

Nombre y Apellido:

Legajo:

## Examen Parcial de Programación 2

### 2do Examen Parcial

Tiempo mínimo para el examen: 1 hora reloj.

Tiempo máximo para el examen: 2 horas reloj.

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 copiar_lista(self) -> 'ListaDoblementeEnlazada':  
        pass  
  
    def dar_vuelta_lista(self) -> None:  
        pass  
  
    def borrar_duplicados(self) -> None:  
        pass  
  
    def extender (self, other: 'ListaDoblementeEnlazada') -> None:  
        pass  
  
    def contar (self, dato: Any) -> int:  
        pass
```

Imagine que ya están implementados los siguientes métodos:

1. `agregar_al_principio`: que agrega un nodo con el dato al principio de la lista.
2. `agregar_al_final`: que agrega un nodo con el dato al final de la lista.
3. `eliminar_al_principio`: que elimina el nodo al principio de la lista.
4. `eliminar_al_final`: que elimina el nodo al final de la lista.
5. `longitud`: que devuelve la cantidad de datos guardados en la lista.

Implemente los siguientes métodos:

1. `copiar_lista`: devuelve otra lista con el mismo contenido
2. `dar_vuelta_lista`: que da vuelta la lista.
3. `borrar_duplicados`: que borra elementos duplicados de la lista.
4. `extender`: que toma otra lista como parámetro y extiende la lista (`self`) con los elementos de la otra lista (`other`).
5. `contar`: que cuenta cuántas veces aparece el dato en la lista.

2. Considere las siguientes clases:

```
class _NodoArbol:
    def __init__(self, cargo: Any = None,
                 izq: '_NodoArbol' = None,
                 der: '_NodoArbol' = None):
        self.cargo = cargo
        self.izq = izq
        self.der = der

class ArbolBinarioDeBusqueda:
    ''' Clase que implementa un árbol binario
    de búsqueda '''
    def __init__(self):
        self.raiz = None

    def agregar_dato(self, dato: Any):
        pass

    def devolver_min(self) -> Any:
        pass

    def devolver_in_order(self) -> list[Any]:
        pass

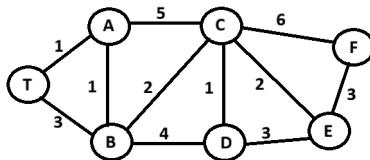
# Ejemplo de uso
arbol = ArbolBinarioDeBusqueda()
datos = [5, 3, 8, 1, 4, 7, 9]

for dato in datos:
    arbol.agregar_dato(dato)

print("Impresión en orden:")
arbol.devolver_in_order() # [1, 3, 4, 5, 7, 8, 9]
```

y complete las implementaciones de los métodos especificados:

1. `agregar_dato` (de la clase `ArbolBinarioDeBusqueda`): que permite agregar un dato en el árbol binario de búsqueda.
  2. `devolver_min` (de la clase `ArbolBinarioDeBusqueda`): que devuelve el menor dato del árbol binario de búsqueda.
  3. `devolver_in_order` (de la clase `ArbolBinarioDeBusqueda`): que devuelve los datos del árbol en una lista, teniendo en cuenta el recorrido In Order.
3. Una empresa quiere construir una red de comunicación.



En la figura anterior los nodos representan las oficinas, las aristas los enlaces entre las diferentes oficinas y los pesos la distancia que abarcaría dichos enlaces. Para minimizar gastos, se pretende construir una red entre todos las oficinas tal que la distancia total de los enlaces sea mínima.

1. Determine una posible red, aplicando el algoritmo de Kruskal.
2. Dado un grafo cualquiera, ¿Hay siempre un único árbol recubridor mínimo en el grafo?