Home works 5 & 6

Policy on plagiarism

This is an individual homework. While you may discuss the ideas and algorithms or share the test cases with others, at no time may you read, possess, or submit the solution code of anyone else (including people outside this course), submit anyone else's solution code to your account on the gradebot, or allow anyone else to read or possess your source code. We will detect plagiarism using automated tools and will prosecute all violations to the fullest extent of the university regulations, including failing this course, academic probation, and expulsion from the university.

Goal

- Programming in Python
- String
- Recursion
    - Tree
    - Graph

Background

- Control flow graph: understand the definition of basic blocks and control flow graphs
- Boolean program, this is optional. You might read this material if you want to know more. For the homework, we defined as below a more restricted syntax of Boolean programs.

Syntax of Matrices

- `Matrix` is defined in Backus-Naur Form (BNF), see BNF Wiki, there does not exist any blank space in a matrix. You have to output the control flow graph of a program in the syntax of `Martix`

```
Matrix::= "[" Rows "]"

    Rows::= Row | Row ";" Rows

    Row::= element | element "," Row

    elment::= 0 | 1
```

Syntax of Boolean Programs

- `Boolean Program` is defined in Extended Backus-Naur Form (EBNF), see EBNF Wiki.
- A Boolean program consists of at least one variable declaration
    - A Boolean program always contain exactly one procedure **main** which is the entry of the

  program, the procedure end by a **return** statement
  - For the sake of simplicity, we assume that all the programs do not have nested `while/if` statements, i.e., in the body of `while`, there is no `while/if` statements, in the body of `if`, there is no `while/if` statements

```
prog  ::= decls proc                 # A program is a list of (global) Boolean
variable declarations followed by the definition of the main procedure

decls ::= decl | decl decls          # Declaration of Boolean variables
decl  ::= "bool" id "=" v ";"        # Declaration of one variable, its value is
`v`, e.g., `bool x=True;`
id    ::= [a-zA-Z0-9][a-zA-Z0-9_]*   # An identifier excluding keywords 'if', 'fi',
'while', 'done', 'return' and 'pass',
v     ::= "True" | "False"           # Each variable in the program has to take a
Boolean value

proc  ::= "main"  "(" ")"  "{" stmts "}"     # Definition of the `main` procedure,
e.g., `main() { x=True; return x;}`

stmts ::= id ":" stmt | id ":" stmt stmts      # Sequence of labeled statements, id
is the label of the statement
stmt  ::= id "=" expr ";"                      # Assignment,  e.g., `x=True; y=x; `
       | "if" "(" expr ")"  stmts  "fi"         # `if` statement like in python,
e.g., `if (x) pass; x=False; fi`
       | "while" "(" expr ")"  stmts  "done"    # `while` statement for loop, e.g.,
`while(x) pass; x=False; done`
       | "return" id ";"                        # For the **evaluate** method,
e.g., `return x;`
       | "pass" ";"                             # Same as `pass` in Python

expr  ::= expr "&" expr          # Logical `and` operator
       | expr "|" expr           # Logical `or` operator
       | "!" expr                # Logical `not` operator, priority exactly follows
the priority of these operators in Python, i.e., `not > and > or`
       | "(" expr ")"
       | v                       # constant "True" | "False"
       | id                      # Boolean variable which must be declared at
"Declaration of Boolean variables"
```

Note that keywords are case sensitive, e.g., if != IF != If, etc.

Description

- Define a class called **program** in which you should implement the following instance methods
- **getCFG** method: outputs the control flow graph of the Boolean program in the syntax of Matrix,
  1. each node corresponds to a basic block which consists of a sequence of statement
  2. each node is named by the label of the first statement of the corresponding basic block
  3. rows and columns of the matrix are in ascending order in the meaning of the names of the nodes

- **evaluate** method outputs the return value of the **main** procedure

Howework 5

- Implement the **getCFG** method

Homework 6

- Implement the **evaluate** method
- The **evaluate** method outputs the string *infinite* if the Boolean program does not terminate, i.e., the Boolean program has an infinite loop

Example

```
s="""
    bool x = True;
    bool y = False;
    bool z = True;
    bool a = True;
    main()
    {
    1:  x= !y;
    2:  z= !x;
    3:  if ( (x & y) | (! z) )
    4:      y= !y;
    5:      pass;
        fi
    6:  x=!y;
    7:  z=!z;
    8:  while ( ( x | y) & (a | z) )
    9:      a=!y;
    10:     y=!z;
        done
    11: return x;
    }
"""
p =  program(s)

p.evaluate()
>>>False

>>>p.getCFG()
[0,0,1,1,0,0;0,0,0,0,0,0;0,0,0,1,0,0;0,0,0,0,1,0;0,1,0,0,0,1;0,0,0,0,1,0]
```

- The control flow graph is shown in the figure
- `[1,2,3]`, `[4,5]`, `[6,7]`, `[8]`, `[9,10]`, `[11]` are basic blocks. The nodes are in the control flow graph are `1`, `4`, `6`, `8`, `9`, `11`.

Matrix representing of this graph is

```
     1  11 4  6  8  9     # nodes represent basic blocks
   -------------------
1  | 0  0  1  1  0  0 |   # edges (1,4), (1,6)... between nodes denote the flow of
program
11 | 0  0  0  0  0  0 |
4  | 0  0  0  1  0  0 |
6  | 0  0  0  0  1  0 |
8  | 0  1  0  0  0  1 |
9  | 0  0  0  0  1  0 |
   -------------------
```

```
s="""
    bool x = True;
    bool y = False;
    bool z = True;
    bool a = True;
    main()
    {
    1:  x= !y;
    2:  z= !x;
    3:  if ( (x & y) | (! z) )
    4:      y= !y;
    5:      pass;
        fi
    6:  x=!y;
    7:  z=!z;
    8:  while ( ( x | y) & (a | z) )
    9:      a=!y;
    10:     y=z;
        done
    11: return x;
    }
"""
p =  program(s)


p.evaluate()
>>>infinite
```

Check in

- check in `program.py` file into Online Judge