A REPORT ON

Automatic Problem Generator

BY :
Nikhil Dilip Agrawal | 2016A4PS0248P

COURSE : Study Oriented Project
INSTRUCTOR : Dr. Kamlesh Tiwari

1.Problem Statement

To Develop an algorithm which generates a similar problem as provided by the user using constraints provided.

2. Background of the problem

a. Description of the selected application domain

Generating problems that are similar to a given problem has many applications.It can help avoid copyright issues. It may not be legal to publish problems from textbooks on course websites. Hence, instructors resort to providing indirect pointers to textbook problems as part of assignments. So, instead of taking problems from exact same textbook, they can provide question and similar problem can be generated using the system.

b. Motivation of the problem

A problem generation tool can provide instructors with a fresh source of problems to be used in their assignments or lecture notes. Also, it can help prevent plagiarism in classrooms or massive open online courses since each student can be provided with a different problem of the same difficulty level.

c. Technical issues included

Technical issues include the vast variety of question that exist and number of different techniques with which the questions can be solved. Along with the variety of question, the constraints placed on the variables differ from domain to domain. Hence, forming a generalized automatic similar problem generation is a very difficult task with using NLP techniques.

3. Related Work: Literature survey

3.1. Automatic problem generation for Natural deduction

Automatic problem is generated by creating the data structure called universal proof graph. Universal proof graph is hypergraph with nodes representing propositions and edges representing the inference rules.

3.2. Automatic problem generation from the paragraph

Different algorithm and NLP techniques have been proposed to date using which automatic questions can be generated from a given paragraph.

4. System Description

The proposed algorithm is divided into total seven steps:

I. Question input from the user:

Input Format :

1. In first step, user will provide a question in which the variable will be preceded by "#" symbol.

2. In second step, constraints will be provided by the user in the format given below.

a) First of all, type of each variable is declared. Declaration of values can take place in any order. Below is the form of declaration for each type of variable.

i. If the variable is integer then the form of declaration is "*int(Var,l,r)*" where "Var" is variable name and "l" is lower bound of range and "r" is upper bound of range.
eg. int(X,1,22) means X is integer whose value lie between 1 and 22.

ii. If the variable is float then the form of declaration will be "*float(Var,p,l,r)*" where "Var" is variable name, "p" is precision, "l" is lower bound of range and "r" is upper bound of range.

eg. float(X,1,1,22) implies X will be in the range of 1 and 22 and decimal precision is 1.

iii. If the variable is string then the form of declaration is *"literal(Var,string1,string2,..)"* where "Var" is variable and "string1","string2".. are the possible string values which Var can take. Number of arguments in literal function can be large too.

eg. literal(X, Apple, Banana, Carrot) implies X is a string variable which can take value of "Apple","Banana" and "Carrot".

b) Then, other constraints on variables are provided depending on the constraints which user want to put on the question. The functions or interdependent relation added in the code are :

i. less(A,B):

It evaluates to True if the expression A is less than B else False.

e.g. less(5+X,Y) => 5+X < Y

less(19, 3*X + 4*Y) => 19<3*X + 4*Y

ii. equal(A,B)

It evaluates to True if the expression A is equal to B else False.

e.g. equal(X+Y,19) => X+Y = 19, it implies X and Y can only take those values whose sum add up to 19.

equal(X-3,2) => X-3 = 2

iii. notequal(A,B)

It works opposite to equal.

iv. divisible(A,B)

It evaluates to True if A is divisible by B.

e.g diviable(X,Y) => X = m*Y for some integer value of m.

divisible(6,3) => evaluates to true since 6 is divisible by 3.

v. mod(A,B,M)

It evaluates to true if A%B is M

e.g. mod(X,Y,10) implies X when divided by Y leaves remainder 10.

vi. coprime(A,B)

It evaluates to True if A and B are coprime.

eg. coprime(10,8) return false
coprime(X,11) => X can only take those value which are coprime to 11

vii. prime(A)

It evaluates to true if A is prime. It also implies that A can take only those values which are prime.

eg. prime(X) => Implies that X is prime number.

NOTE : Variable Declaration should be in uppercase and constraints should be in lower case(except the Variables).

II. Extracting the variables and the type of each variable:

In this step all the variable from the question are extracted and are divided into respective category of integer, float and string.

```
def extract_variable_name(question, variable_names):
    #----Here Name of all variable has been extracted----#
    for i in range(0,len(question)):
        if(question[i] != '#') :
            continue
        i  = i + 1;
        temp = ""
        while(i<len(question) and question[i] is not ' ' and question[i] is not '\t'):
            temp += question[i]
            i = i + 1;
        if(temp not in variable_names):
            variable_names.append(temp)
            |
    for i in range(0,len(variable_names)):
        if('\t' in variable_names[i]):
            print("entered")
            variable_names[i] = variable_names[i].replace('\t','')
```

```
def get_category_of_each_variable(variable_names, all_constraints):
    all_int_var = []
    all_object_var = []
    toDelete = []
    toAdd = []
    for i in range(0, len(all_constraints)):
        constraint = all_constraints[i]
        if("int" in constraint):
            constraint = constraint.replace('(',' ')
            constraint = constraint.replace(')',' ')
            constraint = constraint.replace(',',' ')
            constraint = constraint.split()
            toAdd.append('range(' + constraint[1]+','+constraint[2]+','+constraint[3]+')')
            toDelete.append(all_constraints[i])
            all_int_var.append(constraint[1])
        if("float" in constraint):
            constraint = constraint.replace('(',' ')
            constraint = constraint.replace(')',' ')
            constraint = constraint.replace(',',' ')
            constraint = constraint.split()
            toAdd.append('precision(' + constraint[1] + ',' + constraint[2]+')')
            toAdd.append('range(' + constraint[1] + ',' + constraint[3] + ',' + constraint[4] + ')')
            toDelete.append(all_constraints[i])
            all_int_var.append(constraint[1])
    for i in range(0, len(toDelete)):
        all_constraints.remove(toDelete[i])
    for i in range(0, len(toAdd)):
        all_constraints.append(toAdd[i])
    for i in range(0,len(variable_names)):
        if(variable_names[i] not in all_int_var):
            all_object_var.append(variable_names[i])
    return all_int_var, all_object_var
```

III. Extracting pure dependent function and variable interdependent functions :

Pure dependent functions are defined as those functions which depend on only one of the variables while interdependent functions depend on more than one variables.

While extracting the pure dependent function, the functions are mapped to corresponding variable.

In interdependent variable functions, graph is created in which edges are interdependent variable function while nodes are the variables.

```
#----Pure Variable Function are extraced---#
pure_variable_function = dict()
for i in range(0,len(variable_names)):
    pure_variable_function[variable_names[i]] = []
    for j in range(0,len(all_constraints)):
        if("int" in all_constraints[j] or "object" in all_constraints[j]):
            continue;
        if(variable_names[i] in all_constraints[j]):
            flag = True
            for k in range(0,len(variable_names)):
                if(i is k):
                    continue
                if(variable_names[k] in all_constraints[j]):
                    flag = False
            if(flag):
                pure_variable_function[variable_names[i]].append(all_constraints[j])
print(pure_variable_function)
```

```
#----Extracting Depending Variable Function---#
dependent_variable_function = set()
for i in range(0,len(variable_names)):
    for j in range(0,len(all_constraints)):
        if("int" in all_constraints[j] or "object" in all_constraints[j]):
            continue;
        if(variable_names[i] not in all_constraints[j]):
            continue
        if(variable_names[i] in all_constraints[j]):
            flag = False
            for k in range(0,len(variable_names)):
                if(i is k):
                    continue;
                if(variable_names[k] in all_constraints[j]):
                    flag = True;
            if(flag):
                dependent_variable_function.add(all_constraints[j])
#print(dependent_variable_function)
#----Done Extracting Dependent variable function----#
```

IV. Generating randomized values for variables:

After the graph is created, the randomized value will be generated which will satisfy only the pure variable functions. This will limit the total number of combination of the values which variables can take.

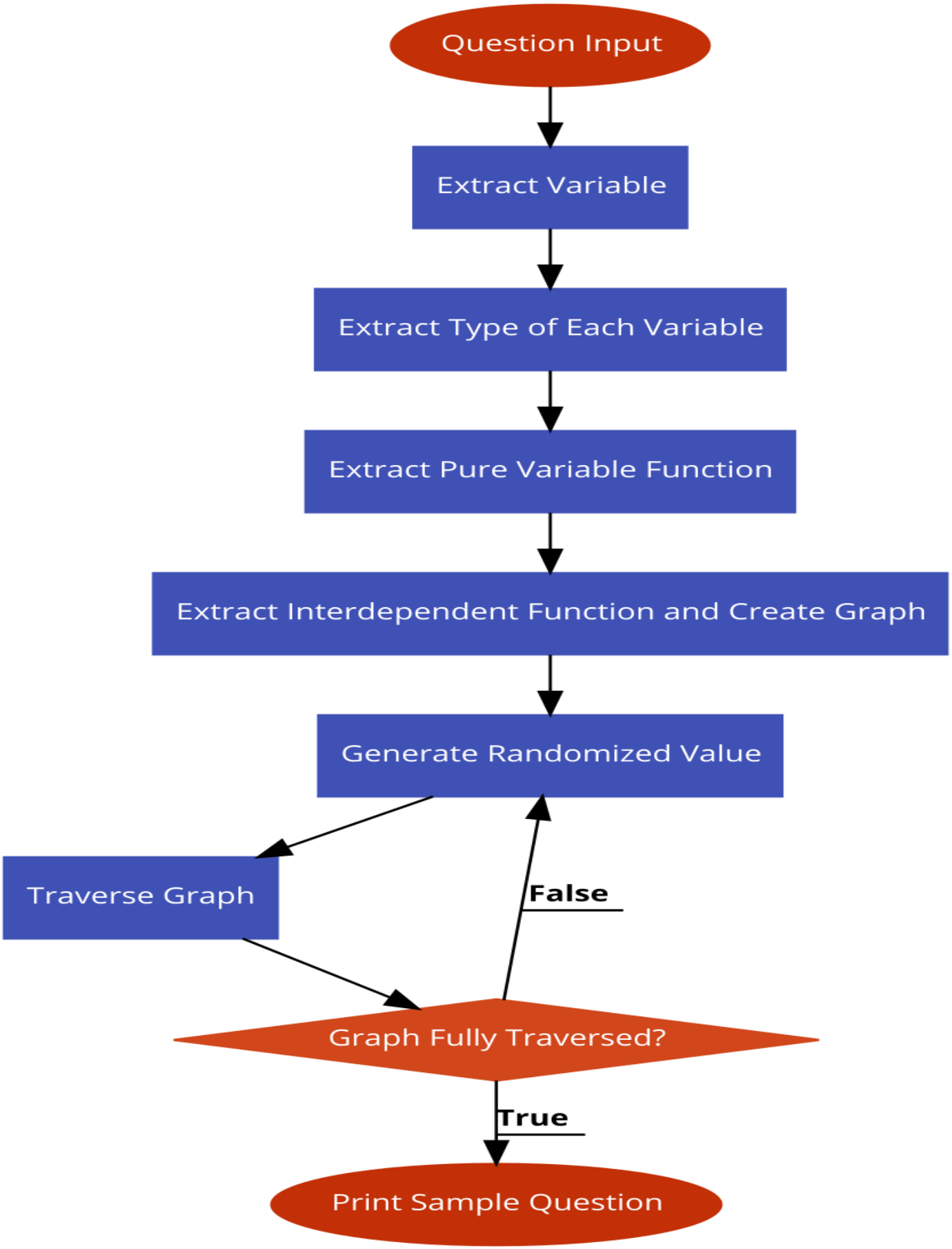
V. Graph traversal:

After the randomized value is assigned to each variable using pure variable function, graph is traversed. Now using the Depth First Search(dfs) method, traverse

the graph by evaluating all the edge. We will traverse each edge only when the function corresponding to that edge has be evaluated and has been satisfied. After dfs on each connected component is done, we check if all the edges has been traversed or not.
If all edges are traversed then, our generated variables satisfied all the constraint provided by the user else go to step IV.

VI. Question generation:

Here, we are not changing any structure of the question, we will be replacing only the variables generated from the step IV and V.


Question Input
Extract Variable
Extract Type of Each Variable
Extract Pure Variable Function
Extract Interdependent Function and Create Graph
Generate Randomized Value
Traverse Graph
False
Graph Fully Traversed?
True
Print Sample Question

8. Features of GUI :

I. *Save :*

Save button will save the entered question and constraint in the directory in which python file is present with the name of unique ID provided in the ID entry button by the user

.

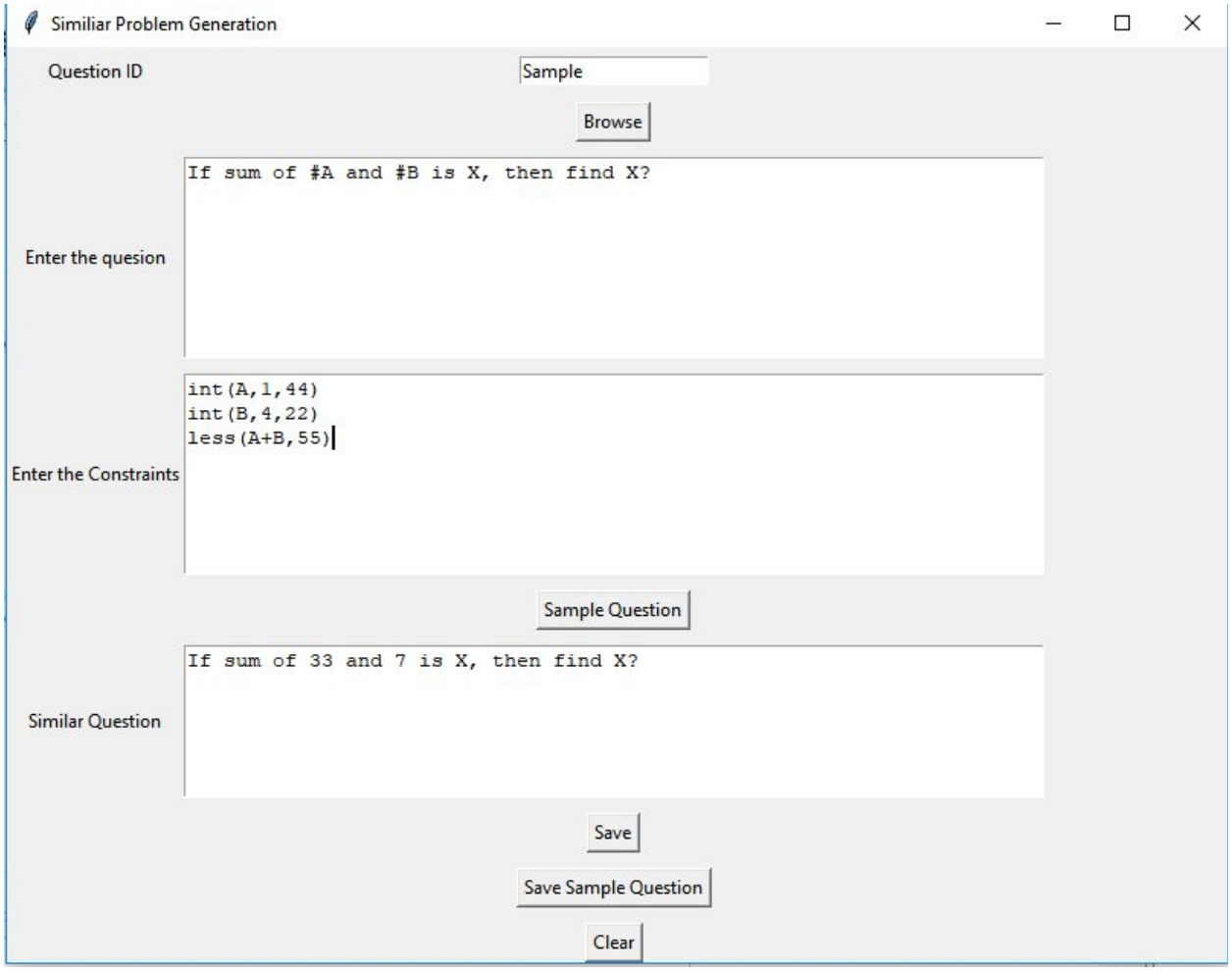
II. Save Sample question :

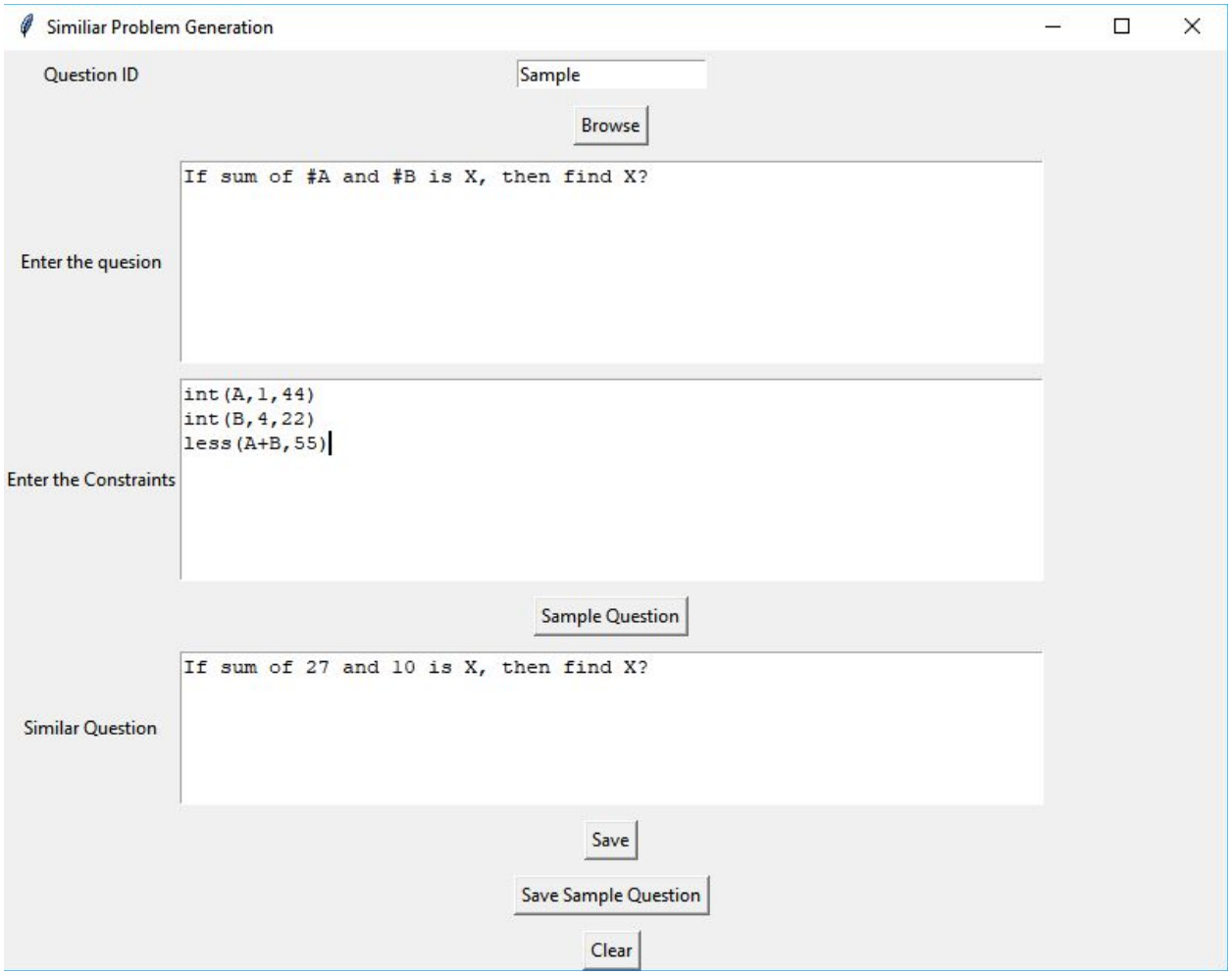
“Save sample question” button will save the generated sample question in the .txt file with the name of ID of the question

III. Browse :

“Browse” button will help in retrieving already inserted question and constraint automatically by entering the question ID in ID box.

9. Experimental Results :


Similiar Problem Generation
Question ID
Sample
Browse
Enter the quesion
If sum of #A and #B is X, then find X?
Enter the Constraints
int(A,1,44)
int(B,4,22)
less(A+B,55)
Sample Question
Similar Question
If sum of 33 and 7 is X, then find X?
Save
Save Sample Question
Clear



10. Conclusion and future work

Using the above method we can generate a question in randomized way. In the generated question, the structure of question is exactly same and only variables are changing. So, using NLP techniques, generated question can be rearranged to form a same question but with different structure. Also, carrying the work forward, generated question can be sorted on the basis of toughness of question which can help student to get more difficult problem once they are capable of solving easier one.