Report for Project 3

Author 王子腾

Date: 2019-12-25

Chapter 1: Introduction

1.1 Problem Description

> There are many cars entering and leaving the campuses everyday, and the cars' **tracks** are recorded by gates with a special form:
>
> > **plate_number hh:mm:ss status**
>
> With the available data collected within one day, we should distinguish different cars , answer the current **number** of cars according to the queries time, and tell the car that stayed for longest time **plate number** as well as the **time** they parked. We are expected to come up with sorting and mapping algorithms to process the data.

1.2 Our Tasks

To figure out the problem, we need to do these tasks:

> - Collect the input data and **store** them;
> - Calculate the current **number** of cars and output it according to the given query time;
> - Use **sorting** method to find the car(s) that parked for the longest time period, output the **plate(s)** and **time** in increasing order.

1.3 Our Thinking

The problem asks us to store the data and do the sorting to get the answer.

What needs to be done:

> 1. Use struct-array to store the input information.
> 2. Sort the array according to the **dictionary order** of cars' plates and initialise the car information.
> 3. Sort the array according to the **time order**, then update the the real-time car number information.
> 4. Output the results.

To simplify our program, we use ***qsort*** function defined in <stdlib.h> to make the array in order, which askes us to implement ***cmp*** and ***cmp2*** to do the *compare* tasks. And to the convience of finding the parking time, we use hash-table to distinguish and match the plate to the unique element.

Chapter 2: Algorithm Specification

2.1 Storing Input

We defined a struct and using arrays to store the cars information as well as the data records.

Node Structure

```c
struct Node {
    int t, in;
    char name[10];
}
```

We use this structure to store the input sequences and the cars information, time ***t*** is transfered into seconds for convience;

2.2 Sorting Node-Array

We use ***qsort*** with cmp1 to sort node array by dictionary order, then judge if the input is valid according to the adjacent status for one car. Initialise the car information and using *hash table* to save the time cost.

Pseudo-code

```
qsort(node_array, dic_order);
for(N-1 times){
    intialise car_array;
    map[hash(plate)] += time_cost;
    maxtime = max(maxtime,map[hash(plate)]);
```

Complete C Code

```c
 qsort(node, N, sizeof(struct Node), cmp);
 for (int i = 0; i < N - 1; i++) {
    if (strcmp(node[i].name, node[i + 1].name) == 0 && node[i].in &&
(!node[i + 1].in)) {
        car[carCnt++] = node[i];
        car[carCnt++] = node[i + 1];
        if (map[hash(node[i].name)] == 0) {
            strcpy(nameList[nameCnt++], node[i].name);
        }
        map[hash(node[i].name)] += (node[i + 1].t - node[i].t);
        maxTime = maxTime < map[hash(node[i].name)] ?
map[hash(node[i].name)] : maxTime;//update maxTime
    }
}
```

2.3 Sorting Car-Array

With the initialised car information, we are able to do the sorting work by **time cost** so that we can easily find the car(s) that parked for longest time period. According to the problem, we also need to figure out the **time-marked number** of cars, so we need to update the number of car.

Pseudo-code

```
qsort(car_array, increasing order);
initialise stay[0];
for(i:1 to carCnt-1)
    stay[i] = stay[i-1] + status_i;
```

Complete C Code

```
qsort(car, carCnt, sizeof(struct Node), cmp2);
stay[0] = car[0].in;
for (int i = 1; i < carCnt; i++) {
    stay[i] = stay[i - 1] + (car[i].in ? 1 : -1);
}
```

2.4 Output Results

Having processed the data, we still need to respond to the query. One easy method is to transfer the query time to seconds, compare it with the recorded time and then output the **number** of cars during query period. At last, we select the car(s) that parked as long as ***maxtime***, output its(their) **plate number(s)**, and finally print ***maxtime*** on screen.

Pseudo-code

```
for(i:0 ~ K-1){
    Input(query)
    for(j:0 ~ carCnt){
        if(car_time > query_time){
            Output(num_of_cars);
            break;}
        else if(reach End)
            Output(num_at_end);
for(i:0 ~ nameCnt)
    if(match)   Output(plate);
Output(maxtime);
```

Complete C Code

```
for (int i = 0; i < K; i++) {
    scanf("%d:%d:%d", &hh, &mm, &ss);
    tmpTime = hh * 3600 + mm * 60 + ss;
    for (; j < carCnt; ++j)//increasing query order
    {
        if (car[j].t > tmpTime) {
            printf("%d\n", j ? stay[j - 1] : 0);
            break;
        } else if (j == carCnt - 1) {
            printf("%d\n", stay[j]);
        }
    }
}
```

Chapter 3: Test Result

3.1 Test Case Generate

Besides PTA's test case, we generate 4 test cases for some of cases. For convenience, we also provide an *generator program* in folder for you to test the program.

3.2 Test Cases

Test case 1

Test 1 handles the **normal** case with 20 input and 10 queries.

Test case 2

Test 2 handles the case where there is only **one** car going in and out the campus repeatedly.

Test case 3

Test 3 handles the case where there is only one **valid** car. We give 20 records and among these records there are 19 distinct car plates. Only one car entered and left the campus.

Test case 4

Test 4 handles the case with **large** number. We give 10000 records and 80000 queries.

3.3 Status: Pass

Chapter 4: Analysis and Comments

4.1 Complexity Analysis

###4.1.1 Space Complexity
In hash, we choose 122777 as the key. It is an experiential number selected from prime numbers. This key is neither too large nor that easy to collide. As a result, we declare an array map[hashKey], for which the space complexity is O(hashKey). For nameList, stay, node and car, they all use O(N) space. Other variables are all O(1) and our program consumes no extra space. As a result, the time complexity for the whole program is **O(hashKey+N)**.

###4.1.2 Time Complexity
Function cmp, cmp2 and hash all take O(1) time. To specify this, note that the string is all 7 characters and the mapping function is simple. Suppose we have a plate number with 3 letters and 4 digits, then we need to do (int)log122777(36^{3*10}4)=1 module operation and 7 multiplications, so it runs quite fast. Print function has O(N) time complexity.
In Process function, the record reading part takes O(N) time and the query reading part takes O(K) time. Function qsort is a built-in function in standard lib, which takes O(NlogN) time to sort the node array. In worst case, every car is valid so it takes O(N) time to build map array and to find maxTime. Since there are N records, the number of valid cars which is represented by carCnt will not exceed N, so it takes O(N) time to build the stay array to mark car stayed in campus when a car enters.
To sum up, the whole function takes O(N+NlogN+N+NlogN+N+K) = **O(NlogN+K)** time.

4.2 Comments

The first qsort can either distinguish invalid cars or sort valid cars in time order. It puts records of the same car together regardless time. Then we could build the car array which actually matters. In car array, adjacent elements start with ‘in’ represents a car entering and leaving the campus. We record the time a car stayed in the campus to find the longest time.
The second qsort sorts the car array in time order regardless which car it is. As a result, we can get the number of cars staying in the campus when a car enters or leaves the campus. Since car array is in time order, when we handle queries, we only need to determine which interval the

query is in and then we know how many cars were in the campus.
The map array is used to store the time one car staying in the campus. However, the identification of cars cannot be the index of an array, so we use hash table. The mapping function is simple and works well with PTA test data, that is, it needs no collision handling, thanks to the key value. However, complex car plates may cause collision and require collision handling, because we cannot always select a larger prime number (Note that 36^7 will exceed 10^10).

Further Optimization

Int hashVal[hashKey];
Use another int array to store the hash value when record the time a car staying. Another big array may be expensive as to space complexity, but it avoids invoking hash function repeatedly. To be more specific, if there are X valid cars, it could save 3X hash invoking.

Chapter 5: Working Arrangements

Programming: 施楚宁

Testing program(main function): 施楚宁

Testing data and plot: 孙恺元

Algorithm analysis: 孙恺元

Documentation: 王子腾

Appendix A: Source Code in C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXN 10010//max n range
#define hashKey 122777//prime number for hash table
int N, K, carCnt, nameCnt;//N, K, car valid in and out count, Car plate name count
int map[hashKey];//map to store car stay stay
char nameList[MAXN][10];//save valid car plate name

int maxTime, stay[MAXN];//longest stay stay, stay car count for each car in chronological
struct Node {
    int t, in;//time, in or out
    char name[10];//plate name
} node[MAXN], car[MAXN];//original input data, valid car data

int cmp(const void *a, const void *b) {//compare function for original input
    struct Node *aa = (struct Node *) a;
    struct Node *bb = (struct Node *) b;
    if (strcmp(aa->name, bb->name) != 0) {
        return strcmp(aa->name, bb->name);//Firstly, compare in Lexicographic order
    } else {
        return aa->t - bb->t;//Secondly, compare in chronological order
    }
}

int cmp2(const void *a, const void *b) {//compare function for original input
    struct Node *aa = (struct Node *) a;
    struct Node *bb = (struct Node *) b;
    return aa->t - bb->t;//compare in chronological order
}

int hash(const char *name) {//simple hash table without collision handle
    int res = 0;
    for (int i = 0; i < 7; ++i) {
        res = (res * 36 + (name[i] > 'A' ? (name[i] - 'A' + 10) : (name[i] - '0'))) % hashKey;
    }
    return res % hashKey;
}

void Process() {
    int hh, mm, ss, tmpTime, j;//temporary hour,minute,second,time count; counter j
    char tmpIn[5];//temporary in or out save
    scanf("%d%d", &N, &K);
    for (int i = 0; i < N; i++) {
        scanf("%s %d:%d:%d %s", node[i].name, &hh, &mm, &ss, tmpIn);
        node[i].in = (tmpIn[0] == 'i') ? 1 : 0;//judge car is in or out by first letter
        node[i].t = hh * 3600 + mm * 60 + ss;// calculate time
    }
    qsort(node, N, sizeof(struct Node), cmp);// sort with cmp function
    for (int i = 0; i < N - 1; i++) {
        if (strcmp(node[i].name, node[i + 1].name) == 0 && node[i].in && (!node[i + 1].in)) {
            //if a car in sorted list appear in 'in' and 'out' order mean its a valid
            car[carCnt++] = node[i];
            car[carCnt++] = node[i + 1];
            if (map[hash(node[i].name)] == 0) {
                strcpy(nameList[nameCnt++], node[i].name);//save plate name to list
            }
            map[hash(node[i].name)] += (node[i + 1].t - node[i].t);//add car stay time count
            maxTime = maxTime < map[hash(node[i].name)] ? map[hash(node[i].name)] : maxTime;//update maxTime
        }
    }
    qsort(car, carCnt, sizeof(struct Node), cmp2);// sort with cmp2 function
    stay[0] = car[0].in;// reset stay initial condition
    for (int i = 1; i < carCnt; i++) {
        stay[i] = stay[i - 1] + (car[i].in ? 1 : -1);//simplified prefix sum
    }
    j = 0;// reset counter j
    for (int i = 0; i < K; i++) {
        scanf("%d:%d:%d", &hh, &mm, &ss);
        tmpTime = hh * 3600 + mm * 60 + ss;// calculate the time
        for (; j < carCnt; ++j) {//because K input is in chronological order, stay list only need to traverse one time
            if (car[j].t > tmpTime) {// find the time point match input time
                printf("%d\n", j ? stay[j - 1] : 0);
                break;
            } else if (j == carCnt - 1) {//special judgement for the last one
                printf("%d\n", stay[j]);
            }
        }
    }
}

void Print() {
    for (int i = 0; i < nameCnt; i++) {// traverse name list
        if (map[hash(nameList[i])] == maxTime) {
            printf("%s ", nameList[i]);// print plate name satisfying max time
        }
    }
    printf("%02d:%02d:%02d", maxTime / 3600, (maxTime % 3600) / 60, maxTime % 60);//print the max time
}

int main() {
    Process();//input and process
    Print();//print max stay time car info
    return 0;
}
```

Declaration

We hereby declare that all the work done in this project titled "Project 3" is of our independent effort as a group.