ExamTex - Team Number 27

1. Data Structures Used:

   a. Arrays: Arrays have been used throughout the program for general execution. However, the most important interpretation is of making arrays to store pointers to our questions. Every type of question has a different struct. Now every input question of the bank gets allocated memory and has a unique pointer to its struct in memory. These pointers get stored in arrays depending on their struct types.

   b. Stacks: Stacks have been used while inputting of data. The starting of brackets are stored in the stack. Later on, when a closing bracket is encountered, the opening bracket is popped from the stack.

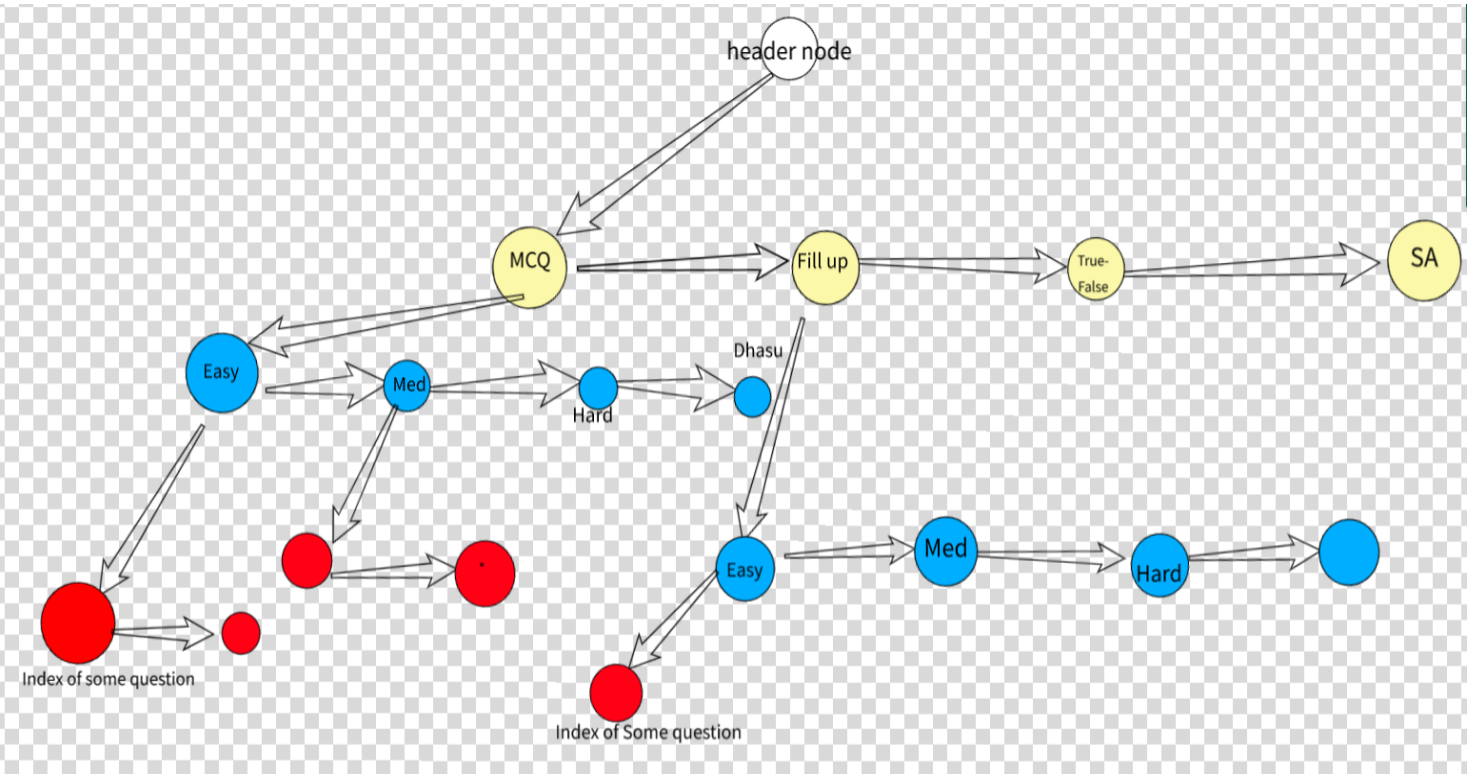
   c. Tree: A general tree has been used for making each question bank. Our tree has been implemented in a manner minimizing the searching required while forming the question paper. Since the tree is not binary, each node of the tree contains a pointer to the next sibling and a pointer to its first child.

      The header file points to the first type of question (Multiple Choice Questions) indicated by a zero. The next sibling of MCQ is Fill in the blanks(indicated by 1), followed by True-False(indicated by 2) and Short Answer(indicated by 3). Each node pertaining to the level of difficulty has nodes of difficulties Easy, Medium, Hard and Dhasu(0, 1, 2 and 3 respectively).

      Now, as explained before, the pointers for each input question of the bank gets stored in dynamic arrays. The indexes of these memory locations in arrays get stored in the nodes pointed by the difficulty levels. For example, consider a question of easy difficulty and mcq type.

      Let its index in an array related to mcq struct be zero. Then, MCQ->first_child will be Easy and the first child of Easy would be 0 (The index in array) . MCQ->next_sibling would be Fill in the blanks which will have its own difficulty level nodes.

      The below diagram represents a visualization of the graph we have implemented. The White node indicates the header node. The Yellow nodes indicate the nodes for type of questions. The blue nodes indicate the nodes for difficulty of questions. The red nodes indicate the indexes of the questions.



2. Algorithms used and their Time Complexity:

   a. Tree Traversal: Since each node is being visited once, the time complexity is O(n) considering that n is the number of nodes.

   b. Insertion of node in Tree: This requires maximum traversal of n nodes if ‘n’ is the number of nodes in the current graph. Since the input tells us the difficulty of the question and the type of question, we travers to the nodes under the region of given difficulty pointed to by the given question type node. Hence, the worst case time complexity is O(n). The best case time complexity of O(1) which would definitely be the case when we are inserting a node for the first time in the tree.

   c. Searching for nodes in tree: The time complexity for this scenario is also O(n) in the worst case where the last node to be searched is the desired node. The best case is O(1) where the first node to be searched is the desired node or there are no available nodes.

   d. Deletion of node in tree: It is nothing but searching for the node in the tree and removing the node if found. All the nodes containing indexes of the desired questions are nothing but the leaf nodes. As a result, removing a question, is basically removing a leaf node which does not require any readjustment of the tree. It only requires freeing of allocated memory to the node. As a result, it shares similar time complexity as deletion of nodes.

3. Overall Explanation of Our program:

    a. Addition of Question Bank: Each inputted question is allocated memory and stored in some array related to its type of question, as explained earlier. A tree is created containing nodes corresponding to question and difficulty types. Now, the indexes of these questions are inserted in this tree at the last level (leaf nodes) as explained earlier. All leaves are traversed till no element is found. Memory is allocated for the node and the element is inserted there.

    b. Adding Questions to Question Bank: This is a similar implementation where instead of creating a new tree, we are adding questions to an already existing tree.

    c. Print Existing Question Bank: The tree is wholly traversed and all the questions are printed according to their type. All the questions are printed on the terminal for quicker review by the user.

    d. Deleting Question bank: The Question bank's memory is freed in the program.

    e. Generating Question Paper:
        i. The first task is to read the question paper input file and obtain data of what exactly is required in the question paper. (The number of questions of what type and what difficulty).

        ii. According to the given data, we generate fully randomised indexes according to the constraints. Then we access the pointers to the questions from the arrays (as we know the indexes).

        iii. For MCQ type, the user also has an option of putting more than 3 incorrect options and more than 1 correct option. The program randomly selects 3 incorrect options and 1 correct option and forms the MCQ to be printed in the question paper.

        iv. The program checks for the available questions in the question bank of a particular difficulty and a particular type. If the number of required questions is more than the number of available questions, the question paper is not generated.

        v. Number of generated Files: A feature to generate more than one files with the same question paper input file provided. Let us assume you need to prepare question papers for 100 students from a common question bank. However, you do not want every student to have the same question paper. Our program will generate 100 randomized question papers according to the given constraints to satisfy the requirement.

vi. All the output Question papers have been converted to markdown so that they can be easily converted to pdf format using software like “Pandoc’, etc.

4. Contributions:
   a. Implementation of Question Bank Tree and functions associated with it: Tanmay Goyal
   b. Input of Question bank file and Allocating memory and structs for questions: Laksh balani
   c. Input of Question paper file and creating sample files: Siddh Jain
   d. Generating question papers and implementation of the functions associated with it: Pratham Gupta

5. Marks File:

   a. It should take input of total number of students who have given a particular Exam having questions from the existing question bank.

   b. Now for every question we will also have input of the number of students who could answer the question. We will already have initial difficulty level of each question from the question bank input file.

   c. Consider the sample input file:
   \students{100}
   \question{position = 1, number = 50}
   \question{position = 2, number = 30}

   d. Here, \students will indicate the number of students who have taken the exam previously. Now, ” position” in \question will indicate the position of the question in question bank input file. The “number” attribute would indicate the number of students who have answered the question correctly.

   e. Implementation of changing difficulty: To improve the level of difficulty of every question, we can try to obtain the average number of people who have answered question of that difficulty correct. Let us try to understand this with an example. Let us assume that total number of students in class is = 100. Let us take the number of questions of difficulty easy to be 3 and each question answered by 40, 50 and 60 people respectively. Also similarly for medium difficulty: 25, 50 and 0. Considering that only 3 difficulties exist, for hard we can take the numbers to be = 10, 18 and 2. So, the average number of people who have answered the question correctly:
   -> For easy: 50
   -> For Medium: 25
   -> For hard: 10

Now, we can say that all the questions answered by > 50 students are definitely easy and that answered by > 10 students are definitely hard. So these questions would not have their difficulty level updated. Those questions with difficulty between 50 and 37.5 (25 + (50-25)/2) Can be said to be easy, and those questions between 27.5 and 25 should be considered medium. Similarly, those questions answered by number of students between 10 and 17.5 should be considered to be Hard and those between 17.5 and 25 should be Medium. As a result, the question of initial difficulty medium and successfully answered by 50 people will be converted to easy and that by 0 people will be converted to hard. Also, the question of initial difficulty hard and successfully answered by 18 people will be converted to medium.

f. Another way of implementation is to obtain the average number of marks obtained for a question and the total number of marks of the question. Instead of considering the number of people who answered the question, we would consider the average number of marks in that question.