Resolução - Meteor Challenge

Rodrigo Augusto Valeretto

Irei descrever por meio deste documento a solução implementada por mim para resolver o desafio “MeteorChallenge”. A implementação foi realizada em python, mais especificamente na versão 3.9.5. As bibliotecas utilizadas foram NumPy, PIL e time (usado para estimar a eficiência). O link para o github com algumas soluções está abaixo:

- https://github.com/RodrigoValeretto/MeteorChallenge

Código

Inicialmente utilizei a biblioteca PIL (python image Library) para abrir o arquivo de imagem e salvar sua referência em uma variável, com o numpy transformei esse arquivo em um vetor de bits (mais especificamente o rgba de cada um deles).

Utilizamos então uma função “totuple” para transformar esses arrays gerados em tuplas, isso é feito pois o python aceita melhor a comparação entre tuplas do que entre arrays.

Observando a figura percebemos que próximo ao centro da mesma existe uma grande quantidade de chão, e esse chão é mais alto do que o do início da imagem. Também notamos que a partir do momento em que chega ao chão não existem mais estrelas. A fim de diminuir o número de iterações e melhorar a eficiência do programa, realizamos uma busca na matriz de bytes a fim de encontrar o primeiro byte de cor preta (nível do chão). Isso é feito utilizando a coluna fixa na metade do comprimento total da imagem (int (w/2)) e iterando as linhas.

Fazemos uma iteração quadrática com o intuito de encontrar o nível da água, visto que ele é o mesmo para todos os bytes de água, assim, logo que encontramos o primeiro byte azul salvamos o índice de sua linha (nível da água) e paramos a iteração. Para melhorar a eficiência dessa busca, iteramos primeiramente em linhas e depois em colunas, além disso começamos a analisar as linhas que estão da metade da altura para baixo pois podemos perceber analisando a imagem que o nível da água fica abaixo desse ponto.

Tendo o nível da água em mãos uma nova iteração é executada para encontrar todas as colunas da imagem em que existe um byte de água, isso é feito fixando a linha no nível da água encontrado anteriormente e iterando as colunas, os índices das colunas que contém bytes azuis são adicionados à uma lista (waterCols). Isso é importante para encontrarmos quantos meteoros atingem a água sem precisar fazer muitas iterações.

Passamos então à iteração responsável por contar o número de estrelas e meteoros, essa iteração é quadrática e percorre toda a matriz de bytes, ao longo do caminho verifica se o byte é vermelho ou branco e incrementa os contadores de meteoros e estrelas respectivamente. Para melhorar a eficiência, as linhas são iteradas até o valor do nível do solo, encontrado anteriormente, as colunas são iteradas normalmente. Ao encontramos um meteoro verificamos também se o índice da coluna em que ele está se encontra no vetor “waterCols” em que guardamos os índices das colunas que possuem água, se sim, significa que esse meteoro (ao cair perpendicularmente) cairá na água.

No link do github citado existem duas implementações, a primeira e mais simples (branch “old-version”), e a que descrevo aqui, que é minha resolução final (branches “main” e “alternative”), podemos perceber através das medidas de tempo que o algoritmo teve uma melhora de até 3 vezes sua antiga eficácia.

Os resultados obtidos foram:

Número de estrelas	315
Número de meteoros	328
Número de meteoros que caem na água	105