

Q1\_1:

1. Kiểm tra điều kiện ban đầu:

if name.startswith("B") or price > 100:

return

Nếu ‘name’ bắt đầu bằng chữ "B" hoặc ‘price’ > 100, hàm sẽ kết thúc ngay lập tức và không thêm nút mới vào danh sách liên kết

2. Tạo một đối tượng Node mới:

n = Node(Car(name, price))

Tạo một đối tượng Car với name và price đã cho, sau đó tạo một đối tượng Node mới với Car object này.

3.Kiểm tra nếu danh sách liên kết rỗng:

if linklist.isEmpty():

linklist.head = linklist.tail = n

Nếu danh sách liên kết trống, thì head và tail của danh sách liên kết đều trỏ đến node mới tạo.

4. Trường hợp danh sách liên kết không rỗng:

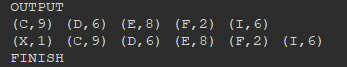
else:

linklist.tail.next = n

linklist.tail = n

Nếu danh sách liên kết không rỗng, node mới được thêm vào cuối danh sách thông qua linklist.tail.next và sau đó cập nhật linklist.tail để trỏ đến node mới này, làm cho node này trở thành node cuối cùng của danh sách liên kết.

Q1\_2:



1. Tạo một đối tượng Node mới:

node = Node(Car(name, price))

Tạo một đối tượng Car với name và price đã cho, sau đó tạo một đối tượng Node mới với Car object này.

2. Kiểm tra nếu danh sách liên kết là None (rỗng):

if linkList == None:

linkList.head = linkList.tail = node

Nếu danh sách liên kết là None (rỗng), thì head và tail của danh sách liên kết đều trỏ đến node mới tạo. Điều này có nghĩa là danh sách liên kết chỉ chứa một node và đó là node mới vừa tạo.

3. Trường hợp danh sách liên kết không rỗng:

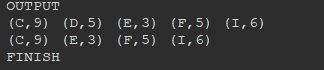
else:

node.next = linkList.head

linkList.head = node

Nếu danh sách liên kết không rỗng, node mới được thêm vào đầu danh sách. node.next được thiết lập để trỏ đến linkList.head, sau đó linkList.head được cập nhật để trỏ đến node mới. Điều này có nghĩa là node mới trở thành node đầu tiên của danh sách liên kết.

Q1\_3:



1. Kiểm tra điều kiện cho node đầu danh sách:

if self.check(linkList.head.data.Price):

linkList.head = linkList.head.next

Nếu giá trị thuộc tính Price của node đầu danh sách thỏa mãn điều kiện được kiểm tra bởi self.check, thì node đầu tiên sẽ bị xóa bằng cách cập nhật linkList.head để trỏ đến node tiếp theo.

2. Kiểm tra điều kiện cho các node còn lại trong danh sách:

else:

cu = linkList.head

while cu.next:

if self.check(cu.next.data.Price):

break

cu = cu.next

Nếu node đầu tiên không thỏa mãn điều kiện, một vòng lặp while được sử dụng để tìm node đầu tiên trong danh sách mà giá trị thuộc tính Price thỏa mãn điều kiện.

3. Xóa node thỏa mãn điều kiện:

if cu.next == None:

return

else:

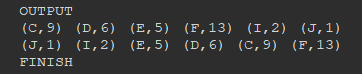
if cu.next == linkList.tail:

linkList.tail = cu

cu.next = cu.next.next

Nếu không có node nào thỏa mãn điều kiện, hàm kết thúc. Nếu node thỏa mãn điều kiện là node cuối cùng (cu.next == linkList.tail), thì cập nhật linkList.tail để trỏ đến node trước đó. Cuối cùng, node thỏa mãn điều kiện sẽ bị xóa bằng cách cập nhật cu.next để trỏ đến node sau nó.

Q1\_4:



1. Kiểm tra nếu danh sách liên kết là rỗng:

if linkList == None:

return

Nếu danh sách liên kết là None, hàm kết thúc ngay lập tức.

2. Sắp xếp danh sách liên kết:

i = linkList.head

while i.next:

j = i.next

while j:

if i.data.Price > j.data.Price:

t = i.data

i.data = j.data

j.data = t

j = j.next

i = i.next

Sắp xếp danh sách liên kết theo giá trị tăng dần của thuộc tính Price bằng cách so sánh giá trị của node i với các node j sau nó.

Nếu giá trị Price của node i lớn hơn giá trị Price của node j, thì hoán đổi giá trị Price của hai node này.

Vòng lặp while trong j được sử dụng để so sánh i với tất cả các node sau nó.

Vòng lặp while ngoại cùng với i được sử dụng để di chuyển qua từng node trong danh sách liên kết cho đến khi danh sách đã được sắp xếp.