Programmer’s Study Group

Week 02

Linked List

- Single Linked List
- Double Linked List

```
// Single Linked List Node
Class ListNode {
        int val;
        ListNode next;
        ListNode(int val) {
                this.val = val;
                this.next = null;
        }
}
```

Basic algorithms

- Find a node / traverse a linked list
- Create a list
- Insert a node
- Delete a node
- Find middle of a list
- Merge two sorted list
- Merge sort a list

Find a node / traverse a linked list

```
public boolean find(ListNode head, int target) {
    while (head != null) {
        if (head.val == target) {
            return true;
        }
        head = head.next;
    }
    return false;
}
```

Create a list

- Given an input integer array, return a newly created Linked List

```
public ListNode createList(int[] nums) {
    if (nums == null) return null;
    ListNode dummyHead = new ListNode(0);
    ListNode current = dummyHead;
    for (int i = 0; i < nums.length; i++) {
        ListNode newNode = new ListNode(nums[i]);
        current.next = newNode;
        current = newNode;
    }
    return dummyHead.next;
}
```

Insert a node

```
// insert "node" after "prev"
…
node.next = prev.next;
prev.next = node;
…
```

- Insertion sort
  - Create an empty result list
  - Insert every node into result list
  - Time Complexity?

```
public ListNode insertionSortList(ListNode head) {
    ListNode dummyHead = new ListNode(0);
    while(head != null) {
      ListNode node = head, prev = dummyHead;
      head = head.next;
      while (prev != null) {
        if (prev.next == null || prev.next.val >= node.val) {
          //insert node after prev
          node.next = prev.next;
          prev.next = node;
          break;
        }
        prev = prev.next;
      }
    }
    return dummyHead.next;
}
```

Delete a node

- If you can access its previous node…

```
prev.next = prev.next.next;
```

- If you were only given the pointer to the node… (it’s not the last node)

```
node.val = node.next.val;
node.next = node.next.next;
```

Find middle of a list

- Naïve solution

- Solution with fast/slow pointers

```
 ListNode slow = head, fast = head;
while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
 }
 return slow;
```

- Time complexity?

Merge two sorted lists

```
ListNode merge(ListNode l1, ListNode l2) {
    ListNode dummyHead = new ListNode(0);
    ListNode p = dummyHead;
    while (l1 != null && l2 != null) {
        if (l1.val < l2.val) {
            p.next = l1;
            l1 = l1.next;
        } else {
            p.next = l2;
            l2 = l2.next;
        }
        p = p.next;
    }
    if (l1 != null)     p.next = l1;
    if (l2 != null)     p.next = l2;
    return dummyHead.next;
}
```

Merge sort a list

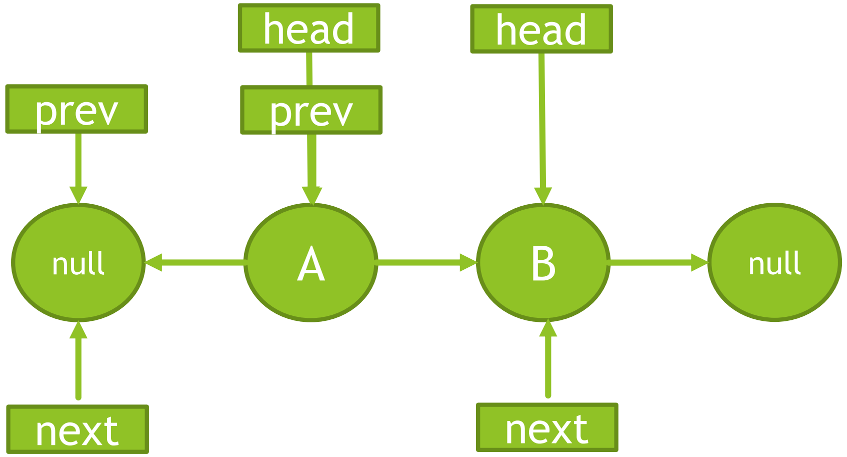
- Cut list into two
- Sort each of them recursively
- Merge two sorted lists

```
public ListNode sortList(ListNode head) {
        // base case
        if (head == null || head.next == null)
                return head;
        // step 1. cut the list to two halves
        ListNode prev = null, slow = head, fast = head;
        while (fast != null && fast.next != null) {
                prev = slow;
                slow = slow.next;
                fast = fast.next.next;
        }
        prev.next = null;
        // step 2. sort each half
        ListNode l1 = sortList(head);
        ListNode l2 = sortList(slow);
        // step 3. merge l1 and l2
        return merge(l1, l2);
}
```

Reverse linked list

- Iterative way

```
ListNode prev = null, next = null;
while (head != null) {
        next = head.next;
        head.next = prev;
        prev = head;
        head = next;
}
return prev;
```



- Recursive way

Advanced topics

- Find intersection of two linked list
  - Use two pointers
  - Think about how two pointers can meet?
- Find cycle of a linked list
  - Use two pointers too
  - What’s their travel speed?
  - You need to do math for Cycle II…

Questions?

Graph

- Directed / Undirected
- Weighted/ Unweighted
- Representation
  - 2D Matrix
  - Graph nodes
  - Which is better?

	A	B	C
A	0	1	0
B	1	0	1
C	0	1	0

```
class GraphNode {
    int label;
    List<GraphNode> neighbors;
        GraphNode(int x) {
        label = x;
        neighbors = new ArrayList<GraphNode>();
    }
 }
```

Traverse

- DFS – Recursive

```
if (visited.contains(node)) return;
visited.add(node);
for (GraphNode neighbor: node.neighbors) {
        traverse(neighbor);
}
```

- BFS - Iterative

```
void traverse (GraphNode node) {
        Queue<GraphNode> queue = new LinkedList<>();
        queue.add(node);
        while (!queue.isEmpty()) {
                node = queue.poll();
                visited.add(node);
                for (GraphNode neighbor: node.neighbors) {
                        if (!visited.contains(neighbor)) {
                                queue.add(neighbor);
                        }
                }
        }
}
```

Clone

```
Map<Integer, GraphNode> created = new HashMap<>();
public GraphNode cloneGraph(GraphNode node) {
    if (node == null) return null;
    if (created.containsKey(node.label))
        return created.get(node.label);

    GraphNode newNode = new GraphNode(node.label);
    created.put(newNode.label, newNode);
    for (GraphNode neighbor : node.neighbors) {
        newNode.neighbors.add(cloneGraph(neighbor));
    }

    return newNode;
}
```

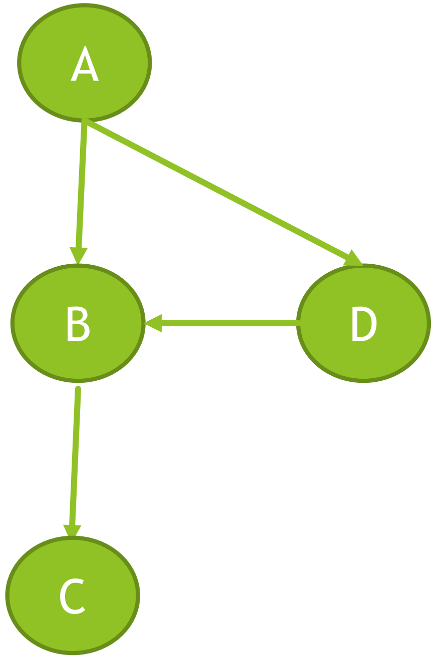
Clone linked list & tree?

- Clone Linked list with a random pointer
  - What’s the difference?
- Clone a tree
  - What’s the difference again?

Topological sort

- To sort nodes with order relationship
  - Task scheduling
  - Convert graph to a tree

- Algorithm
  - 1. Put nodes with incoming = 0 to a queue
  - 2. Dequeu, decrease incoming for each child node
  - 3. Add children with 0 incoming to the queue
  - 4. Repeat 2 until the queue is empty



Questions?

Homework!

- https://leetcode.com/problems/remove-duplicates-from-sorted-list/
- https://leetcode.com/problems/swap-nodes-in-pairs/
- https://leetcode.com/problems/rotate-list/
- https://leetcode.com/problems/partition-list/
- https://leetcode.com/problems/reverse-linked-list-ii/
- https://leetcode.com/problems/odd-even-linked-list/
- https://leetcode.com/problems/intersection-of-two-linked-lists/
- https://leetcode.com/problems/linked-list-cycle/
- https://leetcode.com/problems/linked-list-cycle-ii/

- https://leetcode.com/problems/clone-graph/
- https://leetcode.com/problems/course-schedule/
- https://leetcode.com/problems/course-schedule-ii/