DFS LAB – ASSIGNMENT 3

MTech(CS) I year 2019–2020

Deadline: 1 November, 2019

Total: 60 marks

SUBMISSION INSTRUCTIONS

1. Naming convention for your programs: `cs19xx-assign3-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: `cs1940-assign3-x19jdh4-prog1.c`, `ppo03wws-cs1940-assign3-prog2.c`, `cs1940-assign3-prog2-jsiwm7de.c`

2. To submit a file, go to the directory containing the file and run the following command from your account on the server (IP address: 192.168.64.35)

   ```
   cp -p name-of-your-file ~dfslab/2019/assign3/cs19xx/
   ```

   If you want to submit your files from a computer with a different IP address, run

   ```
   scp -p name-of-your-file \
   mtc19xx@www.isical.ac.in:/user1/perm/pdslab/2019/assign3/cs19xx/
   ```
   (enter your password when prompted).

   To submit all `.c` and `.h` files at one go, use
   `cp -p *.c *.h ~dfslab/2019/assign3/cs19xx/` or similar.

NOTE: Unless otherwise specified, all programs should take the required input from stdin, and print the desired output to stdout.

Q1. Suppose a cultural event is to be organized by the ISI Club. They have initially planned the said event in the form of a series of performances. The performances are stored by the names of corresponding performers in some arbitrary order. You need to replan the cultural event so that the total payment made to the performers is minimised.

Note that, all the performers charge the same amount but a repeated performance requires a bonus amount to be paid. The bonus amount is a function of the number of repeated performances. For example, if there are k successive performances by the same performer then the performer will get the total bonus amount of $b(k-1)$, accumulating the bonus amount of b for each of the latter $k-1$ performances.

Write a program that will take a series of names of performers from the user and rearrange those names in the best possible way (minimizing the computational complexity) such that the above criteria get satisfied. Note that, there might exist multiple correct answers but it is sufficient to return any one of those.

[20 marks]

Input Format

Input will be provided as command-line arguments as a series of strings representing the names of performers.

Output Format

Output is the revised order of the names of performers to be printed on the standard output.

Sample Input 0

```
Atif Atif Atif Atif
```

Sample Output 0

```
Atif Atif Atif Atif
```

Sample Input 1

```
Hariharan Hariharan Hariharan Arijit Shreya
```

Sample Output 1

```
Hariharan Arijit Hariharan Shreya Hariharan
```

Sample Input 2

```
Arijit Hariharan Shreya Shreya Shreya
```

Sample Output 2

```
Shreya Hariharan Shreya Arijit Shreya
```

Q2. Let $\vec{\mathbf{x}} = \langle x_1,\ x_2,\ \ldots,\ x_n \rangle$ and $\vec{\mathbf{y}} = \langle y_1,\ y_2,\ \ldots,\ y_n \rangle$ be two vectors from $\mathbb{R}^n$. Suppose that we define an operation $\circledcirc$ as follows:

$$\vec{\mathbf{x}} \circledcirc \vec{\mathbf{y}} = \langle x_1 y_1,\ x_2 y_2,\ \ldots,\ x_n y_n \rangle.$$

In other words, the $\circledcirc$-product of two vectors from $\mathbb{R}^n$ is another vector in $\mathbb{R}^n$ that is obtained by multiplying the corresponding components of $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$.

Given $\vec{\mathbf{z}} \in \mathbb{R}^n$, and a set $S \subset \mathbb{N}$, your task is to find whether there exist $\vec{\mathbf{x}}, \vec{\mathbf{y}} \in \mathbb{R}^n$ such that

- $\{x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n\}$ are all **distinct** primes from S, **and**
- $\vec{\mathbf{z}} = \vec{\mathbf{x}} \circledcirc \vec{\mathbf{y}}$.

[20 marks]

Input Format

Input will be provided via the standard input in the following format. The first line of input will specify n; the second line contains the values of z_1, z_2, ..., z_n; the third and last line contains the elements of the set S separated by spaces.

Output Format

Your program should print `TRUE` to standard output if $\vec{\mathbf{x}}$ and $\vec{\mathbf{y}}$ exist as specified above; otherwise, it should print `FALSE`.

Sample Input 0

```
2
33 35
3 4 5 7 11 10 13 2 1
```

Sample Output 0

```
TRUE
```

Sample Input 1

```
3
11 15 77
1 2 3 4 5 6 7 8 9 17 19 20
```

Sample Output 1

```
FALSE
```

Q3. A unit of program execution that accesses and possibly updates various data items in a database is known as a transaction. Thus, a transaction comprises a set of instructions. A schedule is a sequence of instructions that specify the chronological order in which instructions of concurrent transactions are executed. A schedule of transactions is `serial` when the execution of one transaction is to be completed entirely before starting another transaction. Consider the following schedule that is not a serial one but equivalent to T_2 `is followed by` T_1. Note that, the data items that are accessed by this schedule are A and B.

We can test the `conflict serializability` of a schedule through constructing precedence graphs. Let us define a precedence graph as follows.

Given the schedule S, a precedence graph is defined as a directed graph $G = (V, E)$, wherein the set of vertices V denotes all the transactions participating in S, and E consists of all the edges $T_i \rightarrow T_j$ for which one of the following three conditions holds in S:

No.	Transaction T_1	Transaction T_2
1		read(A)
2		A = A - 10
3		write(A)
4	read(A)	
5	T = A * 0.05	
6	A = A - T	
7	write(A)	
8		read(B)
9		B = B + 10
10		write(B)
11		commit
12	read(B)	
13	B = B + T	
14	write(B)	
15	commit	

(a) T_i executes write(Q) before T_j executes read(Q).

(b) T_i executes read(Q) before T_j executes write(Q).

(c) T_i executes write(Q) before T_j executes write(Q).

The precedence graph of a `conflict serializable` schedule does not contain a cycle. Write a program that can verify whether a given schedule of transactions is `conflict serializable` or not.

[25 marks]

Input Format

Input will be provided in multiple files (name to be taken from command-line arguments) in the following format.

Each file will include the instructions of a single transaction. The first line of input in each file will contain n, where n is the number of instructions in the transaction. This will be followed by n more lines. Each of these remaining lines will correspond to one instruction preceded by its instruction number in the schedule.

Output Format

Output is to be printed on the standard output. Output will print whether the schedule is `Conflict Serializable` or `Not Conflict Serializable` as defined earlier.

Command-line Arguments

```
./prog3 <input_filename1> <input_filename2> ... <input_filenameN>
```

Sample Input 0

```
Contents of input1.txt:
7
1 read(A)
2 A = A - 10
3 write(A)
8 read(B)
9 B = B + 10
10 write(B)
11 commit

Contents of input2.txt:
8
4 read(A)
5 T = A * 0.05
6 A = - T
7 write(A)
12 read(B)
13 B = B + T
14 write(B)
15 commit

Input from command-line:
./prog3 input1.txt input2.txt
```

Sample Output 0

```
Output on the terminal:
Conflict Serializable
```

Sample Input 1

```
Contents of input1.txt:
2
2 write(A)
3 read(B)
```

```
Contents of input2.txt:
3
1 read(A)
4 write(A)
5 read(B)

Contents of input3.txt:
3
6 write(A)
7 read(B)
8 write(B)

Input from command-line:
./prog3 input1.txt input2.txt input3.txt
```

Sample Output 1

```
Output on the terminal:
Not Conflict Serializable
```