Final Project Report

隊伍: 舒潔三層抽取式衛生紙90抽

組員: B03902001 駱定暄, B03902039 施秉志, B03902107 鄭格承

Data Structure

1. Index: **STL set**, Transaction History: **STL vector**

   The reasons we decided to use **STL set** for Index are:
   (1) I had used STL set in my hw2 & hw6 before I started, so it is relatively familiar.

   (2) The elements in **STL set** are all unique, which is highly according to the account system. Besides, it will sort the elements automatically, which brings the efficiency when it goes to output something such as results of **find** or recommended IDs.

   The reason we decided to use **STL vector** for Transaction History is:
   The sequence of **STL vector** by pushing_back is what we need.

2. Index: **Trie**, Transaction History: **STL vector**

3. Index: **Unordered_Map**, Transaction History: **STL vector**

As for Account Information, we designed a class : **Customer**, to contain it. The private part inside are **password**, **money** and **history**; and the public part are **ID** and some function that have to access private part, such like **authentication**, **deposit**, **withdraw**, **merge**, **search** and so on.

Algorithm (implement)

First, we outline a concept here about how we divided the project into several parts. The **main** function is designed only to read in the command**,** and then distribute the work to some function called **processXXX**.

Here are the functions(which according to the commands), and its **time complexity.**

* N: number of accounts(IDs), H: number of History, s: length of string(1~100)

- login: calls set.find(), O(logN).
- create: calls set.find(), O(logN); conditionally calls **listing function**.
- delete: calls set.find(), O(logN).
- merge: calls set.find() twice, O(2logN); merging two history, $O(H_1+H_2)$.
- deposit: nothing special, O(1).
- withdraw: checking money, O(1).
- transfer: calls set.find(), O(logN); conditionally calls **listing function**.
- find: processes the finding string(ex. *x?0E), O(s);
  check all strings(IDs), O(N * s).
- search: go through the History, O(H).
- **listing function**:

  1.transfer: find IDs which exist: O(N).

  2.create: find IDs which don't exist: very tricky…

We classified the case by modifying length or not:

(1) shorten length (2) same length (3) extend length

and we use some recursion function to determine how scores are distributed.

(see in next page)

```
void extend(int cnt, int n_var, set<string> &listset2, string ID, int pos){
    if(cnt == 0){
        vector<int> needChange;
        pick_or_not(n_var, 1, needChange, listset2, ID, pos);
        return;
    }
    for(int i = 0; i < 62; i++){
        string ex_string = ID + strtable[i];
        extend(cnt - 1, n_var, listset2, ex_string, pos);
    }
}
```

extend is how we determine the length that the string will extend.

```
void pick_or_not(int n, int h, vector<int> NC, set<string> &listset2, string sample, int pos){
    if(n < h && n != 0){

        return;
    }else if(n == 0){
            //cout << "yeah!!!" << endl;
        all_digit_change(1, NC, listset2, sample, pos, sample);
        return;
    }
    pick_or_not(n, h+1, NC, listset2, sample, pos); // not pick
    NC.push_back(h);
    pick_or_not(n-h, h+1, NC, listset2, sample, pos); // pick
    //cout << "fuck yeah!!!" << endl;
}
```

pick_or_not is how we determine the position that the character will change.

```
void all_digit_change(int h, vector<int> &NC, set<string> &listset2, string &sample, int pos, const string origin){
    if(h > (int)NC.size()){
        listset2.insert(sample);
        return;
    }
    for(int i = 0; i < 62; i++){
        if(strtable[i] == origin[pos-NC[h-1]])
            continue;
        sample[pos-NC[h-1]] = strtable[i];
        all_digit_change(h+1, NC, listset2, sample, pos, origin);
    }
}
```

all_digit_change is how we determine the changing part of **pick_or_not**.

Test

How to compile your code and use the system

how to compile your code and use the system