ECEN/CSCE 689

Introduction to Formal Verification

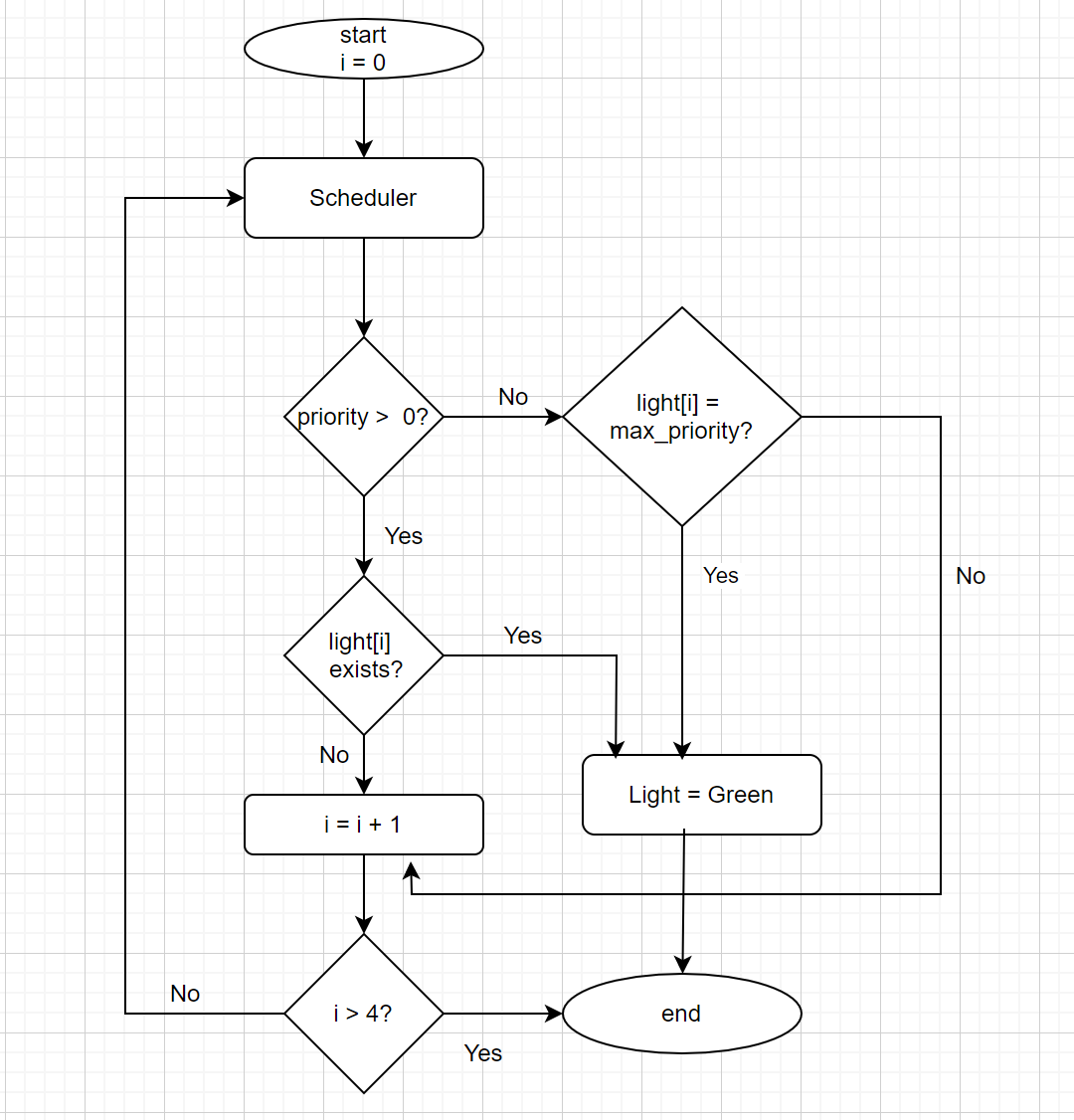
Project Phase A

Team ID: 10

I-Group

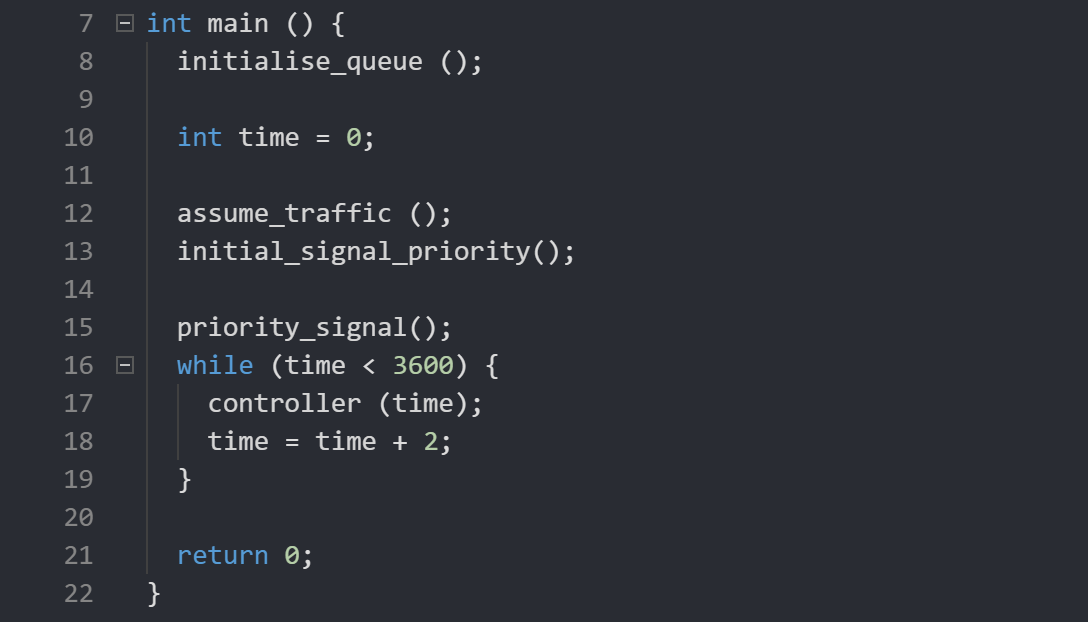
Architecture

**Algorithm**



The above flowchart is for a single junction.  
  
The traffic system has 9 junctions. Each road segment towards the junction has a signal light, which means that there can be maximum of 4 lights at a junction. The lights only switch between red and green. Currently, it is assumed that the light switching happens every 2 seconds. This time is a configurable parameter and can be changed later to maximize the vehicle throughput during the integration phase. Also, vehicle traffic is assumed randomly to check the number of collisions, number of u-turns and number of red light violations.

The light switching is mainly based on two scenarios: one with priority and one without priority. As seen in the above flowchart, scheduler checks if there is any priority assigned to any signal light at a junction. If there is no priority, it switches the first light in the queue to green and returns. At the next time instance, first light is switched to red and the second signal light is turned to green and so on. If there is any priority assigned to any signal light, then the corresponding light is turned to green and all other lights remain at red. How each signal obtains priority and how the queue is maintained will be shown below.



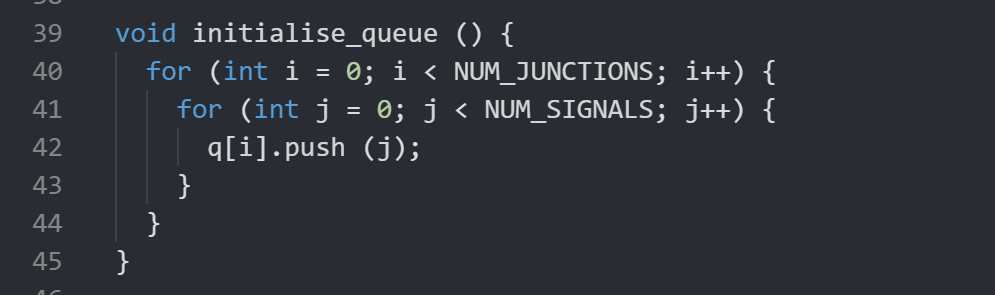
In the main function, queues are initialized which stores the signal light information about which light should be turned to green at each time instance.

Random traffic assumption is made in the assume\_traffic () function.

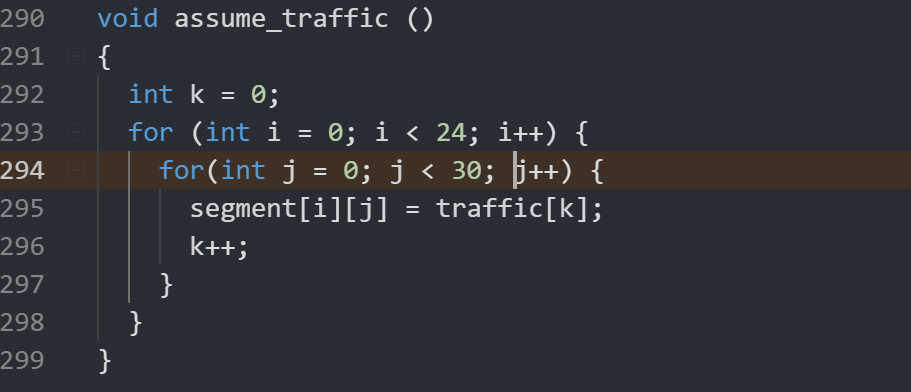
In the function, initial\_signal\_priority (), one light at each junction is turned to green, which gives us the initial signal information.

The priority\_signal () function assigns values from the road segments to each light in the junction, which is checked to decide which signal light is to be given the highest priority.

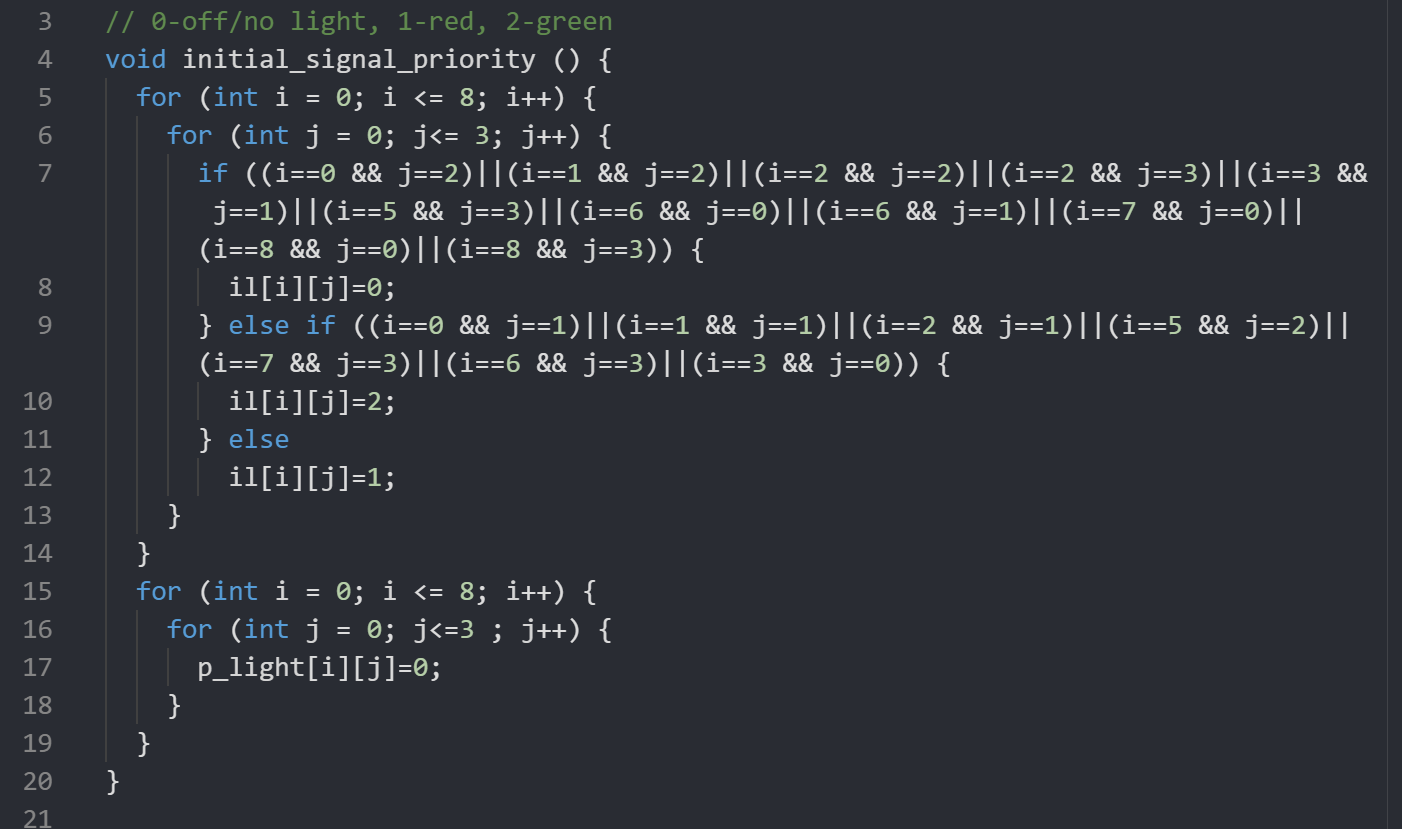
controller () functions manages light switching for every junction in the traffic system. This is called every 2 seconds and the process is continued for 3600 seconds, which is 1 hour.



There are 9 separate queues, one for each junction, and the four lights are pushed to queue in order. Queue “q” is a global variable, which is shared by all functions.

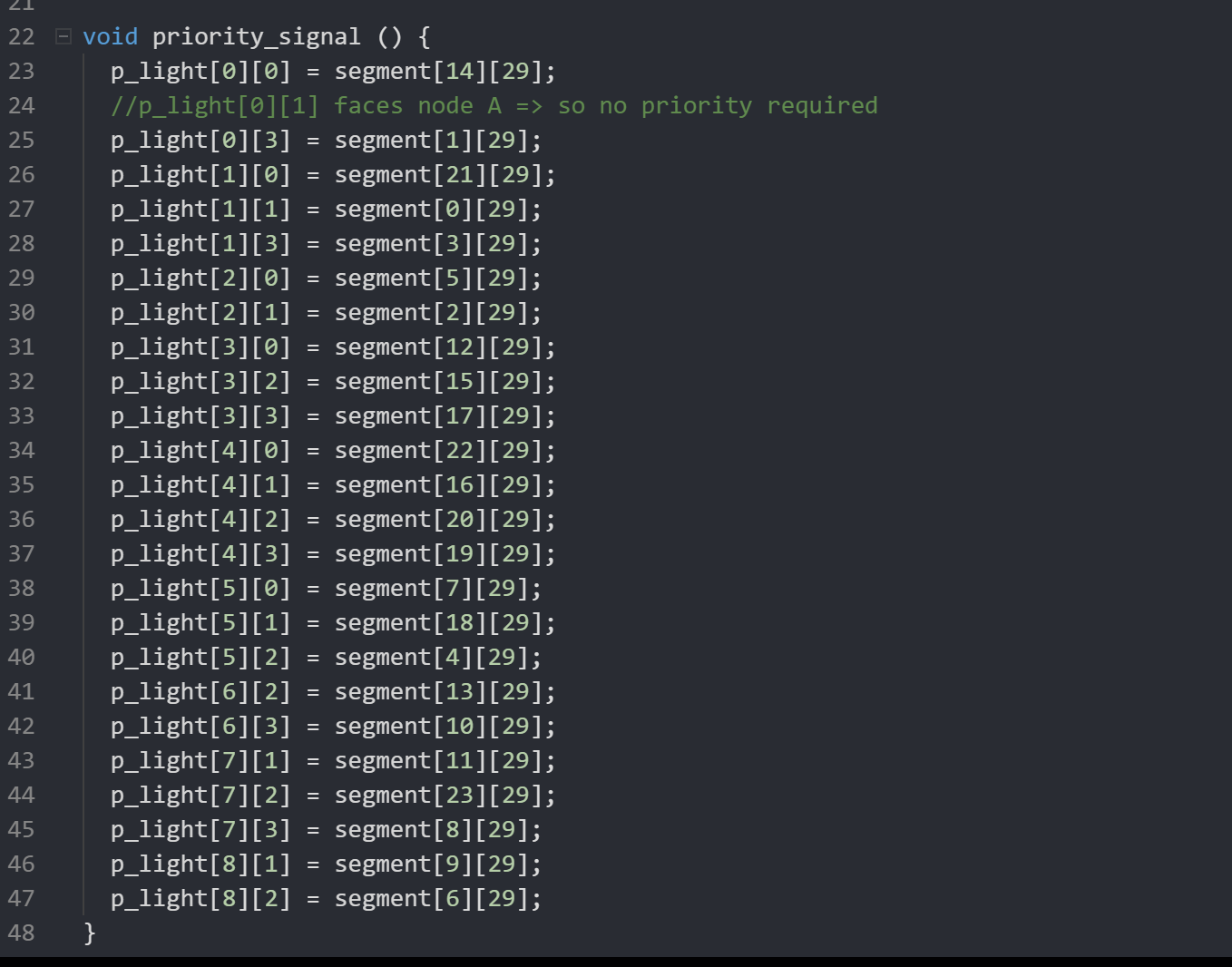


In this function, random vehicle traffic is assumed based on the values given in traffic[] variable. segment[24][30] is a global variable shared between i-group and v-group as described in the architecture section. During Phase B, segment[i][j] value is obtained from the actual vehicle movement.



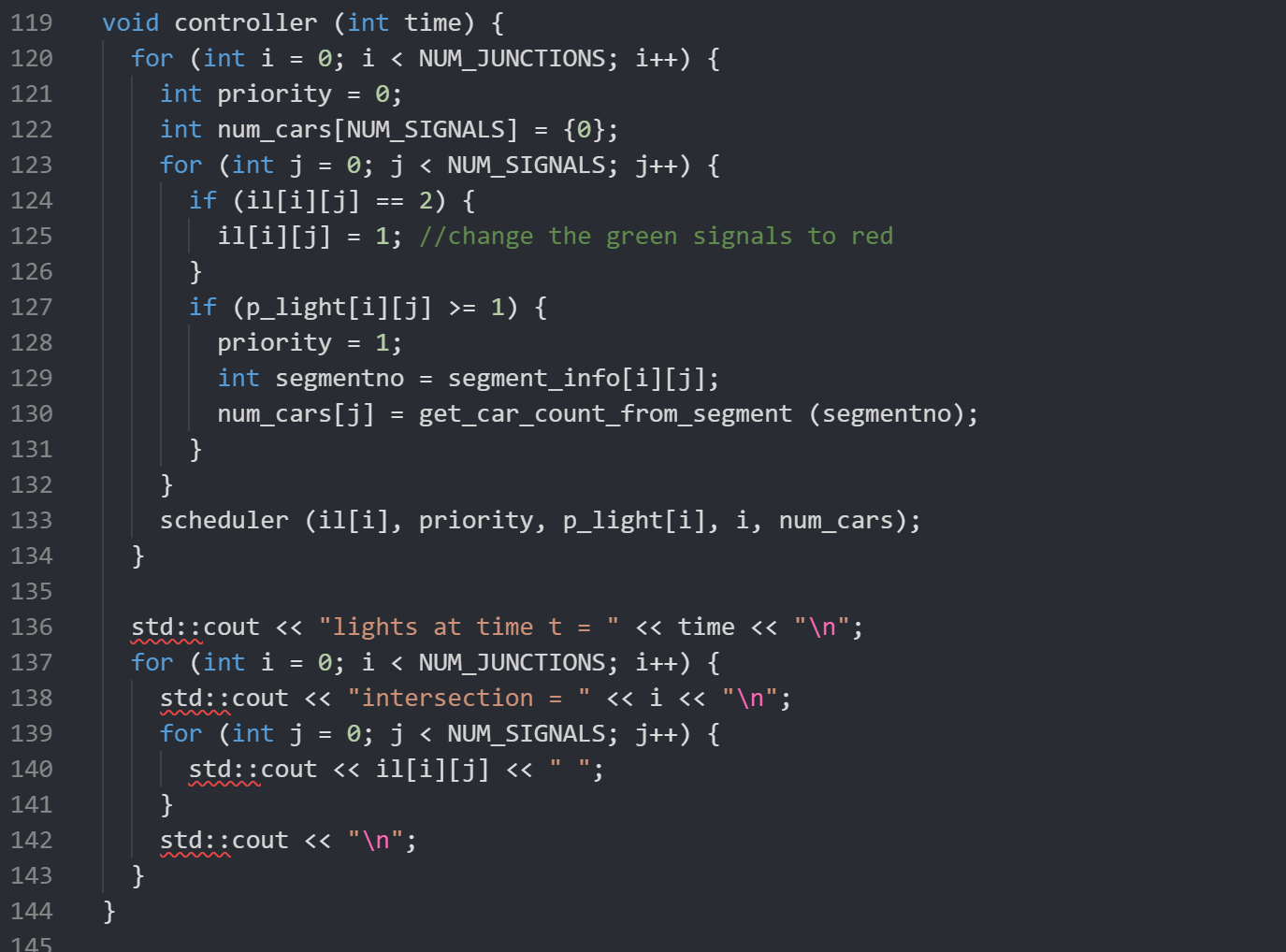
In this function, at each junction, one signal light (i.e “il” variable in the function) is kept as green and rest of the lights are red. The value 0 indicates that the signal light is not present (Light is present only if there is a road segment opposite to it), value 1 indicates that the light is currently red and value 2 indicates that the signal light is green.

p\_light variable indicates priority for each light, and it is initialized to 0, as there is no vehicle traffic initially.



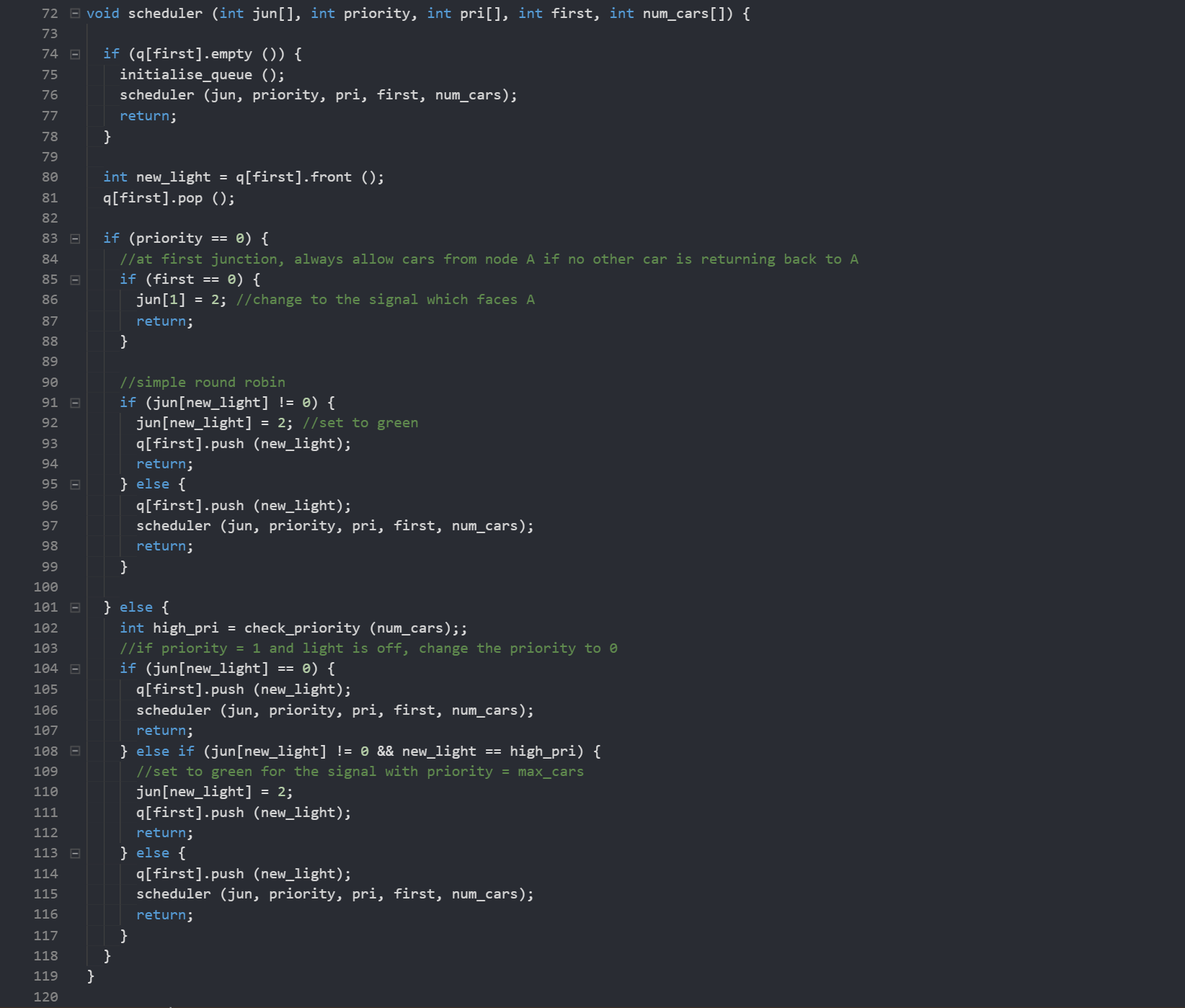
priority\_signal () assigns priority for each light based on the segment corresponding to it. The car id present at the last position of the segment is assigned as the priority.

For example, if segment 14 has car number 32, then the priority assigned for light il[0][0] is 32.



Controller handles the signal light switching and prints the changed values of lights at each junction. Initially, the green lights in each junction are changed to green, which avoids the green light violations (i.e., multiple lights are green at the same junction/intersection at the same time).

The p\_light value greater than 0 implies that there is a vehicle at the 29th or last position of the corresponding segment. If p\_light is greater than 0, then the number of cars is counted from segment (in positions 18 to 29, last 10 positions). This is to understand the priority of each light. After having this information, scheduler function is called, which decides which light is to be turned green based on the priority information for each junction.



Scheduler switches the light signals based on two scenarios: one when there is no priority assigned for any light and other when there is some priority for any of the light.

When there is no priority for any light in the signal, simple round robin algorithm is followed. This means that, the light switching happens every two seconds in a circular manner.

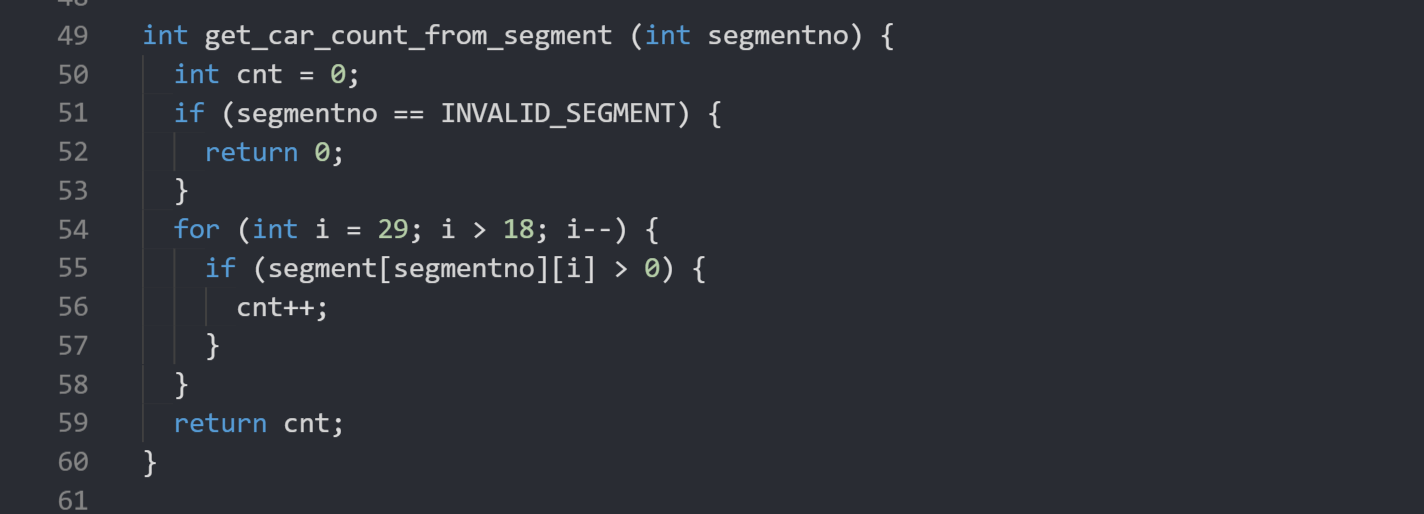
For example, if light0 is green at time = 0, then at time = 2, light1 will be turned to green and so on until light0 is turned back to green at time = 8. Each light is turned green alternatively one after the other.

Here, one special case is considered for the junction 0 or the first junction. In this junction, vehicles are coming from node A at most of the times. Hence the light corresponding to that segment (i.e., il[0][1]) is kept at green. Only when there are vehicles that are returning to A, the other lights are turned green. This helps in maximizing the number of vehicles generated.

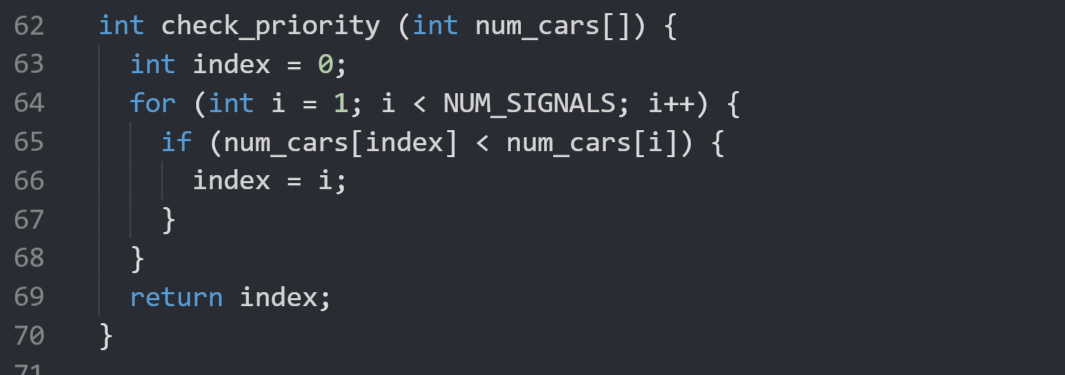
new\_light contains the value popped from queue. If new\_light is 0, it pushed back to queue and scheduler is called again. Recursive technique is being used to make sure each light is being checked. If new\_light is not 0, then the light is changed to green and pushed back to queue. Queue is First In First Out, hence when the light is pushed to queue, it is added at the back of the queue. This makes sure that, at the next time instance, the next light at the intersection is changed to green.

When priority is not 0, it means that there are vehicles at one or more segments of the junction. If vehicles are present at the corresponding junction for the segment, that light is given the highest priority. But if more than one segments have vehicles in their last position, there is conflict for which light needs to turned green. This conflict is resolved by the check\_priority () function (Assigns priority based on number of cars at each segment).

After knowing which light has highest priority, that light is turned green and pushed to back of queue in the same way as above. Other cases like new\_light is zero and if new\_light doesn’t have the highest priority, then it is pushed back to queue and scheduler is called in recursive manner.



In the get\_car\_count\_from\_segment, segment number is received as input and the number of cars in that segment from 18th to 29th positions (last 10 positions) is counted and returned as output. Last 10 positions are considered as these are near to the light and hence they reach the last position much faster, when compared to the other cars in the segment.



check\_priority () is a simple function (same as finding maximum element in an array), which calculates the segment which has highest number of cars and returns the index for that. This index is the light with highest priority.

Source Code

Results

Testcase 1: No vehicle traffic

In this testcase, no traffic is assumed for the entire duration of 3600 seconds. In this case, the lights at a junction, switch to green one after the other every two seconds.

Testcase result for the entire duration can be found in <https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/notraffic_output.txt>

Switching of lights at every time instant is visible in <https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/signal_gif.gif>

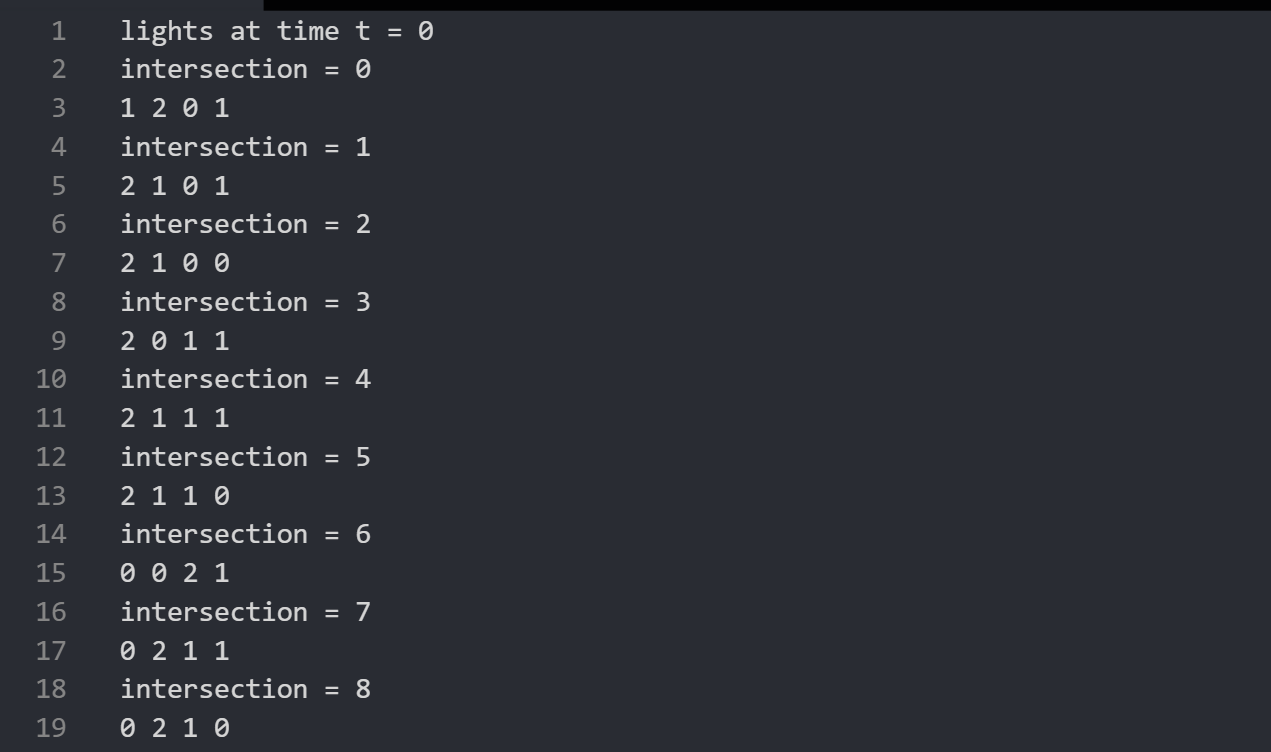
All the output plots are in the link <https://github.com/rishyasankar/Formal_verification_project/tree/main/PHASE%20A%20SUBMISSION/i-group/output%20plots>

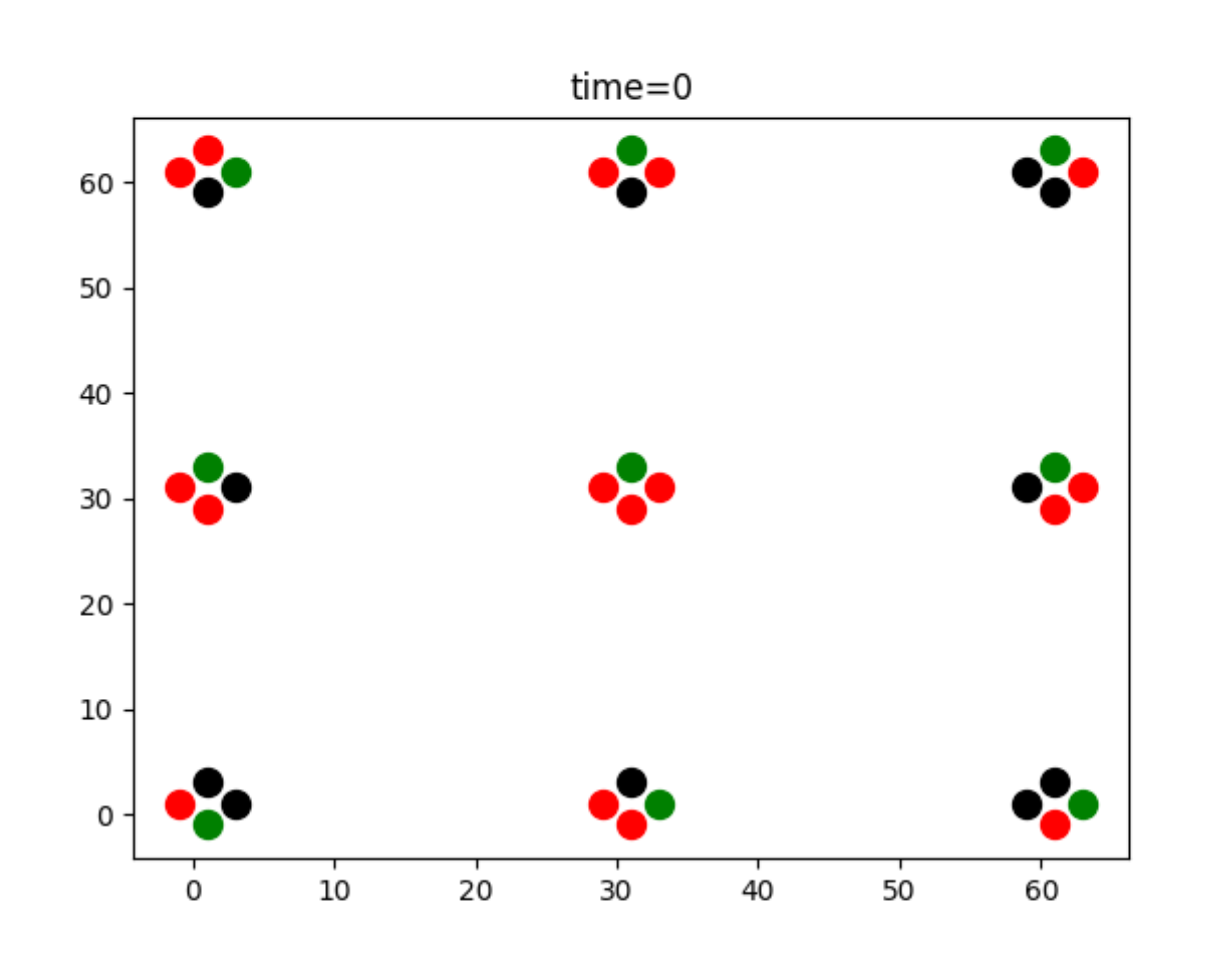
(plots with names notraffic\*.png)

At time t = 0

For intersection 0, always the light opposite to node A is green. This is not changed at any time instance since there is no vehicle traffic.

For intersection 1, it is seen that light 0 is green.





The black dots indicate that there are no lights at those places (il[][] = 0).

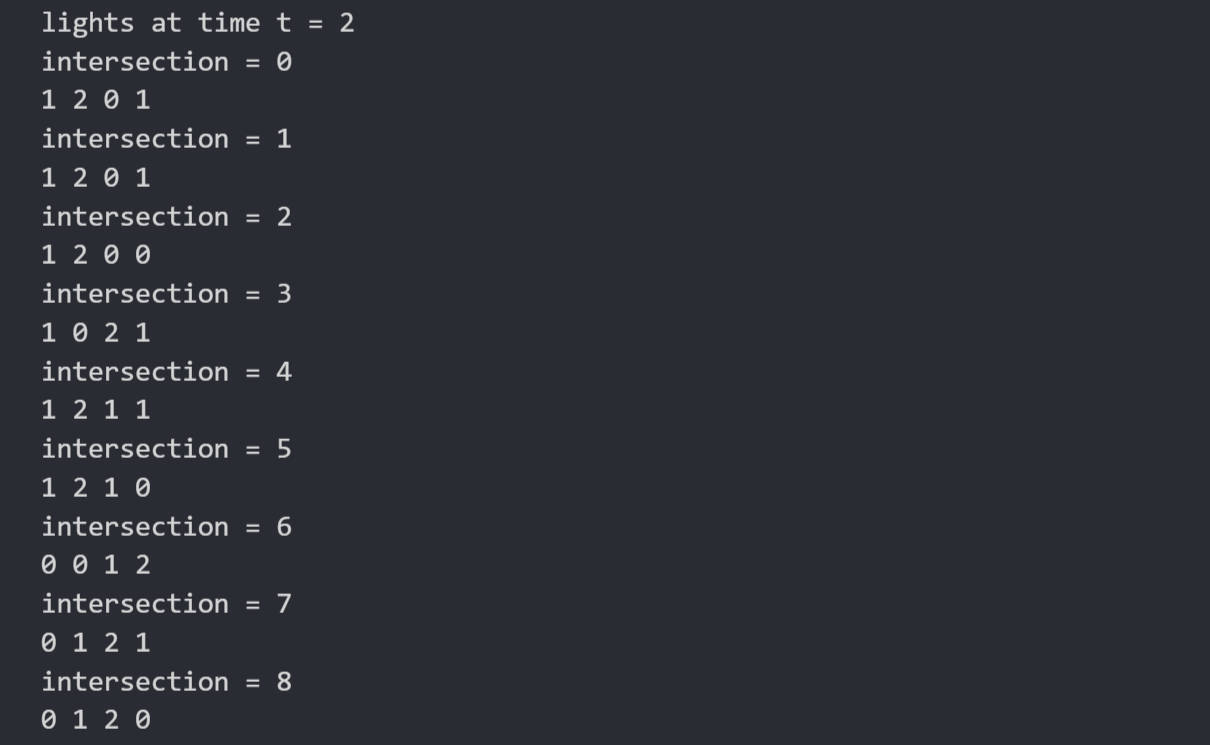
At time t = 2,

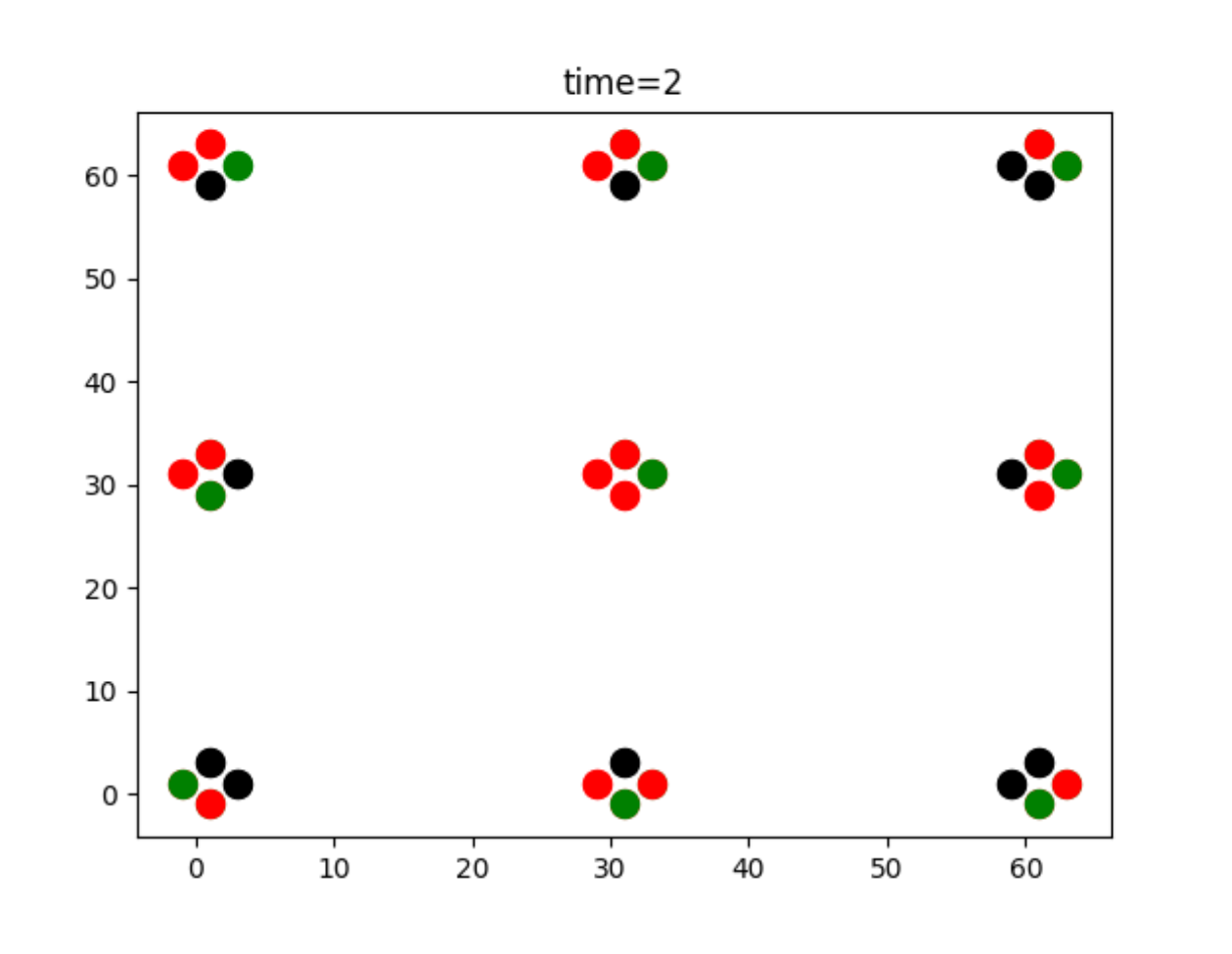
For intersection 0, the light opposite to node A , which is light 1 is still green.

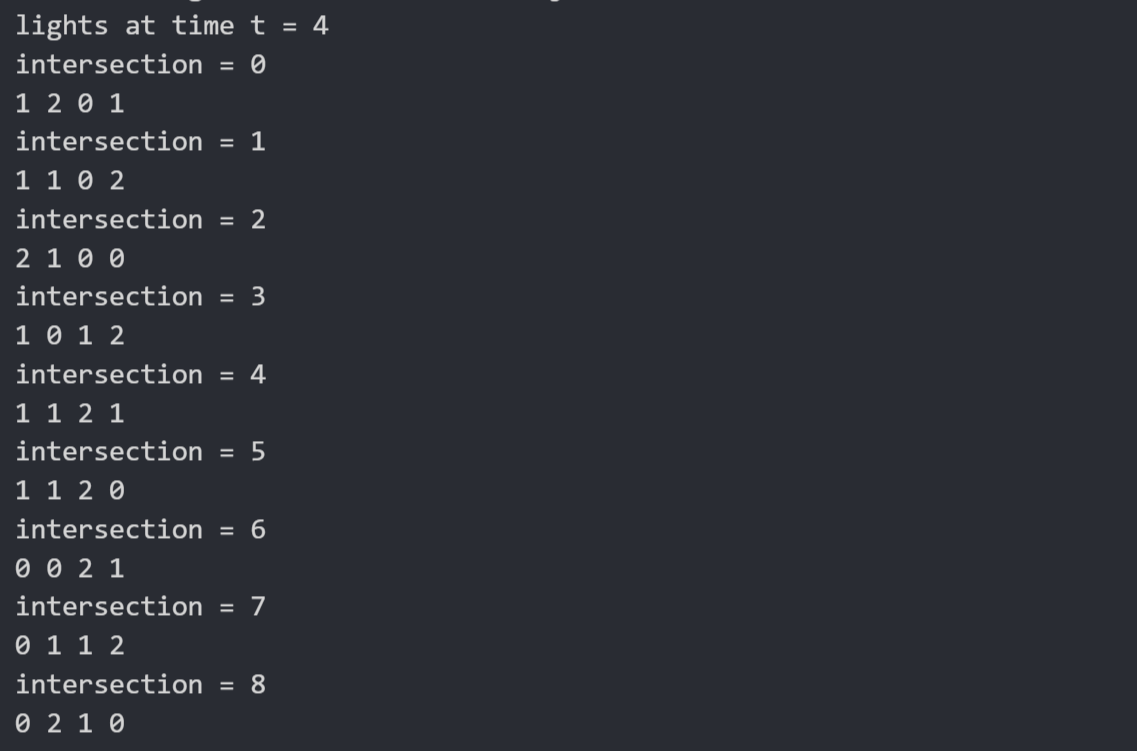
At intersection 1, light 0 is turned to red and light 1 is switched to green.

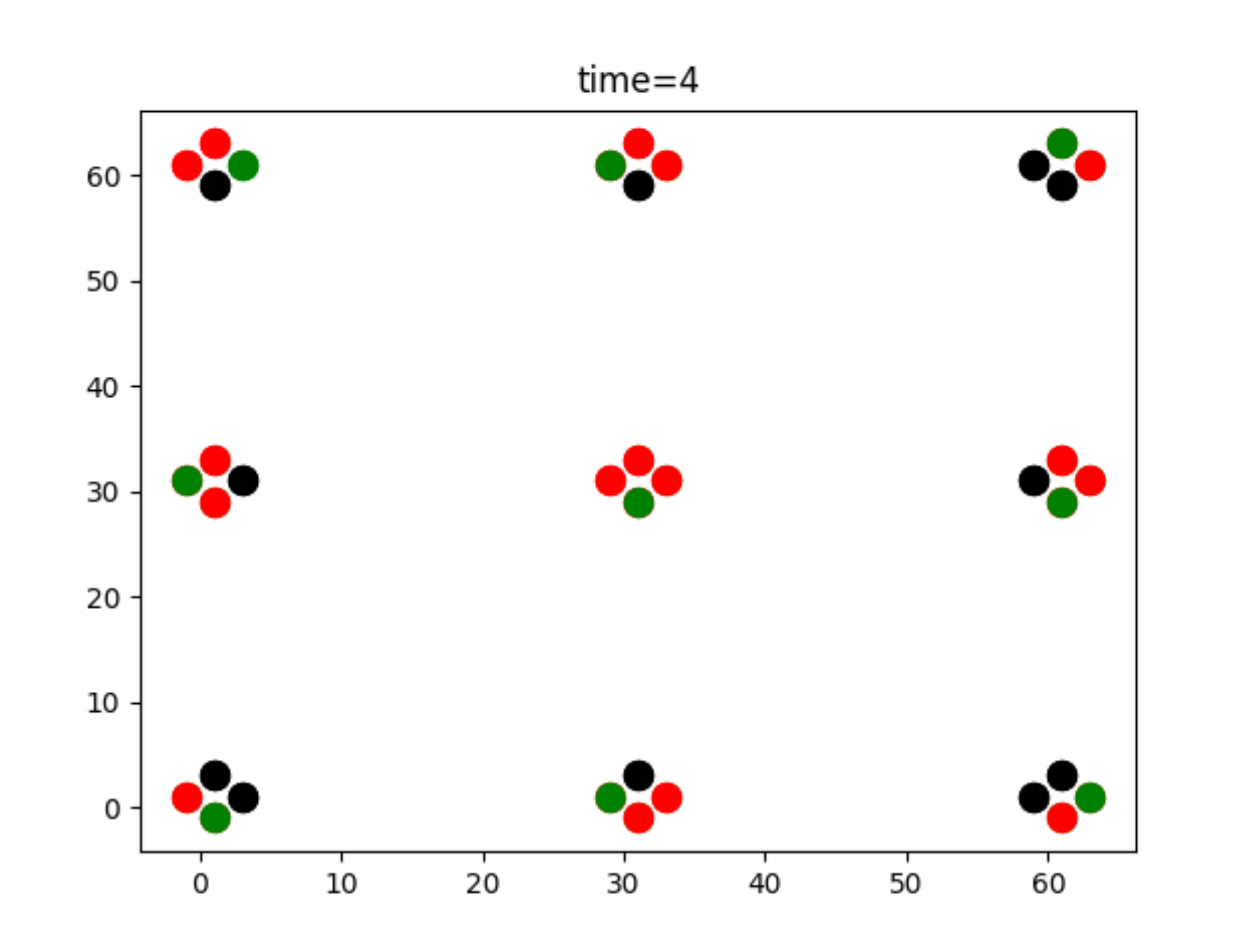
The same behavior can be observed at all the intersections. And the same continues at all times.

Screenshots for time t =2 and 4 are attached below.









Testcase 2: Random traffic

In this case, same random traffic is assumed at all time intervals.

Light switching at every junction happens based on priority.

Testcase result for the entire duration can be found in <https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/traffic1_output.txt>

Switching of lights at every time instant is visible in <https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/signal_traffic_gif.gif>

All the output plots are in the link <https://github.com/rishyasankar/Formal_verification_project/tree/main/PHASE%20A%20SUBMISSION/i-group/output%20plots>

(plots with names traffic\*.png)

Assumed vehicle traffic is

{0,0,0,24,0,23,0,22,0,21,0,0,0,19,0,18,0,17,0,0,0,15,0,0,0,0,0,12,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,9,0,8,0,0,0,6,0,5,0,4,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,10,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,25,0,0,0,0,0,0,0,0,0,20,0,0,0,0,0,0,0,16,0,0,0,14,0,13,0,0,0,11,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

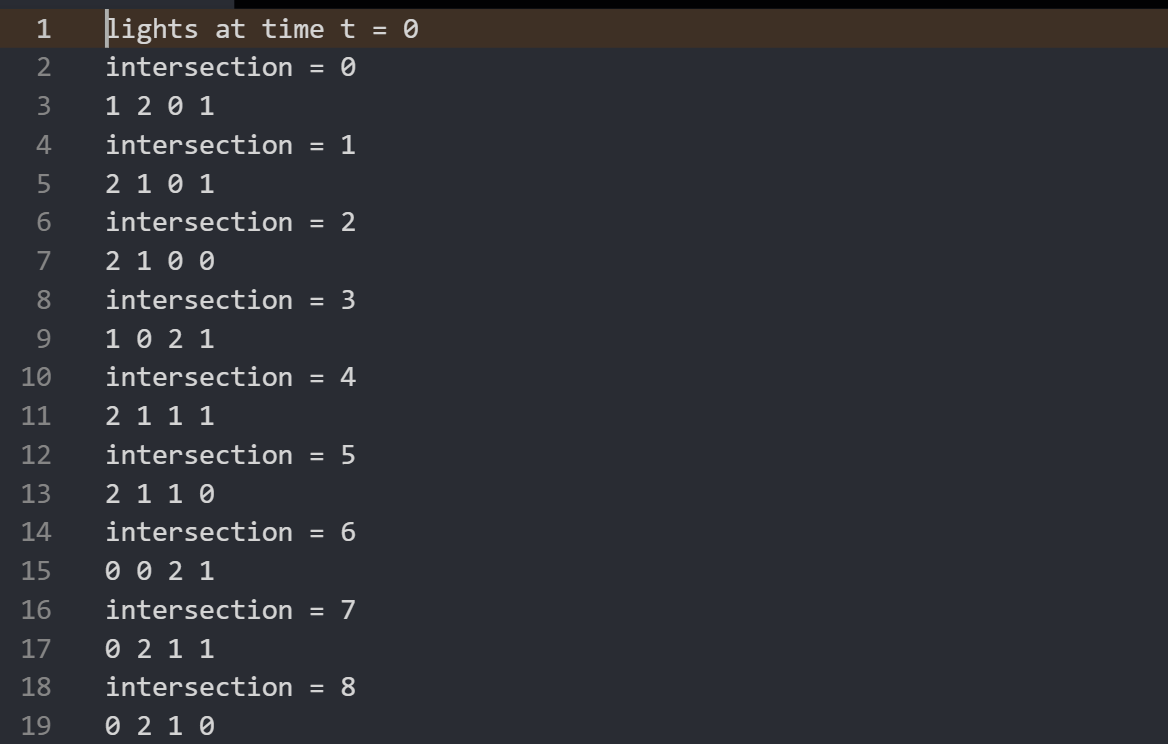
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

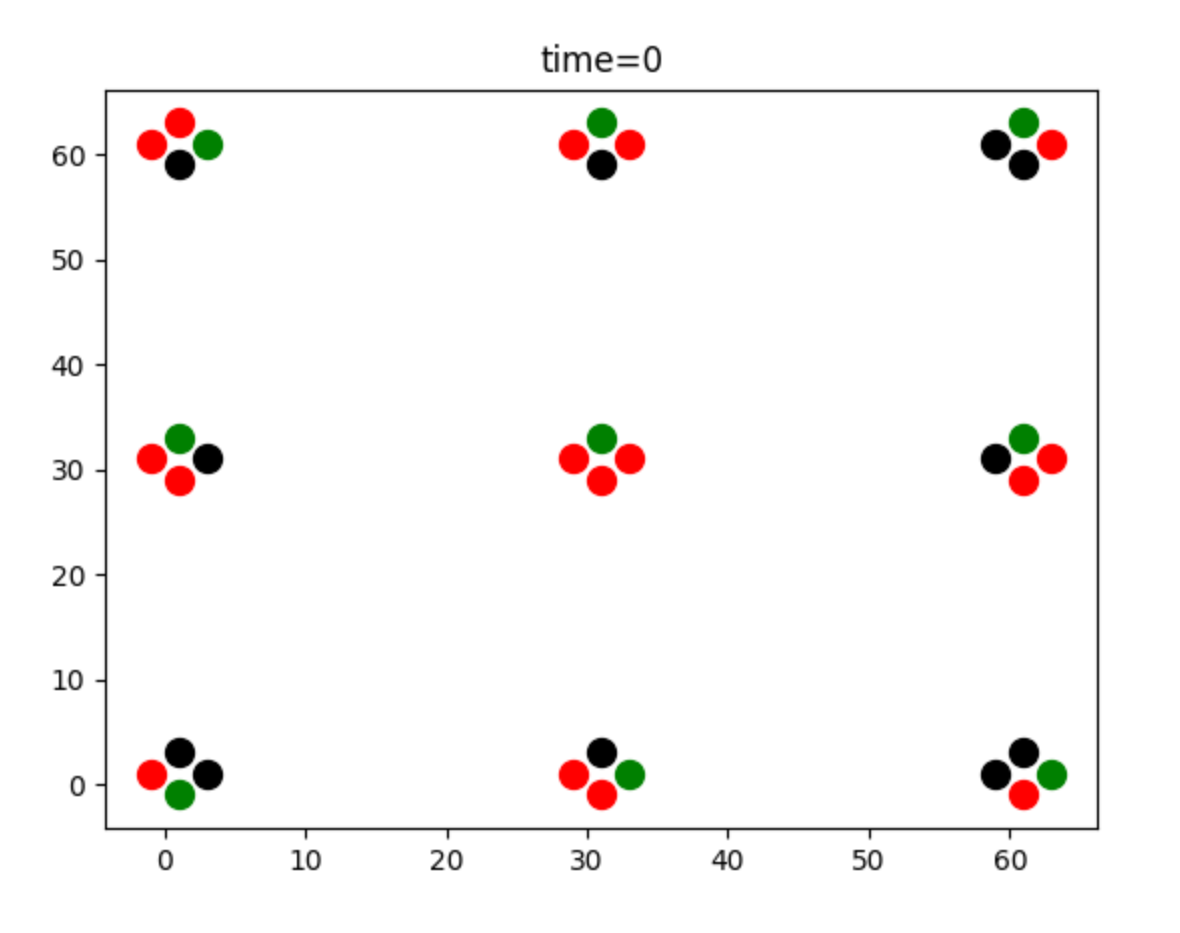
Car 11 is at 29th position of segment 15.

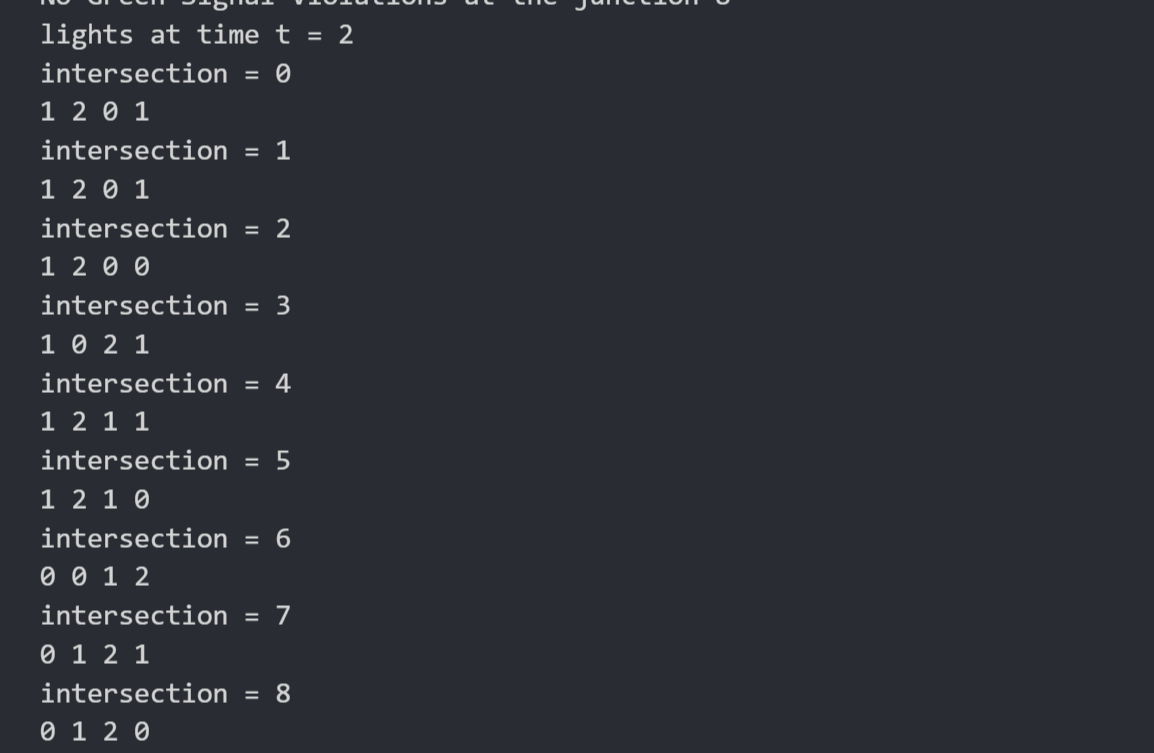
From the architecture diagram, it is seen that segment 15 is opposite light 2 of intersection 3.

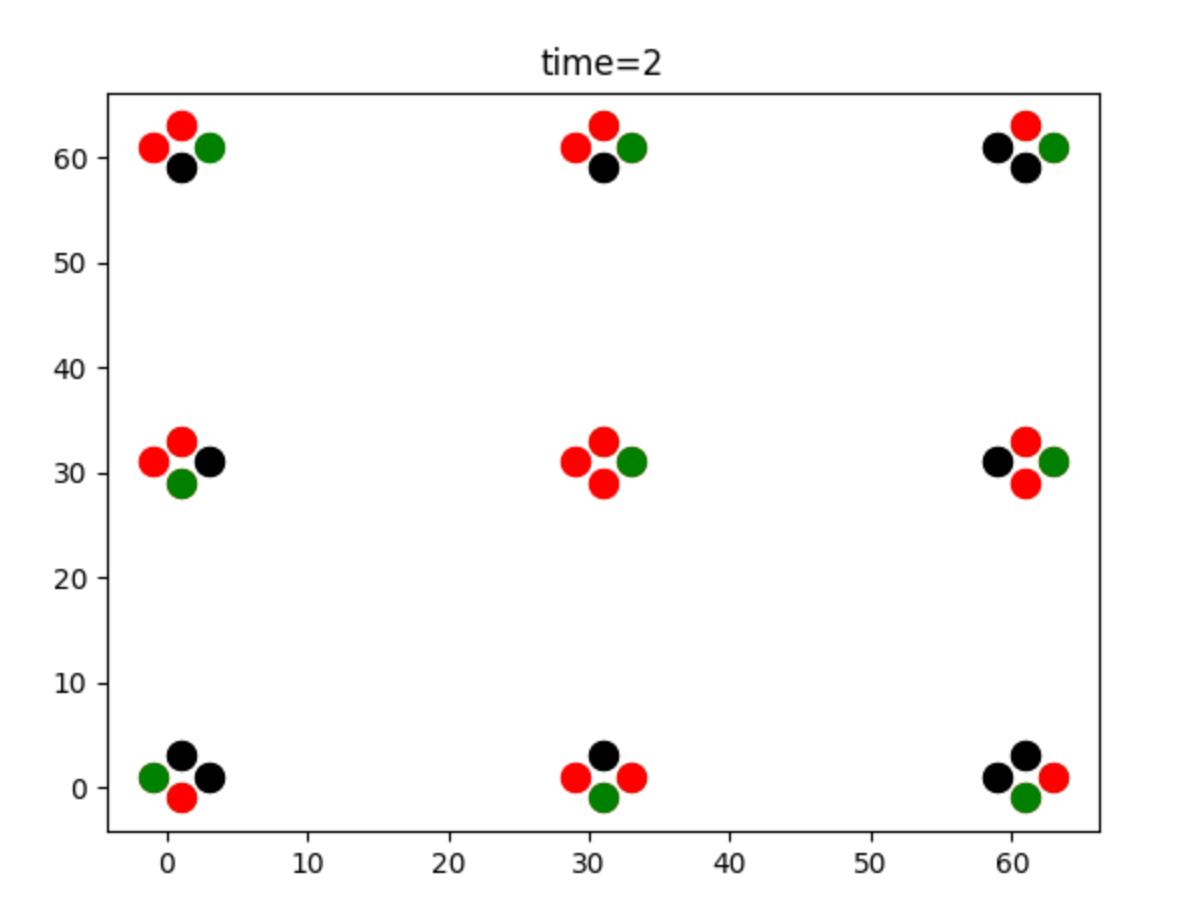
So, at every time instant the light 2 of intersection 3 should be green, so that the vehicle can cross the junction.

In the attached screenshots for time t = 0 and 2, it is clearly visible the above mentioned light is not switching, where as all other lights are switched to green alternatively in the same as no traffic schenario.





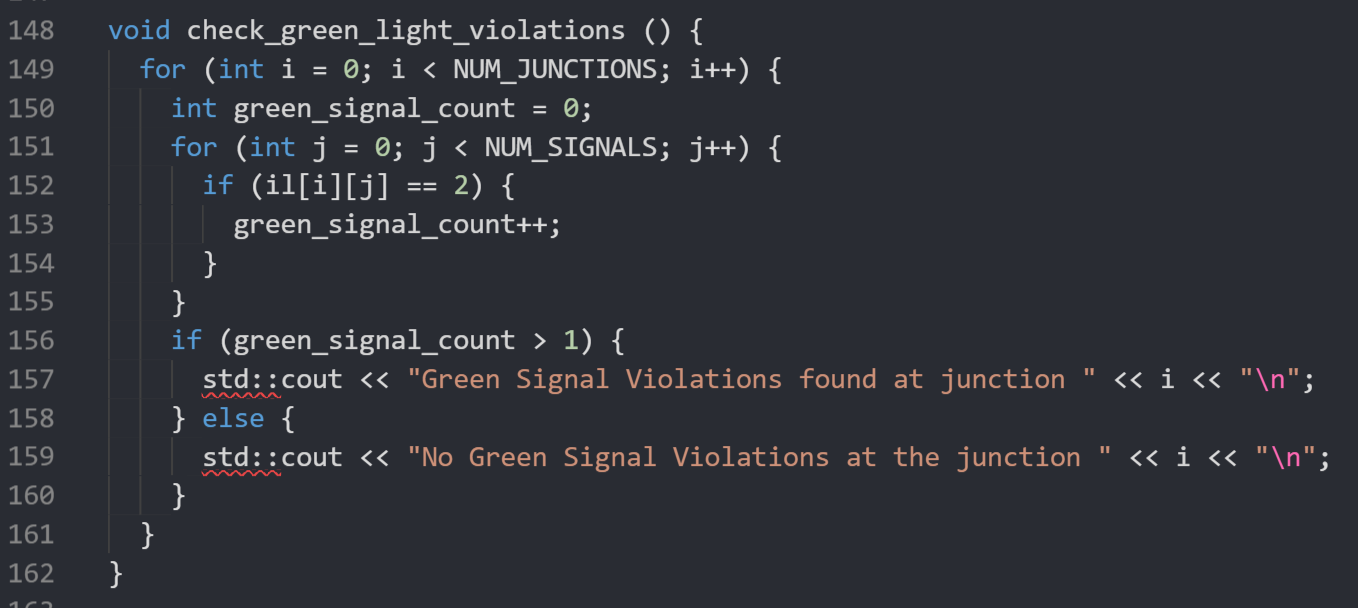




Testcase 3: Green Light Violations

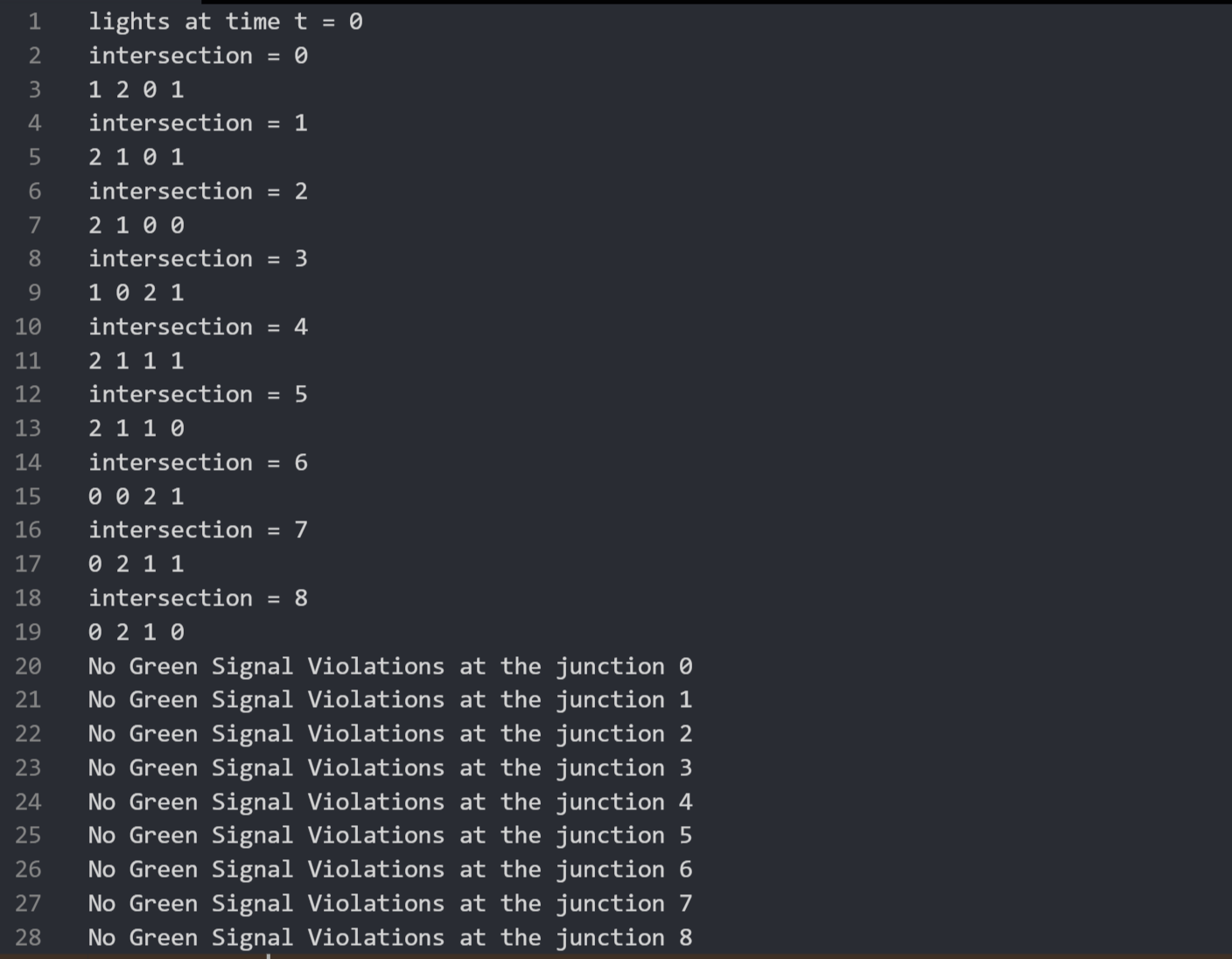
Green light violations occur when there are more than one green light at any intersection.

For the same above random traffic, green light violations are checked by the below function.



In every junction, number of green lights are counted by iterating over the loop. If this number is greater than 1, then it can be said that green light violations have occurred.

The green light violations are zero in this case, as each green light is turned to red before switching any light to green.



The result for green light violations for entire duration can be found at

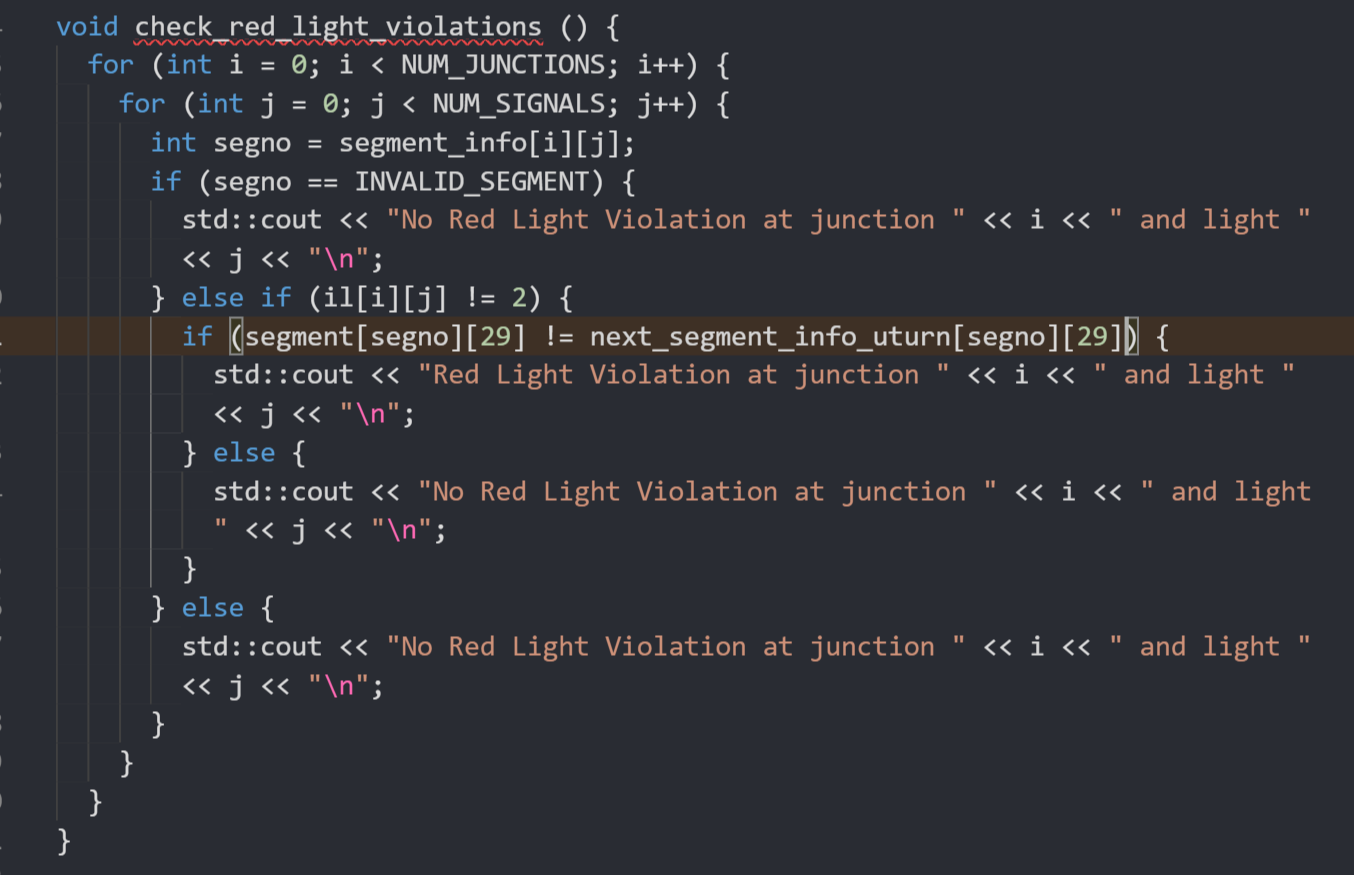
<https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/traffic1_output.txt>

and

<https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/traffic3_output.txt>

Testcase 4: Red Light Violations

Red light violations occur when vehicles cross the junction even when the corresponding traffic light is red.



If il[][] is not green, then we check for 29th position of segment at current time and 29th position of segment at next time instance. If both are not equal, then red light violations have occurred.

For this, different vehicle is assumed at time t = 0, 2 and 4 and then it is kept unchanged throughout.

The assumed traffic for these time instances are as below, where traffic5 is for t = 0, traffic6 is for t = 2 and traffic7 is for t = 4:

int traffic5[]={0,0,0,0,0,0,138,0,0,0,0,0,135,0,0,0,0,0,0,0,131,0,130,0,129,0,0,0,127,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,27,0,0,0,25,0,0,0,0,0,0,0,

126,0,125,0,124,0,123,0,122,0,121,60,0,89,59,0,58,0,117,0,56,0,115,0,54,53,0,112,52,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

111,0,110,109,49,0,78,0,77,0,106,0,75,0,0,0,0,102,42,0,101,69,39,0,99,98,38,0,0,96,

0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,57,0,0,0,0,0,0,0,0,0,0,0,

0,5,0,34,0,0,63,0,92,0,0,0,0,0,0,0,0,0,0,0,26,0,0,0,0,0,0,0,0,0,

21,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,70,0,9,0,8,0,0,0,

0,0,20,0,0,0,48,16,0,0,45,44,14,0,74,0,13,0,72,0,0,0,10,0,0,0,0,37,7,0,

36,0,0,0,0,0,33,0,32,0,0,0,30,0,0,0,0,0,0,0,0,0,0,0,24,0,23,0,0,0,

0,0,35,0,0,0,2,0,62,61,31,0,0,0,0,0,28,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,108,0,107,0,0,0,105,0,0,0,0,0,0,0,41,0,0,0,0,0,0,0,97,0,

51,0,50,0,0,0,0,0,47,0,46,0,0,0,0,0,43,0,0,0,0,0,40,0,0,0,0,0,0,0,

0,65,0,0,64,0,0,0,0,0,0,0,0,0,119,0,0,0,0,0,116,0,55,0,114,0,113,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

141,0,140,0,139,0,0,0,137,0,136,0,0,0,134,0,133,0,132,0,0,0,0,0,0,0,128,0,0,0,

0,0,0,0,0,0,0,0,0,0,1,0,120,0,29,0,118,0,0,0,0,0,0,0,0,0,0,0,22,0,

0,0,0,0,0,0,0,0,0,76,0,0,0,0,0,73,0,0,0,0,71,0,0,0,0,0,68,0,0,0,

0,0,0,0,19,0,18,0,0,0,0,0,15,0,0,0,0,0,12,0,0,0,0,0,0,0,0,0,0,0,

6,0,0,4,0,93,0,0,0,0,91,0,0,0,0,0,88,0,0,86,0,0,85,0,84,0,83,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,80,0,79,0,0,0,0,0,0,0,0,0,104,0,103,0,0,0,0,0,100,0,0,0,0,0,67,66,

81,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,95,0,94,0,0,0,0,0,0,90,0,0,0,0,0,0,87,0,0,0,0,0,0,0,0,0,82,0};

int traffic6[]={0,0,0,0,0,0,0,138,0,0,0,0,0,135,0,0,0,0,0,0,0,131,0,130,0,129,0,0,0,127,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,27,0,0,0,25,0,0,0,0,0,0,

0,126,0,125,0,124,0,123,0,122,0,121,60,0,89,59,0,58,0,117,0,56,0,115,0,54,53,0,112,52,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,111,0,110,109,49,0,78,0,77,0,106,0,75,0,0,0,0,102,42,0,101,69,39,0,99,98,38,0,0,

0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,57,0,0,0,0,0,0,0,0,0,0,

0,0,5,0,34,0,0,63,0,92,0,0,0,0,0,0,0,0,0,0,0,26,0,0,0,0,0,0,0,0,

0,21,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,70,0,9,0,8,0,0,

0,0,0,20,0,0,0,48,16,0,0,45,44,14,0,74,0,13,0,72,0,0,0,10,0,0,0,0,37,7,

0,36,0,0,0,0,0,33,0,32,0,0,0,30,0,0,0,0,0,0,0,0,0,0,0,24,0,23,0,0,

0,0,0,35,0,0,0,2,0,62,61,31,0,0,0,0,0,28,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,108,0,107,0,0,0,105,0,0,0,0,0,0,0,41,0,0,0,0,0,0,0,97,

0,51,0,50,0,0,0,0,0,47,0,46,0,0,0,0,0,43,0,0,0,0,0,40,0,0,0,0,0,0,

0,0,65,0,0,64,0,0,0,0,0,0,0,0,0,119,0,0,0,0,0,116,0,55,0,114,0,113,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,141,0,140,0,139,0,0,0,137,0,136,0,0,0,134,0,133,0,132,0,0,0,0,0,0,0,128,0,0,

0,0,0,0,0,0,0,0,0,0,0,1,0,120,0,29,0,118,0,0,0,0,0,0,0,0,0,0,0,22,

0,0,0,0,0,0,0,0,0,0,76,0,0,0,0,0,73,0,0,0,0,71,0,0,0,0,0,68,0,0,

0,0,0,0,0,19,0,18,0,0,0,0,0,15,0,0,0,0,0,12,0,0,0,0,0,0,0,0,0,0,

96,6,0,0,4,0,93,0,0,0,0,91,0,0,0,0,0,88,0,0,86,0,0,85,0,84,0,83,0,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,80,0,79,0,0,0,0,0,0,0,0,0,104,0,103,0,0,0,0,0,100,0,0,0,0,67,66,

0,81,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,95,0,94,0,0,0,0,0,0,90,0,0,0,0,0,0,87,0,0,0,0,0,0,0,0,0,82};

int traffic7[]={142,0,0,0,0,0,0,0,138,0,0,0,0,0,135,0,0,0,0,0,0,0,131,0,130,0,129,0,0,127,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,27,0,0,0,25,0,0,0,0,0,

66,0,126,0,125,0,124,0,123,0,122,0,121,60,0,89,59,0,58,0,117,0,56,0,115,0,54,53,0,112,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

52,0,111,0,110,109,49,0,78,0,77,0,106,0,75,0,0,0,0,102,42,0,101,69,39,0,99,98,38,0,

0,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,57,0,0,0,0,0,0,0,0,0,

0,0,0,5,0,34,0,0,63,0,92,0,0,0,0,0,0,0,0,0,0,0,26,0,0,0,0,0,0,0,

0,0,21,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,70,0,9,0,8,0,

0,0,0,0,20,0,0,0,48,16,0,0,45,44,14,0,74,0,13,0,72,0,0,0,10,0,0,0,0,37,

0,0,36,0,0,0,0,0,33,0,32,0,0,0,30,0,0,0,0,0,0,0,0,0,0,0,24,0,23,0,

7,0,0,0,35,0,0,0,2,0,62,61,31,0,0,0,0,0,28,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,108,0,107,0,0,0,105,0,0,0,0,0,0,0,41,0,0,0,0,0,0,0,

0,0,51,0,50,0,0,0,0,0,47,0,46,0,0,0,0,0,43,0,0,0,0,0,40,0,0,0,0,0,

0,0,0,65,0,0,64,0,0,0,0,0,0,0,0,0,119,0,0,0,0,0,116,0,55,0,114,0,113,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

0,0,141,0,140,0,139,0,0,0,137,0,136,0,0,0,134,0,133,0,132,0,0,0,0,0,0,0,128,0,

0,0,0,0,0,0,0,0,0,0,0,0,1,0,120,0,29,0,118,0,0,0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,0,0,76,0,0,0,0,0,73,0,0,0,0,71,0,0,0,0,0,68,0,

22,0,0,0,0,0,19,0,18,0,0,0,0,0,15,0,0,0,0,0,12,0,0,0,0,0,0,0,0,0,

0,96,6,0,0,4,0,93,0,0,0,0,91,0,0,0,0,0,88,0,0,86,0,0,85,0,84,0,83,0,

0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

82,0,0,0,80,0,79,0,0,0,0,0,0,0,0,0,104,0,103,0,0,0,0,0,100,0,0,0,0,67,

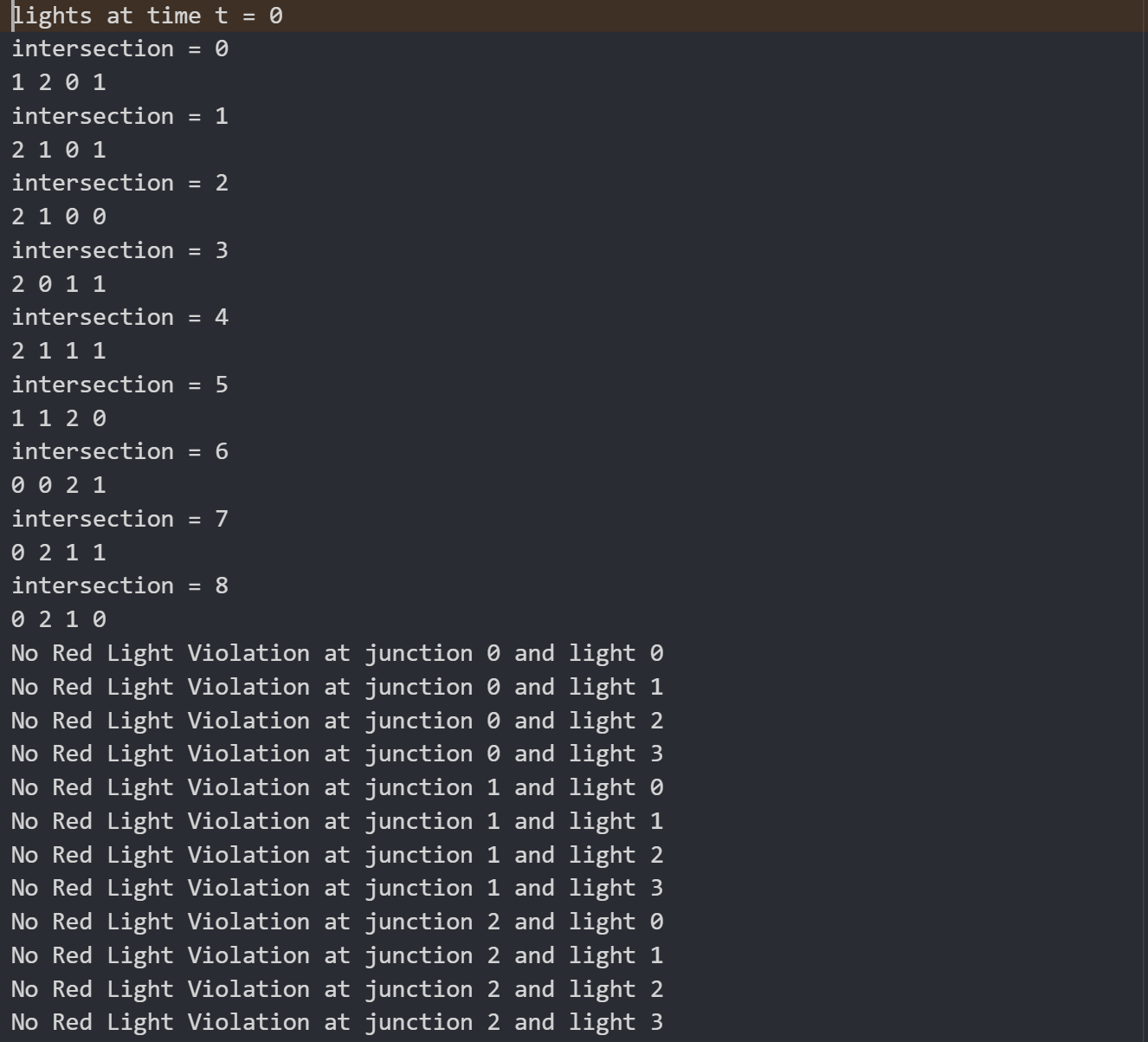
0,0,81,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,

97,0,0,0,95,0,94,0,0,0,0,0,0,90,0,0,0,0,0,0,87,0,0,0,0,0,0,0,0,0};

The full result for the testcase is found at

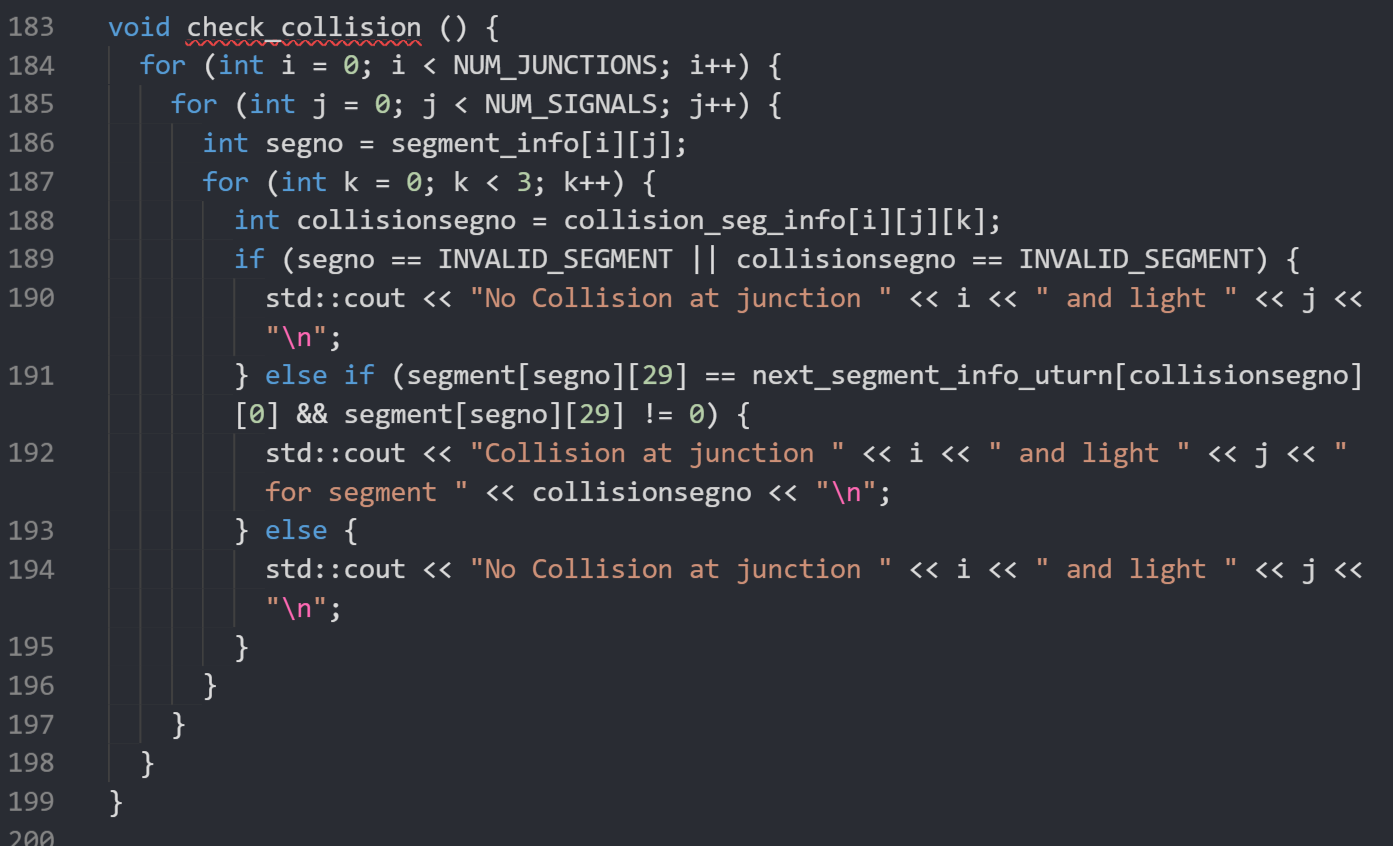
<https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/redlight_violation567_output%20(1).txt>

Here, it can be seen that no red light violations have occurred from assumed traffic.



Testcase 5: Collisions

Collisions occur when two vehicles try to move to the same position in a segment.



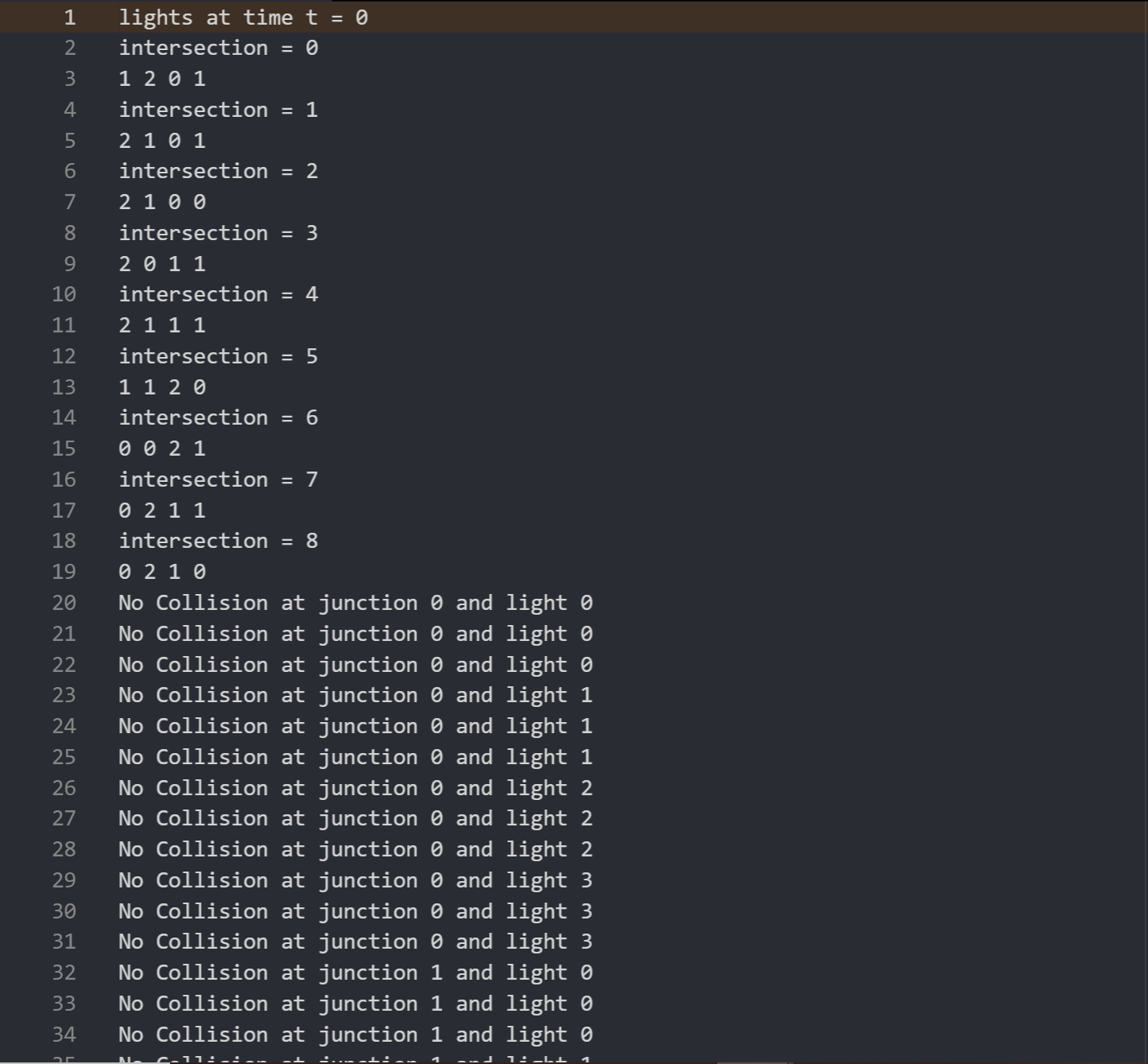
Each vehicle can move to either of the three segments (left, right or straight). So if the vehicle moves to either of them when that position is not free, then collision happens. So the vehicle information at current time is compared with each of the three segments in next time instant.

For this testcase also, same traffic as case 4 is assumed.

Full result can be obtained in

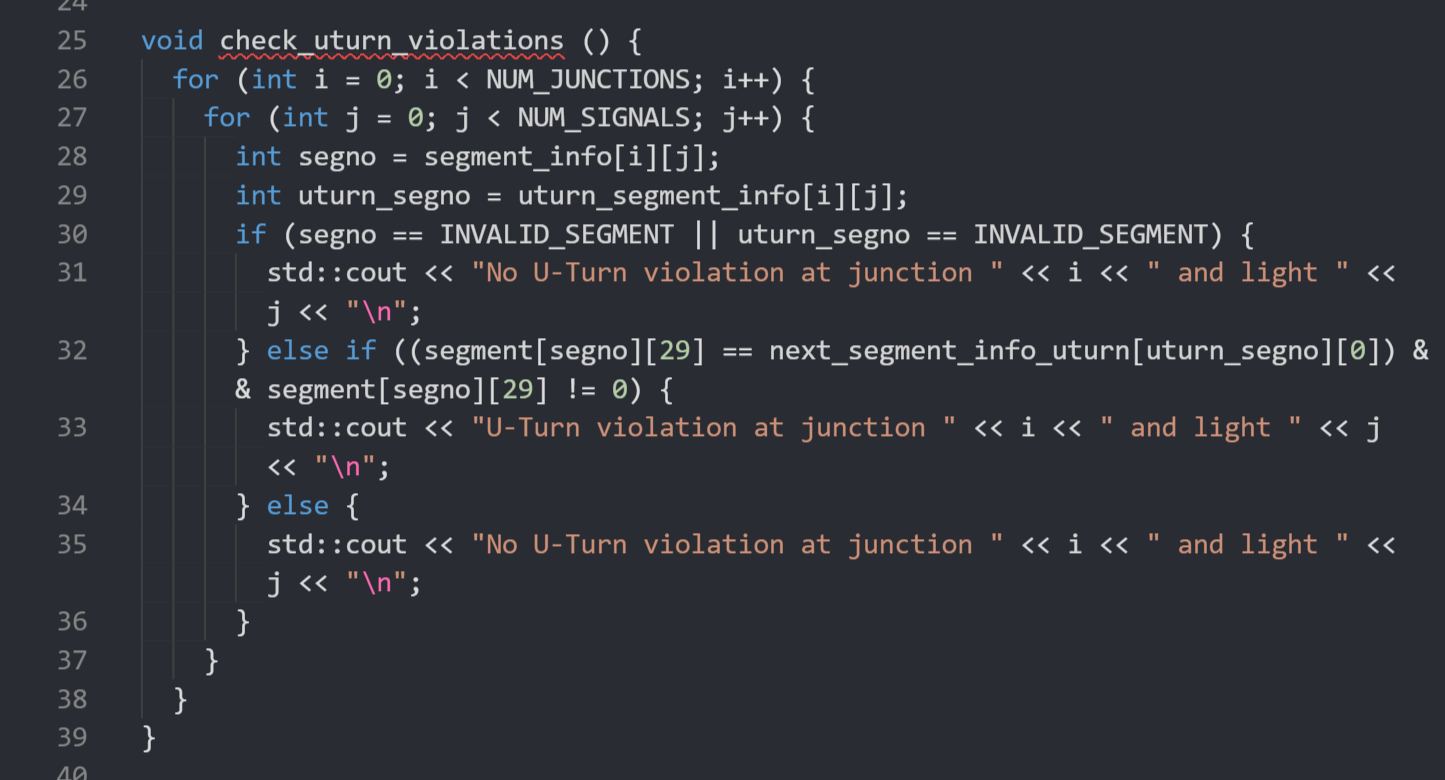
<https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/check_collision567_output.txt>

No vehicle collision is seen for the traffic assumed.



Testcase 6: U-Turn Violations

U-turn violations occur when the car takes u-turn at any signal (which is not allowed).



For u-turn violations, position 0 of the adjacent segment in next time instant is compared with position 29 of the segment at current time.

Same traffic is assumed as Testcase 4 and Testcase 5.

Link to the full result is found at

<https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/uturn_violation567_output.txt>

It is seen no u-turn violations are found for the assumed traffic.

