A. Alpacas y Guanacos

Límite de tiempo: 2 segundos

Límite de memoria: 256 megabytes

En pleno altiplano chileno, los cuidadores de una estancia tienen un tremendo enredo con sus animales. En un terreno abierto pastan dos tipos de animales: alpacas y guanacos.

El problema es que las alpacas y los guanacos no se llevan bien: si encierras a ambos en un mismo corral, se agarran a escupos y pelean, dejando la escoba. Además, como el terreno no tiene rejas perimetrales, si un corral no queda completamente cerrado respecto al borde del mapa, se arrancan a la cordillera.

Para salvar la estancia, tienes que armar corrales cerrados usando un catálogo de rejas. Tu objetivo es colocarlas de forma inteligente para maximizar la ganancia neta.

El terreno corresponde a una grilla cuadrada de $N \times N$ celdas, con filas y columnas numeradas del 1 al N. Inicialmente, cada celda está vacía o contiene a lo más un animal.

Un corral es una componente conexa de celdas libres (conectadas ortogonalmente: arriba, abajo, izquierda o derecha) que se encuentra completamente rodeada por rejas y aislada del exterior. Para que un corral sea válido y genere felicidad, debe estar estrictamente cerrado sin alcanzar los bordes del terreno (fila 1, fila N, columna 1 o columna N); si una zona de celdas libres toca el perímetro exterior, se considera campo abierto y los animales en ella se arrancan a la cordillera. Asimismo, todos los animales dentro del corral deben pertenecer a la misma especie, ya que si se encierran alpacas y guanacos juntos, se agarran a escupos y pelean, dejando la escoba y arruinando la felicidad del corral. Por último, un corral cerrado que no contenga ningún animal tampoco produce ganancia alguna.

Cada animal necesita pasto para alimentarse, pero su capacidad de consumo tiene un límite: cada individuo puede aprovechar a lo más $A_{\max}$ celdas de pastizal. Si construyes un corral demasiado grande para la cantidad de animales que hay adentro, el pasto excedente no se aprovecha, se seca o se llena de paja brava, sin aportar ni un peso adicional.

De esta forma, si un corral cerrado válido P contiene $C(P) \geq 1$ animales de la misma especie y abarca un área de $\text{Área}(P)$ celdas libres, su área efectiva de pastoreo se calcula como:

$$\text{ÁreaEfectiva}(P) = \min\Big(\text{Área}(P), C(P) \cdot A_{\max}\Big)$$

A los animales les gusta compartir, por lo que la felicidad obtenido por el corral corresponde a la cantidad de animales multiplicada por el pasto que efectivamente aprovechan:

$$\text{Felicidad}(P) = C(P) \cdot \text{ÁreaEfectiva}(P)$$

Cualquier zona que no cumpla con las condiciones de un corral válido (por ejemplo, corrales abiertos que tocan el borde, corrales con especies mezcladas o corrales vacíos) tiene un felicidad de 0.

Tienes a disposición un catálogo con M tipos de piezas de reja modulares, identificadas del 1 al M según su orden de aparición en la entrada. Cada pieza tiene dimensiones de alto H y ancho W, un costo base B, y una matriz donde `#` representa un bloque de reja sólida y `.` representa espacio libre. Puedes usar la cantidad de piezas que quieras y rotarlas en $0°, 90°, 180°$ o $270°$ en sentido horario.

Al colocar una pieza, se indica su número en el catálogo ($1 \leq i \leq M$), la coordenada (r, c) de su esquina superior izquierda ($1 \leq r, c \leq N$) y su ángulo de rotación. Toda pieza colocada debe quedar completamente dentro de los límites del terreno ($r + H' - 1 \leq N$ y $c + W' - 1 \leq N$, donde H' y W' son sus dimensiones ya rotadas). Los bloques de reja de distintas piezas se pueden cruzar y solapar libremente en la grilla. Sin embargo, por el bienestar de los animales, no se puede construir encima de ninguno de ellos: si un bloque `#` se coloca sobre una celda ocupada por una alpaca o un guanaco, la solución queda descalificada y el puntaje final es 0.

En la cordillera las rejas escasean y los obreros te van a cobrar más caro a medida que les pidas repetir el mismo modelo de reja.

La m-ésima vez que usas una pieza de tipo k ($m = 1, 2, 3, ...$), su colocación cuesta:

$$\text{Costo}(k, m) = B_k \cdot m$$

Así, si usas K_k copias de la pieza k, el costo acumulado de ese tipo de pieza es:

$$\text{CostoTotal}_k = B_k \cdot \frac{K_k(K_k + 1)}{2}$$

El costo total de construcción es la suma de los costos de todas las piezas usadas:

$$\text{CostoTotalRejas} = \sum_{k=1}^{M} \text{CostoTotal}_k$$

Tu ganancia neta es la suma de los felicidades brutos de todos los corrales válidos menos lo que gastaste en rejas:

$$\text{Puntaje} = \max\left(0, \sum_{P} \text{Felicidad}(P) - \text{CostoTotalRejas}\right)$$

Cualquier solución inválida (piezas que se salen de la grilla, rejas encima de los animales o formato incorrecto) obtiene puntaje 0. Tu objetivo es maximizar este puntaje neto.

Entrada

La primera línea contiene dos enteros N y $A_{\max}$ ($10 \leq N \leq 70$, $10 \leq A_{\max} \leq 100$), el tamaño del terreno ($N \times N$) y el límite de pastoreo por animal.

La segunda línea contiene dos enteros C y M ($1 \leq C \leq 100$, $1 \leq M \leq 50$), la cantidad de animales y la cantidad de tipos de piezas en el catálogo.

Las siguientes C líneas describen a los animales. Cada una tiene tres enteros r_i, c_i y especie_i ($1 \leq r_i, c_i \leq N$, $\text{especie}_i \in \{1, 2\}$): la fila, columna y especie del animal (`1` para Alpaca, `2` para Guanaco). No hay dos animales en la misma celda.

Luego vienen las descripciones de los M tipos de piezas del catálogo, según su orden de aparición (1 a M):

- Una línea con tres enteros: H, W y B ($1 \leq H, W \leq N$, $B \geq 1$), con el alto, ancho y costo base de la pieza.
- H líneas con un string de W caracteres (`#` o `.`) que muestran la forma de la pieza.

Salida

La primera línea debe contener un entero $K \geq 0$, la cantidad total de piezas de reja que vas a colocar.

Luego deben venir K líneas describiendo la colocación de cada pieza mediante cuatro enteros separados por espacio:

$$i \quad r \quad c \quad \text{rotación}$$

donde:

- i ($1 \leq i \leq M$) es el número de la pieza según su orden de aparición en el catálogo.
- r y c ($1 \leq r, c \leq N$) son las coordenadas de la fila y columna (indexadas desde 1) de la esquina superior izquierda de la pieza en la grilla tras aplicarle la rotación.
- `"rotación"` es el ángulo en grados (0, 90, 180 o 270).

Si prefieres no poner ninguna reja, imprime únicamente una línea con el número `0`.

Scoring

Cada caso de prueba se evalúa por separado. Para cada caso i, los puntos se normalizarán en una escala de 0 a 100 en función de la mejor solución obtenida por la organización (S_i^*):

$$\text{Puntaje}_i = \min\left(100, \left(\frac{S_i}{S_i^*}\right)^2 \cdot 100\right)$$

donde S_i es la ganancia neta obtenida por tu solución. Si la solución resulta inválida en un caso, el puntaje para ese caso será 0.

El puntaje final de tu entrega corresponderá al promedio de los puntajes obtenidos en todos los casos de prueba:

$$\text{PuntajeFinal} = \frac{1}{T}\sum_{i=1}^{T}\text{Puntaje}_i$$

donde T es la cantidad total de casos de prueba.

Ejemplos

Entrada 1	Salida 1
8 10 3 2 3 3 1 4 3 1 7 6 2 1 4 1 #### 4 1 1 # # # #	4 1 2 2 0 1 5 2 0 2 2 2 0 2 2 5 0

Nota

En el ejemplo anterior, el terreno es de 8×8 y $A_{\max} = 10$. Hay 3 animales: dos alpacas en $(3,3)$ y $(4,3)$, y un guanaco en $(7,6)$.

La solución coloca 4 piezas de reja formando un corral cerrado entre las filas 2 a 5 y las columnas 2 a 5:
- Se usan 2 copias de la pieza 1 (horizontal 1×4, costo base 1) en $(2,2,0)$ y $(5,2,0)$.
- Se usan 2 copias de la pieza 2 (vertical 4×1, costo base 1) en $(2,2,0)$ y $(2,5,0)$.

Adentro del corral quedan 4 celdas libres ($(3,3)$, $(3,4)$, $(4,3)$ y $(4,4)$), aisladas del borde del mapa (filas 1 y 8, columnas 1 y 8). Como adentro solo hay alpacas, el corral es válido:
- $\text{ÁreaEfectiva} = \min(4, 2 \cdot 10) = 4$.
- $\text{Felicidad} = 2 \cdot 4 = 8$.

El costo de las piezas con cobro progresivo es:
- Pieza 1: $1 \cdot \frac{2 \cdot 3}{2} = 3$.
- Pieza 2: $1 \cdot \frac{2 \cdot 3}{2} = 3$.
- $\text{CostoTotalRejas} = 3 + 3 = 6$.

El guanaco en $(7,6)$ quedó afuera en campo abierto (escapado), por lo que no genera conflicto ni felicidad. El puntaje neto final es:

$$\text{Puntaje} = \max(0, 8 - 6) = 2.$$

B. Minas y Mazmorras

Límite de tiempo: 2 segundos

Límite de memoria: 256 megabytes

Emilia estaba jugando con sus amigas a *Minas y Mazmorras*, el MMO RPG del momento. Decidieron aventurarse en las Minas de Rayse y avanzaban en grupo cuando Emilia se detuvo un instante a tomar agua. En ese momento, ¡el resto del grupo cayó en una trampa de teletransportación! Por pura suerte, Emilia quedó fuera del alcance de la trampa.

Luego de utilizar sus habilidades, descubrió que las *Minas de Rayse* se componen de N salas unidas por caminos estrechos que toman exactamente 1 minuto en atravesar. Gracias a los cursos de matemáticas discretas, Emilia reconoció que las *Minas de Rayse* tienen forma de árbol.

Cada una de las K amigas de Emilia se encuentra en una sala p_i distinta de las demás, y tiene h_i puntos de salud. Cada minuto que pasen sin recibir la ayuda de Emilia pierden 1 punto de salud; si alcanzan 0 puntos de salud su personaje muere permanentemente.

Por suerte, Emilia puede salvarlas. Específicamente, Emilia salva a su i-ésima amiga si alcanza la sala p_i a más tardar en el minuto h_i, notar que salvar a una de sus amigas ocurre de forma instantánea.

¡Ayuda a Emilia a salvar la mayor cantidad de sus amigas!

Entrada

La primera línea contiene 2 enteros N $(1 \leq N \leq 2 \cdot 10^5)$ y s $(1 \leq s \leq N)$, la cantidad de salas de las *Minas de Rayse* y la sala en la que se encuentra Emilia inicialmente.

Las siguientes $N-1$ líneas contienen pares de enteros $1 \leq u, v \leq N$, indicando que las salas u y v se encuentran conectadas por un camino estrecho. Se garantiza que las salas y los caminos estrechos forman un árbol.

La siguiente línea contiene 1 entero K $(1 \leq K \leq 20)$, el número de amigas de Emilia.

Siguen K líneas, la i-ésima línea contiene 2 enteros p_i $(1 \leq p_i \leq N)$ y h_i $(1 \leq h_i \leq 2 \cdot 10^5)$, la sala en la que se encuentra la i-ésima amiga de Emilia y sus puntos de salud, respectivamente.

Se garantiza que todos los p_i son distintos.

Salida

Imprime un único entero, el máximo número de amigas que Emilia puede salvar.

Ejemplos

Entrada 1	Salida 1
1 1 1 1 1	1

Entrada 2	Salida 2
3 1 1 2 2 3 1 3 1	0

Entrada 3	Salida 3
3 1 1 2 2 3 2 3 2 1 1	2

Nota

En el caso de ejemplo 1, Emilia ya se encuentra en la sala 1 en el minuto 0, cuando su amiga aún tiene 1 punto de salud, por lo que la salva al instante.

En el caso de ejemplo 2, a Emilia le toma 2 minutos llegar a la sala 3, pero su amiga perdió todos sus puntos de salud en el minuto 1, por lo que no alcanza a brindarle su ayuda.

En el caso de ejemplo 3, Emilia salva a la amiga de la sala 1 en el minuto 0, luego camina hasta la sala 3 y llega en el minuto 2, justo cuando la salud de la amiga en la sala 3 llega a 0.

C. Casino Compacto

Límite de tiempo: 2 segundos

Límite de memoria: 512 megabytes

El Casino Compacto (ex Casino Codificadas) volvió a abrir sus puertas al público luego de años sin funcionar. Entre las razones de su clausura se encontraba el uso desmedido de memoria de sus máquinas, por lo que decidieron cambiar de enfoque completamente: desarrollar juegos de azar que premiaran el uso eficiente de recursos.

Como parte del equipo de desarrollo del Casino Compacto, deberás diseñar una estrategia para probar uno de los juegos nuevos que obtenga el mayor puntaje posible (para calibrar algunos parámetros y evitar que el casino quede en negativo).

Jugarás en un grafo conexo no dirigido con n vértices y m aristas. Tu posición inicial será el vértice s.

El juego se desarrolla por rondas. Al comienzo de cada ronda aparece una moneda invisible en un vértice escogido uniformemente al azar entre todos los vértices distintos de tu posición actual.

Para encontrarla, puedes consultar la distancia mínima entre tu posición y la moneda pagando q fichas, o moverte a otro vértice pagando una ficha por cada arista recorrida.

Al capturar una moneda comienza inmediatamente una nueva ronda.

También puedes terminar una ronda sin capturar la moneda pagando r fichas.

El juego finaliza cuando te quedas sin fichas.

Tu estrategia deberá escribirse usando el pequeño lenguaje de la máquina del Casino Compacto.

La máquina posee una única celda de memoria que almacena un entero. Al comienzo de cada ronda, su valor es -1.

Tu programa puede contener las siguientes instrucciones:

Instrucción	**Descripción**
`MOVE v`	Recorre un camino mínimo hasta el vértice `v`, pagando una ficha por cada arista recorrida. Si existen varios caminos mínimos, en cada paso se escoge el vecino de menor índice que permita continuar por uno de ellos. Si no hay suficientes fichas para completar el recorrido, el programa termina inmediatamente.
`ASK`	Paga `q` fichas y guarda en la celda de memoria la distancia mínima entre tu posición y la moneda. Si no hay suficientes fichas, el programa termina inmediatamente.
`REROLL`	Paga `r` fichas y comienza una nueva ronda sin capturar la moneda actual. Si no hay suficientes fichas, el programa termina inmediatamente.
`GOTO etiqueta`	Continúa la ejecución desde la instrucción identificada por `etiqueta`.
`IF d etiqueta`	Si el valor almacenado en la celda de memoria es igual a `d`, continúa la ejecución desde `etiqueta`. En caso contrario, continúa con la siguiente instrucción.

La máquina ejecuta las instrucciones comenzando desde la primera. Salvo por `GOTO` e `IF`, después de cada instrucción continúa con la siguiente. Si alcanza el final del programa, este termina.

Puedes declarar etiquetas escribiendo su nombre seguido de `:` antes de una instrucción. Cada etiqueta debe tener un nombre único y estar asociada a una instrucción existente.

```
inicio:
MOVE 5
ASK
IF 1 cerca
REROLL

cerca:
GOTO inicio
```

Las etiquetas, líneas vacías y comentarios no cuentan como instrucciones. Los comentarios comienzan con `#` y se extienden hasta el final de la línea. Las instrucciones son sensibles a mayúsculas y minúsculas.

Cada instrucción ocupa exactamente una unidad de memoria. El **tamaño de tu programa** es la cantidad de instrucciones que contiene, independientemente de cuántas veces sean ejecutadas.

Durante un `MOVE`, una moneda se captura al pasar por su vértice. Al capturarla comienza inmediatamente una nueva ronda, la celda vuelve a valer -1 y aparece una nueva moneda en un vértice distinto de tu posición actual. El `MOVE` continúa hasta su destino, por lo que puede capturar más de una moneda.

Si no tienes suficientes fichas para completar un `MOVE` o ejecutar un `ASK` o `REROLL`, el programa termina y conserva el puntaje obtenido. `GOTO` e `IF` no consumen fichas.

Durante una partida pueden ejecutarse como máximo 10^6 instrucciones. Al alcanzar este límite, el programa termina y conserva su puntaje.

Un programa inválido obtiene puntaje 0.

Por ejemplo, el programa anterior tiene tamaño 5.

Tu puntaje dependerá tanto de la cantidad de monedas capturadas como del tamaño de tu programa.

Entrada

La primera línea de cada caso de prueba contiene 6 enteros n, m, s, T, q y r, la cantidad de vértices y aristas del grafo, el vértice inicial del jugador, la cantidad inicial de fichas, el costo de consultar la distancia a la moneda y el costo de terminar la ronda actual sin capturarla, respectivamente.

Las siguientes m líneas contienen 2 enteros u y v, indicando que existe una arista no dirigida entre los vértices u y v. $(1 \leq u, v \leq n)$

El grafo entregado es conexo y sus vértices están numerados desde 1 hasta n.

Salida

Este es un problema **output-only**. Deberás enviar un archivo `.zip` que contenga un programa para cada caso de prueba.

El programa correspondiente al caso i debe encontrarse en un archivo llamado `i.out`. Por ejemplo:

```
submission.zip
├── 1.out
├── 2.out
├── 3.out
├── 4.out
├── 5.out
├── 6.out
├── 7.out
└── 8.out
```

Cada archivo debe contener un programa válido escrito utilizando las instrucciones descritas anteriormente. Puedes utilizar una estrategia distinta para cada caso de prueba.

Dentro de cada archivo, cada instrucción o etiqueta debe escribirse en una línea distinta. También puedes incluir líneas vacías y comentarios comenzados por `#`.

Si el archivo correspondiente a un caso de prueba no está presente en el `.zip`, obtendrás 0 puntos para ese caso.

La ejecución de cada programa comienza en su primera instrucción y finaliza al alcanzar el final del archivo o cuando se cumple alguna de las condiciones de término descritas anteriormente.

Scoring

El problema contiene 8 casos de prueba públicos, que serán evaluados de manera independiente.

Para cada caso de prueba, tu programa será ejecutado 10 veces. Las posiciones de las monedas serán generadas utilizando semillas fijadas por el juez, por lo que enviar nuevamente el mismo programa producirá siempre el mismo resultado.

Sea C_i la cantidad total de monedas capturadas durante las 10 ejecuciones del caso i, y sea S_i el tamaño del programa correspondiente, es decir, su cantidad de instrucciones.

Definimos el multiplicador de tamaño como

$$M(S) = \begin{cases} 1 & \text{si} \quad S \leq 10 \\ \frac{1}{\sqrt{1+\left(\frac{S-10}{30}\right)^2}} & \text{si} \quad S > 10 \end{cases}.$$

El puntaje bruto de una solución para el caso i es

$$R_i = C_i \cdot M(S_i).$$

Luego, el puntaje bruto de cada caso se normaliza utilizando un valor de referencia fijado por el juez para dicho caso.

El porcentaje obtenido en el caso i es proporcional a su puntaje bruto y está limitado a un máximo de 100.

Los 8 casos tienen el mismo peso en el puntaje total. Por lo tanto, cada caso puede aportar como máximo 12.5 puntos, para un máximo total de 100 puntos.

Los programas de hasta 10 instrucciones no reciben penalización por tamaño. A partir de ahí, la penalización aumenta gradualmente a medida que crece el programa.

Si el programa correspondiente a un caso es inválido, su puntaje para ese caso será 0.

D. Guardias

Límite de tiempo: 1 segundos

Límite de memoria: 256 megabytes

El museo de la ciudad contrató guardias nocturnos. Cada guardia está parado en una posición fija con una linterna potente y apunta en una de cuatro direcciones: arriba, abajo, izquierda o derecha.

El museo es una grilla de $n \times m$ celdas, con filas numeradas de 1 a n (de arriba hacia abajo) y columnas de 1 a m (de izquierda a derecha). La luz de la linterna de un guardia que apunta hacia abajo ilumina una celda en la fila siguiente, tres celdas en la fila de más abajo, cinco en la siguiente, y así: en la fila k-ésima debajo del guardia ilumina un segmento de $2k-1$ celdas centrado en su columna. Las zonas iluminadas hacia la derecha, arriba e izquierda se definen de forma análoga. Un guardia no ilumina su propia celda, y la luz no se detiene en ningún obstáculo ni se apaga con la distancia.

Si dos guardias se iluminan **mutuamente** (cada uno está en la zona iluminada del otro), quedan cegados y puede ocurrir un accidente. El jefe de seguridad no quiere accidentes: para todo par de guardias A y B, o bien A no está en la zona iluminada de B, o bien B no está en la zona iluminada de A.

En el museo hay p **obras de arte**. La obra t está en la celda (x_t, y_t) y tiene un valor v_t. Aunque la luz llega muy lejos, un guardia solo distingue lo que está cerca. La **distancia** de una celda iluminada a su guardia es el número de filas que los separan (si el guardia apunta arriba o abajo) o el número de columnas que los separan (si apunta a la izquierda o a la derecha); es decir, la celda está en la k-ésima fila o columna de la zona iluminada. La **vigilancia** de un guardia es la suma de los valores de las obras que están en su zona iluminada a distancia **a lo más R**. Las obras más lejanas no cuentan, pero OJO que los guardias más lejanos quedan igualmente cegados. Una obra iluminada por varios guardias suma su valor **una vez por cada guardia** que la ilumina: cuantos más guardias la vigilan, más segura está.

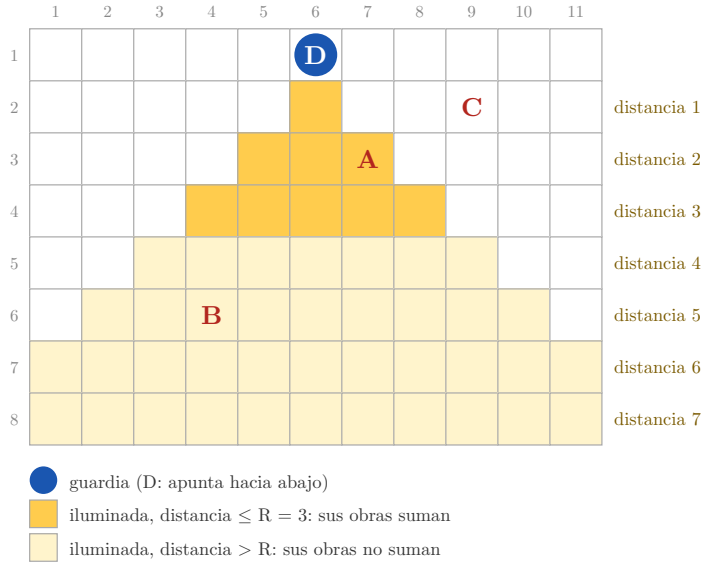


Figure 1: Zona iluminada del guardia en $(1, 6)$, que apunta hacia abajo, con $R = 3$. En la fila k-ésima debajo del guardia la luz cubre $2k-1$ celdas. La obra **A** está a distancia $2 \le R$ y suma a su vigilancia. La obra **B** está iluminada pero a distancia $5 > R$, así que no suma. La obra **C** no está iluminada.

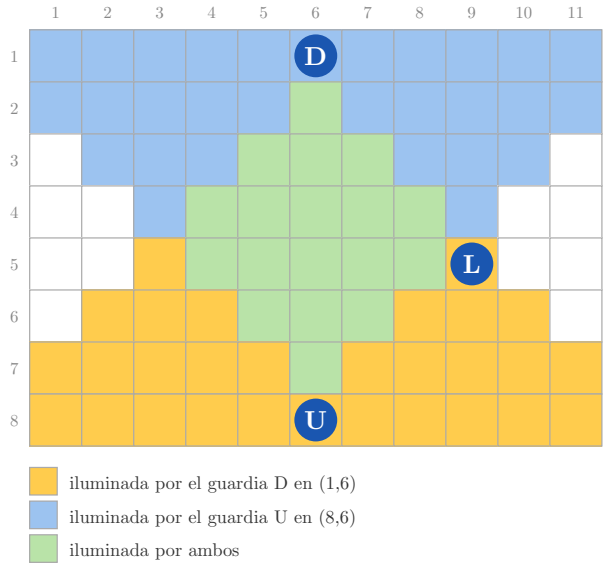


Figure 2: El guardia en $(1, 6)$ apunta hacia abajo y el guardia en $(8, 6)$ apunta hacia arriba: cada uno está en la zona iluminada del otro, así que se cegan y habría un accidente, aunque estén a distancia 7 (la luz ciega a cualquier distancia; R solo limita qué obras se vigilan). El guardia en $(5, 9)$ apunta a la izquierda: lo ilumina el guardia en $(1, 6)$, pero su propia luz va hacia la izquierda y no alcanza a ninguno de los otros dos, así que no hay accidente con él.

Los guardias no pueden moverse, pero sí girar. En un paso, el jefe puede girar a un guardia 90° en sentido horario o antihorario, y cada paso cuesta C (el guardia debe recibir instrucciones por radio). Girar 180° cuesta dos pasos.

Ayuda al jefe de seguridad a elegir la dirección final de cada guardia, sin accidentes, de modo que maximice

$$\text{Seguridad} = \sum_{\text{guardias}} \text{vigilancia} - C \cdot (\text{cantidad total de pasos}).$$

Entrada

La primera línea contiene cuatro enteros n, m, C y R ($1 \le n, m \le 5000$, $1 \le C \le 10^4$, $1 \le R \le 5000$).

Las siguientes n líneas contienen m caracteres cada una: `U`, `D`, `L`, `R` o `.`. `U` es un guardia que apunta hacia arriba, `D` hacia abajo, `L` hacia la izquierda, `R` hacia la derecha, y `.` es una celda sin guardia.

La siguiente línea contiene un entero p ($0 \le p \le 10^5$). Cada una de las siguientes p líneas contiene tres enteros x_t, y_t y v_t ($1 \le x_t \le n$, $1 \le y_t \le m$, $1 \le v_t \le 10^4$): la fila, la columna y el valor de una obra. No hay dos obras en la misma celda. Una obra puede estar en una celda con guardia; en ese caso, ese guardia no la vigila, pero otros sí pueden.

Salida

Imprime n líneas de m caracteres con la dirección final de cada guardia. Los guardias deben estar exactamente en las mismas posiciones que en la entrada.

Si hay dos guardias que se iluminan mutuamente, o el formato es incorrecto, el puntaje del caso es 0.

Scoring

Cada caso se evalúa por separado. Si S_i es la seguridad de tu configuración en el caso i y S_i^* es la mejor seguridad conocida por la organización, sea $x_i = S_i/S_i^*$. Tu puntaje en ese caso es $f(x_i)$, donde

$$f(x) = \begin{cases} 0 & \text{si } x \le 0 \\ 100x & \text{si } 0 \le x \le 0.5 \\ 50 + \frac{500}{9}(x - 0.5) & \text{si } 0.5 \le x \le 0.95 \\ 75 + 500(x - 0.95) & \text{si } 0.95 \le x \le 1 \\ 100 & \text{si } x \ge 1 \end{cases}$$

Es decir, f une con segmentos rectos estos puntos:

$x = S_i/S_i^*$	0	0.5	0.95	1
Puntaje	0	50	75	100

Por ejemplo, $x = 0.8$ da 66.67 puntos y $x = 0.99$ da 95 puntos.

En todos los casos de prueba $S_i^* > 0$. El puntaje final es el promedio sobre todos los casos.

Ejemplos

Entrada 1	Salida 1
4 5 2 3 R...L .D... ...U. L.... 4 2 5 5 4 3 4 1 3 3 4 5 6	R...D .R... ...R. U....

E. Escribiendo TCS por todas partes

Límite de tiempo: 1 segundos

Límite de memoria: 256 megabytes

A Ximena le encanta trabajar en TCS, pero además le apasiona la sigla **TCS**. Por eso quiere diseñar la publicidad definitiva: un cartel en el que la palabra `TCS` aparezca la mayor cantidad de veces posible.

El cartel es un rectángulo de n filas y m columnas, y en cada casilla se escribe exactamente una letra. Como cualquier otra letra solo estorbaría, Ximena usará únicamente las letras `T`, `C` y `S`.

Una **aparición** de `TCS` es una de estas dos cosas:

- una aparición **horizontal**: tres casillas consecutivas $(i, j), (i, j+1), (i, j+2)$ de una misma fila que se leen `T`, `C`, `S` de izquierda a derecha;
- una aparición **vertical**: tres casillas consecutivas $(i, j), (i+1, j), (i+2, j)$ de una misma columna que se leen `T`, `C`, `S` de arriba hacia abajo.

Antes de pintar el cartel, Ximena quiere saber cuál es la mayor cantidad de apariciones de `TCS` que se pueden lograr, y tener un diseño que lo consiga. ¡Ayúdala!

Entrada

La primera línea contiene un entero t ($1 \le t \le 10^4$), la cantidad de casos de prueba.

Cada caso de prueba es una línea con dos enteros n y m ($1 \le n, m \le 10^6$).

La suma de $n \cdot m$ sobre todos los casos de prueba no supera 10^6.

Salida

Para cada caso de prueba, imprime la mayor cantidad posible de apariciones k. Luego imprime n líneas de m caracteres cada una (`T`, `C` o `S`): un cartel que tenga exactamente k apariciones.

Si hay varios carteles óptimos, puedes imprimir cualquiera de ellos.

Ejemplos

Entrada 1	Salida 1
4 1 5 3 3 4 5 6 6	1 TCSTC 3 TCS TCS TCS 8 STCST TCSTC CSTCS STCST 16 TCSTCS CSTCST STCSTC TCSTCS CSTCST STCSTC

Nota

En el segundo caso de prueba, las tres filas se leen `TCS`, lo que da 3 apariciones. Uno podría esperar lograr 6 en un cartel de 3×3 (una en cada fila y una en cada columna), pero es imposible: la casilla $(1, 2)$ tendría que ser `C` para que la fila 1 se lea `TCS`, y a la vez `T` para que la columna 2 se lea `TCS`.

En el tercer caso de prueba, el cartel tiene 3 apariciones horizontales y 5 verticales.

F. El dron perdido

Límite de tiempo: 2 segundos

Límite de memoria: 256 megabytes

Un dron de reparto sobrevuela una zona de $n \times m$ celdas cuando, en plena garúa, pierde la señal. Cada celda tiene cobertura de antena o no la tiene. El dron está en alguna celda **sin** cobertura, pero no sabes en cuál.

Aunque no puedes recibir nada del dron, aún puedes enviarle órdenes. Las filas se numeran de 1 a n de arriba hacia abajo, y las columnas de 1 a m de izquierda a derecha. Cada orden es una letra:

Orden	**Efecto**
`U`	Una celda hacia arriba (la fila disminuye en 1).
`D`	Una celda hacia abajo (la fila aumenta en 1).
`L`	Una celda hacia la izquierda (la columna disminuye en 1).
`R`	Una celda hacia la derecha (la columna aumenta en 1).

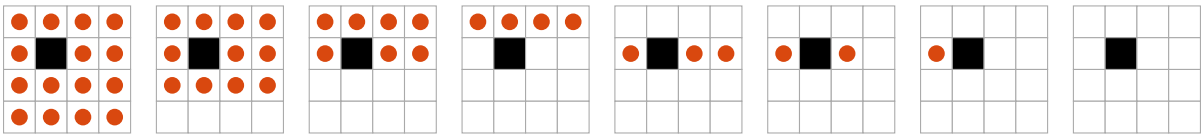
El dron obedece estas reglas:

1. **Geocerca:** si una orden lo sacaría de la zona, el dron se queda en su celda actual.
2. **Reconexión:** si después de alguna orden el dron está en una celda con cobertura, se reconecta y lo encuentras. Las celdas con cobertura no bloquean el paso.

Tu tarea es encontrar una secuencia de órdenes, lo más corta posible, que garantice encontrar el dron sin importar en qué celda sin cobertura empezó. La secuencia debe tener como máximo 5000 órdenes.

Ejemplo

En la siguiente zona de 4×4 la única celda con cobertura (negra) está en la fila 2, columna 2. Los puntos marcan las celdas donde el dron todavía podría estar después de cada orden de la secuencia `UUUDLLR`, de 7 órdenes:



inicio — 1. orden: `U` — 2. orden: `U` — 3. orden: `U` — 4. orden: `D` — 5. orden: `L` — 6. orden: `L` — 7. orden: `R`

Tras la tercera `U`, todos los drones que no se reconectaron quedaron apretados contra el borde superior por la geocerca. Después de la última orden ya no queda ninguna posición posible: la secuencia garantiza encontrar el dron. No existe una secuencia válida más corta para esta zona.

Entrada

Este es un problema **output-only**. Recibirás el archivo `problem-f-inputs.zip` con 10 casos de prueba: `01.png`, `02.png`, …, `10.png`.

Cada caso es el mapa de cobertura de una zona, como imagen PNG en escala de grises de m píxeles de ancho y n píxeles de alto ($100 \leq n, m \leq 500$). Cada píxel es una celda; el píxel de la fila i y columna j de la imagen corresponde a la celda de la fila i y columna j de la zona.

- Negro (valor 0): celda con cobertura.
- Blanco (valor 255): celda sin cobertura.

Cada mapa tiene al menos una celda de cada tipo. Algunos de ellos:



Salida

Deberás enviar un archivo `.zip` que contenga la respuesta para cada caso de prueba. La respuesta del caso `i.png` debe encontrarse en un archivo llamado `i.out`. Por ejemplo:

```
submission.zip
├── 01.out
├── 02.out
├── ...
└── 10.out
```

Puedes omitir algunos archivos; si el archivo de un caso no está presente, obtendrás 0 puntos para ese caso.

Cada archivo debe contener una sola cadena formada por los caracteres `U`, `D`, `L` y `R`: tu secuencia de órdenes para ese caso, de largo como máximo 5000.

Puntaje

Los 10 casos se evalúan de manera independiente y cada uno vale como máximo 10 puntos, para un máximo total de 100 puntos.

Una respuesta es **válida** si respeta el formato y encuentra al dron sin importar en qué celda sin cobertura empezó. Una respuesta inválida obtiene 0 puntos en ese caso.

Sea L_i el largo de tu secuencia para el caso i, y sea B_i el largo de la secuencia de referencia del jurado para ese caso. Si tu respuesta es válida, tu puntaje en el caso i es

$$P_i = 10 \cdot \min\left(1, \frac{B_i}{L_i}\right)^2.$$

Es decir, igualar o superar al jurado otorga los 10 puntos completos, y una secuencia el doble de larga que la del jurado otorga 2.5 puntos.