National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B. Tech.(CSE)-Monsoon 2024
CS2091E Data Structures Laboratory
Assignment 4

Submission deadline (on or before): 04/09/2024, 11:59 PM

Policies for Submission and Evaluation:

- You must submit all the solutions of this assignment following the below-mentioned guidelines in the Eduserver course page, on or before the submission deadline.
- Ensure that your programs will compile and execute using GCC compiler without errors. The programs should be compiled and executed in the SSL/NSL.
- During the evaluation, failure to execute programs without compilation errors may lead to zero marks for that evaluation.
- Your submission will also be tested for plagiarism, by automated tools. In case your code fails to pass the test, you will be straightaway awarded zero marks for this assignment and considered by the examiner for awarding an F grade in the course. Detection of ANY malpractice related to the lab course can lead to awarding an F grade in the course.

Naming Conventions for Submission

- Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip`

  (Example: *ASSG1_BxxyyyyCS_LAXMAN.zip*). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.
- The source codes must be named as

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example *ASSG1_BxxyyyyCS_LAXMAN_1.c*). If you do not conform to the above naming conventions, your submission might not be recognized by our automated tools and hence will lead to a score of 0 marks for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

- Violation of academic integrity will be severely penalized. Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign an F grade in the course. The department policy on academic integrity can be found at: https://minerva.nitc.ac.in/?q=node/650.

Questions

1. You are given a Binary Max Heap H with unique keys. Perform the following sequence of operations on H where each key is an integer $n \in [1, 10^6]$. Implement a menu-driven program with the following operations:

Operations

(a) $InsertKey(H, n)$: Insert the given key n into the Max Heap H. After insertion, Perform the necessary operations to maintain the heap property. Duplicate keys are not allowed; if n already exists, skip the insertion.

(b) $FindMax(H)$: return and print the element of a Max heap H with the largest key.

(c) $ExtractMax(H)$: Extract and print the maximum key from the Max Heap H. After extraction, Perform the necessary operations to maintain the heap property and print the max-heap as a space-separated list of elements in level-order traversal.

(d) $KthLargest(H, k)$: Find and print the k-th largest element in the max-heap H without removing it. If k is greater than the number of elements in the heap, print -1.

(e) $DeleteKey(H, x)$: Delete the key x from the max-heap H. Perform the necessary operations to maintain the heap property. After deletion, print the max-heap as a space-separated list of elements in level order traversal. If H has one element that element is x, print the empty heap as 0. If the key x is not present, print -1.

(f) $ReplaceKey(H, old\_val, new\_val)$: Replace the value *old_val* in the max-heap H with *new_val*. If *new_val* is lesser than *old_val*, ensure that the heap property is maintained by performing necessary operations. If *old_val* is not present, print the absolute difference between *old_val* and *new_val*. Otherwise, After replacement, print the max-heap as a space-separated list of elements in level-order traversal.

Input Format

- Each line contains a character from {'`a`', '`b`', '`c`', '`d`', '`e`', '`f`', '`g`'} followed by zero or more positive integers n.
- Character *'a'* is followed by a positive integer n. Perform the *InsertKey(H, n)* operation.
- Character *'b'* performs the *FindMax(H)* operation.
- Character *'c'* performs the *ExtractMax(H)* operation.
- Character *'d'* is followed by a positive integer k. Perform the *KthLargest(H, k)* operation.
- Character *'e'* is followed by a positive integer x. Perform the *DeleteKey(H, x)* operation.
- Character *'f'* is followed by two positive integers *old_val* and *new_val* separated by a space. Perform the *ReplaceKey(H, old_val, new_val)* operation.
- Character `'g'` is to terminate the sequence of operations.

Output Format

- The output (if any) of each command should be printed on a separate line. However, no output is printed for `'a'` and `'g'`.
- For Option *'b'*: Print the maximum key in the heap H. If the heap is empty, print -1.

- For Option *'c'*: Print the maximum key extracted from the heap H. After extraction, print the max-heap as a space-separated list of elements in level-order traversal. If the heap is empty, print -1.
- For Option *'d'*: Print the k-th largest element in the max-heap H. If k is greater than the number of elements in the heap, print -1.
- For Option *'e'*: Print the max-heap as a space-separated list of elements in level-order traversal after deleting the key x. If H has one element and that element is x, print the empty heap as 0. If the key x is not present, print -1.
- For Option *'f'*: After replacement, print the max-heap as a space-separated list of elements in the level-order traversal. If *old_val* is not present in the heap, Print the absolute difference between *old_val* and *new_val*.

Sample test case 1

Input:

```
b
a 15
a 10
a 20
a 17
a 30
b
c
d 7
d 2
e 10
f 17 25
f 10 30
b
g
```

Output:

```
-1
30
30 20 17 15 10
-1
17
20 17 15
25 20 15
20
25
```

2. You are tasked with developing a patient management system for an emergency room using a priority queue PQ implemented as a min-heap. Each patient has a unique identifier *patient_id*, a name *name*, and a severity level *severity*. The priority queue is used to manage patients based on the severity of their condition. The emergency room has the following rules:

   - Lower priority values indicate more severe conditions (e.g., a priority of 1 is more urgent than a priority of 5).
   - Each patient has a unique priority, ensuring no two patients have the same severity level.

   The system must handle various operations efficiently to manage the patient queue, ensuring that the heap property is maintained after each operation. The operations are described below:

   Operations

   (a) $AdmitPatient(PQ, patient\_id, name, severity)$:
   - Admit a new patient with *patient_id*, *name* and a *severity* level into the priority queue *PQ*.
   - After each admission, print the current state of the priority queue as a space-separated list of *patient_id*s in level-order traversal (based on severity).

   (b) $TreatPatient(PQ)$:
   - Treat and remove the patient with the most severe condition (i.e., the patient with the lowest severity value) from the priority queue *PQ*.
   - Print the *patient_id* and *name* of the patient that was treated. Given that no two patients have the same severity, only one patient will be treated.
   - If the priority queue is empty, i.e., if there are no patients to treat, print `-1`

   (c) $UpdateSeverity(PQ, patient\_id, new\_severity)$:
   - Update the severity of an existing patient identified by *patient_id* to *new_severity*. Ensure that the new severity is unique. If the patient is absent in the queue, print `-1.`
   - After updating the severity, print the updated priority queue in level-order traversal to reflect the changes.

   (d) $QueryPatient(PQ, patient\_id)$:
   - Check if a patient with *patient_id* is in the queue and print their *name* and *severity* level. If the patient is not found, print `-1`

   (e) $FindMostSevere(PQ)$:
   - Print the *patient_id name* and *severity* of three patients with the most severe condition (i.e., three patients with the lowest severity value) without removing them from the queue. If there are fewer than three patients, print as many as available. If the queue is empty, print `-1`

   Input Format

   - Each line contains a character from {'`a`', '`b`', '`c`', '`d`', '`e`'} followed by at-most two positive integers n.
   - Character `'a'` is followed by a space separated two positive integers, *patient_id* and *severity* followed by a string separated by a space. Perform the *AdmitPatient(PQ, patient_id, name, severity)* operation.
   - Character `'b'` performs the *TreatPatient(PQ)* operation.
   - Character `'c'` is followed by a space separated two positive integers, *patient_id* and *new_severity*. Perform the *UpdateSeverity(PQ, patient_id, new_severity)* operation.
   - Character `'d'` is followed by a positive integer, *patient_id*. Perform the *QueryPatient(PQ, patient_id)* operation.
   - Character `'e'` performs the *FindMostSevere(PQ)* operation.

Output Format

- The output (if any) of each command should be printed on a separate line. However, no output is printed for `‘g’`.
- For option `’a’`: Print the current state of the priority queue after the admission in level-order traversal.
- For option `’b’`: Print the *patient_id* and *name* of the patient that was treated, or `-1` if the queue is empty.
- For option `’c’`: Print the updated priority queue in level-order traversal, or `-1` if the patient is not found.
- For option `’d’`: Print the *name* and *severity* of the patient, or `-1` if the patient is not found.
- For option `’e’`: Print the *patient_id*, *name* and *severity* of three most severe patients in the increasing order of severity (space-separated). Each patient’s details should be printed in a new line. If there are fewer than three patients, print as many as available in separate lines. Print `-1` if the queue is empty.

Sample test case 1

Input:

```
a 101 5 Asma
a 102 3 Sajeena
a 103 4 Sourav
b
c 103 2
d 101
a 105 1 Ruchi
a 107 3 Parth
a 120 4 Rohan
e
d 102
```

Output:

```
101
102 101
102 101 103
102 sajeena
103 101
asma 5
105 101 103
105 107 103 101
105 107 103 101 120
105 Ruchi 1
103 Sourav 2
107 Parth 3
-1
```

3. You are tasked with designing a job dispatcher using a priority queue *Que* implemented as a min-heap. The job dispatcher selects a job to be executed from a pool of jobs based on its priority. Each job has a unique job-id *JID* ($1 \leq JID \leq 100$) and a priority p ($1 \leq p \leq 10$).Only one job can be executed at a time, and the pool of jobs can be updated with more jobs as they arrive. New jobs are added to the pool upon arrival and are removed from the pool when they are scheduled for execution.

   From the pool, the job dispatcher selects the job with the highest priority for execution. Job with the lowest priority value has higher priority(e.g., a priority of 1 is higher than a priority of 5). The selected job is then removed from the pool after execution.

   Note:

   - The priority queue should be implemented using a min-heap, where jobs with lower numerical priority values are given precedence.
   - The *JID* values are unique. There are no two jobs with the same priority *p*.

   Your program should be implemented using the following functions:

   Operations

   (a) *Add(Que, JID, p):* It adds a new job with job-id *JID* and priority *p* to the pool of jobs.

   (b) *Schedule(Que):* It schedules the job with the highest priority from the pool of jobs for execution and then removes it from the pool of jobs. Print *-1* if the pool is empty.

   (c) *Next-job(Que):* It displays the *JID* of the next job to be executed, which will be the job with the highest priority from the pool of jobs. The job is not removed from the pool. Print *-1* if the queue is empty.

   (d) *Replace-priority(Que, JID, np):* It replaces the priority of job *JID* in the pool of jobs with the new priority *np*. If the *JID* is not in the pool of jobs, print *-1*. After updating the priority, the job should be repositioned within the heap to maintain the min-heap property.

   (e) *Display(Que):* It prints the details (*JID* and *p*) of each job in the pool of jobs. Print *-1* if the pool is empty.

   Input Format:

   - Each line contains a character from {'a', 'b', 'c', 'd', 'e', 'g'} followed by zero or more positive integers *n*.
   - Character 'a' is followed by two positive integers *JID* and *p*. Perform the *Add(Que, JID, p)* operation.
   - Character 'b' performs the *Schedule(Que)* operation.
   - Character 'c' performs the *Next-job(Que)* operation.
   - Character 'd' is followed by two positive integers *JID* and *np*. Perform the *Replace-priority(Que, JID, np)* operation.
   - Character 'e' performs the *Display(Que)* operation.
   - Character 'g' is to terminate the sequence of operations.

   Output Format:

   - The output (if any) of each command should be printed on a separate line. However, no output is printed for 'a' and 'g'.
   - For Option 'b': Print the *JID* of the scheduled job. If the pool is empty, print `-1`.
   - For Option 'c': Print the *JID* of the next job to be executed. If the pool is empty, print `-1`.

- For Option ‘d’: After replacing the priority, if the *JID* is found in the pool, no output is required. If the *JID* is not found, print `-1`.
- For Option ‘e’: Print the job details (*JID* and *p*) of each job as a space-separated list of elements in the level-order traversal. Where each pair of (JID p) should be printed in a new line. If the pool is empty, print `-1`.

Sample test case 1

Input:

```
c
a 18 3
a 17 2
a 10 4
a 16 1
a 9 5
e
c
b
d 20 1
d 9 6
e
c
g
```

Output:

```
-1
16 1
17 2
10 4
18 3
9 5
16
16
-1
17 2
18 3
10 4
9 6
17
```