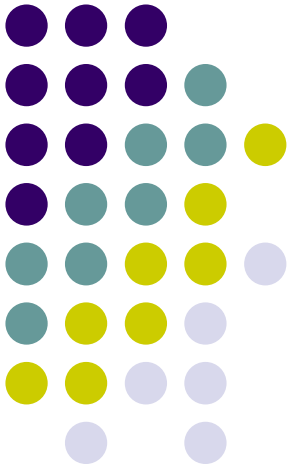
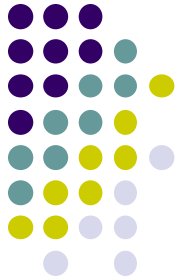
Análise de algoritmos

SCC-214 Projeto de Algoritmos

Thiago A. S. Pardo



Análise de algoritmos



- Precauções
  - Alguns dizem que a expressão correta é “f(n) é O(g(n))”
    - Seria considerado redundante e inadequado dizer “f(n) ≤ O(g(n))” ou (ainda pior) “f(n) = O(g(n))”
    - Não é incorreto (embora não seja usual) dizer “f(n) ϵ O(g(n))”, já que o operador *Big-oh* representa todo um conjunto de funções

Precauções

- A análise assintótica é uma ferramenta fundamental ao projeto, análise ou escolha de um algoritmo específico para uma dada aplicação

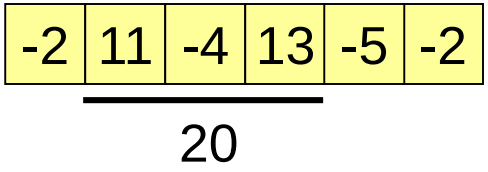
- No entanto, deve-se ter sempre em mente que essa análise "esconde" fatores assintoticamente irrelevantes, mas que em alguns casos podem ser relevantes na prática, particularmente se o problema de interesse se limitar a entradas (relativamente) pequenas

  - Por exemplo, um algoritmo com tempo de execução da ordem de $10^{100}n$ é $O(n)$, assintoticamente melhor do que outro com tempo 10 n log n, o que nos faria, em princípio, preferir o primeiro

  - No entanto, 10^{100} é o número estimado por alguns astrônomos como um limite superior para a quantidade de átomos existente no universo observável!

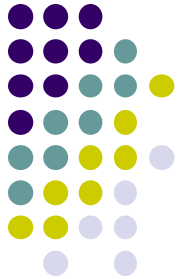
Exercício

- Problema da maior soma de subseqüência em um vetor



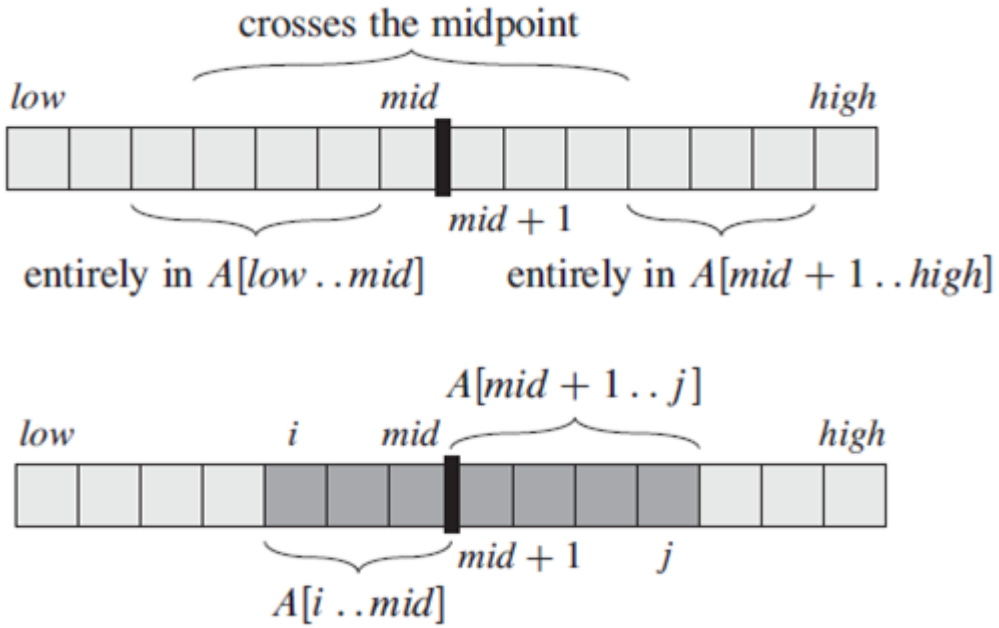
- Implemente um programa para resolver o problema e analise-o

Exercício



- Maior soma de subseqüência em um vetor
  - Algoritmo de $O(n^2)$

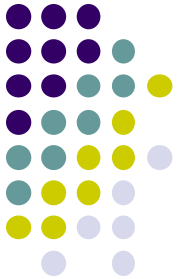
Solução mais eficiente



Soma que passa pelo meio

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

Soma que passa pelo meio



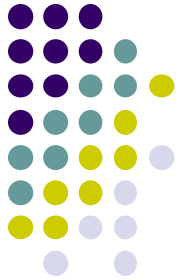
```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

Qual a complexidade?

Solução

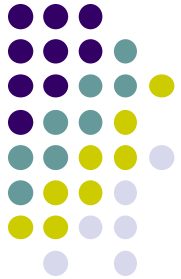
```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

Análise de complexidade



- T(1) = 1 (ou constante)
- T(n) = ?
  - Criação da variável mid → O(c)
  - Chamada para esquerda → T(n/2)
  - Chamada para direita → T(n/2)
  - Combinação das soluções → O(n)
  - Retorno da solução → O(c)
- T(n) = 2 T (n/2) + O(n) + O(c)

Exercício



- Maior soma de subseqüência em um vetor
  - Algoritmo de O(n log n)

Exercício

- Implemente um programa para calcular x^n e analise-o

Exercício

- Exponenciação
  - O(n)

Exercício

- Exponenciação
  - O(log n)

Exercício para casa

- Análise do algoritmo de Euclides
  - Um dos algoritmos mais antigos conhecidos (300 anos antes de Cristo)
  - Cômputo do máximo divisor comum entre dois números m e n, com m≥n
    - mdc(50,15)=5

```
int mdc(int m, int n) {
    int aux=0;
    while (n>0) {
        aux=m%n;
        m=n;
        n=aux;
    }
    return(m);
}
```