Concorrência das IA´s na Programação

Mário Leite

…

Eu já havia feito uma postagem abordando esse tema, mas devido à importância deste assunto é importante relembrar e discutir o "***medo dos programadores com a concorrência das IA's***". Esse "medo" pode até assombrar e se tornar uma realidade se não nos informarmos melhor sobre o cenário atual do desenvolvimento de sistemas. Eu digo e afirmo: "*uma IA pode codificar melhor que um programador, mas não pode programar*".

Existe uma diferença fundamental entre "codificar" e "programar":

- **Codificar**: Basicamente, é transformar ideias ou algoritmos em códigos que uma máquina entende. A IA hoje pode fazer isto muito bem: gerar código, corrigir sintaxe, sugerir trechos, até refatorar funções; isto é a parte mecânica (braçal) da programação que será cada vez mais automatizada.
- **Programar:** É criar a estratégia, a lógica e o raciocínio para resolver um problema específico apresentado pelo usuário. Isto requer entendimento profundo do problema, criatividade e visão de alto nível. Por exemplo: decidir que tipo de algoritmo usar para gerar primos, ou otimizar busca, ou construir um modelo de dados complexo. Essa parte é difícil de substituir, porque envolve intuição humana, experiência e *insight*; coisas que uma IA (ainda) não domina plenamente.

Mesmo com IA's avançadas (generativas) o papel do programador é fundamental, pois:

- Elas não substituem a criatividade que projeta soluções.
- IA é uma ferramenta que acelera o trabalho, mas não cria soluções novas do zero sem orientação humana.
- Programadores que sabem pensar em algoritmos, otimização e *design* de sistemas continuarão essenciais.

As IA's só podem:

- **Gerar código** automaticamente a partir de uma descrição (bem específica) do problema.
- **Corrigir erros de sintaxe** ou sugerir melhorias em funções existentes.
- **Automatizar tarefas repetitivas**, como validações, CRUDs ou *scripts* simples.

Portanto, em termos de codificação pura, sim: a IA pode acelerar muito o trabalho; mas isso é apenas a parte mecânica da programação; não é a criação da solução do problema. Para demonstrar isto vamos criar um exemplo concreto mostrando como o programador decide a lógica do algoritmo para encontrar **N** números primos usando a estratégia **6k ± 1**, com códigos em **Visualg** e **Python**. Assim, após verificar divisibilidade por **2** e **3**, apenas números desta forma são testados; isto reduz drasticamente o número de divisões para números grandes. O programador decidiu essa lógica; a IA poderia gerar código de primalidade, mas talvez não escolhesse a otimização **6k ± 1** sozinha.

Conclusão: A IA não substitui a análise e conhecimento humano e, consequentemente, o melhor algoritmo a ser usado. O medo de substituição dos programadores pelas IA's é exagerado se considerarmos apenas resolução de problemas complexos; para codificação rotineira, sim: IA pode fazer grande parte do trabalho. Mas o cerne da programação - criar algoritmos eficientes e inteligentes - ainda depende do humano. Pela **figura 1** a linha azul (método simples): número de divisões cresce rápido com primos maiores; a linha laranja (**6k ± 1**) mostra um crescimento muito mais lento, mostrando eficiência superior; visualmente, fica claro que a escolha do algoritmo tem grande impacto, algo que um programador experiente pode decidir. A **tabela 1** apresenta os resultados numéricos, indicando os ganhos com o algoritmo correto. E para demonstrar que a programação (por humano) sobrepõe à codificação (por IA's), apresentamos duas soluções computadorizadas para encontrar os 1000 primeiros números primos: uma em Visualg (planejada por humano) e outra codificada por uma IA, em Python.

FINALMENTE: E se ensinarmos as IA's a usar o melhor algoritmo?! Sim; mas QUEM as ensinou?!

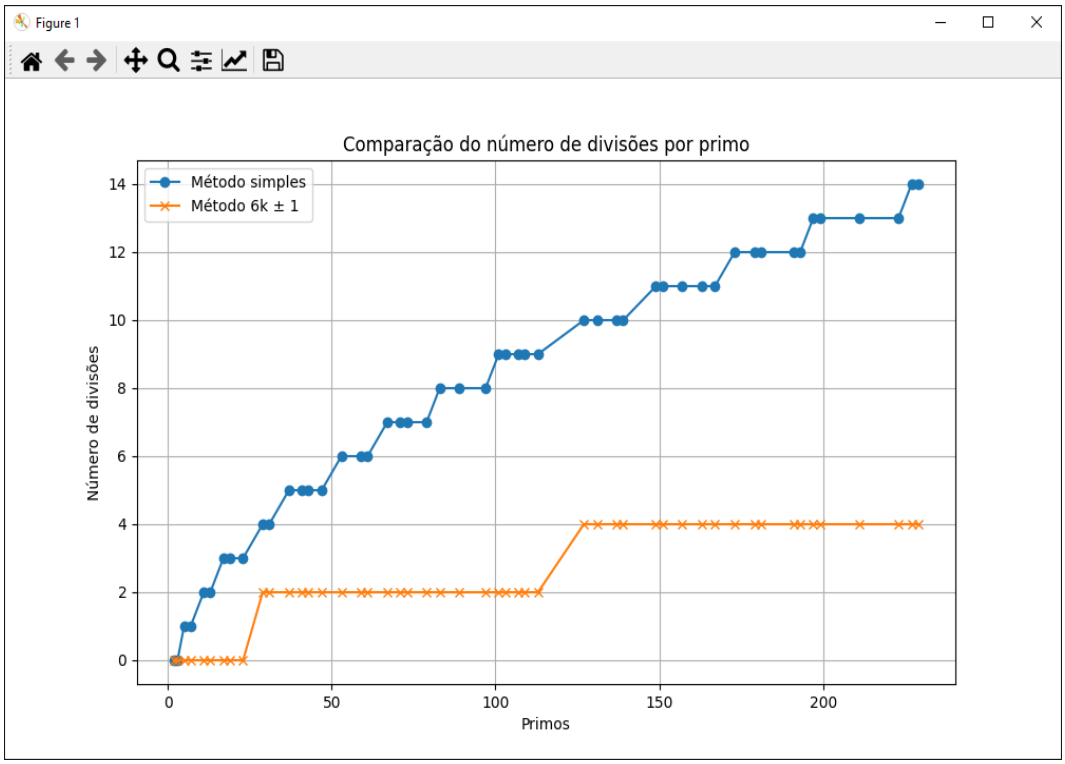

Figura 1 - Comparando Programação com Codificação

Primo	Divisões (método simples)	Divisões (6k ± 1)	Observação
2	0	0	Teste inicial
3	0	0	Teste inicial
5	1 (testa 3)	0	6k ± 1 pula 3
7	2 (3, 5)	1 (5)	Menos divisões
11	4 (3, 5, 7)	2 (5, 7)	Ganho crescente
13	5 (3, 5, 7, 9, 11)	3 (5, 7, 11)	Ganho mais evidente
17	6 (3, 5, 7, 9, 11, 13)	3 (5, 7, 11)	50% menos divisões
19	7 (3, 5, 7, 9, 11, 13, 15)	3 (5, 7, 11)	Muito mais eficiente
23	8 (até 21)	4 (5, 7, 11, 13)	Redução significativa
29	10 (até 27)	4 (5, 7, 11, 13)	Ganho aumenta com n

Tabela 1 - Tabela de comparação de divisões

```
// Primos6kmaismenos1.alg
Algoritmo "Primos6kmaismenos1"
var
    n, cont, num, limite, i: inteiro

funcao ehPrimo(num: inteiro): logico
var
    i, limite: inteiro
inicio
    Se(num < 2) Entao
        Retorne Falso
    Senao Se (num <= 3) Entao
        Retorne Verdadeiro
    Senao Se ((num mod 2 = 0) ou (num mod 3 = 0)) Entao
        Retorne Falso
    Fimse

    limite <- Trunc(RaizQ(num))
    i <- 5
    Enquanto (i <= limite) Faca
        Se ((num mod i = 0) ou (num mod (i+2) = 0)) Entao
            Retorne Falso
        FimSe
        i <- i + 6
    FimEnquanto
    Retorne Verdadeiro
fimfuncao

Inicio
    Escreva("Digite quantos números primos deseja: ")
    Leia(n)

    cont <- 0
    num <- 2

    Enquanto (cont < n) Faca
        Se(ehPrimo(num)) Entao
            Escreval(num)
            cont <- cont + 1
        FimSe
        num <- num + 1
    FimEnquanto
FimAlgoritmo  #Fim do programa "Primos6kmaismenos1.alg"
```

```
// Primos6kmaismenos1.py
import math
import matplotlib.pyplot as plt

#Função para contar divisões no método simples
def FazerDivisaoSimples(num):
    if(num < 2): return 0
    cont = 0
    for i in range(2, int(math.sqrt(num)) + 1):
        cont += 1
        if(num % i == 0):
           break
    return cont

#Função para contar divisões no método 6k ± 1
def FazerDivisao6k(num):
    if(num < 2):  return 0
    if(num <= 3): return 0
    if(num % 2 == 0 or num % 3 == 0): return 1
    count = 0
    limite = int(math.sqrt(num))
    i = 5
    while(i <= limite):
        count += 2  #i e i+2
        i += 6
    return count

# Lista de primeiros N primos
N = 50
LstPrimos = []
num = 2
while(len(LstPrimos) < N):
    ehPrimo = True
    for i in range(2, int(math.sqrt(num)) + 1):
        if (num % i == 0):
            ehPrimo = False
            break
    if(ehPrimo):
        LstPrimos.append(num)
    num += 1

#Contagem de divisões
divSimples = [FazerDivisaoSimples(p) for p in LstPrimos]
div6k = [FazerDivisao6k(p) for p in LstPrimos]

#Plotagem
plt.figure(figsize=(10,6))
plt.plot(LstPrimos, divSimples, label="Método simples", marker='o')
plt.plot(LstPrimos, div6k, label="Método 6k ± 1", marker='x')
plt.xlabel("Primos")
plt.ylabel("Número de divisões")
plt.title("Comparação do número de divisões por primo")
plt.legend()
plt.grid(True)
plt.show()
#Fim do programa "Primos6kmaismenos1.py"
```