Shanghai Metro DESIGN1

Decision 1 - DATA STRUCTURE
ordered container, bounded, no repetitions
the number of elements (pairs station from station to)
is known, P = N*N (order of 100*100)
N = number of stations

Decision 2 DATA STRUCTURE
key is pair stationFrom-stationTotypedef
struct trip {
char* fromStation;
char* toStation;
char* fromToKey;
int nTrips;

}TYPE;
ASSUMPTION: no two stations with the same name

Decision3 - ALGORITHM
Step1
we produce all pairs of stations and we insert them in ordered container (this has to be repeated only if new station is added to metro), once per year?
time analysis: proportional to P

Step2
each time a trip is finished call (millions of times per day)
addOneTrip (from, to)
(searches the pair from-to in the container, nTrip++)

time analysis: proportional to P/2 (circa 50000)
(if binarySearch then would be ln2(P))
Step3 call function – once per day
computeMostFrequentTrip()
Time analysis: proportional to P
ASSUMPTION: only one max, no ties
SPACE analysis: container with P elements

DESIGN2
Store all trips

Data structure: container , unbounded, one single trip is an element (if 3 trips from A to B, then 3 elements in the container), overall millions of trips = T

Step 1
Each time there is a new trip, addTrip()
Space: if 30 bytes for one element, T*30bytes

Step2
Go through all trips, find trips with same pair from to, count them
At same time store current max
Time analysis: P * T

Overall (MUCH) worse than design1, both in time and space

DESIGN3
Assumption: trips are not evenly distributed, but follow Pareto law
(so for instance 80% of trips happens between 20% of pairs of stations)
Then design1, but key is nTrips

Decision 1 - DATA STRUCTURE

ordered container, bounded, yes repetitions
the number of elements (pairs station from station to)
is known, P = N*N (order of 100*100)
N = number of stations

Decision 2 DATA STRUCTURE

key is nTrips

```
struct trip {
    char* fromStation;
    char* toStation;
    char* fromToKey;
    int nTrips;

}TYPE;
```

Decision3 - ALGORITHM
Step1
we produce all pairs of stations and we insert them in ordered container (this has to be repeated only if new station is added to metro), once per year?
time analysis: proportional to P

Step2
each time a trip is finished call (millions of times per day)
addOneTrip (from, to)
(searches the pair from-to in the container, nTrip++
Also changes the position of the pair using nTrip as ordering element
)

time analysis:
addOneTrip is called M times
   search(Pair) – x
    average = 0.8* P *.1   +   0.2* P *.9 = P(.08+.18)
   (actual analysis decide if better than design 1 requires to measure the parameters in the search)

   reorderAllPairs
    not always, when happens exchange lements, constant

Design 3-1
Compute all pairs upfront and insert them
 They all have nTrip = 0,
  There will be more reordering

Design 3-2
No computing pairs upfront, insert new pair the first time it is encountered
  Likely the more frequent trips happen more and stay on top, there will be average reordering

Design3-3
Use order of pairs from previous day, insert them upfront
  There will be less reordering