



Club de Algoritmia US { [Universidad de Sevilla]

Sesión 10: Pilas y Colas

}

Solución al problema 71 (LeetCode)

```
def simplifyPath(self, path: str) -> str:
    res = []
    i = 0
    while i < len(path):
        j = i+1
        while j < len(path) and path[j] != "/":
            j += 1
        word = path[i+1:j]
        if j - i > 1 and word != ".":
            if word == "..":
                if len(res)>0:
                    res.pop()
            else:
                res.append(path[i:j])
        i = j
    return "".join(res) if len(res)>0 else "/"
```



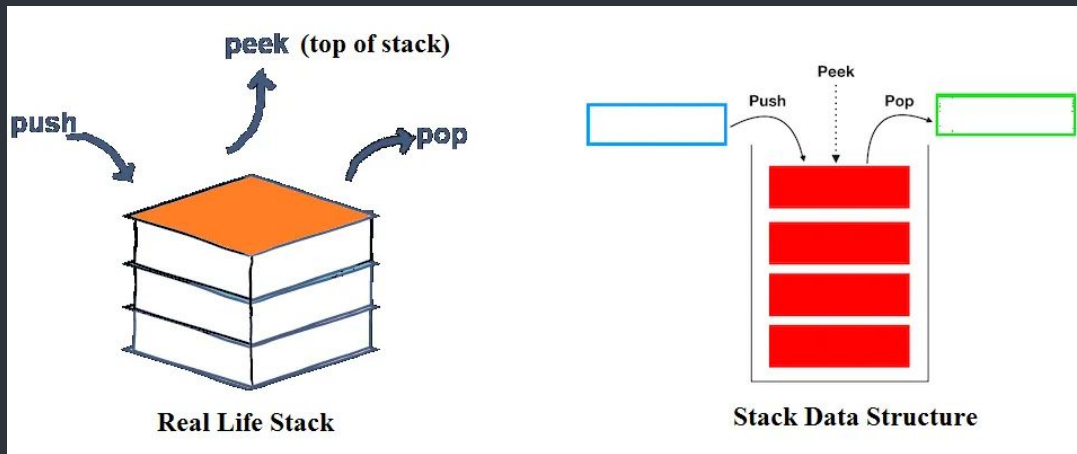
Lista

- Una lista es un tipo de datos abstracto.
- Todo el mundo sabe lo que es una lista:
 - Secuencia ordenada de elementos
 - Puedes añadir elementos
 - Puedes acceder a los elementos
 - Puedes eliminar elementos
- Nadie sabe cómo funcionan las listas por dentro (porque pueden funcionar de muchas formas distintas)



Stack (Pila)

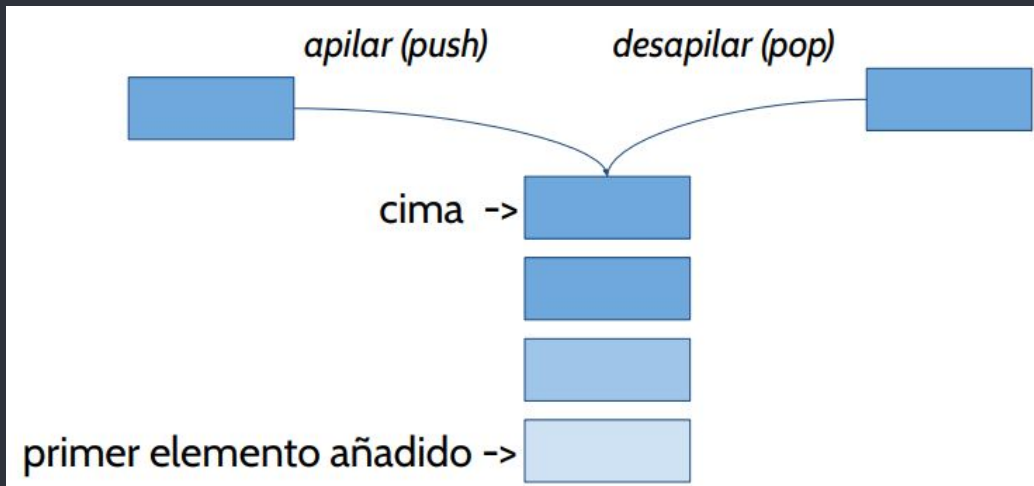
Tipo de datos abstracto estilo lista en la que podemos insertar o extraer elementos en una única posición llamada cima (head), siguiendo el modo de acceso **LIFO** (last-in, first-out).





Stack: Operaciones de manipulación

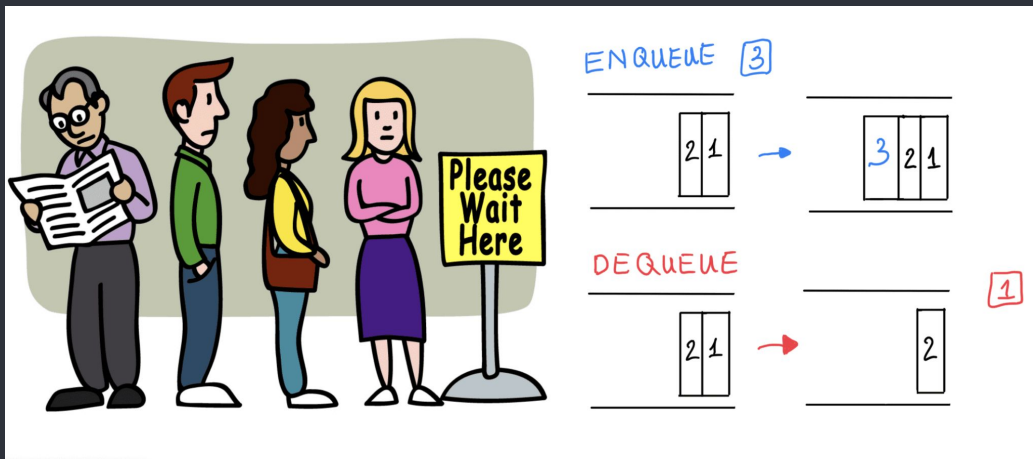
- Apilar (push): Añade un elemento e en la cima de la pila
- Desapilar (pop): Borra y devuelve el elemento que está en la cima de la pila





Queue (Cola)

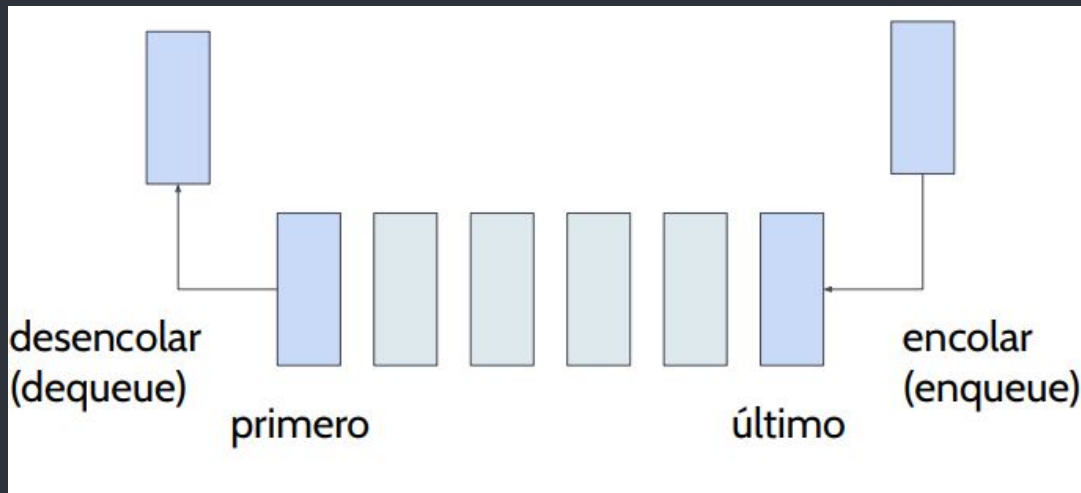
Tipo de datos abstracto estilo lista en el que las inserciones y extracciones siguen una política **FIFO** (first-in, first-out), lo que significa que el primer dato en entrar es el primer dato en salir.





Queue: Operaciones de manipulación

- Encolar (enqueue): Añade el elemento al final de la cola
- Desencolar (dequeue): Elimina y devuelve el primer elemento de la cola





Implementaciones: 'Array dinámico'

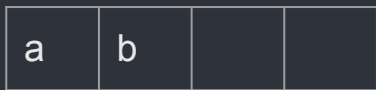
Un array dinámico es una estructura de datos que almacena elementos en posiciones adyacentes de la memoria. A diferencia de los arrays habituales en lenguajes como C, los arrays dinámicos pueden aumentar su tamaño cuando se alcanza el límite de su capacidad.

Veamos cómo se harían las operaciones *push* y *pop*.



Implementaciones: 'Array dinámico'

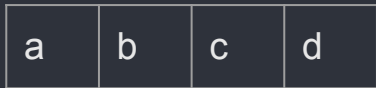
Añadir elemento (hay espacio)



start=0, head=1, size=4

start=0, head=2, size=4

Añadir elemento (no hay espacio)



start=0, end=3, size=4

start=x, end=x+4, size=8



Implementaciones: 'Array dinámico'

Eliminar elemento



start=0, end=4, size=8



start=0, end=3, size=8



Implementaciones: 'Linked list'

Una *linked list* es una lista en la que cada elemento cuenta con un puntero al siguiente elemento.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|----|-----|--------|----|------|----|----|-----|------|----|
| | | b;11 | | | | | | d;14 | |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| | c;8 | | | e;22 | | | | | |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| | | f;null | | | | | a;2 | | |

start=27, end=22



Implementaciones: 'Doubly linked list'

Igual que una *linked list* pero con punteros en ambos sentidos.

| | | | | | | | | | |
|----|-------|-----------|----|--------|----|----|----------|---------|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| | | b;27;11 | | | | | | d;11;14 | |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| | c;2;8 | | | e;8;22 | | | | | |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| | | f;14;null | | | | | a;null;2 | | |

start=27, end=22



Comparativa

Un mismo tipo de datos abstractos puede tener distintas complejidades para las mismas operaciones dependiendo de la estructura de datos que se utilice para implementarlo.

| | Array dinámico | Linked list | Doubly linked list |
|--------|---------------------|-------------|----------------------|
| push | $O(1)$ (amortizado) | $O(1)$ | $O(1)$ (ambos lados) |
| pop | $O(1)$ | $O(1)$ | $O(1)$ (ambos lados) |
| access | $O(1)$ | $O(n)$ | $O(n)$ |
| insert | $O(n)$ | $O(1)$ | $O(1)$ |
| remove | $O(n)$ | $O(1)$ | $O(1)$ |



Tipos estilo lista en Python

Array dinámico → Tipo “list”

Doubly linked list → Tipo “deque”

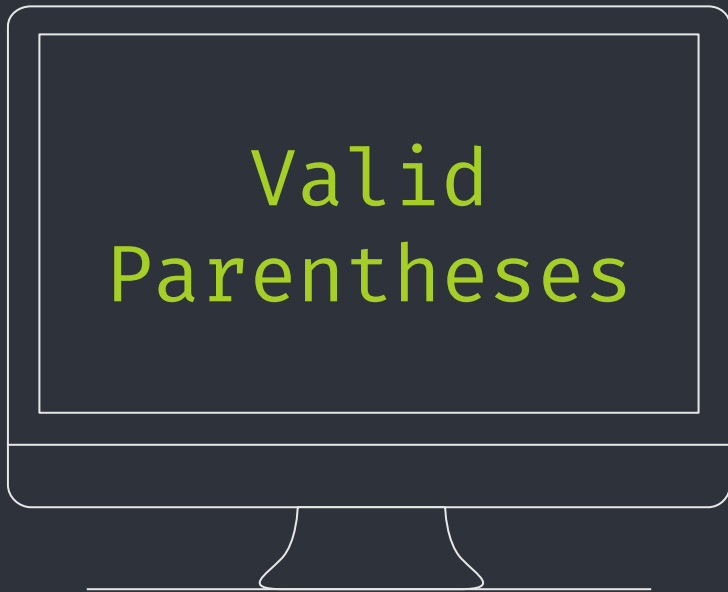
Generalmente interesa utilizar el tipo “list”.

Cuando queremos un comportamiento estilo pila o cola es más eficiente utilizar el tipo “deque”

```
from collections import deque
```



LeetCode - Número 20



1
2
3
4
5
6
7
8
9
10
11
12
13
14



Problema: 'Valid Parentheses'

{

Determina si una cadena "s" que contiene solo '(', ')', '{', '}', '[', y ']', es válida. Para que sea válida, los corchetes deben cerrarse con el mismo tipo de corchete y en el orden correcto, asegurándose de que cada corchete cerrado tenga su correspondiente corchete abierto del mismo tipo.

Input:

s = "()[]{}"

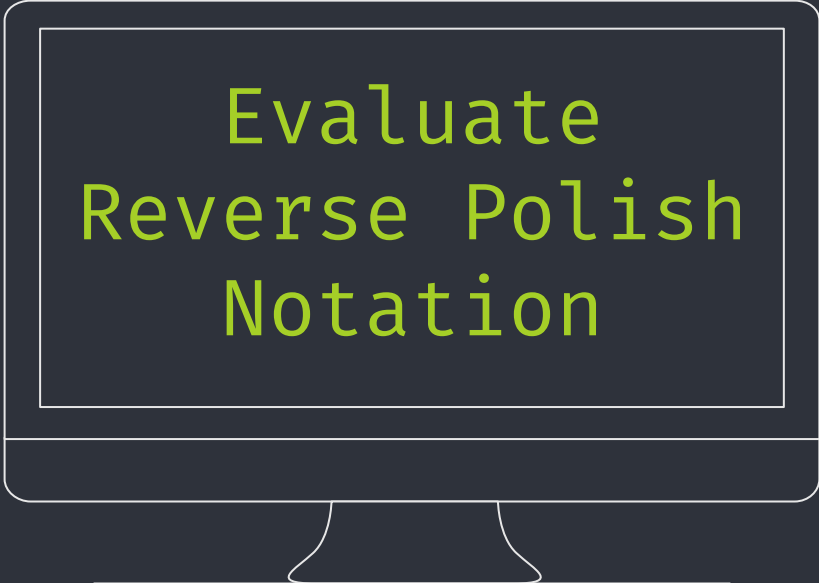
Output:

true

}



LeetCode - Número 153



Evaluate
Reverse Polish
Notation



Problema: 'Evaluate RPN'

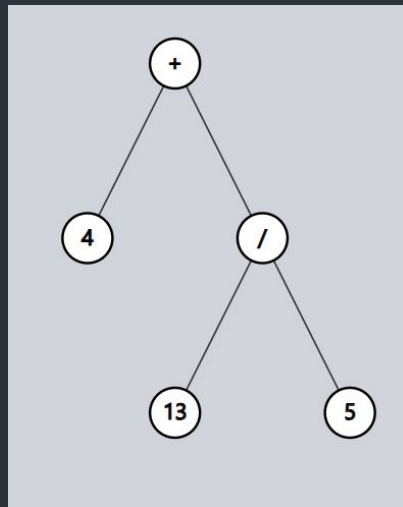
{ Tienes un array de caracteres que representa una expresión aritmética en la notación Reverse Polish. Evalúa la expresión

Input:

s = ["4", "13", "5", "/", "+"]

Output: 6

Explanation: $(4 + (13 / 5)) = 6$



Solución (pila)

```
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []
        operators = {
            "+": lambda a, b: a + b,
            "-": lambda a, b: a - b,
            "*": lambda a, b: a * b,
            "/": lambda a, b: int(a / b) # Redondear hacia cero
        }

        for token in tokens:
            if token in operators:
                b = stack.pop()
                a = stack.pop()
                stack.append(operators[token](a, b))
            else:
                stack.append(int(token))

        return stack.pop()
```



LeetCode - Número 649

Dota2 Senate

1
2
3
4
5
6
7
8
9
10
11
12
13
14



Problema: 'Dota2 Senate'

```
{
  El senado del juego Dota2 está compuesto por senadores de
  dos partidos: los Radiant y los Dire. Las votaciones sobre
  cambios en el juego se deciden por rondas. En cada ronda,
  cada senador disponible puede ejercer uno de dos derechos:

  1. Vetar a otro senador: el senador objetivo pierde su
  derecho al voto.

  2. Anunciar la victoria: si todos los senadores que aún
  tienen derecho a votar pertenecen a su partido, puede
  anunciar la victoria.
}
```



Problema: 'Dota2 Senate'

{

La entrada consiste en una cadena senate que representa la afiliación partidista de cada senador, donde 'R' representa a un senador de los Radiant y 'D' a uno de los Dire.

El objetivo es predecir qué partido finalmente anunciará la victoria y decidirá el cambio en el juego.

Importante: los senadores votan en orden

}



Problema: 'Dota2 Senate'

{

Input:

senate = "RD"

Output: "Radiant"

}

Input:

senate = "RDD"

Output: "Dire"

Solución (colas)

```
class Solution:
    def predictPartyVictory(self, senate: str) -> str:
        radiantQueue = deque()
        direQueue = deque()

        n = len(senate)

        for i in range(n):
            if senate[i] == 'R':
                radiantQueue.append(i)
            else:
                direQueue.append(i)

        while radiantQueue and direQueue:
            radiant_index = radiantQueue.popleft()
            dire_index = direQueue.popleft()

            if radiant_index < dire_index:
                radiantQueue.append(radiant_index + n)
            else:
                direQueue.append(dire_index + n)

        return "Radiant" if radiantQueue else "Dire"
```



end.py

workshop.java

1 ¡Gracias!
2
3
4 ¡Nos vemos la semana que
5 viene!
6
7
8
9

- 10 • Próxima sesión: Árboles y colas de prioridad
11
12
13
14

