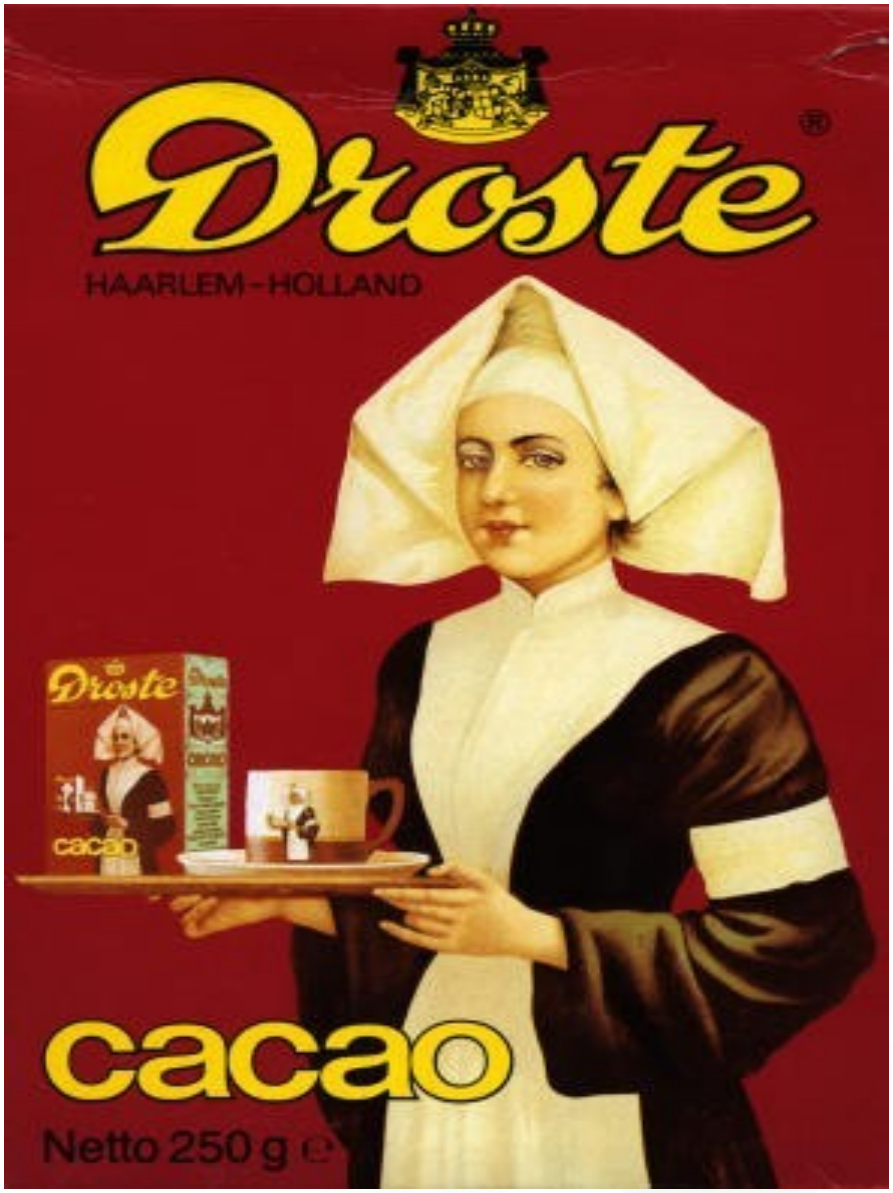
BC1424
Algoritmos e Estruturas de Dados I

Aula 02:
Recursão/Recursividade

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015



Anuncio de cacao com uma imagem recursiva.

“To understand recursion, we must first understand recursion”
– folclore

“Ao tentar resolver o problema, encontrei obstáculos dentro de obstáculos.
Por isso, adotei uma solução recursiva”
– um aluno

Recursão

- O conceito de recursão é de fundamental importância em computação!

- Muitos problemas computacionais têm a seguinte propriedade:

  Cada instância do problema contém uma instância menor do mesmo problema.

  → Dizemos que esses problemas têm ***estrutura recursiva***.

(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

Recursão

Para resolver um tal problema é natural aplicar o seguinte método:

- Se a instância em questão é pequena:
  → Resolva-a diretamente
  (use força bruta se necessário)
- Senão
  → Reduza-a a uma instância menor do mesmo problema
  → Aplique o método à instância menor
  e volte à instância original.

A aplicação do método produz um algoritmo recursivo.

(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

Exercício 00: Fatorial

Considere a função fatorial:

F(n) = n!

para um número inteiro não-negativo arbitrário

- Como n! = n*(n-1)!, para n>=1, e
- 0! = 1

F(n) = F(n-1)*n

Exercício 00: Fatorial

```c
#include <stdio.h>

int F(int n) {
    if (n==0)
        return 1;
    else
        return F(n-1)*n;

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", F(num));

    return 0;
}
```

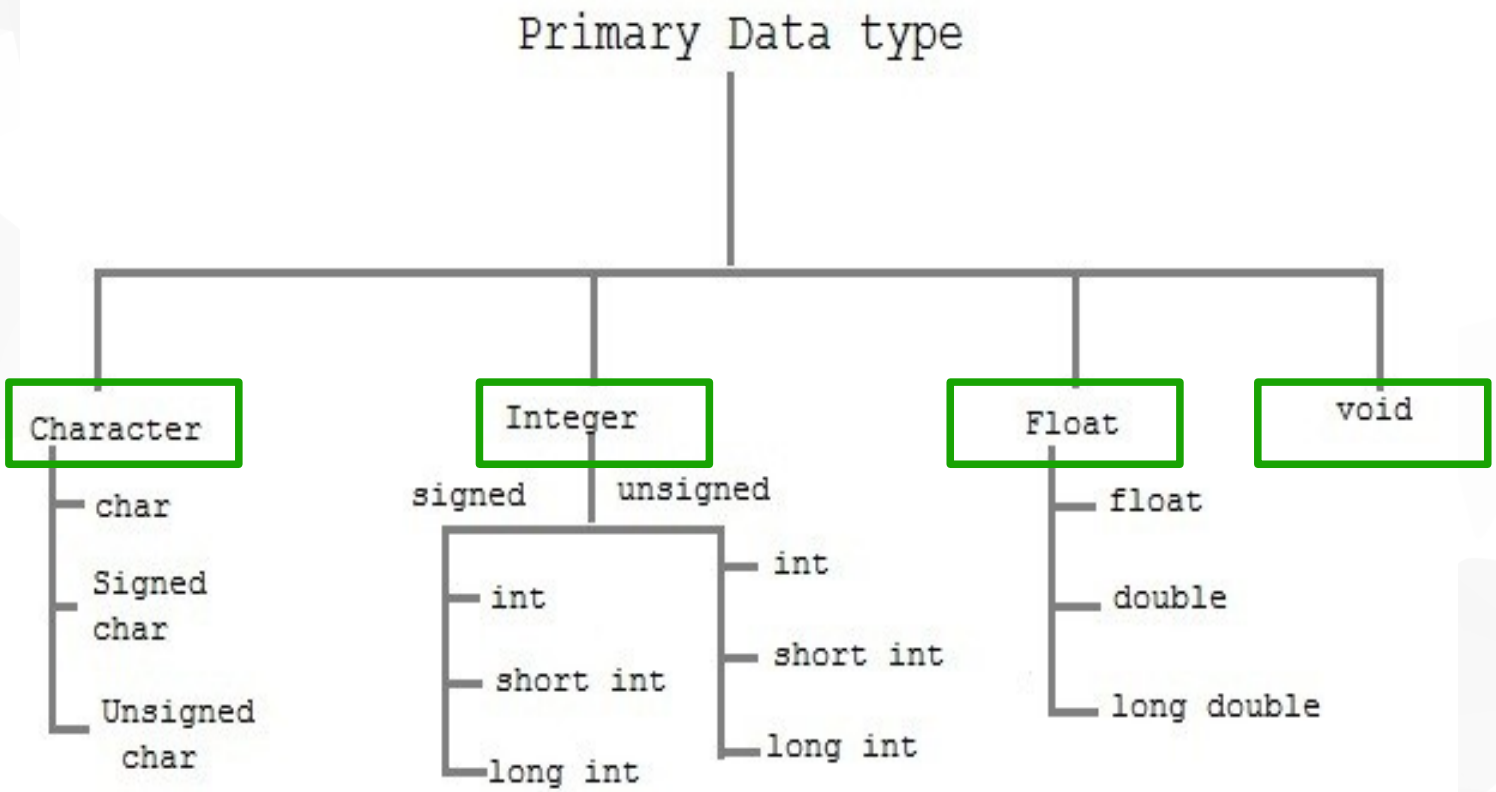
```
jmenac@aed1:~/workspace/aed1-02 $ gcc fatorial.c -o fatorial
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
6
720
```

Exercício 00: Fatorial

```
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
16
2004189184
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
17
-288522240
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
18
-898433024
```

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

Linguagem C: Tipos de dados



(*) Fonte: http://www.studytonight.com/c/datatype-in-c.php

Exercício 00: Fatorial

```c
#include <stdio.h>

long int F(int n) {
    if (n==0)
        return 1;
    else
        return F(n-1)*n;

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%ld\n", F(num));

    return 0;
}
```

Número de vezes em que a função **F** é chamada? **n**

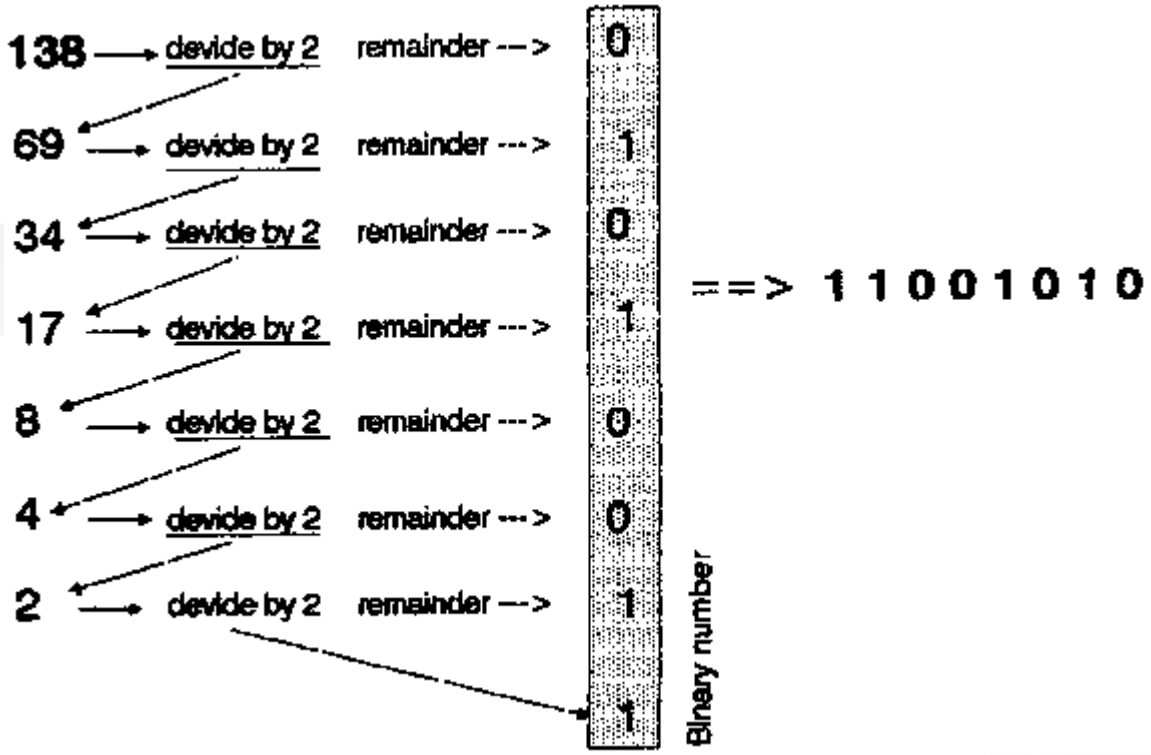
Exercício 00: Fatorial

```
jmenac@aed1:~/workspace/aed1-02 $ gcc fatorial.c -o fatorial
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
16
20922789888000
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
17
355687428096000
jmenac@aed1:~/workspace/aed1-02 $ ./fatorial
18
6402373705728000
```

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

Exercício 01: Número de digitos binários

Algoritmo que calcula o número de digitos binários de um número **n** (inteiro decimal positivo)



Exercício 01: Número de digitos binários

```c
#include <stdio.h>

int bin(int n) {
    if (n==1)
        return 1;
    else
        return bin(n/2) + 1;

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", bin(num));

    return 0;
}
```

Número de vezes em que a função **bin** é chamada?

Exercício 01: Número de digitos binários

```c
#include <stdio.h>

int bin(int n) {
    if (n==1)
        return 1;
    else
        return bin(n/2) + 1;

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", bin(num));

    return 0;
}
```

Número de vezes em que a função **bin** é chamada? **floor(log2(n))**

Exercício 02: Função Q

```c
#include <stdio.h>

int Q(int n) {
    if (n==1)
        return 1;
    else
        return Q(n-1) + 2*n - 1;

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", Q(num));

    return 0;
}
```

Qual o valor que a função Q calcula? → **resolva a recorrência**

Exercício 02: Função Q

```c
#include <stdio.h>

int Q(int n) {
    if (n==1)
        return 1;
    else
        return Q(n-1) + 2*n - 1;

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", Q(num));

    return 0;
}
```

Qual o valor que a função Q calcula? → **resolva a recorrência = n^2**

Exercício 03: Função M

```c
#include <stdio.h>

int M(int v[], int n) {
    if (n==1)
        return v[0];
    else {
        int x;
        x = M(v, n-1);
        if (x <= v[n-1])
            return x;
        else
            return v[n-1];
    }

}

int main() {
    int v[] = {1,2,3,4,5,6,7,-5,100};
    printf("%d\n", M(v,9));
    return 0;
}
```

Qual o valor que a função **M** calcula?

Qual o número total de comparações?

Escreva uma versão iterativa da função M.

Exercício 03: Função M

```c
int MIter(int v[], int n) {
    int i;
    int min = v[0];
    for (i=1; i<n; i++){
        if (min>v[i])
            min = v[i];
    }
    return min;
}
```

Versão iterativa

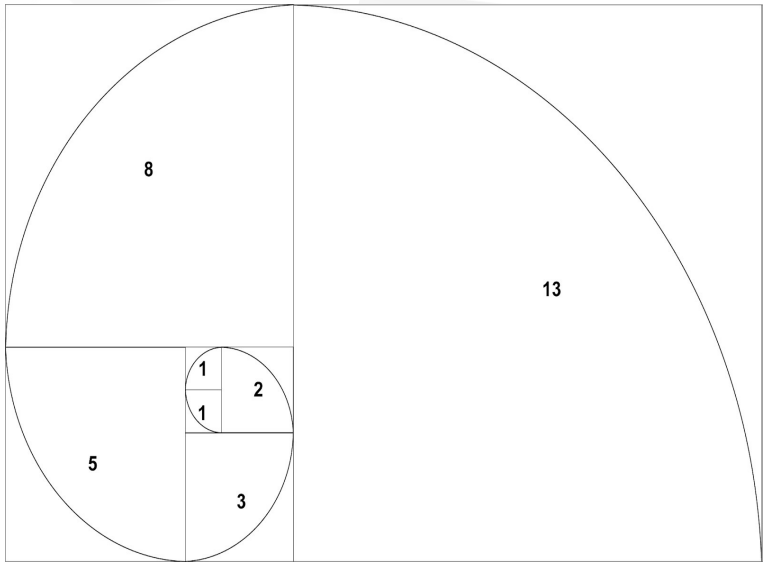
Exercício 04: Função M2

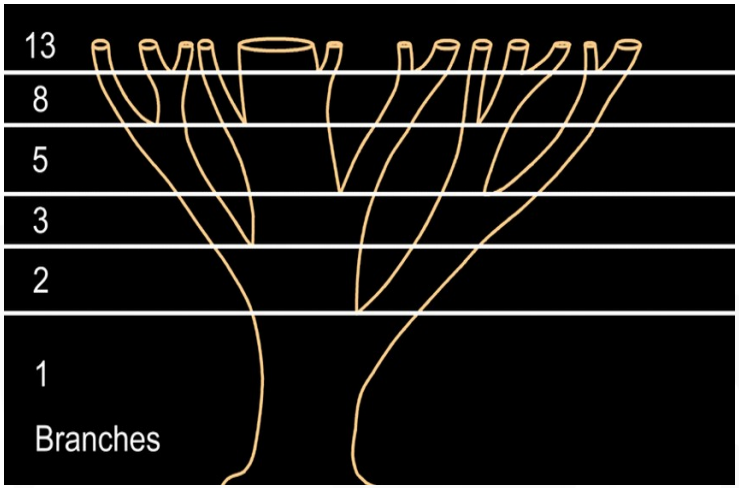
- Considere outro algoritmo para resolver o problema anterior (identificar um “menor elemento” em um vetor).

```
int M2(int v[], int l, int r) {
    if (l==r)
        return v[l];
    else {
        int x1, x2;
        x1 = M2(v, l, (l+r)/2);
        x2 = M2(v, (l+r)/2+1, r);

        if (x1 <= x2)
            return x1;
        else
            return x2;
    }

}
```

Fibonacci

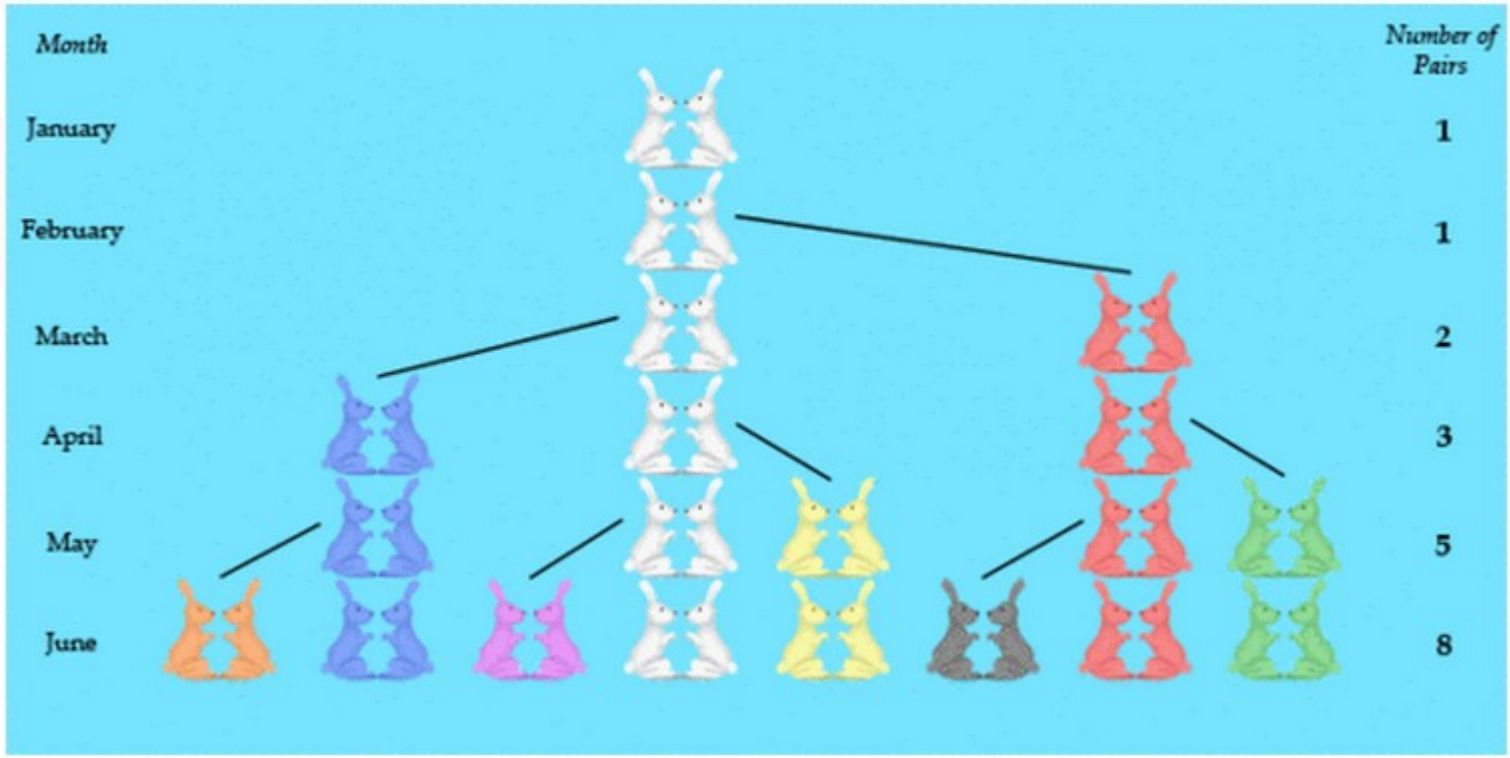
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765





(*) fonte http://britton.disted.camosun.bc.ca/fibslide/jbfibslide.htm

Os números de Fibonacci foram propostos por Leonardo di Pisa (Fibonacci), em 1202, como uma solução para o problema de determinaro tamanho da população de coelhos



(*) fonte http://www.oxfordmathcenter.com/drupal7/node/487

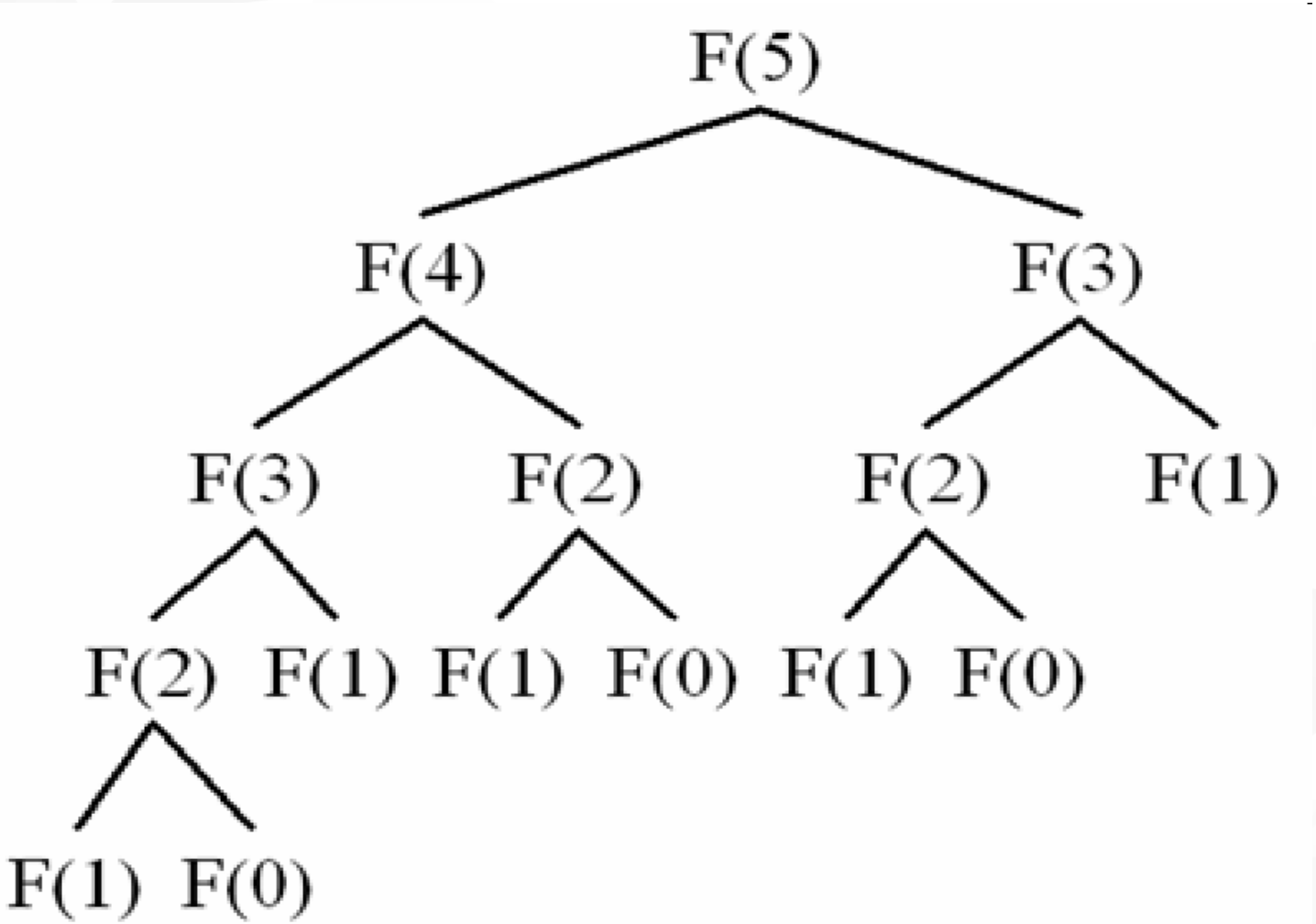
Exercício 05: Fibonacci

```c
#include <stdio.h>

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

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", Fib(num));

    return 0;
}
```

Exercício 05: Fibonacci

```
5
Fibonacci: 5
        Fibonacci: 4
                Fibonacci: 3
                        Fibonacci: 2
                                Fibonacci: 1
                                Fibonacci: 0
                        Fibonacci: 1
                Fibonacci: 2
                        Fibonacci: 1
                        Fibonacci: 0
        Fibonacci: 3
                Fibonacci: 2
                        Fibonacci: 1
                        Fibonacci: 0
                Fibonacci: 1
5
```

```c
#include <stdio.h>

int Fib(int n, int c) {
    int i;
    for (i=0; i<c; i++)
        printf("\t");
    printf("Fibonacci: %d \n", n);

    if (n<=1)
        return n;
    else {
        return Fib(n-1, c+1)+Fib(n-2, c+1);
    }

}

int main() {
    int num;

    scanf("%d", &num);
    printf("%d\n", Fib(num, 0));

    return 0;
}
```


F(5)
F(4)
F(3)
F(3)
F(2)
F(2)
F(1)
F(2)
F(1)
F(1)
F(0)
F(1)
F(0)
F(1)
F(0)

Recursão

- Uma função recursiva é aquela que se chama a si mesma (obrigatoriamente)?

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

}
```

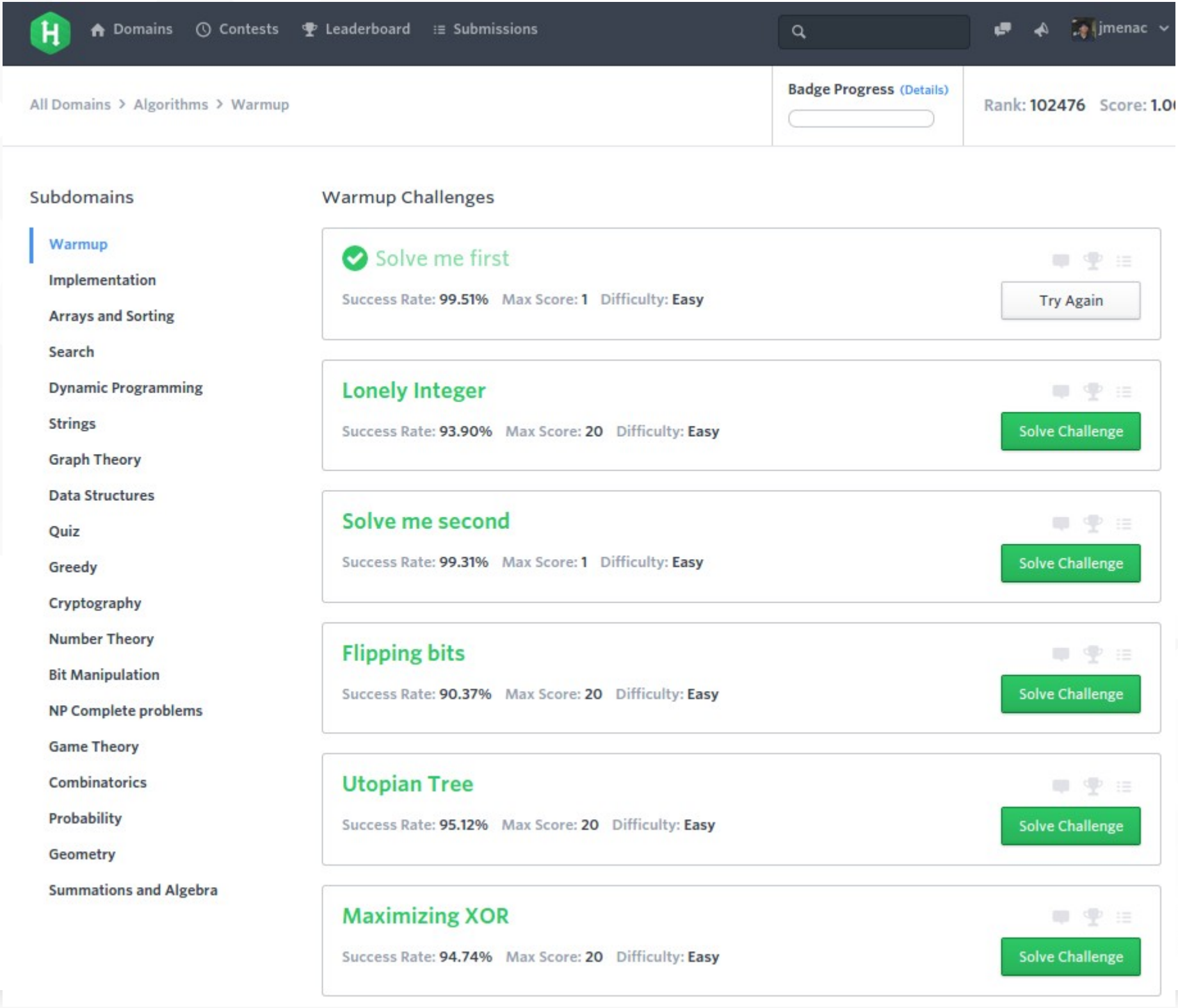
Recursão

Uma função recursiva não necessariamente é aquela que se chama a si mesma

```
int F1(parametros)
{

   .
   .
   .

   F2(parametros)
   .
   .
}
```

```
int F2(parametros)
{

   .
   .
   .

   F1(parametros)
   .
   .
}
```

Sobre a lista de exercícios


Domains
Contests
Leaderboard
Submissions
jmenac
All Domains > Algorithms > Warmup
Badge Progress (Details)
Rank: 102476 Score: 1.0
Subdomains
Warmup
Implementation
Arrays and Sorting
Search
Dynamic Programming
Strings
Graph Theory
Data Structures
Quiz
Greedy
Cryptography
Number Theory
Bit Manipulation
NP Complete problems
Game Theory
Combinatorics
Probability
Geometry
Summations and Algebra
Warmup Challenges
Solve me first
Success Rate: 99.51% Max Score: 1 Difficulty: Easy
Try Again
Lonely Integer
Success Rate: 93.90% Max Score: 20 Difficulty: Easy
Solve Challenge
Solve me second
Success Rate: 99.31% Max Score: 1 Difficulty: Easy
Solve Challenge
Flipping bits
Success Rate: 90.37% Max Score: 20 Difficulty: Easy
Solve Challenge
Utopian Tree
Success Rate: 95.12% Max Score: 20 Difficulty: Easy
Solve Challenge
Maximizing XOR
Success Rate: 94.74% Max Score: 20 Difficulty: Easy
Solve Challenge

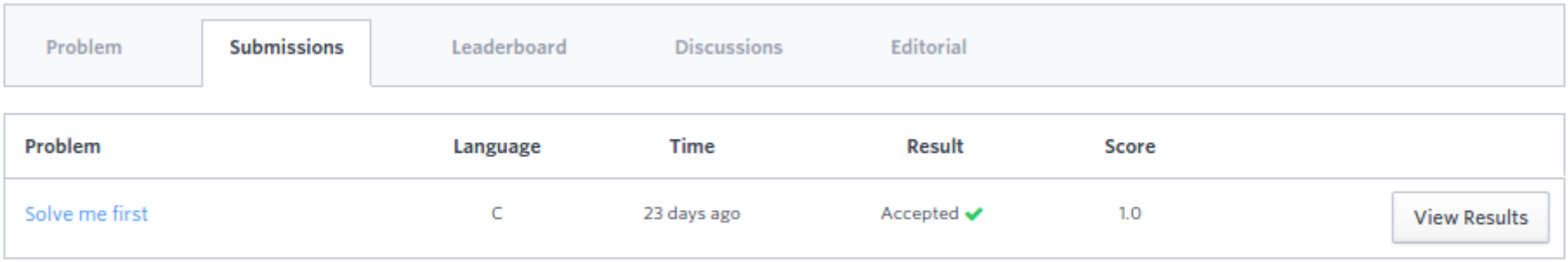
Current Buffer (saved locally, editable)

C

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


int solveMeFirst(int a, int b) {
  return a+b;
}
int main() {
  int num1,num2;
  scanf("%d %d",&num1,&num2);
  int sum;
  sum = solveMeFirst(num1,num2);
  printf("%d",sum);
  return 0;
}

```

Line: 1 Col: 1

Upload Code as File **Test against custom input** Run Code Submit Code


Problem
Submissions
Leaderboard
Discussions
Editorial
Problem
Language
Time
Result
Score
Solve me first
C
23 days ago
Accepted
1.0
View Results

Lista 01: https://www.hackerrank.com

- **Solve me first**
- **Solve me second**
- **Lonely integer**
- Flipping bits
- Maximizing XOR

Será utilizado um programa de deteção de plágio em todas as submissões!

Data: 15/Fevereiro (domingo) até às 23h50.

Envio: Através do Tidia.

Arquivos:

Para cada exercício-problema deverá submeter:

- O código fonte: nome do arquivo → RA_nomeDoProblema.c
- O comprovante de aceitação (screenshot) → RA_nomeDoProblema.pdf

Exemplo: 10123456_solveMeFirst.c
10123456_solveMeFirst.pdf