ASSIGNMENT 3: HIDDEN ORGS FINDER

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 an investigative agency working on uncovering the hidden connections between the various drug agencies. You have got telephone records of various telephone numbers, some of which are believed to be associated with the mafia. Some external sources have suggested that there are two different drug agencies (of sizes k1 and k2 each). Your goal is to automatically uncover the two drug agencies from the telephone records. To solve this problem (for our assignment), you make a few assumptions.

1. Each person (in a drug agency or not) 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. The two drug agencies do not have any common person.

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 numbers K_1 and K_2 ($K_1 >= K_2$), output two subgraphs of G (say G_1, G_2) such that G_1 is of size K_1 and G_2 is of size K_2, and both G_is are complete graphs (i.e., all nodes connected to each other in G). Finally, G_1 and G_2 do not have any common nodes. Note that there are no self loops in G. Sample cases are shown in Sample_graphs.

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 two complete subgraphs. You will output these subgraphs in the given output format. Note that you can make only one call to miniSAT.

You are being provided a problem generator that takes inputs |G|, K1 and K2, and generates random problems with those parameters.

Input format:

The first line has four numbers: number of vertices in G, number of edges in G, K1 and K2.

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

7 10 4 2

1 2

1 3

1 4

1 6

4 5

3 2

4 2

5 3

3 4

6 7

Output format:

Each subgraph will be prefaced with a #i indicating that it is the i^{th} subgraph. Post that, mention the vertices in one line. For the solution to the above example

#1

1 2 3 4

#2

6 7

If the problem is unsatisfiable output a 0.

Scenario (Part II): Same setting, except that there is only one drug agency, and your intelligence is weak and **you don't know the size of the agency**. Your goal is to automatically uncover the largest possible drug agency from the telephone records. To solve this problem (for our assignment), you make a few assumptions.

1. Each person has exactly one phone.
2. The drug agency is very close-knit: if two people are in the same agency they must have called each other.

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, output one subgraph of G (say G_1) such that G_1 is of the maximum size possible, and G_1 is a complete graph. For example above, the output subgraph will be of size 4 with nodes {1,2,3,4}.

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 the complete subgraph. You will output this subgraph in the given output format. Here you will be allowed to make multiple calls to miniSAT.

You may use the same problem generator as before and just remove K1 and K2 as inputs to create an input for this part.

Input format:

The first line has two numbers: number of vertices in G and number of edges in G.

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

7 10

1 2

1 3

1 4

1 6

4 5

3 2

4 2

5 3

3 4

6 7

Output format:

The subgraph will be prefaced with a #1 which will indicate only one largest subgraph of the original graph. Post that, mention the vertices in one line. For the solution to the above example

#1

1 2 3 4

If the problem is unsatisfiable output a 0.

Code

1. You may program the software in any of C++, Java or Python. The versions of the compilers that will be used to test your code are
   JAVA: java version "1.7.0_79" (OpenJDK)
   Python 3.6
   g++ 4.8.1
2. Executing the command **"./run1.sh test"** will take as input a file named **test.graphs (Part I)** and produce a file **test.satinput** – the input file for minisat. You can assume that test.graphs exists in the present working directory. ('test' is a parameter and can be changed when running).
3. Executing the command **"./run2.sh test"** will use the generated **test.satoutput, test.graphs** (and any other temporary files produced by run1.sh) and produce a **file test.mapping** – the mapping in the output format described above. You can assume that test.graphs, test.satoutput (and other temp files) exist in the present working directory.
4. Similarly ./run3.sh will be used to evaluate Part II.
5. The TA will execute your scripts as follows:
   ./compile.sh
   ./run1.sh test
   ./minisat test.satinput test.satoutput
   ./run2.sh test
   ./run3.sh newtest
   When we call "./run1.sh test", you can assume that test.graphs exists in the present working directory. When we call "./run2.sh test", you can assume that test.graphs, 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. Note that "test" and "newtest" are just placeholders – the name of the file can be anything.

Please note that you are NOT allowed to call minisat within run1.sh or run2.sh. The TA will call minisat. On the other hand, you ARE allowed to call minisat multiple times within run3.sh. 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 (like 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 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 is being provided?

A problem generator for G and K1 and K2 where G does have two 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 two complete subgraphs of necessary sizes. To run the generator use the command "python problemGenerator.py <number of vertices> <K1> <K2>", which will generate the input file "test.graph". To test your code use "python checker.py <input graph file> <output subgraphs file> <problem 1/2>". For checking, provide the '1' for checking problem 1 and '2' for checking problem 2. 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" 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
   run3.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 two lines as follows
   First line should be just a number between 1 and 3. Number 1 means C++. Number 2 means Java and Number 3 means Python.

   Second line should mention names of all students 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 two 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 A1).

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++, 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. Please do not use ChatGPT or other large language models for solutions 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.
6. Your code will be automatically evaluated. You get a minimum of 20% penalty if your output is not automatically parsable.
7. 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.