A. Alpacas e Guanacos

Límite de tiempo: 2 segundos

Límite de memoria: 256 megabytes

Em pleno altiplano chileno, os cuidadores de uma fazenda estão com uma tremenda confusão com seus animais. Em um terreno aberto pastam dois tipos de animais: alpacas e guanacos.

O problema é que as alpacas e os guanacos não se dão bem: se você prender os dois em um mesmo curral, eles começam a cuspir uns nos outros e brigar, armando o maior caos. Além disso, como o terreno não tem cercas no perímetro, se um curral não ficar completamente fechado em relação à borda do mapa, os animais fogem para a cordilheira.

Para salvar a fazenda, você precisa montar currais fechados usando um catálogo de cercas. Seu objetivo é posicioná-las de forma inteligente para maximizar o ganho líquido.

O terreno corresponde a uma grade quadrada de $N \times N$ células, com linhas e colunas numeradas de 1 a N. Inicialmente, cada célula está vazia ou contém no máximo um animal.

Um curral é uma componente conexa de células livres (conectadas ortogonalmente: acima, abaixo, à esquerda ou à direita) que está completamente cercada por cercas e isolada do exterior. Para que um curral seja válido e gere felicidade, ele deve estar estritamente fechado, sem alcançar as bordas do terreno (linha 1, linha N, coluna 1 ou coluna N); se uma região de células livres toca o perímetro exterior, ela é considerada campo aberto e os animais nela fogem para a cordilheira. Da mesma forma, todos os animais dentro do curral devem pertencer à mesma espécie, já que, se alpacas e guanacos forem presos juntos, eles começam a cuspir uns nos outros e brigar, armando o maior caos e arruinando a felicidade do curral. Por fim, um curral fechado que não contenha nenhum animal também não produz ganho algum.

Cada animal precisa de pasto para se alimentar, mas sua capacidade de consumo tem um limite: cada indivíduo pode aproveitar no máximo $A_{\max}$ células de pastagem. Se você construir um curral grande demais para a quantidade de animais que há dentro, o pasto excedente não é aproveitado, seca ou se enche de palha-brava, sem render nem um centavo a mais.

Dessa forma, se um curral fechado válido P contém $C(P) \geq 1$ animais da mesma espécie e abrange uma área de $\text{Área}(P)$ células livres, sua área efetiva de pastagem é calculada como:

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

Os animais gostam de compartilhar, então a felicidade obtida pelo curral corresponde à quantidade de animais multiplicada pelo pasto que efetivamente aproveitam:

$$\text{Felicidade}(P) = C(P) \cdot \text{ÁreaEfetiva}(P)$$

Qualquer região que não cumpra as condições de um curral válido (por exemplo, currais abertos que tocam a borda, currais com espécies misturadas ou currais vazios) tem felicidade 0.

Você tem à disposição um catálogo com M tipos de peças de cerca modulares, identificadas de 1 a M segundo sua ordem de aparição na entrada. Cada peça tem dimensões de altura H e largura W, um custo base B e uma matriz onde `#` representa um bloco de cerca sólida e `.` representa espaço livre. Você pode usar quantas peças quiser e girá-las em $0°, 90°, 180°$ ou $270°$ no sentido horário.

Ao posicionar uma peça, indica-se seu número no catálogo ($1 \leq i \leq M$), a coordenada (r, c) de seu canto superior esquerdo ($1 \leq r, c \leq N$) e seu ângulo de rotação. Toda peça posicionada deve ficar completamente dentro dos limites do terreno ($r + H' - 1 \leq N$ e $c + W' - 1 \leq N$, onde H' e W' são suas dimensões já rotacionadas). Os blocos de cerca de peças diferentes podem se cruzar e se sobrepor livremente na grade. No entanto, pelo bem-estar dos animais, não é permitido construir em cima de nenhum deles: se um bloco `#` for colocado sobre uma célula ocupada por uma alpaca ou um guanaco, a solução é desclassificada e a pontuação final é 0.

Na cordilheira as cercas são escassas, e os operários vão cobrar mais caro à medida que você pedir para repetir o mesmo modelo de cerca.

A m-ésima vez que você usa uma peça do tipo k ($m = 1, 2, 3, ...$), sua colocação custa:

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

Assim, se você usar K_k cópias da peça k, o custo acumulado desse tipo de peça é:

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

O custo total de construção é a soma dos custos de todas as peças usadas:

$$\text{CustoTotalCercas} = \sum_{k=1}^{M} \text{CustoTotal}_k$$

Seu ganho líquido é a soma das felicidades brutas de todos os currais válidos menos o que você gastou em cercas:

$$\text{Pontuação} = \max\left(0, \sum_{P} \text{Felicidade}(P) - \text{CustoTotalCercas}\right)$$

Qualquer solução inválida (peças que saem da grade, cercas em cima dos animais ou formato incorreto) obtém pontuação 0. Seu objetivo é maximizar essa pontuação líquida.

Entrada

A primeira linha contém dois inteiros N e $A_{\max}$ ($10 \le N \le 70$, $10 \le A_{\max} \le 100$), o tamanho do terreno ($N \times N$) e o limite de pastagem por animal.

A segunda linha contém dois inteiros C e M ($1 \le C \le 100$, $1 \le M \le 50$), a quantidade de animais e a quantidade de tipos de peças no catálogo.

As C linhas seguintes descrevem os animais. Cada uma tem três inteiros r_i, c_i e espécie_i ($1 \le r_i, c_i \le N$, $\text{espécie}_i \in \{1, 2\}$): a linha, a coluna e a espécie do animal (`1` para Alpaca, `2` para Guanaco). Não há dois animais na mesma célula.

Em seguida vêm as descrições dos M tipos de peças do catálogo, segundo sua ordem de aparição (1 a M):

- Uma linha com três inteiros: H, W e B ($1 \le H, W \le N$, $B \ge 1$), com a altura, a largura e o custo base da peça.
- H linhas com uma string de W caracteres (`#` ou `.`) que mostram o formato da peça.

Salida

A primeira linha deve conter um inteiro $K \ge 0$, a quantidade total de peças de cerca que você vai posicionar.

Em seguida devem vir K linhas descrevendo a colocação de cada peça por meio de quatro inteiros separados por espaço:

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

onde:

- i ($1 \le i \le M$) é o número da peça segundo sua ordem de aparição no catálogo.
- r e c ($1 \le r, c \le N$) são as coordenadas da linha e da coluna (indexadas a partir de 1) do canto superior esquerdo da peça na grade após aplicar a rotação.
- `"rotação"` é o ângulo em graus (0, 90, 180 ou 270).

Se preferir não colocar nenhuma cerca, imprima apenas uma linha com o número `0`.

Pontuação

Cada caso de teste é avaliado separadamente. Para cada caso i, os pontos serão normalizados em uma escala de 0 a 100 em função da melhor solução obtida pela organização (S_i^*):

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

onde S_i é o ganho líquido obtido pela sua solução. Se a solução for inválida em um caso, a pontuação desse caso será 0.

A pontuação final da sua submissão corresponderá à média das pontuações obtidas em todos os casos de teste:

$$\text{PontuaçãoFinal} = \frac{1}{T}\sum_{i=1}^{T} \text{Pontuação}_i$$

onde T é a quantidade total de casos de teste.

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

No exemplo acima, o terreno é de 8×8 e $A_{\max} = 10$. Há 3 animais: duas alpacas em $(3,3)$ e $(4,3)$, e um guanaco em $(7,6)$.

A solução posiciona 4 peças de cerca formando um curral fechado entre as linhas 2 a 5 e as colunas 2 a 5:

- São usadas 2 cópias da peça 1 (horizontal 1×4, custo base 1) em $(2,2,0)$ e $(5,2,0)$.
- São usadas 2 cópias da peça 2 (vertical 4×1, custo base 1) em $(2,2,0)$ e $(2,5,0)$.

Dentro do curral ficam 4 células livres ($(3,3)$, $(3,4)$, $(4,3)$ e $(4,4)$), isoladas da borda do mapa (linhas 1 e 8, colunas 1 e 8). Como dentro há apenas alpacas, o curral é válido:

- $\text{ÁreaEfetiva} = \min(4, 2 \cdot 10) = 4$.
- $\text{Felicidade} = 2 \cdot 4 = 8$.

O custo das peças com cobrança progressiva é:

- Peça 1: $1 \cdot \frac{2 \cdot 3}{2} = 3$.
- Peça 2: $1 \cdot \frac{2 \cdot 3}{2} = 3$.
- $\text{CustoTotalCercas} = 3 + 3 = 6$.

O guanaco em $(7,6)$ ficou do lado de fora, em campo aberto (fugiu), portanto não gera conflito nem felicidade. A pontuação líquida final é:

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

B. Minas e Masmorras

Límite de tiempo: 2 segundos

Límite de memoria: 256 megabytes

Emilia estava jogando com suas amigas *Minas e Masmorras*, o MMO RPG do momento. Elas decidiram se aventurar nas *Minas de Rayse* e avançavam em grupo quando Emilia parou por um instante para beber água. Nesse momento, o resto do grupo caiu em uma armadilha de teletransporte! Por pura sorte, Emilia ficou fora do alcance da armadilha.

Depois de usar suas habilidades, ela descobriu que as *Minas de Rayse* são compostas por N salas ligadas por passagens estreitas, e que atravessar cada uma delas leva exatamente 1 minuto. Graças às aulas de matemática discreta, Emilia reconheceu que as *Minas de Rayse* têm formato de árvore.

Cada uma das K amigas de Emilia está em uma sala p_i diferente das demais e tem h_i pontos de vida. A cada minuto que passam sem receber a ajuda de Emilia, elas perdem 1 ponto de vida; se chegarem a 0 pontos de vida, suas personagens morrem permanentemente.

Por sorte, Emilia pode salvá-las. Mais precisamente, Emilia salva sua i-ésima amiga se chegar à sala p_i no minuto h_i ou antes. Salvar uma amiga é instantâneo, então Emilia pode fazê-lo mesmo que apenas passe pela sala.

Ajude Emilia a salvar o maior número possível de amigas!

Entrada

A primeira linha contém 2 inteiros N ($1 \leq N \leq 2 \cdot 10^5$) e s ($1 \leq s \leq N$), a quantidade de salas das *Minas de Rayse* e a sala onde Emilia se encontra inicialmente.

As próximas $N-1$ linhas contêm pares de inteiros $1 \leq u, v \leq N$, indicando que as salas u e v estão conectadas por uma passagem estreita. É garantido que as salas e as passagens estreitas formam uma árvore.

A linha seguinte contém 1 inteiro K ($1 \leq K \leq 20$), o número de amigas de Emilia.

Seguem K linhas; a i-ésima linha contém 2 inteiros p_i ($1 \leq p_i \leq N$) e h_i ($1 \leq h_i \leq 2 \cdot 10^5$), a sala onde se encontra a i-ésima amiga de Emilia e seus pontos de vida, respectivamente.

É garantido que todos os p_i são distintos.

Salida

Imprima um único inteiro, o número máximo de amigas que Emilia pode 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

No exemplo 1, Emilia já está na sala 1 no minuto 0, quando sua amiga ainda tem 1 ponto de vida, então ela a salva imediatamente.

No exemplo 2, Emilia leva 2 minutos para chegar à sala 3, mas sua amiga perdeu todos os pontos de vida no minuto 1, então Emilia não consegue ajudá-la a tempo.

No exemplo 3, Emilia salva a amiga da sala 1 no minuto 0, depois caminha até a sala 3 e chega no minuto 2, exatamente quando os pontos de vida da amiga da sala 3 chegam a 0.

C. Cassino Compacto

Límite de tiempo: 2 segundos

Límite de memoria: 512 megabytes

O Cassino Compacto (ex Cassino Codificadas) voltou a abrir suas portas ao público depois de anos sem funcionar. Entre os motivos de seu fechamento estava o uso excessivo de memória de suas máquinas, por isso decidiram mudar completamente de abordagem: desenvolver jogos de azar que recompensassem o uso eficiente de recursos.

Como parte da equipe de desenvolvimento do Cassino Compacto, você deverá projetar uma estratégia para testar um dos novos jogos que obtenha a maior pontuação possível (para calibrar alguns parâmetros e evitar que o cassino fique no prejuízo).

Você jogará em um grafo conexo não direcionado com n vértices e m arestas. Sua posição inicial será o vértice s.

O jogo é dividido em rodadas. No início de cada rodada, uma moeda invisível aparece em um vértice escolhido uniformemente ao acaso entre todos os vértices diferentes da sua posição atual.

Para encontrá-la, você pode consultar a distância mínima entre sua posição e a moeda pagando q fichas, ou se mover para outro vértice pagando uma ficha por cada aresta percorrida.

Ao capturar uma moeda, uma nova rodada começa imediatamente.

Você também pode encerrar uma rodada sem capturar a moeda pagando r fichas.

O jogo termina quando você fica sem fichas.

Sua estratégia deverá ser escrita usando a pequena linguagem da máquina do Cassino Compacto.

A máquina possui uma única célula de memória que armazena um inteiro. No início de cada rodada, seu valor é -1.

Seu programa pode conter as seguintes instruções:

Instrução	**Descrição**
`MOVE v`	Percorre um caminho mínimo até o vértice `v`, pagando uma ficha por cada aresta percorrida. Se existirem vários caminhos mínimos, em cada passo é escolhido o vizinho de menor índice que permita continuar por um deles. Se não houver fichas suficientes para completar o percurso, o programa termina imediatamente.
`ASK`	Paga `q` fichas e armazena na célula de memória a distância mínima entre sua posição e a moeda. Se não houver fichas suficientes, o programa termina imediatamente.
`REROLL`	Paga `r` fichas e inicia uma nova rodada sem capturar a moeda atual. Se não houver fichas suficientes, o programa termina imediatamente.
`GOTO etiqueta`	Continua a execução a partir da instrução identificada por `etiqueta`.
`IF d etiqueta`	Se o valor armazenado na célula de memória for igual a `d`, continua a execução a partir de `etiqueta`. Caso contrário, continua com a próxima instrução.

A máquina executa as instruções começando pela primeira. Exceto por `GOTO` e `IF`, após cada instrução a execução continua com a próxima. Se chegar ao fim do programa, ele termina.

Você pode declarar etiquetas escrevendo seu nome seguido de `:` antes de uma instrução. Cada etiqueta deve ter um nome único e estar associada a uma instrução existente.

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

perto:
GOTO inicio
```

Etiquetas, linhas vazias e comentários não contam como instruções. Os comentários começam com `#` e se estendem até o fim da linha. As instruções diferenciam letras maiúsculas de minúsculas.

Cada instrução ocupa exatamente uma unidade de memória. O **tamanho do seu programa** é a quantidade de instruções que ele contém, independentemente de quantas vezes elas sejam executadas.

Durante um `MOVE`, uma moeda é capturada ao passar pelo vértice em que ela está. Ao capturá-la, uma nova rodada começa imediatamente, a célula volta a valer -1 e uma nova moeda aparece em um vértice diferente da sua posição atual. O `MOVE` continua até seu destino, portanto pode capturar mais de uma moeda.

Se você não tiver fichas suficientes para completar um `MOVE` ou executar um `ASK` ou `REROLL`, o programa termina e mantém a pontuação obtida. `GOTO` e `IF` não consomem fichas.

Durante uma partida podem ser executadas no máximo 10^6 instruções. Ao atingir esse limite, o programa termina e mantém sua pontuação.

Um programa inválido recebe pontuação 0.

Por exemplo, o programa anterior tem tamanho 5.

Sua pontuação dependerá tanto da quantidade de moedas capturadas quanto do tamanho do seu programa.

Entrada

A primeira linha de cada caso de teste contém 6 inteiros n, m, s, T, q e r, representando, respectivamente, a quantidade de vértices e arestas do grafo, o vértice inicial do jogador, a quantidade inicial de fichas, o custo de consultar a distância até a moeda e o custo de encerrar a rodada atual sem capturá-la.

As próximas m linhas contêm 2 inteiros u e v, indicando que existe uma aresta não direcionada entre os vértices u e v. $(1 \leq u, v \leq n)$

O grafo fornecido é conexo e seus vértices são numerados de 1 até n.

Saída

Este é um problema **output-only**. Você deverá enviar um arquivo `.zip` contendo um programa para cada caso de teste.

O programa correspondente ao caso i deve estar em um arquivo chamado `i.out`. Por exemplo:

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

```
├── 7.out
└── 8.out
```

Cada arquivo deve conter um programa válido escrito utilizando as instruções descritas anteriormente. Você pode utilizar uma estratégia diferente para cada caso de teste.

Dentro de cada arquivo, cada instrução ou etiqueta deve ser escrita em uma linha diferente. Você também pode incluir linhas vazias e comentários iniciados por `#`.

Se o arquivo correspondente a um caso de teste não estiver presente no `.zip`, você receberá 0 pontos nesse caso.

A execução de cada programa começa em sua primeira instrução e termina ao chegar ao fim do arquivo ou quando alguma das condições de término descritas anteriormente for satisfeita.

Scoring

O problema contém 8 casos de teste públicos, que serão avaliados de forma independente.

Para cada caso de teste, seu programa será executado 10 vezes. As posições das moedas serão geradas utilizando sementes fixadas pelo juiz, portanto, enviar novamente o mesmo programa produzirá sempre o mesmo resultado.

Seja C_i a quantidade total de moedas capturadas durante as 10 execuções do caso i, e seja S_i o tamanho do programa correspondente, ou seja, sua quantidade de instruções.

Definimos o multiplicador de tamanho como

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

A pontuação bruta de uma solução para o caso i é

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

Em seguida, a pontuação bruta de cada caso é normalizada utilizando um valor de referência fixado pelo juiz para esse caso.

A porcentagem obtida no caso i é proporcional à sua pontuação bruta e limitada a um máximo de 100.

Os 8 casos têm o mesmo peso na pontuação total. Portanto, cada caso pode contribuir com no máximo 12.5 pontos, para um máximo total de 100 pontos.

Programas com até 10 instruções não recebem penalização por tamanho. A partir desse ponto, a penalização aumenta gradualmente à medida que o programa cresce.

Se o programa correspondente a um caso for inválido, sua pontuação para esse caso será 0.

D. Guardas

Límite de tiempo: 1 segundos

Límite de memoria: 256 megabytes

O museu da cidade contratou guardas noturnos. Cada guarda está parado em uma posição fixa com uma lanterna potente e aponta em uma de quatro direções: para cima, para baixo, para a esquerda ou para a direita.

O museu é uma grade de $n \times m$ células, com linhas numeradas de 1 a n (de cima para baixo) e colunas numeradas de 1 a m (da esquerda para a direita). A luz da lanterna de um guarda que aponta para baixo ilumina uma célula na linha seguinte, três células na linha abaixo dela, cinco na seguinte, e assim por diante: na k-ésima linha abaixo do guarda, ilumina um segmento de $2k-1$ células centrado na sua coluna. As zonas iluminadas para a direita, para cima e para a esquerda são definidas de forma análoga. Um guarda não ilumina a sua própria célula, e a luz não é bloqueada por nenhum obstáculo nem se apaga com a distância.

Se dois guardas se iluminam **mutuamente** (cada um está na zona iluminada do outro), eles ficam ofuscados e pode ocorrer um acidente. O chefe de segurança não quer acidentes: para todo par de guardas A e B, ou A não está na zona iluminada de B, ou B não está na zona iluminada de A.

No museu há p **obras de arte**. A obra t está na célula (x_t, y_t) e tem valor v_t. Embora a luz chegue muito longe, um guarda só distingue o que está perto. A **distância** de uma célula iluminada até o seu guarda é o número de linhas que os separam (se o guarda aponta para cima ou para baixo) ou o número de colunas que os separam (se aponta para a esquerda ou para a direita); ou seja, a célula está na k-ésima linha ou coluna da zona iluminada. A **vigilância** de um guarda é a soma dos valores das obras que estão na sua zona iluminada a distância **no máximo** R. As obras mais distantes não contam, mas ATENÇÃO: os guardas mais distantes ficam ofuscados do mesmo jeito. Uma obra iluminada por vários guardas soma o seu valor **uma vez para cada guarda** que a ilumina: quanto mais guardas a vigiam, mais segura ela está.

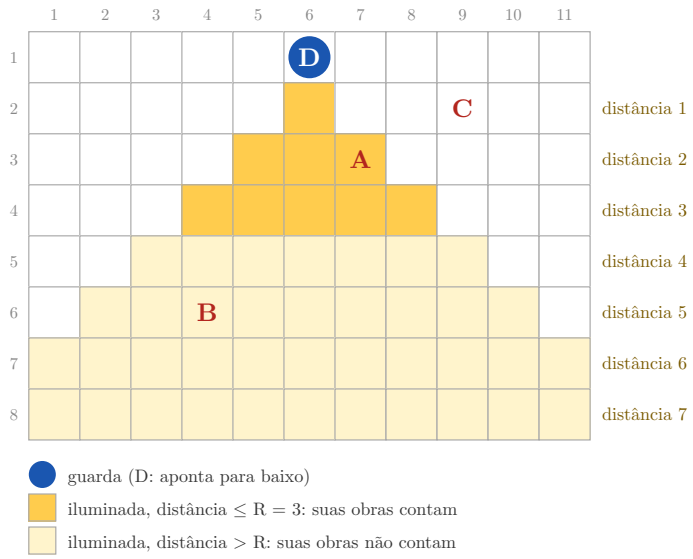


Figure 1: Zona iluminada do guarda em $(1, 6)$, que aponta para baixo, com $R = 3$. Na k-ésima linha abaixo do guarda, a luz cobre $2k-1$ células. A obra **A** está a distância $2 \le R$ e soma à sua vigilância. A obra **B** está iluminada, mas a distância $5 > R$, então não soma. A obra **C** não está iluminada.

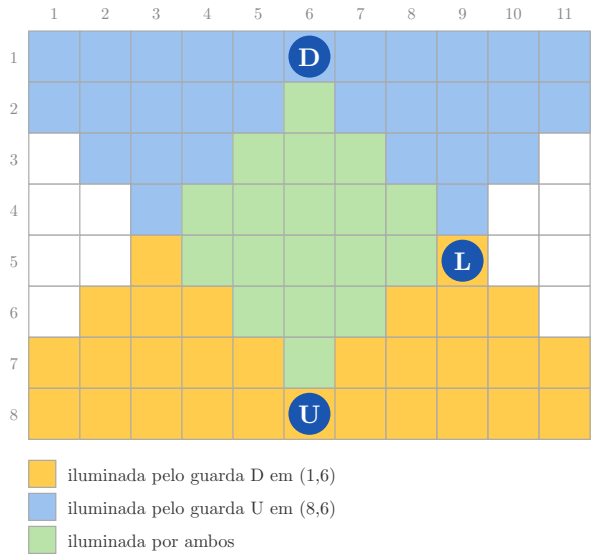


Figure 2: O guarda em $(1, 6)$ aponta para baixo e o guarda em $(8, 6)$ aponta para cima: cada um está na zona iluminada do outro, então eles se ofuscam e haveria um acidente, mesmo estando a distância 7 (a luz ofusca a qualquer distância; R só limita quais obras são vigiadas). O guarda em $(5, 9)$ aponta para a esquerda: ele é iluminado pelo guarda em $(1, 6)$, mas a sua própria luz vai para a esquerda e não alcança nenhum dos outros dois, então não há acidente com ele.

Os guardas não podem se mover, mas podem girar. Em um passo, o chefe pode girar um guarda 90° no sentido horário ou anti-horário, e cada passo custa C (o guarda precisa receber instruções pelo rádio). Girar 180° custa dois passos.

Ajude o chefe de segurança a escolher a direção final de cada guarda, sem acidentes, de modo a maximizar

$$\text{Segurança} = \sum_{\text{guardas}} \text{vigilância} - C \cdot (\text{quantidade total de passos}).$$

Entrada

A primeira linha contém quatro inteiros n, m, C e R ($1 \le n, m \le 5000$, $1 \le C \le 10^4$, $1 \le R \le 5000$).

As próximas n linhas contêm m caracteres cada uma: `U`, `D`, `L`, `R` ou `.`. `U` é um guarda que aponta para cima, `D` para baixo, `L` para a esquerda, `R` para a direita, e `.` é uma célula sem guarda.

A linha seguinte contém um inteiro p ($0 \le p \le 10^5$). Cada uma das próximas p linhas contém três inteiros x_t, y_t e v_t ($1 \le x_t \le n$, $1 \le y_t \le m$, $1 \le v_t \le 10^4$): a linha, a coluna e o valor de uma obra. Não há duas obras na mesma célula. Uma obra pode estar em uma célula com guarda; nesse caso, esse guarda não a vigia, mas outros guardas podem vigiá-la.

Salida

Imprima n linhas de m caracteres com a direção final de cada guarda. Os guardas devem estar exatamente nas mesmas posições que na entrada.

Se houver dois guardas que se iluminam mutuamente, ou se o formato estiver incorreto, a pontuação do caso é 0.

Pontuação

Cada caso é avaliado separadamente. Se S_i é a segurança da sua configuração no caso i e S_i^* é a melhor segurança conhecida pela organização, seja $x_i = S_i/S_i^*$. A sua pontuação nesse caso é $f(x_i)$, onde

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

Ou seja, f liga com segmentos de reta estes pontos:

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

Por exemplo, $x = 0.8$ dá 66.67 pontos e $x = 0.99$ dá 95 pontos.

Em todos os casos de teste, $S_i^* > 0$. A pontuação final é a média sobre todos os 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. TCS por toda parte

Límite de tiempo: 1 segundos

Límite de memoria: 256 megabytes

A Ximena adora trabalhar na TCS, mas, além disso, é apaixonada pela sigla **TCS**. Por isso, ela quer criar a propaganda definitiva: um cartaz em que a palavra `TCS` apareça o maior número de vezes possível.

O cartaz é um retângulo com n linhas e m colunas, e em cada casa se escreve exatamente uma letra. Como qualquer outra letra só atrapalharia, a Ximena vai usar apenas as letras `T`, `C` e `S`.

Uma **ocorrência** de `TCS` é uma destas duas coisas:

- uma ocorrência **horizontal**: três casas consecutivas $(i, j), (i, j+1), (i, j+2)$ de uma mesma linha que formam `T`, `C`, `S` da esquerda para a direita;
- uma ocorrência **vertical**: três casas consecutivas $(i, j), (i+1, j), (i+2, j)$ de uma mesma coluna que formam `T`, `C`, `S` de cima para baixo.

Antes de pintar o cartaz, a Ximena quer saber qual é o maior número de ocorrências de `TCS` que é possível obter, e ter um desenho que alcance esse número. Ajude-a!

Entrada

A primeira linha contém um inteiro t ($1 \le t \le 10^4$), o número de casos de teste.

Cada caso de teste é uma linha com dois inteiros n e m ($1 \le n, m \le 10^6$).

A soma de $n \cdot m$ sobre todos os casos de teste não excede 10^6.

Salida

Para cada caso de teste, imprima o maior número possível de ocorrências k. Em seguida, imprima n linhas com m caracteres cada (`T`, `C` ou `S`): um cartaz que tenha exatamente k ocorrências.

Se houver vários cartazes ótimos, você pode imprimir qualquer um deles.

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

No segundo caso de teste, as três linhas formam `TCS`, o que dá 3 ocorrências. Poderíamos esperar conseguir 6 em um cartaz 3×3 (uma em cada linha e uma em cada coluna), mas isso é impossível: a casa $(1, 2)$ teria que ser `C` para que a linha 1 forme `TCS` e, ao mesmo tempo, `T` para que a coluna 2 forme `TCS`.

No terceiro caso de teste, o cartaz tem 3 ocorrências horizontais e 5 verticais.

F. O drone perdido

Límite de tiempo: 2 segundos

Límite de memoria: 256 megabytes

Um drone de entregas sobrevoa uma área de $n \times m$ células quando, em plena garoa, perde o sinal. Cada célula tem ou não tem cobertura de antena. O drone está em alguma célula **sem** cobertura, mas você não sabe em qual.

Embora você não possa receber nada do drone, ainda pode enviar comandos a ele. As linhas são numeradas de 1 a n de cima para baixo, e as colunas de 1 a m da esquerda para a direita. Cada comando é uma letra:

Comando	**Efeito**
`U`	Uma célula para cima (a linha diminui em 1).
`D`	Uma célula para baixo (a linha aumenta em 1).
`L`	Uma célula para a esquerda (a coluna diminui em 1).
`R`	Uma célula para a direita (a coluna aumenta em 1).

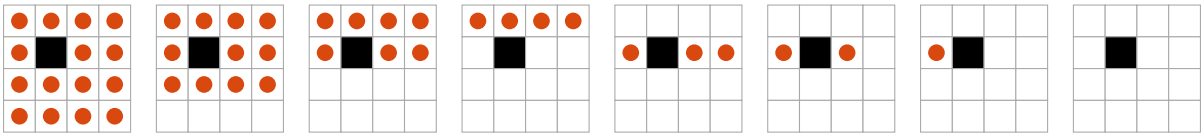
O drone obedece a estas regras:

1. **Geocerca:** se um comando o tirasse da área, o drone permanece em sua célula atual.
2. **Reconexão:** se, após algum comando, o drone estiver em uma célula com cobertura, ele se reconecta e você o encontra. As células com cobertura não bloqueiam a passagem.

Sua tarefa é encontrar uma sequência de comandos, a mais curta possível, que garanta encontrar o drone independentemente da célula sem cobertura em que ele começou. A sequência deve ter no máximo 5000 comandos.

Exemplo

Na área de 4×4 a seguir, a única célula com cobertura (preta) está na linha 2, coluna 2. Os pontos marcam as células onde o drone ainda poderia estar após cada comando da sequência `UUUDLLR`, de 7 comandos:



Após o terceiro `U`, todos os drones que não se reconectaram ficaram espremidos contra a borda superior pela geocerca. Após o último comando, não resta nenhuma posição possível: a sequência garante encontrar o drone. Não existe sequência válida mais curta para esta área.

Entrada

Este é um problema **output-only**. Você receberá o arquivo `problem-f-inputs.zip` com 10 casos de teste: `01.png`, `02.png`, …, `10.png`.

Cada caso é o mapa de cobertura de uma área, como uma imagem PNG em escala de cinza com m pixels de largura e n pixels de altura ($100 \leq n, m \leq 500$). Cada pixel é uma célula; o pixel da linha i e coluna j da imagem corresponde à célula da linha i e coluna j da área.

- Preto (valor 0): célula com cobertura.
- Branco (valor 255): célula sem cobertura.

Cada mapa tem pelo menos uma célula de cada tipo. Alguns deles:



Saída

Você deverá enviar um arquivo `.zip` contendo a resposta de cada caso de teste. A resposta do caso `i.png` deve estar em um arquivo chamado `i.out`. Por exemplo:

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

Você pode omitir alguns arquivos; se o arquivo de um caso não estiver presente, você obterá 0 pontos nesse caso.

Cada arquivo deve conter uma única string formada pelos caracteres `U`, `D`, `L` e `R`: sua sequência de comandos para esse caso, de comprimento no máximo 5000.

Pontuação

Os 10 casos são avaliados de forma independente e cada um vale no máximo 10 pontos, para um total máximo de 100 pontos.

Uma resposta é **válida** se respeita o formato e encontra o drone independentemente da célula sem cobertura em que ele começou. Uma resposta inválida obtém 0 pontos nesse caso.

Seja L_i o comprimento da sua sequência para o caso i, e seja B_i o comprimento da sequência de referência dos juízes para esse caso. Se sua resposta for válida, sua pontuação no caso i é

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

Ou seja, igualar ou superar os juízes concede os 10 pontos completos, e uma sequência com o dobro do comprimento da dos juízes concede 2.5 pontos.