Merge K sorted linked lists

Wednesday, January 19, 2022 8:09 PM

Merge K sorted linked lists

Medium Accuracy: 41.42% Submissions: 40124 Points: 4

Given **K** sorted linked lists of different sizes. The task is to merge them in such a way that after merging they will be a single sorted linked list.

Example 1:

Input:
K = 4
value = {{1,2,3},{4 5},{5 6},{7,8}}
Output: 1 2 3 4 5 5 6 7 8
Explanation:
The test case has 4 sorted linked
list of size 3, 2, 2, 2
1st list 1 -> 2-> 3
2nd list 4->5
3rd list 5->6
4th list 7->8
The merged list will be
1->2->3->4->5->5->6->7->8.

Example 2:

Input:
K = 3
value = {{1,3},{4,5,6},{8}}
Output: 1 3 4 5 6 8
Explanation:
The test case has 3 sorted linked
list of size 2, 3, 1.
1st list 1 -> 3
2nd list 4 -> 5 -> 6
3rd list 8
The merged list will be
1->3->4->5->6->8.

Your Task:

The task is to complete the function **mergeKList**() which merges the K given lists into a sorted one. The **printing** is done **automatically** by the **driver code**.

Expected Time Complexity: O(nk Logk)

Expected Auxiliary Space: O(k)

Note: n is the maximum size of all the **k** link list

From <https://practice.geeksforgeeks.org/problems/merge-k-sorted-linked-lists/1>

```
import java.util.Arrays;
import java.util.Comparator;
public class MergeK_SortedLinkedLists {
    class Node {
        int data;
        Node next;
        Node(int key) {
            data = key;
            next = null;
        }
    }

 void print(Node head)
    {

            //  printing  the  list
            if (head==null) {
                System.out.println("NULL");
            }
            Node temp=head;
            while (temp!=null) {
                System.out.print( temp.data+" -->");
                temp=temp.next;
            }
            System.out.print("NULL");
            System.out.println();

    }
    Node mergeKList(Node[]arr,int K)
    {
        //Add your code here.

          Arrays.sort(arr,new Comparator<Node>() {
            @Override
            public int compare(Node nodeOne, Node nodeTwo)
            {
                if (nodeOne.data == nodeTwo.data)
                return 0;
            else if (nodeOne.data > nodeTwo.data)
                return 1;
            else
                return -1;
            }
        });
        Node head=null;
        Node temp=head;
        boolean gotHead=false;
        for (int i = 0; i < arr.length; i++) {
            if (!gotHead) {
```

```
                head=merge(head, arr[i]);
                temp=head;
             gotHead=true;
             print(head);

            }
            else{
                temp=merge(temp, arr[i]);
            print(temp);
            }
        }



        return temp;

    }
    Node merge(Node a,Node b)
    {
        if(a==null)return b;
        if (b==null)return a;
        Node result;
        if (a.data<b.data) {
            result=a;
            result.next=merge(a.next, b);
        }
        else{
            result=b;
            result.next=merge(a, b.next);
        }
       return result;
    }

}
```

Given K sorted linked lists of size N each, merge them and print the sorted output.

Examples:

Input: k = 3, n = 4
list1 = 1->3->5->7->NULL

list2 = 2->4->6->8->NULL

list3 = 0->9->10->11->NULL

Output: 0->1->2->3->4->5->6->7->8->9->10->11

Merged lists in a sorted order

where every element is greater

than the previous element.

Input: k = 3, n = 3

list1 = 1->3->7->NULL

list2 = 2->4->8->NULL

list3 = 9->10->11->NULL

Output: 1->2->3->4->7->8->9->10->11

Merged lists in a sorted order

where every element is greater

than the previous element.

From <https://practice.geeksforgeeks.org/problems/merge-k-sorted-linked-lists/1#>

Method 1 (Simple) Approach: A Simple Solution is to initialize result as first list. Now traverse all lists starting from second list. Insert every node of currently traversed list into result in a sorted way.

From <https://practice.geeksforgeeks.org/problems/merge-k-sorted-linked-lists/1#>

```
// C++ program to merge k sorted
// arrays of size n each
#include <bits/stdc++.h>
using namespace std;
// A Linked List node
struct Node {
    int data;
    Node* next;
};
/* Function to print nodes in
   a given linked list */
void printList(Node* node)
{
    while (node != NULL) {
        printf("%d ", node->data);
        node = node->next;
    }
}
// The main function that
// takes an array of lists
// arr[0..last] and generates
// the sorted output
Node* mergeKLists(Node* arr[], int last)
{
    // Traverse form second list to last
    for (int i = 1; i <= last; i++) {
        while (true) {
            // head of both the lists,
            // 0 and ith list.
            Node *head_0 = arr[0], *head_i = arr[i];
            // Break if list ended
            if (head_i == NULL)
                break;
            // Smaller than first element
            if (head_0->data >= head_i->data) {
                arr[i] = head_i->next;
```

```
                head_i->next = head_0;
                arr[0] = head_i;
            }
            else
                // Traverse the first list
                while (head_0->next != NULL) {
                    // Smaller than next element
                    if (head_0->next->data
                        >= head_i->data) {
                        arr[i] = head_i->next;
                        head_i->next = head_0->next;
                        head_0->next = head_i;
                        break;
                    }
                    // go to next node
                    head_0 = head_0->next;
                    // if last node
                    if (head_0->next == NULL) {
                        arr[i] = head_i->next;
                        head_i->next = NULL;
                        head_0->next = head_i;
                        head_0->next->next = NULL;
                        break;
                    }
                }
        }
    }
    return arr[0];
}
// Utility function to create a new node.
Node* newNode(int data)
{
    struct Node* temp = new Node;
    temp->data = data;
    temp->next = NULL;
    return temp;
}
// Driver program to test
// above functions
int main()
{
    // Number of linked lists
    int k = 3;
    // Number of elements in each list
    int n = 4;
    // an array of pointers storing the
    // head nodes of the linked lists
    Node* arr[k];
    arr[0] = newNode(1);
    arr[0]->next = newNode(3);
    arr[0]->next->next = newNode(5);
    arr[0]->next->next->next = newNode(7);
    arr[1] = newNode(2);
    arr[1]->next = newNode(4);
    arr[1]->next->next = newNode(6);
    arr[1]->next->next->next = newNode(8);
    arr[2] = newNode(0);
    arr[2]->next = newNode(9);
```

```
    arr[2]->next->next = newNode(10);
    arr[2]->next->next->next = newNode(11);
    // Merge all lists
    Node* head = mergeKLists(arr, k - 1);
    printList(head);
    return 0;
}
```

Output:

```
0 1 2 3 4 5 6 7 8 9 10 11
```

Complexity Analysis:

- **Time complexity:** O(N_2), where N is total number of nodes, i.e., N = kn.
- **Auxiliary Space:** O(1).
  As no extra space is required.

From <https://practice.geeksforgeeks.org/problems/merge-k-sorted-linked-lists/1#>

Merge k sorted linked lists | Set 2 (Using Min Heap)

- Difficulty Level : Medium
- Last Updated : 30 Oct, 2021

Given **k** linked lists each of size **n** and each list is sorted in non-decreasing order, merge them into a single sorted (non-decreasing order) linked list and print the sorted linked list as output.

Examples:

```
Input: k = 3, n =  4
list1 = 1->3->5->7->NULL
list2 = 2->4->6->8->NULL
list3 = 0->9->10->11->NULL
Output: 0->1->2->3->4->5->6->7->8->9->10->11
Merged lists in a sorted order
where every element is greater
than the previous element.
```

```
Input: k = 3, n =  3
list1 = 1->3->7->NULL
list2 = 2->4->8->NULL
list3 = 9->10->11->NULL

Output: 1->2->3->4->7->8->9->10->11
Merged lists in a sorted order
where every element is greater
than the previous element.
```

Source: Merge K sorted Linked Lists | Method 2

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

An efficient solution for the problem has been discussed in **Method 3** of this post.

Approach: This solution is based on the **MIN HEAP** approach used to solve the problem 'merge k sorted arrays' which is discussed here.

MinHeap: A Min-Heap is a complete binary tree in which the value in each internal node is smaller than or equal to the values in the children of that node. Mapping the elements of a heap into an array is trivial: if a node is stored at index k, then its left child is stored at index 2k + 1 and its right child at index 2k + 2.

1. Create a min-heap and insert the first element of all the 'k' linked lists.
2. As long as the min-heap is not empty, perform the following steps:
- Remove the top element of the min-heap (which is the current minimum among all the elements in the min-heap) and add it to the result list.
- If there exists an element (in the same linked list) next to the element popped out in previous step, insert it into the min-heap.
3. Return the head node address of the merged list.
Below is the implementation of the above approach:

C++

```
// C++ implementation to merge k
// sorted linked lists
// | Using MIN HEAP method
#include <bits/stdc++.h>
using namespace std;

struct Node
{
    int data;
    struct Node* next;
};

// Utility function to create
// a new node
struct Node* newNode(int data)
{
    // Allocate node
    struct Node* new_node = new Node();

    // Put in the data
    new_node->data = data;
    new_node->next = NULL;

    return new_node;
}

// 'compare' function used to build
```

```
// up the priority queue
struct compare
{
    bool operator()(
         struct Node* a, struct Node* b)
    {
        return a->data > b->data;
    }
};

// Function to merge k sorted linked lists
struct Node* mergeKSortedLists(
             struct Node* arr[], int k)
{
    // Priority_queue 'pq' implemented
    // as min heap with the help of
    // 'compare' function
    priority_queue<Node*, vector<Node*>, compare> pq;

    // Push the head nodes of all
    // the k lists in 'pq'
    for (int i = 0; i < k; i++)
        if (arr[i] != NULL)
            pq.push(arr[i]);

      // Handles the case when k = 0
      // or lists have no elements in them
      if (pq.empty())
        return NULL;

      struct Node *dummy = newNode(0);
      struct Node *last = dummy;

    // Loop till 'pq' is not empty
    while (!pq.empty())
    {
        // Get the top element of 'pq'
        struct Node* curr = pq.top();
        pq.pop();

        // Add the top element of 'pq'
        // to the resultant merged list
        last->next = curr;
        last = last->next;

        // Check if there is a node
         // next to the 'top' node
        // in the list of which 'top'
        // node is a member
        if (curr->next != NULL)

        // Push the next node of top node
        // in 'pq'
        pq.push(curr->next);
    }
```

```
    // Address of head node of the required
    // merged list
    return dummy->next;
}

// Function to print the singly
// linked list
void printList(struct Node* head)
{
    while (head != NULL)
    {
        cout << head->data << " ";
        head = head->next;
    }
}

// Driver code
int main()
{
    // Number of linked lists
    int k = 3;

    // Number of elements in each list
    int n = 4;

    // An array of pointers storing the
    // head nodes of the linked lists
    Node* arr[k];

    // Creating k = 3 sorted lists
    arr[0] = newNode(1);
    arr[0]->next = newNode(3);
    arr[0]->next->next = newNode(5);
    arr[0]->next->next->next = newNode(7);

    arr[1] = newNode(2);
    arr[1]->next = newNode(4);
    arr[1]->next->next = newNode(6);
    arr[1]->next->next->next = newNode(8);

    arr[2] = newNode(0);
    arr[2]->next = newNode(9);
    arr[2]->next->next = newNode(10);
    arr[2]->next->next->next = newNode(11);

    // Merge the k sorted lists
    struct Node* head = mergeKSortedLists(arr, k);

    // Print the merged list
    printList(head);

    return 0;
}
```

Output:

0 1 2 3 4 5 6 7 8 9 10 11

Complexity Analysis:

- **Time Complexity:** O(N * log k) or O(n * k * log k), where, 'N' is the total number of elements among all the linked lists, 'k' is the total number of lists, and 'n' is the size of each linked list.
  Insertion and deletion operation will be performed in min-heap for all N nodes.
  Insertion and deletion in a min-heap require log k time.
- **Auxiliary Space:** O(k).
  The priority queue will have atmost 'k' number of elements at any point of time, hence the additional space required for our algorithm is O(k).

From <https://www.geeksforgeeks.org/merge-k-sorted-linked-lists-set-2-using-min-heap/>

In this post, **Divide and Conquer** approach is discussed. This approach doesn't require extra space for heap and works in O(nk Log k)

It is known that merging of two linked lists can be done in O(n) time and O(1) space (For arrays O(n) space is required).

1. The idea is to pair up K lists and merge each pair in linear time using O(1) space.
2. After first cycle, K/2 lists are left each of size 2*N. After second cycle, K/4 lists are left each of size 4*N and so on.
3. Repeat the procedure until we have only one list left.

From <https://practice.geeksforgeeks.org/problems/merge-k-sorted-linked-lists/1#>

```
// Java program to merge k sorted arrays of size n each
public class MergeKSortedLists {
    /* Takes two lists sorted in increasing order, and merge
    their nodes together to make one big sorted list. Below
    function takes O(Log n) extra space for recursive calls,
    but it can be easily modified to work with same time and
    O(1) extra space  */
    public static Node SortedMerge(Node a, Node b)
    {
        Node result = null;
        /* Base cases */
        if (a == null)
            return b;
        else if (b == null)
            return a;
        /* Pick either a or b, and recur */
        if (a.data <= b.data) {
            result = a;
            result.next = SortedMerge(a.next, b);
        }
        else {
            result = b;
            result.next = SortedMerge(a, b.next);
        }
        return result;
    }
    // The main function that takes an array of lists
    // arr[0..last] and generates the sorted output
```

```
    public static Node mergeKLists(Node arr[], int last)
    {
        // repeat until only one list is left
        while (last != 0) {
            int i = 0, j = last;
            // (i, j) forms a pair
            while (i < j) {
                // merge List i with List j and store
                // merged list in List i
                arr[i] = SortedMerge(arr[i], arr[j]);
                // consider next pair
                i++;
                j--;
                // If all pairs are merged, update last
                if (i >= j)
                    last = j;
            }
        }
        return arr[0];
    }
    /* Function to print nodes in a given linked list */
    public static void printList(Node node)
    {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }
    public static void main(String args[])
    {
        int k = 3; // Number of linked lists
        int n = 4; // Number of elements in each list
        // an array of pointers storing the head nodes
        // of the linked lists
        Node arr[] = new Node[k];
        arr[0] = new Node(1);
        arr[0].next = new Node(3);
        arr[0].next.next = new Node(5);
        arr[0].next.next.next = new Node(7);
        arr[1] = new Node(2);
        arr[1].next = new Node(4);
        arr[1].next.next = new Node(6);
        arr[1].next.next.next = new Node(8);
        arr[2] = new Node(0);
        arr[2].next = new Node(9);
        arr[2].next.next = new Node(10);
        arr[2].next.next.next = new Node(11);
        // Merge all lists
        Node head = mergeKLists(arr, k - 1);
        printList(head);
    }
}
class Node {
    int data;
    Node next;
    Node(int data)
    {
        this.data = data;
```

```
    }
}
// This code is contributed by Gaurav Tiwari
```

Complexity Analysis:

- **Time Complexity:** O(nk logk).
  As outer while loop in function mergeKLists() runs log k times and every time it processes nk elements.
- **Auxiliary Space:** O(1).
  As no extra space is required.

Method 4: This solution is based on the **MIN HEAP** approach used to solve the problem 'merge k sorted arrays' which is discussed here.

MinHeap: A Min-Heap is a complete binary tree in which the value in each internal node is smaller than or equal to the values in the children of that node. Mapping the elements of a heap into an array is trivial: if a node is stored at index k, then its left child is stored at index 2k + 1 and its right child at index 2k + 2.

The process must start with creating a MinHeap and inserting the first element of all the k linked list. Remove the root element of Minheap and insert it in the output list and insert the next element from the linked list of removed element. To get the result the step must continue until there is no element in the MinHeap left.

From <https://practice.geeksforgeeks.org/problems/merge-k-sorted-linked-lists/1#>

```
// Java implementation to merge
// k sorted linked lists
// Using MIN HEAP method
import java.util.PriorityQueue;
import java.util.Comparator;
public class MergeKLists {
    // function to merge k
    // sorted linked lists
    public static Node mergeKSortedLists(
        Node arr[], int k)
    {
        Node head = null, last = null;
        // priority_queue 'pq' implemeted
        // as min heap with the
        // help of 'compare' function
        PriorityQueue<Node> pq
            = new PriorityQueue<>(
                new Comparator<Node>() {
                    public int compare(Node a, Node b)
                    {
                        return a.data - b.data;
                    }
                });
        // push the head nodes of all
        // the k lists in 'pq'
        for (int i = 0; i < k; i++)
            if (arr[i] != null)
                pq.add(arr[i]);
        // loop till 'pq' is not empty
        while (!pq.isEmpty()) {
            // get the top element of 'pq'
            Node top = pq.peek();
            pq.remove();
            // check if there is a node
```

```
            // next to the 'top' node
            // in the list of which 'top'
            // node is a member
            if (top.next != null)
                // push the next node in 'pq'
                pq.add(top.next);
            // if final merged list is empty
            if (head == null) {
                head = top;
                // points to the last node so far of
                // the final merged list
                last = top;
            }
            else {
                // insert 'top' at the end
                // of the merged list so far
                last.next = top;
                // update the 'last' pointer
                last = top;
            }
        }
        // head node of the required merged list
        return head;
    }
    // function to print the singly linked list
    public static void printList(Node head)
    {
        while (head != null) {
            System.out.print(head.data + " ");
            head = head.next;
        }
    }
    // Utility function to create a new node
    public Node push(int data)
    {
        Node newNode = new Node(data);
        newNode.next = null;
        return newNode;
    }
    public static void main(String args[])
    {
        int k = 3; // Number of linked lists
        int n = 4; // Number of elements in each list
        // an array of pointers storing the head nodes
        // of the linked lists
        Node arr[] = new Node[k];
        arr[0] = new Node(1);
        arr[0].next = new Node(3);
        arr[0].next.next = new Node(5);
        arr[0].next.next.next = new Node(7);
        arr[1] = new Node(2);
        arr[1].next = new Node(4);
        arr[1].next.next = new Node(6);
        arr[1].next.next.next = new Node(8);
        arr[2] = new Node(0);
        arr[2].next = new Node(9);
        arr[2].next.next = new Node(10);
        arr[2].next.next.next = new Node(11);
```

```
        // Merge all lists
        Node head = mergeKSortedLists(arr, k);
        printList(head);
    }
}
class Node {
    int data;
    Node next;
    Node(int data)
    {
        this.data = data;
        next = null;
    }
}
// This code is contributed by Gaurav Tiwari
```