Aluno: André Santos Rocha
Ra: 235887

Problema 1: A - Cards

O problema nos fornece uma string com os caracteres ‘z’, ‘e’, ‘r’,’o’,’n’,’e’. Esses caracteres estão embaralhados. Devemos ordená-los de tal forma a obter o maior número binário possível.

1.1 Ideia de solução

Para este problema é possível desenvolver uma solução simples. Primeiramente, é preciso identificar a quantidade de ‘zero’ e ‘one’. Para isso, podemos contar as aparições dos caracteres ‘n’ e ‘o’. Cada aparição do caracter ‘n’ na entrada contabiliza um ‘one’. Após contabilizar as aparições de ‘n’ (num_n) e as de ‘o’ (num_o), o número de ‘zero’ será num_o - num_n. Por fim, para criar o maior número possível, é preciso que coloquemos o dígito 1 nas posições mais significativas e o dígito 0 nas posições menos significativas. Sendo assim, printamos um num_n vezes o número 1 e depois num_o - num_n vezes o número 0. Com isso, o problema está solucionado.

1.2 Detalhes de implementação

No nosso output, os dígitos 1 e 0 devem estar espaçados. Portanto, a cada vez que printarmos um dígito, printaremos também um espaço “ “.

Problema 2: G - Fox and Snake

O problema nos fornece dois inteiros n e m (3<= n, m <= 50). Devemos criar uma tabela n por m, preenchendo-na de acordo com o padrão da cobra. Esse padrão consiste nas seguintes regras:

1. A cobra é representada pelo caractere ‘#’.
2. O corpo da cobra começa na posição (1,1) e se estende até (1,m).
3. Daí o corpo da cobra salta duas linhas, chegando na posição (3,m).
4. O corpo da cobra se estica da posição (3,m) até a (3,1).
5. O corpo da cobra salta duas linhas novamente, se estica até (5,m) e assim por diante.
6. Os espaços não ocupados pelo corpo da cobra são preenchidos com ‘.’ .

1.1 Ideia de solução

A ideia principal desse problema é iterar pela tabela, printando ‘.’ ou ‘#’ quando necessário. Supondo que, dado n e m, nossa tabela tenha n linhas, numeradas de 0 a n-1, e m colunas, numeradas de 0 a m-1. Para cada linha, passaremos por todas as colunas, preenchendo com o caractere adequado. A primeira linha é totalmente preenchida com ‘#’, como explicado nas regras. A próxima linha a ser totalmente preenchida é a linha 2, pois o corpo da cobra salta duas linhas. Após a 2 será a 4, 6,8….ou seja, todas as linhas pares são preenchidas totalmente com ‘#’. Portanto, cada vez que iteramos por uma linha, verificamos se seu índice é par. Caso seja par, preenchemos-na com ‘#’, para todas as colunas. Agora nos resta o caso em que as linhas são ímpares. Nas linhas ímpares, todas as colunas de uma linha são preenchidas com ‘.’, exceto as colunas m-1 ou 0. Na primeira linha de número ímpar, a linha de número 1, todas as colunas exceto a m-1 são preenchidas com ‘.’. A coluna m-1 é preenchida com ‘#’. No caso da linha 3, a coluna 0 é que será preenchida com ‘#’. Então, para as linhas ímpares, preenchemos tudo com ‘.’, intercalando o ‘#’ entre as posições nas colunas 0 e m-1, começando pela m-1.

1.2 Detalhes de implementação

Para criar o efeito de intercalar o ‘#’ entre m-1 e 0 nas linhas ímpares, podemos fazer criar uma variável “intercala” inicializada com valor 1. Daí, haverá duas possibilidades se a linha não for par:

1) Se estivermos na coluna de posição 0 e intercala = 0, printamos o ‘#’ e intercala passa a ser 1.
2) Se estivermos na coluna de posição m–1 e intercala = 1, printamos o ‘#’ e intercala passa a ser 0.

Além disso, outro detalhe na implementação é que sempre que não estivermos em uma linha ímpar e nenhuma das condições acima for atendida, preenchemos a posição com ‘.’. Por fim, o último detalhe é que, a cada vez que terminarmos a iteração pelas colunas de uma linha, printamos um ‘\n’ para saltar para a próxima linha.