Prova, Implementação e Análise de Problemas Selecionados

Carlos Mattoso, Ian Albuquerque e Leonardo Kaplan

PUC-Rio

I. Introdução

Apresentou-se três problemas com o objetivo de desenvolver-se códigos com diferentes estruturas de dados e algoritmos para solucioná-los, assim como analisar-se o desempenho das implementações destes algoritmos com respeito ao tempo de CPU.

É possível enunciar-se um teorema correspondente para cada um dos problemas descritos, o que permite descrever-los, demonstrar seu entendimento em detalhe, explicar sua fundamentação e justificar sua corretude através de uma prova por indução matemática, montar um algoritmo com base neles e apresentar dados a respeito do tempo de execução dos algoritmos para diversas instâncias. Dessa forma, discriminamos para cada um dos três problemas seus teoremas correspondentes, cada um com seu enunciado, prova, algoritmo resultate, comentários e resultados.

As provas dos teoremas se baseiam principalmente no princípio de indução matemática de Peano, conforme descrito abaixo:

Se ϕ é um predicado unário tal que: $\phi(0)$ é verdade, e para todo número natural n, se ϕ(n) é verdadeiro, então ϕ(S(n)) também o é, então ϕ(n) é verdadeiro para todo número natural n.

Assim, para provar-se um teorema para diversos valores em um domínio discreto, basta provar-se os chamados Teoremas de Caso Base, em que prova-se o teorema para valores específicos e usualmente de simples entendimento do domínio, assim como os Teoremas do Passo Indutivo, em que prova-se que caso o teorema seja verdade para algum valor do domínio, o teorema também será verdade para outros valores do domínio diferentes do da hipótese.

Dessa forma, segue-se a desenvolvimento de cada um dos problemas solicitados.

II. Problema 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:
$x^{k+1} - y^{k+1} =$
$x^{k+1} - x^k * y + x^k * y - y^{k+1} =$
$x^k * (x - y) + y * (x^k - y^k)$

Como, pela hipótese indutiva, temos que $x^k - y^k = q_k * (x - y)$, podemos escrever:
$x^{k+1} - y^{k+1} =$
$x^k * (x - y) + y * (x^k - y^k) =$
$x^k * (x - y) + y * q_k * (x - y) =$
$(x^k + y * q_k) * (x - y)$

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 o seguinte algoritmo para determinar Q[k], dados x, y e k.

```
function q(int x, int y, unsigned int k){
 // Caso Base
  if( k == 1 )
    return 1
 // Passo Indutivo
  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.

É importante ressaltar que em casos para os quais **x == y**, o algoritmo retornará valo-

res diferentes dos esperados, apresentando comportamento indesejado. Contudo, dados o teorema enunciado e sua respectiva prova, não é incumbência do algoritmo desta derivado tratar de tais casos. Assim, chamadas ao algoritmo devem ser feitas apenas para **x != y**.

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:
      -22189779019502168049065398650468 49

x = 24, y = 23, k=24
  Quocient:
      85348501285378214765891571590073 5
```

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. Problema 2

I. Enunciado

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

II. Prova

Define-se:
d_x := algarismo de valor absoluto x
$N_{n,m} := \{x|\ x = a_n \dots a_2 a_1$ tal que $a_i \in \{d_1, d_2, \dots, d_m\}$, 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 $(\mathbf{n}, \mathbf{m})$ Sabe-se enumerar explicitamente os elementos do conjunto:
$E_{n,m} = \{x|\ x \in N_{n,m}$ 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.

Prova por indução no parâmetro de indução (n):
Teorema(s) do(s) Caso(s) Base(s):

Teorema 2 (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 2 (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.
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 (em n):
Teorema 2 (k,m) $\rightarrow$ ***Teorema 2 (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:
$E^{d_i}_{k,m} := \{x|x \in E_{k,m}$ e todos os algarismos de x são diferente de $d_i\ \}$

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

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

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** $(\mathbf{n}, \mathbf{m})$ podemos construir o seguinte algoritmo:

```
Numeros geraNumeros( int n, int m )
{
    // Casos Bases
    if (n == 0)
    {
        Numeros E_{0,m};
        E_{0,m} = ∅;
        return E_{0,m};
    }
    if (n == 1)
    {
        Numeros E_{1,m};
        E_{1,m} = {d_1, d_2, ..., d_m};
```

```
        return E_{1,m};
    }
    // Passo Indutivo
    else
    {
        Numeros E_{n,m};
        Numeros E_{n-1,m};
        E_{n-1,m} = geraNumeros(n - 1, m);
        for(int i from 1 to m by 1)
        {
            E^{d_i}_{n-1,m} = E_{n-1,m} sem elem. com d_i
            for(int y in E^{d_i}_{n-1,m})
            {
                add (concatena(y,d_i)) to E_{n,m};
            }
        }
        return E_{n,m};
    }
}
```

IV. Comentários

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

Nas saídas disponibilizadas, os diferentes números gerados são exibidos através da sequência de dígitos que os formam, sendo estes delimitados cada um por colchetes. Além disso, para cada instância de teste, à direita de **m** e **k** são apresentados seus valores, também estes delimitados por colchetes.

Em anexo disponibilizamos o arquivo ***ex2-small.log*** que apresenta o resultado da execução do algortimo para **m** de **1 até 9** e **k** de **1 até 9**, possibilitando checar se a implementação funciona pelo menos para pequenos valores. Abaixo disponibilizamos um resultado para visualização.

```
m[ 3 ] k[ 3 ]
#numbers: 6

[1][2][3]
[1][3][2]
[2][1][3]
[2][3][1]
[3][1][2]
[3][2][1]
```

Para valores de **m** maiores que 9, apresentamos alguns testes no arquivo ***ex2-small-2.log***. Abaixo, alguns resultados para o caso de m = 20 e k = 4:

```
m[ 20 ] k[ 4 ]
  #numbers: 116280
  Trial time: 0s
  [1][2][3][4]
  [1][2][3][5]
  [1][2][3][6]
      ...
  [20][19][18][15]
  [20][19][18][16]
  [20][19][18][17]
```

Para maiores valores, conseguimos fazer nosso algoritmo funcionar de maneira satisfatória até **m = 17** e **k = 7**, como apresentado na tabela abaixo. Os tempos de execução para cada instância de teste são indicados na tabela, sendo o valor médio de **115.7529** segundos. Pode-se observar um crescimento considerável quando da adição de um novo dígito. Os dados da tabela encontram-se no arquivo ***ex2-huge.log***.

m	k	#numbers	Time (s)
10	6	151200	0.203125
10	7	604800	0.828125
11	6	332640	0.421875
11	7	1663200	2.3125
12	6	665280	0.8125
12	7	3991680	5.45312
13	6	1235520	2.03125
13	7	8648640	12.7656
14	6	2162160	5.875
14	7	17297280	66.4375
15	6	3603600	16.2031
15	7	32432400	216.891
16	6	5765760	8.35938
16	7	57657600	519.156
17	6	8910720	16.0625
17	7	98017920	978.234

O algoritmo implementado acaba necessitando de grandes quantidades de memória, o que acaba sendo um considerável gargalo em sua execução. Para os maiores casos apresentados abaixo, o algoritmo consumia em torno de **40GB**, fazendo uso principalmente de memória virtual montada no HDD, afetando seu desempenho.

IV. Problema 3

I. Enunciado

Seja um conjunto n de equipes $e_1, e_2, \ldots, e_n$. Deseja-se construir as n-1 rodadas de um campeonado onde todos jogam contra todos. Assuma que $n = 2^k$ para alguma k. Enuncia-se abaixo o teorema de que sabe-se construir as n-1 rodadas de n/2 jogos cada:

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:
$E = \{e_1, e_2, \ldots, e_{2n}\}$
Separemos dois subconjuntos de E:
$E_1 = \{e_1, e_2, ..., e_n\}$
$E_2 = \{e_{n+1}, e_{n+2}, \ldots, e_{2n}\}$
Observe que:
$card(E_1) = card(E_2) = n = 2^k$
$E_1 \cup E_2 = E$
$E_1 \cap E_2 = \emptyset$
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 $R^{k+1} = \{R_1^{k+1}, R_2^{k+1}, \ldots, R_{2n-1}^{k+1}\}$ tal que:

Primeiras n-1 rodadas (Partidas que ocorrem internamente entre equipes de E_1 e E_2):
$R_i^{k+1} = R_i^{k,E_1} \cup R_i^{k,E_2}\ \forall i \in \mathbb{N}, 1 \leq i \leq (n-1)$

Próximas n rodadas (partidas que envolvem uma equipe de E_1 com uma equipe de E_2):
$R_{n+\tau}^{k+1} = \{(e_{1+((j+\tau)\%n)}, e_{j+n}) \in \mathbb{E}^2 | j \in \mathbb{N}, 1 \leq j \leq n\ \}\ \forall \tau \in \mathbb{Z}, 0 \leq \tau \leq (n-1)$

Observar que $e_{1+((j+\tau)\%n)} \in E_1$ e $e_{j+n} \in E_2$ para todos os valores de j e τ nos limites estipulados.
Notar ainda que dado um j, variando-se τ 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 elemntos 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 $card(R^{k+1}) = 2n - 1$, 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 R^0;
        R^0 = ∅;
        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, ..., e_{n/2}};
        E_2 = {e_{n/2+1}, e_{n/2+2}, ..., e_n};
        R^{k-1,E_1} = geraRodadas(k - 1, E_1);
        R^{k-1,E_2} = geraRodadas(k - 1, E_2);
        for(int i from 1 to (n/2 - 1) by 1)
        {
            R^k_i = R^{k-1,E_1}_i ∪ R^{k-1,E_2}_i
        }
        for(int τ from 0 to (n/2 - 1) by 1)
        {
            R^k_{n/2+τ} = ∅;
            for(int j from 1 to (n/2 - 1) by 1)
            {
                add (e_{1+((j+τ)%n/2)}, e_{j+n/2}) to R^k_{n/2+τ};
            }
        }
        return R^k;
    }
}
```

IV. Comentários

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 os últimos valores de k computados o uso de memória atingia valores maiores que 8GB.

V. Conclusão

Todos os algoritmos resultantes das provas por indução conseguiram resolver os problemas apresentados. Contudo, para valores de seus respectivos parâmetros que produzam grandes resultados, todos os algoritmos acabaram pecando em termos de desempenho, principalmente em uso de memória.

O primeiro foi o mais eficiente e de mais fácil otimização, tendo sido ainda assim necessário o uso de uma biblioteca externa para manipulação de grandes inteiros.

Quanto ao segundo e terceiro, as operações matemáticas utilizadas na pseudocódigo são aquelas básicas de conjuntos, principalmente união, e assim constituem-se certas ineficiências inerentes às estruturas utilizadas pelo algoritmo. Em termos de memória, são gerados milhões de valores já para pequenos parâmetros, o que acaba explodindo seu uso. Sendo então necessário o uso de bastante memória virtual (armazenada no HDD), o desempenho do programa sofre significativamente.