

Tecnicatura Universitaria en **PROGRAMACIÓN**

Listas enlazadas



Pilas y Colas

Las pilas y colas son estructuras de datos que se utilizan para gestionar elementos en una orden específica. A continuación, te explico cómo implementarlas en Python.



Pilas (Stacks)

Una pila es una estructura de datos que sigue el principio de "LIFO" (Last In, First Out), lo que significa que el elemento agregado más recientemente es el primero en ser eliminado. En Python, se puede implementar una pila utilizando una lista y métodos para agregar y eliminar elementos.

Implementación de una Pila en Python

Para implementar una pila en Python, puedes utilizar una lista y definir métodos para agregar y eliminar elementos. Aquí te muestro un ejemplo:

```
1 class Pila:
2     def __init__(self):
3         self.elementos = []
4
5     def push(self, elemento):
6         self.elementos.append(elemento)
7
8     def pop(self):
9         if self.elementos:
10            return self.elementos.pop()
11        else:
12            return None
13
14    def peek(self):
15        if self.elementos:
16            return self.elementos[-1]
17        else:
18            return None
19
20    def is_empty(self):
21        return len(self.elementos) == 0
22
```

Colas (Queues)

Una cola es una estructura de datos que sigue el principio de "FIFO" (First In, First Out), lo que significa que el elemento agregado primero es el primero en ser eliminado. En Python, se puede implementar una cola utilizando una lista y métodos para agregar y eliminar elementos.

Implementación de una Cola en Python

Para implementar una cola en Python, puedes utilizar una lista y definir métodos para agregar y eliminar elementos. Aquí te muestro un ejemplo:

```
1 class Cola:
2     def __init__(self):
3         self.elementos = []
4
5     def enqueue(self, elemento):
6         self.elementos.append(elemento)
7
8     def dequeue(self):
9         if self.elementos:
10            return self.elementos.pop(0)
11        else:
12            return None
13
14    def peek(self):
15        if self.elementos:
16            return self.elementos[0]
17        else:
18            return None
19
20    def is_empty(self):
21        return len(self.elementos) == 0
22
```

enqueue:

La operación enqueue se utiliza para agregar un elemento al final de la cola.

Cuando se realiza una operación enqueue, el elemento se agrega al final de la cola.

La implementación típica de enqueue en Python utiliza el método `append()` de la lista para agregar el elemento al final de la cola.

dequeue:

La operación dequeue se utiliza para eliminar y devolver el elemento del frente de la cola.

Cuando se realiza una operación dequeue, el elemento del frente de la cola se elimina y se devuelve.

La implementación típica de dequeue en Python utiliza el método `pop(0)` de la lista para eliminar y devolver el primer elemento de la cola.

enqueue y dequeue no son palabras reservadas o comandos integrados en el lenguaje Python. Son términos generales utilizados en la teoría de estructuras de datos para describir las operaciones de agregar y eliminar elementos de una cola.

Ventajas y Desventajas

Ambas estructuras de datos tienen ventajas y desventajas:

Ventajas de las pilas:

Facilitan la implementación de algoritmos que requieren acceder a los elementos más recientes.

Son más eficientes para operaciones de inserción y eliminación en el final.

Desventajas de las pilas:

No permiten acceder directamente a elementos específicos.

No son adecuadas para operaciones que requieren acceder a elementos en una posición específica.

Ventajas y Desventajas

Ambas estructuras de datos tienen ventajas y desventajas:

Ventajas de las colas:

Permiten acceder directamente a los elementos más antiguos.

Son adecuadas para operaciones que requieren acceder a elementos en una posición específica.

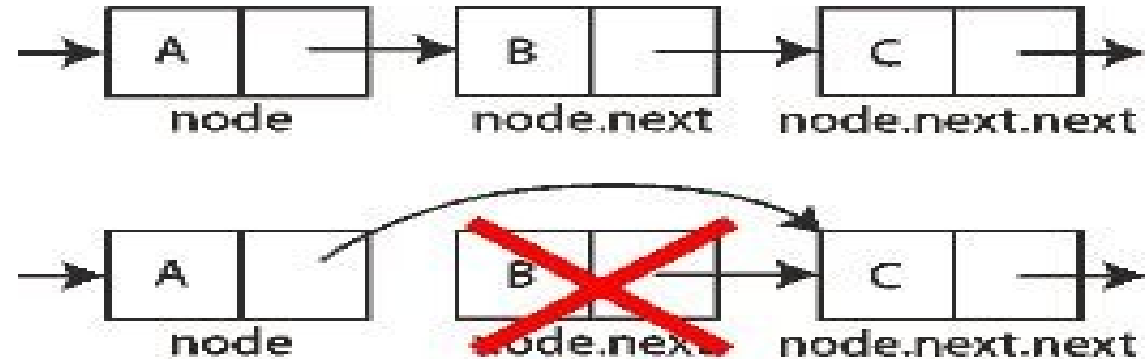
Desventajas de las colas:

No son tan eficientes para operaciones de inserción y eliminación en el final.

Requieren más memoria para almacenar los elementos.

En resumen, las pilas son adecuadas para operaciones que requieren acceder a los elementos más recientes, mientras que las colas son adecuadas para operaciones que requieren acceder a los elementos más antiguos.

Listas Enlazadas



Las listas enlazadas en Python son estructuras de datos dinámicas que permiten almacenar y manipular elementos de manera eficiente.

Una lista enlazada es una estructura de datos que se compone de nodos, cada uno de los cuales contiene un valor y un puntero al siguiente nodo en la lista. Esta estructura permite agregar o eliminar elementos en cualquier posición de la lista sin necesidad de reordenar todos los elementos, lo que la hace especialmente útil para aplicaciones que requieren operaciones de inserción y eliminación frecuentes.

Implementación de una lista enlazada en Python

```
1  class Nodo:
2      def __init__(self, valor):
3          self.valor = valor
4          self.siguiente = None
5
6
7  class ListaEnlazada:
8      def __init__(self):
9          self.cabeza = None
10
11      def agregar_al_inicio(self, valor):
12          nuevo_nodo = Nodo(valor)
13          nuevo_nodo.siguiente = self.cabeza
14          self.cabeza = nuevo_nodo
15
16      def imprimir(self):
17          actual = self.cabeza
18          while actual:
19              print(actual.valor)
20              actual = actual.siguiente
21
```

Clase de nodo: Primero, se define una clase `Nodo` que contiene un valor y un puntero al siguiente nodo en la lista.

Clase de lista enlazada: Luego, se define una clase `LinkedList` que contiene una lista de nodos y métodos para insertar, eliminar y buscar elementos.

Ventajas y desventajas

Ventajas:

Dinámico: La cantidad de nodos en una lista enlazada no es fija y puede crecer o disminuir según sea necesario.

Eficiente: Las operaciones de inserción y eliminación son más rápidas que en matrices, ya que no se requiere reordenar todos los elementos.

Flexibilidad: Las listas enlazadas se pueden utilizar para implementar estructuras de datos como pilas y colas.

Ventajas y desventajas

Desventajas:

Memoria: Las listas enlazadas utilizan más memoria que las matrices debido al almacenamiento de los punteros.

Acceso secuencial: Los nodos en una lista enlazada deben leerse en orden desde el principio, lo que puede ser un problema en aplicaciones que requieren acceso directo a elementos individuales.

Dificultades en la reversa: Las listas enlazadas pueden ser engorrosas para navegar hacia atrás, especialmente si no se utiliza una lista doblemente enlazada.

¿PREGUNTAS?

