Búsqueda Binaria

Quimey Vivas [1]
Leopoldo Taravilse
Facundo Gutierrez

[1] Avature Argentina

Training Camp 2018

Contenidos

Contenidos

Buscando en listas ordenadas

- Uno de los problemas más comunes que existen es el de querer buscar si un número aparece o no en una lista ordenada.

Buscando en listas ordenadas

- Uno de los problemas más comunes que existen es el de querer buscar si un número aparece o no en una lista ordenada.
- Es trivial ver que se puede resolver el problema en $O(n)$ donde n es la cantidad de elementos de la lista, simplemente buscando uno por uno en orden.

Buscando en listas ordenadas

- Uno de los problemas más comunes que existen es el de querer buscar si un número aparece o no en una lista ordenada.
- Es trivial ver que se puede resolver el problema en $O(n)$ donde n es la cantidad de elementos de la lista, simplemente buscando uno por uno en orden.
- Hay una forma de aprovechar el hecho de que el arreglo está ordenado. Esta forma de buscar eficientemente se conoce como búsqueda binaria.

Ejemplo de Búsqueda Binaria

Supongamos que queremos buscar si el número 22 está en el siguiente arreglo:

```
1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128
```

Ejemplo de Búsqueda Binaria

Supongamos que queremos buscar si el número 22 está en el siguiente arreglo:

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128

Podemos preguntarnos ¿Es el primer elemento del arreglo el 22? No, ¿Es el segundo elemento del arreglo el 22? Tampoco... y así hasta que nos preguntamos ¿Es el décimo elemento del arreglo el 22? Sí!

Ejemplo de Búsqueda Binaria

Con búsqueda binaria podemos buscar así

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Con búsqueda binaria podemos buscar así

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Con búsqueda binaria podemos buscar así

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Con búsqueda binaria podemos buscar así

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Con búsqueda binaria podemos buscar así

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

Notemos que en 4 pasos pudimos encontrar el 22 cuando antes nos llevaba 10 pasos.

Código de la búsqueda binaria

```
1  # A es el arreglo ordenado, v es el valor a buscar
2  # Si v esta en A entonces esta en el intervalo [L, R) de A
3  def busqueda_binaria(A, v):
4      L = 0
5      R = len(A)
6      while R - L > 1:
7          M = (L + R) / 2
8          if v < A[M]:
9              R = M
10         else:
11             L = M
12     if A[L] == v:
13         return L
14     return None  # Not found
```

¿Funciona? ¿Encuentra la primera aparición de *v*? ¿La última? ¿Una cualquiera?

Ejemplo de Búsqueda Binaria

Buscar un número que no está es mucho más costoso con búsqueda lineal, sin embargo con búsqueda binaria es igual de rápido que buscar un número que si está. Busquemos por ejemplo el 23.

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 22 31 35 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Buscar un número que no está es mucho más costoso con búsqueda lineal, sin embargo con búsqueda binaria es igual de rápido que buscar un número que si está. Busquemos por ejemplo el 23.

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** 35 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Buscar un número que no está es mucho más costoso con búsqueda lineal, sin embargo con búsqueda binaria es igual de rápido que buscar un número que si está. Busquemos por ejemplo el 23.

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** 35 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Buscar un número que no está es mucho más costoso con búsqueda lineal, sin embargo con búsqueda binaria es igual de rápido que buscar un número que si está. Busquemos por ejemplo el 23.

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** 35 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Buscar un número que no está es mucho más costoso con búsqueda lineal, sin embargo con búsqueda binaria es igual de rápido que buscar un número que si está. Busquemos por ejemplo el 23.

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** 35 44 45 56 87 89 95 99 100 103 112 128

Ejemplo de Búsqueda Binaria

Buscar un número que no está es mucho más costoso con búsqueda lineal, sin embargo con búsqueda binaria es igual de rápido que buscar un número que si está. Busquemos por ejemplo el 23.

1 2 4 6 8 14 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128 .

1 2 4 6 8 **14** 15 16 21 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 **14** 15 16 **21** 22 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 **21** **22** 31 **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** **35** 44 45 56 87 89 95 99 100 103 112 128

1 2 4 6 8 14 15 16 21 **22** **31** 35 44 45 56 87 89 95 99 100 103 112 128

Complejidad de la búsqueda binaria

La complejidad de la búsqueda binaria es $O(\log n)$ donde n es el tamaño del arreglo. Esto es fácil de probar ya que en cada paso se reduce el espacio de búsqueda a la mitad.

Contenidos

Funciones monótonas

Cuando tenemos una función f que cumple

$$x > y \Rightarrow f(x) \geq f(y)$$

decimos que f es monótona creciente. Si en cambio f cumple

$$x > y \Rightarrow f(x) \leq f(y)$$

decimos que f es monótona decreciente. Si f es monótona creciente o monótona decreciente decimos que f es monótona.
Cuando una función es monótona creciente (decreciente) podemos aplicar búsqueda binaria para buscar el menor valor de x tal que $f(x) \geq y$ ($f(x) \leq y$) para un y dado.

Ejemplo de búsqueda binaria con funciones monótonas

Pedro estudia Ciencias de la Computación y le quedan por cursar n materias. Él sabe que si hace una sóla materia le va a costar 1 unidad de esfuerzo aprobarla, pero si hace una segunda materia le cuesta 2 unidades de esfuerzo para aprobar la segunda materia, y en general, si hace i materias le cuesta i unidades de esfuerzo aprobar la i-ésima materia.
El sabe que hacer un esfuerzo de t unidades o más lo va a estrezar y por lo tanto va a dejar la carrera. Cuántos cuatrimestres necesita para recibirse?

Ejemplo de búsqueda binaria con funciones monótonas

- Lo primero que tenemos que hacer es calcular cuántas materias puede hacer por cuatrimestre. Luego es una división.

Ejemplo de búsqueda binaria con funciones monótonas

- Lo primero que tenemos que hacer es calcular cuántas materias puede hacer por cuatrimestre. Luego es una división.
- Para calcular cuántas materias puede hacer por cuatrimestre, sabemos que ese número x cumple con $\frac{x(x+1)}{2} < t$. Luego hacemos búsqueda binaria para encontrar ese máximo valor posible de x.

Contenidos

Búsqueda binaria continua

- Así como usamos búsqueda binaria cuando tenemos un dominio discreto, también podemos usar búsqueda binaria cuando el dominio es continuo.

Búsqueda binaria continua

- Así como usamos búsqueda binaria cuando tenemos un dominio discreto, también podemos usar búsqueda binaria cuando el dominio es continuo.
- Por ejemplo, tenemos una función f monótona creciente y continua y queremos hallar el mínimo x tal que $f(x) \geq y$.

Búsqueda binaria continua

- Así como usamos búsqueda binaria cuando tenemos un dominio discreto, también podemos usar búsqueda binaria cuando el dominio es continuo.
- Por ejemplo, tenemos una función f monótona creciente y continua y queremos hallar el mínimo x tal que $f(x) \geq y$.
- Como f es continua, si el mínimo x_0 existe tenemos qué $f(x_0) = y$, es decir $x_0 = f^{-1}(y)$.

Ejemplo de búsqueda binaria para encontrar el inverso de una función

- Dado un número x, calcular la raiz cuadrada de x

Ejemplo de búsqueda binaria para encontrar el inverso de una función

- Dado un número x, calcular la raiz cuadrada de x
- ¿Cuál sería el criterio de terminación? ¿Cuántas iteraciones necesitamos?

Ejemplo de búsqueda binaria para encontrar el inverso de una función

- Dado un número x, calcular la raiz cuadrada de x
- ¿Cuál sería el criterio de terminación? ¿Cuántas iteraciones necesitamos?
- A diferencia del caso de la búsqueda binaria discreta, cuando el dominio de la búsqueda binaria es continuo tenemos que poner un punto de corte arbitrario para la búsqueda binaria.

Ejemplo de búsqueda binaria para encontrar el inverso de una función

- Dado un número x, calcular la raiz cuadrada de x
- ¿Cuál sería el criterio de terminación? ¿Cuántas iteraciones necesitamos?
- A diferencia del caso de la búsqueda binaria discreta, cuando el dominio de la búsqueda binaria es continuo tenemos que poner un punto de corte arbitrario para la búsqueda binaria.
- Podríamos cortar por ejemplo cuándo $R - L < 10^{-9}$ que nos da una precisión aceptable.

Ejemplo de búsqueda binaria para encontrar el inverso de una función

- Dado un número x, calcular la raiz cuadrada de x
- ¿Cuál sería el criterio de terminación? ¿Cuántas iteraciones necesitamos?
- A diferencia del caso de la búsqueda binaria discreta, cuando el dominio de la búsqueda binaria es continuo tenemos que poner un punto de corte arbitrario para la búsqueda binaria.
- Podríamos cortar por ejemplo cuándo $R - L < 10^{-9}$ que nos da una precisión aceptable.
- Cuando el ciclo termina, tanto L como R contienen la respuesta con un error de a lo sumo 10^{-9}.

Ejemplo de búsqueda binaria para encontrar el inverso de una función

- Dado un número x, calcular la raiz cuadrada de x
- ¿Cuál sería el criterio de terminación? ¿Cuántas iteraciones necesitamos?
- A diferencia del caso de la búsqueda binaria discreta, cuando el dominio de la búsqueda binaria es continuo tenemos que poner un punto de corte arbitrario para la búsqueda binaria.
- Podríamos cortar por ejemplo cuándo $R - L < 10^{-9}$ que nos da una precisión aceptable.
- Cuando el ciclo termina, tanto L como R contienen la respuesta con un error de a lo sumo 10^{-9}.
- Método de Newton

Contenidos

Búsqueda binaria con predicados booleanos

- Ya vimos como hace búsqueda binaria sobre funciones que tienen una imagen numérica y que son monotonas.

Búsqueda binaria con predicados booleanos

- Ya vimos como hace búsqueda binaria sobre funciones que tienen una imagen numérica y que son monotonas.
- Cuando la imagen de la función es booleana, por ejemplo, dado un predicado booleano P, encontrar el mínimo x tal que $P(x)$ es verdadero, también podemos definir monotonía y aplicar búsqueda binaria.

Búsqueda binaria con predicados booleanos

- Ya vimos como hace búsqueda binaria sobre funciones que tienen una imagen numérica y que son monotonas.
- Cuando la imagen de la función es booleana, por ejemplo, dado un predicado booleano P, encontrar el mínimo x tal que $P(x)$ es verdadero, también podemos definir monotonía y aplicar búsqueda binaria.
- Cuando un predicado es falso para todos los $x < x_0$ y verdadero para los $x \geq x_0$ decimos que el predicado monótono. Esto equivale con asignarle 0 a falso y 1 a verdadero, en este caso el predicado es monótono creciente.

Contenidos

- El caso no monónoto:

- El caso no monónoto: Si estamos tratando de encontrar x tal que $f(x) = y$ y f es continua podemos usar búsqueda binaria aunque f no sea monónotona (por Bolzano).

- El caso no monónoto: Si estamos tratando de encontrar x tal que $f(x) = y$ y f es continua podemos usar búsqueda binaria aunque f no sea monónotona (por Bolzano).
- Sin cota superior:

- El caso no monónoto: Si estamos tratando de encontrar x tal que $f(x) = y$ y f es continua podemos usar búsqueda binaria aunque f no sea monónotona (por Bolzano).
- Sin cota superior: Si no tenemos una cota superior podemos usar búsqueda binaria para encontrarla.

- El caso no monónoto: Si estamos tratando de encontrar x tal que $f(x) = y$ y f es continua podemos usar búsqueda binaria aunque f no sea monónotona (por Bolzano).
- Sin cota superior: Si no tenemos una cota superior podemos usar búsqueda binaria para encontrarla.
- Búsqueda ternaria

- El caso no monónoto: Si estamos tratando de encontrar x tal que $f(x) = y$ y f es continua podemos usar búsqueda binaria aunque f no sea monónotona (por Bolzano).
- Sin cota superior: Si no tenemos una cota superior podemos usar búsqueda binaria para encontrarla.
- Búsqueda ternaria Si f es *unimodal* podemos usar búsqueda ternaria para encontrar el máximo (si f no es unimodal encontramos un extremo local).

- El caso no monónoto: Si estamos tratando de encontrar x tal que $f(x) = y$ y f es continua podemos usar búsqueda binaria aunque f no sea monónotona (por Bolzano).
- Sin cota superior: Si no tenemos una cota superior podemos usar búsqueda binaria para encontrarla.
- Búsqueda ternaria Si f es *unimodal* podemos usar búsqueda ternaria para encontrar el máximo (si f no es unimodal encontramos un extremo local).
- Optimización $\Rightarrow$ Decisión

Contenidos

Ejemplo 1

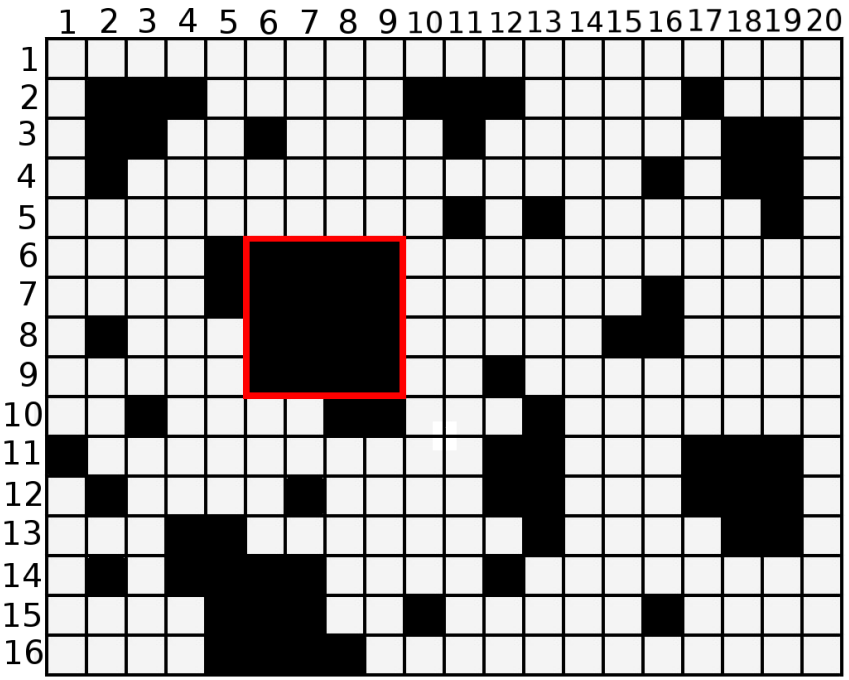
Máximo Cuadrado en un Tablero

Tenemos un tablero de $m \times n$ (m filas y n columnas). Algunas casillas del tablero están pintadas de blanco y otras de negro. Buscamos encontrar el tamaño del lado del subtablero cuadrado más grande de casillas negras.

Ejemplo 1

Máximo Cuadrado en un Tablero

Tenemos un tablero de $m \times n$ (m filas y n columnas). Algunas casillas del tablero están pintadas de blanco y otras de negro. Buscamos encontrar el tamaño del lado del subtablero cuadrado más grande de casillas negras.



Solución

Cosas que vamos a usar

Supongamos que tenemos una función booleana $f(k, i, j)$ que **nos dice si hay un cuadrado de casillas negras de lado** k en el tablero cuya casilla superior izquierda está situada en la fila i y la columna j. Por ejemplo, en la imagen anterior, $f(2, 2, 2)$ es `true` y $f(3, 2, 2)$ es `false`.

Solución

Cosas que vamos a usar

Supongamos que tenemos una función booleana $f(k, i, j)$ que **nos dice si hay un cuadrado de casillas negras de lado** k en el tablero cuya casilla superior izquierda está situada en la fila i y la columna j. Por ejemplo, en la imagen anterior, $f(2, 2, 2)$ es `true` y $f(3, 2, 2)$ es `false`.

1. Solución: Podemos **recorrer todas las casillas**, y en cada una preguntarle a la función f si existe un tablero de tamaño $0, 1, 2, \ldots, \min(n, m)$ inclusive. El valor más grande que devolvió `true` será el lado del cuadrado más grande que tiene la esquina superior izquierda en la casilla que estamos analizando. Tomando el máximo de este número para todas las casillas resolvemos el problema.

Solución

Cosas que vamos a usar

Supongamos que tenemos una función booleana $f(k, i, j)$ que **nos dice si hay un cuadrado de casillas negras de lado** k en el tablero cuya casilla superior izquierda está situada en la fila i y la columna j. Por ejemplo, en la imagen anterior, $f(2, 2, 2)$ es `true` y $f(3, 2, 2)$ es `false`.

1. Solución: Podemos **recorrer todas las casillas**, y en cada una preguntarle a la función f si existe un tablero de tamaño $0, 1, 2, \ldots, \min(n, m)$ inclusive. El valor más grande que devolvió `true` será el lado del cuadrado más grande que tiene la esquina superior izquierda en la casilla que estamos analizando. Tomando el máximo de este número para todas las casillas resolvemos el problema. ¿Cuántas veces estamos llamando a la función f?

Solución

Cosas que vamos a usar

Supongamos que tenemos una función booleana $f(k, i, j)$ que **nos dice si hay un cuadrado de casillas negras de lado** k en el tablero cuya casilla superior izquierda está situada en la fila i y la columna j. Por ejemplo, en la imagen anterior, $f(2, 2, 2)$ es `true` y $f(3, 2, 2)$ es `false`.

1. Solución: Podemos **recorrer todas las casillas**, y en cada una preguntarle a la función f si existe un tablero de tamaño $0, 1, 2, \ldots, \min(n, m)$ inclusive. El valor más grande que devolvió `true` será el lado del cuadrado más grande que tiene la esquina superior izquierda en la casilla que estamos analizando. Tomando el máximo de este número para todas las casillas resolvemos el problema. ¿Cuántas veces estamos llamando a la función f?

   Respuesta : $\boxed{n \cdot m \cdot \min(n, m) \sim n^3}$

Solución

¿Cómo podemos **acelerar la solución**? (hacer menos llamados a la f)

Solución

¿Cómo podemos **acelerar la solución**? (hacer menos llamados a la f)

Búsqueda binaria

Solución

¿Cómo podemos **acelerar la solución**? (hacer menos llamados a la f)

Búsqueda binaria Ok... ¿Pero en qué?

Solución

¿Cómo podemos **acelerar la solución**? (hacer menos llamados a la f)

Búsqueda binaria Ok... ¿Pero en qué?

2. En la solución anterior, cuando estamos parados en una casilla, podemos hacer búsqueda binaria en **el lado del cuadrado que buscamos**.

Solución

¿Cómo podemos **acelerar la solución**? (hacer menos llamados a la f)

Búsqueda binaria Ok... ¿Pero en qué?

2. En la solución anterior, cuando estamos parados en una casilla, podemos hacer búsqueda binaria en **el lado del cuadrado que buscamos**.

¿Cuántas veces estamos llamando a la función f?

Solución

¿Cómo podemos **acelerar la solución**? (hacer menos llamados a la f)

Búsqueda binaria Ok... ¿Pero en qué?

2. En la solución anterior, cuando estamos parados en una casilla, podemos hacer búsqueda binaria en **el lado del cuadrado que buscamos**.

¿Cuántas veces estamos llamando a la función f?

Respuesta : $\boxed{n \cdot m \cdot \lg(\min(n, m)) \sim n^2 \cdot \lg(n)}$

Ejemplo 2

Problema

Fito tiene que cruzar un cuarto lleno de dinosaurios desde la esquina superior izquierda hasta la esquina inferior derecha. Son dadas las posiciones de los dinosaurios y las dimensiones del cuarto. Se sabe que si pasa a distancia menor a T de un dinosaurio, este lo ve y se lo come. Cuál es el mínimo T tal que Fito puede lograr su objetivo?

Contenidos

Problema:

Dado un arreglo de n números **positivos**, y un número x, nos interesa saber si existe un subarreglo cuya suma sea x.

Ejemplos:

- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 8$
  La respuesta es : "El subarreglo [2..4] suma 8"
- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 4$
  La respuesta es : "No hay subarreglo de A que sume 4"

Problema:

Dado un arreglo de n números **positivos**, y un número x, nos interesa saber si existe un subarreglo cuya suma sea x.

Ejemplos:

- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 8$
  La respuesta es : "El subarreglo [2..4] suma 8"
- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 4$
  La respuesta es : "No hay subarreglo de A que sume 4"

Problema:

Dado un arreglo de n números **positivos**, y un número x, nos interesa saber si existe un subarreglo cuya suma sea x.

Ejemplos:

- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 8$
  La respuesta es : "El subarreglo [2..4] suma 8"
- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 4$
  La respuesta es : "No hay subarreglo de A que sume 4"

Problema:

Dado un arreglo de n números **positivos**, y un número x, nos interesa saber si existe un subarreglo cuya suma sea x.

Ejemplos:

- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 8$
  La respuesta es : "El subarreglo [2..4] suma 8"
- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 4$
  La respuesta es : "No hay subarreglo de A que sume 4"

Problema:

Dado un arreglo de n números **positivos**, y un número x, nos interesa saber si existe un subarreglo cuya suma sea x.

Ejemplos:

- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 8$
  La respuesta es : "El subarreglo [2..4] suma 8"
- $A = [2, 3, 2, 5, 1, 5, 2, 3]$. Buscamos sumar $x = 4$
  La respuesta es : "No hay subarreglo de A que sume 4"

Solución

1. **Probar todos los subarreglos posibles**. Es decir, todos los subarreglos $[i..j]$ con $0 \leq j < n$ e $0 \leq i \leq j$. Para cada uno de ellos calcular `suma`(i,j), la suma de los números en el subarreglo especificado y ver si es exactamente x.
   Complejidad :

Solución

1. **Probar todos los subarreglos posibles**. Es decir, todos los subarreglos $[i..j]$ con $0 \leq j < n$ e $0 \leq i \leq j$. Para cada uno de ellos calcular `suma`(i,j), la suma de los números en el subarreglo especificado y ver si es exactamente x.
   Complejidad : $\mathcal{O}(n^3)$ u $\mathcal{O}(n^2)$ dependiendo de la implementación de la función `suma`.

Solución

1. **Probar todos los subarreglos posibles**. Es decir, todos los subarreglos $[i..j]$ con $0 \leq j < n$ e $0 \leq i \leq j$. Para cada uno de ellos calcular `suma`(i,j), la suma de los números en el subarreglo especificado y ver si es exactamente x.
   Complejidad : $\mathcal{O}(n^3)$ u $\mathcal{O}(n^2)$ dependiendo de la implementación de la función `suma`.
2. Utilizar una **ventana deslizante** para resolver el problema. La idea es mantener **dos índices** que son los extremos de nuestra ventana deslizante, al extremo izquierdo lo llamaremos "i" y al derecho lo llamaremos "j". Ambos extremos comienzan en el principio del arreglo, y en todo momento vamos a **mantener** cuánto vale **la suma de los números dentro de la ventana [i..j)** (notar que no incluimos el extremo derecho).

Análisis

- ¿Cuántos pasos hace el algoritmo? Respuesta :

Análisis

- ¿Cuántos pasos hace el algoritmo? Respuesta : En cada paso, o bien **aumenta i** o **aumenta j**. Cada uno de ellos puede ser aumentado a lo sumo n veces. Por lo tanto al cabo de $2n$ **pasos** en el peor caso, finaliza nuestro algoritmo.

Análisis

- ¿Cuántos pasos hace el algoritmo? Respuesta : En cada paso, o bien **aumenta i** o **aumenta j**. Cada uno de ellos puede ser aumentado a lo sumo n veces. Por lo tanto al cabo de $2n$ **pasos** en el peor caso, finaliza nuestro algoritmo.
- ¿Por qué es importante que los números sean **positivos**? Respuesta :

Análisis

- ¿Cuántos pasos hace el algoritmo? Respuesta : En cada paso, o bien **aumenta i** o **aumenta j**. Cada uno de ellos puede ser aumentado a lo sumo n veces. Por lo tanto al cabo de $2n$ **pasos** en el peor caso, finaliza nuestro algoritmo.
- ¿Por qué es importante que los números sean **positivos**? Respuesta : Porque nos permite saber que si en algún momento $\texttt{suma} > x$, entonces todas las ventanas que tienen el mismo valor de i y un j mayor tendrán una suma mayor y podemos no mirarlas (de alguna forma, podemos afirmar que **"ya nos pasamos"**).

2SUM

Problema:

Dado un arreglo de n números, y un número x. Queremos encontrar dos números del arreglo que sumen x, o reportar que no existe tal par.

Ejemplos:

- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 12$ La respuesta es : "Sumando $A[0]$ y $A[5]$ obtenemos 12"
- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 4$ La respuesta es : "No hay dos elementos de A que sumen 4"

2SUM

Problema:

Dado un arreglo de n números, y un número x. Queremos encontrar dos números del arreglo que sumen x, o reportar que no existe tal par.

Ejemplos:

- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 12$ La respuesta es : "Sumando $A[0]$ y $A[5]$ obtenemos 12"
- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 4$ La respuesta es : "No hay dos elementos de A que sumen 4"

2SUM

Problema:

Dado un arreglo de n números, y un número x. Queremos encontrar dos números del arreglo que sumen x, o reportar que no existe tal par.

Ejemplos:

- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 12$ La respuesta es : "Sumando $A[0]$ y $A[5]$ obtenemos 12"
- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 4$ La respuesta es : "No hay dos elementos de A que sumen 4"

2SUM

Problema:

Dado un arreglo de n números, y un número x. Queremos encontrar dos números del arreglo que sumen x, o reportar que no existe tal par.

Ejemplos:

- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 12$ La respuesta es : "Sumando $A[0]$ y $A[5]$ obtenemos 12"
- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 4$ La respuesta es : "No hay dos elementos de A que sumen 4"

2SUM

Problema:

Dado un arreglo de n números, y un número x. Queremos encontrar dos números del arreglo que sumen x, o reportar que no existe tal par.

Ejemplos:

- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 12$ La respuesta es : "Sumando $A[0]$ y $A[5]$ obtenemos 12"
- $A = [7, 10, 4, 1, 9, 5, 9, 6]$. Buscamos sumar $x = 4$ La respuesta es : "No hay dos elementos de A que sumen 4"

Solución

Observación clave

Es importante notar que el orden de los números en el arreglo A no juega ningún rol. Entonces podemos asumir que tienen el orden que nos convenga. En particular, podemos asumir que el arreglo está **ordenado en forma creciente** (u ordenarlo en $\mathcal{O}(n \lg(n))$)

Solución

Observación clave

Es importante notar que el orden de los números en el arreglo A no juega ningún rol. Entonces podemos asumir que tienen el orden que nos convenga. En particular, podemos asumir que el arreglo está **ordenado en forma creciente** (u ordenarlo en $\mathcal{O}(n \lg(n))$)

1. Probar todos los pares de índices, y ver si esos dos números suman o no x. De existir un par que sí, reportarlo. Complejidad $\mathcal{O}(n^2)$

Solución

Observación clave

Es importante notar que el orden de los números en el arreglo A no juega ningún rol. Entonces podemos asumir que tienen el orden que nos convenga. En particular, podemos asumir que el arreglo está **ordenado en forma creciente** (u ordenarlo en $\mathcal{O}(n \lg(n))$)

1. Probar todos los pares de índices, y ver si esos dos números suman o no x. De existir un par que sí, reportarlo. Complejidad $\mathcal{O}(n^2)$
2. Utilizar una variante de la técnica que vimos recién para resolver el problema. En lugar de tener dos índices que siempre aumentan, ahora tendremos dos índices. **Uno siempre aumenta, el otro siempre disminuye**. Por lo tanto también haremos **a lo sumo 2n pasos**. Complejidad $\mathcal{O}(n)$.

Solución

Observación clave

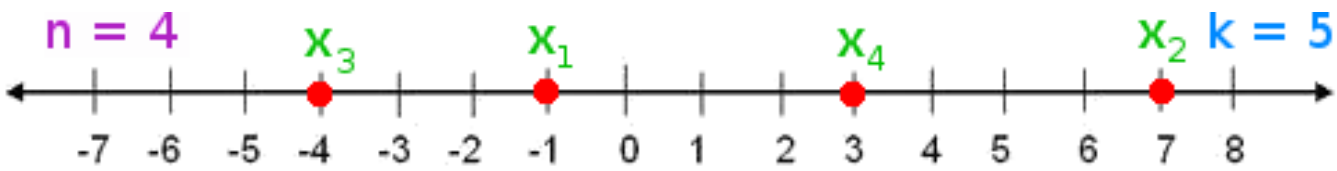
Es importante notar que el orden de los números en el arreglo A no juega ningún rol. Entonces podemos asumir que tienen el orden que nos convenga. En particular, podemos asumir que el arreglo está **ordenado en forma creciente** (u ordenarlo en $\mathcal{O}(n \lg(n))$)

1. Probar todos los pares de índices, y ver si esos dos números suman o no x. De existir un par que sí, reportarlo. Complejidad $\mathcal{O}(n^2)$
2. Utilizar una variante de la técnica que vimos recién para resolver el problema. En lugar de tener dos índices que siempre aumentan, ahora tendremos dos índices. **Uno siempre aumenta, el otro siempre disminuye**. Por lo tanto también haremos **a lo sumo 2n pasos**. Complejidad $\mathcal{O}(n)$.
3. Búsqueda binaria, hashmap, ...

K-ésima distancia entre puntos de la recta

Problema:

Dados n puntos en la recta numérica $(x_1, x_2, \ldots, x_n)$, y un número k. Si listamos todas las distancias entre pares de puntos (x_i, x_j) con $i < j$ y ordenamos estas distancias (puede haber repetidas). Queremos saber qué distancia queda en el k-ésimo lugar.



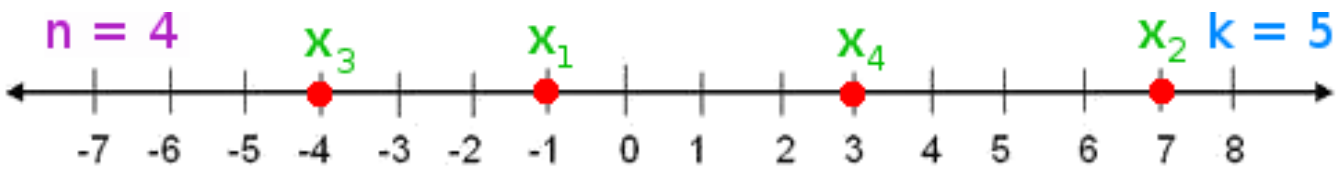
1. $\text{dist}(x_1, x_2) = 8$
2. $\text{dist}(x_1, x_3) = 3$
3. $\text{dist}(x_1, x_4) = 4$
4. $\text{dist}(x_2, x_3) = 11$
5. $\text{dist}(x_2, x_4) = 4$
6. $\text{dist}(x_3, x_4) = 7$

$$D = [3, 4, 4, 7, 8, 11]$$

K-ésima distancia entre puntos de la recta

Problema:

Dados n puntos en la recta numérica $(x_1, x_2, \ldots, x_n)$, y un número k. Si listamos todas las distancias entre pares de puntos (x_i, x_j) con $i < j$ y ordenamos estas distancias (puede haber repetidas). Queremos saber qué distancia queda en el k-ésimo lugar.



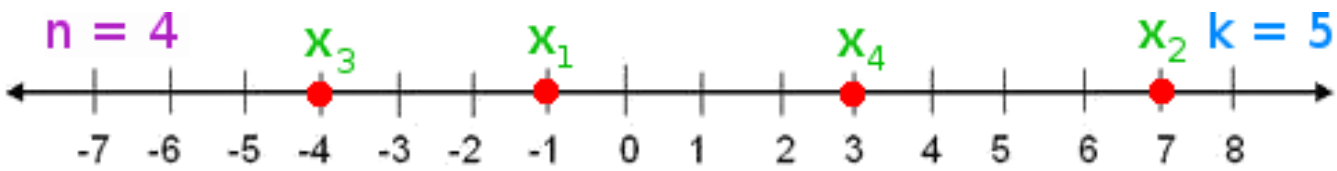
1. $\text{dist}(x_1, x_2) = 8$
2. $\text{dist}(x_1, x_3) = 3$
3. $\text{dist}(x_1, x_4) = 4$
4. $\text{dist}(x_2, x_3) = 11$
5. $\text{dist}(x_2, x_4) = 4$
6. $\text{dist}(x_3, x_4) = 7$

$$D = [3, 4, 4, 7, 8, 11]$$

K-ésima distancia entre puntos de la recta

Problema:

Dados n puntos en la recta numérica $(x_1, x_2, \ldots, x_n)$, y un número k. Si listamos todas las distancias entre pares de puntos (x_i, x_j) con $i < j$ y ordenamos estas distancias (puede haber repetidas). Queremos saber qué distancia queda en el k-ésimo lugar.



1. $\text{dist}(x_1, x_2) = 8$
2. $\text{dist}(x_1, x_3) = 3$
3. $\text{dist}(x_1, x_4) = 4$
4. $\text{dist}(x_2, x_3) = 11$
5. $\text{dist}(x_2, x_4) = 4$
6. $\text{dist}(x_3, x_4) = 7$

$$D = [3, 4, 4, 7, 8, 11]$$

Solución

1. Calcular todas las distancias, ordenarlas, y ver qué elemento está en el k-ésimo lugar. Complejidad : $\mathcal{O}(n^2 + n^2 \lg(n^2))$

Solución

1. Calcular todas las distancias, ordenarlas, y ver qué elemento está en el k-ésimo lugar. Complejidad : $\mathcal{O}(n^2 + n^2 \lg(n^2))$
2. Usar **Binary Search** y una **Ventana deslizante**.

Solución

1. Calcular todas las distancias, ordenarlas, y ver qué elemento está en el k-ésimo lugar. Complejidad : $\mathcal{O}(n^2 + n^2 \lg(n^2))$
2. Usar **Binary Search** y una **Ventana deslizante**. Haremos búsqueda binaria en la distancia. Formalmente, la propiedad será: $P(d)$: La cantidad de distancias menores que **d** es menor que k

Solución

1. Calcular todas las distancias, ordenarlas, y ver qué elemento está en el k-ésimo lugar. Complejidad : $\mathcal{O}(n^2 + n^2 \lg(n^2))$
2. Usar **Binary Search** y una **Ventana deslizante**. Haremos búsqueda binaria en la distancia. Formalmente, la propiedad será: $P(d)$: La cantidad de distancias menores que **d** es menor que k
   Ordenados los puntos, para una **distancia** d **fija**, podemos ir moviendo una ventana deslizante que **para cada extremo izquierdo nos diga cuántos puntos están a distancia menor que** d .

Solución

1. Calcular todas las distancias, ordenarlas, y ver qué elemento está en el k-ésimo lugar. Complejidad : $\mathcal{O}(n^2 + n^2 \lg(n^2))$
2. Usar **Binary Search** y una **Ventana deslizante**. Haremos búsqueda binaria en la distancia. Formalmente, la propiedad será:
   $P(d)$: La cantidad de distancias menores que **d** es menor que k
   Ordenados los puntos, para una **distancia** d **fija**, podemos ir moviendo una ventana deslizante que **para cada extremo izquierdo nos diga cuántos puntos están a distancia menor que** d . Complejidad : $\mathcal{O}(\underbrace{n \lg(n)}_{\text{Ordenar}} + \underbrace{\lg(n)}_{\text{BB}} \cdot \underbrace{n}_{\text{VD}}) = \mathcal{O}(n \lg(n))$

Tarea

- http://codeforces.com/problemset/problem/760/B
- https://icpcarchive.ecs.baylor.edu/external/44/4478.pdf
- https://www.spoj.com/problems/TAP2015B/
- http://www.spoj.com/problems/DINOSM/