SpeedUp

Número avaliado	É Primo?	Tempo médio de Execução do Algoritmo (milissegundos)	Desvio padrão
7	Sim	0.00	0.00
37	Sim	0.00	0.00
8431	Sim	0.06	0.02
13033	Sim	0.09	0.03
524287	Sim	3.13	0.42
664283	Não	0.00	0.00
3531271	Sim	21.32	2.89
2147483647	Sim	12503.37	833.33

Número avaliado	Tempo médio de Execução do Algoritmo (milissegundos)		SpeedUp (Em relação ao algoritmo fornecido)	
	Algoritmo 2	Algoritmo 3	Algoritmo 2	Algoritmo 3
7	0.00	0.00	0.87	0.95
27	0.00	0.00	0.62	1.00
8431	0.00	0.00	42.69	44.67
13033	0.00	0.00	61.58	45.66
524287	0.00	0.00	839.22	1236.75
664283	0.01	0.00	2.12	2.67
2147483647	0.17	0.08	72609.58	151862.79

1. Detalhamento dos algoritmos:

Algoritmo 2: Para o algoritmo 2, foi proposta uma melhoria com base no algoritmo fornecido, pois uma vez que para um dado valor a ser testado n, qualquer valor acima de $\sqrt{n}$ é apenas um espelho da multiplicação de um múltiplo já calculado anteriormente e, portanto, não precisa ser calculado.

Exemplo: Para n = 100 ($\sqrt{100} = 10$), temos os seguintes múltiplos: 2(x50), 4(x25), 5(x20), 10(x10), 20(x5), 25(x4), 50(x2). Pode-se notar que todos os valores acima de 10 são apenas multiplicações invertidas (espelhadas) que já foram calculadas. Portanto não precisam ser contabilizadas quando buscamos validar se o número é primo.

Algoritmo 3: O algoritmo 3 utiliza como base o algoritmo 2, porém acrescenta a otimização conhecida como $(6k \pm 1)$. Assim, além de testarmos apenas valores até $\sqrt{n}$, também aproveitamos do fato de que todo número inteiro é múltiplo de $(6k + i)$ para k $\in$ Z_+^* e i pertencente ao conjunto [-1, 0, 1, 2, 3, 4]. Como qualquer valores de $(6k + 0)$, $(6k + 2)$ e $(6k + 4)$ é sempre múltiplo de 2 e qualquer valor de $(6k + 3)$ é sempre múltiplo de 3, basta testar se n é divisível por 2 ou 3 em primeiro momento e após isso sobram apenas os números consecutivos de $(6k + (-1))$ e $(6k + 1)$ para serem testados.

2. Captura de tempo:

Para realizar a contagem de tempo, foi utilizada a biblioteca time.h, na qual através da função clock(), recupera-se o valor do clock no momento. Com isso, basta armazenar o valor de clock antes e depois da execução do algoritmo para termos quantos ciclos de clock foram utilizados. Para converter ciclos de clock em segundos, a biblioteca fornece a constante CLOCKS_PER_SEC e para o valor ser passado para milissegundos divide-se por mil.

3. Código fonte:

Todo o código desenvolvido, assim como o output da execução com maiores detalhes de tempo por iteração (30 iterações foram utilizadas no cálculo) podem ser encontradas em: https://github.com/12pedro07/FEI-CS/tree/main/CC7261-SistemasDistribuidos/SpeedUp_PrimeNumbers

4. Avaliação do SpeedUp:

Através dos resultados de SpeedUp, pode-se notar que a melhora de tempo em função do aumento no tamanho de n (para n sendo o valor inteiro avaliado) é exponencial, ou seja, quanto maior o número, mais maior será a diferença de tempo entre algoritmos. Porém caso o número avaliado seja múltiplo de um inteiro de valor baixo, este tempo tende a diminuir (um exemplo são números pares, que para qualquer algoritmo, sempre cairão no caso base e terão seu resultado quase instantaneamente).

Nome: Pedro Henrique Silva Domingues
R.A.: 22.218.019-2