CS263

ASSIGNMENT 8

NAME:

ARCHIT AGRAWAL

ROLL NO. :

202051213

SECTION:

A

1. **Suppose your course instructor has made n groups of students in the class which contains unqeual number students. For each group a TA has been assigned. Each TA's has checked the assignment of their respective group and sorted the assignments in their roll number. TA's has to submit the total assignments to the course instructor in sorted form. How TA's will combine the assignments so that they minimized the total number of comparison to sort the files.**

This problem can be visualised as "Merging K sorted lists". Each TA has sorted their respective group based on roll numbers. This can be assumed to be given K sorted list. Each list has students of a particular group sorted according to their roll number. The TA's has to submit the total assignments to the course instructor in sorted form. That is, they have to return the K lists, merged together and sorted according to their roll number.

For example:

Suppose TA 1 has 3 students with roll number {1, 5, 8}

TA 2 has 3 students with roll numbers {2, 4, 6}

TA 3 has 2 students with roll number {3, 7}

These TA's have to return another list to the course instructor which will be {1, 2, 3, 4, 5, 6, 7, 8}.

Algorithm

The input is a linkedlist array "lists". Length of "lists" i.e number of TA's is k

- Make a node 'head' and initialise it 'null'. This node will be the head of the node that we have to return (say 'answer'). Make another node, 'prev' which stores the last inserted node in the 'answer' list.
- If the length of the 'lists' array is 0, it means there are 0 TA's. Hence, return null.
- Build a minheap (or priority queue) and insert the first node of each of the k lists if they are not null. The comparisons will be done on the basis of the rollNumber i.e lowest roll number (out of the present in the heap) will be the root node of heap.
- Run a while loop until this minheap is empty, and do the following:

1) Remove the root node (or dequeue) from the minheap and store it in another temporary node (say ‘min’)
2) Append a new node to the ‘answer’ list with its value being the roll number of the ‘min’ node. To do so, we don’t need to traverse the ‘answer’ list again and again. Simply, update the next node of ‘prev’ to ‘min’ and then update ‘prev’ node to ‘min’.
3) Insert the next node to ‘min’ is it is not null.

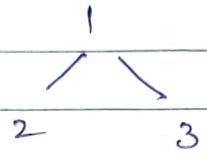
- Return the ‘head’ node.

Working of The Above Algorithm

list 1 → 1, 5, 8
list 2 → 2, 4, 6
list 3 → 3, 7

Build minheap with first Node of each list

∴ heap is

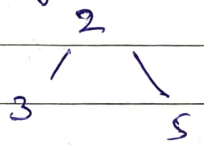


→ Remove root node and insert it in answer list

∴ answer → 1.

→ Insert next node to 1 in list 1 (i.e. 5) to heap and heapify

∴ heap

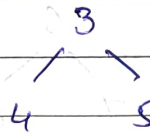


→ Remove root node and insert it in answer list

∴ answer → 1 → 2

→ Insert next node to 2 in list 2 to heap and heapify

∴ heap

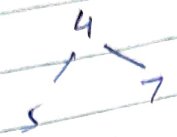


→ Remove root node and insert it in answer list

∴ answer → 1 → 2 → 3

→ Insert next node to 3 in list 3 to heap and heapify

∴ heap →

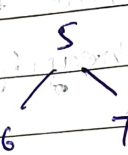


→ Remove root node and insert it in answer list

∴ answer → 1 → 2 → 3 → 4

→ Insert next node to 4 in list 4 to heap and heapify.

heap →

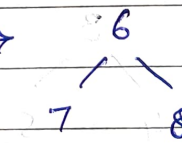


→ Remove root node and insert it in answer list.

answer → 1 → 2 → 3 → 4 → 5

→ Insert next node to 5 in list 1 to heap and heapify.

heap →



→ Remove root node and insert it in answer list

ans → 1 → 2 → 3 → 4 → 5 → 6

→ Now, next node to 6 in list 2 doesn't exist, hence, no insertion in heap takes place.
Heapify
∴ heap → 7
⌀

→ Remove root node and insert it in answer list

Answer: 1 → 2 → 3 → 4 → 5 → 6 → 7

→ No next node to 7 exists in list 3. ~~Insert~~
Heapify.

heap → 8

→ Remove ~~root~~ root node and insert it in ~~answer~~ answer list.

answer: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8

→ Heap is empty. Exit the loop and return answer list.

Below is the implementation of the above algorithm.

```
public ListNode mergeKLists(ListNode[] lists) {
        ListNode head = null;
        ListNode prev = null;
        int k = lists.length;

        if(k == 0) return null;

        PriorityQueue<ListNode> pq = new PriorityQueue<ListNode>(k, (l1, l2) -
> l1.val - l2.val);


        for(int i = 0; i < k; i++){
            if(lists[i] != null)
                pq.add(lists[i]);
        }

        while(!pq.isEmpty()){
            ListNode sol = new ListNode();
            ListNode min;
            min = pq.poll();
            sol.val = min.val;
            if(prev == null){
                head = sol;
            } else{
                prev.next = sol;
            }
            prev = sol;

            if(min.next != null) pq.add(min.next);
        }

        return head;
    }
```

Analysis:

Space complexity is clearly O(k), where k is the number of groups divided in the class. This is because we only require a minheap of size k.

Now, the time complexity. Let us assume that the maximum size of any group is n. Therefore, in the worst case, a total of (n * k) students are there in the classroom. First, we are building a heap of size k which takes O(k) time. After that for each of the (n * k) students (in worst case), we are heapifying this

heap. Since, heapify has a complexity of O(log(k)), the complexity of this task is O(n*k*log(k)).

Therefore, the time complexity of above algorithm is O(n*k*log(k)) + O(k) = O(n*k*log(k)), where k is number of groups in the class and there are a maximum of n students in any group.

2. In your Gandhinagar city, there are various locations such as Vidhan sabha, Achardham, Gandhi Asharam,...,etc. There exists a road network that connects all the locations. Due to elections in Gujarat and rallies, all the paths between any two locations act as one-way. You and your friends have decided to visit all the major locations as today is a holiday. You have to start your traveling plan from the hostel and after visiting each location you return to the hostel. The condition is you cannot follow the same path which you have already visited. Write an algorithm that gives you the efficient route to successfully execute your plan otherwise you drop your today's plan.

Algorithm

- Mark all the nodes as unvisited and create and empty path and make the pathExist false.
- Start from the vertex v1 and visit the next vertex (use adjacency list).
- Keep track of visited nodes to avoid cycles.
- Add current vertex to result (taking integer array here) to keep track of path from vertex v1.
- Now if you look carefully, the new problem is to find paths from the current vertex to destination. For instance as per the example above, start from vertex 0 and visit vertex 1. Now all the paths from vertex 1 to vertex 5 will be included in our final result if we add vertex 0. So make a recursive call with source as vertex 1 and destination as vertex 5.
- Once reach to the destination vertex, print the path.
- Mark the current node as unmarked and delete it from path.
- Now visit the next node in adjacency list in step 1 and repeat all the steps (loop)

Analysis

This is a brute force approach, where each possibility is checked. Since, each location can be selected in any order. If we try finding the recursion tree, at, i^{th} level there will be l^i recursive calls, where l is number of locations and since in each call DFS is called, therefore the overall complexity will be $O(l^i. (V + E))$, where V is the number of vertices and E is the number of edges.