# Cycle Rental Porta: Data Structures and Algorithms Project

Vrihad Agarwal 150100004
Arpit Singh 150070059
Krish Mehta 150070002
Harsh Mundra 150070024

April 26, 2017

## 1 Problem Statement and Description

As things go virtual, so does stock management and product searching.
So, our goal here was to implement a program that can:

- Store, access and edit the database of cycles
- Search for queries of customers, based on Brand, Model, Price etc of the cycle
- Maintain a Waitlist of customers for every model

## 2 Overview of the report

The link to project program file:
`https://drive.google.com/drive/folders/0B1L8DK9UW53FVmhUMC1NR1hMa2c`
The report contains the following:

- Details of Program flow and features
- The Searching and Filtering Algorithms used
- Challenges faced
- Possible Improvisations

# 3 Details of Program Flow and Features

The program offers user 5 choices:

1. Adding a new cycle to the Database
2. Edit the parameters of any cycle in stock
3. Remove a cycle from stock (damage, ageing etc)
4. Rent a cycle
5. Return the rented cycle

The database entry addition, editing and deletion are straightforward database access and edit operations.

Renting a cycle would further involve searching of the database for the cycle the customer needs. This done by a two stage elimination:

1. Searching based on Brand, Price etc using Prefix Trie Searching
2. Filtering using simple yes/no elimination on all parameters as desired

# 4 Data Structures and Algorithms Explored

There were a lot of different data structures and algorithms required to meet the requirements of different aspects of the implementation. The following were the main implementations that needed specific Data Structures:

- Customer Waitlist - Queue Data Structure
- Brand/Model Cycle Database - Trie Data Structure
- Customers who rented a given cycle - List Data Structure
- Prefix Trie based searching for customer's query

### 4.1 Details of Prefix Trie Search

The cycles are stored in a Trie Data Structure. The strucutre is initially like a null root node, about to have 26 children corresponding to each alphabet and each of these children has the same, and it goes on. Now, when a database instance is to be created(stored) in the Trie, it starts from the first alphabet and populates the nodes of the prefix trie.

## 5 Challenges Faced

We faced a few challenges like:

- Building the Prefix trie search
- Overall flow of the program
- Error/exception handling

## 6 Possible Improvisation

We could have added some features and/or failsafes if there would've been more time:

- Admin password login provision for database access/edit rights
- Error/Exception handling code
- Trie could have been added with special character and number handling