RELATÓRIO DO EP 1 – MAC0121

Aluno: Adriano Elias Andrade N° USP: 13671682

Data: 11/09/2022 Professor: Guilherme Oliveira Mota

Conceitos matemáticos e simplificações usadas:

1) Dobros: A conjectura diz que para números pares, o cálculo aplicado é x/2. Portanto, se ao dividir um número por dois, avança-se um passo, então para o dobro 2n de um número n, a quantidade de passos é uma a mais do que n: basta dividir por dois. Aplicando essa regra consecutivamente, é possível descobrir a quantidade de passos para qualquer número da forma $2^m \cdot n$, que possui m passos a mais que n.

2) Vetor de memória: É usado também um vetor de memória MEM, que armazena o resultado de cada número, de 1 a f, e ao invés de calcular todos os passos de cada número, pode verificar se já não foi calculado nos dobros (1), a cada passo dado.

3) Passo ímpar duplo: Para qualquer número i ímpar, é possível afirmar que $3 \cdot i + 1$ é par. Isso é verdade pois um ímpar vezes um ímpar (3 · i) tem resultado ímpar, que por sua vez, se somado a um, é par. Assim, podemos afirmar que nessa conjectura, seguido de um ímpar, sempre haverá um par, e portanto, é possível dar dois passos de uma vez. Isso é útil quando aplicamos as regras em ordem invertida, pois se primeiro dividirmos um número por 2, e então multiplicarmos por 3, e somar $\frac{1}{2}$, sempre estaremos trabalhando com números menores (pela metade), ignorando o passo intermediário, que terá um resultado maior.

Exemplo:

Ordem normal: 5 → 16 → 8 (máx 16)

Ordem invertida: 5 → 2,5 → 8 (máx 8 – metade)

Isso evita dificuldades durante o cálculo e permite calcular intervalos maiores antes de atingir um overflow.

Traduzindo esse raciocínio para C, trabalhando com inteiros: O número ímpar i, da forma 2n + 1, quando dividido por 2 sempre terá 0,5 de parte não inteira (n + $\frac{1}{2}$), o que é ignorado no programa. Isso então é multiplicado por 3, se soma $\frac{1}{2}$, e agora temos 3n, quando deveríamos ter 3n + 1,5 + 0,5 = 3n + 2. Para corrigir esse erro, basta escrever essa conta no código da maneira: i / 2 * 3 + 2.

Informações interessantes obtidas:

Observando alguns resultados, imediatamente é possível notar a aleatoriedade do número de passos. Para alguns números como 26 temos 9 passos, mas para o seu sucessor, 27, são 111 passos até o 1. Assim como o 27, existem alguns números que dão saltos na quantidade de passos, imprevisivelmente.

Por outro lado, essa aleatoriedade muitas vezes se repete para números próximos. Foram vistos vários casos em que múltiplos números seguidos (mais de 4) possuem a mesma quantidade de passos. Como não foi possível achar um padrão para essas ocorrências, nada foi implementado a respeito.

Maior intervalo calculado: 1 a $8{,}5 \cdot 10^8$ (57min 33s).

Esse intervalo foi o maior alcançado, não por conta de um overflow durante os cálculos, mas pela incapacidade de alocar uma memória maior, que guarde os resultados num vetor de long long int.

É interessante notar, que para o intervalo de 1 a 10^4 o tempo foi 3.5s, de 1 a 10^5 5.3s e de 1 a 10^6, 9s. Isso indica que o crescimento no tempo para esses valores não foi linear, mas logarítmico. De 1 a $1 \cdot 10^8$, o tempo foi de 6min 46s, o que continua sendo 45% de uma projeção linear em relação ao 10^6. No entanto, de $1 \cdot 10^8$ para $8{,}5 \cdot 10^8$, o aumento de tempo é praticamente linear.