Olimpiada de Informática

—

—

Índice general

Parte I

Elementos básicos

Parte II

Búsquedas

Capítulo 1

Búsqueda exhaustiva
Fuerza bruta

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 esta parte trabajaremos un poco más en desarrollar esta intuición e ideas.

En la olimpiada de informática también debemos buscar cosas. Ya sea encontrar la solución al problema o solo utilizar una búsqueda como paso intermedio, ser buenos haciendo búsquedas nos abrirá la puerta a muchos problemas y técnicas.

Para comenzar con entendiendo las búsquedas comenzaremos con la búsqueda más sencilla, la búsqueda exhaustiva, también conocida como fuerza bruta.

Lo primero que se requiere para poder buscar es definir el espacio de búsqueda, ¿dónde podría estar lo que queremos encontrar? De la respuesta dependerá como haremos la búsqueda. Por ejemplo, si buscamos la posición de un valor en un arreglo, el espacio de búsqueda es el arreglo.

Una vez que definamos donde podría estar la respuesta, lo que hacemos con la

exhaustiva es explorar absolutamente todo el espacio de búsqueda, todos los candidatos que podrían ser lo que buscamos hasta dar con la respuesta. Por esto se le conoce como fuerza bruta, porque aprovecha el poder computacional para procesar todo hasta dar con la respuesta.

Se te enseñara las búsquedas exhaustivas más comunes, así como una forma general. En concreto, los espacios de búsquedas que usaremos son:

- Espacio lineal
- Parejas de elementos
- Ordenes (Permutaciones)
- Subcojuntos
- Cadena de decisiones

A continuación, veamos cada uno:

1.1. 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.

1.1.1. Ejemplo: Encontrar posición en un arreglo

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;
}
```

1.1.2. 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 $O(N)$.

1.1.3. Ejemplo: Contar formas de dividir

Veamos otro problema de 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 practica

Ejercicio 1.1 *TODO Frecuencia de K*
(omegaup.com/arena/problem/frecuencia-de-k)

Ejercicio 1.2 *TODO Multiplos de cinco*
(omegaup.com/arena/problem/multiplos-cinco)

Ejercicio 1.3 *TODO: Divisores del entero*
(omegaup.com/arena/problem/divisores-entero)

Ejercicio 1.4 *A la suma de digitos*
(omegaup.com/arena/problem/m-suma-digitos)

Ejercicio 1.5 *Escalera de fer*
(omegaup.com/arena/problem/)

1.2. 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.

1.2.1. 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.

1.2.2. 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.

Problemas de práctica

Ejercicio 1.6 *Bicicleta de Karel I*
(omegaup.com/arena/problem/bicicleta-de-karel-i)

Ejercicio 1.7 *Contar capicúas*
(omegaup.com/arena/problem/Contar-capicuas)

Ejercicio 1.8 *Cuenta primos*
(omegaup.com/arena/problem/Cuenta-primos)

Parte III

Matemáticas olímpicas

Parte IV

STL y estructura de datos

Parte V

Grafos

Parte VI

Técnicas de resolución de problemas

Parte VII

Problemas no estándar