Department of Computer Engineering, METU

CENG 140

Fall 2019-2020

THE 1

Yusuf Mücahit Çetinkaya
yusufc@ceng.metu.edu.tr

Due date: 15.11.2019 Friday, 23:55

1 Overview

Recursion is a concept in computer science that a function calls itself by breaking the input into pieces directly or indirectly to solve problems. These functions are called recursive functions. The recursion method perfectly fits for some problems such as Towers of Hanoi, tree traversals, graph searches, etc. However, sometimes iterative solutions are simpler and more appropriate.

In this assignment, you are expected to implement some functions with iterative and recursive approaches. Signatures of the functions are given in header files(*utils.h, draw.h*). Additionally, stub codes are available for the source files. Util functions are already implemented and you will use them directly without a change.

2 Tasks

2.1 Draw Matrix(40 Pts.)

A lower triangular matrix is a special square matrix that all entries above the main diagonal are zero. For this task, you will print a regular lower triangular matrix but in mirrored on the y-axis. Opposite diagonal entries are always 1 and increments by one while moving on the x-direction.

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 1 & 2 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & 1 & \dots & \text{n-3} & \text{n-2} \\ 0 & 1 & 2 & \dots & \text{n-2} & \text{n-1} \\ 1 & 2 & 3 & \dots & \text{n-1} & \text{n} \end{bmatrix}$$

An alternant matrix is a matrix formed by applying a finite list of functions pointwise to a fixed column of inputs. For this assignment, there are 5 functions that are located in *utils.c*. If the number of columns is more than 5, you will take the modulo of the column number.

$$\begin{bmatrix} \text{f}_0(0) & \text{f}_1(0) & \text{f}_2(0) & \dots & \text{f}_{(n-1)\%5}(0) & \text{f}_{n\%5}(1) \\ \text{f}_0(0) & \text{f}_1(0) & \text{f}_2(0) & \dots & \text{f}_{(n-1)\%5}(1) & \text{f}_{n\%5}(2) \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ \text{f}_0(0) & \text{f}_1(0) & \text{f}_2(1) & \dots & \text{f}_{(n-1)\%5}(n-3) & \text{f}_{n\%5}(n-2) \\ \text{f}_0(0) & \text{f}_1(1) & \text{f}_2(2) & \dots & \text{f}_{(n-1)\%5}(n-2) & \text{f}_{n\%5}(n-1) \\ \text{f}_0(1) & \text{f}_1(2) & \text{f}_2(3) & \dots & \text{f}_{(n-1)\%5}(n-1) & \text{f}_{n\%5}(n) \end{bmatrix}$$

- You will implement two methods for drawing matrix. Namely, **void draw_matrix_rec(int row, int col, int size, int is_alternant)** and **void draw_matrix_itr(int size, int is_alternant)**.
- **void draw_matrix(int size, int is_alternant, int is_recursive)** function is already implemented and given to you.
- The size of the matrix is passed to the function with **size** parameter.

- If **is_alternant** value is evaluated as true, you will use **apply_function** method given to you in utils. Otherwise, you will use the value of the entry directly as visualized above.
- If **is_recursive** parameter is true, the method calls recursive **void draw_matrix_rec** function. Otherwise, it calls iterative **void draw_matrix_itr** function.
- Entries are seperated with a single space character(␣). At the end of the rows, there is no space character but a new line character(**\n**). At the last row, you will not print any new line character.
- You will manipulate **row** and **col** variables as control variables to conduct the recursion in **void draw_matrix_rec** function.
- **void draw_matrix_rec** must not include any loop or global, static variable.
- **void draw_matrix_rec** should call itself. In other words, it uses direct recursion technique. For more information click!
- Implementing **void draw_matrix_rec** function without recursive fashion will not give you any points.
- You will implement the same logic in **void draw_matrix_itr** function with iterative manner. The outputs of recursive and iterative functions with same parameters should be identical. You should not use any global, static variable.
- Use **unsigned long** data type for getting rid of overflow problems.

2.2 Binary Tree(60 Pts.)

Binary tree is a data structure where each node in the tree has up to two child nodes. It is a recursive data structure by its nature. More clearly, each node in the tree can be accepted as a root of subtree. In this assignment, the values in the nodes of the tree are integer and split with a ratio, which is in the range of(0,1). The value of the left child node equals to integer value of $ratio \times root$ and the value of the right child node is $(1 - ratio) \times root$. Figure 1 visualizes a binary tree that is split with ratio of 0.3. You will calculate the depth and draw a binary tree to the console without using any data structure.

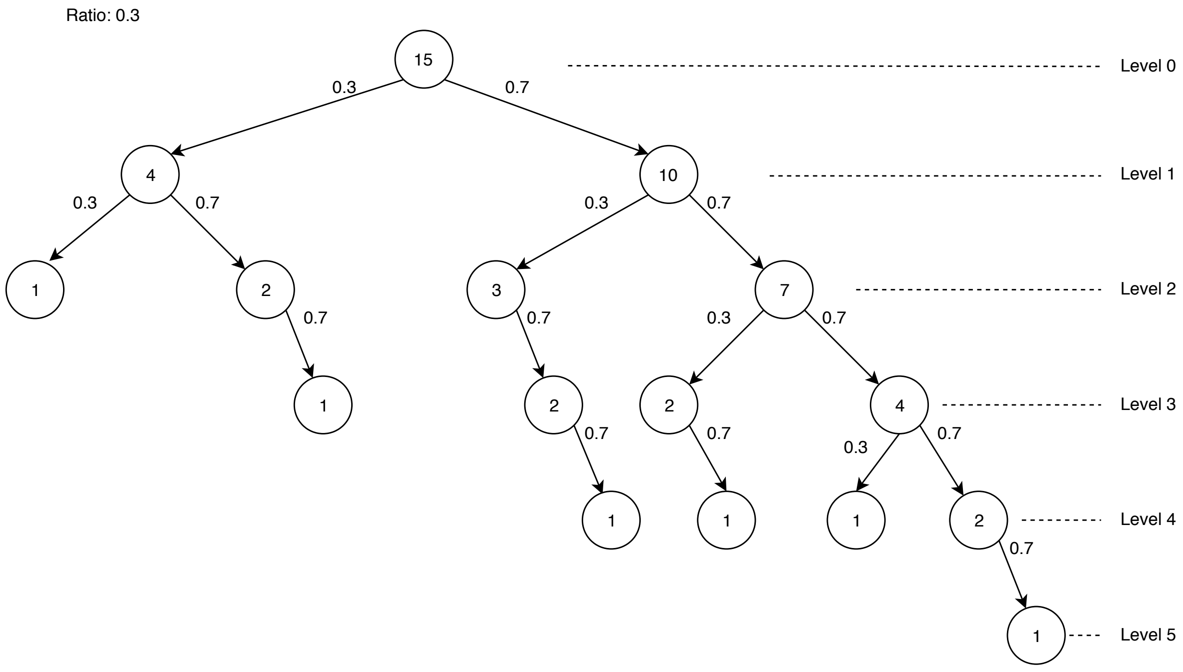

Figure 1: Binary tree that is split with 0.3 ratio. Node values are integer. If the value of the node is lower than 1, it is skipped. The depth of the tree is 6.

- You will implement two methods related to binary tree. Namely, **int find_depth(int root, float ratio)** and **void draw_binary_tree_rec(int root, int max, int level, int depth, float ratio)**.

- **void draw_binary_tree(int root, int depth, float ratio)** function is already implemented and given to you.
- The value of the root node is passed to the functions with **root** parameter.
- The splitting ratio is passed to the functions with **ratio** parameter.
- You will manipulate **root** and **level** variables as control variable to conduct the recursion in **int find_depth** function.
- **int find_depth** function will return the depth of the tree. More explicitly, the level of the deepest node in the tree plus 1.
- **void draw_binary_tree** function will print the tree in row-based with in order traversal (left-child / root / right-child).
- You will manipulate **root** and **level** variables as control variable to conduct the recursion in **void draw_binary_tree** function.
- Tab character (**\t**) will be used to visualize the level. Number of tab character is $depth-(level+1)$ for each node.
- There is one new line character(**\n**) at the end of output for both functions.
- Using any high-level data structure (pointer, array, etc.) is **forbidden**.
- You will use direct recursion for **int find_depth** and **void draw_binary_tree** function.Implementing these functions without recursive fashion will not give you any points. They must not include any loop or global, static variable.

3 Regulations

- **Input-Output format:** Input-output formats are given below. You will read inputs within main function in **main.c** file. There will be 3 different operations which are represented with 3 different characters.
  - 'm': draw_**m**atrix
  - 'f': **f**ind_depth
  - 'd': **d**raw_binary_tree

  Input Format:

```
<opt_char>
if opt_char == 'm':
    <size> <is_alternant> <is_recursive>
if opt_char == 'f':
    <root> <ratio>
if opt_char == 'd':
    <root> <depth> <ratio>
```

  Example 1:
  Draw a simple matrix with the size of 6 with iterative function.
  Input:

```
m
6 0 0
```

  Output:

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

Example 2:
Draw an alternant matrix with the size of 6 with recursive function.

Input:

```
m
6 1 1
```

Output:

```
0 2 4 8 16 1
0 2 4 8 17 2
0 2 4 9 48 3
0 2 5 24 259 4
0 3 12 89 1040 5
1 6 31 264 3141 6
```

Example 3:
Find the depth of the tree where root is 81 and split ratio is 0.41.

Input:

```
f
81 0.41
```

Output:

```
Depth of the tree[size: 81, ratio:0.41] is 8.
```

Example 4:
Draw binary tree where root is 15, depth is 6 and split ratio is 0.3.

Input:

```
d
15 6 0.3
```

Output:

```
                        1
                                4
                        2
                1
                                        15
                        3
                2
        1
                                10
                2
        1
                        7
        1
                4
        2
1
```

- **Programming Language:** C
- **Libraries and Language Elements:**
  You should not use any library other than *"stdio.h"* and *"math.h"*. You can use conditional clauses (switch/if/else if/else), loops (for/while). **You can NOT use any further elements beyond that (this is for students who repeat the course).** You can define your own helper functions, provided that you have implemented 4 functions given to you in **Tasks** section.

- **Compiling and running:**
  You should be able to compile your codes and run your program with given **Makefile**:

  ```
  >_ make the1
  >_ ./the1
  ```

  If you are working with ineks or you are working on Ubuntu OS, you can feed your program with input files instead of typing inputs. This gives the input from stdin and an equivalent of typing inputs. This way you can test your inputs faster:

  ```
  >_ ./the1 < inp1.txt
  >_ ./the1 < inp2.txt
  ```

- **Submission:**
  You will use CengClass system for the homework just like Lab Exams. You can use the system as editor or work locally and upload the source files. Late submission IS NOT allowed, it is not possible to extend the deadline and **please do not ask for any deadline extensions**.

- **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as example. You can check your grade with sample test cases via CengClass system but do not forget it is not your final grade. Your output must give the exact output of the expected outputs. It is your responsibility to check the correctness of the output with the invisible characters. Otherwise, you can not get grade from that case. If your program gives correct outputs for all cases, you will get 100 points.

- **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. Sharing code between each other or using third party code is strictly forbidden. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.