Design Underground System

题目

Implement the `UndergroundSystem` class:

- `void checkIn(int id, string stationName, int t)`
  - A customer with a card id equal to `id`, gets in the station `stationName` at time `t`.
  - A customer can only be checked into one place at a time.
- `void checkOut(int id, string stationName, int t)`
  - A customer with a card id equal to `id`, gets out from the station `stationName` at time `t`.
- `double getAverageTime(string startStation, string endStation)`
  - Returns the average time to travel between the `startStation` and the `endStation`.
  - The average time is computed from all the previous traveling from `startStation` to `endStation` that happened **directly**.
  - Call to `getAverageTime` is always valid.

You can assume all calls to `checkIn` and `checkOut` methods are consistent. If a customer gets in at time **t1** at some station, they get out at time **t2** with **t2 > t1**. All events happen in chronological order.

Structure of the Code

```
checkInData = a new HashMap (id -> startStation, checkInTime)
journeyData = a new HashMap (startStation, endStation -> total, count)
```

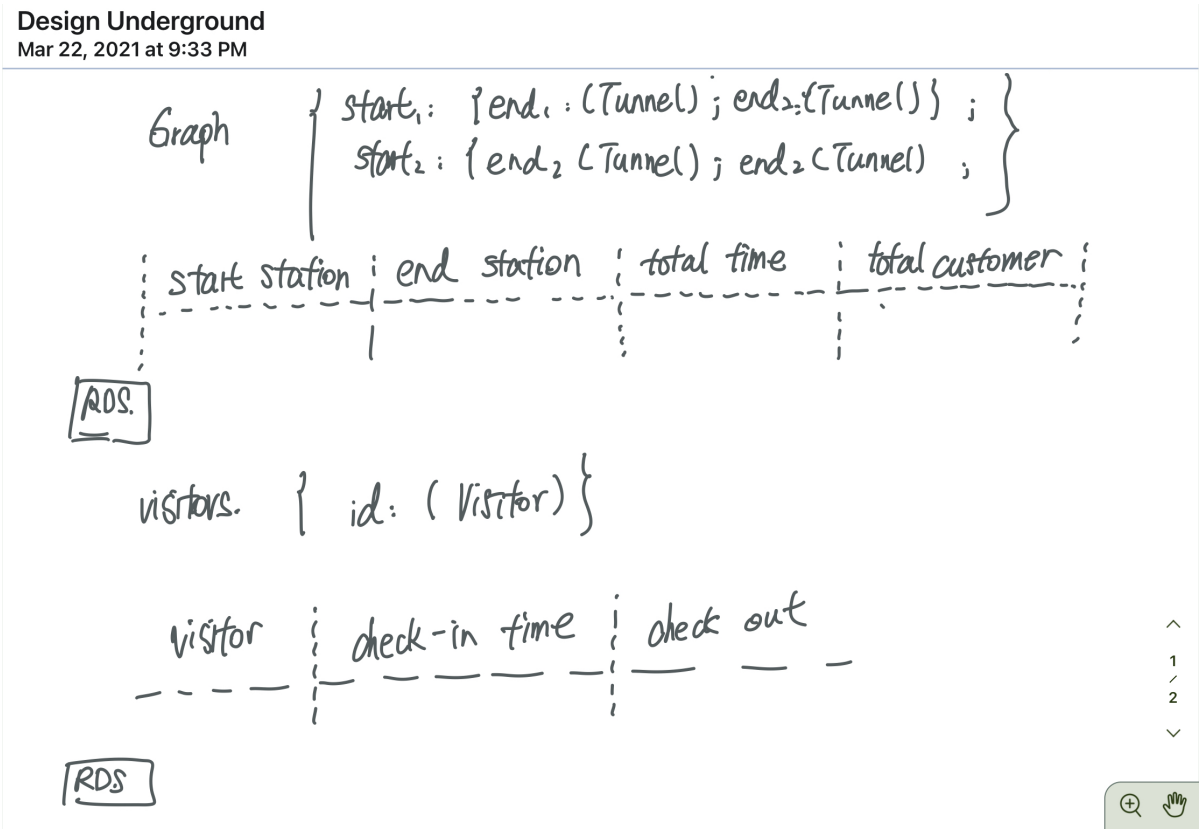
My Solution:

```
class Tunnel:
    def __init__(self):

        self.totalCustomer = 0
        self.averageTime = 0

    def update (self, duration):
        self.averageTime = (self.averageTime * self.totalCustomer + duration) / (self.totalCustomer + 1)
        self.totalCustomer += 1
```

```
class Visitor:
    def __init__(self, startTime, startStation):
        self.startTime = startTime
        self.startStation = startStation

class UndergroundSystem:

    def __init__(self):
        # graph =  { Station: {Station: Tunnel}}
        self.graph = defaultdict (lambda: defaultdict (Tunnel))

        # visitors = { id : Visitor }
        self.visitors = {}

    def checkIn(self, id: int, stationName: str, t: int) -> None:
        assert (id not in self.visitors)
        self.visitors[id] = Visitor(t, stationName)

    def checkOut(self, id: int, stationName: str, t: int) -> None:
        assert (id in self.visitors)

        # Update average Time
        duration = t - self.visitors[id].startTime
        self.graph[self.visitors[id].startStation][stationName].update (duration)

        # Send the visitor out
        del self.visitors[id]


    def getAverageTime(self, startStation: str, endStation: str) -> float:
        return self.graph[startStation][endStation].averageTime
```

Follow-Ups/ System Design

1. **存储:** it would not be realistic to store the data in volatile computer memory. In practice, computers fail (e.g. lose power) so we need to store the data in a permanent medium, such as a database.



- Vistor 比较适合RDS
- Graph我觉得AWS的Neptune可以让Query写起来更直接

2. **同步:** This is a lot of data that one computer would need to receive through its network connection! To make this work, we'd probably be using more than one computer. This introduces concurrency issues that would need to be addressed.
3. **计算:** Avoid Integer Overflow:
    1. 计算Average的时候可以total Time拆开
4. **设计理念:**You can use a tuple as a key if all of the elements contained in the tuple are immutable. (If the tuple contains mutable objects, it cannot be used as a key.) Hence a tuple to be used as a key can contain strings, numbers, and other tuples containing references to immutable objects.