Homework 1
CSE4051 Program Verification, Fall 2025
Woosuk Lee
due: 10/1(Wed), 23:59

- You must use Python 3 for this assignment.
- Submit one python file per problem via the submission system in the course website. Make sure that your files run without errors.
- You are not allowed to use any external libraries except for Z3 for Python.
- All problems can be solved by encoding them as SAT or MaxSAT problems (reasoning over values other than Boolean values (e.g., integers) is not allowed).

Exercise 1 DIMACS format is a standard format for specifying SAT problems. A DIMACS file consists of a preamble line followed by lines representing clauses. The preamble line starts with the letter 'p', followed by the problem type (usually 'cnf' for conjunctive normal form), the number of variables, and the number of clauses.

Each clause is represented on a separate line, with positive integers representing variables and negative integers representing negated variables.

Each clause ends with a '0'. For example, the clause $x_1 \vee \neg x_2 \vee x_3$ would be represented as "1 -2 3 0".

Here is an example of a DIMACS file representing a simple SAT problem:

```
p cnf 3 2
1 -2 0
-1 2 3 0
```

This file specifies a SAT problem with 3 variables (namely, x_1, x_2, and x_3) and 2 clauses. The first clause is $x_1 \vee \neg x_2$ and the second clause is $\neg x_1 \vee x_2 \vee x_3$.

Write a python program that reads a DIMACS file and outputs the number of all possible assignments of variables that satisfy all the clauses in the file.

You can assume that the input file is named "input.dimacs" and is located in the same directory as your program.

Here is the starter code:

```
from z3 import *
def read_dimacs(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()

    # fill here

    return num_vars, clauses

def count_satisfying_assignments(num_vars, clauses):
    # fill here
    return count

if __name__ == "__main__":
    num_vars, clauses = read_dimacs("input.dimacs")
    result = count_satisfying_assignments(num_vars, clauses)
    print(result)
```

Exercise 2 There are n switches controlling a single light. The light is on if (1) an odd number of switches are on, (2) two adjacent switches are not both on, and (3) the first or the last switch is on. Write a program that takes n as input and outputs the number of all possible assignments of the switches that turn on the light. The input as a single integer n will be given via standard input.

HINT: You may use Xor operator in Z3 to express the parity constraint (1).

Here is the starter code:

```
from z3 import *
def count_light_on_configurations(n):
    # fill here
    return count

if __name__ == "__main__":
    n = int(input().strip())
    result = count_light_on_configurations(n)
    print(result)
```

Exercise 3 Recall the seat assignment problem discussed in the class. In the class, we considered the case where three people and three seats are given. Now, consider the general case where n people and n seats are given. There are three kinds of constraints: (1) person i cannot sit in seat j, (2) person i must sit in seat j, and (3) person i and person j cannot sit next to each other. The indices of people and seats are 0-based (i.e., from 0 to $n-1$). Write a program that takes n and a set of constraints as input and outputs the number of all possible assignments of people to seats that satisfy all the constraints. The input is of the following format:

```
[the number of people/seats n]
1 i j (person i cannot sit in seat j)
...
2 i j (person i must sit in seat j)
...
3 i j (person i and person j cannot sit next to each other)
...
```

For example, the following input specifies the case where there are 4 people and 4 seats, person 1 cannot sit in seat 2, person 2 must sit in seat 3, and person 3 and person 4 cannot sit next to each other.

```
4
1 0 1
2 1 2
3 2 3
```

You can assume that the input will be given as a file named "input.txt" in the same directory as your program.

Here is the starter code:

```python
from z3 import *
def read_input(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()
    # fill here
    return n, constraints

def count_valid_assignments(n, constraints):
    # fill here
    return count

if __name__ == "__main__":
    n, constraints = read_input("input.txt")
    result = count_valid_assignments(n, constraints)
    print(result)
```

Exercise 4 Suppose you are organizing a tennis league with n matches and $2n$ players, where each player has a skill level represented by an integer. You want to schedule matches between players such that the difference in skill levels between any two players in a match is less than or equal to a given threshold k. Write a program that takes as input the number of matches n, the skill levels of the $2n$ players, and the threshold k, and outputs the number of all possible ways to schedule the matches satisfying the skill level constraint. The input will be given in a file named "input.txt" in the following format:

```
[n]
[skill_level_1] [skill_level_2] ... [skill_level_2n]
[k]
```

For example, the following input specifies the case where there are 2 matches (4 players), the skill levels of the players are 1, 3, 5, and 7, and the threshold is 2.

```
2
1 3 5 7
2
```

Here is the starter code:

```python
from z3 import *
def read_input(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()
    #fill here
    return n, skill_levels, k

def count_match_schedules(n, skill_levels, k):
    # fill here
    return count

if __name__ == "__main__":
    n, skill_levels, k = read_input("input.txt")
    result = count_match_schedules(n, skill_levels, k)
    print(result)
```

Exercise 5 Given a directed graph, a starting node, an ending node, and a set of constraints on the nodes, write a program that finds the shortest path from the starting node to the ending node that satisfies all the constraints. The constraints can include: (1) certain nodes must be included in the path, (2) certain nodes must be excluded from the path, (3) certain node must be visited if another node is visited, and (4) certain node cannot be visited if another node is visited.

The indices of nodes are 0-based (i.e., from 0 to n-1). The starting node is 0 and the ending node is n-1. You may assume that there is no cycle in the graph.

The input will be given in a file named "input.txt" in the following format:

```
[n] [m] (number of nodes and edges)
[u1] [v1] (edge from u1 to v1)
...
[um] [vm] (edge from um to vm)
1 [node1] (node1 must be included)
1 [node2] (node2 must be included)
...
2 [node3] (node3 must be excluded)
2 [node4] (node4 must be excluded)
...
3 [node1] [node2] (if node1 is visited, node2 must be visited)
...
4 [node1] [node2] (if node1 is visited, node2 cannot be visited)
...
```

For example, the following input specifies a graph with 4 nodes and 4 edges, where there is an edge from node 0 to node 1, from node 1 to node 2, from node

0 to node 2, and from node 2 to node 3. The starting node is 0 and the ending node is 3. If node 0 is visited, node 1 must be visited, and if node 1 is visited, node 2 cannot be visited.

```
4 4
0 1
1 2
0 2
2 3
3 0 1
4 1 2
```

The output should be a sequence of integers representing the nodes in the shortest path from the starting node to the ending node that satisfies all the constraints. If no such path exists, the program should output "No path". For example, for the above input, the output should be:

```
No path
```

HINT: You may encode the problem as a partial weighted MaxSAT problem.

Here is the starter code:

```python
from z3 import *
def read_input(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()
    # fill here
    return n, edges, constraints

def find_shortest_path(n, edges, constraints):
    start = 0
    end = n - 1
    # fill here

if __name__ == "__main__":
    n, edges, constraints = read_input("input.txt")
    result = find_shortest_path(n, edges, constraints)
    print(result)
```

Exercise 6 You are planning your course schedule for the next semester. You are given a graph representing course prerequisites, a list of courses you have already taken. You should take at least 1 course in the next semester, and all courses are offered in the next semester. Write a program that computes the number of all possible combinations of courses you can take in the next semester while satisfying the prerequisite requirements.

The indices of courses are 0-based (i.e., from 0 to n-1). The input will be given in a file named "input.txt" in the following format:

```
[n] [m] (number of courses and prerequisite relations)
[u1] [v1] (course u1 is a prerequisite for course v1)
```

```
...
[um] [vm] (course um is a prerequisite for course vm)
[k] (number of courses already taken)
[c1] [c2] ... [ck] (list of courses already taken)
```

For example, the following input specifies a scenario with 4 courses and 3 prerequisite relations, where course 0 is a prerequisite for course 1, course 1 is a prerequisite for course 2, and course 0 is a prerequisite for course 3. You have already taken course 2 (courses 0 and 1).

```
4 3
0 1
1 2
0 3
2
0 1
```

Here is the starter code:

```python
from z3 import *
def read_input(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()
    # fill here
    return n, edges, taken_courses

def count_course_combinations(n, edges, taken_courses):
    # fill here
    return count

if __name__ == "__main__":
    n, edges, taken_courses = read_input("input.txt")
    result = count_course_combinations(n, edges, taken_courses)
    print(result)
```

Exercise 7 Recall the graph coloring problem discussed in the class. In the class, we considered the case where a graph is colored with three colors. In this exercise, given a graph, compute the minimum number of colors needed to color the graph such that no two adjacent nodes have the same color. The indices of nodes are 0-based (i.e., from 0 to n-1). The input will be given in a file named "input.txt" in the following format:

```
[n] (number of nodes)
[u1] [v1] (edge from u1 to v1)
...
[um] [vm] (edge from um to vm)
```

For example, the following input specifies a graph with 4 nodes and 4 edges, where there is an edge from node 0 to node 1, from node 1 to node 2, from node 0 to node 2, and from node 2 to node 3.

```
4
0 1
1 2
0 2
2 3
```

Here is the starter code:

```python
from z3 import *
def read_input(file_path):
    with open(file_path, 'r') as f:
        lines = f.readlines()
    # fill here
    return n, edges

def min_colors(n, edges):
    # fill here

if __name__ == "__main__":
    n, edges = read_input("input.txt")
    result = min_colors(n, edges)
    print(result)
```