octubre 2025

GREEDY

Resumen | Robber | Fiu

Joaquín - Steven - Elías - Alexander - Paula

HOY



MOTIVACIÓN

Greedy es como la vida: tomas decisiones buscando lo mejor en ese momento,

sin saber si será lo mejor al final.

GREEDY

Decisiones locales óptimas

Se basa en tomar decisiones locales óptimas en cada paso con la “esperanza” de alcanzar una solución global óptima

No tiene recuerdos

No considera ni revisa las decisiones tomadas anteriormente, solo se enfoca en la decisión actual basada en la información disponible en ese momento.

Agregación gradual

Comienza con una solución vacía y agrega gradualmente elementos o toma decisiones que maximicen una métrica o función objetivo en cada paso.

No garantiza optimalidad global

No garantiza una solución óptima global y puede quedar atrapado en mínimos o máximos locales o soluciones parcialmente satisfactorias. (Requiere de un problema con subestructura óptima)

GREEDY

→ EJERCICIO 1

HOUSE ROBBER!

Eres un ladrón profesional que planea robar casas dispuestas a lo largo de una calle.
Cada casa tiene una cierta cantidad de dinero escondida, pero el único impedimento para robarlas todas es que las casas adyacentes tienen sistemas de seguridad conectados, y la policía será alertada si se roban dos casas consecutivas la misma noche.
Entonces, dado un arreglo entero **nums** que representa la cantidad de dinero en cada casa, devuelve la cantidad máxima de dinero que puedes robar esta noche sin alertar a la policía.

Escribe el pseudocódigo que permita resolver esta situación y determina la complejidad en tiempo y memoria de tu respuesta.

~~HOUSE ROBBER!~~ SOLUCIÓN

```
HouseRobber(nums):
    prev ← 0
    prev_prev ← 0
    for each num in nums:
        mejor_opción ← max(num + prev_prev, prev)
        prev_prev ← prev
        prev ← mejor_opción
    return max(prev, prev_prev)
```

~~HOUSE ROBBER!~~ SOLUCIÓN

```
HouseRobber(nums):
    prev ← 0
    prev_prev ← 0
    for each num in nums:
        mejor_opción ← max(num + prev_prev, prev)
        prev_prev ← prev
        prev ← mejor_opción
    return max(prev, prev_prev)
```

Tiempo: **O(N)**

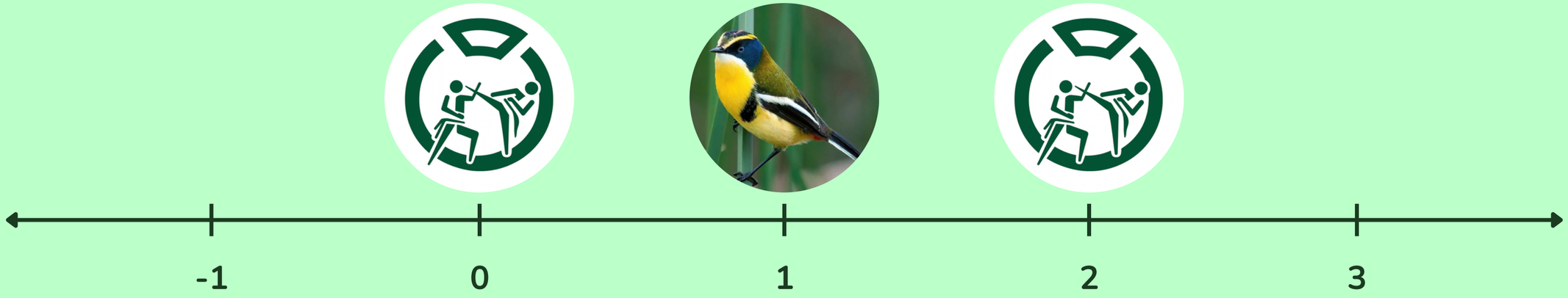
Espacio: **O(1)**

GREEDY

→ EJERCICIO 2

I2-2023-2 FIU

Para satisfacer la demanda de peluches de Fiu, la mascota de los Juegos Panamericanos y Parapanamericanos, se deben construir tiendas cerca de los recintos deportivos. Consideremos el problema de minimizar la cantidad de tiendas, para lo cual, representamos los recintos como puntos R = {r1, ..., rn } en una misma recta numérica, diciendo que un recinto está cubierto si en la recta hay una tienda a lo más a L kilómetros de el. Por ejemplo, si L = 1 y R = {0, 2} la solución óptima es ubicar una tienda entre ambos recintos



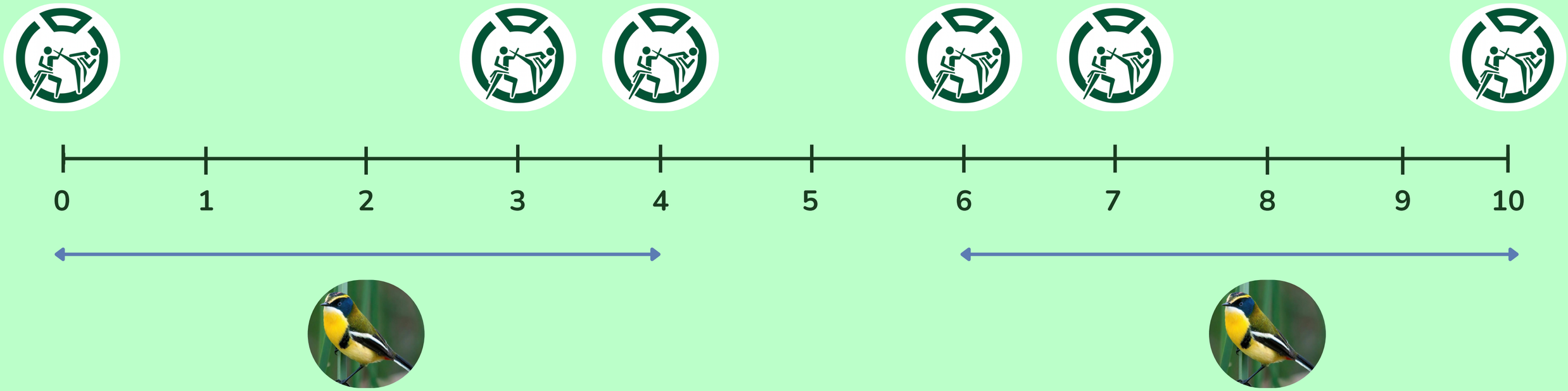
I2-2023-2 FIU I)

i) Se propone la siguiente estrategia para escoger dónde colocar tiendas: “**ubicar la siguiente tienda en la posición que maximiza el número de recintos nuevos cubiertos y que no habían sido cubiertos por tiendas añadidas antes**". Demuestre que esta estrategia no es óptima en todos los casos.

~~I2-2023-2 FIUI~~) SOLUCIÓN

a) Se propone la siguiente estrategia para escoger dónde colocar tiendas: **“ubicar la siguiente tienda en la posición que maximiza el número de recintos nuevos cubiertos y que no habían sido cubiertos por tiendas añadidas antes"**. Demuestre que esta estrategia no es óptima en todos los casos.

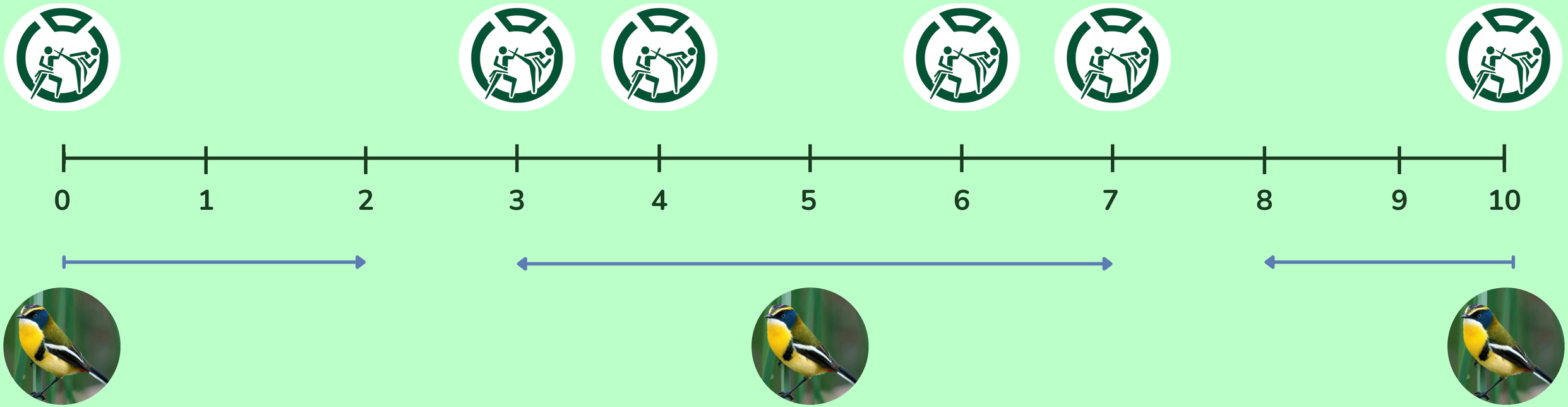
Solución esperada que sí respeta el mínimo!



~~I2-2023-2 FIUI~~) SOLUCIÓN

a) Se propone la siguiente estrategia para escoger dónde colocar tiendas: **“ubicar la siguiente tienda en la posición que maximiza el número de recintos nuevos cubiertos y que no habían sido cubiertos por tiendas añadidas antes"**. Demuestre que esta estrategia no es óptima en todos los casos.

Solución con estrategia (incorrecta)



I2-2023-2 FIU II)

ii) Considere la siguiente estrategia codiciosa: **“Con los recintos en orden creciente, si el primer recinto no cubierto es r, entonces ubicar la siguiente tienda a distancia L hacia adelante de r"**. Proponga el **pseudocódigo de un algoritmo** codicioso que use esta estrategia para retornar la lista más corta con las ubicaciones de las tiendas que cubren el conjunto de recintos R con distancia L. Asuma que R es un arreglo no necesariamente ordenado.

~~I2-2023-2 FIU II)~~ SOLUCIÓN

B) “Con los recintos en orden creciente, si el primer recinto no cubierto es r, entonces ubicar la siguiente tienda a distancia L hacia adelante de r".

Greedy(R, L):

```
1  T ← lista vacía
2  R ← InsertionSort(R)
3  Insertar al final de T el dato R[0] + L
4  i ← 1
5  while i < n :
6      if T.last + L < R[i] :
7          Insertar al final de T el dato R[i] + L
8      i + 1
9  return T
```

I2-2023-2 FIU III)

iii) Si R contiene n recintos y se entrega como arreglo no necesariamente ordenado, **determine si existe distinción entre mejor y peor caso para su algoritmo de (b)** y determine la **complejidad de los casos** identificados

~~I2-2023-2 FIU III)~~ SOLUCIÓN

C) Si R contiene n recintos y se entrega como arreglo no necesariamente ordenado, **determine si existe distinción entre mejor y peor caso para su algoritmo de (b)** y determine la **complejidad de los casos** identificados

Greedy(R, L):

```
1  T ← lista vacía
2  R ← InsertionSort(R)
3  Insertar al final de T el dato R[0] + L
4  i ← 1
5  while i < n :
6      if T.last + L < R[i] :
7          Insertar al final de T el dato R[i] + L
8      i + 1
9  return T
```

Sólo depende de los mejores/peores casos del algoritmo de ordenación. Para el caso específico de InsertionSort, el mejor caso ocurre cuando R está ordenado y el peor, cuando hay desorden en una cantidad lineal de elementos.

La complejidad del loop es O(n) siempre. Luego, como InsertionSort es O(n) en el mejor caso, el mejor caso del algoritmo es lineal O(n) + O(n) = O(n) .
El peor caso es cuadrático $O(n^2) + O(n) = O(n^2)$.

~~I2-2023-2 FIU III)~~ SOLUCIÓN

C) Si R contiene n recintos y se entrega como arreglo no necesariamente ordenado, **determine si existe distinción entre mejor y peor caso para su algoritmo de (b)** y determine la **complejidad de los casos** identificados

Greedy(R, L):

```
1  T ← lista vacía
2  R ← InsertionSort(R)
3  Insertar al final de T el dato R[0] + L
4  i ← 1
5  while i < n :
6      if T.last + L < R[i] :
7          Insertar al final de T el dato R[i] + L
8      i + 1
9  return T
```

Sólo depende de los mejores/peores casos del algoritmo de ordenación. Para el caso específico de InsertionSort, el mejor caso ocurre cuando R está ordenado y el peor, cuando hay desorden en una cantidad lineal de elementos.

La complejidad del loop es O(n) siempre. Luego, como InsertionSort es O(n) en su mejor caso,

Mejor caso es lineal $O(n) + O(n) = O(n)$

Peor caso es cuadrático $O(n^2) + O(n) = O(n^2)$

octubre 2025

GREEDY

Resumen | Robber | Fiu

Joaquín - Steven - Elías - Alexander - Paula