Fundamentals of Data Structures

Laboratory Projects Report

Project 3: Cars on Campus

Group 07

Programmer: Hao Xiangpeng

Tester: Yang Kefan

Report Writer: Li Haipeng

Date: 2016-12-24

Content

[Chapter 1: Introduction 2](#_Toc470356200)

[1.1 Background 2](#_Toc470356201)

[1.2 Problem description 2](#_Toc470356202)

[1.3 Sample problem&solution 2](#_Toc470356203)

[Chapter 2: Algorithm Specification 4](#_Toc470356204)

[2.1 Flow Chart 4](#_Toc470356205)

[2.2 Correctness of Algorithm 4](#_Toc470356206)

[2.3 Data Structure 5](#_Toc470356207)

[2.4 Key Algorithm 5](#_Toc470356208)

[Chapter 3: Testing Results 7](#_Toc470356209)

[Chapter 4: Analysis and Comments 13](#_Toc470356210)

[Appendix: Source Code 15](#_Toc470356211)

[Declaration 20](#_Toc470356212)

# Chapter 1: Introduction

## 1.1 Background

Zhejiang University has 8 campuses and a lot of gates. From each gate we can collect the in/out times and the plate numbers of the cars crossing the gate.

## 1.2 Problem description

**The information we have:**

* The record of cars crossing the gate
  + **Plate number:** a string of 7 English capital letters or 1-digit numbers
  + **Time when it get in:** represents the time point in a day by “hour: minute: second”
  + **Time when it get out:** represents the time point in a day by “hour: minute: second”

**The purpose we want to achieve:**

* **Task 1:** Given a certain time, the number of cars parking on campus
* **Task 2:** At the end of the day, the cars that have parked for the longest time period.

## 1.3 Sample problem&solution

**Input:**

16 7

JH007BD 18:00:01 in

ZD00001 11:30:08 out

DB8888A 13:00:00 out

ZA3Q625 23:59:50 out

ZA133CH 10:23:00 in

ZD00001 04:09:59 in

JH007BD 05:09:59 in

ZA3Q625 11:42:01 out

JH007BD 05:10:33 in

ZA3Q625 06:30:50 in

JH007BD 12:23:42 out

ZA3Q625 23:55:00 in

JH007BD 12:24:23 out

ZA133CH 17:11:22 out

JH007BD 18:07:01 out

DB8888A 06:30:50 in

05:10:00

06:30:50

11:00:00

12:23:42

14:00:00

18:00:00

23:59:00

**Output:**

1

4

5

2

1

0

1

JH007BD ZD00001 07:20:09

# Chapter 2: Algorithm Specification

## 2.1 Flow Chart

Information Input

Sort all the records by in-time using quick sort

Eliminate the invalid items (including the records that are not paired)

Do the query for the given time and output\*

Sort the records by plate number and merge the same car’s parking time

Sort the records by total parking time

Output\*\* car(s) that have parked for the longest time period

\* : Task 1

\*\*: Task 2

## 2.2 Correctness of Algorithm

* We use function block “Sort all the records by in-time using quick sort” to make query easier.
* In function block “Eliminate the invalid items”, all invalid records are deleted so that we get the right result.
* When we do the query, we pick the records of which the in-time is earlier than the given time and the out-time is later than the given time. So we can find all the parking cars
* In some cases, a car enters and leaves the campus more than one time, so we have to merge them to get the total time using “Sort the records by plate number and merge the same car’s parking time” function block.
* At last, we Sort the records by total parking time and output the cars that parking for longest time. Mission completed.

## 2.3 Data Structure

* Records of cars

|  |
| --- |
| *struct* OneItem { |
| *long* *long* plate\_number; |
| *int* time; |
| *int* status; |
| *int* valid; |
| *int* stay\_time; |
| }; |

* + plate\_number：we convert the string to long long type so that we can easily sort the data
  + time: equal to “hour \* 3600 + minute \* 60 + second”
  + status : 1 represents “in” while -1 represents “out”
  + valid: represent whether a record is valid or not
  + stay\_time: the total time of parking

## 2.4 Key Algorithm

* Quick sort (We call the qsort() function in <stdlib.h> and write functions that compares two elements.)
  + Compare by in-time

|  |
| --- |
| *int* comp\_by\_in\_time(const *void* \**a*, const *void* \**b*) { |
| const *struct* OneItem \*aa = (*struct* OneItem \*) a; |
| const *struct* OneItem \*bb = (*struct* OneItem \*) b;//cast the void\* to target type |
| return aa->time - bb->time;// compare by their in time |
| } |

* + Compare by stay-time

|  |
| --- |
| *int* comp\_by\_stay\_time(const *void* \**a*, const *void* \**b*) { |
| const *struct* OneItem \*aa = (*struct* OneItem \*) a; |
| const *struct* OneItem \*bb = (*struct* OneItem \*) b; |
| if (bb->stay\_time != aa->stay\_time)//cast the void\* to target type |
| return bb->stay\_time - aa->stay\_time;//compare by their in time |
| else// if they have the same in time |
| return bb->plate\_number < aa->plate\_number;//sort by their plate\_number |
| } |

* + Compare by plate number

|  |
| --- |
| *int* comp\_by\_stay\_plate(const *void* \**a*, const *void* \**b*) { |
| const *struct* OneItem \*aa = (*struct* OneItem \*) a; |
| const *struct* OneItem \*bb = (*struct* OneItem \*) b;//cast the void\* to target type |
| return bb->plate\_number > aa->plate\_number;// sort by their plate\_number |
| } |

* + The way to call them:
    - qsort (void\* base, size\_t num, size\_t size, int (\*compar)(const void\*,const void\*));
    - Parameters
      * base
        + Pointer to the first object of the array to be sorted, converted to a void\*.
      * num
        + Number of elements in the array pointed to by base.
        + size\_t is an unsigned integral type.
      * size
        + Size in bytes of each element in the array.
        + size\_t is an unsigned integral type.
      * compar
        + Pointer to a function that compares two elements.
* Eliminating the invalid items

|  |  |
| --- | --- |
|  | |
| |  | | --- | | *struct* OneItem result\_items[10000];//since we do not now how many valid item | | *int* result\_number = 0;//mark the valid number | | for (*int* i = 0; i < RECORD; i++) { | | if (all\_items[i].valid == 0) { | | continue;//already marked | | } else if (all\_items[i].valid == 1) { | | result\_items[result\_number].stay\_time = all\_items[i].stay\_time; | | result\_items[result\_number].plate\_number = all\_items[i].plate\_number; | | result\_items[result\_number].status = all\_items[i].status; | | result\_items[result\_number].time = all\_items[i].time; | | result\_number += 1;//copy the value to the target | | continue; | | } | | if (all\_items[i].status == -1) { | | all\_items[i].valid = 0;//if out first invalid then | | continue; | | }        //valid=-1 and status = 1 | | *int* valid\_flag = 0; | | for (*int* j = i + 1; j < RECORD; j++) { | | if (all\_items[i].valid != -1) | | continue; | | if (all\_items[i].plate\_number == all\_items[j].plate\_number && | | all\_items[i].status == 1 && all\_items[j].status == -1) { | | all\_items[i].valid = 1;// if they have the same plate number | | all\_items[j].valid = 1;// and they have different status, they got paired | | all\_items[i].stay\_time = all\_items[j].time - all\_items[i].time; | | all\_items[j].stay\_time = 0;// set the stay time, in to the correct out to zero | |  | | result\_items[result\_number].stay\_time = all\_items[i].stay\_time; | | result\_items[result\_number].plate\_number = all\_items[i].plate\_number; | | result\_items[result\_number].status = all\_items[i].status;//copy the source information | | result\_items[result\_number].time = all\_items[i].time;//to the target one | | result\_number += 1;//increment the counter | | valid\_flag = 1; | | break; | | } else if (all\_items[i].plate\_number == all\_items[j].plate\_number && | | all\_items[i].status == 1 && all\_items[j].status == 1) { | | all\_items[i].valid = 0;//if we encountered a same status | | valid\_flag = 1;// it should be marked as illegal | | break; | | } | | } | | if (valid\_flag == 0) { | | all\_items[i].valid = 0;//not found and mark illegal | | } | | } | |  |

* Merge the stay-time

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | for (*int* i = 0; i < result\_number - 1; i++) {//merge the same car's plate time | | if (result\_items[i].plate\_number == result\_items[i + 1].plate\_number) { | | result\_items[i + 1].stay\_time += result\_items[i].stay\_time; | | result\_items[i].stay\_time = 0;//set one of the source to zero | | } | | } | |

* + Before merging we use quick sort to rearrange them by plate number
  + For the records with the same plate number, add them together

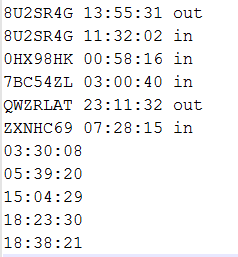
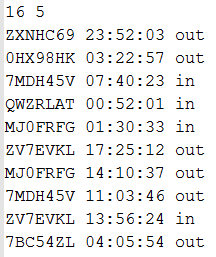
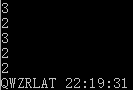
# Chapter 3: Testing Results

In the following test cases, a script is written in order to generate random input.

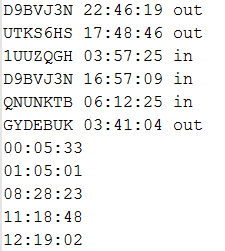
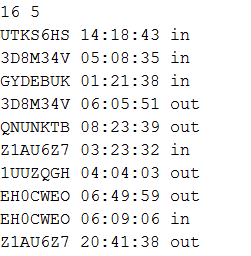
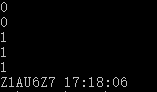
//test.js  
"use strict";  
**const** fs = require("fs");  
**let** shuffle = (a)=>{  
 **for** (**let** i = a.length; i; i--) {  
 **let** j = Math.floor(Math.random() \* i);  
 [a[i - 1], a[j]] = [a[j], a[i - 1]];  
 }  
}  
**let** randomString = (length)=>{  
 **let** text = "";  
 **let** possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";  
 **for**( **let** i=0; i < length; i++ )  
 text += possible.charAt(Math.floor(Math.random() \* possible.length));  
 **return** text;  
}  
**let** pad = (num, n)=> {  
 **var** len = num.toString().length;  
 **while**(len < n) {  
 num = "0" + num;  
 len++;  
 }  
 **return** num;  
}  
**const** recordCount = 16,  
 queryCount = 5;  
**let** record= [];  
**for**(**let** i=0;i<recordCount/2;i++){  
 **let** car\_plate=randomString(7);  
 **let** time = [];  
 **for**(**let** j=0;j<2;j++) {  
 time.push(`${pad(Math.floor(Math.random() \* 24),2)}:${pad(Math.floor(Math.random() \* 60),2)}:${pad(Math.floor(Math.random() \* 60),2)}`)  
 }  
 **for**(**let** j=0;j<2;j++) {  
 record.push({  
 state:time[j]<time[1^j]?"in":"out",  
 car\_plate: car\_plate,  
 time: time[j]  
 });  
 }  
}  
**let** query=[];  
**for**(**let** i=0;i<queryCount;i++){  
 query.push(`${pad(Math.floor(Math.random() \* 24),2)}:${pad(Math.floor(Math.random() \* 60),2)}:${pad(Math.floor(Math.random() \* 60),2)}`)  
}  
shuffle(record);  
query.sort();  
**let** fileData = `${recordCount} ${queryCount}\n`;  
**for**(**let** i=0;i<recordCount;i++){  
 fileData +=`${record[i].car\_plate} ${record[i].time} ${record[i].state}\n`;  
}  
**for**(**let** i=0;i<queryCount;i++){  
 fileData += `${query[i]}\n`;  
}  
fs.writeFile('test',fileData,(err)=>{  
 **if**(err) console.log(err);  
})

The generated test cases and the program results are listed below:

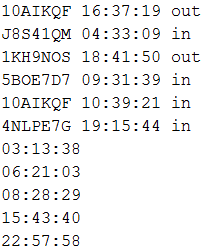
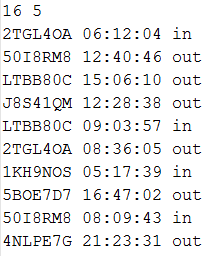
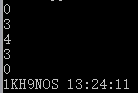
Test Case 1:

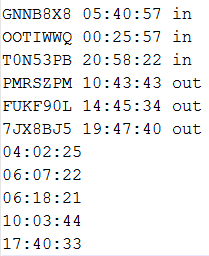
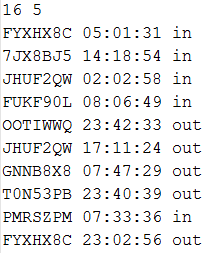
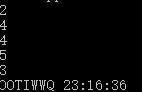
Test Case 2:

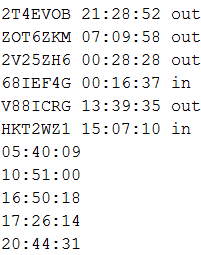
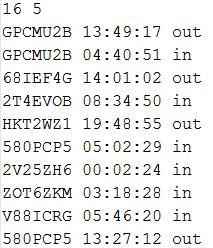
Test Case 3:

Test Case 4:

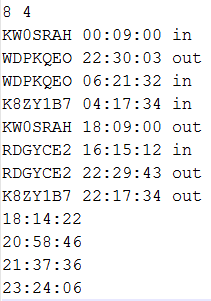
Test Case 5

The five test cases listed above are all generated randomly. Passing these test cases can convincingly prove the validity of this program in most common cases. However, hand-made test cases are needed to test the program’s behavior in edge situations.

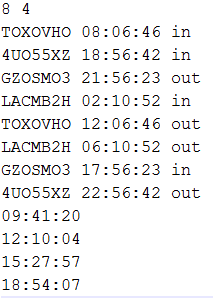
The next two test cases are to test program’s behavior when multiple solutions exist.

Test Case 6

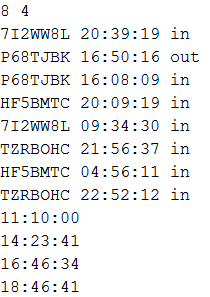
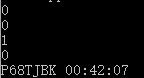
:

Test Case 7:

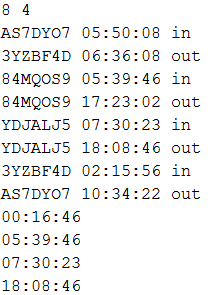
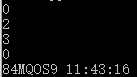
The next test case is to test program’s behavior when only one car is paired in the input.

Test Case 8:

The last test case is to test program’s behavior when the query time is exactly at the point when a car is moving in/out. The result under such situations are not mentioned in the problem specification. But we assume both moving in and out happen before the query time.

Test Case 9:

# Chapter 4: Analysis and Comments

This program has successfully passed all the test cases above, therefore its validity has been proved both in common cases and in edge situations.

We denote **N** as the number of records and **K** as the number of queries. In order to analyze the complexity of this program, we can divide it into several parts and analyze the complexity of each part respectively.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | Quick sort(average complexity) | Pair in&out | Find plate number | Output | Total |
| O(N+K) | O(NlogN) | O(N^2) | O(N) | O(K) | O(N^2 + K) |

As the form illustrates, we expect the time complexity of the whole program to be **O(N^2 + K)**. In order to prove this conclusion, we do mass data test for **N** and **K** respectively(so we can observe more cleverly).

The graphs above are drawn using1000,2000,3000,5000,10000 and 15000 lines data. The result is very consistent with our expectation. So we can conclude the average time complexity is **O(N^2 + K)**.

Square time complexity is acceptable in this problem, but there is still room for improvement. When finding the longest stay time, we can simply find out the maximal element(**O(N)**) instead of using a quick sort(**O(NlogN)**). And also, in order to reduce the time complexity of pairing the in and out record, we can store the records in an array and use a hash function to locate the element. Then only constant time is needed to find out an element with certain plate number. The ideal time complexity of the whole program would be **O(NlogN + K)** in this way.

# Appendix: Source Code

|  |  |
| --- | --- |
|  | |
| |  |  | | --- | --- | | 1 | // | | 2 | // Created by Haoxiangpeng on 12/20/2016. | | 3 | // Created and tested as per C99 standard | | 4 | // | | 5 | //Copyright (c) <year> <copyright holders> | | 6 |  | | 7 | //Permission is hereby granted, free of charge, | | 8 | // to any person obtaining a copy of this software and associated documentation files (the "Software"), | | 9 | // to deal in the Software without restriction, including without limitation the rights to | | 10 | // use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, | | 11 | // and to permit persons to whom the Software is furnished to do so, subject to the following conditions: | | 12 | //The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. | | 13 | //THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, | | 14 | // INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, | | 15 | // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. | | 16 | // IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, | | 17 | // WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. | | 18 |  | | 19 |  | | 20 | #include <stdio.h> | | 21 | #include <stdlib.h> | | 22 | #include <string.h> | | 23 |  | | 24 |  | | 25 | *struct* OneItem { | | 26 | *long* *long* plate\_number; | | 27 | *int* time; | | 28 | *int* status;// in:1 out:0 | | 29 | *int* valid; | | 30 | *int* stay\_time; | | 31 | }; | | 32 |  | | 33 | *long* *long* str\_to\_long(*char* \**str*) { | | 34 | *long* *long* num = 1;// convert the string to long long type | | 35 | *int* i = 0;// so that we can easily sort the data | | 36 | while (str[i]) { | | 37 | num \*= 100; | | 38 | num += str[i]; | | 39 | i++; | | 40 | }// to avoid conflict | | 41 | return num; | | 42 | } | | 43 |  | | 44 | *char* \*long\_to\_string(*long* *long* *num*) { | | 45 | *char* \*plate\_number;// convert the long long type back to the string | | 46 | plate\_number = malloc(sizeof(*char*) \* 7); | | 47 | for (*int* i = 6; i >= 0; i--) { | | 48 | *int* temp = num % 100; | | 49 | plate\_number[i] = (*char*) temp; | | 50 | num /= 100;// to avoid the conflict | | 51 | } | | 52 | plate\_number[7] = '\0';// to mark the stop of the plate\_number | | 53 | return plate\_number; | | 54 | } | | 55 |  | | 56 | *int* comp\_by\_in\_time(const *void* \**a*, const *void* \**b*) { | | 57 | const *struct* OneItem \*aa = (*struct* OneItem \*) a; | | 58 | const *struct* OneItem \*bb = (*struct* OneItem \*) b;//cast the void\* to target type | | 59 | return aa->time - bb->time;// compare by their in time | | 60 | } | | 61 |  | | 62 | *int* comp\_by\_stay\_time(const *void* \**a*, const *void* \**b*) { | | 63 | const *struct* OneItem \*aa = (*struct* OneItem \*) a; | | 64 | const *struct* OneItem \*bb = (*struct* OneItem \*) b; | | 65 | if (bb->stay\_time != aa->stay\_time)//cast the void\* to target type | | 66 | return bb->stay\_time - aa->stay\_time;//compare by their in time | | 67 | else// if they have the same in time | | 68 | return bb->plate\_number < aa->plate\_number;//sort by their plate\_number | | 69 | } | | 70 |  | | 71 | *int* comp\_by\_stay\_plate(const *void* \**a*, const *void* \**b*) { | | 72 | const *struct* OneItem \*aa = (*struct* OneItem \*) a; | | 73 | const *struct* OneItem \*bb = (*struct* OneItem \*) b;//cast the void\* to target type | | 74 | return bb->plate\_number > aa->plate\_number;// sort by their plate\_number | | 75 | } | | 76 |  | | 77 |  | | 78 | *int* main() { | | 79 | *unsigned* *int* RECORD, QURERY; | | 80 |  | | 81 | *struct* OneItem \*all\_items; | | 82 | *int* \*all\_queries; | | 83 |  | | 84 | scanf("%d%d", &RECORD, &QURERY); | | 85 | all\_items = (*struct* OneItem \*) malloc(sizeof(*struct* OneItem) \* RECORD);//all the records | | 86 | all\_queries = (*int* \*) malloc(sizeof(*int*) \* QURERY);// all the queries | | 87 | for (*int* i = 0; i < RECORD; i++) { | | 88 | *char* plate\_number[10], tem\_status[5]; | | 89 | *int* hour, minute, second; | | 90 | scanf("%s %d:%d:%d %s", plate\_number, &hour, &minute, &second, tem\_status);//read from stdin | | 91 | all\_items[i].plate\_number = str\_to\_long(plate\_number); | | 92 | all\_items[i].time = hour \* 3600 + minute \* 60 + second; | | 93 | all\_items[i].status = strcmp(tem\_status, "in") ? -1 : 1;//if in then 1 else -1 | | 94 | all\_items[i].valid = -1;//initialize the data | | 95 | all\_items[i].stay\_time = 0;//initialize the data | | 96 | } | | 97 | for (*int* i = 0; i < QURERY; i++) { | | 98 | *int* hour, minute, second; | | 99 | scanf("%d:%d:%d", &hour, &minute, &second); | | 100 | all\_queries[i] = hour \* 3600 + minute \* 60 + second;// convert the time to int, so that we can sort it | | 101 | } | | 102 | qsort(all\_items, RECORD, sizeof(*struct* OneItem), comp\_by\_in\_time);// Sort all the records per in time | | 103 |  | | 104 | // eliminate the invalid items | | 105 | *struct* OneItem result\_items[10000];//since we do not now how many valid item | | 106 | *int* result\_number = 0;//mark the valid number | | 107 | for (*int* i = 0; i < RECORD; i++) { | | 108 | if (all\_items[i].valid == 0) { | | 109 | continue;//already marked | | 110 | } else if (all\_items[i].valid == 1) { | | 111 | result\_items[result\_number].stay\_time = all\_items[i].stay\_time; | | 112 | result\_items[result\_number].plate\_number = all\_items[i].plate\_number; | | 113 | result\_items[result\_number].status = all\_items[i].status; | | 114 | result\_items[result\_number].time = all\_items[i].time; | | 115 | result\_number += 1;//copy the value to the target | | 116 | continue; | | 117 | } | | 118 | if (all\_items[i].status == -1) { | | 119 | all\_items[i].valid = 0;//if out first invalid then | | 120 | continue; | | 121 | }        //valid=-1 and status = 1 | | 122 | *int* valid\_flag = 0; | | 123 | for (*int* j = i + 1; j < RECORD; j++) { | | 124 | if (all\_items[i].valid != -1) | | 125 | continue; | | 126 | if (all\_items[i].plate\_number == all\_items[j].plate\_number && | | 127 | all\_items[i].status == 1 && all\_items[j].status == -1) { | | 128 | all\_items[i].valid = 1;// if they have the same plate number | | 129 | all\_items[j].valid = 1;// and they have different status, they got paired | | 130 | all\_items[i].stay\_time = all\_items[j].time - all\_items[i].time; | | 131 | all\_items[j].stay\_time = 0;// set the stay time, in to the correct out to zero | | 132 |  | | 133 | result\_items[result\_number].stay\_time = all\_items[i].stay\_time; | | 134 | result\_items[result\_number].plate\_number = all\_items[i].plate\_number; | | 135 | result\_items[result\_number].status = all\_items[i].status;//copy the source information | | 136 | result\_items[result\_number].time = all\_items[i].time;//to the target one | | 137 | result\_number += 1;//increment the counter | | 138 | valid\_flag = 1; | | 139 | break; | | 140 | } else if (all\_items[i].plate\_number == all\_items[j].plate\_number && | | 141 | all\_items[i].status == 1 && all\_items[j].status == 1) { | | 142 | all\_items[i].valid = 0;//if we encountered a same status | | 143 | valid\_flag = 1;// it should be marked as illegal | | 144 | break; | | 145 | } | | 146 | } | | 147 | if (valid\_flag == 0) { | | 148 | all\_items[i].valid = 0;//not found and mark illegal | | 149 | } | | 150 | } | | 151 |  | | 152 | // Print the query result | | 153 | *int* car\_number = 0, query\_index = 0, result\_index = 0; | | 154 | while (query\_index < QURERY) { | | 155 | if (result\_items[result\_index].time <= all\_queries[query\_index]) {//earlier than queried,status can only be 1 or -1 | | 156 | car\_number += (result\_index < result\_number ? result\_items[result\_index].status : 0);//if reached the end, status shall not change | | 157 | result\_index++;// go to next car item | | 158 | } else { | | 159 | printf("%d\n", car\_number);//out put the status number | | 160 | query\_index++;//next query | | 161 | } | | 162 | } | | 163 |  | | 164 | // Sort by plate number first, so that we can merge the time with O(N) | | 165 | qsort(result\_items, (*size\_t*) result\_number, sizeof(*struct* OneItem), comp\_by\_stay\_plate); | | 166 | for (*int* i = 0; i < result\_number - 1; i++) {//merge the same car's plate time | | 167 | if (result\_items[i].plate\_number == result\_items[i + 1].plate\_number) { | | 168 | result\_items[i + 1].stay\_time += result\_items[i].stay\_time; | | 169 | result\_items[i].stay\_time = 0;//set one of the source to zero | | 170 | } | | 171 | } | | 172 |  | | 173 | // Sort, the first key is stay time, second key is plate number | | 174 | qsort(result\_items, (*size\_t*) result\_number, sizeof(*struct* OneItem), comp\_by\_stay\_time); | | 175 |  | | 176 | *int* index\_number = 0; | | 177 | while (1) {//output the plate number | | 178 | printf("%s ", long\_to\_string(result\_items[index\_number].plate\_number));//judge if we have two same entry | | 179 | if (result\_items[index\_number].stay\_time != result\_items[index\_number + 1].stay\_time) break; | | 180 | index\_number++;//go to next | | 181 | } | | 182 | printf("%02d:%02d:%02d", result\_items[0].stay\_time / 3600, (result\_items[0].stay\_time % 3600) / 60, | | 183 | result\_items[0].stay\_time % 3600 % 60);//out put the top time | | 184 | } | |  |

# Declaration

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

Duty Assignments:

**Programmer:** Hao Xiangpeng

**Tester:** Yang Kefan

**Report Writer:** Li Haipeng