Automatic Generation of Programming Word Problems

Rajas Vanjape, Vinayak Athavale, Manish Shrivastava
International Institute of Information Technology Hyderabad
(rajas.vanjape,vinayak.athavale)@research.iiit.ac.in, m.shrivastava@iiit.ac.in

Abstract

Programming word problems are problems that can be solved using computer programs or algorithms. They are used to test a person's knowledge in applying different programming constructs to solve problems in real life and are frequently used as assignments in introductory programming courses. This paper outlines a method to automatically generate simple, realistic programming word problems. We first synthesize a problem automatically in a high-level graph-based notation and then incorporate it into a story. We perform a study with human judges to determine the robustness of our method and find that the problems generated are sufficiently realistic.

1 Introduction

A Programming Word Problem (PWP) is a problem written in natural language which can be solved using a computer program (Athavale et al. 2019). They have specified input/output formats, and there exists an algorithm which will give the correct answer for any input. They contain a certain narrative and describe a problem. They are frequently used in introductory programming assignments to test the knowledge of a student by making him/her solve a real life problem using a computer program.

Writing new problems like these require a huge amount of manual effort as they require the creation of 2 things: first, the generation of the actual problem and second, a narrative in which one can present the problem. In this paper we outline a method to automatically generate PWPs which require an algorithm to solve them.

PWPs can be classified into two types.

- Simple Narrative - These problems don't contain any real world object or story and ask for finding certain quantities from input.
- Story Based - These problems contain a story relating to an imaginary scenario.

We define a new graph based representation for a program which is loosely based on the graphical representation of algorithms (Jones 2019). Nodes are various datatypes and edges are operations on these nodes. We then create a few examples of programming problems in this representation. We use templates to make a story for each operation and then combine the stories together using certain rules. We generate problems of both types, simple narrative and story based.

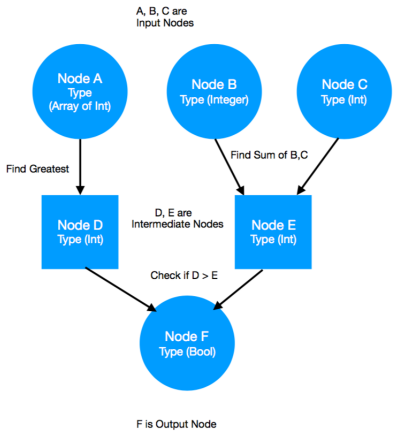

Figure 1: Graphical representation for the problem: *Check if the greatest number in array A is greater than the sum of B and C*

2 Formalization of a Programming Problem

In this section, we formalize a PWP using a graphical representation.

Programming word problem (PWP)

It is the set of sentences S which contains known quantities K and unknown quantities U. And there exists an algorithm M which when applied on K will compute U [1].

Definitions

Nodes Nodes are datatypes used in any general purpose programming language. eg: integer, array[of integers],

[1] We only generate programming problems having only one unknown quantity

string[array of characters], boolean[construct having value T or F]. We propose 3 types of nodes

- start nodes (corresponding to inputs)
- intermediate nodes corresponding to intermediate states in the computation
- terminal nodes (corresponding to outputs)

Transition Operation or step which will take us from one node/datatype to another. This will generally refer to one computation.

Graphical representation

A PWP is represented as an inverted tree, with input nodes represented as leaves and the output node as the root. Each node represents a variable which is used in the programming problem. Every node except the input nodes have incoming edges from all immediate nodes on which the variable depends on. An example is shown in Figure 1. The objective of the PWP is to find out the value of output nodes from given input nodes. All variables which are used while defining a problem are present as some nodes in the graph. In this work we only consider graphs having one output node.

3 Simple Narrative Problem Generation

In this section we describe the process of converting graph representation to a simple narrative based programming problem. We do this by generating templates for each node in the graph, organize them in a particular order and then combine them to make the final PWP.

Templates

We manually created a set of templates for each operation.

Input Nodes Input nodes have templates which are just a definition. They are of the form *__NAME__ is a __datatype__*.

Intermediate Nodes The template for every intermediate node has the operation performed to generate the node along with the operands for the operation. *__NAME__ is the __operation__ on __Node1__ and __Node2__* .

Output Nodes The templates for output nodes are of the form *Find __NAME__ which is made by __operation__ on __Node1__ and __Node2__* .

Combining the statements

To make this problem look more natural we combine certain statements and remove the definition statements of certain intermediate nodes.

Considering the problem in Figure 1: We replace *D* by *greatest number of A*. and *E* by *sum of B and C* and then remove the definition statements of node D and E.

Making the final problem as follows: *A is an array. B is a Number. C is a number. Check if the greatest number of A is greater than sum of B and C*.

Node	Template
A	Charlotte has n boxes containing chocolates.
B	B is number of chocolates with Laura
C	C is number of chocolates with Paul
D	D is maximum number of chocolates in Charlotte's boxes
E	E is the sum of B and C
F	Find if D is greater than E

Table 1: Theme based Templates for the nodes in Figure 1

Choosing which nodes to remove While combing the statements in the problem we remove certain nodes. We only combine statements in such a way that intermediate nodes having only one out degree are removed. This will ensure that the generated problem is consistent.

4 Story Based Problem Generation

Templates

We created 3 different themes for story based problems. For each theme and each operation, we created a template. An example of theme based templates for problem in Figure 1 is given in Table 1. The method used for combining and ordering statements is same as that for simple narrative problems.

The final generated problem is *Charlotte has n boxes containing chocolates. Find if the maximum number of chocolates in Charlotte's boxes is greater than sum of number of chocolates with Paul and Laura*.

5 Related Work

Our work relates to Generating Programming Exercises. (Zavala and Mendoza 2018) generated programming exercises using a semantic-based Automatic Item Generation Approach for generating contextualized programming exercises. They start with a template containing variables and formulas which are resolved at the time of generation. This approach is able to generate programming exercises in different contexts but lacks in generating completely different algorithmic exercises.

6 Evaluation and Results

We automatically created a few graph representations. The input given to the algorithm are the input nodes and N (number of nodes) We generated 14 simple narrative and 6 story based problems from them. 5 students who had previously been Teaching Assistants of Introductory Programming Courses were asked to rate the 20 generated problems on parameters of relevance on a scale of 1 - 5. Evaluators were also additionally asked if the given problems were grammatically correct. The mean relevance score for generated simple narrative problems came out to be 4.06 while that for generated story based problems was 3.87.

References

Athavale, V.; Naik, A.; Vanjape, R.; and Shrivastava, M. 2019. Predicting algorithm classes for programming word problems. In *Proceedings of the 5th Workshop on Noisy User-generated Text, W-NUT@EMNLP 2019, Hong Kong, China, November 4, 2019*, 84–93.

Jones, J. 2019. Abstract syntax tree implementation idioms.

Zavala, L., and Mendoza, B. 2018. On the use of semantic-based aig to automatically generate programming exercises. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, 14–19. New York, NY, USA: ACM.