Aula 19

MC 102 - Algoritmos e Programação de Computadores

Recursão II

Mais sobre Recursividade

Alguns problemas podem parecer não serem recursivos. Porém, pode ser possível encontrar uma relação de recorrência para uma solução recursiva.

Exemplo:

Encontrar o Máximo Divisor Comum entre dois valores inteiros pode ser resolvido recursivamente?

MDC Iterativo

Máximo Divisor Comum de A e B

```
int mdc(int a, int b) {
  int r;

  while (b != 0) {
    r = a % b;
    a = b;
    b = r;
  }
  return a;
}
```


MDC Recursivo

Máximo Divisor Comum de A e B

- **Caso base??**

MDC Recursivo

Máximo Divisor Comum de A e B

- **Caso base**
  - MDC de A e B é A se B = 0.

MDC Recursivo

Máximo Divisor Comum de A e B

- **Caso base**
  - MDC de A e B é A se B = 0.
- **Chamadas Recursivas ?**

MDC Recursivo

Máximo Divisor Comum de A e B

- **Caso base**
  - MDC de A e B é A se B = 0.
- **Chamadas Recursivas**
  - MDC de A e B é MDC(B, resto da divisão de A por B)

MDC Recursivo

Máximo Divisor Comum de A e B

```
int mdc(int a, int b) {
  if (b == 0) {
    return a;
  else
    return mdc(b, a % b);
}
```

Caso base, condição em que facilmente se resolve o problema.

MDC Recursivo

Máximo Divisor Comum de A e B

```
int mdc(int a, int b) {
  if (b == 0) {
    return a;
  else
    return mdc(b, a % b);
}
```

Caso base, condição em que facilmente se resolve o problema.

O Quadrado Perfeito

Um número é dito quadrado perfeito se puder ser escrito como o quadrado de um número natural.

$1 = 1^2$

$4 = 2^2$

$9 = 3^2$

$16 = 4^2$

$25 = 5^2$

Mas dá para usar recorrência com isso?

O Quadrado Perfeito

Podemos reescrever estes números:

$Q(1) = 1 = 1^2$

$Q(2) = Q(1) + 3 = 1 + 3 = 2^2$

$Q(3) = Q(2) + 5 = 1 + 3 + 5 = 3^2$

$Q(4) = Q(3) + 7 = 1 + 3 + 5 + 7 = 4^2$

$Q(5) = Q(4) + 9 = 1 + 3 + 5 + 7 + 9 = 5^2$

...

$Q(n) = Q(n-1) + (2n-1) = 1 + 3 + 5 + ... + (2n-1) = n^2$

Qual a relação de recorrência para Q(n)?

O Quadrado Perfeito

- **Caso Base**
  - O quadrado perfeito de 0 é 0
- **Chamadas Recursivas**
  - O quadrado perfeito $Q(n) = Q(n-1) + 2n - 1$

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

População de Coelhos

Dados um casal de coelhos:

- No primeiro mês, nasce apenas 1 casal
- Os novos casais só podem se reproduzir após o segundo mês de vida
- Não há problemas genéticos no cruzamento consangüíneo
- Todo mês, cada casal fértil dá a luz a um novo casal
- Os coelhos nunca morrem

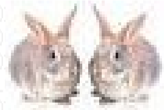
Quantos pares de coelhos haverá após n meses?

População de Coelhos

A solução é o **Número de Fibonacci**, da forma:

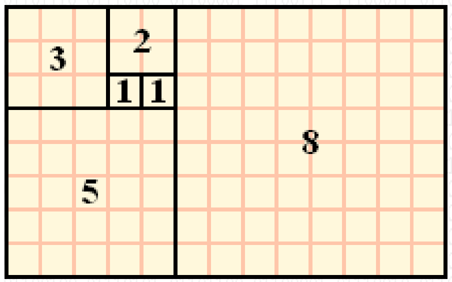
- **Caso Base**
  - F(0) = 1 e F(1) = 1
- **Chamadas Recursivas**
  - **Número de Fibonacci** F(n) = F(n - 1) + F(n - 2)

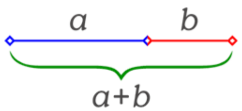
```
int fibonacci(int n) {
   if (n <= 1)
      return 1;
   else
      return (fibonacci(n-1) + fibonacci(n-2));
}
```



Números de Fibonacci

Uma grade preenchida com quadrados cujos lados são **números de Fibonacci**, formando sucessivamente retângulos cada vez maiores e que tendem à **razão áurea**





$$\frac{(a+b)}{a}=\frac{a}{b}=1.618033989$$

Problema 1 - Razão Áurea

A razão áurea pode ser aproximada pelas seguintes funções:

$$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\ldots}}}} \qquad 1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1}\cdots}}}$$

Desenvolva uma função recursiva baseada na série de frações (à direita) que receba um valor de partida (neste caso 1) e calcule a série até que a diferença entre o valor de partida e o cálculo atual seja menor que 10^{-6}.

Busca Binária Recursiva

Dado um **vetor ordenado**, como usar a busca binária de forma recursiva?

A definição de busca binária já é recursiva naturalmente, pois basea-se na técnica de divisão e conquista da forma:

1. Se não existir elemento a ser pesquisado (inicio > fim), não é possível encontrar o elemento (Caso Base 1).
2. Caso contrário, esolhe-se o elemento mediado (pivô) e verifica se é o valor procurado. Se for, retora sua posição (Caso Base 2).
3. Senão, busca-se no subvetor superior ao pivô, caso o valor procurado seja maior que o pivô, ou no subvetor inferior ao pivô, se o valor procurado for inferior (Chamadas Recursivas).

Busca Binária Recursiva

BuscaBinRec(vetor, ini, fim, x)
Entrada:
Vetor Ordenado, ini (**0**), fim (**14**) e x (valor procurado **33**)
Algoritmo
Se (ini > fim)
retorna -1
meio = (fim + ini) / 2
Se (vetor[meio] = x)
retorna meio
Senão, se (x < vetor[meio])
retorna BuscaBinRec(vetor, ini, meio-1, x)
Senão
retorna BuscaBinRec(vetor, meio+1, fim, x)

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ ini														↑ fim

Busca Binária Recursiva

BuscaBinRec(vetor, 0, 14, 33)
Entrada:
Vetor Ordenado, ini (**0**), fim (**14**) e x (valor procurado **33**)
Algoritmo
Se (ini > fim)
retorna -1
meio = (fim + ini) / 2
Se (vetor[meio] = x)
retorna meio
Senão, se (x < vetor[meio])
retorna BuscaBinRec(vetor, ini, meio-1, x)
Senão
retorna BuscaBinRec(vetor, meio+1, fim, x)

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ ini							↑ meio							↑ fim

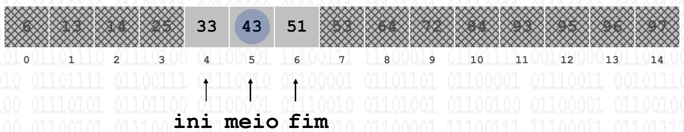
Busca Binária Recursiva

BuscaBinRec(vetor, 0, 6, 33)
Entrada:
Vetor Ordenado, ini (**0**), fim (**6**) e x (valor procurado **33**)
Algoritmo
Se (ini > fim)
retorna -1
meio = (fim + ini) / 2
Se (vetor[meio] = x)
retorna meio
Senão, se (x < vetor[meio])
retorna BuscaBinRec(vetor, ini, meio-1, x)
Senão
retorna BuscaBinRec(vetor, meio+1, fim, x)

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑ ini			↑ meio			↑ fim								

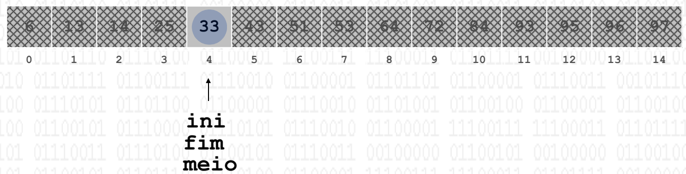
Busca Binária Recursiva

BuscaBinRec(vetor, 4, 6, 33)
Entrada:
Vetor Ordenado, ini (**4**), fim (**6**) e x (valor procurado **33**)
Algoritmo
Se (ini > fim)
retorna -1
meio = (fim + ini) / 2
Se (vetor[meio] = x)
retorna meio
Senão, se (x < vetor[meio])
retorna BuscaBinRec(vetor, ini, meio-1, x)
Senão
retorna BuscaBinRec(vetor, meio+1, fim, x)



Busca Binária Recursiva

BuscaBinRec(vetor, 4, 4, 33)
Entrada:
Vetor Ordenado, ini (**4**), fim (**4**) e x (valor procurado **33**)
Algoritmo
Se (ini > fim)
retorna -1
meio = (fim + ini) / 2
Se (vetor[meio] = x)
retorna meio
Senão, se (x < vetor[meio])
retorna BuscaBinRec(vetor, ini, meio-1, x)
Senão
retorna BuscaBinRec(vetor, meio+1, fim, x)

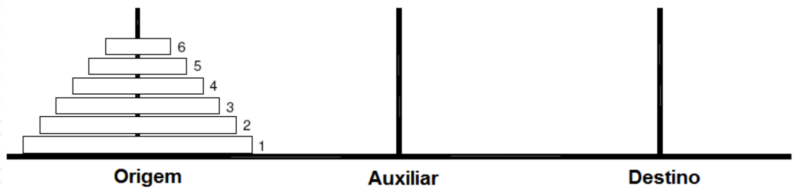


Busca Binária Recursiva

```
int BinSRec(int vetor[], int i, int f, int x)
 {
 int meio;

 if (i > f)
   return -1;

 meio = (f + i)/2;
 if (vetor[meio]==x)
     return meio;

 if (x < vetor[meio])
     BinSRec(vetor, i, meio - 1, x);
 else
     BinSRec(vetor, meio + 1, f, x);
}
```

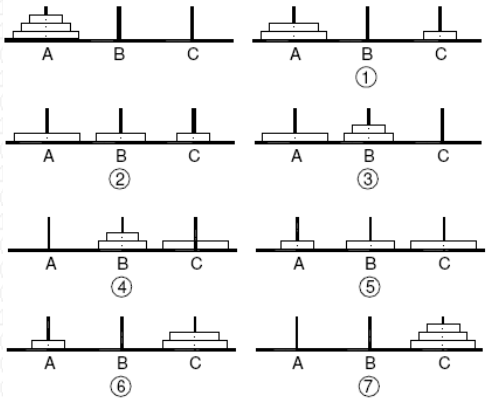

Torre de Hanoi

1883, o matemático francês Édouard Lucas publicou este quebra-cabeça que consiste em mover N discos da torre de origem para a Destino usando uma torre auxiliar. Discos menores não podem ficar embaixo de discos maiores. Como mover com o menor número de movimentos se somente o disco superior pode ser movido por vez?



Torre de Hanoi

Vamos analisar como resolver o problema usando apenas 3 discos, já que para 1 e 2 discos o problema torna-se trivial. Considere A a torre de Origem, B a Auxiliar e C a de Destino.



Foram necessários 7 movimentos para a solução do problema.

Torre de Hanoi

Recursividade:

Hanoi(n, **Origem**, Auxiliar, **Destino**)

Se n = 1

  Move Disco 1 da Origem para Destino

Senão

  Hanoi(n-1, **Origem**, Destino, **Auxiliar**)

  Move Disco n da Origem para Destino

  Hanoi(n-1, **Auxiliar**, Origem, **Destino**)

Torre de Hanoi

```
void move(unsigned int n, char A, char B) {
  printf("Mova o disco %2d de %c para %c\n", n, A, B);
}


void hanoi(unsigned int n, char A, char B, char C) {
   /* A: Origem, B: Auxiliar, C: Destino */
   if (n == 1)
     move(n, A, C);
   else {
     hanoi(n-1, A, C, B);
     move(n, A, C);
     hanoi(n-1, B, A, C);
   }
 }
```


Exercício 1

Escreva a função recursiva **comb(n,k)** que representa o número de grupos distintos com k pessoas que podem ser formados a partir de n pessoas:

$$Comb(n,k) = \begin{cases} n & \text{se} \quad k = 1 \\ 1 & \text{se} \quad k = n \\ Comb(n-1, k-1) + Comb(n-1, k) & \text{se} \quad 1 < k < n \end{cases}$$

Exercício 2

Faça uma função recursiva para calcular a função de Ackerman para dois inteiros positivos m e n, conforme definição abaixo:

$$A(m,n) = \begin{cases} n+1 & \text{se } m = 0 \\ A(m-1, 1) & \text{se } m > 0 \text{ e } n = 0 \\ A(m-1, A(m, n-1)) & \text{se } m > 0 \text{ e } n > 0. \end{cases}$$

Exercícios 1 e 2

```
unsigned int Comb(unsigned int n, unsigned int k) {
  if (k==1)
    return n;
  else if (k==n)
    return 1;
  else
    return Comb(n-1,k-1) + Comb(n-1,k);
}

unsigned int Ack(unsigned int m, unsigned int n) {
  if (m==0)
    return n+1;
  else if (n == 0) /* m > 0 por ser usigned */
    return Ack(m-1, 1);
  else
    return Ack(m-1, Ack(m, n-1));
}
```