CSE 431/531 Analysis of Algorithms
Project: Random walk on graph with states

Chen Xu

Due Date: August 15th, 2023 23:59 PM EST.

1 Problem statement

In this project, we will write code to solve a puzzle. We are given a 2-regular directed graph $G = (V, E)$ (Recall that every vertex in a 2-regular directed graph has exactly two outgoing edges). On every vertex, there is a switch that has two states: **ON**/**OFF**. On two outgoing edges of every vertex, one edge has label **a** while the other has label **b**. For this project, self-loop edges (v, v) that point to the vertex v itself are also considered.

Initially a robot is put on some vertex. The robot takes our command and moves along the edge. We send the edge label to move the robot. When the robot **enters a vertex through an edge** it toggles the switch on the destination vertex. Here is an illustration:

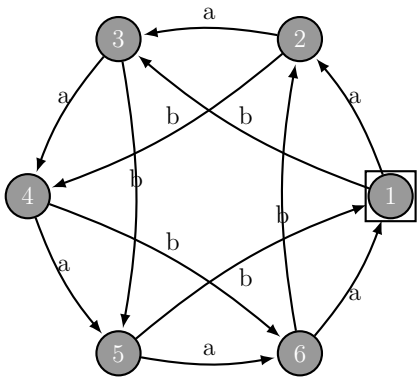

Figure 1: The robot is placed at node 1

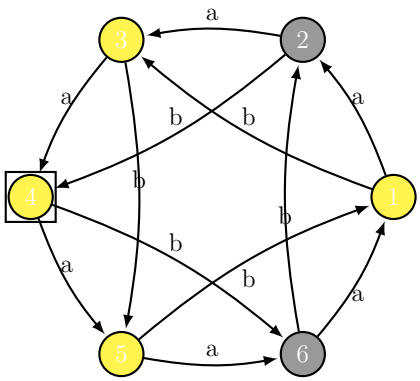

Figure 2: After the input *aabbab*, the vertices $1, 3, 4, 5$ are ON and $2, 6$ are OFF, the robot is at vertex 4

Complete the following programming tasks using C / C++, JAVA, C#, Python, Matlab. The console output is required while the file output is optional. We suggest taking some input from a txt file because some test cases can be huge. The difficulty of the programming tasks are rated with stars:

1. ★ Implement the basic *SGraph* class that contains the label information of the 2-regular digraph and the ON / OFF switches on the vertices. This should include some proper constructors. You can hard-code the instances but constructing from a string is recommended. A reset function is recommended too to reset all switches to OFF states.

2. An enumerator is a class that enumerates elements by some sequence. It typically has two interfaces:

   (1) SetCurrent(current) – It sets the states inside the enumerator object so that it knows what is the current element.

   (2) Next() – It returns the next one in the sequence and increment the states by one.

   We would like to enumerate all of the 2-regular digraphs that we are interested. For example, we want to study the property on all 2-regular digraph of vertex size m.

   (a) ★★ Implement an efficient enumerator for all 2-regular digraph with different labels.

(b) ★★★★ Implement an efficient enumerator for all **Strongly-Connected** 2-regular digraph with different labels.

(c) ★★★★ Implement an efficient enumerator for all strongly connected 2-regular digraphs with each vertex having in-degree 2 too.

(d) ★★★★ Implement an efficient enumerator for all strongly connected 2-regular digraphs with each vertex having in-degree 2 and one incoming edge marked with a and the other incoming edge marked with b.

3. ★★ Implement a public function $Move(initpos, input)$ in *SGraph* class. The *initpos* is the starting vertex of the robot. The *input* is the string of sequential edge labels. After executing this function the *SGraph* object should contain the correct outcome of the movement i.e. the states of the switches. The return value is the final position of the robot.

4. (a) ★★★ Implement a public function $Count(initpos, dest, n)$ in *SGraph* class. The *initpos* is the starting vertex of the robot. The *dest* is the destination vertex. We want to count the number of different commands of length n that move the robot from initpos to dest. The return value is that number.

   (b) ★★★★ Implement a public function $SCount(initpos, dest, n, states)$ in *SGraph* class. The *initpos* is the starting vertex of the robot. The *dest* is the destination vertex. The *states* is an 0/1 array of specific ON/OFF states that we are interested (size is the number of vertices). We want to count the number of different commands of length n that move the robot from initpos to dest **and** with the final switches equal to the given states exactly. The return value is that number.

5. ★★★ Implement a public function $Solve(initpos, states)$ in *SGraph* class. The *initpos* is the starting vertex of the robot. The *states* is the 0/1 array of the interested ON/OFF states of the switches. This function returns one of the shortest commands that move the robot to toggle the switches to match the *states*.

6. ★★★★★ Implement a public function $FindHardest(m)$ in *SGraph* class. It returns the hardest instance of size m of **2-regular strongly**

connected di-graph with a starting vertex and a possible 0/1 assignment of switches such that the **shortest commands** that move the robot to reach that matching switch states is **maximized**.

You can use the enumerator that we have implemented to find the instance. We are curious about the case $m = 4$, what is the hardest instance? Output it somewhere in a readable .txt file.

For each numbered tasks we ask you to complete ★★★ and below. While ★★★★ and above are bonus. Submit your code in .zip file and write an .md/.txt file on how to run your code.