Solução e análise do problema de programação "Contando Palíndromos"

Raphael Rocha da Silva

20 de novembro de 2016

É apresentado neste texto o problema *Contando Palíndromos*, que foi elaborado com molde em competições de programação, como a Maratona de Programação da Sociedade Brasileira de Computação. Após descrever o problema, é mostrada uma solução que procura seguir uma linha de raciocínio que um competidor poderia usar durante uma prova. Em seguida, a complexidade da solução é analisada, a fim de estimar-se quão eficiente pode-se esperar que a solução de um competidor seja. Então, é justificado o nível de dificuldade do problema, levando-se em conta quanto tempo demora-se para encontrar uma solução, para implementá-la, quais técnicas o competidor deve saber e quais dificuldades e erros podem prejudicá-lo.

1 O Problema

O problema consiste em contar a quantidade de números palíndromos que existem entre dois números dados.

Entrada
A entrada contém vários casos de teste, cada um formado por dois números inteiros A e B, com $1 \leq A \leq B \leq 1\,000\,000\,000 = 10^9$. O caso $A = B = 0$ indica que o programa deve ser terminado.

Saída
Para cada caso de teste, a saída deve conter um número igual à quantidade de números palíndromos que há entre A e B, intervalo fechado. Esse número deve ser seguido por uma quebra de linha.

Exemplo de entrada e saída

Entrada:

```
1 12
10 15
11 11
12 14
200 300
25 1542
0 0
```

Saída:

```
10
1
1
0
10
102
```

2 Solução

Para simplicidade, pode-se escrever uma função que retorne a quantidade de palíndromos *menores* que um determinado número. Assim, para dizer quantos palíndromos existem entre A e B, basta subtrair o número de palíndromos menores que A do número de palíndromos menores que $B + 1$.

Com isso, a partir daqui, o problema a ser resolvido cai no que enuncia o Problema 1:

Problema 1 *Escrever um algoritmo que informe quantos palíndromos menores do que um número N existem.*

Assuma que $\boldsymbol{N}$ seja um inteiro positivo de $\boldsymbol{D}$ dígitos. Esses rótulos serão usados durante todo este texto.

A resolução do problema estará apoiada em técnicas de Análise Combinatória.

Para manter a solução simples, divertida e de fácil entendimento, é interessante dividir a contagem em várias partes. Serão contados separadamente:

- Para cada d de 1 até $D-1$:
  a quantidade de palíndromos de d dígitos que existem.
- A quantidade de palíndromos de D dígitos que são menores/iguais a N.

Assim, o Problema 1 será dividido no que enunciam os Problemas 2 e 3 das próximas seções.

2.1 Contagem de palíndromos de d dígitos

Problema 2 *Escrever um algoritmo que retorne a quantidade de palíndromos de d dígitos que existem.*

Escrever uma função que retorne quantos palíndromos de uma certa quantidade de dígitos existem é muito simples. É o que faz o Algoritmo 1, escrito em C++.

Algoritmo 1 – contaPaliDigs
Retorna a quantidade de palíndromos de d dígitos que existem

```
1  int contaPaliDigs (int d)
2  {
3    if (d%2==0)
4      return 9*pow(10, d/2-1);
5    else
6      return 9*pow(10, (d-1)/2);
7  }
```

O Algoritmo 1 usa um raciocínio simples de Análise Combinatória. Tome um número com d dígitos. Conta-se de quantas maneiras diferentes esses dígitos podem ser preenchidos de modo que o número seja palíndromo. Considere inicialmente que d seja par.

- O primeiro dígito pode ser $\{1, 2, \ldots, 9\}$ (ele não pode ser 0, senão o número não teria d dígitos);
- O último dígito deve ser igual ao primeiro (senão o número não seria palíndromo);
- O segundo dígito pode ser $\{0, 1, 2, \ldots, 9\}$;
- O penúltimo dígito deve ser igual ao segundo;
- Segue até completar todos os dígitos.

Assim, há 9 possibilidades para o primeiro dígito, 10 para o segundo, 10 para o terceiro, e assim sucessivamente, até o dígito $\frac{d}{2}$. Para os demais dígitos, há apenas uma possibilidade para cada um, porque eles devem repetir os dígitos que já foram escolhidos.

Usa-se então esse raciocínio para contar o número total de palíndromos de d dígitos. Porém, deve-se dividir a contagem em dois casos, que dependem da paridade de d, como descrito a seguir.[1]

1 O Algoritmo 1 pode ainda ser reduzido se seu valor de retorno for igual a $9 \times 10^{\frac{d+1}{2}-1}$ independentemente de d ser par ou ímpar, considerando que a divisão é feita dentro dos números inteiros. Neste texto ele foi escrito da outra maneira para favorecer a legibilidade.

O número total de palíndromos de d dígitos para d par é:

$$9 \times \underbrace{10 \times 10 \times \ldots \times 10}_{\frac{d}{2}-1 \text{ vezes}} \times \underbrace{1 \times 1 \times 1 \times \ldots \times 1}_{\frac{d}{2} \text{ vezes}}.$$

Para d ímpar (e diferente de 1), sobra um dígito no centro, que pode assumir qualquer valor. O número de palíndromos é então contado a partir do cálculo abaixo:

$$9 \times \underbrace{10 \times 10 \times \ldots \times 10}_{\frac{d-1}{2}-1 \text{ vezes}} \times 10 \times \underbrace{1 \times 1 \times 1 \times \ldots \times 1}_{\frac{d-1}{2} \text{ vezes}}.$$

2.2 Contagem de palíndromos menores que N

Como determinado no Problema 1, é preciso contar quantos palíndromos menores que um número N existem. O Algoritmo 1 resolve o Problema 2 e ajuda nessa tarefa, porque ele resolve parte do problema inicial, restando apenas um outro problema, mais específico:

Problema 3 *Contar quantos palíndromos de D dígitos menores do que N existem, sendo D o número de dígitos de N.*

Antes de ver o algoritmo que resolve esse problema, deve-se compreender algumas informações preliminares, narradas a seguir.

Assuma que N tenha mais de um dígito. Separa-se N em duas metades, $\boldsymbol{M_a}$ e $\boldsymbol{M_b}$, usando-se as fórmulas:

$$M_a = \frac{N}{10^{(D+1)/2}}$$
$$M_b = N\%(10^{D/2})^{2}$$

Por exemplo,

- Para $N = 123456$,
  $M_a = 123$ e $M_b = 456$.
- Para $N = 1234567$,
  $M_a = 123$ e $M_b = 567$.

Os números M_a e M_b serão chamados respectivamente de **primeira metade** e **segunda metade** de N.

Para D ímpar, essas metades não incluem o dígito da posição central (chamado doravante de "**dígito do meio**").

Neste texto, se um número A for formado pelos dígitos de um outro número B na ordem reversa, diz-se que "$\boldsymbol{A}$ **é o reflexo de** $\boldsymbol{B}$", e escreve-se $\boldsymbol{A = \overleftrightarrow{B}}$. Por exemplo, 123 é o reflexo de 321.

Aqui também, como na Seção 2.1, os passos para a contagem variarão um pouco dependendo da paridade de D. Os dois casos aparecem separadamente a seguir.

Caso 1: D par

Considere a primeira metade, M_a, de N. Pode-se seguramente contar todos os palíndromos cuja primeira metade seja *menor* do que M_a, porque é sabido que eles são menores do que N. Para saber quantos palíndromos existem nesse intervalo (sempre considerando apenas números de D dígitos), basta perceber que todos os palíndromos têm a primeira metade igual ao reflexo da segunda metade. Assim, escolhida a primeira metade, há apenas uma maneira de escolher a segunda. Ou seja, basta contar de quantas formas a primeira metade pode ser escolhida.

2 O operador % é comumente usado em Programação. A operação $A\%B$ significa "o resto da divisão de A por B".

Agora, falta saber quantos são os números menores que M_a que têm *o mesmo número de dígitos* de M_a. Para isso, basta calcular o número $M_a - 10^{D_a-1}$, onde D_a é o número de dígitos de M_a. Por exemplo, para $N = 421555$, deve-se incluir na contagem os números 100001, 101101, 102201, ..., 419914 e 420024, que são $M_a - 10^{D_a-1} = 421 - 100 = 321$ números. (Ainda não foi contado o número 421124; isso será feito no parágrafo a seguir.)

Contou-se, até aqui, todos os palíndromos cuja primeira metade é no máximo $M_a - 1$, porque é sabido que todos eles são menores do que N. Se a primeira metade de um número for *igual* a primeira metade de N (rotulada M_a) isso pode não ser verdade. Novamente: todos os palíndromos têm a primeira metade igual ao reflexo da segunda metade. Assim, só é contado o palíndromo cuja primeira metade é M_a se o reflexo de M_a for menor do que segunda metade de N. No exemplo do número 421555, deve-se contar o número 421124 porque $\overleftrightarrow{421} = 124 < 555$. Se fosse maior, o palíndromo gerado dessa forma seria maior do que N, e não seria desejável contá-lo.

Caso 2: D ímpar

Raciocínio similar ao Caso 1 é aplicado na primeira parte deste.

Considere a primeira metade, M_a, de N. Pode-se, também aqui, contar todos os palíndromos cuja primeira metade seja menor do que M_a, porque todos os números cuja primeira metade é menor do que M_a são menores do que N.

Para saber quantos palíndromos existem nesse intervalo, novamente pode-se usar o fato de todos os palíndromos terem a primeira metade igual ao reflexo da segunda metade. Assim, escolhida a primeira metade, há apenas uma maneira de escolher a segunda.

Todavia, agora, o número de dígitos é ímpar, o que significa que há um dígito a mais para se escolher: o dígito do meio, que não está incluso nas metades. Essa escolha pode ser feita de 10 formas, pois qualquer dígito serve para obter um número palíndromo.

Agora, veja o que ocorre para os palíndromos cuja primeira metade é *igual* a M_a.

Seja M o dígito do meio de N.

Pode-se seguramente contar os palíndromos cuja primeira metade seja M_a e cujo dígito do meio estiver entre 0 e $M - 1$, pois, de fato, são todos menores que N. Por exemplo, para $N = 4443111$, pode-se incluir na contagem 4440444, 4441444 e 4442444, mas não 4443444, por este último ser maior que N.

Entretanto, se o reflexo de M_a for menor que M_b, pode-se contar também o palíndromo cuja primeira metade é M_a e cujo dígito do meio é M. É o caso de $N = 4443888$, onde, aí sim, deve-se incluir o número 4443444, desprezado no parágrafo anterior.

Toda essa análise está sintetizada no Algoritmo 2, que apresenta uma maneira de contar todos os palíndromos menores que N de D dígitos.

No Algoritmo 2, usa-se o ponto como **operador de concatenação**. Assim, $\boldsymbol{A.B}$ significa "o número formado pela concatenação dos dígitos de A com os de B. Por exemplo, se $A = 123$ e $B = 89$, então $A.B = 12389$.

Se $D = 1$, não use o Algoritmo 2: esse é o único caso em que não se pode dividir N em duas metades. Mas também é um caso óbvio: se N tem apenas um dígito, todos os números menores que N são palíndromos. O Algoritmo 3 trata isso em suas primeiras linhas, quando retorna $N - 1$ se $N \leq 9$.

O Algoritmo 3, escrito em C++, faz exatamente o que descreve o Algoritmo 2, exceto que ele usa fórmulas (apresentadas anteriormente neste texto) para contar os números em vez de contar um por um.

Algoritmo 2 – Conta Palíndromos
Conta os palíndromos menores que N que têm D dígitos

```
1  função Conta Palíndromos(N)
2      D ← quantidade de dígitos de N;
3      M_a ← primeira metade de N;
4      M_b ← segunda metade de N;
5      D_a ← quantidade de dígitos de M_a;
6      M_0 ← menor inteiro de D_a dígitos;[a]
7      Se D for par,
8          Para cada m_a de M_0 até M_a − 1,
9              Conte o número m_a.reverso(m_a).
10         Se reverso(M_a) for menor que M_b
11             Conte o número M_a.reverso(M_a).
12     Se D for ímpar,
13         Para cada m_a de M_0 até M_a − 1,
14             Para cada c de 0 a 9,
15                 Conte o número m_a.c.reverso(m_a).
16         M ← dígito do meio de N;
17         Para cada m de 0 até M − 1
18             Conte o número M_1.m.reverso(M_1);
19         Se reverso(M_a) for menor que M_b
20             Conte o número M_a.M.reverso(M_a).
```

Algoritmo 3 – contaPaliDigsMenor
Calcula a quantidade de palíndromos menores que N que têm D dígitos

```
int contaPaliDigsMenor (int N)
{
  if (N <= 9)
    return N-1;
  int D = digs(N);
  int Ma = N/(pow(10,(D+1)/2));
  int Mb = N%(pow(10,D/2));
  int Da = digs(Ma);
  int M0 = pow(10,Da-1);

  int c;

  if (D%2==0) {
    c = (Ma-1)-M0 +1;
    if (reflexo(Ma) < Mb) {
      c++;
    }
  }
  else {
    c = ((Ma-1)-M0 +1)*10 + (M-1) +1;
    int M = (N/pow(10,D/2))%10;
    if (reflexo(Ma) < Mb) {
      c++;
    }
  }
  return c;
}
```

[a] O menor inteiro de 1 dígito é 1, e não 0.

Os Algoritmos 4 e 5 são usados pelo Algoritmo 3 para encontrar o reflexo de um número e para encontrar a quantidade de dígitos de um número, respectivamente.

A função pow(b, e), usada nos Algoritmos 1 e 3, retorna o número b^e.

Algoritmo 4 – reflexo
Retorna o reflexo de n

```
int reflexo (int n)
{
  int d = digs(n);
  int r = 0;
  while (n>0) {
    r += n%10*pow(10,d-1);
    d--;
    n = n/10;
  }
  return r;
}
```

Algoritmo 5 – digs
Retorna a quantidade de dígitos de n

```
int digs (int n)
{
  int c = 0;
  while (n>0) {
    c++;
    n = n/10;
  }
  return c;
}
```

2.3 Resumo da solução

- O problema principal foi reduzido ao Problema 1: escrever um algoritmo que diga quantos palíndromos menores do que um número N existem;
- O Problema 1 foi dividido em duas outras partes:
  - Problema 2: Contar a quantidade de palíndromos de d dígitos que existem;
  - Problema 3: Contar a quantidade de palíndromos de D dígitos menores do que N existem, sendo D o número de dígitos de N.
- O Problema 2 foi resolvido com o Algoritmo 1;
- O Problema 3 foi resolvido com o Algoritmo 3.

De posse dos algoritmos que resolvem os Problemas 2 e 3, pode-se resolver o Problema 1 como descreve o Algoritmo 6.

Algoritmo 6 – **contaPali**
Conta quantos palíndromos menores que N existem

```
1 função contaPali(N)
2     c ← 0;
3     D ← quantidade de dígitos de N;
4      Para d de 1 até D − 1
5         c ← c + contaPaliDigs(d);
6     c ← c + contaPaliDigsMenor(N);
7     Retorna c.
```

A função principal da solução é mostrada no Algoritmo 7. Como as soluções desenvolvidas contam palíndromos *menores* que um determinado número e o enunciado do problema pede que eles sejam contados em um *intervalo fechado* $[A, B]$, deve-se fazer $c_A =$ contaPali(A) para contar todos os palíndromos *menores* que A e $c_B =$ contaPali$(B + 1)$ para contar todos os palíndromos *menores ou iguais* a B. Depois, faz-se $c_B - c_A$ para contar todos os palíndromos menores ou iguais a B que não são menores do que A, ou seja, que são maiores ou iguais a A, estando portanto no intervalo fechado $[A, B]$.

Algoritmo 7 – **main**
Função principal

```
1 função main
2     Lê dois inteiros A e B;
3     c_A = contaPali(A);
4     c_B = contaPali(B + 1);
5     Informa o valor c_B − c_A.
```

3 Eficiência

Nesta seção, cada função utilizada será analisada para, no fim, saber qual é o custo de tempo do programa no pior caso.

A palavra "operação" será usada para referir-se a qualquer operação de adição, subtração, multiplicação, divisão ou comparação de números inteiros menores do que 10^9.

3.1 Tempo de execução das funções

3.1.1 Função digs

A função digs, descrita no Algoritmo 5, faz duas operações para cada dígito do número em seu argumento. Como esse número não é maior do que N, o número máximo de operações é igual a

$$2D.$$

3.1.2 Função pow

A função pow, descrita no Algoritmo 8, faz duas operação para cada unidade do expoente. Como essa função nunca é chamada com expoente maior que D, o número máximo de operações é igual a

$$2D.$$

3.1.3 Função contaPaliDigs

A função contaPaliDigs, descrita no Algoritmo 1, faz 4 operações e chama a função pow com expoente nunca maior que $\frac{D+1}{2} - 1$. Somando $4 + 2 \times (\frac{D+1}{2} - 1)$, o total de operações fica

$$D + 3.$$

Algoritmo 8 – POW
Retorna b^e

```
1  int pow (int b, int e)
2  {
3    int r = 1;
4    while (e--)
5      r *= b;
6    return r;
7  }
```

3.1.4 Função REFLEXO

A função REFLEXO, descrita no Algoritmo 4, faz:

- 1 chamada à função DIGS: $1 \times 2D$;
- D chamadas à função POW: $D \times 2D$;
- D vezes um conjunto de 5 operações.

Isso totaliza

$$2D^2 + 7D.$$

3.1.5 Função CONTAPALIDIGSMENOR

Para analisar esta função, considerar-se-á apenas o caso D ímpar, que é nitidamente o caso onde há mais operações. A função CONTAPALIDIGSMENOR, descrita no Algoritmo 3, faz, para $D > 1$ ímpar:

- 1 chamada à função DIGS: $2D$;
- 4 chamadas à função POW: $4 \times 2D$;
- 2 chamadas à função REFLEXO(n) (com $n \leq \frac{D}{2}$): $2 \times \left(2\left(\frac{D}{2}\right)^2 + 7\left(\frac{D}{2}\right)\right)$;
- Não mais que 20 outras operações.

Isso totaliza

$$D^2 + 17D + 20.$$

3.1.6 Função CONTAPALI

A função CONTAPALI, descrita no Algoritmo 6, faz:

- 1 chamada à função DIGS: $1 \times 2D$;
- Não mais que D chamadas à função CONTAPALIDIGS: $D \times (D + 3)$;
- 1 chamada à função CONTAPALIDIGSMENOR: $1 \times (D^2 + 17D + 20)$;
- Não mais que $2D$ outras operações.

Isso totaliza

$$2D^2 + 24D + 20.$$

3.1.7 Total

Na função principal MAIN, a função CONTAPALI é chamada duas vezes (uma para cada número da entrada). Assim, o número máximo de operações será

$$4D^2 + 48D + 40. \tag{1}$$

Para as restrições de entrada impostas na descrição do problema, D jamais passa de 10 (porque N não pode ser maior que 10^9), e então número de operações para cada caso de entrada nunca será maior do que 1000.

3.2 Complexidade

Como $D = \lfloor \log_{10} N \rfloor + 1$,[3] pode-se concluir do valor em (1) que o programa é de complexidade $O(\log^2 N)$.

3 $\lfloor \log_{10} N \rfloor$ quer dizer "o piso de $\log_{10} N$", o maior inteiro menor ou igual a $\log_{10} N$.

3.3 Otimizações possíveis

Pode-se facilmente modificar o programa para que ele armazene alguns valores frequentemente calculados. Dessa forma, é possível aumentar a rapidez da execução.

Um pré-processamento que pode ser feito é o das saídas do Algoritmo 1 (contaPaliDigs) para d de 1 a 9, que serão os únicos casos necessários dentro dos limites da entrada.

Também pode-se fazer o mesmo para a função pow, que é chamada sempre com base igual a 10 e expoente menor que D.

Se os valores de retorno das funções pow e contaPaliDigs fossem calculados e armazenados previamente (de modo que cada uma delas passasse a ter complexidade $O(1)$), em vez do valor em (1), o número total de operações seria

$$46D + 24, \tag{2}$$

e isso significa que o programa passaria a ter complexidade $O(\log N)$, e que para o maior caso dentro das restrições do problema, seriam gastos menos de 490 operações.

Para os limites impostos, o tempo de execução das duas soluções não é tão discrepante (o que é esperado, já que $\log N$ e $\log^2 N$ não divergem de maneira rápida entre si). A Figura 1 compara os valores 1 e 2.

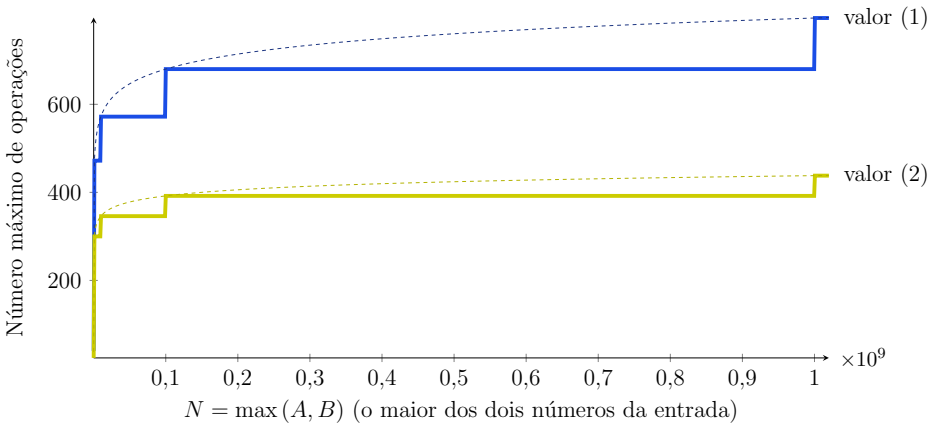

Figura 1: Tempo esperado de execução do programa, comparando os valores (1) e (2). Para traçar as linhas contínuas, foi usado $D = \lfloor \log_{10} N \rfloor + 1$, que é igual ao número de dígitos de N. As linhas tracejadas foram traçadas considerando-se $D = \log_{10} N + 1$, e servem para dar melhor noção sobre o crescimento das funções.

3.4 Soluções menos eficientes

Se for elaborado um algoritmo que teste cada número, a complexidade será $O(N \log^2 N)$ (ou $O(N \log N)$, se houver pré-processamento), porque, além de percorrer cada número até N, também será necessário analisar os dígitos de cada número.

Se a função contaPaliDigs não for usada e for substituída pela contaPaliDigsMenor (por exemplo, para contar quantos palíndromos de três dígitos existem, é possível chamar contaPaliDigsMenor(999) em vez de contaPaliDigs(3)) obteria-se uma solução em $O(\log^3 N)$.

4 Nível de Dificuldade

A maior parte da dificuldade do problema cai sobre:

1. Perceber que o problema deve ser resolvido com Combinatória;

2. Perceber que o problema é simplificado se forem contados os palíndromos de 1 até A e de 1 até $B + 1$, em vez de contar diretamente os de A até B;
3. Perceber que se deve contar, separadamente, quantos palíndromos com cada quantidade de dígitos existem;
4. Escrever um algoritmo que conte quantos palíndromos de d dígitos existem;
5. Escrever um algoritmo que retorne o reflexo de um número;
6. Escrever um algoritmo que conte quantos palíndromos de D dígitos menores ou iguais a N existem.

Uma pessoa que tenha noções de Combinatória consegue facilmente contar quantos palíndromos contendo um determinado número de dígitos existem. Portanto, o item 4 seria facilmente resolvido.

Os itens 2 e 5 são supostamente realizados sem dificuldade alguém que tenha suficiente prática de programação.

O item 3 requer mais atenção. Um equívoco possível seria tentar contar, de uma só vez, todos os palíndromos que tenham até d dígitos assumindo que o primeiro dígito da esquerda pode ser 0. Tal contagem não fornece o valor correto porque, para $d = 6$, estaria incluso na contagem, por exemplo, o número $003300 = 3300$, que não é palíndromo.

O item 1 pode variar muito de indivíduo para indivíduo, já que pode depender das referências que ele possui. Mas uma pessoa que tenha familiaridade com Combinatória pode deduzi-lo imediatamente.

Uma vez deduzidos os itens 1 e 3, não deve haver dificuldades para se concluir o item 4.

O item 6 é certamente o mais trabalhoso, e pode ser local de algumas tentativas frustradas.

Para concluir, nota-se os seguintes pontos que facilitam a resolução do problema:

- As técnicas de contagem usadas podem ser deduzidas mesmo sem grande conhecimento teórico;
- A solução não exige técnicas avançadas de programação;
- Pode-se usar apenas variáveis simples para implementar o programa.

E os pontos que podem dificultar a resolução:

- A solução do problema depende de várias observações que não podem ser ignoradas para que ela funcione corretamente;
- Requer-se um bom tempo de formulação antes da implementação da solução;
- A solução é composta de várias partes;
- Pode haver problemas ao definir-se bem as partes da solução, e fazê-las operar em conformidade;
- Pode ser trabalhoso e demorado localizar erros e corrigi-los.

Tendo como referência os itens listados acima, o problema pode ser classificado como nível médio.

5 Conclusão

O problema que foi apresentado neste documento tem as seguintes características:

- **Categoria**: Matemática →Aritmética →Combinatória.
- **Dificuldade**: Médio.
- **Complexidade esperada**: O($\log^2$N).

Sua solução é simplificada se for quebrada em passos de contagem, e contém várias exceções que devem ser tratadas. Um competidor teria grande vantagem se observasse isso antes de começar a implementar o programa.

É esperado que uma pessoa que tenha experiência com Análise Combinatória consiga obter uma solução sem grandes problemas e uma pessoa que apenas tenha uma vaga noção dessa área consiga, depois de alguns testes, deduzir a solução.

Sobre este trabalho

Este documento foi originalmente submetido como trabalho avaliativo da disciplina de Desafios de Programação, ministrada pelo professor Anderson de Araújo na Faculdade de Computação da Universidade Federal de Mato Grosso do Sul, em Campo Grande, em março de 2016. A versão que você tem em mãos foi revisada e publicada em raphael.neocities.org em novembro de 2016. A data da revisão mais recente está na primeira página.

Informações adicionais

A folha de descrição do problema, uma solução em C++ e arquivos de teste podem ser encontrados em raphael.neocities.org/portpholio/contapali.

Sobre o autor

Raphael Rocha da Silva é formado em Engenharia de Computação pela Universidade Federal de Mato Grosso do Sul, tendo concluído o curso em setembro de 2016. Mais informações em raphael.neocities.org.

Agradecimentos

- Ao amigo Vinícius Candia, colega de olimpíada de Matemática, que leu este artigo e fez sugestões de revisão. Também dele partiu a ideia, há alguns anos, de fazer um programa que conta a quantidade de palíndromos entre dois números.
- Ao amigo Lucas Tsutsui, colega de faculdade, cobaia deste problema, que desenvolveu uma solução a partir da folha de descrição que eu o enviei, o que me permitiu saber se o problema estava formulado adequadamente e resolvido corretamente por mim, além de saber qual caminho uma pessoa como ele, com muita experiência em maratonas de programação, tenderia a seguir. (A propósito, a solução dele ficou muito parecida com a mostrada neste texto.)
- À amiga Andressa Sousa, colega de olimpíada de Matemática, que respondeu a meus pedidos de sugestões e ajudou a batizar alguns termos da nomenclatura adotada neste trabalho.

Este texto foi produzido em LaTeX, com o pdfTeX versão 3.14159265-2.6-1.40.16 (TeX Live 2015/Debian) e uso dos pacotes Algorithm, Listings, Algpseudocode, TikZ, PGFPlots e outros. 0312162008