Chicos Mosby

Puntos		Límite de memoria	32MB
Límite de tiempo (caso)	0.5s	Límite de tiempo (total)	30s

Historia

Mientras estabas practicando arduamente para la OMI 2016, el estafador William Zabka se ha robado tu piña de la suerte. Bien sabes que sin tu piña de la suerte no vas a conseguir ninguna medalla en la olimpiada, así que le pides al detective/arquitecto Ted Mosby que, junto a los famosos chicos Mosby, te ayude a encontrar a Zabka.

Zabka se escondió en una cuadrícula rectangular de $N \times M$. Por suerte Ted y sus chicos desarrollaron un dron que les permite buscar a Zabka dentro de un cuadrado, sin importar su tamaño, en tan solo un minuto.

Con su dron, los chicos Mosby piensan ubicar a Zabka con el siguiente método:

- Primero encuentran el cuadrado más grande que quepa en el área de la cuadrícula que aún no han revisado.
- Ubican ese cuadrado pegado a uno de los extremos del área de la cuadrícula que aún no se ha revisado.
- Revisar el área cuadrada en 1 minuto.
- Volver al paso 1 hasta que el área rectangular haya sido revisada completamente.

Problema

Escribe un programa que dadas las dimensiones de N y M de la cuadrícula donde se escondió Zabka, calcule cuál será la cantidad máxima de minutos que tardarán en encontrarlo. Es decir, la cantidad de cuadrados que tendrán que revisar.

Entrada

Tu programa debe leer del teclado la siguiente información:

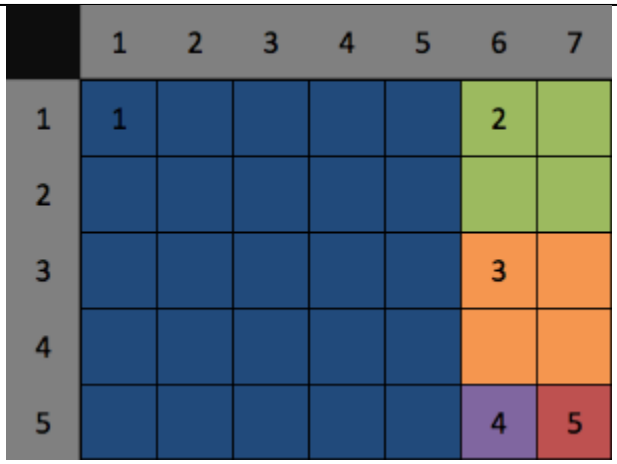
- Una línea con dos números enteros: N y M, las dimensiones de la cuadricula rectangular.

Salida

Tu programa debe escribir en la pantalla un único número que representa el tiempo que tomará revisar toda la cuadrícula.

Ejemplo

Entrada	Salida	Descripción
3 2	3	La cuadrícula es de 3×2. Mosby inicia buscando en un cuadrado de 2×2. Después de realizar la búsqueda, queda una línea sin buscar de 1×2 en la cuadrícula. Mosby checa un cuadrado de 1×1 y finalmente busca en el cuadrado de 1×1 restante. En total Se requirió de 3 minutos para revisar la cuadrícula completa.
5 7	5	La forma de búsqueda se muestra en la siguiente imagen.



Evaluaciòn

Para un conjunto de casos **NO** agrupados con un valor de 60 puntos, se cumple que $1\leq N,M\leq10$

Para otro conjunto de casos agrupados con un valor de 40 puntos, se cumple que $1\leq N,M<2$

Fiesta en la Piscina

Puntos		Límite de memoria	32MB
Límite de tiempo (caso)	0.1s	Límite de tiempo (total)	30s

Historia

Estás en una fiesta cool en la piscina donde hay N personas además de ti. Se sabe que en toda fiesta hay un chico demasiado cool que es conocido por todos, pero que no conoce absolutamente a ninguno de los asistentes de la fiesta.

Como eres un ñoñomi y no gustas de los chicos populares, quieres encontrar a ese chico cool y darle una lección (de programación). Por simplicidad, la gente está numerada de 1 a N. Para encontrar a ese chico cool puedes preguntarle a cualquier persona A: "¿conoces a la persona B en la fiesta?". Tu tarea es hacer todas las preguntas necesarias para encontrar al chico cool de esa fiesta.

Problema

Escribe un programa que, dado el número N de personas que están en la fiesta, realice las preguntas adecuadas al programa evaluador para encontrar al chico cool.

Por supuesto, es posible que esta fiesta se haya hecho durante la OMI y no tenga ningún chico cool.

Entrada y Salida

Este problema es interactivo, por lo que no tendrás que leer la entrada ni imprimir la salida, sino implementar en tu código la función `BusquedaCool` y mandar llamar la función del evaluador `Preguntar` para completar tu tarea.

Para obtener más información sobre los detalles de implementación de este problema debes revisar el texto del problema en la plataforma *OmegaUp*.

Restricciones

$1 \leq N \leq 2{,}000$ Número de personas en la fiesta además de ti. Para todos los casos de prueba puedes llamar la función `Preguntar` a lo más **6,000** veces.

Ejemplo

En el texto del problema que se encuentra en la plataforma de *OmegaUp* aparecen ejemplos de prueba, así como información sobre cómo crear tus propios casos y corridas de prueba.

Implementación

Tu función `BusquedaCool()`

C/C++ `int BusquedaCool(int N);`

Pascal `function BusquedaCool(var N: LongInt): LongInt;`

Descripción

El evaluador buscará en tu código esta función y la llamará con el número N como parámetro. Tu implementación deberá utilizar la función `Preguntar` para encontrar al chico cool de la fiesta y, una vez que lo encuentre, devolver el número que identifica a ese chico.

Si en la fiesta no se encontró a ningún chico cool, tu función deberá devolver `0`.

Parámetros

- `N`: Un entero indicando la cantidad de personas que hay en la fiesta.

Función del evaluador `Preguntar()`

C/C++ `int Preguntar(int A, int B);`

Pascal `function Preguntar(var A, B: LongInt): LongInt;`

Descripción

Llama a esta función para saber si la persona A conoce a la persona B . Si la persona A conoce a la persona B, la función devolverá `1`; si no la conoce, devolverá `0`.

Dado que todas las personas son complicadas, siempre que preguntes **si una persona se conoce a sí misma** (por ejemplo `Preguntar(1, 1)`), esta función devolverá `0`.

En caso de que realices alguna pregunta a personas que no están presentes en la fiesta ($A,B<1$ o $A,B>N$), la función `Preguntar` te devolverá `-1`.

Sí, aunque sabemos que duele, lamentablemente tú nunca serás el chico cool.

Parámetros

- A: La persona a la que se le pregunta.
- B: La persona por la que preguntas.

Rutina de Ejemplo

Suponiendo que en la fiesta hubieran 2 personas y que cada uno conoce a:

- Persona #1: No conoce a nadie.
- Persona #2: Conoce a la persona 1.

Entrada	Salida	Descripción
`Función llamada`	`Valor devuelto`	Explicación
`BusquedaCool(2)`	`1`	Esta es la llamada que el evaluador realiza a la función `BusquedaCool`. En este ejemplo la función devolverá `1` de acuerdo a las llamadas a la función `Preguntar` que se muestran a continuación.
`Preguntar(1, 2)`	`0`	La persona `1` no conoce a la persona `2`.
`Preguntar(2, 1)`	`1`	La persona `2` sí conoce a la persona `1`. Como ya preguntaste a todas las personas que podías preguntar y sabes que la persona 1 no conoce a nadie y la persona 2 sí conoce a la persona 1, entonces la persona 1 debe ser el chico cool. En este momento tu función `BusquedaCool` deberá devolver `1`, indicando que la persona número 1 es el chico cool de la fiesta.

Experimentación

El evaluador de prueba recibe el archivo `sample.in`, con dos enteros N y X en la primera línea, representando el número de personas en la fiesta y, de antemano, el número del chico cool.

Después seguirán N líneas; la i-ésima línea contendrá N números `0` o `1`. Si el j-ésimo número de esta línea es `1` significa que la persona con índice i **SÍ** conoce a la persona con índice j. Si el número es `0`, significa que i **NO** conoce a j.

Para el caso de ejemplo anterior, `sample.in` se vería de la siguiente manera:

```
2 1 0 0 1 0
```

Existen 2 personas y la persona 1 es el chico cool. Por lo tanto: 1 no se conoce a sí mismo ni a la persona 2; 2 conoce a la persona 1 y no se conoce a sí mismo.

Otro Ejemplo:

```
3 1 0 0 0 1 0 1 1 0 0
```

este ejemplo lo puedes copiar en el `sample.in`

Durante la ejecución, el evaluador de prueba imprimirá algunos mensajes para ayudarte a depurar tu solución. Al finalizar la ejecución de tu función, el evaluador de prueba imprimirá la cantidad de veces que llamaste a la función `Preguntar` y te dirá si encontraste al chico cool.

Evaluación

Si durante la ejecución de tu programa excede el límite de llamadas permitidas a `Preguntar`, recibirás 0 puntos. Si al finalizar la ejecución de tu solución no encontraste al chico cool, y hay un chico cool en la fiesta, tu puntaje será de 0 puntos. Por otro lado, si encontraste al chico cool sin haber excedido el límite de llamadas permitidas y tu función no excede el límite de tiempo obtendrás el 100% de los puntos.

Para un grupo de casos con valor de 28 puntos se cumple que $N \leq 50$
Para otro grupo de casos con valor de 36 puntos se cumple que $N \leq 1{,}000$
El resto de los casos estará agrupado en un grupo con valor de 36 puntos.

Pescador

Puntos		Límite de memoria	32MB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	30s

Historia

En la ciudad de Alvarado, Veracruz vive un pescador que tiene una pequeña lanchita, una red de pesca de 1 m^2 y, novedosamente, una app llamada iFish para salir a pescar en un mar rectangular de $w \times h$ m^2, el cual puedes ver como un plano cartesiano.

iFish es muy poderosa, le puede decir al pescador cuántos bancos de peces hay en un área rectángular del mar, sin embargo, como usa la versión gratuita, la app tiene dos restricciones: puede hacer un número de preguntas limitado y cuando no pregunta no puede saber exactamente dónde están los peces en esa área del mar, sólo sabe cuántos hay.

En iFish, el pescador especifica 2 puntos con las coordenadas de las esquinas opuestas de un rectángulo. La app responde diciendo el número de bancos de peces en el área.

El pescador quiere aprovechar al máximo la versión gratuita. Para ello te ha pedido que lo ayudes a diseñar un plan para capturar **a todos** los peces del mar usando la menor cantidad de veces su red.

Recuerda que el pescador puede preguntar cuántos bancos de peces hay en **cualquier tamaño** de área dentro del mar, sin embargo su red mide 1 m^2, por lo tanto, sólo puede usar su red en un cuadro con área de 1 m^2.

Problema

Escribe un programa que, dados w y h, el ancho y largo del área del mar, n, el número total de bancos de peces en todo el mar, y k, el número máximo de consultas que permite la versión gratuita de la app, ayude al pescador a capturar los n bancos de peces tirando su red la menor cantidad posible de veces.

Entrada y Salida

Este problema es interactivo, por lo que no tendrás que leer la entrada ni imprimir la salida, sino implementar en tu código la función `Pescar` y mandar a llamar las funciones del evaluador `UsarApp` y `TirarRed` para completar tu tarea.

Internamente el evaluador llevará el registro de cuantos bancos de peces quedan en el mar, tu programa no necesitará imprimir ni devolver nada, solo asegurarse de que hayas capturado, mediante la función `TirarRed`, todos los bancos de peces del mar.

Ejemplo

En el texto del problema que se encuentra en la plataforma de *OmegaUp* aparecen ejemplos de prueba, así como información sobre cómo crear tus propios casos y corridas de prueba.

Implementación

Tu procedimiento `Pescar()`

C/C++ `void Pescar(int w, int h, int n, int k);`

Pascal `procedure Pescar(var w, h, n, k: LongInt);`

Descripción

El evaluador buscará en tu código esta función y la llamará con los parámetros `w`, `h`, `n` y `k`. Tu implementación deberá utilizar las funciones `UsarApp` y `TirarRed` para atrapar a todos los peces. En cada caso de prueba **solo se llamará a esta función una vez**.

Parámetros

- `w`: El ancho del área que abarca el mar.
- `h`: La altura del área que abarca el mar.
- `n`: El número de bancos de peces que hay en el mar.
- `k`: El número de consultas que puedes hacer con la app.

Función del evaluador `UsarApp()`

C/C++ `int UsarApp(int x1, int y1, int x2, int y2);`.

Pascal `function UsarApp(var x1, y1, x2, y2: LongInt): LongInt;`.

Descripción

Esta es la función que deberás llamar cuando quieras consultar cuántos bancos de peces hay en cierta área. Recibe como parámetros dos puntos que describen esquinas opuestas del área rectangular que deseas consultar.

Si las coordenadas se encuentran fuera del área de cobertura o el rectángulo de consulta tiene área igual a cero, la función devolverá `-1`. En caso contrario, devolverá el número de bancos de peces que se encuentran dentro del rectángulo consultado.

Las coordenadas con las que debes preguntar siempre serán enteras. Los bancos de peces **NO** se moverán de su lugar.

Parámetros

- `x1`: La coordenada *X* de una esquina del rectángulo.
- `y1`: La coordenada *Y* de una esquina del rectángulo.
- `x2`: La coordenada *X* de la esquina opuesta del rectángulo.
- `y2`: La coordenada *Y* de la esquina opuesta del rectángulo.

Función del evaluador `TirarRed()`

C/C++ `int TirarRed(int x1, int y1, int x2, int y2);`.

Pascal `function TirarRed(var x1, y1, x2, y2: LongInt): LongInt;`.

Descripción

Esta es la función que deberás llamar cuando quieras capturar a los peces que se encuentren en cierta área. Recibe como parámetros dos puntos que describen las esquinas opuestas del área rectangular que deseas consultar.

Esta función tiene una restricción adicional: la red del pescador mide 1 m^2, por lo tanto, las coordenadas que proporciones no podrán formar un rectángulo de área mayor a 1 m^2.

Si las coordenadas se encuentran fuera del área de cobertura o el rectángulo de consulta tiene área igual a cero o mayor a 1 m^2, la función devolverá `-1`. En caso contrario, capturará a todos los peces que se encuentren dentro del área del rectángulo indicado.

Nota: Una vez que captures un banco de peces, este ya no volverá a contar en las siguientes consultas de `UsarApp`.

Parámetros

- `x1`: La coordenada *X* de una esquina del rectángulo.
- `y1`: La coordenada *Y* de una esquina del rectángulo.
- `x2`: La coordenada *X* de la esquina opuesta del rectángulo.
- `y2`: La coordenada *Y* de la esquina opuesta del rectángulo.

Rutina de Ejemplo

Entrada	Salida	Descripción
`Función llamada`	`Valor devuelto`	Descripción
`Pescar(5, 5, 6, 25)`	`-`	Esta será la llamada inicial a tu procedimiento `Pescar`.
`UsarApp(0, 0, 5, 5)`	`6`	El pescador consulta el número de bancos de peces que hay en todo el mar. Con esta llamada verifica que, en efecto, existen 6 bancos en el mar.
`UsarApp(1, 3, 4, 3)`	`-1`	El pescador hace una consulta en un área de 0, es decir, en una línea. La aplicación responderá `-1` dado que es imposible responder esa consulta.
`TirarRed(1, 3, 2, 2)`	`0`	El pescador tira la red en un cuadrado de 1 m^2 y no atrapa a ningún banco de peces.
`TirarRed(2, 4, 4, 5)`	`-1`	El pescador no puede extender la red más de 1 m^2, por ello, esta función devuelve `-1`.

Experimentación

El evaluador de prueba recibe el archivo `sample.in` con los valores w , h , n y k en la primer línea. En las siguientes n líneas se encontrarán las coordenadas x y y de cada uno de los bancos de peces que se encuentran en el mar. Las coordenadas de los bancos deben de ser **NO** enteras.

Para el caso de ejemplo anterior, `sample.in` se vería de la siguiente manera:

```
5 5 6 25 0.2 0.8 2.1 2.5 4.7 0.9 2.3 4.6 1.5 4.2 1.9 3.4
```

Donde el tamaño del mar es de 5×5 , existen 6 bancos de peces y puedes tirar la red 25 veces. A partir de la línea 2, se encuentran las coordenadas de los 6 bancos de peces, siendo la primera la posición la x, y la segunda y.

Puedes modificar el contenido de `sample.in` para probar distintos casos con tu solución.

Durante la ejecución, el evaluador de prueba imprimirá algunos mensajes para ayudarte a depurar tu solución. Al finalizar la ejecución de tu función, el evaluador de prueba imprimirá la cantidad de veces que llamaste a las funciones `UsarApp` y `TirarRed`, seguido de un mensaje que te dirá si lograste capturar todos los bancos de peces o no.

IMPORTANTE: El evaluador de prueba y el evaluador final que se usará para calificar tu solución son diferentes. El evaluador de prueba tiene la intención de facilitar la depuración de tu solución y puede implementar algunas de sus funciones de manera ineficiente, sin embargo puedes asumir con seguridad que **el tiempo de ejecución del evaluador final no afectará el tiempo total de tu solución**.

Evaluación

Si durante la ejecución de tu programa excedes el límite de consultas que puedes hacer a la app o tiras la red más de 100,000 veces, recibirás 0 puntos. Si al finalizar la ejecución de tu solución no lograste atrapar a todos los peces tu puntaje será de 0 puntos. Por otro lado, si logras capturar a todos los peces y no excedes el límite de llamadas para ninguna función del evaluador, recibirás un puntaje de al menos 20% de los puntos. El 80% del puntaje restante se otorgará en función del número de veces que llamaste a TirarRed, por lo tanto, entre menos veces uses esta función, mayor será tu puntaje. Nota que cada caso se puntuará de manera independiente.

A continuación se muestran las llamadas para los 20 casos de prueba donde los parámetros de la llamada son:

`Pescar`(w,h,n,k) las dimensiones del mar, el número de bancos y la cantidad de veces que puedes preguntar al app.

- **Caso 01**: `Pescar(10, 10, 20, 100)`.
- **Caso 02**: `Pescar(18, 12, 50, 200)`.
- **Caso 03**: `Pescar(20, 24, 75, 360)`.
- **Caso 04**: `Pescar(36, 44, 200, 900)`.
- **Caso 05**: `Pescar(60, 80, 330, 2400)`.
- **Caso 06**: `Pescar(100, 150, 365, 5000)`.
- **Caso 07**: `Pescar(130, 120, 240, 3900)`.
- **Caso 08**: `Pescar(350, 350, 2450, 14000)`.
- **Caso 09**: `Pescar(450, 600, 1150, 8000)`.
- **Caso 10**: `Pescar(800, 600, 1800, 15000)`.
- **Caso 11**: `Pescar(800, 800, 2400, 12000)`.
- **Caso 12**: `Pescar(800, 1000, 800, 10000)`.
- **Caso 13**: `Pescar(3000, 2000, 400, 8000)`.
- **Caso 14**: `Pescar(5000, 5000, 850, 10000)`.
- **Caso 15**: `Pescar(7000, 9000, 1200, 17000)`.
- **Caso 16**: `Pescar(10000, 10000, 1250, 17700)`.

Para los últimos 4 casos de prueba se asegura que existe un rectángulo de coordenadas enteras en el mar con un área de n **m²** **que contiene a los** n **bancos de peces**.

- **Caso 17**: `Pescar(200, 360, 1050, 720)`.
- **Caso 18**: `Pescar(250, 220, 800, 650)`.
- **Caso 19**: `Pescar(280, 320, 1950, 890)`.
- **Caso 20**: `Pescar(280, 400, 800, 700)`.

All-Out Attack

Puntos		Límite de memoria	256MB
Límite de tiempo (caso)	1s	Límite de tiempo (total)	60s

Historia

Practicar para la OMI es muy estresante, así que te pones a jugar un juego de video para relajarte. En dicho juego tienes un equipo de 4 personajes. Cada personaje cuenta con un número entero de puntos de energía que pueden usar para hacer ataques. En este juego realizar cualquier ataque cuesta K puntos de energía.

Como en los mejores juegos de este tipo, cada personaje puede atacar sólo con cierto tipo de ataques. En este juego existen 7 tipos: Normal, Fuego, Hielo, Aire, Electricidad, Luz y Oscuridad. Cada personaje puede realizar ataques de un subconjunto de estos tipos. Por ejemplo, un personaje de tu equipo podría solamente realizar ataques de tipo Normal y de Fuego, mientras otro personaje podría realizar ataques de tipo Aire y Electricidad, y así para los demás personajes.

El siguiente nivel que jugarás tiene N enemigos, numerados del 1 al N, cada enemigo es vulnerable a **sólo un tipo de ataque, si recibe un ataque de este tipo serà derrotado inmediatamente, ademas un ataque de este tipo es la ùnica forma de vencerlo**. Para terminar el nivel debes vencer a todos y cada uno de los enemigos. Lleno de determinación, has decidido terminar este juego de una vez por todas.

Problema

Escribe un programa que, dada la descripción de los tipos de ataque que pueden realizar los personajes de tu equipo, los puntos de energía que tiene cada uno, la cantidad de enemigos en el siguiente nivel y las vulnerabilidades de cada uno, determine la estrategia para finalizar el nivel. **Se asegura que siempre habrá una forma de finalizar el nivel.**

Entrada

Tu programa debe leer del teclado la siguiente información.

- Una línea con los números N y K, la cantidad de enemigos y el costo en energía de realizar un ataque.

- En las siguientes 4 líneas recibirás una cadena de caracteres P_i y un entero E_i . Cada línea representa a un personaje de tu equipo, la cadena P_i puede contener cualquier subconjungo de los caracteres `N`, `F`, `H`, `A`, `E`, `L`, `O`, que representan los tipos de ataque que puede realizar el i-ésimo personaje de tu equipo. El entero E_i representa la cantidad de energía del personaje al inciar el nivel.
- Finalmente habrá N líneas, cada una con un único carácter V_j que puede ser `N`, `F`, `H`, `A`, `E`, `L` o `O` y que representa la vulnerabilidad del enemigo j-ésimo.

Salida

Para cada uno de los miembros de tu equipo (en el mismo orden en el que se presentan en la entrada) debes imprimir dos líneas.

En la primera de estas líneas, para el miembro i , debes imprimir un entero M_i representando la cantidad de enemigos que el i-ésimo personaje debe vencer. En la siguiente línea, imprime M_i enteros; cada entero representa el índice de uno de los enemigos que ese personaje debe vencer.

Para obtener puntos en este problema es necesario que imprimas una estrategia válida para vencer a los N enemigos sin rebasar los puntos de energía de cada personaje ni sus restricciones en el tipo de ataques que pueden realizar. En caso de haber múltiples soluciones cualquiera de ellas será considerada correcta.

Restricciones

$1 \leq K \leq 10$ El costo de realizar un ataque. $1 \leq E_i \leq 5{,}000{,}000$ La cantidad de energía inicial del personaje i. $1 \leq N \leq 50{,}000$ El número de enemigos.

Ejemplo

Entrada	Salida	Descripción
5 1 NAE 3 A 1 EH 2 A 3 N E	1 1 0 2 4 2 2	El miembro 1 del equipo ataca al enemigo 1 con un ataque Normal. El miembro 2 de tu equipo no ataca a ningún enemigo. El miembro 3 del equipo vence al enemigo 2 y 4 con un ataque Electrico. Finalmente, el miembro 4 ataca a los enemigos 3 y 5 con ataques de Aire.

Entrada	Salida	Descripción
A E A	3 5	

Para un grupo de casos de prueba con valor de 23 puntos, $N \leq 10$

Para un grupo de casos de prueba con valor de 26 puntos, $N \leq 50$

Para un grupo de casos de prueba con valor de 22 puntos, $N \leq 100$ N≤100