浙江大学

Fundamentals of Data Structures

Project3

Project Report

2020~2021 秋冬学期　2020 年 12 月 30 日

Arrangement of Computation Tasks

Categories

Project3

Chapter 1: Introduction (6 pts.)

1.1 Background and significance of topic selection

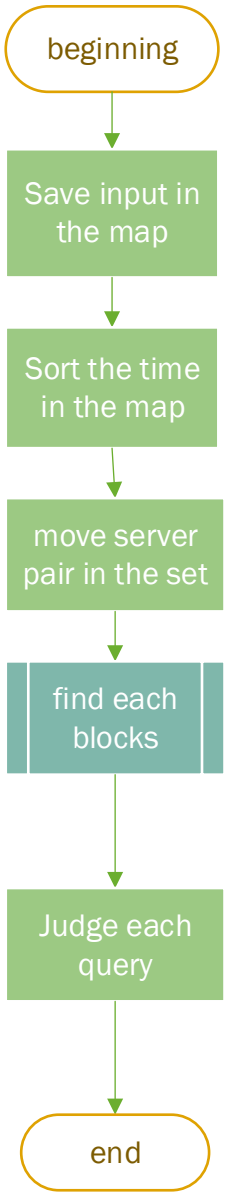
We often need to allocate time and money to some tasks, and they are often parallel, this program can help us to quickly calculate the longest running time of some servers that runs in parallel. There are plenty of servers running at different times of the day. we use all available server information and find the total number of valid startup times. Proper planning can save resources and increase efficiency

1.2 Our goals

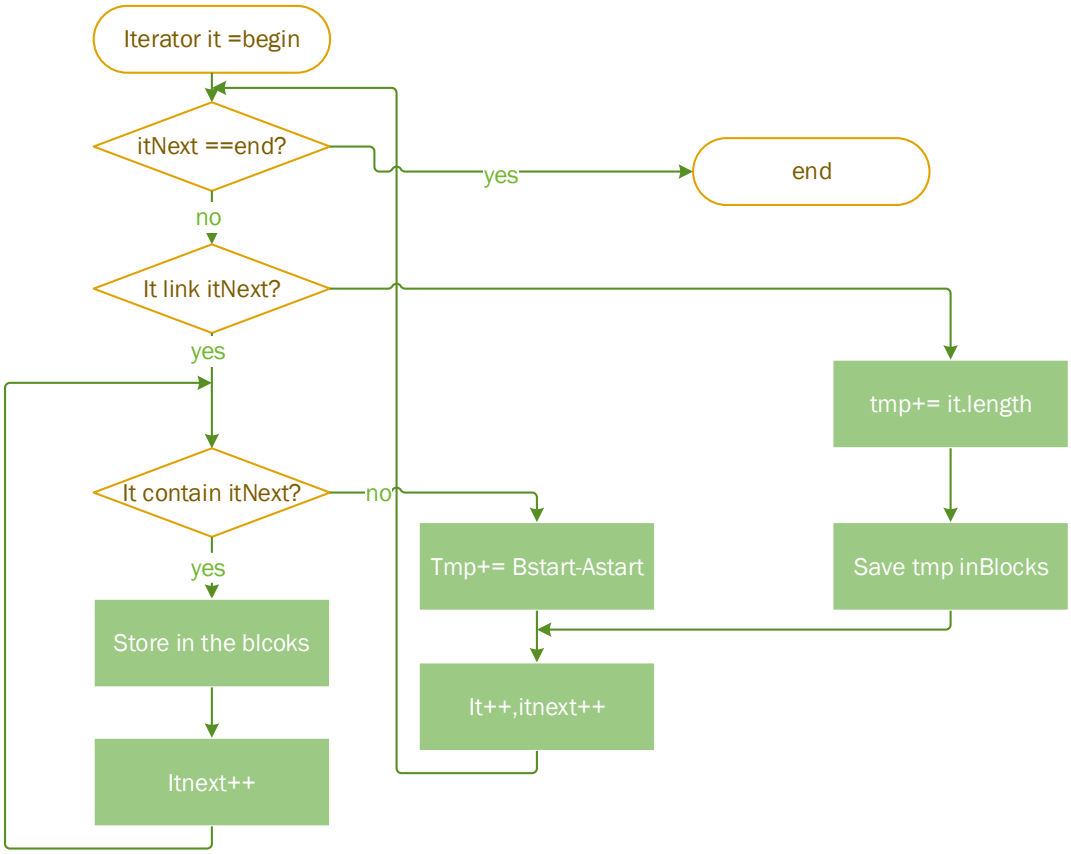
Calculate the longest running time of some servers that runs in parallel, and when a task can start, i.e. how much free allocation to choose from. Find the longest task that can run, not add tasks up.

Chapter 2: Algorithm Specification (12 pts.)

2.1 Overall architecture design



Find each blocks algorithm is following:



2.2 Algorithm design

First put each server in the map, according to the time, then minutes, and then map sort each server
And then put it in the set. Map is to organize every server.
In fact, there is no need to distinguish between server name in set.
Then line up the time for the server to start. Because the time guaranteed for the first server will not be their next repeat.
If you're looking for the longest, look at the earliest start time it,
How to process the itnext ends earlier than it, starts later than it? directly add in vector。
How to process the itnext ends later than it, starts later than it? link them。
The specific implementation is:
If it “contains” itnext, then itnext is added to the block. And then itnext;
until it is not included, to see if the next server starts before it ends,
If there is one, shows that itnext is connected to it, then temptime = itnextstat-itstat.
Then it = itnext; itnext++; Continue traversing and the next server has not started before it ends, until no server starts before it ends.
Finally, if there is no connection, then the length of the temptime += it. Then add temptime to the vector

2.3Main data structures

```
class time{
public:
    time(int h,int m,int s); // have three para
    int hours;int minutes;int seconds;
};

class server{  //every pair
public:
    server(time start,time end);
    time start;
    time end;
    string name;
    bool operator< (const server &b) const {  // sort by the starting time
point ascending
        return comp(this->start, b.start); // true is earlier,  false;
    }
};
```

I save the class time in the map first,

```
map<string, std::vector<time>> m1;
```

and then move the servers pair in the set,

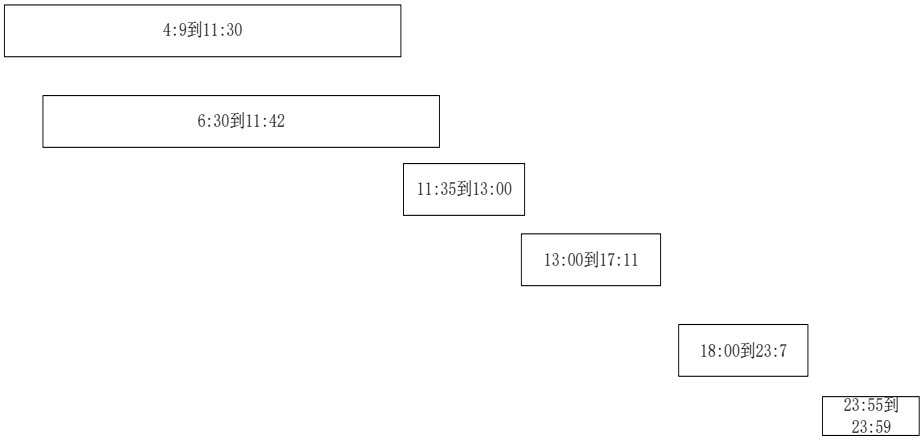
```
set<server> ser1;// store all the time blocks, each server has <start,end>
```

save the blocks time in vector

```
vector<int> blocks;//save each blcoks
```

Chapter 3: Testing Results (20 pts.)

test case	Actual behavior of my program
12 7 jh007bd 18:00:01 zd00001 11:30:08 db8888a 13:00:00 za3q625 23:59:50	31801 1201 0 13202

za133ch 13:00:00 zd00001 04:09:59 za3q625 11:42:01 za3q625 06:30:50 za3q625 23:55:00 za133ch 17:11:22 jh007bd 23:07:01 db8888a 11:35:50 08:30:01 12:23:42 05:10:00 04:11:21 00:04:10 05:06:59 00:05:11  one comprehensive test. Three blocks link,	20063 64556 13385
8 7 jh007bd 18:00:01 db8888a 13:00:00 za3q625 23:59:50 za133ch 13:00:00 za3q625 23:55:00 za133ch 17:11:22 jh007bd 23:07:01 db8888a 11:35:50 08:30:01 12:23:42 05:10:00 04:11:21 00:04:10 05:06:59	18420 0 0 0 3342 37805 2 37622

00:05:11 No link	
2 2 db8888a 13:00:00 db8888a 11:35:50 04:11:21 00:04:10 Smallest server, one can, one cannot	5050 0 4801
6 7 zd00001 11:30:08 db8888a 13:00:00 zd00001 04:09:59 za3q625 11:42:01 za3q625 06:30:50 db8888a 11:35:50 08:30:01 12:23:42 05:10:00 04:11:21 00:04:10 05:06:59 00:05:11 all linked	31801 1201 0 13202 16721 31552 13383 31491
0 7 08:30:01 12:23:42 05:10:00 04:11:21 00:04:10 05:06:59 00:05:11 No record	don't have any server record!!

Chapter 4: Analysis and Comments (10 pts.)

time complexities of the algorithms: O(nlogn)
because
save in map, it is implemented with red and black trees.
Insert in set, it is implemented with red and black trees. O(logn)
Save server pair in the set: O(nlogn)
Traverse the set: O(n) traverse all elements.

space complexities of the algorithms
The space complexities is O(n)
Because we use the Map O(n),Set O(n)

Appendix: Source Code

```
//
// Created by 12638 on 2020/12/26.
// the longest computation task you could run,
// and find the total amount of valid starting time for a given computation
task
/*
 * first line is analysis the server and calculate the longest time ,  then
analysis each query.
 * the key point is that how can we sort by hours.
 * first sort each server,  the sort each server begin point.
Just traverse, the first one, the first a end, if there's another b, then
temptime = bstat-astat
And then cycle the next one as well.
If not, then maxtime = aend-astat;
Recycling b. The algorithm should be nlogn's
 */
#include<string>
#include<iostream>
#include <utility>
#include<vector>
#include <map>
#include <algorithm>
#include <set>

#define SEC_PER_MIN 60
using namespace std;
class time{
public:
    time(int h,int m,int s); // have three para
    int hours;int minutes;int seconds;
};
time::time(int h, int m, int s):hours(h),minutes(m),seconds(s) {} //
constructor
//define smaller a earlier, return 1. b  earlier, return 0;
bool comp(const time &a, const time &b){  // equal time will return true,
We judge it earlier so we can link them.
    if (a.hours > b.hours)
```

```
        return false;             // bigger is later.
    else  if (a.hours < b.hours) {
            return true;
        }
    else{           //else if(a.hours == b.hours)
        if(a.minutes < b.minutes) {
            return true;
        } else if(a.minutes >b.minutes ){
            return false;
        }
        else{
            if (a.seconds >= b.seconds) {
                return false;// same time return false. such as sample
13:00:00
            }else  return true;
        }
    }
}
int timeSub(const time &a, const time &b){   // a should later than  b
    return
(a.hours-b.hours)*SEC_PER_MIN*SEC_PER_MIN+(a.minutes-b.minutes)*SEC_P
ER_MIN+(a.seconds-b.seconds);
}
class server{  //every pair
public:
    server(time start,time end);
    time start;
    time end;
    string name;
    bool operator< (const server &b) const {  // sort by the starting time
point ascending
        return comp(this->start, b.start); // true is earlier,  false;
    }
};
server::server(time start, time end):start(start),end(end) {} //init
list

int main() {
    int sec, min, hours, N, K, i;
    cin >> N >> K;
    string tempS;
    map<string, std::vector<time>> m1;//Save input in the map
    char dev;
    for (i = 0; i < N; i++) { // input all the time point
```

```
        cin >> tempS;cin >> hours;cin >> dev; //end when meet space
        cin >> min;cin >> dev;cin >> sec;// enter ':'
        std::vector<time> tmpV;
        time t1(hours, min, sec);// call the constructor to init the t1
        tmpV.clear();
        tmpV.push_back(t1);//record the hour, minute and seconds
        if (m1.empty()) { // if don't have any server, add directly,
            m1.insert(make_pair(tempS, tmpV)); //mapPerson.insert(pair <
int,string > (1,"Jim"));
        }//else compare.
        else { // find the map server then put the time into
            if (m1.count(tempS)  ) { // exist return 1 , else return 0
                m1.find(tempS)->second.push_back(t1);  // find returns an
iterator to it if found,otherwise it returns an iterator to map::end.
            } else {
                m1.insert(make_pair(tempS, tmpV));// finally we will sort
the vector in map
            }
        }
    }// it is difficult to change the vector in the map
    map<string, std::vector<time>>::iterator mapIt;// define iterator as
a global variable for change data
    std::vector<time> *vt;//pointer for change data
    for (mapIt = m1.begin(); mapIt != m1.end(); mapIt++) { //Sort the time
in the map
        vt = &mapIt->second;
        sort(vt->begin(), vt->end(), comp);//define a customer compare
rule
    }// sort finished, we can use iterator,notice we need change value
so don't use const iterator
    // sort default from small to big , sort finished in 12.28 23:26, we
can
//    for(auto it : m1){
//        vector<time>vt = it.second;
//        for (auto & it2 : vt){
//            cout << "hours: " << it2.hours << " min: " << it2.minutes <<
endl;
//        } //for debug
//    }// before there, is all  correct  10:58 12/30

    //first we Save server pair in the set
    set<server> ser1;// store all the time blocks, each server has
<start,end>
    for (auto it : m1) {// move all into set. they will be sorted
```

```
automatically.
        int count = it.second.size();// the number of one server's all time
point
        for (int i = 0; i + 1 < count; i += 2) {
            ser1.insert(server(it.second[i], it.second[i + 1]));// insert
into set
        }//Save server pair in the set
    }// before there, is all  correct  10:59 12/30
    //In fact, there is no need to distinguish between servers in the set,
we now save , may delete it in the future

    auto it = ser1.begin(); // traverse the set.
    set<server>::iterator itNext;
    if(!ser1.empty()) {
        itNext = std::next(it);// find next the beginning time .
    }
    else{
        cout <<"don't have any server record!!";
        return 0;
    }
    vector<int> blocks;//save each blocks
    if(itNext == ser1.end()){
        blocks.push_back(timeSub(it->end, it->start));
    }// only have one element;
    while (itNext != ser1.end()) { // traverse all blocks
        int qTmp = 0;// query tmp store current length of time.
        while (itNext != ser1.end() &&
               comp(itNext->start, it->end)) {  // if there is a blocks
"itNext" start before the "it" end,
            while (itNext != ser1.end() && comp(itNext->end, it->end)) { //
itNext end before it end,  cannot ignore
                blocks.push_back(timeSub(itNext->end, itNext->start) );//
the total number of valid time points for starting the task.
                itNext++; //traverse all "it contains "
            }// have processed all "it contains ".
            if (itNext != ser1.end() && comp(itNext->start, it->end))
{   // itNext can link after the it 这一段连起来的算完.
                qTmp += timeSub(itNext->start, it->start);// add then jump
out the itNext
                it = itNext;
                itNext = std::next(it);
            }
//            cout << "hour: " << it->start.hours << " min: " <<
it->start.minutes << endl;
```

```
//          cout << "itNext->start next hour: " << itNext->start.hours
<< " min: " << itNext->start.minutes << endl;
       }// if don't have any blocks can continue
       qTmp += timeSub(it->end, it->start);// finished the link
       blocks.push_back(qTmp);// push the link blocks
       if(itNext!= ser1.end()){
       it = itNext; itNext++;}// continue traverse
    }//
    sort(blocks.begin(), blocks.end());
    printf("%d\n", blocks.back());//output the the longest computation
task you could run
    // the index of vector , then we compare them with query  the length
of one link
    for(i = 0;i < K;i++ ){  // input the queries
       cin >> hours;cin >> dev; // enter ':'
       cin >> min;cin >> dev;cin >> sec;// enter ':'
       int QueryL = timeSub(time(hours,min,sec),time(0,0,0));
       int total = 0;
       for(int i : blocks){
          if(QueryL < i)
             total += i- QueryL+1;//can process the query
       }
       printf("%d\n",total);// output each query results
    }
    return 0;
}
```

Declaration

I hereby declare that all the work done in this project titled " Arrangement of Computation Tasks " is of my independent effort.