Algoritmos e Estruturas de Dados

técnicas de desenho de algoritmos

2022-2023

Carlos Lisboa Bento

1

Técnicas de Projecto de Algoritmos

conceito e aplicações

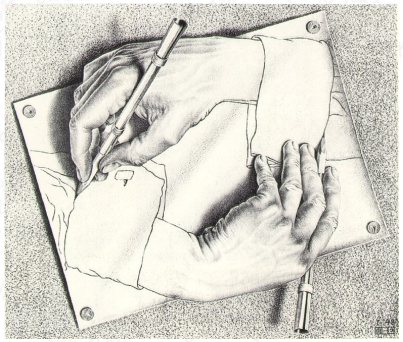
- Recursão
- Divisão e Conquista (Divide and Conquer)
- Programação Dinâmica

2

Técnicas de Projecto de Algoritmos

RECURSÃO: aplicações

Ex5: Escher



Exemplos de recursividade na NATUREZA; ARTE; na TECNOLOGIA

5

Técnicas de Projecto de Algoritmos

RECURSÃO: aplicações

Ex7: Babuscas

Exemplos de recursividade na NATUREZA; ARTE; na TECNOLOGIA

7

Técnicas de Projecto de Algoritmos

RECURSÃO: conceito e aplicações

Um método recursivo é definido em termos de uma instância mais simples dele próprio assim se caminha em direcção a um caso de base que deve ser calculado sem recurso ao processo recursivo.

9

Técnicas de Projecto de Algoritmos

RECURSÃO: conceito e aplicações

10

Técnicas de Projecto de Algoritmos

RECURSÃO: aplicações

Ex8: Consideremos o cálculo da soma dos N primeiros inteiros.

Caso de base: S(1) = 1

Caso recursivo: S(N) = S(N-1) + N

```
// Calcula a soma dos n primeiros
// inteiros positivos.

public static long s (int n)
{
    if (n == 1) return 1;
    else return s( n-1 ) + n;
}
```


11

Técnicas de Projecto de Algoritmos

RECURSÃO: aplicações

Ex11: Imprimir o número N em qualquer base

```
final  static String
    DIGIT_TABLE = "0123456789abcdef";

// Recursive routine
public static void printInt ( int n, int base )
  {
    if( n >= base )
      printInt ( n / base, base );

    System.out.print(
      DIGIT_TABLE.charAt (n % base ));
  }
```


12

Técnicas de Projecto de Algoritmos

RECURSÃO

Conclusões intermédias:

- A recursão é apelativa para determinados problemas

... Que custos ?

Quando deve (não deve) ser seguida ?

16

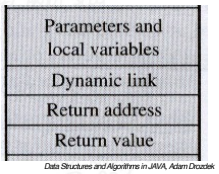
Técnicas de Projecto de Algoritmos

RECURSÃO: implementação

Activação de um método

main() ➔f1()

Durante a execução de um método é mantida na pilha run-time um REGISTO DE ACTIVAÇÃO com:



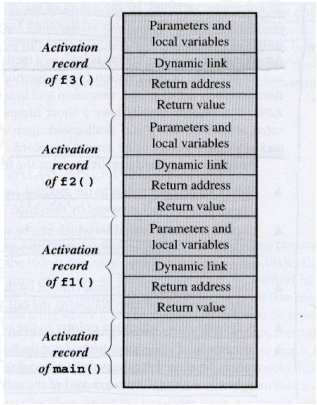
Data Structures and Algorithms in JAVA, Adam Drozdek

17

Técnicas de Projecto de Algoritmos

RECURSÃO: implementação

main() ➔f1()➔f2()➔f3()



18

Técnicas de Projecto de Algoritmos

RECURSÃO: Anatomia de uma Chamada Recursiva

$$x^n = \begin{cases} 1 & if\ n = 0 \\ x \bullet x^{n-1} & if\ n > 0 \end{cases}$$

```
/* 102 */ double power (double x, int n) {
/* 103 */      if (n == 0)
/* 104 */           return 1.0;
                // else
/* 105 */           return x * power(x,n-1);
           }
```

Data Structures and Algorithms in JAVA, Adam Drozdek

```
call 1      power(x,4)
call 2          power(x,3)
call 3              power(x,2)
call 4                  power(x,1)
call 5                      power(x,0)
call 5                      1
call 4                  x
call 3              x·x
call 2          x·x·x
call 1      x·x·x·x
```

Data Structures and Algorithms in JAVA, Adam Drozdek

20

Técnicas de Projecto de Algoritmos

RECURSÃO: versão iterativa

```
/* 102 */ double power (double x, int n) {
/* 103 */     if (n == 0)
/* 104 */         return 1.0;
          // else
/* 105 */         return x * power(x,n-1);
          }
```

Data Structures and Algorithms in JAVA, Adam Drozdek

```
double nonRecPower(double x, int n) {
  double result = 1;
  if (n > 0)
  for (result = x; n > 1; --n)
       result *= x;
  return result;
}
```

;-)

Data Structures and Algorithms in JAVA, Adam Drozdek

Técnicas de Projecto de Algoritmos

Recursão Terminal

```
void tail (int i) {
  if (i > 0) {
    System.out.print (i + "");
    tail(i-1);
  }
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

Recursão não terminal

```
void nonTail (int i) {
  if (i > 0) {
    nonTail(i-1);
    System.out.print (i + "");
    nonTail(i-1);
  }
}
```

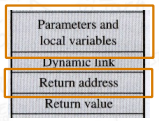
Data Structures and Algorithms in JAVA, Adam Drozdek

Técnicas de Projecto de Algoritmos

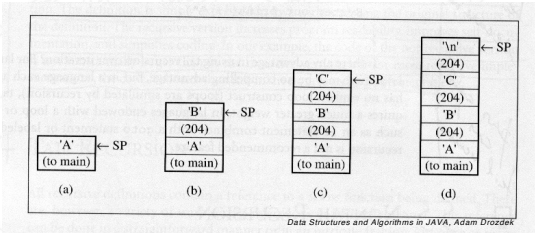
Recursão Não Terminal

```
/* 200 */ void reverse() {
/* 201 */      char ch = getChar();
/* 202 */      if (ch != '\n') {
/* 203 */          reverse();
/* 204 */          System.out.print(ch);
               }
          }
```

Data Structures and Algorithms in JAVA, Adam Drozdek



Data Structures and Algorithms in JAVA, Adam Drozdek



Data Structures and Algorithms in JAVA, Adam Drozdek

33

Técnicas de Projecto de Algoritmos

Recursão Não Terminal

```
/* 200 */ void reverse() {
/* 201 */      char ch = getChar();
/* 202 */      if (ch != '\n') {
/* 203 */          reverse();
/* 204 */          System.out.print(ch);
               }
          }
```

Data Structures and Algorithms in JAVA, Adam Drozdek

Versão Iterativa #1

```
void simpleIterativeReverse() {
        String stack = new String();
        int top = 0;
        try { stack = buffer.readLine();
        } catch (IOException io) {
        }
        for (top = stack.length() - 1; top >= 0; top--)
            System.out.print(stack.charAt(top));
    }
```

Data Structures and Algorithms in JAVA, Adam Drozdek

34

Técnicas de Projecto de Algoritmos

Recursão Não Terminal

```
/* 200 */ void reverse() {
/* 201 */      char ch = getChar();
/* 202 */      if (ch != '\n') {
/* 203 */          reverse();
/* 204 */          System.out.print(ch);
               }
          }
```

Data Structures and Algorithms in JAVA, Adam Drozdek

Versão Iterativa #2

```
void iterativeReverse() {
    char[] stack = new char[80];
    int top = 0;
    stack[top] = getChar();
    while (stack[top] != '\n')
        stack[++top] = getChar();
    for (top -= 1; top >= 0; top--)
        System.out.print(stack[top]);
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

35

Técnicas de Projecto de Algoritmos

Recursão Não Terminal

```
/* 200 */ void reverse() {
/* 201 */      char ch = getChar();
/* 202 */      if (ch != '\n') {
/* 203 */          reverse();
/* 204 */          System.out.print(ch);
               }
          }
```

Versão Iterativa #3

```
void nonRecursiveReverse() {
    Stack st = new Stack();
    char ch = getChar();
    while (ch != '\n') {
        st.push(new Character(ch));
        ch = getChar();
    }
    while (!st.isEmpty())
        System.out.print((Character) st.pop());
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

36

Técnicas de Projecto de Algoritmos

Recursão Indirecta

f() ➔ f_1() ➔ f_2() ➔ ... ➔ f_n() ➔ f()

f() ➔ g_1() ➔ g_2() ➔ ... ➔ g_m() ➔ f()

receive() ➔ decode() ➔ store() ➔ receive() ➔ decode() ...

```
receive(buffer)
   while buffer is not filled up
      if information is still incoming
         get a character and store it in buffer;
      else exit();
      decode(buffer);

decode(buffer)
   decode information in buffer;
   store(buffer);

store(buffer)
   transfer information from buffer to file;
   receive(buffer);
```


Técnicas de Projecto de Algoritmos

Recursão Imbricada

$$h(n) = \begin{cases} 0 & if\ n = 0 \\ n & if\ n > 4 \\ h(2 + h(2n)) & if\ n \le 4 \end{cases}$$

Função de Ackermann

$$A(n,m) = \begin{cases} m+1 & if\ n = 0 \\ A(n-1,1) & if\ n > 0, m = 0 \\ A(n-1, A(n,m-1)) & \text{para os restantes casos} \end{cases}$$

$$A(3,m) = 2^{m+3} - 3$$

$$A(4,1) = 2^{2^{16}} - 3 = 2^{65536} - 3 > 10^{80}$$

Técnicas de Projecto de Algoritmos

Tipos de Recursão

- Terminal (subst. autom. por iteração)
- Não terminal
- Indirecta
- Imbricada

39

Técnicas de Projecto de Algoritmos

RECURSÃO: limitações

Vejamos o caso dos números de Fibonnacci

F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2) p/ n > 1

Como a definição é recursiva torna-se tentador uma implementação recursiva

```
// Calculo do nésimo valor de Fibonacci
// versão incorrecta
public static long fib (int n)
{
  if (n <= 1) return n;
  else return fib(n-1) + fib(n-2);
}
```

PROBLEMA: ***o cálculo por exemplo de F(40) já demora vários minutos !!***

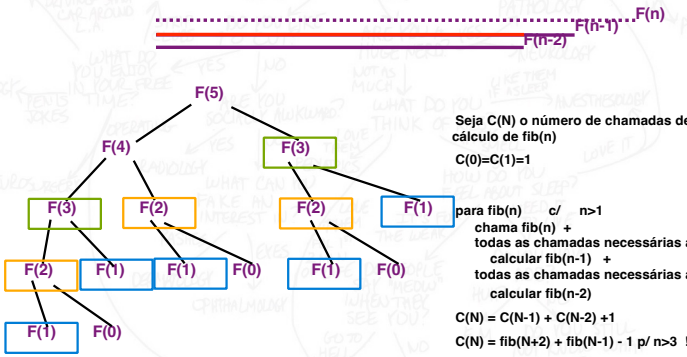
PORQUÊ??

40

Técnicas de Projecto de Algoritmos

RECURSÃO: limitações

... analisemos



Seja C(N) o número de chamadas de fib no cálculo de fib(n)

C(0)=C(1)=1

para fib(n) c/ n>1
chama fib(n) +
todas as chamadas necessárias a calcular fib(n-1) +
todas as chamadas necessárias a calcular fib(n-2)

C(N) = C(N-1) + C(N-2) +1

C(N) = fib(N+2) + fib(N-1) - 1 p/ n>3 !!!!

Assim: p/ N=40 fib(40)=102.334.155 c/ um núm. total de cham. recur.s de 300 x 10^6

41

Técnicas de Projecto de Algoritmos

RECURSÃO: limitações

Conclusão:

Verificar que a resolução recursiva da mesma instância de um problema não conduza à repetição do trabalho computacional.

42

Técnicas de Projecto de Algoritmos

RECURSÃO: limitações

Solução: PROGRAMAÇÃO DINÂMICA

43

Técnicas de Projecto de Algoritmos

RECURSÃO: em destaque

- Um método recursivo é composto por:
- um ou mais casos de base,
- um ou mais casos recursivos.

- Os casos recursivos devem ser definidos em termos de instâncias mais simples deles próprios que "tendem" para um caso de base.

- Existem vários padrões de recursão: terminal (subst. autom. por iteração); não terminal; indirecta; imbricada.

- É comum existir uma *driver routine* que testa a validade da primeira chamada e que a seguir chama a rotina recursiva.

- O processo recursivo pode ser removido recorrendo a uma pilha.

- A recursão não deve ser usada como substituto de estruturas simples de repetição.

- Não repetir trabalho computacional resolvendo a mesma instância de um problema em chamadas recursivas distintas.

44

Técnicas de Projecto de Algoritmos

RECURSÃO: erros comuns

- Caso de base não definido.
- Os casos recursivos não convergem para o caso de base.
- Chamadas recursivas que se sobrepõem (ex. sequência de Fibonnacci) e que conduzem a complexidade exponencial.
- Recursão facilmente substituível por iteração

45

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

Abordagem:

1. **Dividir um problema em subproblemas mais simples a serem resolvidos recursivamente.**
2. **Construir a solução para o problema inicial a partir das soluções para os subproblemas.**

Exemplo:

Probl: Dada uma sequência de inteiros (eventualmente negativos) $A_1, A_2, \ldots, A_N$, encontrar (e identificar a sequência correspondente a) máximo valor de $\sum_{k=i}^{j} A_K$. A subsequência é zero se todos os inteiros forem negativos.

Consideremos por exemplo a sequência de entrada {4, -3, 5, -2, -1, 2, 6, -2} e vamos assim dividir esta sequência em duas metades.

A subsequência máxima pode ocorrer de três formas:

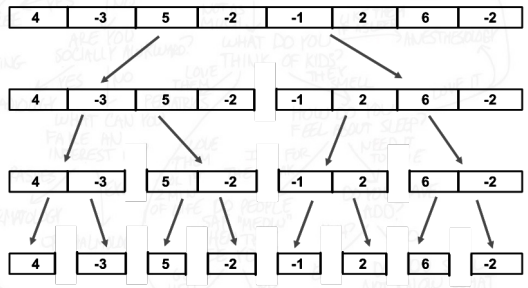
Forma #1: está toda na primeira metade da sequência.
Forma #2: está toda na segunda metade da sequência.
Forma #3: tem início na primeira metade e termina na segunda metade da sequência.

46

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

Forma #1: está toda na primeira metade da sequência.
Forma #2: está toda na segunda metade da sequência.

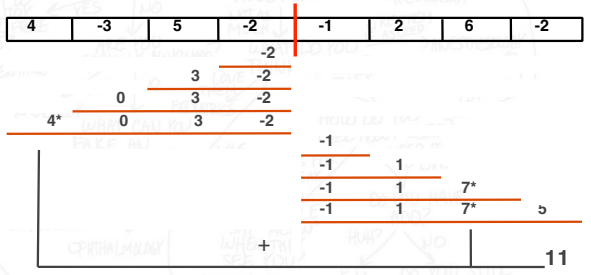


47

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

Forma #3: tem início na primeira metade e termina na segunda metade da sequência.



48

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

Algoritmo

1. Recursivamente calcula a subsequência de soma máxima que se encontra totalmente na primeira metade — **CASO 1**

2. Recursivamente calcula a subsequência de soma máxima que se encontra totalmente na segunda metade — **CASO 2**

3. Calcula por via de dois ciclos consecutivos a subsequência de soma máxima que tem início na primeira metade e se prolonga pela segunda metade — **CASO 3**

4. Selecciona a maior das três somas calculadas em 1. 2. e 3.

49

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

```
private static int maxSumRec( int [ ] a, int left, int right )
  {
      int maxLeftBorderSum = 0, maxRightBorderSum = 0;
      int leftBorderSum = 0, rightBorderSum = 0;
      int center = ( left + right ) / 2;

      if( left == right )  // Base case
          return a[ left ] > 0 ? a[ left ] : 0;

      int maxLeftSum  = maxSumRec( a, left, center );
      int maxRightSum = maxSumRec( a, center + 1, right );

      for( int i = center; i >= left; i-- )
      {
          leftBorderSum += a[ i ];
          if( leftBorderSum > maxLeftBorderSum )
              maxLeftBorderSum = leftBorderSum;
      }

      for( int i = center + 1; i <= right; i++ )
      {
          rightBorderSum += a[ i ];
          if( rightBorderSum > maxRightBorderSum )
              maxRightBorderSum = rightBorderSum;
      }

      return max3( maxLeftSum, maxRightSum,
                   maxLeftBorderSum + maxRightBorderSum );
  }
```


50

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

```
/**
 * Return maximum of three integers.
 */
private static int max3( int a, int b, int c )
{
    return a > b ? a > c ? a : c : b > c ? b : c;
}

/**
 * Driver for divide-and-conquer maximum contiguous
 * subsequence sum algorithm.
 */
public static int maxSubSum4( int [ ] a )
{
    return maxSumRec( a, 0, a.length - 1 );
}
```


51

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

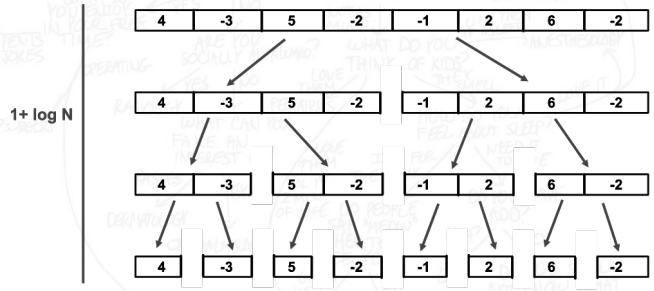
```
/**
 * Simple test program.
 */
public static void main( String [ ] args )
{
    int a[ ] = { 4, -3, 5, -2, -1, 2, 6, -2 };
    int maxSum;

    maxSum = maxSubSum4( a );
    System.out.println( "Max sum is " + maxSum );
}
}
```


52

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

Forma #1: está toda na primeira metade da sequência.
Forma #2: está toda na segunda metade da sequência.

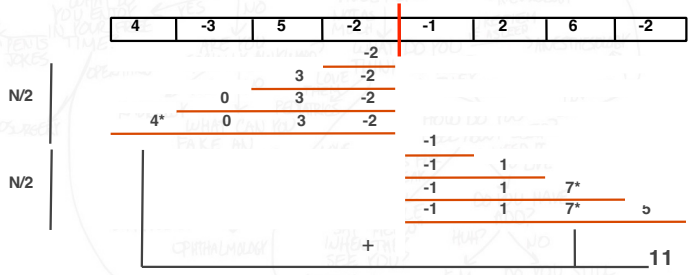


53

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer)

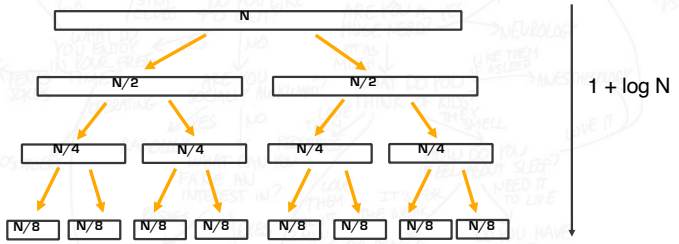
Forma #3: tem início na primeira metade e termina na segunda metade da sequência.



54

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer): análise de complexidade



55

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer): análise de complexidade

```
private static int maxSumRec( int [ ] a, int left, int right )
  {
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
      return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
      leftBorderSum += a[ i ];
      if( leftBorderSum > maxLeftBorderSum )
        maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
      rightBorderSum += a[ i ];
      if( rightBorderSum > maxRightBorderSum )
        maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
  }
```

T(N): tempo que demora resolver o problema da subsquência de soma máxima para um vector de comprimento N

N=1; vamos necessitar de tempo constante; vamos designar esse tempo por 1 unidade de tempo; e temos T(1) = 1

As duas chamadas recursivas resolvem problemas de dimensão N/2; temos então que necessitam de 2 * T(N/2)

O(N)

Temos então:

T(1) = 1

T(N) = 2T(N/2) + O(N)

56

Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer): análise de complexidade

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
        return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
}
```

Temos então:

T(1) = 1

T(N) = 2T(N/2) + O(N)

Assumindo N é uma potência de 2 e substituindo o termo O(N) por N temos:

T(1) = 1

T(N) = 2T(N/2)+N (Eq. 1)

Para N suficientemente grande temos:

T(N/2) = 2T(N/4) + N/2 (Eq. 2)

Substituindo em Eq. 1:

T(N) = 2 * (2T(N/4) + N/2) + N = 4T(N/4) + 2N (Eq. 3)

Usando agora Eq.1 para N/4 temos

T(N/4) = 2T(N/8)+N/4 (Eq. 4)

Substituindo em Eq. 3:

T(N) = 8T(N/8) + 3N (Eq. 5)

... continuando o processo indutivo temos:

$T(N) = 2^kT(N/2^k) + kN$ (Eq. 5)

... fazendo k = log N (pois assumimos $N = 2^k$), temos:

T(N) = NT(1) + log N * N = N logN + N ou seja O(N logN)

57

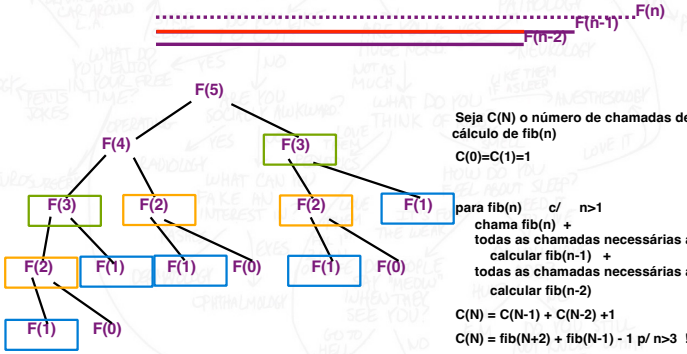
Técnicas de Projecto de Algoritmos

Dividir e Conquistar (Divide and Conquer): análise de complexidade

58

Técnicas de Projecto de Algoritmos
PROGRAMAÇÃO DINÂMICA

... RECORDEMOS a solução recursiva para a sequência de Fibonacci



Seja C(N) o número de chamadas de fib no cálculo de fib(n)

C(0)=C(1)=1

para fib(n) c/ n>1
chama fib(n) +
todas as chamadas necessárias a calcular fib(n-1) +
todas as chamadas necessárias a calcular fib(n-2)

C(N) = C(N-1) + C(N-2) +1

C(N) = fib(N+2) + fib(N-1) - 1 p/ n>3 !!!!

Assim: p/ N=40 fib(40)=102.334.155 c/ um núm. total de cham. recur.s de 300 x 10^6

59

Técnicas de Projecto de Algoritmos
PROGRAMAÇÃO DINÂMICA

SOLUÇÃO:
guardar os resultados intermédios!

60

Técnicas de Projecto de Algoritmos
PROGRAMAÇÃO DINÂMICA

Abordagem Top-Down

- Dividir o problema em subproblemas
- Resolver os subproblemas e memorizar as soluções
- Usar as soluções memorizadas quando de novo necessárias

61

Técnicas de Projecto de Algoritmos
PROGRAMAÇÃO DINÂMICA

Abordagem Bottom-Up

- Todos os sub-problemas cujo resultado é necessário resolvidos antecipadamente
- Sub-soluções aplicadas na resolução do problema de maior dimensão
- VANTAGENS/DESVANTAGENS:
  - menor dimensão da pilha de recursão e menor número de chamadas de funções
  - muitas das vezes não é intuitiva a delimitação de que sub-problemas têm de ser resolvidos para chegar à solução do problema global

62

Técnicas de Projecto de Algoritmos

PROGRAMAÇÃO DINÂMICA: exemplo, seq. de Fibonacci

Usando só recursão

```
function fib(n)
    if n = 0 return 0
    if n = 1 return 1
    return fib(n − 1) + fib(n − 2)
```


63

Técnicas de Projecto de Algoritmos

PROGRAMAÇÃO DINÂMICA: exemplo, seq. de Fibonacci

Usando programação dinâmica

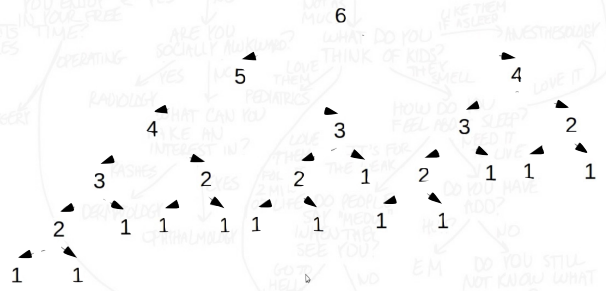
```
var m := map(0 → 0, 1 → 1)
function fib(n)
    if map m does not contain key n
        m[n] := fib(n − 1) + fib(n − 2)
    return m[n]
```


64

Técnicas de Projecto de Algoritmos

PROGRAMAÇÃO DINÂMICA: exemplo, seq. de Fibonacci

Usando programação dinâmica



65

Técnicas de Projecto de Algoritmos

PROGRAMAÇÃO DINÂMICA: exemplo, seq. de Fibonacci

66

Análise de Complexidade

… bom trabalho!

67