INDIAN STATISTICAL INSTITUTE

MTech(CS) I year 2022-2023

Subject: Computing Laboratory

Lab Test 2 (October 21, 2022)

Total: 60 marks Duration: 3 hours

SUBMISSION INSTRUCTIONS

1. Please make sure that your file names are of the form `mtc22xx-progy.c`, where `xx` is your 2-digit roll number and `y` is the question number (`1a`, `1b` or `2`).
2. To submit, please upload your files to `https://www.dropbox.com/request/4coNld9ZfJb7NmRNB5o9`.
3. Please make sure that your programs adhere strictly to the specified input and output format. Do not print extra strings asking the user for input, debugging messages, etc. These will cause the automatic checking system to fail.

Q1. (20 marks) You have to write a program to re-arrange the nodes in a given linked list so that all odd valued nodes come before all even valued nodes. The relative ordering of even-valued nodes (resp., odd-valued nodes) should not change.

(a) (10 marks) Write your program using the traditional pointer-based implementation of linked lists.

(b) (10 marks) Re-write your program using the "alternative" implementation discussed in class.

Input format: The elements of the linked list will be given to you in order as command-line arguments.

Output format: Your program should print the elements of the re-arranged list in order to standard output.

Sample input 1:

```
$ ./a.out 1
```

Sample output 1:

```
1
```

Sample input 2:

```
$ ./a.out 2 8 6 0
```

Sample output 2:

```
2 8 6 0
```

Sample input 3:

```
$ ./a.out 4 9 6 3 2 2 8 19
```

Sample output 3:

```
9 3 19 4 6 2 2 8
```

Since this is an entirely straightforward problem, no credit will be given for simply generating the desired output if your program does not use the specified implementation.

Q2. (40 marks) A k-ary *Boolean function* is a function of the form $f : \{0,1\}^k \rightarrow \{0,1\}$, i.e., it takes k bits (0 or 1) as inputs, and produces an output that can be either 0 or 1 for a given input. For this problem, we will assume $k > 0$.

Boolean functions are often described using truth-tables. The truth table for a k-ary Boolean function has 2^k rows, each corresponding to a possible combination of the inputs. The example below shows the truth-tables corresponding to the well-known AND and OR Boolean functions for 2 variables.

x_1	x_2	AND(x_1, x_2)
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	OR(x_1, x_2)
0	0	0
0	1	1
1	0	1
1	1	1

In this question, we will first consider representing a Boolean function using a binary tree.

$$f(x,y,z) = x \cdot y + \bar{y} \cdot z$$

Truth Table

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

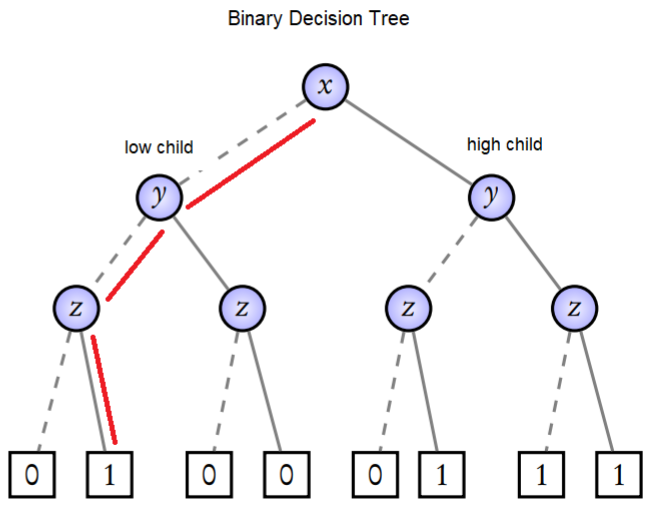

Figure 1: A sample Binary Tree Representation

(a) *Constructing a binary tree corresponding to a Boolean function* (20 marks).

Figure 1 shows how the Boolean function $f(x, y, z) = x.y + \bar{y}.z$ can be represented using a binary tree. In general, given any k-ary Boolean function, the corresponding binary tree can be constructed using the following rules.

- The tree will have $k + 1$ levels, numbered $1, 2, \ldots, k + 1$. These levels correspond respectively to the input variables x_1, x_2, ..., x_k (in that order), and the output.
- All the nodes at level i will be labelled with the value x_i. The leaf nodes (at level $k + 1$) will be labelled with either **0** or **1**.
- Each internal (non-leaf) node in the tree will have **exactly** 2 children.

- Note that the tree will have exactly 2^k leaf nodes. Each path from the root to a leaf represents one of the possible 2^k inputs to f, as explained below.
- Suppose P is a path from the root (x_1) to a leaf. If P goes from a node at level i to its **left** (or low) child at level $i+1$, then the input x_i is taken to have the value **0**. Conversely, if P goes from a node at level i to its **right** (or high) child, the input x_i is taken to have the value **1**.
- Finally, the value at the leaf node reached via path P denotes the output of f for the input corresponding to P.

In Figure 1, for example, the path highlighted by the red color corresponds to the 2nd row of the truth table for f (i.e., to the minterm $\bar{x}.\bar{y}.z$).

Write a program that reads the truth-table for a Boolean function f from a file, and constructs the corresponding binary tree, as specified above. Use the following `typedef`s to represent a node and the tree.

```
typedef struct {
    int data;
    int left, right, level;
} TNODE;
```

```
typedef struct {
    int root, num_nodes;
    TNODE *nodelist;
} TREE;
```

The names of the fields are self-explanatory. For the internal nodes, the values of `data` and `level` should be the same. For a leaf node, `data` should store the corresponding output value of f (0 or 1).

Input format: The truth table will be stored in a file named `input.txt`. The first line of this file will contain the value of k. Each of the remaining 2^k lines of `input.txt` will contain $k+1$ 0s and 1s, separated by spaces. The first k digits will represent one of the possible inputs for f; the last digit will be the value of the corresponding output.

Output format: Your program should construct the corresponding tree in such a way that the root node corresponds to the first element (index 0) of `nodelist`. It should then serially print the elements of the `nodelist` array for this tree in a file named `output2a.txt`. Each element should be printed on a separate line, and the 4 integers stored in the element should be separated by a single blank. You may use the following function for this part.

```
void print_tree(TREE *t, char *filename)
{
    FILE *fp;
    TNODE *node;

    if (NULL == (fp = fopen(filename, "w")))
        ERR_MESG("print_tree: error opening output file");
    for (node = t->nodelist; node - t->nodelist < t->num_nodes; node++)
        fprintf(fp, "%d %d %d %d\n", node->data, node->left, node->right,
                node->level);
    fclose(fp);
    return;
}
```

Please download testcases2.zip from the course page for additional examples / testcases, and their worked out solutions.

(b) *Removal of duplicate leaves.* (20 marks).

We will now look at (simplified) *Binary Decision Diagrams* (BDD), a more compact way of representing

Boolean functions. If your program for part (a) is correct, you should see that each of the 2^k lines corresponding to a leaf node is either `0 -1 -1 k+1` or `1 -1 -1 k+1`. We want to obtain a more compact representation of the Boolean function by eliminating all these duplicate lines, and keeping only **two** leaves, corresponding to the values **0** and **1**, respectively.

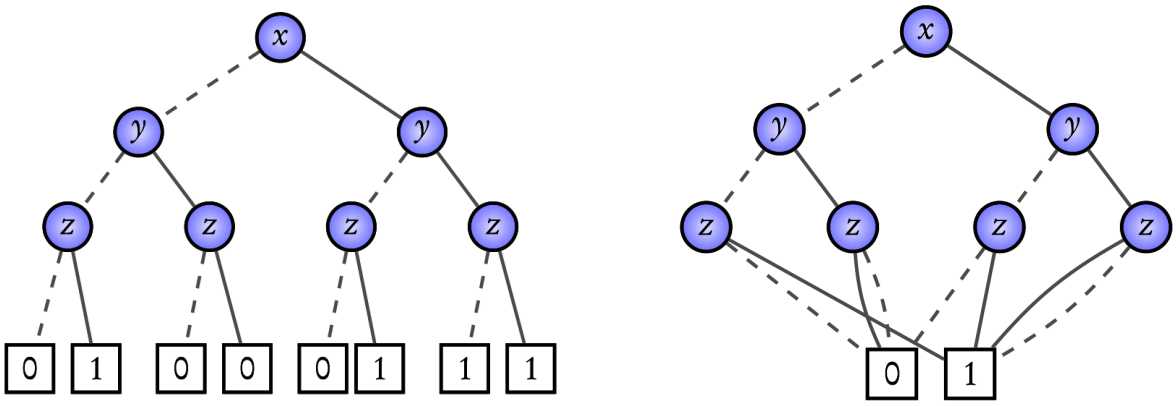

Figure 2: Obtaining the Reduced BDD

Figure 2 shows both the original binary tree corresponding to the function f from Figure 1, as well as the compact representation described above. The compact representation is not a binary tree, but it can still be represented using the same structures as in part (a).

Write a function `remove_duplicate_leaves()` to obtain the compact representation described above from a given binary tree representating a Boolean function. Your function should take the given binary tree as an input parameter (along with any other parameters that you want to pass); it should return a structure of type TREE (say `t_out`) that contains the compact representation, and has the following properties.

(i) `t_out.root` $= 2$;

(ii) `t_out.num_nodes` $= 2^k + 1$;

(iii) `t_out.nodelist[0]` and `t_out.nodelist[1]` correspond to **all** leaves with value **0** and **1**, respectively.

(iv) the `left` and `right` fields of any node at level k should now be either `0` or `1`, depending on whether the left and right children were leaves with value **0** or **1**.

Thus, `t_out` should look like the following.

Index	data	left	right	level
0	0	-1	-1	k+1
1	1	-1	-1	k+1
root → 2	1	3	...	1
3	2	...	...	2
...	...	...	...	...

Your program should save the compact representation in a file named `output2b.txt`, using the `print_tree()` function provided in part (a). Thus, the first two lines of `output2b.txt` should correspond to the two leaves, and the root node should be on the 3rd line of the file.