Solution Review: Pairs with Sums

This lesson contains the solution review for the challenge of finding pairs in a doubly linked list which sum up to the given number.

we'll cover the following ^ Implementation Explanation

Here is an overview of our solution to finding a pair of nodes in a doubly linked list that sums up to a specific number.

Implementation

Here is the coding solution in Python for the previous challenge:

```
def pairs_with_sum(self, sum_val):
    pairs = list()
    p = self.head
    q = None
    while p:
        q = p.next
        while q:
        if p.data + q.data == sum_val:
              pairs.append("(" + str(p.data) + "," + str(q.data) + ")")
        q = q.next
        p = p.next
    return pairs
```

Explanation

On **line 2**, pairs is initialized to an empty Python list. In the next lines (**lines 3-4**), p and q are set equal to self.head and None respectively. We will use both these pointers (p and q) to make pairs out of the doubly linked list by using two while loops afterward.

The outer while loop on line 5 will run until p becomes equal to None while

the inner loop on **line** 7 will run for every iteration of the outer loop until **q** becomes **None**.

On **line 6**, we set **q** equal to **p.next** as we have to start pairing nodes from the next node of the current node as we would already have checked the pairing with all previous nodes. Then we'll check if the sum of **p.data** and **q.data** equals **sum_value** or not on **line 8**. If it is, then we append **p.data** and **q.data** to the list we declared at the beginning of the **pairs_with_sum** method. If it does not, we move on to the next node of **q** by updating **q** to **q.next** on **line 10**. In each iteration of the inner loop, we pair the data of **p** with all the data of the nodes after **p** using **q** and then check each of these pairs to see if they sum up to **sum_value**. This process is repeated for every node in the linked list as **p** updates to **p.next** on **line 11** in the outer **while** loop. After the outer loop terminates, **pairs** is returned from the method on **line 12**.

You can play around with all the methods that we have implemented for the DoublyLinkedList class in the code widget provided below.

```
class Node:
                                                                                        G
 def __init__(self, data):
   self.data = data
   self.next = None
   self.prev = None
class DoublyLinkedList:
 def init (self):
   self.head = None
  def append(self, data):
   if self.head is None:
     new node = Node(data)
     new node.prev = None
     self.head = new_node
   else:
     new node = Node(data)
     cur = self.head
     while cur.next:
         cur = cur.next
     cur.next = new node
     new node.prev = cur
     new_node.next = None
 def prepend(self, data):
   if self.head is None:
     new_node = Node(data)
     new_node.prev = None
     self.head = new_node
   else:
     new_node = Node(data)
     self.head.prev = new_node
     new_node.next = self.head
```

```
self.head = new_node
    new_node.prev = None
def print_list(self):
  cur = self.head
  while cur:
    print(cur.data)
    cur = cur.next
def add_after_node(self, key, data):
  cur = self.head
  while cur:
    if cur.next is None and cur.data == key:
      self.append(data)
      return
    elif cur.data == key:
      new_node = Node(data)
      nxt = cur.next
      cur.next = new_node
      new_node.next = nxt
      new_node.prev = cur
      nxt.prev = new_node
      return
    cur = cur.next
def add_before_node(self, key, data):
  cur = self.head
  while cur:
    if cur.prev is None and cur.data == key:
      self.prepend(data)
      return
    elif cur.data == key:
      new_node = Node(data)
      prev = cur.prev
      prev.next = new_node
      cur.prev = new_node
      new_node.next = cur
      new_node.prev = prev
      return
    cur = cur.next
def delete(self, key):
  cur = self.head
  while cur:
    if cur.data == key and cur == self.head:
      # Case 1:
      if not cur.next:
        cur = None
        self.head = None
        return
      # Case 2:
      else:
        nxt = cur.next
        cur.next = None
        nxt.prev = None
        cur = None
        self.head = nxt
        return
    elif cur.data == key:
        # Case 3:
```

```
if cur.next:
          nxt = cur.next
          prev = cur.prev
          prev.next = nxt
          nxt.prev = prev
          cur.next = None
          cur.prev = None
          cur = None
          return
        # Case 4:
      else:
          prev = cur.prev
          prev.next = None
          cur.prev = None
          cur = None
          return
    cur = cur.next
def delete_node(self, node):
  cur = self.head
  while cur:
    if cur == node and cur == self.head:
      # Case 1:
      if not cur.next:
       cur = None
        self.head = None
       return
      # Case 2:
      else:
       nxt = cur.next
       cur.next = None
       nxt.prev = None
       cur = None
        self.head = nxt
        return
   elif cur == node:
      # Case 3:
      if cur.next:
       nxt = cur.next
       prev = cur.prev
        prev.next = nxt
       nxt.prev = prev
       cur.next = None
       cur.prev = None
       cur = None
        return
      # Case 4:
      else:
        prev = cur.prev
        prev.next = None
        cur.prev = None
        cur = None
        return
    cur = cur.next
def reverse(self):
    tmp = None
    cur = self.head
```

```
while cur:
          tmp = cur.prev
          cur.prev = cur.next
          cur.next = tmp
          cur = cur.prev
      if tmp:
          self.head = tmp.prev
  def remove_duplicates(self):
      cur = self.head
      seen = dict()
      while cur:
          if cur.data not in seen:
              seen[cur.data] = 1
              cur = cur.next
          else:
              nxt = cur.next
              self.delete_node(cur)
              cur = nxt
  def pairs_with_sum(self, sum_val):
    pairs = list()
    p = self.head
    q = None
    while p:
      q = p.next
     while q:
        if p.data + q.data == sum_val:
            pairs.append("(" + str(p.data) + "," + str(q.data) + ")")
        q = q.next
      p = p.next
    return pairs
dllist = DoublyLinkedList()
dllist.append(1)
dllist.append(2)
dllist.append(3)
dllist.append(4)
dllist.append(5)
print(dllist.pairs_with_sum(5))
```

Now this lesson marks an end to the content on linked lists. Get ready to solve problems using another data structure in the next chapter!