

**课 程 实 验 报 告**

**课程名称： 算法设计与分析**

**专业班级： CS1503班**

**学 号： U201514559**

**姓 名： 周铭昊**

**指导教师： 华多强**

**报告日期： 2018年1月10日**

**计算机科学与技术学院**

# 目 录

[实验一 最近点对问题 1](#_Toc419182939)

[1.1 实验内容与要求 1](#_Toc419182940)

[1.2 算法设计 1](#_Toc419182941)

[1.3 实验结果与分析 2](#_Toc419182942)

[1.4 编程技术与方法 3](#_Toc419182943)

[1.5 源程序及注释 3](#_Toc419182944)

### 实验一 最近点对问题

### 实验内容与要求

最近点对问题的算法实现主要针对平面上的二维点，采用欧几里得距离。

输出哪些点对是最近点对

输出这些最近点对的距离是多少

### 1.2 算法设计

采用分治法，求最近点对。

先把n个点按照x坐标升序排列，找到一个中数mid。如果n<=3，直接计算所有点对的距离进行比较得出最短距离d并且保存最近点对。否则根据mid把所有点划分为左右两个部分：(0~mid)和(mid+1~n)，分别计算出这两个部分中的最短点对距离d1、d2，取d1和d2的最小值为d，在n个点中把所有与mid点的横坐标距离超过d的点排除(因为它们之间的最短距离一定比d要长)，剩余的点再按照y坐标升序排序，如果其中某两个点y轴距离大于d，则不计算(筛选掉距离一定比d大的点对)，直接计算下一对，直到所有点对都进行比较得到最短距离d3，如果d3<=d，则更新（或者添加）新的最近点对。

### 1.3 实验结果与分析

对于测试样例如图1.1所示的点，程序运行结果如图1.2所示，可知五个点连起来为十字架形，因此程序运行正确。

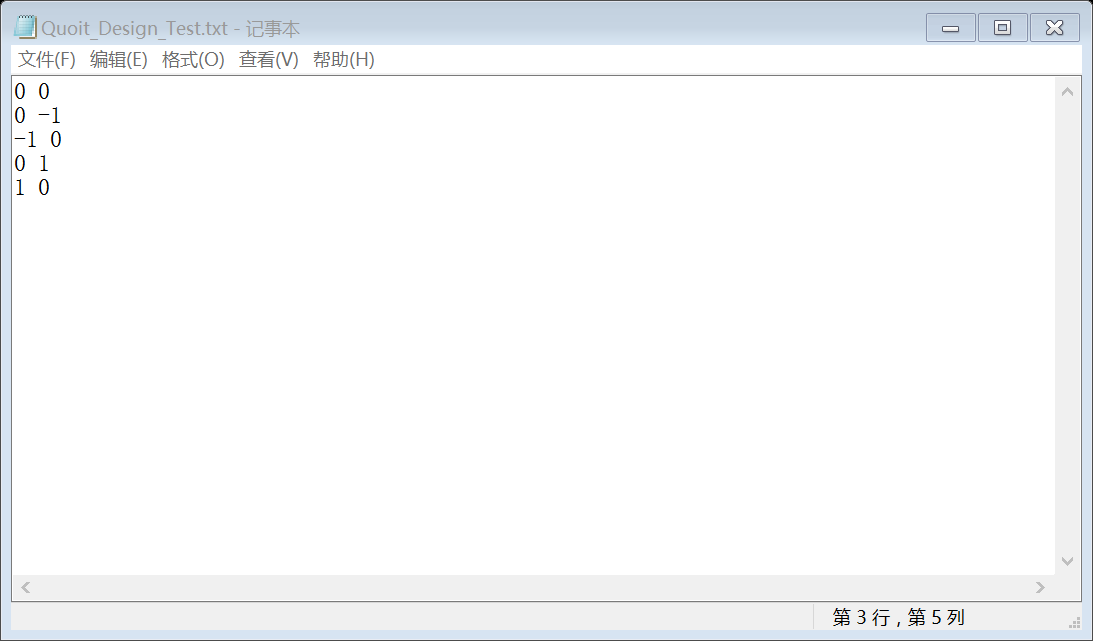
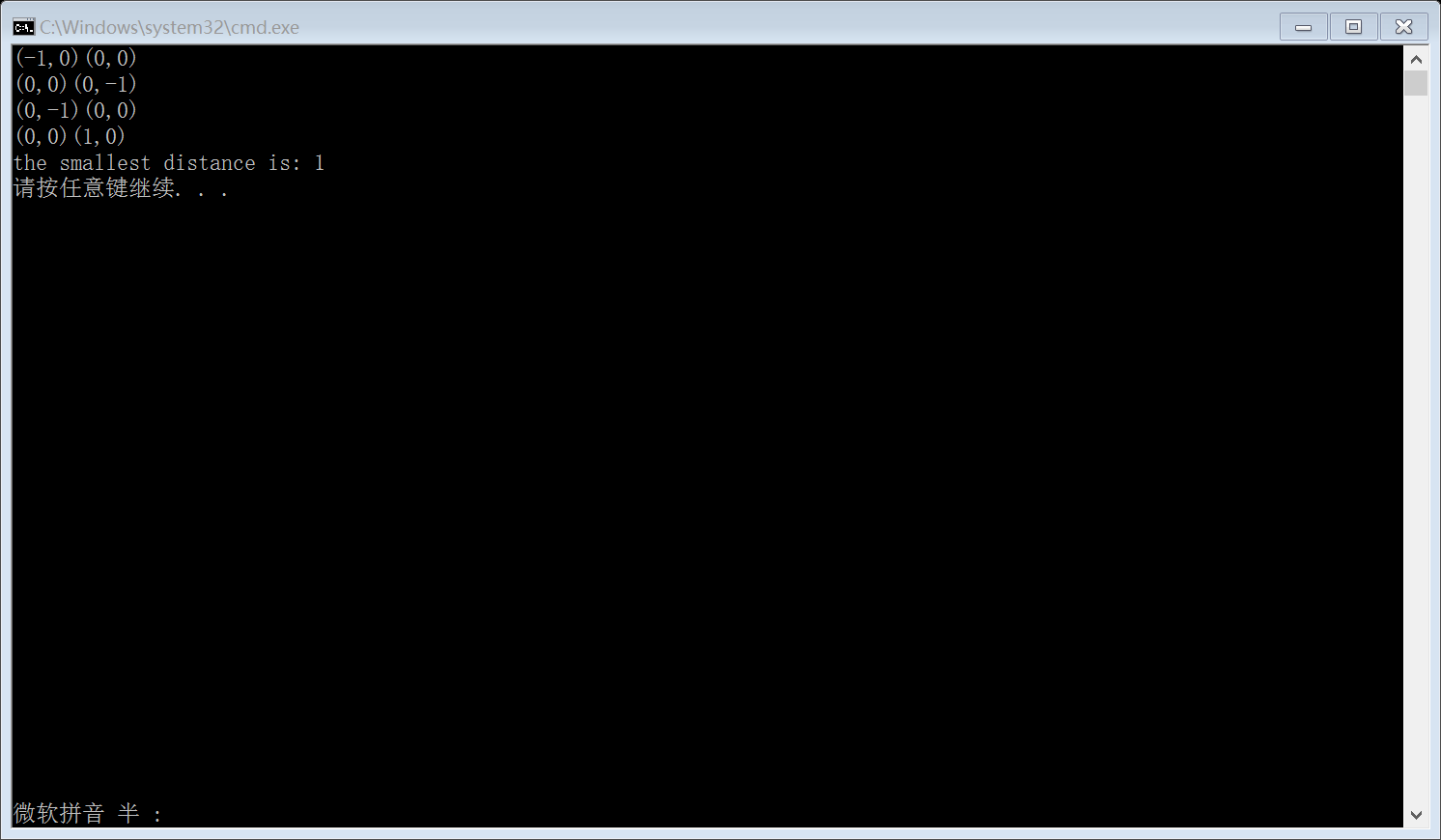
 

图1.1 测试样例1 图1.2 测试结果1

对于测试样例如图1.3所示的点，程序运行结果如图1.4所示，程序运行正确。

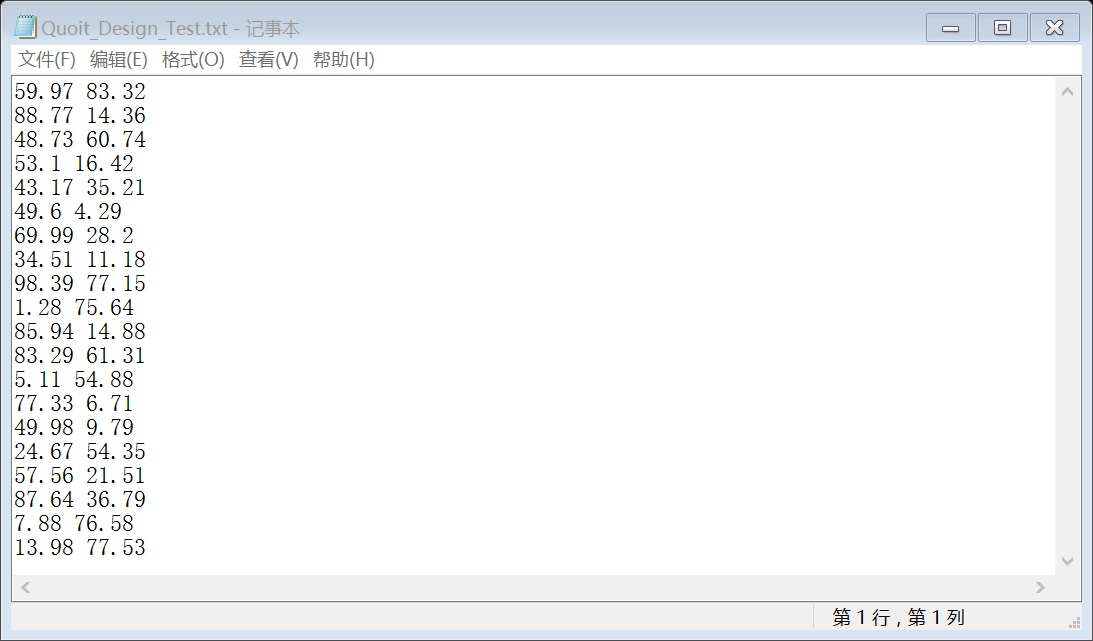
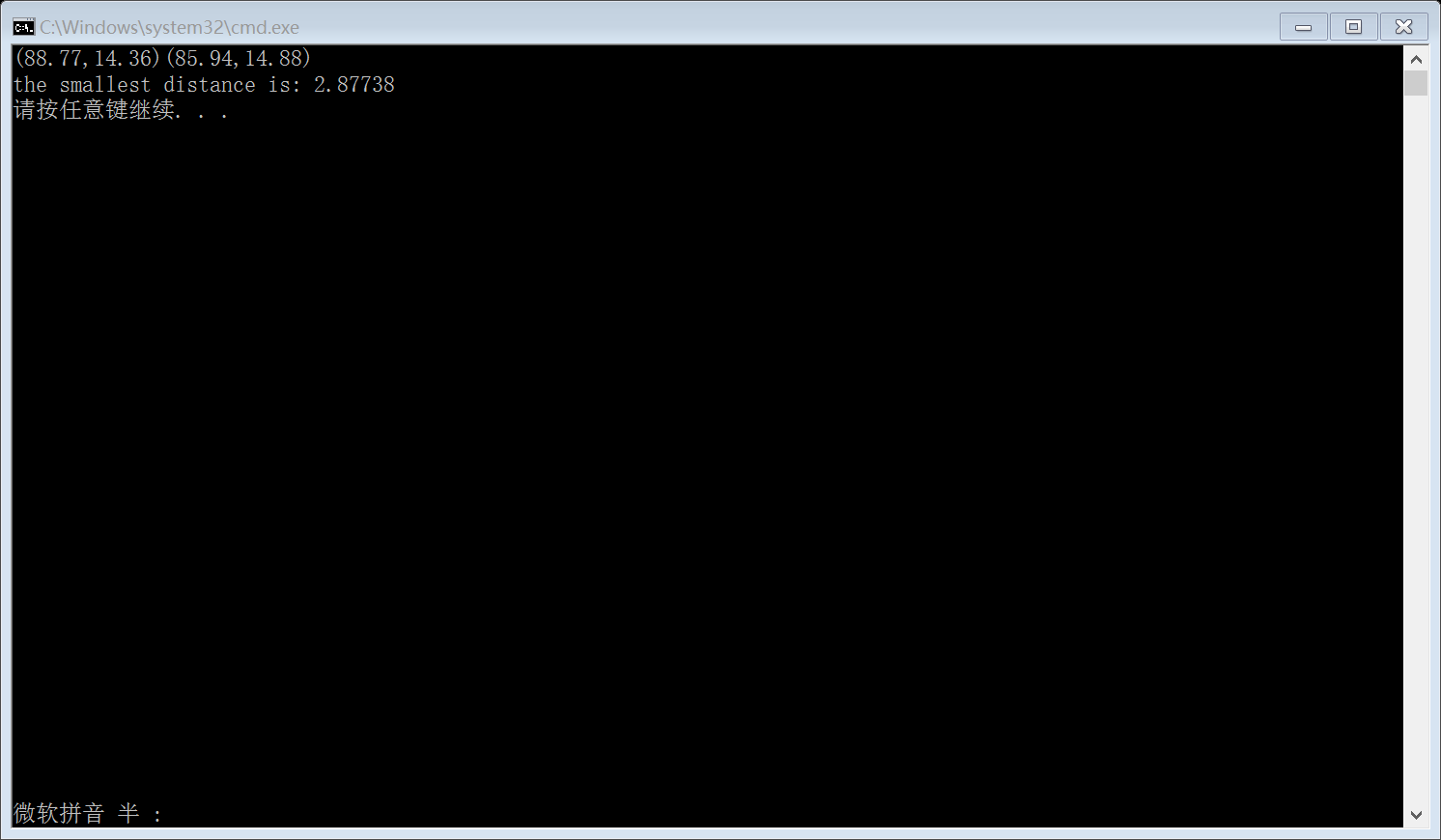
 

图1.3 测试样例2 图1.4 测试结果2

### 1.4 编程技术与方法

定义一个点的结构体MyPoint用于保存点的x、y坐标，再定义一个类MyPair用于保存所有最近点对信息和最短距离dis以当前距离的最近点对的对数num。

在MyPair类中定义两个函数：NewDis和MorePoint。前者用于更新新的最短距离和点对信息，后者用于增加相同最短距离的点对信息。

排序的方式采用库中函数sort进行排序，定义两种排序方式，一种是按x升序，一种是按y升序。

所有测试用例通过文件形式获取，便于修改测试多种不同例子。样例保存路径为D:\\Quoit\_Design\_Test.txt

为了更方便的操作最近点对类MyPair中的内容，定义一个函数compare\_p，输入两个点a，b和它们之间的距离d，如果MyP中已经存在这组点对则跳过，如果d<当前保存的最短距离dis，则调用NewDis函数更新最近点对，如果d=当前保存的最短距离dis，则调用MorePoint函数新添点对。最终输出MyP中保存的所有点对和最短距离。

遇到的一个问题是有一个函数命名为compare编译不通过，而且报错挺奇怪的，修改成compare\_p以后就可以了.

### 1.5 源程序及注释

#include <iostream>

#include <sstream>

#include <fstream>

#include <algorithm>

#include <math.h>

#define MAX 20

using namespace std;

struct MyPoint

{

double x,y;

}p[MAX];

class MyPair//save the closest MyP(may have more than one MyP)

{

public:

MyPoint p1[MAX];

MyPoint p2[MAX];

int num;

double dis;

public:

MyPair()

{

num = 0;

dis = 99999999;

};

~MyPair(){};

void NewDis(MyPoint a,MyPoint b,double d)//reload the new pair and delete the old

{

num=0;

dis=d;

p1[num].x=a.x;

p1[num].y=a.y;

p2[num].x=b.x;

p2[num].y=b.y;

num++;

return;

};

void MorePoint(MyPoint a,MyPoint b)//new pair with the same distance

{

p1[num].x=a.x;

p1[num].y=a.y;

p2[num].x=b.x;

p2[num].y=b.y;

num++;

return;

};

}MyP;

bool cmpx(MyPoint a,MyPoint b)//sort according to x

{

return a.x < b.x;

}

bool cmpy(MyPoint a,MyPoint b)//sort according to y

{

return a.y < b.y;

}

double MyDistance(MyPoint a,MyPoint b)//caculate the diatance of two MyPoint

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

void compare\_p(MyPoint a,MyPoint b,double d)

{

for(int i=0;i<MyP.num;i++)

if((MyP.p1[i].x==a.x)&&(MyP.p1[i].y==a.y)&&(MyP.p2[i].x==b.x)&&(MyP.p2[i].x==b.x))

return;

if(d<MyP.dis)

MyP.NewDis(a,b,d);

else if(d==MyP.dis)

MyP.MorePoint(a,b);

return;

}

double solve(int low,int high)//core

{

double d;

if(high-low==0)//only one point

{

return 0;

}

if(high-low==1)//two points

{

d = MyDistance(p[low],p[high]);

compare\_p(p[low],p[high],d);

return d;

}

if(high-low==2)//three points

{

double d1,d2,d3;

d1=MyDistance(p[low],p[low+1]);

d2=MyDistance(p[low+1],p[high]);

d3=MyDistance(p[low],p[high]);

if((d1<d2)&&(d1<d3))//d1 is the smallest

{

d=d1;

compare\_p(p[low],p[low+1],d);

}

if((d2<d1)&&(d2<d3))//d2 is the smallest

{

d=d2;

compare\_p(p[low+1],p[high],d);

}

if((d3<d1)&&(d3<d2))//d3 is the smallest

{

d=d3;

compare\_p(p[low],p[high],d);

}

if((d1==d2)&&(d1<d3))//d1 and d2 is smaller than d3

{

d=d1;

if(d<MyP.dis)

{

MyP.NewDis(p[low],p[low+1],d);

MyP.MorePoint(p[low+1],p[high]);

}

else if(d==MyP.dis)

{

MyP.MorePoint(p[low],p[low+1]);

MyP.MorePoint(p[low+1],p[high]);

}

}

if((d1==d3)&&(d1<d2))//d1 and d3 is smaller than d2

{

d=d1;

if(d<MyP.dis)

{

MyP.NewDis(p[low],p[low+1],d);

MyP.MorePoint(p[low],p[high]);

}

else if(d==MyP.dis)

{

MyP.MorePoint(p[low],p[low+1]);

MyP.MorePoint(p[low],p[high]);

}

}

if((d2==d3)&&(d2<d1))//d2 and d3 is smaller than d1

{

d=d2;

if(d<MyP.dis)

{

MyP.NewDis(p[low+1],p[high],d);

MyP.MorePoint(p[low],p[high]);

}

else if(d==MyP.dis)

{

MyP.MorePoint(p[low+1],p[high]);

MyP.MorePoint(p[low],p[high]);

}

}

return d;

}

else//more than three points

{

int mid = (low+high)/2;

double d1 = solve(low,mid);

double d2 = solve(mid+1,high);

double d3 = 99999999;

d = d1 < d2 ? d1 : d2;

int i,k=0;

MyPoint temp[MAX];

for(i = low; i <= high; i++)

{

if(fabs(p[mid].x-p[i].x) <= d)

temp[k++] = p[i];

}

sort(temp,temp+k,cmpy);

for(i=0;i<k;i++)

{

for(int j = i+1; j < k && temp[j].y-temp[i].y<=d; j++)

{

d3 = MyDistance(temp[i],temp[j]);

if(d3<=d)

{

d = d3;

compare\_p(temp[i],temp[j],d);

}

}

}

return d;

}

}

int main(void)

{

int i=0;

int n=0;

ifstream readFile("D:\\Quoit\_Design\_Test.txt");

while(!readFile.eof())//read file

{

readFile>>p[i].x>>p[i].y;

i++;

n++;

}

sort(p,p+n,cmpx);//sort according to x

solve(0,n-1);

for(i=0;i<MyP.num;i++)

{

cout<<"("<< MyP.p1[i].x <<","<<MyP.p1[i].y<<")";

cout<<"("<< MyP.p2[i].x <<","<<MyP.p2[i].y<<")"<<endl;

}

cout<<"the smallest distance is: "<< MyP.dis<<endl;

// for(i=0;i<n;i++)

// cout<<"("<< p[i].x <<","<<p[i].y<<")"<<endl;

// cout << n << endl;

//cout << "over" << endl;

return 0;

}

### 实验二 大整数相乘

### 2.1 实验内容与要求

利用分治法设计一个计算两个n位的大整数相乘的算法，要求计算时间低于O(n2)。

大整数（big integer）：位数很多的整数，普通的计算机不能直接处理，如： 9834975972130802345791023498570345

对大整数的算术运算，显然常规程序语言是无法直接表示的。编程实现大整数的加、减、乘运算，需考虑操作数为0、负数、任意位等各种情况

要实现两个大整数的加法、减法和乘法

输入：两个大整数

输出：加法结果，减法结果和乘法结果

### 2.2 算法设计

首先对于符号位，直接取出，再分成++，+-，-+，--四种类型调用相应的加减和乘，因此这里只考虑正数的运算（0归为+类，只要没有-开头就是+）。

对于大整数相加，算法过程就是模拟手算，两个数从最低位开始相加，大于9则进一位，同时本位的数-10，每位的数等于下一位的进位加上两数对应位之和。采用字符串形式保存大整数，则第一个字符对应的是最低位，因此最后的运算结果要通过reverse函数进行倒转。

对于大整数相减，首先判断两个数是否相等，相等则直接输出0，否则把较大的数s1减去较小的数s2，同样模拟手算过程，两个数从最低位开始相减，如果s1的最低位s1[0]大于等于s2的最低位s2[0]，则运算结果的最低位即为s1[0]-s2[0]，否则s1要向高一位借位，这时候需要判断高一位是否为0，如果为0，则把这位赋值为9，再向更高一位借位，因为s1>s2，所以一定不会出现无位可借的情况，如此循环直到成功借位，运算结果的最低位即为s1[0]+10-s2[0]，每一位的运算步骤与最低位相同。最后判断在一开始是否将s1与s2的位置调转过，如果调转过则在结果前加上一个-号表示结果为负数。

对于大整数相乘，采用分治算法。把第一个大整数X(m位)分为A,B两个部分，第二个大整数Y(n位)分为C，D两个部分，A、C位低n/2位，B、D为高n/2位，因此最终结果X\*Y=（A\*10^(m/2)+B）\*( C\*10^(n/2)+D)，化简后得到另一种表达式，即X\*Y=A\*C\*10^(m/2+n/2)+A\*D\*10^(m/2)+B\*C\*10^(n/2)+B\*D，即分别处理A\*C,A\*D,B\*C,B\*D后，把他们的运算结果相加可以得到X\*Y的值。

### 2.3 实验结果与分析

对于测试样例100 -100，程序运行结果如图2.1所示，易知100+(-100)=0;100-(-100)=200,100\*(-100)=-10000，因此程序运行正确。

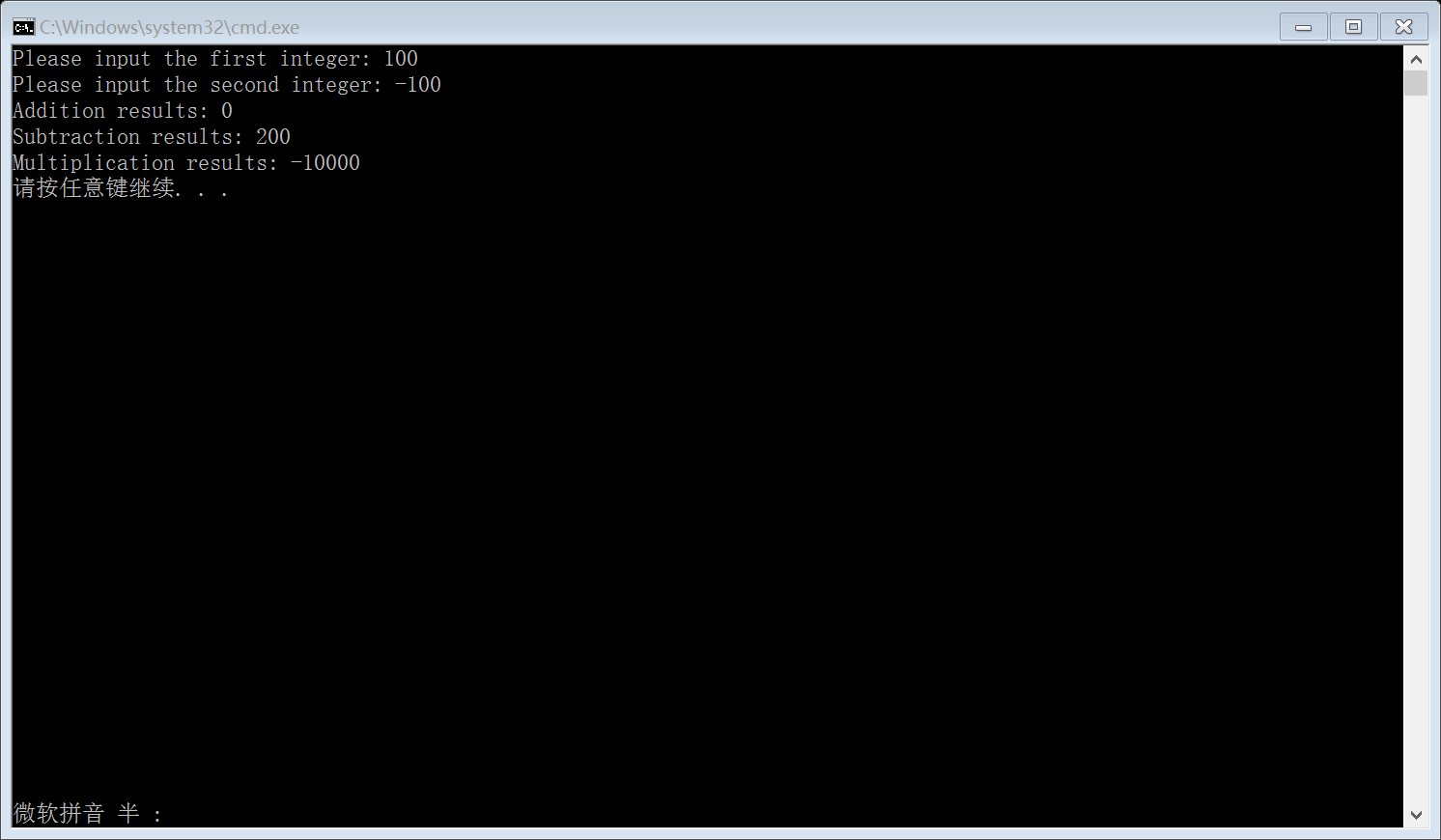


图2.1 测试结果1

对于测试样例123456789123456789和987654321987654321，程序运行结果如图2.2所示，可知程序运行正确。

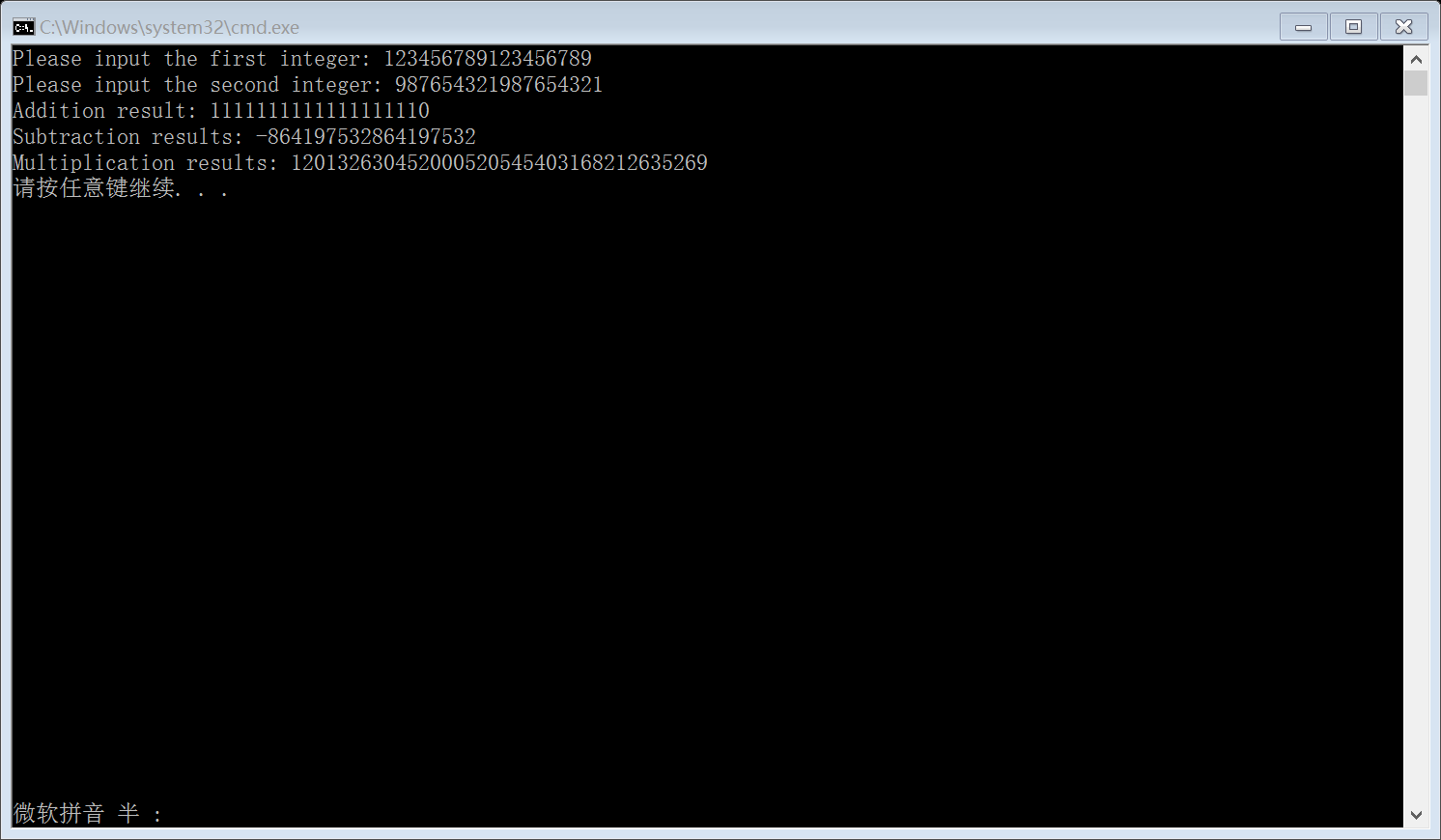


图2.1 测试结果2

### 2.4 编程技术与方法

通过字符串形式来存储大整数，判断输入的两个字符串第一个字符是否为“-”，再把符号位去除，调用对应的函数，例如s1为负数，s2为正数，那么加法结果输出为sub(s2,s1)，即s2-s1；减法结果输出为add(s1,s2)，即s1+s2，然后再把结果加上一个“-”号；乘法结果首先判断结果是否为0，如果不为0则在mul(s1,s2)的运算结果前加一个“-”号。

定义一个字符串s用于保存结果，s初始化为拥有10000个字符“0”的字符串。每次进行运算操作之前都要把s恢复初始状态。最终输出的运算结果就是从字符串s中第一个不为0的地方开始输出。

为了表示数字0（相等数相减），当结果为0时，通过s.resize(1,'0')把字符串s的大小缩减到1，然后直接返回s，这样输出结果的时候正好输出为0

Exchange函数用于判断两个大整数的大小关系，再把较大的数放在s1字符串，较小的数放在s2字符串，便于减法和乘法操作，再用一个flag来标识是否进行过交换。

在分治乘法过程中X\*Y=A\*C\*10^(m/2+n/2)+A\*D\*10^(m/2)+B\*C\*10^(n/2)+B\*D，因此定义了字符串RAC表示A\*C\*10^(m/2+n/2)，字符串AD表示A\*D\*10^(m/2)，字符串BC表示B\*C\*10^(n/2)，字符串BD表示B\*D最终结果即为RAC+RAD+RBC+BD。

考虑到有可能即使拆分以后，现有的数据类型longlong还是无法进行运算，因此自定义了一个字符串的乘法函数caculate，来进行乘法运算操作，过程与加减类似，即模拟手算过程，外部循环m次(s1的位数)，内部循环n次(s2的位数)，每位相乘结果如果大于10则进位，上一位的值等于本位乘积temp加上进位。

### 2.5 源程序及注释

#include <iostream>

#include <string>

#include <cstring>

#include <algorithm>

#define MAX 10000

using namespace std;

string s(MAX, '0');//save the results

bool flag = false;//exchange s1 and s2

void exchange(string &s1, string &s2)//s1 is larger than s2

{

flag = false;//initial

if (s1.length()<s2.length())

{

string temp;

temp = s1;

s1 = s2;

s2 = temp;

flag = true;

}

if (s1.length()==s2.length())

{

if (s1 < s2)

{

string temp;

temp = s1;

s1 = s2;

s2 = temp;

flag = true;

}

}

return;

}

string add(string s1, string s2)//s1+s2 99+1

{

string result;

int carry = 0;

s.resize(MAX,'0');//resize

s.assign(MAX,'0');//assign s

exchange(s1, s2);//confirm s1 is longer

reverse(s1.begin(),s1.end());//make the first char(s[0]) unit's digit

reverse(s2.begin(),s2.end());

for(int i=0;i<s1.length();i++)//s1.length()>=s2.length()

{

if(i<s2.length())

{

carry=s1[i]-'0'+s2[i]-'0'+carry;//two addends and one from low carry

if(carry>9)

{

s[i]=carry%10+'0';

carry = carry/10;

}

else

{

s[i]=carry+'0';

carry = 0;

}

}

else

s[i]=s1[i];

}

if (s[s2.length()] == '9')

{

for (int j = s2.length(); j <=s1.length(); j++)

{

if (s[j] >= '9')

{

s[j] = '0';

s[j + 1] = s[j + 1] + carry;

}

}

}

else

s[s2.length()]=s[s2.length()]+carry;

reverse(s.begin(),s.end());

result = s.substr(s.find\_first\_not\_of('0'));

return result;

}

string sub(string s1, string s2)//s1-s2

{

int i,j;

int carry = 0;

string result;

s.resize(MAX,'0');//resize s

s.assign(MAX,'0');//assign s

exchange(s1, s2);//confirm s1 is larger

if(s1 == s2)//result equals 0

{

s.resize(1,'0');//resize s

s.assign(1,'0');//assign s

return s;

}

else

{

reverse(s1.begin(), s1.end());//make the first char(s[0]) unit's digit

reverse(s2.begin(), s2.end());

for (i = 0; i < s1.length(); i++)

s[i]=s1[i];

for (i = 0; i < s2.length(); i++)//s1.length()>=s2.length()

{

if (s[i] >= s2[i])

s[i] = s[i] - s2[i] + '0';

else

{

if(s[i+1]=='0')

for (j = i+1; j < s1.length()-1; j++)

{

if (s[j] <= '0')

{

s[j] = '9';

s[j + 1] = s[j + 1] - 1;//brorrow

}

}

else

s[i + 1] = s[i + 1] - 1;//brorrow

s[i] = s[i] + 10 - s2[i] + '0';

}

}

reverse(s.begin(), s.end());

if(flag==true)//have exchanged

s.insert(s.find\_first\_not\_of('0'), "-");

}

result = s.substr(s.find\_first\_not\_of('0'));

return result;

}

string caculate(string s1, string s2)//s1\*s2

{

string result;

s.resize(MAX, '0');//resize

s.assign(MAX, '0');//assign s

reverse(s1.begin(), s1.end());

reverse(s2.begin(), s2.end());

for (int i = 0; i<s1.length(); i++)

for (int j = 0; j<s2.length(); j++)

{

int temp = (s1[i] - '0')\*(s2[j] - '0');

s[i + j + 1] = s[i + j + 1] - '0' + (s[i + j] - '0' + temp) / 10 + '0';

s[i + j] = (s[i + j] - '0' + temp) % 10 + '0';

}

reverse(s.begin(), s.end());

if (s.find\_first\_not\_of('0') == string::npos)//until over

{

s.resize(1,'0');//resize s

s.assign(1,'0');//assign s

return s;

}

else

{

result = s.substr(s.find\_first\_not\_of('0'));

return result;

}

}

string mul(string s1, string s2)//s1\*s2

{

string result;

exchange(s1, s2);//confirm s1 is longer

int len = s1.length();

if (s1.find\_first\_not\_of('0') == string::npos || s2.find\_first\_not\_of('0') == string::npos)//if s1 or s2 = 0

{

int len = s1.length() + s2.length();

result.resize(1, '0');

result.assign(1, '0');

return result;

}

if(s1.length()==1||s2.length()==1)//the end of recursion

{

result = caculate(s1, s2);

return result;

}

else

{

string A = s1.substr(0, s1.length() /2);

string B = s1.substr(s1.length() /2);

string C = s2.substr(0, s2.length() /2);

string D = s2.substr(s2.length() /2);

string AC = mul(A, C);

string AD = mul(A, D);

string BC = mul(B, C);

string BD = mul(B, D);

string RAC=AC;

for(int i=0;i<(s1.length()-s1.length()/2+s2.length()-s2.length()/2);i++)

{

RAC += "0";

}

string RAD=AD;

for(int i=0;i<(s1.length() - s1.length()/2);i++)

{

RAD += "0";

}

string RBC=BC;

for(int i=0;i<(s2.length() - s2.length()/2);i++)

{

RBC += "0";

}

result = add(RAC,RAD);

result = add(result,RBC);

result = add(result,BD);

}

return result;

}

int main(void)

{

string s1,s2,result;

cout << "Please input the first integer: " ;

cin>>s1;

cout << "Please input the second integer: " ;

cin>>s2;

if (s1[0] != '-'&&s2[0] != '-')//both are positive integers

{

result = add(s1, s2);

cout << "Addition result: " << result << endl;

result = sub(s1, s2);

cout << "Subtraction results: " << result << endl;

result = mul(s1, s2);

cout << "Multiplication results: " << result << endl;

}

if (s1[0] == '-'&&s2[0] != '-')//s1 is negative

{

s1.erase(0, 1);

result = sub(s2, s1);

cout << "Addition results: " << result << endl;

result = add(s1, s2);

result.insert(result.find\_first\_not\_of('0'), "-");

cout << "Subtraction results: " << result << endl;

result = mul(s1, s2);

if(result.find\_first\_not\_of('0') == string::npos)

cout << "Multiplication results: " << "0" << endl;

else

{

result.insert(result.find\_first\_not\_of('0'), "-");

cout << "Multiplication results: " << result << endl;

}

}

if (s1[0] != '-'&&s2[0] == '-')//s2 is negative

{

s2.erase(0, 1);

result = sub(s1, s2);

cout << "Addition results: " << result << endl;

result = add(s1, s2);

cout << "Subtraction results: " << result << endl;

result = mul(s1, s2);

if (result.find\_first\_not\_of('0') == string::npos)

cout << "Multiplication results: " << "0" << endl;

else

{

result.insert(result.find\_first\_not\_of('0'), "-");

cout << "Multiplication results: " << result << endl;

}

}

if (s1[0] == '-'&&s2[0] == '-')//both are negative integers

{

s1.erase(0, 1);

s2.erase(0, 1);

result = add(s1, s2);

result.insert(result.find\_first\_not\_of('0'), "-");

cout << "Addition results: " << result << endl;

result = sub(s2, s1);

cout << "Subtraction results: " << result << endl;

result = mul(s1, s2);

cout << "Multiplication results: " << result << endl;

}

return 0;

}

### 实验三 最优二分查找树

### 实验内容与要求

15.5-1最优二分查找树

题目要求为教材15.5-1题目

输入：例如教材上图15-10中的root表

输出：见教材15.5-1

### 3.2 算法设计

最优子结构：如果一棵最优二叉查找树T有一棵包含关键字ki,..,kj的子树T'，那么这棵子树T'对于关键字Ki,...,kj和伪关键字di-1,...dj的子问题也必定是最优的。因此通过递归算法求解本问题。从root[1,n]开始向下递推，找出树根，左子树和右子树。

root[1,n]为根节点，如果i>root[i][j]-1则代表遇到伪关键字，把d添加到左孩子节点，如果j<root[i][j]+1也代表遇到伪关键字，把d添加到右孩子节点。当i<=root[i][j]-1时代表遇到内部节点，递归调用本算法生成左子树；当j>=root[i][j]+1时代表遇到内部节点，递归调用本算法生成右子树。

### 3.3 实验结果与分析

对于如图3.1所示的测试样例，程序运行结果如图3.2所示，可知程序运行正确。

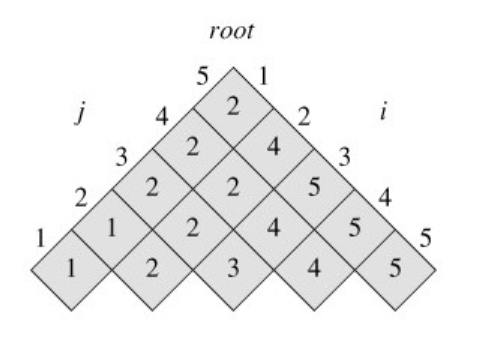
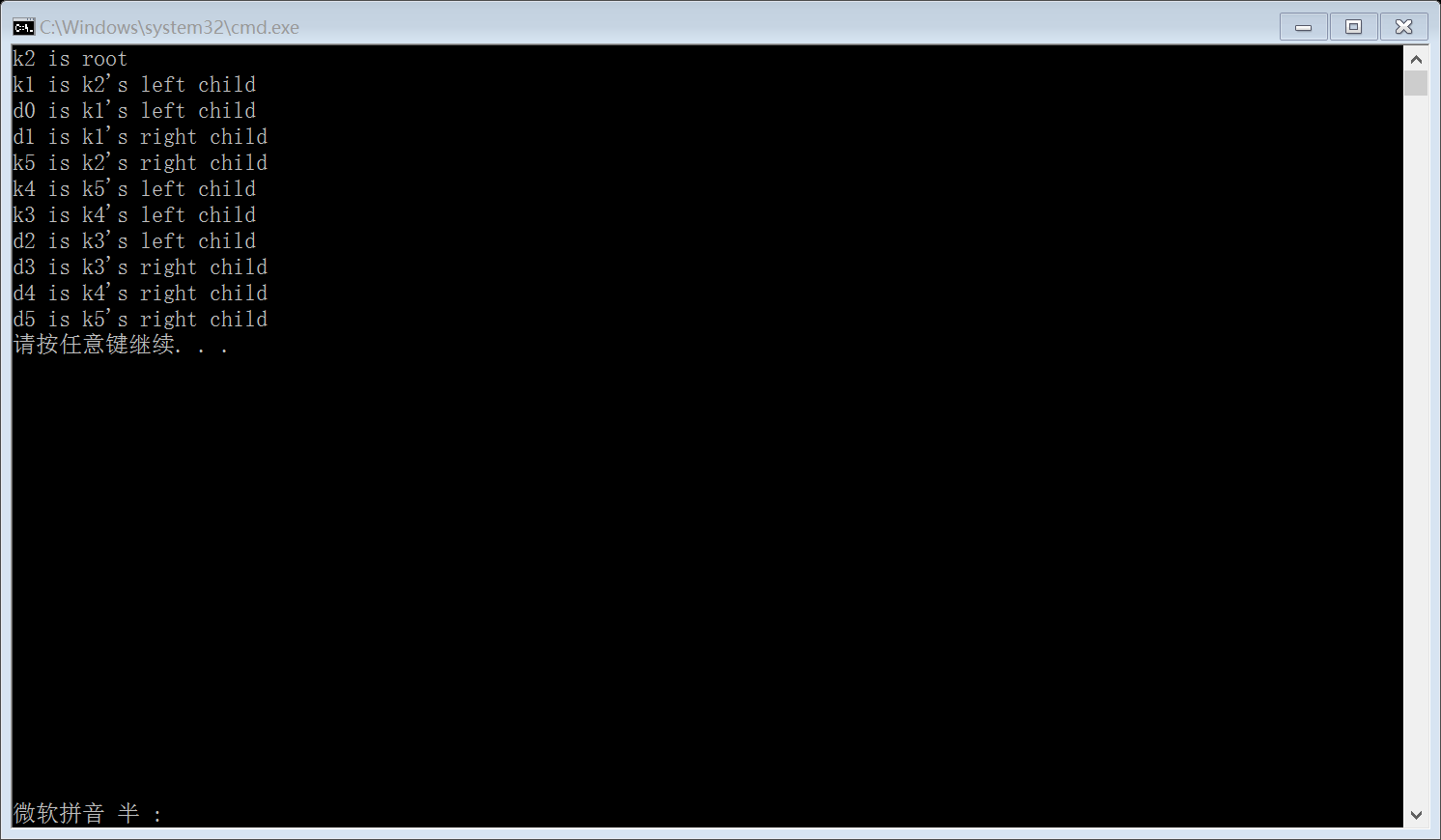
 

图3.1 测试样例 图3.2 测试结果

### 3.4 编程技术与方法

用一个二维数组保存root表（各子树的根），我们只使用表中满足1<=i<=j<=n的表项root[i][j]，因此把不需要用到的表项赋值为0。root数组大小为MAX，因此n=MAX-1。

### 3.5 源程序及注释

#include<iostream>

using namespace std;

#define MAX 6

int root[MAX][MAX] = {

0, 0, 0, 0, 0, 0,

0, 1, 1, 2, 2, 2,

0, 0, 2, 2, 2, 4,

0, 0, 0, 3, 4, 5,

0, 0, 0, 0, 4, 5,

0, 0, 0, 0, 0, 5,

};

void CONSTRUCT\_OPTIMAL\_BST(int i, int j)

{

int s = root[i][j];

if(i==1&&j==MAX-1)//root

cout << "k" << root[i][j] << " is root" << endl;

if(i>s-1)//虚拟键

cout << "d" << root[i][j] - 1 << " is k" << root[i][j] << "'s left child" << endl;

else//内部节点

{

cout << "k" << root[i][s-1] << " is k" << root[i][j] << "'s left child" << endl;

CONSTRUCT\_OPTIMAL\_BST(i, root[i][j] - 1);

}

if(j<s+1)//虚拟键

cout << "d" << root[i][j] << " is k" << root[i][j] << "'s right child" << endl;

else//内部节点

{

cout << "k" << root[s+1][j] << " is k" << root[i][j] << "'s right child" << endl;

CONSTRUCT\_OPTIMAL\_BST(root[i][j]+1, j);

}

}

int main(void)

{

CONSTRUCT\_OPTIMAL\_BST(1, MAX-1);//MAX-1=n

return 0;

}

### 实验四 最短路径

### 实验内容与要求

每对结点之间的最短路径(Floyd-Warshall算法）

补充ALL-PATHS算法，增加前驱矩阵(Chp.25.2)，使得在求出结点间的最短路径长度矩阵A后，能够推导出每对结点间的最短路径。

输入：一个带权图

输入参考：用一个文件，第一行输入两个数，分别表示几个节点几条边，接下来的n行表示这n条边的起点、终点和权值，如图4.1所示：

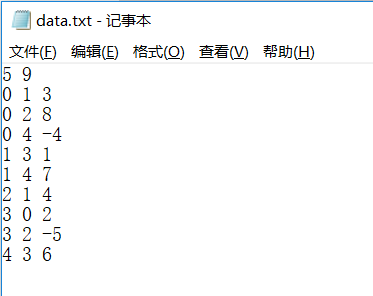


图4.1 测试样例

输出：图中每对节点之间的最短路径

### 4.2 算法设计

采用Floyd-Warshall算法，使用自底向上的算法以递增次序来计算 d(k)ij 的值，首先初始化图G的邻接矩阵A和前驱矩阵D，对于i=j的情况令A[i][j]=0，其余情况A[i][j]初始化为MAX，前驱矩阵所有节点初始化为MAX。

哈哈哈Hi hi hi

### 4.3 实验结果与分析

对于如图4.1所示的测试样例，程序运行结果如图4.2所示，第一个矩阵为图G的邻接矩阵，第二个矩阵为前驱矩阵（用1000代替NIL），可知程序运行正确。

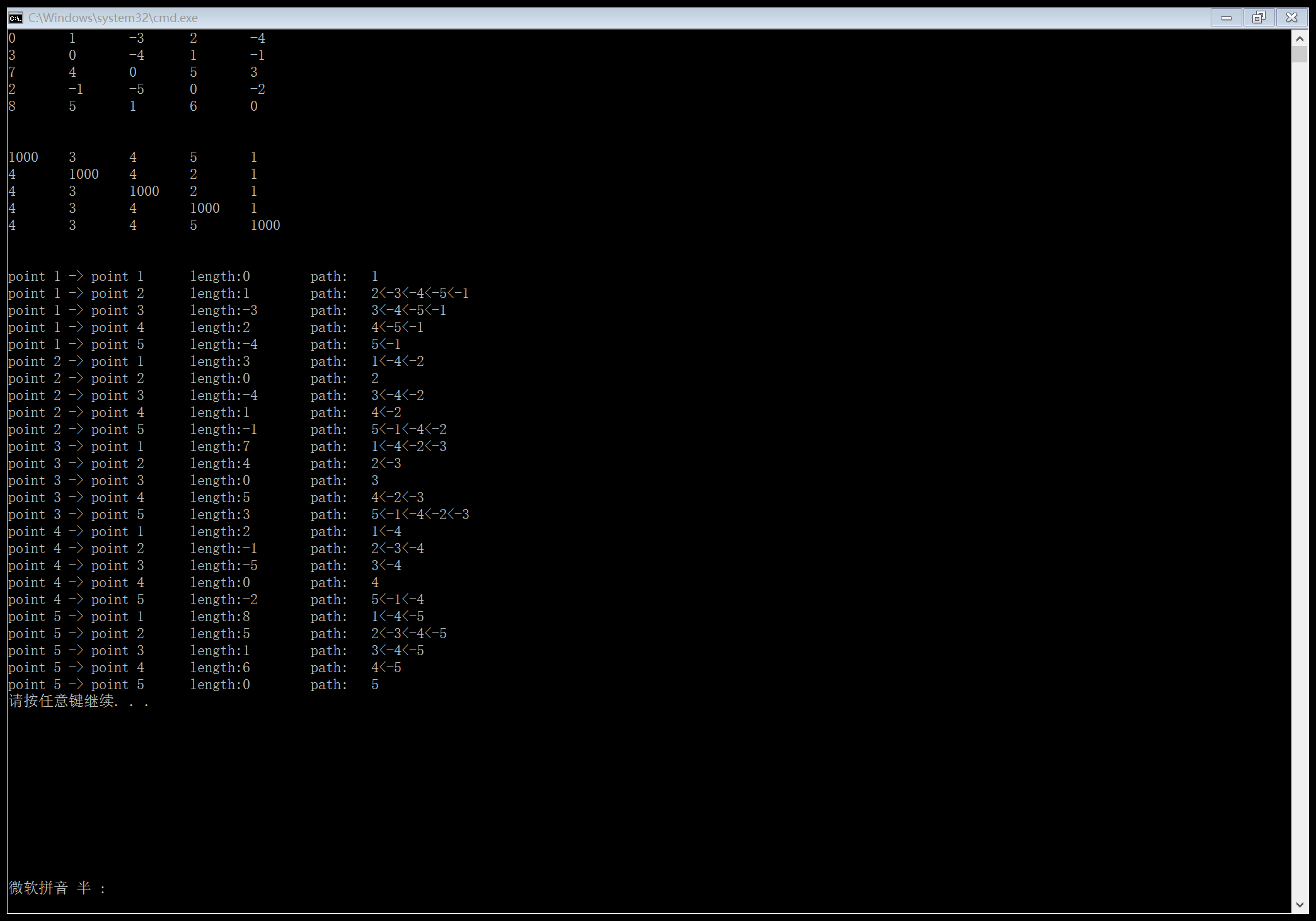


图4.2 测试结果

### 4.4 编程技术与方法

定义一个类MyPair保存所有最近点对

哈哈哈Hi hi hi

### 4.5 源程序及注释

哈哈哈Hi hi hi

哈哈哈Hi hi hi