CURSO DE PROGRAMACIÓN COMPETITIVA

ESTRUCTURAS DE DATOS Y COMPLEJIDAD

CURSO DE PROGRAMACIÓN COMPETITIVA URJC - 2026

Organizadores:

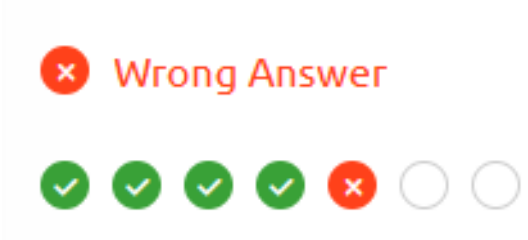
- Isaac Lozano (isaac.lozano@urjc.es)
- **Sergio Salazar** (sergio.salazar@urjc.es)
- Adaya Ruiz (am.ruiz.2020@alumnos.urjc.es)
- Olga Somalo (o.somalo.2021@alumnos.urjc.es)
- **Lucas Martín** (lucas.martin@urjc.es)
- Iván Penedo (ivan.penedo@urjc.es)
- Alicia Pina (alicia.pina@urjc.es)
- Sara García (sara.garciar@urjc.es)
- Raúl Fauste (raul.fauste@urjc.es)

Complejidad

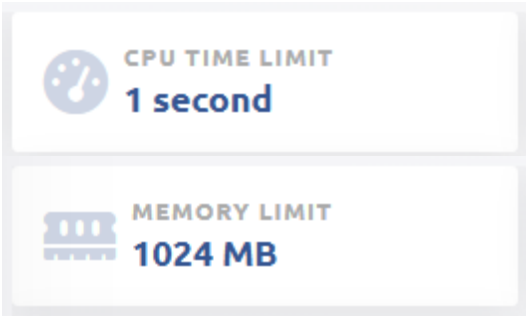
Complejidad

¿Cómo obtenemos un **ACCEPTED** en un problema de programación competitiva?

- El algoritmo debe ser **CORRECTO**
  - Para **todas** las entradas contempladas por los jueces el algoritmo debe dar la salida esperada.



- El algoritmo debe ser **EFICIENTE**
  - El tiempo de ejecución para **todas** entrada debe ser menor al tiempo límite establecido.



Complejidad

¿Cuánto va a tardar mi código?

- Vamos a medir cuántas operaciones hace el programa en **relación al tamaño de la entrada** (n).
- Búscamos una idea rápida e intuitiva
- Siempre consideraremos el **peor caso posible!**

Complejidad

Las operaciones **básicas** se consideran iguales. Todas ellas “cuestan” 1 iteración.

- `1+1, a+b, a*b, a/b,`
- `a or b, a and b, a xor b, …`
- `print(“hola”)`
- `var = 3`
- `if(True)`
- `…`

O(1)

Complejidad

Concatenar un conjunto **FIJO** de operaciones básicas se va a considerar una sola iteración…

- Pero estamos haciendo 3 operaciones… ¿No debería ser O(3)?
- Si… ¡Pero ese valor no depende de la entrada! Por lo tanto, no cambiará nada si el número que entra es muy grande…

```
if __name__ == '__main__':
    n = input()
    a = True
    b = False
    c = True
    print(a or b and c)
```

Complejidad

Concatenar un conjunto de operaciones que depende de la entrada **NO** es una iteración >>> **BUCLES**

- ¡El número de operaciones comienza a depender de la entrada!
- O(1*n) = **O(n)**

```
if __name__ == '__main__':
    n = input()
    for i in range(n):
        print(i)
```

Complejidad

Concatenar un conjunto de operaciones que depende de la entrada **NO** es una iteración >>> **BUCLES**

- ¡Da igual que haya varias operaciones en el cuerpo del bucle!
- O(10*n) = **O(n)**

```
if __name__ == '__main__':
    n = input()
    for i in range(n):
        a = i*2
        b = a + 3
        print(b-a)
        a = b*a
        print(a*b)
```

Complejidad

¿Y si nos saltamos iteraciones?

- Hacemos la mitad de operaciones… O(n/2)
- OJO! Ya hemos visto que no nos importan las constantes: O(n/2) = O(n)

```
if __name__ == '__main__':
    n = input()
    for i in range(0,n,2):
        print(i)
```

- Esto hace que `i` vaya de dos en dos hasta `n`.

Complejidad

Concatenar un conjunto de bucles **IMPORTA.**

- Ambos bucles dependen de la entrada del problema!
- Se hacen n iteraciones de un bucle que hace n operaciones: O(n*n) = **O(n^2)**

```
if __name__ == '__main__':
    n = input()
    for i in range(n):
        for j in range(n):
            a = i*2
            b = j + 3
            print(b-a)
            a = b*a
            print(a*b)
```

Complejidad

¿Y si no están concatenados?

- Aunque ambos bucles dependen de n, es como si en el cuerpo del bucle aparecieran ambas operaciones…
- O(n+n) = O(2*n) = **O(n)**

```
if __name__ == '__main__':
    n = input()
    for i in range(n):
        print(i)
    for i in range(n):
        print(2*i)
```

Complejidad

¿Y si no están concatenados, con cuál me quedo?

- Ahora los bucles tienen complejidades distintas…
- Nos quedamos con el **PEOR** caso!
- O(n^2+n) = **O(n^2)**

```
if __name__ == '__main__':
    n = int(input())
    for i in range(n):
        print(n)
    for i in range(n):
        for j in range(n):
            print(i*j)
```

Complejidad

¿Y si la cantidad de operaciones cambia de forma dinámica durante el bucle? >>> **RECURSIÓN**

- Por cada iteración se harán dos operaciones `solve`, que llamarán a dos operaciones `solve`, que llamarán a otras dos operaciones `solve`…

- O(2*2*2*2…) = **O(2^n)**

  n

```
def solve(n):
    if(n==0):
        print("HOLA")
    else:
        solve(n-1)
        solve(n-1)
```

Complejidad

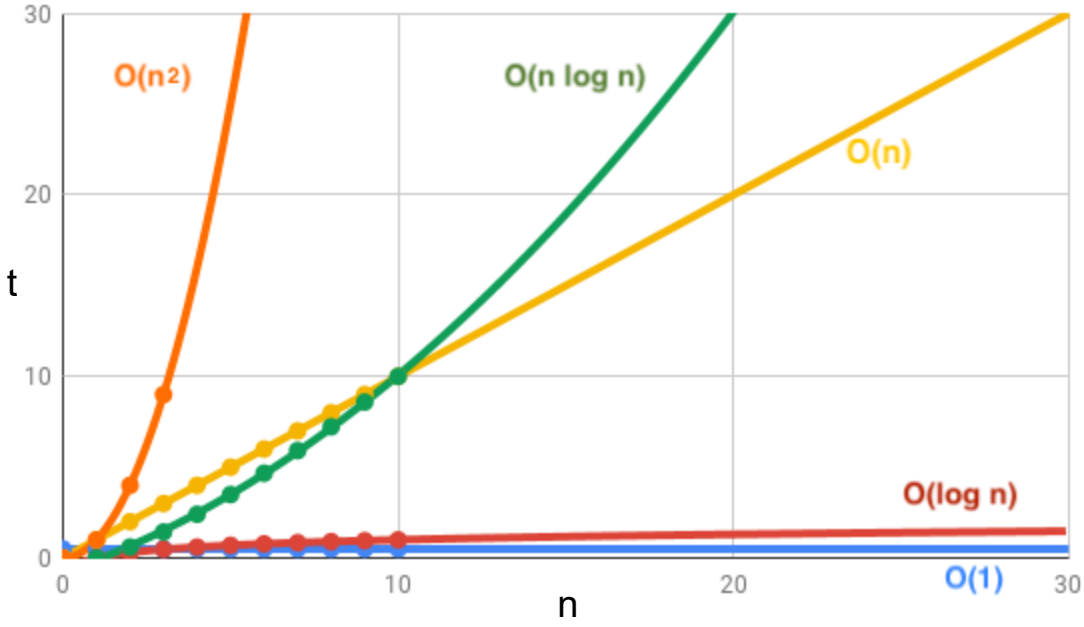
¿Y si la cantidad de operaciones cambia de forma dinámica durante el bucle, pero **REDUCIENDOSE**?

- Por cada iteración el tamaño de la entrada se divide a la mitad.
- Es la operación contraria a la exponenciación: O(log n)

```
def solve(n):
    if(n<1):
        print(n)
    else:
        solve(n/2)
```

Complejidad

El término de complejidad funciona cuando los tamaños son grandes, no nos importa que pasa cuando la entrada es pequeña, siempre se resolverá rápido…



Complejidad

¿Y esto para qué?

Complejidad	Operaciones por segundo
O(1)	:D
O(log n)	:)
O(n)	1,000,000
O(n * log n)	100,000
O(n^2)	1000 - 3000
O(n^3)	100 - 300
O(2^n)	20
O(n!)	12

- Todos los problemas te indican el tamaño de la entrada
- Podemos aproximar si nuestro algoritmo dará **TLE** antes de enviarlo.

Complejidad

¿Y esto para qué?

Complejidad	Operaciones por segundo
O(1)	:D
O(log n)	:)
O(n)	1,000,000
O(n * log n)	100,000
O(n^2)	1000 - 3000
O(n^3)	100 - 300
O(2^n)	20
O(n!)	12

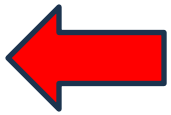
The length of the input string is at least 2 and at most 100 000 characters.

CPU TIME LIMIT
3 seconds

Complejidad

¿Y esto para qué?

Complejidad	Operaciones por segundo
O(1)	:D
O(log n)	:)
O(n)	1,000,000
O(n * log n)	100,000
O(n^2)	1000 - 3000
O(n^3)	100 - 300
O(2^n)	20
O(n!)	12



The length of the input string is at least 2 and at most 100 000 characters.

Complejidad

¿Y esto para qué?

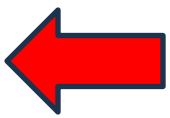
Complejidad	Operaciones por segundo
O(1)	:D
O(log n)	:)
O(n)	1,000,000
O(n * log n)	100,000
O(n^2)	1000 - 3000
O(n^3)	100 - 300
O(2^n)	20
O(n!)	12

input is an integer n $(2 \leq n \leq 100)$

Complejidad

¿Y esto para qué?

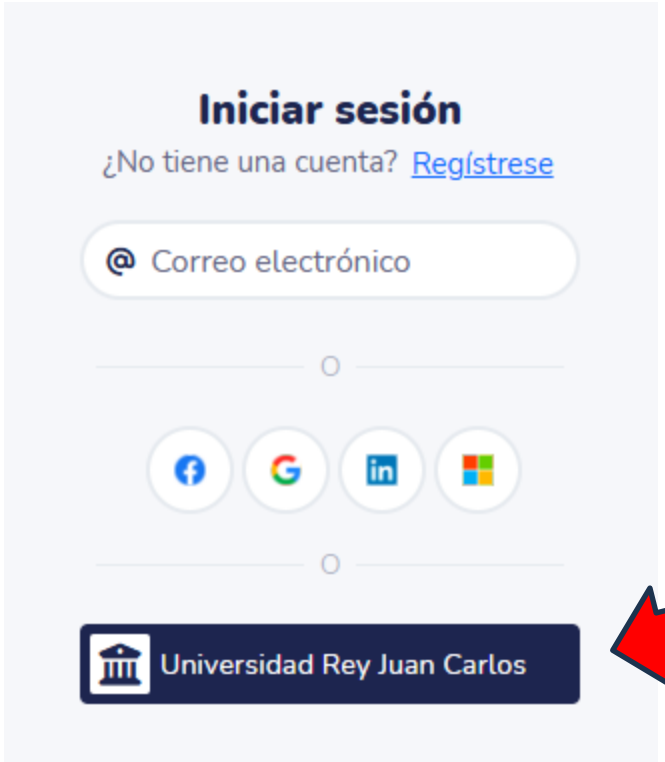
Complejidad	Operaciones por segundo
O(1)	:D
O(log n)	:)
O(n)	1,000,000
O(n * log n)	100,000
O(n^2)	1000 - 3000
O(n^3)	100 - 300
O(2^n)	20
O(n!)	12

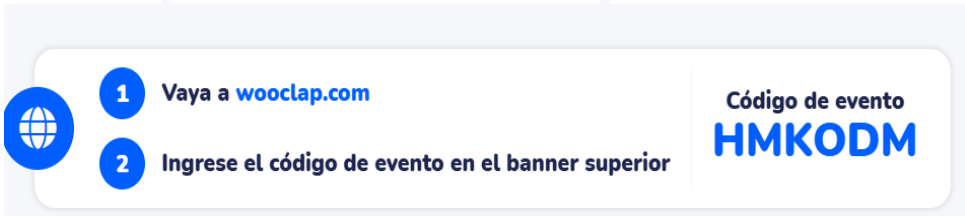


input is an integer n $(2 \leq n \leq 100)$

Complejidad

app.wooclap.com/HMKODM





ESTE ES EL MÉTODO DE ASISTENCIA DE HOY!!!

Estructuras de Datos 1

Estructuras de Datos Principales

- Listas
- Arrays
- Strings
- Pilas
- Colas
- Set

LISTAS

- **Estructura de datos dinámica** que permite almacenar una **secuencia de elementos**.
- Puede contener distintos tipos de datos. ¡Python no te avisa!

```
1   # Definición de una lista
2   mi_lista = [1, 2, 3, 4, 5]
3
4   print(mi_lista)  # Salida: [1, 2, 3, 4, 5]
5
```

LISTAS

- Bloque contiguo de memoria.
- **Redimensión** Automática.
- Se **dobla el tamaño** según necesidad **copiando** los elementos al nuevo bloque más grande (O(n)).

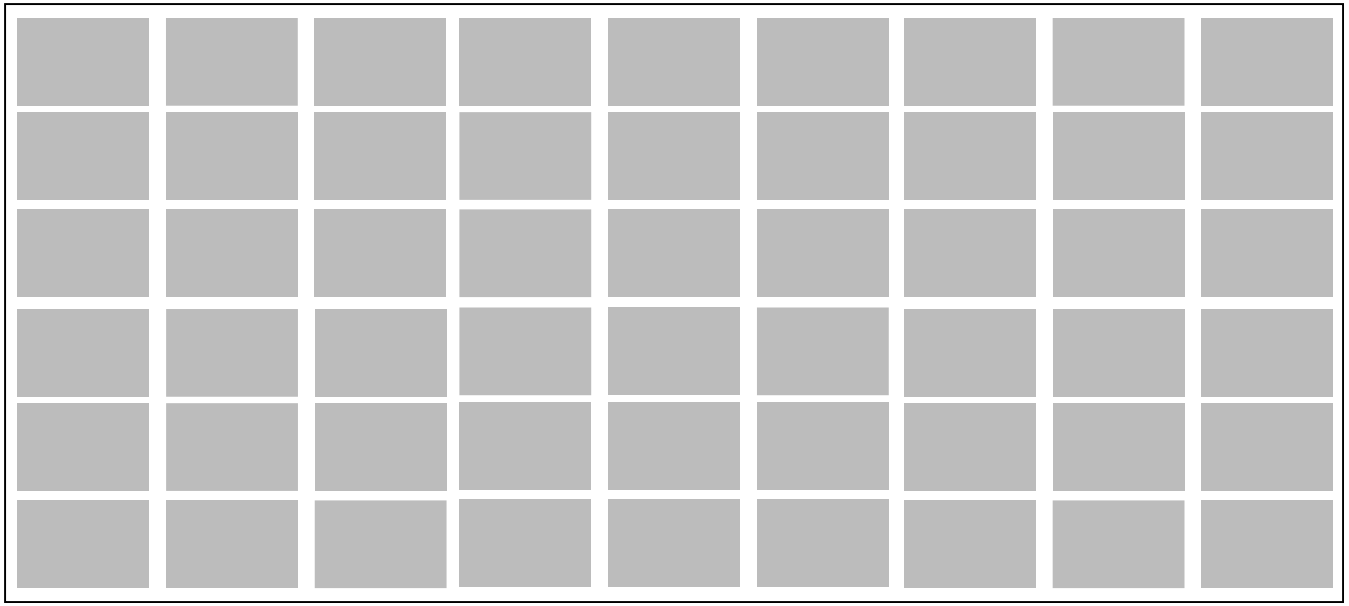
- **Complejidad**:
  - + Buscar un elemento
  - + Eliminar un elemento
  - + Añadir un elemento
  - + Modificar un elemento

LISTAS

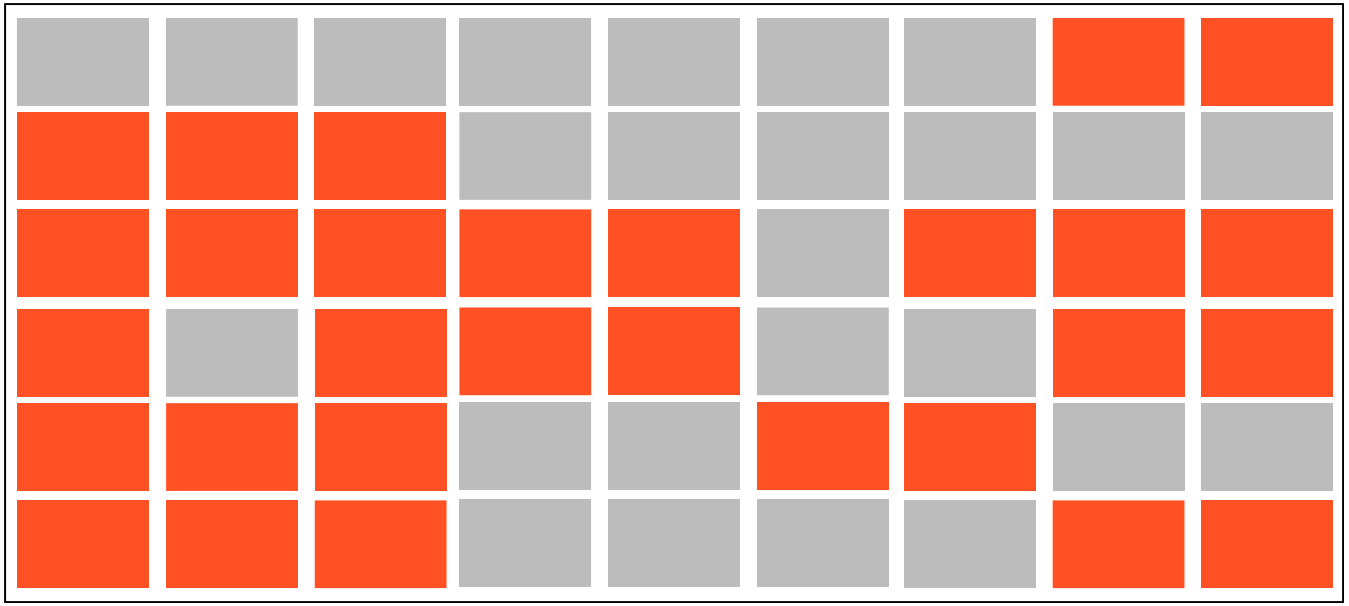
- Búsqueda
  - Mediante índice: O(1)
  - Mediante Valor: **O(n)**
- Eliminación **O(n)**
- Añadir al final O(1)
- Insertar en medio **O(n)**
- Modificar O(1)

```
 6  mi_lista[3] #Índice
 7
 8  if valor in mi_lista: #Valor
 9      print("Encontrado")
10
11
12  mi_lista.pop(indice) #Índice
13  mi_lista.remove(valor) #Valor
14
15  mi_lista.append(valor)
16  mi_lista.insert(indice, valor)
17
18
19  mi_lista[indice] = valor
```

Funcionamiento Interno

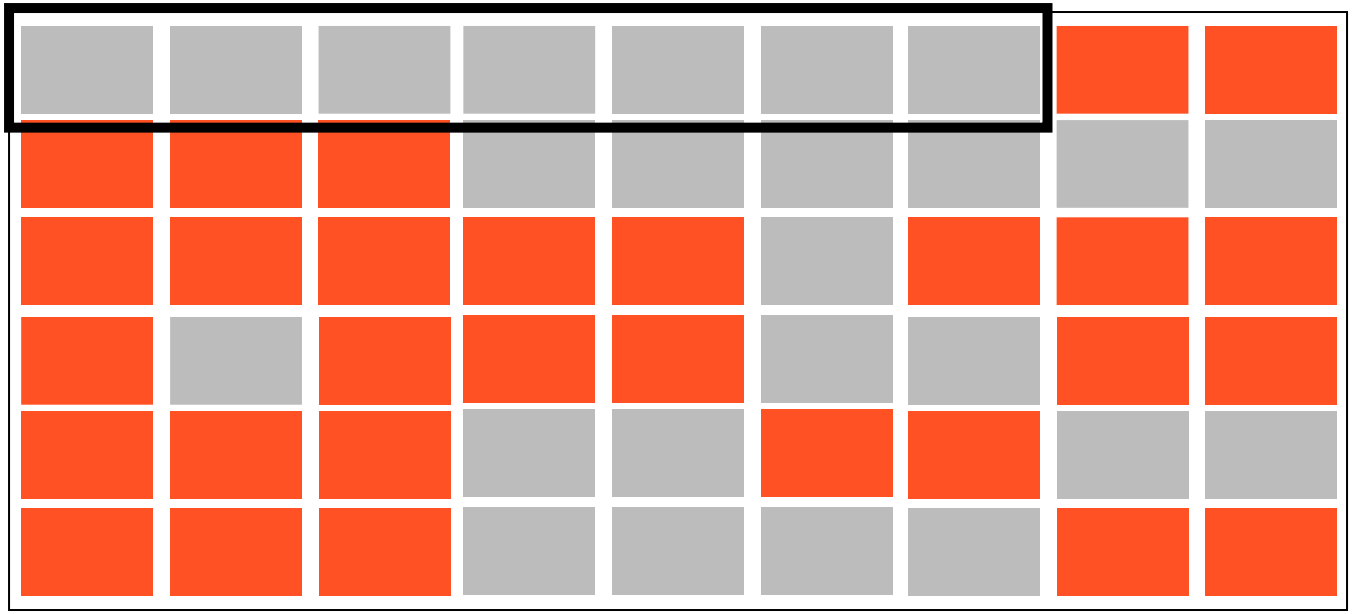


Funcionamiento Interno



Funcionamiento Interno

Listas

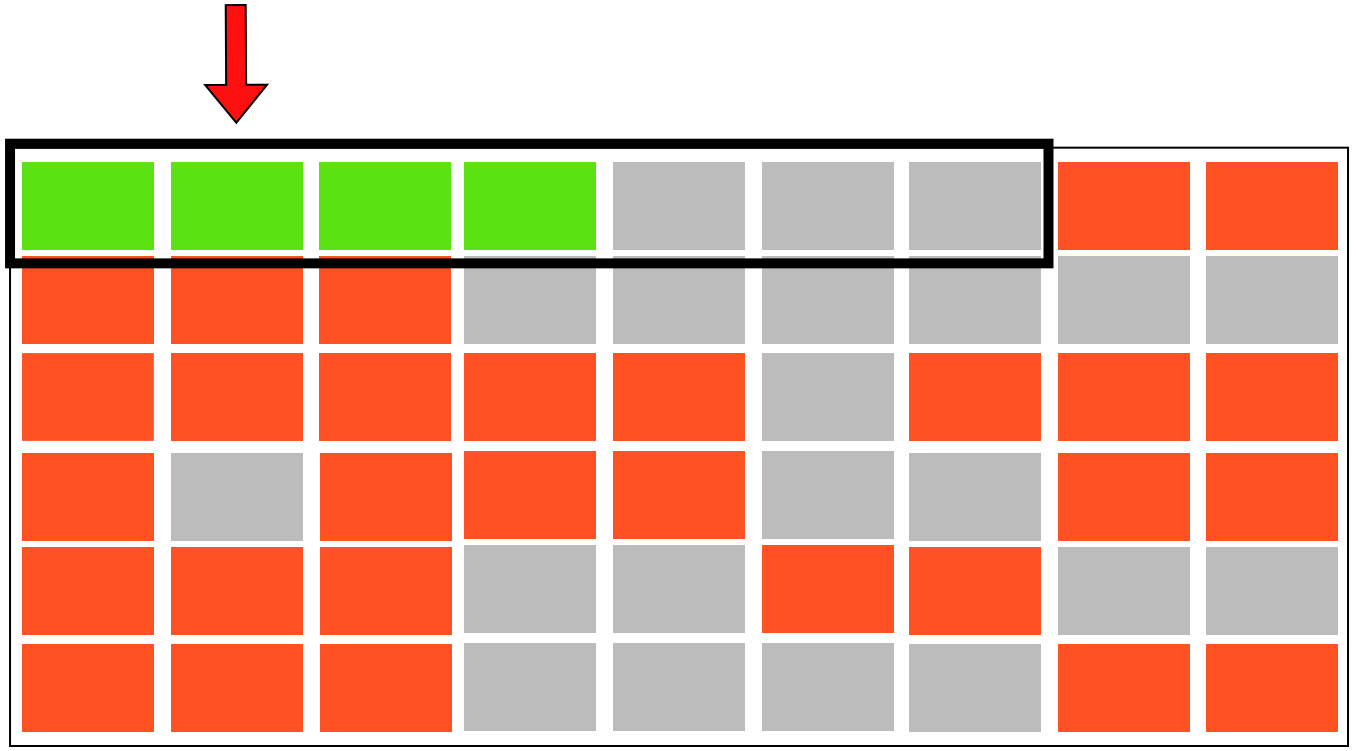


Funcionamiento Interno

Listas

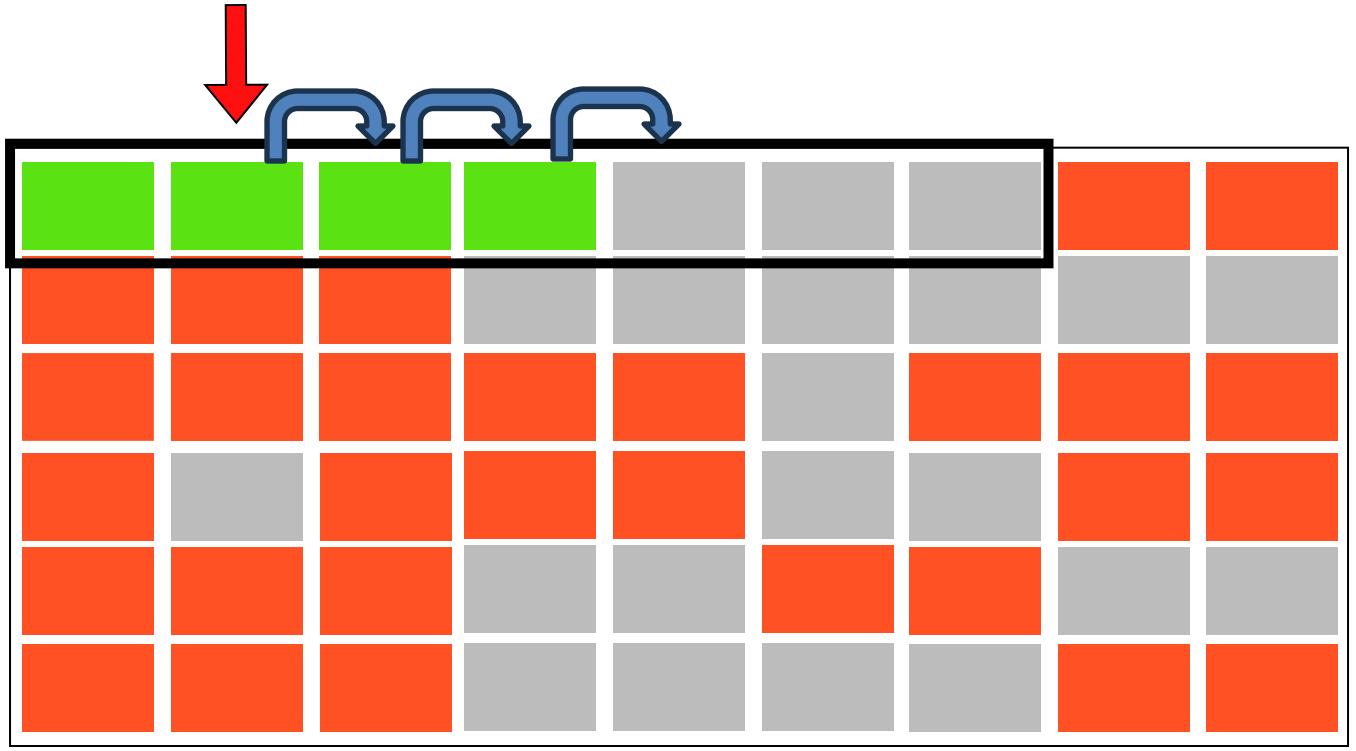
Funcionamiento Interno

Listas: Inserción



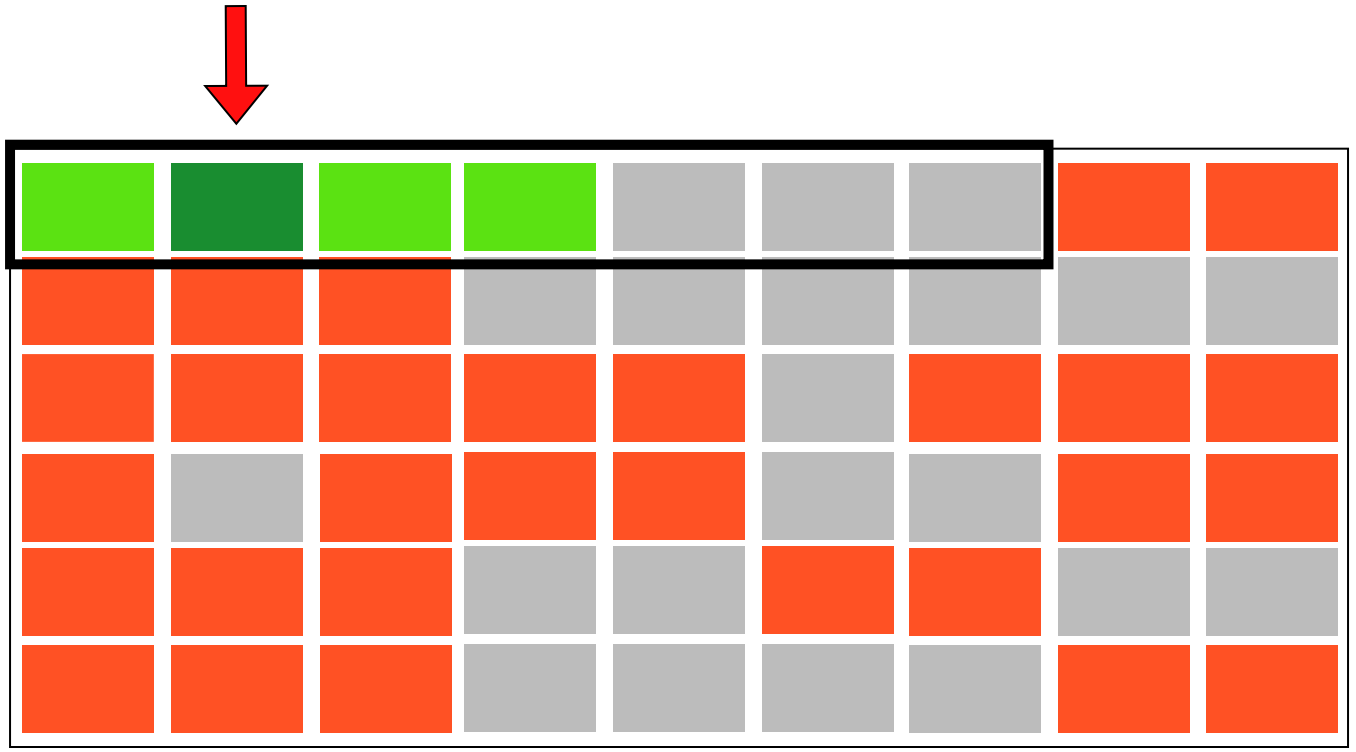
Funcionamiento Interno

Listas: Inserción **O(n)**



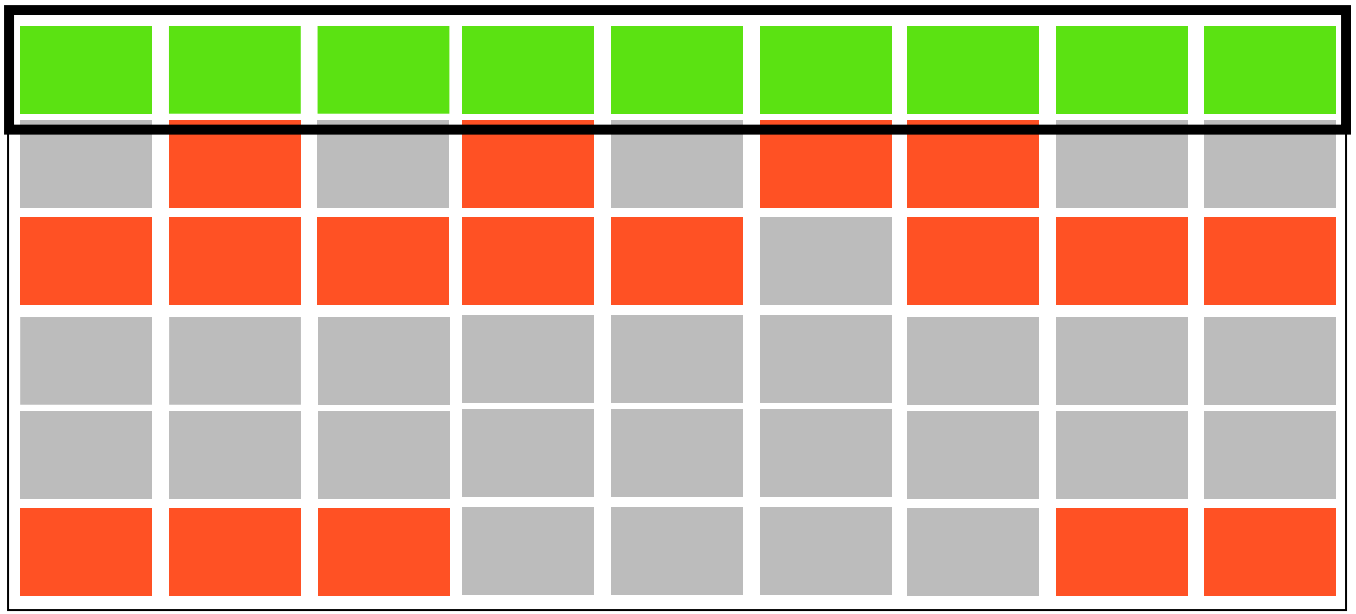
Funcionamiento Interno

Listas: Modificación **O(1)**



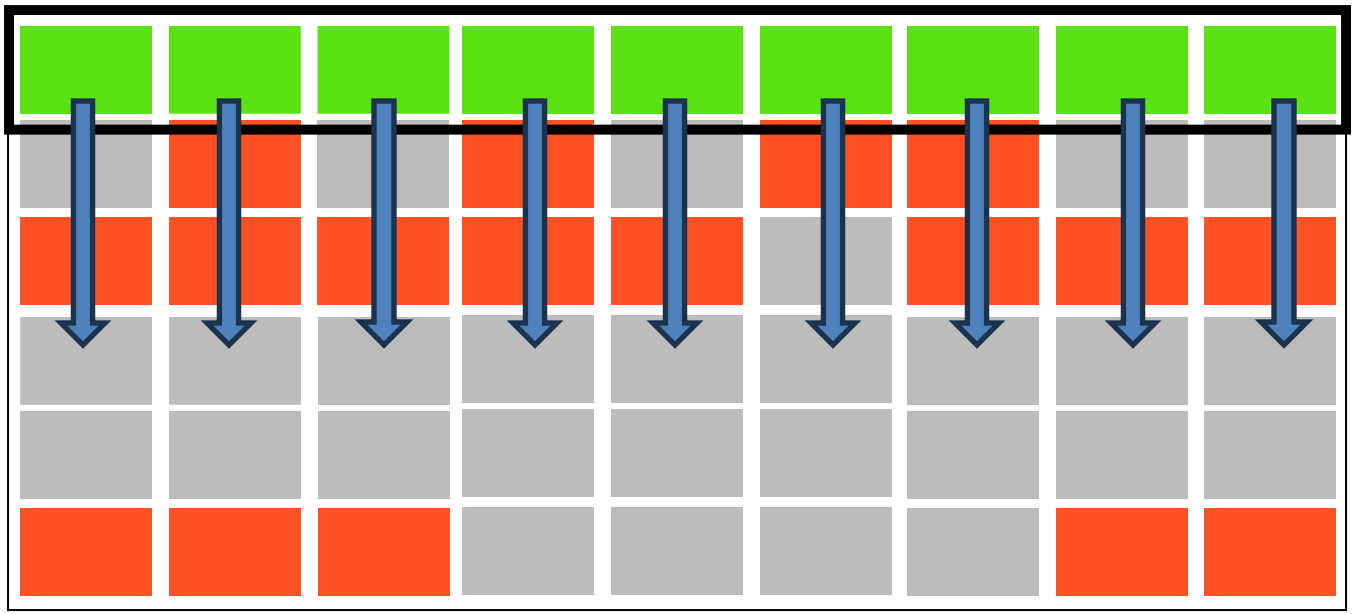
Funcionamiento Interno

Listas: Redimensionado **O(n)**



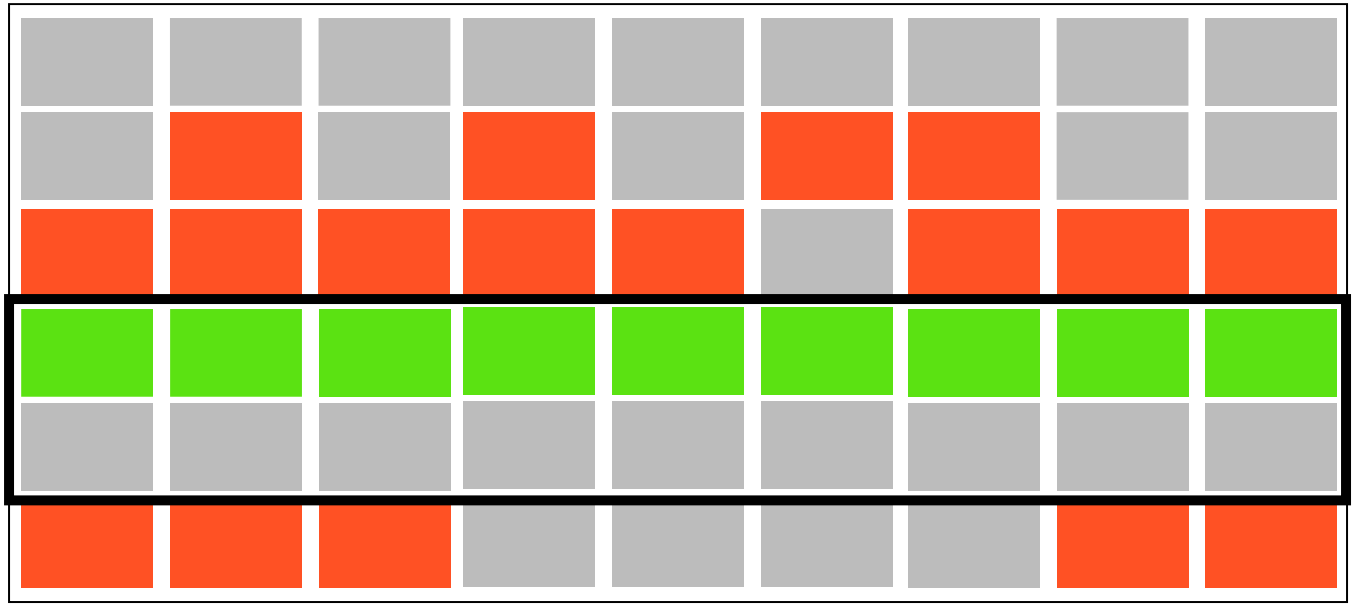
Funcionamiento Interno

Listas: Redimensionado **O(n)**



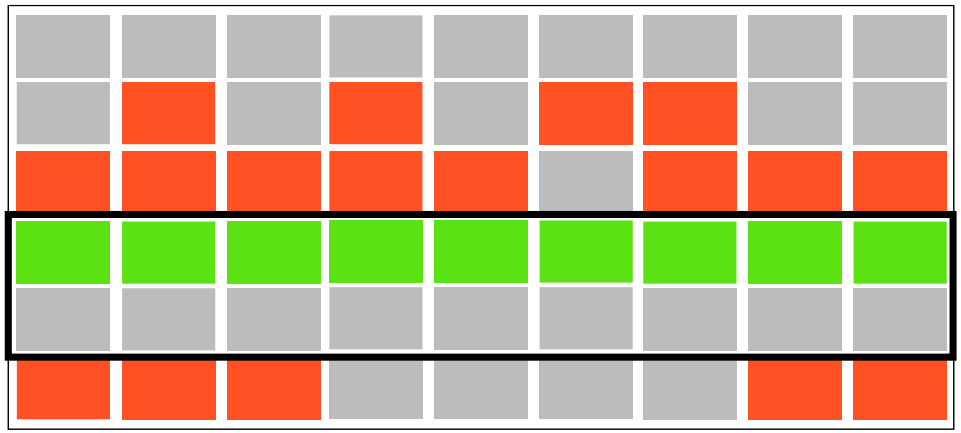
Funcionamiento Interno

Listas: Redimensionado **O(n)**



LISTAS

- Búsqueda
  - Mediante índice: O(1)
  - Mediante Valor: **O(n)**
- Eliminación **O(n)**
- Añadir al final O(1)
- Insertar en medio **O(n)**
- Modificar O(1)



ARRAYS

- En Python no existen Arrays como en **C** o **C++**, se usan **listas**.

STRINGS

- Listas inmutables de caracteres.
- Mismo acceso que listas
- Concatenación **O(n)**

```
mi_string = "hola"

elemento_string = mi_string[3]

concatenado_str = mi_string + " mundo"


mi_string.upper()    # Mayúsculas
mi_string.lower()    # Minúsculas
mi_string.replace("o", "O")
mi_string.split(" ")  # Dividir
```

STRINGS

- Modificar strings mejor con listas.

```
mi_lista = list(mi_string)
mi_lista.append("!")    # Añadir
mi_lista.pop()          # Eliminar último
mi_lista.insert(1, "E")  # Insertar
mi_lista.remove("o")    # Eliminar valor
mi_lista.reverse()      # Invertir
mi_lista.sort()         # Ordenar
```

Apaxiaaaaaaaaaaaans!

- https://open.kattis.com/problems/apaxiaaans
- ID: apaxiaans
- Ejemplos:

rooobert → robert

Roooooobertapalaxxxxios → robertapalaxios

PILAS (STACKS)

- Estructura **LIFO** (Last In, First Out).
- **collections.deque :**
  - Lista doblemente **enlazada**
  - Más eficiente que **list.**

- Aplicaciones:
  - Backtracking (DFS).
  - Paréntesis balanceados, evaluación de expresiones.
  - Historial de deshacer/rehacer en editores.

PILAS (STACKS)

- Estructura **LIFO** (Last In, First Out).
- **collections.deque :**
  - Lista doblemente **enlazada**
  - Más eficiente que **list.**

```
2  from collections import deque
3
4  pila = deque()
5
```

- Aplicaciones:
  - Backtracking (DFS).
  - Paréntesis balanceados, evaluación de expresiones.
  - Historial de deshacer/rehacer en editores.

PILAS (STACKS)

- Complejidad Operaciones:
- Apilar: **O(1)**
- Desapilar (Pop): **O(1)**
- Ver cima (Top): **O(1)**
- Comprobar vacía: **O(1)**

```
2   from collections import deque
3
4   pila = deque()
5
6   pila.append(10) # Apilar
7   pila.pop()      # Desapilar
8
9   top = pila[-1]
10  esta_vacia = len(pila) == 0
```

COLAS (QUEUES)

- Estructura **FIFO** (First In, First Out).

- **collections.deque :**
  - Doble **lista enlazada**
  - Más eficiente que **list.**

- Aplicaciones:
  - BFS en grafos.
  - Orden de llegada.
  - Colas de impresión, eventos, supermercados.

```
2  from collections import deque
3
4  cola = deque()
5
```

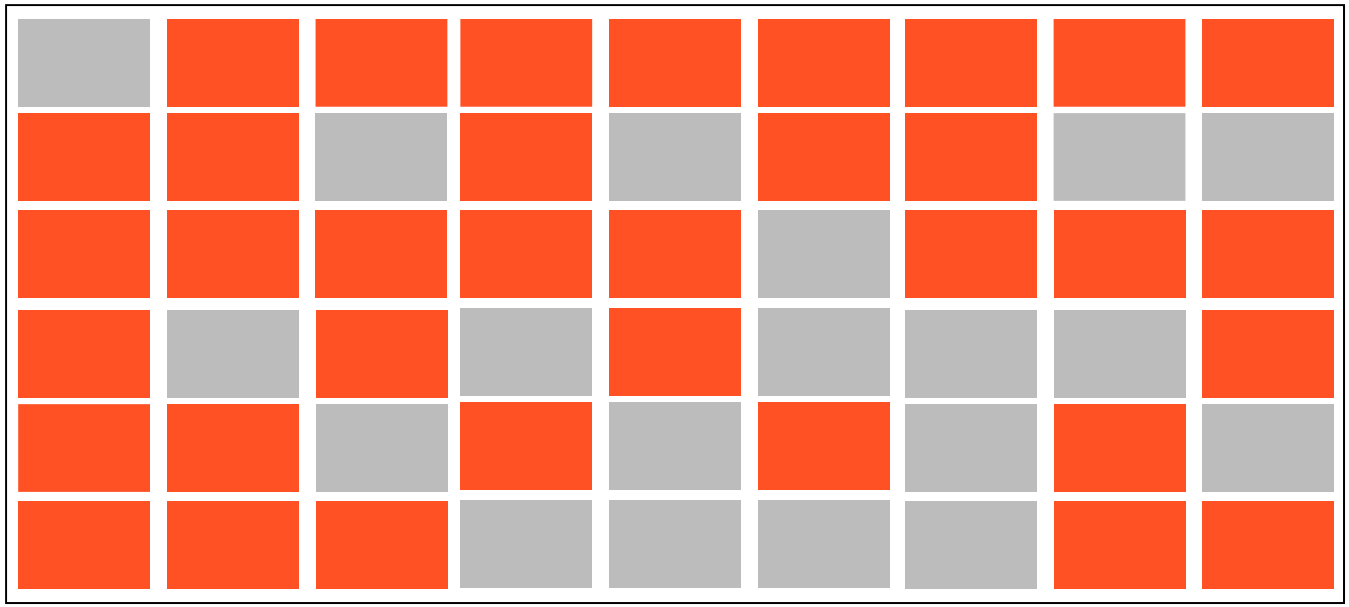
COLAS (QUEUES)

- Complejidad Operaciones:
- Encolar: **O(1)**
- Desencolar: **O(1)**
- Ver frente (Front): **O(1)**
- Comprobar vacía: **O(1)**

```
2   from collections import deque
3
4   cola = deque()
5
6   cola.append(10)    # Encolar
7   cola.popleft()     # Desencolar
8
9   frente = cola[0]
10  esta_vacia = len(cola) == 0
```

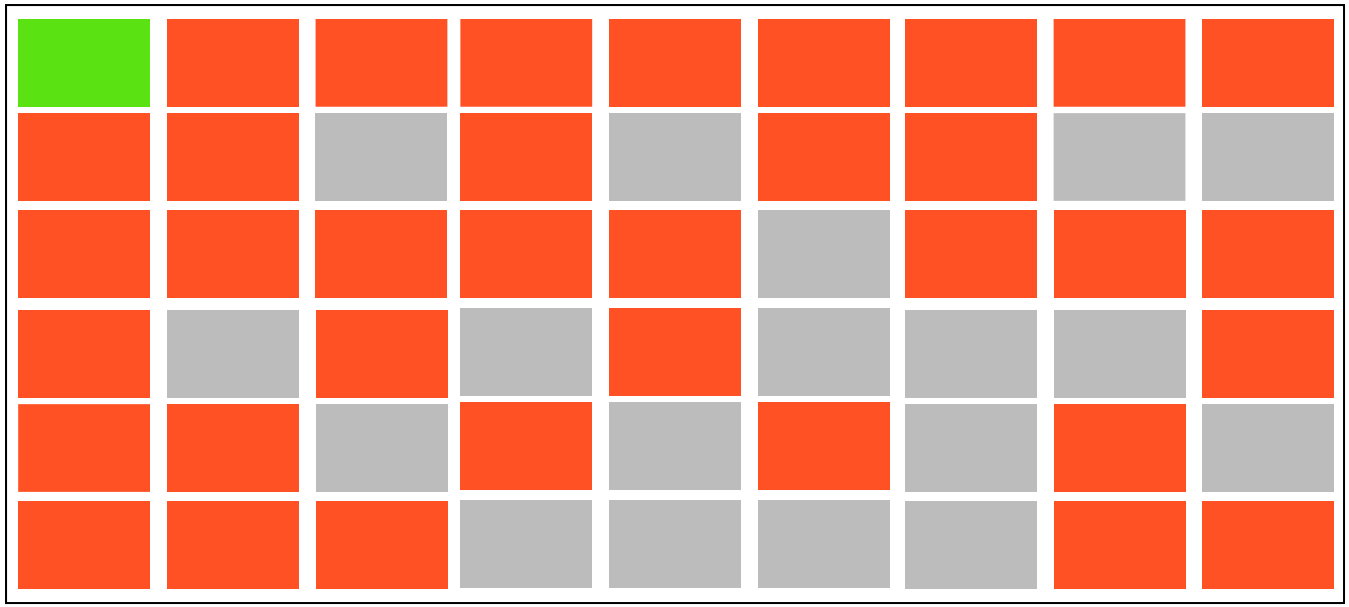
Funcionamiento Interno

Pilas y Colas (Lista doblemente enlazada)



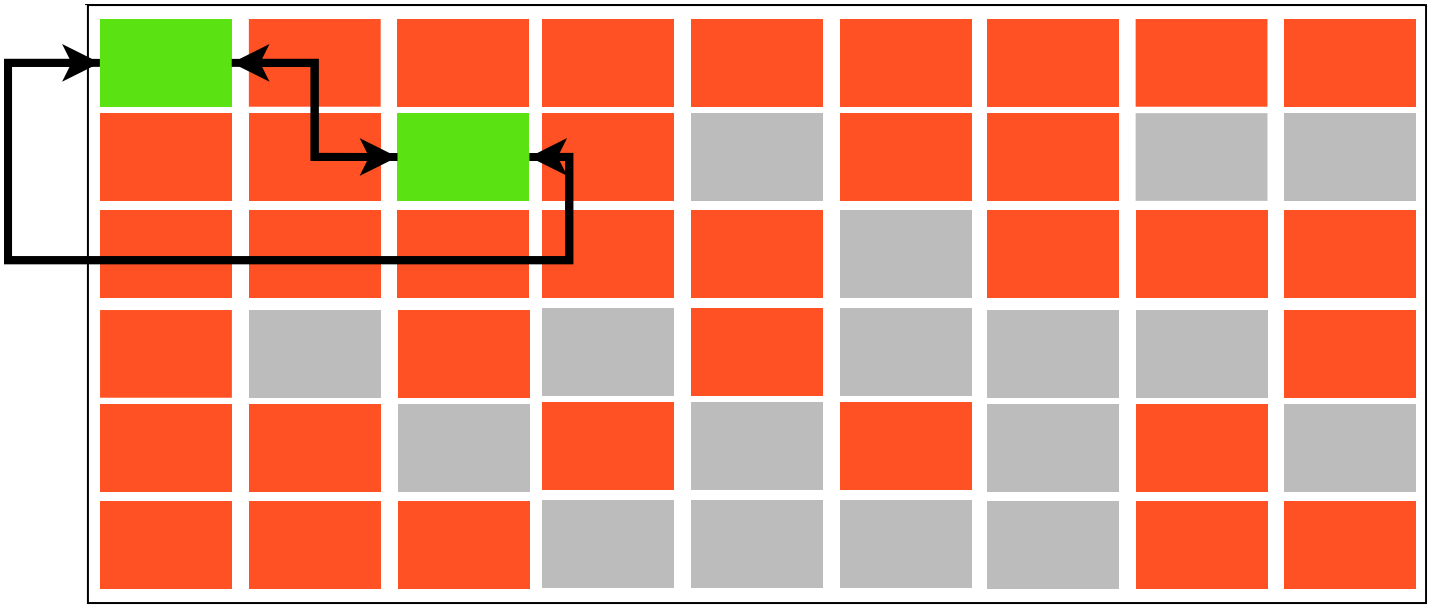
Funcionamiento Interno

Pilas y Colas (Lista doblemente enlazada)



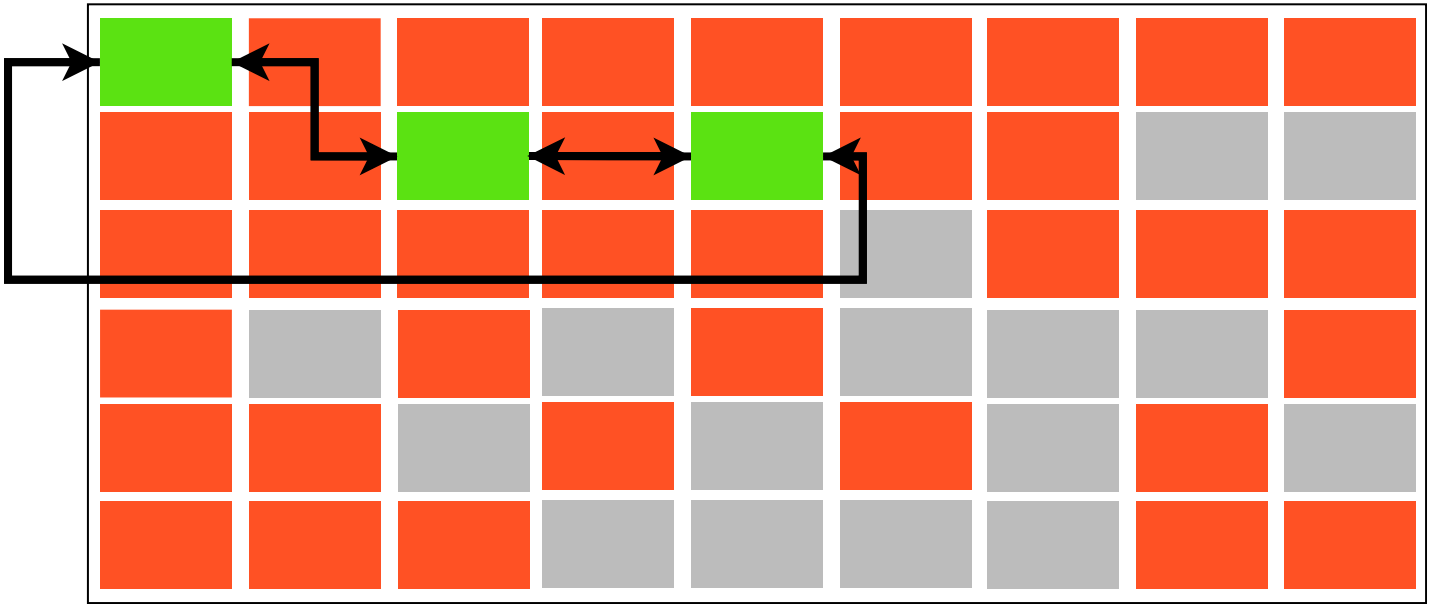
Funcionamiento Interno

Pilas y Colas (Lista doblemente enlazada)



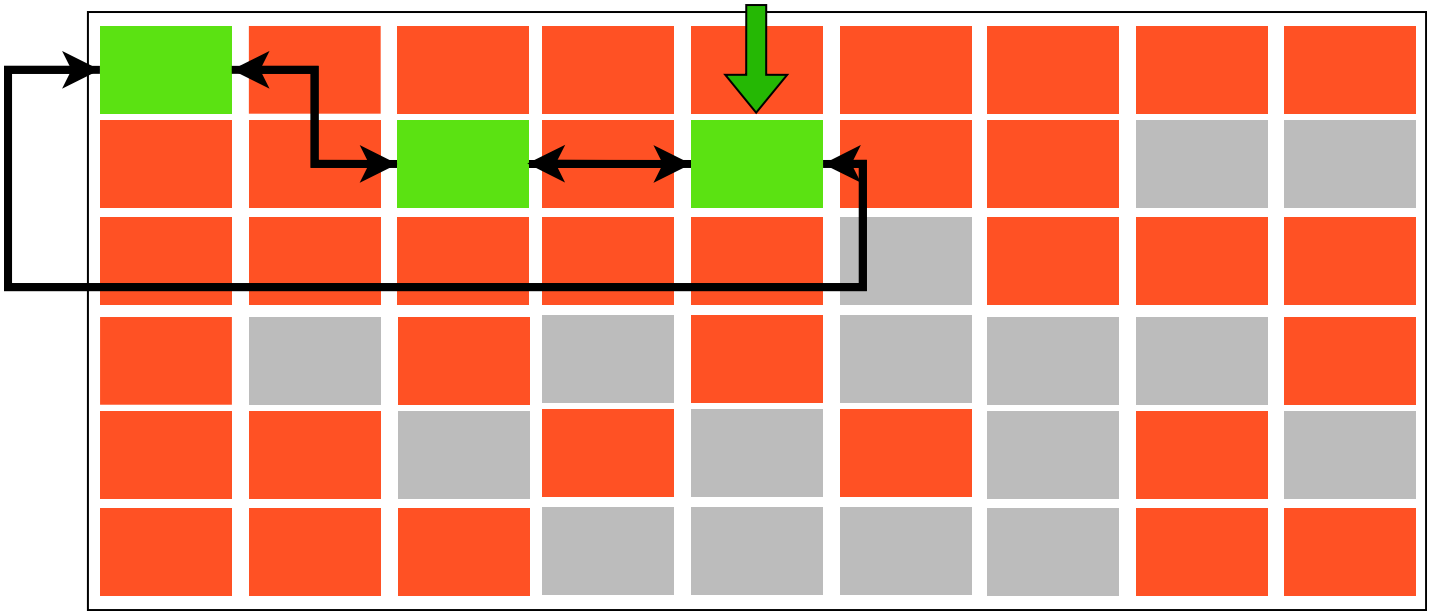
Funcionamiento Interno

Pilas y Colas (Lista doblemente enlazada)
Inserción O(1)



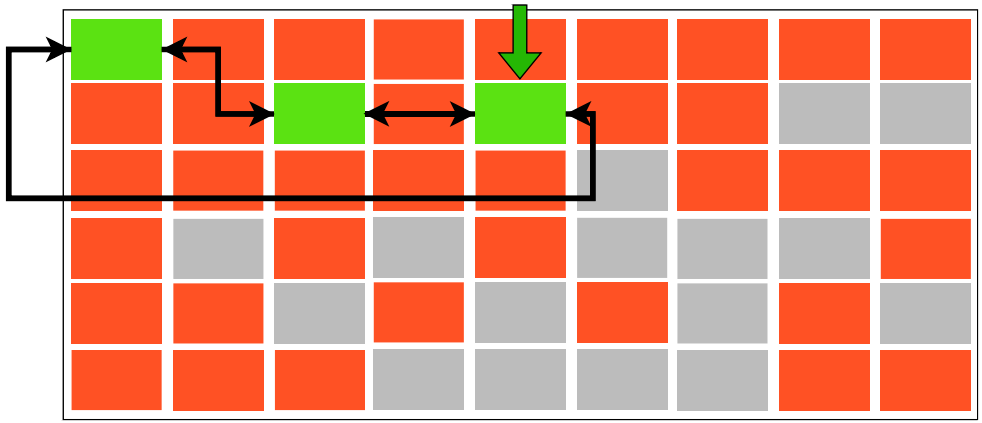
Funcionamiento Interno

Pila (Lista doblemente enlazada)



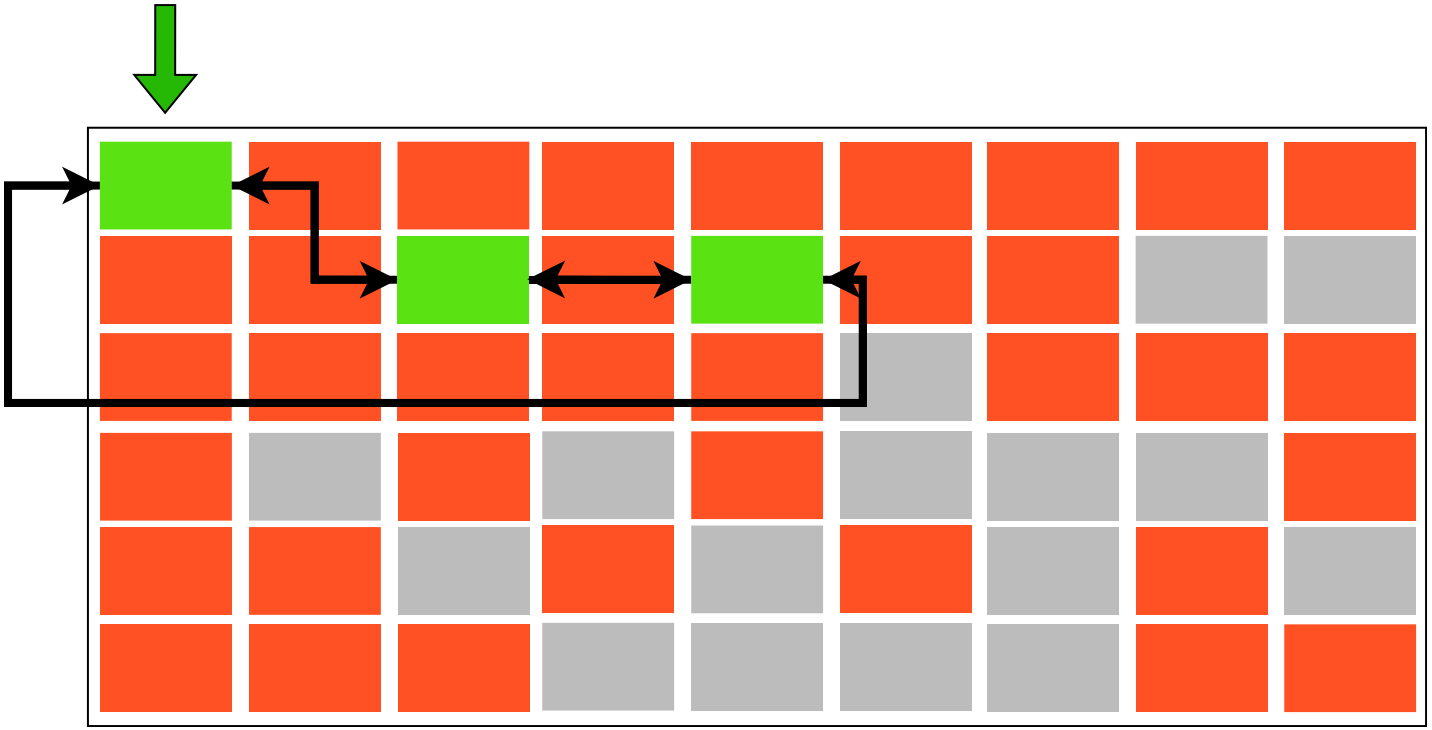
PILAS (STACKS)

- Complejidad Operaciones:
- Apilar: **O(1)**
- Desapilar (Pop): **O(1)**
- Ver cima (Top): **O(1)**
- Comprobar vacía: **O(1)**



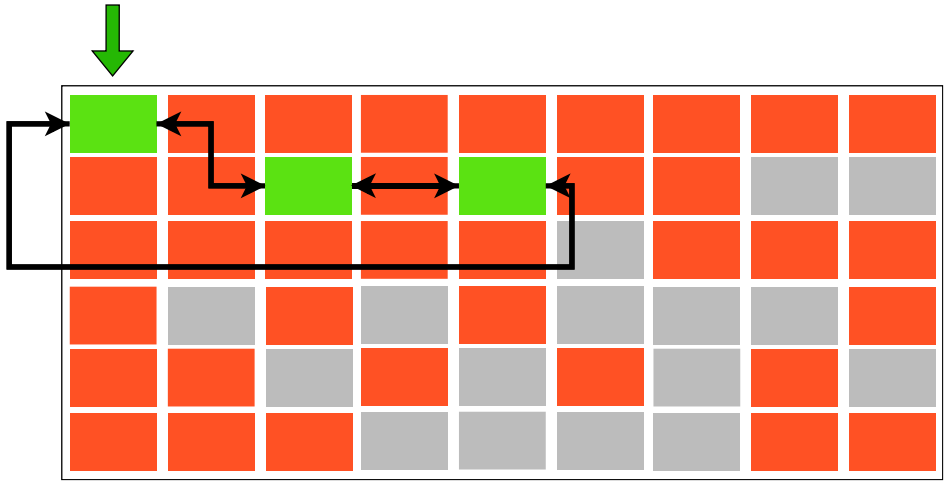
Funcionamiento Interno

Cola (Lista doblemente enlazada)



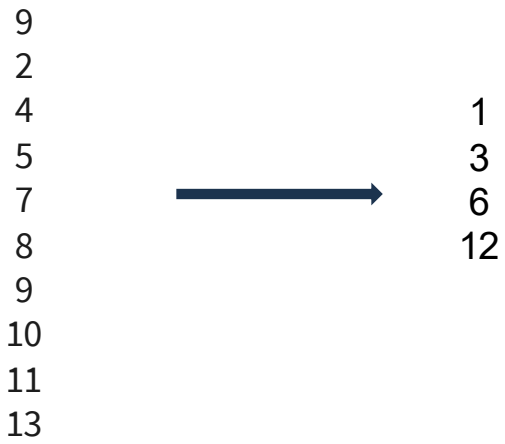
COLAS (QUEUES)

- Complejidad Operaciones:
- Encolar: **O(1)**
- Desencolar: **O(1)**
- Ver frente (Front): **O(1)**
- Comprobar vacía: **O(1)**



Missing Numbers

- https://open.kattis.com/problems/missingnumbers
- ID: missingnumbers
- Ejemplos:



COJUNTOS (SETS)

- Estructura de datos que almacena elementos **únicos**.
- Basado en una tabla **hash**.
- Aplicaciones:
  - **Búsqueda** rápida de elementos.
  - Eliminación de **duplicados**.
  - Operaciones de **conjuntos** (unión, intersección, diferencia).

```
2   set1 = set()
3   set2 = {30, 40, 50}
4
```

COJUNTOS (SETS)

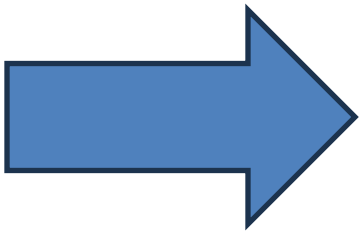
- Complejidad Operaciones:
- Inserción: **O(1)**
- Eliminación: **O(1)**
- Búsqueda: **O(1)**
- Unión, Intersección, Diferencia: **O(n)**

```
 2  set1 = set()
 3  set2 = {30, 40, 50}
 4
 5  set1.add(10)    # Insertar
 6  set1.remove(20)  # Eliminar
 7
 8  existe = 10 in set1
 9
10  union = set1 | set2
11  interseccion = set1 & set2
12  diferencia = set1 - set2
```

Funciones Hash

Conjunto de Strings de longitud hasta 10

- Hola
- Mañana
- Qué tal?
- Alberto
- ….



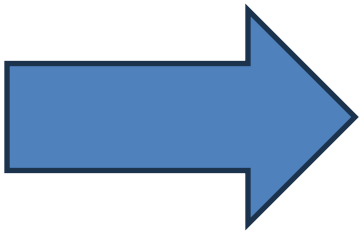
Indices de una lista de N posiciones

0,1,…, N-1

Funciones Hash

Conjunto de Strings de longitud hasta 10

- Hola
- Mañana
- Qué tal?
- Alberto
- ….



$$f(s)$$

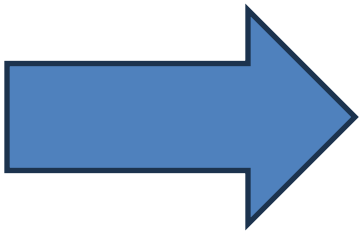
Indices de una lista de N posiciones

0,1,…, N-1

Funciones Hash

Conjunto de Strings de longitud hasta 10

- Hola
- Mañana
- Qué tal?
- Alberto
- ….



$$f(s)$$

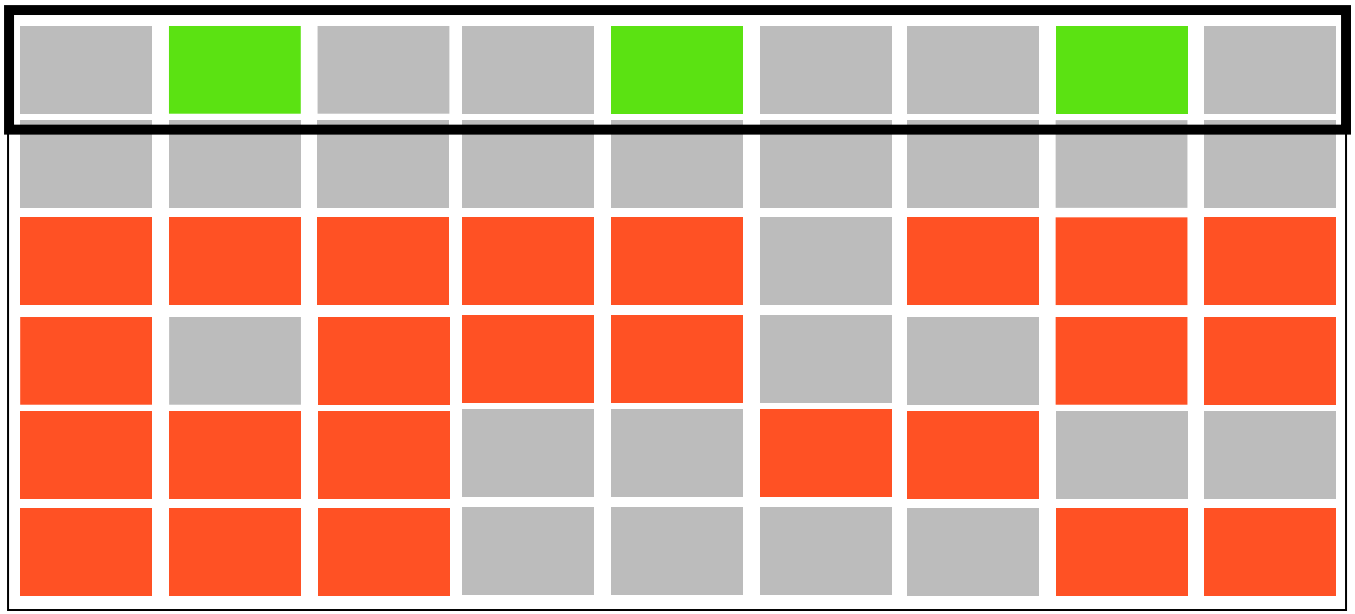
Indices de una lista de N posiciones

0,1,…, N-1

$$f(s) = bits(S) \mod N$$

Funcionamiento Interno

Conjuntos (Tabla Hash)

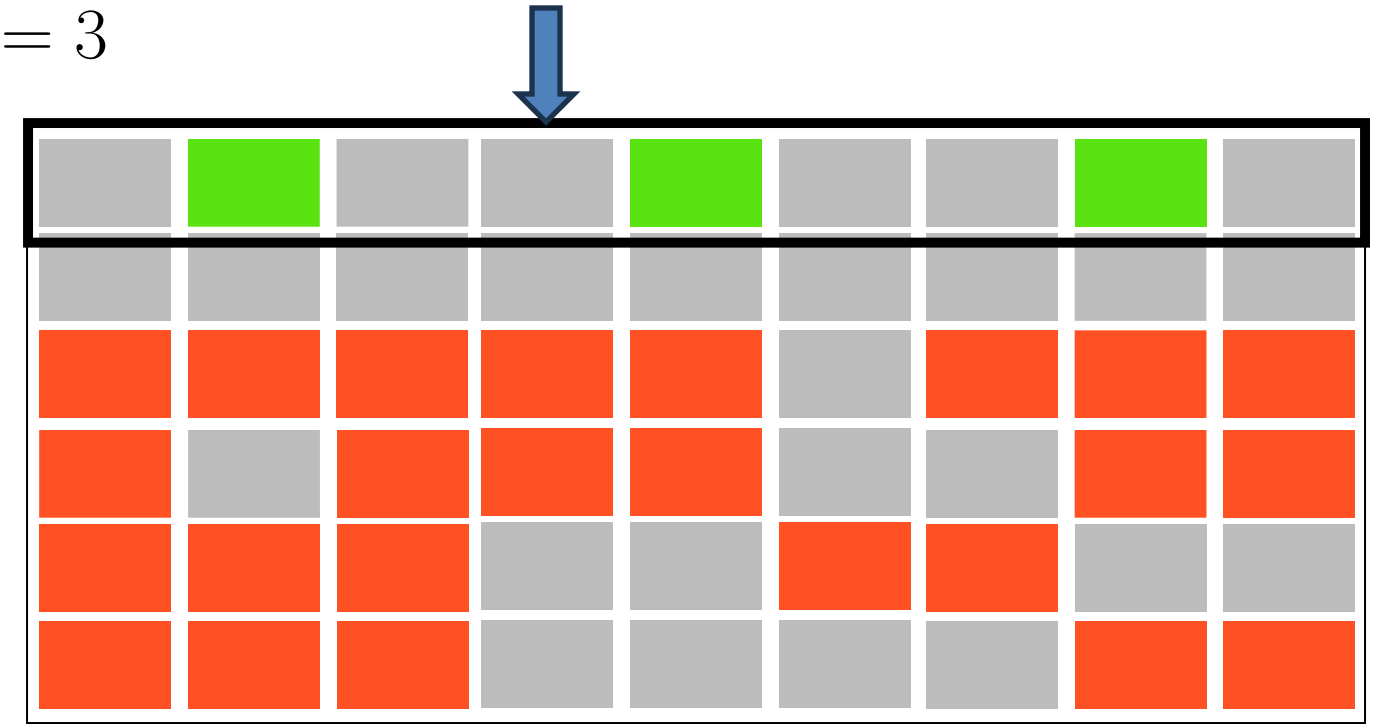


Funcionamiento Interno

Conjuntos (Tabla Hash)

s = hola $\quad f(s) = bits(S) \mod N = 103413 \mod 9 = 3$

$f(s) = 3$

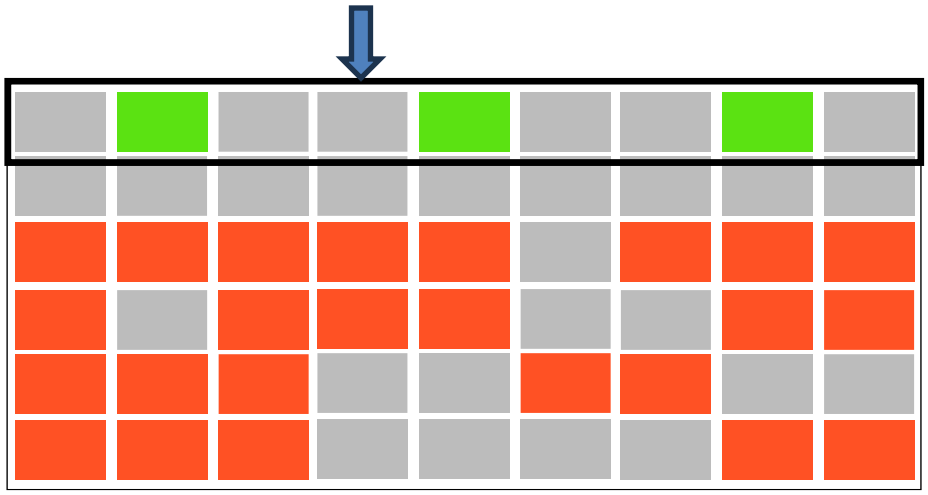


COJUNTOS (SETS)

- Complejidad Operaciones:
- Inserción: **O(1)**
- Eliminación: **O(1)**
- Búsqueda: **O(1)**
- Unión, Intersección, Diferencia: **O(n)**

s = hola

$$f(s) = 3$$

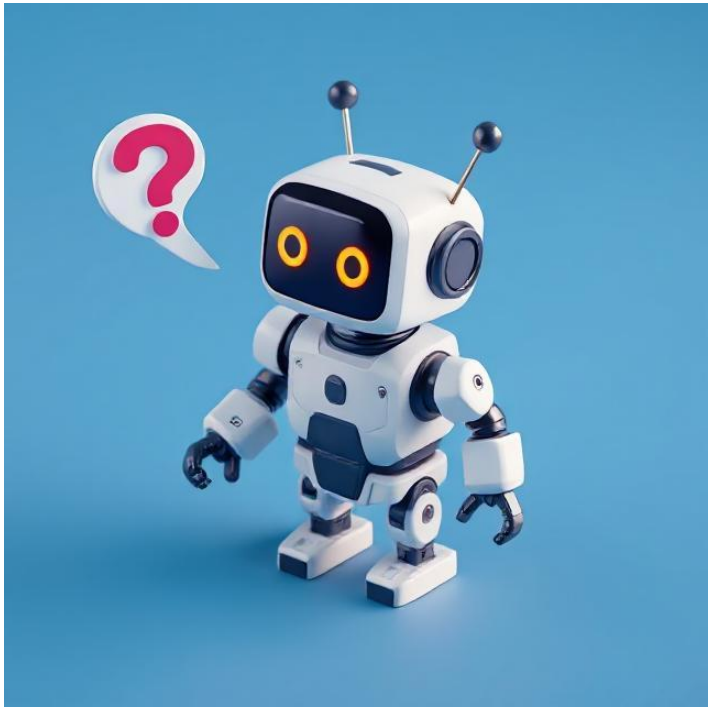


No Duplicates

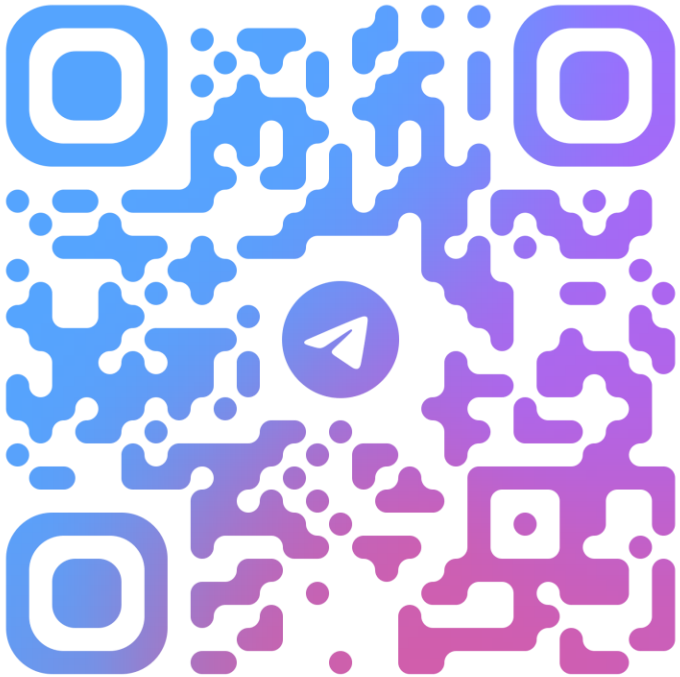
- https://open.kattis.com/problems/nodup
- ID: nodup
- Ejemplos:

THE RAIN IN SPAIN	⟶	yes
IN THE RAIN AND THE SNOW		no

¿PREGUNTAS?



HASTA LA SEMANA QUE VIENE!



CP-2026

Formulario