

In the *Deleted sequence* problem, you maintain a dynamic set M of elements from the domain  $\{1,2,\ldots,n\}$  under the operations INSERT and DELETEMIN. The input is a sequence S of n INSERT and m DELETEMIN calls (possibly interleaved), where each key in  $\{1,2,\ldots,n\}$  is inserted exactly once. Your goal is to determine which key is returned by each DELETEMIN call. The expected output is an array DeletedKeys[1:m], where DeletedKeys[i] is the key returned by the ith DELETEMIN call (for  $i=1,2,\ldots,m$ ). Note that you are allowed to see the entire sequence S before determining any of the returned keys. To develop an algorithm for this problem, break the sequence S into homogenous subsequences  $I_1, D, I_2, D, I_3, \ldots, I_m, D, I_{m+1}$ , where each S represents a single DELETEMIN call and each S represents a (possibly empty) sequence of INSERT calls. For each subsequence S initially place the keys inserted by these operations into a set S0, which is empty if S1 is empty. Then execute the DELETEDSEQUENCE function.

```
1 function DELETEDSEQUENCE(m, n)

2 | for i = 1 to m do

3 | determine j such that i \in K_j

4 | if j \neq m+1 then

5 | DeletedKeys[j] = i

6 | let l be the smallest value greater than j for which set K_l exists

7 | K_l = K_j \cup K_l, destroying K_j

8 | return DeletedKeys
```

Choose an appropriate data structure and describe how to efficiently implement Deleted Sequence. Give as tight a bound as you can on the worst case running time of your implementation. For full credit, you should choose the best data structure.

### 1 Main Idea

The key insight is to use a Union-Find data structure with path compression to efficiently manage the sets  $K_j$  and implement the merging operations. Each element is represented as a node in the Union-Find structure, with the root of each set serving as its representative.

### 2 Data Structure

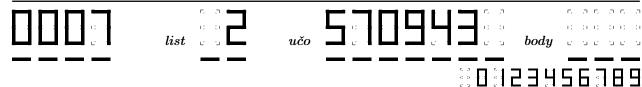
We will use Union-Find with:

- Path compression for Find operations.
- Union-by-rank for merging sets.
- An additional array to track which set  $(K_j)$  each element belongs to.

### 3 Implementation

Below is the pseudocode for the algorithm:

Name: Andrii Bezrukov UČO: 570943



```
1 function DELETEDSEQUENCE(m, n, S)
       // Initialize data structures
       Parent[1...n] \leftarrow (1,2,...,n)
 \mathbf{2}
       \mathtt{Rank[1...n]} \leftarrow (0,0,\ldots,0)
 3
       SetID[1...n]
 4
       \texttt{DeletedKeys}[1...m] \leftarrow (0,0,...,0)
 5
       j \leftarrow 1
 6
       foreach operation op in S do
           if op is Insert(x) then
 8
              SetID[x] \leftarrow i
 9
           else if op is DeleteMin then
10
            j \leftarrow j+1
11
       // Define Find with path compression
       function FIND(x)
12
           if Parent[x] \neq x then
13
              Parent[x] \leftarrow FIND((Parent[x]))
14
           else
15
              return Parent[x]
16
          return Parent[x]
17
       // Define Union by rank
       function UNION(x, y)
18
           rootX \leftarrow \text{FIND}(x)
19
           rootY \leftarrow FIND(y)
20
           if rootX = rootY then
\mathbf{21}
            return
22
           if Rank[rootX] < Rank[rootY] then
23
             Parent[rootX] \leftarrow rootY
24
           else if Rank[rootX] > Rank[rootY] then
25
            Parent[rootY] \leftarrow rootX
26
           else
27
               Parent[rootY] \leftarrow rootX
28
               \mathtt{Rank}[rootX] \leftarrow \mathtt{Rank}[rootX] + 1
29
       for i = 1 to n do
30
           determine j such that i \in K_i
                                                                                    // from SetID[i]
31
           if j \neq m+1 then
32
               \texttt{DeletedKeys}[j] \leftarrow i
33
               let l be the smallest value greater than j for which set K_l exists
34
               for each x with SetID[x] = j do
35
                  SetID[x] \leftarrow l
36
                  // Assume y is a representative element in K_l
                  if FIND(x) \neq FIND(y) then
37
                      UNION(x,y)
38
39
       return DeletedKeys
```

Name: Andrii Bezrukov		<b>UČO:</b> 570943	
	list		
		 ::::::::::::::::::::::::::::::::	

## 4 Time Complexity Analysis

- Initialization: O(n) time to set up the data structures and process the input sequence.
- Find operations with path compression:  $O(n \cdot \alpha(n))$  total time for all Find operations, where  $\alpha(n)$  is the inverse Ackermann function.
- Union operations:  $O(n \cdot \alpha(n))$  total time for all Union operations.
- Overall:  $O(n \cdot \alpha(n))$ , nearly linear since  $\alpha(n)$  grows extremely slowly.

# 5 Worst-Case Time Complexity

In practice, the overall time complexity is  $O(n \cdot \alpha(n))$ , which for all practical purposes can be considered almost linear—effectively O(n) because  $\alpha(n)$  is a very slowly growing function.