Jhony Minetto Araújo 14/11/2021

Primeiramente, considerando os quatro problemas introduzidos no projeto, as funções que pareciam mais desafiadoras – e que no final realmente acabaram sendo – foram a primeira, do FourCC (*Four Character Code*), e a função desafio, de avaliar códigos válidos. Apesar dos outros dois problemas demandarem mais tempo para se escrever a solução – principalmente em relação ao primeiro –, em termos de abstração e lógica eles eram mais simples de se resolver.

A primeira função se apresentou como um "desafio" no momento de contato inicial com o problema, pois envolve operações *bit a bit* e manipulação de números hexadecimais, que, não sendo algo com os quais tenho costume, foram mais difíceis de visualizar. Para desenvolver a solução, realizei alguns *testes de mesa* em conjunto com a calculadora do Windows 10 (no modo *Programador*), seguindo a lógica dada pelo problema para ilustrar sua lógica. Logo percebi que o problema era meramente uma questão de somar as sequências na ordem desejada, ao mesmo tempo multiplicando-as por potências de 2 para 'deslocá-las' para a posição certa. Os expoentes de dois foram facilmente determinados considerando que o tipo *char* tem tamanho de 1 byte (8 bits). Isso significa que o operador Left Shift (<<) pode ser usado na forma `character << n*8` para colocar a sequência na n-ésima.

A segunda função, para calcular a intersecção dos retângulos foi facilmente implementada. Ao se pensar no problema no plano cartesiano, percebe-se que a intersecção se dará sempre que as coordenadas ordenadas do canto superior esquerdo (SE) e inferior direito (ID) forem menor (SE) e maior (ID), respectivamente, ao mesmo tempo que as coordenadas abscissas forem maior (SE) e menor (ID).

A última função formal do trabalho, sobre aproximar raízes, também foi facilmente resolvida, dado que toda a lógica do problema já estava descrita no enunciado. A maior dificuldade que se apresentou foi calcular as raízes de números entre 0 e 1. Isso implicou em mudar a lógica inicial do programa, que antes dava o limite máximo para a raiz como sendo o próprio número – já que para números fora desse intervalo isso é verdade–, para considerar o limite máximo como 1. Caso contrário o loop nunca teria fim, pois como a raiz seria maior que o próprio número, a função nunca acharia um candidato melhor que o anterior.

Por fim, a função mais desafiadora de todo o trabalho foi a função desafio. Inicialmente, ao tentar resolver o problema, procurei por padrões entre os números gerados, convertendo-os para decimal para procurar alguma regra de formação válida, o que acabou somente me desviando da solução correta, pois passei um bom tempo procurando por tais relações – números primos, combinações lineares entre os números, permutações das sequências, entre outros. A primeira solução que consegui encontrar era inválida, pois se baseava em recursão, que, apesar de funcionar, era claramente ineficiente, pois demorava um tempo considerável para checar mesmo as sequências pequenas, de 11 e 13 bits. Então, após um período de pesquisa e tentativa e erro, encontrei a sequência de Catalan, que, curiosamente, seguia a mesma ordem dada na tabela de sequências válidas no enunciado. Acabei por encontrar um artigo[1] que analisa objetos de programação relacionados à sequência, que proveu uma lógica usando o conceito de árvores binárias que produzia os mesmos resultados que o problema requisitava. Ao implementar em código, após alguns *debugs*, o programa produziu a saída esperada, e de forma muito mais eficiente.

[1] https://www.researchgate.net/publication/45904772_Coding_objects_related_to_Catalan_numbers