Shraddha Hegde, Siya Sonpatki, Jamie Liu

Description

This problem involves the Blackbox solution for solving 2-SAT problem, provided in this GitHub repository: https://github.com/arunptl100/SAT-Solver/blob/master/sat-solver.py

Otherwise, the main task to complete for this problem was to parse the inputs in a meaningful way and then translate the inputs to a form translatable by the given Blackbox solution.

After establishing the code from GitHub earlier, the first thing with did in our main function was to parse each instance of the light switches in their maps into a readable form for the 2-SAT solver. We parse in the file and arrange it into a 2-D array format with individual arrays with the following attributes:

- The 2D array is has n rows (number of lights)
- Each beginning with the state of the lightbulb (0, 1) followed by the switches associated with that lightbulb.

Once the file is parsed into that format, we reduce the arrays into a 2-SAT pronlem via. The following:

- Assuming the light is on (and there's two switches), we form the clauses (A V B) ∧ (!A V !B).
- If the light is off, we create the constraints: (!A V B) ∧ (A V !B)

We format the given array through the build_cnf_formula( ) by combining all the individual clauses into a long string of combined constraints. From there, we're able to plug in the long constraint into the two_cnf_solver and get the correct inputs.

Time Complexity

Deconstructing the Algorithm:

- ***parse_instance* function - O(num_bulbs x num_switches)**
    - Description: parses input to build circuit for switches and bulbs
    - Complexity:
        - Reading data for bulb states and switch connections – O(*num_bulbs*) + O(*num_switches x avg_connections*)
        - Worst case: each switch connects to every bulb – O(*num_switches x num_bulbs)*
- ***build_cnf_formula* function - O(num_bulbs)**

    - Description: converts the circuit into a 2-CNF formula
    - Complexity:
        - Iterates over each bulb to sum to two clauses – O(*num_bulbs*)
        - Each individual clause – O(1)
        - Total complexity = O(num_bulbs) * O(1) = O(num_bulbs)
- ***two_sat_solver*** **function - O(num_switches + num_bulbs)**
    - Description: determines whether the 2-CNF formula is satisfiable – uses a directed graph and Kosaraju's algorithm to find SCCs
    - Complexity:
        - Building directed graph – O(*num_clauses*) = O(*num_bulbs*)
        - Kosaraju's Algorithm – O(*num_switches* + *num_bulbs*) total runtime:
            - First DFS – O(|V| + |E|), |V| = # of variables and |E| = # of edges
            - Transposing the graph - O(|V| + |E|)
            - Second DFS on transposed graph - O(|V| + |E|)
            - Total runtime = O(|V| + |E|) = O(*num_switches* + *num_bulbs*)
- ***find-contradiction function -*** **O(num_bulbs)**
    - Description: determines if there are any contradictions within the SCCs
    - Complexity:
        - Iterates through each component and literal within the component – O(*num_bulbs*)
- ***can_turn_off_lights*** **(main) function - O(num_switches + num_bulbs)**
    - Description: processes two instances and writes results to output.txt file
    - Complexity: for *each* instance, the dominant runtime is a result of two_sat_solver – O(*num_switches* + *num_bulbs)*

Overall Time Complexity:

- For two instances:

    O(2 x (n*um_switches* + *num_bulbs*))

    = O(*num_switches* + *num_bulbs*)

- Worst-case:

    O(2 x (*num_switches x num_bulbs*)

    = **O(*num_switches* + *num_bulbs*)**

Proof of Correctness

To show that our reduction works, we need to prove that the 2-SAT instance created by our method is satisfiable if and only if there is a way to toggle the switches so that all the lights turn off. This gives us two possible parts:

1. If it's possible to toggle the switches and turn off all the lights, then the 2-SAT instance is satisfiable-
   - If we have a valid way to toggle the switches so that all lights are off. For each light Li , connected to two switches S1 and S2 , the light turns off when the number of toggles between S1 and S2 is even.
     - This happens if both switches are in the same state (either both toggled or both untoggled).
     - In terms of logic, this means S1 =S2 .
   - In the 2-SAT problem, we represent this condition using two clauses:
     - (¬S1 ∨ S2): If S1 is untoggled, S2 must also be untoggled.
     - (¬S2 ∨ S1): If S2 is untoggled, S1 must also be untoggled.
   - Together, these clauses guarantee that S1 and S2 are in the same state, which matches the condition for the light being off.
   - Since all the lights are off in this configuration, the truth values of S1 ,S2 ,... satisfy all the clauses in the 2-SAT instance. Therefore, the 2-SAT formula is satisfiable.
2. If the 2-SAT instance is satisfiable, then it's possible to toggle the switches to turn off all the lights.
   - Now, if the 2-SAT formula is satisfiable. This means we have a truth assignment (true/false values) for the switches S1 ,S2 ,... that satisfies every clause in the formula.
     - For a clause like (¬S1 ∨ S2), the truth values ensure that $S1S_1$S1 and S2 meet the condition for turning off the light.
   - Since all the clauses in the formula are satisfied, this truth assignment guarantees that for every light Li , the switches S1 and S2 toggle the light an even number of times. This means all lights will be off.
   - By toggling a switch if its value is true, we can turn off all the lights.

   Conclusion of proof:
   - If there's a way to toggle the switches and turn off all the lights, the 2-SAT instance is satisfiable.
   - If the 2-SAT instance is satisfiable, there's a way to toggle the switches to turn off all the lights.

- This proves that the reduction works.