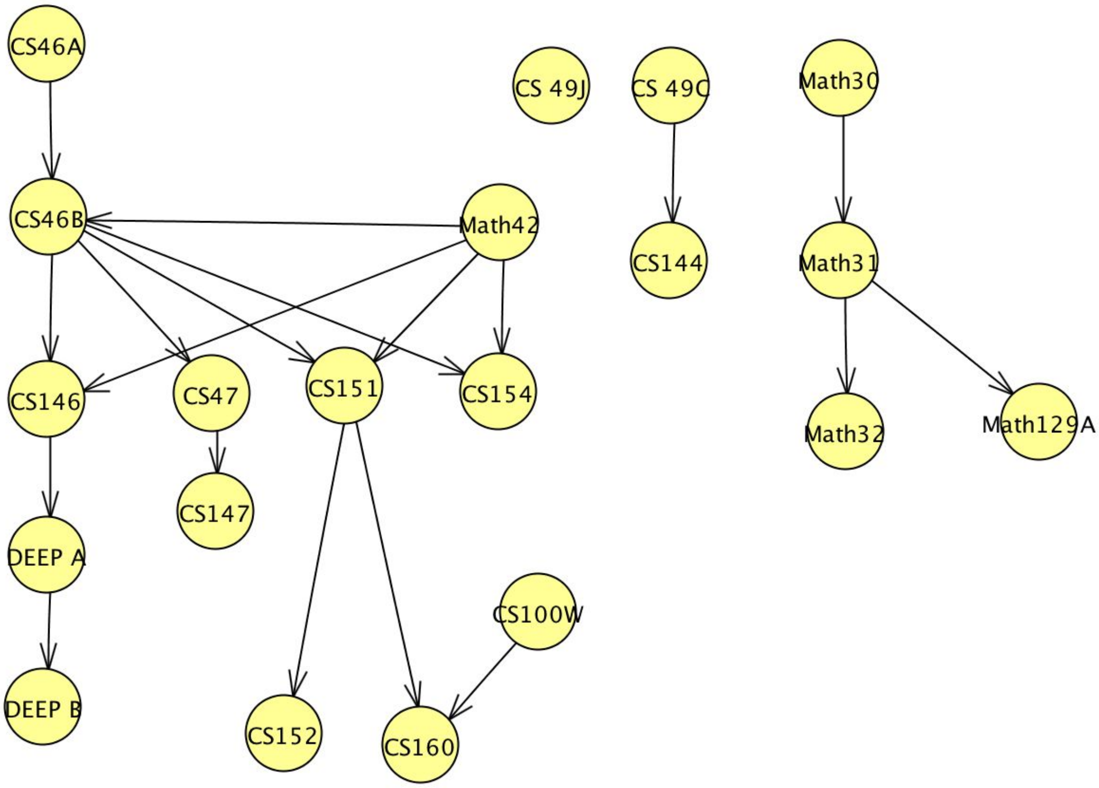
CS156-2
10/31/17

AI HW 3 Questions 1 - 6, follow up of sat_solver port

1) Consider the problem of getting a BSCS at SJSU. Simplify the prerequisite graph of CS courses by only consider deep course sequences (i.e., CS157A, CS157B; ; etc) which have CS146 only as a prerequisite for the first course. Treat all these sequences as the same sequence CSDeepA and CSDeepB. Write down the the task of just scheduling your CS courses as a CSP. You should have sufficient constraints so that a solution would mean that you have satisfied all undergrad CS requirements (you can ignore general ed and other requirements). Your domains should be the possible semesters to start a course. For example, 1 means you start the course the first semester, 2 the second, etc. Let 8 be the highest semester number and assume you can take at most 5 courses a semester.

*Note that most elective courses are excluded for simplicity but CS144 was included due to some questions directly related to this course.
Variables:
{
CS46A, CS46B, CS47, CS49C, CS49J, CS100W, CS130, CS144, CS146,
CS147, CS151, CS152, CS154, CS160, CSDeepA, CSDeepB,
Math30, Math31, Math32, Math42, Math129A
}
Domain: Each course has domain: {1, 2, 3, 4, 5, 6, 7, 8}
Constraints:
<(less than): defined as the lesser course being a direct prerequisite for the larger course. When the variables take values between 1-8, the prerequisite course must have a lower value
atMost5variablesEquals: defined as a constraint that restricts having more than 5 variables set to the value specified in the parameter
{
CS46A < CS46B
CS46B < CS47
CS46B < CS130
CS46B < CS146
CS46B < CS151
CS46B < CS154
CS47 < CS147
CS100W < CS160
CS151 < CS152
Math30 < Math31

```
        Math31 < Math32
        Math31 < Math129A
        Math42 < CS146
        Math42 < CS151
        Math42 < CS154
        Math42 <= CS46B
        atMost5variablesEqual(1), atMost5variablesEqual(2),
        atMost5variablesEqual(3), atMost5variablesEqual(4),
        atMost5variablesEqual(5), atMost5variablesEqual(6),
        atMost5variablesEqual(7), atMost5variablesEqual(8)
}
```



2) Let CS46B be the variable corresponding to the class CS46B. If we use the degree heuristic to choose the first variable in backtracking search to solve our CSP and if we break ties by picking the smaller numbered course, what is the variable X that would be considered in backtracking search in your CSP and what course does it correspond to? What would be the result of the REVISE(csp, X,CS46B)?

The next node to be assigned following the degree heuristic would be Math 42 with a degree of 4, which ties with CS46B's degree of 4 but is broken because 42 < 46. The backtracking algorithm would then try all the domain values of Math42 as assignments, and check them for consistency with the node's neighbors. First it would try to assign Math42 = 1. REVISE(csp, Math42, CS46B) would return false, and remove no values from CS46B's domain which since every value in CS46B's domain {1, 2, 3, 4, 5, 6, 7, 8} fits the constraint Math42 <= CS46B since all the values are less than or equal to 1. The revise operation would continue to revise CS146, CS151, and CS154's domains as well to remove 1 from each of theirs.

3) Suppose we choose CS49C to be taken in the first semester. Show all of the domains for each course after doing an arc consistency check (you only need to list the ones that change), in doing the check carefully trace the AC3 algorithm. In terms of works you did, how did this compare to the REVISE operation of the previous problem.

CS49C = 1
After AC3 only the variable CS144's domain would be changed to not include the value '1' due to 49C being taken in the first semester and the value '1' for CS144 would violate arc consistency.
It's different from the revise in #2 because it would only affect 144's domain. In #2, 3 domains were revised and one remained unchanged (46B).

4) From this updated CSP, using the MINIMUM REMAINING VALUE heuristic, give the variable to consider next. Break ties by choosing the course of smaller number.
Next to consider would be CS144 because it would be the only unassigned course with a domain size of 7 and not 8.

5) Using the variable from the last step, determine the value for it using the LEAST-CONSTRAINING-VARIABLE heuristic breaking ties by picking the earliest possible semester.
The value for CS 144 would become 2.

6) This next problem is unconnected with the five previous. Come up with a logical formula which expresses $P_{3,3}$, there is a pit in square (3,3), in terms of the variables (more than one) $B_{x,y}$ expressing there is a breeze in square (x,y). Convert this formula to CNF by hand by first doing biconditional elimination, then implication elimination, then using distributivity. Check your work by downloading the software from Russell and Norvig, launching the Python interpreter, typing *from logic import* * then doing to_cnf("your original formula"). Obviously, replace "your original formula" with the formula you had before converting to CNF. Have the transcript of your interaction with Python as part of your solution.

	B_{32}	
B_{23}	P_{33}	B_{43}
	B_{34}	

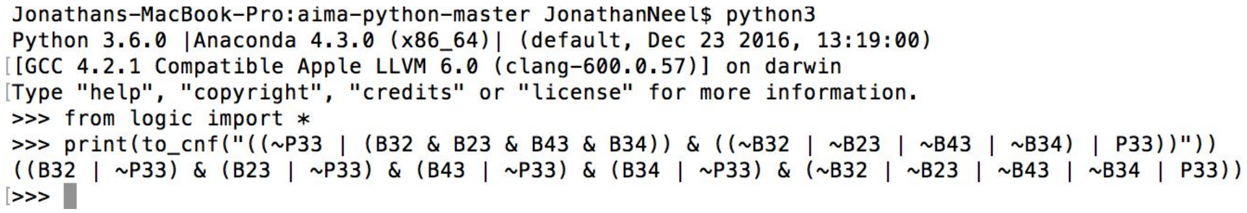
a) Logical Formula:
$P_{33} \Leftrightarrow B_{32} \,\&\, B_{23} \,\&\, B_{34} \,\&\, B_{43}$

b) Biconditional Elimination:
$P_{33} \Rightarrow B_{32} \,\&\, B_{23} \,\&\, B_{34} \,\&\, B_{43}$
&
$B_{32} \,\&\, B_{23} \,\&\, B_{34} \,\&\, B_{43} \Rightarrow P_{33}$

c) Implication Elimination:
$[\sim P_{33} \mid (B_{32} \,\&\, B_{23} \,\&\, B_{34} \,\&\, B_{43})]$
&
$[(\sim B_{32} \mid \sim B_{23} \mid \sim B_{34} \mid \sim B_{43}) \mid P_{33}]$

d) Distributivity:
$((B_{32} \mid \sim P_{33}) \,\&\, (B_{23} \mid \sim P_{33}) \,\&\, (B_{34} \mid \sim P_{33}) \,\&\, (B_{43} \mid \sim P_{33})$
&
$(\sim B_{32} \mid \sim B_{23} \mid \sim B_{34} \mid \sim B_{43} \mid P_{33}))$

e) To_cnf output of the logical formula from the implication elimination



```
 Jonathans-MacBook-Pro:aima-python-master JonathanNeel$ python3
 Python 3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016, 13:19:00)
[[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
[Type "help", "copyright", "credits" or "license" for more information.
 >>> from logic import *
 >>> print(to_cnf("((~P33 | (B32 & B23 & B43 & B34)) & ((~B32 | ~B23 | ~B43 | ~B34) | P33))"))
 ((B32 | ~P33) & (B23 | ~P33) & (B43 | ~P33) & (B34 | ~P33) & (~B32 | ~B23 | ~B43 | ~B34 | P33))
[>>> 
```

SAT_SOLVER.PY PORT RESOLUTIONS:

CNF 1:
The random cnf generator produced these clauses:
p cnf 3 3

-1 -3 0
3 -2 0
1 2 -3 0
2 -1 0

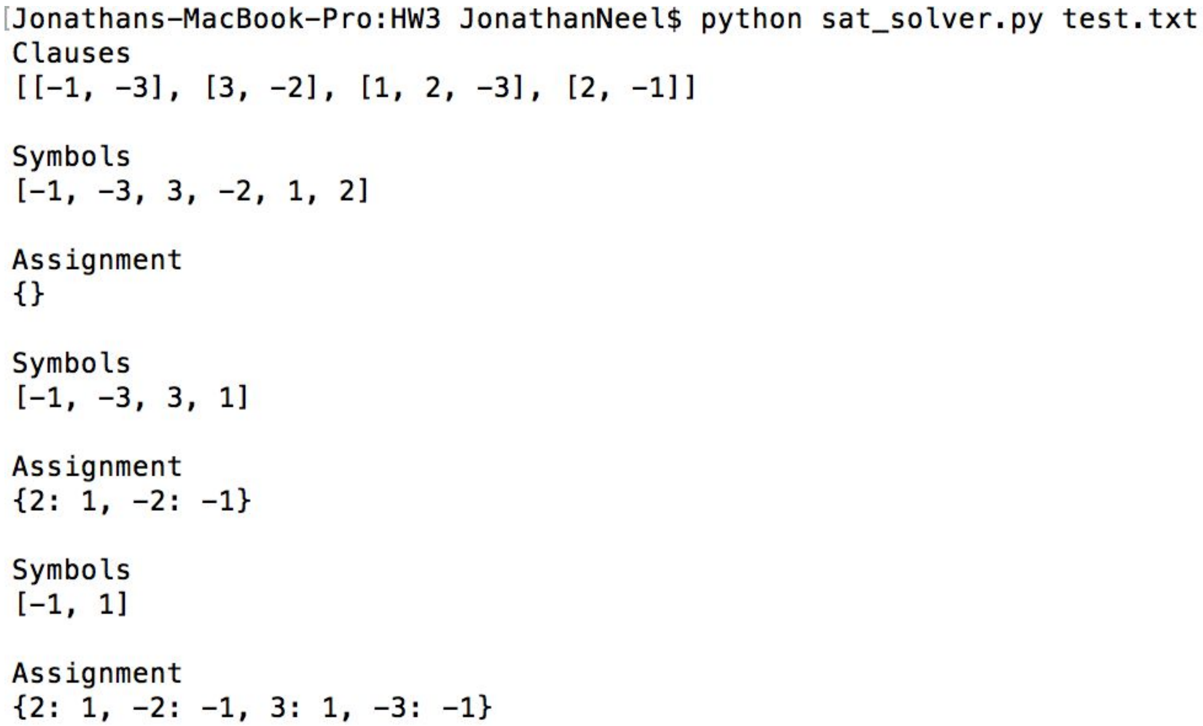
Let's call 1 = A, 2 = B, 3 = C
translating from the sat format, the cnf can be rewritten as:
(~A | ~C) & (C | ~B) & (A | B | ~C) & (B | ~A)

resolution:
R1: { ~A | ~C }
R2: {C | ~B}

R3: {A | B | ~C}
R4: {B | ~A}
R5: {~A | ~B} from R1 and R2
R6: {~A} from R4 and R5
R7: {C | ~A} from R4 and R2
R8: {B} from R7 and R3
R9: {C} from R8 and R2

So the assignment would be A = false, B = true, C=true. If we plug these values back into the original cnf we can verify this as a satisfiable assignment:
(~A | ~C) => (true | false) => true
(C | ~B) =>(true | false) => true
(A | B | ~C) => (false | true | false) => true
(B | ~A) => (true | true) => true
So the original cnf becomes (true & true & true & true) => true
Therefore the assignment was satisfiable.

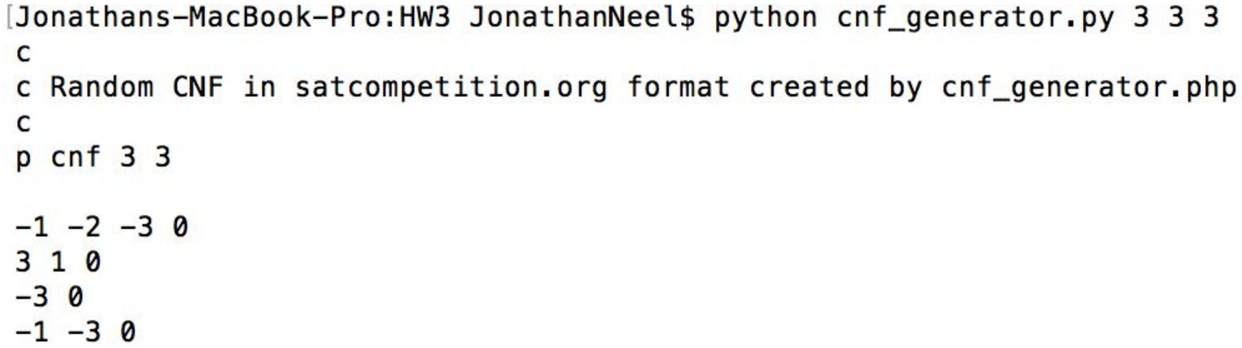
This is the output and assignment from the sat_solver.py function:



```
[Jonathans-MacBook-Pro:HW3 JonathanNeel$ python sat_solver.py test.txt
 Clauses
 [[-1, -3], [3, -2], [1, 2, -3], [2, -1]]

 Symbols
 [-1, -3, 3, -2, 1, 2]

 Assignment
 {}

 Symbols
 [-1, -3, 3, 1]

 Assignment
 {2: 1, -2: -1}

 Symbols
 [-1, 1]

 Assignment
 {2: 1, -2: -1, 3: 1, -3: -1}
```

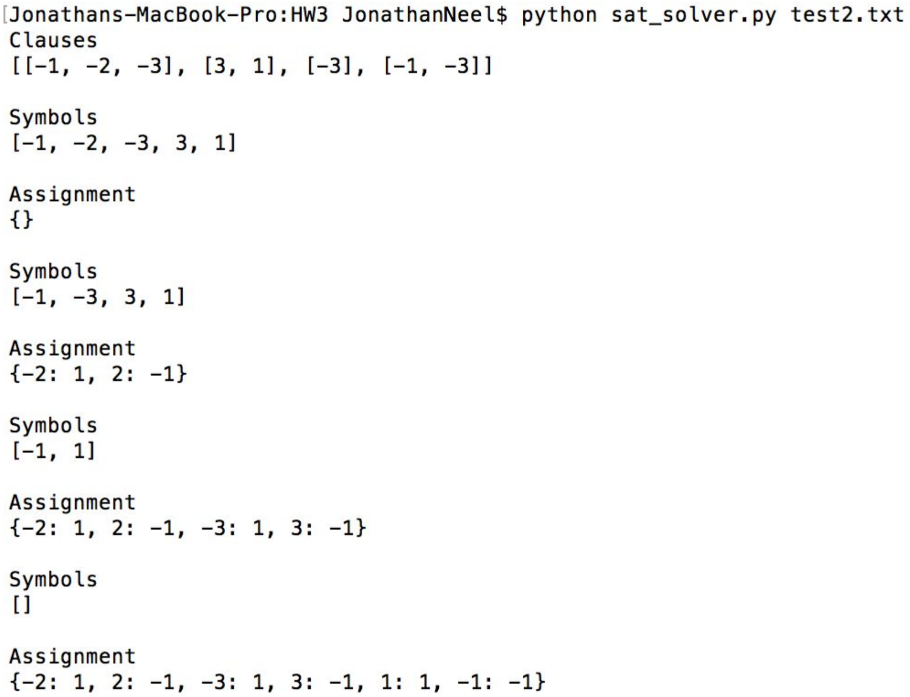
this assignment is the same: 1 (A) is false, and 2 (B) is true, and 3(C) is true.

CNF 2:



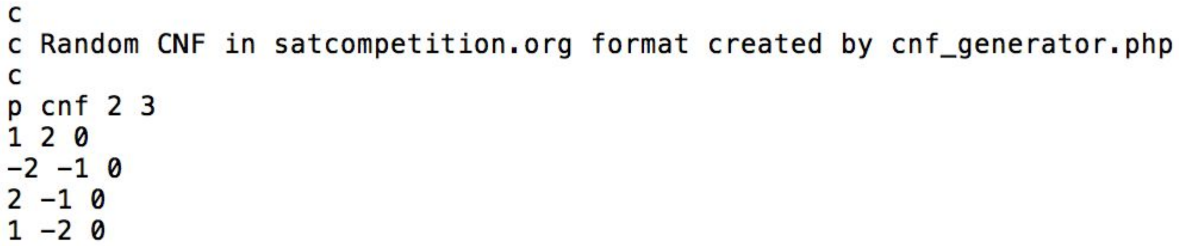
```
[Jonathans-MacBook-Pro:HW3 JonathanNeel$ python cnf_generator.py 3 3 3
 c
 c Random CNF in satcompetition.org format created by cnf_generator.php
 c
 p cnf 3 3

 -1 -2 -3 0
 3 1 0
 -3 0
 -1 -3 0
```

Let’s call 1 = A, 2 = B, 3 = C
translating from the sat format, the cnf can be rewritten as:
(~A | ~B |~C) & (C | A) & (~C) & (~A | ~C)
resolution:
R1: { ~A | ~B |~C }
R2: {C | A}
R3: {~C}
R4: {~A | ~C}
R5: {A} from R2 and R3
R6: {~B | ~C} from R1 and R5
R7: {~B} from R6 and R3
So the assignment would be C = false, A = true, B = false. However technically the value of B is arbitrary and a value of true for B would also be a satisfiable assignment.
This aligns with the assignments provided by the python program.



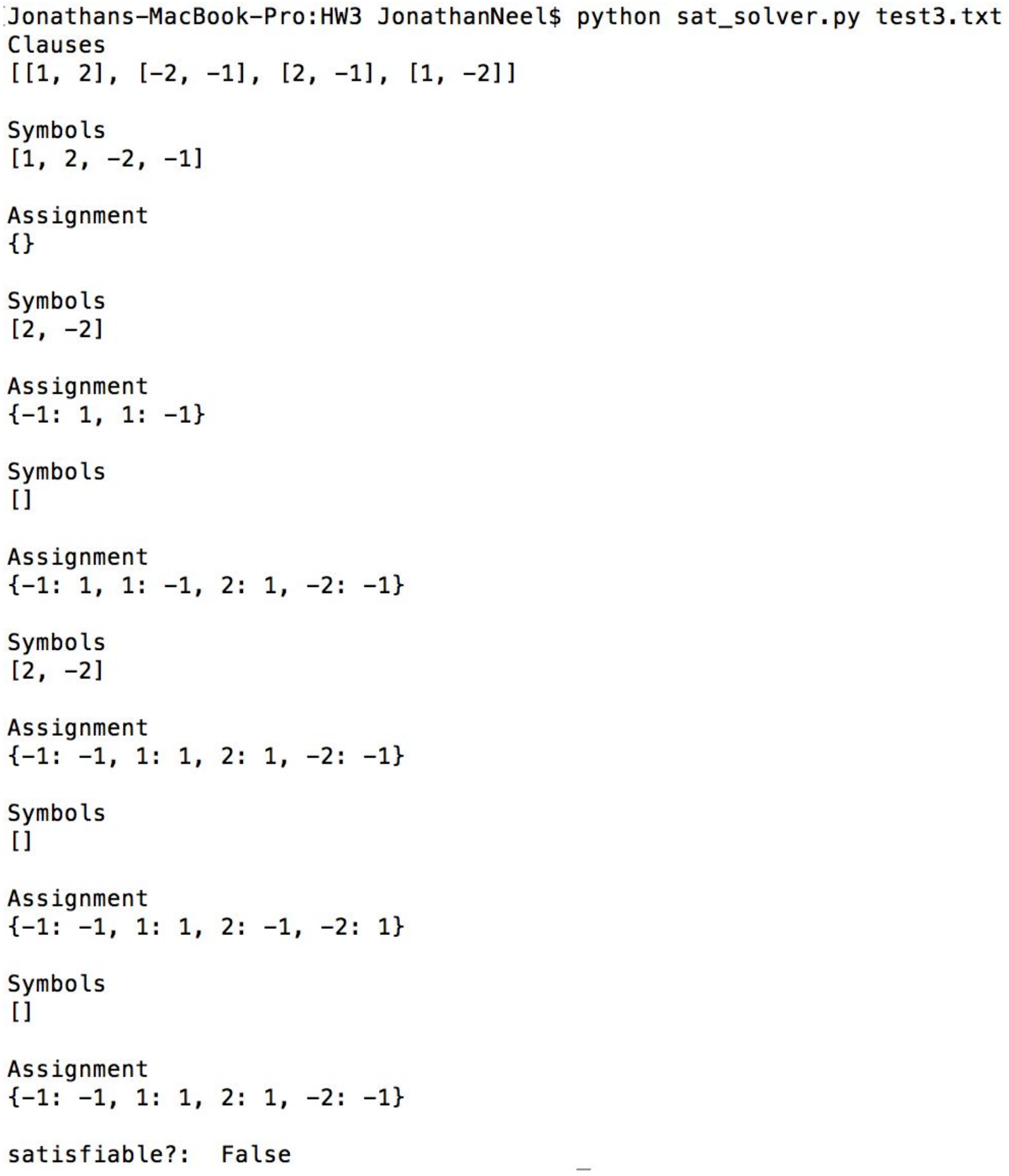
```
[Jonathans-MacBook-Pro:HW3 JonathanNeel$ python sat_solver.py test2.txt
 Clauses
 [[-1, -2, -3], [3, 1], [-3], [-1, -3]]

 Symbols
 [-1, -2, -3, 3, 1]

 Assignment
 {}

 Symbols
 [-1, -3, 3, 1]

 Assignment
 {-2: 1, 2: -1}

 Symbols
 [-1, 1]

 Assignment
 {-2: 1, 2: -1, -3: 1, 3: -1}

 Symbols
 []

 Assignment
 {-2: 1, 2: -1, -3: 1, 3: -1, 1: 1, -1: -1}
```

CNF 3:



```
c
c Random CNF in satcompetition.org format created by cnf_generator.php
c
p cnf 2 3
1 2 0
-2 -1 0
2 -1 0
1 -2 0
```

Let’s call 1 = A, 2 = B. Translating from the sat format, the cnf can be rewritten as:
(A | B) & (~B | ~A) & (B | ~A) & (A | ~B)
resolution:
R1: { A | B }
R2: {~B | ~A}
R3: {B | ~A}
R4: {A | ~B}
R5: {B} from R1 and R3
R6: {~B} from R2 and R4
R7: {} from R5 and R6
Therefore, since we were able to resolve an empty set, there is a contradiction and this CNF unsatisfiable.
Here’s the output of the program being run on this CNF. The output agrees with our resolution.



```
Jonathans-MacBook-Pro:HW3 JonathanNeel$ python sat_solver.py test3.txt
Clauses
[[1, 2], [-2, -1], [2, -1], [1, -2]]

Symbols
[1, 2, -2, -1]

Assignment
{}

Symbols
[2, -2]

Assignment
{-1: 1, 1: -1}

Symbols
[]

Assignment
{-1: 1, 1: -1, 2: 1, -2: -1}

Symbols
[2, -2]

Assignment
{-1: -1, 1: 1, 2: 1, -2: -1}

Symbols
[]

Assignment
{-1: -1, 1: 1, 2: -1, -2: 1}

Symbols
[]

Assignment
{-1: -1, 1: 1, 2: 1, -2: -1}

satisfiable?:  False
```