MC621 - Relatório X

Luiz Felipe Corradini Rego Costa - 230613

Contest 04/10

1. C - Kuroni and the Gifts

1.1 - Ideia da Solução

No problema em questão, precisamos determinar pares de braceletes para a i-ésima filha de Kuroni, tal que cada filha tenha um par de braceletes e colares com soma de brilho distinta de todas as outras. Para isso, utilizaremos uma implementação simples, com complexidade *O(n log n)*, que resolve o problema em questão.

1.2 - Detalhes da Implementação

Como queremos garantir que as somas sejam distintas, e o problema garante que existe tal combinação, começamos por ordenar ambas os vetores de braceletes e colares. Agora, basta iterar pelo seu tamanho, atribuindo para a i-ésima filha, o i-ésimo bracelete e o i-ésimo colar. Com isso, garantimos que, se existe uma distribuição tal que todos os brilhos são diferentes, ela será encontrada, já que será atribuída a menor combinação para a primeira filha, segunda menor para a segunda filha, e assim por diante.

2. H - Longest Palindrome

2.1 - Ideia da Solução

No problema em questão, é solicitado que encontre o maior palíndromo possível dentro de uma determinada *string*. Para isso, iremos utilizar uma solução com programação dinâmica, de modo a salvar o tamanho do maior palíndromo para cada posição distinta da *string.*

2.2 - Detalhes da implementação

Como dito anteriormente, utilizaremos dp para resolver o problema. Para isso, criaremos uma matriz de dp, com tamanho *len(string) x len(string).* Além disso, criaremos uma função recursiva que recebe L e R como parâmetros, sendo L um parâmetro equivalente ao índice pela esquerda, e R pela direita.

Quando *L >= R,* retornamos 0 para aquela posição, já que não temos palíndromo entre *L e R.* Se *L = R,* temos um palíndromo de tamanho 1, então retornamos 1.

Se nenhum desses casos for satisfeito e *string[l] = string[r],* significa que temos um "potencial" palíndromo entre L e R, então incrementamos o tamanho do palíndromo na posição em 2, e chamamos a função para *L + 1* e *R - 1.*

Se não, apenas retornamos o máximo entre (*L, R - 1) e (L+1, R).* Com isso, garantimos que iremos considerar todas as sequências possíveis, e então, encontrar o maior palíndromo possível.