FIAP ON

CÓDIGOS DE ALTA PEFORMANCE

ALGORITMOS RECURSIVOS

PATRICIA MAGNA

3

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

3 ALGORITMOS RECURSIVOS

3.1 Introdução

Provavelmente, você já viu imagens que se repetem ao infinito ao colocarmos um espelho de frente para outro. Isso é uma recursão! Claro, considerando os espelhos como o mesmo objeto. A figura Rercusão na TV vai exemplificar melhor o efeito da recursão de maneira gráfica.

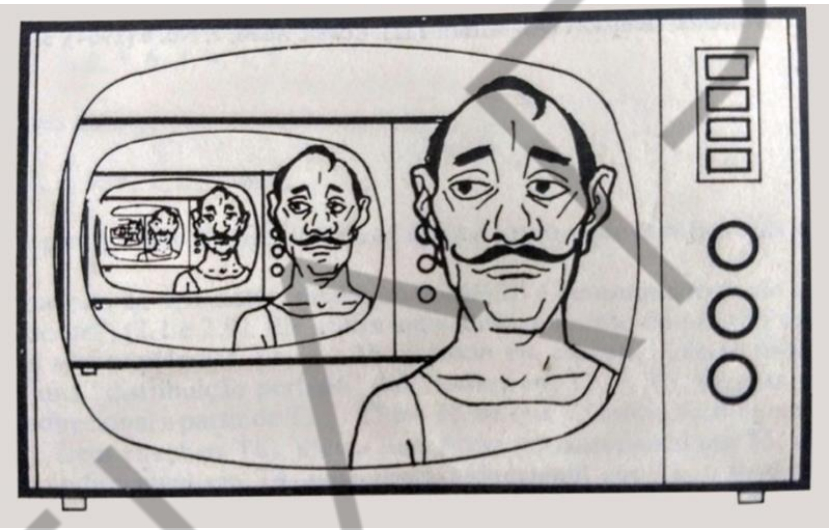

Figura 3.1 Rercusão na TV.
Fonte: Wirth, N. (1989).

Formalmente, a recursão auxilia a definir alguns problemas matemáticos, mas ela pode ser aplicada ao dia a dia de maneira bem constante. Alguns exemplos mais comuns que você pode encontrar na literatura são: conjunto de números naturais, estrutura de dados, como árvores, fatorial, etc,

E já que estamos na matemática, vamos à definição matemática formal para recursividade. Um objeto é dito recursivo se consistir parcialmente ou se for definido em termos de si próprio. Em termos matemáticos, a recursão permite a definição de conjuntos infinitos através de formulações finitas.

Por exemplo, para os números naturais:

- Zero é um número natural;
- O sucessor de um número natural é um número natural.

Recursividade é um conceito no qual a solução de um problema é expressa como combinação de soluções de problemas idênticos, porém menores. A solução de menor problema consiste de um caso extremo, que são premissas sobre as quais a solução recursiva é criada.

A existência da solução para o caso mais simples é necessária para que as equações da definição recursiva possam ser resolvidas.

Calma, não entre em pânico, tudo isso é para auxiliá-lo a aprender a recursão e saber como fazer seus algoritmos melhores. Você vai conseguir destrinchar os caminhos de um código melhor e mais elegante.

3.2 Recursividade em Sistemas Computacionais

A recursividade é um método comum, no qual é feita uma simplificação ou divisão do problema em subproblemas do mesmo tipo, que são resolvidos reaplicando a mesma função. Como técnica de programação, esse método é conhecido como “dividir para conquistar” e é a chave para a construção de muitos algoritmos importantes.

Um programa computacional recursivo pode executar um número, teoricamente, infinito de cálculos sem declaração explícita de repetições. Na prática por limitações de armazenamento em memória e processamento, é obrigatório que esse número seja finito para que exista utilidade prática nos cálculos.

Em geral, um programa recursivo possui um trecho não recursivo. Esse último estabelece uma condição de parada da recursão.

De forma geral:

$$f(x) = \begin{cases} \textbf{f(y), se <condição>} \\ \textbf{valor, caso contrário} \end{cases}$$

A ferramenta necessária e suficiente para implementar recursão é a função que permite a sua própria chamada dentro de seu código.

Existem dois tipos de recursão: direta e indireta. Na recursão direta a função realiza uma chamada a si mesma para obter o resultado. Já na indireta existe uma função A que chama a função B, e essa chama novamente a função A.

3.2.1 CÁLCULO DE FATORIAL

Um problema clássico que pode ser resolvido utilizando recursividade é o problema do fatorial de um número inteiro não negativo. Pela definição matemática:

$$n! = \begin{cases} n*(n-1)!\text{ , se } n \neq 0 \\ 1\text{, caso contrário} \end{cases}$$

Recordando, a implementação iterativa do fatorial em linguagem JAVA seria:

```
int fat=1;
for (int i=n; i>1; i--){
    fat=fat*i;
}
```

Código Fonte 3.1 – Trecho de programa para cálculo da função fatorial usando iteração em JAVA.
Fonte: elaborado pelo autor (2018)

A versão recursiva deste problema seria:

```
public static int fatorial (int n)
{
     if (n==0)
               return(1);
          else
               return(n*fatorial(n-1));
}

public static void main(String[] args) {
      Scanner entra = new Scanner(System.in);
      System.out.print("Digite valor inteiro positivo: ");
      int n = entra.nextInt();
      System.out.println("Fatorial de "+ n + " = " + fatorial(n));
      entra.close();
}
```

Código Fonte 3.2 – Programa com função recursiva para o cálculo da função fatorial em JAVA.
Fonte: elaborado pelo autor (2018)

Para podermos fazer o “teste de mesa”, ou seja, apresentarmos o passo a passo da execução dessa função recursiva é preciso recordar como fazemos o teste de mesa da execução de funções em geral. Suponha que a função main() chama a

função teste(). Quando essa é chamada dizemos, que ativamos a função teste() e, depois de passar os parâmetros necessários à função teste(), sua execução inicia a partir do seu primeiro comando. Quando ela termina, sua ativação é “fechada”, retornando o resultado para função main(), que prossegue sua execução do comando seguinte ao momento da chamada à função teste().

Quando fazemos a chamada de uma função recursiva, não vamos poder referenciar cada chamada pelo nome da função (que será sempre o mesmo) assim, numeramos cada ativação da função para podermos identificar cada ativação específica.

Voltando, agora, ao nosso exemplo de função fatorial recursiva, a cada vez que a função fatorial é chamada, é criada uma nova ativação, ou seja, é utilizada uma alocação de diferentes posições de memória para alocar os parâmetros e a função é iniciada pelo primeiro comando (no caso `if(n==0)`).

Assim, a sequência de ativações que serão executadas será:

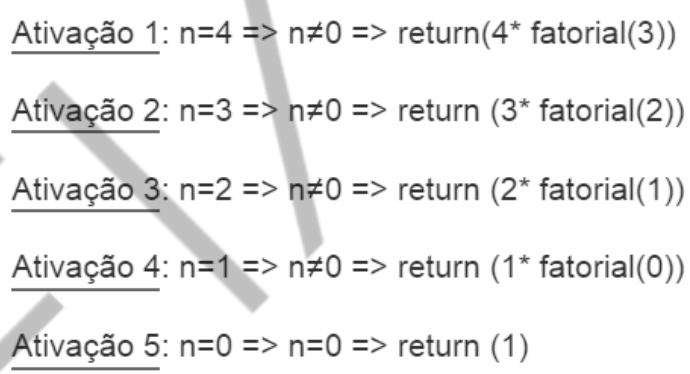

Figura 3.2–Criação das ativações da função fatorial
Fonte: elaborado pelo autor (2018).

Quando termina a ativação 5 da função fatorial, essa retorna para a função que a chamou, que foi a ativação 4 da função fatorial dentro do comando return(). Assim, podemos representar as finalizações e retornos de cada ativação da forma como apresentada na Figura Finalização de cada ativação da função fatorial e seu retorno para ativação anterior.

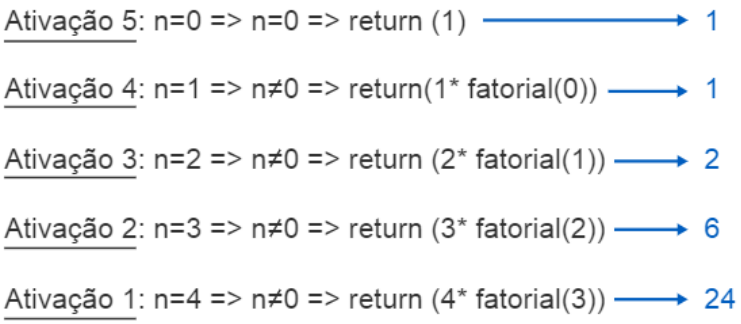

Figura 3.3–Finalização de cada ativação da função fatorial e seu retorno para ativação anterior
Fonte: elaborado pelo autor (2018).

Uma forma diferente de representar as várias ativações de chamadas de funções recursivas é usar um grafo, no qual cada aresta representa uma ativação e o valor do parâmetro de entrada nessa ativação. A Figura Representação gráfica das ativações da função fatorial contém a representação gráfica da execução da função fatorial, que deve ser observada seguindo as setas de cima para baixo, e depois de atingir o último nível de ativação, vai retornando o valor de cada ativação para a ativação anterior.

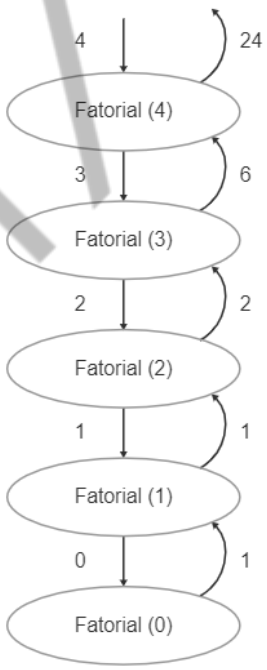

Figura 3.4– Representação gráfica das ativações da função fatorial.
Fonte: elaborado pelo autor (2018).

Outro exemplo de função recursiva é utilizado para o cálculo do máximo divisor comum (MDC) entre dois números, cuja implementação seria:

```
public static int mdc(int n, int m){
       if (n > m)
            return(mdc(m,n));
       else{
            if (n==0)
                  return(m);
            else
                  return(mdc(n,m%n));
            }
       }

public static void main(String[] args) {
       Scanner entra = new Scanner(System.in);
       System.out.print("Digite 2 dados inteiros: ");
       int n = entra.nextInt();
       int m = entra.nextInt();
       System.out.println("mdc de "+ n + "," + m + " = " + mdc(n,m));
       entra.close();
       }
```

Código Fonte 3.3 – Programa com função recursiva para o cálculo da função MDC em JAVA.
Fonte: elaborado pelo autor (2018)

Vamos fazer o passo a passo, supondo que os valores de n e m sejam 8 e 4, respectivamente.

Ativação 1: n=8 e m=4 => n>m => return(mdc(4,8))

Ativação 2: n=4 e m=8 => n<m e n≠0 => return(mdc(4, 0))

Ativação 3: n=4 e m=0 => n>m => return(mdc(0,4))

Ativação 4: n=0 e m=4 => n<m e n=0 => return (4)

Figura 3.5–Criação das ativações da função para calcula máximo divisor comum
Fonte: elaborado pelo autor (2018).

Quando chegamos à ativação 4, o valor de máximo divisor comum entre 8 e 4 é encontrado e o valor 4 é retornado, finalizando a ativação 4, voltando para a função que a chamou, ou seja, a ativação 3. Essa também é finalizada, retornando o resultado e voltando para a ativação 2, que então é finalizada, retornando o resultado. E finalmente, a ativação 1 recebe o resultado e é concluída retornando o valor de resultado (4). Como mostra o esquema da Figura Retorno das ativações da função para calcula máximo divisor comum.

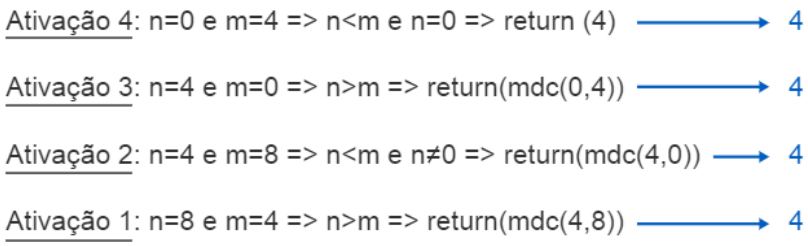

Figura 3.6– Retorno das ativações da função para calcula máximo divisor comum
Fonte: elaborado pelo autor (2018).

3.3 Profundidade da Recursão

Profundidade refere-se ao número de vezes que uma rotina recursiva chama a si própria, até obter o resultado.

Para a função fatorial anterior, com n=4, é 4, pois foi o número de vezes que essa função a chamou. De uma forma mais geral, a profundidade da função recursiva fatorial de n é sempre n. Na representação gráfica, é bem mais fácil identificar a profundidade de uma função recursiva.

Nem sempre é tão simples saber a profundidade da função para o máximo divisor comum entre dois números, já há uma dependência de m e n.

3.4 Utilização de Recursividade em Estruturas de Dados

Até agora, foram apenas apresentados algoritmos de recursividade para resolução de problemas matemáticos, no entanto, é importante dizer que o conceito pode ser usado em diversos tipos de problemas.

Nosso foco é utilizar recursividade para algoritmos de manipulação de estruturas de dados, tanto para sua criação como para o armazenamento e ordenação de dados.

Vamos começar a usá-la na próxima estrutura de dados que estudaremos, chamada de **árvores**. Essas estruturas de dados são amplamente usadas, pois a partir de sua organização no armazenamento dos dados, a consulta e a recuperação de uma informação tornam-se extremamente eficientes. Mas ainda não temos condições de descrever como essa eficiência pode ser obtida, precisamos primeiro

definir e entender o que são árvores. Esse é o tema do próximo capítulo, mas antes, vamos exercitar.

Exercícios de Fixação

Apresente o passo a passo da execução da função mdc supondo que os valores digitados sejam n=28 e m=12.

Ativação 1: n=28 e m=12 => n>m => return(mdc(12,28))

Ativação 2: n=12 e m=28=> n<m e n≠0 => return(mdc(12, 4))

Ativação 3: n=12 e m=4 => n>m => return(mdc(4,12))

Ativação 4: n=4 e m=0 => n>m => return(mdc(0,4))

Ativação 5: n=0 e m=4=>n<m e n=0 => return (4)

REFERÊNCIAS

ASCÊNCIO, A. F. G.; ARAUJO, G. S. **Estruturas de Dados**: Algoritmos, Análise de Complexidade e Implementações em JAVA e C/C++. São Paulo: Pearson Prentice Hall, 2010.

ASCÊNCIO, A. F. G.; CAMPOS, E. A. V. **Fundamentos da Programação de Computadores**: algoritmos, Pascal, C/C++ e Java. São Paulo: Pearson Prentice Hall, 2007.

DEITEL, P. J.; DEITEL, H. M. **Java**: como programar. 8. ed. São Paulo: Pearson Prentice Hall, 2010.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. São Paulo: Pearson Prentice Hall, 2005.

PEREIRA, S. L.; **Estruturas de Dados Fundamentais**: Conceitos e Aplicações. São Paulo: Érica, 1996.

PUGA, S.; RICETTI, G. **Lógica de Programação e Estruturas de Dados**. São Paulo: Pearson Education do Brasil, 2016.

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. **Estruturas de dados e seus algoritmos**. Rio de Janeiro: LTC, 1994.

TENEMBAUM, A. M. et al. **Estruturas de Dados usando C.** São Paulo: Makron Books, 1995.