PANC: Projeto e Análise de Algoritmos

Aula 09: Divisão e Conquista - Subarranjo Máximo e Multiplicação de Matrizes

Breno Lisi Romano

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SÃO PAULO
Campus São João da Boa Vista

Sumário

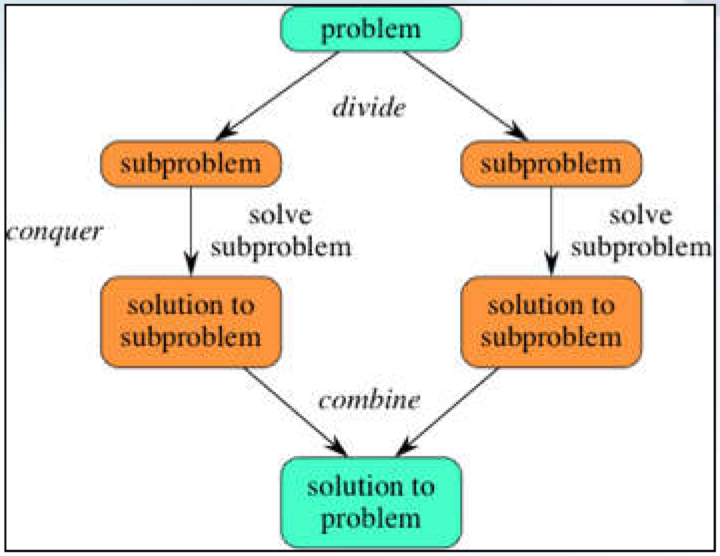
- Revisão de Conteúdo
- O Problema do Subarranjo Máximo
- O Problema da Multiplicação de Matrizes – Algoritmo de Strassen

Recapitulando... (1)

- Divisão e Conquista (ou Dividir e Conquistar, ou ainda, D&C) é um **paradigma de solução de problemas** no qual tentamos **simplificar a solução do problema** original **dividindo-o em subproblemas** menores e **resolvendo-os** (ou "conquistando-os") separadamente
- O processo:
  - **Dividir** o problema original em **subproblemas** – normalmente com a metade (ou algo próximo disto) do tamanho do problema original, porém com a mesma estrutura
  - **Conquistar**, ou determinar a solução dos subproblemas, comumente, de maneira recursiva – que agora se tornam mais "fáceis"
  - Se necessário, **combinar** as soluções dos subproblemas para produzir a **solução completa** para o problema original

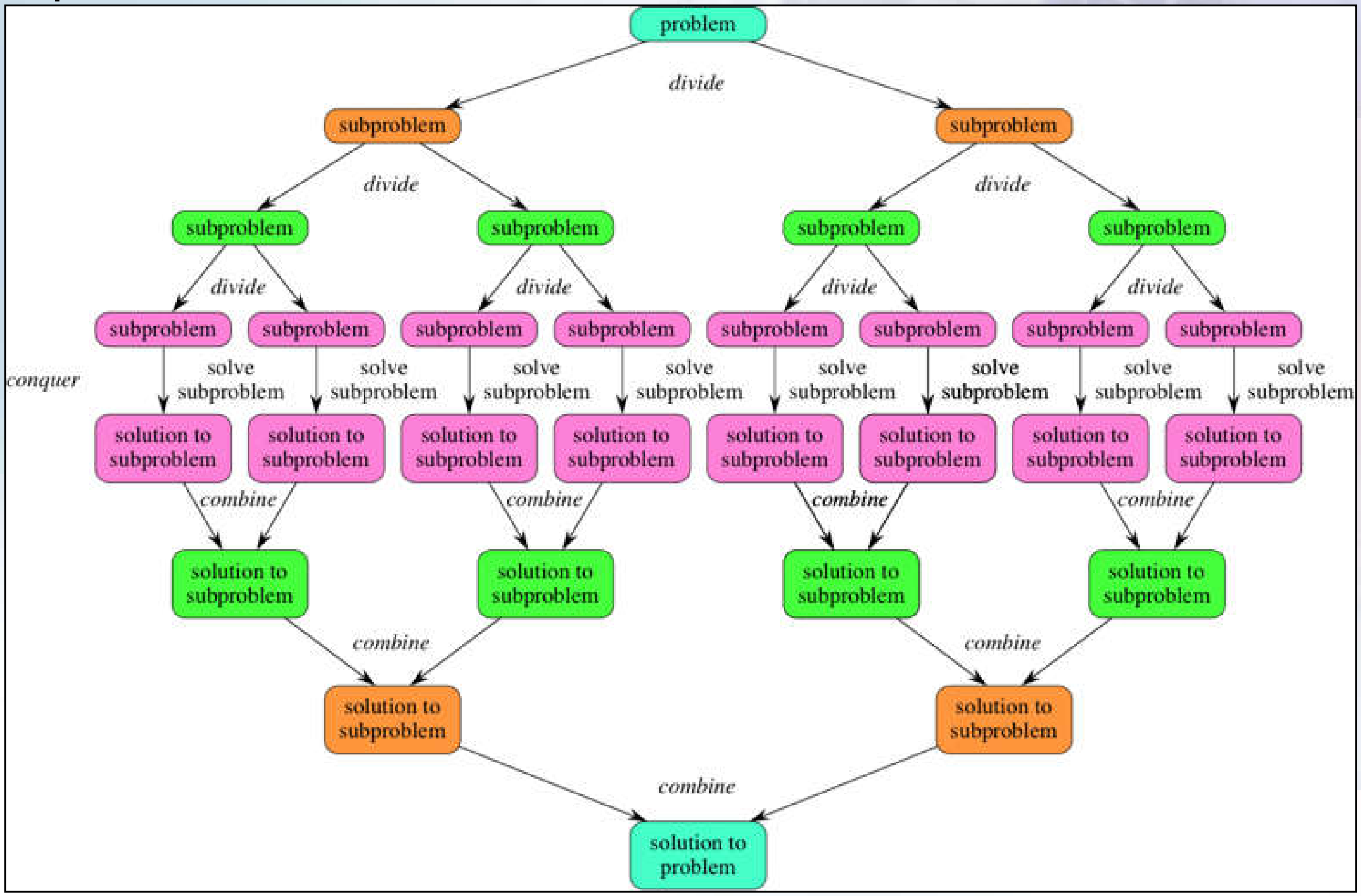
Recapitulando... (2)

- **Passos** de um algoritmo de Divisão e Conquista: **Dividir**, **Conquistar**, **Combinar**
- **Ilustração**: Visualizar um passo, assumindo que cada passo de dividir cria dois subproblemas
  - Alguns algoritmos de dividir-e-conquistar criam mais de dois



Recapitulando... (3)

- **Ilustração**: Se expandirmos mais duas etapas recursivas, ele terá esta aparência



Como se cria pelo menos dois subproblemas, um algoritmo de Divisão e Conquista faz múltiplas chamadas recursivas

Recapitulando... (4)

- Existem **quatro condições** que indicam se o paradigma Divisão e Conquista pode ser aplicado com sucesso:
  - Deve ser possível **dividir o problema** em **subproblemas**
  - A **combinação** de **resultados** deve ser **eficiente**
  - Os **subproblemas** devem possuir **tamanhos parecidos** dentro de um mesmo nível
  - A **solução** dos **subproblemas** são **operações repetidas** ou correlacionadas

Recapitulando... (5)

- **Recorrências e D&C:**
  - As **recorrências** estão **diretamente relacionadas** com o **D&C**, pois caracterizam naturalmente o **tempo de execução** destes algoritmos
  - Para o MergeSort(), por exemplo:
    - $T(1) = 1$
    - $T(n) = 2T(n/2) + n$, para $n > 1$

- **Processo de divisão é sempre ao Meio?**
  - Um algoritmo D&C pode dividir um problema em partes de diferentes tamanhos, como 2/5 e 3/5
  - Por exemplo:
    - Supondo que o processo de combinar soluções seja linear, temos:
      - $T(n) = T(2n/5) + T(3n/5) + n$

Recapitulando... (6)

- **Processo de divisão é sempre ao Meio?**
  - Não necessariamente o tamanho do subproblema será uma fração do problema original
    - Por exemplo, a versão recursiva de busca linear cria, a cada passo, um único subproblema que difere em tamanho do problema original por apenas um elemento
    - Supondo um tempo constante adicional às chamadas recursivas, temos:
      - $T(n) = T(n - 1) + 1$
- **Resolução de Recorrências:**
  - "Resolvemos" uma recorrência quando conseguimos eliminar as referências a si mesma
  - Melhores técnicas: Substituição de variáveis, Iterações, Árvore de recorrência e Teorema Mestre
    - Para resolver a maior parte das recorrências associadas com algoritmos D&C, utiliza-se o Teorema Mestre

Recapitulando... (7)

- Existem alguns **métodos para resolver recorrências**, isto é, para **obter limites assintóticos "Θ" ou "O" para a solução**. Os principais são:
  - **Método de substituição:** arrisca-se um palpite para um limite e então utiliza-se da indução matemática para provar que nosso palpite estava correto
  - **Método da Iteração:** expandir (iterar) a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n e das condições iniciais
  - **Método da árvore de recursão:** converte a recorrência em uma árvore cujos nós representam os custos envolvidos em vários níveis da recursão
    - Usa-se técnicas para limitar somatórios, resolvendo-se a recorrência
  - **Método do Teorema mestre**: dá limites para recorrências da forma **$T(n) = aT(n/b) + f(n)$**, onde $a \geq 1$, $b > 1$ e $f(n)$ é uma função dada. Tais recorrências ocorrem frequentemente.
    - Uma recorrência da forma da equação apresentada caracteriza um algoritmo de divisão e conquista que cria **a subproblemas**, cada um **com 1/b do tamanho do problema original** e no qual as etapas de divisão e conquista, juntas, levam o **tempo f(n)**

Aplicação do paradigma de Divisão e Consquisa.....

O PROBLEMA DO SUBARRANJO MÁXIMO (*MAXIMUM-SUBARRAY PROBLEM*)

Problema: Subarranjo Máximo (1)

- O Problema de **Subarranjo Máximo** (*Maximum-Subarray Problem*) é um conhecido problema computacional que pede que seja **encontrada a subsequência contígua de números cuja soma seja máxima em uma determinada sequência de números**
- Consideremos a sequência -2, 1, -3, 4, -1, 2, 1, -5, 4
  - A subsequência contígua de números de soma máxima é 4, -1, 2, 1, cuja soma é 6
- Este problema foi proposto originalmente em 1977, por Ulf Grenander como um modelo simplificado para estimativa de máxima verossimilhança entre padrões em imagens digitalizadas

Problema: Subarranjo Máximo (2)

- Um subarranjo (segmento) de um array A[1..n] é qualquer subarray na forma A[e..d]
- **Problema:** Dado um array A[1..n] de números inteiros, determinar um subarray A[e..d] de soma máxima
- **Exemplo de Array:**

	1									n
A	31	−41	59	26	−53	58	97	−93	−23	84

- **Subarranjo Máximo:**

	1		3				7			n
A	31	−41	59	26	−53	58	97	−93	−23	84

- **A[e..d] = A[3..7] é o subarranjo máximo de A[]**
- A[3..7] tem soma 187

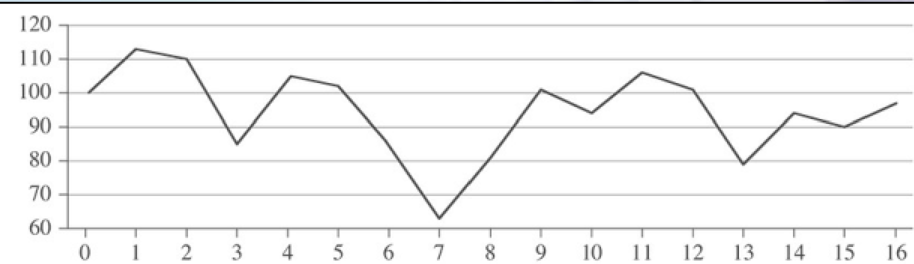
Problema: Subarranjo Máximo – Investindo em Ações (1)

- Mas por quê este problema é importante?

- Investindo em Ações?
  - Para ilustrar, suponha que você esteja interessado em investir em ações
  - Geralmente, adotamos a estratégia “comprar na baixa, vender na alta” → Maximizar os lucros
  - Consideremos algumas simplificações:
    - Só podemos comprar e vender uma única ação
    - A compra e a venda só podem ser realizadas após o fechamento do dia, quando o valor da ação não pode mais variar
    - Em compensação, possuímos o dom de “prever” o valor da ação nos dias seguintes

Problema: Subarranjo Máximo – Investindo em Ações (2)

- Informações sobre o preço das ações após o fechamento em um período de 17 dias:
  - O eixo vertical indica o preço e o eixo horizontal indica o dia
  - Na tabela, a última linha indica a mudança do valor em relação ao dia anterior

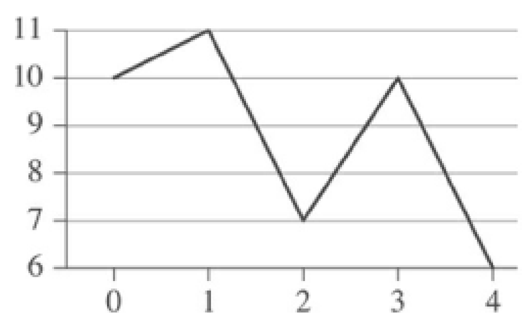


Dia	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Preço	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Mudança		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

  - Maximização do lucro está ligada a estratégia de comprar pelo preço mais baixo e vender pelo mais alto, desde que possível → Bastaria encontrar estes dois picos e, a partir do maior preço, andar para a esquerda (ou do menor preço para a direita) tentando encontrar o intervalo que obtivesse a maior diferença
    - No exemplo, não é possível comprar as ações pelo menor preço possível (depois do dia 7, $63) e vender pelo maior preço possível, depois do dia 1, $113
  - O próximo slide mostra um contraexemplo para esta estratégia “gulosa”

Problema: Subarranjo Máximo – Investindo em Ações (3)

- Um exemplo que nos mostra que o lucro máximo não necessariamente começa no menor preço (dia 4, \$6) ou termina no maior preço (dia 1, \$11)
- Com efeito, o lucro de \$3 é obtido pela compra após o dia 2 (\$7) e venda após o dia 3 (\$10)



Dia	0	1	2	3	4
Preço	10	11	7	10	6
Mudança		1	−4	3	−4

Problema: Subarranjo Máximo – Solução 01 - SEG-MAX-3 (1)

- Determinar o **Subarranjo Máximo de um array A[1..n]** sem a utilização da Divisão e Conquista?
  - A[e..d] é o subarranjo máximo do array A[]
  - somamax recebe o valor do somatório

```
SEG-MAX-3 (A, n)
somamax ← 0
e ← 0    d ← −1    ▷ A[e..d] é vazio
para i ← 1 até n faça
    para f ← i até n faça
        soma ← 0
        para k ← i até f faça
            soma ← soma + A[k]
        se soma > somamax então
            somamax ← soma    e ← i    d ← f
devolva e, d e somamax
```

Problema: Subarranjo Máximo – Solução 01 - SEG-MAX-3 (1)

```
SEG-MAX-3 (A, n)
1   somamax ← 0
2   e ← 0   d ← −1   ▷ A[e..d] é vazio
3   para i ← 1 até n faça
4       para f ← i até n faça
5           soma ← 0
6           para k ← i até f faça
7               soma ← soma + A[k]
8           se soma > somamax então
9               somamax ← soma   e ← i   d ← f
10  devolva e, d e somamax
```

- Vamos realizar a **Análise da Complexidade do algoritmo** SEG-MAX-3?

linha	todas as execuções da linha		
1-2	$=$	2	$=\Theta(1)$
3	$=$	$n+1$	$=\Theta(n)$
4	$=$	$(n+1)+n+(n-1)+\cdots+2$	$=\Theta(n^2)$
5	$=$	$n+(n-1)+\cdots+1$	$=\Theta(n^2)$
6	$=$	$(2+\cdots+(n+1))+(2+\cdots+n)+\cdots+2$	$=\Theta(n^3)$
7	$=$	$(1+\cdots+n)+(1+\cdots+(n-1))+\cdots+1$	$=\Theta(n^3)$
8	$=$	$n+(n-1)+(n-2)+\cdots+1$	$=\Theta(n^2)$
9	$\leq$	$n+(n-1)+(n-2)+\cdots+1$	$=\mathrm{O}(n^2)$
10	$=$	1	$=\Theta(1)$
total	$=$	$\Theta(2n^3+3n^2+n+2)+\mathrm{O}(n^2)$	$=\Theta(n^3)$

- **Complexidade T(n): O**(n^3)

Problema: Subarranjo Máximo – Solução 02 - SEG-MAX-2 (1)

- Determinar o **Subarranjo Máximo de um array A[1..n]** sem a utilização da Divisão e Conquista em um tempo melhor que SEG-MAX-3? Talvez Quadrático?
  - A[e..d] é o subarranjo máximo do array A[]
  - somamax recebe o valor do somatório

```
SEG-MAX-2 (A, n)
    somamax ← 0
    e ← 0    d ← −1    ▷ A[e..d] é vazio
    para i ← 1 até n faça
        soma ← 0
        para f ← i até n faça
            soma ← soma + A[f]
            se soma > somamax então
                somamax ← soma    e ← i    d ← f
    devolva e, d e somamax
```

Problema: Subarranjo Máximo – Solução 02 - SEG-MAX-2 (2)

```
SEG-MAX-2 (A, n)
1  somamax ← 0
2  e ← 0    d ← −1    ▷ A[e . . d] é vazio
3  para i ← 1 até n faça
4       soma ← 0
5       para f ← i até n faça
6            soma ← soma + A[f]
7            se soma > somamax então
8                 somamax ← soma    e ← i    d ← f
9  devolva e, d e somamax
```

- Vamos realizar a **Análise da Complexidade do algoritmo** SEG-MAX-2?

linha		todas as execuções da linha	
1-2	$=$	2	$=\Theta(1)$
3	$=$	$n+1$	$=\Theta(n)$
4	$=$	n	$=\Theta(n)$
5	$=$	$(n+1)+n+\cdots+2$	$=\Theta(n^2)$
6	$=$	$n+(n-1)+\cdots+1$	$=\Theta(n^2)$
7	$=$	$n+(n-1)+\cdots+1$	$=\Theta(n^2)$
8	$\leq$	$n+(n-1)+\cdots+1$	$=\mathrm{O}(n^2)$
9	$=$	1	$=\Theta(1)$
total	$=$	$\Theta(3n^2+2n+2)+\mathrm{O}(n^2)$	$=\Theta(n^2)$

- **Complexidade T(n): O**(n^2)
- **Conseguimos melhorar mais ainda?**

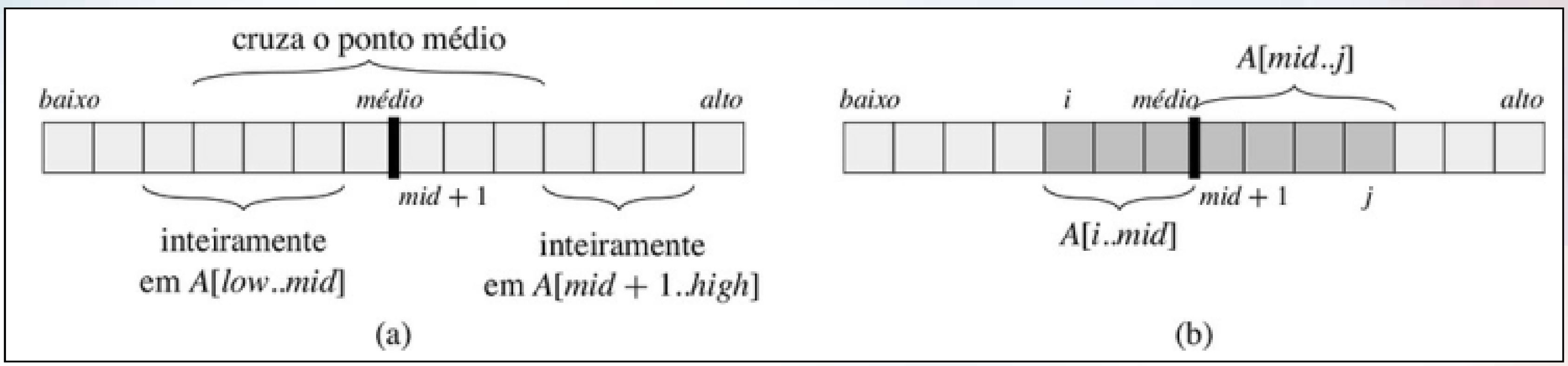
Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC (1)

- O paradigma D&C sugere que, se temos uma sequência A[inicio,..., fim], então a dividamos em duas partes, se possível, de mesmo tamanho
- Assim, teríamos subproblemas A[inicio,..., meio] e A[meio+1,..., fim]
- **Qualquer subsequência A[i...j] deve cair** em exatamente uma das situações abaixo:
  - Completamente na primeira metade A[inicio,..., meio], tal que **inicio ≤ i ≤ j ≤ meio**
  - Completamente na segunda metade A[meio+1,..., fim], tal que **meio < i ≤ j ≤ fim**
  - Cruzando o ponto médio, tal que **inicio ≤ i ≤ meio < j ≤ fim**
- Sendo assim, a s**ubsequência de soma máxima deve possuir a soma maior do que todas as subsequências em A[inicio,.., meio], em A[meio+1, .., fim] e também cruzando o ponto médio**

Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC (2)

a) Possíveis localizações de subarranjos de A[low...high]:
   - inteiramente em **A[low...mid]**
   - inteiramente em **A[mid+1...high]** ou
   - **cruzando o ponto médio mid**

b) Qualquer subarranjo de A[low...high] que cruze o ponto médio compreende dois subarranjos A[i...mid] e A[mid+1...j], onde **low ≤ i ≤ mid e mid < j ≤ high**



Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC (3)

- Pode-se determinar as subsequências de soma máxima de A[inicio;...; meio] e A[meio + 1;...; fim] recursivamente, uma vez que estes dois subproblemas são cópias exatas do problema original
- O que nos faltaria é determinar a subsequência de soma máxima que cruza o ponto médio, depois, determinar dos três qual é o maior
- Podemos facilmente determinar subsequência de soma máxima que cruza o ponto médio, em tempo linear no tamanho A[inicio...fim]
  - Podemos, por exemplo, determinar as subsequências de forma A[i...meio] e A[meio+1...j] e depois combiná-las
  - O algoritmo recebe como entrada o array A e os índices do início, meio e fim, retornando os índices que delimitam a subsequência de soma máxima que cruze o ponto médio entre dois subproblemas

Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC (4)

```
FindMaxCrossingSubarray(A, inicio, meio, fim)
Entrada: Sequência A, índices inicio, meio e fim
somaEsquerda ← −∞; soma ← 0;
para i ← meio até inicio faça
    soma ← soma + A[i];
    se soma > somaEsquerda então
        somaEsquerda ← soma;
        maxEsquerda ← i;
    fim
fim
somaDireita ← −∞; soma ← 0;
para j ← meio + 1 até fim faça
    soma ← soma + A[j];
    se soma > somaDireita então
        somaDireita ← soma;
        maxDireita ← j;
    fim
fim
retorna maxEsquerda, maxDireita, somaEsquerda+somaDireita;
```

Solução do Livro do Cormen

- O algoritmo determina o subarranjo de soma máxima do meio para o início, depois, do meio para o fim
- Ambos subarranjos possuem o elemento do meio → as combinamos para gerar um único subarranjo de soma máxima que cruza o ponto médio
- A cada iteração, os dois laços possuem custo constante → Algoritmo é linear no número de elementos, ou seja, Θ(n)
- De posse deste algoritmo, pode-se projetar um algoritmo D&C para o Maximum-Subarray Problem

Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC (5)

```
FindMaxSubarray(A, inicio, fim)
Entrada: Sequência A, índices inicio e fim
se fim = inicio então
 | retorna inicio, fim, A[inicio];//um único elemento
senão
 | meio ← ⌊(inicio + fim)/2⌋;
 | (inicioEsquerda, fimEsquerda, somaEsquerda)← FindMaxSubarray(A, inicio,
 |   meio);
 | (inicioDireita, fimDireita, somaDireita)← FindMaxSubarray(A, meio+1, fim);
 | (inicioCruzado, fimCruzado, somaCruzado)← FindMaxCrossingSubarray(A,
 |   inicio, meio, fim);
 | se somaEsquerda ≥ somaDireita e somaEsquerda ≥ somaCruzado então
 |  | retorna inicioEsquerda, fimEsquerda, somaEsquerda;
 | senão
 |  | se somaDireita ≥ somaEsquerda e somaDireita ≥ somaCruzado então
 |  |  | retorna inicioDireita, fimDireita, somaDireita;
 |  | senão
 |  |  | retorna inicioCruzado, fimCruzado, somaCruzado;
 |  | fim
 | fim
fim
```

Solução do Livro do Cormen

- As chamadas recursivas de FindMaxCrossingSubarray retornam uma tupla que delimita os elementos do subarranjo de soma máxima, além do valor da soma máxima
- O caso base é termos apenas um elemento, que por si só é um subarranjo de soma máxima
- O passo recursivo determina o meio do subproblema e o divide para determinar os subarranjos de soma máxima do lado esquerdo e do lado direito
- O passo de combinação determina a subsequência de soma máxima que cruza o ponto médio do problema e depois determina dentre as três qual possui a maior soma

Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC (6)

- Vamos realizar a **Análise da Complexidade do algoritmo recursivo?**
  - Para simplificarmos a análise do algoritmo, consideramos que o tamanho do problema é uma potência de 2, de forma que o tamanho de todos os problemas é inteiro e par
  - Seja T(n) o tempo de execução de *FindMaxSubarray()*:
    - T(1) = 1 → o caso base possui apenas operações de tempo constante
    - No passo recursivo, dividimos o problema em duas partes iguais, então necessitamos de T(n/2) de tempo para resolver cada um deles
    - Como resolvemos 2 subproblemas em tempo linear, temos 2T(n/2)
    - Como vimos, *FindMaxCrossingSubarray()* possui tempo Θ(n)
    - Determinar o subarranjo de maior soma, dentre as três calculadas é feito em tempo constante (Θ(1))

Recorrência – FindMaxSubarray

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1, \\ 2T(\frac{n}{2}) + \Theta(n) & \text{se } n > 1 \end{cases}$$

    - **Complexidade T(n): O**(n lgn) → Aplicar o Teorema Mestre para Resolver a Recorrência

Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC - Alternativo (7)

- Determinar o **Subarranjo Máximo de um array A[p..d]** adotando-se o paradigma de Divisão e Conquista alternativo ao Algoritmo do Cormen?

SEG-MAX-DC (A, p, d)

se $p = d$ **então devolva** $\max(0, A[p])$
$q \leftarrow \lfloor (p+d)/2 \rfloor$
$maxesq \leftarrow$ SEG-MAX-DC(A, p, q)
$maxdir \leftarrow$ SEG-MAX-DC$(A, q+1, d)$
$max2esq \leftarrow soma \leftarrow A[q]$
para $i \leftarrow q-1$ **decrescendo até** p **faça**
    $soma \leftarrow soma + A[i]$
    $max2esq \leftarrow \max(max2esq, soma)$
$max2dir \leftarrow soma \leftarrow A[q+1]$
para $f \leftarrow q+2$ **até** d **faça**
    $soma \leftarrow soma + A[f]$
    $max2dir \leftarrow \max(max2dir, soma)$
$maxcruz \leftarrow max2esq + max2dir$
devolva $\max(maxesq, maxcruz, maxdir)$

- **maxesq** é a soma máxima de um segmento de A[p...q]
- **maxdir** é a soma máxima de um segmento de A[q+1...d];
- **maxcruz** é a soma máxima de um segmento da forma A[i...f] com i ≤ q e q + 1 ≤ f

Problema: Subarranjo Máximo – Solução 03 - SEG-MAX-DC - Alternativo (8)

- Vamos realizar a **Análise da Complexidade do algoritmo** SEG-MAX-DC?

linha		todas as execuções da linha	
1-2	$=$	2	$=\Theta(1)$
3	$=$	$T(\lceil \frac{n}{2} \rceil)$	$=T(\lceil \frac{n}{2} \rceil)$
4	$=$	$T(\lfloor \frac{n}{2} \rfloor)$	$=T(\lfloor \frac{n}{2} \rfloor)$
5	$=$	1	$=\Theta(1)$
6	$=$	$\lceil \frac{n}{2} \rceil + 1$	$=\Theta(n)$
7-8	$=$	$\lceil \frac{n}{2} \rceil$	$=\Theta(n)$
9	$=$	1	$=\Theta(1)$
10	$=$	$\lfloor \frac{n}{2} \rfloor + 1$	$=\Theta(n)$
11-12	$=$	$\lfloor \frac{n}{2} \rfloor$	$=\Theta(n)$
13-14	$=$	2	$=\Theta(1)$
total	$=$	$T(\lfloor \frac{n}{2} \rfloor) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n)$	

SEG-MAX-DC (A, p, d)

1 **se** $p = d$ **então devolva** $\max(0, A[p])$
2 $q \leftarrow \lfloor (p+d)/2 \rfloor$
3 $maxesq \leftarrow$ SEG-MAX-DC(A, p, q)
4 $maxdir \leftarrow$ SEG-MAX-DC$(A, q+1, d)$
5 $max2esq \leftarrow soma \leftarrow A[q]$
6 **para** $i \leftarrow q-1$ **decrescendo até** p **faça**
7 $\quad soma \leftarrow soma + A[i]$
8 $\quad max2esq \leftarrow \max(max2esq, soma)$
9 $max2dir \leftarrow soma \leftarrow A[q+1]$
10 **para** $f \leftarrow q+2$ **até** d **faça**
11 $\quad soma \leftarrow soma + A[f]$
12 $\quad max2dir \leftarrow \max(max2dir, soma)$
13 $maxcruz \leftarrow max2esq + max2dir$
14 **devolva** $\max(maxesq, maxcruz, maxdir)$

- **Complexidade T(n): O**(n lgn)
- **Conseguimos melhorar mais ainda?** Sim. Existe um algoritmo linear proposto por Jay Kadane

Problema: Subarranjo Máximo – Comparação dos Algoritmos

- Os quatro algoritmos foram implementados na linguagem de programação Python. A execução foi realizada em uma máquina com processador Intel(R) Corel(TM)2 DUO CPU P8600 @ 2.40GHz 1,58 GHz, Memória RAM de 1,89 GB e sistema operacional Microsoft Windows XP Professional Versão 2002 Service Pack 3

n	$O(n^3)$	$O(n^2)$	$O(n\ log\ n)$	$O(n)$
10^0	3,10 μs	3,10 μs	3,20 μs	3,10 μs
10^1	133,10 μs	39,00 μs	73,40 μs	9,40 μs
10^2	37,00 ms	2,63 ms	3,08 ms	93,00 μs
10^3	32,30 s	243,00 ms	247,18 ms	930 μs
10^4	6,53 h	23,00 s	22,88 s	7,8 ms
10^5	272,01 d	51,67 m	20,95 m	78 ms
10^6	745,18 a	3,59 d	29,55 h	750 ms

- Valores calculador por extrapolação

Aplicação do paradigma de Divisão e Consquisa.....

O PROBLEMA DA MULTIPLICAÇÃO DE MATRIZES – ALGORITMO DE STRASSEN

Problema: Multiplicação de Matrizes

- Uma operação frequentemente utilizada na Matemática e na Computação é a multiplicação de matrizes
  - A manipulação de imagens computadorizadas é feita por meio de transformações geométricas, definidas por operações com matrizes

- Existe mais de um algoritmo que realiza essa operação, iremos estudar sobre:
  - O Algoritmo Tradicional (Ensino Médio)
  - O Algoritmo por Divisão e Conquista
  - O Algoritmo de Strassen

Revisão Rápida da Multiplicação de Matrizes (1)

- Por definição, o produto de matrizes A.B só é possível se o número de colunas de A for igual ao número de linhas de B
  - Por exemplo, o produto das seguintes matrizes não é possível, pois a primeira matriz tem 2 colunas e a segunda tem 3 linhas:

$$\begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 0 & 4 \\ 1 & 8 \end{bmatrix}$$

  - Por outro lado, o exemplo a seguir é possível, pois a definição é satisfeita:

$$\begin{bmatrix} 1 & 2 \\ 0 & 5 \end{bmatrix} \times \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

  - A matriz resultante terá o mesmo número de linhas da primeira matriz e o mesmo número de colunas da segunda matriz → $A_{m \times n} \times B_{n \times p} = C_{m \times p}$
    - No último exemplo, o resultado seria uma matriz com duas linhas e uma coluna

Revisão Rápida da Multiplicação de Matrizes (2)

- Se A = (a_{ij}) e B = (b_{ij}) são matrizes, então no produto C = A × B, definimos a entrada c_{ij}, para i, j = 1, 2, … , n, por:

$$\boxed{c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}}$$

- Isto é, o elemento da linha i e coluna j será a soma dos produtos dos elementos a_{ik} pelos elementos b_{kj}
- A constante n é igual ao número de colunas da primeira matriz (que é igual ao número de linhas da segunda)

Multiplicação de Matrizes: Exemplo Prático

- Calcular o produto das seguintes matrizes:

$$\begin{bmatrix} 3 & 1 \\ 2 & -1 \\ 0 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & -1 & 2 \\ 3 & 0 & 5 \end{bmatrix}$$

- Resolução:
  - O resultado será uma matriz 3x3

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

  - Ilustração do cálculo para o primeiro elemento:

$$c_{11} = \sum_{k=1}^{2} a_{1k} \times b_{k1}$$
$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21}$$
$$c_{11} = 3 \times 1 + 1 \times 3 = 6$$

  - Repetindo o procedimento para os demais:

$$\begin{bmatrix} 3 \times 1 + 1 \times 3 & 3 \times (-1) + 1 \times 0 & 3 \times 2 + 1 \times 5 \\ 2 \times 1 + (-1) \times 3 & 2 \times (-1) + (-1) \times 0 & 2 \times 2 + (-1) \times 5 \\ 0 \times 1 + 4 \times 3 & 0 \times (-1) + 4 \times 0 & 0 \times 2 + 4 \times 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 6 & -3 & 11 \\ -1 & -2 & -1 \\ 12 & 0 & 20 \end{bmatrix}$$

Problema: Multiplicação de Matrizes – O Algoritmo Tradicional

- Vocês conseguem pensar em um algoritmo para resolver este problema?

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

- E a Complexidade deste Algoritmo Tradicional?
  - **T(n) = θ(n^3)** → Laços aninhados triplicados, ocorrendo exatamente n iterações

Problema: Multiplicação de Matrizes – O Algoritmo por Divisão e Conquista (1)

- Quando utilizado a estratégia D&C na matriz C = A . B, assumimos que o valor de n é potência exata de 2 e cada matriz de entrada é de ordem n × n
  - Justificativa: Em cada etapa é dividido cada matriz n × n em quatro n/2 × n/2, e assumindo que n é potência exata de 2, é garantido que, enquanto n ≥ 2 a dimensão n/2 é um inteiro

$$\begin{bmatrix} C_{11} C_{12} \\ C_{21} C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} A_{12} \\ A_{21} A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} B_{12} \\ B_{21} B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{21} \cdot B_{21} \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{aligned}$$

  - É possível utilizar as equações acima para construir um algoritmo de divisão e conquista, mas primeiro, vamos generalizar.

Problema: Multiplicação de Matrizes – O Algoritmo por Divisão e Conquista (2)

- Generalização da para adotar a Estratégia de Divisão e Conquista:

$$\overset{X}{\begin{bmatrix} A & B \\ C & D \end{bmatrix}} \times \overset{Y}{\begin{bmatrix} E & F \\ G & H \end{bmatrix}} = \begin{bmatrix} R & S \\ T & U \end{bmatrix}$$

$$\begin{aligned} R &= AE + BG \\ S &= AF + BH \\ T &= CE + DG \\ U &= CF + DH \end{aligned}$$

Problema: Multiplicação de Matrizes – O Algoritmo por Divisão e Conquista (3)

- O Algoritmo de D&C recebe inteiros X[1...n] e Y[1...n] e devolve X.Y:

$$R = AE + BG$$
$$S = AF + BH$$
$$T = CE + DG$$
$$U = CF + DH$$

MULTI-M (X, Y, n)

se $n = 1$ **devolva** $X \cdot Y$

$(A, B, C, D) \leftarrow$ PARTICIONE(X, n)

$(E, F, G, H) \leftarrow$ PARTICIONE(Y, n)

$R \leftarrow$ MULTI-M$(A, E, n/2)$ + MULTI-M$(B, G, n/2)$

$S \leftarrow$ MULTI-M$(A, F, n/2)$ + MULTI-M$(B, H, n/2)$

$T \leftarrow$ MULTI-M$(C, E, n/2)$ + MULTI-M$(D, G, n/2)$

$U \leftarrow$ MULTI-M$(C, F, n/2)$ + MULTI-M$(D, H, n/2)$

$P \leftarrow$ CONSTRÓI-MAT(R, S, T, U)

devolva P

  - PARTICIONE corresponde a criar a matriz (A,B,C,D) ou (E,F,G,H), com n elementos, a partir de X ou Y, respectivamente
  - CONTRÓI-MAT corresponde a criar a matriz P, com base nos resultados calculados de R, S, T e U

Problema: Multiplicação de Matrizes – O Algoritmo por Divisão e Conquista (4)

- Calcular T(n): Análise da Complexidade do Algoritmo de D&C:

```
MULTI-M (X, Y, n)
1  se n = 1 devolva X · Y
2  (A, B, C, D) ← PARTICIONE(X, n)
3  (E, F, G, H) ← PARTICIONE(Y, n)
4  R ← MULTI-M(A, E, n/2) + MULTI-M(B, G, n/2)
5  S ← MULTI-M(A, F, n/2) + MULTI-M(B, H, n/2)
6  T ← MULTI-M(C, E, n/2) + MULTI-M(D, G, n/2)
7  U ← MULTI-M(C, F, n/2) + MULTI-M(D, H, n/2)
8  P ← CONSTRÓI-MAT(R, S, T, U)
9  devolva P
```

linha		todas as execuções da linha
1	$=$	$\Theta(1)$
2	$=$	$\Theta(n^2)$
3	$=$	$\Theta(n^2)$
4	$=$	$T(n/2) + T(n/2)$
5	$=$	$T(n/2) + T(n/2)$
6	$=$	$T(n/2) + T(n/2)$
7	$=$	$T(n/2) + T(n/2)$
8	$=$	$\Theta(n^2)$
9	$=$	$\Theta(n^2)$
total	$=$	$8\,T(n/2) + \Theta(n^2)$

- **Fórmula de Recorrência:**

$$T(n) = \begin{cases} \theta(1) & se\ n\ =\ 1, \\ 8T(n/2) + \theta(n^2) & se\ n\ >\ 1 \end{cases}$$

- **Complexidade T(n): O**(n^3) Aplicar o Teorema Mestre para Resolver a Recorrência
  - Não é mais rápido que o Algoritmo Tradicional. Será que conseguimos melhorar?

Problema: Multiplicação de Matrizes – O Algoritmo de Strassen (1)

- A principal diferença entre o algoritmo de Strassen e o algoritmo de D&C é que no algoritmo de Strassen são realizadas sete chamadas recursivas, uma a menos que o algoritmo D&C:
  - **Etapa 01:** Dividir as matrizes de entrada A e B e a matriz de saída C em submatrizes n/2 × n/2
  - **Etapa 02:** Criar 10 matrizes S1, S2, ..., S10, cada uma das quais é n/2 × n/2 e é a soma ou diferença de duas matrizes criadas na etapa 1 → Podemos criar todas as 10 matrizes no tempo $\Theta(n^2)$
  - **Etapa 03:** Usando as submatrizes criadas na etapa 1 e as 10 matrizes criadas na etapa 2, calcular recursivamente sete produtos de matrizes P1, P2,..., P7 → Cada matriz Pi é n/2 × n/2.
  - **Etapa 04:** Calcular as submatrizes desejadas C_{11}, C_{12}, C_{21}, C_{22} da matriz resultado C somando e subtraindo várias combinações das Pi matrizes → Podemos calcular todas as quatro submatrizes no tempo $\Theta(n^2)$

Problema: Multiplicação de Matrizes – O Algoritmo de Strassen (2)

- As 10 matrizes a serem criadas na etapa 02 são:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22} \\
S_2 &= A_{11} + A_{12} \\
S_3 &= A_{21} + A_{22} \\
S_4 &= B_{21} - B_{11} \\
S_5 &= A_{11} + A_{22} \\
S_6 &= B_{11} + B_{22} \\
S_7 &= A_{12} - A_{22} \\
S_8 &= B_{21} + B_{22} \\
S_9 &= A_{11} - A_{21} \\
S_{10} &= B_{11} + B_{12}
\end{aligned}
$$

- Na etapa 3, multiplicamos recursivamente matrizes n/2 × n/2 sete vezes para calcular as seguintes matrizes n/2 × n/2, cada uma das quais é a soma ou a diferença de produtos de submatrizes A e B:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \, , \\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \, , \\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \, , \\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \, , \\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \, , \\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \, , \\
P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \, .
\end{aligned}
$$

Observe que as únicas multiplicações que devemos executar são as que se encontram na coluna do meio dessas equações. A coluna do lado direito mostra apenas em que esses produtos são iguais em termos das submatrizes originais criadas na etapa 1.

Problema: Multiplicação de Matrizes – O Algoritmo de Strassen (3)

- A última etapa realiza a soma e subtração das matrizes Pi em quatro submatrizes C:

$$\begin{aligned}
C_{11} &= P5 + P4 - P2 + P6 \\
C_{12} &= P1 + P2 \\
C_{21} &= P3 + P4 \\
C_{22} &= P5 + P1 - P3 - P7
\end{aligned}$$

$$\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}\,, \\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}\,, \\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}\,, \\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}\,, \\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}\,, \\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}\,, \\
P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}\,.
\end{aligned}$$

- Prova para cada valor de C:

$$\begin{array}{rlllllll}
P5 = & +A_{11}B_{11} & +A_{11}B_{22} & +A_{22}B_{11} & +A_{22}B_{22} & & & \\
P4 = & & & -A_{22}B_{11} & & +A_{22}B_{21} & & \\
-P2 = & & -A_{11}B_{22} & & & & -A_{12}B_{22} & \\
P6 = & & & & -A_{22}B_{22} & -A_{22}B_{21} & +A_{12}B_{22} & +A_{12}B_{21} \\
\hline
C_{11} = & +A_{11}B_{11} & & & & & & +A_{12}B_{21}
\end{array}$$

$$\begin{array}{rlll}
P1 = & +A_{11}B_{12} & -A_{11}B_{22} & \\
P2 = & & +A_{11}B_{22} & +A_{12}B_{22} \\
\hline
C_{12} = & +A_{11}B_{12} & & +A_{12}B_{22}
\end{array}$$

$$\begin{array}{rlll}
P3 = & +A_{21}B_{11} & +A_{22}B_{11} & \\
P4 = & & -A_{22}B_{11} & +A_{22}B_{21} \\
\hline
C_{21} = & +A_{21}B_{11} & & +A_{22}B_{21}
\end{array}$$

Provado!!!

$$\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{21} \cdot B_{21} \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}
\end{aligned}$$

$$\begin{array}{rlllllll}
P5 = & +A_{11}B_{11} & +A_{11}B_{22} & +A_{22}B_{11} & +A_{22}B_{22} & & & \\
P1 = & & -A_{11}B_{22} & & & +A_{11}B_{12} & & \\
-P3 = & & & -A_{22}B_{11} & & & -A_{21}B_{11} & \\
-P7 = & -A_{11}B_{11} & & & & -A_{11}B_{12} & +A_{21}B_{11} & +A_{21}B_{12} \\
\hline
C_{22} = & & & & +A_{22}B_{22} & & & +A_{21}B_{12}
\end{array}$$

Problema: Multiplicação de Matrizes – O Algoritmo de Strassen (3)

- **Pseudocódigo do Algoritmo de Strassen (Cormen):**

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}\\
S_2 &= A_{11} + A_{12}\\
S_3 &= A_{21} + A_{22}\\
S_4 &= B_{21} - B_{11}\\
S_5 &= A_{11} + A_{22}\\
S_6 &= B_{11} + B_{22}\\
S_7 &= A_{12} - A_{22}\\
S_8 &= B_{21} + B_{22}\\
S_9 &= A_{11} - A_{21}\\
S_{10} &= B_{11} + B_{12}
\end{aligned}
$$

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}\,,\\
P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}\,,\\
P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}\,,\\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}\,,\\
P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}\,,\\
P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}\,,\\
P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}\,.
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= P5 + P4 - P2 + P6\\
C_{12} &= P1 + P2\\
C_{21} &= P3 + P4\\
C_{22} &= P5 + P1 - P3 - P7
\end{aligned}
$$

Algorithm 3 Strassen(A, B)

```
if A.length == 1 then
    return A[1] · B[1]
end if
Let C be a new n by n matrix
A11 = A[1..n/2][1..n/2]
A12 = A[1..n/2][n/2 + 1..n]
A21 = A[n/2 + 1..n][1..n/2]
A22 = A[n/2 + 1..n][n/2 + 1..n]
B11 = B[1..n/2][1..n/2]
B12 = B[1..n/2][n/2 + 1..n]
B21 = B[n/2 + 1..n][1..n/2]
B22 = B[n/2 + 1..n][n/2 + 1..n]
S1 = B12 − B22
S2 = A11 + A12
S3 = A21 + A22
S4 = B21 − B11
S5 = A11 + A22
S6 = B11 + B22
S7 = A12 − A22
S8 = B21 + B22
S9 = A11 − A21
S10 = B11 + B12
P1 = Strassen(A11, S1)
P2 = Strassen(S2, B22)
P3 = Strassen(S3, B11)
P4 = Strassen(A22, S4)
P5 = Strassen(S5, S6)
P6 = Strassen(S7, S8)
P7 = Strassen(S9, S10)
C[1..n/2][1..n/2] = P5 + P4 − P2 + P6
C[1..n/2][n/2 + 1..n] = P1 + P2
C[n/2 + 1..n][1..n/2] = P3 + P4
C[n/2 + 1..n][n/2 + 1..n] = P5 + P1 − P3 − P7
return C
```

Problema: Multiplicação de Matrizes – O Algoritmo de Strassen (4)

- Calcular T(n): Análise da Complexidade do Algoritmo de Strassen:
  - Vamos considerar que tão logo o tamanho n da matriz atinja 1, efetuamos uma multiplicação escalar simples
  - Quando n > 1, as etapas 1, 2 e 4 levam um tempo total de $\Theta(n^2)$ e a Etapa 03 requer que efetuemos sete multiplicações de matrizes n/2 × n/2

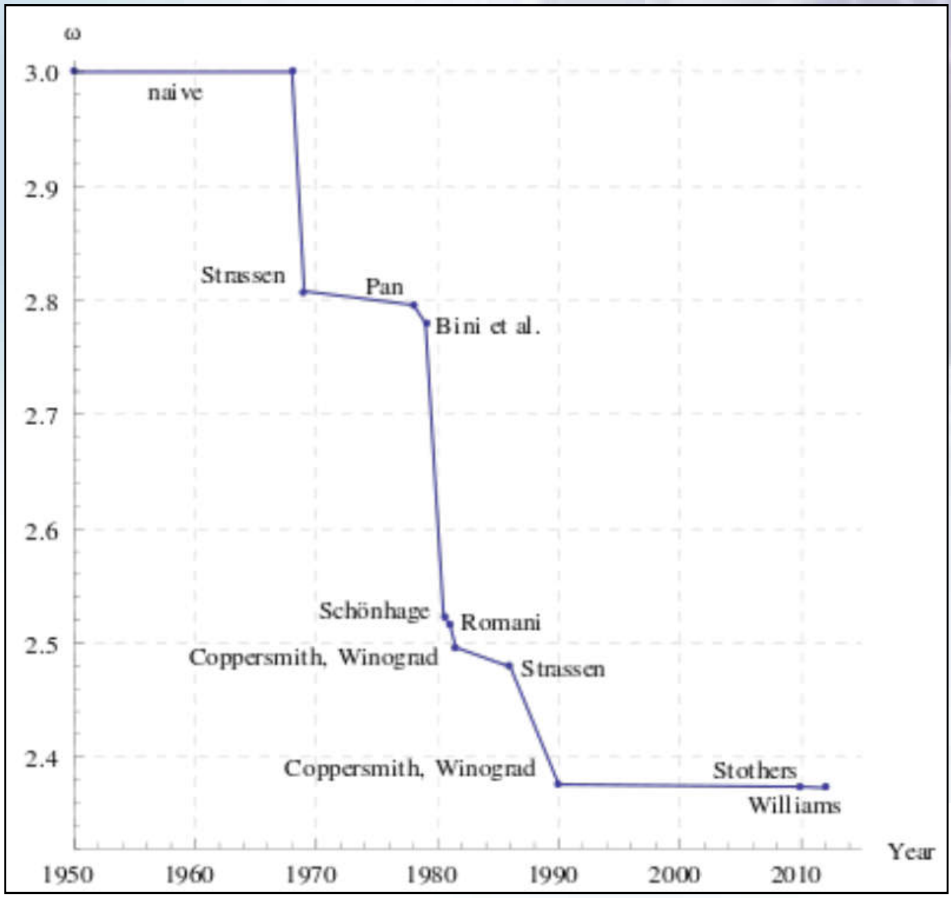
- Por consequência, obtermos a seguinte recorrência para o tempo de execução T(n) do algoritmo de Strassen:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1; \\ 7T(n/2) + \Theta(n^2) & \text{se } n > 1. \end{cases}$$

- **Complexidade T(n): O**($n^{\lg 7}$) → Aplicar o Teorema Mestre para Resolver a Recorrência
  - Ressalta-se que 2,80 < lg 7 < 2,81 →Algoritmo de Strassen é assintoticamente mais rápido que os demais

Problema: Multiplicação de Matrizes – Comparação de Outros Algoritmos

- Outros algoritmos buscam ser mais eficientes que $\theta(n^3)$, inspirados no algoritmo de Strassen, eles apresentam uma complexidade menor que a já calculada



Fonte: Virginia Vassilevska Williams. Multiplying matrices in O (n2. 373), 2014

Proposta de Estudo – Vídeo

- **Strassen's Matrix Multiplication | Divide and Conquer | GeeksforGeeks:**
  - Link: https://www.youtube.com/watch?v=E-QtwPi620I&feature=emb_title

Conclusões (1)

- **Vantagens da Divisão e Conquista:**
  - Torna problemas difíceis mais fáceis pela diminuição do tamanho
  - Tendência a complexidade logarítmica
  - Paralelismo facilitado no processo de “conquista”
  - Em computação aritmética, traz resultados mais precisos em termos de controle de arredondamento
- **Desvantagens da Divisão e Conquista:**
  - O número de chamadas recursivas pode ser um inconveniente para o desempenho
  - Eventual dificuldade para selecionar o caso base
  - Redundância de resolução de subproblemas repetidos, o que pode ser resolvido através do uso de “memorização”

Conclusões (2)

- Começamos a ter uma ideia de quão poderoso pode ser o paradigma de Divisão e Conquista
- Vimos como a Divisão e Conquista pode nos permitir projetar algoritmos assintoticamente mais rápidos do que Força Bruta
- Para alguns problemas, os melhores algoritmos existentes são de Divisão e Conquista
- Em outros casos, podemos fazer ainda melhor
  - Com efeito o melhor algoritmo para o Problema de Subarranjo Máximo possui complexidade linear e não usa D&C

PANC: Projeto e Análise de Algoritmos

Aula 09: Divisão e Conquista - Subarranjo Máximo e Multiplicação de Matrizes

Breno Lisi Romano

Dúvidas???

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista