Binary Search y Problemas Interactivos

Nacional OIA

Pajor Ivo

2019

Introducción

Problema 1

Dado un arreglo ordenado de n números S, decidir si un elemento x se encuentra en él.

Introducción

- Una solución para esto es realizar una búsqueda lineal, pero esta tiene una complejidad $O(n)$ y no utiliza la información de que el arreglo está ordenado.

Introducción

- Una solución para esto es realizar una búsqueda lineal, pero esta tiene una complejidad $O(n)$ y no utiliza la información de que el arreglo está ordenado.
- Busquemos una mejor solución...

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.
- Supongamos que $S = [2, 4, 9, 11, 12]$ y buscamos $x = 3$.

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.
- Supongamos que $S = [2, 4, 9, 11, 12]$ y buscamos $x = 3$.
- Entonces el elemento del medio es 9.

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.
- Supongamos que $S = [2, 4, 9, 11, 12]$ y buscamos $x = 3$.
- Entonces el elemento del medio es 9.
- Ahora notemos que 3 no puede estar adelante del 9.

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.
- Supongamos que $S = [2, 4, 9, 11, 12]$ y buscamos $x = 3$.
- Entonces el elemento del medio es 9.
- Ahora notemos que 3 no puede estar adelante del 9.
- Entonces el 3 deberia estar en $[2, 4]$

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.
- Supongamos que $S = [2, 4, 9, 11, 12]$ y buscamos $x = 3$.
- Entonces el elemento del medio es 9.
- Ahora notemos que 3 no puede estar adelante del 9.
- Entonces el 3 deberia estar en $[2, 4]$
- El siguiente paso sería realizar lo mismo para la lista que contiene a los números no descartados.

Introducción: Idea

- Para mejorar la complejidad veamos que información nos brinda el elemento medio del arreglo.
- Supongamos que $S = [2, 4, 9, 11, 12]$ y buscamos $x = 3$.
- Entonces el elemento del medio es 9.
- Ahora notemos que 3 no puede estar adelante del 9.
- Entonces el 3 deberia estar en $[2, 4]$
- El siguiente paso sería realizar lo mismo para la lista que contiene a los números no descartados.
- Este proceso termina cuando me queda un elemento en la lista y la respuesta dependerá en si es o no el elemento buscado.

Introducción: Implementación

- Utilizaremos dos índices *low* y *high*, para guardar en que rango debería estar el elemento buscado.

Introducción: Implementación

- Utilizaremos dos índices *low* y *high*, para guardar en que rango debería estar el elemento buscado.
- El indice *low* nos indica los números,que sabemos, son menores a x, y high los números que son mayores o iguales a x.

Introducción: Implementación

- Utilizaremos dos índices *low* y *high*, para guardar en que rango debería estar el elemento buscado.
- El indice *low* nos indica los números,que sabemos, son menores a x, y high los números que son mayores o iguales a x.
- Luego en cada paso, buscamos el elemento en el medio de *low* y *high*, llamémoslo *mid*.

Introducción: Implementación

- Utilizaremos dos índices *low* y *high*, para guardar en que rango debería estar el elemento buscado.
- El indice *low* nos indica los números,que sabemos, son menores a x, y high los números que son mayores o iguales a x.
- Luego en cada paso, buscamos el elemento en el medio de *low* y *high*, llamémoslo *mid*.
- Comparamos para saber si $S[mid] < x$ y actualizamos los índices.

Introducción: Código

```
bool estaEnArreglo(int numeroDeseado, vector<int> elementosOrdenados)
{
    const int N = int(elementosOrdenados.size());
    int low = -1; // Aca guardamos hasta donde sabemos que son menores
    int high = N; // Aca guardamos desde donde son mayores o iguales

    while(high-low>1)
    {
        int mid = (high+low)/2;
        if(elementosOrdenados[mid]>=numeroDeseado)
            high = mid; // Siempre high es uno que SI cumple
        else
            low = mid;  // Siempre  low es uno que NO cumple
    }
    // Como desde high sabemos que son todos mayores o iguales, si la posicion high es mayor ya esta,
    // desde aca son todos mayores y el numero no esta. Entonces si esta, esta en la posicion high.
    return high<N && elementosOrdenados[high]==numeroDeseado; // Pensar por que pedimos high<N
}
```

Introduccion: Complejidad

Este algoritmo tiene complejidad $0(lg(n))$ ya que en cada paso reducimos el rango de posibles valores por la mitad.

Idea General

- Ahora extendamos esta idea para una propiedad $P(k)$ con k un número tal que $P(k)$ es falso o verdadero dependiendo del k.

Idea General

- Ahora extendamos esta idea para una propiedad $P(k)$ con k un número tal que $P(k)$ es falso o verdadero dependiendo del k.
- Si p es constantemente False y luego constantemente True (o al revés). Podemos encontrar el k en donde se produce el quiebre.

Idea General

- Ahora extendamos esta idea para una propiedad $P(k)$ con k un número tal que $P(k)$ es falso o verdadero dependiendo del k.
- Si p es constantemente False y luego constantemente True (o al revés). Podemos encontrar el k en donde se produce el quiebre.
- Su implementación es muy parecida a la anterior cambiando en el if por $P(mid)$.
- Veamos un ejemplo...

Ejemplo

Problema 2

En un supermercado hay n filas y en cada una hay a_i personas esperando con $1 \leq i \leq n$. Hay k personas que están llegando al supermercado las cuáles se van a formar en alguna fila, debemos ubicar a las k personas de tal forma de minimizar la cantidad de gente en la fila con mayor cantidad de personas.

Solución

- Podemos hacer este problema usando greedy, pero usemos binary search.

Solución

- Podemos hacer este problema usando greedy, pero usemos binary search.
- Sea $P(r)$ "¿Si r es el máximo de personas que puedo poner en cada fila, existe una configuración que cumpla lo pedido?"

Solución

- Podemos hacer este problema usando greedy, pero usemos binary search.
- Sea $P(r)$ "¿Si r es el máximo de personas que puedo poner en cada fila, existe una configuración que cumpla lo pedido?"
- Notemos que esto va a ser constantemente False hasta que llega a un punto que es True, cuando llega a ese punto es fácil ver que va a seguir siendo True.

Solución

- Podemos hacer este problema usando greedy, pero usemos binary search.
- Sea $P(r)$ "¿Si r es el máximo de personas que puedo poner en cada fila, existe una configuración que cumpla lo pedido?"
- Notemos que esto va a ser constantemente False hasta que llega a un punto que es True, cuando llega a ese punto es fácil ver que va a seguir siendo True.
- Iniciamos $low = 0$ y $high = max(a_i) + k$.

Solución

- Podemos hacer este problema usando greedy, pero usemos binary search.
- Sea $P(r)$ "¿Si r es el máximo de personas que puedo poner en cada fila, existe una configuración que cumpla lo pedido?"
- Notemos que esto va a ser constantemente False hasta que llega a un punto que es True, cuando llega a ese punto es fácil ver que va a seguir siendo True.
- Iniciamos $low = 0$ y $high = max(a_i) + k$.
- Link para enviar el problema (con una leve adicion): https://codeforces.com/problemset/problem/1042/A

Problema (selectivo)

Problema 3: (bailando)

En un concurso de baile compiten F famosos y B bailarines profesionales, $F \leq B$. Cada uno posee cierta altura, con alturas entre 1000 y 2500. Queremos emparejar a todos los famosos con un bailarín (distinto para cada uno), de tal forma que minimicemos la máxima diferencia entre alturas de las parejas.

Idea

- Podemos resolver este problema con...

Idea

- Podemos resolver este problema con...
- BINARY SEARCH!!! :o

Idea

- Podemos resolver este problema con...
- BINARY SEARCH!!! :o
- ¿Cómo?, es muy similiar al problema anterior, sea $P(k) =$ "¿Si k es la máxima diferencia que puede haber entre parejas, existe una configuración que cumpla lo pedido?".

Idea

- Podemos resolver este problema con...
- BINARY SEARCH!!! :o
- ¿Cómo?, es muy similiar al problema anterior, sea $P(k) =$ "¿Si k es la máxima diferencia que puede haber entre parejas, existe una configuración que cumpla lo pedido?".
- Notemos que cumple ser constantemente False y luego ser constantemente True.

Idea

- Podemos resolver este problema con...
- BINARY SEARCH!!! :o
- ¿Cómo?, es muy similiar al problema anterior, sea $P(k) =$ "¿Si k es la máxima diferencia que puede haber entre parejas, existe una configuración que cumpla lo pedido?".
- Notemos que cumple ser constantemente False y luego ser constantemente True.
- Y que podemos responder a esta pregunta en $0(B * log(B))$ (usando sets por ejemplo y haciendo greedy), por lo cual la complejidad total nos queda $0(B * log(B) * log(1500))$, lo cual es suficiente.

Idea

- Podemos resolver este problema con...
- BINARY SEARCH!!! :o
- ¿Cómo?, es muy similiar al problema anterior, sea $P(k) =$ "¿Si k es la máxima diferencia que puede haber entre parejas, existe una configuración que cumpla lo pedido?".
- Notemos que cumple ser constantemente False y luego ser constantemente True.
- Y que podemos responder a esta pregunta en $0(B * log(B))$ (usando sets por ejemplo y haciendo greedy), por lo cual la complejidad total nos queda $0(B * log(B) * log(1500))$, lo cual es suficiente.
- Pueden enviar este problema en el Juez de OIA: http://juez.oia.unsam.edu.ar/#/task/bailando/statement

Consejo

- Notemos que en los dos anteriores problemas nos pedían minimizar un máximo, siempre que los problemas nos hablen de esto es buena idea pensar en binary search, ya que en general se cumple la propiedad de ser constantemente False y luego ser constantemente True.

Consejo

- Notemos que en los dos anteriores problemas nos pedían minimizar un máximo, siempre que los problemas nos hablen de esto es buena idea pensar en binary search, ya que en general se cumple la propiedad de ser constantemente False y luego ser constantemente True.
- Siempre y cuando podamos responder rápido a la pregunta.

Más problemas

Problema 4: (frutales)

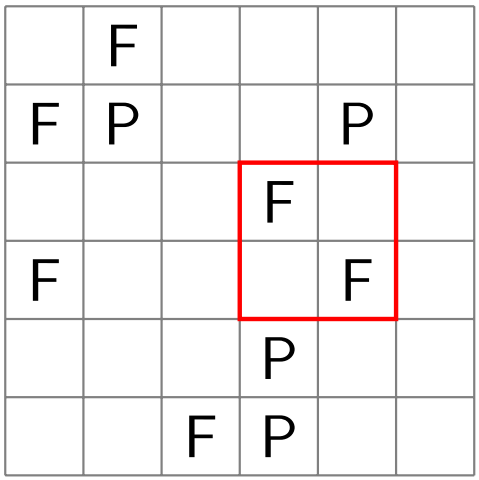
El dueño de un amplio terreno desea construir un invernadero de piso cuadrado para proteger sus frutales de las heladas. El terreno es rectangular y tiene dispersos algunos pinos muy añosos y una gran variedad de frutales. Tratando de encontrar el lugar apropiado, después de varias mediciones el dueño logra cuadricular el terreno de manera tal que en cada cuadrícula le queda un pino, un árbol frutal o simplemente terreno libre. El dueño pretende que el invernadero cubra la mayor cantidad posible de frutales, pero se niega rotundamente a talar un pino. Además, si descubre que puede instalar el invernadero en más de un lugar cubriendo la misma cantidad máxima de frutales, se decide por el de menor superficie.

Problema

	F				
F	P			P	
			F		
F				F	
			P		
		F	P		

¿Dónde colocamos el invernadero?

Problema



Recordemos que nos piden el cuadrado mínimo que contenga la mayor cantidad de frutales.

Solución

- Si nos paramos en alguna casilla del tablero,¿cúal es el máximo cuadrado que tenga como esquina superior izquierda a la casilla en que estoy parado y que no contenga pinos?.

Solución

- Si nos paramos en alguna casilla del tablero,¿cúal es el máximo cuadrado que tenga como esquina superior izquierda a la casilla en que estoy parado y que no contenga pinos?.
- Acá podemos usar binary search + sumas parciales.

Solución

- Si nos paramos en alguna casilla del tablero,¿cúal es el máximo cuadrado que tenga como esquina superior izquierda a la casilla en que estoy parado y que no contenga pinos?.
- Acá podemos usar binary search + sumas parciales.
- En cada casilla del tablero guardamos cuantos pinos hay en el rectángulo que tiene como esquina superior la del tablero y como esquina inferior derecha mi casilla.

Solución

- Si nos paramos en alguna casilla del tablero,¿cúal es el máximo cuadrado que tenga como esquina superior izquierda a la casilla en que estoy parado y que no contenga pinos?.
- Acá podemos usar binary search + sumas parciales.
- En cada casilla del tablero guardamos cuantos pinos hay en el rectángulo que tiene como esquina superior la del tablero y como esquina inferior derecha mi casilla.
- ¿Cómo se vería?

Solución

	F				
F	P			P	
			F		
F				F	
			P		
		F	P		

0	0	0	0	0	0
0	1	1	1	2	2
0	1	1	1	2	2
0	1	1	1	2	2
0	1	1	2	3	3
0	1	1	3	4	4

Solución

- Una vez tengamos las sumas parciales hacemos binary search en donde False sería si tenemos 0 pinos y True cuando tenemos algún pino en un cuadrado de largo k (que tiene como esquina superior izquierda la casilla en que estamos parados).

Solución

- Una vez tengamos las sumas parciales hacemos binary search en donde False sería si tenemos 0 pinos y True cuando tenemos algún pino en un cuadrado de largo k (que tiene como esquina superior izquierda la casilla en que estamos parados).
- Cuando sabemos el máximo cuadrado sin pinos, sabemos que este es el que tiene mayor cantidad de frutales, pero no necesariamente es el mínimo.

Solución

- Una vez tengamos las sumas parciales hacemos binary search en donde False sería si tenemos 0 pinos y True cuando tenemos algún pino en un cuadrado de largo k (que tiene como esquina superior izquierda la casilla en que estamos parados).
- Cuando sabemos el máximo cuadrado sin pinos, sabemos que este es el que tiene mayor cantidad de frutales, pero no necesariamente es el mínimo.
- La cantidad de frutales en este cuadrado lo podemos calcular de la misma manera que los pinos, con sumas parciales.

Solución

- Una vez tengamos las sumas parciales hacemos binary search en donde False sería si tenemos 0 pinos y True cuando tenemos algún pino en un cuadrado de largo k (que tiene como esquina superior izquierda la casilla en que estamos parados).
- Cuando sabemos el máximo cuadrado sin pinos, sabemos que este es el que tiene mayor cantidad de frutales, pero no necesariamente es el mínimo.
- La cantidad de frutales en este cuadrado lo podemos calcular de la misma manera que los pinos, con sumas parciales.
- Sea k la cantidad de frutales en el máximo cuadrado sin pinos, ahora con binary search buscamos el mínimo cuadrado que tenga k frutales.

Solución

- Una vez tengamos las sumas parciales hacemos binary search en donde False sería si tenemos 0 pinos y True cuando tenemos algún pino en un cuadrado de largo k (que tiene como esquina superior izquierda la casilla en que estamos parados).
- Cuando sabemos el máximo cuadrado sin pinos, sabemos que este es el que tiene mayor cantidad de frutales, pero no necesariamente es el mínimo.
- La cantidad de frutales en este cuadrado lo podemos calcular de la misma manera que los pinos, con sumas parciales.
- Sea k la cantidad de frutales en el máximo cuadrado sin pinos, ahora con binary search buscamos el mínimo cuadrado que tenga k frutales.
- Este algoritmo tiene una complejidad de $0(x * y * log(min(x, y)))$, con y la altura del tablero y x el largo.

Solución

- Una vez tengamos las sumas parciales hacemos binary search en donde False sería si tenemos 0 pinos y True cuando tenemos algún pino en un cuadrado de largo k (que tiene como esquina superior izquierda la casilla en que estamos parados).
- Cuando sabemos el máximo cuadrado sin pinos, sabemos que este es el que tiene mayor cantidad de frutales, pero no necesariamente es el mínimo.
- La cantidad de frutales en este cuadrado lo podemos calcular de la misma manera que los pinos, con sumas parciales.
- Sea k la cantidad de frutales en el máximo cuadrado sin pinos, ahora con binary search buscamos el mínimo cuadrado que tenga k frutales.
- Este algoritmo tiene una complejidad de $0(x * y * log(min(x, y)))$, con y la altura del tablero y x el largo.
- Pueden enviar este problema en el Juez de OIA: http://juez.oia.unsam.edu.ar/#/task/frutales/statement

Interactivo: ¿Qué es?

- En un problema interactivo, el input dado puede no estar creado predeterminadamente, sino que se crea específicamente para nuestra solución.

Interactivo: ¿Qué es?

- En un problema interactivo, el input dado puede no estar creado predeterminadamente, sino que se crea específicamente para nuestra solución.
- Existe un "interactor", tal que el output del mismo es el input de nuestro programa y el output de nuestro programa es el input del "interactor".

Interactivo: Ejemplo

Problema 5

Sea x un número del 1 al 1.000.000 que no conocemos, debemos adivinar este número haciendo queries. En estas queries preguntamos sobre un numero z del 1 al 1.000.000 y el sistema nos responde "$<$" si $x < z$ y "$>=$" si $x \geq z$. A lo sumo podemos hacer 25 queries.

Solución

Notemos que este problema lo podemos resolver con binary search, ya que son las mismas preguntas que nos hacemos en el problema 1 (salvo un igual).

Interactivo:Implementación

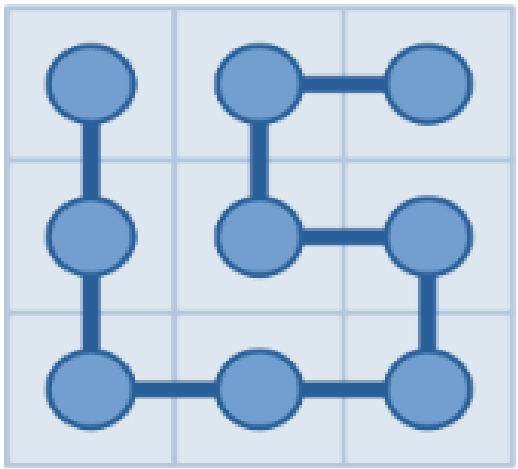
```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long intl;
int main(){
    int low=1,high=1000001;
    while(high-low>1){
        int mid=(high+low)/2;
        cout<<mid<<endl;
        string s;
        cin>>s;
        if(s=="<"){
            high=mid;
        }
        else{
            low=mid;
        }

    }
    cout<<"! "<<low<<endl;

}
```

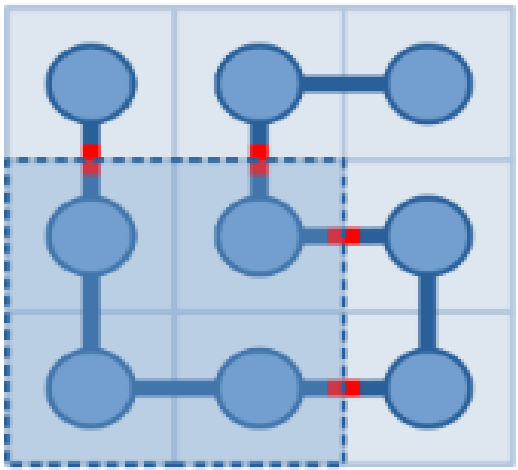
Interactivo: Problema

Problema 6

En un tablero de $n * n$ con $n \leq 1000$, hay una serpiente (que no podemos ver) cuya cabeza y cola se encuentran en diferentes casillas . Su cuerpo es una serie de casillas adyacentes que conectan la cabeza y su cola.
Queremos saber la posición de la cabeza y de la cola, para ello contamos con un dispotivo al que le preguntamos por un rectángulo del tablero y nos dice las veces que la serpiente cruza el borde de ese rectángulo. Encontrar la posición de la cabeza y de la cola, dado que solamente podemos usar el dispositivo 2019 veces.

Problema





Solución

- Notemos que si pregunto por algún rectángulo y el número que me devuelve es par, entonces la cabeza y la cola se encuentran ambos dentro del rectángulo o ambos afuera de este.

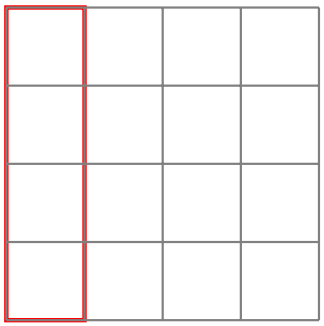
Solución

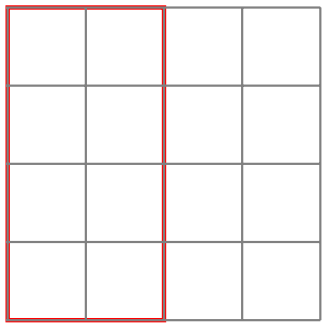
- Notemos que si pregunto por algún rectángulo y el número que me devuelve es par, entonces la cabeza y la cola se encuentran ambos dentro del rectángulo o ambos afuera de este.
- Usando 1998 queries podemos saber en que fila y columna se encuentran.

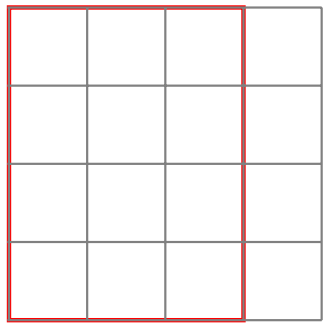
Solución

- Notemos que si pregunto por algún rectángulo y el número que me devuelve es par, entonces la cabeza y la cola se encuentran ambos dentro del rectángulo o ambos afuera de este.
- Usando 1998 queries podemos saber en que fila y columna se encuentran.
- Esto lo hacemos de la siguiente manera.

Solución







Solución

- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.

Solución

- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.
- La primera pregunta en que nos da impar es donde va a estar uno de sus extremos y la última pregunta que nos da impar nos dice donde esta el otro.

Solución

- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.
- La primera pregunta en que nos da impar es donde va a estar uno de sus extremos y la última pregunta que nos da impar nos dice donde esta el otro.
- Esto lo haríamos para saber en que columna están.

Solución

- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.
- La primera pregunta en que nos da impar es donde va a estar uno de sus extremos y la última pregunta que nos da impar nos dice donde esta el otro.
- Esto lo haríamos para saber en que columna están.
- Hacemos lo mismo pero para las filas.

Solución

- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.
- La primera pregunta en que nos da impar es donde va a estar uno de sus extremos y la última pregunta que nos da impar nos dice donde esta el otro.
- Esto lo haríamos para saber en que columna están.
- Hacemos lo mismo pero para las filas.
- Si estan en diferentes filas y columnas, entonces solamente nos falta ver cual de las dos posibles formas es.
- Pero, ¿qué ocurre si están en la misma fila o columna?...

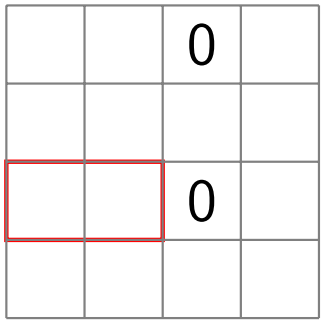
Solución

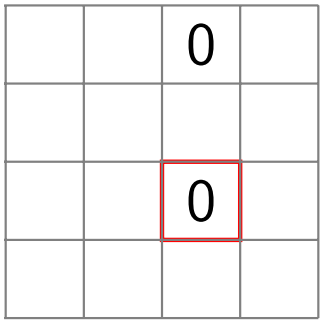
- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.
- La primera pregunta en que nos da impar es donde va a estar uno de sus extremos y la última pregunta que nos da impar nos dice donde esta el otro.
- Esto lo haríamos para saber en que columna están.
- Hacemos lo mismo pero para las filas.
- Si estan en diferentes filas y columnas, entonces solamente nos falta ver cual de las dos posibles formas es.
- Pero, ¿qué ocurre si están en la misma fila o columna?...
- Siempre nos da par!!, pero por suerte no puede ocurrir que estén en la misma fila y columna.

Solución

- Nos importa solo la paridad de la respuesta, notemos que las primeras preguntas nos van a dar par, luego impar y por último vuelven a ser pares.
- La primera pregunta en que nos da impar es donde va a estar uno de sus extremos y la última pregunta que nos da impar nos dice donde esta el otro.
- Esto lo haríamos para saber en que columna están.
- Hacemos lo mismo pero para las filas.
- Si estan en diferentes filas y columnas, entonces solamente nos falta ver cual de las dos posibles formas es.
- Pero, ¿qué ocurre si están en la misma fila o columna?...
- Siempre nos da par!!, pero por suerte no puede ocurrir que estén en la misma fila y columna.
- Supongamos que están en la misma columna, entonces...usamos binary search, ya que sabemos en que filas están.

Solución





Pueden enviar este problema a:
https://codeforces.com/problemset/problem/1153/E

Extras

- Es conveniente usar $low + (high - low)/2$ en vez de $(high + low)/2$ debido a posibles casos de overflow.
- Siempre debemos chequear que las inicializaciones de low y high son correctas.

Fin

¿Preguntas?