DSA HW2 Analysis (Programming Section)

kelvin

Overview

In this assignment, you are asked to process a series of plate-number records on highway cameras, where at most two digits are possibly wrongly identified. You are to support two kinds of queries, **possible** and **anomaly**.

This is a brief sketch of possible approaches and some implementation detail. If you manage to capture most of the important stuffs, the given time limit should be fairly loose.

Anomaly Queries

The anomaly query is actually the easier one of two, because you do not need to consider the existence of wrongly identified digits.

It thus became fairly obvious that we should categorize the records by plate-number and sort them in chronological order. For a particular plate, we can easily discover anomaly records by scanning the records corresponding to the plate-number from earliest to latest. After each record, the vehicle is expected to be between two specific camera. Thus the (at most) two possible next occurence of the vehicle is known, and an anomaly is discovered if the next record is not one of the possibilities.

For handling this part of queries, a vector corresponding to each plate is enough.

Possible Queries

The Obvious Way

Note that the approach we used in processing anomaly queries (categorizing by plate) no longer work, as there is multiple (roughly $\binom{6}{2} \times 25^2 \approx 2000$) possible identified version for a particular plate. Thus one of the trivial approach is to scan through all records, and test one-by-one whether each record could be corresponding to the queried plate, i.e. differs by at most two digits.

This approach is however slow (taking $O(6\#record)$ per query), and will time out in larger test cases.

Trading Space for Time Efficiency

A way to go is to observe that despite the many possibilities of how a plate is identified, it is possible to express them concisely in with the wildcard character. Namely, the set of possible match of a plate "$p =$ "ABCDEF"" could be expressed with $S = WC(p) =$ "ABCD$**$"|"ABC$*$E$*$"|"ABC$**$F"|$\ldots$|"$**$CDEF". Note that the originally more than 2000 possibilities is cut down to just $\binom{6}{2} = 15$.

This hints us to instead categorize plates by their **matching wildcard**. In particular, we keep a vector for all possible representation with exactly two wildcards, and each record is appended to it's corresponding 15 wildcard vectors.

When processing a query asking for a particular plate p, we gather those records in vectors corresponding to $WC(p)$, and merge them in chronological order. When the number of relevant record (corresponding to a particular plate) is relatively sparse (in comparison to the whole list of records), this method is tremendously more efficient.

Implementation Details

STL Map is a perfectly suitable storage structure to serve the purpose of this problem (as is a hash map or a balanced tree; some more preprocessing to sort all possible plate occurences work as well. The key here is to be able to retreive the vector storing vectors corresponding to a specific plate in $O(\lg N)$ time).

Specifically, let `map<string,vector<Record>> rec` be the aforementioned map. An example way of retreiving or appending records into one of the vector is then

```
rec["DSA*1*"].push_back(Record("DSA014",time,camera))
```

Some other tricks (STL functionality) are helpful in this homework as well. For example, if we want to generate all possible “mask” (wildcard plate) for a particular plate p, we can simply make use of `next_permutation`:

```
char mask[] = "....**";
do {
    string str = p;
    for(int i=0; i<6; i++)
        if(mask[i]=='*') str[i] = '*';
    /* do something to the wildcard plate str */
} while(next_permutation(mask,mask+6));
```

When searching records from a given start date till an end date, the STL function `lower_bound` may be of help. For removeing duplicated records from a sorted array (you may need to do this when handling the possible query, where you may have a same record mapping to multiple wildcard plate), STL function `unique` is useful:

```
sort(arr.begin(),arr.end()); // arr is sorted now
arr.resize(unique(arr.begin(),arr.end())-arr.begin());
```

There, one line and you’re done. So sometimes making use of existing libraries could really save your time.

There are even more things that you can do to improve your efficiency, though it might not be worth your effort. For example for merging all the records from different cameras in chronological order, it could be done in $O(N \lg k)$, where N is the total number of records and k is the number of cameras. A way to do this is through *divide and conquer* (on camera), another way is to utilize the *heap* data-structure that you just learnt in class. Since you can still always achieve the same in $O(N \lg N)$ by a simple sort, it may not be meaningful to go through all the hassle. It’s still fun to think about different ways to improve things though.