Hélio Hideki Assakura Moreira

MAC 0122 – EP1

Descobrindo os números

Instituto de Matemática e Estatística - USP

São Paulo – 2014

Objetivo do EP

Dado um número **n**, $0 \leq n \leq 2000000000$, achar o menor inteiro **k** tal que os primeiros dígitos de **2^k** sejam **n**. Porém, mais da metade dos dígitos à direita de **2^k** serão desconsiderados. A entrada **n = 0** indica o final do programa.

Conceitos matemáticos usados:

Partindo da seguinte desigualdade, temos:

$$\mathbf{n * 10^m < 2^k < (n+1) * 10^m}$$

Sendo **m** a quantidade de dígitos de **n**, somado 1, garantindo que **n * 10^m** seja o menor número que contém os primeiros dígitos entrados e que mais da metade dos dígitos tenham sido descartados, e **n+1** seja o próximo número, fazendo com que **2^k**, caso exista, esteja entre eles. Aplicando log na base 10:

$$\log_{10} n + m < k * \log_{10} 2 < \log_{10}(n+1) + m$$

Dividindo por $\log_{10} 2$:

$$(\log_{10} n + m) / \log_{10} 2 < k < (\log_{10} (n+1) + m) / \log_{10} 2$$

Com essa desigualdade, garantimos que $(\log_{10} n + m) / \log_{10} 2$ será sempre menor que **k**, e que $(\log_{10} (n+1) + m) / \log_{10} 2$ será sempre maior que **k**. Assim, ao pegarmos a parte inteira de cada termo, $(int)\ (\log_{10} n + m) / \log_{10} 2 < (int)\ (\log_{10} (n+1) + m) / \log_{10} 2$. Logo, o programa deverá procurar números, **somando 1 em m para cada verificação**, até que a diferença entre a parte inteira dos números seja diferente de 0. Isso significará que a desigualdade é verdadeira, e que **k** estará entre eles. Para isso, devemos achar o **m** inicial. Para isso foi criada a função *digitos,* que retorna a quantidade de dígitos do **n** inserido pelo usuário. Para garantir que pelo mais da metade dos dígitos foi retirada, é acrescido 1 nesse valor.

Problemas

Devido à quantidade de casas decimais a ser considerada na conta, podemos ter alguns erros de aproximação, influenciando no problema quando **n** e **m** chegam a valores altos.

Tempo de execução

Para valores pequenos, como os exemplos dados no enunciado do problema (“Para n = 65 seu programa deverá imprimir 16 pois 2^16 = **65**536. Para n = 10 seu programa deverá imprimir 20 pois 2^20 = **10**48576.”), a execução é rápida.

Porém, para valores mais altos, o tempo começa a ficar considerável. Ele depende do valor de **m**. Foi usado o comando *time* ***(time ./ep1)*** e compilado com ***gcc –Wall –std=c99 –pedantic –O3 –o ep1 ep1.c -lm***.

```
Insira um numero n, 0 <= n <= 2000000000

10
20

real	0m3.229s
user	0m0.000s
sys	0m0.001s
```

```
Insira um numero n, 0 <= n <= 2000000000

1999999999
1923400331

real	0m43.911s
user	0m36.073s
sys	0m0.054s
```

```
Insira um numero n, 0 <= n <= 2000000000

65
16

real	0m1.521s
user	0m0.001s
sys	0m0.001s
```

```
Insira um numero n, 0 <= n <= 2000000000

2000000000
7693601321

real	2m29.434s
user	2m24.217s
sys	0m0.772s
```

```
Insira um numero n, 0 <= n <= 2000000000

123456789
62759188

real	0m7.870s
user	0m1.221s
sys	0m0.003s
```

```
Insira um numero n, 0 <= n <= 2000000000

999999999
1923400330

real	0m47.374s
user	0m36.019s
sys	0m0.047s
```

Entradas e saídas do programa

```
Insira um numero n, 0 <= n <= 2000000000

65
k:	16
m:	4

real	0m0.965s
user	0m0.001s
sys	0m0.001s
```

```
Insira um numero n, 0 <= n <= 2000000000

123456789
k:	62759188
m:	18892391

real	0m7.717s
user	0m1.192s
sys	0m0.002s
```

```
Insira um numero n, 0 <= n <= 2000000000

10
k:	20
m:	6

real	0m4.568s
user	0m0.001s
sys	0m0.001s
```

```
Insira um numero n, 0 <= n <= 2000000000

999999999
k:	1923400330
m:	579001185

real	0m42.389s
user	0m35.481s
sys	0m0.036s
```

Entradas e saídas do programa, com m

O maior valor de n, 0 ≤ n ≤ 2000000000

Para os valores de **n** propostos no problema, temos que o maior valor possível é 2000000000. Porém, o tempo que leva para calculá-lo é bem maior que do restante, pois o valor de **m** é também maior. Para esse valor, temos o tempo entre **2m20s** e **2m25s**.

```
Insira um numero n, 0 <= n <= 2000000000

2000000000
k:      7693601321
m:      2316004764

real    2m26.599s
user    2m21.962s
sys     0m0.157s
```

```
Insira um numero n, 0 <= n <= 2000000000

2000000000
k:      7693601321
m:      2316004764

real    2m25.048s
user    2m20.664s
sys     0m0.422s
```

```
Insira um numero n, 0 <= n <= 2000000000

2000000000
k:      7693601321
m:      2316004764

real    2m26.876s
user    2m22.680s
sys     0m0.160s
```

```
Insira um numero n, 0 <= n <= 2000000000

2000000000
k:      7693601321
m:      2316004764

real    2m27.720s
user    2m23.149s
sys     0m0.839s
```

Valores maiores que n

O programa também consegue calcular o **k** para valores além dos propostos pelo programa. O fator limitante é o tamanho que **m** chega, além da aproximação da expressão usada para calcular o **k**.

```
Insira um numero n, 0 <= n <= 2000000000

4123456789
k:      4188118034
m:      1260749145

real    1m20.956s
user    1m17.832s
sys     0m0.102s
```

```
Insira um numero n, 0 <= n <= 2000000000

6123456789
k:      5710148977
m:      1718926113

real    1m49.499s
user    1m46.680s
sys     0m0.133s
```

```
Insira um numero n, 0 <= n <= 2000000000

5123456789
k:      1695314816
m:      510340603

real    0m37.035s
user    0m32.075s
sys     0m0.123s
```

```
Insira um numero n, 0 <= n <= 2000000000

7123456789
k:      797819235
m:      240167512

real    0m17.165s
user    0m14.655s
sys     0m0.020s
```