Actividad 1

Estructuras de Datos lineales: Pila dinámica

Carmen Witsman García

1. Introducción

En esta actividad, implementaremos una **pila dinámica**, con estructura LIFO (Last-in, First-out), de las siguientes **formas**:

- Con la técnica de construcción de **lista enlazada**: Clase `LinkedStack`
- Con soporte de **Lista de Python**: Clase `ListStack`
- Con soporte de **estructura DeQue**: Clase `DqueStack`
- Con soporte de estructura **LifoQueue**: Clase `LifoQueueStack`

Todas las clases que crearemos tendrán los siguientes **métodos**:

- `__init__` - Método constructor con el que inicializaremos la pila vacía
- `push(e)` - Apila un elemento (e) al inicio de la pila
- `pop()` - Extrae y devuelve el primer elemento de la pila
- `peek()` - Devuelve la cima de la pila (el primer elemento)
- `empty()` - Comprueba si la pila está vacía
- `clear()` - Elimina todos los elementos de la pila

Comentaremos para cada clase la forma en la que implementamos dichos métodos, y mostraremos ejemplos de uso de cada uno de ellos.

Al final, realizaremos una comparativa de estos 4 métodos a nivel de rendimiento y facilidad de uso y de comprensión.

2. Infome

Los distintos constructores que hemos empleado en cada método son:

- Clase `LinkedStack`

  Creamos un puntero "first": `self.first = None`

- Clase `ListStack`

  Creamos una lista para almacenar los elementos de la pila: `self.pila = []`

- Clase `DqueStack`

  Creamos una pila vacía con la función queue.deque(): `self.pila = dq()`

- Clase `LifoQueueStack`

  Creamos una pila vacía con la función queue.LifoQueue(): `self.pila = lq()`

Facilidad de uso y comprensión:

- Mediante el sistema de **lista enlazada**, hay que actualizar el puntero, lo que hace que el código sea más elaborado y menos fácil de comprender. En lugar de utilizar funciones útiles para la implementación de los métodos, tenemos que trabajar con sentencias "if" y haciendo uso de otra clase auxiliar Nodo(), lo que no es nada eficiente.
- Usando **de soporte una lista de Python** podemos iterar, lo que hace más fácil almacenar los elementos de la pila. Además, podemos hacer uso de funciones útiles para implementar en los métodos de la clase:
  - insert(index, e): Nos permite insertar un elemento (e) en el inicio de la pila (índice=0)
  - pop(index): Nos permite extraer y devolver el elemento del inicio de la pila (índice=0)
- Con **queue.deque()** tenemos funciones que facilitan el uso y comprensión del código:
  - appendleft(e): Nos permite apilar un elemento al inicio de la pila
  - popleft(): Nos permite extraer y devolver el elemento del inicio de la pila

  Además, hace que escribamos el código en menos líneas.
- Con **queue.LifoQueue()** también tenemos funciones específicas para pilas:
  - put(e): Nos permite apilar un elemento al inicio de la pila
  - get(): Nos permite extraer y devolver el elemento del inicio de la pila

Aun así, no posee una función para mostrar elementos, ya que solo elimina el elemento inicial con get(). Por ello, hubo que hacer más uso de líneas de código almacenando el elemento, para luego reintroducirlo en la pila y mostrarlo mediante peek().

Después de implementar los 6 métodos en las diferentes clases, llego a la conclusión de que **el código más fácil de usar y comprender es el de DqueStack** gracias a las funciones de queue.deque.

No es así con queue.LifoQueue ya que sus funciones son más limitadas, además de que no hay manera de iterar en la pila.

3. Comparativa de rendimiento

Resultados

- `DqueStack`: La implementación DqueStack es la más rápida con un tiempo promedio de ejecución de 5.65 microsegundos por loop.
- `ListStack`: La implementación ListStack se ubica en segundo lugar con un tiempo promedio de ejecución de 8.51 microsegundos por loop.
- `LinkedStack`: La implementación LinkedStack es un poco más lenta que ListStack con un tiempo promedio de ejecución de 14.8 microsegundos por loop.
- `LifoQueueStack`: La implementación LifoQueueStack es la más lenta con un tiempo promedio de ejecución de 59.3 microsegundos por loop (y solo se ejecutaron 10,000 loops en la medición).

Conclusiones

DqueStack aprovecha la eficiencia de collections.deque para realizar operaciones de push y pop de forma rápida.

ListStack utiliza listas de Python, que son eficientes para la inserción y eliminación al final (operaciones LIFO de la pila).

LinkedStack implica la creación y manipulación de nodos enlazados, lo que puede ser ligeramente más lento que las estructuras de datos basadas en arrays.

LifoQueueStack parece tener una implementación menos eficiente para las operaciones de pila, ya que su tiempo de ejecución es significativamente mayor y se ejecutaron menos loops en la medición.