

# Prova, implementação e análise de problemas selecionados

CARLOS MATTOSO, IAN ALBUQUERQUE E LEONARDO KAPLAN

PUC-Rio

## Resumo

*Neste projeto procuramos provar e implementar soluções para os problemas apresentados em aula da forma mais clara possível. Otimizações foram feitas desde que não obscurassem as induções que originam os algoritmos.*

## I. INTRODUÇÃO

**N**os foram apresentados 3 teoremas para provar e implementar:  
Para o primeiro teorema, a prova nos foi apresentada, restando apenas a codificação.

No segundo, primeiro enunciamos e provamos um teorema equivalente ao proposto, para então responder o que foi perguntado.

Provamos e implementamos o último teorema, sendo  $k = 15$  o maior valor calculado.

Todos os algoritmos foram implementados em C++ , alguns com o uso de bibliotecas externas a fim de otimização.

## II. TEOREMA 1

### I. Enunciado

**Teorema 1 :**  $x^n - y^n$  é divisível por  $x - y$  para quaisquer  $x$  e  $y$  inteiros e todos o valores de  $n$  inteiros e maiores que zero.

### II. Prova

A prova é feita por indução matemática utilizando  $k$  como parâmetro de indução. O teorema 1 pode ser enunciado:

**Teorema 1 (k):**  $x^k - y^k$  é divisível por  $x - y$  para quaisquer  $x$  e  $y$  inteiros e todos os valores de  $k$  inteiros e maiores que zero.

### Teorema do Caso Base:

#### Teorema 1 (1):

Seja  $k=1$  (o menor valor para o qual  $k$  tem que ser verdade). Nesse caso temos que provar que  $x-y$  é divisível por  $x-y$  para qualquer valor de  $x$  e  $y$ . O que é trivialmente verdade, sendo o quociente,  $q_1$ , igual a 1.

### Teorema do Passo Indutivo:

#### Teorema 1 (k) $\rightarrow$ Teorema 1 (k+1)

Desejamos provar que se o teorema 1 é verdade para um  $k$  fixo, isto é, podemos assumir que:  $x^k - y^k = q_k * (x - y)$ , onde  $q_k$  é um inteiro, então é possível mostrar que  $x^{k+1} - y^{k+1} = q_{k+1} * (x - y)$  para algum  $q_{k+1}$  inteiro. Ou seja, temos mostrar que é verdade também para  $k+1$ . Isto é, que podemos obter  $q_{k+1}$  inteiro a partir de  $q_k$  se o teorema 1 é verdade para  $k$ .

Como:

$$\begin{aligned} x^{k+1} - y^{k+1} &= \\ x^{k+1} - x^k * y + x^k * y - y^{k+1} &= \\ x^k * (x - y) + y * (x^k - y^k) & \end{aligned}$$

Como, pela hipótese indutiva, temos que

$$\begin{aligned} x^k - y^k &= q_k * (x - y), \text{ podemos escrever:} \\ x^{k+1} - y^{k+1} &= \\ x^k * (x - y) + y * (x^k - y^k) &= \\ x^k * (x - y) + y * q_k * (x - y) &= \\ (x^k + y * q_k) * (x - y) & \end{aligned}$$

Como  $x$  é inteiro,  $x^k$  é inteiro. Como  $y$  e  $q + k$  são inteiros seu produto também é inteiro. Portanto,  $(x^k + y * q_k)$  é inteiro e  $q_{k+1} = (x^k + y * q_k)$  é inteiro, ou seja:  $x^{k+1} - y^{k+1} = (x^k + y * q_k) * (x - y) = q_{k+1} * (x - y)$

Assim:  $q_{k+1} = (x^k + y * q_k)$

### III. Algoritmo Resultante

Com base na prova por indução, pode-se extrair um algoritmo para determinar  $Q[k]$ , dados  $x$ ,  $y$  e  $k$ .

---

```
function q(int x, int y, unsigned int k){
    if( k == 1 )
        return 1
    else
        return x^(k-1) + y * q(x, y, k-1)
}
```

---

### IV. Comentários

O algoritmo funcionou como esperado para todos os valores testados. A fim de checar de certa forma a corretude do código, foi criado um programa em *Python* para gerar em torno de quatro milhões de quocientes, para diferentes valores de  $x$ ,  $y$  e  $k$ . Comparando-se com a saída de nossa implementação do algoritmo acima, constatou-se que os resultados foram idênticos.

Em anexo apresentamos o arquivo *ex1-small.log* que mostra uma série de resultados de nossos testes. Alguns resultados:

---

```
x[-25] y[-24] k[2]
Quocient: -49
```

```
x[-25] y[-24] k[24]
Quocient:
-2218977901950216804906539865046849
```

```
x[24] y[23] k[24]
```

Quocient:

853485012853782147658915715900735

---

Finalmente, fizemos um teste com **364.500.000** iterações, aproximadamente, sendo o tempo total necessário de execução de **210.094s** (disponível em *ex1-huge.log*), evidenciando quão otimizado encontra-se o código. As classes desenvolvidas apresentam um bom nível de otimização, armazenando valores relevantes para consultas futuras.

Como é necessário saber diferentes potências de  $x$ , a classe de exponenciação lembra-se das já calculadas. Além disso, como é necessário saber valores passados de quocientes, a classe que implementa o algoritmo lembra-se de quocientes passados (dados  $x$  e  $y$ ).

## III. TEOREMA 2

### I. Enunciado

**Teorema 2** O número de números inteiros cujos dígitos pertencem ao conjunto  $\{1, 2, \dots, m\}$  de  $K$  dígitos diferentes é dado pelo produto  $m.(m - 1) \dots (m - k + 1)$ .

**Define-se:**

$d_x :=$  algarismo de valor absoluto  $x$

$N_{n,m} := \{x \mid x = a_n \dots a_2 a_1 \text{ tal que } a_i \in \{d_1, d_2, \dots, d_m\}, \text{ o conjunto de dígitos} \} =$  números de  $n$  dígitos com dígitos com valor até  $m$ .

Podemos enunciar o seguinte teorema, válido para todo  $n$  e  $m$  inteiros, tal que  $n, m \geq 1$ :

**Teorema 2 ( $n, m$ )** Sabe-se enumerar explicitamente os elementos do conjunto:

$E_{n,m} = \{x \mid x \in N_{n,m} \text{ e } \forall i, j \in \mathbb{N}, 1 \leq i, j \leq n ((i \neq j) \rightarrow (a_i \neq a_j))\} =$  números de  $n$  dígitos com dígitos distintos com valor até  $m$ .

### II. Prova

(Prova por indução no parâmetro  $n$ )

**Teorema(s) do(s) Caso(s) Base(s):****Teorema 3 (0,m):** (Caso Degenerado)

$$n = 0 \rightarrow N_{0,m} = \emptyset$$

$$E_{0,m} \subseteq N_{0,m} \rightarrow E_{0,m} = \emptyset$$

$E_{0,m}$  não possui elementos, logo podemos enumerar seus elementos como: {}

**Teorema 3 (1,m):**

Na definição de  $E_{1,m}$  como  $1 \leq i, j \leq 1 \rightarrow i = j$  então  $((i \neq j) \rightarrow (a_i \neq a_j))$  é sempre verdadeiro.

$$\text{Logo, } E_{1,m} = N_{1,m} = \{d_1, d_2, \dots, d_m\}$$

Assim podemos enumerar os elementos de  $E_{1,m}$  explicitamente como  $\{d_1, d_2, \dots, d_m\}$ .

**Teorema do Passo Indutivo:****Teorema 3 (k,m)  $\rightarrow$  Teorema 3 (k+1,m)**

Pela hipótese indutiva sabe-se enumerar explicitamente o conjunto  $E_{k,m}$

Deseja-se enumerar explicitamente o conjunto  $E_{k+1,m}$  utilizando-se do fato de conhecer-se a enumeração de  $E_{k,m}$ .

**Define-se:**

$$E_{k,m}^{d_i} = \{x | x \in E_{k,m} \text{ e todos os algarismos de } x \text{ são diferentes de } d_i\}$$

Por construção  $E_{k,m}^{d_i} \subset E_{k,m}$ , logo sabe-se enumerar explicitamente os elementos de  $E_{k,m}^{d_i}$  por hipótese indutiva.

Façamos:

$$E_{k+1,m} = \bigcup_{i \in \{1,2,\dots,m\}} \{x | x = y \text{ concatenado com } d_i, \forall y \in E_{k,m}^{d_i}\}$$

Observar que  $x \in N_{k+1,m}$  e que  $\forall i, j \in \mathbb{N}, 1 \leq i, j \leq (k+1) ((i \neq j) \rightarrow (a_i \neq a_j))$  conforme desejado.

**III. Algoritmo Resultante**

Com base na prova por indução em n do **Teorema 2 (n,m)** podemos construir o seguinte algoritmo:

**Numeros** *geraNumeros*( **int** n, **int** m )

```
{
    // Casos Bases
    if (n == 0)
    {
        Numeros E0,m;
        E0,m = ∅;
        return E0,m;
    }
    if (n == 1)
    {
        Numeros E1,m;
        E1,m = {d1, d2, ..., dm};
        return E1,m;
    }
    // Passo Indutivo
    else
    {
        Numeros En,m;
        Numeros En-1,m;
        En-1,m = geraNumeros(n - 1, m);
        for(int i from 1 to m by 1)
        {
            En-1,mdi = En-1,m sem elem. com di
            for(int y in En-1,mdi)
            {
                add (concatena(y, di)) to En,m;
            }
        }
        return En,m;
    }
}
```

**IV. Comentarios**

Primeiro provamos o teorema de que sabe-se enumerar todos estes números. Depois, implementamos o algoritmo resultante da prova, que enumera todos os m.(m-1)...(m - k + 1) números, o que permite contá-los.

**IV. TEOREMA 3****I. Enunciado**

**Teorema 3(k)** Sabe-se construir  $2^k - 1$  rodadas de  $2^k - 1$  jogos onde cada equipe enfrenta uma

equipe diferente em cada rodada.

## II. Prova

### Teorema(s) do(s) Caso(s) Base(s):

#### Teorema 3 (0): (Caso Degenerado)

$k = 0 \rightarrow n = 2^0 = 1$  equipe

Conjunto de Equipes  $E = \{e_1\}$

Não há equipes suficiente para montar nenhum jogo. Como não é possível montar nenhum jogo, não é possível montar nenhuma rodada. Logo: 0 rodadas são possíveis de serem formadas.

Conjunto de Rodadas  $R^0 = \emptyset$

Número de rodadas  $= 1 = 2^0 - 1 = (n - 1)$

#### Teorema 3 (1):

$k = 1 \rightarrow n = 2^1 = 2$  equipes

Conjunto de Equipes  $E = \{e_1, e_2\}$

Há somente duas equipes. Logo, podemos montar o seguinte conjunto de rodadas:

Conjunto de Rodadas  $R^1 = \{R_1\}$

$R_1^1 = \{(e_1, e_2)\}$

Número de rodadas  $= 1 = 2^1 - 1 = (n - 1)$

Com essa configuração de rodadas,  $e_1$  joga com  $e_2$ , satisfazendo a condição de que cada time jogue com todos os outros.

### Teorema do Passo Indutivo:

#### Teorema 3 (k) $\rightarrow$ Teorema 3 (k+1)

Pela hipótese indutiva sabe-se construir o conjunto de rodadas  $R^k$  para um número de equipes  $n = 2^k$ .

Deseja-se provar que sabe-se construir o conjunto de rodadas  $R^{k+1}$  para um número de equipes  $n' = 2^{k+1} = 2 * 2^k = 2n$ .

Sejam o conjunto de equipes:

$E = \{e_1, e_2, \dots, e_{2n}\}$

Separemos dois subconjuntos de E:

$E_1 = \{e_1, e_2, \dots, e_n\}$

$E_2 = \{e_{n+1}, e_{n+2}, \dots, e_{2n}\}$

Observe que:

$\text{card}(E_1) = \text{card}(E_2) = n = 2^k$

$E_1 \cup E_2 = E$

$E_1 \cap E_2 = \emptyset$

Pela hipótese indutiva, sabe-se construir as rodadas referentes a  $E_1$  e  $E_2$ , uma vez que em cada time existem exatos  $n = 2^k$  equipes. Faltam apenas as partidas que envolvem uma equipe de  $E_1$  com uma equipe de  $E_2$

Denotemos, respectivamente, o conjunto de rodadas relativos a  $E_1$  e  $E_2$  como  $R^{k,E_1}$  e  $R^{k,E_2}$  e cada rodada como  $R_i^{k,E_1}$  e  $R_j^{k,E_2}$  para  $1 \leq i, j \leq (n - 1)$ .

Montemos o conjunto de rodadas  $R^{k+1}$  como  $R^{k+1} = \{R_1^{k+1}, R_2^{k+1}, \dots, R_{2n-1}^{k+1}\}$  tal que:

Primeiras  $n-1$  rodadas (Partidas que ocorrem internamente entre equipes de  $E_1$  e  $E_2$ ):

$R_i^{k+1} = R_i^{k,E_1} \cup R_i^{k,E_2} \quad \forall i \in \mathbb{N}, 1 \leq i \leq (n - 1)$

Próximas  $n$  rodadas (partidas que envolvem uma equipe de  $E_1$  com uma equipe de  $E_2$ ):

$R_{n+\tau}^{k+1} = \{(e_{1+((j+\tau)\%n)}, e_{j+n}) \in \mathbb{E}^2 \mid j \in \mathbb{N}, 1 \leq j \leq n\} \quad \forall \tau \in \mathbb{Z}, 0 \leq \tau \leq (n - 1)$

Observar que  $e_{1+((j+\tau)\%n)} \in E_1$  e  $e_{j+n} \in E_2$  para todos os valores de  $j$  e  $\tau$  nos limites estipulados.

Notar ainda que dado um  $j$ , variando-se  $\tau$  nos limites estipulados, têm-se que  $e_{j+n} \in E_2$  joga com todos os elementos de  $E_1$  e que variar  $j$  nos limites estipulados equivale a percorrer-se os elementos de  $E_2$ .

Uma vez que todo elemento de  $E_1$  e  $E_2$  jogou com os próprios elementos de seu respectivo conjunto assim como com todos os elementos do outro conjunto e como  $E_1 \cup E_2 = E$  então  $R^{k+1}$  é um conjunto de rodadas que satisfaz o teorema. Observar que  $\text{card}(R^{k+1}) = 2n - 1$ , conforme esperado.

Sabe-se, portanto, construir o conjunto de rodadas  $R^{k+1}$  para um número de equipes.

## III. Algoritmo Resultante

Com base na prova por indução em  $k$  do Teorema 3(k) podemos construir o seguinte algoritmo:

```
Rodadas geraRodadas( int k, Equipes E )
{
    // Casos Bases
    if (k == 0)
    {
        Rodadas R0;
```

```

     $R^0 = \emptyset$ ;
    return  $R^0$ ;
}
if (k == 1)
{
    Rodadas  $R^1$ ;
     $R_1^1 = \{(e_1, e_2)\}$ ;
    return  $R^1$ ;
}
// Passo Indutivo
else
{
    Rodadas  $R^k$ ;
    int  $n = 2^k$ ;
     $E_1 = \{e_1, e_2, \dots, e_{\frac{n}{2}}\}$ ;
     $E_2 = \{e_{\frac{n}{2}+1}, e_{\frac{n}{2}+2}, \dots, e_n\}$ ;
     $R^{k-1, E_1} = \text{geraRodadas}(k-1, E_1)$ ;
     $R^{k-1, E_2} = \text{geraRodadas}(k-1, E_2)$ ;
    for(int i from 1 to  $(\frac{n}{2} - 1)$  by 1)
    {
         $R_i^k = R_i^{k-1, E_1} \cup R_i^{k-1, E_2}$ 
    }
    for(int  $\tau$  from 0 to  $(\frac{n}{2} - 1)$  by 1)
    {
         $R_{\frac{n}{2}+\tau}^k = \emptyset$ ;
        for(int j from 1 to  $(\frac{n}{2} - 1)$  by 1)
        {
            add  $(e_{1+((j+\tau)\% \frac{n}{2})}, e_{j+\frac{n}{2}})$  to  $R_{\frac{n}{2}+\tau}^k$ ;
        }
    }
    return  $R^k$ ;
}
}

```

#### IV. Resultados

Utilizando o algoritmo a cima construímos um programa que gera as rodadas para  $2^k$  equipes, uma vez dado um k. Testamos esse programa coletando informações sobre o tempo de execução para diversos valores de k. Abaixo, segue-se alguns dos principais resultados encontrados:

##### Results for k=0:

1 teams.  
0 rounds with 0 matches per round.

0 total number of matches.  
3016214 runs in 5s  
Average time per run: 1.65771e-006s  
Rounds generated:

##### Results for k=1:

2 teams.  
1 rounds with 1 matches per round.  
1 total number of matches.  
2012070 runs in 5s  
Average time per run: 2.485e-006s  
Rounds generated:  
Round 1: {(1,2)}

##### Results for k=2:

4 teams.  
3 rounds with 2 matches per round.  
6 total number of matches.  
1039078 runs in 5s  
Average time per run: 4.81196e-006s  
Rounds generated:  
Round 1: {(1,2),(3,4)}  
Round 2: {(2,3),(1,4)}  
Round 3: {(1,3),(2,4)}

##### Results for k=3:

8 teams.  
7 rounds with 4 matches per round.  
28 total number of matches.  
461214 runs in 5s  
Average time per run: 1.0841e-005s  
Rounds generated:

Round 1: {(1,2),(3,4),(5,6),(7,8)}  
Round 2: {(2,3),(1,4),(6,7),(5,8)}  
Round 3: {(1,3),(2,4),(5,7),(6,8)}  
Round 4: {(2,5),(3,6),(4,7),(1,8)}  
Round 5: {(3,5),(4,6),(1,7),(2,8)}  
Round 6: {(4,5),(1,6),(2,7),(3,8)}  
Round 7: {(1,5),(2,6),(3,7),(4,8)}

##### Results for k=14:

16384 teams.  
16383 rounds with 8192 matches per round.  
134209536 total number of matches.  
1 runs in 55.2969s  
Average time per run: 55.2969s

**Results for k=15:**

32768 teams.

32767 rounds with 16384 matches per round.

536854528 total number of matches.

1 runs in 793.641s

Average time per run: 793.641s

É de simples verificação que o programa baseado no algoritmo descrito gerou corretamente rodadas para valores de  $k$  pequenos ( $k \in \{0, 1, 2, 3\}$ ). Porém, como o número de partidas cresce exponencialmente, para valores de  $k$  maiores que 3 não é prático avaliar diretamente a saída do programa manualmente.

Somente foi possível rodar o programa para  $k \leq 15$  (um total de 536.854.528 jogos) devido ambas falta de memória e tempo de execução alto. Uma vez que para  $k=15$  levou-se aproximadamente 13 minutos para calcular-se a resposta (em comparação a aproximadamente 1 minuto necessário para  $k=14$ ) é razoável estimar que para  $k=16$  levaria-se algumas horas e para  $k=17$  levaria-se mais de um dia. Além disso o programa possui um custo alto de

memória devido ao alto número de partidas envolvidas. Para  $k=14$  já são necessários mais de 2GB de memória disponível.

## V. CONCLUSÃO

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultrices scelerisque semper. Donec cursus euismod enim eget sodales. Duis sit amet lacinia leo, at tristique libero. Suspendisse eget placerat felis. Nulla non nibh pharetra, sodales mauris et, tristique arcu. Donec placerat ornare convallis. Vestibulum elementum enim eu aliquet faucibus. Donec rutrum orci at erat tempus, ac pellentesque sapien facilis. Morbi suscipit euismod ligula, vel laoreet ligula euismod in. Phasellus suscipit neque tellus, ut eleifend dui sollicitudin vel. Vivamus ut hendrerit justo.

Curabitur vitae massa ullamcorper, ullamcorper metus nec, condimentum purus. Aenean nec lacus metus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Morbi tincidunt est ut quam molestie convallis.