

Universidad Americana

Facultad de Ingeniería y Arquitectura



Algoritmos y Estructuras de Datos, Grupo 4

Manejo de Funciones, Clases y Paquetes

Realizado por:

Joaquín Alberto Pérez Zúñiga

Nicolás Rafael Laguna Vallejos

Solieth Valentina Trejos Pérez

Docente:

MSc. César Marín López

Implementación de la Clase LinkedList()

1. node.py:

```
1  """
2  Implementación de un nodo en una lista enlazada.
3  """
4
5  from typing import (
6      Generic,
7      Optional,
8      TypeVar
9  )
10
11  T = TypeVar("T")
12
13
14  class Node(Generic[T]):
15      """
16      Representa un nodo en una lista enlazada,
17      con el dato que almacena, y el proximo elemento en la lista.
18      """
19
20      def __init__(self, data: T, next_node: Optional["Node"] = None):
21          self.data = data
22          self.next = next_node
23
```

2. linked_list.py:

```
1  """
2  Implementación personalizada de una lista enlazada.
3  """
4
5  from typing import (
6      Generic,
7      Optional,
8      TypeVar
9  )
10
11  from .node import Node
12
13  T = TypeVar("T")
14
15
16  class LinkedList(Generic[T]):
17      """
18      Implementación personalizada de una lista enlazada.
19      """
20
21      def __init__(self) -> None:
22          self.head: Optional[Node] = None
23          self.size = 0
24
25      def __len__(self) -> int:
26          return self.size
27
28      def __repr__(self) -> str:
29          return self.__str__()
30
```

```

16  class LinkedList(Generic[T]):
31      ...def __str__(self) -> str:
32      ...    self_str = "[ "
33
34      ...    current = self.head
35      ...    while current is not None:
36      ...        if current.next is not None:
37      ...            self_str += f"{str(current.data)}, "
38      ...        else:
39      ...            self_str += f"{str(current.data)} ]"
40      ...        current = current.next
41
42      ...    return self_str
43
44      ...def is_empty(self) -> bool:
45      ...    """
46      ...    Determina si la lista esta vacía.
47      ...    """
48
49      ...    return self.head is None
50
51      ...def add_at_front(self, data: T) -> None:
52      ...    """
53      ...    Agrega un nuevo elemento al inicio de la lista enlazada.
54      ...    """
55
56      ...    self.head = Node(data=data, next_node=self.head)
57      ...    self.size += 1
58

```

```

16  class LinkedList(Generic[T]):
59      ...def add_at_end(self, data: T) -> None:
60      ...    """
61      ...    Agrega un nuevo elemento al final de la lista enlazada.
62      ...    """
63
64      ...    new_node = Node(data)
65
66      ...    if self.is_empty():
67      ...        self.head = new_node
68      ...        self.size += 1
69      ...    return
70
71      ...    current = self.head
72      ...    while True:
73      ...        if current.next is not None:
74      ...            current = current.next
75      ...        else:
76      ...            current.next = new_node
77      ...            self.size += 1
78      ...    return
79

```

```

16 class LinkedList(Generic[T]):
80     def update_node(self, index: int, new_data: T) -> None:
81         """
82         Actualiza el valor de un nodo en la lista.
83         """
84
85         if self.is_empty():
86             raise IndexError("Índice inválido: no hay elementos en la lista.")
87         if index < 0:
88             raise IndexError("Índice inválido: debe ser mayor o igual a cero.")
89         if index >= self.size:
90             raise IndexError(
91                 "Índice inválido: es fuera de límites; no corresponde a un nodo."
92             )
93
94         i = 0
95         current = self.head
96         while current is not None and i < index:
97             i += 1
98             current = current.next
99
100         if current is not None:
101             current.data = new_data
102

```

```

16 class LinkedList(Generic[T]):
103     def search(self, data: T) -> Optional[Node]:
108         if self.is_empty():
109             return None
110
111         current = self.head
112         while current is not None:
113             if current.data == data:
114                 return current
115             current = current.next
116
117         return None
118
119     def delete(self, data: T, count: int = 1) -> None:
120         """
121         Elimina nodos con el valor especificado.
122         """
123
124         if self.is_empty():
125             return
126         if self.size == 1:
127             self.head = None
128
129         deletions = 0
130         previous = None
131         current = self.head
132
133         while current is not None and deletions < count:
134             if current.data == data:
135                 if previous is None:
136                     self.head = current.next
137                 else:
138                     previous.next = current.next
139                 deletions += 1
140
141             previous = current
142             current = current.next

```

```

16     class LinkedList(Generic[T]):
144         ... def delete_first(self) -> None:
145             ...     """
146             ...     Elimina el primer nodo de la lista.
147             ...     """
148
149             ...     if self.is_empty():
150                 ...         return
151
152             ...     if self.head.next is None:
153                 ...         self.head = None
154             ...     else:
155                 ...         self.head = self.head.next
156
157         ... def delete_last(self) -> None:
158             ...     """
159             ...     Elimina el último nodo de la lista.
160             ...     """
161
162             ...     if self.is_empty():
163                 ...         return
164             ...     if self.size == 1:
165                 ...         self.delete_first()
166                 ...         return
167
168             ...     previous = None
169             ...     current = self.head
170
171             ...     while True:
172                 ...         if current.next is None:
173                     ...             previous.next = None
174                     ...             return
175                 ...             previous = current
176                 ...             current = current.next
177

```

Ejercicio 1

1. estudiante.py:

```
1 """
2 Implementación de la clase Estudiante.
3 """
4
5 class Estudiante:
6     """
7     Representa los datos de un estudiante.
8     """
9
10    def __init__(
11        self,
12        carnet: str,
13        nombres: tuple[str, str],
14        apellidos: tuple[str, str],
15        genero: str,
16        peso: float,
17        estatura: float,
18        promedio: float
19    ) -> None:
20        self.carnet = carnet
21        self.nombres = nombres
22        self.apellidos = apellidos
23        self.genero = genero
24        self.peso = peso
25        self.estatura = estatura
26        self.promedio = promedio
27
```

```
1 """
2 Una escuela de educación primaria requiere un algoritmo que muestre los datos de los
3 estudiantes de un salón de clase ordenados de forma ascendente, según un parámetro
4 indicado; este parámetro puede ser cualquiera de los siguientes campos:
5 carnet, nombres, apellidos, genero, peso, estatura, promedio.
6 """
7
8 from typing import Union
9
10 from . import Estudiante, LinkedList
11
12
13 def input_estudiantes(lista: LinkedList[Estudiante]) -> None:
14     """
15     Guarda los datos de estudiantes ingresados en una lista enlazada.
16     """
17
18     i = 1
19
20     while True:
21         print("\nPara terminar, dejar todos los campos vacíos.\n")
22
23         try:
24             carnet = input(f"Ingrese el carnet del estudiante #{i}: ")
25             nombres = input(f"Ingrese los nombres del estudiante #{i}: ")
26             apellidos = input(f"Ingrese los apellidos del estudiante #{i}: ")
27             genero = input(f"Ingrese el género del estudiante #{i}: ")
28
29             peso = input(f"Ingrese el peso del estudiante #{i} (en kg): ")
30             if peso != "" and float(peso) <= 0:
31                 raise ValueError
32
33             estatura = input(f"Ingrese la estatura del estudiante #{i} (en m): ")
34             if peso != "" and float(estatura) <= 0:
35                 raise ValueError
36
```

2. main.py:


```

13 def input_estudiantes(lista: LinkedList[Estudiante]) -> None:
14     .....
15     ..... estatura = input(f"Ingresa la estatura del estudiante #{i} (en m): ")
16     ..... if peso != "" and float(estatura) <= 0:
17     .....     raise ValueError
18     .....
19     ..... promedio = input(f"Ingresa el promedio del estudiante #{i}: ")
20     ..... if peso != "" and float(promedio) <= 0:
21     .....     raise ValueError
22     .....
23     ..... except (TypeError, ValueError):
24     .....     input(
25     .....         f"\n¡Error! El peso, promedio y estatura del estudiante deben"
26     .....         f"+ " ser un número real positivo, intente nuevamente..."
27     .....     )
28     .....
29     ..... continue
30     .....
31     ..... if all(
32     .....     x == ""
33     .....     for x in (carnet, nombres, apellidos, genero, peso, estatura, promedio)
34     ..... ):
35     .....     break
36     .....
37     ..... i += 1
38     ..... lista.add_at_end(Estudiante(
39     .....     carnet=carnet,
40     .....     nombres=tuple(nombres.split()),
41     .....     apellidos=tuple(apellidos.split()),
42     .....     genero=genero,
43     .....     peso=float(peso),
44     .....     estatura=float(estatura),
45     .....     promedio=float(promedio)
46     ..... ))
47     .....
48     ..... print()
49     .....
50     .....
51     .....
52     .....
53     .....
54     .....
55     .....
56     .....
57     .....
58     .....
59     .....
60     .....
61     .....
62     .....
63     .....
64     .....
65     .....
66     .....
67     .....
68     .....

```

```

69 def print_ordenado(lista: LinkedList[Estudiante], filtro: int) -> None:
70     .....
71     ..... lista_filtrada: list[tuple[str, Union[str, float]]] = []
72     .....
73     ..... # hay que extraer los nombres de todos los estudiantes de la lista
74     ..... # y emparejarlos con el dato que se desea visualizar y ordenar
75     .....
76     ..... current = lista.head
77     ..... while current is not None:
78     .....     ..... # construir un string con el primer nombre y primer apellido
79     .....     ..... key = f"{current.data.nombres[0]} {current.data.apellidos[0]}"
80     .....     .....
81     .....     ..... # extraer el atributo correspondiente al filtro especificado
82     .....     ..... match filtro:
83     .....     .....     case 1:
84     .....     .....         atributo = "Carnet"
85     .....     .....         value = current.data.carnet
86     .....     .....     case 2:
87     .....     .....         atributo = "Nombres"
88     .....     .....         value = current.data.nombres
89     .....     .....     case 3:
90     .....     .....         atributo = "Apellidos"
91     .....     .....         value = current.data.apellidos
92     .....     .....     case 4:
93     .....     .....         atributo = "Género"
94     .....     .....         value = current.data.genero
95     .....     .....     case 5:
96     .....     .....         atributo = "Peso"
97     .....     .....         value = current.data.peso
98     .....     .....     case 6:
99     .....     .....         atributo = "Estatura"
100     .....     .....         value = current.data.estatura
101     .....     .....     case 7:
102     .....     .....         atributo = "Promedio"
103     .....     .....         value = current.data.promedio
104     .....     .....
105     .....     ..... # guardar la entrada en la lista
106     .....     ..... lista_filtrada.append((key, value))
107     .....     ..... current = current.next
108     .....
109     .....
110     .....
111     .....

```

```

69  def print_ordenado(lista: LinkedList[Estudiante], filtro: int) -> None:
109  |    ...# guardar la entrada en la lista
110  |    ...lista_filtrada.append((key, value))
111  |    ...current = current.next
112  |
113  |    ...# ordenar la lista de tuplas por el segundo elemento (el atributo del objeto)
114  |    ...lista_filtrada.sort(key=lambda x: x[1])
115  |
116  |    ...print()
117  |    ...for i, estudiante in enumerate(lista_filtrada):
118  |    |    ...print(f"{i + 1}. {atributo} de {estudiante[0]}: {estudiante[1]}")
119  |    ...print()
120  |
121  |
122  def main() -> None:
123  |    """
124  |    Ejecución del programa.
125  |    """
126  |
127  |    ...lista: LinkedList[Estudiante] = LinkedList()
128  |    ...print("\nPrimeramente, debe ingresar los datos de los estudiantes de la clase:")
129  |    ...print("-----")
130  |
131  |    ...input_estudiantes(lista)
132  |    ...print("\nAhora, seleccione el campo por el cual desea ordenar los datos:")
133  |    ...print("1. Carnet\n2. Nombres\n3. Apellidos\n4. Genero\n5. Peso\n6. Estatura\n7. Promedio")
134  |
135  |    ...seleccion = input("=> ")
136  |
137  |    ...if seleccion.isnumeric():
138  |    |    ...print_ordenado(lista, int(seleccion))
139  |    ...else:
140  |    |    ...input("\n¡Error! No seleccionó una opción válida, intente nuevamente...\n")
141  |
142  |
143  |
144  if __name__ == "__main__":
145  |    ...main()

```

Ejercicio 2

1. ruta.py:

```
6 class Ruta:
7     """
8     ... Representa una ruta en un mapa.
9     """
10
11     def __init__(self):
12         """# Diccionario que representa el grafo de estaciones
13         ...# Cada estación tiene una lista de tuplas (estación_destino, tiempo_minutos)
14         ...self.estaciones = {}
15
16         ...# Inicializar el mapa con algunas estaciones de ejemplo
17         ...# En una aplicación real, esto podría cargarse desde un archivo o base de datos
18         ...self.inicializar_mapa_ejemplo()
19
20     def inicializar_mapa_ejemplo(self):
21         """
22         ...Inicializa un mapa de ejemplo con estaciones y tiempos entre ellas.
23         """
24
25         ...# Formato: "estación": [(estación_destino, tiempo_en_minutos), ...]
26         ...self.estaciones = {
27             ..."Terminal Central": [("Plaza Mayor", 10), ("Parque Industrial", 15)],
28             ..."Plaza Mayor": [
29                 ...("Terminal Central", 10),
30                 ...("Universidad", 8),
31                 ...("Centro Comercial", 5),
32             ...],
33             ..."Parque Industrial": [("Terminal Central", 15), ("Zona Residencial", 12)],
34             ..."Universidad": [("Plaza Mayor", 8), ("Hospital", 6)],
35             ..."Centro Comercial": [
36                 ...("Plaza Mayor", 5),
37                 ...("Zona Residencial", 7),
38                 ...("Hospital", 9),
39             ...],
40             ..."Zona Residencial": [("Parque Industrial", 12), ("Centro Comercial", 7)],
41             ..."Hospital": [("Universidad", 6), ("Centro Comercial", 9)],
42         ...}
43
```

```
6 class Ruta:
44     def agregar_estacion(self, nombre):
45         """
46         ...Agrega una nueva estación al mapa.
47         """
48
49         ...if nombre not in self.estaciones:
50             ...self.estaciones[nombre] = []
51             ...return True
52         ...return False
53
54     def conectar_estaciones(self, origen, destino, tiempo):
55         """
56         ...Conecta dos estaciones con un tiempo determinado en minutos.
57         """
58
59         ...if origen in self.estaciones and destino in self.estaciones:
60             ...# Verificar si ya existe la conexión
61             ...for i, (est, _) in enumerate(self.estaciones[origen]):
62                 ...if est == destino:
63                     ...# Actualizar tiempo si ya existe la conexión
64                     ...self.estaciones[origen][i] = (destino, tiempo)
65                     ...return True
66
67             ...# Agregar nueva conexión si no existe
68             ...self.estaciones[origen].append((destino, tiempo))
69             ...return True
70         ...return False
71
```

```

6 class Ruta:
72     def obtener_conexiones(self):
73         """
74         Devuelve todas las conexiones del mapa.
75         """
76
77         conexiones = []
78         for origen, destinos in self.estaciones.items():
79             for destino, tiempo in destinos:
80                 conexiones.append((origen, destino, tiempo))
81         return conexiones
82
83     def calcular_ruta_mas_rapida(self, inicio, fin):
84         """
85         Implementa el algoritmo de Dijkstra para encontrar la ruta más rápida.
86         """
87
88         if inicio not in self.estaciones or fin not in self.estaciones:
89             return None, None
90
91         # Inicializar distancias con valores infinitos
92         tiempos = {estacion: float("infinity") for estacion in self.estaciones}
93         tiempos[inicio] = 0
94
95         # Diccionario para almacenar el predecesor de cada estación en la ruta más corta
96         predecesores = {estacion: None for estacion in self.estaciones}
97
98         # Conjunto de nodos no visitados
99         no_visitados = set(self.estaciones.keys())
100
101         while no_visitados:
102             # Encontrar la estación no visitada con el tiempo mínimo actual
103             actual = min(no_visitados, key=lambda x: tiempos[x])
104
105             # Si llegamos al destino o si el tiempo es infinito (no hay ruta)
106             if actual == fin or tiempos[actual] == float("infinity"):
107                 break
108

```

```

6 class Ruta:
83     def calcular_ruta_mas_rapida(self, inicio, fin):
105         # Si llegamos al destino o si el tiempo es infinito (no hay ruta)
106         if actual == fin or tiempos[actual] == float("infinity"):
107             break
108
109         no_visitados.remove(actual)
110
111         # Revisar vecinos de la estación actual
112         for vecino, tiempo in self.estaciones[actual]:
113             # Calcular nuevo tiempo a través de esta ruta
114             nuevo_tiempo = tiempos[actual] + tiempo
115
116             # Si encontramos una ruta más rápida, actualizamos
117             if nuevo_tiempo < tiempos[vecino]:
118                 tiempos[vecino] = nuevo_tiempo
119                 predecesores[vecino] = actual
120
121         # Reconstruir la ruta
122         if tiempos[fin] == float("infinity"):
123             return None, None # No hay ruta
124
125         ruta = []
126         actual = fin
127         while actual:
128             ruta.insert(0, actual)
129             actual = predecesores[actual]
130
131         return ruta, tiempos[fin]
132
133     def mostrar_estaciones(self):
134         """
135         Muestra todas las estaciones disponibles.
136         """
137
138         return list(self.estaciones.keys())
139

```

2. main.py:

```
1  """
2  Se requiere automatizar un mapa que contiene las estaciones de una
3  ruta previamente establecida para una aplicación que indique, a
4  partir de un punto de la ruta, el tiempo estimado para llegar
5  a un destino determinado de la misma.
6  """
7
8  from . import Ruta
9
10
11 def main():
12     """
13     Ejecución del programa.
14     """
15
16     # Crear el mapa de la ruta
17     mapa = Ruta()
18
19     while True:
20         print("\n=== SISTEMA DE CÁLCULO DE TIEMPO DE VIAJE ===")
21         print("1. Ver estaciones disponibles")
22         print("2. Agregar nueva estación")
23         print("3. Conectar estaciones")
24         print("4. Ver todas las conexiones")
25         print("5. Calcular tiempo de viaje")
26         print("6. Salir")
27
28         opcion = input("\nSeleccione una opción (1-6): ")
29
30         if opcion == "1":
31             estaciones = mapa.mostrar_estaciones()
32             print("\nEstaciones disponibles:")
33             for i, estacion in enumerate(estaciones, 1):
34                 print(f"{i}. {estacion}")
35
36         elif opcion == "2":
```

```

11 def main():
36     ..... elif opcion == "2":
37         ..... nombre = input("Ingrese el nombre de la nueva estación: ")
38         ..... if mapa.agregar_estacion(nombre):
39             ..... print(f"Estación '{nombre}' agregada correctamente.")
40         ..... else:
41             ..... print(f"La estación '{nombre}' ya existe.")
42
43     ..... elif opcion == "3":
44         ..... estaciones = mapa.mostrar_estaciones()
45         ..... print("\nEstaciones disponibles:")
46         ..... for i, estacion in enumerate(estaciones, 1):
47             ..... print(f"{i}. {estacion}")
48
49         ..... try:
50             ..... idx_origen = (
51             ..... int(input("\nSeleccione el número de la estación de origen: ")) - 1
52             ..... )
53
54             ..... idx_destino = (
55             ..... int(input("Seleccione el número de la estación de destino: ")) - 1
56             ..... )
57
58             ..... tiempo = int(
59             ..... input("Ingrese el tiempo en minutos entre las estaciones: ")
60             ..... )
61
62             ..... if (
63             ..... 0 <= idx_origen < len(estaciones)
64             ..... and
65             ..... 0 <= idx_destino < len(estaciones)
66             ..... ):
67                 ..... origen = estaciones[idx_origen]
68                 ..... destino = estaciones[idx_destino]
69

```

```

11 def main():
62     ..... if (
63     ..... 0 <= idx_origen < len(estaciones)
64     ..... and
65     ..... 0 <= idx_destino < len(estaciones)
66     ..... ):
67         ..... origen = estaciones[idx_origen]
68         ..... destino = estaciones[idx_destino]
69
70     ..... if mapa.conectar_estaciones(origen, destino, tiempo):
71         ..... print(
72         ..... f"Conexión de {origen} a {destino} ({tiempo} minutos) establecida."
73         ..... )
74     ..... else:
75         ..... print("No se pudo establecer la conexión.")
76     ..... else:
77         ..... print("Selección de estación inválida.")
78     ..... except ValueError:
79         ..... print("Entrada inválida. Ingrese números válidos.")
80
81     ..... elif opcion == "4":
82         ..... conexiones = mapa.obtener_conexiones()
83         ..... print("\nConexiones actuales:")
84         ..... for i, (origen, destino, tiempo) in enumerate(conexiones, 1):
85             ..... print(f"{i}. {origen} → {destino}: {tiempo} minutos")
86
87     ..... elif opcion == "5":
88         ..... estaciones = mapa.mostrar_estaciones()
89         ..... print("\nEstaciones disponibles:")
90         ..... for i, estacion in enumerate(estaciones, 1):
91             ..... print(f"{i}. {estacion}")
92
93     ..... try:
94         ..... idx_inicio = (
95         ..... int(input("\nSeleccione el número de la estación de inicio: ")) - 1
96         ..... )
97

```

```

11 def main():
12     try:
13         idx_inicio = (
14             int(input("\nSeleccione el número de la estación de inicio: ")) - 1
15         )
16
17         idx_fin = (
18             int(input("Seleccione el número de la estación de destino: ")) - 1
19         )
20
21         if 0 <= idx_inicio < len(estaciones) and 0 <= idx_fin < len(estaciones):
22             inicio = estaciones[idx_inicio]
23             fin = estaciones[idx_fin]
24
25             ruta, tiempo_total = mapa.calcular_ruta_mas_rapida(inicio, fin)
26
27             if ruta:
28                 print(f"\nRuta más rápida de {inicio} a {fin}:")
29                 print(" → ".join(ruta))
30                 print(f"Tiempo total estimado: {tiempo_total} minutos")
31             else:
32                 print(f"No se encontró una ruta de {inicio} a {fin}.")
33             else:
34                 print("Selección de estación inválida.")
35         except ValueError:
36             print("Entrada inválida. Ingrese números válidos.")
37
38     elif opcion == "6":
39         print("¡Gracias por usar el sistema de cálculo de tiempo de viaje!")
40         break
41
42     else:
43         print("Opción inválida. Por favor, seleccione una opción del 1 al 6.")
44
45 if __name__ == "__main__":
46     main()
47

```


Ejercicio 3

1. paciente.py:

```
1  """
2  Implementación de la clase Paciente.
3  """
4
5  class Paciente:
6      """
7      Representa los datos de un paciente médico.
8      """
9
10     def __init__(
11         self,
12         nombre: str,
13         edad: int,
14         sintoma: str,
15         prioridad: int
16     ) -> None:
17         self.nombre = nombre
18         self.edad = edad
19         self.sintoma = sintoma
20         self.prioridad = prioridad
21
```

2. main.py:

```
1  """
2  Una clínica recibe pacientes en orden de llegada. Cada paciente debe ser ingresado al
3  sistema con los siguientes datos: nombre completo, edad, síntoma principal y prioridad
4  (de 1 a 5). El sistema debe permitir insertar nuevos pacientes, recorrer la lista
5  para mostrar el orden de atención, y eliminar a un paciente una vez atendido.
6  """
7
8  from os import system
9
10  from . import LinkedList, Paciente
11
12
13  def agregar_paciente(lista: LinkedList[Paciente]) -> None:
14      """
15      Pide los datos del nuevo paciente y lo agrega a la lista.
16      """
17
18      print(f"\nPaciente {len(lista) + 1}:\n")
19
20      try:
21          nombre = input("Nombre: ")
22          edad = int(input("Edad: "))
23          if edad < 0:
24              raise ValueError
25
26          sintoma = input("Síntoma principal: ")
27          prioridad = int(input("Prioridad (1-5): "))
28          if prioridad < 1 or prioridad > 5:
29              raise ValueError
30
31      except (TypeError, ValueError):
32          input(
33              "\nError! La edad debe ser mayor a cero, y la prioridad"
34              + " debe ser un número entre 1 y 5, intente nuevamente..."
35          )
36
37      agregar_paciente(lista)
```

```

13 def agregar_paciente(lista: LinkedList[Paciente]) -> None:
14     """
15     ...
16     """
17     lista.add_at_end(Paciente(
18         nombre,
19         edad,
20         sintoma,
21         prioridad
22     ))
23
24
25 def consultar_pacientes(lista: LinkedList[Paciente]) -> None:
26     """
27     ...
28     Muestra todos los pacientes en la lista, ordenados por su prioridad.
29     ...
30     """
31
32     lista_filtrada: list[tuple[str, int]] = []
33
34     current = lista.head
35     while current is not None:
36         lista_filtrada.append((current.data.nombre, current.data.prioridad))
37         current = current.next
38
39     # ordenar la lista de tuplas por el segundo elemento (el atributo del objeto)
40     lista_filtrada.sort(key=lambda x: x[1])
41
42     print()
43     for i, paciente in enumerate(lista_filtrada):
44         print(f"{i + 1}. Prioridad de {paciente[0]}: {paciente[1]}")
45     print()
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67

```

```

68 def eliminar_paciente(lista: LinkedList[Paciente]) -> None:
69     """
70     ...
71     Pide el nombre del paciente a eliminar despues de ser atendido.
72     ...
73     """
74
75     nombre = input("\nIngrese el nombre del paciente a eliminar: ")
76
77     previous = None
78     current = lista.head
79
80     while current is not None:
81         if current.data.nombre == nombre:
82             if previous is None:
83                 lista.head = current.next
84             else:
85                 previous.next = current.next
86             previous = current
87             current = current.next
88
89
90 def menu_principal(lista: LinkedList[Paciente]) -> None:
91     """
92     ...
93     Loop principal del programa que le muestra un menú de opciones al usuario.
94     ...
95     """
96
97     opcion = ""
98     while opcion != "4":
99         system("cls || clear")
100         print("\nGestión de Pacientes")
101         print("-----\n")
102         print("1. Agregar paciente")
103         print("2. Consultar pacientes")
104         print("3. Eliminar paciente")
105         print("4. Salir")
106
107         opcion = input("\n-> ")
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

88  def menu_principal(lista: LinkedList[Paciente]) -> None:
105      .... match opcion:
106          .... case "1":
107              .... system("cls || clear")
108              .... agregar_paciente(lista)
109          .... case "2":
110              .... system("cls || clear")
111              .... consultar_pacientes(lista)
112              .... input("\nPresione 'Enter' para regresar al menú principal...")
113          .... case "3":
114              .... system("cls || clear")
115              .... eliminar_paciente(lista)
116          .... case "4":
117              .... print("\nSaliendo del programa...\n")
118          .... case _:
119              .... input("\n¡Opción inválida, intente nuevamente!")
120
121
122  def main() -> None:
123      .... """
124      .... Ejecución del programa.
125      .... """
126
127      .... menu_principal(LinkedList())
128
129
130  if __name__ == "__main__":
131      .... main()
132

```

Ejercicio 4

1. acciones_editor.py:

```
1  """
2  Implementa las clases que representan las acciones permitidas en la aplicación.
3  """
4
5
6  class Accion:
7      """
8      Clase base que representa una acción genérica en el editor.
9      """
10
11     def __init__(self, descripcion):
12         self.descripcion = descripcion
13
14     def ejecutar(self, editor):
15         """
16         Método a implementar en las clases derivadas.
17         """
18
19     def deshacer(self, editor):
20         """
21         Método a implementar en las clases derivadas.
22         """
23
24
```

```
25 class EscribirTexto(Accion):
26     """
27     Acción de escribir texto en una posición específica.
28     """
29
30     def __init__(self, posicion, texto):
31         super().__init__(f"Escribir '{texto}' en posición {posicion}")
32         self.posicion = posicion
33         self.texto = texto
34
35     def ejecutar(self, editor):
36         editor.texto = (
37             editor.texto[: self.posicion] + self.texto + editor.texto[self.posicion :]
38         )
39         editor.posicion_cursor = self.posicion + len(self.texto)
40
41     def deshacer(self, editor):
42         # Eliminar el texto que se escribió
43         editor.texto = (
44             editor.texto[: self.posicion]
45             + editor.texto[self.posicion + len(self.texto) :]
46         )
47         editor.posicion_cursor = self.posicion
48
```

```

50 class BorrarTexto(Accion):
51     """
52     ...Acción de borrar texto desde una posición específica.
53     ..."""
54
55     def __init__(self, posicion, longitud):
56         super().__init__(f"Borrar {longitud} caracteres desde posición {posicion}")
57         self.posicion = posicion
58         self.longitud = longitud
59         self.texto_borrado = None # Se guardará al ejecutar
60
61     def ejecutar(self, editor):
62         # Guardar el texto que se va a borrar para poder recuperarlo al deshacer
63         self.texto_borrado = editor.texto[self.posicion : self.posicion + self.longitud]
64         editor.texto = (
65             editor.texto[: self.posicion]
66             + editor.texto[self.posicion + self.longitud :]
67         )
68         editor.posicion_cursor = self.posicion
69
70     def deshacer(self, editor):
71         # Restaurar el texto borrado
72         editor.texto = (
73             editor.texto[: self.posicion]
74             + self.texto_borrado
75             + editor.texto[self.posicion :]
76         )
77         editor.posicion_cursor = self.posicion + len(self.texto_borrado)
78

```

```

80 class CopiarTexto(Accion):
81     """
82     ...Acción de copiar texto al portapapeles.
83     ..."""
84
85     def __init__(self, posicion, longitud):
86         super().__init__(f"Copiar {longitud} caracteres desde posición {posicion}")
87         self.posicion = posicion
88         self.longitud = longitud
89         self.texto_anterior_portapapeles = None # Se guardará al ejecutar
90
91     def ejecutar(self, editor):
92         # Guardar el contenido anterior del portapapeles
93         self.texto_anterior_portapapeles = editor.portapapeles
94         editor.portapapeles = editor.texto[
95             self.posicion : self.posicion + self.longitud
96         ]
97
98     def deshacer(self, editor):
99         # Restaurar el portapapeles anterior
100         editor.portapapeles = self.texto_anterior_portapapeles
101

```

```

103 class PegarTexto(Accion):
104     """
105     ...Acción de pegar texto desde el portapapeles.
106     ..."""
107
108     def __init__(self, posicion):
109         super().__init__(f"Pegar texto en posición {posicion}")
110         self.posicion = posicion
111
112     def ejecutar(self, editor):
113         if editor.portapapeles:
114             editor.texto = (
115                 editor.texto[: self.posicion]
116                 + editor.portapapeles
117                 + editor.texto[self.posicion :]
118             )
119             editor.posicion_cursor = self.posicion + len(editor.portapapeles)
120
121     def deshacer(self, editor):
122         if editor.portapapeles:
123             # Eliminar el texto que se pegó
124             editor.texto = (
125                 editor.texto[: self.posicion]
126                 + editor.texto[self.posicion + len(editor.portapapeles) :]
127             )
128             editor.posicion_cursor = self.posicion
129

```

```

1  """
2  Implementación de la clase principal de la aplicación.
3  """
4
5  from .acciones_editor import (
6      ...EscribirTexto,
7      ...BorrarTexto,
8      ...CopiarTexto,
9      ...PegarTexto
10 )
11
12
13 class EditorTexto:
14     """
15     ...Clase principal que simula un editor de texto con historial de acciones.
16     ..."""
17
18     def __init__(self):
19         self.texto = ""
20         self.posicion_cursor = 0
21         self.portapapeles = ""
22         self.historial = [] # Lista de acciones realizadas
23         self.posicion_actual = -1 # Índice de la última acción ejecutada (-1 significa ninguna)
24
25     def ejecutar_accion(self, accion):
26         """
27         ...Ejecuta una nueva acción y la añade al historial.
28         ..."""
29
30         # Si hay acciones en el historial después de la posición actual, se eliminan
31         if self.posicion_actual < len(self.historial) - 1:
32             self.historial = self.historial[:self.posicion_actual + 1]
33
34         # Ejecutar la acción
35         accion.ejecutar(self)
36

```

2. e
d
i
t
o
r.
p
y
:


```

13 class EditorTexto:
24     def ejecutar_accion(self, accion):
25         .....# Añadir al historial
26         .....self.historial.append(accion)
27         .....self.posicion_actual += 1
28
29     def deshacer(self):
30         ....."""
31         .....Deshace la última acción ejecutada.
32         ....."""
33
34     if self.posicion_actual >= 0:
35         .....# Deshacer la acción actual
36         .....self.historial[self.posicion_actual].deshacer(self)
37         .....self.posicion_actual -= 1
38         .....return True
39     .....return False
40
41     def rehacer(self):
42         ....."""
43         .....Rehace la siguiente acción en el historial.
44         ....."""
45
46     if self.posicion_actual < len(self.historial) - 1:
47         .....# Avanzar a la siguiente acción
48         .....self.posicion_actual += 1
49         .....# Ejecutar esa acción
50         .....self.historial[self.posicion_actual].ejecutar(self)
51         .....return True
52     .....return False
53
54
55
56
57
58
59
60
61
62
63
64
65

```

```

13 class EditorTexto:
66     def escribir(self, texto):
67         ....."""
68         .....Método para escribir texto en la posición actual del cursor.
69         ....."""
70
71     accion = EscribirTexto(self.posicion_cursor, texto)
72     self.ejecutar_accion(accion)
73
74     def borrar(self, longitud):
75         ....."""
76         .....Método para borrar texto desde la posición actual del cursor.
77         ....."""
78
79     if self.posicion_cursor - longitud >= 0:
80         .....accion = BorrarTexto(self.posicion_cursor - longitud, longitud)
81         .....self.ejecutar_accion(accion)
82
83     def copiar(self, longitud):
84         ....."""
85         .....Método para copiar texto desde la posición actual del cursor.
86         ....."""
87
88     if self.posicion_cursor + longitud <= len(self.texto):
89         .....accion = CopiarTexto(self.posicion_cursor, longitud)
90         .....self.ejecutar_accion(accion)
91
92     def pegar(self):
93         ....."""
94         .....Método para pegar el texto del portapapeles en la posición actual del cursor.
95         ....."""
96
97     if self.portapapeles:
98         .....accion = PegarTexto(self.posicion_cursor)
99         .....self.ejecutar_accion(accion)
100

```

```

13  class EditorTexto:
101  ... def mostrar_historial(self):
102  ...     """
103  ...     Muestra el historial de acciones.
104  ...     """
105
106  ...     print("\nHistorial de acciones:")
107  ...     for i, accion in enumerate(self.historial):
108  ...         marcador = "→" if i == self.posicion_actual else " "
109  ...         print(f"{marcador} {i+1}. {accion.descripcion}")
110
111  ... def mostrar_estado(self):
112  ...     """
113  ...     Muestra el estado actual del editor.
114  ...     """
115
116  ...     print("\nEditor de Texto")
117  ...     print("=====")
118  ...     print(f"Texto: '{self.texto}'")
119  ...     print(f"Posición del cursor: {self.posicion_cursor}")
120  ...     print(f"Portapapeles: '{self.portapapeles}'")
121
122  ...     # Mostrar posición del cursor visualmente
123  ...     print("\nRepresentación visual:")
124  ...     cursor_visual = " " * self.posicion_cursor + "^"
125  ...     print(self.texto)
126  ...     print(cursor_visual)
127

```

3. main.py:

```
1  """
2  Se desea implementar el historial de acciones realizadas por un usuario en un
3  editor de texto (como escribir, borrar, pegar, copiar). Cada acción debe
4  guardarse en orden y poder recorrerlas en ambas direcciones,
5  simulando las acciones de Deshacer y Rehacer.
6  """
7
8  from . import EditorTexto
9
10
11  def menu_principal():
12      """
13      Función para mostrar el menú principal.
14      """
15
16      print("\n=== EDITOR DE TEXTO ===")
17      print("1. Escribir texto")
18      print("2. Borrar texto")
19      print("3. Copiar texto")
20      print("4. Pegar texto")
21      print("5. Deshacer")
22      print("6. Rehacer")
23      print("7. Mover cursor")
24      print("8. Ver historial")
25      print("9. Salir")
26      opcion = input("Seleccione una opción: ")
27      return opcion
28
```

```
30  def main():
31      """
32      Ejecución del programa.
33      """
34
35      editor = EditorTexto()
36
37      while True:
38          editor.mostrar_estado()
39          opcion = menu_principal()
40
41          if opcion == "1": # Escribir
42              texto = input("Texto a escribir: ")
43              editor.escribir(texto)
44
45          elif opcion == "2": # Borrar
46              try:
47                  longitud = int(input("Número de caracteres a borrar: "))
48                  editor.borrar(longitud)
49              except ValueError:
50                  print("Por favor, ingrese un número válido.")
51
52          elif opcion == "3": # Copiar
53              try:
54                  longitud = int(input("Número de caracteres a copiar: "))
55                  editor.copiar(longitud)
56              except ValueError:
57                  print("Por favor, ingrese un número válido.")
58
59          elif opcion == "4": # Pegar
60              editor.pegar()
61
62          elif opcion == "5": # Deshacer
63              if not editor.deshacer():
64                  print("No hay más acciones para deshacer.")
65
```

```

30 def main():
66     elif opcion == "6": # Rehacer
67         if not editor.rehacer():
68             print("No hay más acciones para rehacer.")
69
70     elif opcion == "7": # Mover cursor
71         try:
72             nueva_posicion = int(
73                 input(f"Nueva posición del cursor (0-{len(editor.texto)}): ")
74             )
75             if 0 <= nueva_posicion <= len(editor.texto):
76                 editor.posicion_cursor = nueva_posicion
77             else:
78                 print("Posición fuera de rango.")
79         except ValueError:
80             print("Por favor, ingrese un número válido.")
81
82     elif opcion == "8": # Ver historial
83         editor.mostrar_historial()
84
85     elif opcion == "9": # Salir
86         print("¡Hasta luego!")
87         break
88
89     else:
90         print("Opción no válida. Intente de nuevo.")
91
92
93 if __name__ == "__main__":
94     main()
95

```