

AI-Driven Traffic Light Control System

(Using C++ and Multiple Data Structures)

AIM:

To design and implement an AI-driven traffic light control system using C++, incorporating fundamental data structures such as graphs, queues, stacks, linked lists, trees, and hash tables to efficiently manage intersections, vehicles, and signal states.

ALGORITHM:

Intersection Graph Algorithm

1. Create an adjacency list to represent intersections (nodes) and connecting roads (edges).
2. Insert edges to model the traffic network.
3. Use this graph to validate signal paths and manage traffic flow at each node.

Vehicle Queue Algorithm:

1. For each lane, create a queue.
2. When a vehicle arrives, enqueue it.
3. When a signal turns green, dequeue vehicles.
4. Queue length is sent to the AI analyzer.

Manual Override & Undo Stack Algorithm:

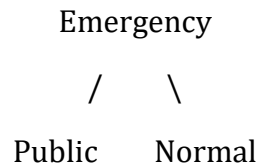
1. When the operator changes the signal state manually, push the previous state into the stack.
2. When "Undo" is pressed, pop the last state and restore it.

Signal Cycle Linked List Algorithm:

1. Create a circular linked list: Green → Yellow → Red → Green.
2. Traverse the list to update signal states in order.

Priority Tree Algorithm:

Create a binary tree:



Based on vehicle type, AI assigns priority:

- Emergency → highest priority
- Public transport → medium
- Normal → lowest

AI Decision Algorithm:

1. Read queue length.
2. If queue > threshold → extend green time.
3. If emergency vehicle detected → jump to high priority.
4. Else → follow normal linked list cycle.

Hash Table Algorithm for Vehicle Mapping:

1. Use a hash function: $\text{index} = \text{vehicleID} \% \text{tableSize}$.
2. Store vehicle type and priority.
3. Use hash table to check if emergency vehicle is present.

C++ PROGRAM:

```
#include <iostream>

#include <vector>

#include <queue>

#include <stack>

#include <unordered_map>

using namespace std;
```

```
// LINKED LIST FOR SIGNAL CYCLE
```

```
struct SignalNode {  
    string color;  
    SignalNode* next;  
    SignalNode(string c) : color(c), next(NULL) {}  
};
```

```
// TREE FOR PRIORITY
```

```
struct PriorityNode {  
    string type;  
    PriorityNode* left;  
    PriorityNode* right;  
    PriorityNode(string t) : type(t), left(NULL), right(NULL) {}  
};
```

```
// GRAPH FOR INTERSECTIONS
```

```
vector<vector<int>> graph(5); // 5 intersections
```

```
void addRoad(int a, int b) {  
    graph[a].push_back(b);  
    graph[b].push_back(a);  
}
```

```
// HASH TABLE FOR VEHICLE ID
```

```
unordered_map<int, string> vehicleTable;
```

```
// QUEUE FOR VEHICLES
```

```
queue<int> vehicleQueue;
```

```
// STACK FOR MANUAL OVERRIDE
stack<string> overrideStack;

// AI DECISION
string AI_Decide(int queueSize, bool emergency) {
    if (emergency)
        return "GREEN (Emergency Priority)";
    else if (queueSize > 5)
        return "GREEN Extended";
    else
        return "NORMAL Cycle";
}

int main() {
    // Build signal cycle (Linked List)
    SignalNode* green = new SignalNode("Green");
    SignalNode* yellow = new SignalNode("Yellow");
    SignalNode* red = new SignalNode("Red");

    green->next = yellow;
    yellow->next = red;
    red->next = green; // Circular

    SignalNode* current = green;

    // Build priority tree
    PriorityNode* root = new PriorityNode("Emergency");
    root->left = new PriorityNode("Public");
    root->right = new PriorityNode("Normal");
```

```
// Build graph
addRoad(0, 1);
addRoad(1, 2);
addRoad(2, 3);

// Insert vehicles
vehicleQueue.push(101);
vehicleQueue.push(102);
vehicleQueue.push(103);

// Hash table entries
vehicleTable[101] = "Normal";
vehicleTable[102] = "Public";
vehicleTable[103] = "Emergency";

cout << "\n--- AI TRAFFIC CONTROL SYSTEM ---\n";

// Check for emergency vehicle
bool emergency = false;
queue<int> temp = vehicleQueue;
while (!temp.empty()) {
    if (vehicleTable[temp.front()] == "Emergency")
        emergency = true;
    temp.pop();
}

// AI Decision
string decision = AI_Decide(vehicleQueue.size(), emergency);
```

```
cout << "\nAI Decision: " << decision << endl;

// Show current signal cycle
cout << "Current Signal: " << current->color << endl;

// Manual override
cout << "\nManual Override: Changing to RED\n";
overrideStack.push(current->color);
current = red;

cout << "Signal is now: " << current->color << endl;

// Undo operation
cout << "\nUndoing Override...\n";
string lastState = overrideStack.top();
overrideStack.pop();

cout << "Restored Signal: " << lastState << endl;

return 0;
}
```

SAMPLE OUTPUT:

--- AI TRAFFIC CONTROL SYSTEM ---

AI Decision: GREEN (Emergency Priority)

Current Signal: Green

Manual Override: Changing to RED

Signal is now: Red

Undoing Override...

Restored Signal: Green

RESULT:

The system successfully simulates:

- AI-based signal timing
- Emergency vehicle priority
- Signal cycle transitions
- Undo operation using stack
- Vehicle management through queues
- Intersection modeling using graphs
- Vehicle lookup using hash tables

All required data structures work together in one integrated system.

CONCLUSION:

The project demonstrates how AI logic combined with multiple data structures can improve real-time traffic signal management. The use of C++ provides efficient memory handling and faster execution, making the system suitable for practical smart-city implementations.