ECS 32B – Introduction to Data Structures
Final Project Part 1

Due: December 5, 2020, 5:00pm PT

Note:

- This is a group project. Each group may have up to three people. You may post on Piazza to find teammates.
- If you do not have a group and are willing to be assigned by the teaching staff, you can put your name and email in this spreadsheet `https://docs.google.com/spreadsheets/d/1S86rbmbzW_HVyO-1cJMWZafD3vQtKCyA6CgQOYiMIA/edit#gid=0` by **Sunday 11/22, 5pm PT**. Then we will randomly form groups from the speadsheet.
- If you wish to have more than three people in your group, you could consider the Design Your Own Project option (requirements are in Page 4).

Overview

The final project is to simulate a delivery truck. Each truck starts out empty but with a map that has the addresses of the post service offices and of the possible places it might need to deliver to. The truck is supposed to pick up all packages from all of the post service offices and deliver all the pakcages (but not necessarilty in that order). The truck also knows what packages there are in each post service office.

Each package has an address and a name to which the package should be delivered. The goal is to pick up and deliver all packages. By the end of Part 2, the fastest groups will gain extra credit.

The final project is broken into two parts. Part 1 is to set up the basic data structures. The description of Part 1 is in Section . Part 2 is to find the fastest route of delivering all packages given a map and the pakcages' information. The details of Part 2 will come later.

Part 2 also has an extra credit opportunity. The groups that can finish delivering the pakcages early will earn some extra credit, depending on your ranking (not cumulative):

- Top 2: 2%
- Top 5: 1.5%
- Top 10: 1%
- Top 15: 0.5%

The exact measure of performance will come later in Part 2's description. The basic operations that would probably be calculated towards the final running time include

- picking up (or adding) a package;
- delivering a pakcage (including finding the package in the truck first);
- calculating which route to take;
- driving from one location to the next.

Part 2 is supposed to be more time consuming than Part 1 and you might also make changes in Part 1 to make Part 2 more efficient. So Part 1 is worth 8% and Part 2 is 16%. (The final project as a whole is 24%.)

Part 1

The framework for Part 1 is provided in "framework.py". It contains two classes: `Package` and `Truck`. You are to finish defining the `Truck` class.

Package

The `Package` class has only one method, the constructor. Each package is initialized with a unique identifier `id` and has four attributes:

- `id` (a string with only numeric characters) for the package's unique identifier.
- `address` (a string) for the address to which the package is supposed to be delivered.
- `office` (a string) for the address of a post service office: if a package is not collected, then the attribute refers to the office where the package is; otherwise, it refers to from which office the package is collected.
- `ownerName` (a string) for the name to which the package is supposed to be delivered, and
- `delivered` (a Boolean) for indicating if the package is delivered or not.
- `collected` (a Boolean) for indicating if the package is collected by a truck or not.

You are free to define accessors and mutators, or you may change the attributes directly. The `Package` class will not be graded.

Truck

The focus of Part 1 is to finish defining the `Truck` class. The framework contains its basic attributes and methods. You are free to add more.

In the framework, a truck has three attributes:

- `id` (a string with only numeric characters): the unique identifier of a truck.
- `size` (a non-negative integer): the size of the truck, i.e., the maximum number of packages can be stored in a truck (assuming all packages are of the same volume).
- `location` (a string): the current location of a truck.
- `packages`: the packages that have been loaded in a truck.

You get to decide the type of the `packages` attribute. It can be a Python list or any of the data structures we have covered (or will cover) in this class.

The methods you need to define include the followings.

- `__init__(self, id, n, loc)`: Each truck is initialized with a unique identifier `id`, a size `n` and a location `loc`. No matter what type you choose for the collection of packages in a truck, it should be initalized to be empty.
- `collectPackage(self, pk)`: collecta package `pk` (an instance of `Package`), i.e., add to `pk` to `self.packages`.
- `deliverPackage(self, pk)`: deliver a pckage `pk`.
- `deliverPackagesByAddress(self, addr)`: deliver all packages that are supposed to be delivered to the input address `addr`.
- `removePackage(self, pk, office)`: remove a package `pk` from the truck and put it back to a post service office `office` without delivering it.
- `driveTo(self, loc1, loc2)`: drive from location `loc1` to `loc2`
- `getPackagesIds(self)`: return a Python list of the ids of the packages in `self.packages`. This is for testing. The order of the ids does not matter.

Notice that when a truck collects a package from a post service office, the truck must be at the office, i.e., the truck's current location is the same as the office. This is similar when delivering and removing a package.

Note

- Do not change the names of the attributes and methods in either class.
- Feel free to add more methods and attributes if needed.

Part 2

(Details will come later.)

Design Your Own Project

You may also design your own project. If you wish to do so, you must talk to me first about your proposed project to make sure it is accetable. If you did your own project without talking to me, you will not be graded properly on Gradescope and hence no points.

The project must also contain two parts:

1. The first part demonstrates the usage of at least one of the linear data structures we have learned in this class, including stack, queue, deque, linked list, and hash table.
2. The second part uses either trees or graphs (we haven't learned yet).

These two parts do not have to be related in any way. They are due at the same time on the same day as the rest of the class.

In addition, you will need to write a **report**. The report should summarize what the project is, what functionalities you want to achieve and what problem(s) to solve. If the two parts of the project are not related, you will need to write two reports, one for each part.

You will also need to write your own **test cases** for both parts. You can email me early on and I will create a separate submission locker for you on Gradescope so that you can test your code.

For Part 2, you can also add additional functionalities as **extra credit** (worth 1% toward the course grade).

If your group has more than three people, you need to state in your reports(s) the extra work handled by the additional teammates.