contagem de instruções - INSERÇÃO:

```
Inserção():   #dados: vetor V, |V|=n
   para i ← 2..n incl.:
      j ← i;   V[0] ← V[i];

      enquanto V[j-1] > V[0]:
         V[j] ← V[j-1]
         j ← j-1

      V[j] ← V[0]
```

Tep(n) = tempo de Inserção, no pior caso =

$(n-1)(t_p + 3t_a + t_a) + (\Sigma_{1 \le j \le n-1} (t_{su}+t_s+2t_{su}+2t_a)j) =$

$n^2(t_s+3t_{su}+2t_a)/2 + n(2t_p+6t_a-t_s-3t_{su})/2 - (t_p+4t_a) =$

$a_1.n^2 + b_1.n + c_1.$

Tem(n) = tempo de Inserção, no melhor caso =

$(n-1)(t_p+4t_a+ t_s+ t_{su}) = \quad b_2.n + c_2.$

COMPLEXIDADE DE ALGORITMOS

É uma importante medida sobre a eficiência de um algoritmo, em termos do tempo de execução ou da memória requerida, em função do tamanho da entrada.

- -Analisa-se o pior caso, em geral
- -Limita-se à instrução mais executada
- -Muitas vezes faz-se uma medição indireta do número de execuções da instrução mais executada. Para ordenação e busca, basta-se contar o número de comparações no processo.
- -Expressa-se os resultados em termos de Ordem de Grandeza

COMPLEXIDADE DE ALGORITMOS

Contagem de comparações na SELEÇÃO

Total de comparações: 21

Exemplo:

	1	2	3	4	5	6	7	
0	E	X	E	M	P	L	O	
1	E	X	E	M	P	L	O	6
2	E	E	X	M	P	L	O	5
3	E	E	L	M	P	X	O	4
4	E	E	L	M	P	X	O	3
5	E	E	L	M	O	X	P	2
6	E	E	L	M	O	P	X	1

Contagem de comparações na INSERÇÃO

Total de comparações: 14

Exemplo:

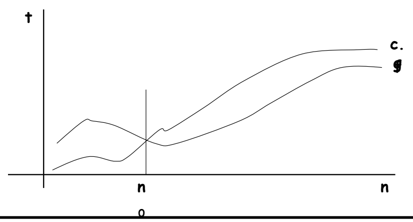
	1	2	3	4	5	6	7	
0	E	X	E	M	P	L	O	
1	E	X	E	M	P	L	O	1
2	E	E	X	M	P	L	O	2
3	E	E	M	X	P	L	O	2
4	E	E	M	P	X	L	O	2
5	E	E	L	M	P	X	O	4
6	E	E	L	M	O	P	X	3

Exercício:

Mostrar os passos da ordenação do MIXSTRING (10 letras, misturando letras do nome de 2 alunos) por SELEÇÃO E INSERÇÃO. Contar comparações.

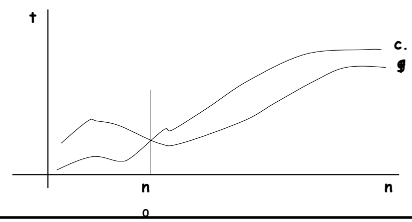
COMPORTAMENTO ASSINTÓTICO DE FUNÇÕES

É uma forma matemática de expressar, simplificadamente, a quantidade de execuções da instrução mais executada em um algoritmo, em função do tamanho da entrada.



Função O (limite superior)

Def: Dadas duas funções f e g sobre N, diz-se que f é O(g), se existirem c e n_0 positivos tal que, $n > n_0 \Rightarrow f(n) \leq c.g(n)$.



Complexidade de Algoritmos

Exemplos:

$f(n) = 10.\log_2 n + 5n$

f é O(n), ou seja g(n) = n, pois, para n > 2,

$f(n) = 10.\log_2 n + 5n \leq 10n + 5n = 15n = 15.g(n)$.

$f(n) = a_o n^k + a_1 n^{k-1} + \ldots + a_k n^0$

f é $O(n^k)$, ou seja $g(n) = n^k$, pois, para n > 0,

$f(n) \leq c.x^k = c.g(n)$, onde $c = |a_o|+|a_1|+\ldots+|a_k|$.

Complexidade de Algoritmos

ORDEM DE GRANDEZA - tempos de execução

n\f(n)	log n	n	n.log n	n^2	n^3	2^n
10	0.000003	0.00001	0.00003	0.0001	0.001	0.01
100	0.000007	0.0001	0.0007	0.01	1	10^{25} anos
1.000	0.00001	0.001	0.01	1	17 min	---
10.000	0.00001	0.01	0.13	2 min	14 dias	---
100.000	0.00002	0.1	1.7	2.7 horas	30 anos	---

Análise da Ordenação por SELEÇÃO:

Complexidade:
Pior caso: $O(n^2)$
Melhor caso: $O(n^2)$

Estabilidade (manutenção da ordem relativa de chaves iguais):
Algoritmo não estável

Memória adicional:
Nenhuma

Usos especiais:
Baixo número de chaves

Análise da Ordenação por INSERÇÃO:

Complexidade:
Pior caso: $O(n^2)$
Melhor caso: $O(n)$

Estabilidade (manutenção da ordem relativa de chaves iguais):
Algoritmo estável

Memória adicional:
Nenhuma

Usos especiais:
Arquivos "quase-ordenados"

Exercício:

Provar que se $f(n) = O(n^k)$ Então $f(n) = O(n^{k+1})$.

Ex: Problema Soma de Duplas (Versão 1):

Dado um vetor ordenado com n números distintos, determinar a quantidade de duplas cuja soma seja dado valor s.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	11	20	24	25	30	32	35	36	39	41	45

s = 50 (5, 45) (11, 39) (20, 30)

```
SomaDuplas1():   #dados: vetor V, |V| = n, inteiro s

    d ← 0
    para i ← 1..n-1 incl.:
        para j ← i+1..n incl.;
            se V[i]+V[j] = s:
                d ← d+1

    escrever(d)
```


Ex: Problema Soma de Duplas (Versão 1):

Dado um vetor ordenado com n números distintos, determinar a quantidade de duplas cuja soma seja dado valor s.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	11	20	24	25	30	32	35	36	39	41	45

s = 50

Complexidade do Algoritmo 1:
Instruções mais executadas:
se V[i]+V[j]= s:
d ← d+1
Número de execuções:
$(n-1)+(n-2)+\ldots+1 = (n-1)n/2 = O(n^2)$

Ex: Problema Soma de Duplas:

Dado um vetor ordenado com n números distintos, determinar a quantidade de duplas cuja soma seja dado valor s.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	11	20	24	25	30	32	35	36	39	41	45

s = 50

Como melhorar o algoritmo para se ter uma solução O(n)?

Ex: Problema Soma de Duplas:

Dado um vetor ordenado com n números distintos, determinar a quantidade de duplas cuja soma seja dado valor s.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	11	20	24	25	30	32	35	36	39	41	45

s = 50

Como melhorar o algoritmo para se ter uma solução O(n)?

Ideia: ordenar o arquivo e processar os elementos da esquerda para a direita procurando se cada um deles tem um par. Note que a procura desse par deve ser feita da direita para a esquerda(!!). O algoritmo, então, usa dois ponteiros, i e j, i começando com 1 e j com n. Quando encontramos um par, V[i]+V[j]=s, o próximo par só pode estar no intervalo de i+1 até j-1(!!). Se, para o elemento da posição i não foi encontrado um par na posição j, então se V[i]+V[j] < s, não há esperança de encontrar um par para esse elemento. Portanto devemos avançar i. Se, entretanto, V[i]+V[j] > s, ainda há esperança de encontrar o par, diminuindo j. O algoritmo correspondente está na página seguinte.

Ex: Problema Soma de Duplas (Versão 2):

Dado um vetor ordenado com n números distintos, determinar a quanti-dade de duplas cuja soma seja dado valor s.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	11	20	24	25	30	32	35	36	39	41	45

s = 50 (5, 45) (11, 39) (20, 30)

```
SomaDuplas2():   #dados: vetor V, |V|=n, inteiro s
    d ← 0
    i ← 1;   j ← n;
    enquanto i < j:
        se V[i]+ V[j] < s:
            i ← i+1
        senão se V[i] + V[j] > s:
            j ← j-1
        senão:
            d ← d+1;   i ← i+1;   j ← j-1;
    escrever(d)
```


Ex: Problema Soma de Duplas:

Dado um vetor ordenado com n números distintos, determinar a quantidade de duplas cuja soma seja dado valor s.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	5	8	11	20	24	25	30	32	35	36	39	41	45

s = 50

Complexidade do Algoritmo 2:

Instruções mais executadas:

```
se V[i]+V[j]= s:
    d ← d+1
```

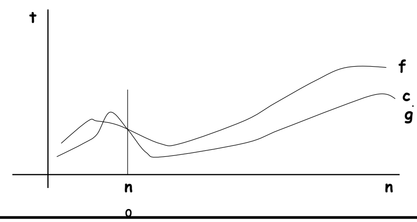
Número de execuções:

≤ n, porque a cada passo do loop o intervalo entre i e j decresce de 1 ou 2.

O Algoritmo é O(n)

Função Ω (limite inferior)

Def: Dadas duas funções f e g sobre N, diz-se que f é Ω(g), se existirem c e n_0 positivos tal que, $n > n_0 \Rightarrow c.g(n) \le f(n)$.



Exemplos:

$f(n) = 10.\log_2 n + 5n$

f é Ω(log n), ou seja g(n) = log n pois, p/ n > 2, e qualquer a > 1,

$f(n) = 10.\log_2 n + 5n \ge 10.\log_2 n + 5.\log_2 n =$
$15.\log_2 n = 15.\log_2 a.\log_a n = c.g(n)$

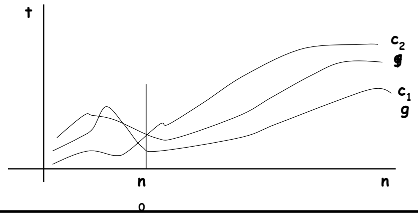
$f(n) = a_o n^k + a_1 n^{k-1} + \ldots + a_k n^0$, com $a_i \ge 0$

f é $\Omega(n^k)$, ou seja $g(n) = n^k$ pois, para n > 0,

$f(n) \ge c.n^k = c.g(n)$, onde $c = \min_{1 \le i \le n} a_i$.

Função Θ (limite assintótico firme)

Def: Dadas duas funções f e g sobre N, diz-se que f é Θ(g), se existirem c_1, c_2 e n_0 positivos tal que, $n > n_0 \Rightarrow c_2.g(n) \geq f(n) \geq c_1.g(n)$.



Exemplo:

$f(n) = a_o n^k + a_1 n^{k-1} + \ldots + a_k n^0$, com $a_i \geq 0$

f é $\Theta(n^k)$, ou seja $g(n) = n^k$ pois, para $n > 0$,

$c_1.n^k \leq f(n) \leq c_2.n^k$, onde

$c_1 = \min_{1 \leq i \leq n} a_i$.
$c_2 = \Sigma_{0 \leq i \leq k} a_i$.

Ferramentas adicionais para a Análise de Algoritmos:

Indução Finita

Recorrências

Recursão

Indução Finita

Técnica para provar teoremas sobre números naturais. A prova é composta de duas partes:

a) tem-se que mostrar que o teorema vale para casos particulares (n = 0 ou 1 ou 2, etc)

b) Supondo-se o teorema válido para valores inferiores a n, prova-se que também é válido para n.

Indução Finita – Exemplo

Teorema: $S_2(n) = \Sigma_{1 \leq i \leq n} i^2 = n(n+1)(2n+1)/6$.

Prova:
a) Para n = 1, $S_2(1) = 1(1+1)(2*1+1)/6 = 1$, OK.

b) Supondo-se o teorema válido para valores inferiores a n, temos:
$S_2(n-1) = (n-1)n(2n-1)/6$. Como $S_2(n)= S_2(n-1)+n^2$, temos:

$S_2(n) = (n-1)n(2n-1)/6+n^2 = ((n-1)n(2n-1)+6n^2)/6 =$
$= n((n-1)(2n-1)+6n)/6 = n(n^2+3n+1)/6 =$
$= n(n+1)(2n+1)/6$,
i.e. o teorema também é verdadeiro para n.

Exercício:

Provar, por **indução finita**, que

$S_1(n) = \Sigma_{1 \leq i \leq n} i = n(n+1)/2$.

Recorrências

Funções sobre números naturais, definidas usando autoreferência. A definição é composta de duas partes:

a) Define-se diretamente a função para alguns números particulares (n = 0 ou 1 ou 2, etc)

b) Em geral, define-se a função para um valor n, lançando-se mão, nessa definição, da mesma função, para valores inferiores a n.

Recorrências – Exemplos

Fatorial:
Fat(0) = 1;
Fat(n) = n.Fat(n-1), para n > 0.

Fibonacci:
Fib(0) = 0;
Fib(1) = 1;
Fib(n) = Fib(n-1)+Fib(n-2), para n > 1.

S2:
$S_2(0) = 0$;
$S_2(n) = S_2(n-1) + n^2$, para n > 0;

Fórmulas fechadas para Recorrências

a) Indução Finita – **Exemplos:**

S_2:
$S_2(0) = 0$;
$S_2(n) = S_2(n-1) + n^2$, para n > 0;

$\Rightarrow$

$S_2(n) = n(n+1)(2n+1)/6$ (já provado)

Fórmulas fechadas para Recorrências

a) Indução Finita – **Exemplos:**

Fibonacci:
Fib(0) = 0; Fib(1) = 1;
Fib(n) = Fib(n-1)+Fib(n-2), para n > 1.
$\Rightarrow$ Fib(n) = $(a^n - \hat{a}^n)/5^{1/2}$, $a = (1+5^{1/2})/2$ $\hat{a} = (1-5^{1/2})/2$

Prova:

a) Verdade para n = 0 e n = 1.
b) Suponhamos válida para valores menores que n. Então:

$$
\begin{aligned}
Fib(n) &= (a^{n-2} + a^{n-1} - \hat{a}^{n-2}-\hat{a}^{n-1})/5^{1/2} = \\
&(a^{n-2}((1+5^{1/2})/2+1)- \hat{a}^{n-2}((1-5^{1/2})/2+1))/5^{1/2} = \\
&(a^{n-2}(3+5^{1/2})/2-\hat{a}^{n-2}(3-5^{1/2})/2)/5^{1/2} = \\
&(a^{n-2}((1+5^{1/2})/2)^2-\hat{a}^{n-2}((1-5^{1/2})/2)^2)/5^{1/2} = \\
&(a^n - \hat{a}^n)/5^{1/2};
\end{aligned}
$$

ou seja, o resultado também vale para n.

Fórmulas fechadas para Recorrências

b) Iteração – **Exemplo:**

T(1) = 1;
T(n) = n + T(n/3), para n > 1.
$\Rightarrow$ T(n) = 3n/2 – 1/2

Prova:

$$
\begin{aligned}
T(n) &= \\
&n + T(n/3) = n + n/3 + T(n/9) = \\
&n + n/3 + \ldots + T(n/3^x)
\end{aligned}
$$

Para $(n/3^x) = 1$, temos $x = \log_3 n$

$T(n) = \Sigma_{0 \le i \le x-1} n/3^i + T(n/3^x) = n(1-1/3^x)/(1-1/3)+1 = 3n/2 - 1/2$.

Fórmulas fechadas para Recorrências

c) Manipulação de Somas – **Exemplo:**

S_2:
$S_2(0) = 0$;
$S_2(n) = S_2(n-1) + n^2$, para n > 0; $\Rightarrow$ $S_2(n) = n(n+1)(2n+1)/6$

Prova:

$S_3(n)+(n+1)^3 = \Sigma_{0 \le i \le n}(i+1)^3 = \Sigma_{0 \le i \le n}(i^3+3i^2+3i+1) =$

$S_3(n)+3S_2(n)+3S_1(n)+n+1$

$3S_2(n) = (n+1)^3 - 3n(n+1)/2 - (n+1) = (n+1)/2(2n^2+4n+2-3n-2) =$

$(n+1)/2(2n^2+n) = n(n+1)(2n+1)/2$

Finalmente, $S_2(n) = n(n+1)(2n+1)/6$

Fórmulas fechadas para Recorrências

d) Caso Particular

T:

$T(1) = c;$

$T(n) = T(n-1) + P_1(k)$, para $n > 1$; $\Rightarrow T(n) = P_2(k+1);$

S_2:

$S_2(0) = 0;$

$S_2(n) = S_2(n-1) + n^2$, para $n > 0$; $\Rightarrow S_2(n) = n(n+1)(2n+1)/6$

$S_2(n) = an^3+bn^2+cn+d$; $\Rightarrow$ $d = 0$, considerando $n = 0$;

Formamos o sistema, considerando n = 1, 2 e 3:

$a+b+c = 1;$

$8a+4b+2c = 5;$

$27a+9b+3c = 14;$

Resolvendo o sistema, temos: $a = 1/3$; $b = 1/2$; $c = 1/6$

ou $S_2(n) = n(n+1)(2n+1)/6$

Fórmulas fechadas para Recorrências

Outras maneiras de resolver:

e) Trocar somas por integrais

f) Usar cálculo finito

g) Usar funções geradoras

Exercício:

Seja a recorrência:

$T(0) = 0;$

$T(n) = 2.T(n-1) + 1$, para $n > 0$;

Provar, por indução finita, que a solução da recorrência é $T(n) = 2^n - 1$

Problema Cinema –

Um grupo de c amigos quer ir ao cinema e sentar o mais próximo possível. É dado o mapa das disponibilidades de assento (uma matriz n x n, contendo 1 quando o assento já está vendido e 0, cc). O critério de proximidade é a menor submatriz (de menor produto "num linhas x num. de colunas") correspondente a, no mínimo, c lugares vagos.

Escrever um algoritmo, de complexidade máxima $O(n^3)$ para resolver esse problema. A entrada são os valores n e c, seguidos da matriz n x n. A saída deve ser a "área" da menor matriz (produto num. linhas x num. colunas).

Problema Cinema – Exemplo, com n = 10, c = 8

Matriz:

1	0	0	1	1	1	0	1	0	1
0	0	1	0	1	0	1	0	0	0
1	0	1	0	1	0	0	1	1	1
0	1	1	1	0	1	1	0	0	0
1	1	1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1	1	0
0	0	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	0
0	0	0	0	0	1	0	0	1	1
1	0	1	1	1	0	1	0	1	0

A solução é a submatriz cujo ponto superior esquerdo é (2,4) e inferior direito, (3,10), com "área" = 14)

Problema Cinema - Versão I - $O(n^6)$

```
Cinema1():   #dados: Matriz M, |M| = nxn
    min ← n*n+1
    para i ← 1..n incl.:
        para j ←1..n incl.:
            para p ← i..n incl.:
                para q ← j..n incl.:
                    t ← 0;
                    para k ← i..p incl.:
                        para l ← j..q incl.:
                            se M[k,l] = 0:
                                t ← t+1
                    se t ≥ c & min > (p-i+1)*(q-j+1):
                        min ← (p-i+1)*(q-j+1)
    escrever (min)
```

Problema Cinema - Versão I

Demonstração da complexidade de Cinema:

A instrução mais executada é a condição mais interna, que corresponde a "varrer" todas as submatrizes consideradas. Portanto, o número de vezes que essa instrução é executada, corresponde à "área" total dessas submatrizes. Existem (n-i+1)*(n-j+1) submatrizes distintas com i linhas e j colunas. Portanto, a "área" total é dada por:

$S(n) = \Sigma_{1 \le i \le n, 1 \le j \le n}(n-i+1)*i*(n-j+1)*j =$

$(n(n+1)(n+2)/6)^2 =$
$O(n^6)$

Problema Cinema – Idéia de um algoritmo $O(n^4)$

Geração da matriz acumulada – Usa coluna 0 e a linha 0 auxiliares

1	0	0	1	1	1	0	1	0	1
0	0	1	0	1	0	1	0	0	0
1	0	1	0	1	0	0	1	1	1
0	1	1	1	0	1	1	0	0	0
1	1	1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1	1	0
0	0	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	1	0
0	0	0	0	0	1	0	0	1	1
1	0	1	1	1	0	1	0	1	0

0	0	0	0	0	0	0	0	0	0	0
0	0	1	2	2	2	2	3	3	4	4
0	1	3	4	5	5	6	7	8	10	11
0	1	4	5	7	7	9	11	12	14	15
0	2	5	6	8	9	11	13	15	18	20
0	2	5	6	8	9	12	14	16	19	21
0	2	5	6	8	10	13	15	17	20	23
0	3	7	8	11	13	17	19	22	25	28
0	3	7	8	11	13	18	20	23	26	30
0	4	9	11	15	18	23	26	30	33	37
0	4	10	12	16	19	25	27	33	36	41

Para testar o número de zeros da submatriz definida por (5,6) e 10,9),
faz-se MA(10,9)-MA(10,5)-MA(4,9)+MA(4,5).
No exemplo: 36 - 19 - 18 + 9 = 8

Problema Cinema - Versão II - $O(n^4)$

```
Cinema2():   #dados: Matriz M, |M| = n × n
    min ← n*n+1
    para i ← 1..n incl.:
        para j ← 1..n incl.:
            para p ← i..n incl.:
                para q ← j..n incl.:
                    t ← MA[p,q]-MA[i-1,q]-MA[p,j-1]+MA[i-1,j-1]
                    se t ≥ c & min > (p-i+1)*(q-j+1):
                        min ← (p-i+1)*(q-j+1)
    escrever (min)
```


Problema Cinema - Exercício:

Gerar a matriz acumulada(n = 5) e indicar a melhor solução para m = 6

0	0	1	1	1
1	1	1	0	0
1	0	1	0	1
0	1	0	1	0
1	0	0	1	1

0	0	0	0	0	0
0					
0					
0					
0					
0					

Problema Cinema - Idéia de um algoritmo $O(n^3)$

a) Gerar a matriz de zeros acumulados, tal que o teste para verificar se qualquer submatriz atende à restrição de ter no mínimo c zeros, seja feita em $O(1)$.
A geração é em $O(n^2)$.

b) Considerar todas as submatrizes distintas de colunas 1 a n, compreendidas entre duas linhas. Isto pode ser feito em $O(n^2)$.

c) Para cada uma das submatrizes do ítem b), fazer uma varredura descobrindo a matriz de menor número de colunas que satisfaz à restrição e ir guardando a de menor área. Isto deve ser feito em $O(n)$.

d) Desta forma, a complexidade total é $O(n^3)$.

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 1 - Geração da matriz acumulada - Usa coluna 0 e a linha 0 auxiliares

0	0	0	0	0	0	0	0	0	0	0
0	0	1	2	2	2	2	3	3	4	4
0	1	3	4	5	5	6	7	8	10	11
0	1	4	5	7	7	9	11	12	14	15
0	2	5	6	8	9	11	13	15	18	20
0	2	5	6	8	9	12	14	16	19	21
0	2	5	6	8	10	13	15	17	20	23
0	3	7	8	11	13	17	19	22	25	28
0	3	7	8	11	13	18	20	23	26	30
0	4	9	11	15	18	23	26	30	33	37
0	4	10	12	16	19	25	27	33	36	41

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 2 - Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 7 e 9.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

Para encontrar a menor submatriz compreendida entre as linhas 7 a 9, o ponteiro j indicará a coluna de fim e o ponteiro i, a coluna de início. Inicialmente ambos estão na coluna 1.

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 2 - Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 1 e 2.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

O ponteiro j caminha para a direita e pára na coluna 5, o primeiro ponto que atende às restrições. O ponteiro i permanece parado na coluna 1. A submatriz encontrada satisfaz às restrições e tem "área" = 3*5 = 15.

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 2 - Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 1 e 2.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

O ponteiro j vai para a coluna 6, que atende às restrições. O ponteiro i vai para a coluna 2. A submatriz encontrada satisfaz às restrições e tem "área" = 3*5 = 15, igual à da primeira submatriz.

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 2 - Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 1 e 2.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

O ponteiro j vai para a coluna 7. O ponteiro i permanece na coluna 2. A submatriz tem "área" = 3*6 = 18, pior que as anteriores.

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 2 - Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 1 e 2.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

O ponteiro j vai para a coluna 8. O ponteiro i vai para a coluna 4. A submatriz tem "área" = 3*5 = 15, igual a das primeiras.

Problema Cinema - Idéia de um algoritmo $O(n^3)$

Etapa 2 - Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 1 e 2.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

O ponteiro j vai para a coluna 9. O ponteiro i permanece na coluna 4. A submatriz tem "área" = 3*6 = 18, pior que a das primeiras.

Problema Cinema – Idéia de um algoritmo $O(n^3)$
Etapa 2 – Geração das submatrizes compreendidas entre linhas.
Exemplo de como seria tratada a submatriz entre as linhas 1 e 2.

linha 6	0	2	5	6	8	10	13	15	17	20	23
linha 7	0	3	7	8	11	13	17	19	22	25	28
linha 8	0	3	7	8	11	13	18	20	23	26	30
linha 9	0	4	9	11	15	18	23	26	30	33	37

Finalmente, o ponteiro j chega à coluna 10. *O* ponteiro i permanece na coluna 4. A submatriz tem "área" = 3*7 = 21, pior que a das primeiras. Portanto, para essa combinação de linhas, a melhor submatriz tem "área" = 15.

FIM