Sort a K Sorted Doubly Linked List

solved by	Senan
Platform	GeeksForGeeks
difficulty	Easy
tags	Heaps Linked List
language	C++
solved on	@20/10/2024
link	https://www.geeksforgeeks.org/problems/sort-a-k-sorted-doubly-linked-list/1
Completion	☑

Intuition

A k-sorted doubly linked list is a list where each node is at most `k` positions away from its correct position in the sorted list. The idea is to use a min-heap (priority queue) to sort the list efficiently. At any point, the heap holds at most `k+1` elements, ensuring that we can place the correct minimum element in its sorted position. The heap helps maintain the order as we process the nodes in the list.

Approach

1. Traverse through the first `k+1` elements and push them into a min-heap.
2. Then, for each node in the list, replace its value with the minimum value in the heap (top of the priority queue).
3. Move to the next node and push its value into the heap while popping the smallest value from the heap to keep it updated.
4. Repeat this process until the entire list is sorted.

Complexity

Time Complexity:

- The time complexity is `O(n log k)`, where `n` is the number of nodes in the doubly linked list and `k` is the maximum distance any node is from its correct position. This is because each insertion and deletion in the priority queue takes `O(log k)`, and we perform this operation `n` times.

Space Complexity:

- The space complexity is `O(k)` because the priority queue stores at most `k+1` elements at any given time.

Code

```
class Solution {
  public:
    // function to sort a k sorted doubly linked list
    DLLNode *sortAKSortedDLL(DLLNode *head, int k) {
```

```
        priority_queue<int, vector<int>, greater<int>> pq;

        DLLNode* temp = head;
        int count = 0;

        while ((count < k+1) && temp) {
            pq.push(temp->data
            temp = temp->next;
            count++;
        }

        DLLNode* traverse = head;
        while (traverse) {
            traverse->data = pq.top();
            pq.pop();
            if (temp) {
                pq.push(temp->data);
                temp = temp->next;
            }
                traverse = traverse->next;
            }

            return head;
        }
};
```