

Programming Assignment

Automata Theory Monsoon 2022, IIIT Hyderabad

August 17, 2022

General Instructions: Read the input and output formats very carefully since the evaluation will be **scripted**. The goal is not to stress test your code but to see if you have understood the underlying concepts used in the implementation. There will be an evaluation/viva after the submission, and a part of the grade would depend on it.

Language constraints: The submission for Q1 must be in Python. You will have to read/write from both a file and stdout. For any further use of other libraries please refer to one of the TAs for approval. Usage of any other language will lead to no evaluation.

1 Cellular Automata

1. [15 points] **Simulating Cellular Automaton**
Due Date: **29th August, 2022 11:59PM**

Given a pattern on a grid, come up with the rules, initial state and the number of iterations for a cellular automaton that will reach the pattern. The cellular automaton you are designing is constrained on a square grid with binary values. Each cell will have 8 neighbours and can have the value 0 or 1.

1.1 Rules

The evolution of the state will happen based on the rules that you will design. The rules must be of the following form:

$$s_{i+1}^a = f(s_i^a, n_i^{a1}, \dots, n_i^{a8}) \quad (1)$$

Here s_i^a is the value of a -th cell at the i -th iteration and n_i^{aj} is the value of j -th neighbour of the a -th cell at i -th iteration. Note that the function f must be Boolean in nature. When programming, it is recommended that you write this function in a different file since you will have to run it on different rules.

1.2 Initial State

The initial state must be described by a **file** that will contain the dimensions of the grid and a list of grid coordinates that will be marked as 1.

- It can be assumed that the dimensions (m, n) of the grid will not exceed the limits of 32, i.e. $1 \leq m, n \leq 32$.
- Contrast to Conway's game of life, the grid will *not* be infinite and any values beyond the grid can be assumed to be of value 0.
- You will have k marked cells in the initial state, you are free to choose k .
- $(2, 1)$ represents bottom-most row's 2nd cell.

The format for the initial state file will be:

```
m n k
x1 y1
⋮
xk yk
```

1.3 Rendering

From the rule and the initial state the user must be able to see the render of the grid from the initial state. User is then prompted to input the number of iterations to apply on the current state. Once these iterations are applied the new grid must be rendered.

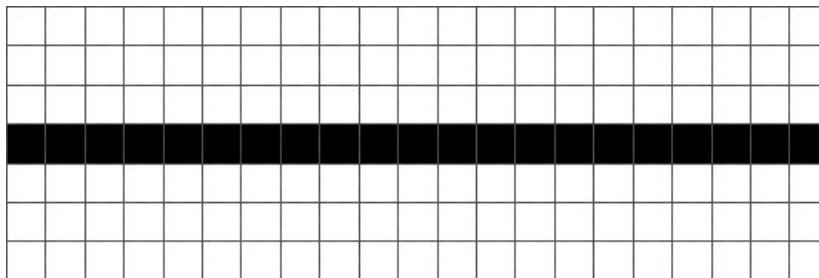
- If the given iteration number is -1 , the program must terminate.
- Each render must be accompanied by the iteration number the render represents from the **initial state**.
- You do not have to worry about smooth transitions or other graphical details, the bare requirement is for the grid to be visible and legible.

1.4 Generating Diagrams

Given below are states of grids that you must generate. Along with the states required, you have been given the number of iterations from the initial state that it will take to reach the given state.

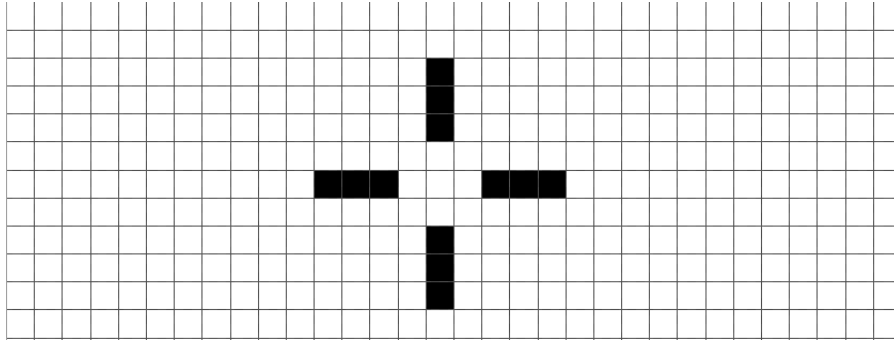
Your task is to give the configuration file of the initial state along with the rules.

1.4.1 Line



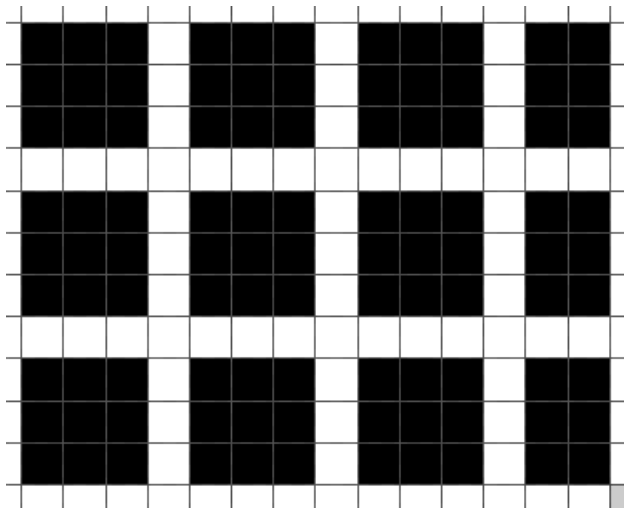
Iterations: 20 **Dimensions:** 21×7

1.4.2 Cross



Iterations: 34 Dimensions: 9×9

1.4.3 Grid



Iterations: 1 Dimensions: 16×13

1.5 Writing to a file

While the rendering is happening, the last render displayed must also be written into the file `output.txt` in the same folder. This will be subjected to automated testing. The black cells must be marked with a 1 and the white cells with 0. There must be no spaces between the numbers and each row must end with a newline.

The final output must match the dimensions mentioned under the diagram. It can be assumed that the marked values occur in the centre of the diagram.

The file must be generated in the same folder as the folder from where the python script is executed. `python gen1/rule.py` must start the rendering and when exited, the final answer must be present in `q1`.

1.6 Marks Distribution

- 50% of your marks will be allocated based on the functioning of your Renderer and correctness.

- 20% will be allocated based on viva on your code and concepts regarding cellular automaton.
- 30% will be based on the generated diagrams

2 Games

2. [15 points] Rock-Paper-Scissors

Pre-Final Evaluation Due Date: **6th September 2022, 11:59 PM**

Final Evaluation Due Date: **9th September 2022, 11:59PM**

In this assignment, you will write a program which, when given as input, a finite state automaton (FSA) which plays Rock-Paper-Scissors (RPS) with an unknown initial state, outputs a FSA that beats it in as many rounds as possible.

2.1 Game Description - For the uncultured

In each round, two players simultaneously choose one of rock, paper and scissors. Scissors beats paper, rock beats scissors, and paper beats rock. It's a draw if they choose the same. The game is repeated for multiple such rounds.

2.2 Finite State Automaton

Each state of a RPS FSA is described by c : what the FSA will play in the current round, and $r\ p\ s$: the new state if the opponent plays rock, paper and scissors respectively.

2.3 Task

You are given the opponent's FSA with $n \leq 30$ states but not its initial state. You have to submit the following files: 1. A program that outputs an FSA with $m \leq 1000$ states and a fixed initial state such that it beats the opponent FSA as many times as possible. We will simulate 10,000 rounds for each possible initial state of the opponent FSA. 2. A file containing a single FSA which your batchmates programs will be evaluated against. The better your test case, i.e. the harder it is to win against this FSA, the greater your score.

2.4 Input Format

Applies to both the input to the program you submit and the FSA test case you submit.

```

n
c1 r1 p1 s1
⋮
cn rn pn sn

```

2.5 Output Format

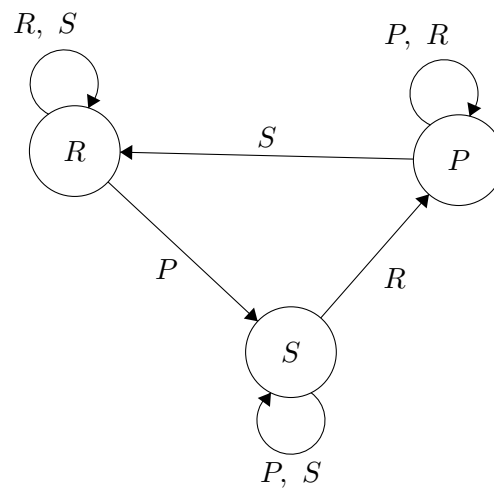
The output format for your program.

```
m  
c1 r1 p1 s1  
⋮  
cm rm pm sm
```

2.6 Example

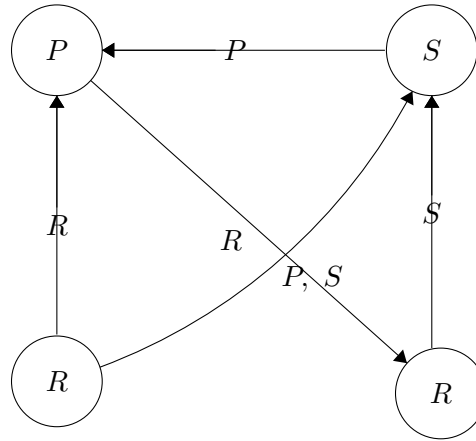
2.6.1 Sample Input

```
3  
R 1 2 1  
S 3 2 2  
P 3 3 1
```



2.6.2 Sample Output

```
4  
R 2 3 3  
P 4 1 1  
S 1 2 1  
R 1 1 3
```



2.6.3 Sample Explanation

We can see that the input FSA does not take a particularly smart approach, only changing its move when it loses in a way that beats the opponents previous move. Since we know this, we can design the output FSA to be 1 step ahead of this process. Initially, it outputs rock (state 1 bottom left), and then based on the opponents response it transitions to a state such that it wins every future round (states 2-4). Thus, if the opponent input state is Rock or Paper (states 1 or 3), it wins 9,999 rounds and if it is Scissors (state 2) it wins all 10,000 rounds. We cannot do better than this against this input FSA.

2.7 Scoring

- There will be a pre-final evaluation a few days before the deadline. You will have to submit your progress on both the program and the testcase. We will use this to provide feedback on common mistakes, as well as an opportunity for you to see how your approach is performing.
- The weightages for the different evaluations are the following: Pre-final evaluation (20%), Final evaluation (60%), Viva (20%). Within the 2 evaluations, there is 75% – 25% split between the performance of your program and the testcase. Thus, for example, your program in the pre-final evaluation holds an overall weightage of $0.20 * 0.75 = 15\%$.
- If your program wins (draws not counted) $x\%$ of rounds tested against, you will receive $\min(0, 2x - 100)\%$ points for this component. This means a program that wins only 50% rounds gets $2 * 50 - 100 = 0\%$ points. In the final evaluation which consists of $0.60 * 0.75 = 45$ points, a program that wins 90% rounds gets $(2 * 90 - 100)\% * 45 = 80\% * 45 = 36$ points for this component. Note that your program will be tested against other cases as well, not just those submitted by your batchmates.
- For scoring the testcases, we will use a relative scale. Among the the top $(10 * k)\%$ cases (in terms of numbers of rounds lost or drawn by the opponent program) will receive $100 - 10(k - 1)\%$ marks for the smallest value of k that applies. So in the final evaluation where the total weightage of the test case is $0.60 * 0.25 = 15\%$, the top 10% cases ($k = 1$) will receive full 15 points, the top 10 – 20% cases ($k = 2$) receive $100 - 10(2 - 1)\% * 15 = 90\% * 15 = 13.5$ points, top 20 – 30% 12 points and so on until the bottom 10% cases receive 1.5 points.

Please ensure you strictly follow the I/O format mentioned above and the submission format mentioned below as the evaluation is automated. We will be forced to give you a 0 otherwise. Please

post any general queries which others might share on Moodle, and for relevant queries specific to your approach reach out to **Shashwat Goel** on Teams.

Submission details

The submission must be a zip file containing the code and a report explaining the approach for the solutions.

```
rollnumber_prog1.zip
→ q1
  → ./src
  → ./gen1
    → ./config.txt
    → ./rule.py
  ...
→ q2
  → rps.cpp
  → fsm.txt
  ...
→ report.pdf
```

In $q1$, 'src' should contain all the code pertaining to the rendering of the systems. Describe in your report where code related to each sub-part of $q1$ is present in 'src'.