ASSIGNMENT 3: METRO MAP PLANNING

Goal: The goal of this assignment is to take a complex new problem and formulate and solve it as a SAT problem. Formulation as SAT is a valuable skill in AI that will come in handy whenever you are faced with a new problem in NP class. SAT solvers over the years have become quite advanced and are often able to scale to decently sized real-world problems. This assignment will be done in two parts.

Scenario (Part I): You are the designer of a city's underground metro system, and are planning the metro map of the city. You can assume the city to be a grid of size NxM, with rows (y-axis) 0, … M-1, and columns (x-axis) 0, … N-1, and the position (0,0) representing the topmost left corner of the city. Based on the city's requirements, K different metro lines (numbered 0 to K-1) are proposed with a start (s_k) and end (e_k) location identified for each metro line. Note that all start and end locations are unique points in the city's grid. You need to define the path for each metro rail under the following constraints:

1. There is at most 1 metro line under any location
2. Every metro line k is a path from s_k to e_k, for every k.
3. There are atmost J turns in any metro line (J may be a small number such as 1, 2 or 3).

We will use miniSAT, a complete SAT solver for this problem. Your code will read a graph in the given input format. You will then convert the mapping problem into a CNF SAT formula. The encoding time and encoding size should be polynomial in the size of the original graph. Your SAT formula will be the input to miniSAT, which will return with a variable assignment that satisfies the formula (or an answer "no", signifying that the problem is unsatisfiable). You will then take the SAT assignment and output the metro map. Note that you can make only one call to miniSAT.

You are being provided a problem generator that takes inputs N, M, K and J, and generates random problems with those parameters.

Input format:

The first line is either a 1 or a 2. 1 represents Scenario #1. Second line has four numbers: number of columns in the grid (N), number of rows in the grid (M), number of metro lines (K) and max number of turns allowed in a single line (J).

Each subsequent line represents a metro line in order 0 to K-1. It has four number representing (x,y) location of start, and (x,y) locations of end on the grid.

As an example, consider this:

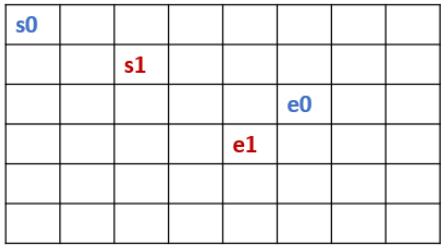
1

8 6 2 1

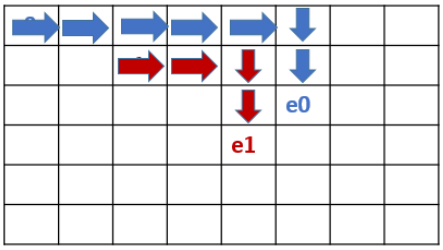
0 0 5 2

2 1 4 3

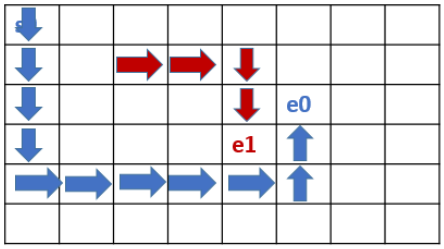
It is depicted pictorially as follows:



valid



invalid



Notice that second solution is invalid because the 0^{th} metro line makes two turns whereas J=1.

Output format:

The output format for valid solution will be written as follows:

R R R R R D D 0

R R D D 0

Here L represents left, R right, D down, and U up. The first row will be the path for 0^{th} metro line, and so on. Each row ends with a 0.

If the problem is unsatisfiable output only a single 0.

Scenario (Part II): Same setting, except that there are some popular parts of the city (cells in the grid) and some metro line must go through them. The change in the input format will be that first line will be 2. And the second line will have five numbers. Fifth number will be the P – the number of popular cells. And after defining all starts and ends, the last row will represent all P popular cells in (x,y) format. For example, in the example above, if (1,0) and (3,1) are popular cells, then the input format will look like:

2

8 6 2 1 2

0 0 5 2

2 1 4 3

1 0 3 1

Code

1. You may program the software in any of C++, or Python. The versions of the compilers that will be used to test your code are
   Python 3.6
   g++ 4.8.1
2. Executing the command **"./run1.sh test"** will take as input a file named **test.city (Part I)** and produce a file **test.satinput** – the input file for minisat. You can assume that test.city exists in the present working directory. ('test' is a placeholder parameter and can be changed when running).
3. Executing the command **"./run2.sh test"** will use the generated **test.satoutput, test.city** (and any other temporary files produced by run1.sh) and produce a **file test.metromap** – the mapping in the output format described above. You can assume that test.city, test.satoutput (and other temp files) exist in the present working directory.
4. The TA will execute your scripts as follows:
   ./compile.sh
   ./run1.sh filename
   ./minisat filename.satinput filename.satoutput
   ./run2.sh filename

   When we call "./run1.sh test", you can assume that test.city exists in the present working directory. When we call "./run2.sh test", you can assume that test.city, test.satinput and test.satoutput exist in the present working directory, along with any other temporary files created by "./run1.sh test". And so on.

Please note that you are NOT allowed to call minisat within run1.sh or run2.sh. The TA will call minisat. Your code will be killed after the given problem cutoff time, so please write good code. However, cutoff time will be not be too short (such as 1-2 minutes only) – it will be larger for large problems, so you need not over-optimize I/O code. That said, time taken to solve the overall problem will be a factor in your final score.

Useful resources

1. http://minisat.se/MiniSat.html: The MiniSat page
2. http://www.dwheeler.com/essays/minisat-user-guide.html: MiniSat user guide

What to submit?

1. Submit your code in a .zip file named in the format **<EntryNo>.zip.** If there are two members in your team it should be called <EntryNo1>_<EntryNo2>.zip. Make sure that when we run "unzip yourfile.zip" it contains a directory with the same name as the zip file and the following files are present in that directory:
   compile.sh
   run1.sh
   run2.sh
   writeup.txt

   You will be penalized for any submissions that do not conform to this requirement.

   We will run your code on a few sample problems and verify the ability of your code to find solutions within a cutoff limit. The cutoff limits will be problem dependent and your translation does not need to depend on the cutoff limit, therefore it is not part of the input format. Of course, better translations will scale better, fetching more points.

2. The writeup.txt should have three lines as follows
   First line should be just a number between 1 and 2. Number 1 means C++. Number 2 means Python.

   Second line should start with this honor code. "Even though I/we have taken help from the following students and LLMs in terms of discussing ideas and coding practices, all my/our code is written by me/us."
   Third line should mention names of all students and LLMs you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone else say None.
   After these first three lines you are welcome to write something about your code, though this is not necessary.

Code verification before submission: Your submission will be auto-graded. This means that it is absolutely essential to make sure that your code follows the input/output specifications of the assignment. Failure

to follow any instruction will incur a significant penalty. The details of code verification will be shared on Piazza (similar to A2).

Evaluation Criteria

1. Final competition on a set of similar problems. The points awarded will be your normalized performance relative to other groups in the class.
2. Extra credit may be awarded to standout performers.

What is allowed? What is not?

1. You may work in teams of two or by yourself. We do not expect a different quality of assignment for 2 people teams. At the same time, please spare us the details in case your team cannot function smoothly. Our recommendation: this assignment may be a little hard for students with limited prior exposure to logic. If you are such a student, work in teams if you can find a workable partner. If you are good at logic, the assignment is quite easy and a partner should not be required.
2. You can use any language from C++, or Python for translation into and out of Minisat, as long as it works on our test machines. We will NOT be responsible for differences in versions leading to execution failures.
3. You must not discuss this assignment with anyone outside the class. You cannot ask LLMs to write code for you. However, you are allowed to give the LLM your code in case you are stuck in debugging, so that you can learn about your mistakes fast, and do not waste time in debugging. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class outside your team.** Please read academic integrity guidelines on the course home page and follow them carefully.
4. You are also allowed to use an LLM's help in modeling the problem as SAT. That said, the fun of the assignment is in learning how to convert problems to SAT. By doing this you will lose out on important education.
5. Please do not search the Web for solutions to the problem.
6. Please do not use ChatGPT or other large language models for creating direct solutions (code) to the problem. Our TAs will ask language models for solutions to this problem and add its generated code in plagiarism software. If plagiarism detection software can match with TA code, you will be caught.
7. Your code will be automatically evaluated. You get a *minimum* of 20% penalty if your output is not automatically parsable.
8. We will run plagiarism detection software. Any team found guilty of either (1) sharing code with another team, (2) copying code from another team, (3) using code found on the Web, will be awarded a suitable strict penalty as per IIT and course rules.