1. **(Extra edge)** In this problem, a rooted tree is a directed graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents. The input is a directed graph that is a rooted tree with n nodes, with one additional directed edge added. The added edge connects two different vertices, and was not an edge that already existed. Assuming that the graph is represented as a list of edges, develop an algorithm returning an edge that can be removed so that the resulting graph is a rooted tree of n nodes.
2. **(Longest consecutive subsequence)** Given an unsorted array of integers, find the length of the longest consecutive elements sequence. For example, given an array $[100, 4, 200, 1, 3, 2]$, the longest consecutive elements sequence is $[1, 2, 3, 4]$ of length 4.
3. Give a sequence of m MAKE-SET, UNION, and FIND-SET operations, $n < m/2$ of which are MAKE-SET operations, that takes $\Omega(m \log n)$ time when we use union by rank only.
4. **(Off-line minimum)** The off-line minimum problem asks us to maintain a dynamic set T of elements from the domain $\{1, 2, \ldots, n\}$ under the operations INSERT and EXTRACT-MIN. We are given a sequence S of n INSERT and m EXTRACT-MIN calls, where each key in $\{1, 2, \ldots, n\}$ is inserted exactly once. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically, we wish to fill in an array $extracted[1 \ldots m]$, where for $i = 1, 2, \ldots, m$, $extracted[i]$ is the key returned by the rth EXTRACT-MIN call. The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys.

   (a) In the following instance of the off-line minimum problem, each INSERT is represented by a number and each EXTRACT-MIN is represented by the letter E :
   4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5 .
   Fill in the correct values in the extracted array.

Algorithm 1 OFF-LINE-MINIMUM(m, n)

```
1: for $i \leftarrow r - 1$ to $n$ do
2:     determine $j$ such that $i \in K_j$
3:     if $j \neq m + 1$ then
4:         $extracted[j] \leftarrow i$
5:         let $l$ be the smallest value greater than $j$ for which set $K_l$ exists
6:         $K_l \leftarrow K_j \cup K_l$, destroying $K_j$
7:     end if
8: end for
```

To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by a sequence $I_1, E, I_2, E, \ldots, I_m, E, I_{m+1}$ where each E represents a single EXTRACT-MIN call and each I_j represents a (possibly empty) sequence of INSERT calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following.

(b) Argue that the array extracted returned by OFF-LINE-MINIMUM is correct.

(c) Describe how to use a disjoint-set data structure to implement OFFLINE- MINIMUM efficiently. Give a tight bound on the worst-case running time of your implementation.

5. **(Tarjan's off-line least-common-ancestors algorithm)** The least common ancestor of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T. In the off-line least-common-ancestors problem, we are given a rooted tree T and an arbitrary set $P = \{(u, v)\}$ of unordered pairs of nodes in T, and we wish to determine the least common ancestor of each pair in P. To solve the off-line least-common-ancestors problem, the following procedure performs a tree walk of T with the initial call $LCA(root[T])$. Each node is assumed to be colored WHITE prior to the walk.

Algorithm 2 $LCA(u)$

```
1:  $MAKE-SET(u)$
2:  $ancestor[FIND-SET(u)] \leftarrow u$
3:  for each child $v$ of $u$ in $T$ do
4:      $LCA(v)$
5:      $UNION(u, v)$
6:      $ancestor[FIND-SET(u)] \leftarrow u$
7:      $color[u] \leftarrow BLACK$
8:  end for
9:  for each node $v$ such that $(u, v) \in P$ do
10:     if $color[v] == BLACK$ then
11:         print The least common ancestor of $u$ and $v$ is $ancestor[FIND-SET(v)]$
12:     end if
13: end for
```

(a) Argue that line 11 is executed exactly once for each pair $(u, v) \in P$.

(b) Argue that at the time of the call $LCA(u)$, the number of sets in the disjoint-set data structure is equal to the depth of u in T.

(c) Prove that LCA correctly prints the least common ancestor of u and v for each pair $(u, v) \in P$.

(d) Analyze the running time of LCA.