Traffic Simulation

Simulate the Flow of Vehicular Traffic Using Priority Queue

SPRING 2020

COURSE

Program Structure and Algorithms (INFO 6205)

INSTRUCTOR

Dr. Robin Hillyard

PRESENTERS

Rituja Mahajan
(NUID 001023617)

Sajal Sood
(NUID 001054338)

Naresh Agarwal
(NUID 001054600)

Table of Contents

Chapter 1: Overview

Traffic and congestion phenomena belong to our everyday experience. Many factors can contribute to traffic congestion such as:

- Accidents and breakdowns
- Road construction and repair
- Harsh weather conditions

One can't always predict where these disturbances will occur, but they still heavily impact traffic flow.

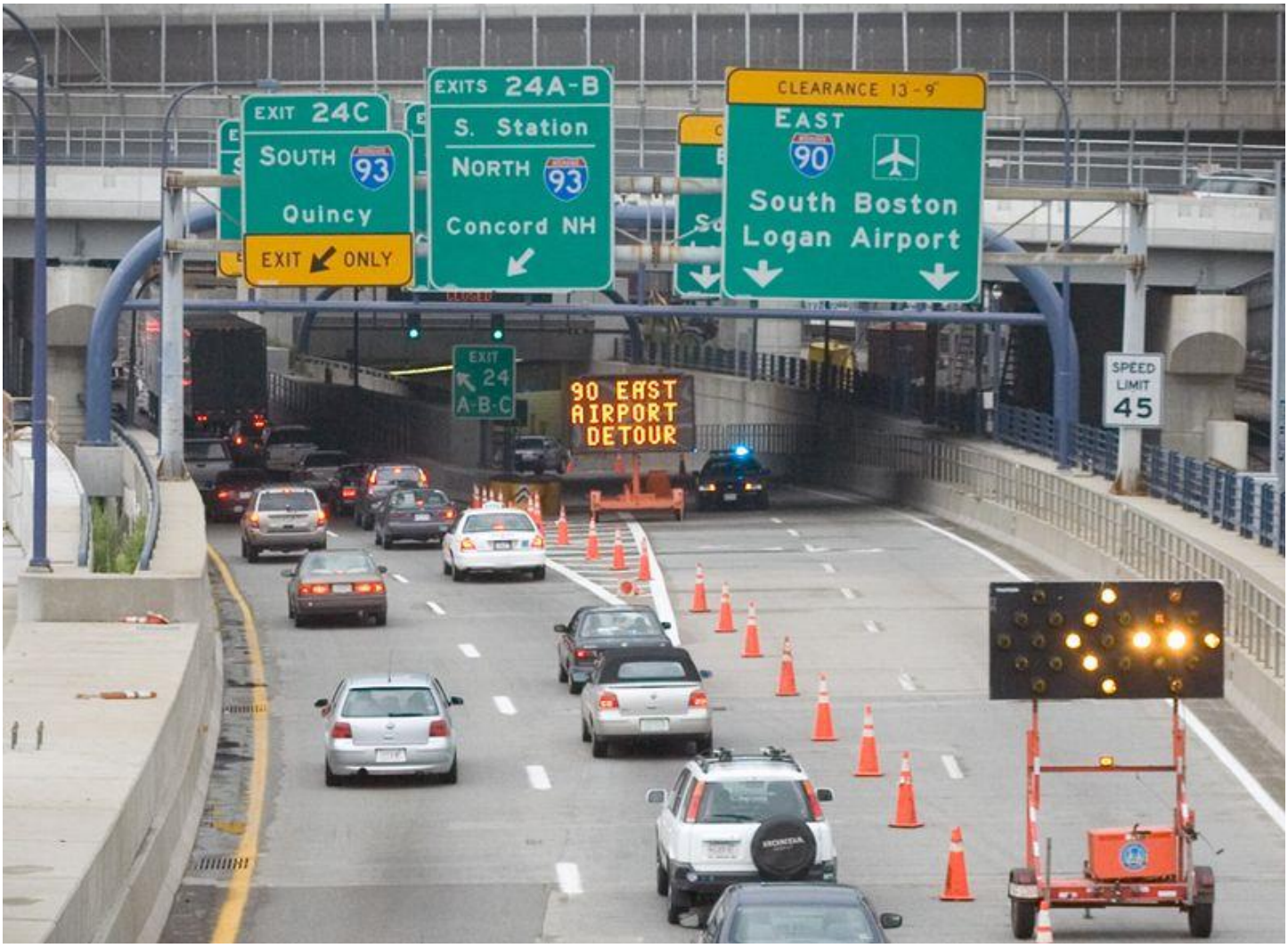


Figure 1.1 Traffic Merge at I-90

Chapter 2: Objective

To simulate a Traffic Model by considering how vehicles avoid each other.

- Dynamic management of vehicle traffic using priority queue
- Generating steady flow of vehicles around bottlenecks

Vehicles are in steady flow although at random positions. A situation arises where a lane is dropped: perhaps four lanes to three or three lanes to two. When two lanes are forced to merge, that lane will merge with its neighbor. In order to negotiate such a situation, traffic must slow down.

In order to simulate the traffic, maintain a **Priority Queue** like the one in the elastic collisions. Potential collisions that happen soon will be at top of the priority queue. When such potential collisions are avoided, they may set up other potential collisions.

Factors to consider are:

- The stopping distance for a vehicle traveling at a speed (for simplicity, we assume all vehicles are of the same type)
- The width of a vehicle (as a factor of lane width)
- The density of traffic (number of vehicles passing a point per minute)

Factors which we assume are constants:

- Road condition (dry)
- Distance to previous and next interchange (infinite)
- Lane width
- The speed limits

Other flow interrupters:

- Police vehicle
- Crash

Chapter 3: Introduction

Traffic simulation is the mathematical modeling of transportation-system, which is designed to predict the behavior and/or outcome of a real-world or physical system.

The below **Figure 3.1** shows the process of building a traffic simulation model, and the interplay between vehicle, variables, and factors.

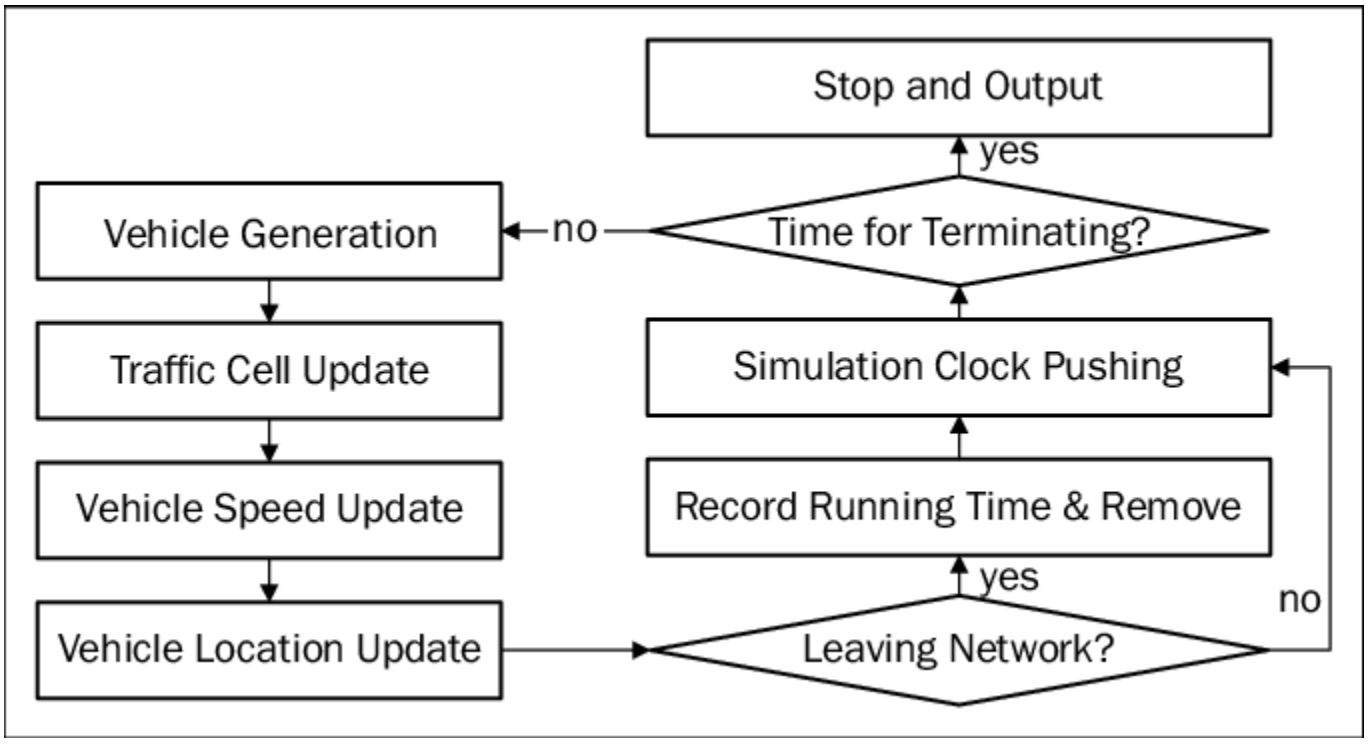


Figure 3.1 Computer Model Process

Traffic Model is a mathematical model of real-world traffic, usually, but not restricted to, road traffic. Traffic modeling draws heavily on theoretical foundations like network theory and certain theories from physics like the kinematic wave model. The interesting quantity being modeled and measured is the traffic flow, i.e. the throughput of mobile units (e.g. vehicles) per time and transportation medium capacity (e.g. road or lane width).

Models can teach researchers and engineers how to ensure an optimal flow with a minimum number of traffic congestion.

Types of Traffic Model:

1. Microscopic
2. Macroscopic

Chapter 4: Implementation

Microscopic Traffic flow Model is used, where traffic flow is assumed to depend on individual mobile units, i.e. cars or vehicle, which are explicitly determined as shown in **Figure 3.2** below:

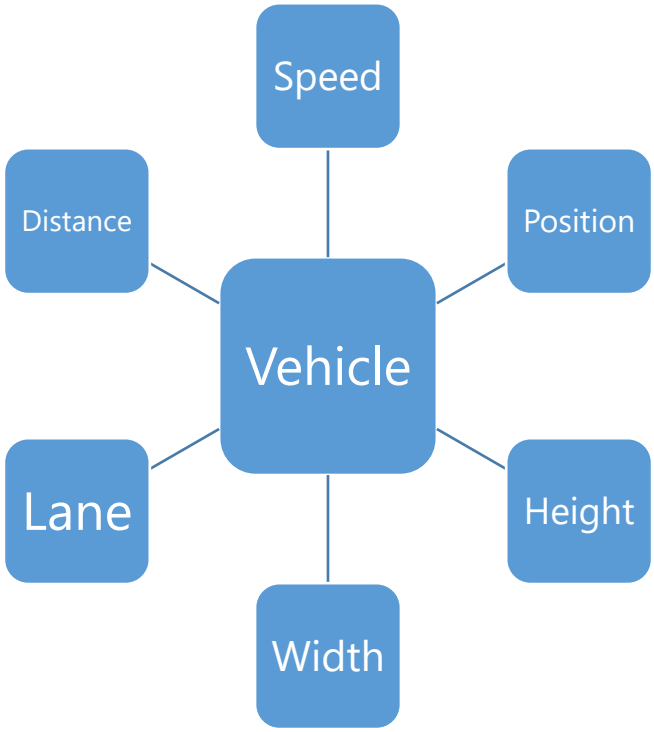


Figure 3.2 Car or Vehicle Class

The Traffic modeled in the simulation is based on the **Intelligent Driver Model (IDM).**

- The models introduced in this project are derived from assumptions about real driving behavior such as keeping a "safe distance" from the leading vehicle, driving at a desired speed, or preferring accelerations to be within a comfortable range.

- The acceleration is a strictly decreasing function of the speed. Moreover, the vehicle accelerates towards a desired ***speed v_0*** if not constrained by other vehicles or obstacles.

- The acceleration is an increasing function of the ***distance s*** to the leading vehicle.

```
private void checkSafeDistance(Car currCar, Car preCar) {
    if (currCar.getVehLocationX() + vehicleWidth < (preCar.getVehLocationX() - currCar.getStopDistance())) {
        if (currCar.getVehSpeed() == 0) {
            currCar.setVehSpeed(8);
        }
        currCar.setVehLocationX(currCar.getVehLocationX() + currCar.getVehSpeed());
    } else {
        currCar.setVehSpeed(0);
    }
}
```

- If other vehicles or obstacles are outside the interaction range of the vehicle then they do not influence the behavior of the vehicle.

- A minimum gap - bumper-to-bumper **distance d_0** to the leading vehicle is maintained (also during a standstill). However, there is no backwards movement if the gap has become smaller than **s_0** by past events. The vehicle stops for the leading vehicle to be at a distance for the gap to be greater.

- Lane changes take place if there is no vehicle in another lane and the change can be performed safely ('safety criterion').

- The safety criterion is satisfied if the new follower of the target lane after a possible change does not exceed a certain limit, this means, the safety criterion fulfills.

```
//Method to get all colliding cars
private ArrayList<Car> GetCollidingCars(ArrayList<Car> from,int rearSafeDistance) {
    ArrayList<Car> collisionCars = new ArrayList<Car>();
    for (Car car : from) {
        if (car.getVehLocationX() + vehicleWidth < rearSafeDistance) {
            collisionCars.add(car);
        }
    }
    return collisionCars;
}

//Method to check if car can merge
private void CheckPriorityQueue(ArrayList<Car> cars) {
    for (Car car : cars) {
        addToPriorityQueue(car);
    }
}

//Method to add to the priority queue
private void addToPriorityQueue(Car car) {
    if (!pQueue.contains(car) && car.getVehLocationX() > 0) {
        pQueue.add(car);
    }
    deQueueEnqueue();
}
```

Chapter 5: Simulation

The simulation instigates scenarios with stationary bottlenecks which are the work zones (lane closing) at two lanes – **Lane 3** and **Lane 2**

1. With the initial settings of the simulation, traffic breaks down near the bottleneck region as shown in **Figure 5.1** below.

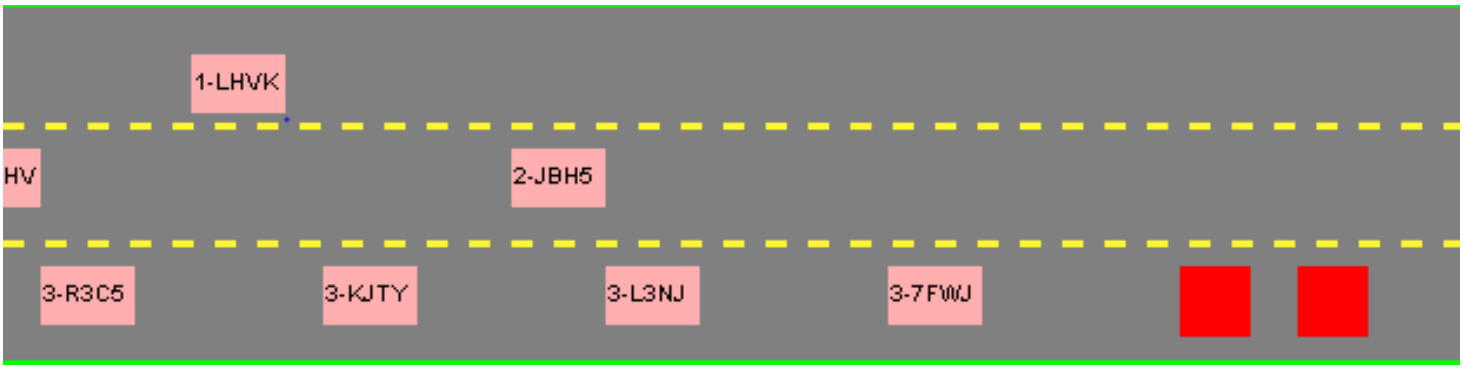


Figure 5.1 Car Approaching Bottleneck in **Lane 3**

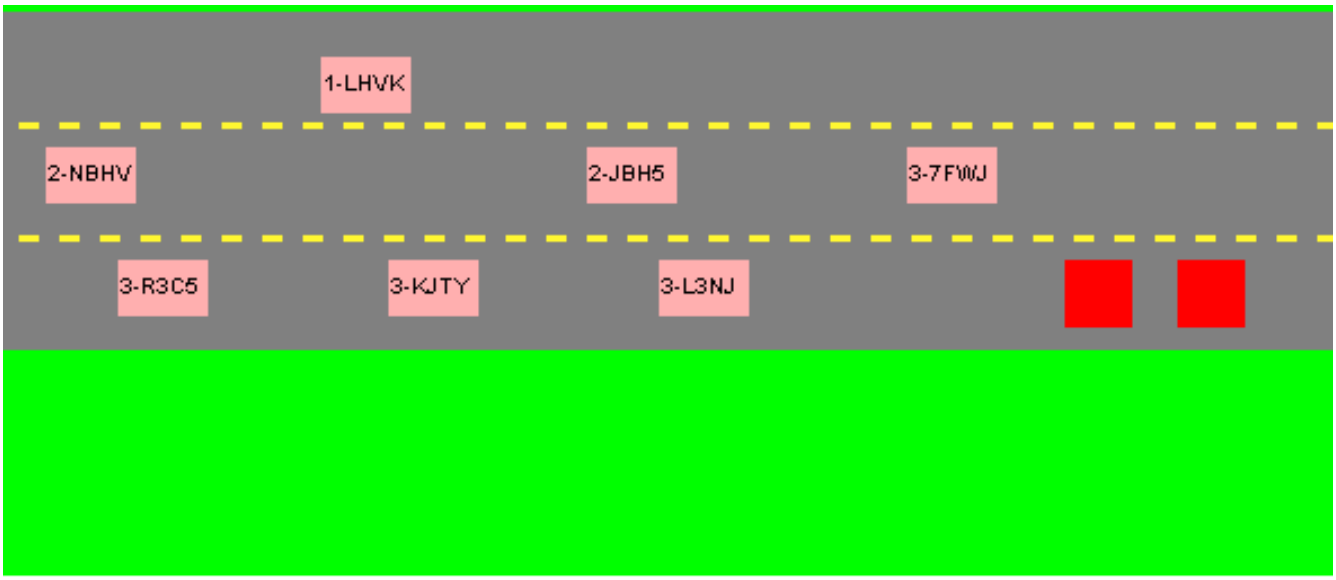
i) The car **3-7FWJ** in **Lane 3** approaches the bottleneck

ii) When the bottleneck is in the range of the vehicle's safe distance, the vehicle halts

iii) At halt, the car checks to see if it can merge into its neighbor lane i.e. **Lane 2**

2. A priority queue is set up during the **Step 1** to see all probable collisions which could occur due to merging as shown in **Figure 5.2** below.

   When the car **3-7FWJ** from **Lane 3** merges into **Lane 2,** the Priority Queue is updated with all probable collisions from both the lanes.



Priority Queue(Elements and priority)

3-L3NJ - 1

3-7FWJ - 1

3-R3C5 - 0

3-KJTY - 0

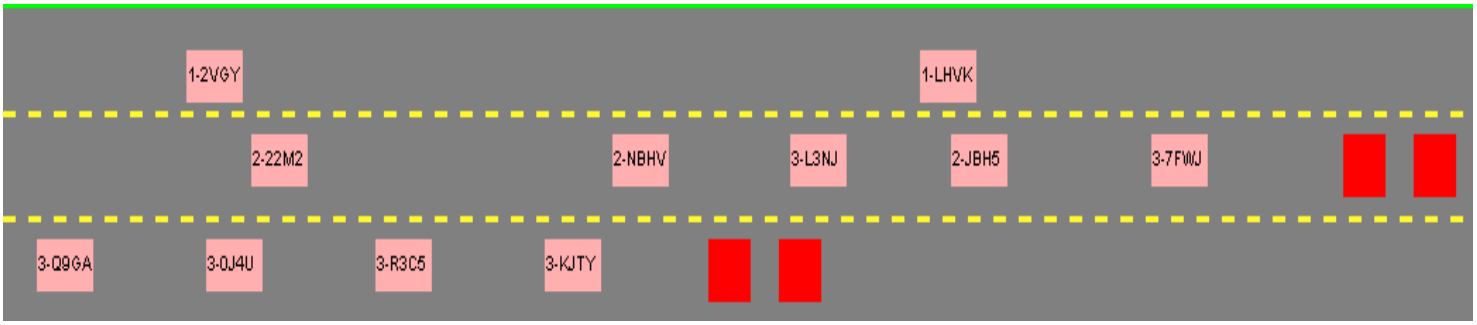
Figure 5.2 Priority Queue with car merge into **Lane 2**

i. The priority queue has the probable collisions in **Lane 3** with their priority

ii. A **1** priority are the elements which are bound to collide before the elements with priority **0**

iii. All the possible collisions in the current lane are added to the priority queue

3. Similarly, another bottleneck is placed in **Lane 2** for cars incoming from the same lane and the cars that were merged from **Lane 3** as shown in **Figure 5.3** below.



Priority Queue(Elements and priority)

3-7FWJ - 1

3-KJTY - 1

3-R3C5 - 0

3-0J4U - 0

3-L3NJ - 0

3-Q9GA - 0

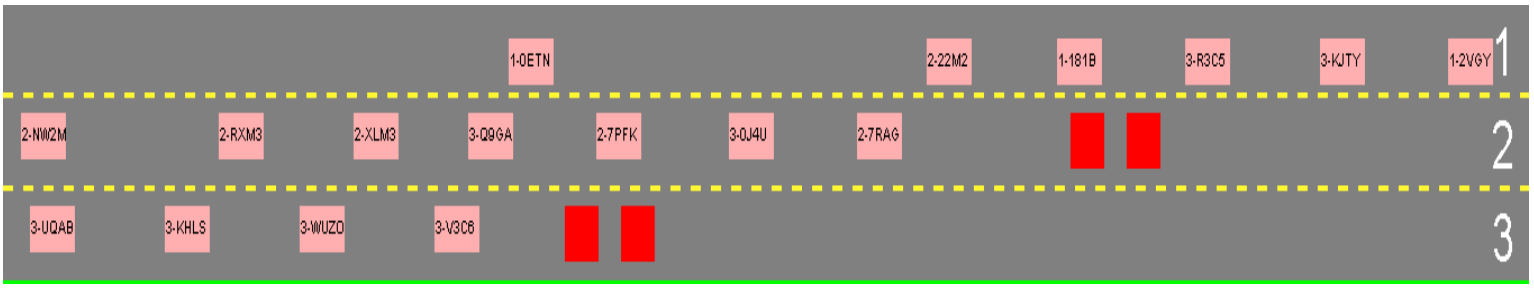
Figure 5.3 Cars Approaching Bottleneck in **Lane 2**

i. The same car **3-7FWJ** approaches bottleneck in the **Lane 2**

ii. Also, the **Lane 3** has new probable collisions set up because of bottleneck in the **Lane 3**

iii. Priority queue is updated with the elements with high priority of colliding which is **3-7FWJ** and **3-KJTY** in **Lane 2** and **Lane 3** respectively

4. Last step in the simulation is when the priority queue is updated for the cars that are merged from **Lane 2** into **Lane 1** as show in the **Figure 5.4** below.



Priority Queue(Elements and priority)

3-V3C6 - 1
3-WUZO - 0
2-7RAG - 1
3-R3C5 - 0
2-22M2 - 0
2-7PFK - 0
3-UQAB - 0
3-Q9GA - 0
3-KHLS - 0
3-0J4U - 0
2-RXM3 - 0
2-XLM3 - 0
3-KJTY - 0

Figure 5.4 Priority queue with the car merge into **Lane 1**

Chapter 6: Conclusion

Priority Queue data structure is an abstract data type that provides a way to maintain a set of elements, each with an associated value called key. It is known that a queue follows First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the Priority Queue comes into play.

Max-priority queue

- **insert**: add an element to the priority queue
- **maxElement**: return the largest element in the priority queue
- **removeMaxElement**: remove the largest element from the priority queue

Priority Heap.

- Elements in this class are in natural order or depends on the Constructor we used at this the time of construction.
- It doesn't permit null pointers.
- It doesn't allow inserting a non-comparable object, if it relies on natural ordering.

Heap Property

- There are two kinds of binary heaps: max-heaps and min-heaps. In both kinds, the values in the nodes must satisfy a **heap property**. For the remainder of this section, we will discuss max-heaps. Min-heaps are analogous.
- In a **max-heap**, nodes must satisfy the **max-heap property**: every node must be greater than or equal to its children.
- Below are some examples of valid max-heaps: all are nearly complete binary trees where the values in the nodes satisfy the max-heap property.

Operations

- **maxHeapify(i)** corrects a single violation of the heap property in a subtree with root at **i**. This procedure assumes the left and right children of the element at **i** are valid max-heaps.

- To correct a violation at index **i**, we find **largest**, the largest of **left(i)** and **right(i)**. Then we swap the elements at indices **i** and **largest (A[i]** and **A[largest])** and call **maxHeapify(largest)** until the violation no longer exists.

- Time complexity of **maxHeapify** is **O(logn)** where n is the number of elements in the heap.

- A more precise time complexity of calling **maxHeapify(i)** is **O(h)** where **h** is the height of the subtree with root at node **i**. If the number of elements in the subtree is $\boldsymbol{n_i}$the time complexity is $\boldsymbol{O \log n_i}$

- **buildMaxHeap()** produces a max-heap from an unordered array

- We observe that the leaves of the heap are always valid max-heaps. So, to build a max heap, we need to correct violations on nodes that are not leaves. (If the array that we use to represent our heap is 0-indexed, these would nodes at indices between $\frac{\{n-2\}}{\{2\}}$ and **0** inclusive).

As discussed above, like heaps we can use priority queues in scheduling of traffic. When there are **N cars** in queue, each having its own priority. If the job with maximum priority will be completed first and will be removed from the queue, we can use priority queue's operation **extract_maximum** here. If at every instant we must add a new car in the queue, we can use **insert_value** operation as it will insert the element in $\boldsymbol{O\ log\ N}$ and will also maintain the property of max heap.

Chapter 7: References

1. Martin Treiber, A. K. (2013). *Traffic Flow Dynamics*. Springer, Berlin, Heidelberg.
2. *Understanding of the Simulation Model* . (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Traffic_model