东南大学
网络空间安全学院

算法作业 #4

学生姓名: 李盛   学号: *229221*

Course: 算法设计与分析 – Professor: 陶军
Due date: *5* 月 *31* 日, *2022* 年

第一题

某军事基地通过密码锁进入，通过前期分析密码锁上的指纹，知道密码是由 4 个数字 0,6,8,9 组成。现可以通过入侵密码锁后的传输线路，来模拟密码的输入。请设计算法来实现密码的组合，给出输出格式。如用到回溯或分支限界方法，请画出解空间树。

a) Problem statement
Solution matrix $\mathbf{X} = (x_1, x_2, \ldots, x_n)$, where x_i represents i^{th} digit of the n-digit password and $x_i \in S_i = \{a_{i1}, a_{i2}, a_{i3}, a_{i4}\}$, where $a_{i1} = 0$, $a_{i2} = 6$, $a_{i3} = 8$, $a_{i4} = 9$ $\forall 1 \leq i \leq n$.
Pseudocode is stated as the following.

Algorithm 1 Password cracker using recursive backtracking

```
1: $X = \phi$
2: $x_1 = a_{11}$
3: if $X = (x_1)$ is the partial solution then
4:     continue to extend $x_2$ with $S_2$
5: else   Choose the next element of $S_1$ to be $x_1$
6: end if
7: Recursively repeat the above steps until cracking the password
```

b) 解空间树
Assume the password is "069", the path to find the password is shown as below.

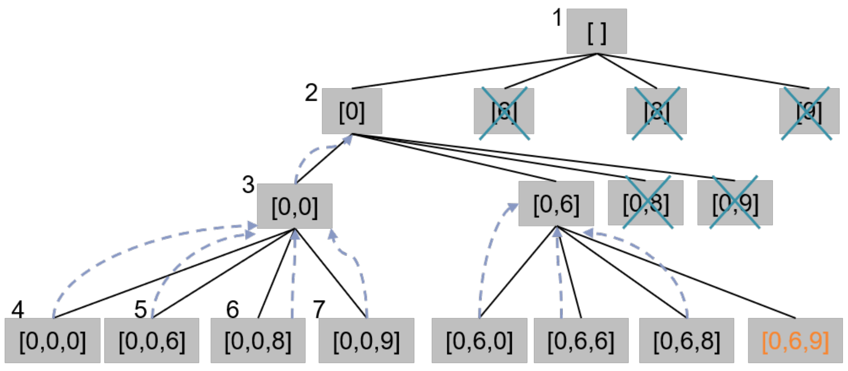
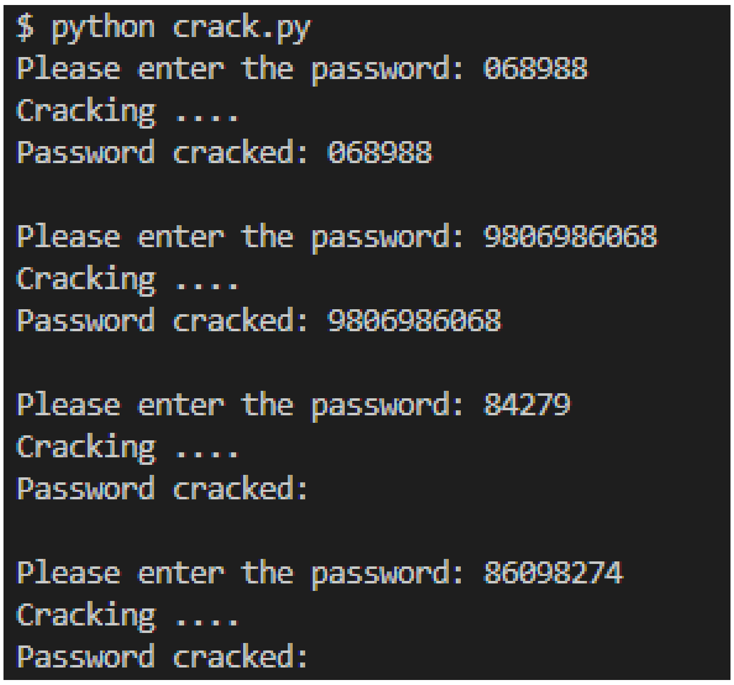


Figure 1: Solution tree of question 1

c) Implementation code

```
def login(password, PASSWORD):
    return password == PASSWORD

def crack(soFar, PASSWORD, maxLength):
    """ Password cracking
    Inputs:
    soFar          the current node
    PASSWORD       the secret password
    maxLength          maximum password length

    Return:
    password       the cracked password
    """
    if login(soFar, PASSWORD):
            return soFar

    if len(soFar) == maxLength:
            return ""

    for i in ['0','6','8','9']:
            password = crack(soFar + i, PASSWORD,
               maxLength)
            if (password != ""):
                    return password

    return ""

if __name__ == "__main__":
    while True:
        PASSWORD = input('Please enter the password: ')
        result = crack("", PASSWORD, 10)
        print('Cracking ....')
        print('Password cracked: {}\n'.format(result))
```

d) The test case is shown as Figure 2. We find the corresponding password if its format is correct and return the empty string if the password consists of any illegal number or exceeds the maximum length of 10.



```
$ python crack.py
Please enter the password: 068988
Cracking ....
Password cracked: 068988

Please enter the password: 9806986068
Cracking ....
Password cracked: 9806986068

Please enter the password: 84279
Cracking ....
Password cracked:

Please enter the password: 86098274
Cracking ....
Password cracked:
```

Figure 2: Test cases of question 1

第二题

> 某地区军事冲突中，A 军需要进行阵地（N 个阵地）的物资补给，包括：弹药，谷物、肉罐头和饮用水等几大类物资。由于战场补给需要讲究效率，因此已经把这些物资按照每个种类的不同数量进行了配给，装入了 N 个统一大小集装箱中，有的弹药多一些，有的食物多（但是没法再开箱调整）。每个阵地会将自己的需求报给后勤部门。现要求为后勤部门设计算法，将这 N 个集装箱空投到 N 个阵地上，使得各阵地的匹配度之和最高。

a) 证明该问题是 NP-完全问题。

b) 请对阵地的匹配度进行建模，合理地反映收到的集装箱与该阵地物资需求的匹配程度。

c) 基于该匹配度模型（函数），以匹配度之和最高为目标，设计军需物资分配算法。

a) Our problem of interest is an optimization problem, in which each feasible solution(i.e. N containers' allocation to N battle fields) has an associated value, and we wish to find a feasible solution with the best matching value. NP-completeness applies directly not to optimization problems, however, but to decision problems, in which the answer is simply “yes” or “no” . In our case, a decision problem related is: given a allocation plan, and a decimal k, does the corresponding inverse matching score exists at most k. we can provide evidence that this decision problem is “hard” , we could also therefore provide evidence that its related optimization problem is “hard” . Thus, even though it restricts attention to decision problems, the theory of NP-completeness often has implications for optimization problems as well.

b) We define the following demand matrix $\mathbf{D}$ to represent material needs for N battle fields.

$$\mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1N} \\ d_{21} & d_{22} & \dots & d_{2N} \\ \dots & \dots & \dots & \dots \\ d_{M1} & d_{M2} & \dots & d_{MN} \end{bmatrix}$$

where column j encodes the needs information (ammo d_{1j}, grain d_{2j}, water d_{3j} ... amount to M materials) of j^{th} battle field.

Accordingly, we could define the following military supply matrix $\mathbf{S}$ to represent materials allocation for N battle fields.

$$\mathbf{S} = \begin{bmatrix} s_{11} & s_{12} & \dots & s_{1N} \\ s_{21} & s_{22} & \dots & s_{2N} \\ \dots & \dots & \dots & \dots \\ s_{M1} & s_{M2} & \dots & s_{MN} \end{bmatrix}$$

where column j encodes the allocation information (ammo s_{1j}, grain s_{2j}, water s_{3j} ... amount to M materials) of j^{th} container.

The matching score m_{ij} between the i^{th} battle field and j^{th} container is modeled as the normalized cross correlation between the i^{th} column of $\mathbf{D}$ and j^{th} column of $\mathbf{S}$.

c) We define the i^{th} column of $\mathbf{D}$ as $\overrightarrow{d_i} = [d_{1i}, d_{2i}, \dots, d_{Mi}]^T$ and the j^{th} column of $\mathbf{S}$ as $\overrightarrow{s_j} = [s_{1j}, s_{2j}, \dots, s_{Mj}]^T$. The matching score function m_{ij} could be written as:

$$m_{ij} = \frac{\overrightarrow{d_i} \cdot \overrightarrow{s_j}}{\| \overrightarrow{d_i} \| \| \overrightarrow{s_j} \|}$$

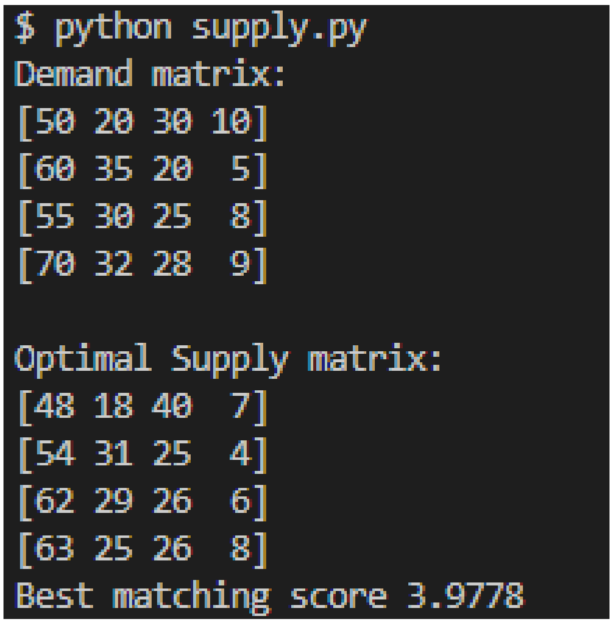
We could generate all unique permutations of $\mathbf{S}$. Therefore, the overall matching score $m = \sum_{i=1}^{N} m_{ii}$

d) Since this problem is proved as NP-complete, we would then do better to spend our time solving a tractable special case, rather than searching for a fast algorithm that solves the problem exactly. The code is implemented as following.

```
import numpy as np
import itertools

D = np.array([[50,20,30,10],[60,35,20,5],\
              [55,30,25,8],[70,32,28,9]])
S = np.array([[63,25,26,8],[54,31,25,4],\
              [62,29,26,6],[48,18,40,7]])
ms_max = 0
for s in itertools.permutations(S):
    ms = 0
    for i in range(4):
        di = D[i]
        si = s[i]
        d_norm = np.linalg.norm(di)
        s_norm = np.linalg.norm(si)
```

```
                ms += np.dot(di, si)/(d_norm*s_norm)
            if ms > ms_max:
                ms_max = ms
                optimal_S = s

    print('Demand matrix:')
    print('\n'.join(str(row) for row in D))
    print('')
    print('Optimal Supply matrix:')
    print('\n'.join(str(row) for row in optimal_S))
    print('Best matching score {:.4f}'.format(ms_max))
```

e) We initialize the demand and supply matrix as D and S respectively and the test result is shown as Figure 3.



```
$ python supply.py
Demand matrix:
[50 20 30 10]
[60 35 20  5]
[55 30 25  8]
[70 32 28  9]

Optimal Supply matrix:
[48 18 40  7]
[54 31 25  4]
[62 29 26  6]
[63 25 26  8]
Best matching score 3.9778
```

Figure 3: Test result of question 2

第三题

第二次世界大战期间，美国派遣一个由若干架轰炸机和战斗机组成的战斗机群前往欧洲战场执行战斗任务。考虑到飞机之间必须保持一定的间距以防止意外撞机，指挥部必须实时获得机群中两机之间的最短距离。假设机群处于同一个二维水平面，安全距离为 100 米。指挥部现已获得各机的 GPS 坐标，请聪明的你设计一个算法，尽可能快的帮指挥部判断是否存在危险可能。

a) 请用分治法分析相应的时间复杂度；

b) 请在 O(n) 时间复杂度下求解该问题，给出伪代码及具体的代码实现。（提示：可以考虑随机算法）

Solution.

a) Pseudocode with divide and conquer algorithm

Algorithm 2 Find the closest pair of points using divide and conquer algorithm

```
function CLOSESTPAIR($i, j, Q$)
    $n = j - i + 1$;
    if $n < 3$ then
        return $ID\_1, ID\_2, distance$
    end if
    $Q_L = Q[i : i + \lceil n/2 \rceil - 1]$
    $Q_R = Q[i + \lceil n/2 \rceil : j]$
    $(ID\_1, ID\_2, distance)$=closestPair($i, i + \lceil n/2 \rceil - 1, Q_L$)
    $(ID\_1, ID\_2, distance)$=closestPair($i + \lceil n/2 \rceil, j, Q_R$)
    Keep record of the minimum of $distance, ID\_1, ID\_2$
    根据鸽巢原理：Yd= 落在临界区的某个点，Yd 与前 6 个与后 6 个计算距离，更新最小值 d 和 id。
    return $ID\_1, ID\_2, distance$
end function
```

1) Define Q as the collection of sets of all planes' coordinates and ID number.

2) As a pre-processing step, the input array is sorted according to x coordinates.

3) Divide and conquer
计算 Q 中各点 x-坐标的中位数 m，用垂线 x=m 划分成两个大小相等的子集合 L 和 R，L 中点在 x=m 左边，R 中点在 x=m 右边.

4) Recursion
Compare $(p1, p2) \in L, (q1, q2) \in R$, and compute the minimum distance.

5) Critical point
在临界区查找距离小于 d 的点对 (p_l, q_r)，$p_l \in L, q_r \in R$;
如果找到，则 (p_l, q_r) 是 Q 中最接近点对；
否则 (p_1, p_2) 和 (q_1, q_2) 中距离最小者为 Q 中最接近点对；

Time complexity analysis:
$T(n) = O(1) \quad n < 3$
$T(n) = 2T(n/2) + O(n)\ n \geq 3$
Apply Master method, we have:
$T(n) = O(nlog(n))$

b) Pseudocode with randomized algorithm

Algorithm 3 Find the closest pair of points with randomized algorithm

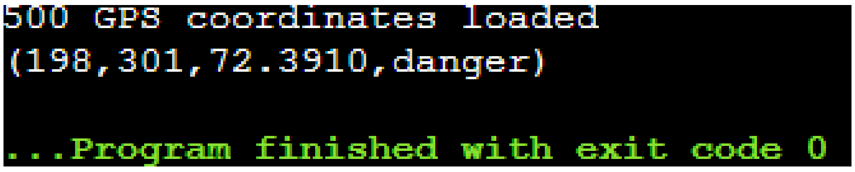
```
1: 读取 csv 文件，构造点集 S
2: 点集 $S(|S| = n)$ 中随机取子集 T，使得 $|T| = \lfloor\sqrt{n}\rfloor$
3: T 中点两两计算距离，求出 T 中最近点对距离 $\delta(T)$
4: 以 $\delta(T)$ 为尺寸构造网格
5: 找到一个略小于 $c \times n$ 的素数 p，其中 c 为常数
6: 用散列函数 $H(i, j, p)$ 把 S 中的点存至长为 p 的散列表，$(i, j)$ 表示方格坐标
7: 散列表中找到点数最多的方格，$(i, j)$, 若点数 $\leq \sqrt{n}$, 则计算方格，$(i, j)$ 中最近点对
   距离 ，否则，将方格 $(i, j)$ 一分为四，点数最多的字方格若还 $> \sqrt{n}$, 继续拆分直至
   点数 $\leq \sqrt{n}$, 计算最后拆分点数最多的方格的最近点距
8: 以 $\delta$ 为尺寸构造网络
9: 最近点对必落在某个 $2\delta \times 2\delta$ 的方格中，计算所有 $2\delta \times 2\delta$ 的方格中点对距离，找
   到最小值
10: 比较最小值与距离阈值判断是否安全
```

c) Implementation code

```
#include <iostream>
#include <vector>
#include <sstream>
#include <cmath>
#include <time.h>
#include <fstream>

#define pi 3.1415926
using namespace std ;
const double R = 6371393;
//平面中点的结构体：包含id、经度、纬度;
struct Spot {
    string id ;
    double lat ;
    double lon ;
} ;
//边界的结构体，分别是x、y的最大最小值，即平面中的点的左右上
   下的最值;
struct boundryLimitation {
    double xLeftLimit ;
    double xRightLimit ;
    double yButtomLimit ;
    double yTopLimit ;
} ;
//输出结果的结构体，包含两个点的id、两个点的距离、安全与否；
struct Res {
    string idOne ;
    string idTwo ;
    double distance ;
    bool safeOrNot ;
} ;
//节点的结构体，该节点中，包含该节点存储的平面中的点，以及指
   向下一个节点的指针；
struct Node{
    Spot data ;
    Node* next ;
    Node(){ next = nullptr ; }
} ;
//存着头节点的链表
struct NodeList {
    Node* head ;
} ;

//getSpotList方法：通过csv文件初始化链表，链表存储着包含csv
   文件中所有的点的节点。
vector<Spot> getSpotList(){
    vector<Spot> spotlist ;
    ifstream fin("/Users/sufen/1.csv",ios::in);
    string currentRow;
    getline(fin,currentRow);
    while(getline(fin,currentRow)){
        istringstream readstr(currentRow);
        Spot currentSpot ;
        getline(readstr,currentSpot.id,',');
        string tmpStr;
        getline(readstr,tmpStr,',');
        istringstream streamOne(tmpStr);
        streamOne >> currentSpot.lat;
        getline(readstr ,tmpStr,',');
        istringstream streamTwo(tmpStr);
        streamTwo >> currentSpot.lon ;
        spotlist.push_back(currentSpot);
    }
    return spotlist ;
}
//弧度与角度转化：一个圆的弧度是2pi 而一个圆又是360度（角度
   制），故pi=180度；
double degreeToRadian (double degree){
    return degree*pi/180.0;
}
```

```
//通过平面中两点的经纬度值(坐标),计算两点间的距离(米);
double getDistance (double latOne,double lonOne,double
   latTwo,double lonTwo) {
    double latOneForRadian = degreeToRadian(latOne) ;
    double latTwoForRadian = degreeToRadian(latTwo);
    double tmpOne = latOneForRadian−latTwoForRadian ;
    double tmpTwo = degreeToRadian(lonOne)−degreeToRadian(
       lonTwo) ;
    double tmpThr = 2*R*asin(sqrt(pow(sin(tmpOne/2) ,2)+ cos(
       latOneForRadian)*cos(latTwoForRadian)*pow(sin(tmpTwo
       /2) ,2)));
    return tmpThr;
}

//getBoundryLimitation方法,输入存储节点的数组,返回该数组中
   所有节点的上下左右边界的最值,以boundryLimitation结构体的
   形式返回。
boundryLimitation getBoundryLimitation(vector<Spot> spotlist
   ){
    double xLeftLimit = spotlist[0].lat;
    double xRightLimit = spotlist[0].lat;
    double yButtomLimit = spotlist[0].lon ;
    double yTopLimit = spotlist[0].lon ;
    for(int i =1; i<spotlist.size(); i++){
        if(spotlist[i].lat<xLeftLimit) {
            xLeftLimit = spotlist[i].lat;
        }
        if(spotlist[i].lat>xRightLimit) {
            xRightLimit = spotlist[i].lat;
        }
        if(spotlist[i].lon <yButtomLimit) {
            yButtomLimit = spotlist[i].lon ;
        }
        if(spotlist[i].lon >yTopLimit) {
            yTopLimit = spotlist[i].lon ;
        }
    }
    boundryLimitation res= boundryLimitation {};
    res.xLeftLimit = xLeftLimit ;
    res.xRightLimit = xRightLimit ;
    res.yTopLimit = yTopLimit ;
    res.yButtomLimit = yButtomLimit ;
    return res;
}

//getMinDistance方法,输入为存储着节点的数组,返回该数组中所
   有节点中两两节点的最小值;
double getMinDistance(vector<Spot> spotlist){
    double minDistance = INT_MAX;
    string idOne ;
    string idTwo ;
    for(int i=0;i<spotlist.size(); i++){
        for(int j=i+1; j<spotlist.size(); j++){
            double distance = getDistance(spotlist[i].lat,
               spotlist[i].lon,spotlist[j].lat, spotlist[j].
               lon);
            if(distance<minDistance){
                minDistance = distance;
            }
        }
    }
    return minDistance;
}

//listAddLast方法，输入链表和节点，将节点插入链表末尾;
int listAddLast(NodeList * nodelist , Node *node){
    if(nodelist->head == nullptr) {
        nodelist->head = node ;
    } else {
        Node* cur=nodelist->head ;
        while(cur->next != nullptr) {
            cur = cur->next ;
        }
        cur ->next = node ;
    }
    return 0 ;
}

double hashForDelta(int p , vector<Spot> spotlist , double
   xLeftLimit , double yButtomLimit , double delta){
    //res数组：该数组的大小为p，用于存放p个链表;
    vector<NodeList*> res(p) ;
    //初始化res数组，把里面的p个链表的头节点都指向nullptr;
    for(int i=0;i<p;i++){
        res[i]= new(NodeList);
        res[i]->head =nullptr;
    }
    //countNumber数组：存放int值，数组容量为p;
    int countNumber[p];
    //初始化countNumber数组，把里面的值全初始化为0;
    for(int i =0;i<p;i++){
        countNumber[i]=0;
    }
    for(int i=0;i<spotlist.size();i++){
        int x = int (ceil((spotlist[i].lat-xLeftLimit
           +0.0000000001) / delta));
        int y = int (ceil((spotlist[i].lon-yButtomLimit
            +0.000000001) / delta));
        Node* node =new(Node) ;
        node->data = spotlist[i];
        node->next =nullptr;
        countNumber [(x*y)%p]++;
        listAddLast(res[(x*y)%p], node);
    }
    int idWhenMax = 0 ;
    int maxValue = 0 ;
    for(int i=0;i<p;i++){
        if(maxValue < countNumber[i]) {
            idWhenMax = i ;
            maxValue = countNumber[i];
        }
    }

    int size = spotlist.size();
    double newDelta=0;
    vector<Spot> tmp;
    Node* cur = res[idWhenMax]->head ;
    while(cur !=nullptr) {
        tmp.push_back(cur->data);
        cur = cur->next ;
    }

    while(tmp.size()> sqrt(size)){
        vector<Spot> tmpOne ;
        vector<Spot> tmpTwo ;
        vector<Spot> tmpThr ;
        vector<Spot> tmpFour ;
        boundryLimitation boundry = getBoundryLimitation(tmp
            );
        double midx =(boundry.xRightLimit+boundry.xLeftLimit
            ) /2;
        double midy =(boundry.yTopLimit+boundry.yButtomLimit
            ) /2;
        for(int i=0; i<tmp.size();i++){
            if(tmp[i].lon > midy && tmp[i].lat >midx){
                tmpFour.push_back(tmp[i]);
            }
            if(tmp[i].lat<midx && tmp[i].lon >midy){
                tmpThr.push_back(tmp[i]);
            }
            if(tmp[i].lat<midx && tmp[i].lon <midy){
                tmpOne.push_back(tmp[i]);
            }
            if(tmp[i].lat> midx && tmp[i].lon <midy){
                tmpTwo.push_back(tmp[i]);
            }
        }
        tmp.clear();
        if(tmp.size() < tmpOne.size()){
            tmp = tmpOne ;
        }
        if(tmp.size() < tmpTwo.size()){
            tmp = tmpTwo ;
        }
        if(tmp.size()< tmpThr.size()){
            tmp = tmpThr ;
        }
        if(tmp.size()< tmpFour.size()){
            tmp = tmpFour ;
        }
    }
    newDelta = getMinDistance(tmp);
    //返回方格中最近点距newDelta;
    return newDelta ;
}


Res deltaForMinDistansce(int numRow, int numCol , double
   delta , vector<Spot> spotlist , double xLeftLimit , double
    yButtomLimit) {
    vector <vector<NodeList *>> gridRanks(numRow - 1);
    for (int i = 0; i < numRow - 1; i++) {
        gridRanks[i].resize(numCol - 1);
    }
    for (int i = 0; i < numRow - 1; i++) {
        for (int j = 0; j < numCol - 1; j++) {
            gridRanks[i][j] = new(NodeList);
            gridRanks[i][j]->head = nullptr;
        }
    }
    for (int i = 0; i < spotlist.size(); i++) {
        int x = int(ceil((spotlist[i].lat - xLeftLimit +
           0.0000000001) / delta) - 1);
        int y = int(ceil((spotlist[i].lon - yButtomLimit +
           0.000000001) / delta) - 1);
        if (x > 0 && y > 0) {
            Node *node = new(Node);
            node->data = spotlist[i];
            listAddLast(gridRanks[x - 1][y - 1], node);
        }
        if (x > 0 && y < numCol - 1) {
            Node *node = new(Node);
            node->data = spotlist[i];
            listAddLast(gridRanks[x - 1][y], node);
        }
        if (x < numRow - 1 && y > 0) {
            Node *node = new(Node);
            node->data = spotlist[i];
            listAddLast(gridRanks[x][y - 1], node);
        }
        if (x < numRow - 1 && y < numCol - 1) {
            Node *node = new(Node);
            node->data = spotlist[i];
            listAddLast(gridRanks[x][y], node);
        }
    }
    double minDistance = INT_MAX;
    string idOne;
    string idTwo;
    for (int i = 0; i < numRow - 1; i++) {
        for (int j = 0; j < numCol - 1; j++) {
            Node *cur = gridRanks[i][j]->head;
            vector <Spot> tmpVectorList;
            while (cur != nullptr) {
                tmpVectorList.push_back(cur->data);
                cur = cur->next;
            }
            for (int k = 0; k < tmpVectorList.size(); k++) {
                for (int l = k + 1; l < tmpVectorList.size()
                   ; l++) {
                    double distance = getDistance(
                       tmpVectorList[k].lat, tmpVectorList[k
                       ].lon, tmpVectorList[l].lat,
                                                    tmpVectorList
                                                       [l].lon)
                                                       ;
                    if (distance < minDistance) {
                        minDistance = distance;
                        idOne = tmpVectorList[k].id;
                        idTwo = tmpVectorList[l].id;
                    }
                }
            }
        }
    }
    Res res = Res{};
    res.idOne = idOne;
    res.idTwo = idTwo;
    res.distance = minDistance;
    if (minDistance < 100) {
        res.safeOrNot = false;
    } else {
        res.safeOrNot = true;
    }
    return res;
}

int main() {
    // 通过getSpotList方法获取平面中点的数组spotlist，该数组
       中存储着Spot;
    vector<Spot> spotlist = getSpotList() ;
    //getSpotList方法：通过csv文件初始化链表，链表存储着包含
       csv文件中所有的点的节点。
    int n = spotlist.size();
    int nForSqrt = int(round(sqrt(n)-1)) ;
    vector<Spot> randomSpotList ;
    srand(time(0));
    // 点集spotlist(n个点)中随机取子集randomSpotList，使得|
       randomSpotList|=√n;
    while(randomSpotList.size()<nForSqrt){
        int i = rand()%n ;
        int numOfRand = randomSpotList.size();
        bool tmpBool = true ;
        for(int j=0; j<numOfRand ; j++){
            if(randomSpotList[j].id == spotlist[i].id){
                tmpBool = false ;
            }
        }
        if(tmpBool){
            randomSpotList.push_back(spotlist[i]);
        }
    }

    const int c = 5 ;
    int p=0;
    for(int i=c*n;i>1;i--){
        bool tmpBool = true ;
        for(int j=2;j<i;j++){
            if(i%j==0){
                tmpBool=false ;
                break ;
            }
        }
        if(tmpBool){
            p = i ;
            break ;
        }
    }
    double newDelta = hashForDelta (p,spotlist,boundry.
        xLeftLimit,boundry.yButtomLimit,delta)/111000;
    int numRow = int(ceil((boundry.xRightLimit−boundry.
        xLeftLimit) /newDelta));
    int numCol = int(ceil((boundry.yTopLimit−boundry.
        yButtomLimit) /newDelta));
    Res res=deltaForMinDistansce(numRow,numCol,newDelta,
        spotlist,boundry.xLeftLimit,boundry.yButtomLimit) ;
    cout << "(" << res.idOne << ",";
    cout << res.idTwo << ",";
    cout << res.distance << ",";
    if(res.safeOrNot){
        cout << "safe" << ")" << endl;
    } else {
        cout << "danger" << ")" << endl;
    }
}
```

d) Test case result: load "1.csv" to test the correctness of our algorithm.



```
500 GPS coordinates loaded
(198,301,72.3910,danger)

...Program finished with exit code 0
```

Figure 4: Test result of question 3