Aula 5: Recursividade

Adriano Veloso

Algoritmos e Estruturas de Dados I - DCC/UFMG

Algoritmo Recursivo

- Uma função é recursiva quando ela é definida em termos dela própria
- A linguagem C permite que um programador escreva funções que chamem a si mesmas. Ou seja, há uma ou mais chamadas da função dentro da própria função (chamada direta ou indireta)

Algoritmo Recursivo

- Uma função é recursiva quando ela é definida em termos dela própria
- A linguagem C permite que um programador escreva funções que chamem a si mesmas. Ou seja, há uma ou mais chamadas da função dentro da própria função (chamada direta ou indireta)
- Diretamente recursiva:
  - P **chama** P
- Indiretamente recursiva:
  - P **chama** Q, Q **chama** R, . . ., **chama** P

Algoritmo Recursivo

- Uma função é recursiva quando ela é definida em termos dela própria
- A linguagem C permite que um programador escreva funções que chamem a si mesmas. Ou seja, há uma ou mais chamadas da função dentro da própria função (chamada direta ou indireta)
- Diretamente recursiva:
  - P **chama** P
- Indiretamente recursiva:
  - P **chama** Q, Q **chama** R, . . ., **chama** P
- Conceito poderoso → define conjuntos infinitos com comandos finitos

Exemplo

Para subir uma escada, vamos definir a função que sobe um degrau:

1. Se ao subir esse degrau você atingiu o topo, então PARE
2. Caso contrário:
    1. Avance mais um degrau

Exemplo

Para subir uma escada, vamos definir a função que sobe um degrau:

1. Se ao subir esse degrau você atingiu o topo, então PARE
2. Caso contrário:
    1. Avance mais um degrau

Note que a cada degrau que se sobe, o tamanho do problema diminui

Propriedades

- Algoritmos recursivos são principalmente apropriados quando a solução já está definida em termos recursivos
  - $n! = n \times (n\text{-}1)!$

Propriedades

- Algoritmos recursivos são principalmente apropriados quando a solução já está definida em termos recursivos
  - $n! = n \times (n\text{-}1)!$
- Mesmo assim, isso não garante que a melhor solução seja recursiva

Propriedades

- Algoritmos recursivos são principalmente apropriados quando a solução já está definida em termos recursivos
  - $n! = n \times (n\text{-}1)!$
- Mesmo assim, isso não garante que a melhor solução seja recursiva
- Toda vez que uma função é iniciada recursivamente, um novo conjunto de variáveis locais e de parâmetros é alocado, e somente essas variáveis podem ser referenciadas dentro dessa chamada
  - Ao terminar a chamada de uma função, todas as variáveis são liberadas

Propriedades

Vantagens:

- Redução do tamanho do código fonte
- Permite descrever algoritmos de forma mais clara e concisa

Desvantagens:

- Redução do desempenho devido ao tempo de gerenciamento de chamadas
- Dificuldade de depuração, principalmente quando a recursão for muito profunda

Recursão

Toda recursão é composta por:

- **Caso base**: Uma instância do problema que pode ser solucionada facilmente
- **Chamadas recursivas**: A solução define-se em termos de si mesma, procurando sempre convergir para o caso base. Por exemplo, a soma de uma lista de n elementos pode ser definida a partir da soma de $n - 1$ elementos.

Exemplo

Fatorial:

- $0! = 1$
- $n! = n \times (n\text{-}1)!$

Exemplo

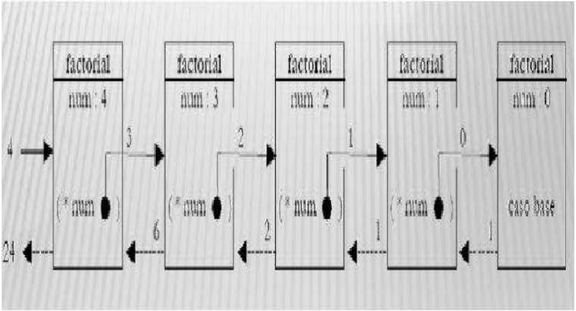
Fatorial:

- 0! $= 1$
- n! $= n \times (n-1)!$

Implementação:

```
int fatorial(int n) {
    if(n==0) return(1);            (caso base)
    return(n*fatorial(n-1));       (chama recursiva)
}
```

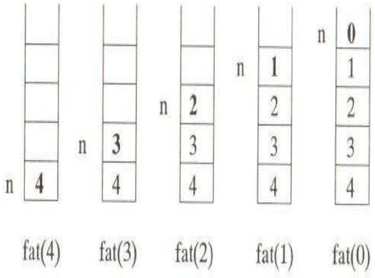
Execução


factorial
num : 4
factorial
num : 3
factorial
num : 2
factorial
num : 1
factorial
num : 0
4
3
2
1
0
(* num ●)
(* num ●)
(* num ●)
(* num ●)
caso base
24
6
2
1
1

Execução

```
fatorial (6)
6 * fatorial (5)
6 * 5 * fatorial (4)
6 * 5 * 4 * fatorial (3)
6 * 5 * 4 * 3 * fatorial (2)
6 * 5 * 4 * 3 * 2 * fatorial (1)
6 * 5 * 4 * 3 * 2 * 1 * fatorial (0)
6 * 5 * 4 * 3 * 2 * 1 * 1
6 * 5 * 4 * 3 * 2 * 1
6 * 5 * 4 * 3 * 2
6 * 5 * 4 * 6
6 * 5 * 24
6 * 120
720
```

Execuçãoem Pilha



Passos

1. Entender o problema
   - 1, 1, 2, 3 ,5, 8, 13, 21, 34, 55, . . .

Passos

1. Entender o problema
   - 1, 1, 2, 3 ,5, 8, 13, 21, 34, 55, ...
2. Formular a solução por meio de funções recursivas
   - Fibonacci(1) $= 1$
   - Fibonacci(2) $= 1$
   - Fibonacci(3) $= 2$
   - Fibonacci(4) $= 3$
   - Fibonacci(5) $= 5$
   - ...

Passos

1. Entender o problema
   - 1, 1, 2, 3 ,5, 8, 13, 21, 34, 55, ...
2. Formular a solução por meio de funções recursivas
   - Fibonacci(1) = 1
   - Fibonacci(2) = 1
   - Fibonacci(3) = 2
   - Fibonacci(4) = 3
   - Fibonacci(5) = 5
   - ...
3. Qual é a definição recursiva?
   - Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
   - Fibonacci(1) = 1, Fibonacci(2) = 1

Passos

1. Entender o problema
   - 1, 1, 2, 3 ,5, 8, 13, 21, 34, 55, . . .
2. Formular a solução por meio de funções recursivas
   - Fibonacci(1) = 1
   - Fibonacci(2) = 1
   - Fibonacci(3) = 2
   - Fibonacci(4) = 3
   - Fibonacci(5) = 5
   - . . .
3. Qual é a definição recursiva?
   - Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
   - Fibonacci(1) = 1, Fibonacci(2) = 1
4. Implementar

Indução

Recursão é uma forma indutiva de definir um algoritmo. Temos dois tipos de indução:

- Indução fraca: a solução de tamanho t pode ser obtida a partir de sua solução de tamanho $t - 1$

Indução

Recursão é uma forma indutiva de definir um algoritmo. Temos dois tipos de indução:

- Indução fraca: a solução de tamanho t pode ser obtida a partir de sua solução de tamanho $t-1$
- Indução forte: a solução de tamanho t depende de suas soluções de tamanho t', para todo $t' < t$. Essa estratégia também é conhecida por **Divisão e Conquista**.

Exemplo

Calcular a soma dos inteiros no intervalo $[m, n]$.

- Ex: $[1, 4] = 1+2+3+4 = 10$.

Exemplo

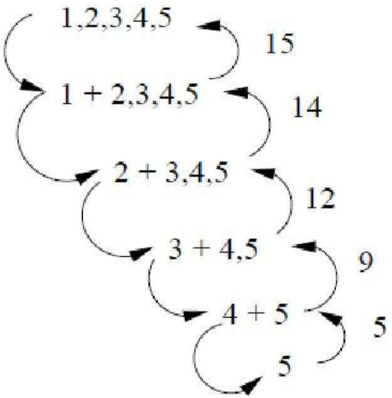
Calcular a soma dos inteiros no intervalo $[m, n]$.

- Ex: $[1, 4] = 1+2+3+4 = 10$.

Solução por indução fraca:

```
int soma(int n, int m) {
    if(m==n) {
        return(m);
    }
    else {
        return(m+soma(m+1, n));
    }
}
```

Indução Fraca



Exemplo

Calcular a soma dos inteiros no intervalo $[m, n]$.

- Ex: $[1, 4] = 1+2+3+4 = 10$.

Exemplo

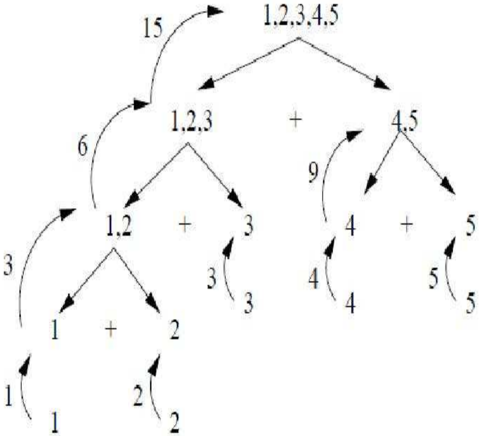
Calcular a soma dos inteiros no intervalo $[m, n]$.

- Ex: $[1, 4] = 1+2+3+4 = 10$.

Solução por indução forte:

```
int soma(int n, int m) {
    if(m==n) {
        return(m);
    }
    else {
        return(soma(m, (m+n)/2), soma((m+n)/2 + 1, n));
    }
}
```

Indução Forte



Contato

adrianov@dcc.ufmg.br