Estruturas de Dados

Aula 2 – Recursividade e Estruturas de Dados Simples

Prof. Ana Carolina Sokolonski

Bacharelado em Sistemas de Informação
Instituto Federal da Bahia – Campus Feira de Santana

2026

Agenda

Recursividade

Algoritmos que se chamam a si mesmos

O que é Recursividade?

▪ Estrutura recursiva

Muitos problemas computacionais têm a propriedade de que **cada instância contém uma instância menor do mesmo problema**. Dizemos que têm **estrutura recursiva**.

▪ Método geral

1. Se a instância for **pequena** → resolva diretamente (**caso base**)
2. Senão → **reduza** a uma instância menor e aplique o mesmo método (**passo recursivo**)

▪ Para evitar loop infinito

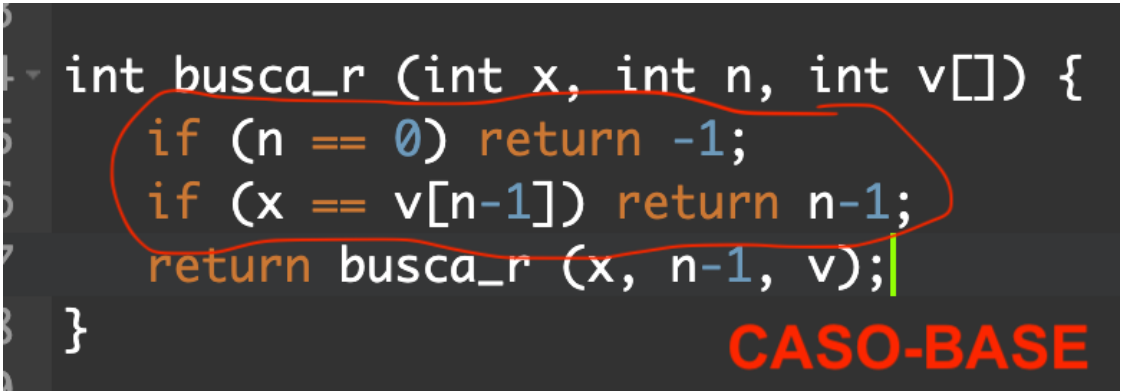
1. Resolva primeiro o **caso base** (instância mínima)
2. Só depois generalize para o **caso recursivo**

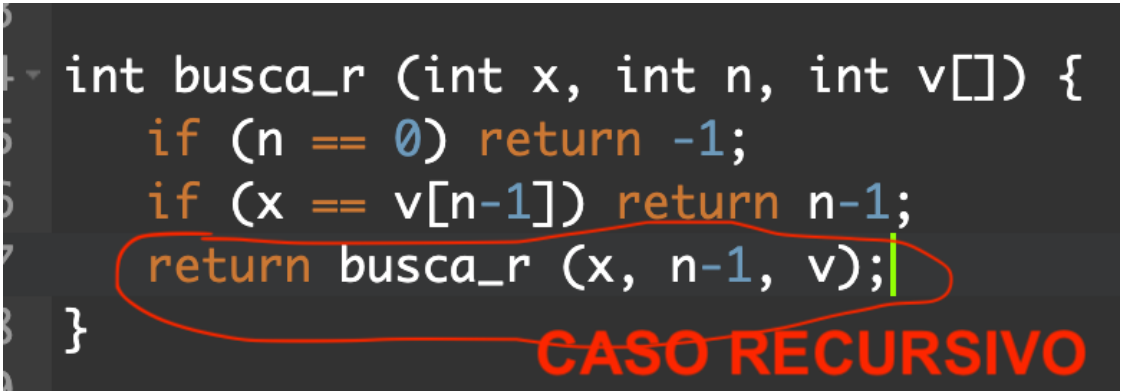
Todo algoritmo recursivo precisa de um caso base bem definido!

Ideia

O programador só precisa mostrar como obter a solução da instância original a partir da solução de uma instância menor; o computador faz o resto. Essa abordagem resulta em um **algoritmo recursivo**.

```
int busca_r (int x, int n, int v[]) {
    if (n == 0) return -1;
    if (x == v[n-1]) return n-1;
    return busca_r (x, n-1, v);
}
```


int busca_r (int x, int n, int v[]) {
if (n == 0) return -1;
if (x == v[n-1]) return n-1;
return busca_r (x, n-1, v);
}
CASO-BASE


int busca_r (int x, int n, int v[]) {
if (n == 0) return -1;
if (x == v[n-1]) return n-1;
return busca_r (x, n-1, v);
}
CASO RECURSIVO

Exercícios

1. Faça um algoritmo recursivo que encontre o **maior valor** de um vetor.
2. Faça um algoritmo recursivo que calcule o **fatorial** de um número e mostre suas parcelas. Ex: $6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$
3. A **sequência de Fibonacci** é definida assim:

   $F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2)$ para $n > 1$

   Implemente em C uma versão **recursiva** e uma **iterativa**.

Ex. 1 — Maior Valor Recursivo (parte 1)

```c
#include <stdio.h>
#include <stdlib.h>

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam] > maior)
        return v[tam];
    else
        return maior;
}
```

Ex. 1 — Maior Valor Recursivo (parte 2)

```c
#include <stdio.h>
#include <stdlib.h>

int maior_r (int tam, int v[]){
    int maior;
    if (tam == 0) return -1;
    if (tam == 1) return v[0];
    maior = maior_r(tam-1,v);
    if (v[tam-1] > maior)
        return v[tam-1];
    else
        return maior;
}

int main (void){
int v1[]={134,3,234,7,567,5,678,2,899,0};
int v[100];
int tam, d=0;

for(tam=0;d!=-1;tam++){
    printf("Digite os elementos do vetor, digite -1 para sair\n");
    scanf("%d",&d);
    if(d!=-1)
     v[tam] = d;
}

  int i = maior_r(tam,v);
  printf("Maior item do Vetor= %d\n", i);
}
```

Ex. 2 — Fatorial Recursivo

```c
#include <stdio.h>
#include <stdlib.h>

int fat (int n){
    if (n == 1){
        printf("1 = ");
        return 1;
    }
    else{
        printf("%d*",n);
        return (n * fat(n-1));
    }
}

int main (void){
int n;

  printf("Informe o número que você deseja saber o fatorial:");
  scanf("%d",&n);
  printf("%d! = ",n);
  printf("%d\n", fat(n));
}
```

Ex. 3 — Fibonacci Iterativo

```c
void fibonacci (int n){
    int fib, fib1, fib2, i;

    if (n == 0) {
        printf("0\n");
    } else if (n == 1){
        printf("0, 1\n");
    } else{
        printf("0, 1, ");
        fib1 = 1;
        fib2 = 0;
        i = 2;
        while (i <= n){
            fib = fib1+fib2;
            if (i==n) printf("%d\n", fib);
            else printf("%d, ", fib);
            fib2 = fib1;
            fib1 = fib;
            i++;
        }
    }

}
```

Ex. 3 — Fibonacci Recursivo

```c
/**A função de Fibonacci é definida assim:   F(0) = 0,   F(1) = 1   e   F(n) = F(n-1) + F(n-2)
para n > 1. Descreva a função F em linguagem C. Faça uma versão recursiva e uma iterativa.**/
int fibonacci (int n){
    if (n == 0) return 0;
    else if (n == 1) return 1;
    else return fibonacci(n-1) + fibonacci(n-2);
}
int main (void){
int n, fib, i;

  printf("Informe o número de fatores das sequências de Fibonacci:");
  scanf("%d",&n);
  for (i = 0; i <= n; i++)
     printf("Parcela %d da sequência = %d\n",i,fibonacci(i));
}
```

Remoção em Vetores

Eliminando elementos

Remoção: Conceito

▪ Problema

Remover o elemento de índice k do vetor $v[0\,..\,(n-1)]$, deslocando os elementos $v[(k+1)\,..\,(n-1)]$ para as posições $[k\,..\,(n-2)]$. O problema é válido se, e somente se, $0 \leq k < n$.

▪ Exemplo

Vetor original: $\{0,\ 11,\ 22,\ \mathbf{33},\ 44,\ 55\}$ (remover índice 3)

Resultado: $\{0,\ 11,\ 22,\ 44,\ 55\}$

▪ Custo

A remoção consome **mais tempo** quanto **menor** for o índice k — mais elementos precisam ser deslocados.

Remoção: Versão Iterativa

```
int remover(int k, int n, int v[]){
    int x = v[k];
    for (int j = k+1; j < n; ++j)
        v[j-1] = v[j];
    return x;
}
```

Remoção: Como usar e Versão Recursiva

Como usar a função

```
// Remove índice 51 de v[0..n-1]
x = remover(51, n, v);
n -= 1; // atualiza n
```

Versão Recursiva

O tamanho da instância é medido por $(n-k)$. A instância é pequena quando $(n-k) = 1$, ou seja, $k = (n-1)$. Esse é o **caso base**. [Cormen et al. 2009]

```
int remover_r (int k, int n, int v[]) {
    int x = v[k];
    if (k < n-1) {
        int y = remover_r (k+1, n, v);
        v[k] = y;
    } return x;
}
```

Inserção em Vetores

Adicionando elementos

Inserção: Conceito e Cuidados

Problema

Dado $v[0\,..\,(n-1)]$, inserir x entre os índices $(k-1)$ e k. Válido para qualquer $k \in [0\,..\,n]$: $k = 0$ insere no início; $k = n$ insere no fim.

Cuidado com overflow

Só insira se $(n+1) \leq N$ (vetor não está cheio). Caso contrário, ocorrerá **transbordamento** (*overflow*).

Como usar a função

```
// Insere 999 entre posições 50 e 51
inserir(51, 999, n, v);
n++; // atualiza n
```

Inserção: Versão Iterativa

```
// Esta função insere x entre as posições k-1 e k do vetor v[0..n-1]
// supondo que 0 <= k <= n.
void inserir (int k, int x, int n, int v[]){
   for (int j = n; j > k; --j)
      v[j] = v[j-1];
   v[k] = x;
}
```

Inserção: Versão Recursiva

```
4  // Esta função insere x entre as posições k-1 e k do vetor v[0..n-1]
5  // supondo que 0 <= k <= n.
6  void inserir_r (int k, int x, int n, int v[]) {
7     if (k == n)  v[n] = x;
8     else {
9        v[n] = v[n-1];
10       inserir_r (k, x, n - 1, v);
11    }
12 }
```

Busca e Remoção Combinadas

Removendo todos os elementos iguais a x

Busca e Remoção: Conceito

Problema

Remover **todas** as ocorrências do valor 2 em $v[0\,..\,(n-1)]$. A função deve devolver o número de elementos após a remoção.

Exemplo

Entrada: $\{11,\ \mathbf{2},\ 22,\ 88,\ \mathbf{2},\ 66,\ 33\}$ $\quad (n = 7)$

Saída: $\{11,\ 22,\ 88,\ 66,\ 33\}$ $\quad (n = 5)$

Busca e Remoção: Versão Iterativa

```
// Esta função elimina todos os elementos 2 de v[0..n-1].
// Supõe apenas que n >= 0. A função deixa o resultado em
// v[0..i-1]  e devolve i.
int buscaRetira2 (int n, int v[]) {
   int i = 0;
   for (int j = 0; j < n; ++j)
      if (v[j] != 2)
         v[i++] = v[j];
   return i;
}
```

▪ Detalhe importante

A instrução `v[i++] = v[j]` equivale a `v[i] = v[j]; ++i;`. Em cada iteração: $v[0\,..\,(i-1)]$ é o resultado parcial da remoção de $v[0\,..\,(j-1)]$, com $i \leq j$.

Busca e Remoção: Versão Recursiva

```
int buscaRetira2_r (int n, int v[]) {
    if (n == 0) return 0;
    int m = buscaRetira2_r (n - 1, v);
    if (v[n-1] == 2) return m;
    v[m] = v[n-1];
    return m + 1;
}
```

▪ Observação

A instrução `v[m] = v[n-1]` coloca `v[n-1]` em sua posição definitiva. A função elimina todos os 2s de $v[0..(n-1)]$, deixando o resultado em $v[0..(i-1)]$ e devolvendo i.

Referências I

CORMEN, T. H. et al. *Introduction to Algorithms*. 2nd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262032937.