## 1 Design

Nice! You got an interview with Chalisee Fahwajiarns, CEO of Pearbnb, the hot new geoponics startup based in the Central Valley. For each of the following scenarios, determine which data structures (doesn't have to be strictly Java) would give the best performance and what algorithms would be used. Additionally, give the worst-case runtime for any operations listed.

Each scenario could have a couple different solutions so we for each we simply list one below.

a. Chalisee says she has a list of $N$ names of crops, where each entry in the list represents an acre of farmland in the Central Valley. Find the number of acres grown for each crop.

**Data Structures**: `HashMap<String, Integer>`

**Algorithm**: Iterate through the list of names, maintaining a mapping from crop name to number of acres, incrementing at each ocurrence.

**Runtime**: $\Theta(N)$

b. Pearbnb is a trusted community marketplace for people to list, discover, and order unique produce and plants around the world. Chalisee wants to start developing auto-complete for search on Pearbnb's website. When a user types in the first $K$ characters of a query, she wants the website to say how many products have the same $K$ character prefix. Assume that no products have a name longer than $M$ and there are $N$ distinct products. Optimize for both constructing the solution and matching a query.

**Data Structures**: `Trie`

**Algorithm**: Construct a trie on the product names using character at each node. Conveniently store how many words use a prefix represented by a node at that node.

**Runtime**: $\Theta(NM)$ for construction and $\Theta(K)$ for query

c. One of the things that Pearbnb does is optimize the profits for farmers. Pearbnb uses a database of $N$ `Orders`. Each `Order` represents an order from a customer for a specific product and has the following: the customer's name, the `Date` the order was made, the `Date` requested for the delivery, the name of the product ordered, the quantity of the product ordered, and the price per unit for the product. Chalisee, a champion for Big Data, wants to run analytics on Pearbnb's database and query for `Orders` requested to be delivered within a certain range of dates for a certain product. Optimize for both constructing the solution and matching a query.

**Data Structures**: `HashMap<String, ArrayList<Order»`

**Algorithm**: For each product, make a mapping from its name to an `ArrayList<Order>` sorted by delivery date for that product. At query time, look up the appropriate list and do a binary search for the indices corresponding to the endpoints. Return a view of that range.

**Runtime**: $\Theta(N \log N)$ for construction and $\Theta(\log N)$ for query. (The worst case is all orders are for the same product)

**13. Reductions (26.5 points).** Often in computer science, problems are just other problems in disguise. Complete each problem below according to the directions given. Many of these problems are very challenging.

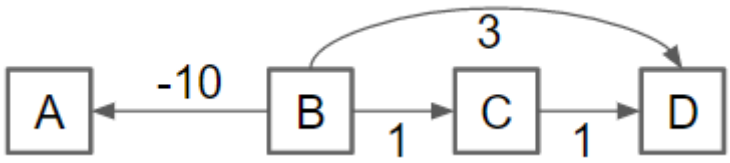
a) (3 pts) Describe an algorithm to find a **maximum** spanning tree. Your algorithm must use Kruskal's as a "black box," that is, without any modifications. Your answer should be brief.

Build a copy of the graph with every edge weight negated. Run Kruskal's.

b) (5 pts) Suppose you want to find the SPT of a graph, but where you redefine the total cost of a path as follows. Let `cost(List<Edge>)` be the sum of the weights of the edges, plus the number of edges. In other words, we want to run Dijkstra's taking into account not just the weights of the edges, but also the number of edges. Describe an algorithm to find this shortest paths tree. Your algorithm must use Dijkstra's as a "black box". Your answer should be brief.

Build a new copy of the graph with +1 added to every edge. Run Dijkstra's.

c) (5 pts) Dijkstra's algorithm sometimes fails on graphs with negative edges. Suppose we have a graph G with a single negative edge with weight -Q, and we want to find the shortest path. Suppose we construct a new graph G' where every edge has Q added to its weight. If we run Dijkstra's on G', is the resulting shortest paths tree always a correct shortest paths tree for G? If yes, explain why. If no, provide a counter-example.

| ○ Yes, because: | ● No, counter-example:<br>  <br><br>SPT with B as source is not correct if we add 10 to every edge (in the new graph G, it'll prefer the path from B to D directly, but that's not actually the shortest path in the original graph from B to D) |
|---|---|

a) https://www.geeksforgeeks.org/nth-node-from-the-end-of-a-linked-list/
(https://www.geeksforgeeks.org/nth-node-from-the-end-of-a-linked-list/)

b) Animal Shelter:
Approach 1:

- Can simply create Dog/Cat classes with arrivalTime instance variable
- 2 queues: one for dogs, one for cats
- If requesting dequeueAny, peek from each queue and select animal with oldest arrivalTime

Approach 2:
Approach if you don't want to store arrivalTime or necessarily have a Dog/Cat object.

- Have a master queue with all the animals (dogs and cats) so the last element on the master queue is always the oldest animal.
- Create 2 other queues. A dog queue with indices that refer to indices of dogs in the master queue. Similar for cats. If you want a dog or cat then pop the index from their queue, retrieve the animal from the index in the master queue, and then remove it from the master queue.

(Note: for this example I assumed cat objects were strings with the word "cat" somewhere and similarly for dogs).

```
class AnimalQueue:
        def __init__(self):
                self.master = []
                self.cat = []
                self.dog = []

        def enqueue(self, x):
                self.master.insert(0, x)
                if "cat" in x:
                        self.cat.append(len(self.master) - 1)
                else:
                        self.dog.append(len(self.master) - 1)

        def dequeue(self):
                a = self.master.pop()
                if "cat" in a:
                        self.cat.pop()
                else:
                        self.dog.pop()
                return a

        def dequeueDog(self):
                i = self.dog.pop()
                a = self.master[i]
                del self.master[i]
                return a

        def dequeueCat(self):
                i = self.cat.pop()
                a = self.master[i]
                del self.master[i]
                return a
```

c) SparseArray:
One approach: HashMap of index -> value. If index does not exist in the map, then its value is zero. Pro: minimal space, efficient. Con: don't know true size of array, very slow sequential access (e.g. if you want to do things like multiply/add two sparsearrays together like vectors)

To fast operations on multiple sparse arrays: Create a SparseArray class with instance variable `ArrayList<Entry>` where Entry has (index, value). Can then also store things like the 'true' size of the whole array as an instance var. Basically Fall 2018 Lab 10 SparseIntVector.

d) https://www.geeksforgeeks.org/median-of-stream-of-integers-running-integers/
(https://www.geeksforgeeks.org/median-of-stream-of-integers-running-integers/)