Indian Statistical Institute

Semester-I 2018-2019

M.Tech.(CS) - First Year

Lab Test IVa (06 November, 2018)

Subject: Data and File Structures Laboratory

Total: 60 marks Duration: 4 hrs.

SUBMISSION INSTRUCTIONS

1. Naming convention for your programs: `cs18xx-test4a-progy.c`
   IMPORTANT: Insert a single alpha-numeric string of your choice, 6-8 characters long, in the name given above as shown in the examples below. Think of this string as something like a security password, except that you are not required to remember the string. Examples: `cs1840-assign3-x19jdh4-prog1.c`, `ppo03wws-cs1840-assign3-prog2.c`, `cs1840-assign3-prog2-jsiwm7de.c`
2. When you have finished, copy all your files to `~dfslab/2018/labtest4a/cs18xx/`.

1. **(25 marks)** Recent developments in multi-core processors and other mainstream hardware like graphics processor units (GPUs) allow the parallelization of multiple processes. This can be efficiently handled by employing parallel algorithms through constructing hash tables. Suppose we plan to design a practical parallel scheme for constructing hash tables on GPUs motivated by the cuckoo hashing technique. For this, we need to hash the different processes based on their process IDs into these hash tables constructed in the GPUs.

   Let us assume that there are n GPUs (indexed as $1, 2, \ldots, n$) that corresponds to separate hash tables. These hash tables contain different sets of process IDs. The GPUs adopt the cuckoo hashing approach for resolving the collision between the process IDs that are to be stored. The assignment of process IDs starts with the 1st GPU. If there is a hash collision on the ith GPU, then the new process (recognized by ID) takes the place of collision (in the hash table), and it finds a place for the old process (recognized by ID) in the $(i \pmod n) + 1$th GPU to resolve the collision, and so on. If we revisit the same place with the same process ID to insert then its called a *hash cycle*. In case of a hash cycle, we consider that the process is placed in none of the n GPUs available. Note that, the old processes have a higher chance of falling into a hash cycle in case of a collision.

   Consider that a family of hash functions are applied on these GPUs. The hash function to be applied on the ith GPU having a hash table of size m, for a process ID k, is given by

   $$h_i(k) = \lfloor k/i \rfloor \pmod m.$$

   Write a program that takes the number of GPUs, size of hash tables on each GPU, and a set of process IDs as user inputs and returns the index of corresponding GPUs into which the input processes are hashed. For simplicity, let us assume that the hash tables have the same size on each GPU.

   ### Input Format

   Input will be provided via standard input in the following format. The first line of input consists of two integers, namely the number of GPUs (n) and the size of hash tables on each GPU (m). It follows by a set of process IDs to be hashed into.

Output Format

Output is to be printed on the standard output in the following format. The output will print the index of corresponding GPUs in which the processes get hashed into. If a process experiences a hash cycle then it simply prints a '−1'.

Sample Input 0

```
1 10
118 216 324 445 566 623 791 812
```

Sample Output 0

```
1 -1 1 1 1 1 1 1
```

Explanation:

The final hash table appears as follows.

GPU 1		791	812	623	324	445	566		118	

- The process ID 566 takes the position of 216 (in GPU 1) to resolve the hash collision.
- The process with ID 216 experiences a hash cycle.

Sample Input 1

```
2 8
12 16 134 563 211 3 5
```

Sample Output 1

```
1 1 1 -1 2 1 1
```

Explanation:

The final hash tables appear as follows.

GPU 1	16			3	12	5	134	
GPU 2		211						

- The process ID 211 takes the position of 563 (in GPU 1) to resolve the hash collision.
- The process ID 3 takes the position of 211 (in GPU 1) to resolve the hash collision.
- The process ID 211 takes the position of 563 (in GPU 2) to resolve the hash collision.
- The process with ID 563 experiences a hash cycle.

Sample Input 2

```
3 10
0 1 2 3 4 5 6 7 8 9 10 11 12 13
```

Sample Output 2

```
-1 2 3 2 1 1 1 1 1 1 1 1 1 1
```

Explanation:

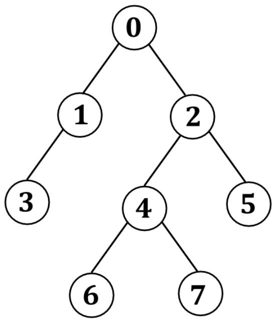
The final hash tables appear as follows.

GPU 1	10	11	12	13	4	5	6	7	8	9
GPU 2	1	3								
GPU 3	2									

- The process ID 10 takes the position of 0 (in GPU 1) to resolve the hash collision.
- The process ID 11 takes the position of 1 (in GPU 1) to resolve the hash collision.
- The process ID 1 takes the position of 0 (in GPU 2) to resolve the hash collision.
- The process ID 12 takes the position of 2 (in GPU 1) to resolve the hash collision.
- The process ID 13 takes the position of 3 (in GPU 1) to resolve the hash collision.
- The process ID 3 takes the position of 2 (in GPU 2) to resolve the hash collision.
- The process ID 2 takes the position of 0 (in GPU 3) to resolve the hash collision.
- The process with ID 0 experiences a hash cycle.

2. **(20 marks)** Suppose a first-n natural binary tree (FnBT) is defined as a binary tree having n nodes that contains the first n natural numbers as its data items. Consider a special representation of such FnBTs. In this representation, a FnBT is given in the form of an array A such that the parent of node i is given by $A[i]$. Note that, there is a correspondence between the index of elements in A and the data items in the tree. For the root node, the parent is denoted by '-1'.

   As for example, the FnBT corresponding to the array representation $A = \{-1, 0, 0, 1, 2, 2, 4, 4\}$ looks like the following.



Write a program to construct the conventional representation of an input FnBT given in the aforementioned form of an array. Recall that the conventional representation of a binary tree can be represented as shown below.

```
typedef struct treeNode{
    DATA d; // Generic data
    struct treeNode *leftChild, *rightChild;
    struct treeNode *parent; // This is optional
}BTNODE;
```

The program should also compute the maximum width of the given binary tree. The maximum width of a tree is the maximum number of nodes at any level, where a level corresponds to all nodes that are at the same distance from the root.

Input Format

Input will be provided via standard input in the following format. The input consists of a set of integers corresponding to the elements of the array A in the order of their appearance.

Output Format

Output is to be printed on the standard output in the following format. The output prints the maximum width of the input FnBT. If the input is not a binary tree then it simply prints a '-1'.

Sample Input 0

```
-1 0 0 1 2 2 4 4
```

Sample Output 0

```
3
```

Sample Input 1

```
-1 0 0 1 1 2 2 3 3 4 4 5 5 6 6
```

Sample Output 1

```
8
```

Sample Input 2

```
-1 0 0 1 2 2 2 4
```

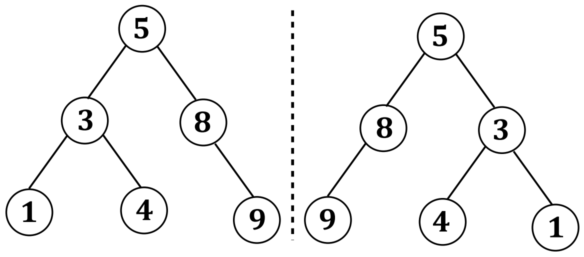
Sample Output 2

```
-1
```

3. **(15 marks)** A binary tree is termed as a *mirror twin* of a binary search tree (BST) if both of them comprises the same set of data items and they are structurally mirror images to each other.

   Given the preorder traversal of a BST and the inorder and preorder traversals of a simple binary tree as user inputs, write a program to determine whether the binary tree is a mirror twin of the BST or not.

   As for example, the binary tree shown below (in the rightside) is a mirror twin of the BST given below (in the leftside).



Input Format

Input will be provided via standard input in the following format. The first line of input consists of two integers, namely the number of data items in both the input trees. It follows by three more input lines. These lines consist of sets of integers corresponding to the data items obtained from the preorder traversal on the BST, followed by the inorder and preorder traversals on the binary tree, respectively.

Output Format

Output is to be printed on the standard output in the following format. The output simply prints 'MIRROR TWIN' or 'NOT MIRROR TWIN' corresponding to whether the input binary tree is a mirror twin to the BST or not.

Sample Input 0

```
6 6
5 3 1 4 8 9
9 8 5 4 3 1
5 8 9 3 4 1
```

Sample Output 0

```
MIRROR TWIN
```

Sample Input 1

```
5 5
7 5 3 1 4
7 5 4 3 1
7 5 3 4 1
```

Sample Output 1

```
MIRROR TWIN
```

Sample Input 2

```
6 6
5 3 1 4 8 9
1 3 4 5 8 9
5 3 1 4 8 9
```

Sample Output 2

```
NOT MIRROR TWIN
```