National Institute of Technology Calicut
Department of Computer Science and Engineering
Fourth Semester B. Tech.(CSE)-Winter 2023-2024
CS2094D Data Structures Laboratory
Assignment Cycle#1
Part C

Submission deadline (on or before): 08.02.2024, 11:00 PM.

Policies for Submission and Evaluation:

- You must submit your assignment in the Eduserver course page, on or before the submission deadline.
- Ensure that your programs will compile and execute without errors using gcc compiler.
- 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 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

  `ASSGC<NUMBER>_<PART>_<ROLLNO>_<BATCHNO>_<FIRST-NAME>.zip`

  (Example: $ASSGC1\_A\_BxxyyyyCS\_CS01\_LAXMAN.zip$). 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>_<PART>_<ROLLNO>_<BATCHNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example: $ASSGC1\_A\_BxxyyyyCS\_CS01\_LAXMAN\_1.c$). 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 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. Given two integer arrays **A** and **B**, implement the following functions on them using **hashing**:

   (a) union(**A**,**B**) or union(**B**,**A**): prints the **union** of the elements of A and B (or B and A) in the order of appearance of elements in A followed by B (or B followed by A).

   (b) intersection(**A**,**B**) or intersection(**B**,**A**): prints the **intersection** of the elements of A and B (or B and A) in the order of appearance of elements in A followed by B (or B followed by A).

   (c) setDifference(**A**,**B**) or setDifference(**B**,**A**): prints the set difference of the elements of A and B (or B and A) in the order of appearance of elements in A (or B).

Input Format:

- The first line contains two integers **m** and **n** (space-separated), which tell the size of input arrays **A** and **B**.
- The second line contains m positive integers (space-separated) in input array **A**.
- The third line contains n positive integers (space-separated) in input array **B**.

The remaining inputs are as follows:

- Each line of input contains three characters separated by a space: the first character from the menu list **['u','i','s','e']**, the second, and third characters specify the arrays A, B, or (B,A).
- Input '**u**' followed by '**A**' '**B**'(or '**B**' '**A**') calls the function union(**A**,**B**) or union(**B**,**A**), which prints the union of the elements of A and B (or B and A) in the order of appearance of elements in A followed by B (or B followed by A).
- Input '**i**' followed by '**A**' '**B**'(or '**B**' '**A**') calls the function intersection(**A**,**B**) or intersection(**B**,**A**), which prints the intersection of the elements of A and B (or B and A) in the order of appearance of elements in A (or B).
- Input '**s**' followed by '**A**' '**B**'(or '**B**' '**A**') calls the function setDifference(**A**,**B**) or setDifference(**B**,**A**), which prints the set difference of the elements of A and B (or B and A) in the order of appearance of elements in A (or B).
- Input '**e**' terminates the execution of the program.

Output Format:

- A line may contain **-1** if there are no elements to print.
- The output of the result of any menu is printed in a new line with a space between each integer.

Constraints:

- Do not use **sorting.**
- Do not consider **duplicate** elements.
- The hash table size should not be greater than **2*max(m, n)**. Use **open addressing** for handling collisions in hashing.
- 1<=A[i],B[i] <=100

Sample Input 1:

```
5 6
1 2 3 4 5
2 5 1 6 7 8
u A B
u B A
i A B
i B A
s A B
```

s B A
e

Sample Output 1:

1 2 3 4 5 6 7 8
2 5 1 6 7 8 3 4
1 2 5
2 5 1
3 4
6 7 8

Sample Input 2:

6 7
1 1 2 2 3 3
3 3 2 2 1 6 5
u A B
u B A
i A B
i B A
s A B
s B A
e

Sample Output 2:

1 2 3 6 5
3 2 1 6 5
1 2 3
3 2 1
-1
6 5

2. You are tasked with implementing a hash table using chaining as the collision resolution method. The hash table should support the following functions:

   (a) insert(**hashTable,key**): Inserts the (integer) **key** at a certain **index** obtained from the hash function below, if there are any collisions, use chaining. Maintain the **sorted order** while chaining. If the key is already present in the hash table, then print -1.

   index=(key)Mod(TableSize) or key%TableSize.

   (b) search(**hashTable,key**): Searches for the **key** in the hash table. If the **key** is found, prints(space-separated) the **index** at which it is stored in the hash table and also its **position** in the chain. If the **key** is not found then print -1.

   Eg: search(15)

   Hash Table: $< index, chain >$
   .
   .
   $4 : 8 -> NULL$
   $5 : 5 -> 15 -> 25 -> NULL$
   .
   .

o/p: 5(index) 2(position in chain)

(c) delete(**hashTable,key**): Deletes the **key** from the hash table and prints(space-separated) the **index** at which it is stored in the hash table and also its **position** in the chain. If the **key** is not found then print -1.

(d) update(**hashTable,oldKey,newKey**): Updates the **oldKey** in the hash table with **newKey** by deleting the **oldKey** and inserting the **newKey** and prints(space-separated) the **index** at which the **oldKey** is stored in the hash table and also its **position** in the chain. If the **oldKey** is not found or the **newKey** is already present in the hash table then print -1.

(e) printElementsInChain(**hashTable,index**): prints all the elements(in sorted order, space-separated) present in the chain at **index** in the hash table. If the slot is empty print -1.

Input Format:

- The first line contains an integer **(TableSize)**, which tells the hash table size.
- Each line of input starts with a **character** from the menu list [**'i','d','u','s','p','e'**] followed by an integer(s), which specify the **key(s)** or **Index,** respectively.
- Input **'i'** followed by an integer **(key)** calls the function insert**(hashTable,key).**
- Input **'d'** followed by an integer **(key)** calls the function delete**(hashTable,key).**
- Input **'u'** followed by two integers **oldKey** and **newKey** calls the function update**(hashTable,oldKey, newKey).**
- Input **'s'** followed by an integer**(key)** calls the function search**(hashTable,key).**
- Input **'p'** followed by an integer**(0 <=index<TableSize)** calls the function printElementsIn-Chain**(hashTable,index)**.
- Input **'e'** terminates the execution of the program.
- All the inputs in a line are separated by space and the integer inputs are positive numbers.

Output Format:

- A line may contain -1.
- The output of the result of any menu is printed in a new line with a space between each integer.

Sample Input 1:

```
5
i 1
i 5
i 10
i 15
i 20
p 0
s 15
d 15
u 15 25
p 4
e
```

The chain position is 1-based indexing

you have to use Linked List for Chaining. The reason is, we won't be knowing how many elements will be inserted in each slot. There will be lots of wastage of space.

Sample Output 1:

```
5 15 20
0 2
0 2
-1
-1
```

```
5 10 15 20
0 3
0 3
-1
-1
```

Sample Input 2:

```
7
i 1
i 2
i 3
i 4
i 5
i 12
p 5
u 12 25
p 5
u 12 25
u 5 25
d 25
s 12
i 5
e
```

Sample Output 2:

```
5 12
5 2
5
-1
-1
3 1
-1
-1
```

```
5 12
5 2
5
-1
-1
4 2
-1
-1
```

3. Our institute decided to split the students into four groups to conduct an event. The procedure for splitting is as follows:

   h(A) = (Sum of ASCII value of the characters in the first name of 'A' + age of 'A') % 4, where 'A' represents a student.

   Eg:- h(Veena) = (86+101+101+110+97+19) % 4 = 2; where, 86 - ASCII(V), 101 - ASCII(e), 110 - ASCII(n), 97 - ASCII(a), 19 - age(Veena).

   If h(A) =0, the student is placed in group 0; if h(A) = 1, the student is placed in group 1; if h(A) = 2, the student is placed in group 2, and if h(A) = 3, the student is placed in group 3.

   Write a program to perform the operations *count the number of students in each group*, *the student list in a group as per the order of insertion*, and *the student list who belongs to the same branch in a group*.

   Note: *Assume all the students are from CS, EC, EE, or CE branches.*

   Input format:

   - First line of the input contains an integer 'n' $\in [1, 10^3]$, the total number of students who are participating in the event.
   - Next 'n' consecutive line contains: *first_name*, *roll_number*, and *age*; separated by single space.
   - The input contains a character 'c' followed by an integer 'k' $\in [0, 3]$ to display the count and student list of the group 'k'.
   - The input contains an integer 'm' $\in [0, 3]$, representing the group number, followed by two characters representing the branch name (both uppercase and lowercase are considered the same branch).

- The input contains a character 'e' to represent the end of the input.

Output format:

- The output (if any) of each command should be printed on a separate line.
- For input lines starting with the character 'c' followed by an integer 'k' $\in [0, 3]$, prints an integer 'x' followed by x number of strings separated by a space.
- For input lines starting with an integer 'm' $\in [0, 3]$ followed by two characters, prints the strings (first_name) separated by a space, if any student exit in the group 'm'. Otherwise, print -1.

Sample Input:

```
4
Veena B220016EC 19
Abu B210051CS 21
Ishan B190016CE 22
Aleena B200036EE 21
c 0
1 CS
c 1
c 2
c 3
3 EC
2 ec
e
```

Incase of c option where the names of students in a particular group have to be printed, what should we do if there is no one in the group? Should we print 0 or -1?

Response: Print 0

Sample Output:

```
0
Abu
2 Abu Ishan
1 Veena
1 Aleena
-1
Veena
```

4. Open addressing is a method for handling collisions in hashing. The three different methods for open addressing are linear probing, quadratic probing, and double hashing. A brief description of the three methods is given below:
In linear probing, the function used to calculate the next location during collision is: $h'(k) = (h(k) + i) \mod m, i = 1, 2 \ldots$.
In quadratic probing, the function used to calculate the next location during collision is: $h'(k) = (h(k) + i^2) \mod m, i = 1, 2 \ldots$
In double hashing scheme, the primary hash function is, $h1(k) = k \mod N$, where N is the table size. The secondary hash function is, $h2(k) = R - (key \mod R)$ where R is the maximum prime number less than the table size. Double hashing can be done using: $(h1(key) + i * h2(key)) \mod N, i = 0, 1, 2 \ldots$.
Given a set of keys and the table size, write a program to print the locations at which the keys are stored using the above-mentioned three methods and also print the total number of collisions that occur during mapping for each of the three methods.

Input format:

- First line of the input contains an integer, the table size.
- Second line contains space-separated(single space) integer numbers, the keys to be inserted.

Output format:

- First line of the output contains space-separated (single space) integers, the locations obtained using linear probing.
- Second line contains an integer, the total number of collisions that occurred during linear probing.
- Third line of the output contains space-separated (single space) integers, the locations obtained using quadratic probing.
- Fourth line contains an integer, the total number of collisions that occurred during quadratic probing.
- Fifth line of the output contains space-separated (single space) integers, the locations obtained using double hashing.
- Sixth line contains an integer, the total number of collisions that occurred during double hashing.

Sample Input:

7
76 93 40 47 10 55

Sample Output:

6 2 5 0 3 1
4
6 2 5 0 3 1
6
6 2 5 1 3 4
2

5. Given an array **A** of N integers and an integer K, return the count of distinct integers in all windows of size K.

Input Format:

- The first line contains two integers **N** and **K** (space-separated), **N** giving size of input arrays **A** and **K** indicating the size of the window.
- The second line contains N positive integers (space-separated) in input array **A**.

Output Format:

- First Line Contains Space Separated Count of distinct integers in each window of size K.

Constraints:

- Time Complexity Should be O(N).

Sample Input 1:

10 4
2 8 2 3 4 5 5 6 5 2

Sample Output 1:

3 4 4 3 3 2 3