Ontiveros Lara Claudia Sarahi. 17212167

Lista Enlazada Simple

La Lista Enlazada Simple es la más fundamental estructura de datos basada en punteros, y del concepto fundamental de ésta derivan las otras estructuras de datos. La lista enlazada asigna espacio para cada elemento por separado, en su propio bloque de memoria, llamado nodo. La lista conecta estos nodos usando punteros, formando una estructura parecida a la de una cadena.

Nodo: (se comportará como un vagón): es un objeto como cualquier otro, y sus atributos serán los encargados de hacer el trabajo de almacenar y apuntar a otro nodo. Cada nodo tiene dos atributos: un atributo "contenido", usado para almacenar un objeto; y otro atributo "siguiente", usado para hacer referencia al siguiente nodo de la lista. Además, como siempre, implementaremos el constructor y el método __str__ para poder imprimir el contenido del nodo.

```
class Nodo(object):
    def __init__(self, dato=None, prox = None):
        self.dato = dato
        self.prox = prox
    def __str__(self):
        return str(self.dato)
```

Ejecutamos este código:

```
>>> v3=Nodo("Bananas")
>>> v2=Nodo("Peras", v3)
>>> v1=Nodo("Manzanas", v2)
>>> print v1
Manzanas
>>> print v2
Peras
>>> print v3
Bananas
```

Podemos considerar una lista enlazada como una **estructura de datos recursiva**. Una lista enlazada puede ser:

- La lista vacía, representada por None, o bien
- Un nodo que contiene un objeto de carga y una referencia a una lista enlazada.

Lista simplemente enlazada.

Operaciones que inicialmente deberá cumplir la clase ListaEnlazada:

- __str__, para mostrar la lista.
- __len__, para calcular la longitud de la lista.
- append(x), para agregar un elemento al final de la lista.
- insert(i, x), para agregar el elemento x en la posición i (levanta una excepción si la posición i es inválida).
- remove(x), para eliminar la primera aparición de x en la lista (levanta una excepción si x no está).
- pop([i]), para borrar el elemento que está en la posición i y devolver su valor. Si no se especifica el valor de i, pop() elimina y devuelve el elemento que está en el último lugar de la lista (levanta una excepción si se hace referencia a una posición no válida de la lista).
- index(x), devuelve la posición de la primera aparición de x en la lista (levanta una excepción si x no está).

Una característica importante de la implementación de listas enlazadas es que borrar el primer elemento es una operación de tiempo constante, es decir que no depende del largo de la lista, a diferencia de las listas de Python, en las que esta operación requiere un tiempo proporcional a la longitud de la lista). Sin embargo no todo es tan positivo: el acceso a la posición p se realiza en tiempo proporcional a p, mientras que en las listas de Python esta operación se realiza en tiempo constante.

Construcción de la lista:

Empezamos escribiendo la clase con su constructor.

```
class ListaEnlazada(object):
    " Modela una lista enlazada, compuesta de Nodos. "

    def __init__(self):
        """ Crea una lista enlazada vacía. """
        # prim: apuntará al primer nodo - None con la lista vacía
        self.prim = None
        # len: longitud de la lista - 0 con la lista vacía
        self.len = 0
```

Ejemplo de método pop():

```
def pop(self, i = None):
    """ Elimina el nodo de la posición i, y devuelve el dato contenido.
        Si i está fuera de rango, se levanta la excepción IndexError.
        Si no se recibe la posición, devuelve el último elemento. """

    # Verificación de los límites
    if (i < 0) or (i >= self.len):
        raise IndexError("Índice fuera de rango")

    # Si no se recibió i, se devuelve el último.
    if i == None:
        i = self.len - 1

    # Caso particular, si es el primero,
    # hay que saltear la cabecera de la lista
    if i == 0:
        dato = self.prim.dato
        self.prim = self.prim.prox

    # Para todos los demás elementos, busca la posición
    else:
        n_ant = self.prim
        n_act = n_ant.prox
        for pos in xrange(1, i):
            n_ant = n_act
            n_act = n_ant.prox

        # Guarda el dato y elimina el nodo a borrar
        dato = n_act.dato
        n_ant.prox = n_act.prox

    # hay que restar 1 de len
    self.len -= 1
    # y devolver el valor borrado
    return dato
```

Referencias:

Aguilar, J. Luis, Martinez, Z. Ignacio, Azuela, F. Matilde, & Garcia, S. Lucas. (1999). Estructura de datos. Libro de problemas. Madrid, España: Mc Graw Hill.

Cairó, O., & Guardati, S. (2002). Estructuras de datos (3rd ed.). México: McGraw-Hill.

Joyanes Aguilar, L., & Zahonero Martínez, I. (1998). Estructura de datos. Algoritmos, abstracción y objetos. (Ed. rev.). Madrid, España: McGraw-Hill.