medium.com

Java algorithms: Merge k Sorted Lists (LeetCode) - Javarevisited - Medium

Ruslan Rakhmedov

4-5 minutes

Photo by Markus Spiske on Unsplash

Task description:

You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order.

Merge 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]
Explanation: The linked-lists are:
[
  1->4->5,
  1->3->4,
  2->6
]
merging them into one sorted list:
1->1->2->3->4->4->5->6

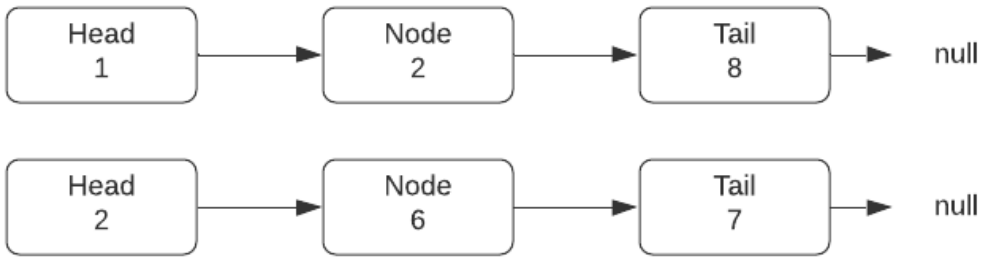
Solution:

Before jumping to the solution to this problem let's recall what the linked list is.

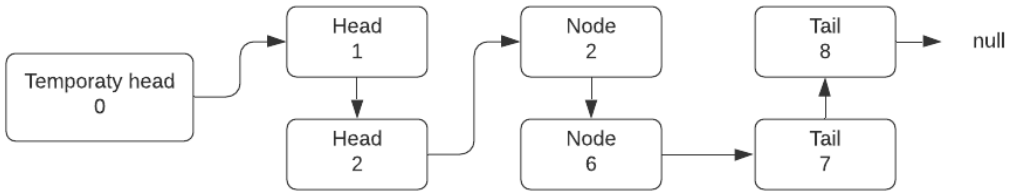
A linked list is a data structure that consisted of nodes connected with each other by one(single linked list) of two(double linked list) pointers. Every node usually contains some value and pointer to the next node. The very first node is called Head and the last node is called Tail. Tail as a next pointer usually has null.

In this task, we are dealing with sorted linked lists. Which means all values in every node are sorted. Every node in every list has value less or equal than value in a node which it points. To visualize it here's an example of sorted linked list.

Having an understanding of what the sorted linked list is we could solve the easier version of our problem. How would you merge 2 sorted linked lists?



And the answer is - extremely simple. Compare heads of each list. Pick the node with the minimum value. Attach it to your answer and adjust the pointer to the next node in the list from which you picked out a node. Do it until one of the heads is non-equal to null. Once you reach that point you could simply attach the rest to your answer.



Now we are close to solving the actual problem. From the easier task, we see that the key to the solution - is taking the node with the smallest value and attaching it to the answer by adjusting the pointer.

Having an array of K sorted list heads we could iterate through it and every time pick a node with the smallest value. It would solve the problem but the time complexity is terrible.

The better way will be to take every list's head and add it to a priority queue. Priority Queue is another data structure that

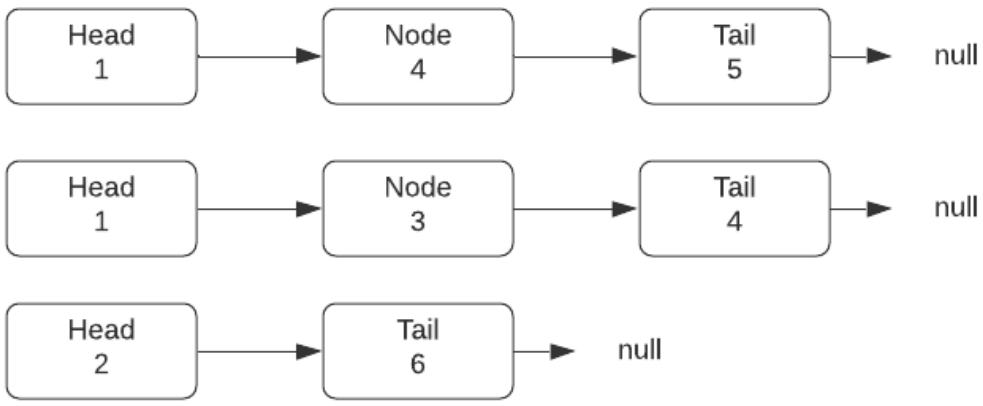
provides getting smallest or biggest value in O(1) constant time. And adding or remove this value in O(log n) time which is good enough for our task. Often priority queue is called a binary heap according to the way of storing elements. I could describe it in a future article.

In line 8 we create a “dummy” node, which is to be our temporary head. It will help us easily return the real head of the sorted linked lists by calling dummy.next

At lines 13–15 we create a Priority Queue. We need to sort elements in it. For this purpose we provide Comparator.

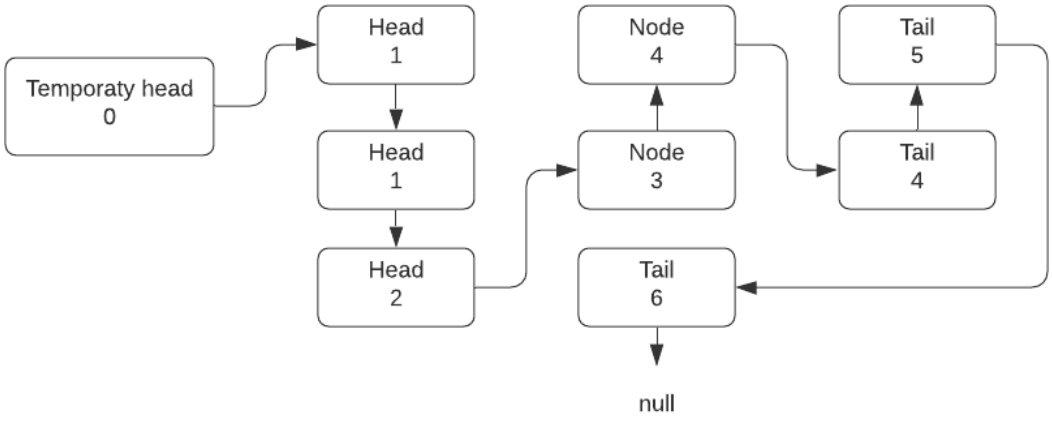
At lines 17–23 we add all non-null heads to the Priority Queue.

At lines 25–42 we have the main while loop which contains the main logic. We update current node by removing the node with the minimum value from the top of our priority queue. Then we update next pointer by getting next node from current. We also need to update the prev pointer. You can think of the prev pointer as a pointer to the temporary tail of the linked list we are building. And last but not least if next link is not null we must push it back to the priority queue.

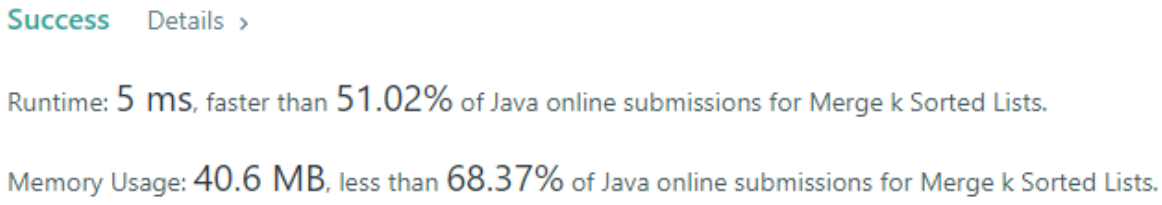


After executing previous steps at given example 1 we will get the

state represented below. I intentionally didn't flat it out. To show you transition from the original state to the end state



The algorithm listed above gives us the following result.



The time complexity for our solution is O(N log K) where N - is a total number of nodes in all lists where K - is a number of sorted linked lists we have.

The space complexity for our solution is O(N) where N - is total number of nodes to create an answer plus O(K) where K - is number of sorted linked lists we have. According to the big O notation we can omit O(K) because it could be equal or less then O(N) in terms of a number of elements.

I hope this article helped you to understand the logic hidden behind this difficult task. Thanks for reading! Looking forward to your feedback. See you soon ✌