ASSIGNMENT 2: DISCOVERY OF DRUG AGENCIES

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.

Scenario: You are an investigative agency working on uncovering the hidden connections between the various drug agencies. You have got telephone records of various telephone numbers which are believed to be associated with the mafia. Some external source has suggested that there are K different drug agencies with several people part of multiple agencies. Your goal is to automatically uncover the various drug agencies from the telephone records. To solve this problem (for our assignment), you make a few assumptions.

1. Each person has exactly one phone.
2. All drug agencies are very close-knit: if two people are in the same agency they must have called each other.
3. People don't directly call anyone outside their agency.
4. No agency is a strict subsidiary of another agency.

You abstract out the problem by creating an undirected graph G, where each node is a person and an edge between two nodes indicates that they had a phone conversation.

Problem Statement: Given an undirected graph G, and a number K, output K subgraphs of G (say G_1, … G_k) such that a node/edge in G is present in at least one of G_i. Moreover, all G_is are complete graphs (i.e., all nodes connected to each other). Finally, no G_i is a subgraph of another G_j. Sample cases are shown here. Note that there are no self loops in G. G only has 1 connected component.

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 convert it into K complete subgraphs. You will output these subgraphs in the given output format.

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

Input format:

The first line has three numbers: number of vertices in G, number of edges in G and K.

Nodes are represented by positive integers starting from 1. Each subsequent line represents an edge between two nodes. An input file that represents the last example in the slide is:

5 8 2

1 2

1 3

1 4

4 5

3 2

4 2

5 3

3 4

Output format:

Each subgraph will be prefaced with a #i $|G_i|$ indicating that it is the ith subgraph of number of vertices $|G_i|$. Post that mention the vertices in one line. For the solution to the above example

#1 4

1 2 3 4

#2 3

3 4 5

If the problem is unsatisfiable output a 0.

Code

Your code must compile and run on machine named 'todi' or any machine with similar configuration present in GCL. We will test you on virtual machines (baadal) with similar configuration, but best to

check your code success via a preliminary submission. Please supply a compile.sh script. Also supply two shell scripts run1.sh, run2.sh:

1. Executing the command "./run1.sh test" will take as input a file named test.graph and produce a file test.satinput – the input file for minisat. You can assume that test.graph exists in the present working directory.
2. Executing the command "./run2.sh test" will use the generated test.satoutput, test.graph (and any other temporary files produced by run1.sh) and produce a file test.subgraphs – subgraphs in the output format described above. You can assume that test.graph, test.satoutput (and other temp files) exist in the present working directory.
3. The TA will execute your scripts as follows:
   ./run1.sh test
   ./minisat test.satinput test.satoutput
   ./run2.sh test

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

While we have not given an explicit time limit in the assignment, we may cut off your program if it takes an excruciatingly long amount of time, say more than an hour or so.

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 is being provided?

A problem generator for G and K where G does have K complete subgraphs is being provided. A check function that tests your output is also being provided. It does not check "unsatisfiable" output and only verifies if your solution provides K complete subgraphs. To run the generator use the command "python problemGenerator.py <number of vertices> <K>", which will generate the input file "test.graph". To test your code use "python checker.py <input graph file> <output subgraphs file>". It will only work for satisfiable cases.

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" the following files are produced in the present working directory:

   compile.sh
   run1.sh
   run2.sh
   Writeup.pdf

   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 and will possibly get higher scores.
2. Submit at-most 1 page writeup (10 pt font) describing your algorithm for translation into SAT and, if you performed, any insights or experiments reporting the scalability of your code. This is not graded but failure to submit a satisfactory writeup will incur negative penalty of 20% of total score. Your writeup will help us identify any common misconceptions and particularly good ideas for discussion in the class.

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 significant penalty.

From this year onwards, we are running a pilot project where we shall be generating a log report for every submission within 12 hours of submission. This log will let you know if your submission followed the assignment instructions (format checker, scripts for compilation & execution, file naming conventions etc.). Hence, you will get an opportunity to resubmit the assignment within half a day of making an inappropriate submission. However, please note that the late penalty as specified on the course web page will still apply for resubmissions beyond the due date. Exact details of log report generation will be notified on Piazza soon.

*Also, note that the log report is an **additional** utility in an experimental stage. In case the log report is not generated, or the sample cases fail to check for some other specification of the assignment, appropriate penalty for not adhering to the input/output specifications of the assignment will still apply at the time of evaluation on real test cases.*

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 actually quite easy and a partner should not be required.
2. You can use any language from C++, Java 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. **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. Please do not search the Web for solutions to the problem.
5. Your code will be automatically evaluated against another set of benchmark problems. You get a zero if your output is not automatically parsable.
6. We will run plagiarism detection software. Any team found guilty will be awarded a suitable penalty as per IIT rules.