Final Year Project

AI Tutor for Programming Class (the educational platform)

Individual Final Report

Group Members:
Kung Tsz Ho     3035176729
Chan Ka Yeung  3035181580
Lor Cheuk Hin  3035185108
Hui Pak Nam     3035185732

Writer: Hui Pak Nam (3035185732)
Supervisor: Dr. C.K. Chui
Date of Submission: April 15, 2018
ABSTRACT

An educational platform called Code Nova is being developed in this project. The goal of this platform is to help users, especially students, to learn programming efficiently and effectively. By data mining techniques, the platform will analyze the learning progress of users. Based on the analysis of a user profile, the system will give some advice or tip to the user on how to modify and improve the code, and that is why this system is called A.I. Tutor. It makes use of artificial intelligence (A.I.). Moreover, some machine learning algorithms will be applied and analyzed on this project to identity different approach to solve one programming task. This platform also enables students to learn in a collaborative environment by sharing their ideas on a problem. This is the final report of the Final Year Project which describes the work that have been done, the challenges that have been encountered and how to resolve these challenges.

ACKNOWLEDGEMENTS

I would like to express my great appreciation to the supervisor, Dr. C.K. Chui for his suggestions and encouragements during the plan and development of this project. I would also like to thank my groupmates Kung Tsz Ho (Bob), Chan Ka Yeung and Lor Cheuk Hin (Marco) for their assistance, collaboration and constructive progress during this Final Year Project.
# TABLE OF CONTENTS

Abstract ........................................................................................................................................... 2  
Acknowledgements .......................................................................................................................... 2  
List Of Figures .................................................................................................................................. 4  
List Of Tables .................................................................................................................................... 6  
1. Introduction ................................................................................................................................. 7  
   1.1 Background .............................................................................................................................. 7  
   1.2 Objectives ................................................................................................................................ 9  
2. Methodology ............................................................................................................................... 11  
   2.1 Selection Of Programming Languages ..................................................................................... 11  
   2.2 Web Framework And Web Design ............................................................................................. 12  
   2.3 Database Schema ..................................................................................................................... 13  
   2.4 System Logic Flow .................................................................................................................... 15  
      2.4.1 Exercise Suggestion .............................................................................................................. 16  
      2.4.2 Peer Helping System .......................................................................................................... 18  
      2.4.3 Submission Engine ............................................................................................................ 19  
      2.4.4 Profile System ................................................................................................................... 20  
      2.4.5 Skill-Tree And Analyzer ...................................................................................................... 21  
      2.4.6 Data Clustering ................................................................................................................ 23  
      2.4.7 Data Cleaning ................................................................................................................... 26  
      2.4.8 Adviser ............................................................................................................................. 27  
3. Experiments And Results ............................................................................................................. 28  
   3.1 Data Preparation ..................................................................................................................... 28  
   3.2 Full Skill-Tree ........................................................................................................................ 32  
   3.3 Analyzer – Tokenization ......................................................................................................... 36  
   3.4 Analyzer – Abstract Syntax Tree .............................................................................................. 37  
      3.4.1 Introduction To Abstract Syntax Tree ................................................................................. 38  
      3.4.2 Tree Conversion In Json Object ......................................................................................... 39  
      3.4.3 Skills Detection ................................................................................................................ 41  
      3.4.4 Obstacles – Lists In Python ............................................................................................... 45  
      3.4.5 Repeated Skills ................................................................................................................ 47  
      3.4.6 Testing Results .................................................................................................................. 48  

4. Improvements .......................................................................................................................... 50
   4.1 Skills Tree Vs Data Clustering ........................................................................................ 50
   4.2 Design Of Analyzer ........................................................................................................ 50
5. Conclusion And Summary .................................................................................................... 52
References .................................................................................................................................. 53

LIST OF FIGURES

Figure 1: The top 10 programming languages. The rankings are produced by IEEE Spectrum among 48 programming languages ................................................................. 11
Figure 2: The homepage of Code Nova .................................................................................. 12
Figure 3: The database schema of the system ................................................................. 13
Figure 4: The logic flow behind the system ........................................................................... 15
   Figure 4a: Exercise suggestion function logic-flow ............................................................. 16
   Figure 4b: Peer helping function logic-flow ......................................................................... 18
   Figure 4c: Submission engine logic-flow ............................................................................. 19
   Figure 4d: Analyzer logic-flow ........................................................................................... 22
   Figure 4e: Data clustering logic-flow .................................................................................. 23
   Figure 4f: Adviser logic-flow .............................................................................................. 27
Figure 5: Exercises Selection in different Modes on Code Nova ........................................... 16
Figure 6: The detailed course description of the course ENGG1111 at HKU ....................... 22
Figure 7: A simplified version of skill-tree of Python ........................................................... 22
Figure 8: Different solutions to ‘Fibonacci Problem’, in particular, the leftmost program is a ‘hardcode’ program, which is a possible solution but is not a recommended approach ...... 24
Figure 9: Different clustering approaches are applied on the same dataset (TSORT problem). K-mean clustering algorithm gives too many clusters to this problem, which is unsatisfactory ...... 25
Figure 10: The results after processing HTML code on the right are displayed on the left, without styling ................................................................................................................. 29
Figure 11a: Test cases generations for ‘Sorting (TSORT)’ ..................................................... 30
Figure 11b: Test cases generations for ‘The Lead Game (TLG)’ ........................................... 30
Figure 12a: A user submits a program to the sorting problem ‘TSORT’ ............................... 31
Figure 12b: All test cases passed because ‘expected output’ and ‘actual output’ are matched… 31
Figure 13: Original skill tree structure ………………………………………………………… 32
Figure 14a: A code using bubble sort algorithm in expectation …………………………… 33
Figure 14b: Skills detection after submitting a code in bubble sort algorithm…………… 33
Figure 15a: A code using build-in sort function without expectation …………………… 33
Figure 15b: Skills detection after submitting a code in Figure 15a ……………………… 34
Figure 16: Current version of skills Tree after adding build-in or member functions … 35
Figure 17: The results after tokenizing a code in Python ………………………………… 36
Figure 18: The results after converting a program in abstract syntax tree ………………… 37
Figure 19: Definition of abstract grammar in Python 3.6 …………………………………… 38
Figure 20: Abstract syntax tree of the code in Figure 18 in JSON format ……………… 39
Figure 21: Abstract syntax tree of the code in Figure 18 in tree diagram ……………… 40
Figure 22: If the node type of an abstract syntax tree is ‘List’, then a list is detected, so the skill-tree is updated …………………………………………………………………………… 41
Figure 23: If the node type with ‘Name’ of an abstract syntax tree is ‘input’, then the skill ‘basic input’ is detected. Similar applies to ‘basic output’ with keyword ‘print’…………………………… 41
Figure 24a: Abstract syntax tree to Program A ……………………………………………… 42
Figure 24b: Abstract syntax tree to Program B ……………………………………………… 42
Figure 25: Recursion detection in the analyzer ……………………………………………… 43
Figure 26: Depth of loops detection and calculation………………………………………… 44
Figure 27: The same variable can be a list or not a list ……………………………………… 45
Figure 28: The calculation of array size and dimension in the analyzer …………………… 46
Figure 29a: Some testing on the analyzer – 1 ………………………………………………… 47
Figure 29b: Some testing on the analyzer – 2 ………………………………………………… 47
Figure 30: Using ‘sorted’ build-in function will regard the same as using list ‘sort’ member function ………………………………………………………………………………… 48
Figure 31a: A correct submission from user ‘tt1050664’ on Codechef ‘Tsort’ problem … 48
Figure 31b: A correct submission from user ‘srikar_ry’ on Codechef ‘Tsort’ problem …… 49
Figure 31c: A correct submission from user ‘adarshpandey’ on Codechef ‘Tsort’ problem…… 49
Figure 32: The A.I. Tutor can give advices to user by displaying the essential part related to a specific skill……………………………………………………………………… 51
LIST OF TABLES

Table 1: Some examples of arrays and their dimensions. ................................................. 45
1. INTRODUCTION

1.1 Background

There is an increasing trend of students learning programming around the world. Programming becomes indispensable in most industries. For example, selling or advertising a product to an online customer via Internet, enhancing network security in a company, predicting the rise and fall in the stock price. Tim Cook, the CEO of Apple Incorporated, one of the largest technology companies in the world, said that learning coding is more important than studying English [1]. Despite there being a growing number of students learning programming, there are not enough teachers with coding skills [2]. Hence, teaching resources are insufficient. Moreover, most universities set a quota limit on the number of enrolled students of their programming courses, so some students are not allowed to enroll his or her desired programming course. Those students would need to self-study, but this would become a bigger challenge. In this project, the coding educational platform Code Nova aims to assist students with difficulties in coding and alleviate the workload of teachers to answer all enquiries from students.

There are many websites and forums dedicated to code learning such as Codecademy [3] and Codechef [4]. Although it provides technical support when a user encounters difficulties in coding, it is commercial and is not designed for a class. It does not provide a workspace for students to discuss a programming task, and teachers cannot track the learning progress of every student online, so it does not enhance the quality of teaching. For instance, if many students in a class do not fully understand a programming skill and their teacher does not know about this, the teacher will not spend more time on this skill to make the concept clear and this lowers the learning effectiveness.

GitHub [5], another popular online platform, allows a group of users working together, but it is also designed for industries or teams working on sophisticated projects. Thus, it is not suitable for simple educational uses. Both platforms mentioned above have their own features and characteristics, but they have also disadvantages. They cannot act as assisting tools in a programming class because they do not facilitate teaching. In this project, Code Nova will provide
a virtual classroom for learning coding. When a teacher assigns classwork to students, the platform records the coding progress of students. Hence, the teacher will know the coding status of every student immediately online. Furthermore, it is easier for the teacher to find the weakness of each student, so more focus will be put on the gap. When a student needs some help, he/she can ask the teacher directly on the platform. Thus, the teacher can directly view what the student has done on the platform and make some comments on the platform, instead of receiving emails from the students, downloading the code and testing the code again. Hence, *Code Nova* is a convenient programming platform for teachers and students, and tutors if necessary.

Other than supporting students in learning coding, *Code Nova* engages students working and learning together. During the process of sharing their ideas and thought, the student who needs help will learn some useful techniques, while the helper will consolidate own knowledge. According to a study [6], students who collaborated with others in the class had a better performance than those who worked individually. In a collaborative learning environment, all students learn faster and more effectively.
1.2 Objectives

The educational platform *Code Nova* is designed for different scales of programming classes at school, as well as persons who are self-learning how to code. There are three main objectives in this project. First, *Code Nova* will be able to track and analyze the learning progress of every user, such that users and their administrators such as teachers will be able to know what tasks the users have completed to facilitate learning and teaching.

Second, the platform itself will be able to provide some advices or suggestions to users when they have difficulties in a programming task, so the users do not need to spend much time to ask another person for help. If that person does not familiar with this problem, he/she will also need to understand the problem before reading the code, which results in spending too much time. If that person does not derive a solution of the problem eventually, the time spent will be wasted. By the aid of artificial intelligence, when a user needs help, the platform instantaneously offers to help, which significantly enhances the learning efficiency. In this project, such technology function is called A.I. Tutor. This tutor acts as a teacher in a class. When a user encounters difficulty on a problem, he/she can ask A.I. Tutor for some hints with an instantaneous response instead of sending an email to his/her teacher to wait for reply.

Finally, there is a peer helping function in the platform. Any artificial intelligence needs to be trained and tested to achieve a high accuracy on its functionalities. For a complicated task, it takes longer time and larger set of training data. Hence, when a user encounters some complicated questions, the A.I. Tutor may not help the student solve this problem in an early development of the educational platform. In this case, the student can use the helping function to find a matching helper. After the matching, the platform will assign both the helper and the user who needs help to a room so they can work together, which has a similar function as *Google Docs* [7]. Different users can create and edit the code concurrently. The platform will also include compiling and testing functions, so all users will view the test results or the error messages from the platform directly. The learning effectiveness is greatly boosted during a collaboration process.
In this final report, it will go over the system overview and describe how these objectives can be accomplished in Chapter 2. Then it is followed by a detailed discussion with some results and experiments about some preparation works, as well as some design and implementation in the analyzer – the skills detecting program in Chapter 3. Next, it provides some improvements that might be feasible in Chapters 4 on our project. Finally, it ends with a short summary and recommendation in Chapter 5.
2. METHODOLOGY

2.1 Selection of Programming Languages

In the early development of *Code Nova*, one programming language had to be selected to implement. Teachers of different programming classes may teach different programming languages, but each programming language has some exclusive characteristics and features. Hence, distinct programming languages contain distinct set of skills. Each skill represents a feature of a programming language, and it may be common among multiple programming languages.

When the first programming language to implement in the system was decided, two factors were considered: (1) popularity, especially for students, and (2) ranges of application. According to a survey conducted by *IEEE (Institute of Electrical and Electronics Engineers) Spectrum* in 2017, Python is the top programming language in terms of popularity (See Figure 1) [8]. Moreover, Python has a wide variety of applications, such as web and internet development, science, education, software development and business. Hence, this programming language is also in high demand by recruiting companies and it attracts many students or graduates to learn.

![Figure 1: The top 10 programming languages. The rankings are produced by IEEE Spectrum among 48 programming languages. [8]](image-url)
Furthermore, there are many Python deep learning modules and libraries, such as SciKit-Learn [9], which is simple and efficient for machine learning and data analysis. It also supports a broad range of algorithms such as data clustering, which will be useful in this project. Therefore, Python is not only the first programming language to be implemented / supported towards the users in Code Nova, but it is also the programming language to develop the whole system in this project. As a result, some useful libraries in Python can be utilized to analyze codes.

2.2 Web Framework and Web Design

To build an online educational platform, a web frame is needed to handle most of the core functions such as authentication. Since Python is selected to develop data analysis and perform machine learning, a web framework using Python is preferred.

Django is selected as the web framework of this project, which is the most popular web framework currently. Other than this, the reason of selection Django as the web framework is that it makes an application cost-effective and efficient to help development as fast as possible. It is also scalable that can meet heavy traffic demand and avoid a lot of common security mistakes [10].

As this project supports peer helping (collaboration) system, WebSocket is used to keep all clients connected to the server. Django channels with Redis are used to handle WebSockets, which is designed for exchanging messages between the client and the web server in real time. This gives more flexibility in development while client can both actively and passively communicate with the server. The also allows the server performing logging and WebSocket management.
2.3 Database Schema

Figure 3 shows the database schema in the system.

- **User (id, username, password, reply_channel)**
  
  Foreign Key: None
  
  This table is used to identify a user. In particular, reply_channel refers to the web socket.

- **Student (id, user, profile)**
  
  Foreign Key: {user} referencing User {id}
  
  This table is used to identify a student. In particular, {profile} refers to the skills equipped in terms of frequency used over all programming tasks on *Code Nova*, as described in Section 2.4.4.
• Room (id, exercise, code, participant, author, chat_history, require_help)
  Foreign Key: {author} referencing Student {user}, {exercise} referencing Exercise {id}
  This table is used to identify a room, which is a workspace created when an individual user
  wants to solve a particular problem. In particular, {participant} refers to the room members,
  chat_history refers to the past conversations among room members and require_help refers
  to whether the room owner needs for help in the peer helping system, as described in
  Section 2.4.2.

• Version (id, room, exercise, code, result, version_tree, cluster, overall_success)
  Foreign Key: {room} referencing Room {id}, {exercise} referencing Exercise {id},
  {cluster} referencing Cluster {id}
  This table is used to identify a submitted version by a particular exercise. In particular,
  {version_tree} refers to the version tree (the set of skills applied) generated by the analyzer,
  as described in Section 2.4.5. {cluster} refers to the data cluster that this code belongs to
  in this exercise, as described in Section 2.4.6. Finally, {overall_success} refers to the
  proportion of test cases that match the expected output.

• Cluster (id, exercise, data_count, center, necessary_skill, redundant_skill, character_skill,
  other_skill)
  Foreign Key: {exercise} referencing Exercise {id}
  This table is used to identify a data cluster, in which {necessary_skill}, {redundant_skill},
  {character_skill} and {other_skill} are exactly the same as the description appeared in
  Section 2.4.6.

• Exercise (id, title, material, test_case, template, common_skill)
  Foreign Key: None
  This table is used to identify an exercise. In particular, {material} refers to the problem
  statements, descriptions, etc. {template} refers to a template code that is given to the
  student. Finally, {common_skill} refers to the skills set after data cleaning, as described in
  Section 2.4.7.
2.4 System Logic Flow

Figure 4 shows the flow of logic behind the system, which briefly describes some of the functionalities and the scenarios that might happen in Code Nova.

In the next few sub-sections, this logic-flow diagram will be used to describe some methodologies such as theories and experiments according to different major functionalities in front-end, as well as some decision making and results in back-end work.
2.4.1 Exercise Suggestion

First of all, when a user of Code Nova logged in, he/she will be directed to a page with 3 different columns (Figure 5). Each column corresponds to a learning mode. Each mode is closely related to the skill equipped by the user. (Here, for the full details of skill and how the system detected a user has equipped that skill, they can be found in Section 2.4.5. For example, if-else statements, for-loop and functions declarations are different skills.)

![Figure 4a: Exercise suggestion function logic-flow]

![Figure 5: Exercises Selection in different Modes on Code Nova]

Suppose User X wants to do an exercise, but he/she does not have any preferences on programming tasks in Code Nova. Then the A.I. Tutor will suggest an exercise to him/her according to the learning mode that User X has picked.

1. ‘Learn New Skill’: This learning mode is suitable for users who want a new challenge or learn new programming knowledges. The system will randomly select a new skill, in which User X has not equipped it yet. Then the system will search for a task that is solved by most users using that skill. When a searching result is found, the A.I. Tutor will suggest
the corresponding exercise to User X. The user can choose whether he/she accepts the suggestion. If yes, he/she will be directed to the page of the corresponding task.

2. ‘Reinforce Skill’: This learning mode is suitable for users who wants to have better preparation and revision on what he/she has learnt. The system will randomly select a skill that has been equipped by the user. Then the system will search for a task that is solved by most users using that skill, but User X has not yet solved it. When a searching result is found, the A.I. Tutor will suggest the corresponding exercise to User X that is similar to scenario/Mode (1).

In the searching approach, the system will avoid selecting a task in which it requires a skill that User X has not yet equipped. For instance, suppose User X has equipped skills S₁, S₂, S₃ (detected by A.I. Tutor). If User X picks this mode ‘reinforce skill’ for exercises suggestion, then the A.I. Tutor will attempt searching all programming tasks that have not yet solved by X. According to all the received correct submissions of a programming task, the A.I. Tutor is able to find what (common) approaches that are required to solve that task by brute-force (or more intelligently, by data clustering, so the A.I. Tutor can distinguish the major clusters that are using a particular set of skills). If there exists a submission that requires the set of skills that is the subset of skills S₁, S₂, S₃, then this task is suitable for User X to reinforce skill. Otherwise, this task is not preferred and not suggested by the A.I. Tutor.

3. ‘Learn Specific Skill’: This learning mode is suitable for users who wants to train a particular skill or knowledge. This is different from the ‘learn new skill’, because it is arbitrarily assigned by the A.I. Tutor. Suppose User X has not yet learnt ‘loops’ and ‘recursion’. However, he/she knew that ‘recursion’ might be too difficult for this user, so he/she does not want to learn ‘recursion’ before learning ‘loops’. Then if he/she picks the mode ‘learn new skill’, the system would possibly assign ‘recursion’, that is not User X wants to learn. Hence, User X can pick a skill (e.g. ‘loops’) to learn among all available skills on Code Nova. Then the system will search for a task that is solved by most users using that skill, and similar to Scenario/Mode (1). After the user is directed to the task page, he/she can try to solve this problem using ‘loops’.

17
2.4.2 Peer Helping System

Peer helping system (or the collaborating system) is one of the main features of this project. It allows different users in different places coding and working at the same coding environment together. To build a helping function, it should enable two or more programs on different devices or computers communicating via a network by connecting to sockets. It can be done by multi-threading.

Suppose a user needs help and uses the helping function, the A.I. Tutor will search for an online user who is able to offer help. Undoubtedly, if the system randomly picks an online user, the chance of successfully providing help is very low because the user may not know the required programming language of the problem, or the user knows that language but he/she may not be acquainted with it. Then the helping function becomes ineffectiveness. Hence, the A.I. Tutor should be strong enough to do a better search such that the user will not waste time on an inefficient guidance from the wrong helper.

To solve this issue, suppose User X asks A.I. Tutor for help on Question A. Then A.I. Tutor will help find a helper for User X. Helpers who have solved Question A will have a higher priority to be select as a helper. Then the A.I. Tutor will do a data clustering on those helpers’ profile (refer to Section 2.4.4) against the skills (features) that have been equipped. To find the most suitable person, the A.I. Tutor will prefer one that has abilities (the set of skills that he/she equipped) that are close to User X. The A.I. Tutor will apply the Euclidean Distance Formula to find the closeness between the features (or the center of their data clusters) corresponding to 2 different users.
$\text{distance}(p, q) = \sqrt{\sum_{s \in \text{skills equipped}} (p_s - q_s)^2}$

(Here, $p_s$ refers to the frequency that User $p$ applies skill $s$ on Code Nova. Similar applies to $q_s$.)

Hence, the helper matched by the A.I. Tutor is expected not to use advanced skills to teach User X, which leads to a better scenario that the helper seeker can understand what the helper says more easier. In addition, that helper has sufficient knowledge to teach the help seeker how to solve a programming task which had been solved by the helper before.

When the platform successfully finds a helper, the server will direct this helper to a workspace such that both the helper and the user who needs help will connect in the same room and they will be able to discuss the problem with each other. The system will keep track of all actions from different users on the coding environment and synchronize among all users’ workspace. Hence, all users in the same room can work and edit the code together, similar to Google Docs [7].

### 2.4.3 Submission Engine

Similar to other programming platforms such as Codecademy, after a user finishes and submits an exercise, the compiler will check whether there is a compilation error in his/her code. If there is a compilation error, then the A.I. Tutor will notify the user where the error comes from. Next, if there are no compilation errors, the system will use different testcases as input to test the submitted program. The user will receive the full feedback against test cases, for instance, what are the expected (correct) output and the output generated by the user’s program, and evaluate their difference.

*Figure 4c: Submission engine logic-flow*
A user can save his work if he/she has not yet finished the exercise. After making a save, he/she can return to the room and continue editing or reviewing his/her code. Thus, a user does not need to worry about whether their works are lost after logging out or closing the browser, so he/she can spend over days to solve a question if necessary.

When a user submits a program which generates correct output, he/she solves this task. Hence, the system will save his program in the database for further analysis, and update personal profile, which will be described in details in Section 2.4.4.

2.4.4 Profile System

Every user will have an individual profile. The data logger in the platform will record his or her learning progress and equipped skills of a programming language. This record will be saved in the database. Suppose a user submits a code to a programming task which generates correct answers, the analyzer (refer to Section 2.4.5) will analyze the structure of the submitted code. After the analysis, the system will find out the skills involved inside the program. Finally, it will update the user profile and report that this problem was solved by this user and record which skills were applied to this problem.

Basically, this profile system records three important set of data. The first is the set of programming tasks that a user has solved. The second is the set of skills equipped, which is stored in a JSON tree. Each node of the tree corresponds to a skill, which is associated with a number: the frequency that a user has applied this skill over all programming tasks on Code Nova. The third is the combination of the previous two: for each programming task that a user solved, the set of skills that the latest version of submission has been applied on that task. This profile system is useful when the peer helping system, as mentioned in Section 2.4.2, wants to find a suitable person who has similar profile as the help seeker.
2.4.5 Skill-Tree and Analyzer

Over the descriptions of the previous section, skill is indispensable to the whole project. *Code Nova* aims to provide an educational platform for programming beginners to practice and discover more on a programming language. A user can apply his programming knowledge and reinforce skills by solving different tasks on *Code Nova*. Moreover, if a user wants to learn more about a programming language, he/she can learn something new and something more beyond his/her knowledge.

By studying the frequency of each skill occurred in all correct and wrong submitted codes of a specific problem, the A.I. Tutor can analyze what the common feature (skills) of a correct program is. Oppositely, the A.I. Tutor can analyze what the common mistakes were made by the whole class, for instance, the A.I. Tutor can distinguish which skill is essential in this problem, such that it is very likely that a submission that does not possess that essential skill is wrong.

A skill-tree can be regarded as unlocking different achievements in a video game. Some challengers intend to unlock all the achievements in the game and master this game. If a skill-tree is well designed and a programmer wants to master a programming language, he or she can attempt the related problems of each skill of that language on the platform.

As mentioned in Section 2.1, *Python* was chosen to be the first programming language to implement in *Code Nova*. The course curriculums of some elementary programming classes such as ‘ENGG1111 Computer Programming and Applications’ at the University of Hong Kong [11] were considered to build a skill-tree (Figure 6). Fundamental concepts of programming include input and output (I/O), control structures (if-condition, for-loop, while-loop), functions and recursion, arrays or multi-dimension arrays, data structures. It is considered that some skills can group together, e.g. for-loop and while-loop are both loops. Hence, some skills (e.g. for-loop) could be a sub-category of another skill (e.g. loop). Therefore, these elements were put into the skill-tree. Figure 7 shows a simplified skill-tree on the platform.

The full considerations of the skill-tree construction and their details will be shown in Section 3.2.
Figure 6: The detailed course description of the course ENGG1111 at HKU.

Figure 7: A simplified version of skill-tree of Python

Analyzer is a data processor that reads a user’s program and returns the list of skills equipped by the user. It acts as an important role in the whole platform because the profile system, the data clustering and the advisor replies on the results provided by the analyzer. For example, the profile system stores the frequency of a particular skill that applied by a user over all programming tasks on Code Nova.

Figure 4d: Analyzer logic-flow
The advisor will make a comparison between two codes from two different users over skills applied to suggest how a code can be improved by adding a specific skill on it or removing a specific skill from it.

The full implementation and consideration inside the analyzer will be covered in Section 3.4.

2.4.6 Data Clustering

Recall that one of the objectives of this project is that A.I. Tutor is to provide some advices or suggestions to users when they encounter difficulties in a programming task. However, it is not reasonable that the A.I. Tutor solves the programming tasks itself and then report to the user what methods the Tutor are using in the solution. In this project, A.I. Tutor will learn from the correct submissions submitted by the users, and report to the user that what methods the correct solution is using. This achieves the same outcome.

However, there are too many ways to solve a programming task. Take a typical ‘Fibonacci Problem’ as an example, this problem asks the user to write a program to read one positive integer \( n \) and print the \( n \)-th Fibonacci number. Refer to Figure 8, there are two major approaches to solve this problem in general: One approach is using loops, and the other approach is using recursion. However, if the upper limit of \( n \) is very low (e.g. 20), then the user can do hardcode (i.e. typing the output directly that corresponds to input, which is possibly equal to 1, 2, 3, ..., 20). If the A.I. Tutor learns from the ‘hardcode’ solution, it does not make sense again that the Tutor suggests a user to solve this problem using if-else condition only, without loops or functions.
Figure 8: Different solutions to ‘Fibonacci Problem’, in particular, the leftmost program is a ‘hardcode’ program, which is a possible solution but is not a recommended approach.

To help A.I. Tutor identify all possible approaches, data clustering is used. For each submission, the frequency of using each skill is used to be a feature. For example, if there is no for-loop in a program such as the leftmost program in Figure 8, the frequency of skills corresponding to ‘for-loop’ will be 0. There is an extra reason of doing data clustering. From two sets of features with one or two features having a small difference (e.g. one program used 2 if-conditions, while another program used 3 if-conditions), it is unreasonable to deduce that they are two different approaches. Hence, by doing the data clustering, the A.I. Tutor may place two data with similar sets of features into the same cluster, which means that they have almost the same approach. By comparing the size of each cluster, the A.I. Tutor can draw a conclusion that the largest cluster represents a good approach to the task, and the skills that mostly used by the largest cluster should be necessary to solve the task.

There are many ways to do data clustering. In this project, scikit-learn [9], a popular machine learning module in Python is focused and applied. This module has many functions to do clustering. During the development stage of this project, 3 different clustering algorithms were picked to do some experiments: k-means, mean-shift and affinity propagation. Principal Component Analysis (PCA) is applied on a dataset collected, which aims to reduce the dimensionality (depends on the number of features of each data) of a data set, while remaining the variation existing in the dataset, up to the largest extent. It was found that k-mean clustering algorithm generated too many clusters (almost double of the other algorithms), which had unsatisfactory result to identify and differentiate all approaches. Hence, k-mean clustering algorithm was eliminated among three.
Between the other two, it was found that the mean shift algorithm generates a reasonable number of clusters by testing. Moreover, by checking the necessary skills detected after each clustering algorithm, it was found that the mean shift algorithm tended to give a better result with matched expectation. Thus, the mean shift algorithm was adopted between this and affinity propagation.

Figure 9: Different clustering approaches are applied on the same dataset (TSORT problem). K-mean clustering algorithm gives too many clusters to this problem, which is unsatisfactory. (Remarks: This dataset will be described later in Section 3.1.)
After the data clustering, the database will give an identifier (primary key) to each cluster. Each cluster has 4 different defined skill sets:

- **Necessary skills set:** It includes all the skills where each of them was applied by not less than 65% of correct submissions. Hence, this set shows all the skills which are widely used, and tend to be essential elements to solve the task.

- **Redundant skills set:** It includes all the skills where each of them was applied by not more than 10% of correct submissions. Hence, this set shows all the skills which are used in some correct submission inside this cluster, but not important at all.

- **Character skills set:** It includes all the skills where each of them was applied by some correct submission, but not all correct submissions. Hence, this set shows all the skills which appear to be optional.

- **Other skills set:** It involves 4 special skills appearing in the skill tree, they are: ‘maximum if-depth’, ‘maximum loop-depth’, ‘maximum array size’ and ‘maximum array dimension’. They are statistical results that describe the usage of if-conditions, loops and arrays. For each of these statistical values among all correct submissions, their mode represents the majority view on the depth of if-conditions, loops, as well as the total size and dimensions of arrays required to solve the task.

### 2.4.7 Data Cleaning

In the recent version of skill tree as described in Section 2.4.5, there are about 140 different features / skills. However, it is very unlikely that a program does not involve all these features, but approximately 15 in average. Hence, to a specific task, there are a lot of unused features or rarely used features among all successful submissions. If every feature is put into account to do the clustering, it may contribute some low-level data error such that the result may become unsatisfactory. Hence, data cleaning with noises removal is necessary. The simplest way to do such data cleaning, is to remove all unused features among all submissions (i.e. the skills that have zero frequency on a task) before doing the clustering. The remaining skills set after the data cleaning is called ‘common skills’, which are stored in the database.
2.4.8 Adviser

Adviser is a system that the A.I. Tutor gives useful hints and suggestion to users when they need help or improvements on their solution. The most basic scenario is if a user encounters compilation error, then the A.I. Tutor will provide some suggestions that where the mistake is. If a user encounters ‘wrong answer’ in some test case, then the A.I. Tutor will provide some test cases in which the student generates a wrong out.

After the data processing in the analyzer and the data analysis using clustering, the A.I. Tutor learns from the data clusters and their related skills. After that, if a user has no ideas on how to write a program which generates matched output and he/she gets stuck in a task, then he/she can ask the A.I. Tutor (advisor) for help. This time, the A.I. Tutor displays some other useful information about this task to the user that there are different clusters related to this task with each cluster represents an approach. Other than directly knowing ‘necessary skills’, ‘redundant skills’, ‘character skills’ and ‘other skills’ in each of the clusters that are stored in the database, the A.I. Tutor will also give some hints to the user by suggesting 3 of the essential skills that the user can apply them in the code first. After the user uses these skills and still gets stuck in the task, the A.I. Tutor will suggest some improvement (e.g. another 3 essential skills that can be applied in the code), so the user can proceed his/her work.

Figure 4f: Adviser logic-flow
3. EXPERIMENTS AND RESULTS

In the chapter, some preparatory works will be introduced. In addition, some decision making and testing regarding to data analysis (analyzer) will be described in detail.

3.1 Data Preparation

In order to achieve a better accuracy of some functions in the system, such as analyzer, an open dataset is used to do the testing and check whether the generated results exactly match the expected results or not. This dataset [12] is called the ‘Codechef Competitive Programming’, which includes all the problem statements and solutions provided by users on the Codechef Website [4], which is also a programming platform for programmers training algorithms by hosting several virtual programming contests.

In the dataset, there are some programs using other programming language such as C++ and Java. Since Code Nova only supports programs submitted in Python 3 in this phase, the solutions submitted in Python 3 are taken from the dataset only.

Since Code Nova is designed for new programming learners, some main focuses are put on some ‘beginner’ and ‘easy’ difficulty-level questions rated by Codechef because other difficulties (‘Medium’, ‘Hard’, ‘Challenge’) might involve advanced algorithms such as binary search and graphs searching in which they are beyond the scope of this project.

After the filtering, it is found that a beginner-level question ‘Turbo Sort (TSORT)’ [13] has the highest number of submissions. Hence, all ‘TSORT’ submissions are used to test the accuracy of the analyzer, as well as data clustering. (‘TSORT’ is a standard sorting problem. A user is required to write a program which reads an integer first: the number of integers to be read in an array, then it reads all integers in the array, and finally output a list of integers in non-descending order.)
In order to utilize the dataset properly, a sorting question is also uploaded to *Code Nova*, with problem descriptions implemented in HTML as shown in Figure 10.

**Sorting (TSORT)**

*Source: Codechef* [https://www.codechef.com/problems/TSORT](https://www.codechef.com/problems/TSORT)*

Given the list of numbers, you are to sort them in non decreasing order.

**Input Specification**

\( T \): the number of numbers in list, then \( T \) lines follow. Each line contains one integer: \( N \).

**Output Specification**

Output given numbers in non decreasing order.

**Sample Input**

```
5
5
6
7
1
```

**Sample Output**

```
1
3
5
6
7
```

**Constraints**

- \( 1 \leq T \leq 30 \)
- \( 0 \leq N \leq 100 \)

*Figure 10: The results after processing HTML code on the right are displayed on the left, without styling.*

Moreover, the compiler and the advisor in *Code Nova* should know whether the newly received solution by a user is correct or not. Hence, test cases data of this problem is also required. To generate multiple test cases easily, a test case generator is also written for this question in C/C++ to avoid artificial features hiding in the test cases (unless some test cases are special boundary cases).
For example, Figure 11a and 11b shows the results in JSON format of the corresponding test case generator for problems ‘Sorting (TSORT)’ and ‘The Lead Game (TLG)’.

**Figure 11a:** Test cases generations for ‘Sorting (TSORT)’.

**Figure 11b:** Test cases generations for ‘The Lead Game (TLG)’.
After uploading test cases in JSON format on the server, users can start the exercises to check whether they will obtain the same output against the expected output corresponds to each test case (Figures 12a and 12b)

![Sample Input and Output](image)

**Figure 12a:** A user submits a program to the sorting problem ‘TSORT’.

<table>
<thead>
<tr>
<th>Input</th>
<th>18</th>
<th>8</th>
<th>8</th>
<th>true</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 12b:** All test cases passed because ‘expected output’ and ‘actual output’ are matched.
3.2 Full Skill-Tree

As mentioned in Section 2.4.5, the design of skill tree is mainly based on the current course curriculums of some elementary programming classes such as:

- Basic Input & Output
- Conditions (e.g. if-condition)
- Loops (e.g. for-loop and while-loop)
- Functions (with Recursion)
- Classes

Hence, the skill tree was designed as shown in Figure 13, which included the above skills and features.

Figure 13: Original skill tree structure
Normally, it is expected that users apply some sorting algorithms such as bubble sort and merge sort, to finish the sorting task (TSORT), such as the code as shown in Figure 14a. A code using bubble sort algorithm is usually implemented with a nested loop. For instance, a nested for-loop between Line 4 and Line 9 in Figure 14a. Any sorting algorithm in $O(n^2)$ time complexity such as bubble sort, insertion sort and selection sort is normally implemented in nested loop(s).

![Figure 14a: A code using bubble sort algorithm in expectation.](image)

Figure 14b: Skills detection after submitting a code in bubble sort algorithm.

However, it is found that some codes found in the dataset to the sorting problem ‘TSORT’ used an unexpected but acceptable method, such as using python build-in sort function, which were undiscovered by the skill tree. The code as shown in Figure 15a does not contain any nested loops.

![Figure 15a: A code using build-in sort function without expectation.](image)
Hence, to detect more methods in the analyzer, different python build-in functions are added in the skill tree. Similarly, some methods functions on string and list and some functions from some imported module (e.g. Collections) are included in the skill tree.

Apart from the skills as specified above, there are four exclusive indicators for judging the quality of a submitted code, they are: the maximum depths of if-conditions, the maximum depth of loops, the maximum dimension of arrays and the maximum size of arrays (if applicable). They may reflect the time complexity and the memory complexity of a solution. Take a simple example. Some users might submit solutions in nested loops, but the depths of loops are different. It is preferred to write a solution in $O(n^2)$ time using loops with depth of 2 rather than $O(n^3)$ time using loops with depth of 3. If a user submits a program with higher depths of conditions / loops or larger dimensions of arrays than most of other users, then his/her program might have poor quality and can be improved. Hence, it can be concluded that a correct program using lower depths of conditions / loops or smaller dimensions of arrays involves skills, so these four exclusive indicators are also put into the skills tree, to wait for an analysis.

After including skills as specified above, the new skill tree (which is the current version of skill tree) is constructed as shown in Figure 16.
Figure 16: Current version of skills Tree after adding build-in or member functions
3.3 Analyzer – Tokenization

Python, the implementing programming language of the platform, has many powerful functions. One of the functions is tokenizer, which transforms and rewrites a code into a list of separate elements in the program (Figure 17). The results of this function give the range of the line and column coordinates of each element, the type of the element such as ‘NAME’ (functions and variables) and ‘STRING’ (strings), as well as the value of the element.

This function may be useful in the project because if the code structure is analyzed directly by compiling and running the original code, infinite loop or compilation error may be encountered so no information in the program will be collected. If the code is pre-processed by tokenization, the structure of the code can be collected in a shorter time.

*Figure 17: The results after tokenizing a code in Python.*
Although tokenization provides a result in table form. There are some potential disadvantages of doing tokenization to analyze the program.

1. The program is split into small pieces, such as left parenthesis, double star, etc. However, it contributes nothing to the defined skill tree, i.e. it involves no skills.

2. Some keywords in Python with different usage, such as ‘for’ and ‘range’, are categorized as ‘NAME’ in the tokenization results. Hence, it makes the analysis become more difficult because most skills in the skill tree involves keywords in Python, e.g. ‘input’ and ‘print’ for basic input/output, ‘if’ for conditions, ‘def’ for functions. Since they have the same category ‘NAME’, this becomes less meaningful to the data analysis.

Hence, during the skills detection process, we adopt another approach in the analyzer – analyze its abstract syntax tree, which is similar to tokenization but display in a form with hierarchy.

### 3.4 Analyzer – Abstract Syntax Tree

Every programming language has its own syntax grammar. An abstract syntax tree of a program describes the program in terms of the syntax grammar in a hierarchy. To support conversion a Python program to its abstract syntax tree, a module ‘ast’ is imported (Figure 18).

![Figure 18: The results after converting a program in abstract syntax tree.](image)
3.4.1 Introduction to Abstract Syntax Tree

A Python program mainly consists of statements and expressions. A statement is possibly an if-condition, a function definition, a while-loop etc., while an expression is possibly an operation, a number, a name call etc. Take if-condition as an example, it consists of three components: the condition, the statements of body if the condition is true and the statements of the body if the condition is false. By this example, it can be imagined that there is a tree with the value of the root is ‘if’. There are three subtrees from this root in which each subtree represents the abstract syntax tree of each component. The definition of statements and expressions are listed in Python 3 Documentation Section 32.2.2 (Figure 19) [14].

![Figure 19: Definition of abstract grammar in Python 3.6](image)
3.4.2 Tree Conversion in JSON Object

According to the abstract syntax tree as shown in Figure 18, it is not easy to read and get the value in the abstract syntax tree. Hence, it would be better to rewrite this tree in other format. In this project, the tree is converted to a JSON object, because a JSON object has a clear and neat structure which consists of name-value pairs like objects, and an ordered list of values like array. Moreover, a JSON object looks similar to Python dictionary. It’s easy to retrieve the values from the object name. Last but not least, JSON is very popular in different systems and areas. That is why it is good to represent the abstract syntax tree in a JSON object.

![Figure 20: Abstract syntax tree of the code in Figure 18 in JSON format.](image)

However, an abstract syntax tree of a code can only be generated if the code has no compilation errors. Hence, its code structure cannot be analyzed when there is compilation error.
This implies that all programs without compilation error can be analyzed its structure, including those with logical error. After receiving a program which has been compiled successfully, it will be converted to an abstract syntax tree in JSON object. Then the A.I. Tutor (analyzer) will study this JSON object and start detecting skills.

**Figure 21:** Abstract syntax tree of the code in Figure 18 in tree diagram.
3.4.3 Skills Detection

The analyzer will first get a skill tree (or version tree) with all node values equal to 0, so it has not yet detected any skills correspond to each of the nodes. There are three ways to detect skills. The first one is to detect by the node type of the abstract syntax tree. For example, if the node type is ‘If’, then it is clearly that the program contains if-condition according to the definition of abstract grammar corresponding to this node type. keywords. There are more similar scenarios, such as:

- Type: For                 Corresponding Skill / Element: For-loop
- Type: While               Corresponding Skill / Element: While-loop
- Type: FunctionDef, AsyncFunctionDef Corresponding Skill / Element: Function
- Type: ClassDef            Corresponding Skill / Element: Class

Figure 22: If the node type of an abstract syntax tree is ‘List’, then a list is detected, so the skill-tree is updated.

The second way is to detect by keyword. Some skills may require a special keyword or involve a specific function. For example, for ‘basic input’, user can read a line by calling a function ‘input()’, and similar to ‘basic output’, user can output using ‘print’ function. The keywords ‘input’ and ‘print’ can be found in the abstract syntax tree under nodes with type ‘Name’ (Figures 21, 23).

Figure 23: If the node type with ‘Name’ of an abstract syntax tree is ‘input’, then the skill ‘basic input’ is detected. Similar applies to ‘basic output’ with keyword ‘print’.
The above two methods are mainly based on the node type of the abstract syntax tree. An example is shown below. There are two different programs in which both of them do not have compilation error. In Program A, fun is a function name, as well as a variable name. In the program 2a, the variable fun is returned, while in Program B, the function fun is returned.

<table>
<thead>
<tr>
<th>Program A</th>
<th>Program B</th>
</tr>
</thead>
<tbody>
<tr>
<td>def fun(n):</td>
<td>def fun(n):</td>
</tr>
<tr>
<td>fun = 1</td>
<td>fun = 1</td>
</tr>
<tr>
<td>return fun</td>
<td>return fun(n - 1)</td>
</tr>
</tbody>
</table>

Both programs have used a function. However, only Program B used recursion.

To detect ‘recursion’, their abstract syntax trees should be observed to find their difference.
From this example, solely detect recursion by function name is not enough, because a function name could be the same as the name of a variable. It is required to check whether the name is come from a function call. In Program B, there is a function call, and there is a node with ‘Call’ type in the abstract syntax tree, while this is not the case in Program A. Therefore, to detect a recursion, one possible method is to check the function name (as an attribute ‘id’ under node ‘Name’) and detect whether the node ‘Name’ is the child of the node ‘Call’.

```
47     elif tree["_type"] == "functionDef":
48         if (tree["name"] == ".init.");
49             classList[0] = True
50             functionName = tree["name"]
51             funList = [False, False]
52             if tree["returns"] != None:
53                 funList[0] = True
54                 treeAnalyser(tree["returns"], tree["_type"], "returns", ifDepth, forDepth,
55                                 for child in tree["body"]:  
56                                 treeAnalyser(child, tree["_type"], "body", ifDepth, forDepth, whileDepth,
57                                 for child in tree["decorator_list"]:
58                                     treeAnalyser(child, tree["_type"], "decorator_list", ifDepth, forDepth, wh
59                                         if funList[0] == False, False):
60                                             vtree["function"]("notRecursion","procedure",ifDepth,"valus" += 1
61                                         elif funList[0] == False, True:
62                                             vtree["function"]("recursion","procedure",ifDepth,"valus" += 1
63                                         elif funList[0] == [True, False]:
64                                             vtree["function"]("notRecursion","function",ifDepth,"valus" += 1
65                                         else:
66                                             vtree["function"]("recursion","function",ifDepth,"valus" += 1
67                             functionName = None
68
69     elif tree["_type"] == "Name":
70         if tree["id"] == "input":
71             vtree["basicIO"]("input",ifDepth,"valus" += 1
72     elif tree["id"] == "print":
73         vtree["basicIO"]("output",ifDepth,"valus" += 1
74     elif (tree["id"] == function Name and parent == "Call" and attr == "func");
75         funList[1] = True
```

**Figure 25: Recursion detection in the analyzer.**

The third and the last method to detect skill is recursively traverse every sub-trees of a syntax tree and record the depths of loops and if-conditions in every sub-tree, so the depth of loops and conditions in the whole tree can be calculated. For example, if a tree with root node type ‘For’ has two children, suppose the first child has depth of loops equal 2, while the second child has depth of loops equal to 1, then the depth of loops of the whole tree will be max(2, 1) + 1 = 3. In general, for any tree with root node type related to ‘loops’ such as ‘For’ and ‘While’, then the depth of loops of this tree will be the maximum depth of loops of all its children in the abstract syntax tree, and then plus 1.
In the implementation of analyzer.py as shown in Figure 26, ‘forDepth’, ‘whileDepth’ are integer datatype which are immutable objects, in which they are used to record the depth of for-loops and the depth of while-loops at a specific position of the abstract syntax tree. ‘loopList’ is a list with two numbers which is a mutable object, in which they are used to record the maximum depth of for-loops and the maximum depth of while-loops before exiting all the loops. It is intentionally to use declare a mutable object to store their values, such that when the child node has finished its visiting and back to the parent, data will not be lost if the updated information about the child is stored in the list.

```
elif tree["type"] == "for":
    forDepth += 1
    if vtree["maxLoopDepth"]["value"] < forDepth + whileDepth:
        vtree["maxLoopDepth"]["value"] = forDepth + whileDepth;
    if loopList[0] < forDepth:
        loopList[0] = forDepth;
    if (forDepth == 1 and whileDepth == 0):
        loopList = [1, 0]
    treeAnalyser(tree["target"], tree["_type"], "target", ifDepth, forDepth, whileDepth, loopList,
                treeAnalyser(tree["_body"], tree["_type"], "body", ifDepth, forDepth, whileDepth, loopList, func, child in tree["body")]:
        treeAnalyser(child, tree["_type"], "body", ifDepth, forDepth, whileDepth, loopList, func)
    if (forDepth == 1 and whileDepth == 0):
        updateLoop(loopList)

elif tree["_type"] == "While":
    whileDepth += 1
    if vtree["maxLoopDepth"]["value"] < forDepth + whileDepth:
        vtree["maxLoopDepth"]["value"] = forDepth + whileDepth;
    if loopList[1] < whileDepth:
        loopList[1] = whileDepth;
    if (forDepth == 0 and whileDepth == 1):
        loopList = [0, 1]
    treeAnalyser(tree["test"], tree["_type"], "test", ifDepth, forDepth, whileDepth, loopList, func,
                for child in tree["_body")]:
        treeAnalyser(child, tree["_type"], "body", ifDepth, forDepth, whileDepth, loopList, func)
    if (forDepth == 0 and whileDepth == 1):
        updateLoop(loopList)
```

def updateLoop(looplist):
    if (looplist[1] == 0):
        if looplist[0] == 1:
            vtree["loop"]["single"]["for"]["value"] += 1
        elif looplist[0] > 1:
            vtree["loop"]["nested"]["forOnly"]["value"] += 1
        elif (looplist[0] == 0):
            if looplist[1] == 1:
                vtree["loop"]["single"]["while"]["value"] += 1
            elif looplist[1] > 1:
                vtree["loop"]["nested"]["whileOnly"]["value"] += 1
            else:
                vtree["loop"]["nested"]["mixed"]["value"] += 1

Figure 26: Depth of loops detection and calculation
3.4.4 Obstacles – Lists in Python

Normally, all the skills can be detected in the above 3 methods with careful observation. However, there some challenges when the dimension of arrays (or lists in Python) is examined.

There are some simple and ideal cases of an array (a list), such as the examples as shown in Table 1. If a list consists of ‘A’, ‘B’, ‘C’ only, its dimension is 1 as it is linear. If a list consists of some lists with dimension 1, the dimension of the whole list will be 2. They can be accurately found by recursion, which has a similar method as to find the depth of loops.

<table>
<thead>
<tr>
<th>Examples of Arrays (Lists)</th>
<th>Dimensions of Arrays (Lists)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>1 (by definition)</td>
</tr>
<tr>
<td>[“A”, “B”, “C”]</td>
<td>1</td>
</tr>
<tr>
<td>[[1, 2, 3], [2, 3, 4], [3, 4, 5]]</td>
<td>2</td>
</tr>
<tr>
<td>[[[1], [2]], [[3], [4]]]</td>
<td>3</td>
</tr>
</tbody>
</table>

*Table 1: Some examples of arrays and their dimensions.*

However, there is a big obstacle in the following scenario. The type of variables declared in a code in Python can be weak-typed and dynamic. For the example as shown in Figure 27, when a new variable is created and a list of numbers is stored inside this variable, it does not adhere to a list type. This variable can be replaced by an integer or a string such that it is not a list anymore, without generating any error message.

*Figure 27: The same variable can be a list or not a list.*
To solve this issue, a dictionary datatype ‘allVar’ is declared to record the following 3 information about each of declared variables with assigned values.

- The datatype of this variable. It could be an integer, a string, a set, a tuple, a list, a dictionary, etc.
- The size of this variable. If it is not a list (non-array), the size is defined as 1. Otherwise if it is a list, it corresponds to the number of elements that this list has contained. This value will be used to calculate the largest size of all arrays later.
- The dimension of this variable. If it is not a list (non-array), the size is defined as 0. Note that it is possible that a list contains different elements with different dimensions. In this case, the dimension is calculated in the way that: the maximum dimension of all elements inside this list, and then plus 1. For instance, suppose L is a list [1, [2], [[3], 4]]. If the analyzer knows that 1 has dimension 0 as it is an integer, [2] has dimension 1 as it is linear and [[3], 4] has dimension 2. Then the whole list should have dimension 3.

Note that ‘allVar’ is a dictionary. Similar to a list in Python, it is a mutable object. Hence, after the value of ‘allVar’ is updated inside a function, the other functions can retrieve the updated values. Using ‘allVar’ information, the analyzer is able to identify the size and the dimension of a list, no matter whether this list stores another variable or another list or not.

![Figure 28: The calculation of array size and dimension in the analyzer.](image)
This detecting approach also applies similarly to detect imported module function by storing the name and ‘asname’ of the corresponding imported module.

### 3.4.5 Repeated Skills

In the design of skills tree as described in Section 3.2, note that all individual Python build-in functions (e.g. map, filter, sorted) and all Python list individual member functions (e.g. .append, .remove, .sort) are regarded as skills. However, for the sorting problem ‘TSORT’, it is found that some users submitted their solution using Python build-in function (sorted), while some other users submitted their solution using Python list member function (.sort). According to the interpretation of the analyzer, they should correspond to different skills. However, since all received solution with build-in function (sorted) is used to sort a list. They are indeed the same skill in this task.

If the design of analyzer does not change, the advisor of the system may generate a wrong advice to users who submitted a program with Python list member function (.sort) that they are recommended to use build-in function (sorted). This is not a reasonable suggestion. Hence, if there exist two skills having the same purpose, they are regarded as the same. For the case mentioned above, using Python build-in function (sorted) will be equivalent of using Python list member function (.sort). Hence, a small exceptional case is included in the analyzer as shown in Figure 30.
Figure 30: Using ‘sorted’ build-in function will regard the same as using list ‘sort’ member function.

Similarly, there are some skills removed because of repetition. If they are not removed, they will contribute heavier weight on data features, so the data clustering may not be accurate. For example:

- Using Python build-in function ‘input’ is regarded as applying ‘Basic Input’ skill.
- Using Python build-in function ‘print’ is regarded as applying ‘Basic Output’ skill.
- Using Python build-in function ‘str’ is regarded as constructing a string.

3.4.6 Testing Results

Applying the dataset in Section 3.1, the accuracy of the analyzer can be tested. Different screenshots of analyzing results against a submitted program are displayed as follows (Figures 31a, 31b and 31c).

Figure 31a: A correct submission from user ‘tt1050664’ on Codechef ‘Tsort’ problem
Figure 31b: A correct submission from user ‘srikar_ry’ on Codechef ‘Tsort’ problem

Figure 31c: A correct submission from user ‘adarshpandey’ on Codechef ‘Tsort’ problem
4. IMPROVEMENTS

There are some improvements and suggestions in some design of this project.

4.1 Skills Tree vs Data Clustering

According to the testing results of data clustering, it is found that it sometimes generates unsatisfactory results such that there are no exclusive skills using in some cluster. Some subtree in the skill tree appears to be too large (such as python build-in functions), and some subtree appears to be too small (such as lambda function). If all features in the skill tree have equal weighting, it makes the weight of small tree too small and a change of value of any node in such tree will be less significant to the data clustering result.

However, if every tree is assigned equal weight, then the change of value of one single node in a small tree will be significant to the clustering result. Oppositely, the change of value of a node inside a large tree, such as python filter build-in function, will be less significant. The effect of such weighting scheme remains negative and unfair.

An alternative way to resolve this problem is to re-design the whole skill tree, such that all trees have almost equal size. However, it is difficult to employ such design. For example, if the skill tree supports different modules functions, while a module has many functions (such as Numpy), then it is unavoidable that the corresponding tree becomes large.

Therefore, the best solution is to improve the weighting scheme by training and testing, such that the A.I. Tutor can compare the significances of different skills. For example, the results of data clustering may be much useful if the weights of ‘loops’ skills and ‘functions’ skills are higher than those of ‘python build-in functions’ skills.

4.2 Design of Analyzer

There is a clear separation between the works in the analyzer and the works in the advisor. The analyzer reads the program and generates a corresponding skill-tree with node values equal to the frequency of using the representing skills. The advisor relies on the skill-tree corresponding to the
user’s code (i.e. the output of the analyzer) to generate suggestion and advices on skills usage. Hence, if the advisor wants to generate more useful advices, the analyzer should tell more information to the advisor.

For example, if a user still gets stuck in a task after reading all advices and suggestions provided by the A.I. Tutor, then the Tutor can give more hints to the user such as providing a sample code of some cluster. However, directly providing the full solution may not be helpful to user because the user does not code a program himself/herself in proper learning. Hence, it might be better that the A.I. Tutor displays some essential part of a solution. For example, if the A.I. Tutor previously suggests the user to apply ‘loops’ on his/her code, but the user does not know where and how to place the loops, then the Tutor can display only the section which contains loops in the solution, such as the code displayed in Figure 32. According to the essential part of a program, the user might be able to capture some idea to solve the problem.

![Figure 32: The A.I. Tutor can give advices to user by displaying the essential part related to a specific skill.](image)

The above functionality can be achieved by expanding the skill tree. Not only the frequency of skill used is recorded, but also the line number(s) involved to that skill, where the line number(s) can be found in the abstract syntax tree.
5. CONCLUSION AND SUMMARY

In summary, this report described the objectives and methodology to implement the educational platform Code Nova, as well as its contribution, especially for students and teachers. With artificial intelligence, it assists users who need help directly and gives some feedbacks to students after they complete a task. Thus, students can learn programming anytime and anywhere without any extra human resources. Learning effectiveness and efficiency are hence greatly enhanced. For users, they can make use of artificial intelligence such as peer helping system and advisor to find a suitable helper for help and ask for advices and suggestions without asking a teacher respectively.

This report also includes the system design, implementation and testing results of the project. Each piece of function has been tested against dataset for many times. However, some inaccuracy might still exist between the data flow of two functions. Take the flow from the analyzer to the data clustering as an example. If there is a minor adjustment on the design of skill tree, or the calculation of detecting a specific skill becomes different, then it modifies the output of the analyzer and greatly affects the performance of data clustering because the existing parameters to a clustering algorithm might not be appropriate after the change. Hence, every decision of changing a design should be careful and precise. At this moment, all the essential features of this project have been completed to achieve the objectives as specified in the beginning. Yet, there is a room of improvement in terms of effectiveness and efficiency of each function. This project provides huge opportunity and thoughts of possibilities that whether artificial intelligence may substitute some duties of a teacher of a programming class. The future development of this project can depend on a research of a machine learning algorithm with extra training and testing to raise the accuracy of skills detection as well as the advices effectiveness to the users.
REFERENCES


[14] Python 3 Documentation Section 32.2. ast – Abstract Syntax Trees [Online]. Available: