Indian Statistical Institute

Semester-I 2018-2019

M.Tech.(CS) - First Year

Lab Test III (09 October, 2018)

Subject: Data and File Structures Laboratory

Total: 60 marks Duration: 4 hrs.

SUBMISSION INSTRUCTIONS

1. Naming convention for your programs: `cs18xx-test3-progy.c`
2. When you have finished, copy all your files to `~dfslab/2018/labtest3/cs18xx/`.

1. **(15 marks)** Let us define a sequence $a_0, a_1, a_2, \ldots, a_{n-1}$ as Λ-*bitonic* if there exists a j, $0 \leq j < n$, such that $a_0 < a_1 < \cdots < a_j > a_{j+1} > \cdots > a_{n-1}$. Consider an $m \times n$ matrix A consisting of integer entries, such that each row and each column of the matrix forms a Λ-bitonic sequence. Write a program to efficiently find the largest element of the matrix.

Input Format

Input will be provided via standard input in the following format. The first line of input consists of a pair of integers, namely the number of rows (m) and number of columns (n) of the matrix A. It follows by the elements of A, providing each row in a single line following their order in the matrix.

Output Format

Output is to be printed on the standard output in the following format. The output will print the largest element of the input matrix A.

Sample Input 0

```
1 8
-10 -5 0 5 10 15 20 10
```

Sample Output 0

```
20
```

Sample Input 1

```
2 2
40 20
30 10
```

Sample Output 1

```
40
```

Sample Input 2

```
3 6
10 19 30 28 26 25
15 25 35 50 41 22
10 20 22 38 40 20
```

Sample Output 2

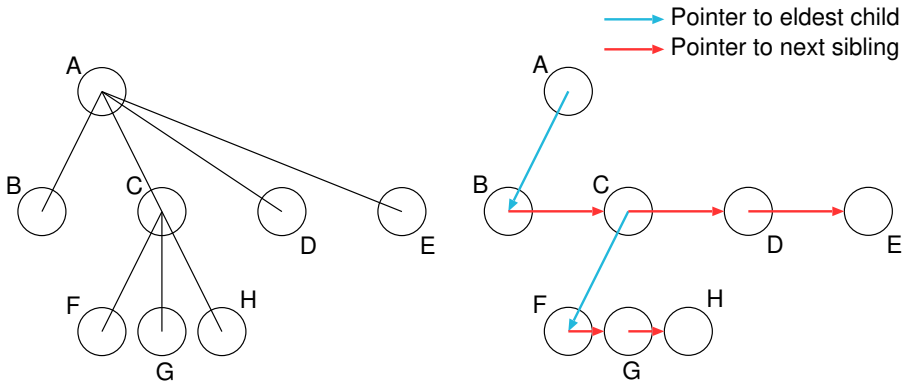
```
50
```

2. **(25 marks)** This question is based on the "alternative" implementation of trees discussed in class, i.e., all nodes of a tree are stored in an array; instead of using a pointer to store the memory address of a node, we use an integer to store its index/location in the array.

   We use the following type definition to represent a node in a tree of arbitrary arity.

```
typedef struct {
    DATA d;
    int eldest_child, next_sibling; // like TREENODE *eldest_child, *next_sibling
} TREENODE;
```

   That is, each node points to its eldest child, and to its right sibling (if these exist). The conventional representation of a tree and its representation using the node definition given above is shown in the figure below. NULL pointers have not been shown in the picture.



   Augment the structure type definition above to incorporate, for each node, the index of its parent, and a non-negative integer specifying the level of the node (the *level* of the root is defined to be 0; if m is a child node of n, then the level of m is one more than the level of n).

   (a) Read in a tree of arbitrary arity from an input file, augment the tree with the information specified above, and write the augmented tree out in a similar format to an output file. Input and output file names will be provided as command-line arguments. Note that all lines in the input and output files (except the first line) will contain 3 columns and 5 columns, respectively.

(b) Given any two nodes in the tree (specified by their line numbers), find the line number of their lowest common ancestor.

Input Format

Input will be provided in a separate file (name to be taken from command-line arguments) in the following format.

The first line of input will contain n, where n is the number of nodes in the binary tree. This will be followed by n more lines. Each of these remaining lines will correspond to one node (considering the first line representing the root node) in the tree and will consist of 3 integers: the data (an integer to be stored in the node), the line number of the node corresponding to the left child (-1 if there is no left child), and the line number corresponding to the right child (-1 if there is no right child).

Output Format

Output is to be printed in a separate file (name to be taken from command-line arguments) in the following format.

The first line of output will contain n, where n is the number of nodes in the binary tree. This will be followed by n more lines. Each of these remaining lines will correspond to one node (considering the first line representing the root node) in the tree and will consist of 5 integers: the data (an integer to be stored in the node), the line number of the node corresponding to the eldest child (-1 if there is no eldest child), the line number corresponding to the next sibling (-1 if there is no next sibling), the line number of the node corresponding to its parent (an integer), and a non-negative integer specifying the level of the node.

Command-line Arguments

Assuming the line numbers of the two nodes, whose common ancestors are to be returned, as $n1$ and $n2$:

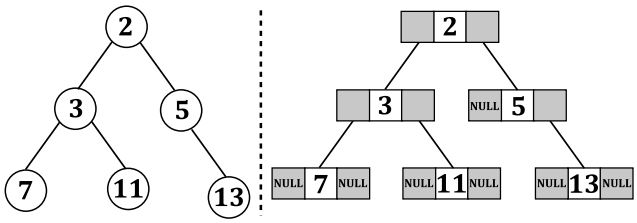
```
./prog2 <input_filename> <output_filename> n1 n2
```

Sample Input 0

```
Contents of input.txt:

6
2 1 2
3 3 4
5 -1 5
7 -1 -1
11 -1 -1
13 -1 -1

Input from command-line:
./prog2 input.txt output.txt 3 5
```



Sample Output 0

```
Contents of output.txt:

6
2 1 -1 -1 0
3 3 2 0 1
5 5 -1 0 1
7 -1 4 1 2
11 -1 -1 1 2
13 -1 -1 2 2

Output on the terminal:
0
```

3. **(20 marks)** A *light curve* is recorded as a sequence of floating point numbers, $a_0, a_1, a_2, \ldots, a_n$, where each a_i represents the brightness of a particular star, as observed by NASA's Kepler Space Telescope at time $t = t_i$ (these measurements are taken approximately in every 30 minutes). The observed brightness (a_i) is represented as a normalized value in $[0, 1]$, with 0 representing darkness, and 1 representing the maximum brightness. The observed brightness of a star may vary for different reasons. For example, when a planet passes in front of a star (this is called a *planetary transit*), it obstructs some of the star's light, thereby making the star appear dimmer for a certain period.

   You have to write an efficient program to automatically detect possible planetary transits using the following approach. Given the following

   - a light curve, i.e., a sequence of floating point numbers $a_0, a_1, a_2, \ldots a_n$ (where $a_i \in [0, 1]$ for $i \in \{1, 2, \ldots, n\}$),
   - m, the total number of points in the light curve that appear dim, and
   - a parameter k,

   find whether there is a sequence of at least k *successive* dim points in the light curve recorded.

   If such a sequence is found, your program should signal a possible planetary transit. Otherwise, it should report that no planetary transits were observed. For full credit, your program must take less than $O(n \log n)$ time, where n is the number of observations in the given light curve.

   Input Format

   Input will be provided via standard input in the following format. The first line of input is a pair of integers, namely the total number of dimmed points m and the window size k. It follows by a number of floating

point values representing the brightness of points on the light curve. The input ends with a '-1' in a new line.

Output Format

Output is to be printed on the standard output in the following format. The output will print TRUE or FALSE to denote whether the presence of a planetary transit is admitted or not, respectively.

Sample Input 0

```
2 3
1.000
0.999
0.700
0.987
0.780
0.967
-1
```

Sample Output 0

```
FALSE
```

Sample Input 1

```
3 3
1.000
0.998
0.700
0.787
0.780
0.967
0.999
-1
```

Sample Output 1

```
TRUE
```

Sample Input 2

```
5 4
1.000
0.999
0.900
0.987
0.780
0.667
0.880
0.867
0.675
0.989
-1
```

Sample Output 2

```
TRUE
```