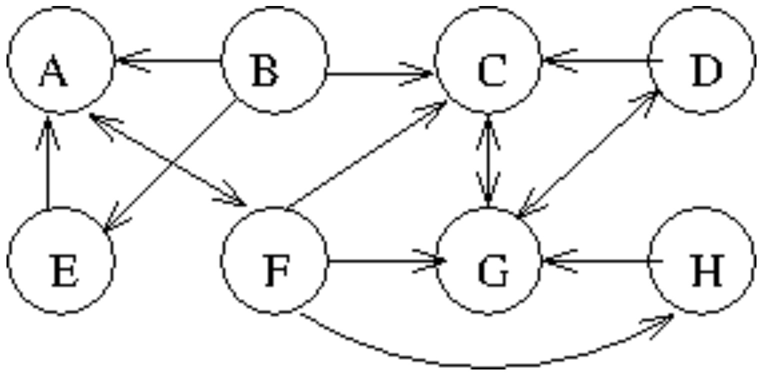
Programming Assignment 3: Propositional Logic

Assigned: Oct. 11
Due: Nov. 1

General Description

The HAMILTONIAN PATH problem is the following: Given a directed graph G, find a path that includes every vertex of G exactly once. For instance in the graph below B,E,A,F,H,G,D,C is a Hamiltonian path.



In this assignment, you will write a three-part program that together solve the Hamiltonian path problem. Part I takes the graph as input and outputs a propositional encoding in CNF. Part II is an implementation of the Davis-Putnam algorithm; it takes a set of clauses as input and outputs a solution, if one exists. The three parts should be separately executable files; the output of part I is the input to part II, and the output of part II is the input to part III.

Propositional encoding of Hamiltonian path

The Hamiltonian path can be encoded as a propositional satisfiability problem as follows:

Let *N* be the number of vertices in the graph. The atoms have the form *UT* where *U* is a vertex in the graph and *T* is a number between 1 and N. The intuitive mean of *UT* is that *U* is the *T*th vertex on some particular path through the graph.

There are the following types of propositions:

- Every vertex is traversed at some time. For each vertex *U* we would have the proposition *U1* V *U2* V ... V *UN*.
  For instance in the graph shown above *A1* V *A2* V *A3* V *A4* V *A5* V *A6* V *A7* V *A8*
  would be a proposition, and likewise for the other vertices.

- No pair of vertices are traversed at the same time. For each pair of vertices *U,W* and each time *T* we have the proposition ¬*UT* V ¬*VT*.
  For instance ¬A1 V ¬B1 and likewise for every pair of vertices and for every time.

- You cannot go from *U* at time *T* to *W* at time *T+1* if there is no edge from *U* to *W*. Thus, for every pair of vertices *U,W* for which there is no edge from *U* to *W* and for every time *T* between 1 and *N1* there is a proposition ¬*UT* V ¬*W(T+1)*.
  For instance in our example, since there is no arc from A to C, there are axioms
  ¬A1 V ¬C2.
  ¬A2 V ¬C3.

   . . .
   ¬A7 V ¬C8.
   and likewise for all the other missing arcs.

- (Optional) At every time there is a vertex. For each time *T* you have the proposition U_1T V U_2T V ... V U_NT.
  For instance in our example, we have the proposition *A*1 V *B*1 V *C*1 V *D*1 V *E*1 V *F*1 V *G*1 V *H*1.
  and likewise for the other times.

- (Optional) No vertex is traversed more than once. For each vertex U, for each times S and T, you have the proposition ¬US V ¬UT. For instance, ¬ A1 V ¬ A2, and likewise for the other vertices and pairs of times.

The optional propositions aren't necesary, but may make it run faster.

Specifications

Input / Output.

All three programs take their input from a text file produce their output to a text file. (If you want, you may use standard input and output.)

Davis-Putnam

The input to the Davis-Putnam procedure has the following form: An atom is denoted by a natural number: 1,2,3 ... The literal P is the same number as atom P; the literal ~P is the negative. A clause is a line of text containing the integers of the corresponding literals. After all the clauses have been given, the next line is the single value 0; anything further in the file is ignored in the execution of the procedure and reproduced at the end of the output file. (This is the mechanism we will use to allow the front end to communicate to the back end.)

The output from the Davis-Putnam procedure has the following form: First, a list of pairs of atom (a natural number) and truth value (either T or F). Second, a line containing the single value 0. Third, the back matter from the input file, reproduced.

Example: Given the input

```
1 2 3
-2 3
-3
0
This is a simple example with 3 clauses and 3 atoms.
```

Davis-Putnam will generate the output

```
1 T
2 F
3 F
0
This is a simple example with 3 clauses and 3 atoms.
```

This corresponds to the clauses

```
P V Q V R.
~Q V R.
~R.
```

If the clauses have no solution, then Davis-Putnam outputs a single line containing a 0, followed by the back-matter in the input file.

Note: Your implementation of Davis-Putnam must work on *any* set of clauses, not just those that are generated by the Hamiltonian path program.

You may assume that there are no more than 250 atoms and no more than 10,000 clauses.

Front end

The front end takes as input a specification of a graph and generates as output a set of clauses to be satisfied. The vertices are labelled A, B, C ... in sequence. (You don't have to deal with graphs with more than 26 vertices).

The format of the input contains the following elements:

- First line: The number of vertices.
- Remaining lines: The list of edges. Each line encodes one edge: start vertex and end vertex.

For example, the encoding of the above graph would be

```
8
A F
B A
B C
B E
C G
D C
D G
E A
F C
F G
F H
G C
G D
H G
```

You may assume that the input is correctly formatted. You do not have to do any error checking on the input. The output consists of

- 1. A set of clauses suitable for inputting to Davis-Putnam as described above.
- 2. A key to allow the back end to translate the numbers used for propositional atoms in the clauses into the correct path. This should consist of a sequence of lines with the number assigned to the propositional atom, the vertex name and sequence number. Thus, for example

  ```
  1 A 1
  2 A 2
  . . .
  8 A 8
  9 B 1
  10 B 2
  11 B 3
  ```

```
. . .
63 H 7
64 H 8
```

Back-end

The back end takes as input the output that Davis-Putnam generates when run on the output of the front end. It generates as output the path that solves the problem; e.g.

If the input indicates that the clauses have no solution, the back end should output the message "NO SOLUTION".

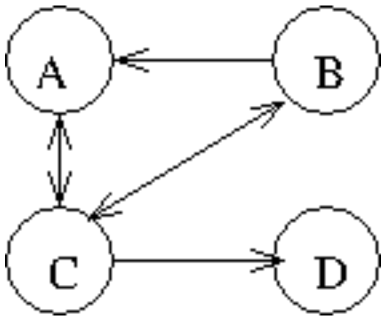
Another example for Davis-Putnam

The following is the input-output pair just for the Davis-Putnam module --- *not* the front and back ends --- corresponding to the example in the class notes.

- Input for Davis-Putnam
- Output from Davis-Putnam

A Simpler Puzzle Example

It would probably be a mistake to begin your testing using the above example, with its 64 propositional atoms and 650+ propositions. Rather, I would suggest that you start by working on the following simple puzzle.



Input for front end for this small puzzle
Propositions for this small puzzle in symbolic form. Note: There is no need for your program to generate any kind of output or data structure in propositional format. This is just to help you check your work.
Output for front end for this small puzzle
These do not include the optional axioms 4.B.

Deliverable

You should upload to NYU Classes. (a) the source code; (b) instructions for running it, if there's anything at all non-obvious about it. Nothing else.

Grading

The Davis-Putnam program is worth 60% of the grade; the front end is worth 35%; the back end is worth 5%. In each of these, a program that does not compile will get a maximum of 10%; a correct program will get 90%; the remaining 10% is for being well-written and well commented.