Design Specification

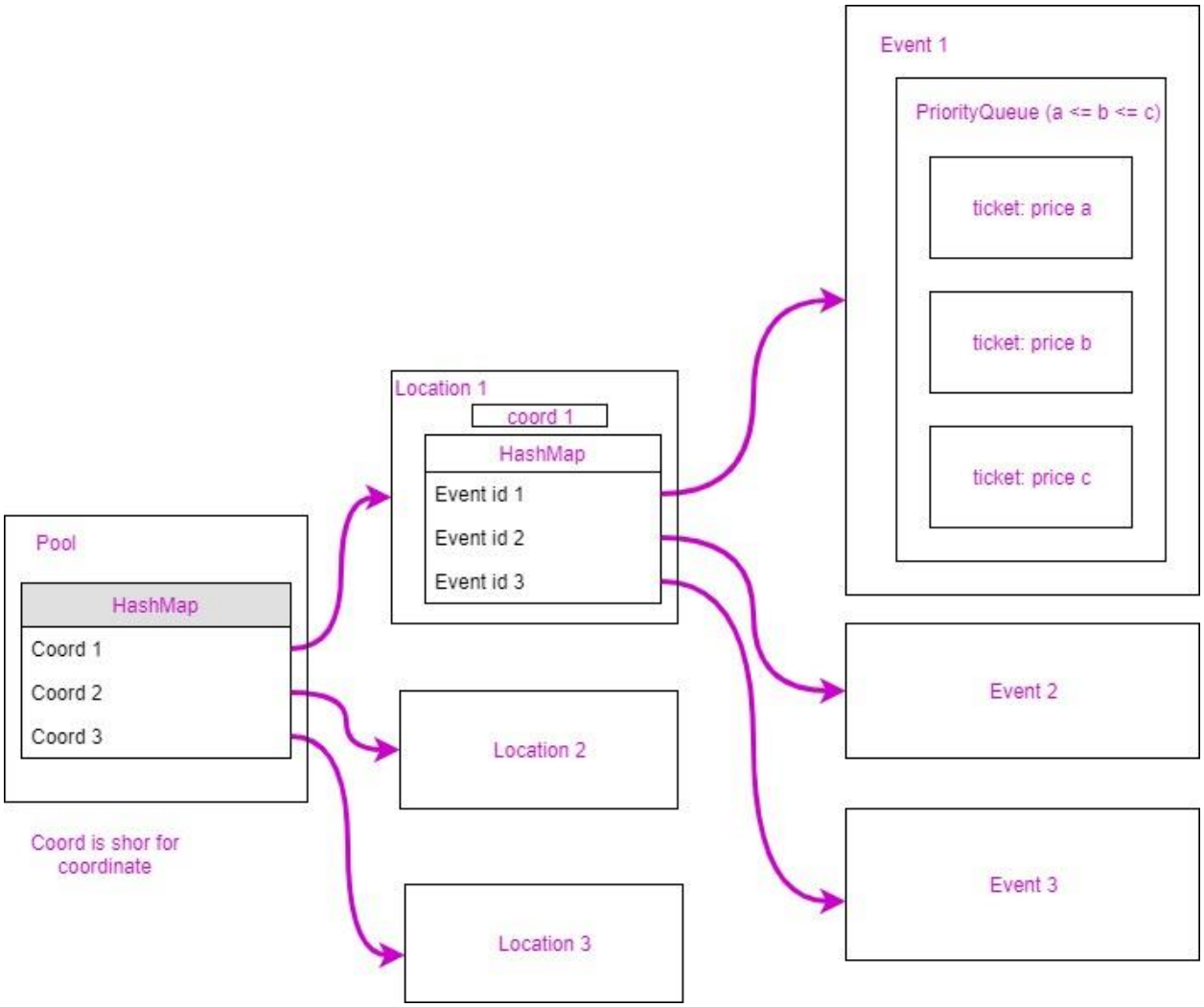
1. Assumptions

a. The max price of a ticket is $10000, you can change maxPrice in test.java.
b. If distance(Event A, user position) equals distance(Event B, user position), the Event which has lower price comes first.
c. The program supports multiple events at the same location

2. Class description

The system is divided into two parts: BasicInfo package and Query package. BasicInfo includes Ticket, Event, Coord, Location, and Pool. Query includes all the query operations to the events pool.

a. BasicInfo: the architecture is as below:



I. Ticket : abstraction of a ticket, contains the price of the ticket

II. Event: abstraction of an event, contains a priority queue for tickets, ordered by price, also contains the coordinate of the event.
III. Location: abstraction of a location, contains a hash map of event and the coordinate of the location.
IV. Pool: a container contains all the locations.
V. Coord: a class of a coordinate.

b. Query: contains a set of functions that do different queries.
I. EventQuery: a class which try to find the five closest events, along with the cheapest ticket price of each event.

Algorithm: use a PriorityQueue to keep at most 5 events, the compare rules are as below:

Sort by the distance(event a, user position) and distance(event, user position), larger one comes first. If the distances are equals, sort by the cheapest price of each event, the expensive one comes first.

```
public int compare(Event a, Event b) {
    int disA = a.getCoord().disTo(pos);
    int disB = b.getCoord().disTo(pos);
    if (disA != disB) {
        return disB - disA;
    }
    if (a.getCheapestTicket() != b.getCheapestTicket()) {
        return (a.getCheapestTicket() < b.getCheapestTicket()) ? 1 : -1;
    }
    return 0;
}
```

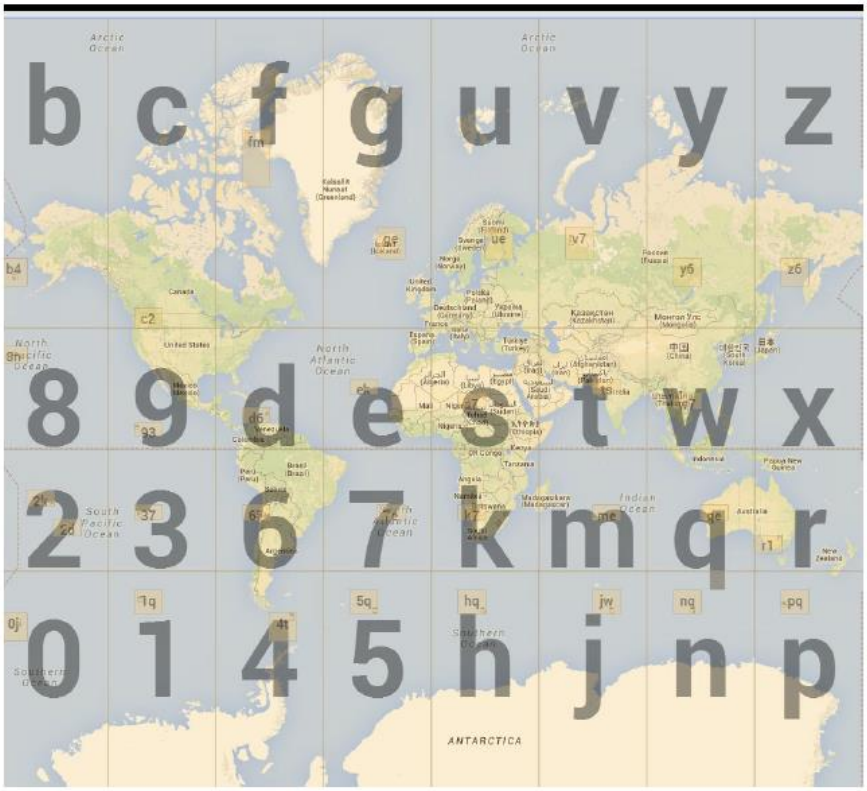
traversal every location and every event, add every event into the PriorityQueue, if the size of the PQ is larger than 5, pop out the top one.

Trick: if the distance(location A, user position) > the distance(PQ.top(), user location) and the size of the PQ is 5, we can skip the location.

Time complexity: NlogK, N is the total number of events and K is the size of the outputs. In our case K is 5.

3. FOR WORLD SIZE

We can use Goehash to store the location information, a base32 number is used to describe a specific of region in the world. The base32 is 0-9, a-z except a, i, l, o, as below:



geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	± 23	± 23	± 2500
2	5	5	± 2.8	± 5.6	±630
3	7	8	± 0.70	± 0.7	±78
4	10	10	± 0.087	± 0.18	±20
5	12	13	± 0.022	± 0.022	±2.4
6	15	15	± 0.0027	± 0.0055	±0.61
7	17	18	±0.00068	±0.00068	±0.076
8	20	20	±0.000085	±0.00017	±0.019

For example:

LinkdeIn HQ: 9q9hu3hhsjxx

Google HQ: 9q9hvu7wbq2s

For an event, we can

Encode the location of the event to the 32base number and save the information into a NoSQL database such as Redis.

For a user position, we can

1. Translate the latitude and longitude to the base32 number, for example 9q9hvu7wbq2s
2. Find the events which have prefix of 9q9hv, add them into our pool and then use EventQuery to search the closest 5 events. The longer the prefix we use, the more accuracy we can get.