2023-06-09 Handout – Linked Lists

Q1. Linked List Cycle ii

Link: https://leetcode.com/problems/linked-list-cycle-ii/

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

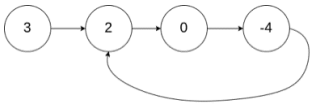
Note: Do not modify the linked list.

Example 1:

Input: head = [3,2,0,-4], pos = 1

Output: tail connects to node index 1

Explanation: There is a cycle in the linked list, where tail connects to the second node.



Example 2:

Input: head = [1], pos = -1

Output: no cycle

Explanation: There is no cycle in the linked list.

Q2. Reverse Linked List ii

Link: https://leetcode.com/problems/reverse-linked-list-ii/

Reverse a linked list from position m to n. Do it in one-pass.

Note: 1 ≤ m ≤ n ≤ length of list.

Example:

Input: 1->2->3->4->5->NULL, m = 2, n = 4

Output: 1->4->3->2->5->NULL

Q3. LRU Cache

Link: https://leetcode.com/problems/lru-cache/

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the LRUCache class:

- LRUCache(int capacity) Initialize the LRU cache with **positive** size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, **evict** the least recently used key.

The functions get and put must each run in O(1) average time complexity.

Example:

Input

```
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
```

Output

```
[null, null, null, 1, null, -1, null, -1, 3, 4]
```

```
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
lRUCache.get(4);    // return 4
```

Q4. Merge k Sorted Lists

Link: https://leetcode.com/problems/merge-k-sorted-lists/description/

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merger all the linked lists into one sorted linked list and return it.

```
Example - 1

Input: lists = [[1->4->5], [1->3->4], [2->6]]

Output: [1->1->2->3->4->4->5->6]
```