Búsquedas

Héctor Fernando Ricárdez Lara

Índice

Introducción

Muchas veces en nuestra vida hemos tenido que buscar algo, una foto en nuestra galería del teléfono, una palabra en el diccionario, una carta dentro de un mazo, etc. Y probablemente, con la experiencia, hemos aprendido algunas intuiciones sobre como buscar cosas, en este libro trabajaremos un poco más en desarrollar esta intuición e ideas.

Igual que en la vida diaria buscamos muchas cosas, en la resolución de problemas nos la pasamos buscando cosas, ya sea: la respuesta, algún valor en un arreglo, la raíz cuadrada de un número, etc.

Es decir, en estos problemas tendremos una lista de objetos y trataremos de encontrar en la lista uno que cumpla alguna propiedad especifica. Es decir, queremos encontrar una respuesta dentro de una lista de candidatos.

Entonces, búsqueda es simplemente encontrar una respuesta dentro en una lista de posibles candidatos.

Búsqueda lineal

La búsqueda lineal es la más sencilla de las búsquedas que hay. ¿Qué es lo que harías si te pido que de una pila de exámenes encuentres el tuyo? Lo que probablemente hagas es una revisar uno por uno, checar de arriba hacia abajo hasta que encuentres el examen con tu nombre en él.

Básicamente, esta idea es la búsqueda lineal, ir revisando de uno por uno toda una lista de candidatos hasta encontrar al que estas buscando o hasta que hayas revisado todos los candidatos.

Entonces, los códigos de búsqueda lineal casi siempre tendrán la siguiente estructura:

```
Itera por cada candidato {
   Si el candidato es lo que buscamos {
      Respuesta = candidato;
      Detener ciclo /* esto es opcional, depende si hay
         varios valores que queramos encontrar. */
   }
}
```

Veamos cómo usar esta técnica para resolver un problema.

Ejemplo 1.1

Descripción

Supongamos que queremos tenemos un arreglo A de enteros distintos y este tiene N elementos en él. Nosotros querremos hacer un código que imprima la posición del arreglo que valga K. O si este valor no existe, que imprima -1.

Límites

- $1 \leq N \leq 10^5$
- $1 \leq K, A[i] \leq 10^9$

Solución

(Recuerda intentar el problema antes de leer la solución)

Lo que este problema nos pide en realidad es buscar dentro del arreglo por el índice del elemento K.

Lo que haremos es revisar todas las posiciones del arreglo hasta encontrar aquella que valga K, si no la encontramos imprimimos -1.

Código

```
int A[100050];
int N, K;
int main () {
   ios_base::sync_with_stdio(0); cin.tie(0);
   cin >> N;
   for (int i=0; i <N; i++){
      cin >> A[i];
   }
   cin >> K;
   int respuesta = -1;
```

```
  for (int i =0; i < N; i++) {
    if (A[i]==K) {
      respuesta = i;
      break;
    }
  }
  cout << respuesta;
}
```

Complejidad

Una pregunta que te has de hacer es: ¿cuál es la complejidad de esta técnica? Y la respuesta es sencilla, en el peor de los casos tenemos que revisar a todos los candidatos, digamos que la cantidad de ellos es iguala a N, entonces la complejidad es $O(N)$.

Ejemplo 1.2

Veamos otro problema para aprender búsqueda lineal.

Descripción

Carlos quiere armar una fiesta, y como le gusta ser un buen anfitrión compro N regalos para sus invitados. Ahora, Carlos quiere darle la misma cantidad de regalos a cada uno de sus invitados sin que sobre ningún regalo no repartido. Como Carlos le gusta contar, ahora se pregunta: ¿Cuántas cantidades diferentes de invitados puede tener?

Entrada

Un entero N, indicando cuantos regalos compró Carlos.

Salida

La cantidad de posibles números de invitados para la fiesta.

Casos ejemplo

Entrada	Salida
12	6
7	1
4	3

Límites

- $1 \leq N \leq 10^5$

Solución

Es fácil ver que el problema en realidad pregunta: ¿Cuántos divisores positivos tiene N?

(Nota: un divisor de N es un número que divide a N sin decimales).

Encontremos todos los divisores de N. Estos se encontrarán entre 1 y N, por lo que podemos iterar i por todo este rango revisando si i es divisor de N.

Código

```
respuesta = 0;
for (int i =1; i <= N; i++) {
   if (N%i==0) {
      respuesta++;
   }
}
cout << respuesta
```

Problemas de práctica

Problema 1.1 Dado una lista de N enteros, cuenta cuantas veces aparece el valor K.

- $1 \leq N \leq 10^5$

Enlace: TODO

Problema 1.2 Dado una lista de N enteros, imprime todos los múltiplos de 5 que estén en la lista y en el mismo orden que aparecen en la lista.

- $1 \leq N \leq 10^5$

Enlace: [TODO]

Problema 1.3 Cuenta cuantos enteros positivos dividen a K exactamente. Similar al ejemplo, pero los límites son más grandes.

- $1 \leq K \leq 10^9$

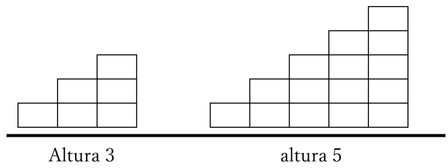
Enlace: 3 [TODO]

Problema 1.4 Encuentra el entero más grande que sea menor o igual a R y que la suma de sus dígitos sea menor o igual que S. Se garantiza que siempre existe este entero.

- $1 \leq S \leq 50$
- $1 \leq R \leq 10^5$

Enlace: omegaup.com/arena/problem/bl-1-4 [TODO]

Problema 1.5 Fernando construye escaleras de ladrillos de la siguiente forma:



El otro día, Fernando obtuvo K ladrillos, ahora se pregunta ¿Qué tan alto que puede construir una escalera? Dado K, responde su pregunta.

- $1 \leq K \leq 10^9$

Casos ejemplo

Entrada	**Salida**
12	4
8	3
20	5

Enlace: [TODO]

Búsqueda lineal con función de validación

Hasta ahorita hemos visto problemas donde revisar si un candidato era la respuesta o no bastaba con un simple condicional, pero este no siempre es el caso.

Varías veces, para revisar si un valor es solución a nuestro problema, vamos a tener que necesitar un poco más de código e ideas. Veamos un problema de este estilo.

Ejemplo 1.3

Descripción

Karel tiene N listones de distintas longitudes enteras. Karel quiere hacer pulseras con ellos, por lo que tomará cada uno de los listones y los cortará para que las pulseras usen segmentos del mismo tamaño.

A Karel le gustan los enteros, entonces la longitud de los segmentos también ha de serla. Además, Karel no quiere que sobre listón sin usar ¿Cuántos diferentes tamaños de segmento se pueden elegir?

Entrada

Un entero N, indicando la cantidad de listones En la siguiente línea, N enteros indicando las longitudes de los listones. Llamemos $A[i]$ a la longitud del listón i.

Salida

La cantidad de opciones para el tamaño de los segmentos

Caso ejemplo

Entrada	Salida	Expliación
5 10 30 20	3	Las longitudes pueden ser 1, 2 y 5

Límites

- $1 \leq N \leq 100$
- $1 \leq A[i] \leq 5000$

Código

Encontremos con búsqueda lineal todos los tamaños de segmento que cumplen y contemos cuantos son.

Primero veamos que los tamaños de listón deben estar entre 1 y 5000. Más concreto, entre 1 y $min(A[1], A[2], \cdots, A[N])$. Esto es porque el tamaño del segmento debe ser entero, debe ser por lo menos 1 (ya que un segmento de tamaño 0 o menor no tiene sentido para este problema) y no puede ser más largo que el listón más corto.

Ya hemos visto como se ve una búsqueda lineal y la que usaremos en este caso sería:

```
cin >> N;
for (int i=0; i< N; i++) {
   cin >> A[i];
}

int minA=A[0];
for (int i=1; i < N; i++) {
   minA=min(minA, A[i]); /* encuentra el liston mas
   pequeno. */
}
respuesta = 0;
```

```
for (int s =1; s <= minA; s++) {
   if (es s es un tamano de segmento valido) {
      respuesta++;
   }
}
cout << respuesta
```

Pero el reto ahora es el chequeo de “es s es un segmento de tamaño valido”.

Para esto necesitamos un poco más de trabajo. Veamos un solo listón. Si queremos cortarlo en segmentos de tamaño s sin que sobre, ¿qué tiene que cumplir s con relación al listón? Así es, que es, que s divida a la longitud del listón. Y podemos ver que s tiene que cumplir esto para todos los listones

Entonces, para ver que s sea una opción válida, hay que ver que s divida a todos los enteros en la lista de listones.

Para lograr esto, creemos una función booleana que se encargue de validar s.

```
bool validar (int s) {
   bool respuesta = true;
   for (int i=0; i< N; i++) {
      if (A[i]%s!=0){
         respuesta = false;
         break;
      }
   }
   return respuesta;
}
```

Entonces con esta función obtenemos que el código de la búsqueda lineal ahora es:

```
respuesta = 0;
for (int s =1; s <= minA; s++) {
   if (validar(s)) {
      respuesta++;
   }
}
cout << respuesta
```

Y con esto logramos completar el problema.

Complejidad

La búsqueda lineal la hacemos sobre el valor de A, pero, además, por cada iteración de la búsqueda lineal, hacemos un ciclo que revisa la condición para que s sea contada.

Entonces, la complejidad nos queda como: $O(Busqueda \times Validar) = O(AN)$.

Como $A \leq 5000$ y $N \leq 100$. Nos queda que $AN \leq 5 \times 10^5$, lo cual corre en menos de un segundo.

Conclusión

Habrá veces que en la búsqueda tengamos que hacer código para validar cada contacto que encontremos, y la complejidad de la validación se meterá como un factor a la complejidad de la búsqueda.

Con esto obtenemos un algoritmo con complejidad $O(Busqueda \times Validar)$.

Problemas de práctica

Problema 1.6 Karel ha comprado una bicicleta eléctrica con la que planea completar un recorrido. El recorrido se puede ver como N colinas en línea recta tal que la i-ésima colina tiene altura h_i. Karel comienza en la colina hasta la izquierda y quiere terminar en la ultima colina de hasta la derecha.

Cuando Karel sube un metro gasta 1 unidad de energía, mientras que bajar un metro recupera 1 unidad de altura. Si Karel en algún momento necesita subir, pero su batería tiene 0 de energía, Karel se quedará atorado y no terminará el recorrido.

Por suerte al inicio hay una estación de recarga donde Karel puede recargar su bicicleta. Como nota, la batería tiene capacidad R y jamás podrá almacena más energía que R.

Actualmente Karel tiene 0 de energía, Determina cuál es la menor cantidad de energía que es necesaria recargar al inicio para completar el recorrido. O determina si es imposible hacer el recorrido con la bicicleta de Karel.

Entrada

La primera línea tiene dos enteros, el valor de N y R.

En la siguiente línea vienen N, enteros separados por espacios, siendo la altura de las colinas de izquierda a derecha. Recuerda que Karel comienza en la primera colina y quiera terminar en la última.

Salida

Un entero, representando la menor cantidad de energía necesaria para completar el recorrido. Si Karel no puede completar el recorrido, imprime -1.

Casos ejemplo

Entrada	Salida	Expliación
6 8 4 6 3 5 7	3	Karel inicia con 3 de energía, moverse de la primera a la segunda colina le toma 2, ahora tiene 1. Luego avanza y se recarga 3, ahora tiene 4. Después continua y se consume 2, ahora tiene 2. Vuelve a avanzar quedándose con 0 de energía. Pero luego avanza y se recarga a 5. Finalmente avanza para termina con 5.
5 6 1 10 1 2 0	-1	

Límites

- $2 \leq N, R \leq 5000$
- $0 \leq h_i \leq 5000$

Fuente: OMIS online 2022.

Enlace: [TODO]

Problema 1.7 Un número capicúa es aquel que no cambia cuando se escribe al revés, por ejemplo 34143 y 1221 son capicúa, pero 145 no lo es porque $145 \neq 541$. Tampoco 30 es capicúa.

Determina cuantos números son capicúas entre L y R.

Ejemplo

Para $L = 1$ y $R = 50$ la respuesta es 13.

Límites

$1 \leq 1 \leq L \leq R \leq 10^5$

Problema 1.8 Dado L y R, encuentra cuantos números primos[1] hay entre L y R, incluyendo L y R.

Entrada

Dos enteros L y R.

Salida

La cantidad de números primos en el rango.

Caso ejemplo

Entrada	**Salida**	**Expliación**
1 7	4	Los primos contados son $2, 3, 5$ y 7

Límites

- $1 \leq L \leq R \leq 10^5$

Enlace: [TODO]

[1] Un número es primo si y solo si tiene exactamente dos divisores positivos, el 1 y el mismo.

Búsqueda completa

La búsqueda completa, también llamada fuerza bruta, es una técnica donde revisamos todos los posibles candidatos donde podría estar el o los valores que buscamos.

Esta búsqueda completa tiende a ser lenta, muchas veces incluso tiene complejidad exponencial y por esto, tiende a no ser una solución para los 100 puntos. Sin embargo, es muy útil conocerla ya que la fuerza bruta es fácil de pensar y siempre encuentra la respuesta si esta existe.

Ademas, la fuerza bruta suele resolvernos una o dos subtarea, dando un puntaje parcial. Esto es una forma de garantizar puntos en problema donde no se nos ocurra ideas mejores, es nuestra e también hay varios problemas que consisten en empezar de la fuerza bruta e ir mejorando la solución hasta que sea suficientemente buena.

También llega a ser útil para encontrar patrones y condiciones en las respuestas que no nos percatemos viendo la redacción, o para encontrar errores en un código que hayas hecho.

Probablemente esto te suene a búsqueda lineal, y esto es resultado de que esta es un tipo de búsqueda completa en la que revisamos de forma secuencial un rango donde se puede encontrar la respuesta. Pero búsqueda completa incluye más tipos de iteraciones que solo revisar los valores de un ciclo.

Las búsquedas completas suelen buscar la respuesta en:

- Los elementos de un arreglo (búsqueda lineal)
- Los valores de un rango (búsqueda lineal)
- Las parejas
- Los subconjuntos
- Los ordenes (permutaciones)

Para esto, a continuación analizaremos cada una de estos casos y veamos como trabajar con ellos.

Pares de elementos

En muchos problemas nos pedirán que encontremos o contemos la cantidad de pares que cumplan alguna condición, o nosotros convertiremos a un problema de este estilo. Para este tipo de problemas será útil conocer como hacer una búsqueda completa que revise todas las posibles parejas de elementos.

Como antes, veamos un problema que puede ser resuelto con esto.

Ejemplo 2.1.1

Fernando necesita K tornillos de la ferretería. Sin embargo, la vida no siempre es fácil y la ferretería no vende exactamente K tornillos.

Sin embargo, venden N cajas de tornillos cada una con C_i tornillos dentro.

Como Fernando tiene una obsesión con no desperdiciar, él solo comprara las cajas de forma que traigan juntas exactamente K tornillos. Además, odia las bolsas de un solo uso que dan en la ferretería por lo que solo comprará dos cajas, una por cada mano.

Entonces, dado el tamaño de las cajas, determina si Fernando puede traer consigo exactamente K tornillos.

Entrada

Dos enteros, N y K, representando cuantas cajas hay y cuantos tornillos se requieren.

En la siguiente línea vendrán N enteros separados por espacios, indicando la cantidad de tornillos en cada caja.

Salida

Deberás imprimir “SI” en caso de que Fernando pueda obtener K tornillos con sus reglas, o “NO” si es imposible.

Casos ejemplo

Entrada	Salida	Expliación
4 6 3 1 8 5	SI	Usa las cajas con 1 y 5 tornillos.
5 10 3 1 8 5 12	NO	

Límites

- $1 \leq N \leq 1000$
- $1 \leq K \leq 10^9$
- $1 \leq C_i \leq 10^9$
- $C_i \neq K$

Enlace: [TODO]

Solución

Lo que nos pregunta el problema es: ¿Existe un par de cajas tal qué sumen K?

Para determinar si existe dicha pareja, lo que haremos será buscar entre todas las parejas de cajas aquella que sume K. Es decir, buscaremos completamente todas las parejas posibles.

Para iterar por todas las parejas lo que haremos es primero definir una pareja como dos índices (i, j), tal que $0 \leq i < j < N$. Como queremos iterar por todos los posibles pares, primero iteraremos por todos los valores de i. Y para cada i, iteraremos por todas las j con las que se puede emparejar. El código se ve como:

```
for (int i =0; i < N; i++) {
   for (int j=i+1; j< N; j++) {
      cout << i<<" "<<j<< "\n";/* imprimimos
      cada par */
   }
}
```

Y ahora que sabemos iterar por todos los pares, lo utilizamos para revisar si existe un par que sume K.

```
for (int i =0; i < N; i++) {
   for (int j=i+1; j< N; j++) {
      if (Caja[i]+Caja[j] == K) {
         cout << "SI";
         exit(0); /* Termina el programa,
         encontramos la respuesta */
      }
   }
}
cout << "NO";
```

Entonces, son estos dos ciclos for nos permiten iterar por toda pareja de elementos en un arreglo. Esta es una herramienta bastante útil para resolver muchos problemas y subtareas.

Complejidad

Bien, ahora hablemos de la complejidad de esta técnica. La complejidad es $O(N^2)$. Esto es porque la cantidad de parejas con N elementos crece en$O(N^2)$.

Pero incluso si no conocemos como crecen las parejas, podemos ver que este ciclo para $i = 0$, itera por $N - 1$ valores de j; para $i = 1$, iteramos por $N-2$ valores de j; para $i = 2$, iteramos por $N-3$ valores de j, y así sucesivamente. De forma que hacemos $(N-1)+(N-2)+\ldots+1+0$ iteraciones de j. Entonces hacemos $1+2+3+\ldots+(N-1)$ iteraciones.

Usando la formula se suma de gauss[2] obtenemos que:

$$1 + 2 + 3 + \ldots + (N - 1) = \frac{N(N - 1)}{2} = \frac{N^2}{2} - \frac{N}{2}$$

Entonces, la complejidad queda como $O(\frac{N^2}{2} - \frac{N}{2}) = O(N^2)$

Por lo tanto, iterar por todos los pares de un arreglo es una técnica de complejidad cuadrada, perfecta para límites hasta $\sim 10^4$.

[2]Formula de gauss para sumar los primeros N naturales: $1+2+3+\ldots+N = \frac{N(N+1)}{2}$

Problemas de práctica

Problema 2.1.1 Definimos una inversión como una parejas (i, j) en un arreglo tal que $1 \leq i < j \leq N$ y también $A_i > A_j$. Es decir, una pareja de números que estén al revés de como deberían estarlo en un orden de menor a mayor.

Dado un arreglo de N enteros, imprime cuantas inversiones hay en él.

Ejemplo

Entrada	Salida	Expliación
4 3 2 6 1	4	Las inversiones son: El 3 con el 2, el 3 con el 1, el 2 con el 1, y el 6 con el 1.

Límites

- $1 \leq N \leq 5000$
- $1 \leq A_i \leq 10^9$

Enlace: [TODO]

Problema 2.1.2 Fernando regreso el otro día a la tienda de tornillos, y esta vez quiere darle una puntuación en un sitio web de guías locales. Fernando juzga la calidad de la tienda en función de cuantas cantidades diferentes de tornillos él puede comprar.

Recordemos que la tienda vende N cajas de tornillos, cada una con C_i dentro. Recordemos que Fernando solo puede tomar una o dos cajas en una compra porque solo tiene dos manos.

Dado las cajas de tornillos, determina la calidad de la tienda.

Ejemplo

Entrada	Salida	Expliación
3 1 3 4	5	Fer puede hacer compras de 1, 3, 4, 5 o 7 tornillos.

Límites

- $1 \leq N \leq 10^5$
- $1 \leq A_i \leq 10^3$

Enlace: [TODO]

Problema 2.1.3 La revista "algofashion" dijo esta semana que los números a la moda son aquellos que pueden ser representados como la suma de dos números pertenecientes a la secuencia de Fibonacci.

Recordemos que la secuencia de Fibonacci empieza con dos 1. Y luego cada número será resultado de la suma de los dos anteriores.

$$F_1 = F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

De forma que los primeros números de la secuencia son:

$$1, 1, 2, 3, 5, 8, 13, 21 \ldots$$

Karel acaba de leer la revista y ahora quiere responder T preguntas, cada pregunta será del tipo: ¿El número x_i está de moda?

Como amigo, debes hacer un código que responda las dudas de Karel.

Entrada

En la primera línea vendrá el valor de T, cuantas preguntas tiene Karel.

En las siguientes T líneas vendrán las preguntas de Karel, una por línea. Cada pregunta consiste en un solo entero x_i.

Salida

Imprime T líneas, cada una siendo la respuesta a una pregunta de Karel. La línea i debe ser "SI" si x_i está a la moda y debe ser "NO" si no está a la moda.

Caso ejemplo

Entrada	**Salida**
3 5 6 10	SI NO SI

Límites

- $1 \leq T \leq 100$
- $1 \leq x_i \leq 10^{18}$

Enlace: [TODO]

Problema 2.1.4 Tú tiene una constructora que busca comprar un terreno para construir un fraccionamiento.

Actualmente, el gobierno de Karelopolis permite que compres un terreno en la región para desarrollo. La región para desarrollo se ve como una cuadricula de N filas y M columnas. El terreno que compres debe ser rectangular y alineada a la cuadricula, es decir, no puedes comprar una celda de la cuadricula de forma incompleta.

Además, tu calculaste cuanto dinero obtendrías si compras cada celda, en concreto, la celda que esta en la fila i y columna j aportaría v_{ij}

pesos. Como hay celdas con valor positivo y otras con valor negativo, te preguntas cual es el máximo valor que puedes obtener.

Por ejemplo, si la región para desarrollo se ve de la siguiente forma:

1	-100	1	2
-3	5	2	-4
10	6	-1	4
-105	5	2	-100

Tu puedes obtener hasta 19 pesos comprando el terreno:

1	-100	1	2
-3	5	2	-4
10	6	-1	4
-105	5	2	-100

Encuentra el máximo valor posible si compras un terreno dada la región para desarrollo.

Entrada

En la primera línea recibes N y M.

En las siguientes N líneas recibirás M enteros representando los valores de v_{ij}.

Salida

Un entero que sea el mayor valor de un terreno.

Ejemplo

Entrada	Salida
4 4 1 -100 1 2 -3 5 2 -4 10 6 -1 4 -105 5 2 -100	19

Límites

- $1 \leq N, M \leq 75$
- $-10^9 \leq v_{ij} \leq 10^9$

Subtareas

- (40 pts) $1 \leq N, M \leq 15$
- (60 pts) Sin consideraciones extra.

Enlace: [TODO]

Problema 2.1.5 Dado un arreglo A de N enteros. Determina si existe un par de elementos que su suma modulo P sea K .

Límites

- $1 \leq N, K, P \leq 10^5$
- $1 \leq A_i \leq 10^9$

Subtareas

- (25 puntos) $1 \leq N \leq 10^3$
- (50 puntos) $1 \leq K, P \leq 10^3$
- (25 puntos) Sin restricciones extra.

Codigo: [TODO]

Subconjuntos

Antes de aprender a buscar subconjuntos, comencemos definiendo que son. Una vez sepamos que son, veremos como resolver problemas con ellos.

Definición de conjuntos y subconjuntos

Un conjunto no es nada mas que una colección de objetos. Por ejemplo, si alguien trae en su mochila un plátano, manzana y naranja, este puede decir que trae un conjunto de frutas compuesto por un plátano, manzana y naranja.

En matemáticas e informática, nosotros estamos trabajando todo el rato con los conjuntos. Ya sean el conjunto de datos de entrada, o los números enteros, tener noción de conjuntos es un requerimiento para el éxito en la olimpiada.

Veamos un poco de notación. Para escribir el conjunto A de números esta conformado por el 3, 5 y 9 escribimos lo siguiente:

$$A = \{3, 5, 9\}$$

TODO: CORREGIR ESTO

Buscar subconjuntos

Veamos como hacer para visitar todos los subconjuntos, lo cuál será necesario para problemas donde nos preguntan por ellos.

Para esto, primero aprendamos a resolver el problema que trata de mostrar los subconjuntos:

Ejemplo 2.2.1

Karel tiene un conjunto formado por las primeras N letras del alfabeto. Ahora Karel que imprimas todos los subconjuntos, uno por cada línea. Puedes imprimirlos en cualquier orden.

Cada subconjunto es representado por las letras en el de la A a la Z. El subconjunto vacío será representado por un asterisco '*'.

Entrada

Un entero N, indicando cuantas letras hay en el conjunto.

Salida

Todos los subconjuntos

Ejemplos

Entrada	**Salida**
2	AB A B *

Entrada	**Salida**
3	ABC AB AC A BC B C *

Límites

- $1 \leq N \leq 20$

Solución, iterar por subconjuntos

Entonces, podemos resolver el problema si obtenemos todos los subconjuntos posibles. Para esto, haremos un algoritmo que construya todos los posibles subconjuntos.

Esto se puede hacer de dos formas principales, una iterativa y otra recursiva. Comencemos viendo la forma con recursión.

Subconjuntos usando recursión

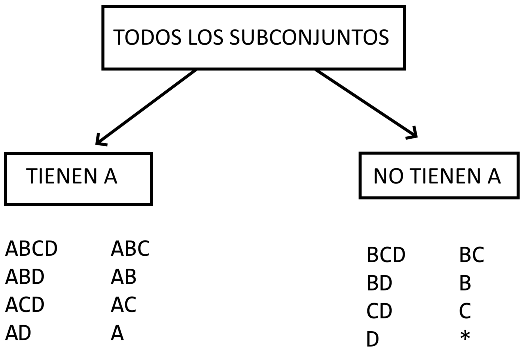
Digamos que $S(conjunto)$ sean todos los subconjuntos del *conjunto*. Por ejemplo:

$S(\{A, B, C\}) = \{\{A, B, C\}, \{A, B\}, \{A, C\}, \{A\}, \{B, C\}, \{B\}, \{C\}, \emptyset\}$

O usando la misma notación que la salida del problema (la que usaremos a partir de aquí):

$$S(ABC) = \{ABC, AB, AC, A, BC, B, C, *\}$$

Veamos un poco como se comporta S, por ejemplo para $N = 4$, en total tenemos 16 subconjuntos los cuales podemos dividir en dos mitades con 8 cada una, los que tienen la A y los que no tienen la A.



Ahora veamos que para los dos grupos, tenemos todos los 8 subconjuntos para BCD, y lo único que los diferencia es si tienen A al inicio o no. Si supiéramos cuales son esos subconjuntos para las ultimas 3 letras, podríamos perfectamente construir $S(ABCD)$,

En concreto, podemos ver que $S(ABCD)$ es igual a $S(BCD)$ con A al inicio y $S(BCD)$ sin nada extra.

Si lo quieres ver en formula sería similar a $S(ABCD) = A \to S(BCD) \cup S(BCD)$, donde $A \to S(BCD)$ significa agregar A a todos los subconjuntos en $S(BCD)$.

Y podemos hacer lo mismo, ver que $S(BCD)$ es otra vez: los subconjuntos de CD con B y sin B.

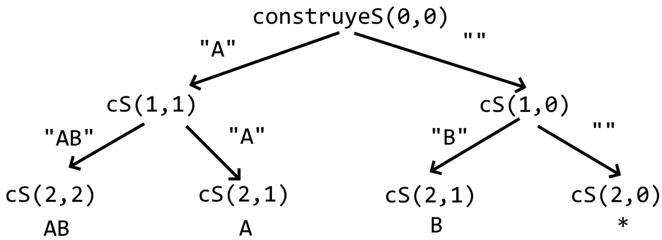
Y de aquí obtenemos nuestro comportamiento recursivo.

Ahora que vemos la recursión, veamos una forma de implementar todo esto para resolver el problema.

Crearemos un metodo construyeSubconjuntos(int pos, int previo) que constuya los subconjuntos usando las letras desde pos hasta $N - 1$.

```
#include <iostream>
using namespace std;
int N;
char subconjunto[25];

// pos implica que letra estamos eligiendo si esta o no A=0, B=1, C=2, ...
// para N=6, pos = 2, es equivalente a S(CDEF)
// previo indica cuantas letras se han agregaron a la construccion.
void construyeSubconjuntos(int pos, int previo) {
   if (pos == N) {
      //Ya no hay mas letras que decidir
      //imprime el subconjunto construido.
      if (previo==0) {
         cout <<"*\n";
      } else {
         cout << subconjunto<<"\n";
      }
      return;
   }
   //agrega la letra pos a la construccion
   subconjunto[previo]=pos+'A';
   construyeSubconjuntos(pos+1, previo+1);

   // Quita la letra pos de la construccion
   subconjunto[previo]='\0';
   construyeSubconjuntos(pos+1, previo);

}

int main () {
   cin >> N;
   construyeSubconjuntos(0,0);
   return 0;
}
```

De forma que el árbol de la recursión para construyeSubconjuntos(0,0) con $N = 2$ se ve:



Un consejo para entender el código es simular la ejecución del programa paso a paso a mano en papel. Esto es útil cualquier otro algoritmo u técnica nueva.

Complejidad

Una vez comprendido esto, pasemos a utilizar esto para resolver problemas.

Ejemplo 2.2.2

Fernando ha llegado a la ferretería con su objetivo frecuente de comprar K tornillos. Sin embargo, la ferretería no siempre vende cajas con exactamente K tornillos dentro.

La ferretería tiene N cajas diferentes en venta, cada uno con C_i tornillos dentro.

Fernando quiere comprar unas cuantas cajas y obtener **exactamente** K tornillos. Por fortuna, esta vez trajo una bolsa y podrá comprar cuantas cajas le sea necesario.

Conociendo las cajas que venden en la Ferretería, determina si Fernando puede comprar los K.

Entrada

Dos enteros N y K, la cantidad de cajas en la Ferretería y cuantos tornillos quiere Fernando.

En la siguiente línea vendrán la cantidad de tornillos en cada caja, los valores C_i, separados por espacios.

Salida

Debes imprimir "SI" en caso de que Fernando pueda comprar exactamente K tornillos. Caso contrario, imprime "NO".

Ejemplos

Entrada	**Salida**	**Expliación**
5 10 2 4 5 3 9	SI	Compra la primera, tercera y cuarta caja.
5 12 4 5 2 11 3	NO	

Límites

- $1 \leq N \leq 20$
- $1 \leq C_i \leq 10^9$
- $1 \leq K \leq 10^9$

Solución

Como siempre, resumamos el problema en menos palabras. En corto, nos preguntan si existe un subconjunto de cajas tal que la suma de sus valores sea exactamente K.

Y como es propio de todos los subtemas de búsqueda completa, veamos todas las posibles soluciones, es decir todos los subconjuntos, hasta que encontremos el que cumpla la condición o nos quedemos sin opciones.

Como ya vimos la forma de iterar por todos los subconjuntos en el ejemplo 3.1, utilicemos esas ideas para resolver este problema.

Lo que podemos hacer es ir construyendo todos los subconjuntos, esta vez en vez de representarlos como una cadena, lo representaremos como su suma. Esto en código se ve:

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int Cajas[25];
int N;
long long K;
// Aqui, llevamos la suma del subconjunto construido
void buscaSubconjunto(int pos, long long suma) {
   if (pos==N) {
      if (suma == K) {
         /* Encontramos un subconjunto con suma K.
         Imprimmos SI y terminemos el programa */
         cout << "SI";
         exit(0);
      }
      return;
   }
   buscaSubconjunto(pos+1, suma+Cajas[pos]);//Incluyamos pos
       en el subconjunto
   buscaSubconjunto(pos+1, suma);//Excluyamos pos del
       subconjunto
}

int main () {
   ios_base::sync_with_stdio(0); cin.tie(0);
   cin >> N >>K;
   for (int i=0; i < N; i++) {
      cin >> Cajas[i];
   }
   buscaSubconjunto(0,0);
   //Si llegamos aqui es porque nunca encontramos la
       respuesta.
   cout << "NO";
   return 0;
}
```

Problemas de práctica

Problema 2.2.1 Warel se infiltro en una joyería con una mochila de capacidad W.

En esta joyería hay N diamantes, cada uno tiene un valor de v_i y un peso de w_i.

Warel puede robar cuantos diamantes quiera, pero el peso total de lo que se lleve no puede exceder a la capacidad de su mochila.

Determina la máxima cantidad de valor que Warel puede robar.

Entrada

En la primera línea viene el valor de N y W.

En la segunda línea vendrán los valores de los diamantes.

En la tercera y última línea vendrán los pesos de los diamantes.

Salida

La máxima suma de valor que Warel puede robar.

Caso Ejemplo

Entrada	**Salida**	**Expliación**
5 6 8 3 6 1 3 5 1 3 2 6	10	Toma el segundo, tercer y cuarto diamante. Tendrás valor de $10 = 3 + 6 + 1$ con peso $6 = 1 + 3 + 2$

Límites

- $1 \leq N \leq 20$
- $1 \leq W \leq 10^9$

- $1 \leq v_i, w_i \leq 10^9$

Enlace: [TODO]

Problema 2.2.2 Imprime todos los números binarios de N bits.

Imprime los números de menor a mayor y agrega 0s a la izquierda para que todos los números tengan la cantidad correcta de bits.

Ejemplo

Entrada	Salida
3	000 001 010 011 100 101 110 111

Límites

- $1 \leq N \leq 20$

Problema 2.2.3 Un día encuentras una maquina que te permite invertir horizontalmente o verticalmente una matriz cuantas veces quieras.

Invertir verticalmente significa intercambiar la primera y ultima fila, la segunda y penúltima fila, la tercera y antepenúltima fila, etc. Mientras que invertir horizontalmente es lo mismo, pero intercambias las columnas en vez de las filas.

Se te dan dos matrices A y B, ambas de tamaño $N \times M$. Determina si la maquina te permite convertir la matriz A en la matriz B.

Entrada

Los valores de N y M.

Las siguientes N filas tendrán M enteros cada una. Siendo la matriz A.

Y otra vez seguirán N filas con M enteros cada una. Los valores de la matriz B.

Salida

Imprime "SI" en caso de que la matriz A pueda convertirse en la matriz B.

Ejemplos

Entrada	**Salida**
2 3 1 2 3 4 5 6 6 5 4 3 2 1	SI
2 2 1 2 3 1 2 3 1 1	NO

Límites

- $1 \leq N, M \leq 500$
- $1 \leq A_{ij}, B_{ij} \leq 10^9$

Enlace: [TODO]

Búsqueda binaria

Esta técnica es una de las más poderosas e importantes en la informática. Es una forma muy eficiente de hacer búsquedas que nos permite resolver problemas antes imposibles.

Utilizada en muchas aplicaciones, nos permite resolver problemas donde una búsqueda completa tardaría miles de millones de años en menos de un segundo. Y esta impresionante herramienta ahora será tuya.

La búsqueda binaria difiere de la completa al evitar revisar absolutamente todas las opciones del espacio de búsqueda. Si no, lo que hace es que en cada paso logra descartar la mitad del espacio de búsqueda al darse cuenta que la respuesta no está allí.

Porque en cada paso va descartando la mitad de las opciones, llega a una única opción (la respuesta si esta existe) en muy poco tiempo. Mientras que la búsqueda completa va eliminando opciones de una en una.

Comparemos un ejemplo de la búsqueda completa vs la binaria, si tuviéramos 128 candidatos.

Número de pasos	Candidatos restantes	
	Completa	Binaria
0	128	128
1	127	64
2	126	32
3	125	16
4	124	8
5	123	4
6	122	2
7	121	1
8	120	
	...	
125	3	
126	2	
127	1	

Como vemos, búsqueda binaria pudo reducir los candidatos a solo uno con siete pasos, mientras que la búsqueda completa requirió de 127.

El motivo por el cuál la búsqueda binaria es tan efectiva, es que realiza $\lceil log_2(candidatos) \rceil$ pasos [3]. Esto es porque $\lceil log_2 \rceil$ nos permite calcular cuantas veces podemos dividir un número entre dos hasta que sea 1.

Veamos un ejemplo que se puede resolver con búsqueda binaria.

[3] $\lceil log_2(A) \rceil$ significa: techo del logaritmo base dos de A.
Recordemos que $log_2(A) = x$ significa que $2^x = A$.
Y el techo nos dice que tomemos el menor entero que sea mayor igual que el valor de adentro, por ejemplo:. $\lceil 3.12 \rceil = 4$

Ejemplo 3.1

Javier es un granjero y esta cansado de que sus animales siempre huyan de su terreno, por lo que ha decidido poner una reja al rededor de todo el terreno.

Sin embargo, Javier no sabe cuantos metros de reja va a necesitar y te ha contratado para que tu le digas esto.

Lo que él si sabe es que el tiene forma cuadrada con lados de longitud entera, además recuerda que este mide A metros cuadrados de área.

Ayuda a Javier para que sepa cuanta reja necesita.

Entrada

Un entero A, el tamaño del terreno en metros cuadrados.

Salida

Un entero indicando cuantos metros de reja necesita para rodear todo el terreno.

Ejemplos

Entrada	**Salida**
36	24
100	40
1	4

Límites

- $1 \leq A \leq 10^{18}$

Subtareas

- (35 pts) $1 \leq A \leq 10^9$
- (65 pts) Sin restricciones adicionales

Solución

Como siempre, antes de leer la solución te invitamos a que intentes el problema.

En este problema nos piden de que dado el área de un cuadrado, imprimamos el perímetro. Por esto, recordemos las formulas del cuadrado.

$$Perímetro = 4 \times Lado$$

$$Área = Lado \times Lado$$

Entonces, si encontramos el lado que nos de el área de entrada, podremos encontrar el perímetro.

Para este problema vamos a ver dos estrategias para resolverlo, la de búsqueda lineal y la binaria.

Comencemos con la que ya deberíamos estar familiarizada, usemos búsqueda lineal.

Búsqueda lineal

Entonces, buscaremos el entero L de entre todos los posibles, que cumpla que $A = 4 \times L$. Para esto, podremos ir probando del 1 en adelante hasta encontrar el que cumpla.

```
long long L=1;
while (L*L != A) {
   L++;
}
```

Esto itera por todos los enteros del 1 a la respuesta que es $\sqrt{A}$. Por lo tanto, su complejidad es $O(\sqrt{A})$. Lo cual corre para la subtarea de 35 puntos, pero no para el límite completo de 10^{18}.

Búsqueda binaria

Ahora resolvamos el problema utilizando búsqueda binaria.

Definamos nuestro espacio de búsqueda, ¿cuáles valores puede ser L? y si lo pensamos puede ser desde 1 hasta 10^9. Porque $10^9 \times 10^9 = 10^{18}$ y de los límites sabemos que $1 \leq A \leq 10^{18}$.

Entonces queremos encontrar la respuesta que esta entre 1 y 10^9, hagamos una función que logre esto llamada buscar que reciba el rango de donde esta la respuesta.

```
//Encuentra L tal que L*L=A, sabiendo que a<=L<=b.
long long buscar(long long a, long long b, long long A);
```

Ahora tenemos 10^9 candidatos donde encontrar la respuesta y queremos reducirlo a la mitad.

Para esto podemos ver que sucede con 5×10^8.

Si resulta que $(5 \times 10^8) \times (5 \times 10^8) < A$, entonces sabemos que cualquier valor menor igual que 5×10^8 no funcionará porque es demasiado pequeño. Por lo tanto la respuesta debe estar entre $5 \times 10^8 + 1$ y 10^9 y los candidatos se redujeron a la mitad.

Pero si en vez sucede que $(5 \times 10^8) \times (5 \times 10^8) >= A$, entonces sabremos que cualquier valor más grande que 5×10^8 nos dará valores más grandes que A y por lo tanto la respuesta no estará allí. Ahora nuestros candidatos son los números entre 1 y 5×10^8, reduciendo los valores que podrían ser la respuesta a la mitad.

Y de hecho, si en general, la respuesta esta entre a y b, nos convendrá preguntar por el punto medio $(a + b)/2$, que nos reducirá el espacio a la mitad.

Una vez que redujimos el rango de la búsqueda, tendremos que encontrar la respuesta en ese nuevo rango, y para esto podemos hacer recursión.

De forma que ahora tenemos:

```
long long buscar(long long a, long long b, long long A) {
   long long m =(a+b)/2;
   if (m*m < A) {
      return buscar(m+1, b, A);
   } else {
      return buscar(a, m, A);
   }
}
```

Ahora, lo que le falta a esa recursión es una condición de paro, saber cuando ya termino y encontramos la respuesta.

Podremos ver que habremos encontrado la respuesta cuando ya estemos seguros de cual es esta. Y esto sucede cuando nuestro rango solo incluye un valor, es decir, cuando $a == b$ se cumpla.

```
long long buscar(long lonng a, long long b, long long A) {
   if (a==b)
      return a;
   long long m=(a+b)/2;
   if (m*m<A) {
      return buscar(m+1, b, A);
   } else {
      return buscar(a, m, A);
   }
}
```

Y con esto, tenemos que `L=buscar(1, 1000000000, A)`.

Ahora, la complejidad de esto es $O(log(\sqrt{A}))$, lo cual corre perfectamente para 10^{18}.

Nota: En este problemas también se pudo haber usado `L=sqrt(A)` con la librería `<math.h>` que calcula el valor rápidamente, pero se uso búsqueda binaria para ejemplificar.

Ejemplo 3.2

Se te da un arreglo A de N enteros diferentes. El arreglo estará en orden creciente, es decir, $A[i] < A[i+1]$.

Deberás responder T preguntas:

Cada pregunta consistirá de un entero q_i y tu deberás imprimir el índice del valor q_i o -1 si este valor no existe.

Entrada

El enteros N.

En la siguiente línea: N enteros separados, los valores del arreglo A.

En la siguiente línea recibirás el entero T.

En las siguientes T líneas recibirás los valores de cada pregunta.

Salida

Imprime la respuesta cada pregunta en una línea en el mismo orden que el de lectura.

Ejemplo

Entrada	Salida
7 2 5 6 7 8 9 10 5 2 6 4 10 5	0 2 -1 6 1

Límites

- $1 \leq N, T \leq 10^5$
- $1 \leq A[i], q_i \leq 10^9$

Enlace: TODO

Solución

Esta ocasión nos piden encontrar el indice de un valor en un arreglo ordenado. Ya hemos visto como hacerlo con búsqueda lineal:

```
int indice(int q) {
   for (int i =0; i < N;i++)
      if (A[i]==q)
         return i;
   return -1;
}
```

Sin embargo, esto no corre en tiempo ya que la complejidad es $O(N)$ por pregunta, siendo en total $O(TN)$ y como $TN = 10^{10}$, obtendremos TLE.

Pero veamos que podemos usar búsqueda binaria. ya que al preguntar por una posición de en medio y discernir si tenemos que buscar adelante o atrás.

```
int binaria(int a, int b, int q) {
   if (a>b)
      return -1;
   int m =(a+b)/2;
   if (A[m]==q)
      return m;
   if (A[m]<q) {
      return binaria(m+1, b, q);
   } else {
      return binaria(a, m-1, q);
   }
}
int indice(int q) {
   return binaria(0, N-1, q);
}
```

La nueva complejidad ahora es $O(logN)$ por pregunta, en total $O(TlogN)$

Dificultades

Ya vimos que la búsqueda binaria tiene una ventaja enorme sobre la búsqueda lineal ya que resuelve el problema en un tiempo mucho menor.

Pero tristemente, no siempre es posible aplicar la búsqueda binaria. Hay veces en que las que no se puede descartar fácilmente la mitad de los candidatos.

Un caso donde puede suceder sería en el ejemplo anterior si el arreglo no estuviese ordenado. En ese caso no podríamos hacer el truco de solo revisar adelante si $A[m] < q$ ya que no nos da suficiente información esa pregunta para eliminar todos los valores de 0 a m.

Por supuesto, el problema anterior tiene corrección para que la búsqueda binaria siga funcionando y muchas veces esto es parte del problema, ¿cómo hago la binaria aquí? Pero hay otras veces que es imposible, o al menos, más allá de los conocimientos actuales.

En esos casos, no quedará de otra más que hacer búsqueda completa o usar una técnica diferente a búsqueda.

Otro ejemplo donde la búsqueda binaria no se puede utilizar es en encontrar el primer divisor que no sea 1 de un entero.

Problemas de práctica

Problema 3.1 Adivina el número que esta pensando OmegaUp.

Deberás implementar una función `void adivina(long long a, long long b)` que descubra el número s que OmegaUp esta pensando. Se cumplirá que $a \leq s \leq b$.

Para adivinar el número podrás llamar a la función `long long pista(long long x)`.

Esta función regresará:

- `-1` si el número que piensa OmegaUp es menor que x.
- `0` si el número que piensa OmegaUp es x.
- `1` si el número que piensa OmegaUp es mayor que x.

La ultima llamada a pista debe ser con el número que OmegaUp esta pensando.

Límites

- $-2^{61} \leq a, b \leq 2^{61}$

Evaluación

Tu puntuación será en base a la cantidad de llamadas que hagas a la función `long long pista(long long x)` de la siguiente manera:

- 100% si haces a lo más $log(b-a+1)$llamadas
- 50% si haces a lo más $2log(b-a+1)$llamadas
- 0% si haces más de $2log(b-a+1)$llamadas

Enlace: www.omegaup.com/arena/problem/COMI-Adivina-el-numero

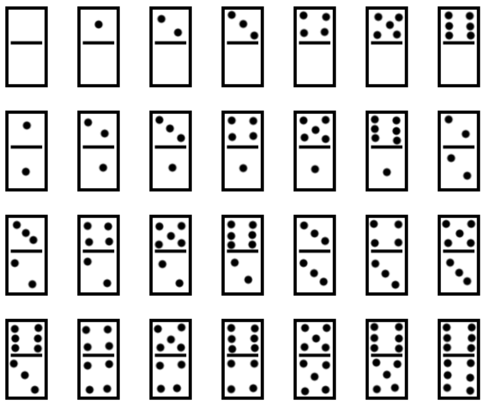
NOTA: Este es un problema interactivo, si no conoces como trabajar con estos, ve la página: **??**

Problema 3.2 Sho colecciona juegos de dominó y quiere jugar.

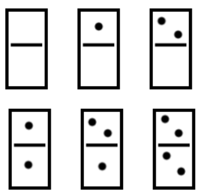
Pero los dominós de Sho son un poco especiales, ya que son k-dominós. El dominó tradicional es del tipo 6-dominó, porque tiene números del 0 al 6, con un total de 28 fichas.

Entonces, un k-dominó tendrá números del 0 al k. Con una ficha por cada par posible de números, incluyendo las mulas (mula es el mismo número emparejado consigo mismo).

Las fichas del 6-dominó son:



Y las del 2-domino son las siguientes seis:



Sho quiere jugar con al menos N fichas, pero como después de jugar tiene que guardar, quiere usar el dominó con menor cantidad de fichas que tenga por lo menos N fichas.

Dado N, encuentra el valor de k para el k-dominó que le sirve a Sho.

Ejemplo

Entrada	**Salida**
30	6
6	2
1000	44

Límites

- $1 \leq N \leq 10^{18}$

Enlace: TODO

Función de validación

Igual que en búsqueda lineal podíamos agregar funciones más complicadas a la hora de buscar la respuesta, lo mismo sucede en búsqueda binaria. A veces requeriremos de más código que una simple comparación para saber en cual mitad esta la respuesta.

Veamos unos ejemplos.

Ejemplo 3.3

TODO: PENSAR UN PROBLEMA CON FORMULA MAS COMPLICADA

Ejemplo 3.4

Karel ha comprado una bicicleta eléctrica con la que planea completar un recorrido. El recorrido se puede ver como N colinas en línea recta tal que la i-ésima colina tiene altura h_i. Karel comienza en la colina hasta la izquierda y quiere terminar en la ultima colina de hasta la derecha.

Cuando Karel sube un metro gasta 1 unidad de energía, mientras que bajar un metro recupera 1 unidad de altura. Si Karel en algún momento necesita subir, pero su batería tiene 0 de energía, Karel se quedará atorado y no terminará el recorrido.

Por suerte al inicio hay una estación de recarga donde Karel puede recargar su bicicleta. Como nota, la batería tiene capacidad R y jamás podrá almacena más energía que R.

Actualmente Karel tiene 0 de energía, Determina cuál es la menor cantidad de energía que es necesaria recargar al inicio para completar el recorrido. O determina si es imposible hacer el recorrido con la bicicleta de Karel.

Entrada

La primera línea tiene dos enteros, el valor de N y R.

En la siguiente línea vienen N, enteros separados por espacios, siendo la altura de las colinas de izquierda a derecha. Recuerda que Karel comienza en la primera colina y quiera terminar en la última.

Salida

Un entero, representando la menor cantidad de energía necesaria para completar el recorrido. Si Karel no puede completar el recorrido, imprime -1.

Casos ejemplo

Entrada	Salida	Expliación
6 8 4 6 3 5 7	3	Karel inicia con 3 de energía, moverse de la primera a la segunda colina le toma 2, ahora tiene 1. Luego avanza y se recarga 3, ahora tiene 4. Después continua y se consume 2, ahora tiene 2. Vuelve a avanzar quedándose con 0 de energía. Pero luego avanza y se recarga a 5. Finalmente avanza para termina con 5.
5 6 1 10 1 2 0	-1	

Límites

- $2 \leq N, R \leq 10^5$
- $0 \leq h_i \leq 10^9$

Fuente: OMIS online 2022.

Enlace: [TODO]

Solución

Este problema 1.6 de búsqueda lineal con validación en la página 13, si no sabes resolverlo con búsqueda lineal para los límites de allí, primero descubre esa solución.

Esta vez, los límites son más estrictos, de forma que la solución estándar con búsqueda lineal no funciona, pero veamos cual es porque nos será útil.

La respuesta siempre estará entre 0 yR. La búsqueda lineal funciona de la siguiente manera.

```
int respuesta=-1;
for (int e=0; e<=R; e++) {
   if (funciona(e)) {
      respuesta=e;
      break;
   }
}
```

Y la función `bool funciona(int e)` te regresa verdadero si Karel puede completar el recorrido comenzando con e de energía.

Esta función simplemente simula el recorrido para ver si Karel se atora en algún momento. Se ve de la siguiente forma:

```
bool funciona(int e) {
   for (int i = 1; i <N; i++){
      e-=A[i]-A[i-1];
      if (e > R)
         e=R; //Limita la energia
      if (e < 0)
         return false;
   }
   return true;
}
```

La función `funciona()` tiene una complejidad de $O(N)$ y como es llamada en R valores, la complejidad total es $O(RN)$.

Pero ahora veamos dos hechos importantes:

- Si funciona(m) cumple, también lo hará cualquier valor mayor que m.
- Si funciona(m) falla, también lo hará cualquier valor menor que m.

Es decir, se ve de la siguiente forma:

TODO PONER IMAGEN AQUI

Esto hace que podamos hacer búsqueda binaria para encontar el primer SI.

El código se verá como:

```
int a=0,b=R;
while(a!=b) {
   int m=(a+b)/2;
   if (funciona(m)) {
      b=m;
   } else {
      a=m+1;
   }
}
int respuesta=-1;
if (funciona(a))
   respuesta=a;
```

Ahora, recordemos que la complejidad de la búsqueda binaria es $O(logR)$, pero en cada paso de la búsqueda estamos llamando a `funciona()` que tiene complejidad $O(N)$, por lo tanto, la complejidad total es $O(NlogR)$.

Problemas de práctica

TODO: PONER PROBLEMAS DE PRACTICA AQUI

Problema 3.3 Palomitas CF

Problema 3.4 Dado un arreglo A de N enteros, determina si existe un subcadena que sume K.

Una subcadena es cualquier subarreglo de elementos continuos.

Búsqueda en los reales

Hay veces en la que no estamos buscando un valor entero, si no en vez querremos encontrar un valor real a cierta precisión.

El problema dirá algo por el estilo de: imprime la respuesta con precisión de 10^{-6}, esto quiere decir que tu respuesta no debe estar alejada de la solución por más de 10^{-6}.

Para estos problemas debes cambiar la condición de paro de $a == b$, por $b - a < precision$. Aunque en realidad, recomiendo poner una precisión un poco más ajustada ya que no tiende a aumentar mucho el costo.

De forma que ahora la binaria se verá como:

```
double a =0, b=1e9;
double epsilon = 1e-8;
while(b-a>= epsilon) {
   double m = (a+b)/2;
   if (funciona(m)) {
      b=m;
   } else {
      a=m;
   }
}
```

Ejemplo 3.5

Calcula la raíz cuadrada de un número A con precisión absoluta de 10^{-4}.

Ejemplo

Entrada	**Salida**
10	3.1622
16	4.0000

Límites

- $1 \leq A \leq 10^9$

Solución

Usamos las mismas observaciones que usamos para el ejemplo 3.1, pero esta vez lo haremos con double y nos detenemos basados en una precisión.

```
double raiz(double A) {
   double a=0, b=A;
   while (b-a >= 1e-5) {
      double m=(a+b)/2;
      if (m*m < A) {
         a=m;
      } else {
         b=m;
      }
   }
   return a;
}
```

Problemas de practica

Problema 3.5 Fernando ha adquirido un gusto por las carreras de coches en la fórmula π.

En este deporte, los coches recorren una pista recta que mide L metros de longitud. Además, dependiendo de resultados anteriores el coche i inicia con x_i metros de ventaja.

Como Fer es un aficionado muy dedicado, se ha percatado de que cada coche se moverá hacia la meta con una velocidad constante, en concreto, el .

Problema 3.6 Foto

www.omegaup.com/arena/problem/carretera

Búsqueda a profundidad

Búsqueda en amplitud

Problemas

P1

P2 Fuerza bruta

P3 Planetas:

www.omegaup.com/arena/problem/planetas

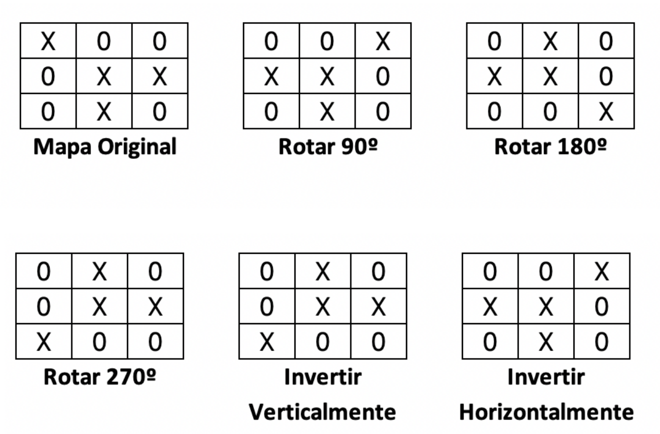
P4 El COMI diseñó unos mapas futuristas. Los mapas futuristas son como los mapas de ahora, pero están impresos en una superficie transparente. Ellos tienen un archivo de varios mapas de la galaxia. Sebastian y Héctor se dieron cuenta de que existen varios mapas repetidos, por lo que desean eliminar cualquier repetición. Para esto, Sebastian toma un mapa, Héctor toma otro y ambos comparan los mapas para determinar si son iguales. Sin embargo, Héctor estaba distraído y algunos mapas se los pasaba en una posición diferente a la original. Ahora necesitan tu ayuda para comparar los dos mapas y saber si son iguales.

Un mapa está representado por una matriz cuadrada de `X`s y `O`s. Dos mapas son iguales si ambos tienen el mismo carácter en la misma coordenada, para todas las coordenadas.

Para validar que dos mapas son iguales, puedes aplicar cualquiera de las siguientes acciones cero o más veces sobre un mapa que quieras comparar con otro:

- Rotarlo 90º.
- Rotarlo 180º.

- Rotarlo 270º.
- Invertirlo horizontalmente.
- Invertirlo verticalmente.



Problema

Se te mostrará dos mapas y debes ayudarles a decir si son iguales o diferentes. Si son iguales, deberás escribir `IGUALES`. Si son diferentes, deberás escribir `DIFERENTES`.

Entrada

En la primera línea N, la longitud del lado de cada mapa. En las siguientes N líneas una cadena de N caracteres `O` ó `X` que representan el primer mapa. En las siguientes N líneas una cadena de N caracteres `O` ó `X` que representan el segundo mapa.

Salida

Debes escribir `IGUALES` si los mapas son iguales después de hacer todas las transformaciones necesarias o `DIFERENTES` en caso contrario.

Ejemplo

Entrada	Salida	Expliación
4 XOOO XXOO OOOO XXXX XOOO XOOO XOXO XOXX	IGUALES	Si el segundo mapa lo giras 90º y además lo inviertes verticalmente, obtienes la misma distribución que el primer mapa.
2 XX OO XO OX	DIFERENTES	No hay manera de transformar el segundo mapa para que se vea tal como el primero.
4 XOOO XXOO OOOO XXXX XOOO XXOO OOOO XXXX	IGUALES	En este caso los mapas ya son iguales, sin necesidad de aplicar ninguna operación.

Límites

$1 \leq N \leq 500$

Subtareas

- (25 puntos)
  - $1 \leq N \leq 10$
  - Se asegura que, en caso de que los mapas sean iguales, no es

necesario aplicar ninguna acción.

- (25 puntos)
  - $1 \leq N \leq 50$
  - Se asegura que, en caso de que los mapas sean iguales, no es necesario aplicar ninguna rotación.
- (25 puntos)
  - $1 \leq N \leq 50$
  - Se asegura que, en caso de que los mapas sean iguales, no es necesario aplicar ninguna inversión vertical o inversión horizontal.
- (25 puntos)
  - $1 \leq N \leq 500$
  - Cualquier acción podría ser necesaria para validar que los mapas son iguales.

Fuente: **OMI 2020**

www.omegaup.com/arena/problem/OMI-2020-Mapas

P5 DUMMY TODO

P6 DUMMY TODO

P7 La siguiente lección en preparatoria requiere que dos temas sean discutidos. El i-ésimo tema es interesante por a_i unidades para el profesor y b_i unidades para los estudiantes.

Un par te temas i y j $(i < j)$ es llamado **bueno** si $a_i + a_j > b_i + b_j$ (es decir, es más interesante para el profesor).

Tu tarea es encontrar el número de parejas de temas **buenas**.

Entrada

Un entero n, la cantidad de temas.

La segunda línea tiene n enteros: $a_1, a_2, \ldots, a_n$, donde a_i es cuan interesante es el tema i para el profesor.

La tercera línea tiene n enteros: $b_1, b_2, \ldots, b_n$, donde b_i es cuan interesante es el tema i para los estudiantes.

Salida

Un entero, la cantidad de parejas buenas.

Ejemplo

Entrada	**Salida**
5 4 8 2 6 2 4 5 4 1 3	7
4 1 3 2 4 1 3 2 4	0

Límites

- $2 \leq N \leq 2 \cdot 10^5$
- $1 \leq a_i, b_i \leq 10^9$

Fuente: codeforces.com

www.codeforces.com/problemset/problem/1324/D

P8 La leyenda dice que el tesoro de Moctezuma está enterrado en el Centro Histórico de la Ciudad de México. El Centro Histórico está representado como una cuadrícula de n filas y m columnas. Gracias a la tecnología de la nueva app iFind puedes desenterrarlo por fin.

iFind es una app donde especificas una casilla (i, j) de la cuadrícula y la app responde cuántos tesoros hay enterrados en el área del rectángulo que abarca de la fila 1 a la fila i y de la columna 1 a la columna j.

Una vez que sabes la casilla exacta donde hay un tesoro debes cavar en esa posición para desenterrarlo.

Se asegura que cada casilla solo puede tener o tesoro y que no hay más de tesoro en cada columna.

Problema

Escribe un programa que dados n y m, el alto y ancho de la cuadrícula y k, el número total de tesoros enterrados, desentierre los k tesoros usando la menor cantidad posible de preguntas a la app.

Interacción

No necesitas leer o escribir[4], debes implementar en tu código la función `BuscaTesoros` y mandar llamar las funciones del evaluador `Preguntar` y `Cavar` para completar tu tarea.

Internamente el evaluador llevará el registro de cuántos tesoros quedan. Tu programa no necesitará imprimir ni devolver nada: solo asegurarse de que hayas desenterrado los tesoros usando la función Cavar.

Implementación

```
void BuscaTesoros(int n, int m, int k);
```

Descripción: El evaluador buscará en tu código esta función y la llamará con los parámetros `n`, `m` y `k`. Tu implementación deberá utilizar las funciones `Preguntar` y `Cavar` para desenterrar todos los tesoros. En cada caso de prueba solo se llamará a esta función una vez.

Parámetos

- `n`: Filas de la cuadrícula.
- `m`: Columnas de la cuadrícula.
- `k`: El número de tesoros enterrados.

[4]Este es un problema interactivo, si no conoces como trabajar con estos, ve la página: **??**

TODO COMPLETAR

Fuente: **OMI 2018**

www.omegaup.com/arena/problem/OMI2018-Tesoro