Binary Search

Universidad Tecnológica Nacional

Taller de Programación Competitiva

Contenidos

Contenidos

Buscando elemento en lista ordenada - Enunciado

Problema motivador

Dada una lista **ordenada** A de N números queremos saber si el elemento X se encuentra y en que posición. Si aparece más de una vez, buscaremos la primera aparición. Si no aparece devolvemos -1. Las posiciones son indexadas en 0.

A[i]	3	5	7	12	22	22	37	40
i	0	1	2	3	4	5	6	7

- $X = 7$, la respuesta es 2
- $X = 22$, la respuesta es 4
- $X = 50$, la respuesta es -1
- $X = 1$, la respuesta es -1
- $X = 15$, la respuesta es -1

Contenidos

Primer intento - Búsqueda lineal

Un primer approach puede ser buscar elemento por elemento:

```
1 def linear_search(A, X):
2     N = len(A)
3     for i in range(N):
4         if A[i] == X
5             return i
6     return -1
```

- Complejidad espacial:

Primer intento - Búsqueda lineal

Un primer approach puede ser buscar elemento por elemento:

```
1 def linear_search(A, X):
2     N = len(A)
3     for i in range(N):
4         if A[i] == X
5             return i
6     return -1
```

- Complejidad espacial: $O(1)$

Primer intento - Búsqueda lineal

Un primer approach puede ser buscar elemento por elemento:

```
1 def linear_search(A, X):
2     N = len(A)
3     for i in range(N):
4         if A[i] == X
5             return i
6     return -1
```

- Complejidad espacial: $O(1)$
- Complejidad temporal: $O(N)$

Contenidos

Refinando la idea

Resolvamos el problema para $X = 7$;

A[i]	3	5	7	12	22	22	37	40
i	0	1	2	3	4	5	6	7

- En vez de mirar todos los números veamos $A[4] = 12$.
- Como la lista está ordenada podemos descartar los índices mayores ($A[4:]$).
- Podemos repetir este proceso, mirando siempre el elemento central y quedándonos con la mitad izquierda o derecha.
- Elegimos mirar el elemento central ya que nos garantiza que sea cual sea la mitad descartada siempre reduciremos el espacio de búsqueda a la mitad.

Contenidos

Segundo intento - Búsqueda binaria

A[i]	$-\infty$	3	5	7	12	22	22	37	40	∞
i	-1	0	1	2	3	4	5	6	7	8

```
1  def binary_search(A, X):
2      N = len(A)
3      a = -1
4      b = N
5      while b - a > 1:
6          c = (a+b)//2
7          if A[c] >= X:
8              b = c
9          else:
10             a = c
11     if b < N and A[b] == X:
12         return b
13     return -1
```

- Complejidad espacial:

Segundo intento - Búsqueda binaria

A[i]	$-\infty$	3	5	7	12	22	22	37	40	∞
i	-1	0	1	2	3	4	5	6	7	8

```
1  def binary_search(A, X):
2      N = len(A)
3      a = -1
4      b = N
5      while b - a > 1:
6          c = (a+b)//2
7          if A[c] >= X:
8              b = c
9          else:
10             a = c
11     if b < N and A[b] == X:
12         return b
13     return -1
```

- Complejidad espacial: $O(1)$

Segundo intento - Búsqueda binaria

A[i]	$-\infty$	3	5	7	12	22	22	37	40	∞
i	-1	0	1	2	3	4	5	6	7	8

```
1  def binary_search(A, X):
2      N = len(A)
3      a = -1
4      b = N
5      while b - a > 1:
6          c = (a+b)//2
7          if A[c] >= X:
8              b = c
9          else:
10             a = c
11     if b < N and A[b] == X:
12         return b
13     return -1
```

- Complejidad espacial: $O(1)$
- Complejidad temporal: $O(logN)$

Búsqueda binaria - Detalles de implementacion

```
1  def binary_search(A, X):
2      N = len(A)
3      a = -1
4      b = N
5      while b - a > 1:
6          c = ( a + b )//2
7          if A[c] >= X:
8              b = c
9          else:
10             a = c
11     if b < N and A[b] == X:
12         return b
13     return -1
```

- $A[a]$ es siempre menor que X
- $A[b]$ es siempre mayor que X
- El bucle termina cuando $[a, b]$ son consecutivos.
- Si el bucle no termina $[a, b]$ tiene al menos 3 elementos. Luego c es siempre distinto de a y b. No necesitamos preocuparnos, c nunca valdrá -1 ni N

Contenidos

Refinando la idea

Dado un predicado monótono sobre los enteros $P(i)$ queremos hallar el índice del **último 0 y el primer 1**.

A[i]	0	0	0	1	1	1	1	1
i	0	1	2	3	4	5	6	7

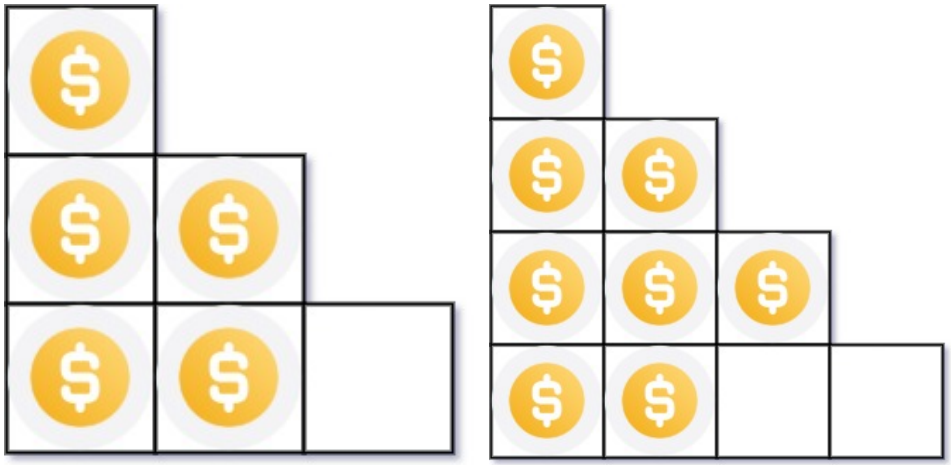
- Un **predicado** es simplemente una función que devuelve $\{0, 1\}$ (verdadero o falso).
- $P : [0, N) \to \{0, 1\}$
- P es monótono significa que $P(i) < P(i + 1)$.
- En criollo, P es falso hasta cierto punto donde comienza a ser verdadero hasta el final.

Contenidos

Buscando elemento en lista ordenada - Enunciado

Problema motivador

Tenemos N ($N \leq 2^{31}$) monedas y teemos que construir una escalera con esas monedas. El primer escalón tiene 1 moneda y cada escalón tiene 1 moneda más que el anterior.
Cual es la cantidad máximas de escalones completos que podemos contruir?



- $N = 5$, respuesta es 2.
- $N = 8$, respuesta es 3.

Fuerza Bruta

```
1 def number_of_staors(N):
2     for k in range(1,N):
3         if k*(k+1)//2 > N:
4             return k-1
5     return -1
```

- Complejidad espacial:

Fuerza Bruta

```
1 def number_of_staors(N):
2     for k in range(1,N):
3         if k*(k+1)//2 > N:
4             return k-1
5     return -1
```

- Complejidad espacial: $O(\sqrt{N})$

Pensando mejor

- La formula $k * (k + 1)//2 > N$ es un predicado $P(k)$
- Es monótono! Para probar que es monótono basta probar alguna de las siguientes:
  - $P(k) \rightarrow P(K + 1)$
  - $\neg P(K) \rightarrow \neg P(K - 1)$ (contrarrecíproco)
- Podemos buscar el último 0 con binary search!