Nombre y Apellido: Legajo:

Examen Parcial de Programación 2

Recuperatorio 2do Examen Parcial

Tiempo mínimo para el examen: 1 hora reloj.
Tiempo máximo para el examen: 2 horas reloj.

El examen se compone los siguientes ejercicios.

1. Considere la siguiente clase `Nodo`:

```
class Nodo:
    def __init__(self, dato = None, siguiente = None, anterior = None):
        self.dato = dato
        self.sig = siguiente
        self.ant = anterior
```

Utilícela para implementar una lista doblemente enlazada:

```
class ListaDoblementeEnlazada:
    def __init__(self):
        self.cabeza = None
        self.ultimo = None
        self.largo = 0

    def agregar_al_principio(self, dato):
        pass

    def agregar_al_final(self, dato):
        pass

    def eliminar_al_principio(self):
        pass

    def eliminar_al_final(self):
        pass

    def eliminar_dato(self, dato):
        pass

    def longitud():
        pass
```

Implemente los métodos especificados:

1. `agregar_al_principio`: Agrega un nodo con el dato al principio de la lista.
2. `agregar_al_final`: Agrega un nodo con el dato al final de la lista.
3. `eliminar_al_principio`: elimina el nodo al principio de la lista.
4. `eliminar_al_final`: elimina el nodo al final de la lista.
5. `eliminar_dato`: elimina el primer nodo que encuentra con el dato buscado, arrancando desde el principio, si lo encuentra.
6. `longitud`: Devuelve la cantidad de datos guardados en la lista.

2. Considere la siguiente implementación del TAD Pila:

```
class Pila:
    def __init__(self):
        self.items = []

    def apilar(self, item):
        self.items.append(item)

    def desapilar(self):
        return self.items.pop() if not self.esta_vacia() else None

    def esta_vacia(self):
        return len(self.items) == 0

    def cima(self):
        return self.items[-1] if not self.esta_vacia() else None
```

   Agregue los siguientes métodos a la clase Pila:

   1. `tamanio`: Devuelve la cantidad de elementos en la pila.
   2. `limpia`: Elimina todos los elementos de la pila.
   3. `copiar`: Devuelve una nueva pila que es una copia de la pila actual.

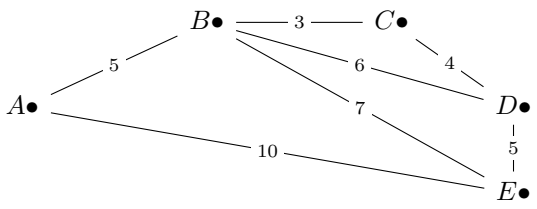
3. Considere la siguiente clase `ArbolBinario`:

```
class ArbolBinario:
    def __init__(self, valor=None, izquierda=None, derecha=None):
        self.valor = valor
        self.izquierda = izquierda
        self.derecha = derecha

    def contar_nodos(arbol: ArbolBinario):
        pass

    def sumar_valores(arbol: ArbolBinario):
        pass
```

   Implemente los métodos especificadas:

   1. `contar_nodos`: Devuelve la cantidad de nodos en el árbol.
   2. `sumar_valores`: Devuelve la suma de todos los valores en el árbol.

4. Dado el siguiente grafo de ciudades y distancias entre ellas:



   Suponga que se quiere encontrar el camino más corto entre la ciudad **A** y la ciudad **D** utilizando el algoritmo de Dijkstra. Describa brevemente los pasos que seguiría para implementar este algoritmo.