<u>ECEN/CSCE 689 PROJECT</u>
<u>PHASE A</u>

<u>MODEL CHECKING OF TRAFFIC CONTROL SYSTEM WITH VEHICLE SIMULATION</u>

TEAM – 10
I-GROUP

SUBMITTED BY
PUNARVI PALLAMREDDY (530005067)
DIVYA SHRIKANT HEGDE (831000995)

PROBLEM STATEMENT

To design a simplified traffic system and verify its properties.

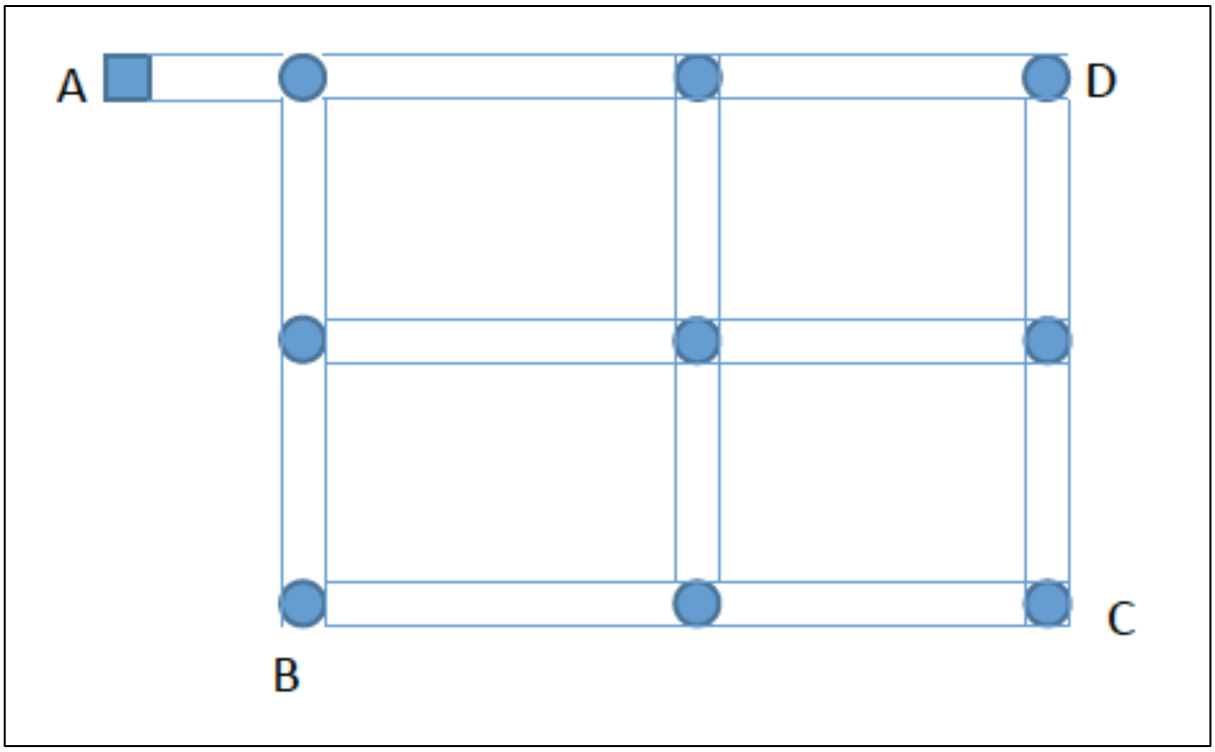


Figure 1 – traffic system

The road system has three horizontal roads and three vertical roads. Each circle is a road intersection. Each vehicle departs from A, visits B, C and D in any order and returns to A. The distance between two neighbouring circles is 0.5 mile and the distance from a square to its adjacent circle is 1/30 mile. A car can either run with speed of 30 miles per hour or stop.
Each road segment between two intersections is divided into 30 uniform-sized slots. Each control step (or time step) is 2 seconds. At each control step, a car must be within a slot. In the next step, it can either stay there or move to the next slot. Along each direction, there is a single lane.
Between two consecutive time steps, there is at most 1 car crossing an intersection. If at any moment, there are more than one car in the same slot, that is counted as a collision. A car must stop at red light and cannot make right turn. The problem formulation is to maximize the total number of cars reaching each of the 4 square node destinations per hour, without any collision, without any red-light signal violation. There are green and red lights, but no yellow lights.

ARCHITECTURE

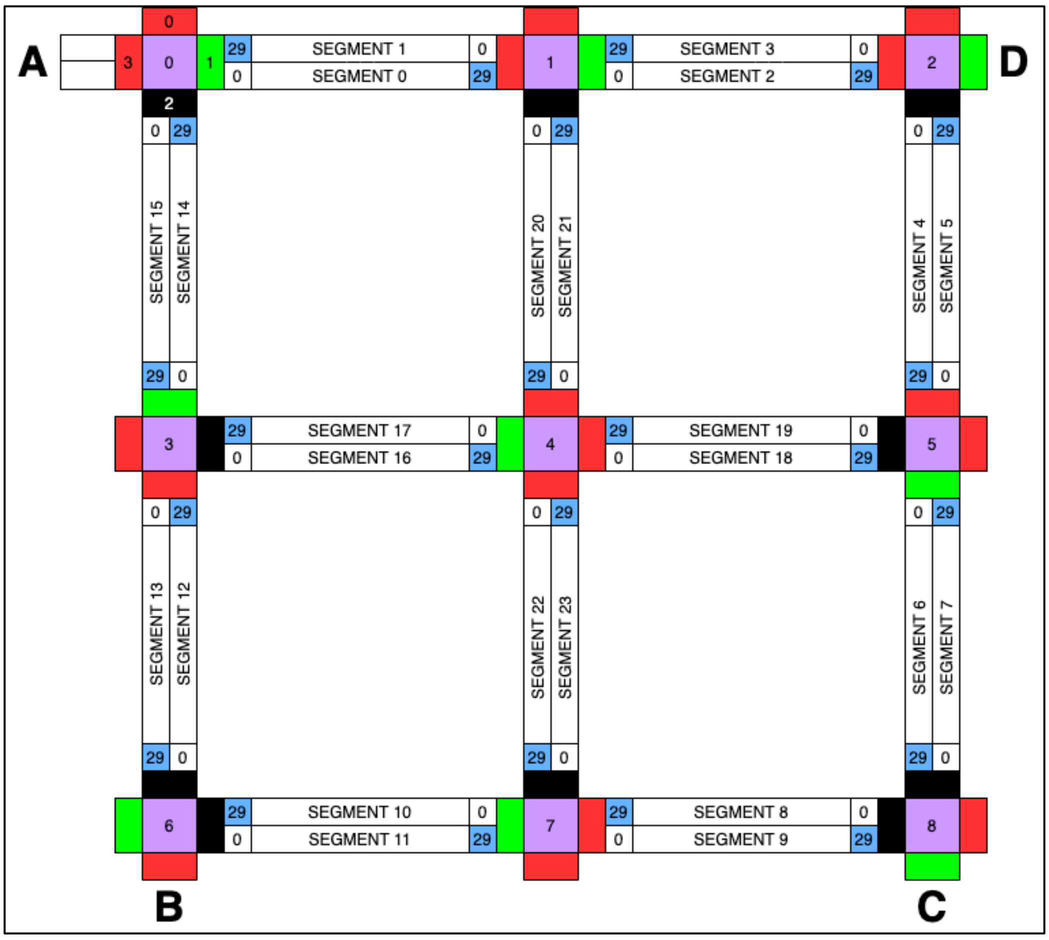


Figure 2 – System Architecture

The road system described in the problem statement is visualized and implemented as shown in figure2. The system has four nodes – A, B, C, D and 9 intersections. The intersections are denoted by purple boxes. At each intersection, there are 4 traffic signal lights. Each traffic light switches between red light and green light only. These lights are denoted in the figure by the red/green/black boxes around the intersection. Black box represents the light that is always switched off or in other words that light is never used since there is no road that follows that light.

The starting point for all vehicles is node A. Vehicle will start from A, visit nodes B, C and D in any order and return to A.

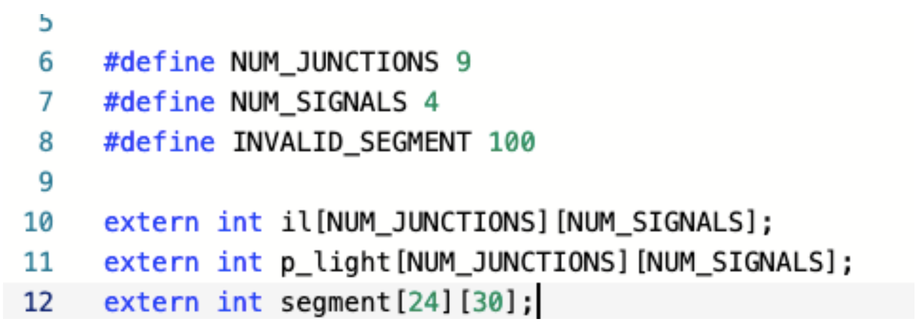
Each road between 2 intersections has been divided into two segments, each for one moving direction. A vehicle moving from intersection 0 to 1 will be traveling in segment 0. Similarly, vehicle moving from intersection 1 to 0 will be traveling in segment 1. Each segment is divided into 30 slots, starting from slot 0 and ending at slot 29. When a vehicle reaches slot 29, it will check for the signal light, whether it is red or green to decide if it should stop at the intersection or move. A vehicle in segment 14 will follow signal light that will be shown in traffic light box 0 at intersection 0. Similarly, cars generated at A will follow the light shown in traffic light box 1. For traffic light box 2, there is no road opposite to it. So, there will be no vehicle following that signal. Hence, it is colored black to denote that light is permanently switched off. Vehicles traveling in segment 1 will follow the signal in traffic light box 3.

The switching of traffic lights is based on the vehicle congestion present at the corresponding four segments at each intersection. Signal will switch to green for the segment which has highest number of vehicles in waiting. For every two seconds, the lights will be switching. At any time, there can only be at the most one green light. U-turns are not allowed. A car must stop at red light and cannot make right turn. Traffic light system (or i-group) does not know where the car will move in next time step. It only receives the information regarding vehicle location and moving directions from v-group.

The constraints of the system are –

- no collision
- no violation of red light signal
- no run of opposite direction lane
- only 1 green light at an intersection at a given time
- no u-turns

Based on the above architectural features, we have designed and implemented a traffic light control system in C++. The design of the system uses the following mapping –



```
#define NUM_JUNCTIONS 9
#define NUM_SIGNALS 4
#define INVALID_SEGMENT 100

extern int il[NUM_JUNCTIONS][NUM_SIGNALS];
extern int p_light[NUM_JUNCTIONS][NUM_SIGNALS];
extern int segment[24][30];
```

1. #define NUM_JUNCTIONS 9
   NUM_JUNCTIONS represents the total number of intersections present in the system. There are 9 intersections.

2. #define NUM_SIGNALS 4
   NUM_SIGNALS represents number of traffic lights at each intersection. At each junction, we have 4 traffic lights.

3. #define INVALID_SEGMENT 100
   INVALID_SEGMENT represents the segment number for the lights that do not have any road/segment opposite to it. For example, light 2 at intersection 0, intersection 1 do not have any road or vehicle following the light. So the light 2 at intersection 0, 1 will be associated with segment numbered 100.

4. extern int il[NUM_JUNCTIONS][NUM_SIGNALS];
   il matrix stores the information of lights at each intersection for every time. It will store values of 0,1 and 2 to show if signal is off, red or green respectively. il[3][3]=2 implies that traffic light 3 at intersection 3 is green.

5. extern int p_light[NUM_JUNCTIONS][NUM_SIGNALS];

p_light is the initial priority given to traffic signals for switching. p_light will take the value from segment information provided by v-group, check if car is present at last slot in the corresponding segment and assign the car ID as priority to the light.
p_light[6][3]=segment[10][29] implies light 3 at intersection 6 will have a priority value assigned to if a car is present at slot 29 in segment 10.

6. extern int segment[24][30];
   segment[][] represents the vehicle location information taken from v-group. Since there are 24 segments and each segment has 30 slots, the segment matrix is sized to 24x30.
   segment[22][17]=9 implies a car with car ID 9 is present in slot 17 of segment 22.

ALGORITHM

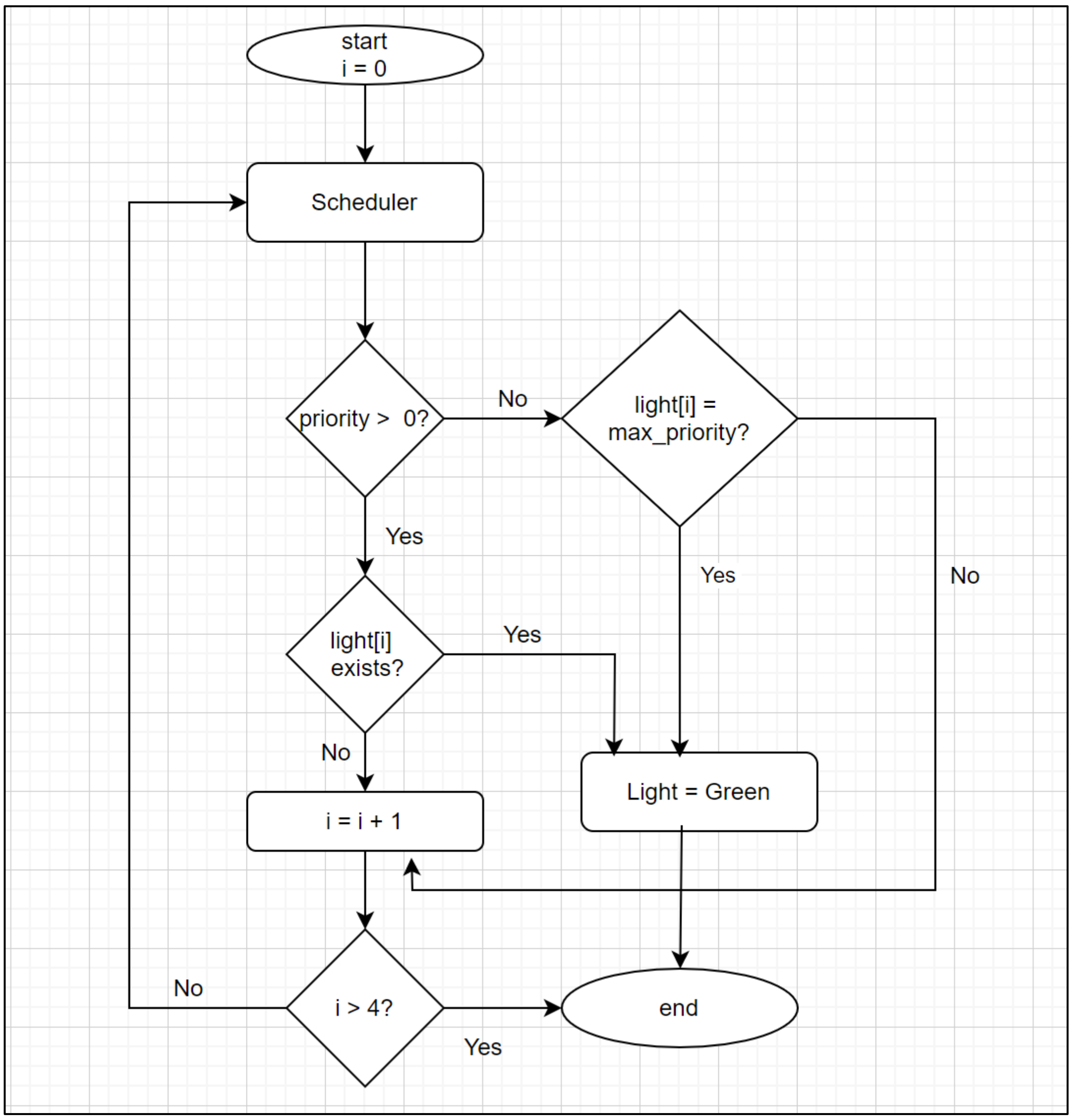


Figure 3 – Algorithm for single junction

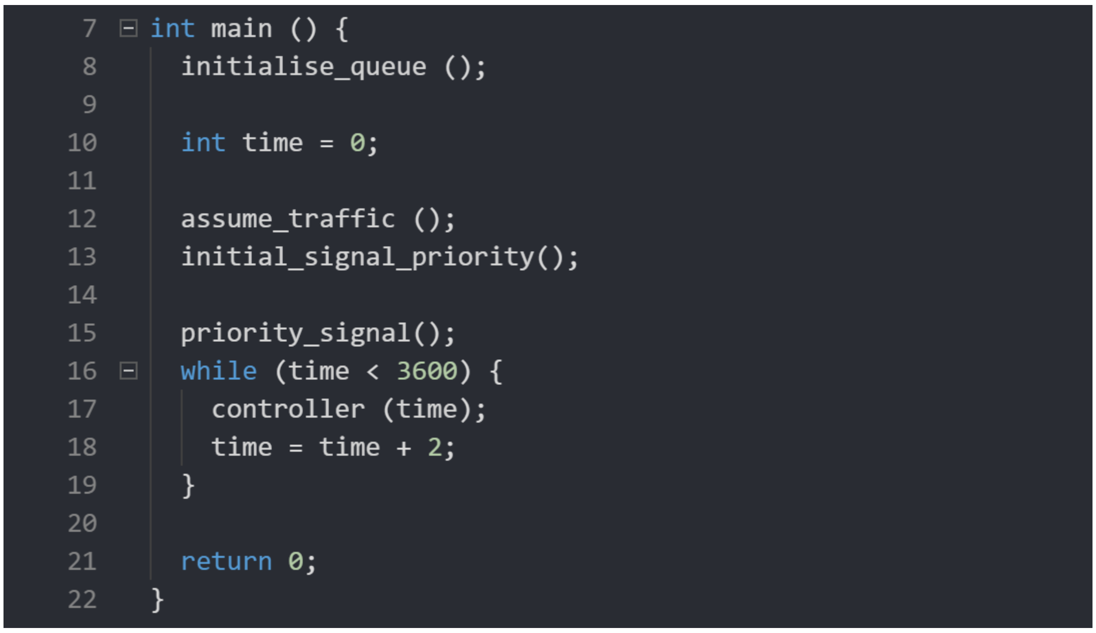
The above flowchart is for a single junction.

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.

MAIN() FUNCTION -

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.

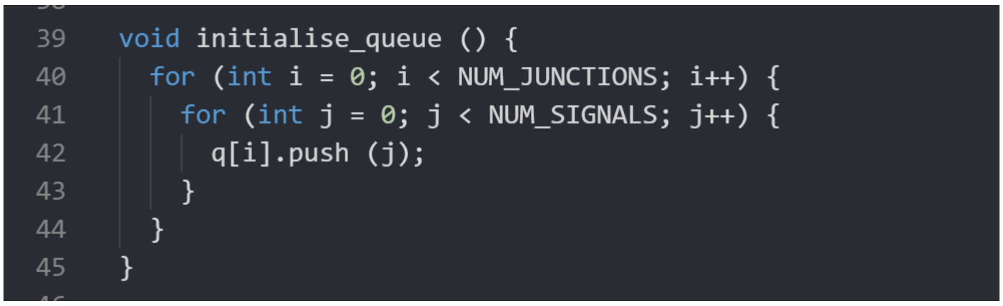


```
int main () {
  initialise_queue ();

  int time = 0;

  assume_traffic ();
  initial_signal_priority();

  priority_signal();
  while (time < 3600) {
    controller (time);
    time = time + 2;
  }

  return 0;
}
```

FIGURE 4 – MAIN FUNCTION

INITIALISE_QUEUE() FUNCTION -

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.

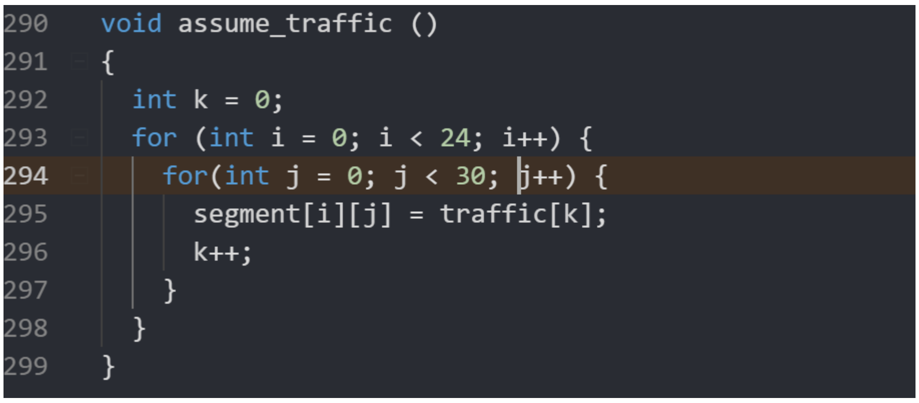


```
void initialise_queue () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      q[i].push (j);
    }
  }
}
```

FIGURE 5 – INITIALISE_QUEUE FUNCTION

ASSUME_TRAFFIC() FUNCTION-

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.

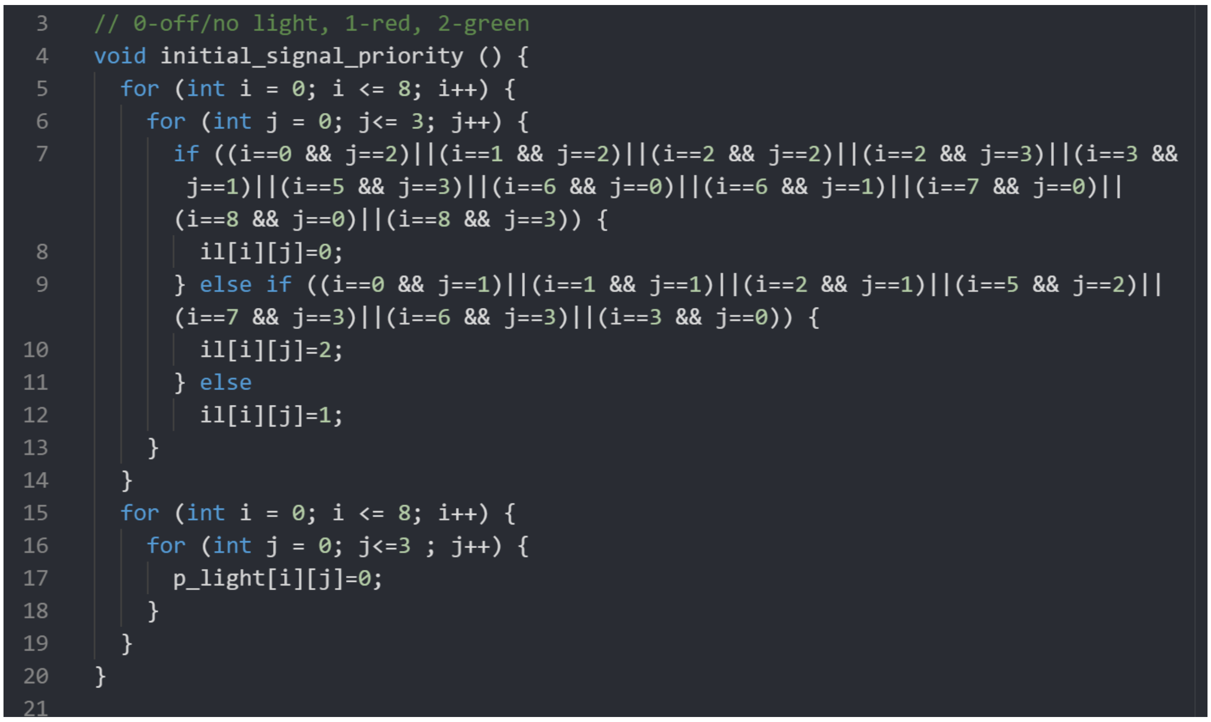


```
void assume_traffic ()
{
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30; j++) {
      segment[i][j] = traffic[k];
      k++;
    }
  }
}
```

FIGURE 6 – ASSUME_TRAFFIC() FUNCTION

INITIAL_SIGNAL_PRIORITY() FUNCTION -

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.

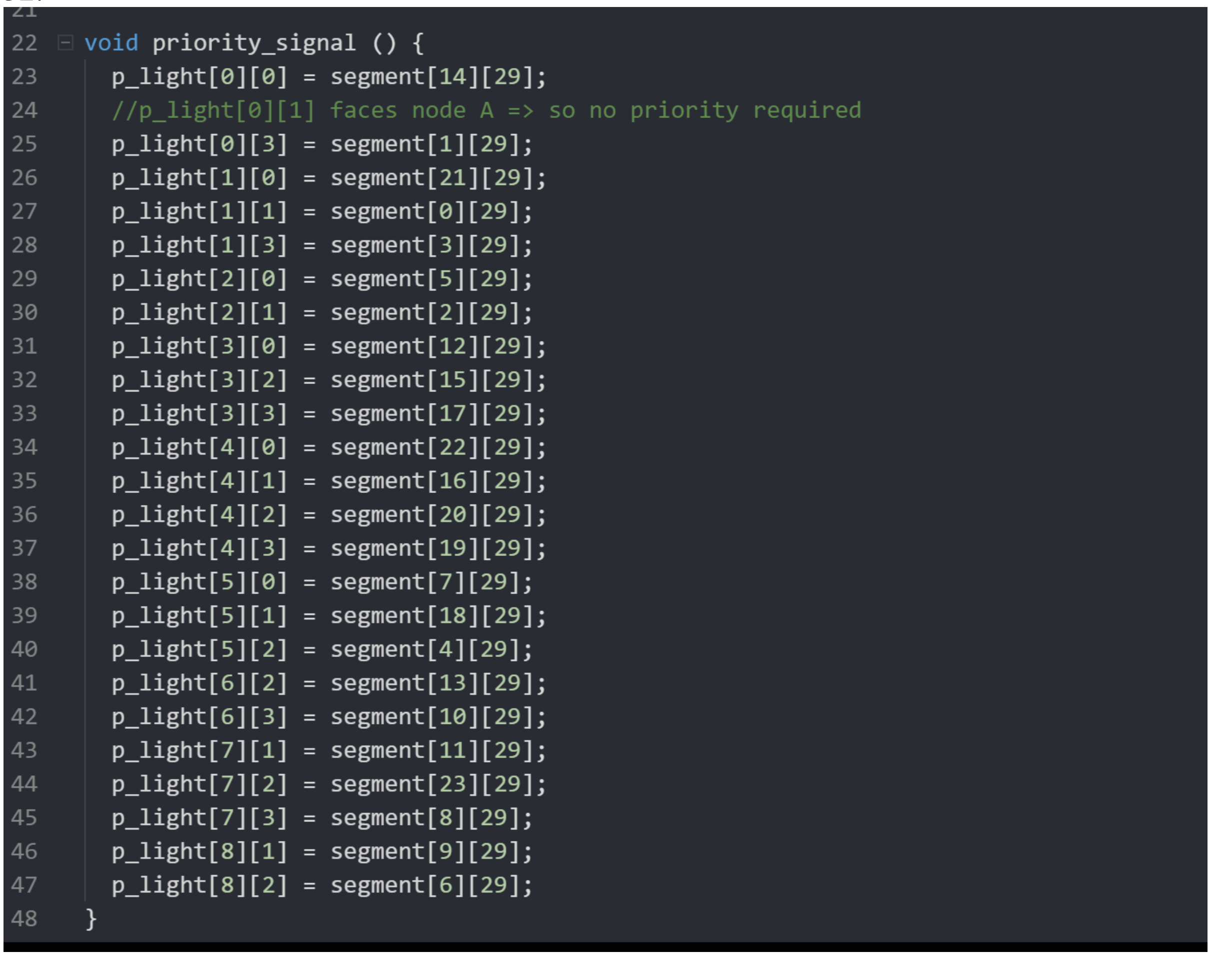


```
// 0-off/no light, 1-red, 2-green
void initial_signal_priority () {
  for (int i = 0; i <= 8; i++) {
    for (int j = 0; j<= 3; j++) {
      if ((i==0 && j==2)||(i==1 && j==2)||(i==2 && j==2)||(i==2 && j==3)||(i==3 &&
       j==1)||(i==5 && j==3)||(i==6 && j==0)||(i==6 && j==1)||(i==7 && j==0)||
      (i==8 && j==0)||(i==8 && j==3)) {
        il[i][j]=0;
      } else if ((i==0 && j==1)||(i==1 && j==1)||(i==2 && j==1)||(i==5 && j==2)||
      (i==7 && j==3)||(i==6 && j==3)||(i==3 && j==0)) {
        il[i][j]=2;
      } else
        il[i][j]=1;
    }
  }
  for (int i = 0; i <= 8; i++) {
    for (int j = 0; j<=3 ; j++) {
      p_light[i][j]=0;
    }
  }
}
```

FIGURE 7 – INITIAL_SIGNAL_PRIORITY FUNCTION

PRIORITY_SIGNAL() FUNCTION -

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.

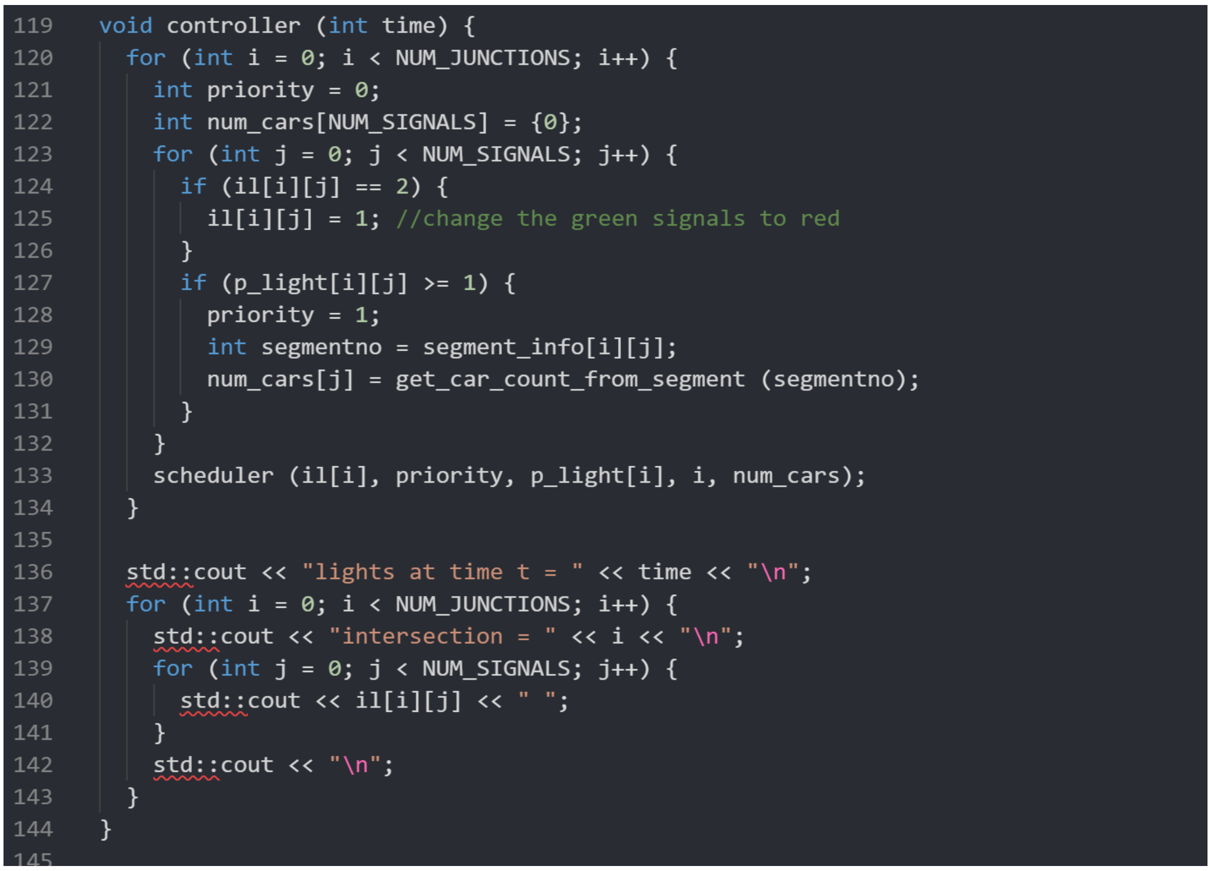


```
void priority_signal () {
    p_light[0][0] = segment[14][29];
    //p_light[0][1] faces node A => so no priority required
    p_light[0][3] = segment[1][29];
    p_light[1][0] = segment[21][29];
    p_light[1][1] = segment[0][29];
    p_light[1][3] = segment[3][29];
    p_light[2][0] = segment[5][29];
    p_light[2][1] = segment[2][29];
    p_light[3][0] = segment[12][29];
    p_light[3][2] = segment[15][29];
    p_light[3][3] = segment[17][29];
    p_light[4][0] = segment[22][29];
    p_light[4][1] = segment[16][29];
    p_light[4][2] = segment[20][29];
    p_light[4][3] = segment[19][29];
    p_light[5][0] = segment[7][29];
    p_light[5][1] = segment[18][29];
    p_light[5][2] = segment[4][29];
    p_light[6][2] = segment[13][29];
    p_light[6][3] = segment[10][29];
    p_light[7][1] = segment[11][29];
    p_light[7][2] = segment[23][29];
    p_light[7][3] = segment[8][29];
    p_light[8][1] = segment[9][29];
    p_light[8][2] = segment[6][29];
}
```

FIGURE 8 –SIGNAL_PRIORITY FUNCTION

CONTROLLER() FUNCTION -

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.



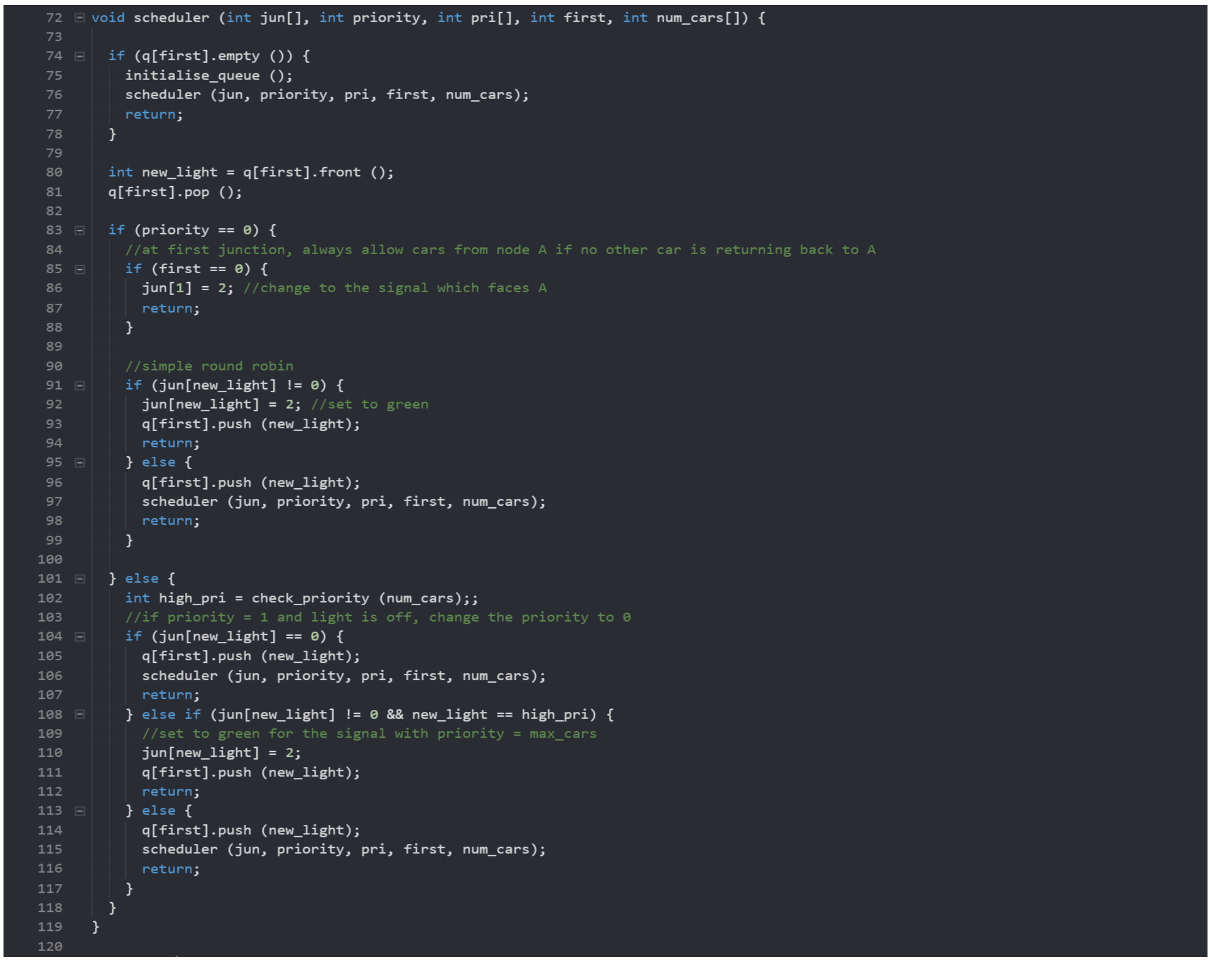
```
void controller (int time) {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    int priority = 0;
    int num_cars[NUM_SIGNALS] = {0};
    for (int j = 0; j < NUM_SIGNALS; j++) {
      if (il[i][j] == 2) {
        il[i][j] = 1; //change the green signals to red
      }
      if (p_light[i][j] >= 1) {
        priority = 1;
        int segmentno = segment_info[i][j];
        num_cars[j] = get_car_count_from_segment (segmentno);
      }
    }
    scheduler (il[i], priority, p_light[i], i, num_cars);
  }

  std::cout << "lights at time t = " << time << "\n";
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    std::cout << "intersection = " << i << "\n";
    for (int j = 0; j < NUM_SIGNALS; j++) {
      std::cout << il[i][j] << " ";
    }
    std::cout << "\n";
  }
}
```

FIGURE 9 – CONTROLLER() FUNCTION

SCHEDULER() FUNCTION -

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.

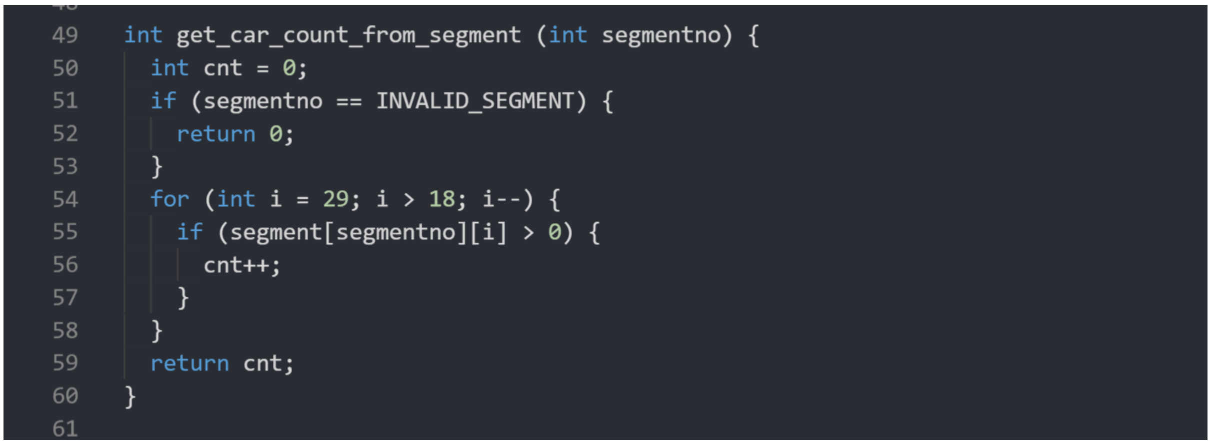


```
void scheduler (int jun[], int priority, int pri[], int first, int num_cars[]) {

  if (q[first].empty ()) {
    initialise_queue ();
    scheduler (jun, priority, pri, first, num_cars);
    return;
  }

  int new_light = q[first].front ();
  q[first].pop ();

  if (priority == 0) {
    //at first junction, always allow cars from node A if no other car is returning back to A
    if (first == 0) {
      jun[1] = 2; //change to the signal which faces A
      return;
    }

    //simple round robin
    if (jun[new_light] != 0) {
      jun[new_light] = 2; //set to green
      q[first].push (new_light);
      return;
    } else {
      q[first].push (new_light);
      scheduler (jun, priority, pri, first, num_cars);
      return;
    }

  } else {
    int high_pri = check_priority (num_cars);;
    //if priority = 1 and light is off, change the priority to 0
    if (jun[new_light] == 0) {
      q[first].push (new_light);
      scheduler (jun, priority, pri, first, num_cars);
      return;
    } else if (jun[new_light] != 0 && new_light == high_pri) {
      //set to green for the signal with priority = max_cars
      jun[new_light] = 2;
      q[first].push (new_light);
      return;
    } else {
      q[first].push (new_light);
      scheduler (jun, priority, pri, first, num_cars);
      return;
    }
  }
}
```

FIGURE 10 –SCHEDULER() FUNCTION

GET_CAR_COUNT() FUNCTION -

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.

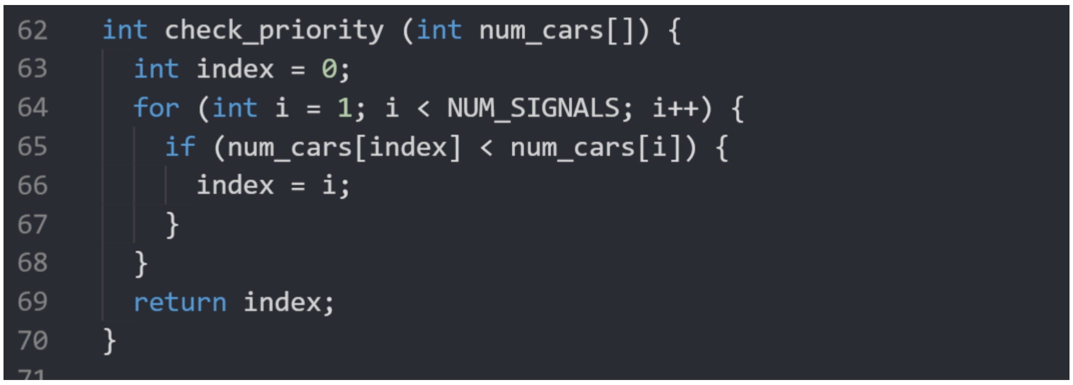


```c
int get_car_count_from_segment (int segmentno) {
  int cnt = 0;
  if (segmentno == INVALID_SEGMENT) {
    return 0;
  }
  for (int i = 29; i > 18; i--) {
    if (segment[segmentno][i] > 0) {
      cnt++;
    }
  }
  return cnt;
}
```

FIGURE 11 – GET_CAR_COUNT() FUNCTION

CHECK_PRIORITY() FUNCTION-

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.



```c
int check_priority (int num_cars[]) {
  int index = 0;
  for (int i = 1; i < NUM_SIGNALS; i++) {
    if (num_cars[index] < num_cars[i]) {
      index = i;
    }
  }
  return index;
}
```

FIGURE 12 – CHECK_PRIORITY() FUNCTION

SOFTWARE CODE AND RESULTS

SOURCE CODE –

Source code is present in Appendix A.

PLOTTING CODE –

Obtained result of traffic lights is plotted into a graph. Plotting of results is done using python. Code for plotting is present in Appendix B.

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.

Result for the entire duration can be found below -

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

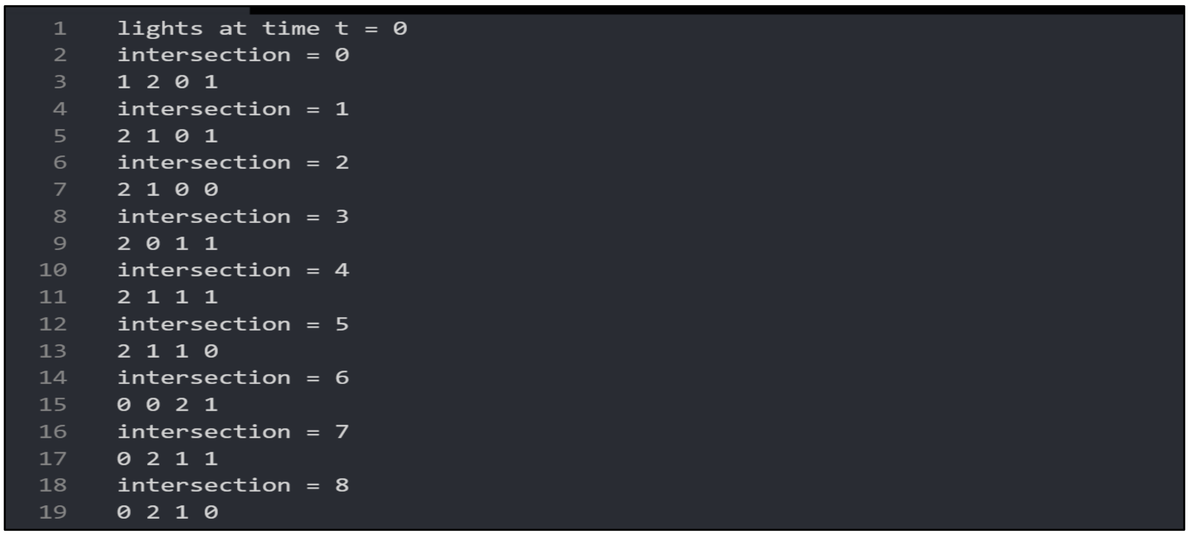
Switching of lights at every time instant is visible below -

https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/signal_gif.gif

All the output plots can be found below (plots with names notraffic*.png) -

https://github.com/rishyasankar/Formal_verification_project/tree/main/PHASE%20A%20SUBMISSION/i-group/output%20plots

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.



```
lights at time t = 0
intersection = 0
1 2 0 1
intersection = 1
2 1 0 1
intersection = 2
2 1 0 0
intersection = 3
2 0 1 1
intersection = 4
2 1 1 1
intersection = 5
2 1 1 0
intersection = 6
0 0 2 1
intersection = 7
0 2 1 1
intersection = 8
0 2 1 0
```

Figure 13 – Output result for test case 1 at t=0

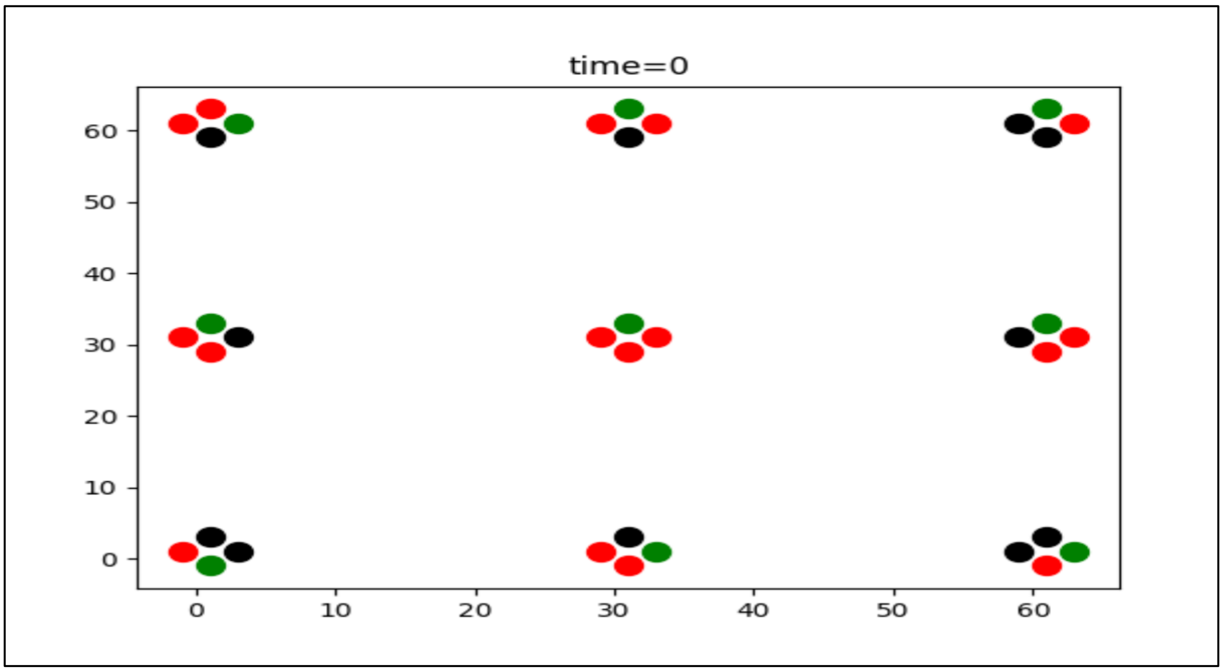
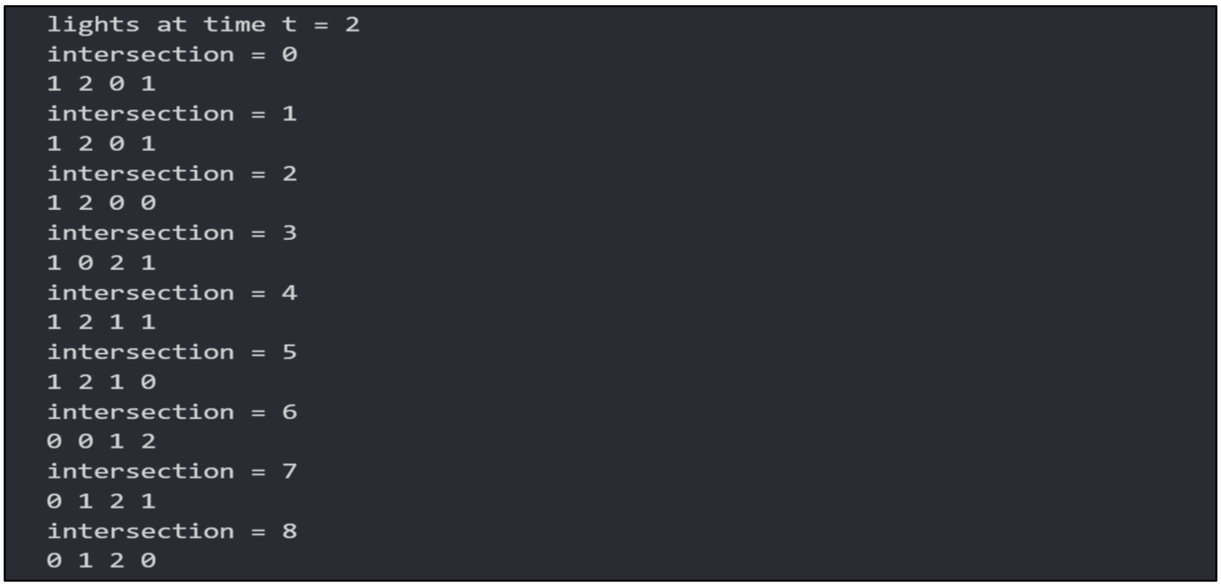


Figure 14 – Plot for test case 1 at t=0

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.



```
lights at time t = 2
intersection = 0
1 2 0 1
intersection = 1
1 2 0 1
intersection = 2
1 2 0 0
intersection = 3
1 0 2 1
intersection = 4
1 2 1 1
intersection = 5
1 2 1 0
intersection = 6
0 0 1 2
intersection = 7
0 1 2 1
intersection = 8
0 1 2 0
```

Figure 15 – Output result for test case 1 at t=2

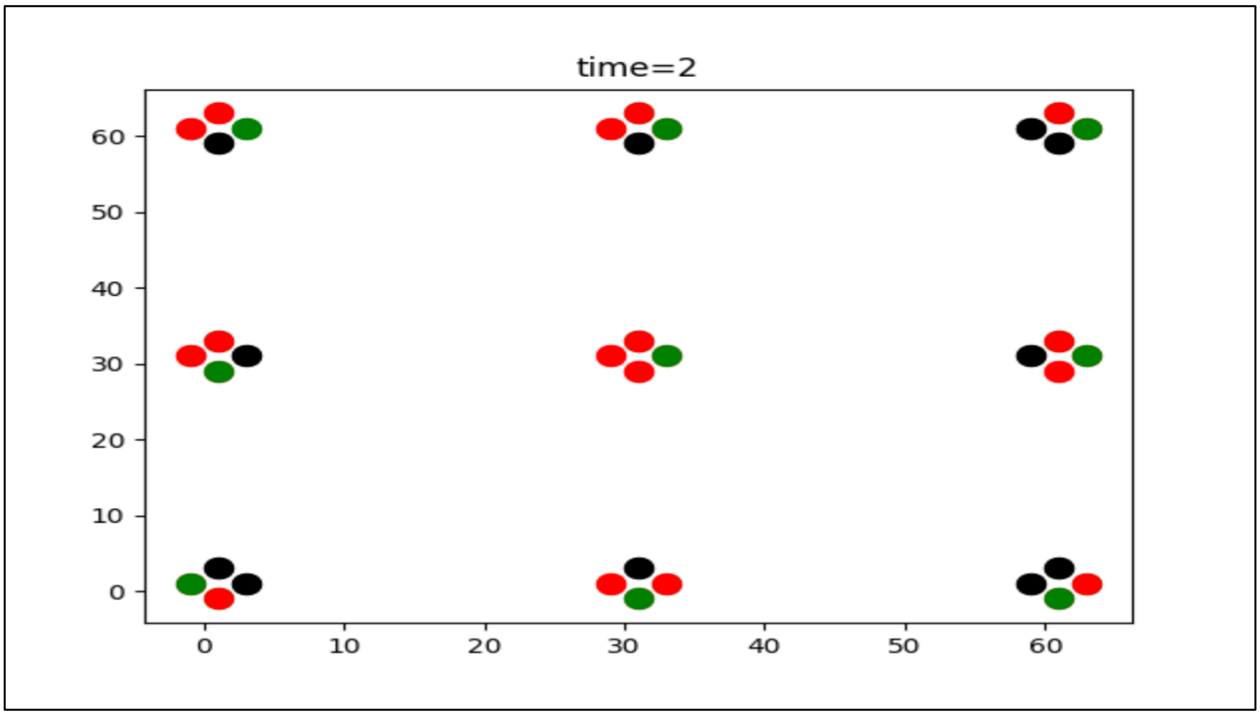
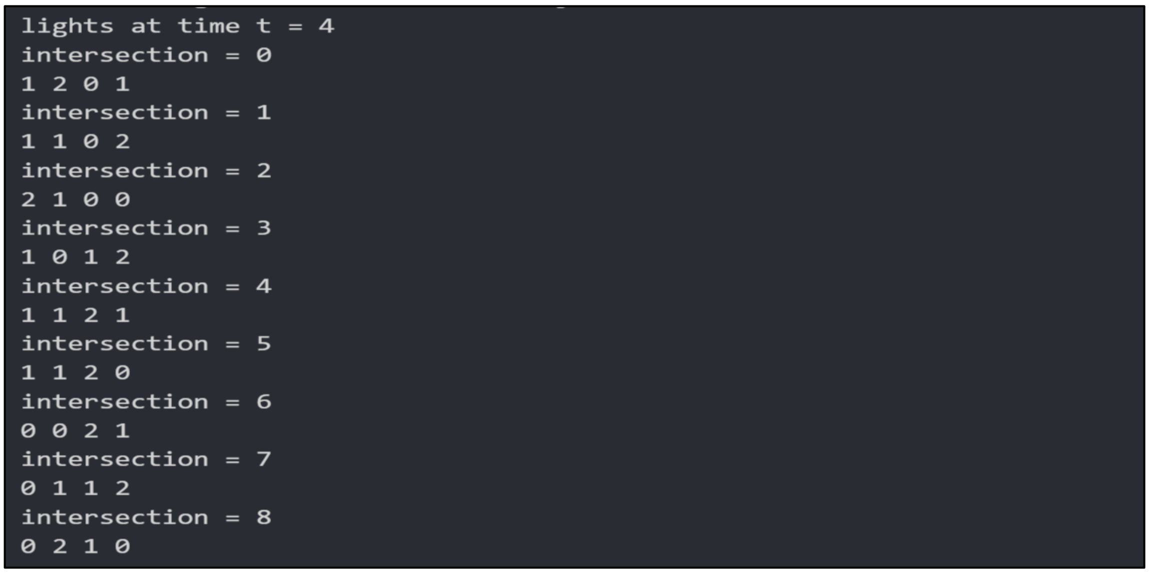


Figure 16 – Plot for test case 1 at t=2



```
lights at time t = 4
intersection = 0
1 2 0 1
intersection = 1
1 1 0 2
intersection = 2
2 1 0 0
intersection = 3
1 0 1 2
intersection = 4
1 1 2 1
intersection = 5
1 1 2 0
intersection = 6
0 0 2 1
intersection = 7
0 1 1 2
intersection = 8
0 2 1 0
```

Figure 17 – Output result for test case 1 at t=4

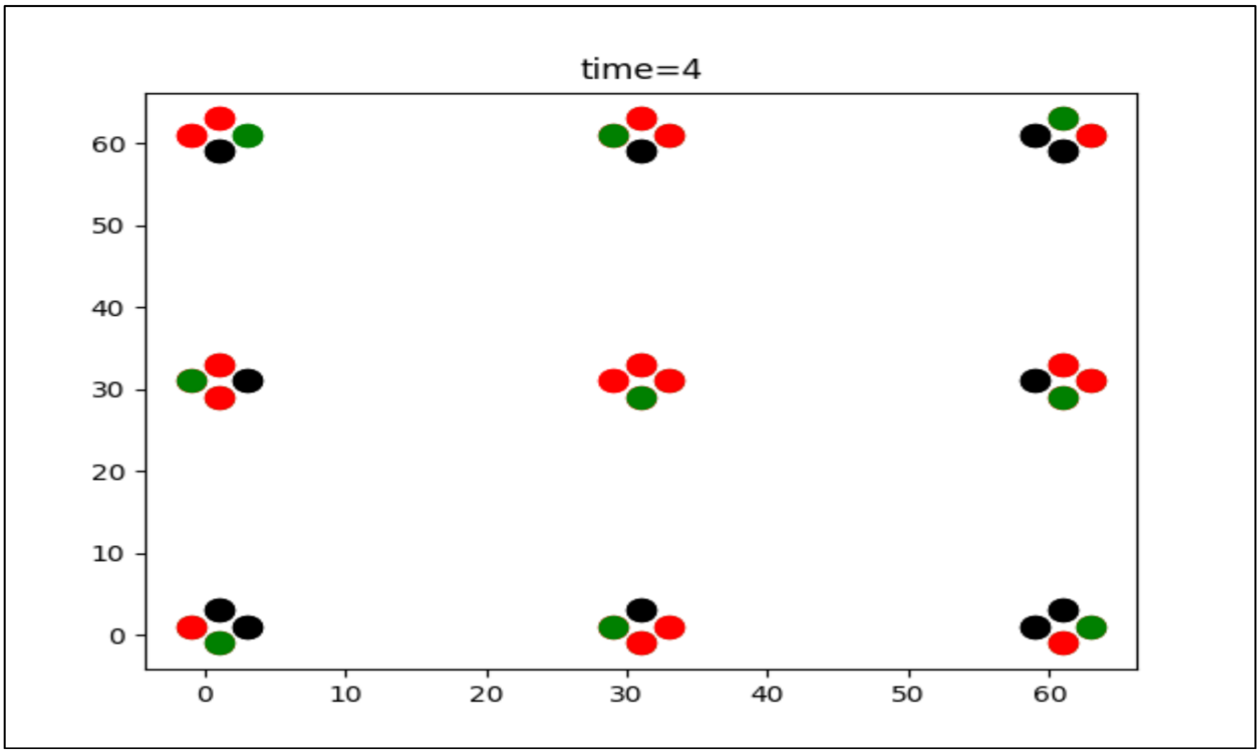


Figure 18 – Plot for test case 1 at t=4

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

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 below -

https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/signal_traffic_gif.gif

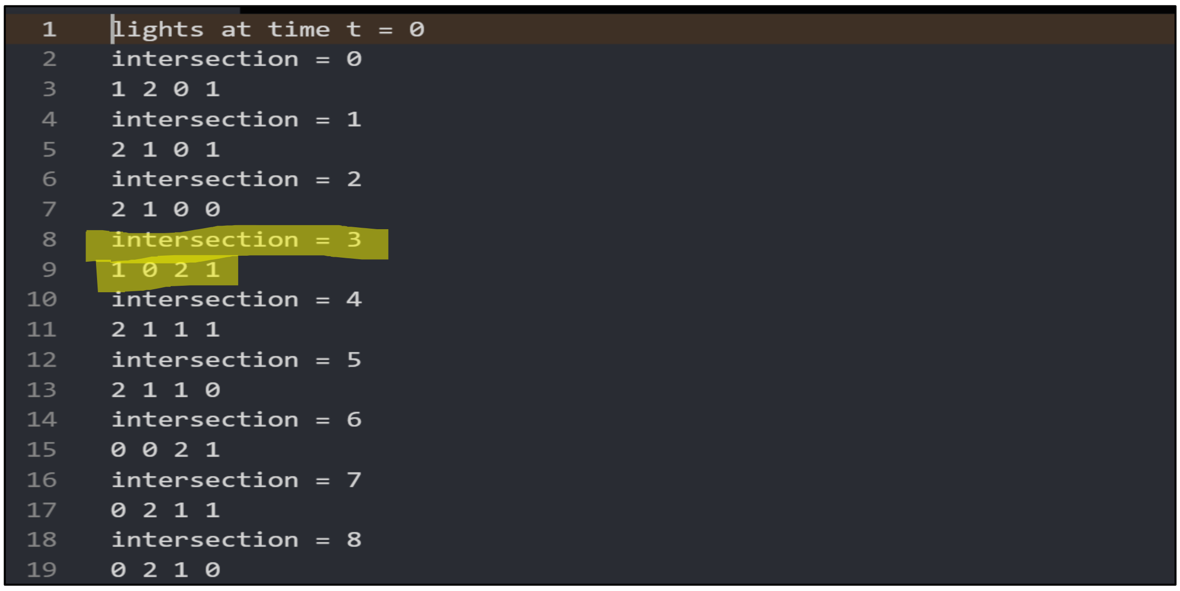
All the output plots can be found below - (plots with names traffic*.png)

https://github.com/rishyasankar/Formal_verification_project/tree/main/PHASE%20A%20SUBMISSION/i-group/output%20plots

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,
0,10,0,0,0,0,0,7,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,
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,
0,25,0,0,0,0,0,0,0,0,20,0,0,0,0,0,0,16,0,0,0,14,0,13,0,0,0,11,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,
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,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

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.



```
lights at time t = 0
intersection = 0
1 2 0 1
intersection = 1
2 1 0 1
intersection = 2
2 1 0 0
intersection = 3
1 0 2 1
intersection = 4
2 1 1 1
intersection = 5
2 1 1 0
intersection = 6
0 0 2 1
intersection = 7
0 2 1 1
intersection = 8
0 2 1 0
```

Figure 18 – Output result for test case 2 at t=0

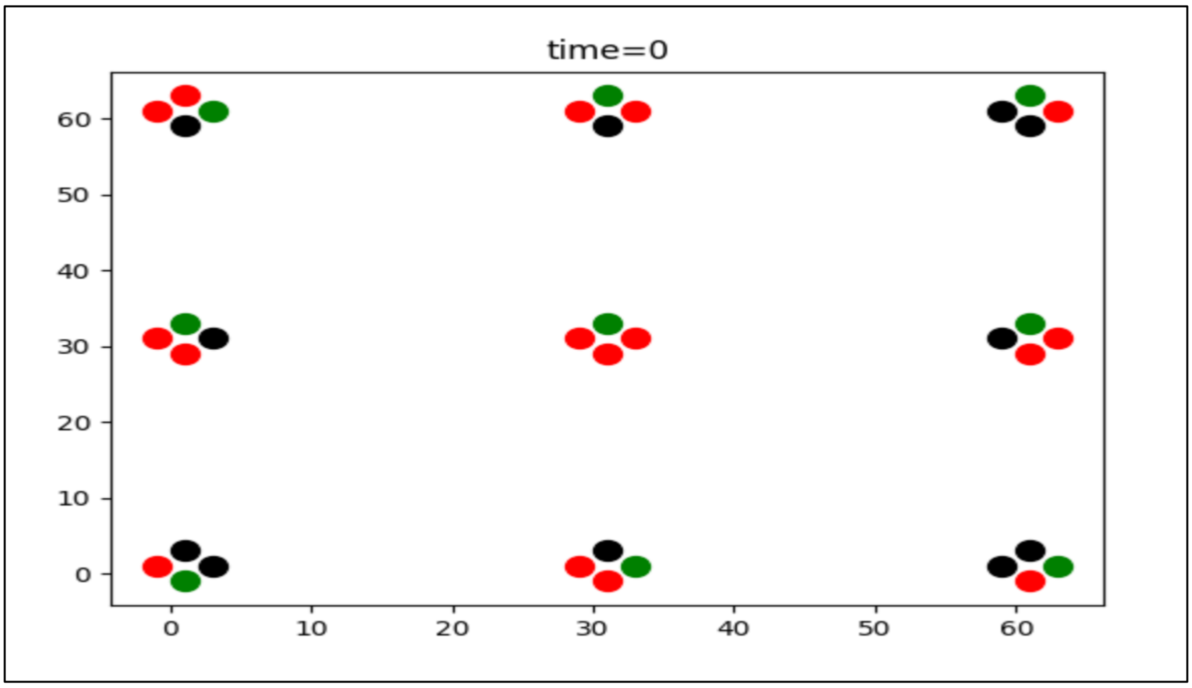
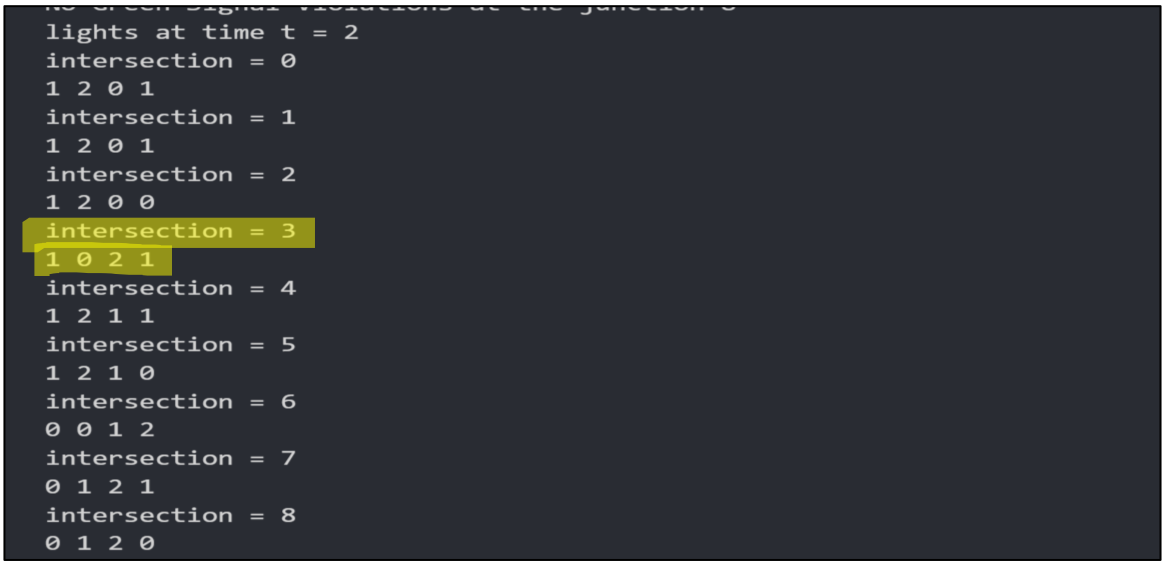


Figure 19 – Plot for test case 2 at t=0



```
lights at time t = 2
intersection = 0
1 2 0 1
intersection = 1
1 2 0 1
intersection = 2
1 2 0 0
intersection = 3
1 0 2 1
intersection = 4
1 2 1 1
intersection = 5
1 2 1 0
intersection = 6
0 0 1 2
intersection = 7
0 1 2 1
intersection = 8
0 1 2 0
```

Figure 20 – Output result for test case 2 at t=2

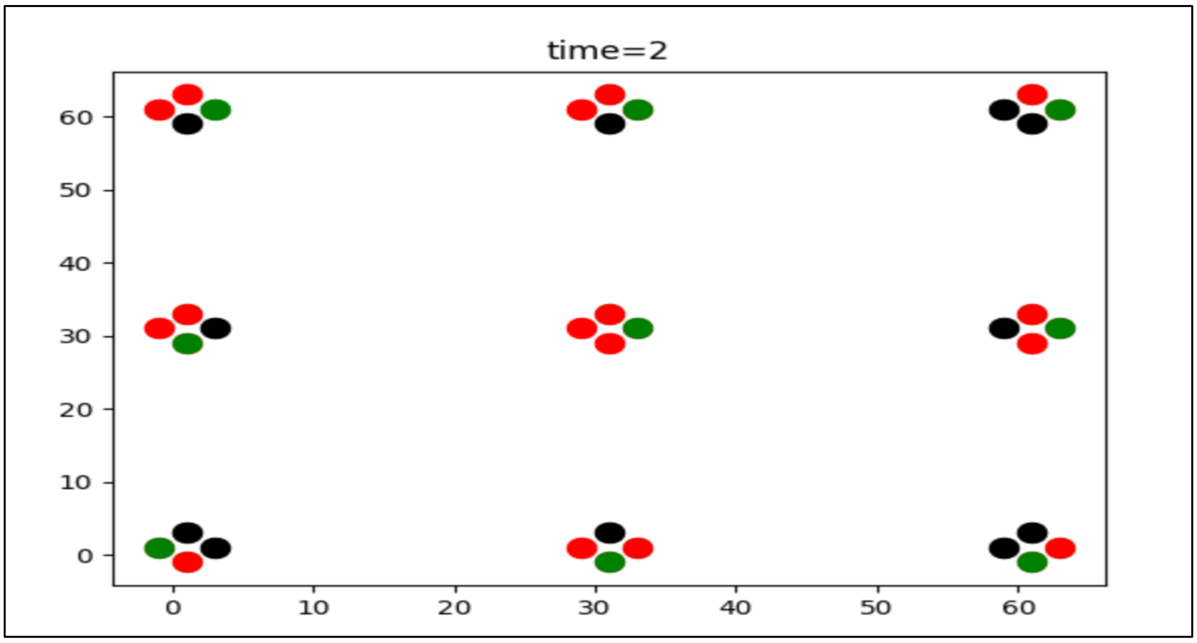
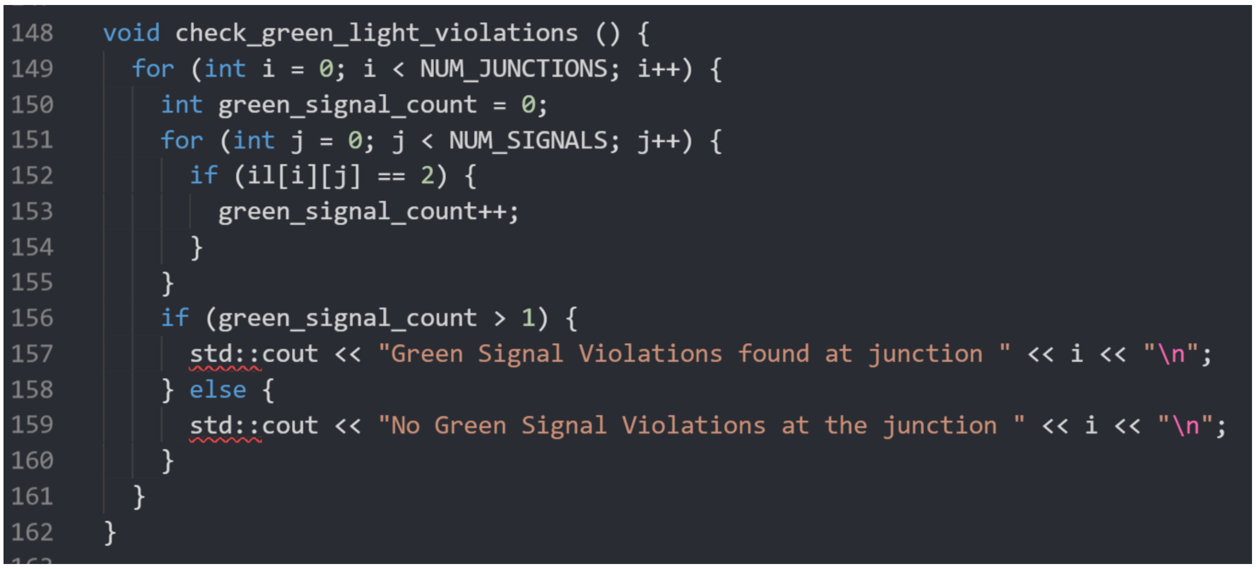


Figure 21 – Plot for test case 2 at t=2

TESTCASE 3: GREEN LIGHT VIOLATIONS

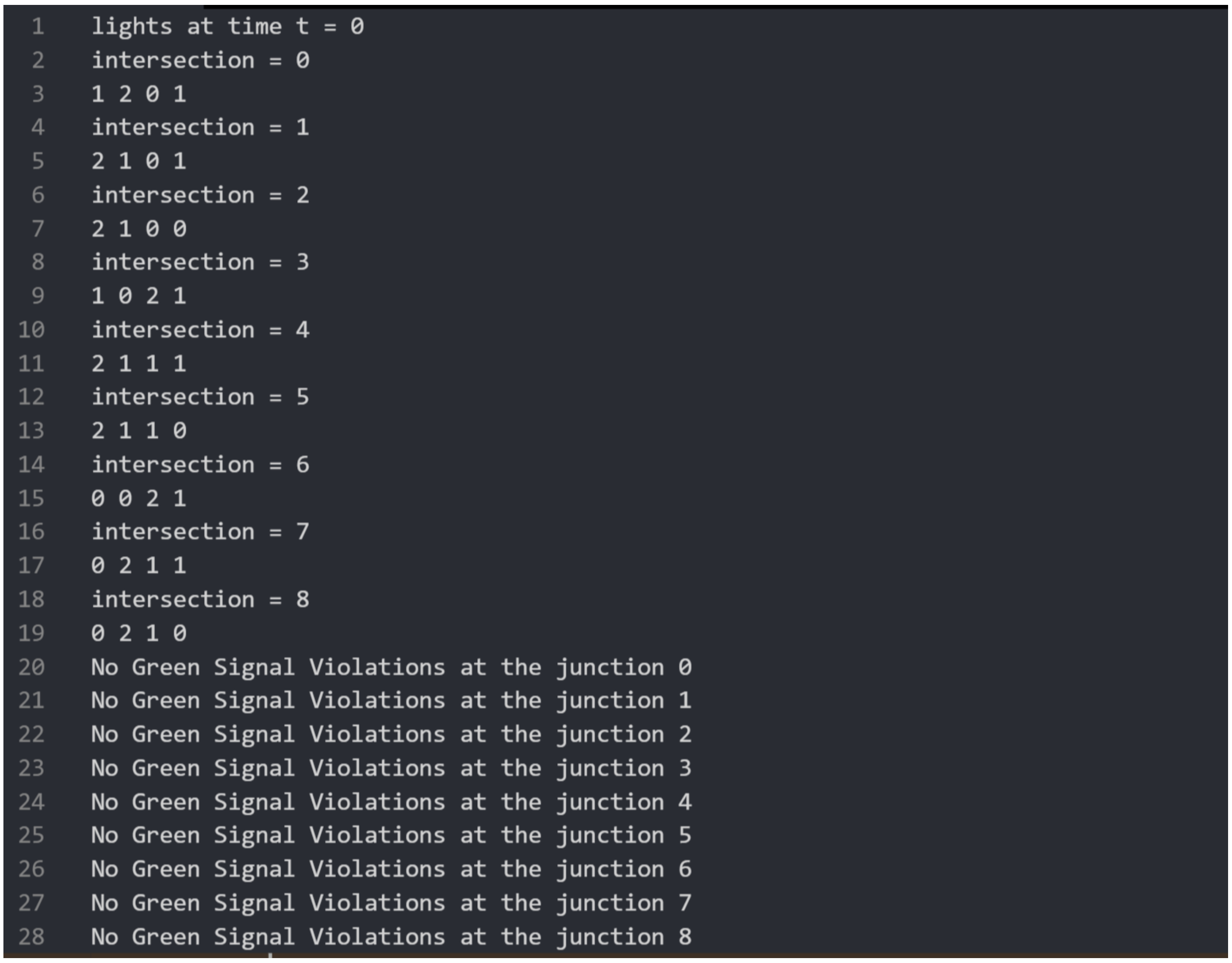
Green light violations occur when there are more than one green light at any intersection. For the same above random traffic, green light violations are checked by the below function.



```cpp
void check_green_light_violations () {
    for (int i = 0; i < NUM_JUNCTIONS; i++) {
        int green_signal_count = 0;
        for (int j = 0; j < NUM_SIGNALS; j++) {
            if (il[i][j] == 2) {
                green_signal_count++;
            }
        }
        if (green_signal_count > 1) {
            std::cout << "Green Signal Violations found at junction " << i << "\n";
        } else {
            std::cout << "No Green Signal Violations at the junction " << i << "\n";
        }
    }
}
```

Figure 22 – check_green_light_violations 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.



```
lights at time t = 0
intersection = 0
1 2 0 1
intersection = 1
2 1 0 1
intersection = 2
2 1 0 0
intersection = 3
1 0 2 1
intersection = 4
2 1 1 1
intersection = 5
2 1 1 0
intersection = 6
0 0 2 1
intersection = 7
0 2 1 1
intersection = 8
0 2 1 0
No Green Signal Violations at the junction 0
No Green Signal Violations at the junction 1
No Green Signal Violations at the junction 2
No Green Signal Violations at the junction 3
No Green Signal Violations at the junction 4
No Green Signal Violations at the junction 5
No Green Signal Violations at the junction 6
No Green Signal Violations at the junction 7
No Green Signal Violations at the junction 8
```

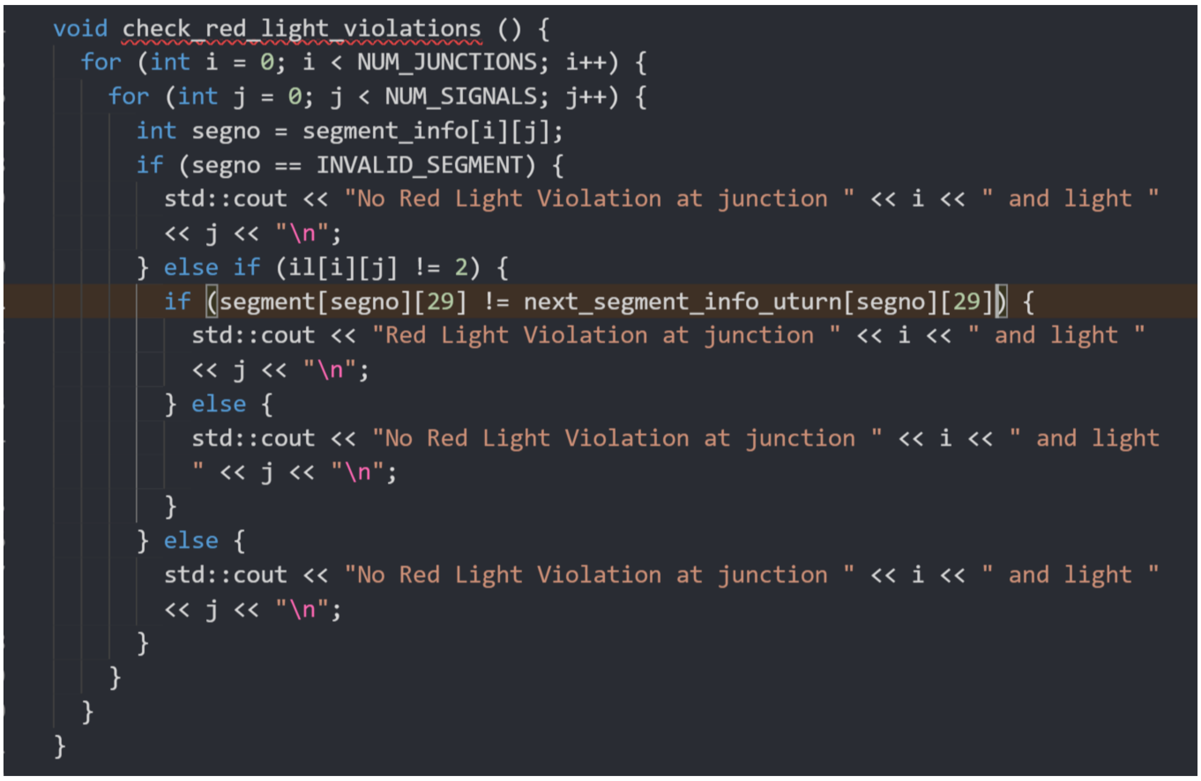
Figure 23 – output result for green light violations

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

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.



```
void check_red_light_violations () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      int segno = segment_info[i][j];
      if (segno == INVALID_SEGMENT) {
        std::cout << "No Red Light Violation at junction " << i << " and light "
        << j << "\n";
      } else if (il[i][j] != 2) {
        if (segment[segno][29] != next_segment_info_uturn[segno][29]) {
          std::cout << "Red Light Violation at junction " << i << " and light "
          << j << "\n";
        } else {
          std::cout << "No Red Light Violation at junction " << i << " and light
          " << j << "\n";
        }
      } else {
        std::cout << "No Red Light Violation at junction " << i << " and light "
        << j << "\n";
      }
    }
  }
}
```

Figure 24 – check_red_light_violations function

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,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,27,0,0,0,25,0,0,0,0,0,0,0,
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,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
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,
0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,57,0,0,0,0,0,0,0,0,0,0,0,0,
0,5,0,34,0,0,63,0,92,0,0,0,0,0,0,0,0,0,0,0,26,0,0,0,0,0,0,0,0,0,0,
21,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,70,0,9,0,8,0,0,0,

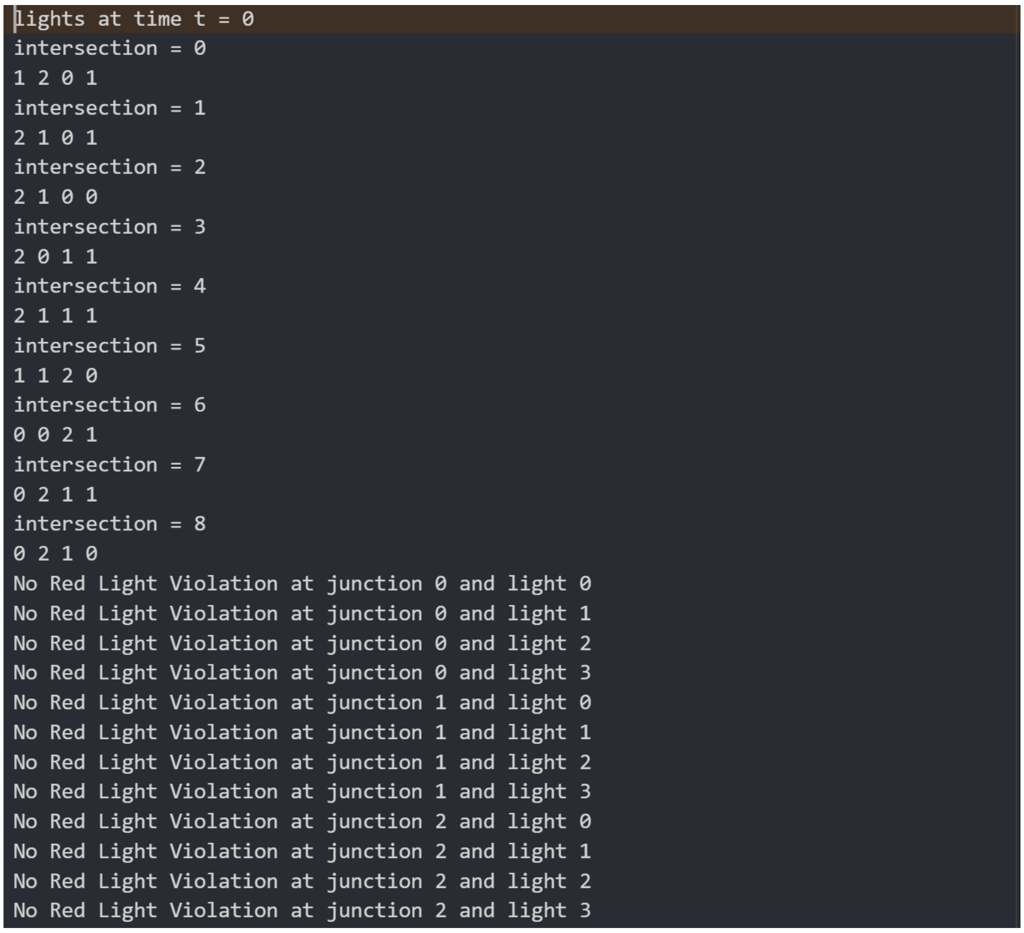
```
0,0,20,0,0,0,48,16,0,0,45,44,14,0,74,0,13,0,72,0,0,0,10,0,0,0,0,37,7,0,
36,0,0,0,0,0,33,0,32,0,0,0,30,0,0,0,0,0,0,0,0,0,24,0,23,0,0,0,
0,0,35,0,0,0,2,0,62,61,31,0,0,0,0,0,28,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,108,0,107,0,0,0,105,0,0,0,0,0,0,0,41,0,0,0,0,0,0,0,97,0,
51,0,50,0,0,0,0,0,47,0,46,0,0,0,0,0,43,0,0,0,0,0,40,0,0,0,0,0,0,0,
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,
0,21,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,70,0,9,0,8,0,0,
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,
0,36,0,0,0,0,0,33,0,32,0,0,0,30,0,0,0,0,0,0,0,0,0,24,0,23,0,0,
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,
0,0,65,0,0,64,0,0,0,0,0,0,0,0,0,119,0,0,0,0,0,116,0,55,0,114,0,113,0,0,
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,
0,0,0,80,0,79,0,0,0,0,0,0,0,0,0,104,0,103,0,0,0,0,0,100,0,0,0,0,0,67,66,
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,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,27,0,0,0,25,0,0,0,0,0,0,
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,
0,0,21,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,70,0,9,0,8,0,
```

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

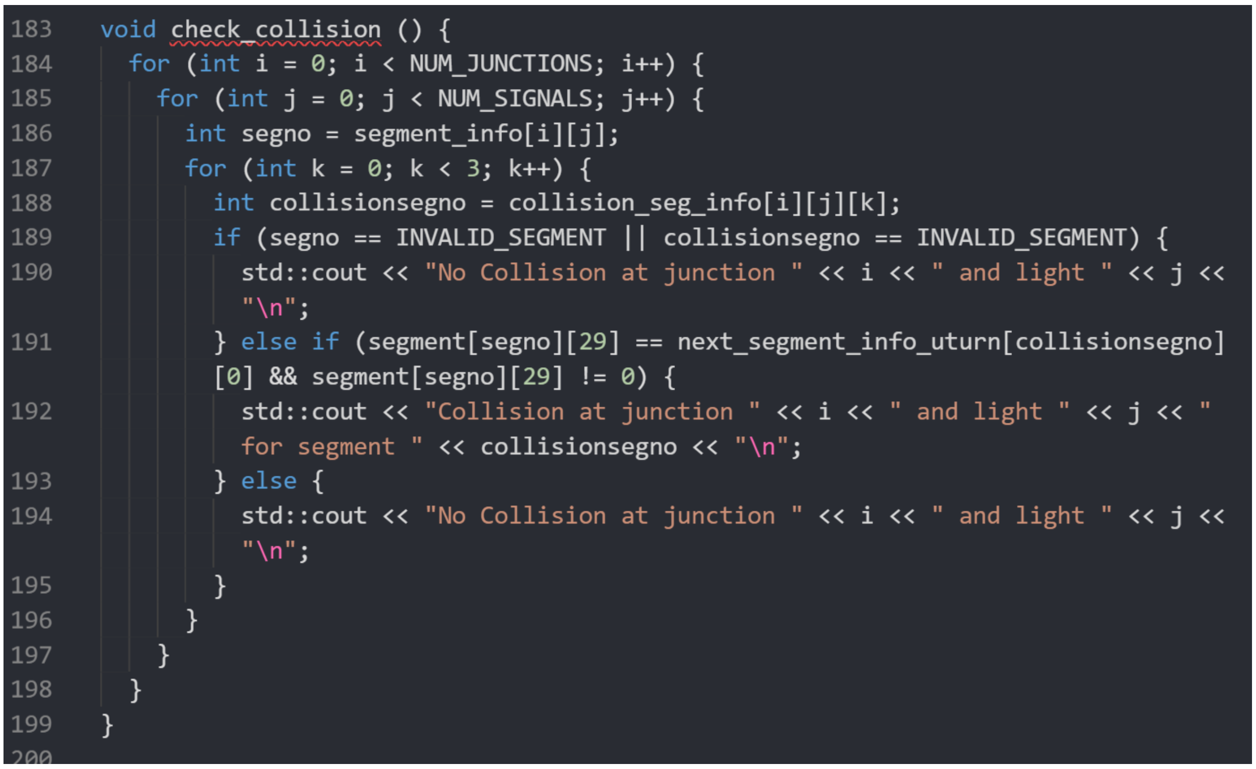
https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE%20A%20SUBMISSION/i-group/redlight_violation567_output%20(1).txt

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


```
lights at time t = 0
intersection = 0
1 2 0 1
intersection = 1
2 1 0 1
intersection = 2
2 1 0 0
intersection = 3
2 0 1 1
intersection = 4
2 1 1 1
intersection = 5
1 1 2 0
intersection = 6
0 0 2 1
intersection = 7
0 2 1 1
intersection = 8
0 2 1 0
No Red Light Violation at junction 0 and light 0
No Red Light Violation at junction 0 and light 1
No Red Light Violation at junction 0 and light 2
No Red Light Violation at junction 0 and light 3
No Red Light Violation at junction 1 and light 0
No Red Light Violation at junction 1 and light 1
No Red Light Violation at junction 1 and light 2
No Red Light Violation at junction 1 and light 3
No Red Light Violation at junction 2 and light 0
No Red Light Violation at junction 2 and light 1
No Red Light Violation at junction 2 and light 2
No Red Light Violation at junction 2 and light 3
```
Figure 25 – output result for red light violations

TESTCASE 5: COLLISIONS

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



```cpp
void check_collision () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      int segno = segment_info[i][j];
      for (int k = 0; k < 3; k++) {
        int collisionsegno = collision_seg_info[i][j][k];
        if (segno == INVALID_SEGMENT || collisionsegno == INVALID_SEGMENT) {
          std::cout << "No Collision at junction " << i << " and light " << j << "\n";
        } else if (segment[segno][29] == next_segment_info_uturn[collisionsegno][0] && segment[segno][29] != 0) {
          std::cout << "Collision at junction " << i << " and light " << j << " for segment " << collisionsegno << "\n";
        } else {
          std::cout << "No Collision at junction " << i << " and light " << j << "\n";
        }
      }
    }
  }
}
```

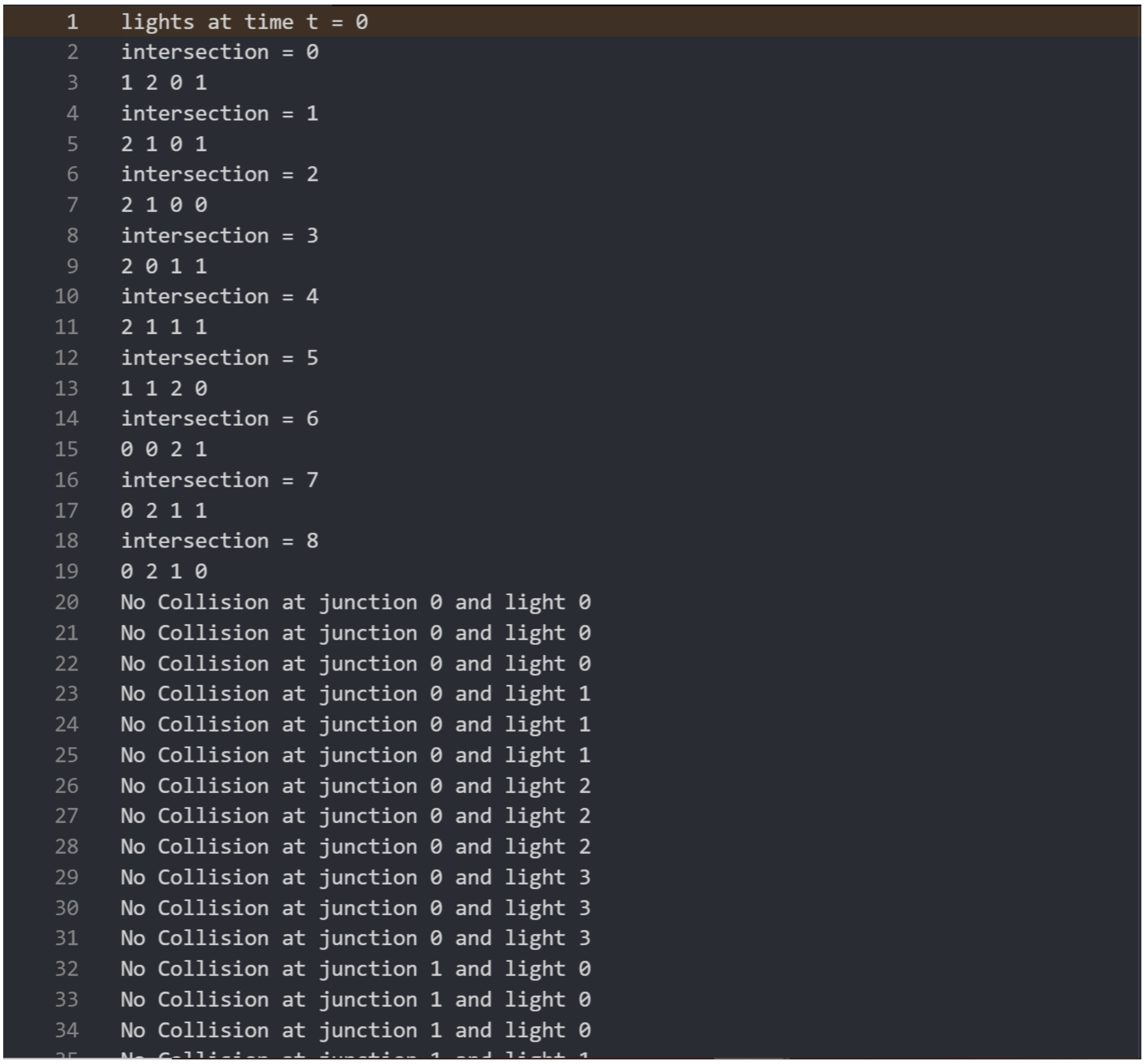
Figure 26 – check_collision function

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.

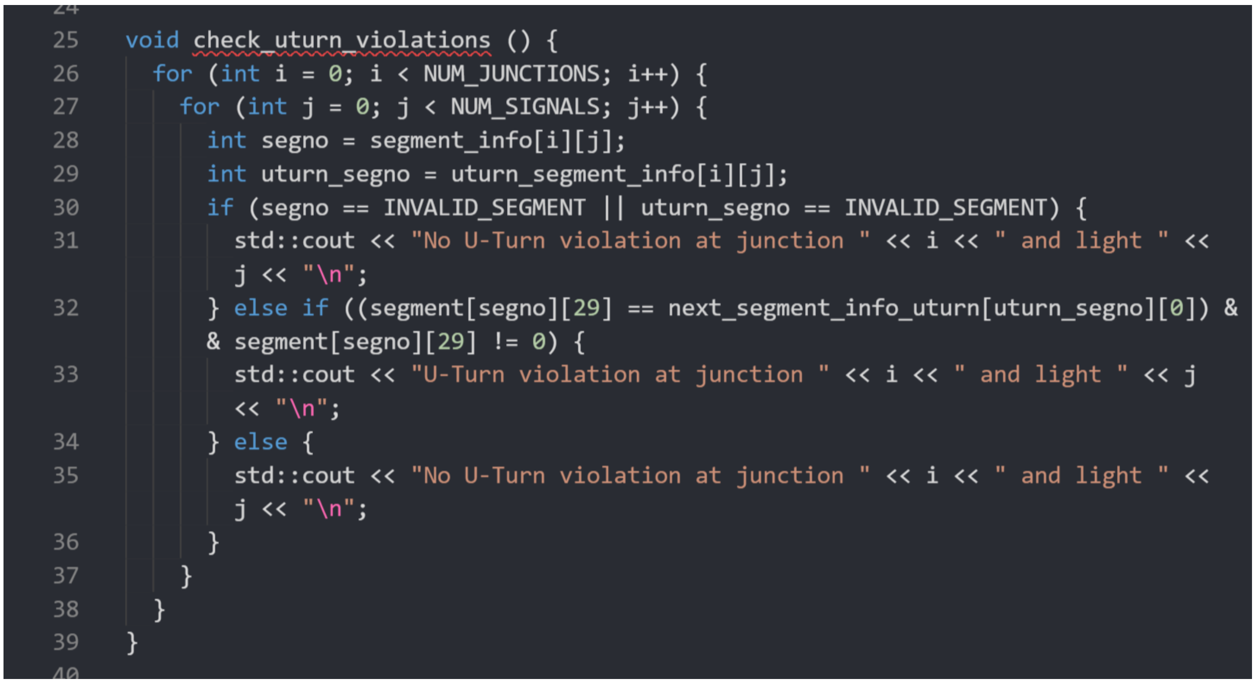


```
lights at time t = 0
intersection = 0
1 2 0 1
intersection = 1
2 1 0 1
intersection = 2
2 1 0 0
intersection = 3
2 0 1 1
intersection = 4
2 1 1 1
intersection = 5
1 1 2 0
intersection = 6
0 0 2 1
intersection = 7
0 2 1 1
intersection = 8
0 2 1 0
No Collision at junction 0 and light 0
No Collision at junction 0 and light 0
No Collision at junction 0 and light 0
No Collision at junction 0 and light 1
No Collision at junction 0 and light 1
No Collision at junction 0 and light 1
No Collision at junction 0 and light 2
No Collision at junction 0 and light 2
No Collision at junction 0 and light 2
No Collision at junction 0 and light 3
No Collision at junction 0 and light 3
No Collision at junction 0 and light 3
No Collision at junction 1 and light 0
No Collision at junction 1 and light 0
No Collision at junction 1 and light 0
```

Figure 27 – output result for collision

TESTCASE 6: U-TURN VIOLATIONS

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



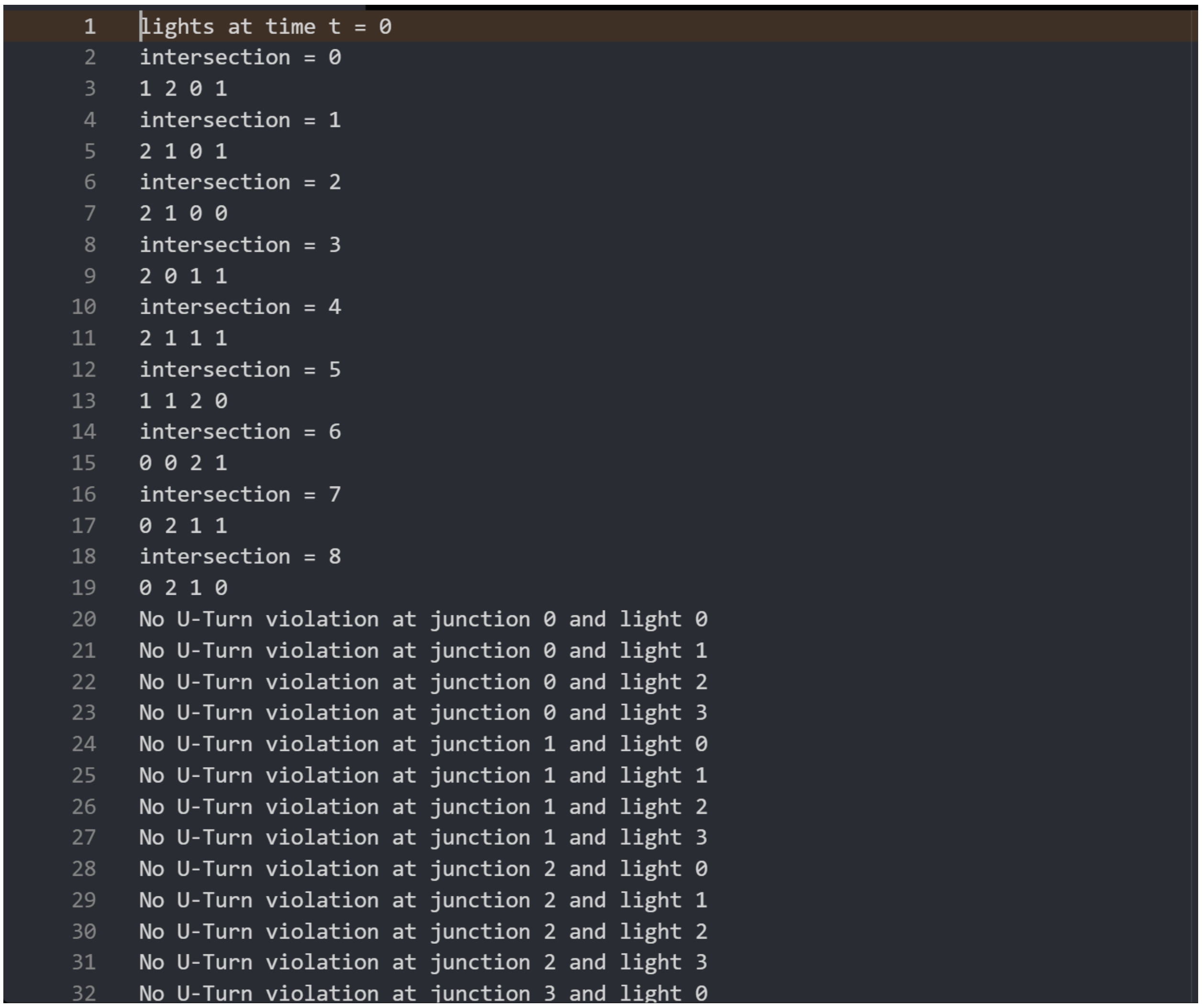
```cpp
void check_uturn_violations () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      int segno = segment_info[i][j];
      int uturn_segno = uturn_segment_info[i][j];
      if (segno == INVALID_SEGMENT || uturn_segno == INVALID_SEGMENT) {
        std::cout << "No U-Turn violation at junction " << i << " and light " << j << "\n";
      } else if ((segment[segno][29] == next_segment_info_uturn[uturn_segno][0]) && segment[segno][29] != 0) {
        std::cout << "U-Turn violation at junction " << i << " and light " << j << "\n";
      } else {
        std::cout << "No U-Turn violation at junction " << i << " and light " << j << "\n";
      }
    }
  }
}
```

Figure 27 – check_uturn_violations function

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.



```
lights at time t = 0
intersection = 0
1 2 0 1
intersection = 1
2 1 0 1
intersection = 2
2 1 0 0
intersection = 3
2 0 1 1
intersection = 4
2 1 1 1
intersection = 5
1 1 2 0
intersection = 6
0 0 2 1
intersection = 7
0 2 1 1
intersection = 8
0 2 1 0
No U-Turn violation at junction 0 and light 0
No U-Turn violation at junction 0 and light 1
No U-Turn violation at junction 0 and light 2
No U-Turn violation at junction 0 and light 3
No U-Turn violation at junction 1 and light 0
No U-Turn violation at junction 1 and light 1
No U-Turn violation at junction 1 and light 2
No U-Turn violation at junction 1 and light 3
No U-Turn violation at junction 2 and light 0
No U-Turn violation at junction 2 and light 1
No U-Turn violation at junction 2 and light 2
No U-Turn violation at junction 2 and light 3
No U-Turn violation at junction 3 and light 0
```

Figure 28 – output result for u-turn violations

TESTCASE 7: THROUGHPUT

Since same traffic is assumed for every time instant and the information about vehicle movement is not available as of now, result for throughput is not available now. Once the vehicle information is obtained in the Phase B, throughput can be easily calculated.

APPENDIX

APPENDIX A – SOURCE CODE

ASSUMPTION OF TRAFFIC – TRAFFICINFO.H

(https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE A SUBMISSION/i-group/trafficinfo.h)

```
#include <iostream>

int traffic1[]={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};

int
traffic2[]={319,0,0,0,317,0,316,0,315,0,314,0,0,0,0,0,0,0,0,0,0,0,308,0,307,0,306,0,305,304,
0,152,151,150,149,0,0,148,0,0,0,0,0,0,0,0,0,0,145,0,0,0,0,0,141,111,0,0,0,0,
242,0,303,0,0,301,270,0,300,238,0,298,267,0,297,0,296,0,0,0,0,0,0,0,0,0,291,0,230,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,215,0,
0,0,288,227,0,0,285,255,225,0,224,0,0,0,282,281,221,0,280,0,0,278,218,0,277,0,276,0,275,0,
0,0,0,0,0,0,241,179,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,232,0,171,0,0,0,
0,243,212,211,181,0,271,0,0,0,209,0,0,0,176,0,0,0,234,0,0,0,202,0,0,199,0,0,229,228,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,190,0,0,0,0,0,187,0,0,0,0,0,
0,0,0,257,197,0,0,0,0,254,164,0,0,0,222,0,251,0,0,188,0,158,0,0,217,0,0,185,155,154,
0,273,213,0,0,0,0,0,0,0,0,0,208,0,207,0,0,0,0,0,0,0,0,0,0,0,201,0,200,0,
0,0,0,0,0,0,0,210,180,0,239,0,0,237,0,0,0,235,204,203,0,0,0,0,0,0,110,109,0,0,
0,0,0,0,287,0,286,0,0,0,0,0,0,0,0,0,0,0,0,279,0,0,0,0,0,0,0,214,0,
0,138,0,0,0,226,165,0,0,194,0,223,191,0,0,220,0,159,0,219,0,0,0,0,0,0,216,0,0,184,
0,0,0,240,0,0,0,0,0,0,0,0,0,0,0,0,236,0,295,0,294,0,233,0,292,231,0,290,0,0,
0,123,0,0,0,0,91,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,137,0,0,
0,0,318,0,0,0,0,0,0,0,0,0,313,0,312,0,311,0,310,0,309,0,0,0,0,0,169,0,0,0,
0,0,0,0,302,0,0,0,0,0,299,0,0,0,0,0,206,205,0,0,0,0,293,0,0,0,0,289,198,196,
0,258,0,0,0,0,0,135,0,0,0,0,0,0,0,0,0,250,249,0,127,0,247,157,0,0,0,0,0,0,
0,0,0,0,195,0,0,0,0,0,0,0,0,0,192,0,0,0,0,189,0,0,0,0,0,0,0,0,0,0,
272,0,182,0,0,0,0,0,0,269,178,0,268,0,0,0,266,265,174,0,0,263,0,0,261,0,0,260,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
167,0,168,166,0,0,256,0,284,162,0,283,253,252,130,0,160,0,0,0,0,0,0,0,156,274,245,244,124
,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,131,0,0,0,0,0,248,0,0,0,0,246,0,125,0,
153,0,0,0,0,0,0,0,0,0,0,0,0,0,175,0,0,0,0,264,142,0,262,172,170,0,0,0,0,259};

int traffic3[]={326,0,0,0,324,0,0,0,322,0,321,0,0,0,319,0,0,0,317,0,316,0,315,0,314,0,0,0,0,0,
160,0,0,0,0,0,156,0,0,0,0,215,124,0,0,152,151,150,149,0,0,148,0,0,0,0,0,0,0,0,
0,0,0,0,0,308,307,306,274,245,244,0,305,304,242,0,303,0,0,301,270,0,300,238,0,298,267,0,2
97,0,
0,0,0,0,0,0,0,0,232,0,171,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
296,0,0,0,0,0,0,0,0,0,291,0,230,0,0,0,288,227,0,0,285,255,225,0,224,0,0,282,281,221,
0,0,0,0,0,0,0,0,187,0,0,0,0,0,0,0,0,0,0,0,241,179,0,0,0,0,0,0,0,0,
0,0,0,189,0,0,218,0,0,0,0,0,0,0,0,243,212,211,181,0,271,0,0,0,209,0,0,0,176,0,
0,0,0,0,0,0,0,0,0,0,201,0,200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,234,0,0,0,202,0,0,199,0,0,229,228,0,0,0,257,197,0,0,0,0,254,164,0,0,0,222,0,
0,0,0,0,0,0,0,0,0,0,0,0,214,0,0,273,213,0,0,0,0,0,0,0,0,0,208,0,207,0,
251,0,0,188,0,248,0,217,0,246,125,0,155,154,0,0,0,0,0,0,0,210,180,0,239,0,0,237,0,0,
236,0,295,0,294,0,233,0,292,231,0,290,0,0,0,0,0,0,287,0,286,0,0,0,0,0,0,0,0,0,
0,235,204,203,0,0,0,0,0,0,110,109,0,0,0,138,0,0,0,226,165,0,0,194,0,223,191,0,0,220,
310,250,249,0,309,0,247,0,0,0,0,0,0,0,0,0,0,240,0,0,0,0,0,0,0,0,0,0,0,0,
0,159,0,219,127,0,0,157,0,0,0,0,0,0,0,123,0,0,0,0,91,0,0,0,0,0,0,0,0,0,
0,0,325,0,0,0,323,0,0,0,0,0,320,0,0,0,318,0,0,0,0,0,0,0,0,0,313,0,312,311,
0,0,0,0,0,0,0,0,0,0,216,0,0,184,0,0,0,0,302,0,0,0,0,0,299,0,0,0,0,0,
0,265,174,0,142,0,261,172,170,0,0,0,0,0,0,258,0,0,0,0,0,135,0,0,0,0,0,0,0,0,
206,205,0,0,0,0,0,0,0,0,0,0,198,196,0,0,0,0,195,0,0,0,0,0,0,0,0,0,192,0,
0,280,190,0,0,278,0,0,277,0,276,0,275,0,272,0,182,0,0,0,0,0,0,269,178,0,268,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,264,0,293,262,0,0,0,0,289,0,259,167,0,168,166,0,0,256,0,284,162,0,283,253,252,130,0,
266,0,0,0,0,263,0,0,0,0,0,260,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,131,
0,0,0,0,279,158,0,0,0,0,0,185,0,0,153,0,0,0,0,0,0,0,0,0,0,0,0,0,175,0};

int traffic4[]={0,0,354,0,353,0,0,0,0,0,350,0,349,0,0,0,347,0,0,0,345,0,0,0,343,0,0,0,341,0,
190,0,0,0,0,0,187,0,0,185,0,184,0,0,0,0,0,241,179,0,0,178,0,0,0,175,0,0,0,
0,339,279,0,0,0,0,0,0,0,0,0,0,0,0,0,302,0,0,0,299,0,329,0,0,0,0,0,326,0,
0,0,0,0,0,0,0,0,0,0,200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
264,0,324,293,262,0,322,0,321,289,259,0,319,0,0,317,256,0,316,315,284,0,314,283,253,252,
0,0,0,0,
0,0,0,0,0,0,0,0,0,0,214,0,0,273,213,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,307,0,274,245,244,216,0,0,0,242,0,0,0,0,270,0,300,238,0,0,267,0,0,0,0,206,205,
0,0,0,0,233,0,0,231,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,230,198,196,0,0,0,227,195,0,0,255,225,0,224,0,0,192,0,221,0,0,
249,0,0,0,247,0,0,0,0,0,0,0,0,0,0,240,0,0,0,0,0,0,0,0,0,0,0,0,236,0,
0,189,0,0,0,0,277,0,0,0,0,0,272,243,0,0,0,0,271,0,0,0,0,0,0,0,0,0,266,0,
325,0,0,0,261,0,0,0,0,0,320,0,258,0,318,0,0,0,0,0,0,0,0,0,0,0,312,311,310,250,
234,0,0,263,0,0,0,0,0,260,229,228,0,0,0,257,0,0,0,0,0,254,164,0,0,222,131,0,251,0,
0,0,278,0,338,337,276,0,336,335,275,0,334,0,333,0,332,0,331,0,0,269,0,328,268,0,0,0,265,0,
0,188,0,158,0,0,0,0,125,0,155,154,153,0,182,0,0,0,0,0,180,0,0,0,0,237,0,0,0,0,
355,0,0,0,217,0,352,0,351,0,0,0,0,0,348,0,0,0,346,0,0,0,344,0,0,0,342,0,0,0,
340,0,0,248,0,0,0,246,0,0,0,0,0,0,0,0,0,0,210,0,330,0,239,0,0,0,327,0,235,204,
0,0,0,0,202,0,0,291,201,0,0,0,0,0,288,0,0,0,285,0,0,0,0,0,0,282,0,0,0,280,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,223,0,0,0,0,220,0,0,
0,0,308,0,306,0,0,0,0,0,305,304,0,0,303,0,301,0,0,0,0,0,298,208,0,297,207,0,296,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
203,0,294,0,323,0,292,199,0,290,0,0,0,0,0,287,197,0,286,0,194,0,0,0,313,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,281,0,0,0,
0,0,309,0,218,0,0,0,0,0,0,0,0,0,212,211,181,0,0,0,0,0,209,0,0,0,176,0,0,295};

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,
0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,57,0,0,0,0,0,0,0,0,0,0,0,
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,
0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,57,0,0,0,0,0,0,0,0,0,0,
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,
97,0,0,0,95,0,94,0,0,0,0,0,0,0,90,0,0,0,0,0,0,87,0,0,0,0,0,0,0,0,0};

int traffic8[]={0,638,0,637,0,636,0,0,0,0,0,0,0,0,0,631,0,0,0,629,0,0,0,0,0,0,0,625,0,0,
473,0,0,0,0,0,0,530,0,0,0,0,467,0,0,0,464,0,432,0,523,0,461,460,0,0,0,0,459,0,
0,623,622,592,0,0,0,0,619,618,558,557,0,0,0,555,0,0,0,613,582,552,0,612,611,551,0,610,0,0,
0,0,0,547,486,0,0,0,0,0,0,543,0,0,0,481,0,0,0,0,0,0,0,0,0,476,0,535,0,0,
548,0,607,577,0,0,0,0,544,0,0,603,0,0,0,601,0,0,0,0,0,0,0,597,0,596,0,595,0,0,
0,0,0,502,0,0,0,0,498,0,0,0,0,0,0,0,0,495,0,493,0,0,0,492,0,0,0,0,0,489,
532,531,0,0,0,561,560,0,589,529,0,497,0,0,0,0,0,0,554,524,0,0,0,0,0,491,0,0,0,0,
0,0,0,514,0,0,0,0,512,511,0,573,0,0,0,0,0,570,0,0,0,508,0,0,0,506,0,504,0,0,
0,517,516,0,0,0,0,0,0,0,0,0,0,0,0,571,0,540,0,479,0,0,0,0,0,0,0,505,0,0,
0,0,0,0,0,0,0,0,0,0,528,0,0,0,0,0,0,525,0,0,0,0,0,0,0,0,0,580,0,0,
0,0,0,0,0,0,0,0,439,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,519,
0,608,0,0,0,546,0,545,0,604,0,0,0,602,0,539,0,600,0,599,0,598,0,0,0,536,0,0,594,534,
0,0,0,0,0,0,0,0,0,0,0,0,0,542,541,450,0,0,0,538,0,537,0,447,0,0,0,0,0,0,
0,563,0,0,0,621,0,620,0,559,0,0,0,617,616,556,0,615,0,614,0,553,0,0,0,0,0,550,549,0,
0,0,0,0,0,0,0,469,0,438,0,0,466,465,0,0,0,433,0,0,0,521,462,0,431,0,0,399,0,0,
0,0,0,0,0,0,0,635,0,634,0,633,0,632,0,0,0,630,0,0,0,628,0,627,0,626,0,0,0,624,
0,533,0,0,0,0,0,0,0,0,0,527,0,0,0,526,0,0,0,0,522,0,0,0,0,0,520,0,0,609,
0,0,0,0,0,0,0,0,0,484,0,0,0,0,0,0,0,478,0,0,0,568,0,0,566,565,475,444,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,507,0,0,0,0,0,0,
0,0,0,591,501,0,0,499,0,0,0,588,0,587,0,0,0,585,494,0,0,583,0,0,0,581,0,0,0,579,
0,472,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
488,0,606,605,576,575,574,483,0,0,0,482,0,572,0,0,0,480,0,569,448,0,477,0,0,0,0,0,564,562,
578,0,0,0,0,0,0,0,0,454,0,0,0,0,0,0,0,0,0,0,0,0,0,567,0,0,0,0,0,0,
0,593,0,0,0,0,0,590,0,0,0,0,0,496,0,586,0,0,0,584,0,0,0,0,0,490,0,0,0,0};

int traffic9[]={639,0,638,0,637,0,636,0,0,0,0,0,0,0,0,0,631,0,0,0,629,0,0,0,0,0,0,0,625,0,
0,473,0,0,0,0,0,0,530,0,0,0,0,467,0,0,0,464,0,432,0,523,0,461,460,0,0,0,0,459,
562,0,623,622,592,0,0,0,0,619,618,558,557,0,0,0,555,0,0,0,613,582,552,0,612,611,551,0,610
,0,
489,0,0,0,547,486,0,0,0,0,0,0,543,0,0,0,481,0,0,0,0,0,0,0,0,0,476,0,535,0,
0,548,0,607,577,0,0,0,0,544,0,0,603,0,0,0,601,0,0,0,0,0,0,0,597,0,596,0,595,0,
0,0,0,0,502,0,0,0,0,498,0,0,0,0,0,0,0,0,495,0,493,0,0,0,492,0,0,0,0,0,
0,532,531,0,0,0,561,560,0,589,529,0,497,0,0,0,0,0,0,554,524,0,0,0,0,0,491,0,0,0,
0,0,0,0,514,0,0,0,0,512,511,0,573,0,0,0,0,0,570,0,0,0,508,0,0,0,506,0,504,0,
0,0,517,516,0,0,0,0,0,0,0,0,0,0,0,0,571,0,540,0,479,0,0,0,0,0,0,0,505,0,
534,0,0,0,0,0,0,0,0,0,0,528,0,0,0,0,0,0,525,0,0,0,0,0,0,0,0,0,580,0,
0,0,0,0,0,0,0,0,0,439,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,608,0,0,0,546,0,545,0,604,0,0,0,602,0,539,0,600,0,599,0,598,0,0,0,536,0,0,594,
519,0,0,0,0,0,0,0,0,0,0,0,0,0,542,541,450,0,0,0,538,0,537,0,447,0,0,0,0,0,
624,0,563,0,0,0,621,0,620,0,559,0,0,0,617,616,556,0,615,0,614,0,553,0,0,0,0,0,550,549,
0,0,0,0,0,0,0,0,469,0,438,0,0,466,465,0,0,0,433,0,0,0,521,462,0,431,0,0,399,0,
0,0,0,0,0,0,0,0,635,0,634,0,633,0,632,0,0,0,630,0,0,0,628,0,627,0,626,0,0,0,
0,0,533,0,0,0,0,0,0,0,0,0,527,0,0,0,526,0,0,0,0,522,0,0,0,0,0,520,0,0,
0,0,0,0,0,0,0,0,0,0,484,0,0,0,0,0,0,0,478,0,0,0,568,0,0,566,565,475,444,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,507,0,0,0,0,0,
0,0,0,0,591,501,0,0,499,0,0,0,588,0,587,0,0,0,585,494,0,0,583,0,0,0,581,0,0,0,
0,0,472,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
609,488,0,606,605,576,575,574,483,0,0,0,482,0,572,0,0,0,480,0,569,448,0,477,0,0,0,0,0,564,
579,578,0,0,0,0,0,0,0,0,454,0,0,0,0,0,0,0,0,0,0,0,0,0,567,0,0,0,0,0,
0,0,593,0,0,0,0,0,590,0,0,0,0,0,496,0,586,0,0,0,584,0,0,0,0,0,490,0,0,0};

int traffic10[]={0,639,0,638,0,637,0,636,0,0,0,0,0,0,0,0,0,631,0,0,0,629,0,0,0,0,0,0,0,625,
0,0,473,0,0,0,0,0,0,530,0,0,0,0,467,0,0,0,464,0,432,0,523,0,461,460,0,0,0,0,
564,562,0,623,622,592,0,0,0,0,619,618,558,557,0,0,0,555,0,0,0,613,582,552,0,612,611,551,0
,610,
0,489,0,0,0,547,486,0,0,0,0,0,0,543,0,0,0,481,0,0,0,0,0,0,0,0,0,476,0,535,
0,0,548,0,607,577,0,0,0,0,544,0,0,603,0,0,0,601,0,0,0,0,0,0,0,597,0,596,0,595,
0,0,0,0,0,502,0,0,0,0,498,0,0,0,0,0,0,0,0,495,0,493,0,0,0,492,0,0,0,0,
0,0,532,531,0,0,0,561,560,0,589,529,0,497,0,0,0,0,0,0,554,524,0,0,0,0,0,491,0,0,
0,0,0,0,0,514,0,0,0,0,512,511,0,573,0,0,0,0,0,570,0,0,0,508,0,0,0,506,0,504,
0,0,0,517,516,0,0,0,0,0,0,0,0,0,0,0,0,571,0,540,0,479,0,0,0,0,0,0,0,505,
0,534,0,0,0,0,0,0,0,0,0,0,528,0,0,0,0,0,0,525,0,0,0,0,0,0,0,0,0,580,
0,0,0,0,0,0,0,0,0,0,439,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
549,0,0,608,0,0,0,546,0,545,0,604,0,0,0,602,0,539,0,600,0,599,0,598,0,0,0,536,0,0,
0,519,0,0,0,0,0,0,0,0,0,0,0,0,0,542,541,450,0,0,0,538,0,537,0,447,0,0,0,0,
0,624,0,563,0,0,0,621,0,620,0,559,0,0,0,617,616,556,0,615,0,614,0,553,0,0,0,0,0,550,
0,0,0,0,0,0,0,0,0,469,0,438,0,0,466,465,0,0,0,433,0,0,0,521,462,0,431,0,0,399,
0,0,0,0,0,0,0,0,0,635,0,634,0,633,0,632,0,0,0,630,0,0,0,628,0,627,0,626,0,0,
0,0,0,533,0,0,0,0,0,0,0,0,0,527,0,0,0,526,0,0,0,0,522,0,0,0,0,0,520,0,
0,0,0,0,0,0,0,0,0,0,0,484,0,0,0,0,0,0,0,478,0,0,0,568,0,0,566,565,475,444,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,507,0,0,0,0,
0,0,0,0,0,591,501,0,0,499,0,0,0,588,0,587,0,0,0,585,494,0,0,583,0,0,0,581,0,0,
0,0,0,472,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,609,488,0,606,605,576,575,574,483,0,0,0,482,0,572,0,0,0,480,0,569,448,0,477,0,0,0,0,0,
0,579,578,0,0,0,0,0,0,0,0,454,0,0,0,0,0,0,0,0,0,0,0,0,567,0,0,0,0,
594,0,0,593,0,0,0,0,0,590,0,0,0,0,0,496,0,586,0,0,0,584,0,0,0,0,0,490,0,0};

int no_traffic[24][30]={0};
extern int segment[24][30];
extern int next_segment_info_uturn[24][30];

void assume_notraffic()
{
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30; j++) {
      segment[i][j] = no_traffic[i][j];
      k++;
    }
  }
}

void assume_traffic1()
{
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30; j++) {
      segment[i][j] = traffic1[k];
      k++;
    }
  }
}

void assume_traffic2()
{
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30; j++) {
      segment[i][j] = traffic2[k];
      k++;
    }
  }
}

void assume_traffic3()
{
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30; j++) {
      segment[i][j] = traffic3[k];
      k++;
    }
  }
}

void assume_traffic4()
{
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30; j++) {
      segment[i][j] = traffic4[k];
      k++;
    }
  }
}

void assume_traffic5()
{
  int k=0;
  for (int i = 0; i < 24; i++) {
    for(int j = 0; j < 30;j++) {
      segment[i][j] = traffic5[k];
      next_segment_info_uturn[i][j] = traffic6[k];
      k=k+1;
    }
  }
}

void assume_traffic6() {
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for (int j = 0; j < 30; j++) {
      segment[i][j] = traffic6[k];
      next_segment_info_uturn[i][j] = traffic7[k];
      k++;
    }
  }
}

void assume_traffic7() {
  int k = 0;
  for (int i = 0; i < 24; i++) {
    for (int j = 0; j < 30; j++) {
      segment[i][j] = traffic7[k];
      k++;
    }
  }
}
```

INITIALIZATION OF SIGNAL LIGHTS AND PRIORITY – SIGNAL.H

(https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE A SUBMISSION/i-group/signal.h)

```
#include "controller.h"

//0-off,1-red,2-green
//initial lights at intersection
void initial_signal_priority()
{
  for(int i=0;i<=8;i++)
  {
    for(int j=0;j<=3;j++)
    {

if((i==0&&j==2)||(i==1&&j==2)||(i==2&&j==2)||(i==2&&j==3)||(i==3&&j==1)||(i==5&&j
==3)||(i==6&&j==0)||(i==6&&j==1)||(i==7&&j==0)||(i==8&&j==0)||(i==8&&j==3))
      {
        il[i][j]=0;
      }
      else if
((i==0&&j==1)||(i==1&&j==1)||(i==2&&j==1)||(i==5&&j==2)||(i==7&&j==3)||(i==6&&j=
=3)||(i==3&&j==0))
      {
        il[i][j]=2;
      }
      else
        il[i][j]=1;
    }

  }
  for(int i=0;i<=8;i++)
    for(int j=0;j<=3;j++)
    {
      p_light[i][j]=0;
    }
}

void priority_signal()
{
  p_light[0][0]=segment[14][29];
  p_light[0][3]=segment[1][29];
  p_light[1][0]=segment[21][29];
  p_light[1][1]=segment[0][29];
  p_light[1][3]=segment[3][29];
  p_light[2][0]=segment[5][29];
  p_light[2][1]=segment[2][29];
  p_light[3][0]=segment[12][29];
  p_light[3][2]=segment[15][29];
  p_light[3][3]=segment[17][29];
  p_light[4][0]=segment[22][29];
  p_light[4][1]=segment[16][29];
  p_light[4][2]=segment[20][29];
  p_light[4][3]=segment[19][29];
  p_light[5][0]=segment[7][29];
  p_light[5][1]=segment[18][29];
  p_light[5][2]=segment[4][29];
  p_light[6][2]=segment[13][29];
  p_light[6][3]=segment[10][29];
  p_light[7][1]=segment[11][29];
  p_light[7][2]=segment[23][29];
  p_light[7][3]=segment[8][29];
  p_light[8][1]=segment[9][29];
  p_light[8][2]=segment[6][29];
}
```

CONTROLLING AND SCHEDULING FOR SWITCHING OF LIGHTS – CONTROLLER.H

(https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE A SUBMISSION/i-group/controller.h)

```
#include <queue>
#include <vector>
using namespace std;

#define NUM_JUNCTIONS 9
#define NUM_SIGNALS 4
#define INVALID_SEGMENT 100
```

```
// global variables
extern int il[NUM_JUNCTIONS][NUM_SIGNALS];
extern int p_light[NUM_JUNCTIONS][NUM_SIGNALS];
extern int segment[24][30];
vector<queue<int>> q(NUM_JUNCTIONS);

//segment information for each light at each intersection to check while assigning priority
int segment_info[NUM_JUNCTIONS][NUM_SIGNALS] = {{14, 100, 100, 1}, {21, 0, 100,
3}, {5, 2, 100, 100}, {12, 100, 15, 17}, {22, 16, 20, 19}, {7, 18, 4, 100}, {100, 100, 13, 10},
{100, 11, 23, 8}, {100, 9, 6, 100}};

int uturn_segment_info[NUM_JUNCTIONS][NUM_SIGNALS] = {{15, 100, 100, 0}, {20,
1, 100, 2}, {4, 3, 100, 100}, {13, 100, 14, 16}, {23, 17, 21, 18}, {6, 19, 5, 100}, {100, 100,
12, 11}, {100, 10, 22, 9}, {100, 8, 7, 100}};

int collision_seg_info[NUM_JUNCTIONS][NUM_SIGNALS][3] = {{{0, 15, 100}, {15,
100, 100}, {100, 100, 0}, {100, 0, 15}}, {{2, 20, 1}, {20, 1, 100}, {1, 100, 2}, {100, 2, 20}},
{{4, 3, 100}, {4, 3, 100}, {3, 100, 100}, {100, 100, 4}}, {{16, 13, 100}, {13, 100, 14}, {100,
14, 16}, {14, 16, 13}}, {{18, 23, 17}, {23, 17, 21}, {17, 21, 18}, {21, 18, 23}}, {{100, 6,
19}, {6, 19, 5}, {19, 5, 100}, {5, 100, 6}}, {{11, 100, 100}, {100, 100, 12}, {100, 12, 11},
{12, 11, 100}}, {{9, 100, 10}, {100, 10, 22}, {10, 22, 9}, {22, 9, 100}}, {{100, 100, 8},
{100, 8, 7}, {8, 7, 100}, {7, 100, 100}}};

extern int next_segment_info_uturn[24][30];

void check_uturn_violations () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      int segno = segment_info[i][j];
      int uturn_segno = uturn_segment_info[i][j];
      if (segno == INVALID_SEGMENT || uturn_segno == INVALID_SEGMENT) {
        std::cout << "No U-Turn violation at junction " << i << " and light " << j << "\n";
      } else if ((segment[segno][29] == next_segment_info_uturn[uturn_segno][0]) &&
segment[segno][29] != 0) {
        std::cout << "U-Turn violation at junction " << i << " and light " << j << "\n";
      } else {
        std::cout << "No U-Turn violation at junction " << i << " and light " << j << "\n";
      }
    }
  }
}

void initialise_queue () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      q[i].push (j);
    }
  }
}

int get_car_count_from_segment (int segmentno) {
  int cnt = 0;
  if (segmentno == INVALID_SEGMENT) {
    return 0;
  }
  for (int i = 29; i > 18; i--) {
    if (segment[segmentno][i] > 0) {
      cnt++;
    }
  }
  return cnt;
}

int check_priority (int num_cars[]) {
  int index = 0;
  for (int i = 1; i < NUM_SIGNALS; i++) {
    if (num_cars[index] < num_cars[i]) {
      index = i;
    }
  }
  return index;
}

void scheduler (int jun[], int priority, int pri[], int first, int num_cars[]) {

  if (q[first].empty ()) {
    initialise_queue ();
    scheduler (jun, priority, pri, first, num_cars);
    return;
  }

  int new_light = q[first].front ();
  q[first].pop ();

  if (priority == 0) {
    //at first junction, always allow cars from node A if no other car is returning back to A
    if (first == 0) {
      jun[1] = 2; //change light to green for the signal which faces A
      return;
    }

    //simple round robin
    if (jun[new_light] != 0) {
      jun[new_light] = 2; //set to green
      q[first].push (new_light);
      return;
    } else {
      q[first].push (new_light);
      scheduler (jun, priority, pri, first, num_cars);
      return;
    }

  } else {
    int high_pri = check_priority (num_cars);;
    //if priority = 1 and light is off, change the priority to 0
    if (jun[new_light] == 0) {
      q[first].push (new_light);
      scheduler (jun, priority, pri, first, num_cars);
      return;
    } else if (jun[new_light] != 0 && new_light == high_pri) {
      //set to green for the signal with priority = max_cars
      jun[new_light] = 2;
      q[first].push (new_light);
      return;
    } else {
      q[first].push (new_light);
      scheduler (jun, priority, pri, first, num_cars);
      return;
    }
  }
}

void controller (int time) {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    int priority = 0;
    int num_cars[NUM_SIGNALS] = {0};
    for (int j = 0; j < NUM_SIGNALS; j++) {
      if (il[i][j] == 2) {
        il[i][j] = 1; //change the green signals to red
      }
      if (p_light[i][j] > 1) {
        priority = 1;
        int segmentno = segment_info[i][j];
        num_cars[j] = get_car_count_from_segment (segmentno);
      }
    }
    scheduler (il[i], priority, p_light[i], i, num_cars);
  }

  std::cout << "lights at time t = " << time << "\n";
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    std::cout << "intersection = " << i << "\n";
    for (int j = 0; j < NUM_SIGNALS; j++) {
      std::cout << il[i][j] << " ";
    }
    std::cout << "\n";
  }
}

void check_green_light_violations () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    int green_signal_count = 0;
    for (int j = 0; j < NUM_SIGNALS; j++) {
      if (il[i][j] == 2) {
        green_signal_count++;
      }
    }
    if (green_signal_count > 1) {
      std::cout << "Green Signal Violations found at junction " << i << "\n";
    } else {
      std::cout << "No Green Signal Violations at the junction " << i << "\n";
    }
  }
}

void check_red_light_violations () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      int segno = segment_info[i][j];
      if (segno == INVALID_SEGMENT) {
        std::cout << "No Red Light Violation at junction " << i << " and light " << j << "\n";
      } else if (il[i][j] != 2) {
        if (segment[segno][29] != next_segment_info_uturn[segno][29]) {
          std::cout << "Red Light Violation at junction " << i << " and light " << j << "\n";
        } else {
          std::cout << "No Red Light Violation at junction " << i << " and light " << j << "\n";
        }
      } else {
        std::cout << "No Red Light Violation at junction " << i << " and light " << j << "\n";
      }
    }
  }
}

void check_collision () {
  for (int i = 0; i < NUM_JUNCTIONS; i++) {
    for (int j = 0; j < NUM_SIGNALS; j++) {
      int segno = segment_info[i][j];
      for (int k = 0; k < 3; k++) {
        int collisionsegno = collision_seg_info[i][j][k];
        if (segno == INVALID_SEGMENT || collisionsegno == INVALID_SEGMENT) {
          std::cout << "No Collision at junction " << i << " and light " << j << "\n";
        } else if (segment[segno][29] == next_segment_info_uturn[collisionsegno][0] &&
segment[segno][29] != 0) {
          std::cout << "Collision at junction " << i << " and light " << j << " for segment " <<
collisionsegno << "\n";
        } else {
          std::cout << "No Collision at junction " << i << " and light " << j << "\n";
        }
      }
    }
  }
}
```

MAIN FUNCTION – MAIN.CPP

(https://github.com/rishyasankar/Formal_verification_project/blob/main/PHASE A SUBMISSION/i-group/main.cpp)

```
#include <iostream>
using namespace std;

#include "signal.h"
#include "trafficinfo.h"

int il[NUM_JUNCTIONS][NUM_SIGNALS];
int p_light[NUM_JUNCTIONS][NUM_SIGNALS];
int segment[24][30];
int next_segment_info_uturn[24][30];
int main () {
  initialise_queue ();
  int time = 0;

  assume_traffic5();
  initial_signal_priority();
  priority_signal();
  while (time < 3600) {
    controller (time);
    check_green_light_violations ();
    //check_uturn_violations ();
    //check_red_light_violations ();
    //check_collision ();
    time = time + 2;
    /*if (time == 2) {
      assume_traffic6 ();
    } else {
      assume_traffic7 ();
    }*/
  }
  return 0;
}
```

APPENDIX B – PLOTTING CODE

```
##for plotting traffic system at different time intervals showing switching of lights

import matplotlib.pyplot as plt

m = 0
g_sig = [[[m for i in range(4)] for j in range(9)] for k in range(1800)]
z=0
yy = 0

with open ("output_traffic3.txt",'r') as f:
  for line in f:
    if "time" in line:
      z = z+1
      yy = 0
    else:
      tmp_lst = []
      for g in line.split():
          tmp_lst.append(int(g))
      g_sig[z-1][yy] = tmp_lst
      yy = yy+1
```

```
x = [[1, 3, 1, -1], [31, 33, 31, 29], [61, 63, 61, 59], [1, 3, 1, -1], [31, 33, 31, 29], [61, 63, 61,
59], [1, 3, 1, -1], [31, 33, 31, 29], [61, 63, 61, 59]]

y = [[63, 61, 59, 61], [63, 61, 59, 61], [63, 61, 59, 61], [33, 31, 29, 31], [33, 31, 29, 31], [33,
31, 29, 31], [3, 1, -1, 1], [3, 1, -1, 1], [3, 1, -1, 1]]

size=100
t=0

for z in range(1800):
  for i in range(9):
    for j in range(len(g_sig[z][i])):
      if g_sig[z][i][j] == 1:
        colour = "red"
      elif g_sig[z][i][j] == 2:
        colour = "green"
      else:
        colour = "black"
      xi = x[i][j]
      yi = y[i][j]
      plt.scatter (xi, yi,s=size, color = colour)
      name = "notraffic"+str(z)+".png"
      plt.title('time=%d' %t)
      plt.savefig(name)
  t=t+2
  print("saved")
```