Indian Institute of Technology, Guwahati

Department of Computer Science and Engineering

Data Structure Lab (CS210)

Assignment: 7

Date: 6th October, 2016.

Total Marks: 30 (offline assignments)

Deadline of Offline Assignment Submission: 19th October, 2016.

Submission Instructions:

- Name of every file should be <Roll_No>_<Question_No>.c. eg. 150101086_1.c, 150101086_2.c and so on.
- Name of the folder you upload should be <Roll_No> eg. 154101086.
- Please do not upload files like a.out, desktop.ini,etc. If there are 'N' questions in assignment, there must be exactly 'N' ".c" or ".cpp" files in the folder. Use only zip to compress the folder to be uploaded.
- You will be awarded ZERO mark if the above rules are violated.
- Please note that you will not be able to submit once the dead line (with 12 hours' late submission) is over.

Offline Assignments:

1. Implement MAKE_SET, FIND_SET, UNION operations using linked list representation of disjoint sets and the weighted union heuristic. Assume that each object x has an attribute rep[x] pointing to the representative of the set containing x and that each set S has attributes head[S], tail[S], and size[S]. The input to your program is K distinct integers numbers. First create K disjoint sets one for each input elements. Next, there would be a menu based implementation for FIND_SET and UNION operations with exit option. For FIND_SET(x) operation, output the set representative of x. For UNION operation, output the resultant disjoint sets. **(10)**
   Input:
   K //<No of Distinct elements>
   <Next K lines contain K input numbers (distinct integers) >

2. Implement MAKE_SET, FIND_SET, UNION operations using Disjoint-set forests and union by rank and path compression heuristics. The input to your program is K distinct integers numbers. First create K disjoint sets one for each input elements. Next, there would be a menu based implementation for FIND_SET and UNION operations with exit option. For FIND_SET(x) operation, output the set representative of x. For UNION operation, output the resultant disjoint sets. **(10)**

3. OFF-LINE-MINIMUM

The ***off-line minimum problem*** asks us to maintain a dynamic set *T* of elements from the domain {1, 2, . . . , *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, . . . , *n*} is inserted exactly once. Note that INSERT and EXTRACT_MIN operations are interleaved in the sequence. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically, we wish to fill in an array *extracted*[1 . . *m*], where for *i* = 1, 2, . . . , *m*, *extracted*[*i*] is the key returned by the *i*th 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.

Implement OFF-LINE-MINIMUM efficiently using disjoint-set data structure. **(10)**

Sample Input:

4, 8, E, 3, E, 9, 2, 6, E, E, E, 1, 7, E, 5.

(In the above instance of the off-line minimum problem, each INSERT is represented by a number and each EXTRACT_MIN is represented by the letter E)