UNIVERSIDAD DIEGO PORTALES

Presentación del Curso

CIT-2000

Estructuras de Datos

Clase 7

Lista Doble y Circular Doble

Profesor: Mauricio Hidalgo

Repaso clase anterior

Practicar lo aprendido sobre Lista Simple y Circular Simple

- Resolver un ejercicio que involucra Lista Simple
- Resolver un ejercicio que involucra Lista Circular Simple

Lista Doble y Circular Doble

Objetivos de la clase

Conocer las listas enlazadas dobles y circular doble

- Conocer las generalidades de un nodo, y de las listas doble y circular doble
- Aprender cómo se implementa un nodo con dos enlaces
- Aprender como se implementa una lista doble
- Aprender como se implementa una lista circular doble

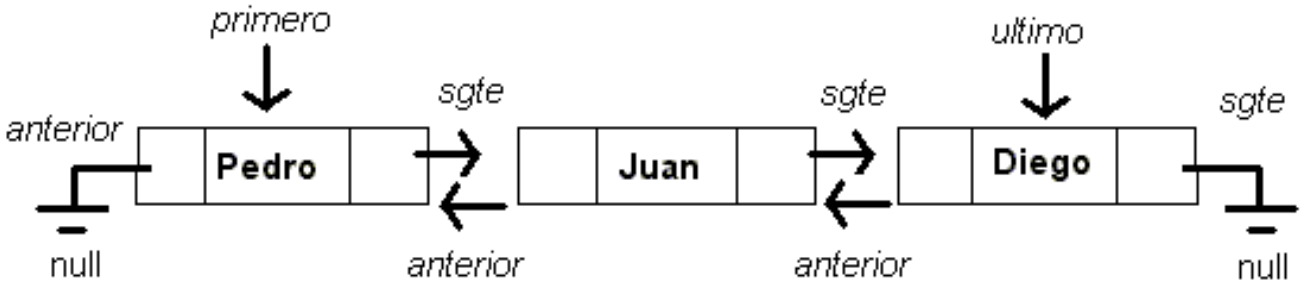
Lista Doblemente Enlazada

¿Qué es una Lista Doblemente Enlazada?

Según Wikipedia

"Es aquella en que cada nodo tiene dos enlaces: uno apunta al nodo anterior, o apunta al valor NULL si es el primer nodo; y otro que apunta al nodo siguiente, o apunta al valor NULL si es el último nodo."

Gráficamente



Lista Doblemente Enlazada

Nodo para lista doblemente enlazada

Es un objeto con:

- ✓ Uno o más atributos (sus elementos)
- ✓ Dos atributos adicionales – Puntero siguiente y Puntero Anterior

Para construirlo:

- ✓ Los atributos guardan los elementos del nodo.
- ✓ El puntero siguiente y el anterior deben ser NULO

Métodos:

- ✓ Solo nos interesa rescatar el dato, por ende, tiene solo un “get”
- ✓ No tiene un “set” elemento ya que eso irá, por añadidura, en la inserción de un nuevo nodo a una lista.

Lista Doblemente Enlazada

Clase Nodo Simple

```
#Clase Nodo con dos punteros
class Nodo(object):
    def __init__(self,elemento):
        #Atributo que tendrá el Nodo - Puede tener más
        self.__elemento=elemento
        #Dos punteros que servirán para "unir" los Nodos
        #cuando se construya la lista. Siguiente y Anterior.
        self.__pSig = None
        self.__pAnt = None

    def getElemento(self):
        return self.__elemento
```

Lista Doblemente Enlazada

Lista Doblemente Enlazada

Es un objeto con:

- ✓ Dos atributos o “etiquetas” (su nodo inicial y su nodo final)

Para construirla:

- ✓ Toda lista parte vacía, por ende, su primer nodo debe ser igual al ultimo nodo y debe ser vacíos. En Python se utiliza la palabra reservada “None”

Sus métodos:

- ✓ Método que nos indique si la lista está vacía – ES MUY IMPORTANTE
- ✓ Un método para retornar el primer o el último Nodo de la lista (los get)
- ✓ Dos métodos para imprimir la lista completa (desde el inicio y desde el final)
- ✓ Dos método que ingresen nodos a la lista: Al inicio y al final
- ✓ Dos método que eliminen Nodos de la lista: Eliminar el primero y eliminar el final

¿En qué cambia entonces?

Lista Doblemente Enlazada

Clase Lista Doblemente Enlazada

```
import ClaseNodoDoble
CN = ClaseNodoDoble
#Clase Lista Doblemente Enlazada
class ListaDoble(object):
    def __init__(self):
        self.__primero = None
        self.__ultimo = None

    def getVacio(self):
        if self.__primero == None:
            return True
```

¿Nota algún cambio respecto a la lista simple?

Lista Doblemente Enlazada

Métodos para imprimir

```
def printListaPrimeroUltimo(self):
    if self.getVacio()==True:
        print ("Lista vacia")
    else:
        validar = True
        temp = self.__primero
        while(validar):
            print (temp.getElemento())
            if temp == self.__ultimo:
                validar=False
            else:
                temp = temp.pSig
```

```
def printListaUltimoPrimero(self):
    if self.getVacio()==True:
        print ("Lista vacia")
    else:
        validar = True
        temp = self.__ultimo
        while(validar):
            print (temp.getElemento())
            if temp == self.__primero:
                validar=False
            else:
                temp = temp.pAnt
```

Comparemos ambos métodos para notar las diferencias

Lista Doblemente Enlazada

Métodos para agregar Nodos

```
def setNodoAlInicio(self,elemento):
    nuevo = CN.Nodo(elemento)
    if self.getVacio()==True:
        self.__primero = self.__ultimo = nuevo
    else:
        nuevo.pSig=self.__primero
        self.__primero.pAnt=nuevo
        self.__primero=nuevo
```

```
def setNodoAlFinal(self,elemento):
    nuevo = CN.Nodo(elemento)
    if self.getVacio()==True:
        self.__primero = self.__ultimo = nuevo
    else:
        self.__ultimo.pSig = nuevo
        nuevo.pAnt = self.__ultimo
        self.__ultimo = nuevo
```

Comparemos ambos métodos para notar las diferencias

Lista Doblemente Enlazada

Métodos para eliminar Nodos

```
def eliminarPrimero(self):
    if self.getVacio()==True:
        print ("Lista vacia. Imposible eliminar")
    elif self.__primero == self.__ultimo:
        self.__primero = None
        self.__ultimo = None
        print ("Elemento eliminado. La lista está vacia")
    else:
        temp = self.__primero
        self.__primero = self.__primero.pSig
        self.__primero.pAnt = None
        temp = None
        print ("Elemento eliminado")
```

```
def eliminarUltimo(self):
    if self.getVacio()==True:
        print ("Lista vacia. Imposible eliminar")
    elif self.__primero == self.__ultimo:
        self.__primero = None
        self.__ultimo = None
        print ("Elemento eliminado. La lista está vacia")
    else:
        temp=self.__ultimo
        self.__ultimo=self.__ultimo.pAnt
        self.__ultimo.pSig = None
        temp = None
        print ("Elemento eliminado")
```

Comparemos ambos métodos para notar las diferencias

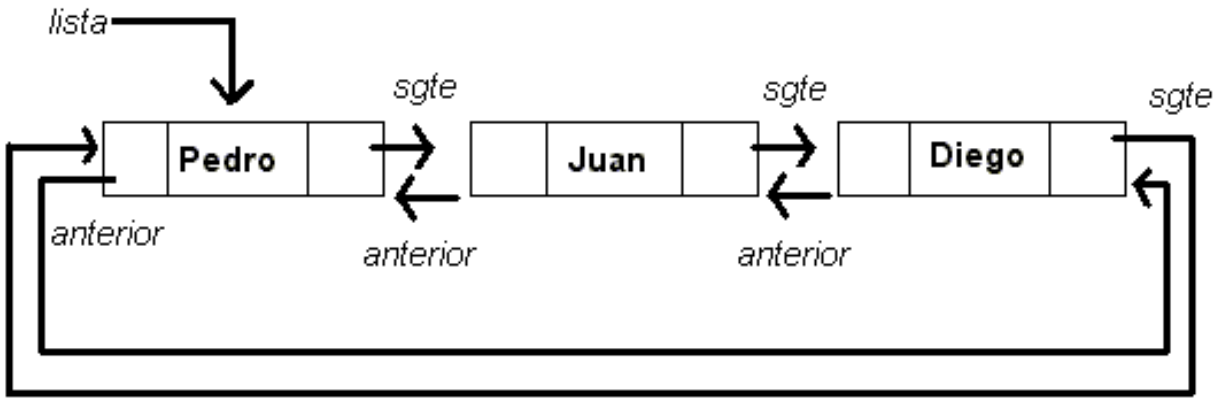
Lista Doblemente Enlazada Circular

¿Qué es una Lista Doblemente Enlazada Circular?

Según Wikipedia

"Es aquella en que cada nodo tiene dos enlaces: uno apunta al nodo anterior, o apunta al último nodo de la lista si es el primer nodo y otro que apunta al nodo siguiente, o apunta al primer nodo de la lista si es el último nodo."

Gráficamente



Lista Doblemente Enlazada Circular

Nodo para lista doblemente enlazada

Es un objeto con

- ✓ Uno o más atributos (sus elementos)
- ✓ Dos atributos adicionales – Puntero siguiente y Puntero Anterior

Para construirlo

- ✓ Los atributos guardan los elementos del nodo
- ✓ El puntero siguiente y el anterior deben ser NULO

Métodos

- ✓ Solo nos interesa rescatar el dato, por ende, tiene solo un “get”
- ✓ No tiene un “set” elemento ya que eso irá, por añadidura, en la inserción de un nuevo nodo a una lista.

Lista Doblemente Enlazada Circular

Clase Nodo

```
#Clase Nodo con dos punteros
class Nodo(object):
    def __init__(self,elemento):
        #Atributo que tendrá el Nodo - Puede tener más
        self.__elemento=elemento
        #Dos punteros que servirán para "unir" los Nodos
        #cuando se construya la lista. Siguiente y Anterior.
        self.__pSig = None
        self.__pAnt = None

    def getElemento(self):
        return self.__elemento
```

Lista Doblemente Enlazada Circular

Lista Doblemente Enlazada Circular

Es un objeto con:

- ✓ Dos atributos o “etiquetas” (su nodo inicial y su nodo final)

Para construirla:

- ✓ Toda lista parte vacía, por ende, su primer nodo debe ser igual al ultimo nodo y debe ser vacíos. En Python se utiliza la palabra reservada “None”

Sus métodos:

- ✓ Método que nos indique si la lista está vacía – ES MUY IMPORTANTE
- ✓ Un método para retornar el primer o el último Nodo de la lista (los get)
- ✓ Dos métodos para imprimir la lista completa
- ✓ Dos método que ingresen elementos (un nuevo nodo) a la lista: Al comienzo y al final
- ✓ Dos método que eliminen Nodos de la lista: Al comienzo y al final

¿En qué cambia si comparamos con la Lista Doblemente Enlazada?

Lista Doblemente Enlazada Circular

Clase Lista Doblemente Enlazada Circular

```
import ClaseNodoDoble
CN = ClaseNodoDoble
#Clase Lista Doblemente Enlazada
class ListaDobleCircular(object):
    def __init__(self):
        self.__primero = None
        self.__ultimo = None

    def getVacio(self):
        if self.__primero == None:
            return True
```

¿Nota algún cambio respecto a la lista doblemente enlazada?

Lista Doblemente Enlazada Circular

Clase Lista Doblemente Enlazada Circular

```
def printListaPrimeroUltimo(self):
    if self.getVacio()==True:
        print ("Lista vacia")
    else:
        validar = True
        temp = self.__primero
        while(validar):
            print (temp.getElemento())
            if temp == self.__ultimo:
                validar=False
            else:
                temp = temp.pSig
```

```
def printListaUltimoPrimero(self):
    if self.getVacio()==True:
        print ("Lista vacia")
    else:
        validar = True
        temp = self.__ultimo
        while(validar):
            print (temp.getElemento())
            if temp == self.__primero:
                validar=False
            else:
                temp = temp.pAnt
```

¿Nota algún cambio respecto a la lista doblemente enlazada?

Lista Doblemente Enlazada Circular

Clase Lista Doblemente Enlazada Circular

```
def setNodoAlInicio(self,elemento):
    nuevo = CN.Nodo(elemento)
    if self.getVacio()==True:
        self.__primero = self.__ultimo = nuevo
        self.__ultimo.pSig = self.__primero
        self.__primero.pAnt = self.__ultimo
    else:
        nuevo.pSig=self.__primero
        self.__primero.pAnt=nuevo
        self.__primero=nuevo
        self.__ultimo.pSig = self.__primero
        self.__primero.pAnt = self.__ultimo
```

¿Nota algún cambio respecto a la lista doblemente enlazada?

Lista Doblemente Enlazada Circular

Clase Lista Doblemente Enlazada Circular

```
def setNodoAlFinal(self,elemento):
    nuevo = CN.Nodo(elemento)
    if self.getVacio()==True:
        self.__primero = self.__ultimo = nuevo
        self.__ultimo.pSig = self.__primero
        self.__primero.pAnt = self.__ultimo
    else:
        self.__ultimo.pSig = nuevo
        nuevo.pAnt = self.__ultimo
        self.__ultimo = nuevo
        self.__ultimo.pSig = self.__primero
        self.__primero.pAnt = self.__ultimo
```

¿Nota algún cambio respecto a la lista doblemente enlazada?

Lista Doblemente Enlazada Circular

Clase Lista Doblemente Enlazada Circular

```
def eliminarPrimero(self):
    if self.getVacio()==True:
        print ("Lista vacia. Imposible eliminar")
    elif self.__primero == self.__ultimo:
        self.__primero = None
        self.__ultimo = None
        print ("Elemento eliminado. La lista está vacia")
    else:
        temp = self.__primero
        self.__primero = self.__primero.pSig
        self.__primero.pAnt = self.__ultimo
        self.__ultimo.pSig = self.__primero
        temp = None
        print ("Elemento eliminado")
```

¿Nota algún cambio respecto a la lista doblemente enlazada?

Lista Doblemente Enlazada Circular

Clase Lista Doblemente Enlazada Circular

```
def eliminarUltimo(self):
    if self.getVacio()==True:
        print ("Lista vacia. Imposible eliminar")
    elif self.__primero == self.__ultimo:
        self.__primero = None
        self.__ultimo = None
        print ("Elemento eliminado. La lista está vacia")
    else:
        temp=self.__ultimo
        self.__ultimo=self.__ultimo.pAnt
        self.__ultimo.pSig = self.__primero
        self.__primero.pAnt = self.__ultimo
        temp = None
        print ("Elemento eliminado")
```

¿Nota algún cambio respecto a la lista doblemente enlazada?

Actividad Propuesta

- ✓ Copie todo el programa de esta clase y ejecútelo para estudiar su funcionamiento. Deberá crear un Main para cada tipo de lista visto este día.

- ✓ Busque la forma de crear un método para agregar o eliminar un nodo en una posición determinada de una lista doblemente enlazada. Si el número donde se desea agregar o eliminar es mayor que el largo de la lista, deberá eliminar el último, si es menor, deberá eliminar el primero.

- ✓ Busque la forma de crear un método que permita, desde una posición "m" determinada de un nodo en la lista, avanzar (a) o retroceder (r) un número "n" de posiciones e imprima el contenido de ambos nodos. Ejemplo:

  listaDoble.imprimeDatosNodosDistanciaN(m,n,a)

  Deberá imprimir los datos del nodo que está a distancia **n** del nodo **m** hacia "**adelante**"

  Recuerde que, siendo una lista del tipo circular, debe "dar la vuelta"