I, Cole Bardin, agree to adhere to the Drexel Academic code of conduct during this assignment; other than the instructor/TA/RCF and the materials provided by the instructor, no other resources were utilized.

Spring CS270: HW#3	**July 15 - Aug 1, 2023**

For this assignment, DrRacket is permitted, but any external website (Google, github, stackoverflow, Chat-GPT, chegg etc) is not allowed, other than the SAT Solver at http://logicrunch.it.uu.se:4096/~wv/minisat/ (and of course, bbLearn)

Q#6 has a small enough clause amt (30, if done correctly) that it can be written manually. For Question#7 though, you must write you OWN generator (otherwise you would be dealing with many hundreds of clauses manually).

This assignment has FOUR different portions to submit: there is this pdf which contains just questions 6-7, the SAT Solver portion of the homework. There is also a separate spot in bbLearn to turn in your source code for Question#7. There is also the first five questions, which are the Racket portion of the assignment, and that .rkt file is turned in at a separate location as well. There is also a section to submit the DIMACS .txt file for Question #7.

Question #6 [25 points]: A train station has seven tracks (denoted as Red, Orange, Yellow, Green, Blue, Indigo, and Violet) that can be set to either running East/West or North/South. Pushing a numbered button rotates exactly three different tracks 90 degrees, according to the following list:

BUTTON	TRACKS ROTATED
1	Red, Yellow, Indigo
2	Red, Orange, Blue
3	Red, Orange, Green
4	Green, Blue, Indigo
5	Yellow, Indigo, Violet
6	Orange, Yellow, Indigo
7	Green, Blue, Violet

Currently, all the tracks are orientated North/South. Your goal is to use the SAT Solver to determine which buttons you should push in order to get every track running East/West. If there is not a way to do it, then explain how you know it cannot be done. If there is more than one way to do it, use the SAT Solver to provide a second way; if there isn't a second way to do it, use the SAT Solver to justify how you know there cannot be another way.

`Make sure to provide all DIMACS along with comments about what each section represents`
`Unlike Question #4, because the DIMACS for this one is so short, you do not need to submit it as a separate file, you can simply copy paste it below (or add a page if need be)`

with variables 1-7 representing whether or not their respective buttons have been pressed

```
p cnf 7 30
-1 -2 3 0
-1 2 -3 0
1 -2 -3 0
1 2 3 0
-2 -3 6 0
-2 3 -6 0
2 -3 -6 0
2 3 6 0
-1 -5 6 0
-1 5 -6 0
1 -5 -6 0
1 5 6 0
-2 -4 7 0
-2 4 -7 0
2 -4 -7 0
2 4 7 0
-3 -4 7 0
-3 4 -7 0
3 -4 -7 0
3 4 7 0
-1 -4 -5 -6 0
-1 -4 5 6 0
-1 4 -5 6 0
-1 4 5 -6 0
1 -4 -5 6 0
1 -4 5 -6 0
1 4 -5 -6 0
1 4 5 6 0
-5 -7 0
5 7 0
```

To generate the DIMACS, for each color, I determined which combinations of buttons involving its color would result in that track being rotated 90 or 270 degrees. This allowed me to logically relate the button presses to each other and get the system into CNF

The colored boxes highlight which colored track the DIMACS line describes a valid combination for

For example: there are 3 buttons that can rotate the red track (1, 2, 3). This means that either exactly one of those buttons can be pressed OR all three of them must be pressed for red's track to be swapped to E/W. Taking those 4 DNF terms and converting to CNF, I get the clauses next to the red box

This returns the satisfied solution:
1 2 3 -4 5 6 -7 0
This means pressing buttons 1, 2, 3, 5, & 6 will turn all tracks from N/S to E/W

This is the only solution to this system. This can be verified by adding the line:
-1 -2 -3 4 -5 -6 7 0 to the DIMACS. This line forces the SATsolver to find a solution that differs by at least 1 term. However, the verdict is UNSATISFIABLE when this line is appended to the DIMACS. This means there is only 1 solution:
1 2 3 -4 5 6 -7 0

Question #7: The Car Race Puzzle

[parts BCFGHIJ are worth 2 pts each, part A is 1 pt, and parts D & E are worth 5 pts each, for a total of 25 points]:

*There are N race cars on a circular track, and each car is painted with a unique label number 00, 01, 02, … N-1 on its hood. This particular car race is just two laps around the track. A car is said to be "***label-shifted***" iff its change in standing in the race between the first lap and its finish is exactly the same as its label. As an example, suppose car #03 after the first lap was in 5th place. If when it crosses the finish line of the second lap it was in either 2nd place or in 8th place, then that means car#03 was label-shifted (2nd place is 3 places better in the order than it was in 5th place, and 8th place would be 3 places worse than it was after the first lap; either one would be considered being label-shifted for that car, since its number was 03). A race is called "***amazing***" if every single car in that race ended up label-shifted.*

Here is an example of an amazing race that happened with N=4 cars:
at the end of the first lap, car#02 was in first place, car#00 was second place, car#01 was third place, and car#03 was fourth place. In the end, when the cars finally crossed the finish line of the 2nd lap, the final order was that car#03 was in first place, car#00 was second place, car#02 in third place, and car#01 in fourth place. We can abbreviate this race in short hand by simply writing the car number orders for the midpoint of the race and the finish line: (2,0,1,3)↠(3 0 2 1).

We can confirm that this example is an amazing race by verifying every car was label-shifted: Car#00: |2 – 2| = 0 (i.e. was in 2nd place and stayed in 2nd place), Car#01: |4 – 3| = 1 (went to 4th place from 3rd place), Car#02: |3 -1| = 2 (went to 3rd place from 1st place), Car#03 = |1 – 4| = 3 (went from 4th place to 1st place)

[a] Note that this is not the only possible N=4 amazing race. Confirm that (1 2 0 3) ↠ (3 1 0 2) is also a four car amazing race.

Car#0: |3-3| = 0
Car#1: |1-2| = 1
Car#2: |2-4| = 2
Car#3: |4-1| = 3

All car numbers equal their change in position.
Meaning this is an 'amazing' race

[b] Manually find an amazing race with N=5 cars (there's more than one, but just finding one is good enough for this part)

(4 1 2 0 3) -> (2 3 1 0 4)

Car#0: |4-4| = 0
Car#1: |2-3| = 1
Car#2: |3-1| = 2
Car#3: |5-2| = 3
Car#4: |1-5| = 4

Try to find an amazing race with N=6 cars (i.e. car labels of 00 through 05) – having difficulty? Let us show how to use the SAT Solver to establish that it's not possible to have an amazing race with 6 cars. The first step is to define our propositional variables. There are many ways to do this, but we'll use the following convention: **var# = 36*(Lap# - 1) + 6*(Position# - 1) + Car# + 1** (if you're curious, the + and – ones are needed since DIMACS and this problem are one-indexed, while the 36 and 6 are coming from the fact that we are essentially using base N=6). Notice that this is a grand total of 72 different variables, starting with 1 ("car#00 is in first place during lap1") up to 72 ("car#05 is in sixth place at the end of lap2")

As an example of using this variable numbering system, suppose we wanted to express the proposition that "After the first lap, car#03 was in fifth place". So, Lap=1, Position=5, Car=3, and plugging that into our equation we see that in DIMACS this would be represented as the variable 28 [since 36*(1-1)+6*(5-1)+3+1=28]. Similarly, you should make sure you can "decode" a variable into its meaning. For instance, the variable number 53 would translate to the proposition "car#04 was in third place" [this is because numbers >36 refer to the second lap, and (53%6)-1=4 would be the car# etc].

[c] verify your understanding by calculating the variable number for "at the finish line of the second lap, car#05 came in fourth place". Also, give the English translation of the variable 12

36*(2-1)+6*(4-1)+5+1 = 60

12 = 36*(1-1) + 6*(2-1) + 5 + 1
after the first lap, car 5 was in 2nd place

[d] It is vital that you do not try to write out all the necessary DIMACS clauses manually (there will typically be thousands); if you do, you will almost surely forget a case or make a typo. Instead, you should write a script to generate it for you; it does not have to be in Racket (in fact, that would be pretty hard since we haven't learned how to do files yet). Instead, you can use any language you are most comfortable with (C++, Java, Python etc.). Just make sure that you thoroughly comment your code and include the source file with your homework submission. The most delicate part of getting a SAT Solver to produce valid answers to puzzles is encoding the implicit restrictions that are so obvious to human beings that they go unstated, however it is still critical that the computer is informed of them. Part of the art/skill of programming is being able to recognize these and specify them in DIMACS. Since this is likely your first experience with this, we will walk you through the process for this problem.

1. **cars don't disappear**. That is to say, that car#03 (for instance) must be somewhere in the race. It didn't fly away. So car#03 is either in first place, or second place, or third place etc. It can't be nowhere. To say that for lap1 in DIMACS would be: 4 10 16 22 28 34 0, and of course it also needs to be true for lap2: 40 46 52 58 64 70 0 (since the number of places might vary in general, you should actually write a loop to generate the line). Moreover, that's not just true for car#03, that was just one example. There would be a similar pair of lines for EVERY car in the race (so this would actually be a nested loop). When done correctly, this part (for N=6) would have 12 clauses with 6 positive integers in each.

2. **a single car can't be in two places at once.** E.g. car#04 can't both be in first place and second place at the same time. In logic terms, at least one (possibly both) of those statements needs to be false. Thus in DIMACS, for Lap1, this would be -5 -11 0. Of course, you need to say this for every car, for each pair of possible positions, and for both laps. When done correctly, this part would have 180 clauses with 2 negative integers in each.

3. **two cars can't be in the same position at the same time.** E.g. car#01 and car#04 can't both be in fifth place for the same lap. Similar to the previous logic, at least one (possibly both) of those statements needs to be false. Thus in DIMACS, for Lap2, this would be -62 -65 0. Of course, you need to say this for every pair of cars, for each position, and for both laps. When done correctly, this part would also have another 180 clauses with 2 negative integers in each.

This is all often referred to as the "boilerplate" part of the code and it comes up in many numerous situations, not just car racing. It's just the basic set up expressing the physical realities of the given scenario (we'll tackle the "puzzle portion" of trying to come up with an *Amazing* race in the next part of the homework). For now, just write source code for this boilerplate and attach it as a separate file (you do not need to copy it here into this pdf). Note that it is important that your program doesn't have "6" cars hard coded in; your program should ask the user to input N, and then generate the DIMACS for N amount of cars. (At this point, when the user enters N=6, it should create a total of 372 clauses for the boilerplate).

Be sure to upload your source code into the section of bbLearn entitled "Race Car Source Code". Inside your code, it should be clearly commented which portions handle the boiler plate (question D), and which handles the "Amazing" criteria (question E).

[e] Now we move beyond just coding the boiler plate physical obvious part of the car race and need to finally tackle the true tricky part of the puzzle – making sure it is an *Amazing* race. To do this, we need to write DIMACS that ensures every car has been label-shifted. As an example, of how to do this, let us consider car#02 and suppose in the first lap it is in third place. In that case, that forces car#02 to be in either first place or fifth place at the end of the second lap. This is exactly the sort of situation that → is perfect for. In logic, we'd write 15 → (39 ∨ 63). But since → isn't legal in CNF (which DIMACS requires), we need to rewrite it using our Boolean rules as ¬15 ∨ (39 ∨ 63), and this can now be expressed as the DIMACS clause -15 39 63 0. This has to be done with every car for every possible position in lap1 (note that due to the set up of our boilerplate and the definition of label-shifting, it suffices to only focus on lap1→lap2 and not the other way around too). If you do it properly, the N=6 case should result in having another 36 clauses mostly with one negative and two positives; note however that not all the clauses have two positives. E.g. if car#02 had been in fifth place in lap1, then the only possibility for its position in lap2 would be third place (since there is no "seventh" place in a 6 car race – make sure that your program checks for this!). And in fact, there are certain situations which cannot happen (e.g. in an Amazing race it is flat out impossible for car#04 to be in third place at lap1, since there would be no legal position for it to be in after lap2, thus in this case your code should simply make the short clause -17 0)

Your program should now be able to write to an output text file everything you need to ask the SAT Solver to make an Amazing race. When N=6, you should make sure that the first line of your text file says "p cnf 72 408" (those specific numbers should not be typed in, your program should compute them).

[f] Run your program setting N=5. Copy the results into the DIMACS window at http://logicrunch.it.uu.se:4096/~wv/minisat/ (link is also in bbLearn). At the bottom of the page are a string of red numbers. The negative integers are false (so no need to translate those), but you should translate the positive ones. Write down the result using the short hand described earlier. Does this match the one you found (it should, although possibly in reverse order)

[g] Now run your code on N=6 and put the DIMACS into the SAT Solver. What does it report? What does that mean in the context of this problem?

[h] Use the SAT Solver to answer the N=7, 8, and 9 car version of this problem. If an amazing race for an amount is impossible, then say that. But if it is possible, then give the shorthand version of the solution `For N=9 only, include the .txt file of DIMACS that your code generated, and upload it to bbLearn in the "Nine Car DIMACS" section.`

[i] Do you think there is a different solution for N=9 other than the one the SAT Solver found? See if it can find one in which car#00 was in third place in the first lap (if the solution you found already had car#00 in third place, then see if you can find out with it in fourth place). Do you think there is a solution with car#00 in first place? Why or why not? (Hint: you don't need the SAT Solver for this last question, see if you can just reason it out).

[j] Do you have any conjectures as to which N's are solvable and which aren't? (You don't need to prove your answer, just give an educated guess that matches the evidence you've seen so far).