

《 算法设计与分析 》

实验报告本

|  |  |
| --- | --- |
| 班 级： | **计211** |
| 学 号： | **21013017** |
| 姓 名： | **蒋文强** |
| 指导教师： | **叶琪** |

信息科学与工程学院

年 月

**实验一 递归与分治算法**

**一、实验目的**

**1、**熟悉数据结构和基本的排序和搜索算法，熟悉编程语言的集成开发环境，掌握程序设计与实现的能力，分析算法的复杂度。

2、通过上机实验，要求掌握递归与分治法算法的问题描述、算法设计思想、程序设计和算法复杂性分析等。

**二、实验装置**

**Win10, Visual C++/Java**

**三、实验内容**

**（**在第一次实验题目库中选择题目，填写题目内容及输入输出要求**）**

1. 假设所有的物件都是一个平面的点。如果点和圆环的距离确实小于圆环的半径那么认为点被圆环圈中了。如果两个物件被放置在了同一个位置，那么圆环的半径被认定为零。

输入：输入包括几组数据。每组数据包括一个整数N(2 <= N <= 100,000)，和场地里物件的总数。接下来的N行，每行包括一组(x, y)，物件的坐标。输入N为0时停止输入。

输出：对于每组数据，单行输出Cyberground经理要求的圆环的半径，精确到小数点后两位。

1. 电梯按照指令上下移动，向上每层6秒，向下每层4秒，每次停下5秒。

输入：

1 2

3 2 3 1

输出：

17

41

1. Number Sequence

一个数列定义如下：f（1）=1，f（2）=1，f（n）=（A\*f（n-1）+B\*f（n-2））mod 7。给定A、B和n，你要计算f（n）的值。

输入：

1 1 3

1 2 10

0 0 0

输出：

2

5

1. 9、汉诺塔II

在汉诺塔原有的规则基础上新加一根柱子，问在一次游戏中使用了N个盘子时，他需要多少次移动才能把他们都移到第三个柱子上？

输入：

1

3

12

输出：

1

5

81

1. Uniform Generator

计算机模拟通常需要随机数。生成伪随机数的一种方式是通过一定形式的函数：

seed（x + 1）= [seed（x）+ STEP]％MOD

其中'％'是模运算符。

这样的函数将生成在0和MOD-1之间的伪随机数（种子）。这种形式的作用的一个问题就是，它们将总是重复地生成相同的模式。

为了最小化这种影响，仔细选择STEP和MOD值，可以使得在0和MOD-1（包括这两者）之间的所有值的均匀分布。

例如，如果STEP = 3和MOD = 5，该函数将在重复周期中生成一系列伪随机数0,3,1,4,2。在该示例中，将在函数的每个MOD迭代中生成包括0和MOD-1之间的所有数字。注意，由于每次产生种子（x）时生成相同种子（x + 1）的函数的性质，意味着如果函数将生成在0和MOD-1之间的所有数字，则它将均匀地生成伪随机数与每个MOD迭代。

如果STEP = 15和MOD = 20，该函数生成系列0,15,10,5（或任何其他重复系列，如果初始种子不是0）。这是一个很差的STEP和MOD选择，因为没有初始种子将生成从0和MOD-1的所有数字。

您的程序将确定STEP和MOD的选择是否将生成伪随机数的均匀分布。

输入：

3 5

15 20

63923 99999

输出：

3 5 Good Choice

15 20 Bad Choice

63923 99999 Good Choice

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**

1. 代码：

#include<iostream>

#include<algorithm>

#include<math.h>

using namespace std;

struct point

{

double x,y;

}a[100001];

//w数组用来保存在分界线附近的点的编号

int w[100001];

//将所有点按x坐标排序

bool cmpx(const point &a,const point &b)

{

return a.x<b.x;

}

//将分界线附近的点按y坐标排序，不需要改动坐标数组，只修改w数组即可

bool cmpy(const int &m,const int &n)

{

return a[m].y<a[n].y;

}

//求两个点之间的距离

double mix(int m,int n)

{

return sqrt((a[m].x-a[n].x)\*(a[m].x-a[n].x)+(a[m].y-a[n].y)\*(a[m].y-a[n].y));

}

double res(int left,int right)

{

if(left==right)

return 10000000;

if(left+1==right)

return mix(left,right);

int mid=(left+right)/2;

//切分平面

double d1=res(left,mid);

double d2=res(mid+1,right);

double d=min(d1,d2);

int k=0;

//找出左右平面中在分界线“附近”的点

for(int i=left;i<=right;i++)

{

//只保留点的坐标即可

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

w[k++]=i;

}

sort(w,w+k,cmpy);

for(int i=0;i<k-1;i++)

{

//只需要向后搜索6个点，再后面的可以舍掉

for(int j=i+1;j<k && j<i+7;j++)

{

if(a[w[j]].y-a[w[i]].y>=d)

break;

d=min(d,mix(w[j],w[i]));

}

}

return d;

}

int main()

{

int i,n;

while(scanf("%d",&n)!=0)

{

if(n==0)

break;

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

scanf("%lf%lf",&a[i].x,&a[i].y);

sort(a,a+n,cmpx);

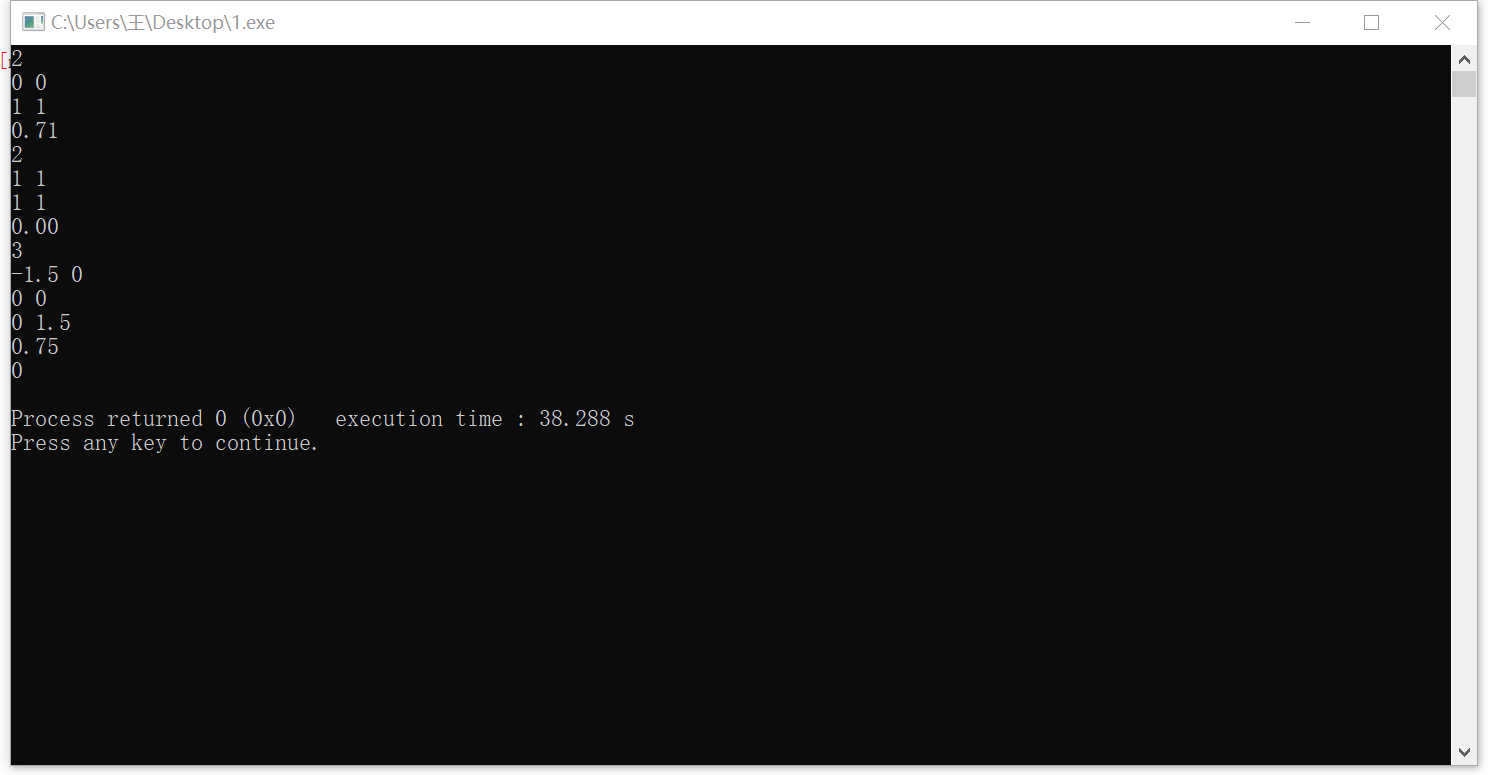
printf("%.2f\n",res(0,n-1) / 2);

}

return 0;

}

结果截图：



设计思路：

核心思想是通过分治策略，在每一步递归中将问题划分为规模较小的子问题，并最终合并子问题的解来得到原问题的解。

首先，通过对输入点按照 x 坐标进行排序，建立了问题的初始状态。接着，通过递归的方式，将问题分解为左右两个子问题，并计算出每个子问题的最近点对的距离。然后，通过在分界线附近寻找可能的最近点对，进一步缩小搜索范围。最后，通过比较左右两个子问题和分界线附近的最近点对的距离，找到全局最近点对的距离。

数据结构：点结构、数组

时间复杂度：O（n\*log（n）)

1. 代码：

#include<iostream>

using namespace std;

int main()

{

int n,now\_floor=0,next\_floor,time=0;

cin>>n;

while(n--){

cin>>next\_floor;

if(next\_floor>now\_floor)//上楼

{

time+=(next\_floor-now\_floor)\*6+5;

now\_floor=next\_floor;

}

else if(next\_floor<now\_floor)

{

time+=(now\_floor-next\_floor)\*4+5;

now\_floor=next\_floor;

}

else

time+=5;

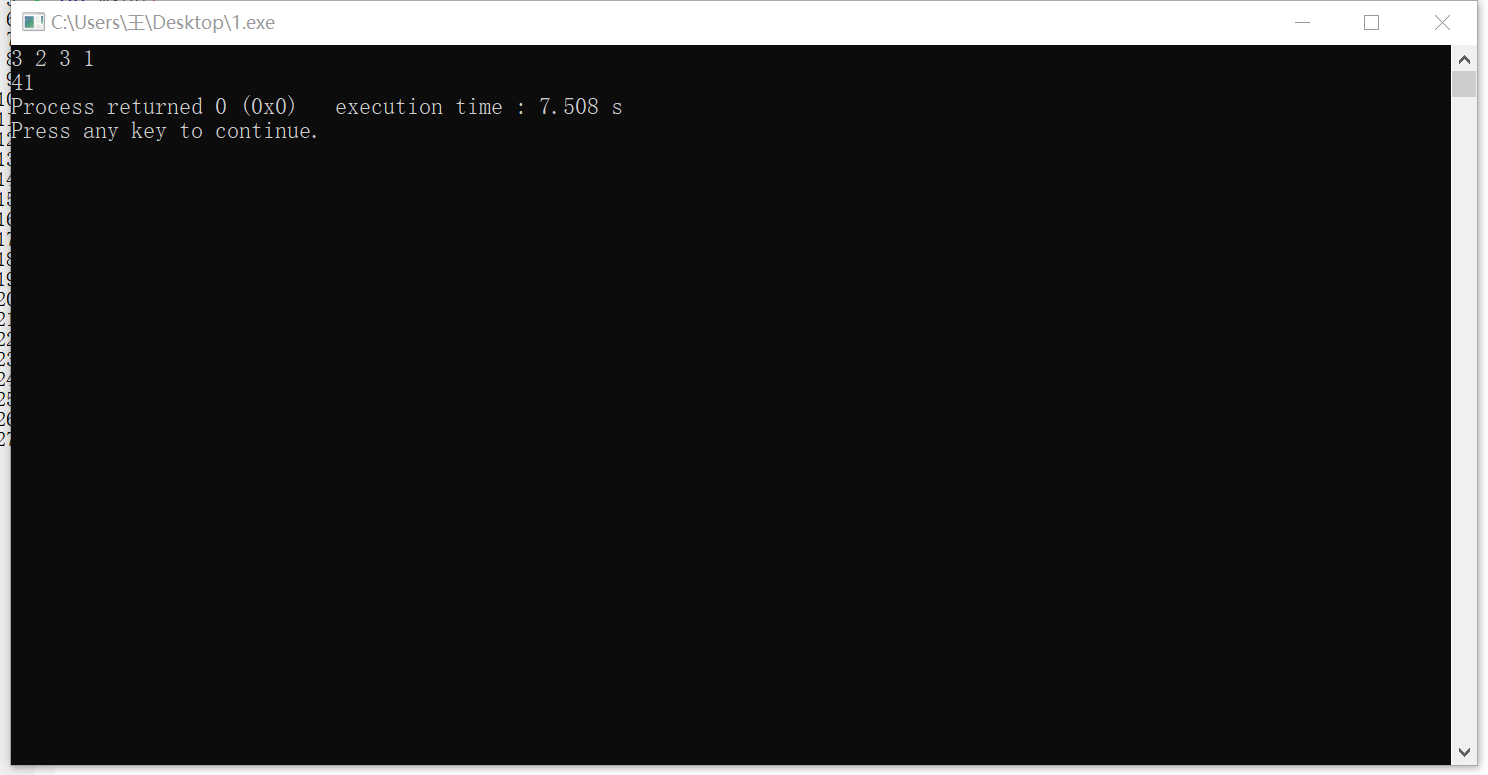
}

cout<<time;

return 0;

}

运行截图：



设计思路：

通过循环处理每个输入的楼层：

若下一目标楼层 next\_floor 大于当前楼层 now\_floor，说明需要上行。电梯上升到 next\_floor，需要时间为 (next\_floor - now\_floor) \* 6 + 5，更新时间并将当前楼层设为 next\_floor。

若 next\_floor 小于当前楼层 now\_floor，说明需要下行。电梯下降到 next\_floor，需要时间为 (now\_floor - next\_floor) \* 4 + 5，更新时间并将当前楼层设为 next\_floor。

若 next\_floor 等于当前楼层 now\_floor，则电梯停留在当前楼层，需要时间为 5。

数据结构：

使用整数变量 ‘n’ 存储总楼层数。

使用整数变量 ‘now\_floor’存储当前所在楼层。

使用整数变量 ‘next\_floor’存储目标楼层。

使用整数变量 ‘time’ 存储总时间。

时间复杂度：O(n)

1. 代码：

#include <iostream>

#include <algorithm>

#include <cstring>

#include <iostream>

using namespace std;

int main(){

int A,B,n;

while(cin >> A >> B >> n){

if(A==0&&B==0&&n==0)return 0;

int f[18];

f[0] = 0;

f[1]=f[2]=1;

for(int i=3;i<17;i++){

f[i] = (A \* f[i - 1] + B \* f[i - 2]) % 7;

}

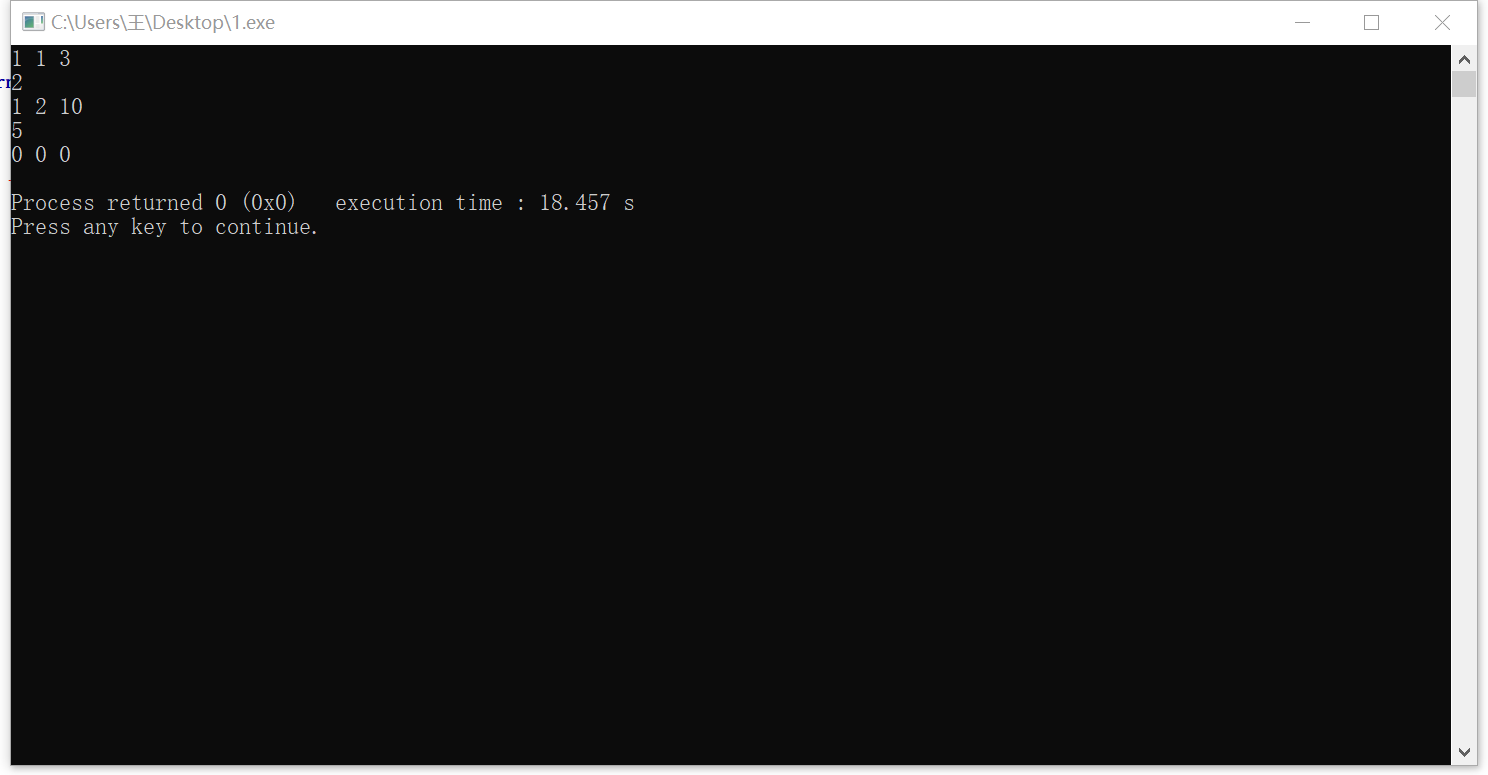
cout << f[n%17] << endl;

}

return 0;

}

运行截图：



设计思路：

可以发现“0 1 1 2 3 4 1 6 0 6 6 5 4 2 6 1”这16个数是一个周期。输出计算得到的 `f[n % 17]` 的值。这里取模 17 是因为在计算过程中，f 的周期为 16。

数据结构：

使用整数数组 ‘f’ 存储计算的结果。

使用整数变量 ‘A’、‘B’、‘n’ 分别表示给定的递推关系参数和计算的次数。

时间复杂度：O(1)

1. 代码：

#include <iostream>

#include <algorithm>

using namespace std;

typedef unsigned long long ull;

ull dp[65], f[65];

int main()

{

for (int i = 1; i <= 64; i++)

dp[i] = f[i] = 2 \* f[i - 1] + 1;

for (int i = 1; i <= 64; i++)

for (int j = 0; j < i; j++)

dp[i] = min(dp[i], 2 \* dp[j] + f[i - j]);

int n;

while (cin >> n)

cout << dp[n] << endl;

system("pause");

return 0;

}

设计思路：

使用数组 dp和f分别存储动态规划的结果。dp[i]表示将i个盘子从源柱子移动到目标柱子的最小步数，f[i]表示移动i个盘子需要的最小步数。设 dp[i] = 2 \* dp[j] + f[i - j]，表示将i个盘子移动到目标柱子的最小步数可以由j个盘子移动到目标柱子的最小步数和i - j个盘子移动到辅助柱子的最小步数组成。这里遍历j取最小值。对于dp和f，初始条件为dp[1] = f[1] = 1。使用两层循环，外层遍历i表示盘子数量，内层遍历j表示将j个盘子移动到目标柱子的最小步数。通过状态转移方程更新dp[i]。

数据结构：

使用数组dp存储动态规划的结果，dp[i]表示将i个盘子移动到目标柱子的最小步数。

使用数组f存储移动i个盘子需要的最小步数。

时间复杂度：

使用两层循环，外层循环遍历i从1到64，内层循环遍历j从0到i - 1，因此总体时间复杂度为O(64^2)。由于是常数级别的循环次数，可以看作是O(1)。

1. 代码：

#include <iostream>

#include <cstdio>

using namespace std;

int gcd(int a, int b)

{

return a%b?gcd(b,a%b):b;

}

int main ()

{

char tab[2][15] = {"Bad Choice", "Good Choice"} ;

int a, b ;

while (~scanf ("%d%d", &a, &b))

{

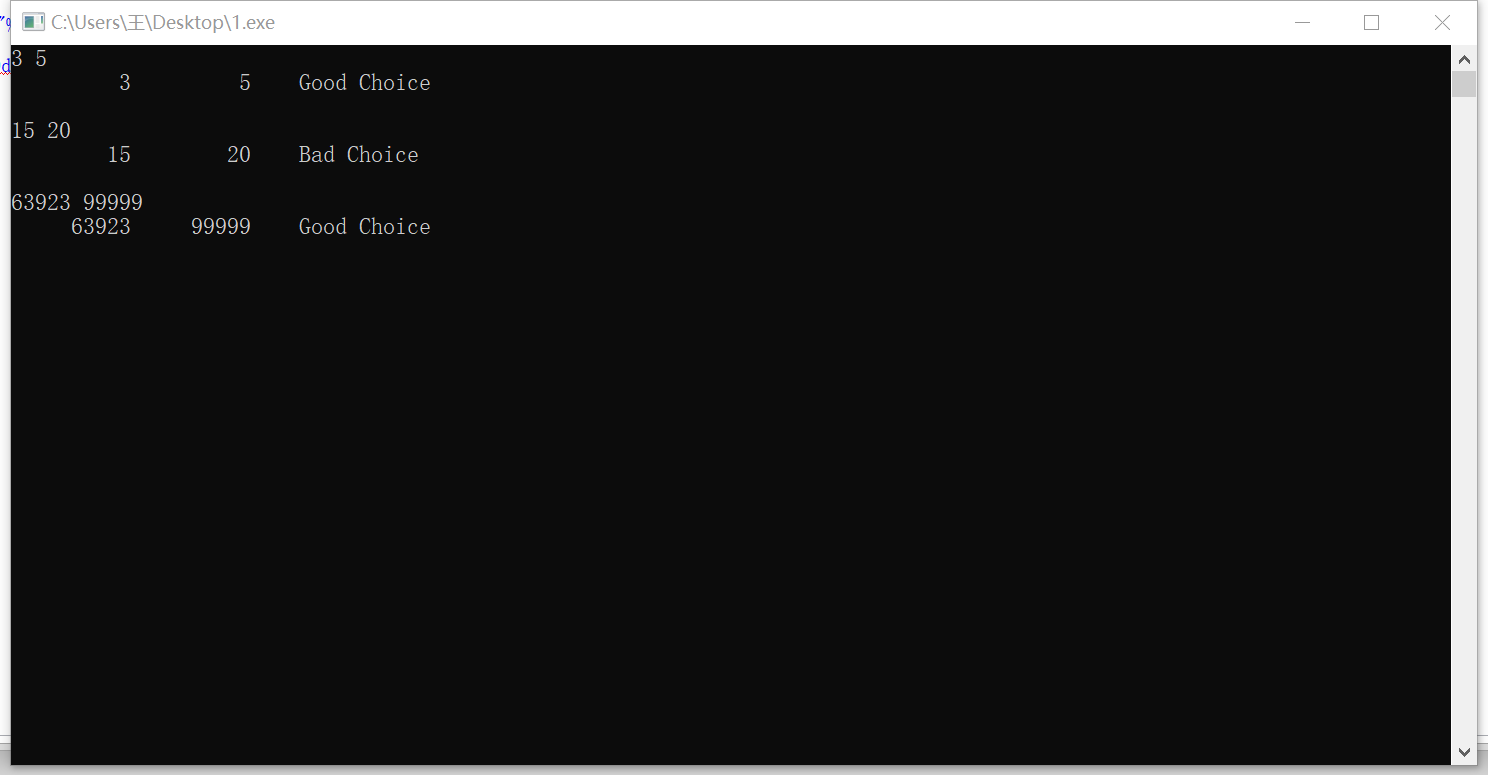
printf ("%10d%10d %s\n\n", a, b, tab[gcd(a,b)==1]) ;

}

return 0 ;

}

运行截图：



设计思路：

使用递归的辗转相除法求两个整数的最大公约数。函数gcd的定义是int gcd(int a, int b)，它返回a和b的最大公约数。递归终止条件是a对b取余等于0，此时b即为最大公约数。

数据结构：

使用两个整数a和b作为输入数据。

使用一个字符数组tab存储两种结果："Bad Choice" 和 "Good Choice"。

时间复杂度：

辗转相除法的时间复杂度取决于两个整数的大小，但在最坏情况下，时间复杂度为O(log(min(a, b)))。由于只进行一次辗转相除法，整体时间复杂度为 O(log(min(a, b)))。

**实验一成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验二 动态规划和贪心算法**

**一、实验目的**

1、理解动态规划法的设计思想，分析是否满足最优子结构性质，刻画其结构特征，递归地定义最优值（动态规划方程），以自底向上的方式计算出最优值，构造最优解。掌握动态规划的算法框架和设计策略。

2、理解贪心算法的设计思想，掌握贪心算法的算法框架和设计策略，选取度量标准，逐步构造最优解。

**二、实验装置**

**Win7, Visual C++/Java**

**三、实验内容**

**（**在第一次实验题目库中选择题目，填写题目内容及输入输出要求**）**

1.

2．

3．

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**

1.

2．

3．

**实验二成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验三 回溯算法**

**一、实验目的**

**1、**理解回溯法的设计思想，回溯法是一个既带有系统性又带有跳跃性的搜索算法。掌握从包含问题的所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树的过程。掌握回溯法的算法框架和设计策略。

**二、实验装置**

**Win7, Visual C++/Java**

**三、实验内容**

**（在第三次实验题目库中选择题目，填写题目内容及输入输出要求）**

1.

2．

3．

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**

1.

2．

3．

**实验三成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**实验四 分支限界算法**

**一、实验目的**

**1、**理解分支限界法的设计思想，掌握分支限界法的算法框架和设计策略。

**2、**通过上机实验，要求掌握算法设计思想、程序设计和算法复杂性分析等。

**二、实验装置**

**Win7, Visual C++/Java**

**三、实验内容**

**（在第四次实验题目库中选择题目，填写题目内容及输入输出要求）**

1.

2．

3．

**四、程序运行结果**

**（说明设计思路，解释使用的数据结构，计算时间复杂度）**

1.

2．

3．

**实验四成绩：\_\_\_\_\_\_\_\_\_\_\_\_ 教师签名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**