LETTCODE Few PREMIUM questions(lintcode)

1.Meeting rooms

```
bool canAttendMeetings(vector<Interval> &intervals) {
        // Write your code here
         sort(intervals.begin(), intervals.end(),
             [](const Interval& x, const Interval& y) { return x.start < y.start;
});
        for (int i = 1; i < intervals.size(); ++i) {
            if (intervals[i].start < intervals[i - 1].end) {
                return false;
            }
        }
        return true;
    }
```

2. Meeting rooms II

```
  class
Solution
{
             /*
               (1) Sort the array by start time
               (2) Use a minheap to keep track of the earliest end time of each
           room
               (3) new room start > current earliest end time then the new
           meeting can just take up the original room
               (4) otherwise, we have to open a new room for them, so we push
           an another end time to minheap

             */
           public:
             int minMeetingRooms(vector<Interval>& intervals) {
               if(!intervals.size()) return 0;
               priority_queue<int, vector<int>, greater<int>> room;
               sort(intervals.begin(), intervals.end(), [](const Interval& lhs,
           const Interval& rhs){
                 return lhs.start < rhs.start;
               });
room.push(intervals[0].end);
for(int i=1; i<intervals.size(); i++) {
   if(intervals[i].start >= room.top())
      room.pop();
   room.push(intervals[i].end);
}
return room.size();
  }
};
```

3.Paint faince

//pepcoding

```
//
There
is a
fence
with n
posts,
each
post
can be
painted
with
one of
the k
colors.
          // You have to paint all the posts such that no more than two adjacent fence posts
          have the same color.
          // Return the total number of ways you can paint the fence.
          // Note:
          // n and k are non-negative integers.
          public int numWays(int n, int k) {
              if(n <= 0) return 0;
              if(n == 1) return k;
              // for the second and following posts, there are two cases, one is same color
          with previous,
              // the other one is diff color than previous, for example
              // case 1: same color, case 2: diff color
              int sameColor = k, diffColor = k * (k - 1);
              int i = 2;
```

```
        while(i < n){
            int pre_sameColor = sameColor, pre_diffColor = diffColor;
            // case 1
            sameColor = pre_diffColor;
            // case 2
            diffColor = pre_sameColor * (k - 1) + pre_diffColor * (k - 1);

            i++;
        }
        return sameColor + diffColor;
    }
```

4. Number of connected component in a undirected graph

[LintCode] Connected Component in Undirected Graph

Find the number connected component in the undirected graph. Each node in the graph contains a label and a list of its neighbors. (a connected component (or just component) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.)

Each connected component should sort by label.

Example

Given graph:

```
A------B  C
 \     |  |
  \    |  |
   \   |  |
    \  |  |
      D   E
```

Return `{A,B,D}, {C,E}`. Since there are two connected component which is `{A,B,D}, {C,E}`

Solution 1. BFS

Algorithm.

Iterate all nodes in the input graph and do the following.

1. For each unvisited graph node s, do a bfs starting from it and add all nodes that can be reached from s to a list as one connected component. Add each visited nodes to a global set that tracks which nodes have been visited.

2. Sort each connected component.

```
/**
 * Definition for Undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     ArrayList<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new
ArrayList<UndirectedGraphNode>(); }
 * };
 */

//Algorithm 1. bfs
public class Solution {
    public List<List<Integer>> connectedSet(ArrayList<UndirectedGraphNode> nodes) {
        List<List<Integer>> results = new ArrayList<List<Integer>>();
        if(nodes == null || nodes.size() == 0)
        {
            return results;
        }
        Set<UndirectedGraphNode> visited = new HashSet<UndirectedGraphNode>();
        //bfs on each node to get all the connected components
        for(UndirectedGraphNode node : nodes)
        {
            if(!visited.contains(node))
            {
                bfs(node, visited, results);
            }
        }
        return results;
    }
    private void bfs(UndirectedGraphNode node,
                     Set<UndirectedGraphNode> visited,
                     List<List<Integer>> results)
    {
        ArrayList<Integer> comp = new ArrayList<Integer>();
        Queue<UndirectedGraphNode> queue = new LinkedList<UndirectedGraphNode>();
        queue.offer(node);
        visited.add(node);
        comp.add(node.label);

        while(!queue.isEmpty())
        {
            UndirectedGraphNode curr = queue.poll();
            for(UndirectedGraphNode neighbor : curr.neighbors)
            {
                if(!visited.contains(neighbor))
                {
                    queue.offer(neighbor);
                    visited.add(neighbor);
                    comp.add(neighbor.label);
                }
            }
        }
        Collections.sort(comp);
        results.add(comp);
    }
}
```

5. Split BST

```
class
Solution
{
          public:
              vector<TreeNode*> splitBST(TreeNode* root, int V) {
                  vector<TreeNode*> res(2,NULL);
                  if(root==NULL) return res;
                  if(root->val>V){
                      res[1]=root;
                      vector<TreeNode*> tmp=splitBST(root->left,V);
                      root->left=tmp[1];
                      res[0]=tmp[0];
                  }else{
                      res[0]=root;
                      vector<TreeNode*> tmp=splitBST(root->right,V);
                      root->right=tmp[0];
                      res[1]=tmp[1];
                  }
                  return res;
              }
          };
```

Encode & decode String

```java
public class Solution {

    public String encode(List<String> strs) {
        // write your code here
         if (strs == null || strs.isEmpty()) {
            return "";
        }
        StringBuilder builder = new StringBuilder();

        for (int i = 0; i < strs.size(); i++) {
            String item = strs.get(i);

            if (":".equals(item)) {
                item += "::";
            } else {
                item += ":";
            }
            if (i < strs.size() - 1) {
                item += ";";
            }
            builder.append(item);
        }
        return builder.toString();
    }

    public List<String> decode(String str) {
        // write your code here
         // write your code here
        if (str == null || str.isEmpty()) {
            return new ArrayList<>();
        }
        String[] split = str.split(";");
        List<String> ans = new ArrayList<>();
        for (String item : split) {
            if (item.startsWith(":")) {
                ans.add(":");
            } else {
                item = item.substring(0, item.length() - 1);
                ans.add(item);
            }
        }
        return ans;
        }
}
```