Algoritmos Recursivos

Prof. Manoel Ribeiro Filho

Algoritmos recursivos

Muitos problemas computacionais têm a seguinte propriedade.
Cada solução de uma instância do problema contém soluções de instâncias menores do mesmo problema.
Dizemos que tais problemas têm estrutura recursiva.
Para resolver um problema desse tipo, aplique o seguinte método:
Se a instância dada é pequena, na qual sabemos imediatamente a solução S
resolva-a diretamente -> **Caso Base**
Se a instância é grande -> **Caso Recursivo**

1. Reduza-a a uma instancia menor
2. Encontre uma solução S da instância menor
3. Use S para construir uma solução da instância original

Em uma linguagem de programação, esse tipo de problema é resolvido com o uso de uma **função recursiva**, quando uma função pode chamar a si própria. Todo cuidado é pouco ao se fazer funções recursivas. A primeira coisa a se providenciar é um critério de parada. Este vai determinar quando a função deverá parar de chamar a si mesma. O critério de parada é o **Caso Base**.

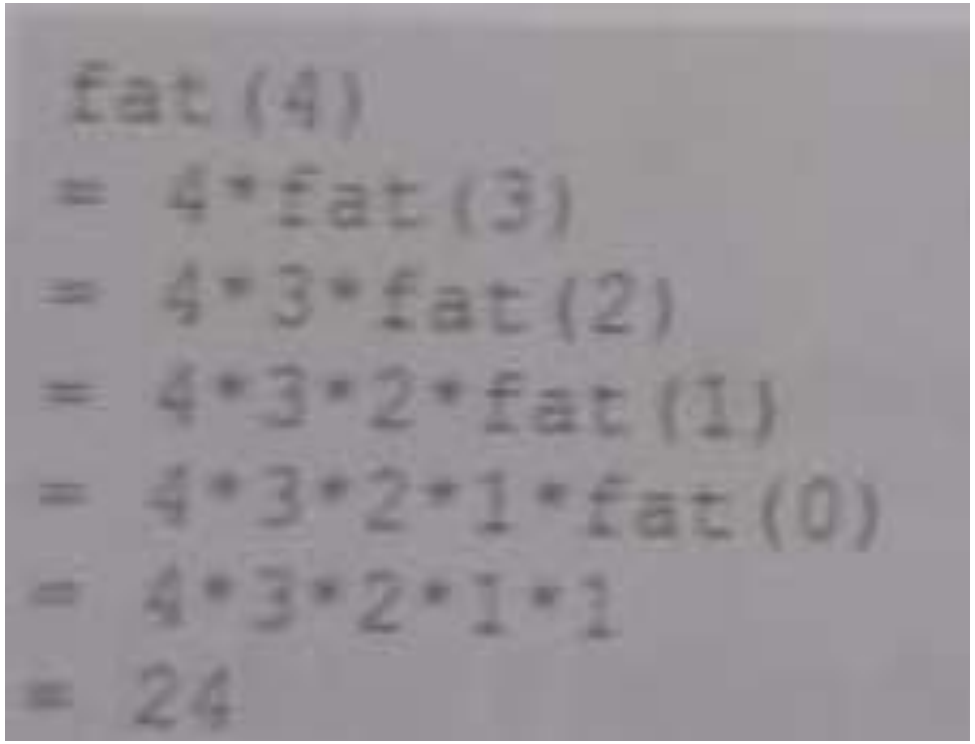
Cálculo do Fatorial

Para n=0 n!=1 caso base
Para n>0 n!=1.2.3..n caso recursivo

Algoritmo Fat(n)
Se n == 0 retorne 1
Se não retorne n*fat(n-1)

```c
#include <stdio.h>
int fat(int n)
{
    if (n==0) return 1;
    else   return n*fat(n-1);
}
 main()
{
    int n;
    printf("\n\nDigite um valor para n: ");
    scanf("%d", &n);
    printf("\nO fatorial de %d e' %d", n, fat(n));
}
```

Uma forma de entender o funcionamento de uma função recursiva é simular sua execução usando **substituições sucessivas**
Nessa simulação, cada chamada da forma fat(n) é substituída por uma expressão da forma n*fat(n-1). Esse processo se repete até que o caso trivial seja alcançado

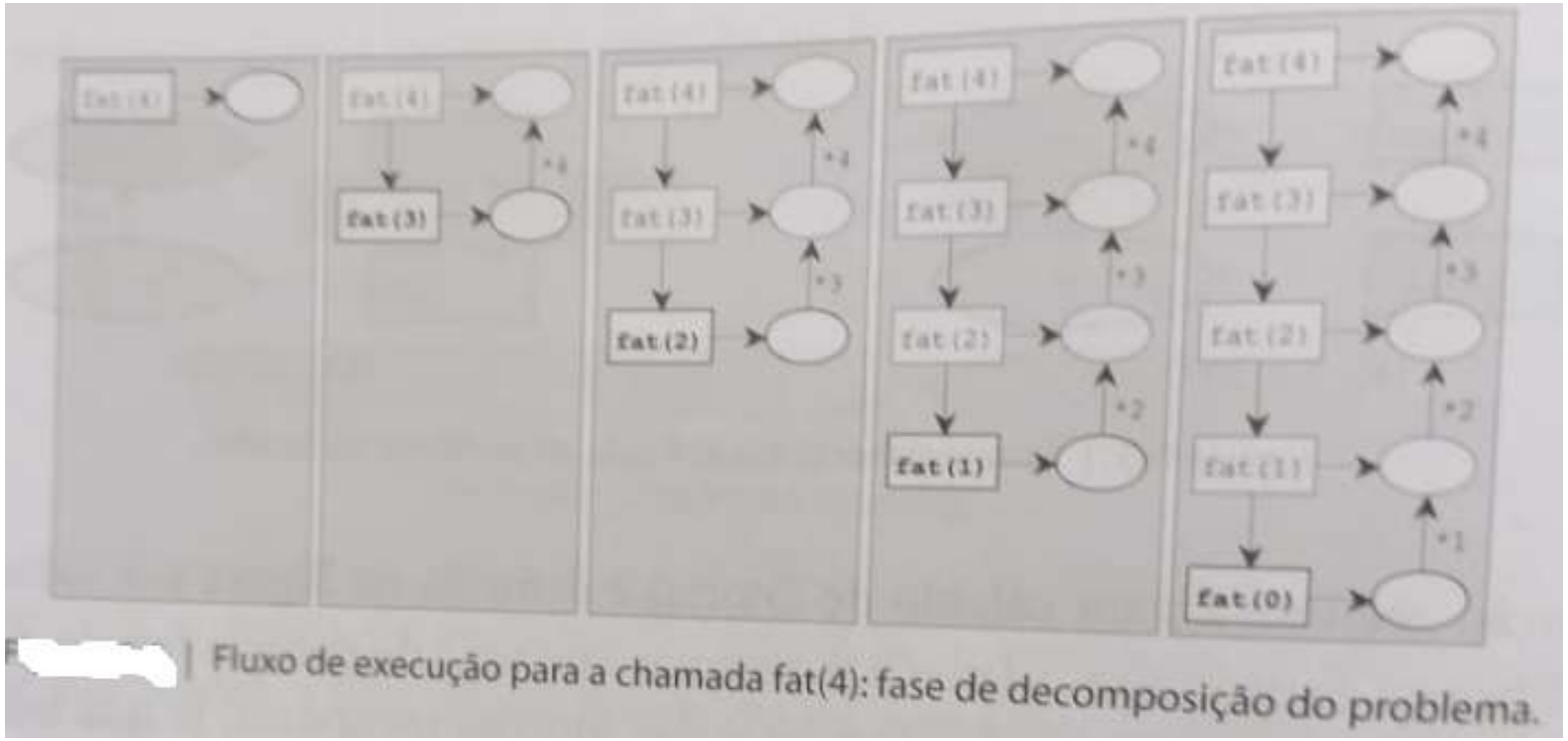


Outra forma de entender a execução de uma função recursiva consiste em desenhar seu **fluxo de execução**.

Esse fluxo de execução tem duas fases:

Fase de decomposição: o passo de recursão é aplicado, repetidamente, até que o caso trivial seja obtido. Nesse ponto, a base de recursão é aplicada e tem início a fase de composição.

Fase de composição: Nessa fase, as soluções dos casos mais simples são usadas para compor as soluções daqueles mais complexas, até que a solução do problema seja obtida.



Fluxo de execução para a chamada fat(4): fase de decomposição do problema.

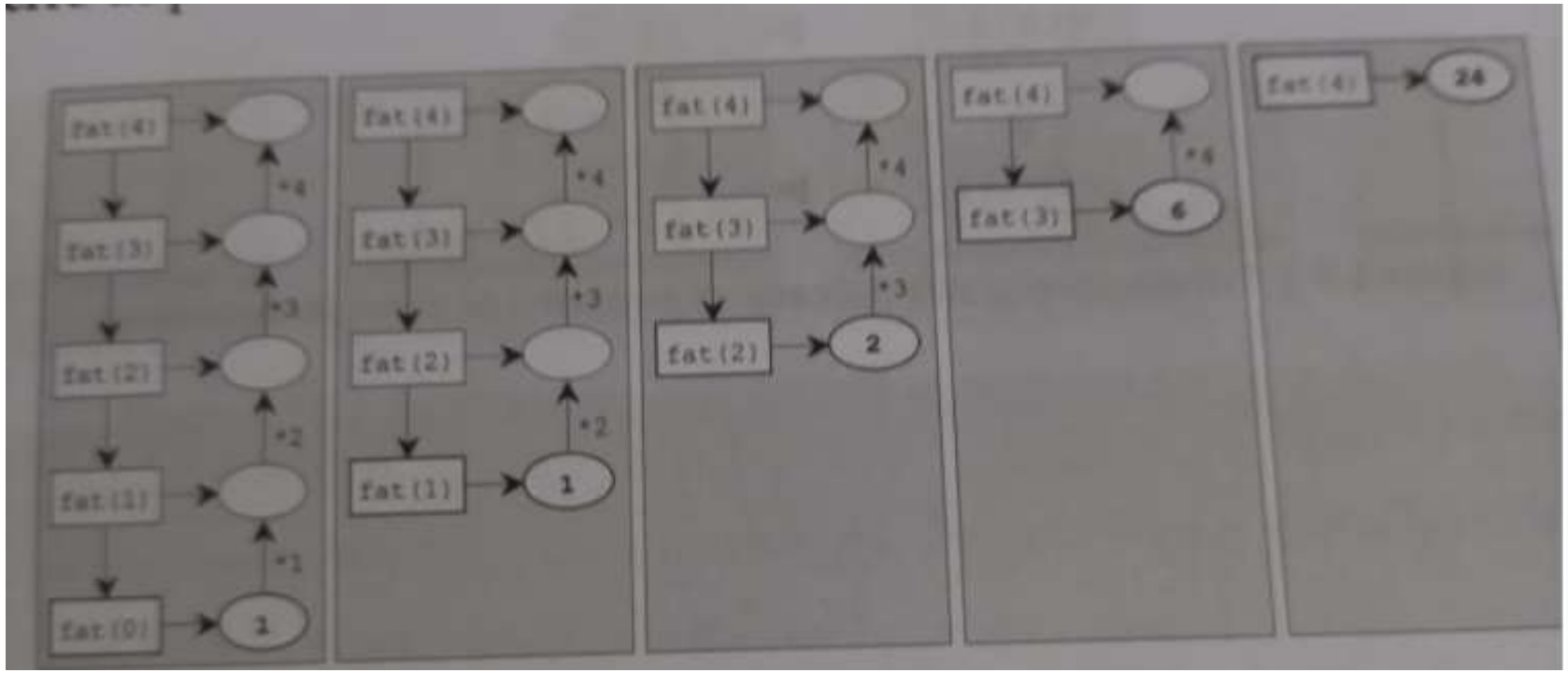
1
2
3
4

Pilha

Na fase de decomposição, a cada chamada recursiva feita, uma multiplicação fica pendente para ser executada mais tarde. Depois, na fase de composição, essas multiplicações são efetuadas na ordem inversa aquela em que eram ficaram pendentes.

Para controlar a ordem em que as operações pendentes devem ser executadas, é usada uma pilha.

Por esse motivo, dependendo da quantidade de chamadas recursivas feita, bem como da capacidade dessa pilha, a execução de uma função recursiva pode terminar com um erro de pilha cheia(stack overflow).

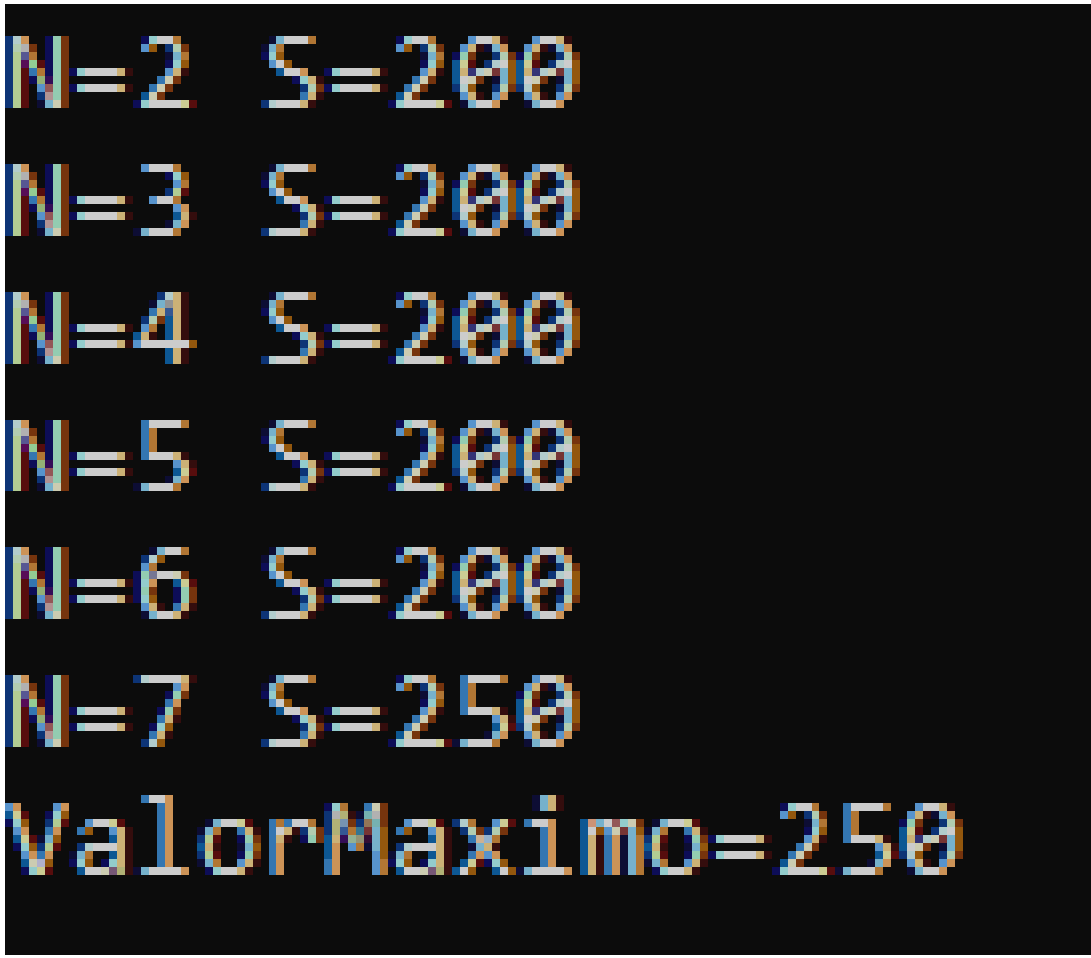


Cálculo do valor máximo de um vetor

```
// O  algoritmo  Max devolve um elemento máximo de V[0..n-1].
// Ele supõe que n >= 1.
Algoritmo Max(V,n)
        Se n== 1 retorne V[0]
        Se não
                S = Max(V,n-1)
                Se  S> V[n-1] retorne S
                Se não retorne V[n-1]
```

```c
#include <stdio.h>
int max(int V[], int n);
main(){
int V[]= {200,5,8,70,2,250,150};
int maximo;
maximo = max(V,7);
printf("ValorMaximo=%d",maximo);
}
int max (int V[], int n) {
        int S;
        if (n == 1) return V[0];
        else {
                S =max(V, n-1);
                printf("N=%d S=%d\n",n,S);
                if (S > V[n-1]) return S;
                else return V[n-1];
                 }
}
```

200
5
8
70
2
250
150



Cálculo da Soma dos elementos de um Vetor

```
// O  algoritmo  Soma devolve a soma dos  elementos de  V[0..n-1].
Algoritmo Soma(V,n)
        Se n== 0 retorne 0
        Se não
                S =Soma(V,n-1)
                S=  S + V[n-1]
                retorne S
```

```
#include <stdio.h>
int somav(int V[], int n);
main(){
int V[]= {200,5,8,70,2,250,150};
int soma;
soma = somav(V,7);
printf("SomaVetor=%d",soma);
}
int somav (int V[], int n) {
                int S;
                if (n == 0) return 0;
                else {
                   S=somav(V, n-1);
                   S=S + V[n-1];
                   printf("n=%d  S=%d\n",n,S);
                              return S;
                               }
}
```

200
5
8
70
2
250
150