ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

DOCUMENTO TÉCNICO

Requisitos

Especificación1

Realizar un código que permita desencriptar un mensaje basándonos en una letra de cada palabra, de tal manera que las letras formarán un mensaje con la primera letra de la primera palabra, la segunda letra de la segunda palabra y así sucesivamente. Si una palabra no tiene suficientes letras se sigue con la siguiente palabra.

Entrada

La primera línea de entrada da el número de casos, T(1<=T<=30), luego hay un espacio en blanco antes del primer caso. Cada caso de prueba representa un mensaje. Cada caso de prueba representa un mensaje, el cual está compuesto por N(1<=N<=100) líneas y cada línea está compuesta entre 1 y 30 palabras. Dos palabras en la misma línea están separadas por una o más líneas en blanco. Una palabra está conformada por letras del abecedario (Mayúsculas y minúsculas) y tiene como máximo 30 letras, os únicos símbolos que pueden entrar son letras y espacios. Debe haber un espacio entre cada mensaje.

Salida

Una cadena que es de la forma "Case #x" donde x es el número del caso correspondiente, y después de esto tantas líneas de texto compuestas por una palabra como líneas de palabras se introdujeron en cada caso.

Especificación2

Siendo la matriz de Hadamard una matriz de tamaño n x n (cuadrada), ese valor n debe ser potencia de 2 (2n) y además los elementos que lo contienen son valores booleanos, representados con [T] y [F].

Esta matriz cumple con una propiedad: los elementos de cada dos filas adyacentes difieren exactamente en n/2 elementos.

Por ejemplo, con n=1; la matriz Hadamard es un arreglo con un único elemento True: H1 = [T].

Entonces lo que hay que determinar es si una matriz es Hadamard o no de acuerdo a la información que se tenga.

Entrada

En la primera línea se encuentra el número n donde n >= 1, el cual indica el tamaño de la matriz (que debe ser cuadrada). En la siguiente línea encuentran n x n valores booleanos (T, F) que

corresponden a la matriz a validar separados por espacio, el llenado de la matriz (específicamente por filas) completa la información necesaria para tener en cuenta para la salida.

Salida

Se debe indicar si la información leída cumple o no a una matriz de Hadamard, o si es imposible hacer un matriz con esos elementos, ya que, el número n no es potencia de 2.

Especificación3

El Buscaminas es un juego de computadora en el que el jugador debe descubrir todas las minas en un campo de dimensiones n × m sin detonar ninguna de ellas. Cada celda del campo puede contener una mina o estar vacía. El juego muestra un número en cada celda vacía que indica la cantidad de minas adyacentes a esa celda.

Entrada

La entrada consta de un número arbitrario de campos. Cada campo comienza con dos números enteros n y m (1 ≤ m, n ≤ 100) que representan el número de filas y columnas del campo, respectivamente. A continuación, siguen n líneas, cada una conteniendo exactamente m caracteres que describen el campo. Los cuadros seguros están representados por el carácter '.' y los cuadros con minas por '*'. La primera línea descriptiva de un campo en la que n = m = 0 representa el final de la entrada y no debe procesarse.

Salida

Para cada campo, se debe imprimir la línea "Field #x:", donde x corresponde al número del campo (comenzando desde 1). Luego, se deben imprimir n líneas que contienen el campo con los caracteres '.' sustituidos por el número de minas adyacentes a ese cuadro. Debe haber una línea en blanco entre dos campos consecutivos.

Diseño

Estrategia1

Por cada caso descifrar un mensaje, en el cual, por cada línea de palabras, según el numero de la palabra ingresada, se toma la letra de la palabra correspondiente a ese número, si el tamaño de la palabra es menor al número se ignora esta palabra y se toma la siguiente, sin aumentar el número. La estrategia utilizada para resolver el problema es estrategia incremental, la idea es almacenar las palabras en una lista y con un temporal verificar si la palabra es tan larga como su posición, si es así añadir la letra correspondiente a la posición en la palabra, a una lista y aumentar el temporal. al final convertir la lista de letras en un Sting, juntar las letras y devolver el mensaje.

La estructura de datos implementada fue la lista, y se implementó al almacenar las palabras a una lista, y de esta palabra, si cumple con los requisitos, almacenar la letra en una lista diferente.

ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Invariante #1 (ver código)

Por cada caso se descifra un mensaje.

- Iniciación: No hay ningún caso.
- Estabilidad: Por cada número dentro del rango establecido debe haber un caso.
- Terminación: Hubo tantos casos como el rango definido.

Invariante #2 (ver código)

Por cada palabra tan larga como index hay una letra sii index > 0.

- Iniciación: Hay un mensaje tan largo como palabras tan largas como el índice en la línea (0 al principio).
- Estabilidad: Por cada palabra del ciclo sii la palabra es tan larga como index se toma la letra y se añade el mensaje final.
- Terminación: No hay más palabras de las cuales se pueda sacar un mensaje.

Estrategia2

Teniendo una matriz con tamaño n >= 1, se crea una matriz de n x n llena de valores False (porque los elementos de la matriz son valores booleanos), también hubiera podido ser un número, pero por comodidad se puso False. Después se llena la matriz con [T] Y [F] y para verificar si la matriz es o no Hadamard, o es imposible crearla; compara cada par de filas y verifica si la cantidad de posiciones en las que tienen el mismo valor es igual a la mitad del tamaño de la matriz (n/2), es decir, aquí es donde se aplica la estrategia de “dividir y conquistar”, ya que el problema se divide en subproblemas más pequeños (comparar pares de filas) y las soluciones a estos subproblemas se combinan para resolver el problema original (verificar si la matriz es una matriz de Hadamard).

Se utilizaron diferentes estructuras de datos como:

- Listas: se utilizan para almacenar múltiples elementos en una sola variable. En el código, las listas se utilizan en varias partes:

  - ✓ En la función crear_matriz(n), se crea una matriz como una lista de listas, donde cada sublista representa una fila de la matriz.

  - ✓ En la función llenar_matriz (matriz, n), se llena la matriz (que es una lista de listas) con valores booleanos ingresados por el usuario.

- Variables booleanas: pueden tener dos valores posibles, True o False. En el código, las variables booleanas se utilizan para representar los valores en la matriz.

ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Invariante #1 (ver código)

Por cada caso se mira si la matriz es Hadamard o no es; o si es imposible.

- Iniciación: No hay ningún caso.
- Estabilidad: Por cada iteración dentro del rango establecido debe haber un caso, dependiendo también de los elementos.
- Terminación: Hubo tantos casos como el rango definido.

Invariante #2 (ver código)

La comparación entre los elementos de la matriz existirá sii tam > 0.

- Iniciación: Hay un caso base donde la matriz es de tamaño 1, entonces debe retornar la posición [0][0] de la matriz.
- Estabilidad: Por cada posición de la matriz dentro del bucle, esta se va comparando con los demás si cumple la condición: los elementos de cada dos filas adyacentes deben tener la mitad del tamaño de la matriz.
- Terminación: Ya recorrió toda la matriz, y terminó de comparar todos los elementos, así muestra la característica de la matriz (es, no es o imposible).

Estrategia3

El objetivo del problema del Buscaminas es determinar el número de minas adyacentes a cada celda de un campo dado. Para conseguirlo, hemos desarrollado un algoritmo que recorre cada celda del campo, cuenta las minas adyacentes y actualiza el campo con esta información.

Nuestra estrategia se basa en un enfoque incremental. A continuación, describimos paso a paso cómo resolvemos el problema:

- **Recorrido del campo:** Iteramos sobre cada celda del campo utilizando dos bucles anidados, uno para las filas y otro para las columnas. Esto nos permite examinar cada celda del campo y determinar su contenido (mina o espacio vacío).
- **Contar las minas adyacentes:** Para cada celda que no contenga una mina, realizamos un recorrido adicional por las celdas adyacentes. Utilizamos dos bucles adicionales para examinar las celdas adyacentes a la celda actual. Si encontramos una mina en alguna de estas celdas adyacentes, incrementamos un contador que lleva la cuenta del número de minas adyacentes encontradas.
- **Actualización del campo:** Una vez que hemos contado las minas adyacentes para una celda concreta, actualizamos el valor de la celda con el número de minas encontradas.

ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Invariante #1

Al contar las minas adyacentes de una casilla determinada, la invariante mantiene que el contador de minas adyacentes aumenta con precisión y refleja el número de minas encontradas hasta ese momento. Esto garantiza que la información sobre las minas adyacentes se mantiene precisa y actualizada en todo momento.

Invariante #2

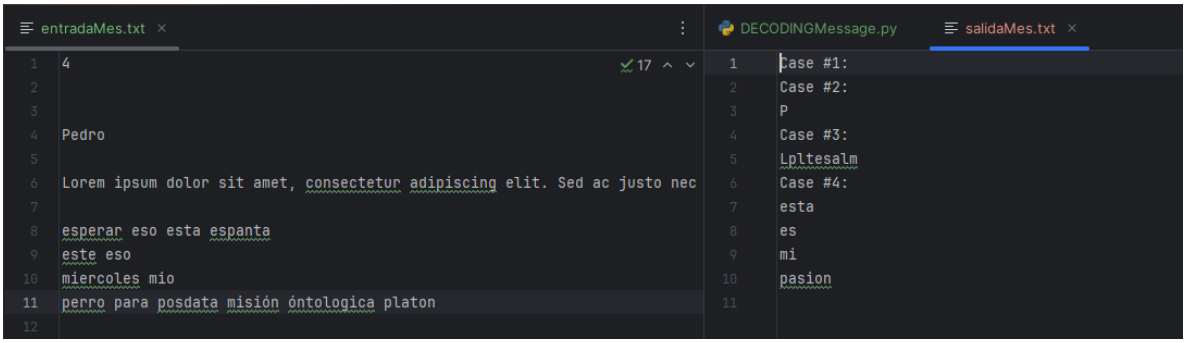
A lo largo del algoritmo, se garantiza que el campo de juego permanece intacto, lo que significa que no se alteran las ubicaciones de las minas ni se modifican las casillas no minadas. Esto garantiza que la representación del campo permanezca coherente y fiel al estado original del juego.

Invariante #3

Durante el proceso de recorrer el campo y acceder a sus celdas, se garantiza que los índices utilizados estén dentro de los límites válidos del campo (es decir, que no sean negativos ni superen el tamaño del campo). Esto evita errores de acceso a memoria no válida y garantiza un comportamiento predecible del algoritmo.

Casos de prueba1

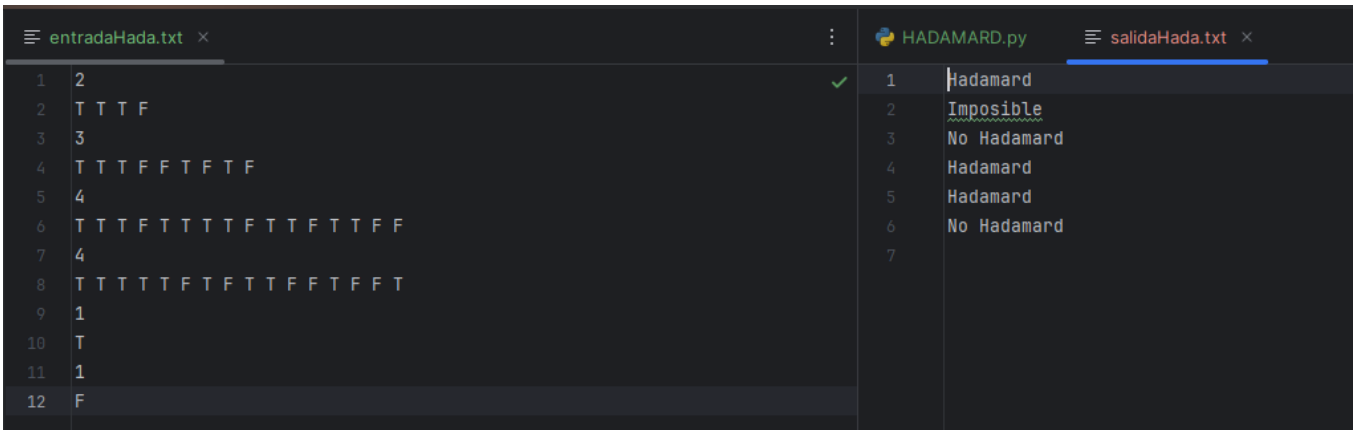
Entrada	Justificación	Salida
	Línea vacía	
Pedro	Una sola línea sencilla	P
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac justo nec urna cursus lacinia. Nullam eget nunc vel sapien vestibulum fermentum.	30 palabras	Lpltesalm
esperar eso esta espanta este eso miercoles mio perro para posdata misión óntologica platon	5 lineas	Case #1: esta es mi pasion



ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Casos de prueba2

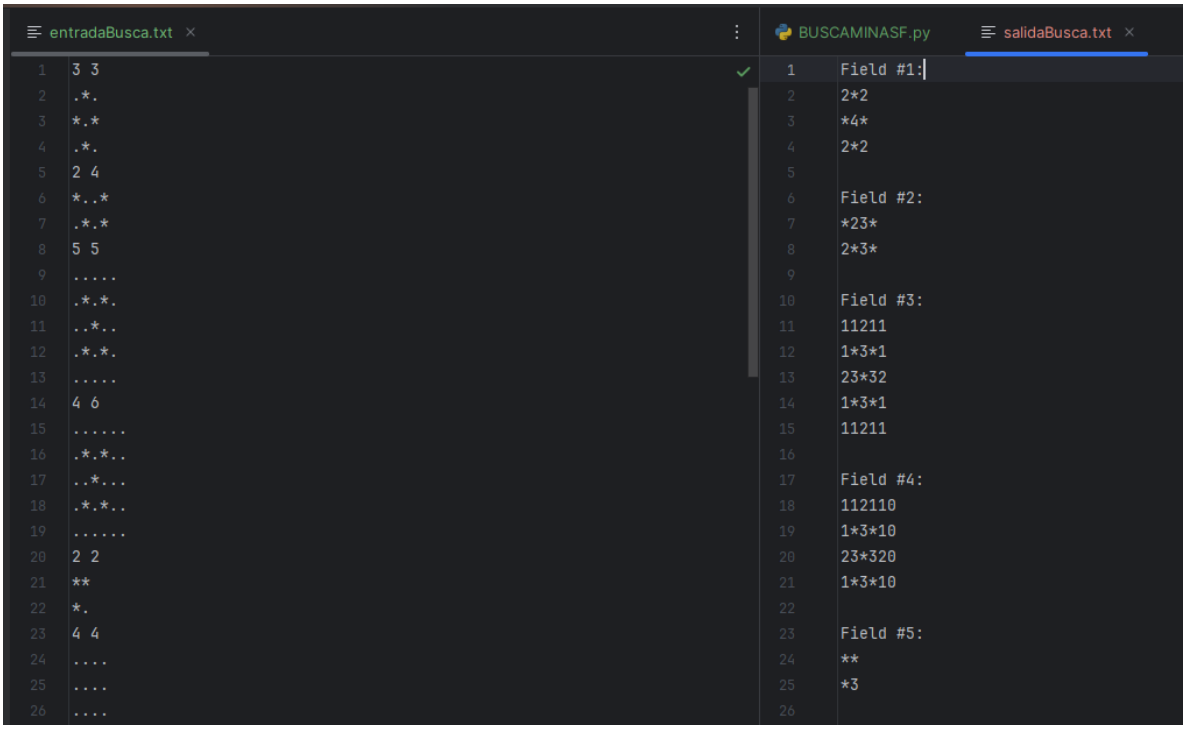
Entrada	Justificación	Salida
2 T T T F	n=2	Hadamard
3 T T T F F T F T F	n = 3	Impossible
4 T T T F T T T T F T T F T T F F	n = 4	No Hadamard
4 T T T T T F T F T T F F T F F T	n = 4	Hadamard
1 T	n = 1	Hadamard
1 F	n=1	No Hadamard



ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Casos de prueba3

Entrada	Justificación	Salida
3 3 .*. *.* .*.	Mismas filas y columnas	Field #1: 2*2 *4* 2*2
2 4 *..* .*.*	Diferentes columnas y filas	Field #2: *23* 2*3*
0 0	Caso base	
4 6 ..... .*.*.. ..*... .*.*..	Diferentes columnas y filas	Field #4: 112110 1*3*10 23*320 1*3*10



ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Análisis

Temporal1

Usando análisis asintótico para determinar la complejidad para el peor y mejor de los casos:

```
from sys import stdin
def write(line):
    words = []
    index = 0
    for i in line: #ciclo de la invariante #1
        if len(i) > index:
            words.append(i[index])
            index += 1
    words = map(str, words)
    print("".join(words))
    return index

def main():
    rango = int(stdin.readline().strip())
    empty = stdin.readline()
    for i in range(rango):#ciclo de la invariante #2
        print("Case #" + str(i + 1) + ":")
        line = stdin.readline().strip().split(" ")
        while len(line[0]) > 0:
            write(line)
            line = stdin.readline().strip().split(" ")

main()
```

```
# costo   pasos
#   1       1
#   1       1
#   1       n
#   1       n - 1
#   1       n - 1
#   1       n - 1
#   1       1
#   1       1
#   1       1
# Total O = (n), omega = 1
# costo   pasos
#   1       1
#   1       1
#   1       j
#   1       j - 1
#   1       j - 1
#   1       j - 1
# write(n)  j - 1
# 1         j - 1
# Total O = (n), omega = 1
```

codesnap.dev

Temporal2

Usando análisis asintótico para determinar la complejidad para el peor y mejor de los casos:

```
def hadamard(matrix, n):
    if n == 1:
        return matrix[0][0]
    for i in range(n):
        for j in range(i + 1, n):
            if sum(matrix[i][k] != matrix[j][k] for k in range(n)) != n / 2:
                return False
    return True
```

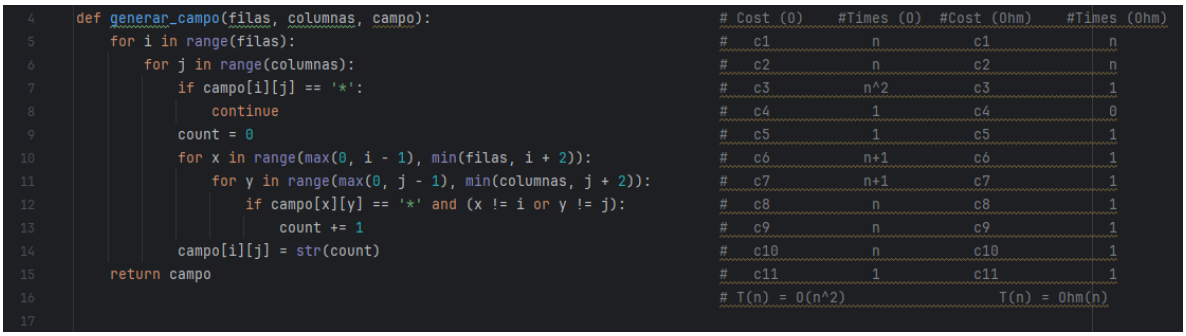
```
#Cost (O)     #Times (O)  #Cost (Ohm)     #Times (Ohm)
#c1               1          c1               1
#c2               1          c2               1
#c3               n          c3               1
#c4             n+1          c4               1
#c5               n          c5               n
#c6               1          c6               1
#c7               1          c7               1

   # T(n) = O(n)                T(n) = Ohm(n)
```

ANDERSSON DAVID SÁNCHEZ MÉNDEZ, CRISTIAN SANTIAGO PEDRAZA RODRÍGUEZ,
JUAN PABLO NIETO CORTES, DAVID EDUARDO SALAMANCA AGUILAR

Temporal3

Usando análisis asintótico para determinar la complejidad para el peor y mejor de los casos:



```python
def generar_campo(filas, columnas, campo):
    for i in range(filas):
        for j in range(columnas):
            if campo[i][j] == '*':
                continue
            count = 0
            for x in range(max(0, i - 1), min(filas, i + 2)):
                for y in range(max(0, j - 1), min(columnas, j + 2)):
                    if campo[x][y] == '*' and (x != i or y != j):
                        count += 1
            campo[i][j] = str(count)
    return campo


# Cost (O)    #Times (O)  #Cost (Ohm)    #Times (Ohm)
#   c1            n            c1             n
#   c2            n            c2             n
#   c3           n^2           c3             1
#   c4            1            c4             0
#   c5            1            c5             1
#   c6           n+1           c6             1
#   c7           n+1           c7             1
#   c8            n            c8             1
#   c9            n            c9             1
#   c10           n            c10            1
#   c11           1            c11            1
# T(n) = O(n^2)                 T(n) = Ohm(n)
```

Documentación1, Documentación2, Documentación3

Dentro del código

Fuentes

/DECODINGMessage.py	/entradaMes.txt	/salidaMes.txt
/HADAMARD.py	/entradaHada.txt	/salidaHada.txt
/BUSCAMINASF.py	/entradaBusca.txt	/salidaBusca.txt