Uma conjectura de Duro Kurepa

Praciano-Pereira, T

28 de maio de 2024
preprints da Sobral Matemática
no. 2024.02
Editor Tarcisio Praciano-Pereira
tarcisio@sobralmatematica.org

Resumo

Uma conjectura de Duro Kurepa citada no livro 1001 problems of classical number theory, [DM07], pede como solução um programa para verificar a conjectura até 1000. Eu escrevi e testei o programa como solicitado até 1000 com resultado positivo depois de 4 milisegundos. Neste artigo estou apresentando dois programas, um escrito em `python` e outro escrito em `calc` para atender ao pedido do exercício mencionado.

palavras chave: conjectura de Duro Kurepa, programa em calc, programa em python,

A conjecture of Duro Kurepa cited in the AMS book "1001 problems of number theory", [DM07], asks for a program to verify the conjecture up to 1000. I wrote the program and the verification up to 1000 took real, 0m39,058s, user 0m39,053s, sys 0m0,004s. I have left a computer running to verify up to 1,000,000 which is the roof of verifications stated in the book. The program is presented in the first section below. I wrote two versions of the program, one in python3 and the other in calc, both presented here.

keywords: calc program, python program, conjecture of Duro Kurepa

*tarcisio@sobralmatematica.org

1

No livro "1001 problems in number theory" esta concjectura aparece como problema 37 na página 19. Primeiro vem a definição de $!n$ como a soma $!n = 0! + 1! \cdots + (n-1)!$ em seguida os autores fazem o desafio que se escreva um programa que teste a conjectura apenas até 1000. Eu escrevi dois programas, um em `calc` e outro em `python` que eu apresento na próxima seção. O programa em `python` se revelou quatro vezes mais efetivo do que o program em `calc` que demorou `user 0m39,053s` o tempo do usuário enquanto que o programa em `python` apontou o tempo do usuário praticamente igual ao do sistema `sys 0m0,004s`.

Em seguida rodei o programa com $n = 1000000$ e depois de 24 horas o programa ainda continua rodando num microcomputador que está sendo usado exclusivamente para este programa ....

Este teste já foi feito, segundo os autores, por dois matemáticos que acho que são poloneses, Ivic e Mijajlovic, e eu estou apenas, por curiosidade, tentando fazer o mesmo num micro de classe corriqueira.

Um teste destes é apenas uma curiosidade, nada que sugira que se pode obter uma demonstração da conjectura com o sucesso deste teste. O que tem de interessante é a capacidade destas duas linguagens de programação, `python` e `calc` de conseguirem efetuar o teste. `calc` é uma linguagem de programação produzida por um grupo que se dedica a problemas da Teoria dos Números e, o *meu programa* se viu menos efetivo do que o o *meu programa* escrito em `python` e observo isto para deixar claro que o meu resultado não pode indicar que a linguagem `calc` é menos efetiva do que a outra. Ambas operam com "*números naturais de qualquer tamanho*", obviamente a capacidade final dos programas depende das possibilidades do processador e da quantidade de memória da máquina, ambos dados cruciais para um desempenho efetivo dos programas. A máquina em que deixei rodando o teste até 1000000 tem apenas 4Gb de memória ainda que tenha 45Gb de área de troca em em uma partição separada do disco, a memória RAM é mais efetiva o que justifica que depois de 24 horas o resultado ainda não tenha sido obtido. Eu vou re-editar este artigo quando a máquina terminar o trabalho ...

2 Descrevendo os programas

Vou começar mostrando programa escrito em `python` por ser esta uma linguagem mais conhecida e usada.

```
from math import gcd;
def fact(n):
                    if (n==0):return 1;
                    else:
                                                    return n*fact(n-1);
def Invfact(n):
                    if(n==1): return 1;
                    else: return Invfact(n-1)+fact(n-1);
def Main(n):
                    I=range(0,n);
                    for k in I:
                                                    if (gcd(Invfact(k),fact(k)) ¿= 2):
                                                    print(k,gcd(Invfact(k),fact(k)));
                    print("Terminou!!
n")
                    print(k); print(gcd(Invfact(k),fact(k)));

Main(1000);
```

Eu também programo em `C++` de onde trago o hábito de criar uma função `Main()` que é a função que executa as tarefas, as outras funções, é função que gerencia um programa. `python` é uma linguagem muito efetiva porque ela tem um pequeno módulo principal que pode ser expandido com o objetivo do programador. Assim a primeira linha do programa importa para núcleo da linguagem a função `gcd`, "*greatest common divisor*", o nosso *MDC*. Esta é uma diferença, possivelmente a mais importante com a linguagem `calc` que é especializada em Teoria dos Números e muito usada na busca do maior número primo então ela já traz incorporada todas as funções mais usadas em Teoria dos Números e possivelmente é isto que justifica a diferença em tempo de processamento dos dois programas que eu escrevi.

Apresento agora o programa em `calc`. Na verdade, primeiro eu escrevi o programa em `calc` e depois a tradução para `python` depois que rodei e testei o programa em `calc`.

```
define Invfact(n) {local soma = 0,k=0;
if(n==1) return 1;
else return Invfact(n-1)+fact(n-1);
}

define Main(n) {local k=0;
for(k=0;k¡n;k++)
        if (gcd(Invfact(k),fact(k)) ¿= 2);
                print k,gcd(Invfact(k),fact(k));
print('k= ',k, 'gcd(Invfact(k),fact(k)) = ', gcd(Invfact(k),fact(k)));
}

Main(1000);
```

Há pequenas diferenças entre os dois programas. Primeiro é que `gcd` é nativo do `calc` o que provavelmente justifica a diferença de desempenho neste caso, `calc` tem no seu *núcleo*

praticamente tudo que for necessário para trabalhar com *Teoria dos Números*, enquanto que `python` tem um *núcleo menor* que você pode especializar de acordo com suas necessidades.

Tudo que eu escrevo é de domínio público e portanto você pode usar livremente os meus programas, obviamente, citar o autor é forma de uso *cidadão do conhecimento*, mas estou exposto ao plágio da famigerada *inteligência artificial*, infelizmente.

Primeiro eu tentei testar a conjectura de *Duro Kurepa* com `geogebra`, mas o fatorial do `geogebra` está limitado a valores de $n \in \mathbf{N}$ menores que 1000, foi quando sai para `calc` e depois para `python`.

Se você conseguir otimizar algum destes programas eu ficaria feliz recebendo uma cópia.

3 Agradecimentos

O meu colega no curso de *geogebra* Erick Rodolfo Souza Trindade, observou erros nos meus programas e mostrou-me a correção com um método do *geogebra* o que agradeço. Os programas foram corrigidos a partir da observação que ele me fez.

referências

[DM07] Jean-Marie De Koninck e Armel Mercier. *1001 problems in classical Number Theory.* 2007.

]