

Recursão

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

- Recursão
- Exemplos de Problemas Recursivos
 - Torre de Hanói
 - Sequência de Fibonacci
- Simulando a Recursividade
- Exercícios

- Na matemática, vários objetos são definidos através de um processo que os produz
- Exemplo: função fatorial
 - Dado um número positivo n , o seu respectivo fatorial é
 - $n! = 1$, se $n = 0$ ou $n = 1$
 - $n! = n * (n - 1) * (n - 2) * \dots * 1$, se $n > 1$

- Exemplo de um algoritmo para o cálculo do fatorial de n

```
int fatorial(int n){  
    int i, f = 1;  
  
    for (i = n; i > 1; i--)  
        f *= i;  
  
    return f;  
}
```

- Esse algoritmo é iterativo

- O algoritmo anterior pode ser traduzido para uma função que retorne $n!$ quando recebe n como parâmetro
 - $n! = 1$, se $n = 0$
 - $n! = n * (n - 1)!$, se $n > 0$
- A função fatorial está definida em termos de si mesma
- Essa é uma definição recursiva do fatorial

Recurso

- Recursão significa repetição
- Uma função recursiva é uma função chama a si própria
- Divisão conquista
- Exemplos de aplicação de recursão: árvores, grafos, ordenação, busca, etc

- Implementação recursiva da função fatorial

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


Recursão

- O critério de parada é uma preocupação (assim como em um comando repetitivo)



- A recursão pode ser infinita

- Três regras para a recursão
 - ① Saber quando parar (critério de parada)
 - Caso base
 - ② Decidir como modificar o seu estado (ou entrada(s)) em direção ao caso base
 - ③ Analisar o problema de forma que possa ser dividido em subproblemas
 - Caso indutivo

- Exemplo: função fatorial

$$n! = \begin{cases} 1, & \text{se } n \leq 1 \text{ (caso base)} \\ n * (n - 1)!, & \text{se } n > 1 \text{ (passo indutivo)} \end{cases}$$

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

- Na função acima, é possível ver que o caso base é tomado quando $n \leq 1$
- Caso $n > 1$, então não estaremos no caso base e o resultado será $n * (n - 1)$

- exemplo

fatorial(5)

5 * fatorial(4)

5 * 4 * fatorial(3)

5 * 4 * 3 * fatorial(2)

5 * 4 * 3 * 2 * fatorial(1)

5 * 4 * 3 * 2 * 1

5 * 4 * 3 * 2

5 * 4 * 6

5 * 24

120

- Exemplo: somatório em vetor

```
int somatorio(int v[], int n){  
    if (n < 1)  
        return 0;  
    else  
        return v[n - 1] + somatorio(v, n - 1);  
}
```

- Caso base: $n < 1$
- Caso indutivo: $n > 1$

- Exemplo: somatório em vetor (exemplo 2)

```
int somatorio_rec(int v[], int i, int m){  
    if (i < m)  
        return v[i] + somatorio(v, i + 1, m);  
    else  
        return 0;  
}
```

```
int somatorio(int v[], int m){  
    return somatorio_rec(v, 0, m);  
}
```

- A *somatorio_rec* é uma imitação explícita de um laço *for*
- Caso base: quando o índice *i* atinge o tamanho do vetor
- Caso indutivo: o índice *i* está dentro dos limites do vetor

- É importante enfatizar que uma função recursiva pode ter mais de um caso base e/ou indutivo
- Para cada chamada de uma função, recursiva ou não, os parâmetros e as variáveis locais são empilhados na pilha de execução
- Execução
 - Em qualquer chamada recursiva é criado um registro de ativação na pilha de execução
 - Na pilha são armazenados o endereço de retorno, os parâmetros, as variáveis locais da função
 - No final da execução, o registro é desempilhado e a execução volta ao subprograma que chamou
 - O processo é repetido até que a pilha esteja vazia ou até que a função recursiva seja completamente finalizada

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

fatorial(5)
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

fatorial(4)
fatorial(5)
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

fatorial(2)
fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

1
fatorial(2)
fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

fatorial(2)
fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

2
fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

fatorial(3)
fatorial(4)
fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

6
fatorial(4)
fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

fatorial(4)
fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

24
fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

fatorial(5)
main()

Recursão

```
int fatorial(int n){  
    if (n <= 1)  
        return 1;  
    else  
        return n * fatorial(n - 1);  
}  
  
int main(){  
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));  
  
    return 0;  
}
```

Pilha de Execução

120
main()

Recursão

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

int main(){
    printf("Fatorial de %d:  %d\n", 5, fatorial(5));

    return 0;
}
```

Pilha de Execução

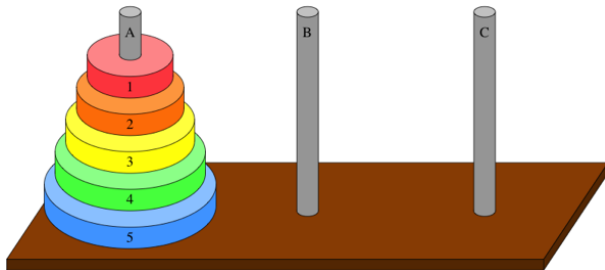
main()

- O desenvolvimento de uma solução recursiva nem sempre é fácil
- Normalmente, não há porque propor uma solução recursiva
- Entretanto, para alguns problemas a solução recursiva é mais elegante
- Há problemas que são recursivos já na sua definição
 - Para estes, a solução recursiva é mais natural

Exemplos de Problemas Recursivos

Exemplos de Problemas Recursivos

Torre de Hanói



- Dado três torres (A , B e C) e n discos de diâmetros diferentes
- O disco de menor diâmetro sempre deve estar em cima do disco de maior diâmetro

Exemplos de Problemas Recursivos

Torre de Hanói

- Inicialmente, todos discos deve estar na torre A
- Problema: como colocar todos os discos na torre C , utilizando a torre intermediária B , sem inverter a ordem dos diâmetros em nenhum torre?
- Se há solução para $n - 1$ discos, então há solução para n discos
- No caso trivial de $n = 1$, a solução é simples
- A solução para n discos é realizada em termos de $n - 1$

Exemplos de Problemas Recursivos

Torre de Hanói

- Procedimento
 - Se $n = 1$, mover o disco da torre A para C
 - Mova os $n - 1$ discos de A para C , usando B como auxiliar
 - Mova o último disco de A para C
 - Mova os $n - 1$ discos de B para C , usando A como auxiliar

Exemplos de Problemas Recursivos

Torre de Hanói

```
void hanoi(char de, char para, char via, int n){  
    if (n >= 1)  
        hanoi(de, via, para, n - 1);  
        printf("disco %d de %c para %c\n", n, de, para);  
        hanoi(via, para, de, n - 1);  
    }  
}
```

Exemplos de Problemas Recursivos

Torre de Hanói

- Exemplo para $de = A$, $para = C$, $meio = B$ e $n = 5$

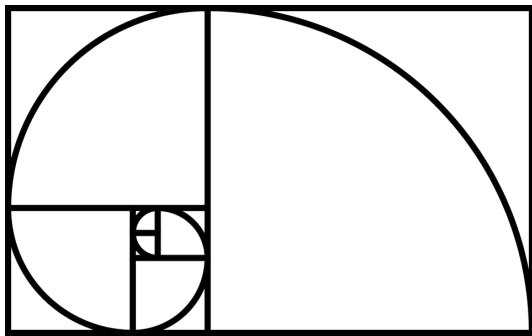
1 disco 1 de A para C
2 disco 2 de A para B
3 disco 1 de C para B
4 disco 3 de A para C
5 disco 1 de B para A
6 disco 2 de B para C
7 disco 1 de A para C
8 disco 4 de A para B
9 disco 1 de C para B
10 disco 2 de C para A
11 disco 1 de B para A
12 disco 3 de C para B
13 disco 1 de A para C
14 disco 2 de A para B
15 disco 1 de C para B
16 disco 5 de A para C

17 disco 1 de B para A
18 disco 2 de B para C
19 disco 1 de A para C
20 disco 3 de B para A
21 disco 1 de C para B
22 disco 2 de C para A
23 disco 1 de B para A
24 disco 4 de B para C
25 disco 1 de A para C
26 disco 2 de A para B
27 disco 1 de C para B
28 disco 3 de A para C
29 disco 1 de B para A
30 disco 2 de B para C
31 disco 1 de A para C

Exemplos de Problemas Recursivos

Sequência de Fibonacci

- Em uma sequência de Fibonacci, a partir de 0 e 1, cada número subsequente é correspondente à soma dos dois anteriores
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, ...



Exemplos de Problemas Recursivos

Sequência de Fibonacci

- Presentes em configurações biológicas: caracol, desenrolar da samambaia
- Aplicação interessante: conversão de milhas para quilômetros
 - Exemplo: para converter 5 milhas, pode ser utilizado o número seguinte da sequência de Fibonacci (8 km)

$$f(n) = \begin{cases} 0, & \text{se } n = 0 \\ 1, & \text{se } n = 1 \\ f(n-1) + f(n-2), & \text{se } n > 1 \end{cases}$$

Exemplos de Problemas Recursivos

Sequência de Fibonacci

```
int fib(int n){  
    if (n <= 0)  
        return 0;  
    else if (n == 1)  
        return 1;  
    else  
        return fib(n - 1) + fib(n - 2);  
}
```

Exemplos de Problemas Recursivos

Sequência de Fibonacci

- Exemplo para $n = 6$

- $\text{fib}(6) = \text{fib}(5) + \text{fib}(4)$
- $= \text{fib}(4) + \text{fib}(3) + \text{fib}(4)$
- $= \text{fib}(3) + \text{fib}(2) + \text{fib}(3) + \text{fib}(4)$
- $= \text{fib}(2) + \text{fib}(1) + \text{fib}(2) + \text{fib}(3) + \text{fib}(4)$
- $= \text{fib}(1) + \text{fib}(0) + \text{fib}(1) + \text{fib}(2) + \text{fib}(3) + \text{fib}(4)$
- $= 1 + \text{fib}(0) + \text{fib}(1) + \text{fib}(2) + \text{fib}(3) + \text{fib}(4)$
- $= 1 + 0 + \text{fib}(1) + \text{fib}(2) + \text{fib}(3) + \text{fib}(4)$
- $= 1 + 1 + \text{fib}(2) + \text{fib}(3) + \text{fib}(4)$
- $= 2 + \text{fib}(1) + \text{fib}(0) + \text{fib}(3) + \text{fib}(4)$
- $= 2 + 1 + \text{fib}(0) + \text{fib}(3) + \text{fib}(4)$
- $= 3 + 0 + \text{fib}(3) + \text{fib}(4)$
- $= 3 + \text{fib}(2) + \text{fib}(1) + \text{fib}(4)$
- $= 3 + \text{fib}(1) + \text{fib}(0) + \text{fib}(1) + \text{fib}(4)$
- $= 3 + 1 + \text{fib}(0) + \text{fib}(1) + \text{fib}(4)$
- $= 4 + 0 + \text{fib}(1) + \text{fib}(4)$
- $= 4 + 1 + \text{fib}(4)$

Exemplos de Problemas Recursivos

Sequência de Fibonacci

- Exemplo para $n = 6$
 - $= 4 + 1 + \text{fib}(4)$
 - $= 5 + \text{fib}(3) + \text{fib}(2)$
 - $= 5 + \text{fib}(2) + \text{fib}(1) + \text{fib}(2)$
 - $= 5 + \text{fib}(1) + \text{fib}(0) + \text{fib}(1) + \text{fib}(2)$
 - $= 5 + 1 + \text{fib}(0) + \text{fib}(1) + \text{fib}(2)$
 - $= 6 + 0 + \text{fib}(1) + \text{fib}(2)$
 - $= 6 + 1 + \text{fib}(2)$
 - $7 + \text{fib}(1) + \text{fib}(0)$
 - $7 + 1 + \text{fib}(0)$
 - $8 + 0$
 - 8

Simulando a Recursividade

Simulando a Recursividade

- Todo problema com solução recursiva pode ser resolvido iterativamente
 - Para isso, deve ser examinado, detalhadamente, os mecanismos usados para implementar a recursividade para que seja possível simulá-los usando técnicas não-recursivas
- A solução recursiva é geralmente mais cara computacionalmente do que a não-recursiva
- A possibilidade de gerar uma solução não-recursiva a partir de um algoritmo recursivo é mais interessante em várias situações

Simulando a Recursividade

- Geralmente, uma versão não-recursiva (iterativa) de um programa executará com mais eficiência
- Na versão iterativa, o trabalho extra dispendido para entrar e sair de um bloco é evitado
- Em programa não-recursivo, muita atividade de empilhamento e desempilhamento pode ser evitada

Simulando a Recursividade

- Exemplo de quando não usar recursividade
 - Sequência de Fibonacci

```
int fib(int n){  
    if (n <= 0)  
        return 0;  
    else if (n == 1)  
        return 1;  
    else  
        return fib(n - 1) + fib(n - 2);  
}
```

- A versão recursiva é extremamente ineficiente, pois refaz os mesmos cálculos diversas vezes

Simulando a Recursividade

- Deve-se evitar uso de recursividade quando existe solução óbvia por iteração

```
int fibonacci(int n){
    int i, fib1, fib2, fibN;

    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else{
        fib1 = 0;
        fib2 = 1;
        fibN = 0;

        for (i = 2; i <= n; i++) {
            fibN = fib1 + fib2;
            fib1 = fib2;
            fib2 = fibN;
        }
        return fibN;
    }
}
```

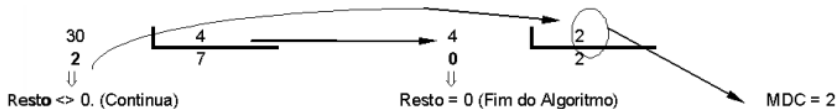
Simulando a Recursividade

- Comparação versões recursiva e iterativa

n	10	20	30	50	100
Recursiva	8 ms	1 s	2 min	21 dias	10^2
Iterativa	1/6 ms	1/3 ms	1/2 ms	3/4 ms	1,5 ms

Simulando a Recursividade

- Exercício: escreva uma função recursiva que calcule o MDC (máximo divisor comum) de 2 números a e b recebidos como parâmetros. Para o cálculo do MDC, deve-se usar o Algoritmo de Euclides. Ex: $a = 30$ e $b = 4$:





Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.



Rosa, J. L. G.
Recursão em C. SCE-181 – Introdução à Ciência da
Computação II.
Slides. Ciência de Computação. ICMC/USP, 2008.



Szwarcfiter, J.; Markenzon, L.
Estruturas de Dados e Seus Algoritmos.
LTC, 2010.



Tenenbaum, A.; Langsam, Y.
Estruturas de Dados usando C.
Pearson, 1995.



Ziviani, N.

Projeto de Algoritmos - com implementações em Java e C++.

Thomson, 2007.