FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

COMPUTER SCIENCE & ENGINEERING

LAB CODE: 18CSC304J

LAB NAME: Compiler Design

Faculty Incharge:
Mrs.J.Jeyasudha(101566)

Academic Year: 2021-2022

REGISTER NO: RA1911033010047

NAME: Kolla Seshu Kiran

FACULRY OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
(UNDER SECTION 3 OF THE UGC ACT, 1956)
S.R.M. NAGAR, KATTANKULATHUR – 603203
Kancheepuram District

BONAFIDE CERTIFICATE

REGISTER NO RA1911033010047

Certified to Be the Bonafide Record of the work done by Kolla Seshu Kiran of CINTEL B.Tech degree course in the practical COMPILER DESIGN in **SRM INSTITUTE OF SCIENCE & TECHNOLOGY**, Kattankulathur during the academic year 2021-2022

Lab Incharge

Date:

HEAD OF DEPARTMENT

Submitted for University Examination held in **SRM INSTITUTE OF SCIENCE & TECHNOLOGY**, at Kattankulathur.

Date **Examiner 1** **Examiner2**

INDEX

Table of Contents

Serial No.	Title	Date	Marks	Sign
1	Experiment 1 - Implementation of Lexical Analyser	10/1/22		
2	Experiment 2 - Conversion from Regular Expression to NFA	27/1/22		
3	Assignment 1 – Compiler Tool IDEONE	27/1/22		
4	Experiment 3 - Conversion from NFA to DFA	8/2/22		
5	Hackerrank Regex Module (1-20)	8/2/22		
6	Experiment 4 - Elimination of Left Recursion and Left Factoring	17/2/22		
7	Experiment 5 - Computation of FIRST and FOLLOW & Predictive Parsing Table	22/2/22		
8	Assignment 2 – Predictive Parsing	22/2/22		
9	Assignment 3 – SLR Parsing	22/2/22		
10	Experiment 6 - Implementation of Shift Reduce Parsing	4/3/22		
11	Experiment 7 - Computation of LEADING and TRAILING	17/3/22		
12	Experiment 8 - Computation of LR (0) items	24/3/22		
13	Experiment 9 - Intermediate code generation – Postfix, Prefix	31/3/22		
14	Experiment 10 - Intermediate code generation – Quadruple, Triple, Indirect triple	31/3/22		
15	Assignment 4 – Cross Compiler	31/3/22		
16	Hackerrank Regex Module (21-31)	31/3/22		
17	Experiment 11 - Implementation of DAG	7/4/22		
18	Experiment 12 - Three Address Code Generation	7/4/22		
19	Experiment 13 - Implementation of storage allocation strategies	18/4/22		
20	Assignment 5 – Global Dataflow Analysis	18/4/22		
21	Hackerrank Regex Module (32-47)	18/4/22		

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 1

Title: Implementation of Lexical Analyser

Aim: To develop lexical analysers to identify identifiers, keywords and operators in c++ programs.

Algorithm:

1. Declare the header files.
2. Open the file, and place a pointer to the first character of it.
3. Read the file until we reach ';' character and store it in a buffer.
4. Using pointers retrieve the tokens from the buffer and store it in a variable.
5. Check whether the token is a delimiter, keyword, operator or a valid identifier using the helper function (is Delimiter, is Operator… etc.).
6. If the token is an identifier and is in a declaration statement add it to the symbol table. If it is not in a statement declaration check whether the identifier is in the symbol table or not. If not, throw an error.
7. If the identifier is not valid, throw an error.
8. In case of no error print the output (type of token).
9. Repeat step 3 until you have reached the end of the file.

RA1911033010047
KOLLA SESHU KIRAN

CODE:

```
// Lexical Analyser
// RA1911033010047
#include<iostream>
#include<fstream>

using namespace std;

// Declaring language constructs
const char delimiters[] = {' ', ';', '(', ')',
                           ']', '[', '{', '}', '"', '#', ';', ',', '\''
                           , ':'};

const char operators[] = {'+', '-', '*', '^',
                          '/', '>', '<', '=', '.', '%', '&', '|', '~', ','};

const string binaryOperators[] = {"<<", "++", "--", "+=", "-=", "*=", "/=", ">>", "&&",
                           "||", "!="};

const string keywords[] = {"if", "else", "while", "do", "break", "continue",
        "int", "double", "float", "return", "char", "string", "case", "sizeof",
            "long", "short", "typedef", "switch", "unsigned", "void", "struct", "goto", "static",
            "class", "for", "continue", "private", "include", "iostream", "using", "namespace", "std", "cout"
            , "endl", "public"};


string symbolTable[1024];
int symbolIndx = 0;

bool isDelimiter(char ch) {
    for(char c: delimiters){
        if(ch == c){
            return true;
        }
    }
    return false;
}
```



```
bool isOperator(char ch){
    for(char c: operators){
        if(ch == c){
            return true;
        }
    }
    return false;
}

bool isKeyWord(string str){
    for(string key: keywords){
        if(key == str){
            return true;
        }
    }
    return false;
}

bool isNum(char c){
    int code = (int)c;
    if(!(48 <= code && code <=57)){
        return false;
    }
    return true;
}

bool isInteger(string str){
    for(char c: str){
        if(!isNum(c)){
        return false;
        }
    }
    return true;
}
```

RA1911033010047
KOLLA SESHU KIRAN

```
72
73
74  bool isReal(string str){
75      bool hasDecimal = false;
76      for(char c: str){
77          if(c == '.'){
78              hasDecimal = true;
79              continue;
80          }
81          if(!isNum(c)){
82          return false;
83          }
84      }
85      return hasDecimal;
86  }
87
88  bool isAlphabet(char c){
89      if((97 <= c && c <= 122) || (65 <= c && c <= 90)){
90          return true;
91      }
92      return false;
93  }
94
95  bool isBinaryOperator(char op1, char op2){
96      for(string s: binaryOperators){
97          if(s[0] == op1 && s[1] == op2){
98              return true;
99          }
100     }
101     return false;
102 }
103
104 void addSymbol(string s){
105     symbolTable[symbolIndx++] = s;
106 }
107
109     Identifier can contain letter, digits and underscores.
110     Names must begin only with a letter or an underscore
111     Names cannot contain whitespace or special characters
112 */
113 bool isIdentifier(string str){
114     if(!isAlphabet(str[0]) && str[0] != '_'){
115         return false;
116     }
117     for(char c: str){
118         if(!isAlphabet(c) && !isNum(c) && (c != '_')){
119             return false;
120         }
121     }
122     return true;
123 }
124
125 bool checkTable(string str){
126     for(int i=0; i<symbolIndx; i++){
127         if(str == symbolTable[i]){
128             return true;
129         }
130     }
131     return false;
132 }
133
134 void parse(string uri){
135     string buff, str;
136     ifstream stream(uri);
137     bool indentifier = false;
138     bool comments = false, multiLineComment = false;
139     bool rvalue = false;
140
141     if(!stream.is_open()){
142         throw runtime_error("Could not open the file");
143         exit(1);
144     }
```


Activate Windows
Go to Settings to activate Windows.

RA1911033010047
KOLLA SESHU KIRAN

```
148     while(getline(stream, str, ';')){
149
150         indentifier = false;
151
152         for(int i=0; i<str.length(); i++){
153
154             // We dont want to show comments, spaces and newline in output hence wer skipping it
155             if(i > 0 && str[i] == '/' && str[i-1] == '*'){
156                 multiLineComment = false;
157                 continue;
158             }
159
160             if(str[i] == ' '|| str[i] == '\n' || str[i] == '\t' || comments == true
161                 || multiLineComment == true){
162                 if(str[i] == '\n')
163                     comments = false;
164                 buff = "";
165                 continue;
166             }
167
168
169             buff = buff + str[i];
170
171             if((i == (str.length() - 1))
172                 || isDelimiter(str[i+1]) || isOperator(str[i+1])
173                 || isDelimiter(str[i]) || isOperator(str[i])){
174
175                 if(isDelimiter(buff[0])){
176
177
178                     if(buff[0] == '"'){
179                         i++;
180                         while(str[i] != '"'){
181                             if(str[i] != '\n')
182                                 cout << str[i];
183                             i++;
184                             if(i >= str.length()) throw runtime_error("missing \'\"\'");
185                         }
186                         cout << " is string" << endl;
187                         continue;
188
189                     }
190
191                     if(buff[0] == '\''){
192                         cout << str[i+1] << " is character" << endl;
193                         i+=2;
194                         cout << (bool)(str[i] != '\'');
195                         if(i < str.length() && str[i] != '\'') throw runtime_error("invalid character declaration");
196                         continue;
197
198                     }
199
200
201                 }
202                 else if(isOperator(buff[0])){
203                     /*
204                         In c++ identifier is also valid after ','. Eg: int a, b;
205                     */
206                     if(indentifier == true && buff[0] == ',') {
207                         indentifier =  true;
208                         rvalue = false;
209                     }
210
211                     if(isBinaryOperator(buff[0], str[i+1])){
212                         cout << buff[0] << str[i+1];
213                         i+=1;
214                     }
215                     // Ignoring comments
216                     else if(buff[0] == '/' && str[i+1] == '/'){
217                         comments = true;
218                         i++;
219                         continue;
```

RA1911033010047
KOLLA SESHU KIRAN

```
                } else if(buff[0] == '/' && str[i+1] == '*'){
                    multiLineComment = true;
                    i++;
                    continue;
                }
                else{
                    cout << buff[0];
                }
                cout << " is operator" << endl;
                if(buff[0] == '=')
                    rvalue = true;
            }
            else if(isKeyWord(buff)){
                cout << buff << " is keyword" << endl;
                indentifier = true;
            }
            else if(isInteger(buff)){
                cout << buff << " is integer" << endl;
            }
            else if(isReal(buff)){
                cout << buff << " is real number" << endl;
            }
            else if(isIdentifier(buff)){
                if(indentifier && !rvalue && !checkTable(buff)){
                    addSymbol(buff);
                    indentifier = false;
                } else {
                    if(checkTable(buff)){
                        indentifier = true;
                    } else{
                        throw runtime_error(buff + " identifier is not declared");
                        buff = "";
                        continue;
                    }
                }
                cout << buff << " is identifier" << endl;
            }else {
                throw runtime_error(buff + " is not valid identifier");
            }

            buff = "";
        }
    }
}

int main() {
    // Set the file location
    parse("./code.txt");

    // cout << "Symbol Table " << endl;
    // for(int i=0; i<symbolIndx; i++){
    //     cout << symbolTable[i] << endl;
    // }
    return 0;
}
```


INPUT:

```
//RA1911033010047
#include <iostream>
using namespace std;
class Student {
   public:
      int id;
      string name;
};
int main () {
    Student s1; //creating an object of Student
    s1.id = 201;
    s1.name = "Sonoo Jaiswal";
    cout << s1.id << endl;
    cout << s1.name << endl;
    return 0;
}
```

RA1911033010047
KOLLA SESHU KIRAN

OUTPUT:

Result: The program for lexical analysis is implemented successfully and obtained successfully.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT- 2

Title: Conversion of Regular Expression to Non deterministic finite automata.

Aim: To design a program that accepts a regular expression and returns the transition table for an NFA.

Algorithm:

1. Start the program
2. The user is prompted to enter a regular expression.
3. Initialize separate variables and functions for converting the regex to postfix and to display the NFA.
4. Create separate methods for different operators like +,*,.
5. By using Switch case Initialize different cases for the input
6. For the' . ' operator Initialize a separate method by using various stack functions and do the same for the other operators like ' * ' and ' + '.
7. The regular expression is in the form of a.b (or) a+b
8. 8.Display the output
9. Stop.

RA1911033010047
KOLLA SESHU KIRAN

CODE:

```cpp
//RA1911033010047
#include <iostream>
#include <stack>
#include <iomanip>

int init[20],fin[20],a=0,b=0;

std::string simplify_input(std::string s);
std::string postfix(std::string s);
int reg_nfa(std::string s,int nfa_table[100][5]);
void print_initial_final();
void print_nfa_table(int nfa_table[100][5],int states);
void initialise(int nfa_table[100][5]);
void reduce_fin(int x);


int main() {

    while(true){
        int nfa_table[100][5];
        initialise(nfa_table);
        int states{};

        std::cout << "Enter your Regular Expression:" << std::endl;
        std::cout << " -- ex: a+b, ab, a* --" << std::endl;
        std::cout << " -- Enter 'exit' to exit the program --" << std::endl;
        std::cout << ">>> ";


        std::string s{};
        std::cin >> s;

        if(s=="exit"){
            break;
        }

        s = simplify_input(s);
    //    std::cout << s << std::endl;

        s = postfix(s);
    //    std::cout << s << std::endl;

        states = reg_nfa(s, nfa_table);
        print_nfa_table(nfa_table, states);
    }


    return 0;
}

void print_nfa_table(int nfa_table[][5],int states){
    std::cout << std::endl;
    for(int i{}; i<60; i++){
        std::cout << "-";
    }
    std::cout << std::endl<< std::endl;
    std::cout << std::setw(43) << "TRANSITION TABLE FOR NFA" << std::endl << std::endl;
    std::cout << std::setw(10) << "States" << std::setw(10) << "a" << std::setw(10) << "b" << std::setw(10) << "e" << std::setw(10)
    for(int i{}; i<60; i++){
        std::cout << "-";
    }
    std::cout << std::endl;
    for(int i{1}; i<states; i++){
        for(int j{}; j<5; j++){
            if(nfa_table[i][j] == -1)
                std::cout << std::setw(10) << "~";
            else
                std::cout << std::setw(10) << nfa_table[i][j];
        }
        std::cout << std::endl;
    }
```

RA1911033010047
KOLLA SESHU KIRAN

```
    std::cout << std::endl;
    for(int i{}; i<60; i++){
        std::cout << "-";
    }
    std::cout << std::endl;

    print_initial_final();
}


void print_initial_final(){
    std::cout << std::endl;
    std::cout << "Initial state is:  ";
    for(int i{}; i<a; i++){
        std::cout << init[i] << " ";
    }
    std::cout << std::endl;
    std::cout << "Final state(s) are:  ";
    for(int i{}; i<b; i++){
        std::cout << fin[i] << " ";
    }
    std::cout << std::endl << std::endl << std::endl;
}


void reduce_fin(int x){
    for(int i=x; i<b-1; i++){
        fin[i] = fin[i+1];
    }
    b -= 1;
}


int reg_nfa(std::string s,int nfa_table[100][5]){
    int l = s.length();
    int states = 1;
    int m,n,j,counter;
```



```
    for(int i{}; i<l; i++){
        char x = s[i];
        switch(x){
            case 'a': {
                nfa_table[states][0] = states;
                init[a] = states;
                a += 1;
                states += 1;
                nfa_table[states-1][1] = states;
                fin[b] = states;
                b += 1;
                nfa_table[states][0] = states;
                states +=1;
                break;
            }
            case 'b': {
                nfa_table[states][0] = states;
                init[a] = states;
                a += 1;
                states += 1;
                nfa_table[states-1][2] = states;
                fin[b] = states;
                b += 1;
                nfa_table[states][0] = states;
                states +=1;
                break;
            }
            case '.': {
                m = fin[b-2];
                n = init[a-1];
                nfa_table[m][3]=n;
                reduce_fin(b-2);
                a -= 1;
                break;
            }
            case '+': {
                for(j=a-1,counter=0;counter<2;counter++){
```

RA1911033010047
KOLLA SESHU KIRAN

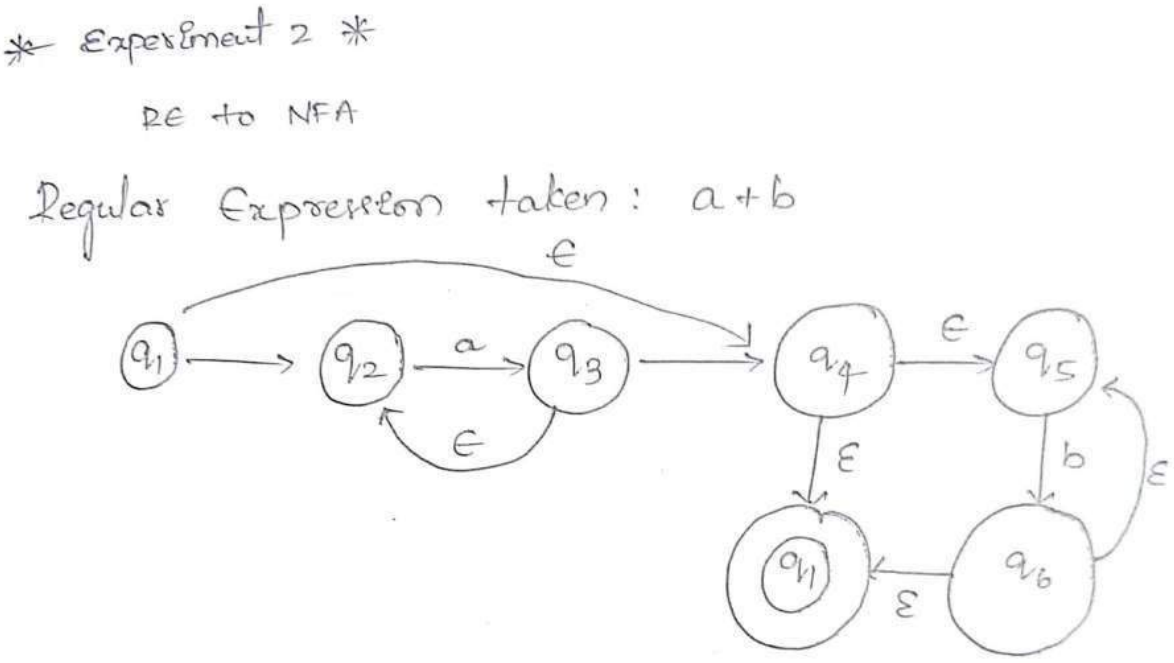
```
147                 m = init[j-counter];
148                 nfa_table[states][3+counter]=m;
149             }
150             a=a-2;
151             init[a]=states;
152             a += 1;
153             nfa_table[states][0] = states;
154             states += 1;
155             for(j=b-1,counter=0;counter<2;counter++){
156                 m = fin[j-counter];
157                 nfa_table[m][3]=states;
158             }
159             b=b-2;
160             fin[b]=states;
161             b += 1;
162             nfa_table[states][0] = states;
163             states += 1;
164             break;
165         }
166         case '*': {
167             m = init[a-1];
168             nfa_table[states][3] = m;
169             nfa_table[states][0] = states;
170             init[a-1] = states;
171
172             states += 1;
173             n = fin[b-1];
174             nfa_table[n][3]=m;
175             nfa_table[n][4]=states;
176             nfa_table[states-1][4]=states;
177             fin[b-1]=states;
178             nfa_table[states][0]=states;
179             states += 1;
180             break;
181         }
182     }
183 }
```


OUTPUT:

```
Enter your Regular Expression:
 -- ex: a+b, ab, a* --
 -- Enter 'exit' to exit the program --
>>> a+b

------------------------------------------------------------

                TRANSITION TABLE FOR NFA

   States       a       b       e       e
------------------------------------------------------------
        1       2       ~       ~       ~
        2       ~       ~       6       ~
        3       ~       4       ~       ~
        4       ~       ~       6       ~
        5       ~       ~       3       1
        6       ~       ~       ~       ~

------------------------------------------------------------

Initial state is:  5
Final state(s) are:  6
```

RA1911033010047
KOLLA SESHU KIRAN

Manual Calculation:



Result: The program to convert a regular expression to non deterministic finite automata.

RA1911033010047

Kolla. Seshu Kiran

* UNIT - I *

* Assignment - I *

* COMPILER DESIGN *

27/1/22

ONLINE COMPILER TOOL

IDEONE :-

It is a free online compiler and debugger, specifically designed for programmers and developers, where you can compile and run code with your custom input in just a matter of seconds. You can compile any programming language with ideone. It also has some awesome features like you can make your code secret to share with specific coders and is the best platform to share your code with your project members.

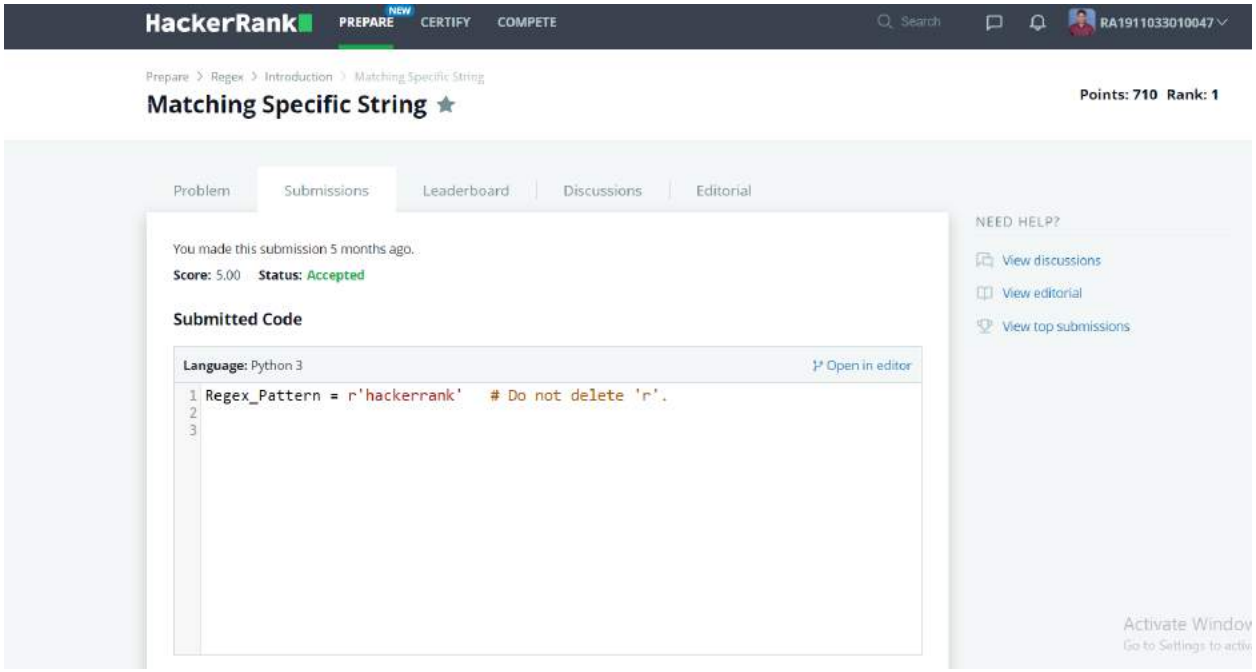
- Very easy to load and a fast platform to quick checking of your code.
- Compile in more than 60 programming languages.
- Saves your codes in an organized manner and share it via the provided link.
- A fully-featured debugging tool that makes debugging easy and fun.
- provides the functionality to make code public, private or secret.

- Embed Source code with the auto-generated javascript code.
- you can enter some notes with your code for an easy understanding of the viewers.
- Useful shortcuts for easy and fast operation
- the Ideone platform is cost free and can be used by anyone.
- It supports colludous platform.
- Ideone is an online compiler and a debugger it allows us to compile Source code and execute it online. and supports more than 60 programming languages.
- It is an online compiler
- Free compiler and debuggers.
- Supports 60 different programming languages.
- we can choose the programming language and enter the Source code and execute the program.
- options to Read Input data from Standard Input are present.
- It is an online IDE, when a compiler is integrated with IDE, we get the entire package at one place so that we can complete the code, compile, debug and execute the program in the same Software.

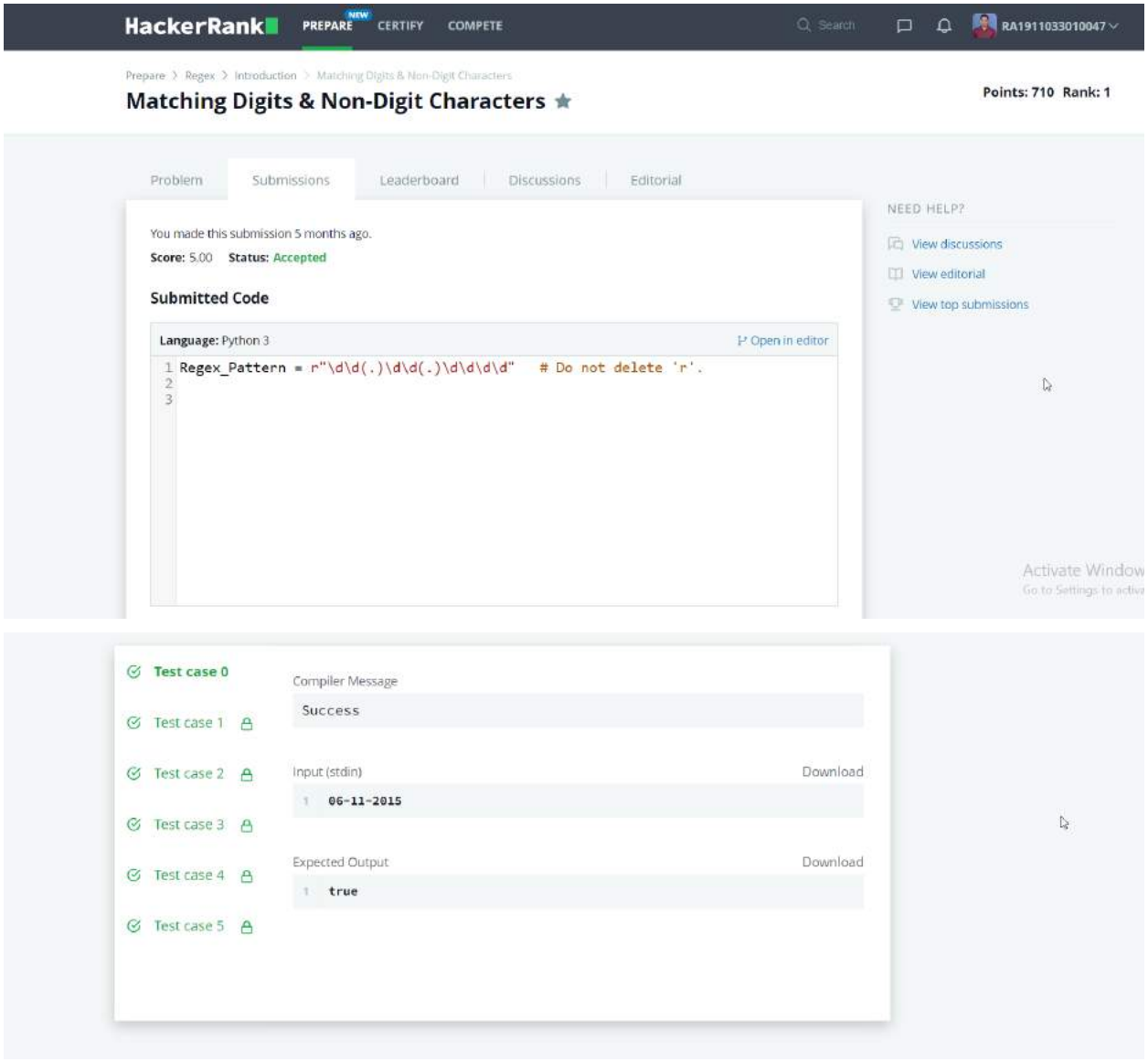
RA1911033010047
KOLLA SESHU KIRAN

HACKERRANK REGEX SOLUTIONS(1 - 20)

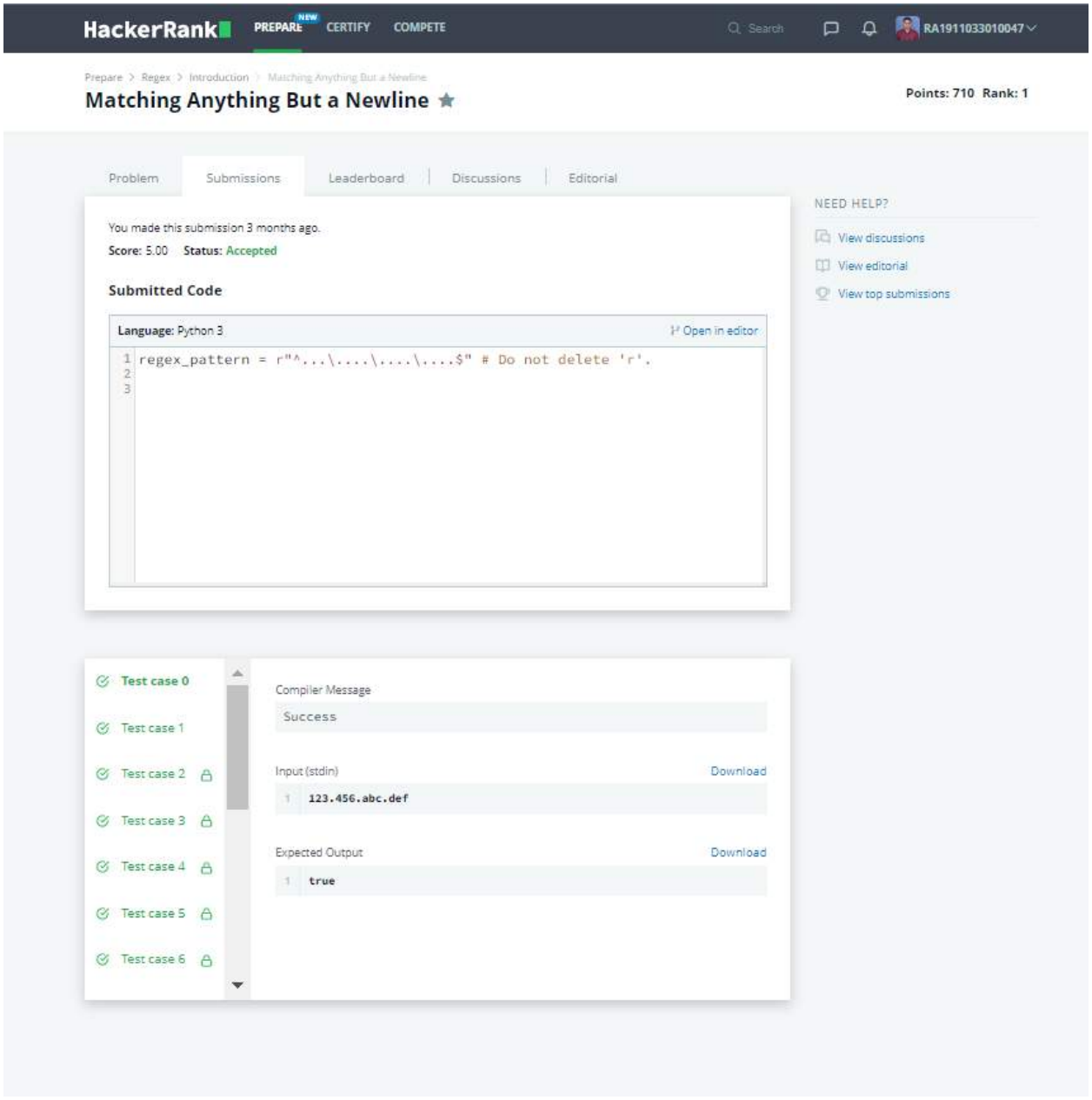
Date: 8-02-2022



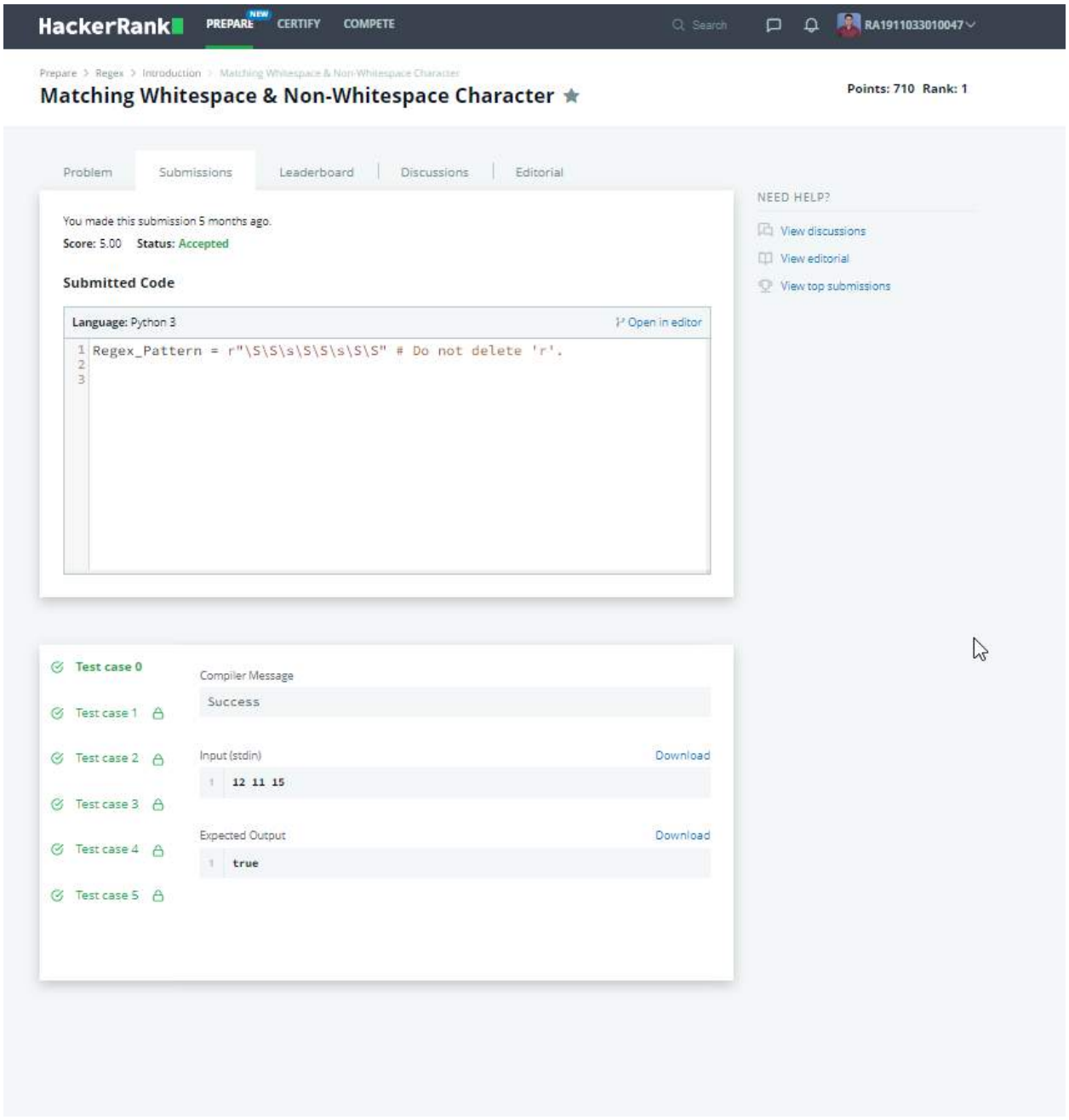
RA1911033010047
KOLLA SESHU KIRAN



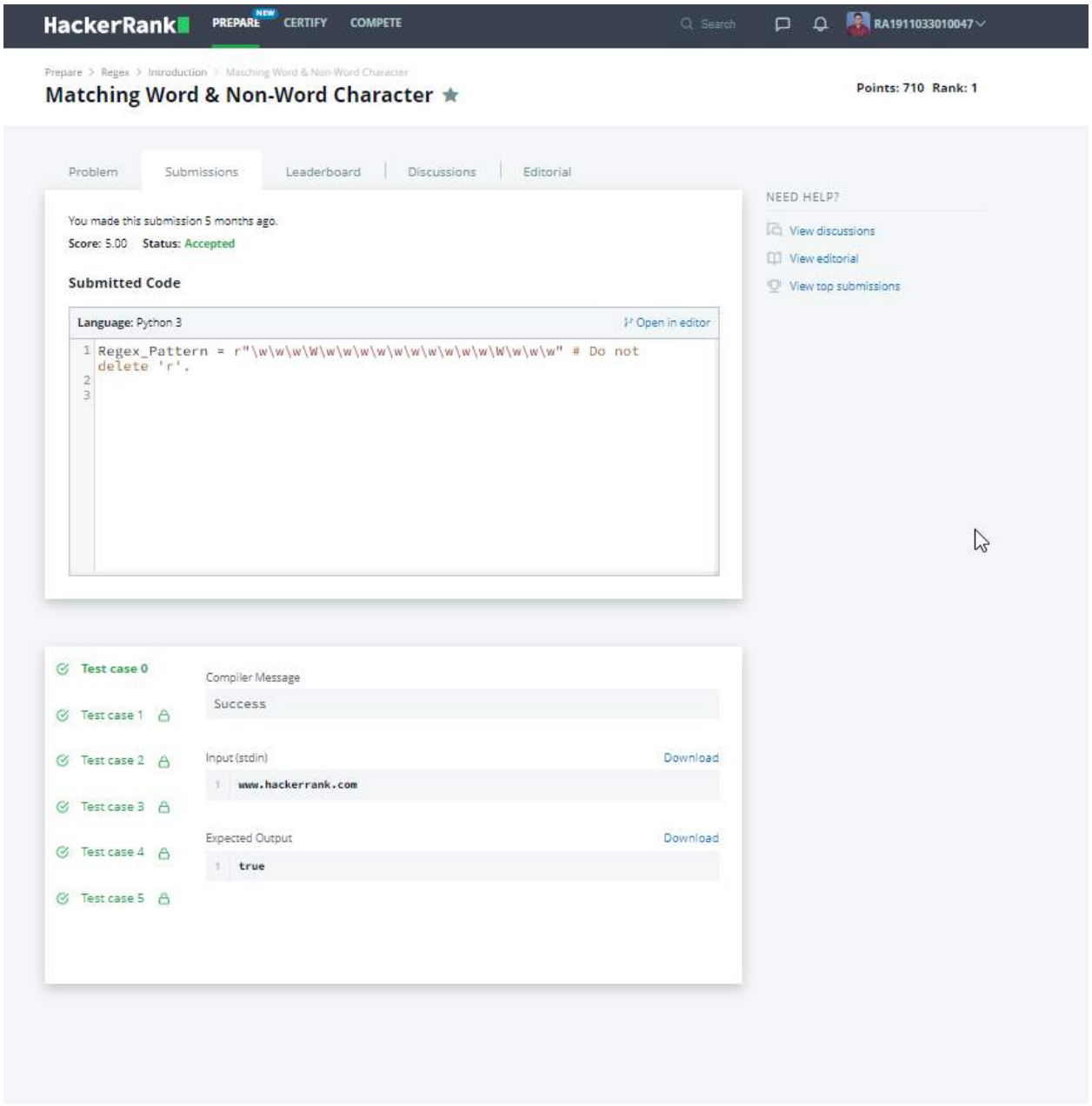
RA1911033010047
KOLLA SESHU KIRAN



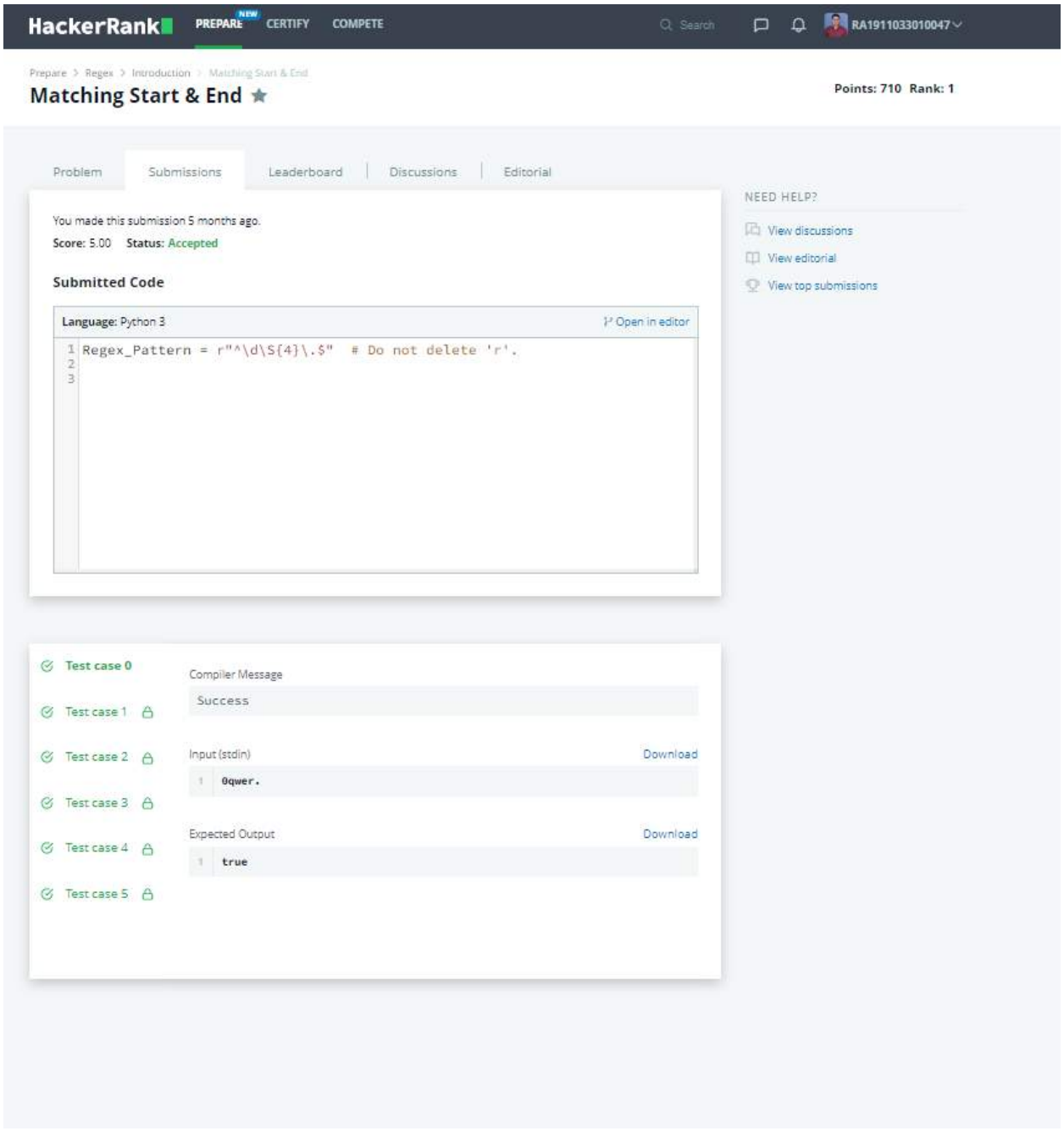
RA1911033010047
KOLLA SESHU KIRAN



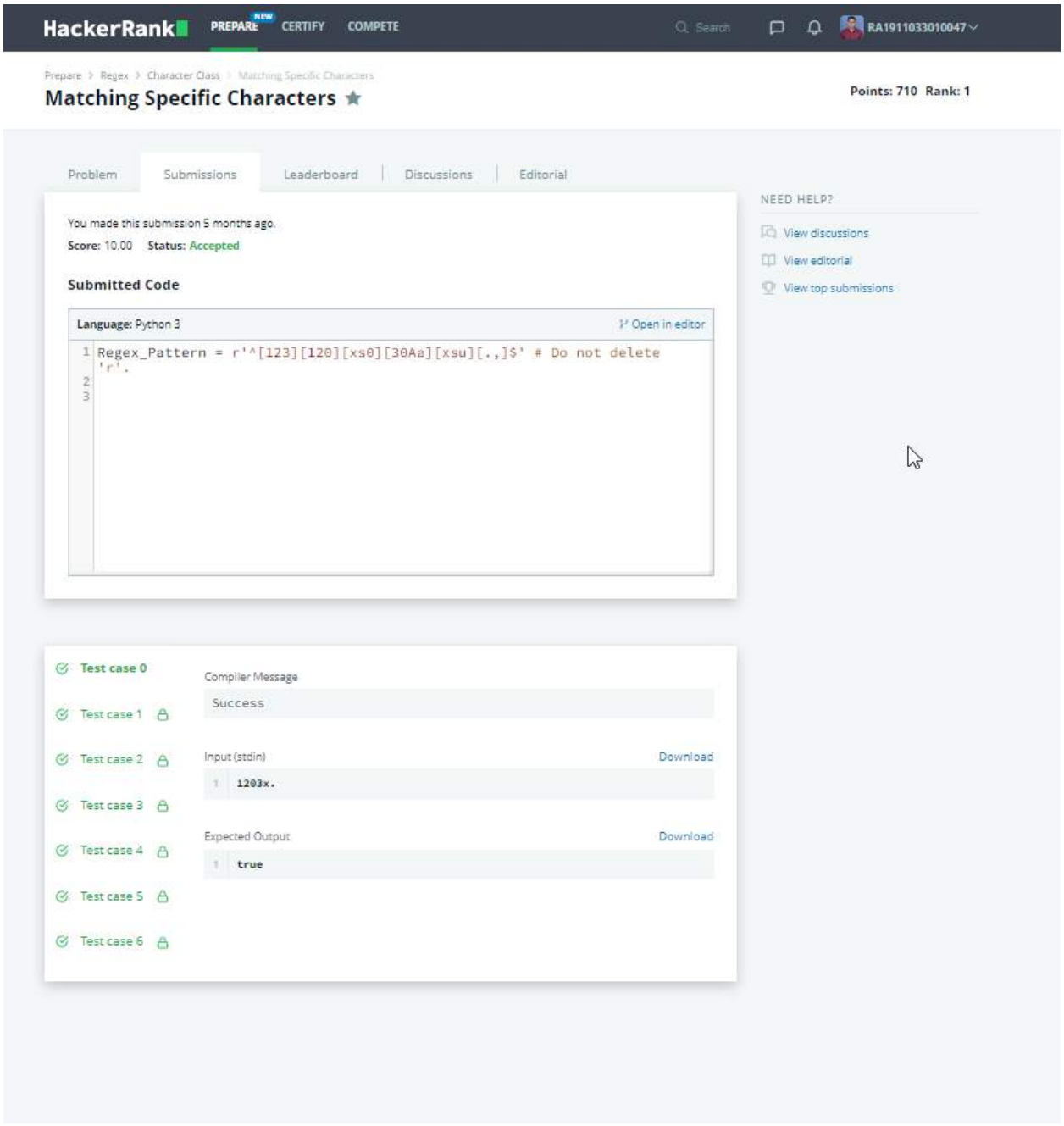
RA1911033010047
KOLLA SESHU KIRAN



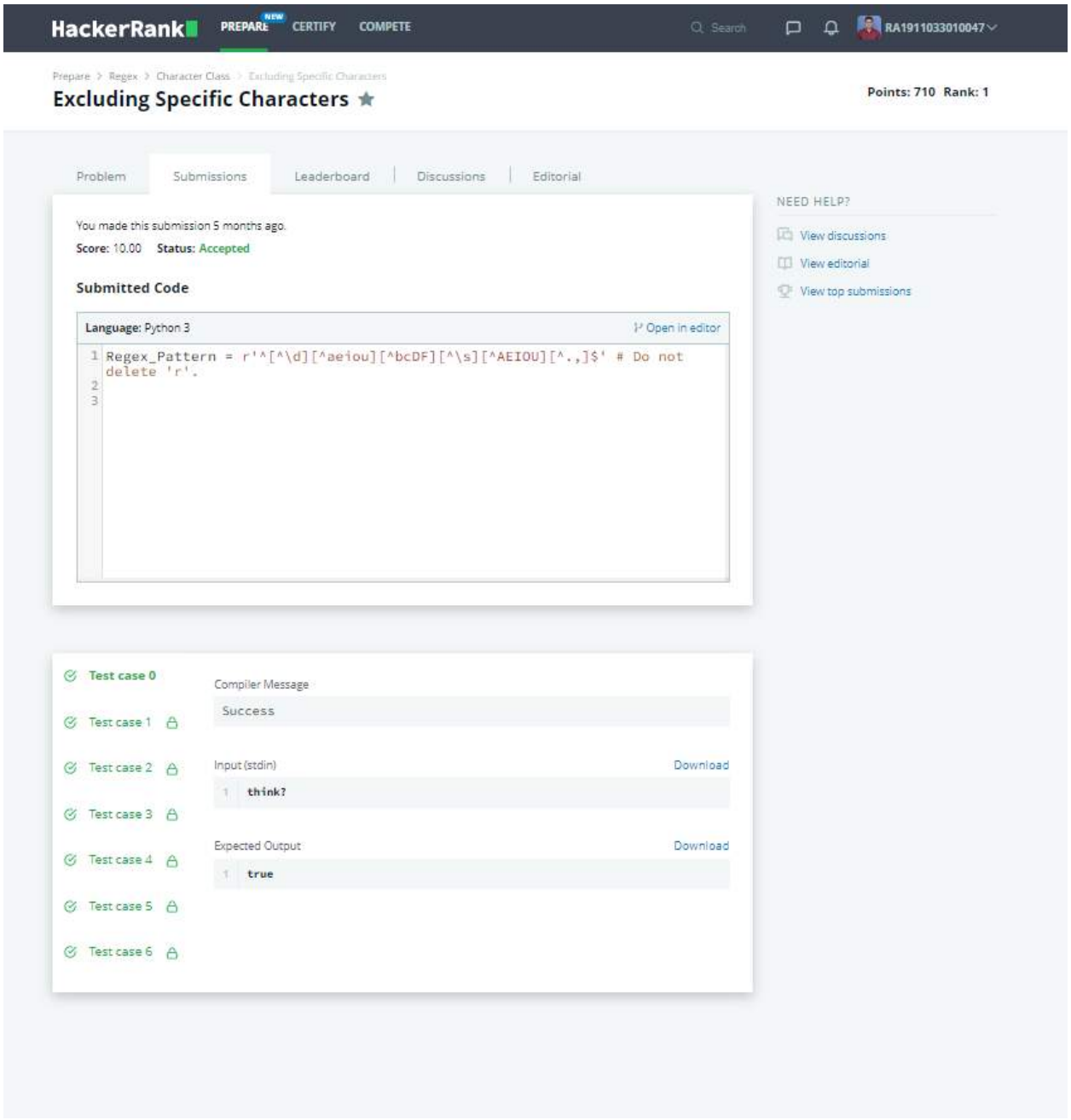
RA1911033010047
KOLLA SESHU KIRAN



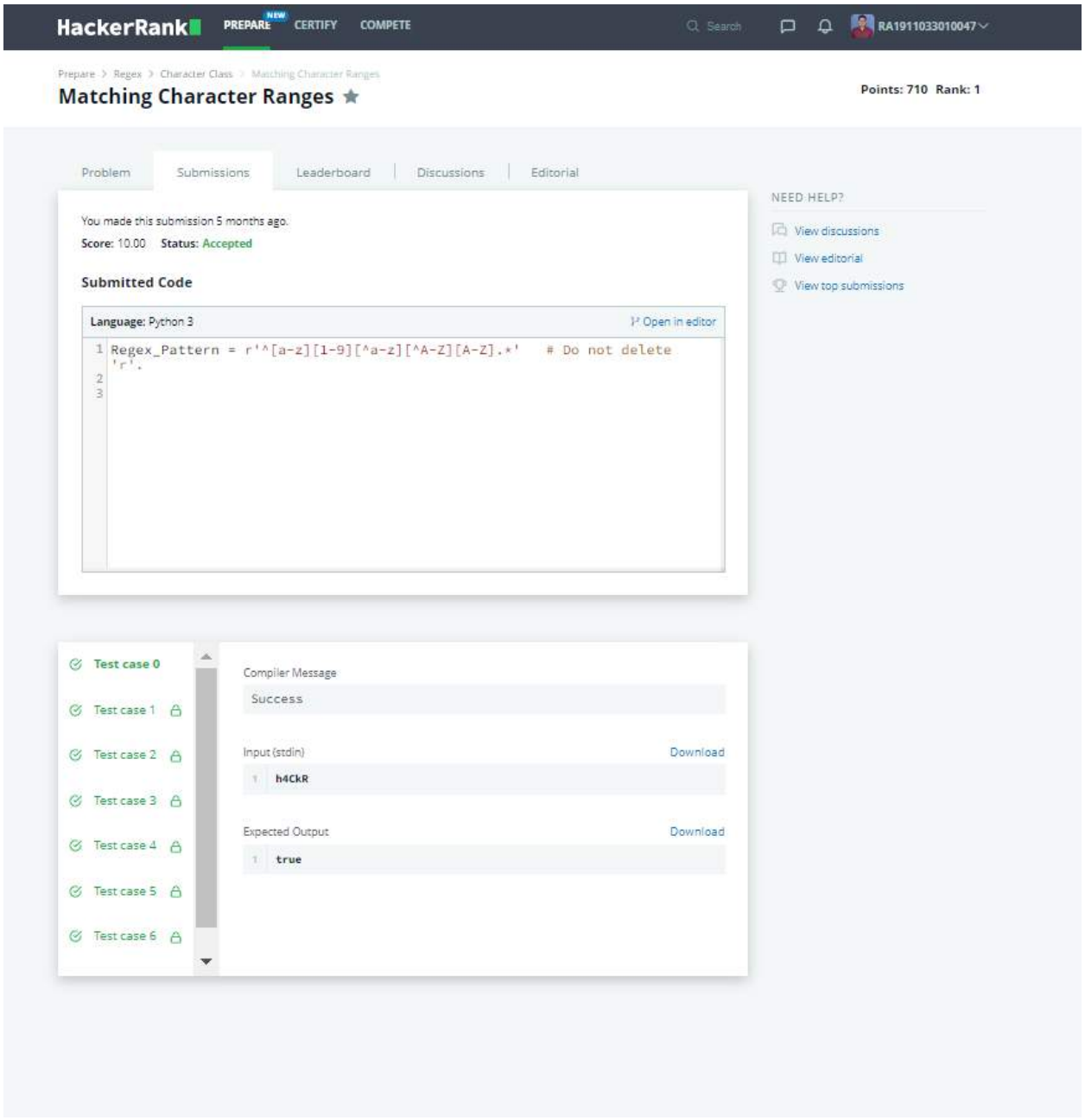
RA1911033010047
KOLLA SESHU KIRAN



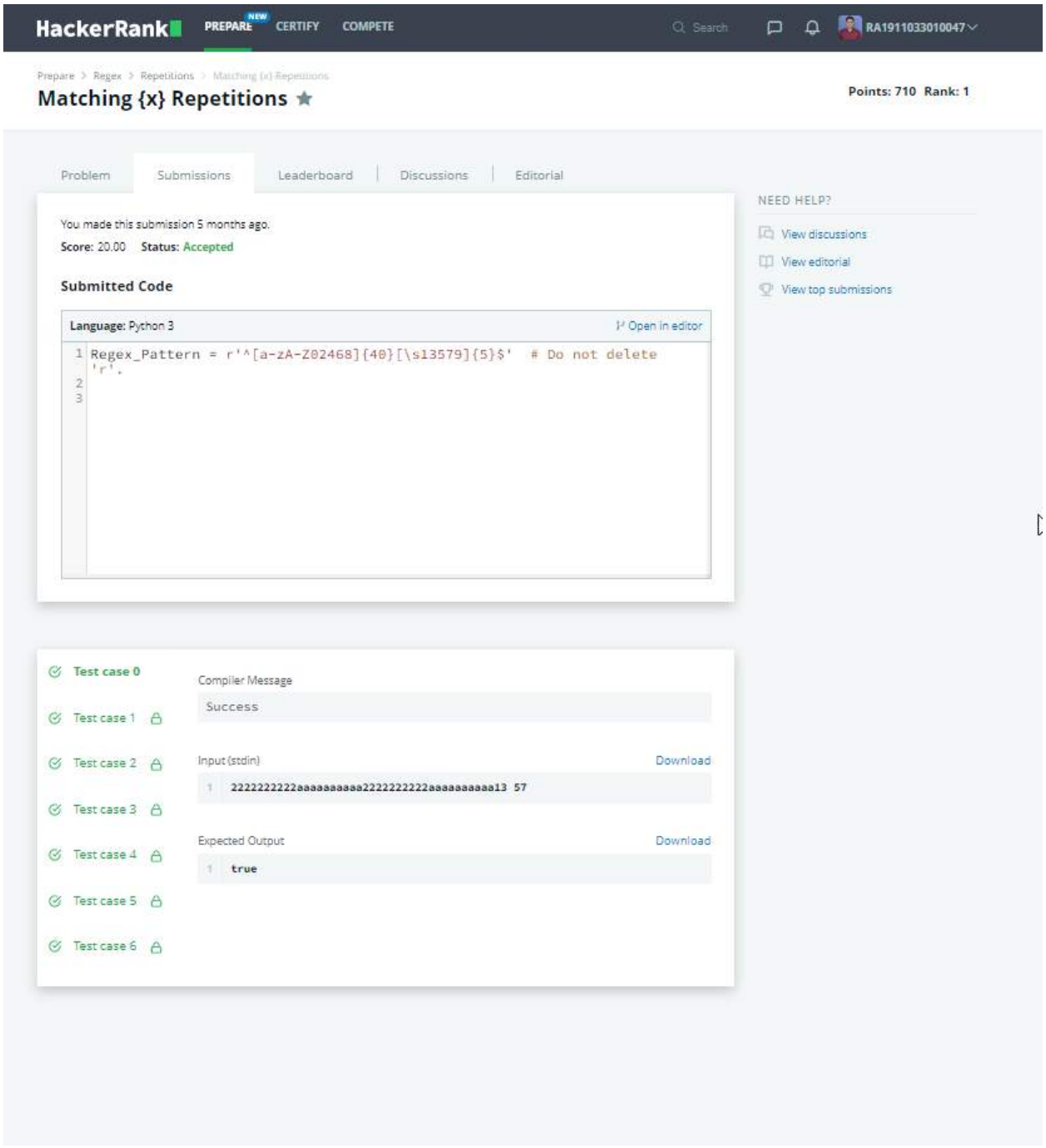
RA1911033010047
KOLLA SESHU KIRAN



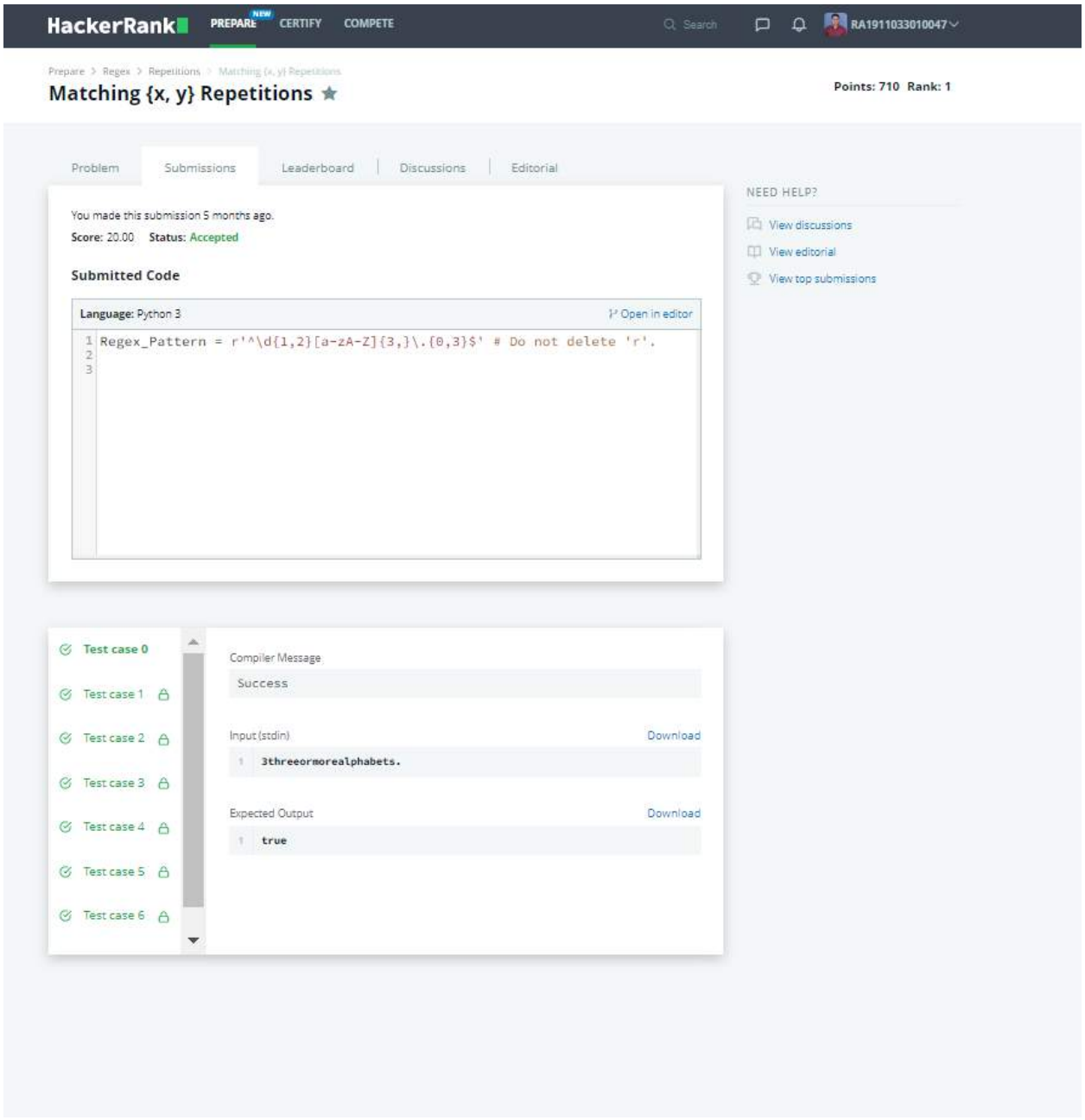
RA1911033010047
KOLLA SESHU KIRAN



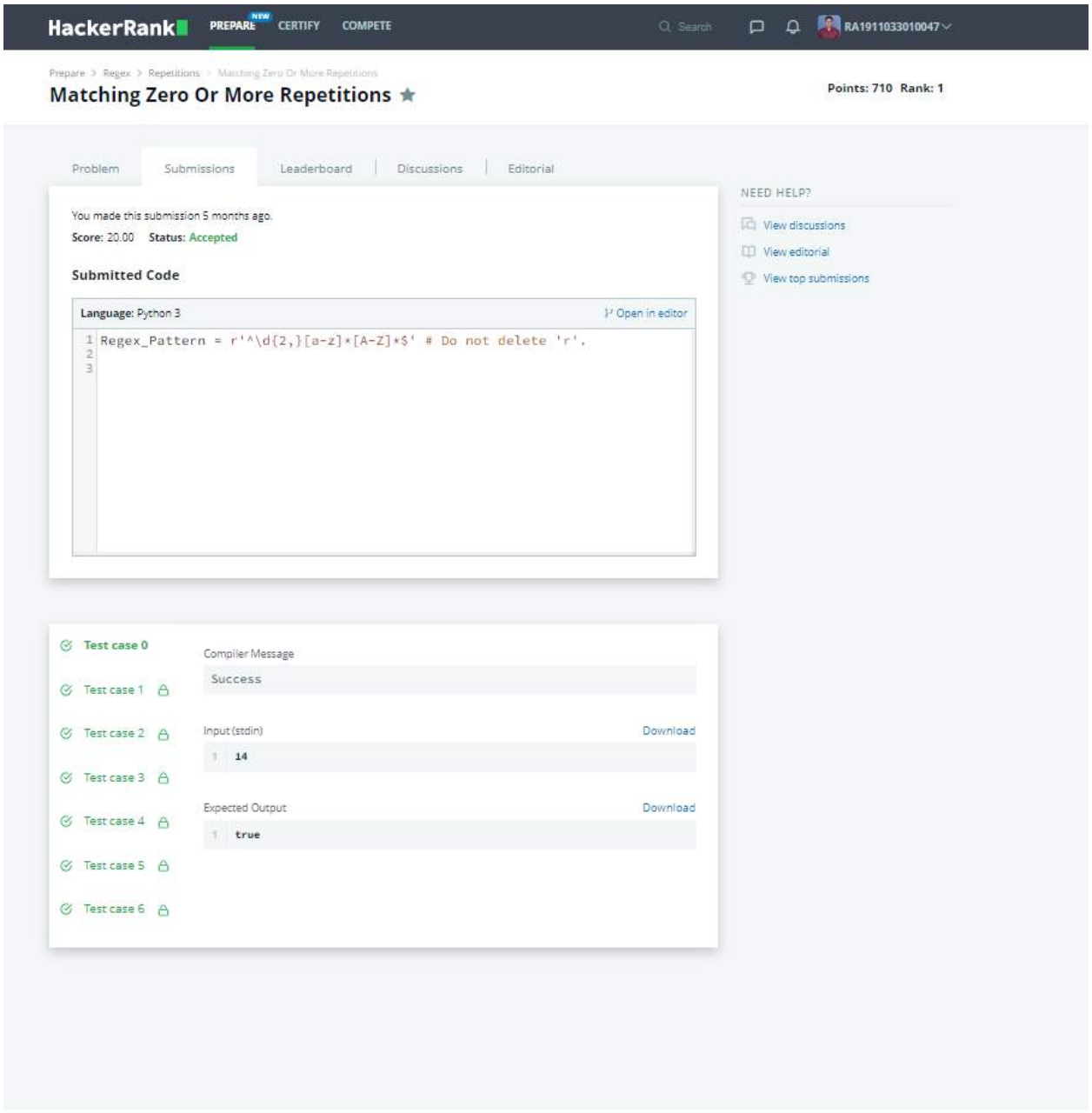
RA1911033010047
KOLLA SESHU KIRAN



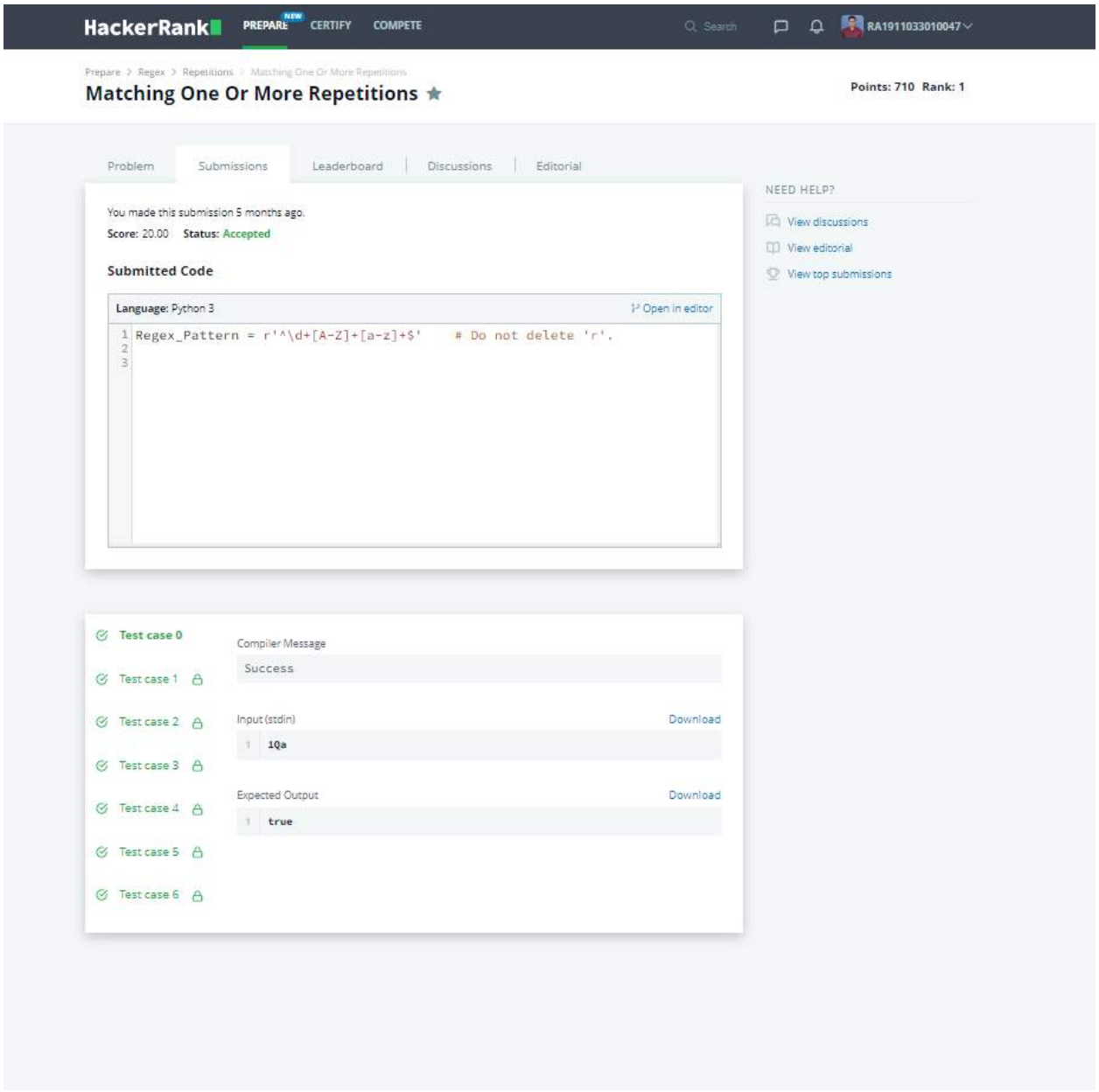
RA1911033010047
KOLLA SESHU KIRAN



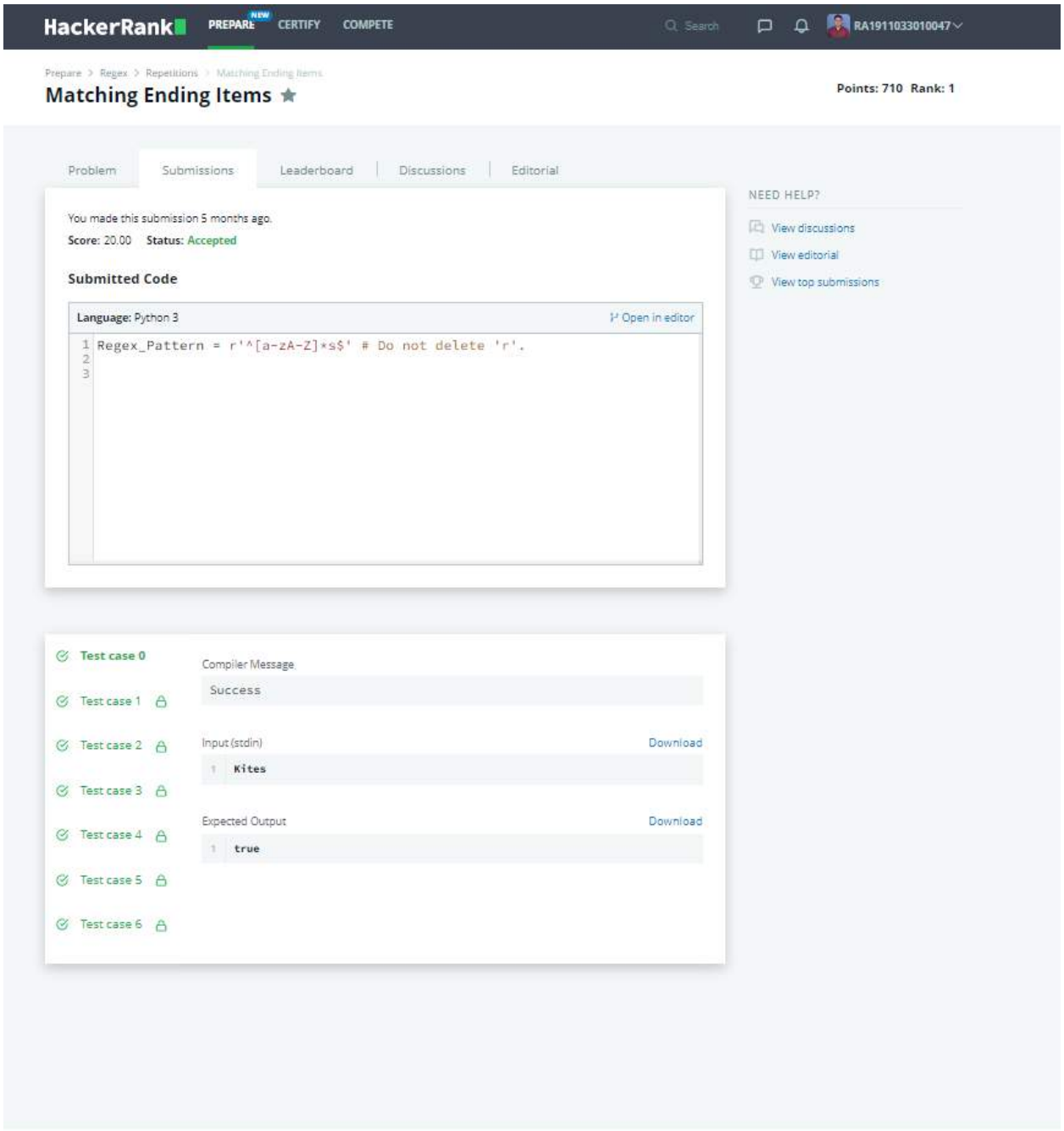
RA1911033010047
KOLLA SESHU KIRAN



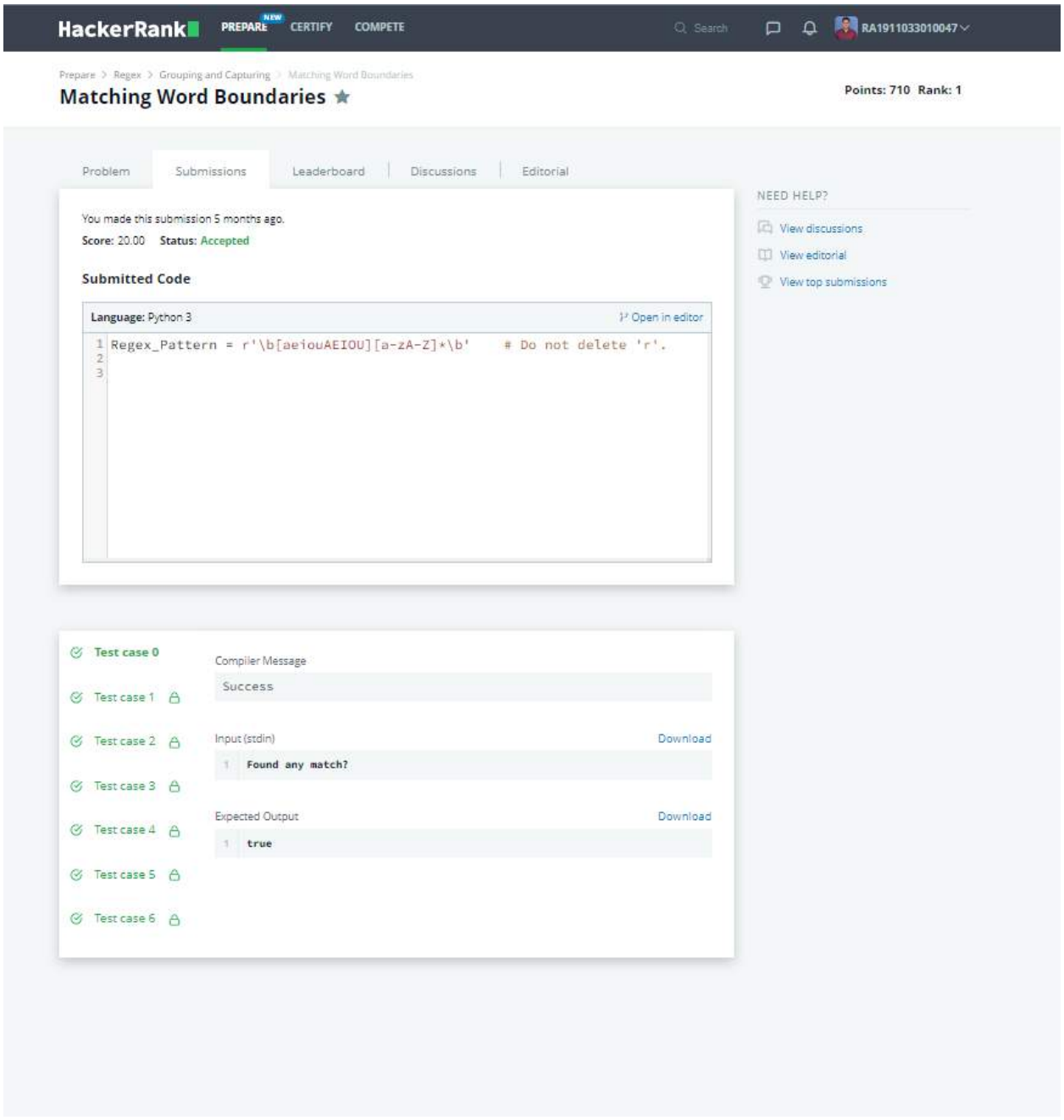
RA1911033010047
KOLLA SESHU KIRAN



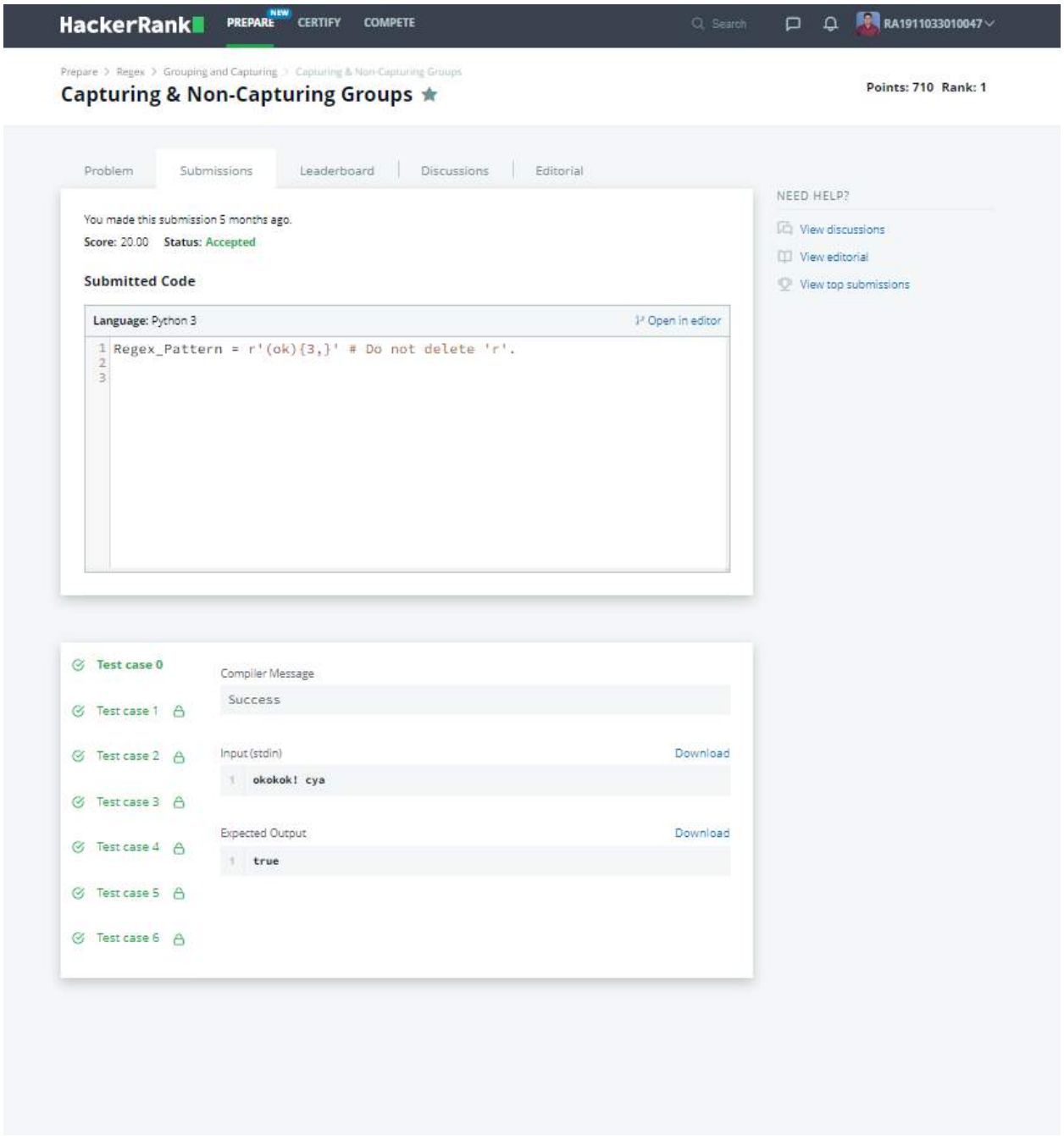
RA1911033010047
KOLLA SESHU KIRAN



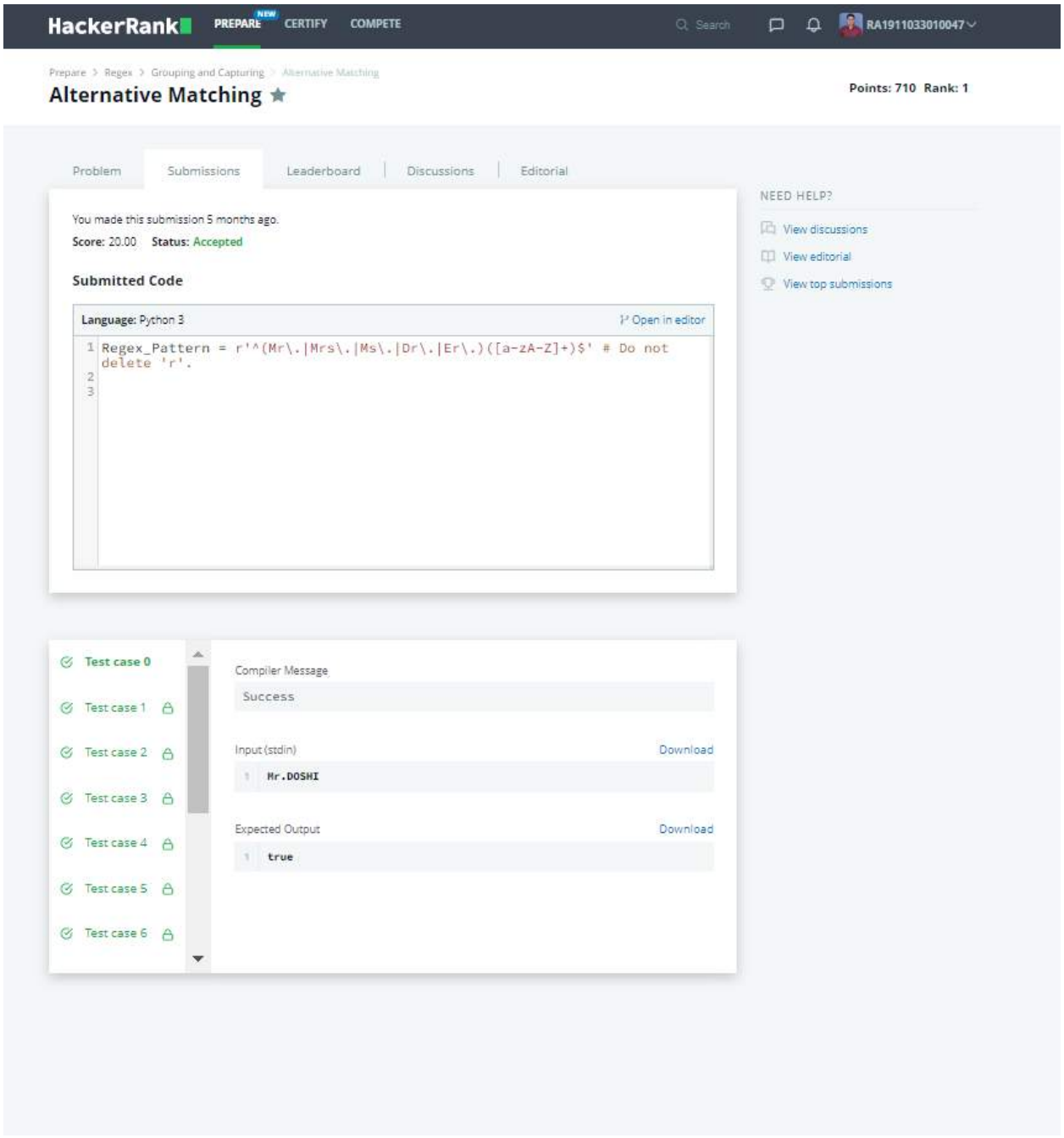
RA1911033010047
KOLLA SESHU KIRAN



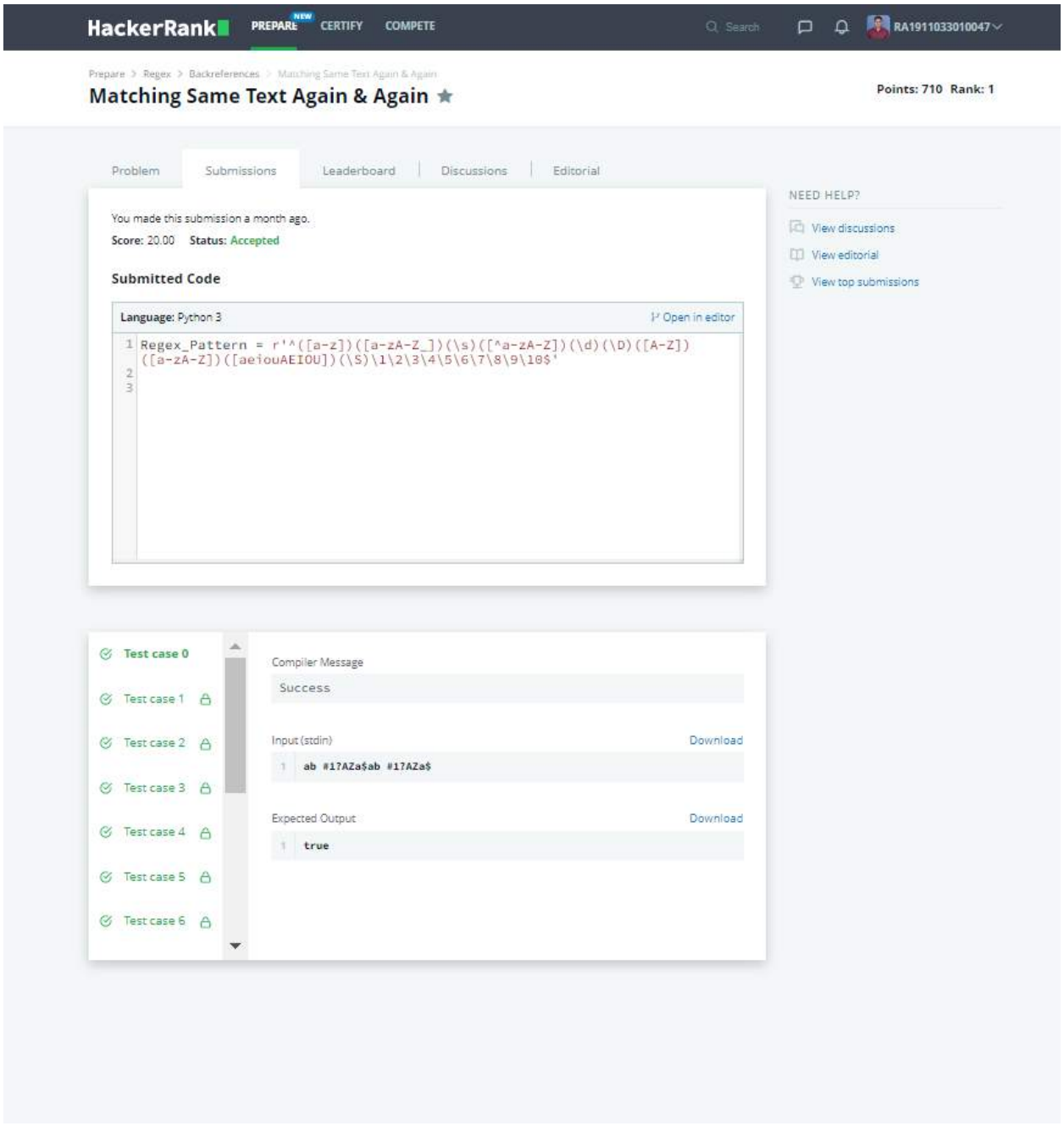
RA1911033010047
KOLLA SESHU KIRAN



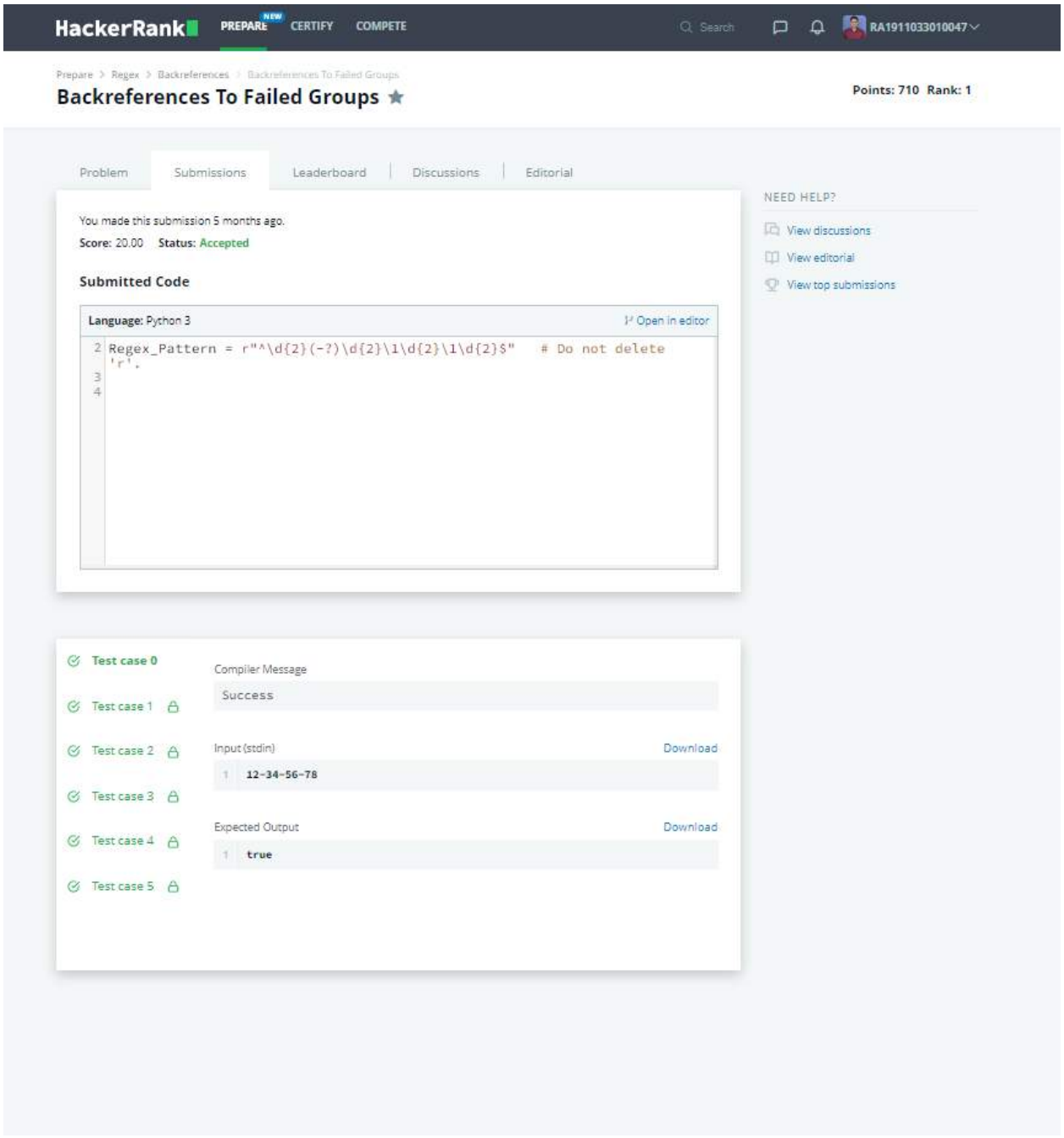
RA1911033010047
KOLLA SESHU KIRAN



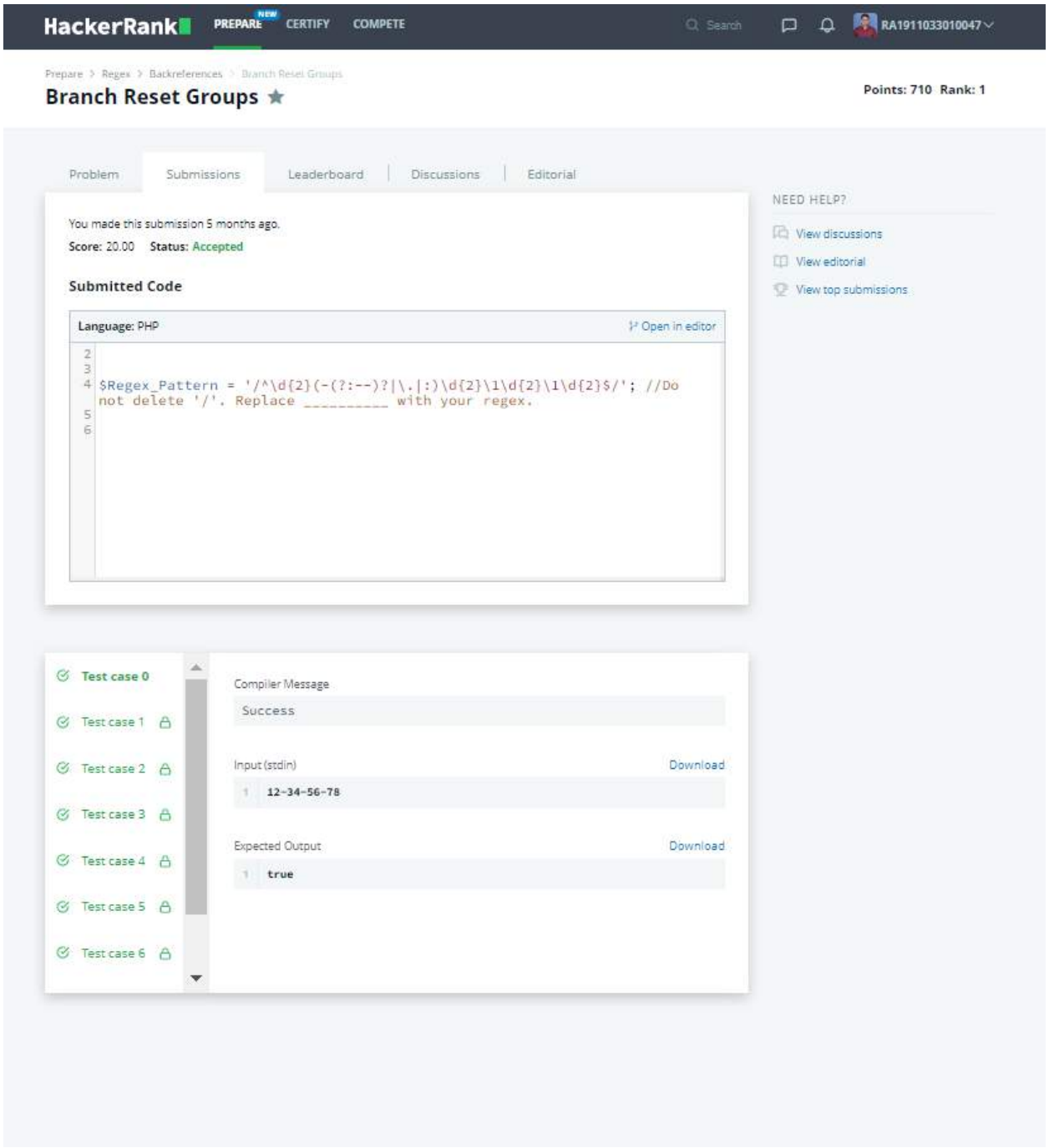
RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT- 3

Title: Conversion of Non deterministic finite automata to deterministic finite automata.

Aim: To implement a program to convert a Non-Deterministic Finite Automata to a Deterministic Finite Automata.

Algorithm:

1. Take the number of states, transition as well as the NFA table as input (state name, path, end state).
2. Store all the input in a data Frame, which is a 2-dimensional array to represent the NFA table. Display the NFA table to the user.
3. Iterate through each state along with its transitions for all input paths.
    a. If there is a transition with multiple endpoints for a particular state for a particular input, then create a new state by combining the end points and find out the transition for the new state of a particular input by union (joining) of the end points transitions, for that particular input. Repeat this for all inputs of the new state to find out the transitions. Store the new state along with its transitions in the final DFA table as well as add it to the current NFA table. Also change the transition for the current state for the current input to the new state.
    b. If transition has a single endpoint for a particular input then make no changes and add it to the final DFA table.
4. Once we have completed iterating through all the states, print out the final DFA table.

RA1911033010047
KOLLA SESHU KIRAN

CODE:

```
#RA1911033010047
import pandas as pd

nfa = {}
n = int(input("Enter no. of states : "))
t = int(input("Enter no. of transitions : "))
for i in range(n):
    state = input("Enter state name : ")
    nfa[state] = {}
    for j in range(t):
        path = input("Enter path : ")
        print("Enter end state from state {} travelling through path {} : ".format(
            state, path))
        reaching_state = [x for x in input().split()]
        nfa[state][path] = reaching_state


print("\nNFA table :- ")
nfa_table = pd.DataFrame(nfa)
print(nfa_table.transpose())

print("\nEnter final state of NFA : ")
nfa_final_state = [x for x in input().split()]

new_states_list = []

dfa = {}
keys_list = list(
    list(nfa.keys())[0])
path_list = list(nfa[keys_list[0]].keys())

dfa[keys_list[0]] = {}
for y in range(t):
    var = "".join(nfa[keys_list[0]][
        path_list[y]])
    dfa[keys_list[0]][path_list[y]] = var
    if var not in keys_list:
        new_states_list.append(var)
        keys_list.append(var)

while len(new_states_list) != 0:
    dfa[new_states_list[0]] = {}
    for _ in range(len(new_states_list[0])):
        for i in range(len(path_list)):
            temp = []
            for j in range(len(new_states_list[0])):
                temp += nfa[new_states_list[0][j]][path_list[i]]
            s = ""
            s = s.join(temp)
            if s not in keys_list:
                new_states_list.append(s)
                keys_list.append(s)
            dfa[new_states_list[0]][path_list[i]] = s

    new_states_list.remove(new_states_list[0])

print("\nDFA table :- ")
dfa_table = pd.DataFrame(dfa)
print(dfa_table.transpose())

dfa_states_list = list(dfa.keys())
dfa_final_states = []
for x in dfa_states_list:
    for i in x:
        if i in nfa_final_state:
            dfa_final_states.append(x)
            break

print("\nFinal states of the DFA are : ", dfa_final_states)
```

RA1911033010047
KOLLA SESHU KIRAN

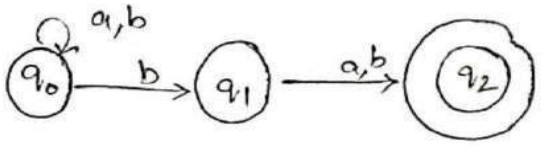
Output:

```
No. of states : 3
No. of transitions : 2
state name : A
path : 0
Enter end state from state A travelling through path 0 :
A
path : 1
Enter end state from state A travelling through path 1 :
AB
state name : B
path : 0
Enter end state from state B travelling through path 0 :
C
path : 1
Enter end state from state B travelling through path 1 :
C
state name : C
path : 0
Enter end state from state C travelling through path 0 :

path : 1
Enter end state from state C travelling through path 1 :
```

```
NFA :-

{'A': {'0': ['A'], '1': ['AB']}, 'B': {'0': ['C'], '1': ['C']}, 'C': {'0': [], '1': []}}

Printing NFA table :-
     0     1
A  [A]   [AB]
B  [C]    [C]
C   []     []
Enter final state of NFA :
C

DFA :-

{'A': {'0': 'A', '1': 'AB'}, 'AB': {'0': 'AC', '1': 'ABC'}, 'AC': {'0': 'A', '1': 'AB'}, 'ABC': {'0': 'AC', '1': 'ABC'}}

Printing DFA table :-
     0    1
A     A   AB
AB   AC  ABC
AC    A   AB
ABC  AC  ABC

Final states of the DFA are :  ['AC', 'ABC']


...Program finished with exit code 0
```

RA1911033010047
KOLLA SESHU KIRAN

NFA :



8 transition states possible.

		a	b
A	ϕ	ϕ	ϕ
B	q_0	q_0	q_0q_1
C	q_1	q_2	q_2
D	q_2^*	ϕ	ϕ
E	q_0q_1	q_0q_2	$q_0q_1q_2$
F	$q_1q_2^*$	q_2	q_2
G	$q_0q_2^*$	q_0	q_0q_1
H	$q_0q_1q_2^*$	q_0q_2	$q_0q_1q_2$

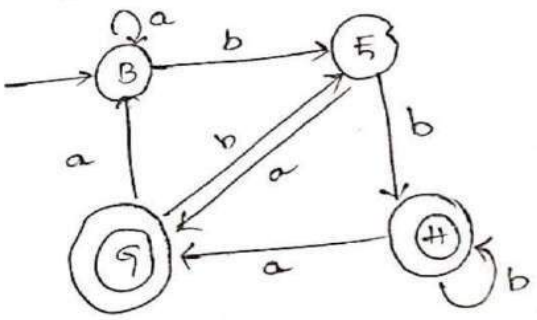
Renaming

	a	b
A	A	A
B	B	E
C	D	D
D*	A	A
E	G	H
F	D	D
G	B	E
H	G	H

To draw the DFA

	a	b
→ B	B	E
E	G	H
G	B	F
H	G	H

DFA :



Result: The program was successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT- 4

Title: Elimination of Left Recursion and Left Factoring

Aim: To write a program to eliminate the left recursion and left factoring for a given grammar.

Algorithm:

Left Recursion:

1. Start.
2. First we take the number of production rules as well the production rules as an input form the user. We also sanitize the input from spaces.
3. For a given production rule =>

a. We split the lexemes on the left and the right side of the "->" symbol. The left lexeme is stored in A variable.

b. We again split the right side lexemes on basis of '|' character and store it in a list (lexemes).

c. For each lexeme on the lexemes list we check if the A string is in the left substring of the lexeme, if so it's a left recursion case and we store the remaining portion of the lexeme in the alpha list.

d. If there is no left recursion we store it in the beta list. e. Repeat this for all the lexemes on the right side.

4. If the alpha list is empty it means there is no left recursion in the production rule, hence we print the production rule as it is.
5. If the alpha list is not empty there is left recursion and we print the production rule in the form of: A->β1A'| β2A'| . . . . .| βnA' A'->α1A'| α2A' | . . . . . |αmA

Where α1, α2….αm are the elements stored in the alpha list and β1, β2…. βn are the elements in the beta list.

6. Repeat step 2 for all the production rules.
7. Stop.

RA1911033010047
KOLLA SESHU KIRAN

CODE FOR LEFT RECURSION:

```
#RA1911033010047
gram = {
    "E":["E+T","T"],
    "T":["T*F","F"],
    "F":["(E)","i"]
}

def removeDirectLR(gramA, A):
    temp = gramA[A]
    tempCr = []
    tempInCr = []
    for i in temp:
        if i[0] == A:
            tempInCr.append(i[1:]+[A+"'"])
        else:
            tempCr.append(i+[A+"'"])
    tempInCr.append(["e"])
    gramA[A] = tempCr
    gramA[A+"'"] = tempInCr
    return gramA


def checkForIndirect(gramA, a, ai):
    if ai not in gramA:
        return False
    if a == ai:
        return True
    for i in gramA[ai]:
        if i[0] == ai:
            return False
        if i[0] in gramA:
            return checkForIndirect(gramA, a, i[0])
    return False

def rep(gramA, A):
    temp = gramA[A]
    newTemp = []
    for i in temp:
        if checkForIndirect(gramA, A, i[0]):
            t = []
            for k in gramA[i[0]]:
                t=[]
                t+=k
                t+=i[1:]
                newTemp.append(t)

        else:
            newTemp.append(i)
    gramA[A] = newTemp
    return gramA

def rem(gram):
    c = 1
    conv = {}
    gramA = {}
    revconv = {}
    for j in gram:
        conv[j] = "A"+str(c)
        gramA["A"+str(c)] = []
        c+=1

    for i in gram:
        for j in gram[i]:
            temp = []
            for k in j:
                if k in conv:
                    temp.append(conv[k])
                else:
                    temp.append(k)
            gramA[conv[i]].append(temp)

    for i in range(c-1,0,-1):
        ai = "A"+str(i)
        for j in range(0,i):
            aj = gramA[ai][0][0]
            if ai!=aj :
```

RA1911033010047
KOLLA SESHU KIRAN

```
            if ai!=aj :
                if aj in gramA and checkForIndirect(gramA,ai,aj):
                    gramA = rep(gramA, ai)

    for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
            if ai==j[0]:
                gramA = removeDirectLR(gramA, ai)
                break

    op = {}
    for i in gramA:
        a = str(i)
        for j in conv:
            a = a.replace(conv[j],j)
        revconv[i] = a

    for i in gramA:
        l = []
        for j in gramA[i]:
            k = []
            for m in j:
                if m in revconv:
                    k.append(m.replace(m,revconv[m]))
                else:
                    k.append(m)
            l.append(k)
        op[revconv[i]] = l

    return op

result = rem(gram)

for i in result:
    print(f'{i}->{result[i]}')
```


Output:

```
E->[['T', "E'"]]
T->[['F', "T'"]]
F->[['(', 'E', ')'], ['i']]
E'->[['+', 'T', "E'"], ['e']]
T'->[['*', 'F', "T'"], ['e']]


...Program finished with exit code 0
Press ENTER to exit console.
```

RA1911033010047
KOLLA SESHU KIRAN

Algorithm for Left Factoring:

1.Start.
2. First we take the production rule as an input form the user. We also sanitize the input from spaces.
3. We split the lexemes on the left and the right side of the “->” symbol.
4. We again split the right side lexemes on basis of ‘|’ character and store it in a list (lexemes).
5. For all the lexemes on the lexemes list we find out the common prefix.
6. We print the first production in the form of if there exists a common prefix:
A->αA’
where alpha(α) represents a common prefix.
7. We remove the common prefix from all the lexemes having it (stores in beta list). We make this as our new lexeme list.
8. We repeat step 5 until there is no common prefix.
9. If the beta list is not empty, we print the final production rule in the form of:
A’->β1| β2| . . . . .| βn|epsilon
where β1, β2…. βn are the elements in the beta list containing the remaining portion of the string after removing all the prefixes.

RA1911033010047
KOLLA SESHU KIRAN

CODE FOR LEFT FACTORING:

```
#RA1911033010047
from itertools import takewhile
def groupby(ls):
    d = {}
    ls = [ y[0] for y in rules ]
    initial = list(set(ls))
    for y in initial:
        for i in rules:
            if i.startswith(y):
                if y not in d:
                    d[y] = []
                d[y].append(i)
    return d

def prefix(x):
    return len(set(x)) == 1


starting=""
rules=[]
common=[]
alphabetset=["A'","B'","C'","D'","E'","F'","G'","H'","I'","J'","K'","L'","M'","N'","O'","P'","Q'","R'","S'","T'","U'","V'","W'","X'

s= "S->iEtS|iEtSeS|a"
while(True):
    rules=[]
    common=[]
    split=s.split("->")
    starting=split[0]
    for i in split[1].split("|"):
        rules.append(i)

    for k, l in groupby(rules).items():
        r = [l[0] for l in takewhile(prefix, zip(*l))]
        common.append(''.join(r))
    for i in common:
        newalphabet=alphabetset.pop()
        print(starting+"->"+i+newalphabet)
        index=[]
        for k in rules:
            if(k.startswith(i)):
                index.append(k)
        print(newalphabet+"->",end="")
        for j in index[:-1]:
            stringtoprint=j.replace(i,"", 1)+"|"
            if stringtoprint=="|":
                print("\u03B5","|",end="")
            else:
                print(j.replace(i,"", 1)+"|",end="")
        stringtoprint=index[-1].replace(i,"", 1)+"|"
        if stringtoprint=="|":
            print("\u03B5","",end="")
        else:
            print(index[-1].replace(i,"", 1)+"",end="")
        print("")
    break
```


Output:

```
S->aZ'
Z'->ε
S->iEtSY'
Y'->ε |eS


...Program finished with exit code 0
Press ENTER to exit console.
```

RA1911033010047
KOLLA SESHU KIRAN

Result: The programs for both the elimination left recursion and left factoring are successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 5

Title: First and follow

Aim: To write a program to find the first and follow of a given grammar.

Algorithm:

For computing the first:

1) If X is a terminal then

   FIRST(X) = {X}

   Example: F -> I | id

   We can write it as FIRST(F) -> { (, id)

2) If X is a non-terminal like E -> T then to get FIRSTI substitute T with other productions

   until you get a terminal as the first symbol

3) If X -> ε then add ε to FIRST(X).

For computing the follow:

1. Always check the right side of the productions for a non-terminal, whose FOLLOW set is

   being found. (never see the left side).

2. (a) If that non-terminal (S,A,B…) is followed by any terminal (a,b…,*,+,(,)…) , then add

   that terminal into the FOLLOW set.

   (b) If that non-terminal is followed by any other non-terminal then add FIRST of other nonterminals into the FOLLOW set.

RA1911033010047
KOLLA SESHU KIRAN

Code:

```
#RA1911033010047
gram = {
    "E":["E+T","T"],
    "T":["T*F","F"],
    "F":["(E)","id"]
}

def removeDirectLR(gramA, A):
    temp = gramA[A]
    tempCr = []
    tempInCr = []
    for i in temp:
        if i[0] == A:
            #tempInCr.append(i[1:])
            tempInCr.append(i[1:]+[A+"'"])
        else:
            #tempCr.append(i)
            tempCr.append(i+[A+"'"])
    tempInCr.append(["e"])
    gramA[A] = tempCr
    gramA[A+"'"] = tempInCr
    return gramA


def checkForIndirect(gramA, a, ai):
    if ai not in gramA:
        return False
    if a == ai:
        return True
    for i in gramA[ai]:
        if i[0] == ai:
            return False
        if i[0] in gramA:
            return checkForIndirect(gramA, a, i[0])
    return False

def rep(gramA, A):
    temp = gramA[A]
    newTemp = []
    for i in temp:
        if checkForIndirect(gramA, A, i[0]):
            t = []
            for k in gramA[i[0]]:
                t=[]
                t+=k
                t+=i[1:]
                newTemp.append(t)

        else:
            newTemp.append(i)
    gramA[A] = newTemp
    return gramA

def rem(gram):
    c = 1
    conv = {}
    gramA = {}
    revconv = {}
    for j in gram:
        conv[j] = "A"+str(c)
        gramA["A"+str(c)] = []
        c+=1

    for i in gram:
        for j in gram[i]:
            temp = []
            for k in j:
                if k in conv:
                    temp.append(conv[k])
                else:
                    temp.append(k)
            gramA[conv[i]].append(temp)


    for i in range(c-1,0,-1):
        ai = "A"+str(i)
```

RA1911033010047
KOLLA SESHU KIRAN

```
            if ai!=aj :
                if aj in gramA and checkForIndirect(gramA,ai,aj):
                    gramA = rep(gramA, ai)

    for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
            if ai==j[0]:
                gramA = removeDirectLR(gramA, ai)
                break

    op = {}
    for i in gramA:
        a = str(i)
        for j in conv:
            a = a.replace(conv[j],j)
        revconv[i] = a

    for i in gramA:
        l = []
        for j in gramA[i]:
            k = []
            for m in j:
                if m in revconv:
                    k.append(m.replace(m,revconv[m]))
                else:
                    k.append(m)
            l.append(k)
        op[revconv[i]] = l

    return op

result = rem(gram)


def first(gram, term):
    a = []
    if term not in gram:
        return [term]
    for i in gram[term]:
        if i[0] not in gram:
            a.append(i[0])
        elif i[0] in gram:
            a += first(gram, i[0])
    return a

firsts = {}
for i in result:
    firsts[i] = first(result,i)
    print(f'First({i}):',firsts[i])


def follow(gram, term):
    a = []
    for rule in gram:
        for i in gram[rule]:
            if term in i:
                temp = i
                indx = i.index(term)
                if indx+1!=len(i):
                    if i[-1] in firsts:
                        a+=firsts[i[-1]]
                    else:
                        a+=[i[-1]]
                else:
                    a+=["e"]
                if rule != term and "e" in a:
                    a+= follow(gram,rule)
    return a

follows = {}
for i in result:
    follows[i] = list(set(follow(result,i)))
    if "e" in follows[i]:
        follows[i].pop(follows[i].index("e"))
    follows[i]+=["$"]

    print(f'Follow({i}):',follows[i])
```


Activate Windows
Go to Settings to activate Windows.

RA1911033010047
KOLLA SESHU KIRAN

Output:

RA1911033010047
KOLLA SESHU KIRAN

Predictive Parsing

Title: Predictive Parsing Table

Aim: To implement the predictive parsing table.

Algorithm:

LL(1) Parsing:Here the 1st L represents that the scanning of the Input will be done from Left to Right manner and second L shows that in this Parsing technique we are going to use Leftmost Derivation Tree, and finally the 1 represents the number of look ahead, means how many symbols are you going to see when you want to make a decision.
Construction of LL(1) Parsing Table:To construct the Parsing table, we have two functions:

1: First(): If there is a variable, and from that variable if we try to drive all the strings then the beginning Terminal Symbol is called the first.

2: Follow(): What is the Terminal Symbol which follows a variable in the process of derivation.

Now, after computing the First and Follow set for each Non-Terminal symbol we have to construct the Parsing table. In the table Rows will contain the Non-Terminals and the column will contain the Terminal Symbols.All
The Null Productions of the Grammars will go under the Follow elements and the remaining productions will lie under the elements of the First set.

RA1911033010047
KOLLA SESHU KIRAN

CODE:

```
#RA1911033010047
gram = {
    "E":["E+T","T"],
    "T":["T*F","F"],
    "F":["(E)","i"]
}

def removeDirectLR(gramA, A):
    """gramA is dictonary"""
    temp = gramA[A]
    tempCr = []
    tempInCr = []
    for i in temp:
        if i[0] == A:
            #tempInCr.append(i[1:])
            tempInCr.append(i[1:]+[A+"'"])
        else:
            #tempCr.append(i)
            tempCr.append(i+[A+"'"])
    tempInCr.append(["e"])
    gramA[A] = tempCr
    gramA[A+"'"] = tempInCr
    return gramA


def checkForIndirect(gramA, a, ai):
    if ai not in gramA:
        return False
    if a == ai:
        return True
    for i in gramA[ai]:
        if i[0] == ai:
            return False
        if i[0] in gramA:
            return checkForIndirect(gramA, a, i[0])
    return False

def rep(gramA, A):
    temp = gramA[A]
    newTemp = []
    for i in temp:
        if checkForIndirect(gramA, A, i[0]):
            t = []
            for k in gramA[i[0]]:
                t=[]
                t+=k
                t+=i[1:]
                newTemp.append(t)

        else:
            newTemp.append(i)
    gramA[A] = newTemp
    return gramA

def rem(gram):
    c = 1
    conv = {}
    gramA = {}
    revconv = {}
    for j in gram:
        conv[j] = "A"+str(c)
        gramA["A"+str(c)] = []
        c+=1

    for i in gram:
        for j in gram[i]:
            temp = []
            for k in j:
                if k in conv:
                    temp.append(conv[k])
                else:
                    temp.append(k)
            gramA[conv[i]].append(temp)


    #print(gramA)
```

RA1911033010047
KOLLA SESHU KIRAN

```
    for i in range(c-1,0,-1):
        ai = "A"+str(i)
        for j in range(0,i):
            aj = gramA[ai][0][0]
            if ai!=aj :
                if aj in gramA and checkForIndirect(gramA,ai,aj):
                    gramA = rep(gramA, ai)

    for i in range(1,c):
        ai = "A"+str(i)
        for j in gramA[ai]:
            if ai==j[0]:
                gramA = removeDirectLR(gramA, ai)
                break

    op = {}
    for i in gramA:
        a = str(i)
        for j in conv:
            a = a.replace(conv[j],j)
        revconv[i] = a

    for i in gramA:
        l = []
        for j in gramA[i]:
            k = []
            for m in j:
                if m in revconv:
                    k.append(m.replace(m,revconv[m]))
                else:
                    k.append(m)
            l.append(k)
        op[revconv[i]] = l

    return op

result = rem(gram)
terminals = []
```



```
for i in result:
    for j in result[i]:
        for k in j:
            if k not in result:
                terminals+=[k]
terminals = list(set(terminals))
#print(terminals)

def first(gram, term):
    a = []
    if term not in gram:
        return [term]
    for i in gram[term]:
        if i[0] not in gram:
            a.append(i[0])
        elif i[0] in gram:
            a += first(gram, i[0])
    return a

firsts = {}
for i in result:
    firsts[i] = first(result,i)
#   print(f'First({i}):',firsts[i])

def follow(gram, term):
    a = []
    for rule in gram:
        for i in gram[rule]:
            if term in i:
                temp = i
                indx = i.index(term)
                if indx+1!=len(i):
                    if i[-1] in firsts:
                        a+=firsts[i[-1]]
                    else:
                        a+=[i[-1]]
                else:
                    a+=["e"]
```

RA1911033010047
KOLLA SESHU KIRAN

```
                if rule != term and "e" in a:
                    a+= follow(gram,rule)
    return a

follows = {}
for i in result:
    follows[i] = list(set(follow(result,i)))
    if "e" in follows[i]:
        follows[i].pop(follows[i].index("e"))
    follows[i]+=["$"]
#   print(f'Follow({i}):',follows[i])

resMod = {}
for i in result:
    l = []
    for j in result[i]:
        temp = ""
        for k in j:
            temp+=k
        l.append(temp)
    resMod[i] = l

# create predictive parsing table
tterm = list(terminals)
tterm.pop(tterm.index("e"))
tterm+=["$"]
pptable = {}
for i in result:
    for j in tterm:
        if j in firsts[i]:
            pptable[(i,j)]=resMod[i[0]][0]
        else:
            pptable[(i,j)]=""
    if "e" in firsts[i]:
        for j in tterm:
            if j in follows[i]:
                pptable[(i,j)]="e"
pptable[("F","i")] = "i"
```



```
toprint = f'{"": <10}'
for i in tterm:
    toprint+= f'|{i: <10}'
print(toprint)
for i in result:
    toprint = f'{i: <10}'
    for j in tterm:
        if pptable[(i,j)]!="":
            toprint+=f'|{i+"->"+pptable[(i,j)]: <10}'
        else:
            toprint+=f'|{pptable[(i,j)]: <10}'
    print(f'{"-":-<76}')
    print(toprint)
```


Output:

```
          |i         |+         |*         |)         |(         |$
--------------------------------------------------------------------------
E         |E->TE'    |          |          |          |E->TE'    |
--------------------------------------------------------------------------
T         |T->FT'    |          |          |          |T->FT'    |
--------------------------------------------------------------------------
F         |F->i      |          |          |          |F->(E)    |
--------------------------------------------------------------------------
E'        |          |E'->TE'   |          |E'->e     |          |E'->e
--------------------------------------------------------------------------
T'        |          |T'->e     |T'->FT'   |T'->e     |          |T'->e

...Program finished with exit code 0
Press ENTER to exit console.
```

Result : The program to find the first and follow function of a given grammar and the predictive parsing table is successfully implemented and output is obtained.

RA1911033010097

Kolla. Seshu Kiran

Compiler Design

* 22/2/22 *

* UNIT - II *

Assignment - 2

* PREDICTIVE PARSING TABLE *

"CSE - SE - M_2"

Q) Seetha is applying for driving license. She checks for the elegibelety on RTO office. It says the first mandatory elegibelety is age. The age (A) must be from 18 to 59. The optional elegebelety is educational qualeffication (Q). It can be undergraduate (ug), post graduate (pg), diploma (dip). Define context free grammar (CGF) for the given scenario. check whether the person who holds a post graduate degree but 61 years is elegible to get the driving license using predictive parsing.

Solution:

the context free grammar for the above given scenario is

$L \rightarrow AQ \mid A$

$Q \rightarrow ug \mid pg \mid dip$

$A \rightarrow 1z \mid yx$

$y \rightarrow 2 \mid 3 \mid 4 \mid 5$

$z \rightarrow 8 \mid 9$

$x \rightarrow 0 \mid 1 \mid y \mid 6 \mid 7 \mid z$

Step 1:-

Elimination of left factoring of production, since there is no left recursion, we will proceed with the left factoring.

$L \to AL'$

$L' \to Q \mid \varepsilon$

$Q \to uq$

$Q \to pq$

$Q \to dip$

$A \to 12$

$A \to yx$

$y \to z$

$y \to 3$

$y \to 4$

$y \to 5$

$z \to 8$

$z \to 9$

$x \to 0$

$x \to 1$

$x \to 4$

$x \to 6$

$x \to 7$

$x \to z$.

Step 2:-

We have to compute the first and follow for the grammar,

First (L) = {1, 2, 3, 4, 5}

First (L') = {ε, uq, pq, dip}

First (Q) = {uq, pq, dip}

First (A) = {1, 2, 3, 4, 5}

First (Y) = {2, 3, 4, 5}

First (z) = {8, 9}

First (X) = {0, 1, 2, ..., 9}

Follow (L) = Follow(L') = Follow(Q) = Follow(A)
= Follow(Z) = Follow(X) = Follow(Y)

= {$, 0, 1, 2, 3, 4, ···, 9}

	0	1	2	3	4	5	6	7	8	9	ug	pg	dip	$
L		L→AL'	L→AL'	L→AL'	L→AL'	L→AL'								
L'											L'→Q	L'→Q	L'→Q	L'→Q
Q											Q→ug	Q→pg	Q→dip	
A		A→1|2|4X	A→1|2|4X	A→1|2|4X	A→1|2|4X	A→1|2|4X								
Y			Y→2	Y→3	Y→4	Y→5								
Z								Z→8		Z→9				
X	X→0	X→1	X→4	X→4	X→4	X→4	X→6	X→7	X→2	X→7				

* Construction of predictive passing *

Now, we will check whether the input string will parse or not therefore.

stack	input	Action
$L	6 | p q $	error

*empty cell represents error.

→ Error because we can't (L, 6) from the above drawn predictive parsing table.

Therefore, the given input string will not parse.

* Assignment - 03 *

RA1911033010047 Date:- 22/2/22

Kolla. Seshukiran

* SLR PARSER *

Q)

Engineering - E

Technology - T

Fundamental - F

knowledge - id.

the grammar is

$E \rightarrow E+T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id.$

Step 1:- Find First and Follow for all the non-terminals.

First(E) = {(, id}

First(T) = {(, id}

First(F) = {(, id}

Follow(E) = {$, +,)}

Follow(T) = {*, $, +,)}

Follow(F) = {*, $, +,)}

Step 2:- Number all the productions.

1. $E \rightarrow E+T$
2. $E \rightarrow T$
3. $T \rightarrow T*F$
4. $T \rightarrow F$

5. $F \rightarrow (E)$

6. $F \rightarrow id$

Step 3:- Write the Augmented Grammar.

$E' \rightarrow E$

$E \rightarrow E+T$

$E \rightarrow T$

$T \rightarrow T*F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$

Step 4:- Construct the canonical collection of sets of LR(0) items.

closure of $\{[E' \rightarrow \cdot E]\}$

$E' \rightarrow \cdot E$
$E \rightarrow \cdot E+T$
$E \rightarrow \cdot T$
$T \rightarrow \cdot T*F$
$T \rightarrow \cdot F$
$F \rightarrow \cdot (E)$
$F \rightarrow \cdot id$
} I_0

goto (I_0, E) – closure $(\{[E' \rightarrow E\cdot], [E \rightarrow E\cdot+T]\})$

$E' \rightarrow E\cdot$
$E \rightarrow E\cdot+T$
} I_1

goto (I_0, T) – closure $(\{E \rightarrow T\cdot], [T \rightarrow T\cdot*F]\})$

$E \rightarrow T\cdot$
$T \rightarrow T\cdot*F$
} I_2

goto (I_0, F) – closure $(\{[T \to F\cdot]\})$

$T \to F\cdot$ $\}$ I_3

goto $(I_0, ()$ – closure $(\{[F \to (\cdot E)]\})$

$F \to (\cdot E)$
$E \to \cdot E + T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$
$\}$ I_4

goto (I_0, id) – closure $(\{F \to id\cdot]\})$

$F \to id\cdot$ $\}$ I_5

goto $(I_1, +)$ – closure $(\{[E \to E + \cdot T]\})$

$E \to E + \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$
$\}$ I_6

goto $(I_2, *)$ – closure $(\{[T \to T * \cdot F]\})$

$T \to T * \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$
$\}$ I_7

goto (I_4, E) – closure $(\{[F \to (E\cdot)], [E \to E\cdot + T]\})$

$F \to (E\cdot)$
$E \to E\cdot + T$
$\}$ I_8

goto(I_4, T) – closure({[E → T·] [T → T· * F]})

E → T·
T → T· * F
} I_2

goto(I_4, F) – closure({[T → F·]})

T → F· } I_3

goto(I_4, ( ) – closure({[F → (·E)]})

F → (·E)
E → · E+T
E → · T
T → · T * F
T → · F
F → · (E)
F → · id
} I_4

goto(I_4, id) – closure({[F → id·]})

F → id· } I_5

goto(I_6, T) – closure({E → E+T·] [T → T· * F]})

E → E+T·
T → T· * F
} I_9

goto(I_6, F) – closure({T → F·]})

T → F· } I_3

goto(I_6, ( ) – closure({[F → (·E)]})

F → (·E)
E → · E+T
E → · T
T → · T * F
} I_4

T → ·F
F → ·(E)
F → ·id
} I_4

goto(I_6, id) – closure({[F → id·]})

F → id· } I_5

goto(I_7, F) – closure({[T → T * F·]})

T → T * F· } I_{10}

goto(I_7, ( ) – closure({[F → (·E)]})

F → (·E)
E → ·E+T
E → ·T
T → ·T * F
T → ·F
F → ·(E)
F → ·id
} I_4

goto(I_7, id) – closure({[F → id·]})

F → id· } I_5

goto(I_8,)) – closure({[F → (E)·]})

F → (E)· } I_{11}

goto(I_8, +) – closure({[E → E+·T]})

E → E+·T
T → ·T * F
T → ·F
F → ·(E)
F → ·id
} I_6.

goto $(I_9, *)$ – closure $(\{[T \to T * \cdot F]\})$

$$\left.\begin{array}{l} T \to T * \cdot F \\ F \to \cdot (E) \\ F \to \cdot id \end{array}\right\} I_7$$

state	Action						GOTO		
	id	+	*	(	)	$	E	T	F
0	S_5			S_4			1	2	3
1		S_6				acc			
2		r_2	S_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	S_5			S_4			8	2	3
5		r_6	r_6		r_6	r_6			
6	S_5			S_4				9	3
7	S_5			S_4					10
8		S_6			S_{11}				
9		r_1	S_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Step1:-

Stack	Input	Action
0	id + id * id $	shift
0id5	+ id * id $	reduce F → id
0F3	+ id * id $	reduce T → F
0T2	+ id * id $	reduce E → T
0E1	+ id * id $	shift.
0E1 + 6	id * id $	shift
0E1 + 6id5	* id $	reduce F → id
0E1 + 6F3	* id $	reduce T → F
0E1 + 6T9	* id $	shift
0E1 + 6T9 * 7	id $	shift
0E1 + 6T9 * 7id5	$	reduce F → id.
0E1 + 6T9 * 7F10	$	reduce T → T * F
0E1 + 6T9	$	reduce E → E+T
0E	$	Accept.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 6

Title: Shift Reduce Parsing

Aim: To perform the shift reduce parsing

Algorithm:

Shift Reduce parser attempts for the construction of parse in a similar manner as done in bottom up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of shift reduce parser is LR parser.
This parser requires some data structures i.e.

1. An input buffer for storing the input string.
2. A stack for storing and accessing the production rules.

Basic Operations –

1.Shift: This involves moving symbols from the input buffer onto the stack.
2. Reduce: If the handle appears on top of the stack then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.
3. Accept: If only the start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accepted action is obtained, it means successful parsing is done.
4. Error: This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

RA1911033010047
KOLLA SESHU KIRAN

CODE:

```
#RA1911033010047
gram = {
    "E":["2E2","3E3","4"]
}
starting_terminal = "E"
inp = "2324232$"
}
starting_terminal = "S"
inp = "i+i*i"
"""
stack = "$"
print(f'{"Stack": <15}'+"|"+f'{"Input Buffer": <15}'+"|"+f'Parsing Action')
print(f'{"-":-<50}')

while True:
    action = True
    i = 0
    while i<len(gram[starting_terminal]):
        if gram[starting_terminal][i] in stack:
            stack = stack.replace(gram[starting_terminal][i],starting_terminal)
            print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Reduce S->{gram[starting_terminal][i]}')
            i=-1
            action = False
        i+=1
    if len(inp)>1:
        stack+=inp[0]
        inp=inp[1:]
        print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Shift')
        action = False

    if inp == "$" and stack == ("$"+starting_terminal):
        print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Accepted')
        break

    if action:
        print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Rejected')
        break
```


Output:

```
Stack          |Input Buffer   |Parsing Action
--------------------------------------------------
$2             |324232$        |Shift
$23            |24232$         |Shift
$232           |4232$          |Shift
$2324          |232$           |Shift
$232E          |232$           |Reduce S->4
$232E2         |32$            |Shift
$23E           |32$            |Reduce S->2E2
$23E3          |2$             |Shift
$2E            |2$             |Reduce S->3E3
$2E2           |$              |Shift
$E             |$              |Reduce S->2E2
$E             |$              |Accepted


...Program finished with exit code 0
```

Result: The program to implement shift reduce parser is successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 7

Title: COMPUTATION OF LEADING AND TRAILING SETS

Aim: To Compute Leading and Trailing Sets for a given grammar.

Algorithm:

1.Leading and Trailing are functions specific to generating an operator-precedence parser, which is only applicable if you have an operator precedence grammar. An operator precedence grammar is a special case of an operator grammar, and an operator grammar has the important property that *no production has two consecutive non-terminals*.
(An operator precedence grammar is, loosely speaking, an operator grammar which can be parsed with an operator precedence parser)

2.Given an operator grammar, the function Leading (resp. Trailing) of a non-terminal produces the set of terminals which could be (recursively) the first (resp. last) terminal in a production for that non-terminal.

3.Another way to think of that a terminal is in the Leading set for a non-terminal is if it is "visible" from the beginning of a production. We consider non-terminals to be "transparent", so a terminal could be visible through a non-terminal or by looking into a visible non-terminal.

RA1911033010047
KOLLA SESHU KIRAN

Code:

```
#RA1911033010047
a = ["E=E+T",
     "E=T",
     "T=T*F",
     "T=F",
     "F=(E)",
     "F=i"]

rules = {}
terms = []
for i in a:
    temp = i.split("=")
    terms.append(temp[0])
    try:
        rules[temp[0]] += [temp[1]]
    except:
        rules[temp[0]] = [temp[1]]

terms = list(set(terms))
print(rules,terms)

def leading(gram, rules, term, start):
    s = []
    if gram[0] not in terms:
        return gram[0]
    elif len(gram) == 1:
        return [0]
    elif gram[1] not in terms and gram[-1] is not start:
        for i in rules[gram[-1]]:
            s+= leading(i, rules, gram[-1], start)
            s+= [gram[1]]
        return s

def trailing(gram, rules, term, start):
    s = []
    if gram[-1] not in terms:
        return gram[-1]
    elif len(gram) == 1:
        return [0]
    elif gram[-2] not in terms and gram[-1] is not start:

        for i in rules[gram[-1]]:
            s+= trailing(i, rules, gram[-1], start)
            s+= [gram[-2]]
        return s

leads = {}
trails = {}
for i in terms:
    s = [0]
    for j in rules[i]:
        s+=leading(j,rules,i,i)
    s = set(s)
    s.remove(0)
    leads[i] = s
    s = [0]
    for j in rules[i]:
        s+=trailing(j,rules,i,i)
    s = set(s)
    s.remove(0)
    trails[i] = s

for i in terms:
    print("LEADING("+i+"):",leads[i])
for i in terms:
    print("TRAILING("+i+"):",trails[i])
```

RA1911033010047
KOLLA SESHU KIRAN

Output:

Result: The Program to compute the leading and trailing sets of a given grammar is successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 8

Title: COMPUTATION OF LR(0) ITEMS

Aim: To compute the lr(0) items for a given grammar.

Algorithm:

- The LR Parser is a Shift-reduce Parser that makes use of a Deterministic Finite Automata, recognizing the Set Of All Viable Prefixes by reading the stack from Bottom To Top.
- If a Finite-State Machine that recognizes viable prefixes of the right sentential forms is constructed, it can be used to guide the handle selection in the Shift-reduce Parser.
- Handle: Handle is a substring that matches the body of a production
- Handle is a Right Sentential Form + position where reduction can be performed + production used for reduction.

LR(0) Items

An LR(0) Item of a Grammar G is a Production of G with a Dot (.) at some position of the right side. Production A → XYZ yields the Four items:

1. A→•XYZ We hope to see a string derivable from XYZ next on the input.
2. A→X•YZ We have just seen on the input a string derivable from X and that we hope to see a string derivable from YZ next on the input.
3. A→XY•Z
4. A→XYZ•

The production A→ε generates only one item, A→•.

Each of this item is a Viable prefixes

Closure Item : An Item created by the closure operation on a state.

♪ Complete Item : An Item where the Item Dot is at the end of the

RA1911033010047
KOLLA SESHU KIRAN

RHS.

CODE:

```python
#RA1911033010047
gram = {
    "S":["CC"],
    "C":["aC","d"]
}
start = "S"
terms = ["a","d","$"]

non_terms = []
for i in gram:
    non_terms.append(i)
gram["S'"]= [start]


new_row = {}
for i in terms+non_terms:
    new_row[i]=""


non_terms += ["S'"]
# each row in state table will be dictionary {nonterms ,term,$}
stateTable = []
# I = [(terminal, closure)]
# I = [("S","A.A")]

def Closure(term, I):
    if term in non_terms:
        for i in gram[term]:
            I+=[(term,"."+i)]
    I = list(set(I))
    for i in I:
        # print("." != i[1][-1],i[1][i[1].index(".")+1])
        if "." != i[1][-1] and i[1][i[1].index(".")+1] in non_terms and i[1][i[1].index(".")+1] != term:
            I += Closure(i[1][i[1].index(".")+1], [])
    return I
```



```python
Is = []
Is+=set(Closure("S'", []))


countI = 0
omegaList = [set(Is)]
while countI<len(omegaList):
    newrow = dict(new_row)
    vars_in_I = []
    Is = omegaList[countI]
    countI+=1
    for i in Is:
        if i[1][-1]!=".":
            indx = i[1].index(".")
            vars_in_I+=[i[1][indx+1]]
    vars_in_I = list(set(vars_in_I))
    # print(vars_in_I)
    for i in vars_in_I:
        In = []
        for j in Is:
            if "."+i in j[1]:
                rep = j[1].replace("."+i,i+".")
                In+=[(j[0],rep)]
        if (In[0][1][-1]!="."):
            temp = set(Closure(i,In))
            if temp not in omegaList:
                omegaList.append(temp)
            if i in non_terms:
                newrow[i] = str(omegaList.index(temp))
            else:
                newrow[i] = "s"+str(omegaList.index(temp))
            print(f'Goto(I{countI-1},{i}):{temp} That is I{omegaList.index(temp)}')
        else:
            temp = set(In)
```

RA1911033010047
KOLLA SESHU KIRAN

```
            if temp not in omegaList:
                omegaList.append(temp)
            if i in non_terms:
                newrow[i] = str(omegaList.index(temp))
            else:
                newrow[i] = "s"+str(omegaList.index(temp))
            print(f'Goto(I{countI-1},{i}):{temp} That is I{omegaList.index(temp)}')

    stateTable.append(newrow)
print("\n\nList of I's\n")
for i in omegaList:
    print(f'I{omegaList.index(i)}: {i}')


#populate replace elements in state Table
I0 = []
for i in list(omegaList[0]):
    I0 += [i[1].replace(".","")]
print(I0)

for i in omegaList:
    for j in i:
        if "." in j[1][-1]:
            if j[1][-2]=="S":
                stateTable[omegaList.index(i)]["$"] = "Accept"
                break
            for k in terms:
                stateTable[omegaList.index(i)][k] = "r"+str(I0.index(j[1].replace(".","")))
print("\nStateTable")

print(f'{" ": <9}',end="")
for i in new_row:
    print(f'|{i: <11}',end="")

print(f'\n{"-":-<66}')
```



```
for i in stateTable:
    print(f'{"I("+str(stateTable.index(i))+")": <9}',end="")
    for j in i:
        print(f'|{i[j]: <10}',end=" ")
    print()
```


Output:

```
Goto(I0,d):{('C', 'd.')} That is I1
Goto(I0,C):{('C', '.aC'), ('S', 'C.C'), ('C', '.d')} That is I2
Goto(I0,S):{("S'", 'S.')} That is I3
Goto(I0,a):{('C', '.aC'), ('C', 'a.C'), ('C', '.d')} That is I4
Goto(I2,d):{('C', 'd.')} That is I1
Goto(I2,C):{('S', 'CC.')} That is I5
Goto(I2,a):{('C', '.aC'), ('C', 'a.C'), ('C', '.d')} That is I4
Goto(I4,d):{('C', 'd.')} That is I1
Goto(I4,C):{('C', 'aC.')} That is I6
Goto(I4,a):{('C', '.aC'), ('C', 'a.C'), ('C', '.d')} That is I4


List of I's

I0: {('C', '.aC'), ("S'", '.S'), ('S', '.CC'), ('C', '.d')}
I1: {('C', 'd.')}
I2: {('C', '.aC'), ('S', 'C.C'), ('C', '.d')}
I3: {("S'", 'S.')}
I4: {('C', '.aC'), ('C', 'a.C'), ('C', '.d')}
I5: {('S', 'CC.')}
I6: {('C', 'aC.')}
['aC', 'S', 'CC', 'd']

StateTable
         |a          |d          |$          |S          |C
----------------------------------------------------------------
I(0)     |s4         |s1         |           |3          |2
I(1)     |r3         |r3         |r3         |           |
I(2)     |s4         |s1         |           |           |5
I(3)     |           |           |Accept     |           |
I(4)     |s4         |s1         |           |           |6
I(5)     |r2         |r2         |r2         |           |
I(6)     |r0         |r0         |r0         |           |
```

Result: The program to compute the lr(0) items for a given grammar is successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 9

Title: Intermediate code generation – Postfix, Prefix.

Aim: To generate the intermediate code and postfix, prefix.

Algorithm:

In the analysis-synthesis model of a compiler, the front end of a compiler translates a source program into an independent intermediate code, then the back end of the compiler uses this intermediate code to generate the target code (which can be understood by the machine).

The benefits of using machine independent intermediate code are:

- Because of the machine independent intermediate code, portability will be enhanced. For example, if a compiler translates the source language to its target machine language without having the option for generating intermediate code, then for each new machine, a full native compiler is required. Because, obviously, there were some modifications in the compiler itself according to the machine specifications.
- Retargeting is facilitated
- It is easier to apply source code modification to improve the performance of source code by optimizing the intermediate code.

Postfix Notation –
The ordinary (infix) way of writing the sum of a and b is with operator in the middle : a + b
The postfix notation for the same expression places the operator at the right end as ab +. In general, if e1 and e2 are any postfix expressions, and
+ is any binary operator, the result of applying + to the values denoted by e1 and e2 is postfix notation by e1e2 +. No parentheses are needed in postfix notation because the position and arity (number of arguments) of the operators permit only one way to decode a postfix expression. In postfix notation the operator follows the operand.

RA1911033010047
KOLLA SESHU KIRAN

If we generate machine code directly from source code then for

target machine we will have n optimizers and n code generators but if we will have a machine independent intermediate code, we will have only one optimizer. Intermediate code can be either language specific (e.g., Bytecode for Java) or language. independent (three-address code).

Example – The postfix representation of the expression (a – b) * (c + d) + (a – b) is : ab – cd + *ab -+.

2. **Three-Address Code –**
   A statement involving no more than three references(two for operands and one for result) is known as three address statements. A sequence of three address statements is known as three address code. Three address statements are of the form x = y op z , here x, y, z will have address (memory location). Sometimes a statement might contain less than three references but it is still called three address statements.
   Example – The three address code for the expression a + b * c + d :
   T1 = b * c
   T2 = a + T1
   T3 = T2 + d
   T 1 , T 2 , T 3 are temporary variables.

3. **Syntax Tree –**
   Syntax tree is nothing more than a condensed form of a parse tree. The operator and keyword nodes of the parse tree are moved to their parents and a chain of single productions is replaced by a single link in the syntax tree; the internal nodes are operators and child nodes are operands. To form a syntax tree put parentheses in the expression, this way it's easy to recognize which operand should come first.

RA1911033010047
KOLLA SESHU KIRAN

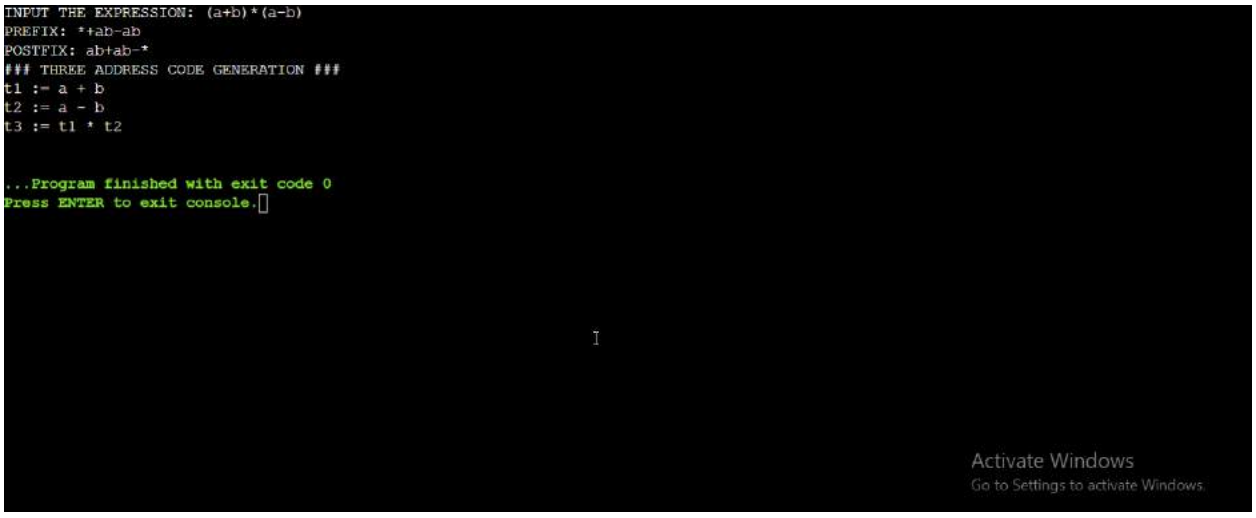
Code:

```
#RA1911033010047
OPERATORS = set(['+', '-', '*', '/', '(', ')'])
PRI = {'+':1, '-':1, '*':2, '/':2}

### INFIX ===> POSTFIX ###
def infix_to_postfix(formula):
    stack = [] # only pop when the coming op has priority
    output = ''
    for ch in formula:
        if ch not in OPERATORS:
            output += ch
        elif ch == '(':
            stack.append('(')
        elif ch == ')':
            while stack and stack[-1] != '(':
                output += stack.pop()
            stack.pop() # pop '('
        else:
            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
                output += stack.pop()
            stack.append(ch)
    # leftover
    while stack:
        output += stack.pop()
    print(f'POSTFIX: {output}')
    return output

### INFIX ===> PREFIX ###
def infix_to_prefix(formula):
    op_stack = []
    exp_stack = []
    for ch in formula:
        if not ch in OPERATORS:
            exp_stack.append(ch)
        elif ch == '(':
            op_stack.append(ch)
        elif ch == ')':
            while op_stack[-1] != '(':
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append( op+b+a )
            op_stack.pop() # pop '('
        else:
            while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append( op+b+a )
            op_stack.append(ch)

    # leftover
    while op_stack:
        op = op_stack.pop()
        a = exp_stack.pop()
        b = exp_stack.pop()
        exp_stack.append( op+b+a )
    print(f'PREFIX: {exp_stack[-1]}')
    return exp_stack[-1]

### THREE ADDRESS CODE GENERATION ###
def generate3AC(pos):
    print("### THREE ADDRESS CODE GENERATION ###")
    exp_stack = []
    t = 1

    for i in pos:
        if i not in OPERATORS:
            exp_stack.append(i)
        else:
            print(f't{t} := {exp_stack[-2]} {i} {exp_stack[-1]}')
            exp_stack=exp_stack[:-2]
            exp_stack.append(f't{t}')
            t+=1

expres = input("INPUT THE EXPRESSION: ")
```

RA1911033010047
KOLLA SESHU KIRAN

```
77 pre = infix_to_prefix(expres)
78 pos = infix_to_postfix(expres)
79 generate3AC(pos)
```


Output:



Result: The program for intermediate code generation and postfix, prefix is successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 10

Title: Intermediate code generation – Quadruple, Triple, Indirect triple

Aim: To implement intermediate code generation - quadruple, triple, indirect triple.

Algorithm:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op c perform the various actions. These are as follows:

1.Invoke a function getreg to find out the location L where the result of computation b op c should be stored.
2.Consult the address description for y to determine y'. If the value of y is currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction MOV y' , L to place a copy of y in L.
3.Generate the instruction OP z' , L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4.If the current value of y or z has no next uses or does not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those registers will no longer contain y or z.

RA1911033010047
KOLLA SESHU KIRAN

Code:

```
#RA1911033010047
OPERATORS = set(['+', '-', '*', '/', '(', ')'])
PRI = {'+':1, '-':1, '*':2, '/':2}

### INFIX ===> POSTFIX ###
def infix_to_postfix(formula):
    stack = [] # only pop when the coming op has priority
    output = ''
    for ch in formula:
        if ch not in OPERATORS:
            output += ch
        elif ch == '(':
            stack.append('(')
        elif ch == ')':
            while stack and stack[-1] != '(':
                output += stack.pop()
            stack.pop() # pop '('
        else:
            while stack and stack[-1] != '(' and PRI[ch] <= PRI[stack[-1]]:
                output += stack.pop()
            stack.append(ch)
    # leftover
    while stack:
        output += stack.pop()
    print(f'POSTFIX: {output}')
    return output

### INFIX ===> PREFIX ###
def infix_to_prefix(formula):
    op_stack = []
    exp_stack = []
    for ch in formula:
        if not ch in OPERATORS:
            exp_stack.append(ch)
        elif ch == '(':
            op_stack.append(ch)
        elif ch == ')':
            while op_stack[-1] != '(':
```

RA1911033010047
KOLLA SESHU KIRAN

```
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append( op+b+a )
            op_stack.pop() # pop '('
        else:
            while op_stack and op_stack[-1] != '(' and PRI[ch] <= PRI[op_stack[-1]]:
                op = op_stack.pop()
                a = exp_stack.pop()
                b = exp_stack.pop()
                exp_stack.append( op+b+a )
            op_stack.append(ch)

    # leftover
    while op_stack:
        op = op_stack.pop()
        a = exp_stack.pop()
        b = exp_stack.pop()
        exp_stack.append( op+b+a )
    print(f'PREFIX: {exp_stack[-1]}')
    return exp_stack[-1]

### THREE ADDRESS CODE GENERATION ###
def generate3AC(pos):
    print("### THREE ADDRESS CODE GENERATION ###")
    exp_stack = []
    t = 1

    for i in pos:
        if i not in OPERATORS:
            exp_stack.append(i)
        else:
            print(f't{t} := {exp_stack[-2]} {i} {exp_stack[-1]}')
            exp_stack=exp_stack[:-2]
            exp_stack.append(f't{t}')
            t+=1

expres = input("INPUT THE EXPRESSION: ")
```



```
pre = infix_to_prefix(expres)
pos = infix_to_postfix(expres)
generate3AC(pos)
def Quadruple(pos):
  stack = []
  op = []
  x = 1
  for i in pos:
    if i not in OPERATORS:
       stack.append(i)
    elif i == '-':
        op1 = stack.pop()
        stack.append("t(%s)" %x)
        print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i,op1,"(-)"," t(%s)" %x))
        x = x+1
        if stack != []:
          op2 = stack.pop()
          op1 = stack.pop()
          print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format("+",op1,op2," t(%s)" %x))
          stack.append("t(%s)" %x)
          x = x+1
    elif i == '=':
      op2 = stack.pop()
      op1 = stack.pop()
      print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i,op2,"(-)",op1))
    else:
      op1 = stack.pop()
      op2 = stack.pop()
      print("{0:^4s} | {1:^4s} | {2:^4s}|{3:4s}".format(i,op2,op1," t(%s)" %x))
      stack.append("t(%s)" %x)
      x = x+1
print("The quadruple for the expression ")
print(" OP | ARG 1 |ARG 2 |RESULT  ")
Quadruple(pos)

def Triple(pos):
        stack = []
        op = []
```

RA1911033010047
KOLLA SESHU KIRAN

```
        x = 0
        for i in pos:
          if i not in OPERATORS:
            stack.append(i)
          elif i == '-':
            op1 = stack.pop()
            stack.append("(%s)" %x)
            print("{0:^4s} | {1:^4s} | {2:^4s}".format(i,op1,"(-)"))
            x = x+1
            if stack != []:
              op2 = stack.pop()
              op1 = stack.pop()
              print("{0:^4s} | {1:^4s} | {2:^4s}".format("+",op1,op2))
              stack.append("(%s)" %x)
              x = x+1
          elif i == '=':
            op2 = stack.pop()
            op1 = stack.pop()
            print("{0:^4s} | {1:^4s} | {2:^4s}".format(i,op1,op2))
          else:
            op1 = stack.pop()
            if stack != []:
              op2 = stack.pop()
              print("{0:^4s} | {1:^4s} | {2:^4s}".format(i,op2,op1))
              stack.append("(%s)" %x)
              x = x+1
print("The triple for given expression")
print("  OP | ARG 1 |ARG 2  ")
Triple(pos)
```


Output:

```
INPUT THE EXPRESSION: (a+b)*(a-b)
PREFIX: *+ab-ab
POSTFIX: ab+ab-*
### THREE ADDRESS CODE GENERATION ###
t1 := a + b
t2 := a - b
t3 := t1 * t2
The quadruple for the expression
 OP | ARG 1 |ARG 2 |RESULT
 +  |  a   |  b   | t(1)
 -  |  b   | (-)  | t(2)
 +  |  a   | t(2) | t(3)
 *  | t(1) | t(3) | t(4)
The triple for given expression
  OP | ARG 1 |ARG 2
 +   |  a   |  b
 -   |  b   | (-)
 +   |  a   | (1)
 *   | (0)  | (2)


...Program finished with exit code 0
Press ENTER to exit console.
```

Result: The program was successfully implemented**.**

RA1911033010047
Kolla. Sesha Kiran

* Compiler Design *

Assignment 4

Date:- 31/3/22

Cross Compiler:-

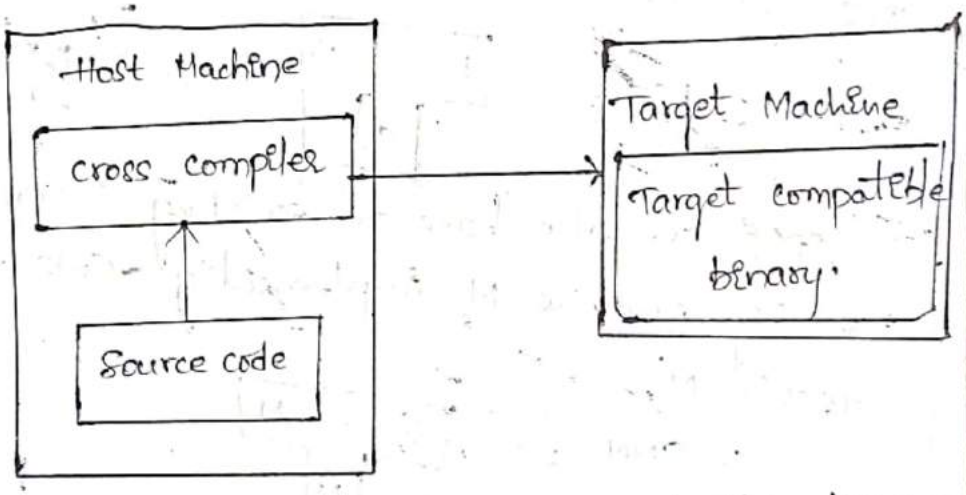
Cross compiler is also known as "retargetable compiler"
GNU Gcc is an example for cross compiler.

A cross compiler is a type of compiler, that generates the code generated targeted to run on a system different the one generating it.

→ For Example, A compiler that runs on windows platform also generates a code that runs on Linux platform is a cross compiler.

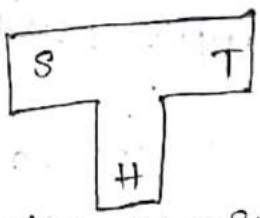
→ The process of creating executable code for different machines is also called "retargeting".

* cross compiler operation *:-



→ A host language is a language in which the compiler is written.

- T- diagram.



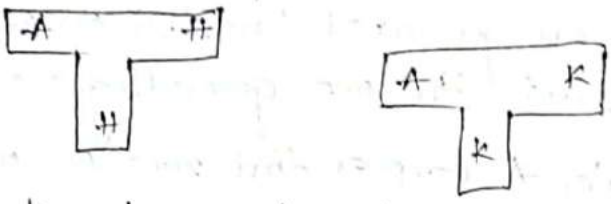
→ Cross compilers are used very often in practice.

T compiles:-

→ If we want a compiler from language A to language B on a machine with language E,

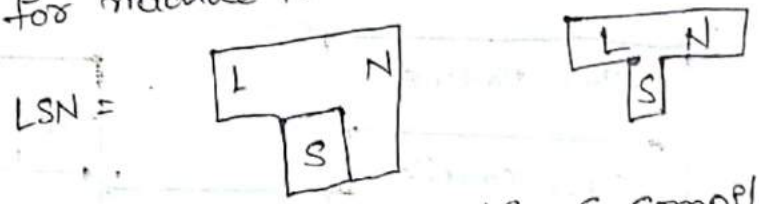
- write one with E.

→ Construct a compiler between a source and a target language using one host language from another host language.

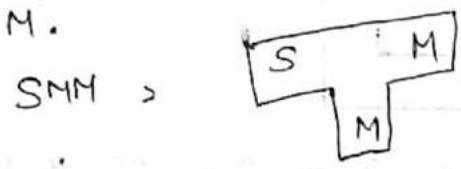


→ If we have to implement, from scratch, a compiler from a high-level language A to a machine which is also a host language.

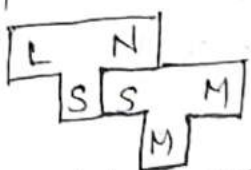
→ Suppose we have cross compiler for a new language L in implementation language S generating code for machine N.



→ Suppose we also have an existing S compiler running machine M implementing code for machine M.



Now, Run LSN through SMM to produce LMN

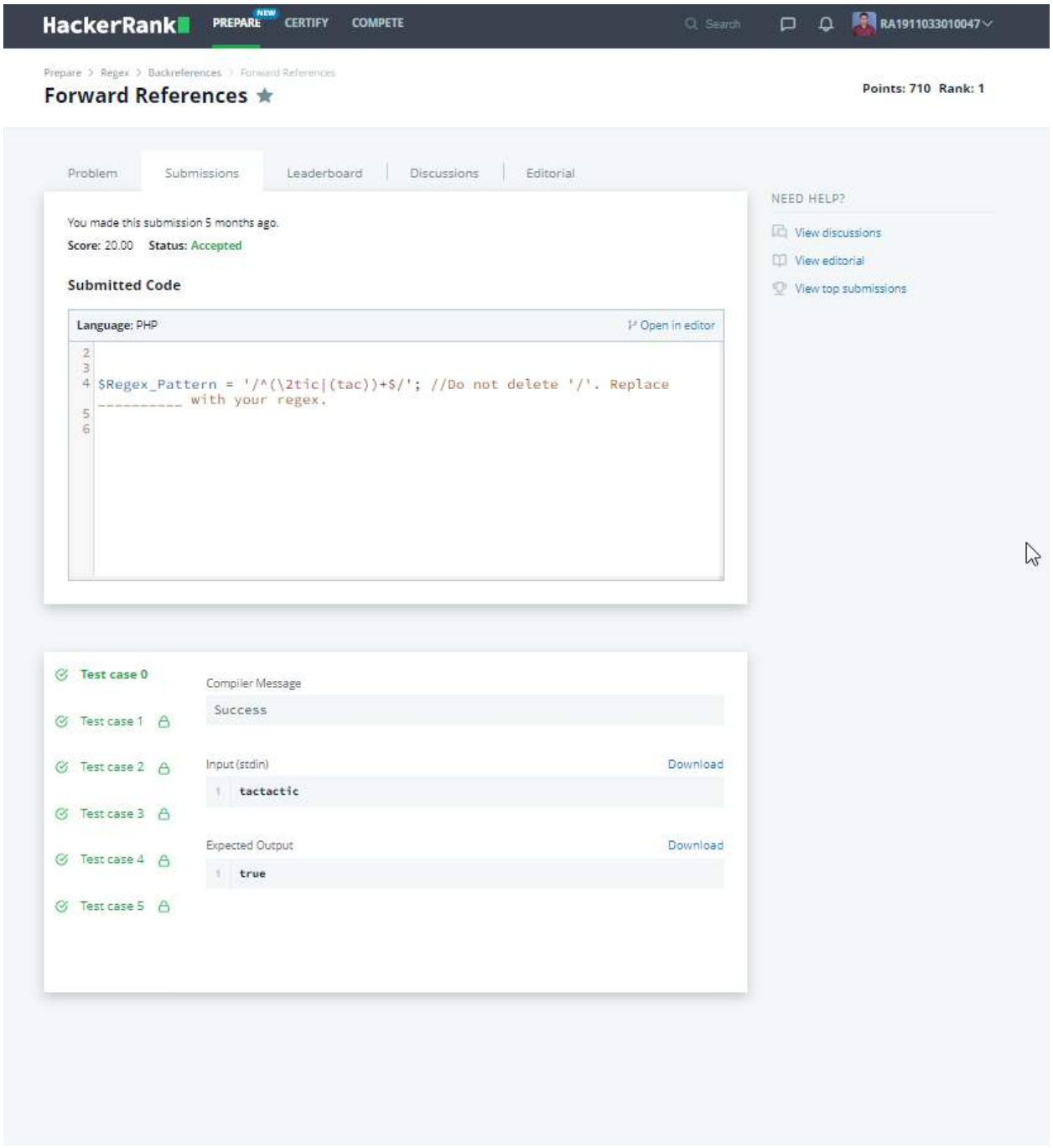


The first T describes a compiler from L to N written in S: second T → S to M written in M.

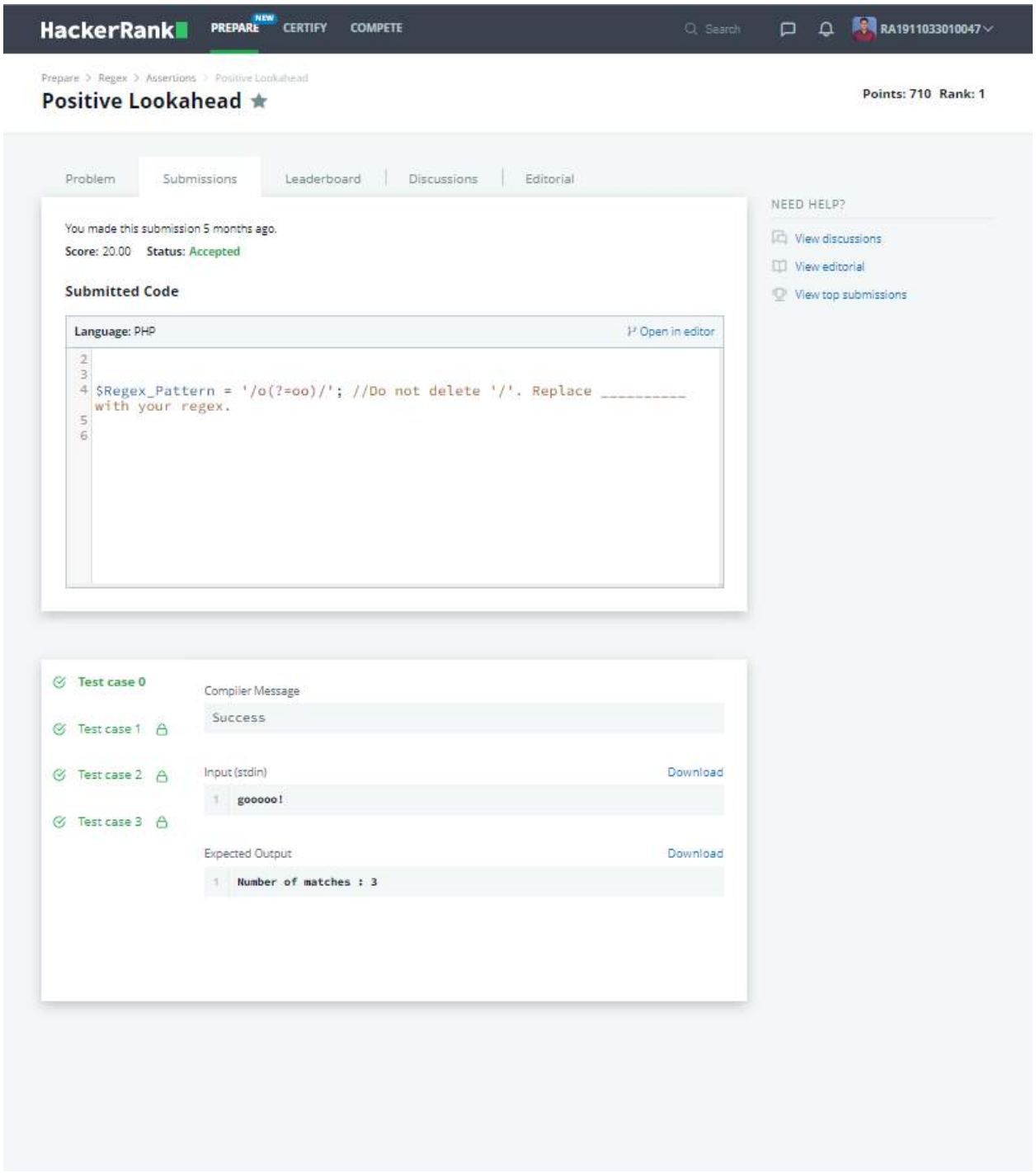
RA1911033010047
KOLLA SESHU KIRAN

HACKERRANK REGEX SOLUTIONS(21 - 31)

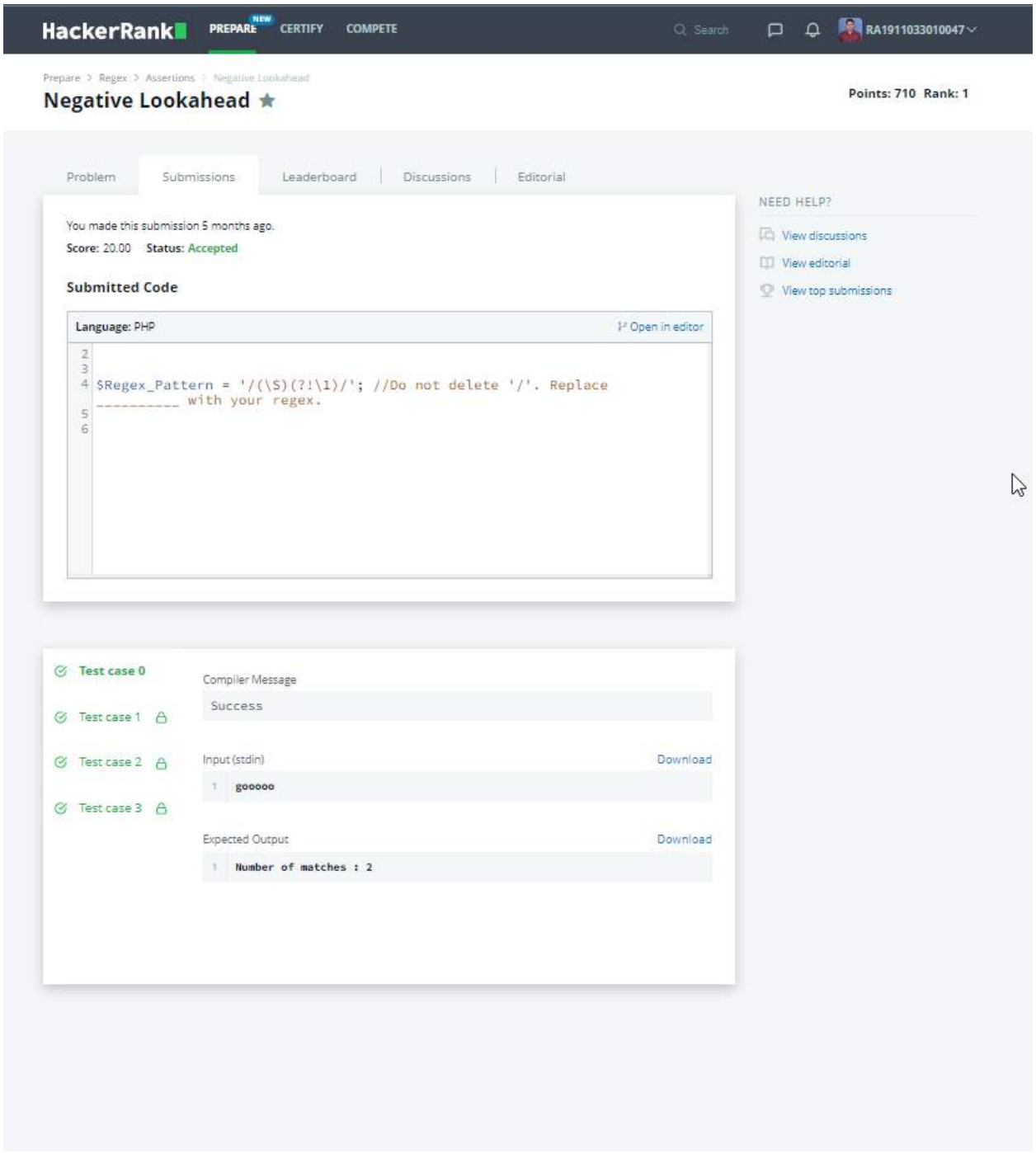
Date: 31-03-2022



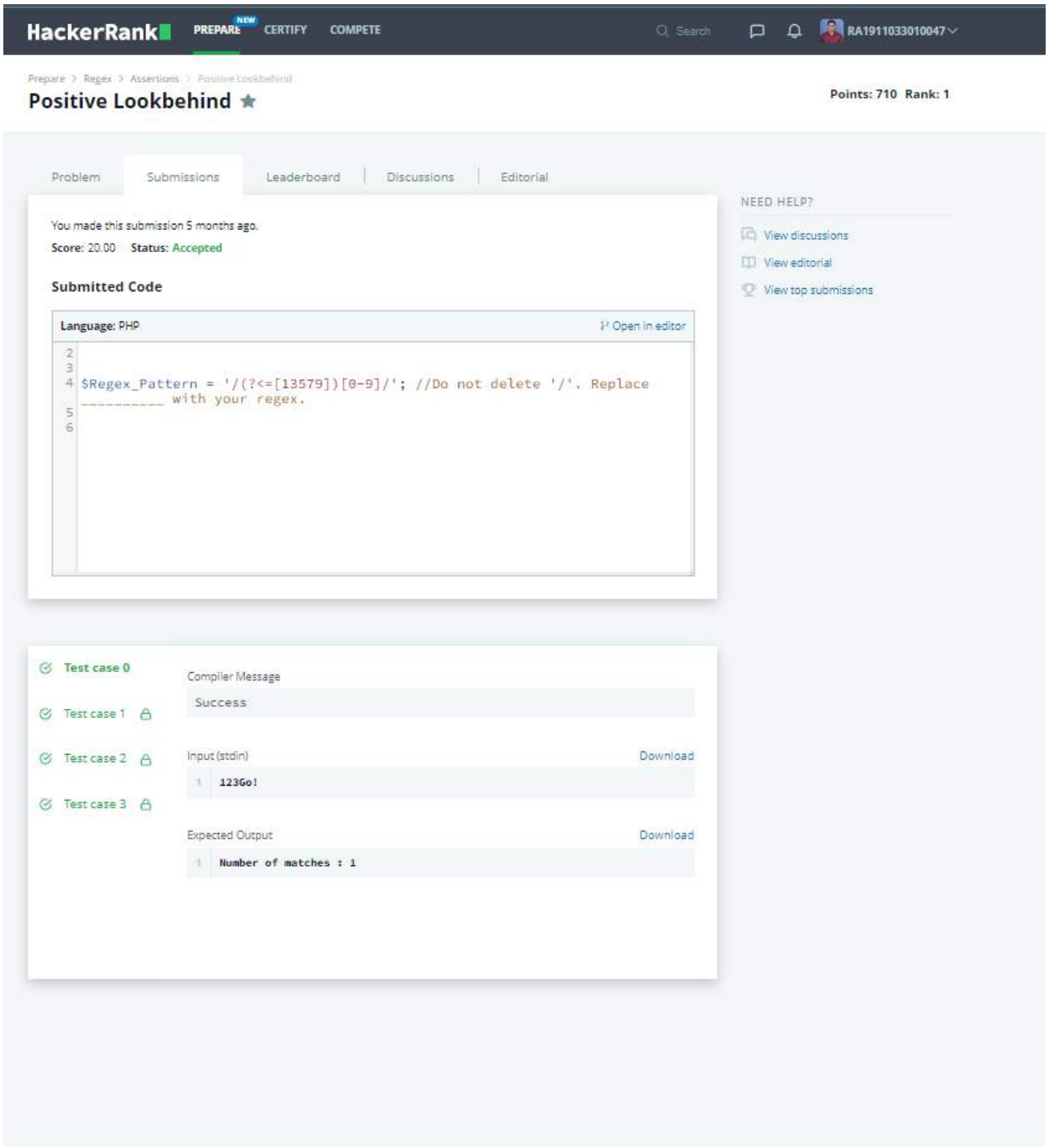
RA1911033010047
KOLLA SESHU KIRAN



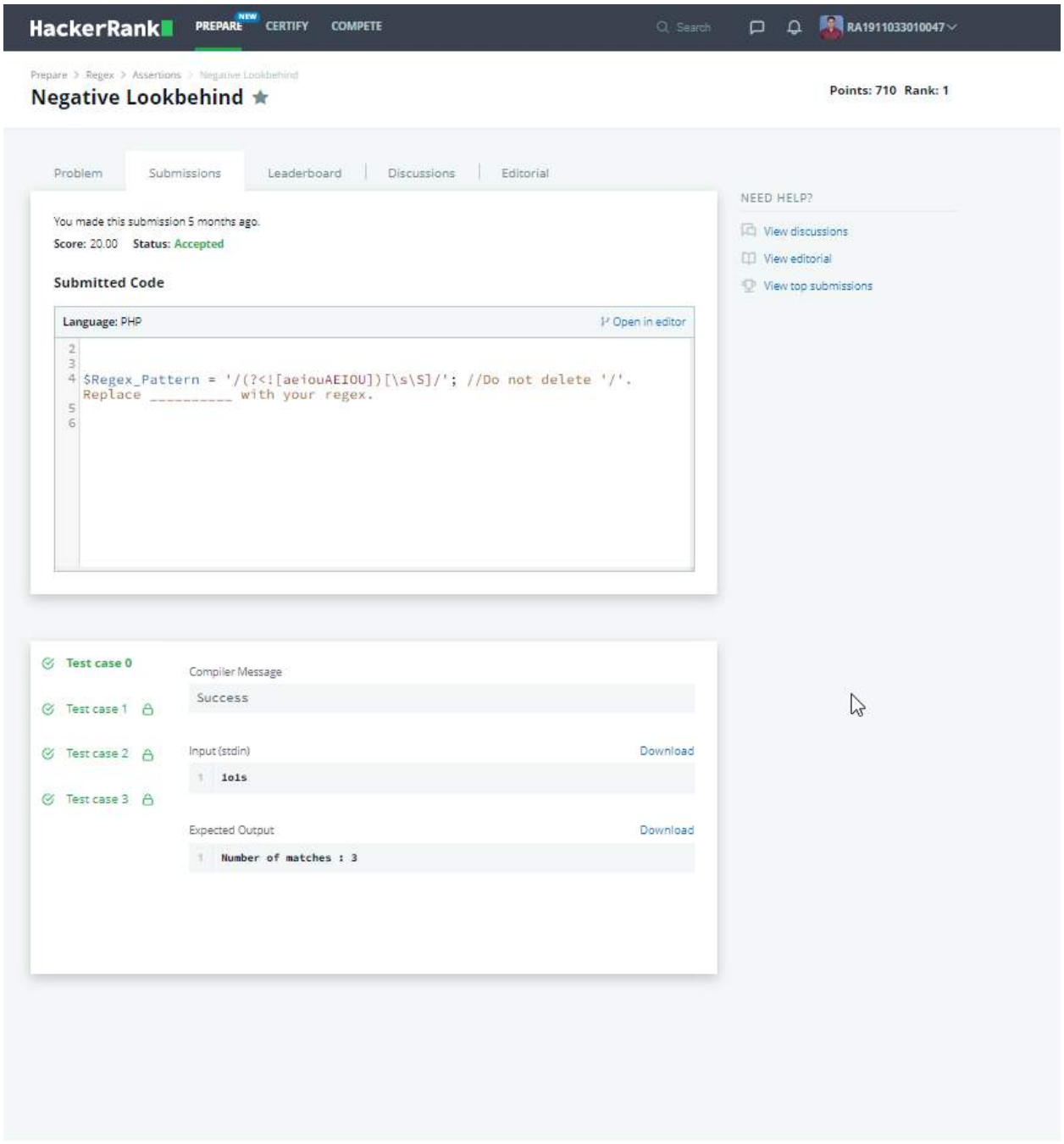
RA1911033010047
KOLLA SESHU KIRAN



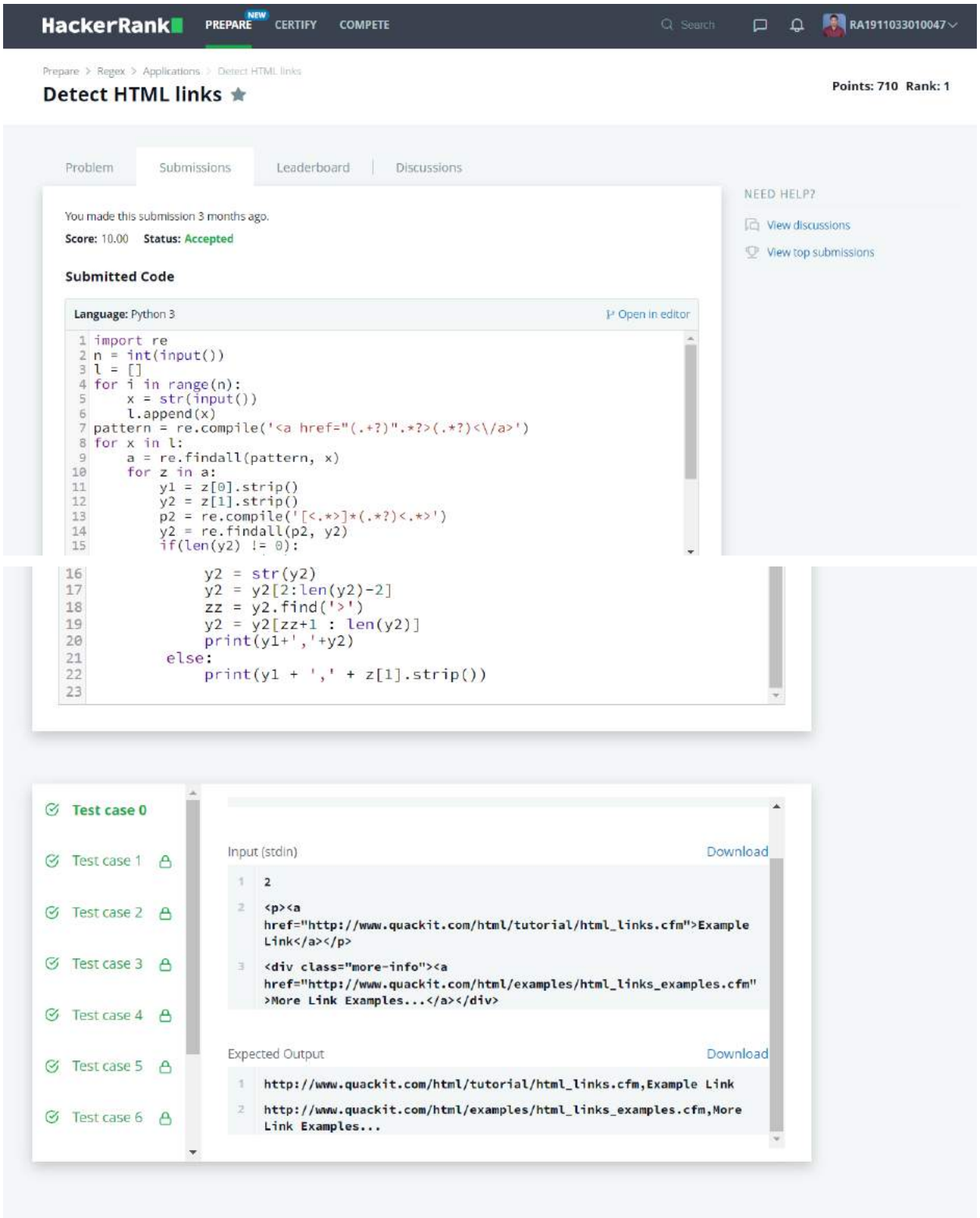
RA1911033010047
KOLLA SESHU KIRAN



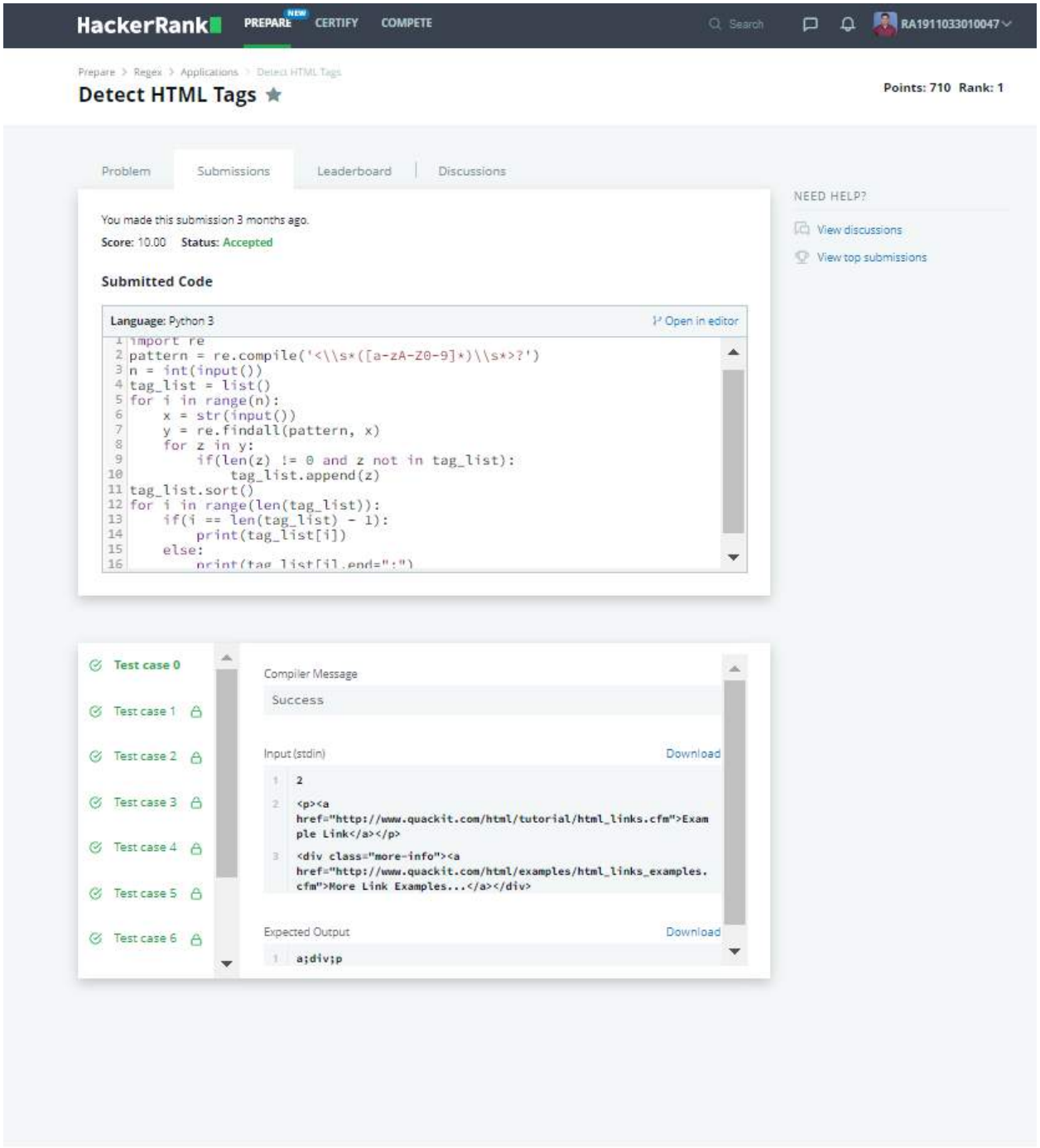
RA1911033010047
KOLLA SESHU KIRAN



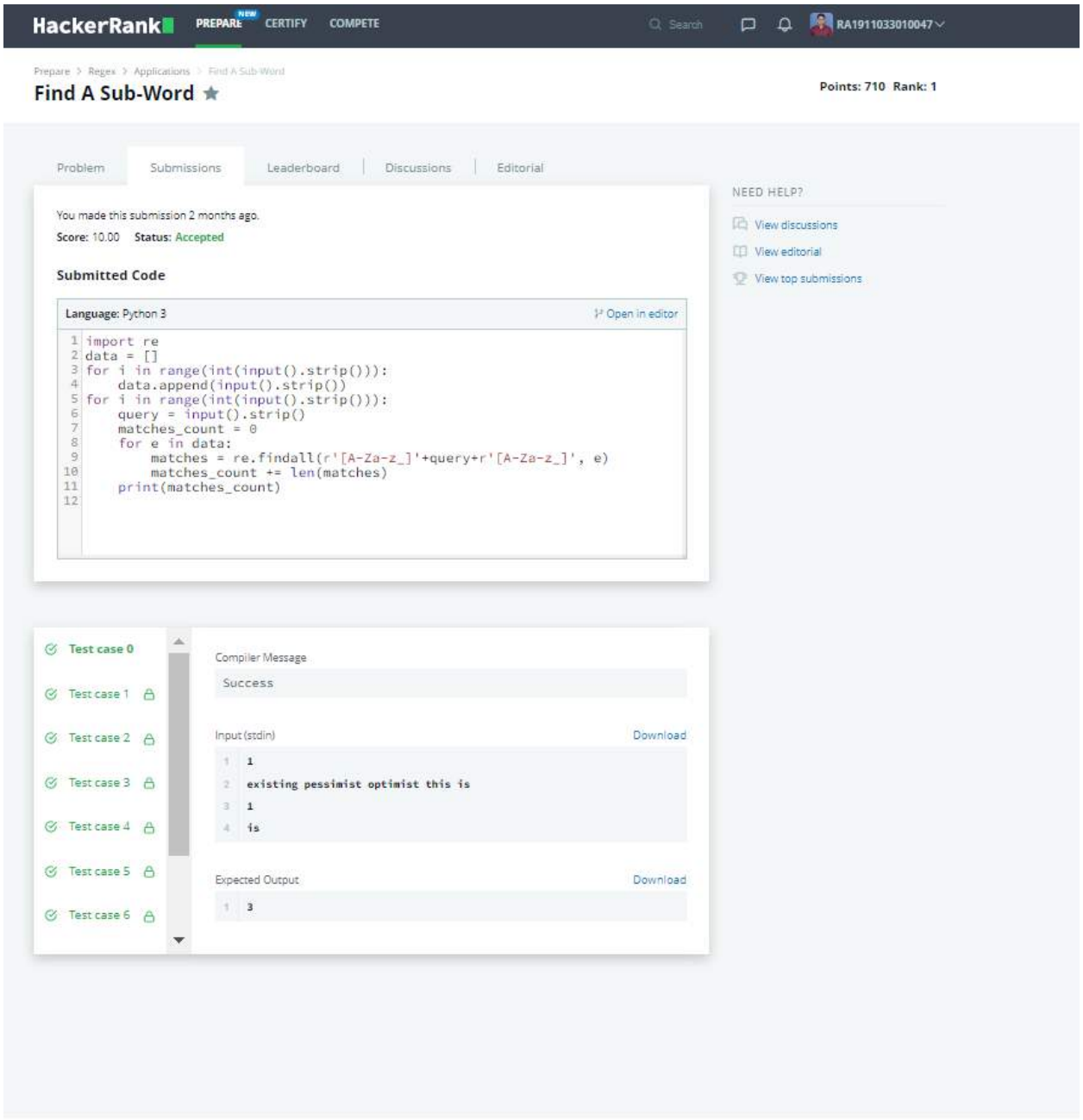
RA1911033010047
KOLLA SESHU KIRAN



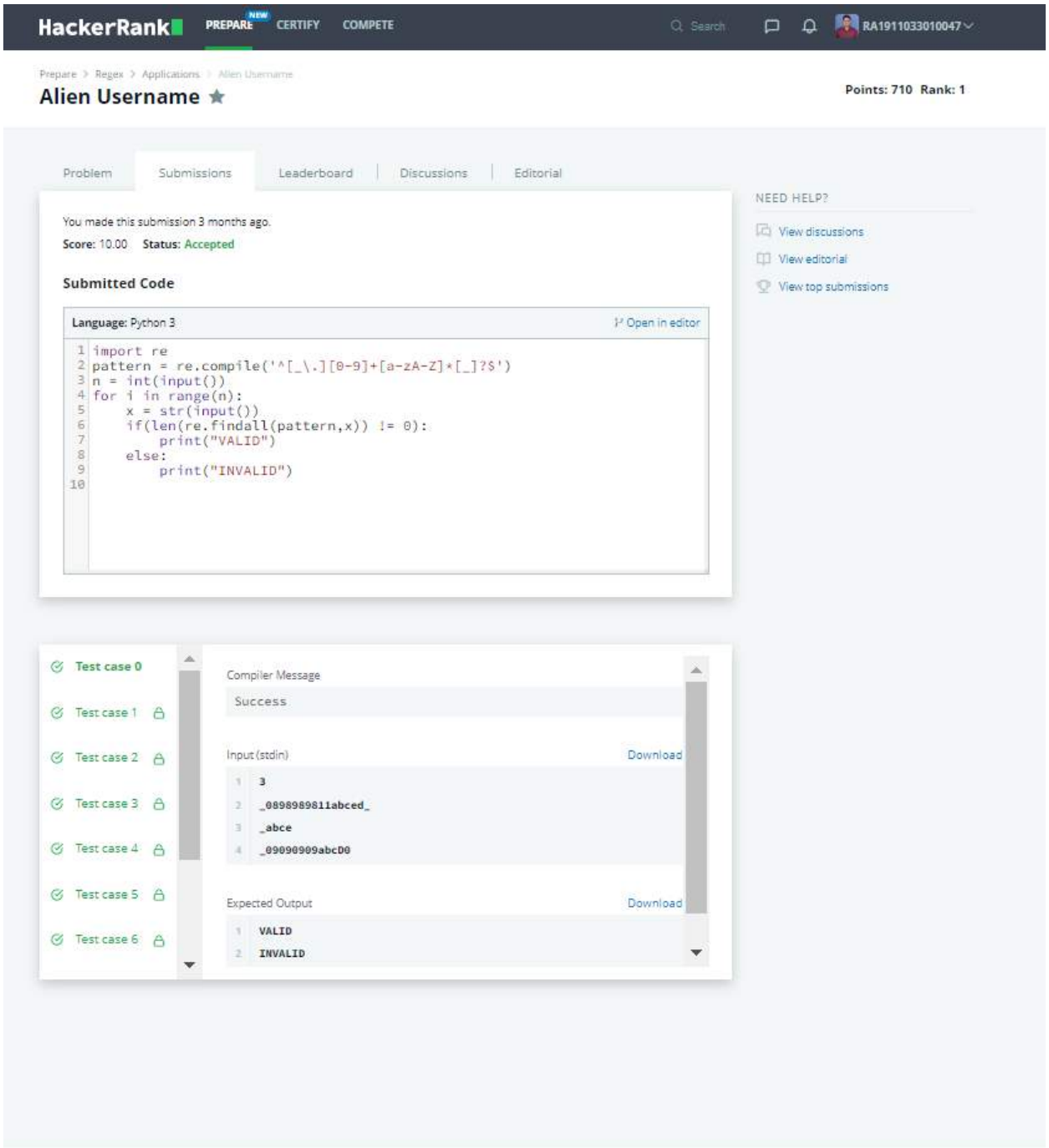
RA1911033010047
KOLLA SESHU KIRAN



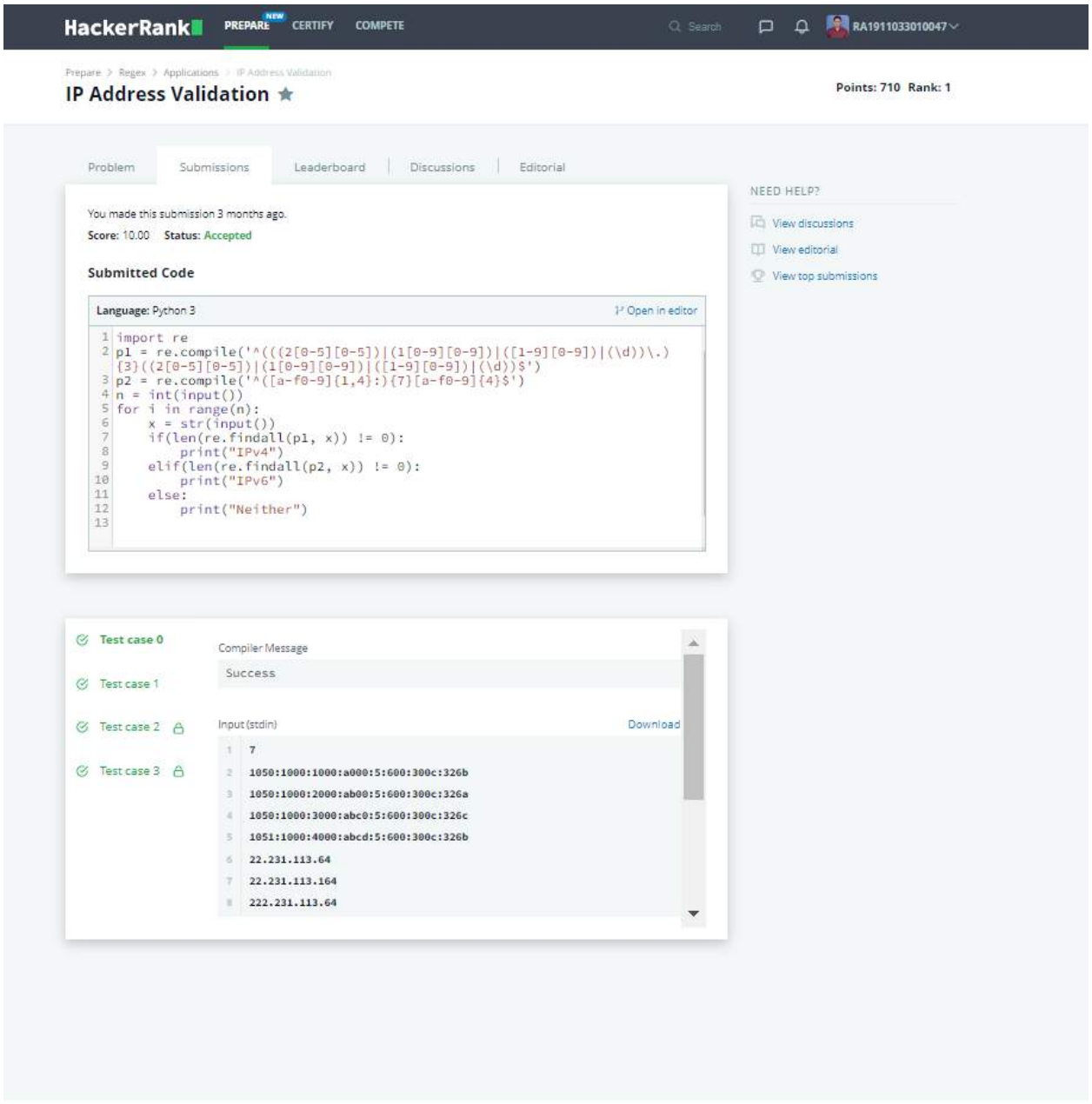
RA1911033010047
KOLLA SESHU KIRAN



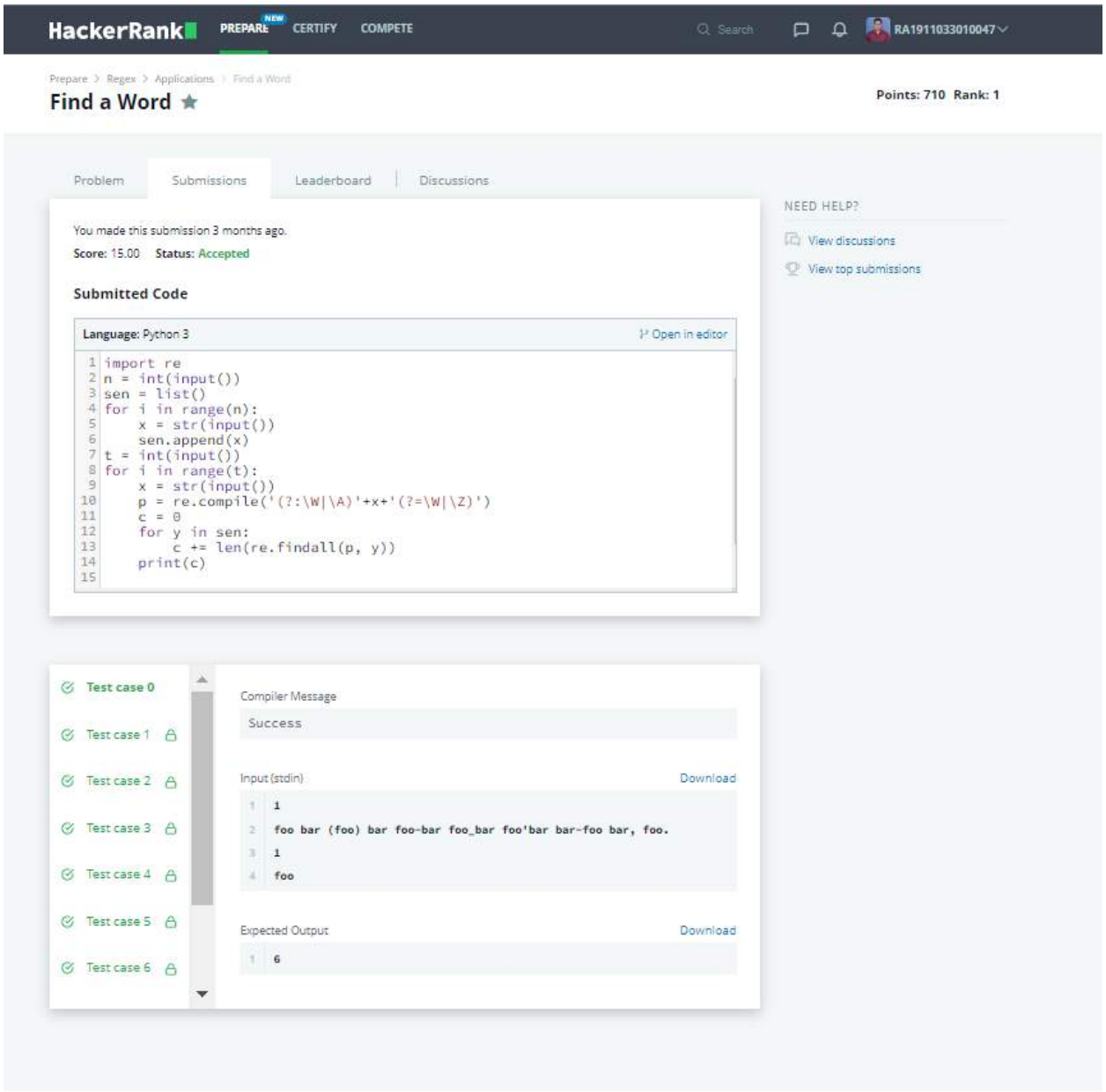
RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 11

Title: Implementation of DAG

Aim: To implement the DAG

Algorithm:

- Case (i) x:= y OP z
- Case (ii) x:= OP y
- Case (iii) x:= y
- Step 1:
  - If y operand is undefined then create node(y).
  - If z operand is undefined then for case(i) create node(z).
- Step 2:
  - For case(i), create node(OP) whose right child is node(z) and left child is node(y).
  - For case(ii), check whether there is node(OP) with one child node(y).
  - For case(iii), node n will be node(y).
- Output:
  - For node(x) delete x from the list of identifiers. Append x to the attached identifiers list for the node n found in step 2. Finally set node(x) to n.

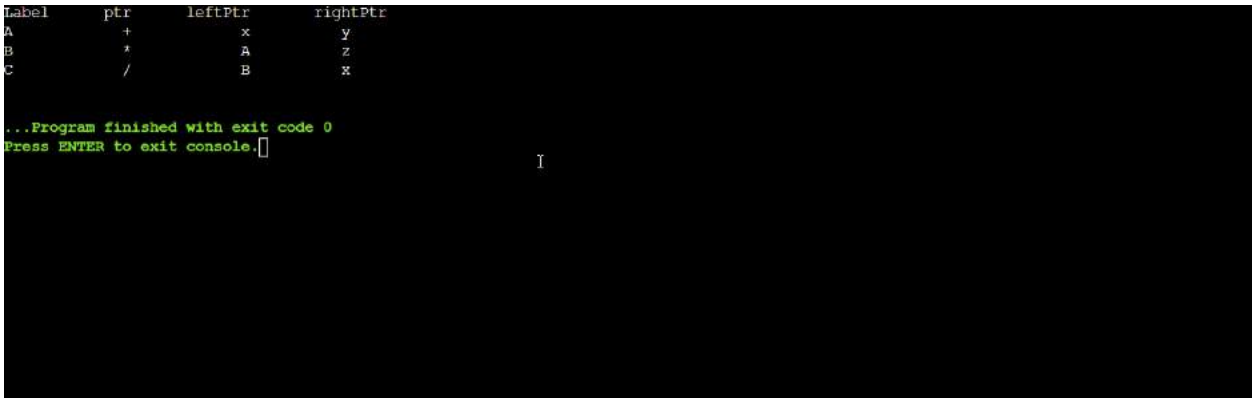
RA1911033010047
KOLLA SESHU KIRAN

Code:

```cpp
//RA1911033010047
#include<iostream>
#include<string>
#include<unordered_map>
using namespace std;
class DAG
{   public:
    char label;
    char data;
    DAG* left;
    DAG* right;

    DAG(char x){
      label='_';
      data=x;
      left=NULL;
      right=NULL;
    }
    DAG(char lb, char x, DAG* lt, DAG* rt){
      label=lb;
      data=x;
      left=lt;
      right=rt;
    }
};

int main(){
    int n;
    n=3;
    string st[n];
    st[0]="A=x+y";
    st[1]="B=A*z";
    st[2]="C=B/x";
    unordered_map<char, DAG*> labelDAGNode;

    for(int i=0;i<3;i++){
        string stTemp=st[i];
        for(int j=0;j<5;j++){
            char tempLabel = stTemp[0];
            char tempLeft = stTemp[2];
            char tempData = stTemp[3];
            char tempRight = stTemp[4];
            DAG* leftPtr;
            DAG* rightPtr;
            if(labelDAGNode.count(tempLeft) == 0){
                leftPtr = new DAG(tempLeft);
            }
            else{
                leftPtr = labelDAGNode[tempLeft];
            }
            if(labelDAGNode.count(tempRight) == 0){
                rightPtr = new DAG(tempRight);
            }
            else{
                rightPtr = labelDAGNode[tempRight];
            }
            DAG* nn = new DAG(tempLabel,tempData,leftPtr,rightPtr);
            labelDAGNode.insert(make_pair(tempLabel,nn));
        }
    }
    cout<<"Label      ptr      leftPtr       rightPtr"<<endl;
    for(int i=0;i<n;i++){
        DAG* x=labelDAGNode[st[i][0]];
        cout<<st[i][0]<<"            "<<x->data<<"            ";
        if(x->left->label=='_')cout<<x->left->data;
        else cout<<x->left->label;
        cout<<"          ";
        if(x->right->label=='_')cout<<x->right->data;
        else cout<<x->right->label;
        cout<<endl;

    }
    return 0;
}
```

RA1911033010047
KOLLA SESHU KIRAN

Output:



Result: The program for implementation of DAG is successfully implemented.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 11

Title: Implementation of three address code

Aim: To write a program to implement three address code:Quadruple,triple,indirect triple

Algorithm:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op c perform the various actions. These are as follows: 1. Invoke a function getreg to find out the location L where the result of computation b op c should be stored.

2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y'. If the value of y is not already in L then generate the instruction MOV y', L to place a copy of y in L.

3. Generate the instruction OP z', L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all

other descriptors.

4. If the current value of y or z has no next uses or does not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x = y op z those registers will no longer contain y or z.

RA1911033010047
KOLLA SESHU KIRAN

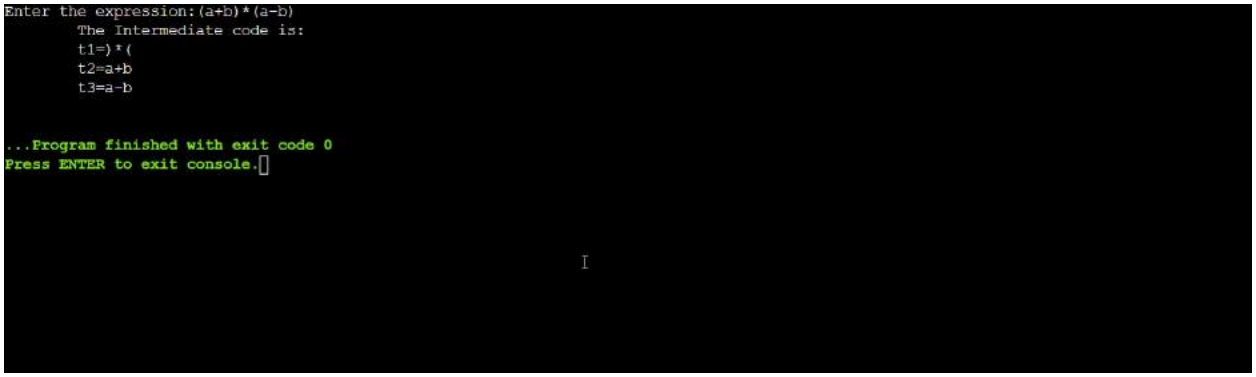
Code:

```c
//RA1911033010047
#include<stdio.h>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
void small();
void dove(int i);
int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;
char sw[5]={'=','-','+','/','*'},j[20],a[5],b[5],ch[2];
void main()
{
printf("Enter the expression:");
scanf("%s",j);
printf("\tThe Intermediate code is:\n");
small();
}
void dove(int i)
{
a[0]=b[0]='\0';
if(!isdigit(j[i+2])&&!isdigit(j[i-2]))
{
a[0]=j[i-1];
b[0]=j[i+1];
}
if(isdigit(j[i+2])){
a[0]=j[i-1];
b[0]='t';
b[1]=j[i+2];
}
if(isdigit(j[i-2]))
{
b[0]=j[i+1];
a[0]='t';
a[1]=j[i-2];
b[1]='\0';
}
if(isdigit(j[i+2]) &&isdigit(j[i-2]))
{
a[0]='t';
b[0]='t';
a[1]=j[i-2];
b[1]=j[i+2];
sprintf(ch,"%d",c);
j[i+2]=j[i-2]=ch[0];
}
if(j[i]=='*')
printf("\tt%d=%s*%s\n",c,a,b);
if(j[i]=='/')
printf("\tt%d=%s/%s\n",c,a,b);
if(j[i]=='+')
printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')
printf("\tt%d=%s-%s\n",c,a,b);
if(j[i]=='=')
printf("\t%c=t%d",j[i-1],--c);
sprintf(ch,"%d",c);
j[i]=ch[0];
c++;
small();
}
void small()
{
pi=0;l=0;
for(i=0;i<strlen(j);i++)
{
for(m=0;m<5;m++)
if(j[i]==sw[m])
if(pi<=p[m])
{
pi=p[m];
 l=1;
 k=i;
}
}
if(l==1)
dove(k);
else
```

RA1911033010047
KOLLA SESHU KIRAN

```
77  exit(0);}
```


Output:



Result: The program was successfully implemented,compiled and run output is obtained.

RA1911033010047
KOLLA SESHU KIRAN

EXPERIMENT - 13

Title:

Aim: To implement various storage allocation techniques algorithms

Algorithm:

Static storage Allocation

- In static allocation, names are bound to storage locations.
- If memory is created at compile time then the memory will be created in a static area and only once.
- Static allocation supports the dynamic data structure that means memory is created only at compile time and deallocated after program completion.
- The drawback with static storage allocation is that the size and position of data objects should be known at compile time.
- Another drawback is restriction of the recursion procedure.

Stack Storage Allocation

- In static storage allocation, storage is organized as a stack.
- An activation record is pushed into the stack when activation begins and it is popped when the activation ends.
- Activation record contains the locals so that they are bound to fresh storage in each activation record. The value of locals is deleted when the activation ends.
- It works on the basis of last-in-first-out (LIFO) and this allocation supports the recursion process.

Heap Storage Allocation

- Heap allocation is the most flexible allocation scheme.
- Allocation and deallocation of memory can be done at any time and at any place depending upon the user's requirement.
- Heap allocation is used to allocate memory to the variables dynamically and when the variables are no more used then claim it back.
- Heap storage allocation supports the recursion process.

RA1911033010047
KOLLA SESHU KIRAN

Code:

1) Stack

```c
//RA1911033010047
#include<stdio.h>
#include<conio.h>

int i, stk[100], top=-1, n;
void show()
{
     for(i=0;i<=top;i++)
     printf("%d\t",stk[i]);
 }
void push()
{
     int item;
     if(top == n-1)
        printf("\nStack is full.");
     else
     {
          printf("\nEnter the item: ");
          scanf("%d",&item);
          stk[++top]=item;
     }}
void pop()
{
         if(top==-1)
             printf("Stack is empty.");
         else
         {
             printf("%d is popped.",stk[top]);
             top--;
 }}
int main()
{
          int i,op;
          printf("Enter the size of the stack: ");
          scanf("%d",&n);
          do
          {
              printf("\n1 : Push");
              printf("\n2 : Pop");
              printf("\n3 : Display");
              printf("\n4 : Exit");
              printf("\nEnter your choice: ");
              scanf("%d",&op);
              switch(op)
              {
                      case 1:
                                  push();
                                  break;
                      case 2:
                                  pop();
                                  break;
                      case 3:
                                  show();
                                  break;
                }
         }while(op!=4);
         getch();
}
```

RA1911033010047
KOLLA SESHU KIRAN

Output:

RA1911033010047
KOLLA SESHU KIRAN

2)Heap:

```
//RA1911033010047
#include <iostream>
#include <cstdlib>
#include <vector>
#include <iterator>
using namespace std;
/*
 * Class Declaration
 */
class Heap
{
    private:
        vector <int> heap;
        int left(int parent);
        int right(int parent);
        int parent(int child);
        void heapifyup(int index);
        void heapifydown(int index);
    public:
        Heap()
        {}
        void Insert(int element);
        void DeleteMin();
        int ExtractMin();
        void DisplayHeap();
        int Size();
};
/*
 * Return Heap Size
 */
int Heap::Size()
{
    return heap.size();
}

/*
 * Insert Element into a Heap
 */
```

RA1911033010047
KOLLA SESHU KIRAN

```
void Heap::Insert(int element)
{
    heap.push_back(element);
    heapifyup(heap.size() -1);
}
/*
 * Delete Minimum Element
 */
void Heap::DeleteMin()
{
    if (heap.size() == 0)
    {
        cout<<"Heap is Empty"<<endl;
        return;
    }
    heap[0] = heap.at(heap.size() - 1);
    heap.pop_back();
    heapifydown(0);
    cout<<"Element Deleted"<<endl;
}

/*
 * Extract Minimum Element
 */
int Heap::ExtractMin()
{
    if (heap.size() == 0)
    {
        return -1;
    }
    else
        return heap.front();
}

/*
 * Display Heap
 */
void Heap::DisplayHeap()
```



```
{
    vector <int>::iterator pos = heap.begin();
    cout<<"Heap -->  ";
    while (pos != heap.end())
    {
        cout<<*pos<<" ";
        pos++;
    }
    cout<<endl;
}

/*
 * Return Left Child
 */
int Heap::left(int parent)
{
    int l = 2 * parent + 1;
    if(l < heap.size())
        return l;
    else
        return -1;
}

/*
 * Return Right Child
 */
int Heap::right(int parent)
{
    int r = 2 * parent + 2;
    if(r < heap.size())
        return r;
    else
 return -1;
}

/*
 * Return Parent
 */
```

RA1911033010047
KOLLA SESHU KIRAN

```
int Heap::parent(int child)
{
    int p = (child - 1)/2;
    if(child == 0)
        return -1;
else
        return p;
}

/*
 * Heapify- Maintain Heap Structure bottom up
 */
void Heap::heapifyup(int in)
{
    if (in >= 0 && parent(in) >= 0 && heap[parent(in)] > heap[in])
    {
        int temp = heap[in];
        heap[in] = heap[parent(in)];
        heap[parent(in)] = temp;
        heapifyup(parent(in));
    }
}

/*
 * Heapify- Maintain Heap Structure top down
 */
void Heap::heapifydown(int in)
{

    int child = left(in);
    int child1 = right(in);
    if (child >= 0 && child1 >= 0 && heap[child] > heap[child1])
    {
       child = child1;
  }
    if (child > 0)
    {
        int temp = heap[in];
```



```
        heap[in] = heap[child];
        heap[child] = temp;
        heapifydown(child);
    }
}

/*
 * Main Contains Menu
 */
int main()
{
    Heap h;
    while (1)
    {
        cout<<"-----------------"<<endl;
        cout<<"Operations on Heap"<<endl;
        cout<<"-----------------"<<endl;
        cout<<"1.Insert Element"<<endl;
        cout<<"2.Delete Minimum Element"<<endl;
        cout<<"3.Extract Minimum Element"<<endl;
        cout<<"4.Print Heap"<<endl;
        cout<<"5.Exit"<<endl;
        int choice, element;
        cout<<"Enter your choice: ";
        cin>>choice;
        switch(choice)
        {
case 1:
            cout<<"Enter the element to be inserted: ";
            cin>>element;
            h.Insert(element);
            break;
        case 2:
            h.DeleteMin();
            break;
        case 3:
            cout<<"Minimum Element: ";
            if (h.ExtractMin() == -1)
```

RA1911033010047
KOLLA SESHU KIRAN

```
            {
                cout<<"Heap is Empty"<<endl;
            }
            else
                cout<<"Minimum Element:  "<<h.ExtractMin()<<endl;
            break;
        case 4:
            cout<<"Displaying elements of Hwap:  ";
            h.DisplayHeap();
            break;
        case 5:
            exit(1);
        default:
            cout<<"Enter Correct Choice"<<endl;
        }
    }
    return 0;
}
```


Output:

```
------------------
Operations on Heap
------------------
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Print Heap
5.Exit
Enter your choice: 1
Enter the element to be inserted: 2
------------------
Operations on Heap
------------------
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Print Heap
5.Exit
Enter your choice: 4
Displaying elements of Hwap:  Heap -->  2
------------------
Operations on Heap
------------------
1.Insert Element
2.Delete Minimum Element
3.Extract Minimum Element
4.Print Heap
5.Exit
```


Result : The programs for different Algorithms for various storage allocation techniques were implemented successfully.

RA1911033010047
Kolla Seshu Kiran

Compiler Design

18/4/22

* Assignment - 5 *

* Global Data-Flow Analysis *

→ To apply global optimizations on basic blocks, data-flow information is collected by solving systems of data-flow equations.

→ Suppose we need to determine the live-variables for a sequence of statement S.

$$in[S] = use[S] \cup (out[S] - def[S])$$

B1 : i := m-1
j := n

B2 : j := j-1

B3 : := i+j

Solution:

$in[B1] = \{m,n\} \cup (\{i,j\} - \{i,j\}) = \{m,n\}$

$out[B1] = in[B2] = \{i,j\}$

$in[B2] = \{j\} \cup (\{i,j\} - \{j\}) = \{i,j\}$

$out[B2] = in[B3] = \{i,j\}$

→ Suppose we need to determine the reaching definitions for a sequence of statements S.

$$out[S] = gen[S] \cup (in[S] - kill[S])$$

B1: d1: i := m-1 — out[B1] = gen[B1] = {d1, d2}

d2: j := n — out[B2] = gen[B2] ∪ {d1}

B2: d3: j := j-1 — = {d1, d3}

B3:

d1 reaches B2 and B3 and d2 reaches B2, but not B3 because d2 is killed in B2.

→ To efficiently optimize the code compiler collects all the information about the program and distribute the information to each block of the flow graph. This process is known as data-flow graph analysis.

→ A framework for proving facts about program

Gen & Kill Sets:-

Gen sets:- A current basic block or instruction is said to be in the GEN set if it creates the definition in any case, the must be in the output.

→ Block B generates def d if d is in B and is not followed in B by any unambiguous defs of the same variable as d.

→ GEN(B) is the set of defs generated by B.

The transfer equation for B is :-

$$OUT(B) = IN(B) - KILL(B) \cup GEN(B)$$

→ Kill sets:- A current basic block or instruction is said to be in the KILL set if it redefines a variable in the Expression. After that, the expression is invalid.

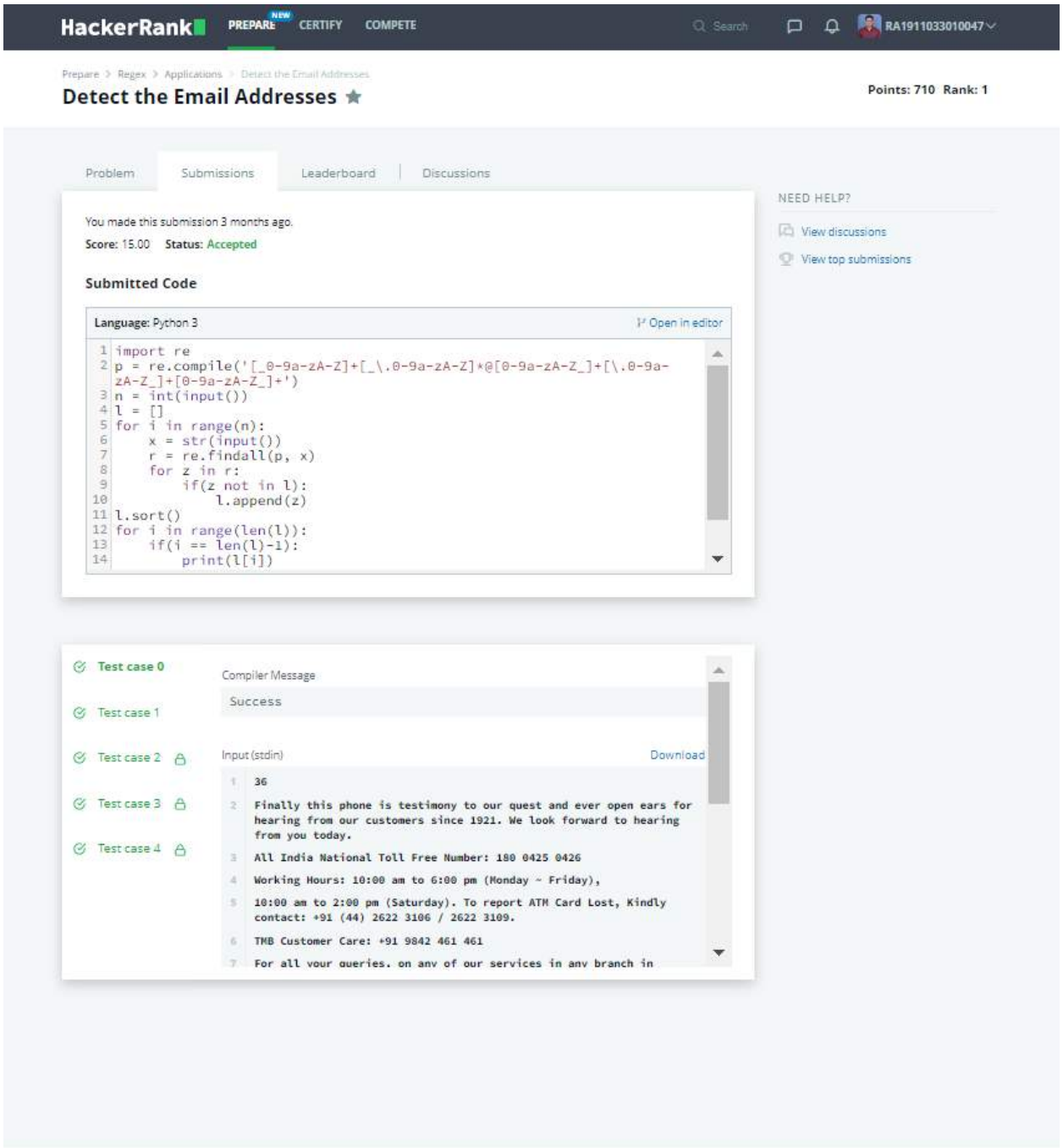
→ Say def d kills defe if d and e define the same variable, and d is unambiguous. Ambiguous defs cannot kill other defs.

→ kill(B) is the set of sets defs not in B that are killed by some def in B.

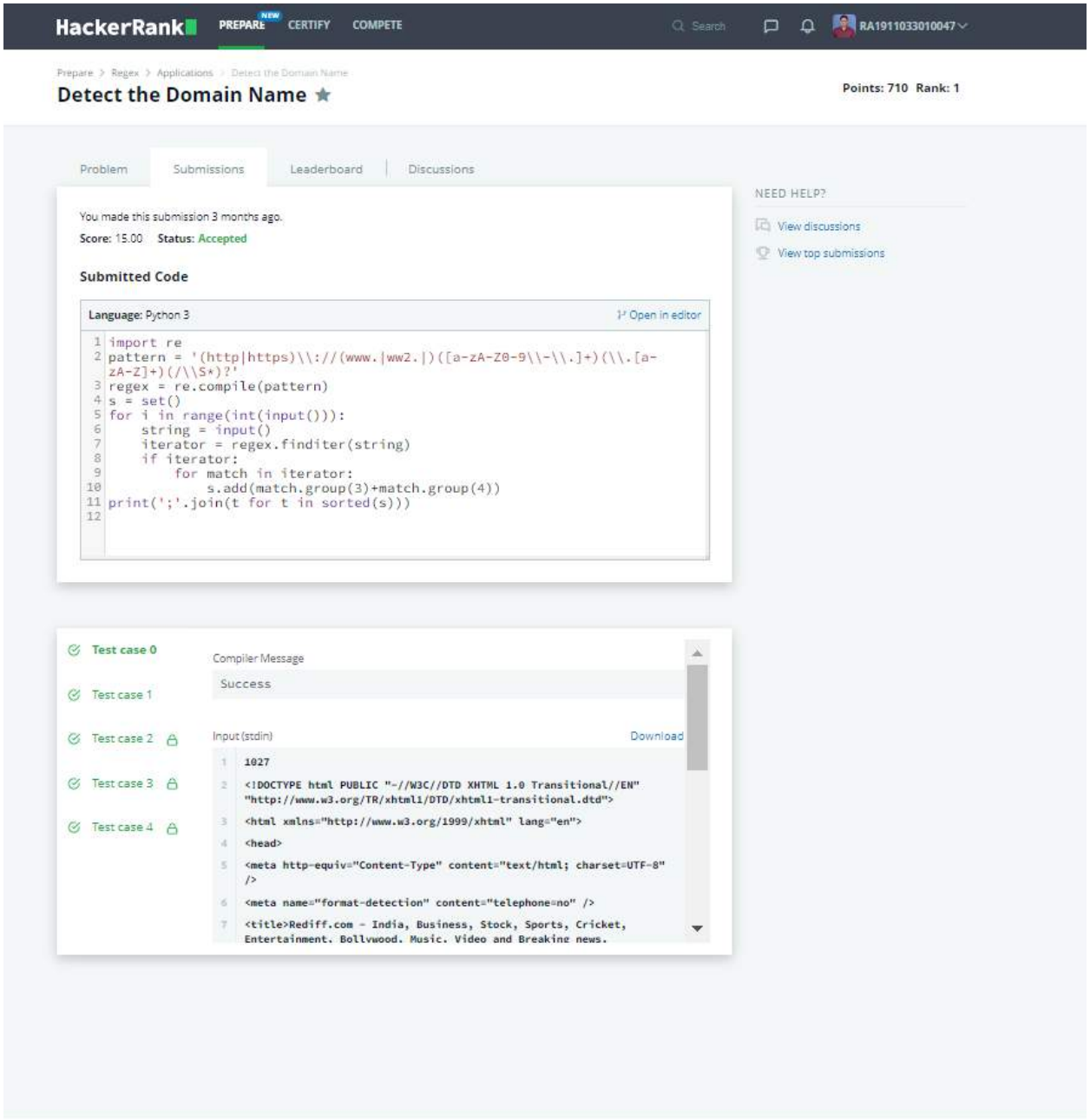
RA1911033010047
KOLLA SESHU KIRAN

HACKERRANK REGEX PROBLEMS(32 - 47)

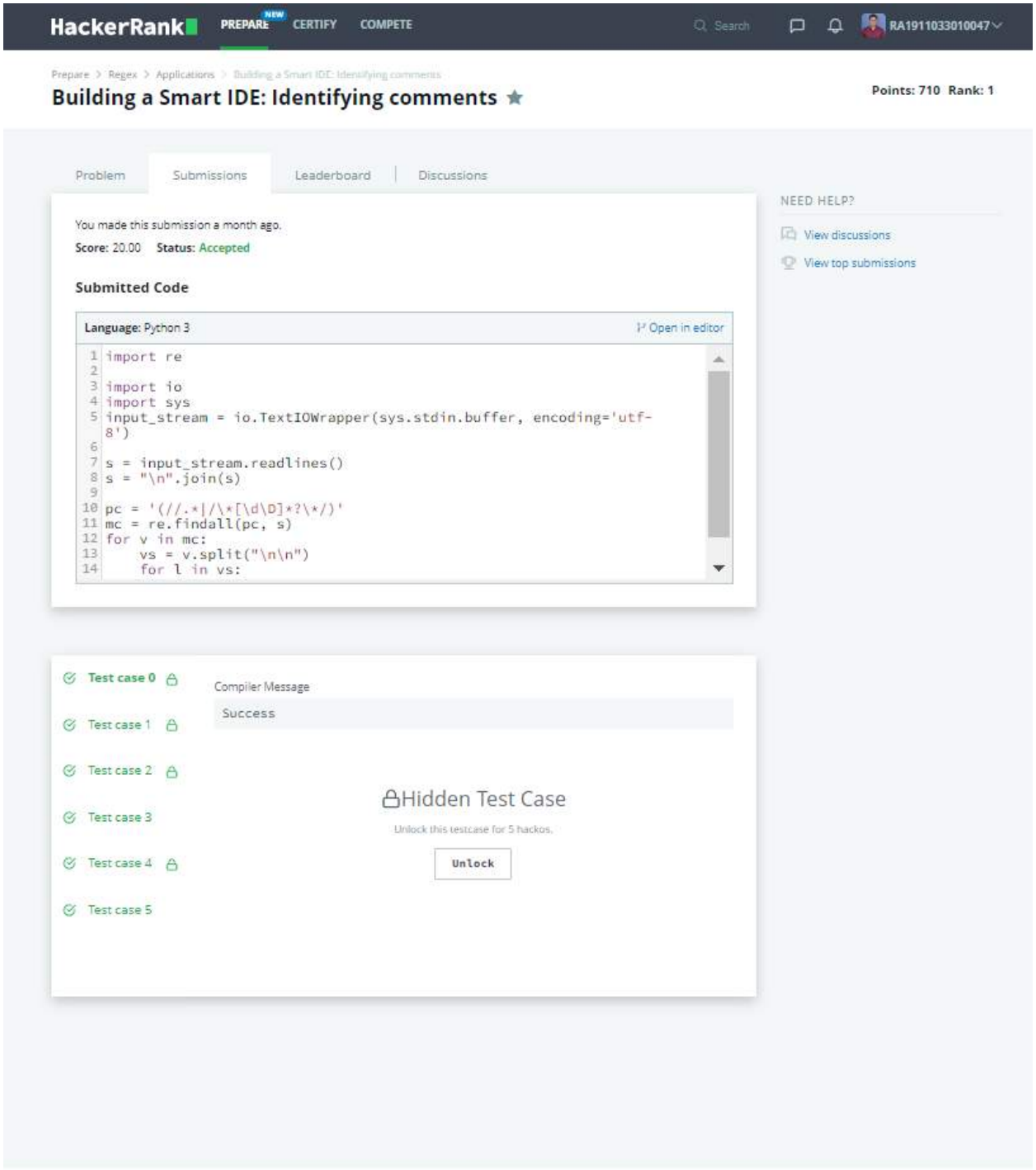
Date: 18-04-2022



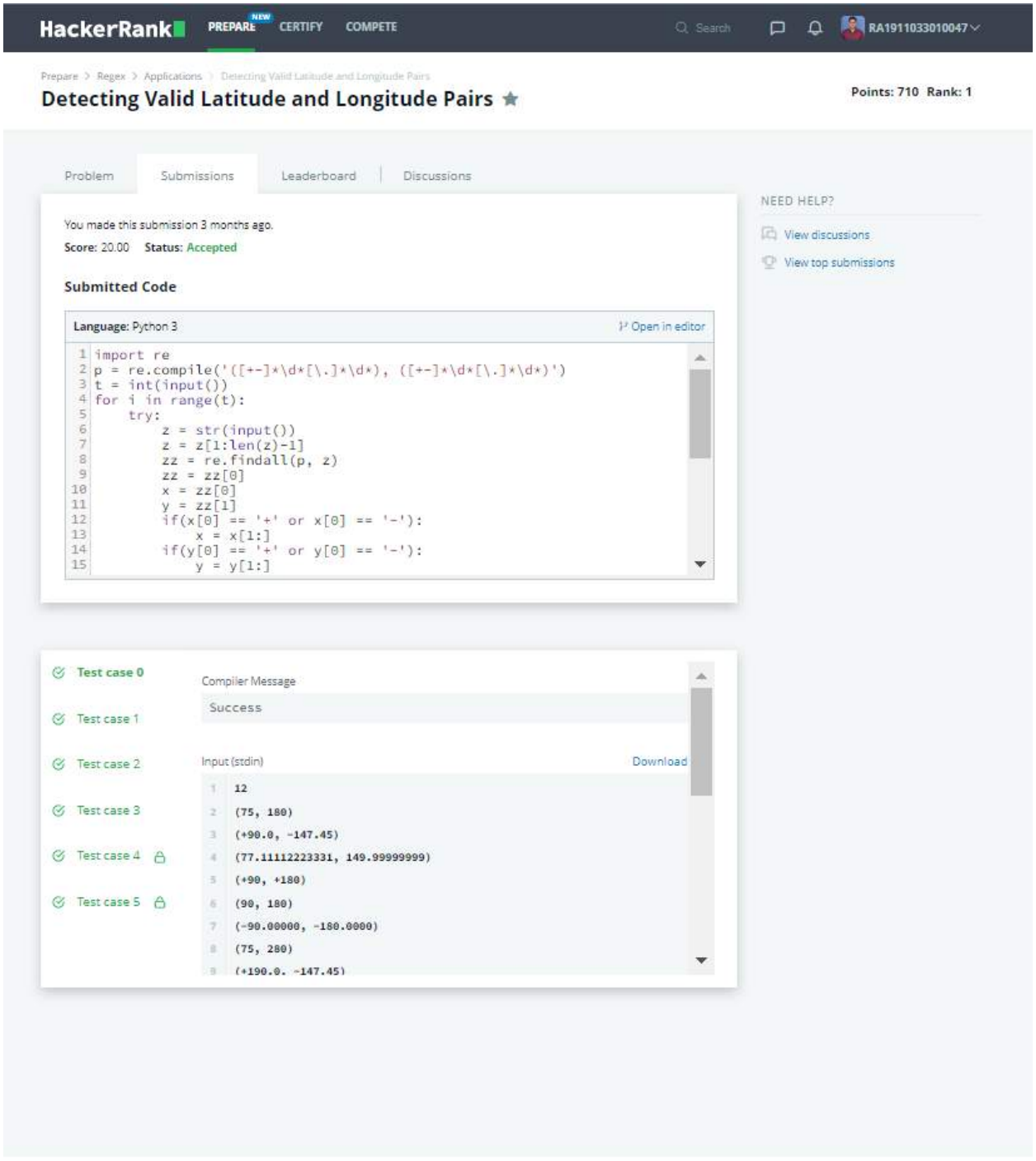
RA1911033010047
KOLLA SESHU KIRAN



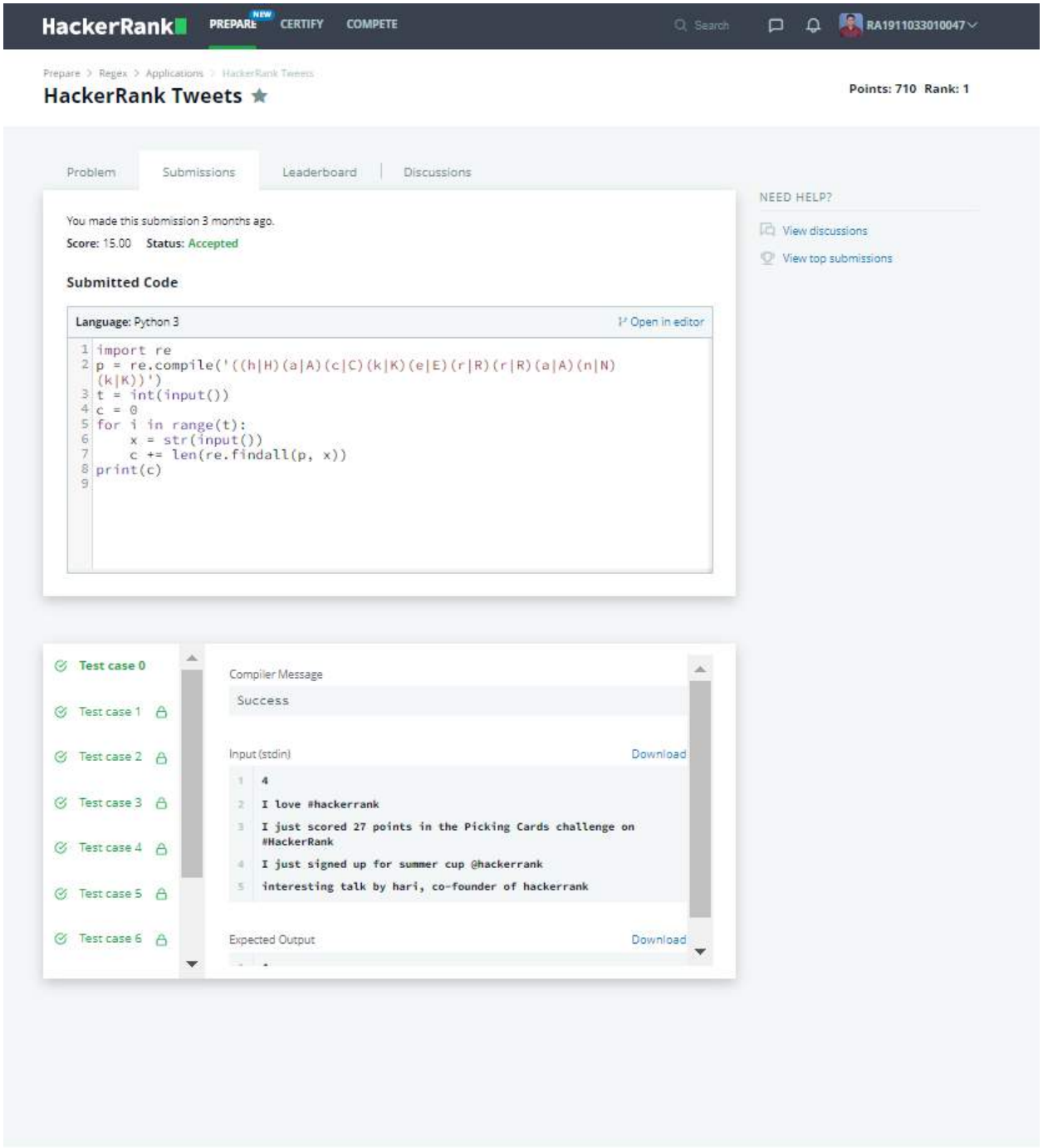
RA1911033010047
KOLLA SESHU KIRAN



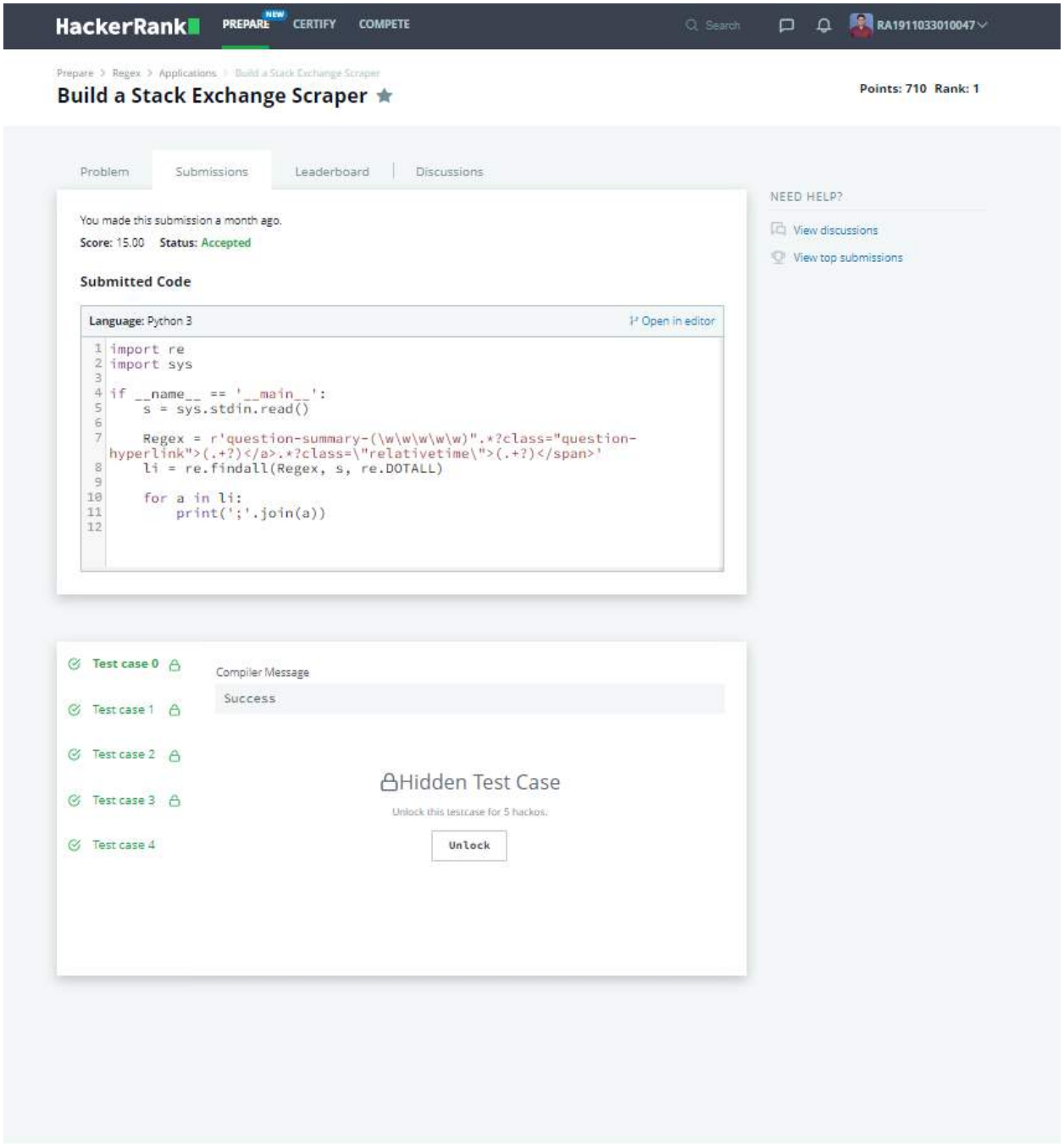
RA1911033010047
KOLLA SESHU KIRAN



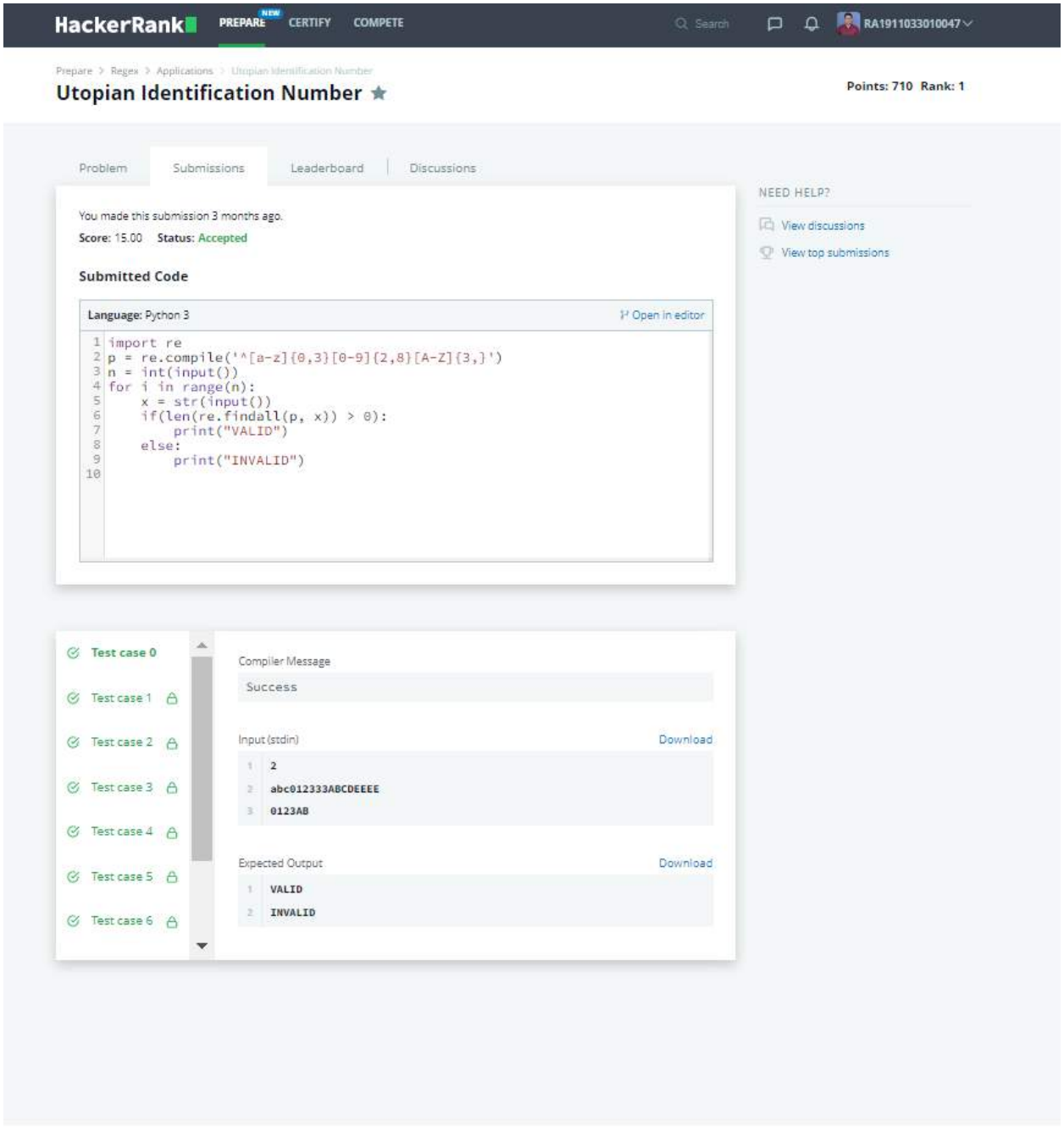
RA1911033010047
KOLLA SESHU KIRAN



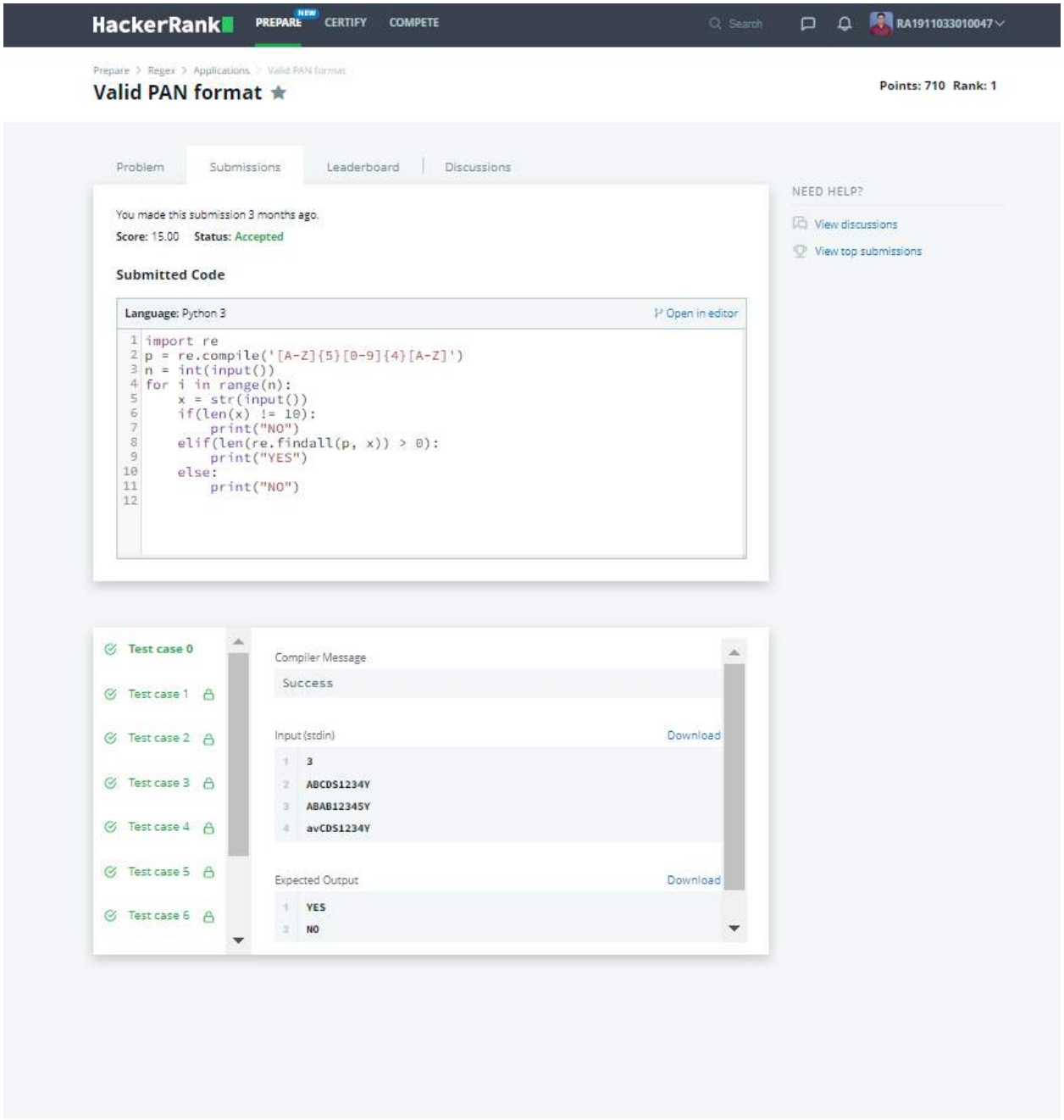
RA1911033010047
KOLLA SESHU KIRAN



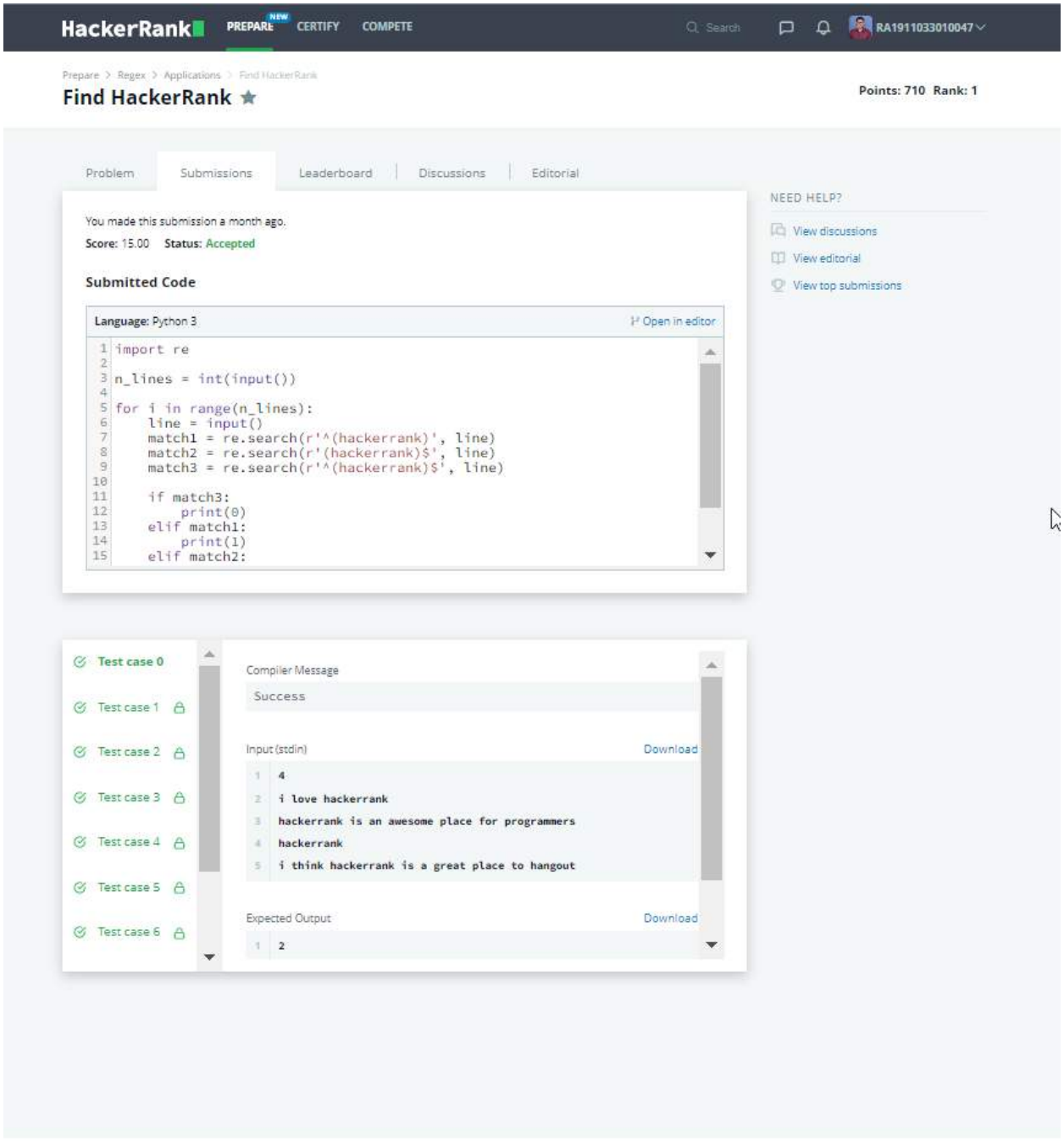
RA1911033010047
KOLLA SESHU KIRAN



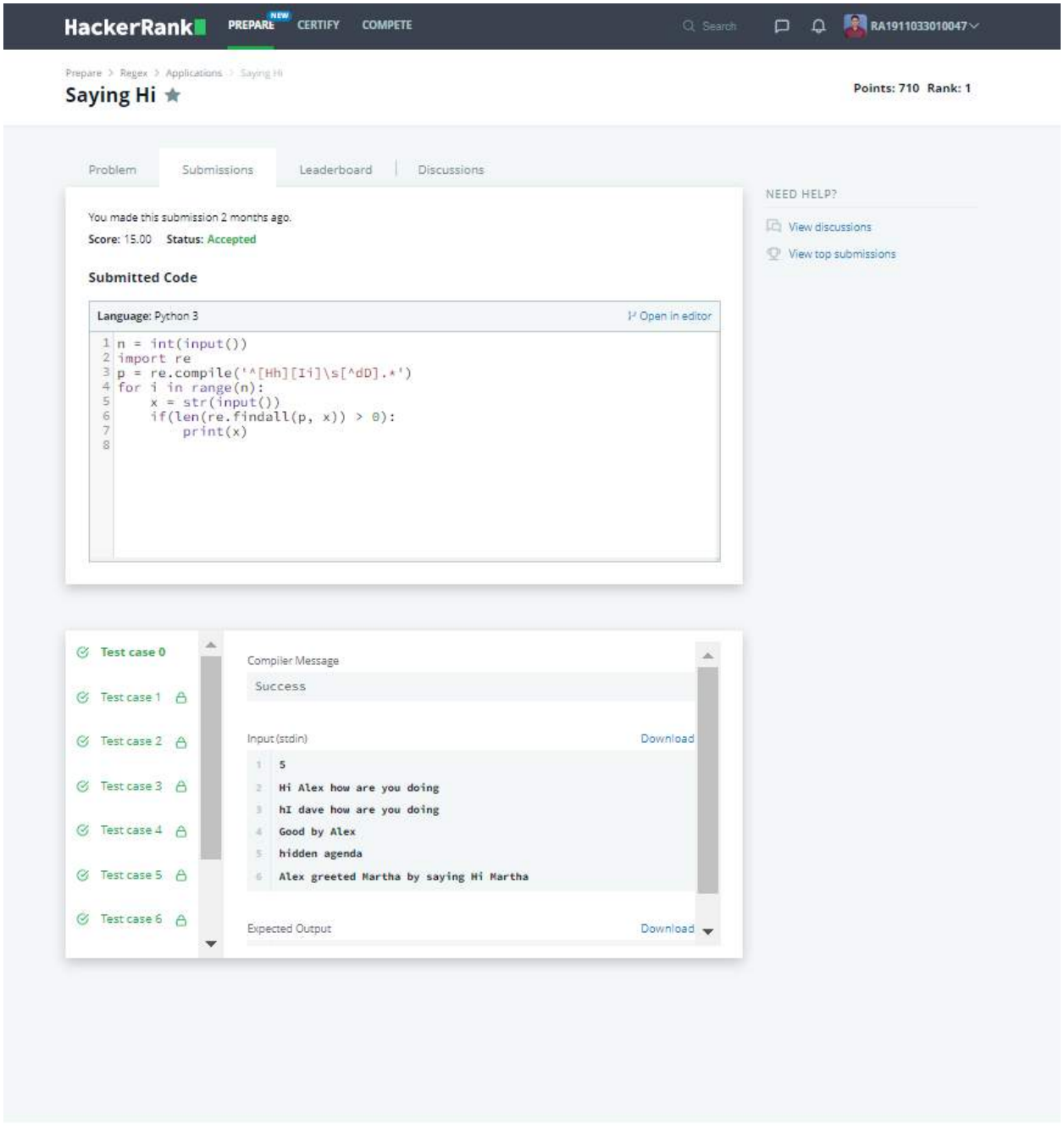
RA1911033010047
KOLLA SESHU KIRAN



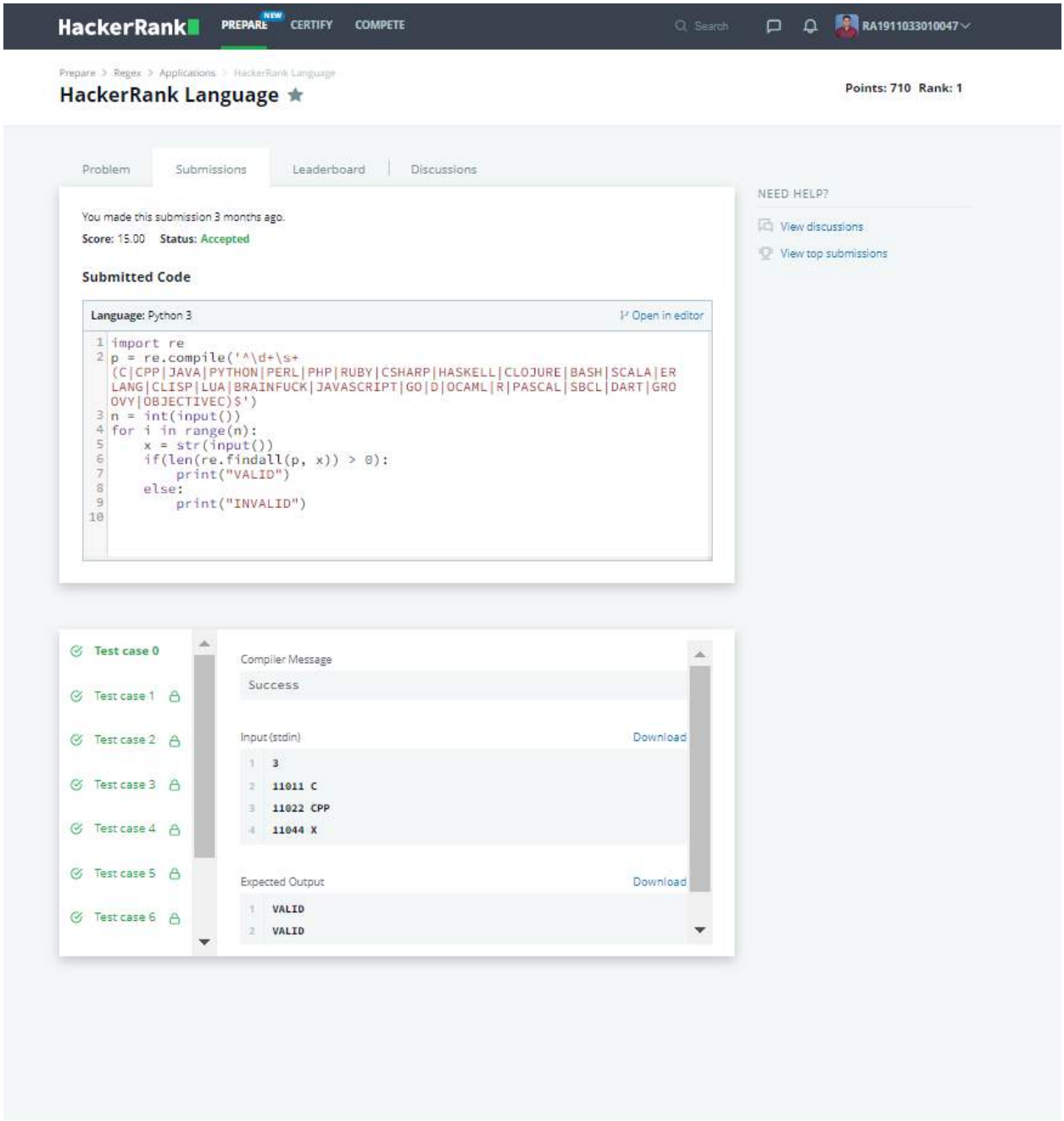
RA1911033010047
KOLLA SESHU KIRAN



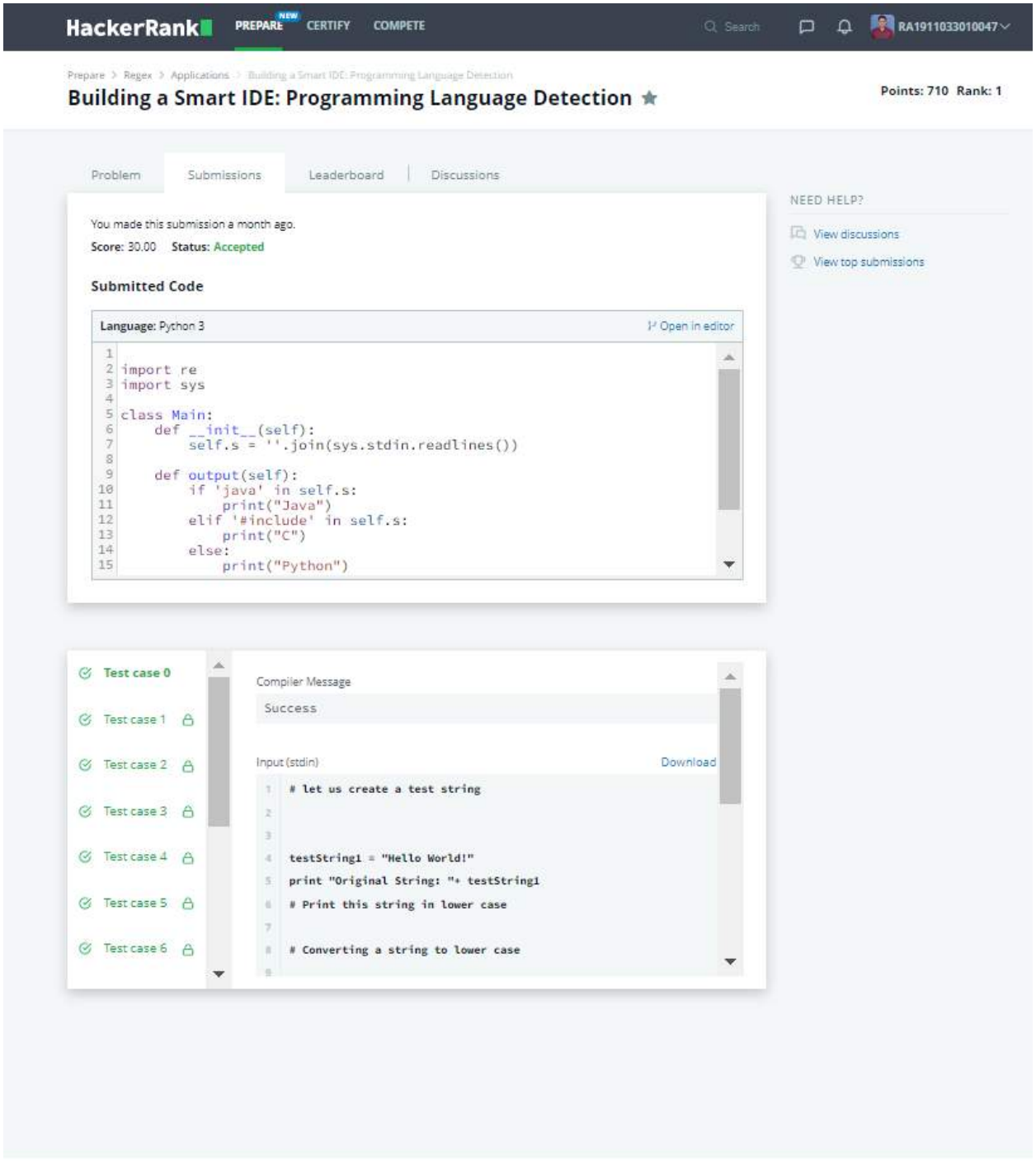
RA1911033010047
KOLLA SESHU KIRAN



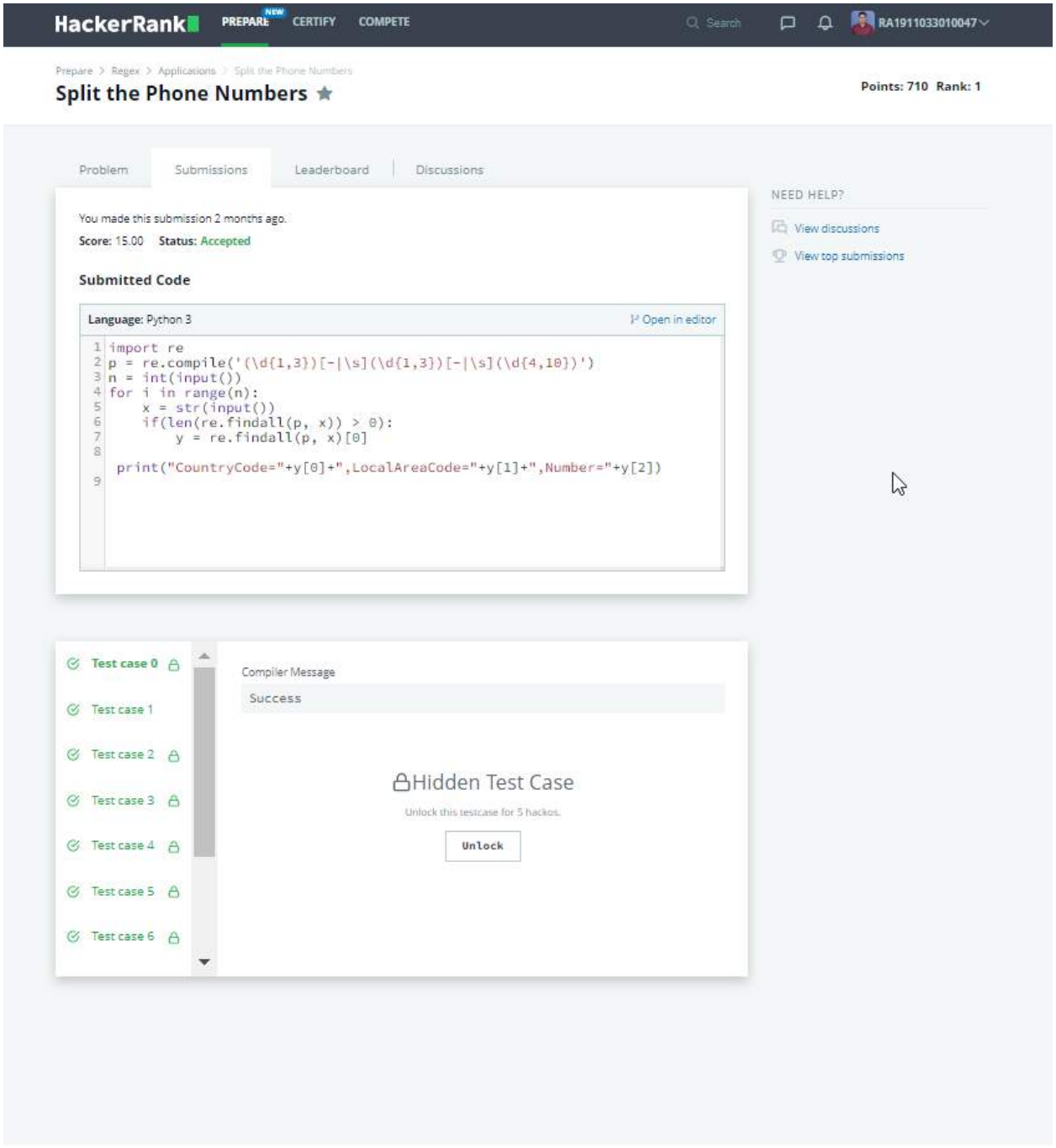
RA1911033010047
KOLLA SESHU KIRAN



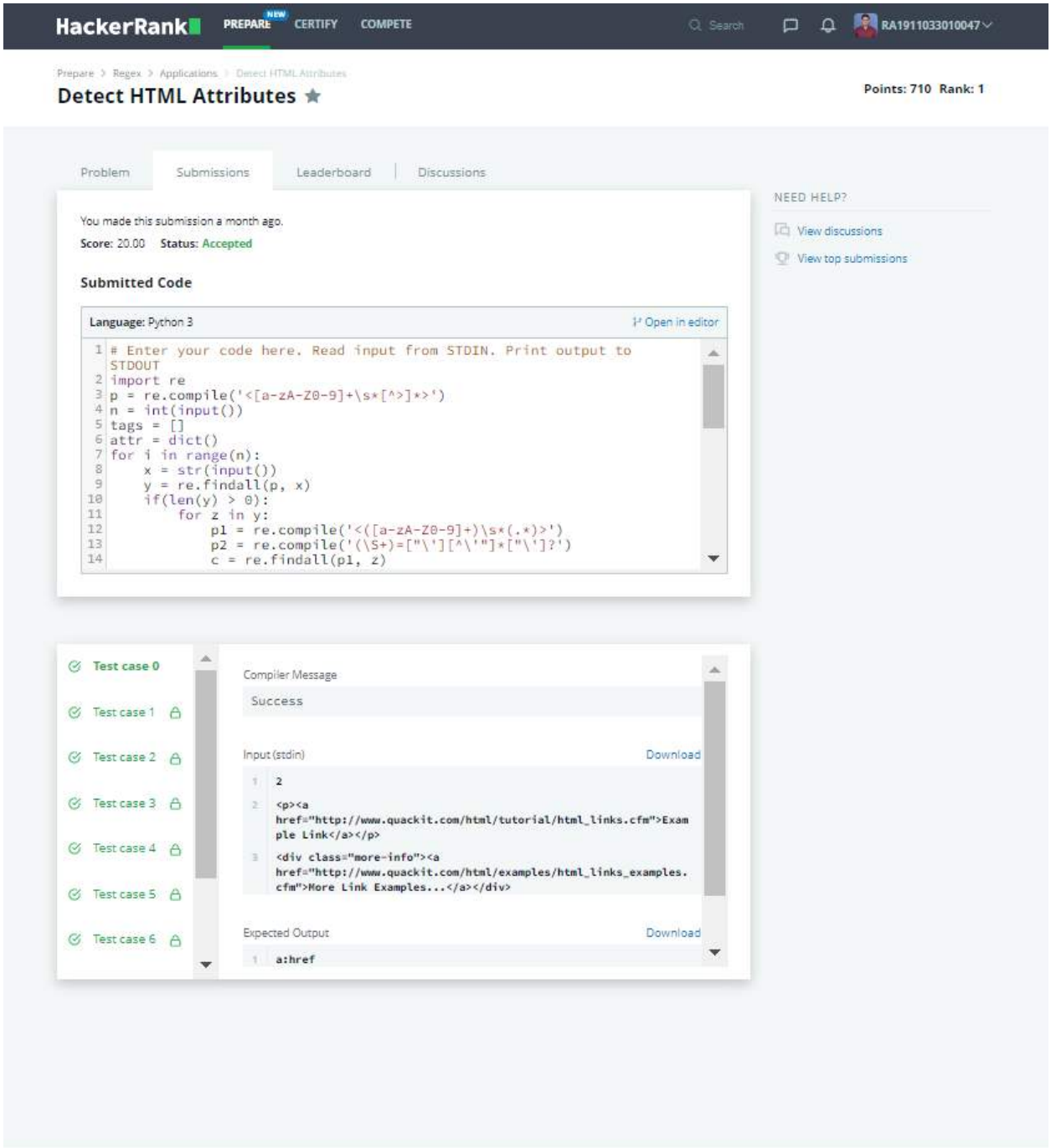
RA1911033010047
KOLLA SESHU KIRAN



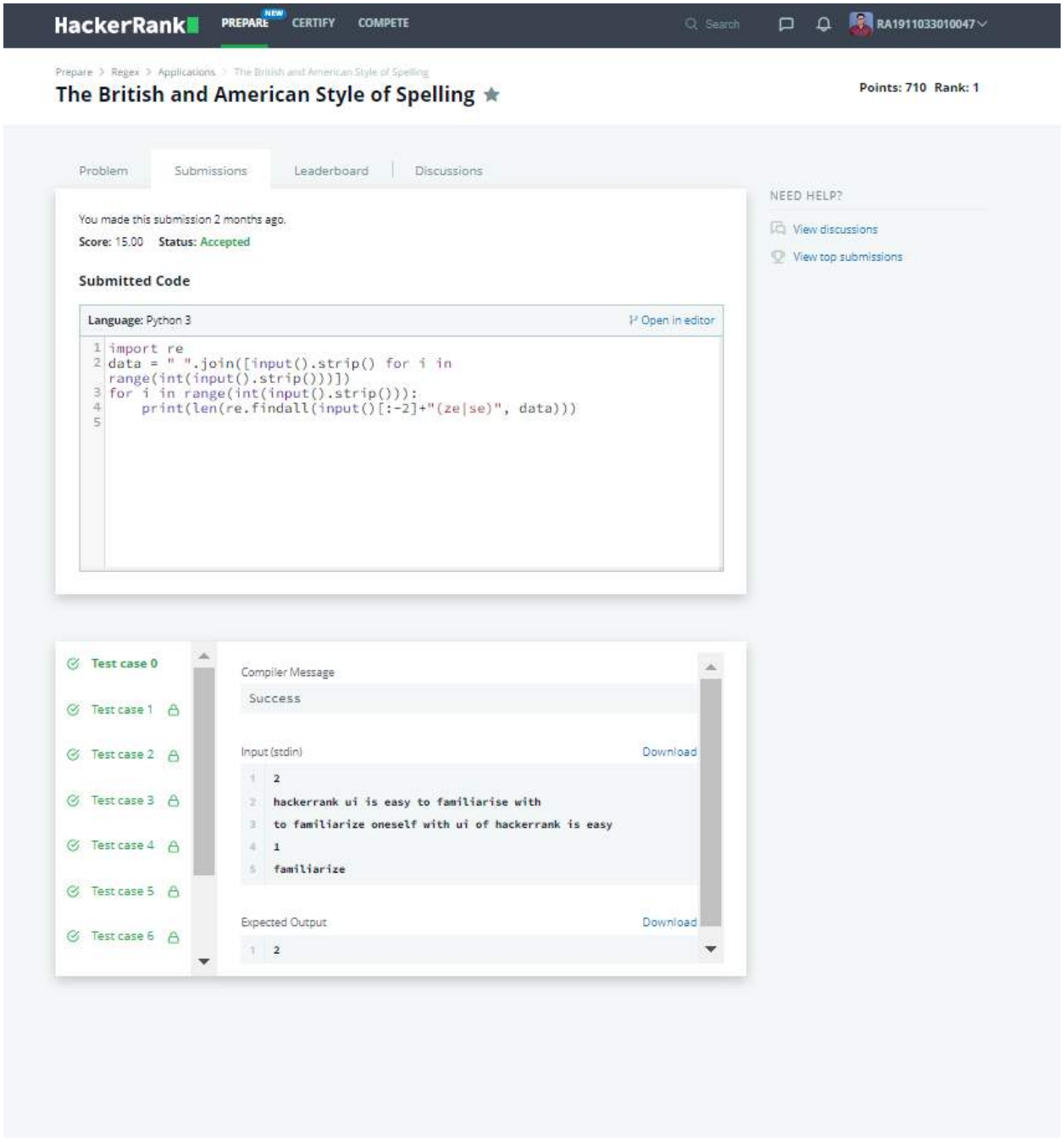
RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN



RA1911033010047
KOLLA SESHU KIRAN