

## MC102

### Algoritmos e Programação de Computadores

Aula de Laboratório 14  
 Instituto de Computação  
 Primeiro Semestre de 2012

11 de junho de 2012



### Recursão

É o processo de resolução de um problema, reduzindo-o em um ou mais subproblemas com as seguintes características:

- 1 São idênticos aos problemas originais;
  - 2 São mais simples de resolver.
- A idéia é que a solução de um problema pode ser expressa da seguinte forma:
    - 1 Definimos a solução para os casos básicos;
    - 2 Definimos como resolver o problema geral utilizando soluções do mesmo problema só que para casos menores.



## Conteúdo

## Cálculo de Potências

Problema: Calcula  $x^n$  para um  $n$  positivo de forma recursiva.

Ex.:  $2^4$

- $2^4 = 2 * 2^3 \Leftarrow$  subdividimos o problema;

### 1 Recursividade



## Cálculo de Potências

Problema: Calcula  $x^n$  para um  $n$  positivo de forma recursiva.

Ex.:  $2^4$

- $2^4 = 2 * 2^3 \Leftarrow$  subdividimos o problema;
- $2^3 = 2 * 2^2 \Leftarrow$  subdividimos o problema;



## Cálculo de Potências

Problema: Calcula  $x^n$  para um  $n$  positivo de forma recursiva.

Ex.:  $2^4$

- $2^4 = 2 * 2^3 \Leftarrow$  subdividimos o problema;
- $2^3 = 2 * 2^2 \Leftarrow$  subdividimos o problema;
- $2^2 = 2 * 2^1 \Leftarrow$  subdividimos o problema;



## Cálculo de Potências

Problema: Calcula  $x^n$  para um  $n$  positivo de forma recursiva.

Ex.:  $2^4$

- $2^4 = 2 * 2^3 \Leftarrow$  subdividimos o problema;
- $2^3 = 2 * 2^2 \Leftarrow$  subdividimos o problema;
- $2^2 = 2 * 2^1 \Leftarrow$  subdividimos o problema;
- $2^1 = 2 * 2^0 \Leftarrow$  Opa!! Temos a solução;
- Chegamos a menor parte do problema!  
Esta é a **condição de parada!**



## Cálculo de Potências

### Solução

Faça o programa para calcular a potência  $x^n$  de forma recursiva.



## Cálculo de Potências

## Solução

Faça o programa para calcular a potência  $x^n$  de forma recursiva.

## Solução

```

1 #include <stdio.h>
2 long pot(long x, long n){
3     if(n == 0)
4         return 1;
5     else{
6         return x*pot(x,n-1);
7     }
8 }
9
10 int main(){
11     long x = 2;
12     long n = 5;
13     printf("A potencia %ld^%ld: %ld\n",x,n, pot(x,n));
14 }

```



## Cálculo de Potências

Vamos pensar a potência  $x^n$ , para um  $n$  positivo de forma genérica. Como fazer de forma recursiva?

$x^n$  é:

- 1 se  $n = 0$ ;



## Cálculo de Potências

Vamos pensar a potência  $x^n$ , para um  $n$  positivo de forma genérica. Como fazer de forma recursiva?

$x^n$  é:

- 1 se  $n = 0$ ;
- $xx^{n-1}$  caso contrário;



## Cálculo de Potências

```
long pot(long x, long n){
```

```
}
```



## Cálculo de Potências

```
long pot(long x, long n){
    if(n == 0)

}
}
```



## Cálculo de Potências

```
long pot(long x, long n){
    if(n == 0)
        return 1;
    else

}
}
```



## Cálculo de Potências

```
long pot(long x, long n){
    if(n == 0)
        return 1;

}
}
```



## Cálculo de Potências

```
long pot(long x, long n){
    if(n == 0)
        return 1;
    else
        return x*pot(x, n-1);

}
}
```



## Cálculo de Potências

Neste caso a solução iterativa é mais eficiente.

```
long pot(long x, long n) {
    long p = 1, i;
    for( i=1; i<=n; i++)
        p = p * x;
    return p;
}
```

- O laço é executado  $n$  vezes.
- Na solução recursiva são feitas  $n$  chamadas, mas tem-se o custo adicional para criação/remoção de variáveis locais na pilha.



## Soma de um vetor

- Dado um vetor, vamos definir  $S(i, n)$  como a soma de  $n$  elementos a partir da posição  $i$ .
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - 1 Se  $n = 1$  então  $S(i, n) = v[i]$ .



## Soma de um vetor

- Dado um vetor, vamos definir  $S(i, n)$  como a soma de  $n$  elementos a partir da posição  $i$ .
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - 1 Se  $n = 1$  então  $S(i, n) = v[i]$ .
  - 2 Se  $n > 1$  então  $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$ .



## Soma de um vetor

- Dado um vetor, vamos definir  $S(i, n)$  como a soma de  $n$  elementos a partir da posição  $i$ .
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - 1 Se  $n = 1$  então  $S(i, n) = v[i]$ .
  - 2 Se  $n > 1$  então  $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$ .
- Para computarmos a soma de todos os elementos de um vetor com  $n$  elementos, devemos calcular  $S(0, n)$ .



## Soma de um vetor

O código recursivo segue abaixo.

## Soma

```

1 #include <stdio.h>
2 // funca teto e chao aqui.
3
4 int soma(int v[], int i, int n){
5     if(n == 1)
6         return v[i];
7     else
8         return soma(v,i,teto(n,2)) +
9             soma(v,i+teto(n,2),chao(n,2));
10 }
11
12 int main(){
13     int vet[10] = {2,2,5,1,8};
14     printf("\n Soma: %d:\n",soma(vet,0,5));
15 }

```



## Busca Binária

Podemos definir recursivamente da seguinte forma:

- 1 Se  $n = 0$  então  $Busca(ini, fim, ch) = -1$ .
- 2 Se  $n = 1$  e se  $ch == vet[meio]$ , então  $Busca(ini, fim, ch) = vet[meio]$ .



## Soma de um vetor

## Soma

```

1 int teto(int numerador, int denominador){
2     if(numerador%denominador == 0) //se divisao for
3         inteira
4         return (numerador/denominador);
5     else
6         return (numerador/denominador + 1);
7 }
8 int chao(int numerador, int denominador){
9     return (numerador/denominador);
10 }

```



## Busca Binária

Podemos definir recursivamente da seguinte forma:

- 1 Se  $n = 0$  então  $Busca(ini, fim, ch) = -1$ .
- 2 Se  $n = 1$  e se  $ch == vet[meio]$ , então  $Busca(ini, fim, ch) = vet[meio]$ .
- 3 Se  $n > 1$  e  $vet[meio] > ch$ , então  $Busca(ini, fim, ch) = Busca(ini, meio - 1, ch)$ .
- 4 Se  $n > 1$  e  $vet[meio] < ch$ , então  $Busca(ini, fim, ch) = Busca(meio + 1, fim, ch)$ .



## Busca Binária

```
1 #include<stdio.h>
2 int busca(int vet[], int ini, int fim, int ch){
3     int meio, res;
4     if(fim < ini) return -1;
5     meio = (ini+fim)/2;
6     if(vet[meio] == ch){
7         return meio;
8     }else{
9         if(vet[meio] > ch)
10            res = busca(vet, ini, meio-1, ch);
11        else res = busca(vet, meio+1, fim, ch);
12    }
13    return res;
14 }
15 int main(){
16     int pos, vet[]={2,4,8,12,22,35,41,82};
17     pos = busca(vet, 0, 8, 22);
18     if (pos == -1) printf("\nChave nao existe.\n");
19     else printf("\nChave na posicao: %d\n",pos);
20 }
```



## Questões?

Obrigado!

Para informação:

Página dos Laboratórios (Tarefas): <http://susy.ic.unicamp.br:9999/mc102ab>

Página do Curso: <http://www.lrc.ic.unicamp.br/~geraldoms/mc102>

E-mail:

[geraldoms@lrc.ic.unicamp.br](mailto:geraldoms@lrc.ic.unicamp.br)

[brhenrique.fischer@gmail.com](mailto:brhenrique.fischer@gmail.com)

