Cars on Campus

2019-12-27

Contents

1. Introduction

1.1 Description

We are suppose to monitor the cars on campus within 24 hours of a day. With the information of license plate numbers of all cars and their times getting in/or the campus, we need to tell the number of cars on campus at any given time. In addition, we should tell the cars which have parked the longest time during the day.

Note that when an `in` record is not paired with an `out` record that is chronically next to it, the `in` record should be ignored. Moreover, a car may get in/out the campus more than once.

2. Algorithm Specification

2.1 Data Structure Analysis

We used two data structure in this code, `Record` and `Period`.

`Record` is used to store the information of one in/out record, while the `period` is used to store the total parking time on campus for each car. They are defined as follow.

```
struct Record {
    char plate_number[MaxCharInPlateNum];       // record the plate number
    int status;                                 // record the in/out status
    int time;                                   // record the time in
seconds
};

struct Period {
    char plate_number[MaxCharInPlateNum];       // record the plate number
    int period;                                 // record the total parking
time
};
```

2.2 Algorithm Specification

In this algorithm, we sort the records for 2 times using `qsort` function.

First, we sort the original records according to the plate number in alphabetical order (if the plate number are same, then sort them in ascending time order). After this sort, all the records with the identical plate number are adjacent. Then, we can delete all the invalid records (those with single in or out status) and get the final records, since the qualified records are those adjacent records with one in and one out and the same plate number.

The second sort is implemented on the final records. **We sort all the valid records according to the ascending time order.** In this way, we're able to determine the number of cars on campus at any specific time. The calculation used an array `count` to record the number of cars on campus at a specific time. For example, `count[i]` represents the number of cars on campus at the the i^{th} smallest time in that ascending time sequence, and it's just the sum of all the elements from `count[0]` to `count[i]`. It's tricky since we had assigned "in" as 1 and "out" as -1 , we can calculate the number of cars on campus by adding all the record status to a specific time. If one car get in and doesn't get out, the sum would add 1, if the car get out, the sum would add 0.

```
procedure main()
    input N    // the number of records
    input K     // the number of queries
    Records = Read(N)
    qsort(Records, cmp_plateNum)
    FianlRecords = Adjust(Records)
    Periods = FindLongestParkingTime(FinalRecords)
    qsort(FinalRecords, cmp_time)
    count = CalculateCount(FinalRecords)
    FindParkingCar(count, K, FinalRecords)
    FindLongestParkingCar(Periods, longest)
    return 0

procedure Read(N)
    for(i=0;i<N;i++)
        input Records[i].plateNumber
        input hour, min, sec
        Records[i].time = convert time into seconds
        input status
        Records[i].status = status=="in"?0:1:-1
    return Records

procedure Adjust(Records)
    for(i=0;i<N-1;i++)
 if(Records[i].plateNumber==Records[i+1].plateNumber&&Records[i].status==1&&Records[i].status==-1)
            FinalRecords[++pointer1]=Records[i]
            FinalRecords[++pointer1]=Records[i+1]
    return FianlRecords

procedure FindLongestParkingTime
        for(i=0;i<N-1;i++)
 if(Records[i].plateNumber==Records[i+1].plateNumber&&Records[i].status==1&&Records[i].status==-1)
            Periods[++pointer2]=FinalRecords[i].plateNumber
            Periods[pointer2]=FinalRecords[i+1].time-FinalRecords[i].time
            if(pointer2>=1&&Periods[pointer2].plateNumber==Periods[pointer2-1].plateNumber)
                Periods[pointer2-1].period += Periods[pointer2].period
                pointer2--

            update longest
    return Periods

procedure CalculateCount(FinalRecords)
    for(i=0;i<pointer1;i++)
        if(i==0)
            count[i] = FinalRecords[i].status
        else count[i] = count[i-1]+FinalRecords[i].status
    return count

procedure FindParkingCar(count, K, FinalRecords)
    index = 0
    for(i=0;i<K;i++)
        input hour, min, sec
        temptime = convert time into seconds
        for(j=index;j<=pointer1;j++)
            if(FinalRecords[j].time>temptime)
                print(count[j-1])
                break
            else if(j==pointer1)
                print(count[j])
        index = j

procedure FindLongestParkingCar(longest, Periods)
    for(i=0;i<pointer2;i++)
        if(Periods[i].period==longest)
            print(Periods[i].plateNuber)
    prnit(longest in specific format)
```

2.3 Correctness of the Algorithm

In this algorithm, we make use of the `qsort` function. First, we read in all the input records. Note that we assign the status to be 1 if it's "in", otherwise it would be 1. This is important for the convenience to compute the number of car on campus during the query section.

Then we sort the records according to the plate number (if the plate number are same, then sort them in ascending time order). Then all the records with the same plate number would be adjacent. According to this property, we can get the valid records. The qualified records are those adjacent records with one in and one out and of the same plate number.

Then we can compute and record the total stay time for each car and get the `longest parking time` of that day.

After that, we sort the valid records according to the ascending time order, and we get an ascending time sequence. To compute the number of cars on campus at any specific time, we initialize a `count` array. It records the number of cars on campus at a specific time. For example, `count[i]` represents the number of cars on campus at the the i^{th} smallest time in that ascending time sequence, and it's just the sum of all the elements from `count[0]` to `count[i]`. It's tricky since we had assigned "in" as 1 and "out" as -1, we can calculate the number of cars on campus by adding all the record status to a specific time. If one car get in and doesn't get out, the sum would add 1, if the car get out, the sum would add 0. With the help of this array, we can get the number of cars on campus easily.

Finally, we search the total stay time of each car, if it's equal to the `longest parkin time` , then we output it's plate number. Since the original records are in alphabetical records, the output sequence is in alphabetical records.

3. Testing Results

- **Test point 0: with unpaired input and parallel maximum time**

> **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

- **Test point 1: There are cars parked all day, and there are cars in and out at t time**

> **Input :**
>
> 20 10

```
BF0037A 00:00:00 in
KM007BD 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
KM007BD 05:09:59 in
ZA3Q625 11:42:01 out
KM007BD 05:10:33 in
ZA3Q625 06:30:50 in
KM007BD 12:23:42 out
TQ332WQ 00:00:00 in
BF0037A 23:59:59 out
ZA3Q625 23:55:00 in
KM007BD 12:24:23 out
ZA133CH 17:11:22 out
KM007BD 18:07:01 out
DB8888A 06:30:50 in
TQ332WQ 23:59:59 out
05:04:00
06:30:50
08:23:40
10:25:25
11:22:17
11:23:56
12:23:42
13:00:00
15:00:39
23:59:59
```

Output :

```
3
6
6
```

> 7
>
> 7
>
> 7
>
> 4
>
> 3
>
> 3
>
> 0
>
> BF0037A TQ332WQ 23:59:59

- **Test point 2: N vehicles are behind T, but ‘in’ is before T**

> **Input :**
>
> 20 10
>
> PQ060ET 04:41:21 in
>
> GR270RT 06:40:25 in
>
> FU264YV 10:46:20 in
>
> EB003WN 08:43:21 in
>
> RK727MW 03:53:47 in
>
> RQ816ND 06:59:31 in
>
> HU619BT 02:35:27 in
>
> YX625OW 00:51:24 in
>
> JC460IX 02:43:15 in
>
> OI374BE 11:19:31 in
>
> PQ060ET 18:21:05 out
>
> GR270RT 19:58:32 out
>
> FU264YV 19:46:44 out
>
> EB003WN 15:28:46 out
>
> RK727MW 22:06:09 out
>
> RQ816ND 12:34:18 out
>
> HU619BT 14:20:33 out
>
> YX625OW 16:02:09 out
>
> JC460IX 16:10:34 out
>
> OI374BE 21:22:06 out
>
> 06:30:50
>
> 08:23:40
>
> 10:25:25

> 11:22:17
>
> 11:23:56
>
> 12:23:42
>
> 13:00:00
>
> 15:00:39
>
> 21:55:33
>
> 22:35:12

> **Output :**
>
> 5
>
> 7
>
> 8
>
> 10
>
> 10
>
> 12
>
> 10
>
> 9
>
> 8
>
> 1
>
> 0
>
> RK727MW 18:12:22

- **Test point 3: Minimum N**

> Because any ‘in’ records that are not paired with an ‘out’ record are ignored, as are ‘out’ records not paired with an ‘in’ record and it is guaranteed that at least one car is well paired in the input, so the minimum N is 2.

> **Input :**
>
> 2 3
>
> JW0137A 23:55:24 out
>
> JW0137A 17:22:18 in
>
> 11:33:10
>
> 20:11:44
>
> 23:58:19

> **Output :**
>
> 0
>
> 1
>
> 0

> ```
> JW0137A 06:33:06
> ```

- **Test point 4: Maximum N**

> **Input :**
>
> See “Max_N.txt” for details.

> **Output :**
>
> ```
> 449
> 553
> 745
> 795
> 965
> 1112
> 1199
> 1224
> 1223
> 1221
> 1231
> 1154
> 939
> 817
> 718
> 620
> 444
> 381
> 271
> 0
> ET547KU 23:37:55
> ```

- **Test point 5: Maximum juxtaposition**

> **Input :**
>
> See “Max_juxtaposition.txt” for details.

> **Output :**
>
> ```
> 3
> 1
> 1
> 1
> ```

> 0
>
> 2
>
> 1
>
> 1
>
> 1
>
> 2
>
> 0
>
> 1
>
> 1
>
> 1
>
> 0
>
> 0
>
> 0
>
> 0
>
> 2
>
> 0
>
> BV623TE CA795KB CA981JO CO137DI GL179TX HE188ZI JI402BF JO304EB JU170OS KJ974FP OM905TA PG938WP PH110OL QR496PN SW390DQ SY927QA TK467NB YC323RE YV445AF ZY812KO 01:00:00

4. Analysis and Comments

4.1 Time Complexity

The average time complexity for this algorithm is $O(NlogN)$, and the worst case would be $O(max(N^2, KN))$.

The `main` function divides into 8 parts. The `Read` function reads in the data in sequence, so it's $O(N)$. The two `qsort` function is the same as *quick sort* , the average time complexity is $O(NlogN)$, and the worst case time complexity is $O(N^2)$. `FindLongestParkingTime` traverse the records linearly, so its time complexity is $O(N)$, as is the `CalculateCount` function. As for the `FindParkingCar` function, we use a variable `index` to avoid the redundant search, since the queries are in ascending time order. So its time complexity is related to K and the worst case would be $O(KN)$. The last function `FindLongestParkingCar` is $O(N)$.

As a result, the average time complexity for this algorithm is $O(NlogN)$, while the worst case would be $O(max(N^2, KN))$.

4.2 Space Complexity

The worst case space complexity and for this algorithm is $O(N)$

We malloced several data structures with the size of N, such as `Records`, `FinalRecords`, `count` ... The `qsort` uses stack to do recursion, so it's average space complexity is $O(logN)$, and the worst case would be $O(N)$.

In all, the worst case space complexity is $O(N)$.

5. Appendix

> Source Code in C

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

#define MaxCharInPlateNum 8     // the maximum number of characters in a plate number
#define MaxCharInStatus 4       // the maximum number of characters in the string status

/**********************************************
       Data Structure Used in This Code

* Record: Used to store the information of one
          in/out record

* Period: Used to store the total stay time on
          campus for each car

***********************************************/

typedef struct Record* PtrToRecord;
typedef struct Period* PtrToPeriod;

struct Record {
    char plate_number[MaxCharInPlateNum];
    int status;
    int time;
};

struct Period {
    char plate_number[MaxCharInPlateNum];
    int period;
};

/************************************************
        Compare Functions Used in qsort

* cmp_plateNum: determines the sort according to the
        plate number of the car in alphabetical order;
        if two records has the same plate number, then
        sort them according to the ascending time order

* cmp_time: determine the sort according to the ascend-
        ing time order.

```

```c
**************************************************/

int cmp_plateNum(const void* a, const void* b) {
    if (strcmp((*(PtrToRecord)a).plate_number, (*(PtrToRecord)b).plate_number) != 0) {
        return strcmp((*(PtrToRecord)a).plate_number, (*(PtrToRecord)b).plate_number);
    }
    else {
        return (*(PtrToRecord)a).time - (*(PtrToRecord)b).time;
    }
}

int cmp_time(const void* a, const void* b) {
    return (*(PtrToRecord)a).time - (*(PtrToRecord)b).time;
}

/**********************************************************
                Functions

* Read: Read in all the information of the input
        records

* Adjust: delete all the invalid records(those single
        in or out)

* FindLongestParkingTime: Find the longest parking time
        on campus during that day.

* CalculateCount: Calculate the count array

* FindParkingCar: Using the count array to respond to
        the querise.

* FindLongestParkingCar: Search Through the Periods and
        output the plate number of those longest stay car
**********************************************************/

PtrToRecord Read(int N);
PtrToRecord Adjust(int N, PtrToRecord Records, int* FinalRecordsPointer);
PtrToPeriod FindLongestParkingTime(int N, int FinalRecordsPointer, int* PeriodsPointer, PtrToRecord FinalRecords, int* longest);
int* CalculateCount(int FinalRecordsPointer, PtrToRecord FinalRecords);
void FindParkingCar(int K, int* count, int FinalRecordsPointer, PtrToRecord FinalRecords);
void FindLongestParkingCar(int longest, PtrToPeriod Periods, int PeriodsPointer);

int main()
{
/**************************************************************************
                    Variables Used

* Records: An structure array which stores all the input records.

* FinalRecords: An structure array that stores the valid records.
            (Haved deleted the single in/out records)

* Periods: An structure array that stores the total stay time of
            each car.

* count: Since all the time in the records are unique, so we can
        sort the records according to the ascending time. So "count"
        is an array that calculate the number of cars on campus at
        a specfic time. For example, count[i] represents the number
        of cars on campus at the the ith smallest time in the ascending
        time sequence.

* FinalRecordsPointer: A pointer for the structure array "FinalRecords".

* PeriodsPointer: A pointer for the structure array "Periods".

* index: Since the queries are given in ascending order of the times. After
        we sort the final records in ascending time order, we can record
        the index of the last search and start the next search from this
        index to avoid redundent search.

* longest: Records the longest parking time on campus that day

*************************************************************************/

    int N, K;       // N is the number of records, K is the number of queries
    int FinalRecordsPointer = -1, PeriodsPointer = -1;
    int longest = 0;  // record the longest stay time
    int* count = NULL;
    PtrToRecord Records = NULL, FinalRecords = NULL;
    PtrToPeriod Periods = NULL;

    scanf("%d %d", &N, &K);

    // a function that reads in all the records information
    Records = Read(N);
    // sort the records according to the plate number
    qsort(Records, N, sizeof(Records[0]), cmp_plateNum);
    // delete all the invalid records (single in/out), get the final records
    FinalRecords = Adjust(N, Records, &FinalRecordsPointer);
    // find the longest parking time on campus
    Periods = FindLongestParkingTime(N, FinalRecordsPointer,&PeriodsPointer, FinalRecords, &longest);
    // sort the final records according to the ascending time order
    qsort(FinalRecords, FinalRecordsPointer + 1, sizeof(struct Record), cmp_time);
    // calculate the count array
    count = CalculateCount(FinalRecordsPointer, FinalRecords);
    // using the count array to get the car number of the queried time
    FindParkingCar(K, count, FinalRecordsPointer, FinalRecords);
    // Search Through the Periods and output the plate number of those longest stay car
    FindLongestParkingCar(longest, Periods, PeriodsPointer);

    return 0;
}

```

```c


PtrToRecord Read(int N)
{
    int i, j, FinalRecordsPointer = -1, PeriodsPointer = -1, index = 0;
    PtrToRecord Records = (PtrToRecord)malloc(N * sizeof(struct Record));

    for (i = 0; i < N; i++) {
        char temp_status[4];
        int hh, mm, ss;
        scanf("%s %d:%d:%d %s\n", Records[i].plate_number, &hh, &mm, &ss, temp_status);     // read in the information
        Records[i].time = hh * 3600 + mm * 60 + ss;
        // if the status is "in", assign value 1, otherwise -1, this makes calculate the array "count" convenient
        Records[i].status = strcmp(temp_status, "in") == 0 ? 1 : -1;
    }

    return Records;
}

PtrToRecord Adjust(int N, PtrToRecord Records, int* FinalRecordsPointer)
{
    int i;
    PtrToRecord FinalRecords = (PtrToRecord)malloc(N * sizeof(struct Record));

    for (i = 0; i < N - 1; i++) {
        // since all the records with the same plate number are adjacent, the qualified records are those adjacent records with one in and one out of the same plate number
        if (strcmp(Records[i].plate_number, Records[i + 1].plate_number) == 0 && Records[i].status == 1 && Records[i + 1].status == -1) {
            FinalRecords[++(*FinalRecordsPointer)] = Records[i];
            FinalRecords[++(*FinalRecordsPointer)] = Records[i + 1];
        }
    }

    return  FinalRecords;
}

PtrToPeriod FindLongestParkingTime(int N, int FinalRecordsPointer, int* PeriodsPointer, PtrToRecord FinalRecords, int* longest)
{
    int i;
    PtrToPeriod Periods = (PtrToPeriod)malloc(N * sizeof(struct Period));

    for (i = 0; i < FinalRecordsPointer; i++) {
        if (strcmp(FinalRecords[i].plate_number, FinalRecords[i + 1].plate_number) == 0 && FinalRecords[i].status == 1 && FinalRecords[i + 1].status == -1) {
            strcpy(Periods[++(*PeriodsPointer)].plate_number, FinalRecords[i].plate_number);   // we assume the plate number is not exist in the Periods
            Periods[(*PeriodsPointer)].period = FinalRecords[i + 1].time - FinalRecords[i].time;     // calculate the stay time and store it

            if ((*PeriodsPointer) >= 1 && strcmp(Periods[(*PeriodsPointer)].plate_number, Periods[(*PeriodsPointer) - 1].plate_number) == 0) {   // the car plate number already exist in the Periods
                Periods[(*PeriodsPointer) - 1].period += Periods[(*PeriodsPointer)].period;  // add the current stay time to the previous stay time
                (*PeriodsPointer)--;    // update the PeriodsPointer
            }

            if ((*longest) < Periods[(*PeriodsPointer)].period) {       // update the longest stay time
                (*longest) = Periods[(*PeriodsPointer)].period;
            }
        }
    }

    return Periods;
}

int* CalculateCount(int FinalRecordsPointer, PtrToRecord FinalRecords)
{
    int i;
    int* count = (int*)malloc(sizeof(int) * FinalRecordsPointer);

    /* the method to calculate the count array is tricky. Since we assigned "in" as 1 and "out" as -1,
        we can calculate the number of cars on campus by adding all the record status to a specific time.
        If one car get in and not get out, the sum would add 1, if the car get out, the sum would add 0.
    */
    for (i = 0; i <= FinalRecordsPointer; i++) {
        if (i == 0)
            count[i] = FinalRecords[i].status;
        else
            count[i] = count[i - 1] + FinalRecords[i].status;
    }

    return count;
}

void FindParkingCar(int K, int* count, int FinalRecordsPointer, PtrToRecord FinalRecords)
{
    int index = 0, i, j;
    /*Since the queries are given in ascending order of the times. After
      we sort the final records in ascending time order, we can record
      the index of the last search and start the next search from this
      index to avoid redundent search.*/

    for (int i = 0; i < K; i++) {
        int hh, mm, ss;
        scanf("%d:%d:%d", &hh, &mm, &ss);
        int temptime = hh * 3600 + mm * 60 + ss;
        for (j = index; j <= FinalRecordsPointer; j++) {
            if (FinalRecords[j].time > temptime) {
                printf("%d\n", count[j - 1]);
                break;
            }
            else if (j == FinalRecordsPointer) {
                printf("%d\n", count[j]);
            }
        }
        index = j;
    }
}

void FindLongestParkingCar(int longest, PtrToPeriod Periods, int PeriodsPointer)
{
    int i;

    // Since the Periods has already in alphabetical order, we can search for the plate number in O(N).
    for (i = 0; i <= PeriodsPointer; i++) {
        if (Periods[i].period == longest) {
            printf("%s ", Periods[i].plate_number);
        }
    }
    // output the plate number in specific format
    printf("%02d:%02d:%02d", longest / 3600, (longest % 3600) / 60, longest % 60);
}



```

6. Declaration

We thereby declare that all the work done in this project is of out independent effort as a group.

7. Duty Assignment

Programmer: 杨云皓

Tester: 张佳文

Report Writer: 王睿