**南京理工大学计算机科学与工程学院**

**算法设计与分析 报告**

**班 级 9181069502**

**学生姓名 黄海浪**

**学 号 9181040G0818**

**起止时间 2020.05.04-2020.05.05**

**教 师 孙廷凯**

目录

[一. 问题描述与实验要求 3](#_Toc39585791)

[二. 符号说明与语法约定 3](#_Toc39585792)

[三. 算法自然语言描述 3](#_Toc39585793)

[四. 算法伪代码描述 3](#_Toc39585794)

[五. 算法的代码及结果 4](#_Toc39585795)

[代码： 4](#_Toc39585796)

[结果： 8](#_Toc39585797)

## 问题描述与实验要求

给你两瓶装满了油的油瓶A和B，其容量分别是5升和3升，再给你一个容量很大（至少8升）的空油瓶C，三个油瓶均没有刻度。现在想量出x升的油（x=1,2,3,4,5,6,7,8），该怎么倒油？试用回溯策略求解，给出一个合适的倒油操作序列。

## 符号说明与语法约定

为了不失一般性，代码能够对给定的三个瓶子容量上限（A、B、C）和一个倒出的油目标值target，以及初始油量A0,B0,C0，即可求得结果。

## 三.算法自然语言描述

回溯法求解

对于每个状态（三个瓶子目前油量）均有6种倒法，为了得到回溯的条件，我们记录之前已经存在的所有状态，那么回溯的条件是之前存在当前的状态或者目前的状态中存在一个油量为target的瓶子。

对于每个状态我们均用6种倒法进行操作，直到和之前状态相同或者得到结果。

## 四. 算法伪代码描述

输入：空

输出：得到目标油量的步骤

主要算法

getAns(index)   //index为目前的状态为第index步

1.  for i<- 1 to 6

2.      step[index][3]=i;               //step[i][j]代表第i个状态 j=0/1/2代表瓶子油量 j=4代表下一步的操作

3.      move(index+1)                   //进行倒油

4.      if index+1 状态存在target then

5.          ++cnt;                      //cnt为目前求得的有多少结果

6.          打印输出并记录              //记录是为了找到最少步骤的解

7.      else if 之前不存在当前状态 then

8.          getAns(index+1)             //进行下一步 否则回溯

9.      end if

10. end for

## 五. 算法的代码及结果

### 代码：

#include <iostream>

#include <string>

#include <time.h>

using namespace std;

typedef long long LL;

const int MAX\_STEP = 0xffff;        //最大步数

const int MAX\_CAP = 0x7fffffff;     //最大值

const string way[6] = {"a-->b", "b-->a","a-->c","c-->a","b-->c","c-->b"};       //移动步骤

int step[MAX\_STEP][4];      //step[i][0/1/2] 分别代表目前瓶子容量 step[i][3] 代表下一步操作

struct {                    //保存最少的 一种 做法

    int len = MAX\_CAP;

    int step[MAX\_STEP][4];

} minStep;

int b1,b2,b3,target;        // 三个瓶子容量  目标

bool isRight(int index);    //判断第x步是否正确 达到target

bool isCircle(int index);   //判断第 index 步是否与之前重复

void move(int index);       //对第index个的进行计算 通过step[index-1];

void show(int step[][4],int index,int cnt);     //显示

void getAns(int index,int &cnt);                //递归实现

int main() {

    cout<<"输入三个瓶子容量(-1代表无限)和目标: ";

    cin>>b1>>b2>>b3>>target;

    b1==-1&&(b1=MAX\_CAP);

    b2==-1&&(b2=MAX\_CAP);

    b3==-1&&(b3=MAX\_CAP);

    cout<<"输入三个瓶子初始值: ";

    cin>>step[0][0]>>step[0][1]>>step[0][2];

    clock\_t start, finish;

    start = clock();

    int index = 0,cnt = 0;

    if (isRight(index)) {

        ++cnt;

        show(step,index,cnt);

        return 0;

    }

    getAns(index,cnt);      //递归实现

    if(cnt==0){

        cout<<"无解"<<endl;

        return 0;

    }

    show(minStep.step,minStep.len,-1);

    finish = clock();

    cout<<endl<<"程序用时" <<(double)(finish - start) / CLOCKS\_PER\_SEC;

    return 0;

}

void getAns(int index,int &cnt) {

    if(index>MAX\_STEP) return;

    for(int i=0; i<6; ++i) {

        step[index][3] = i;

        move(index+1);

        if(isRight(index+1)) {

            ++cnt;

            show(step,index+1,cnt);

        } else if(!isCircle(index+1)) {

            getAns(index+1,cnt);

        }

    }

}

void show(int step[][4],int index,int cnt) {

    if(cnt==-1) {

        cout<<"步数最少的一种："<<endl;

    } else {

        cout<<"第 "<<cnt<<" 种"<<endl;

    }

    cout<<"a -- b -- c"<<endl;

    if (minStep.len>index) {

        minStep.len = index;

        for(int i=0; i<=index; ++i) {

            for(int j=0; j<4; ++j) {

                minStep.step[i][j] = step[i][j];

            }

        }

    }

    for(int i=0; i<index; ++i) {

        cout<<step[i][0]<<"--"<<step[i][1]<<"--"<<step[i][2];

        cout<<"    "<<way[step[i][3]]<<endl;

    }

    cout<<step[index][0]<<"--"<<step[index][1]<<"--"<<step[index][2]<<endl;

    cout<<"步数为："<< index<<endl<<endl;

}

//判断第 index 步是否与之前重复

bool isCircle(int index) {

    bool flag = false;

    for(int i = index-1; i>=0; --i) {

        flag = false;

        for(int j=0; j<3; ++j) {

            if(step[i][j] != step[index][j]) {

                flag = true;

                break;

            }

        }

        if (!flag) return true;

    }

    return false;

}

//判断第x步是否正确 达到target

bool isRight(int index) {

    for(int i=0; i<3; ++i) {

        if(step[index][i]==target) return true;

    }

    return false;

}

void move(int index) {  //对第index个的进行计算 通过step[index-1];

    int lastIndex = index - 1;

    //"a-->b","b-->a","a-->c","c-->a","b-->c","c-->b"

    //0 -- a

    //1 -- b

    //2 -- c

    switch(step[lastIndex][3]) {

        case 0:

            step[index][1] = step[lastIndex][0] + step[lastIndex][1];

            if (step[index][1]>b2) step[index][1] = b2;

            step[index][0] = step[lastIndex][0] - step[index][1] + step[lastIndex][1];

            step[index][2] = step[lastIndex][2];

            break;

        case 1:

            step[index][0] = step[lastIndex][0] + step[lastIndex][1];

            if (step[index][0]>b1) step[index][0] = b1;

            step[index][1] = step[lastIndex][0] - step[index][0] + step[lastIndex][1];

            step[index][2] = step[lastIndex][2];

            break;

        case 2:

            step[index][2] = step[lastIndex][0] + step[lastIndex][2];

            if (step[index][2]>b3) step[index][2] = b3;

            step[index][0] = step[lastIndex][0] - step[index][2] + step[lastIndex][2];

            step[index][1] = step[lastIndex][1];

            break;

        case 3:

            step[index][0] = step[lastIndex][0] + step[lastIndex][2];

            if (step[index][0]>b1) step[index][0] = b1;

            step[index][2] = step[lastIndex][0] - step[index][0] + step[lastIndex][2];

            step[index][1] = step[lastIndex][1];

            break;

        case 4:

            step[index][2] = step[lastIndex][1] + step[lastIndex][2];

            if (step[index][2]>b3) step[index][2] = b3;

            step[index][1] = step[lastIndex][1] - step[index][2] + step[lastIndex][2];

            step[index][0] = step[lastIndex][0];

            break;

        case 5:

            step[index][1] = step[lastIndex][1] + step[lastIndex][2];

            if (step[index][1]>b2) step[index][1] = b2;

            step[index][2] = step[lastIndex][1] - step[index][1] + step[lastIndex][2];

            step[index][0] = step[lastIndex][0];

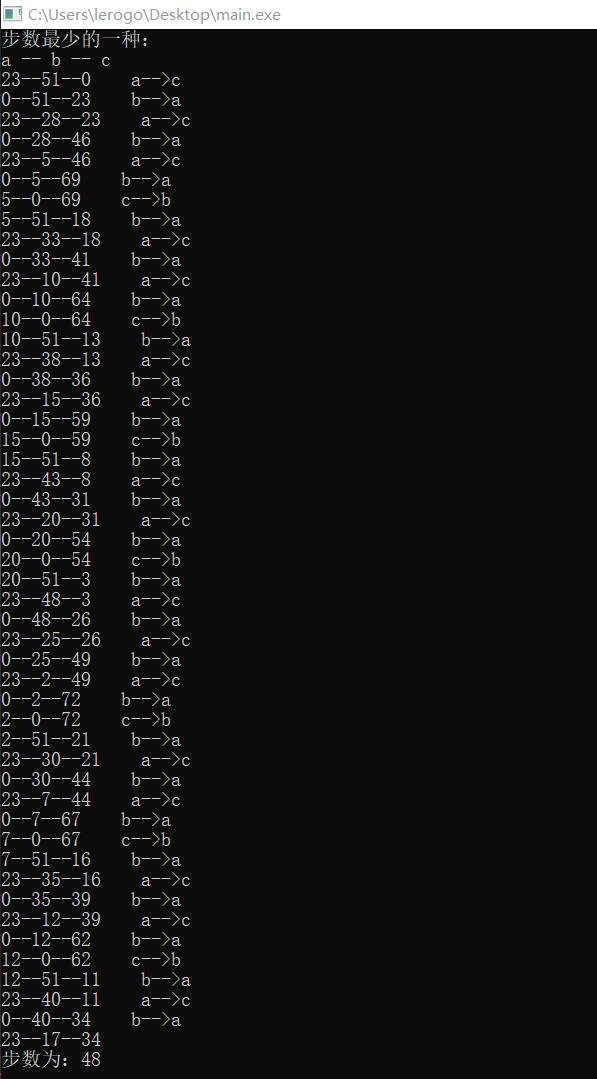
            break;

    }

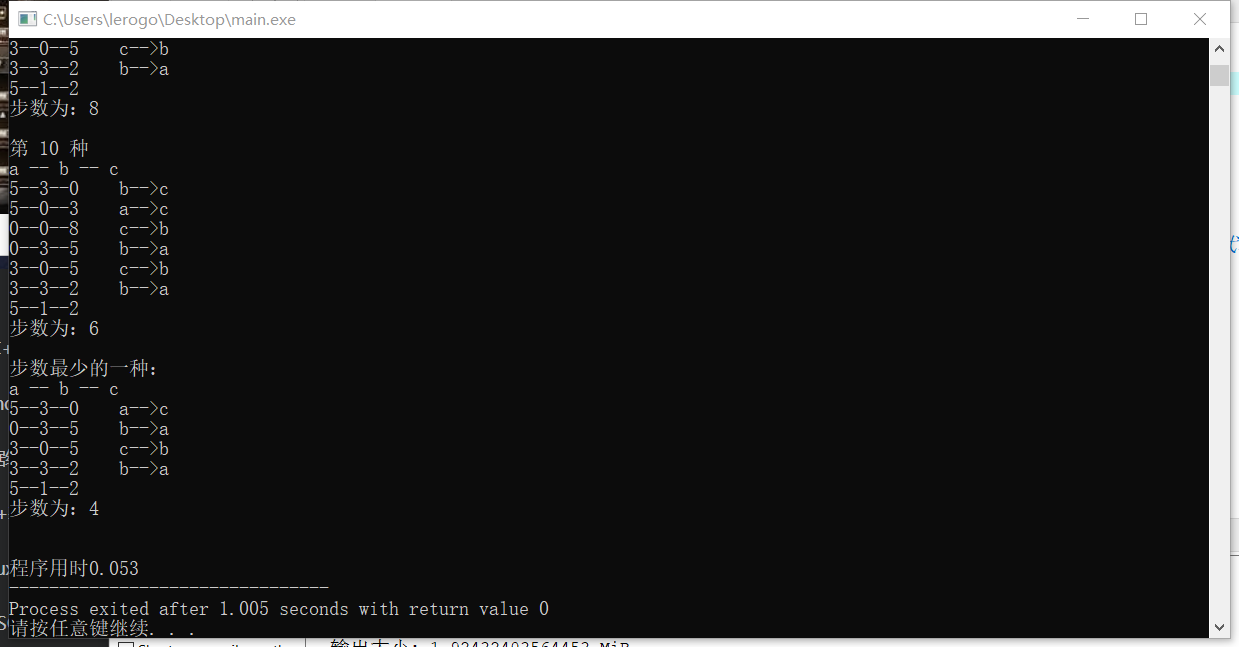
}

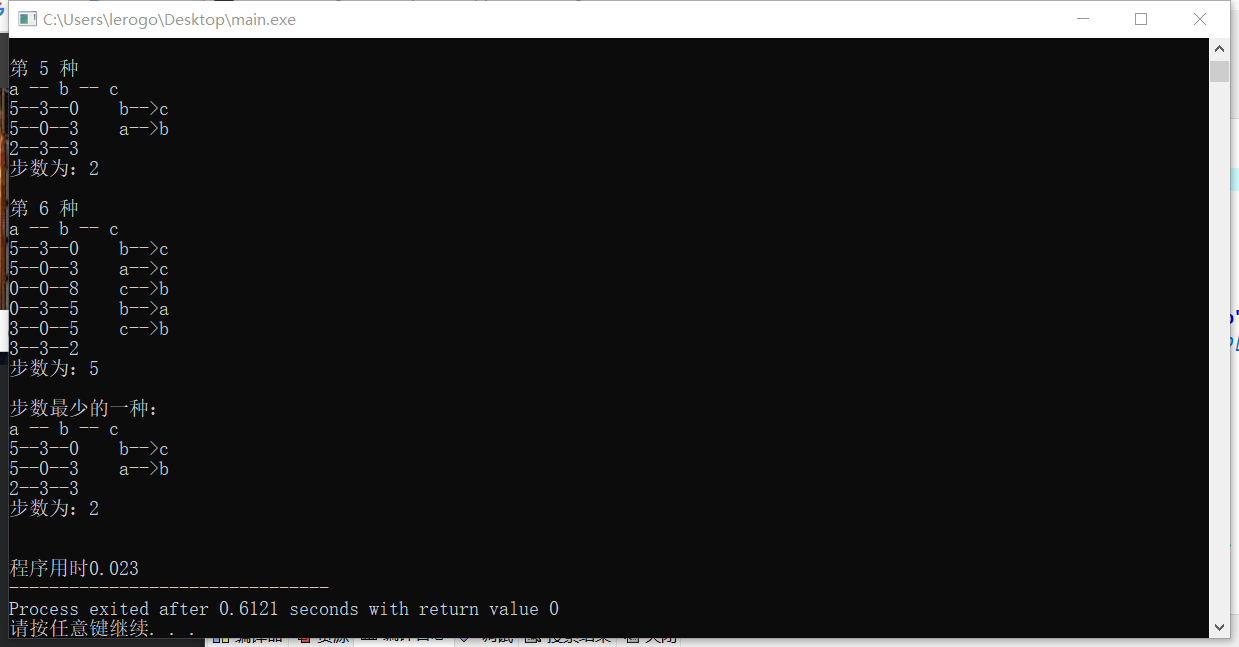
### 结果：

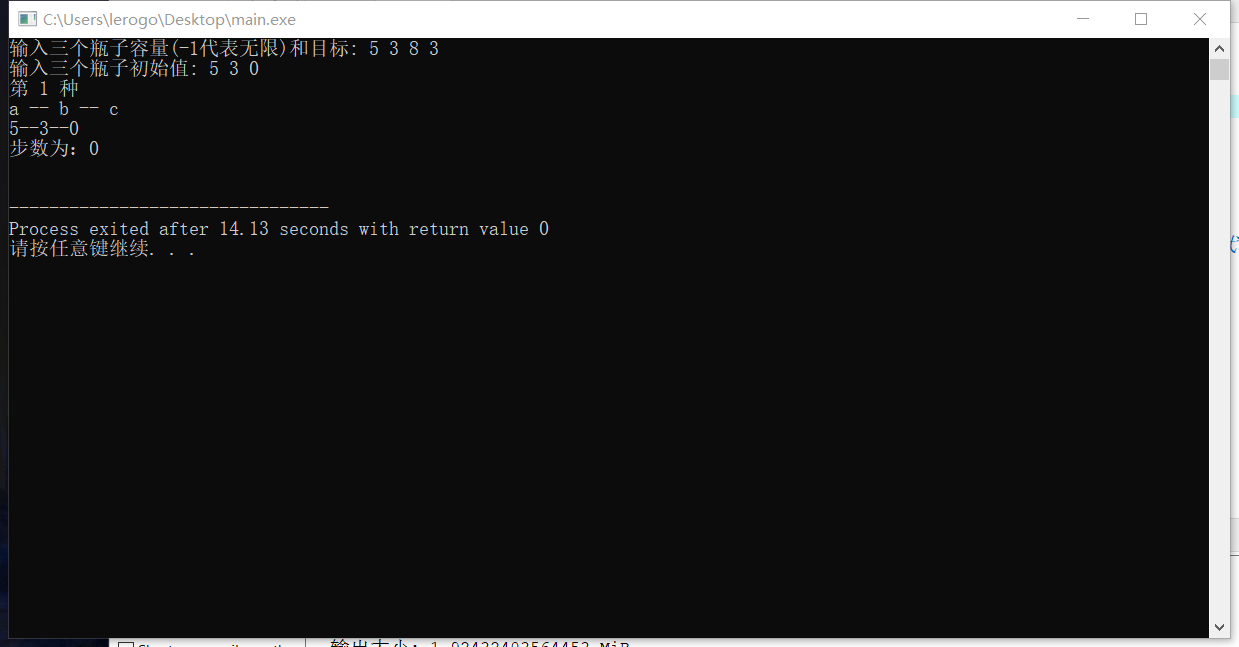
###### 首先测试了两个互质的数23 51 -1（无上限），初始油量为23 51 0，我们想的得到的结果为17。使用上述代码运行的结果如下：

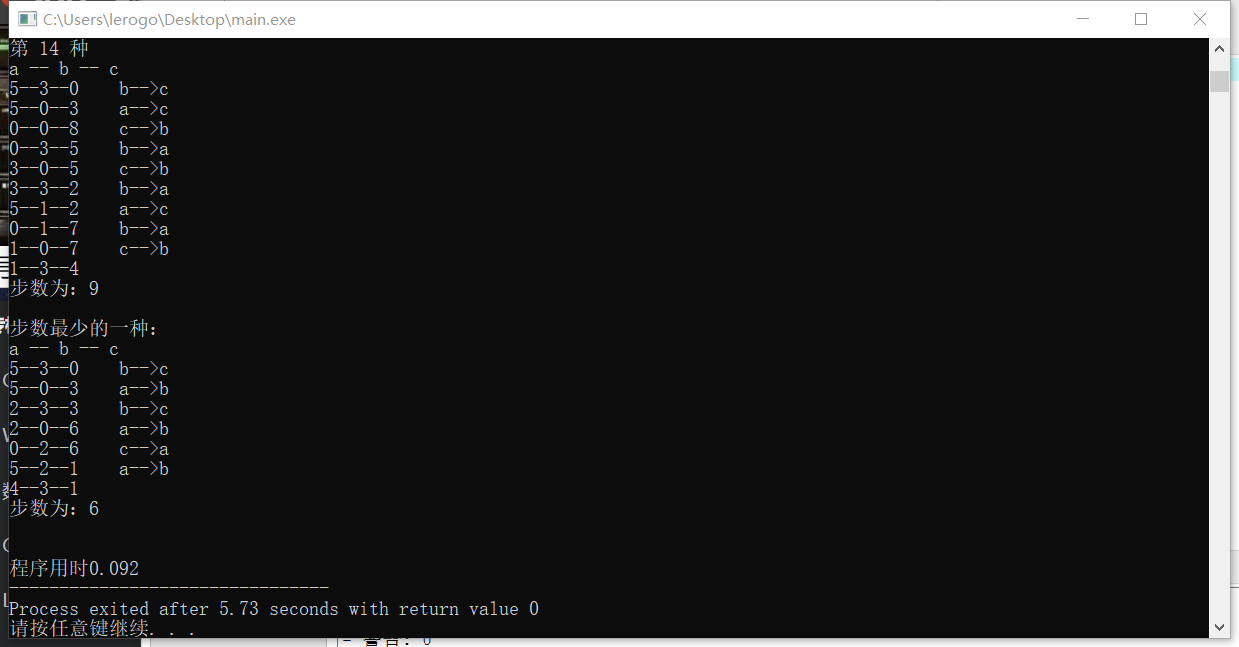


###### 对于油瓶容量上限为“5” “3” “至少8”的三个油瓶A、B、C，以及目标1~8的结果如下：

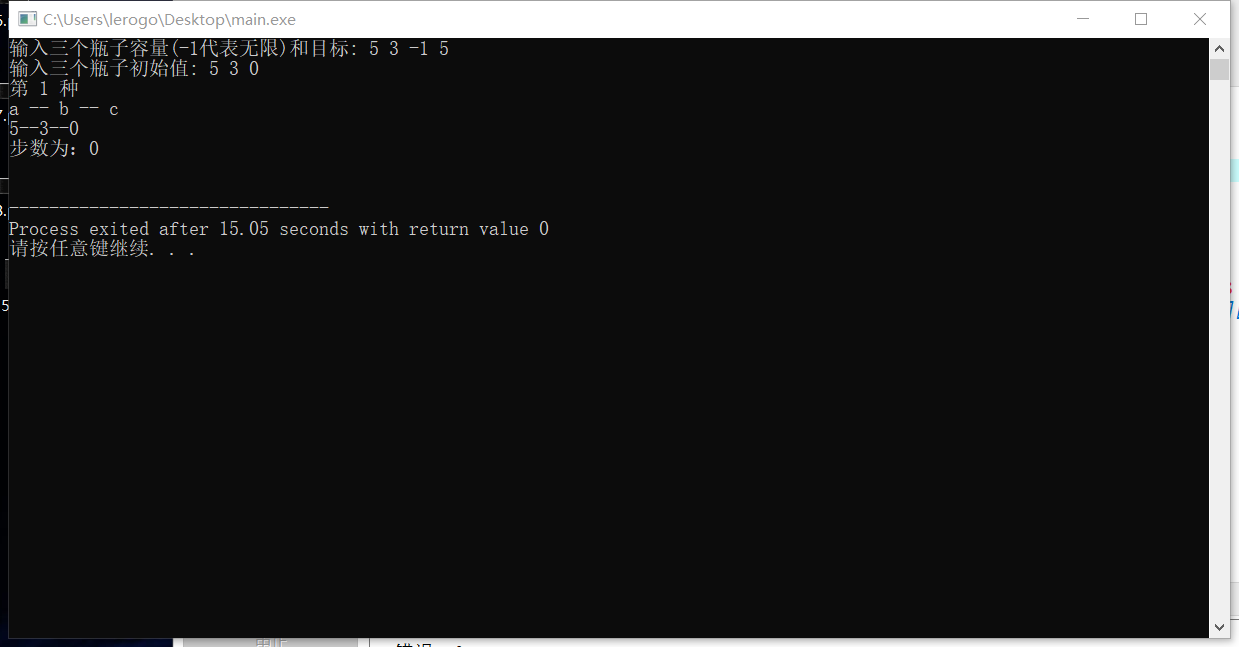
1：

2：

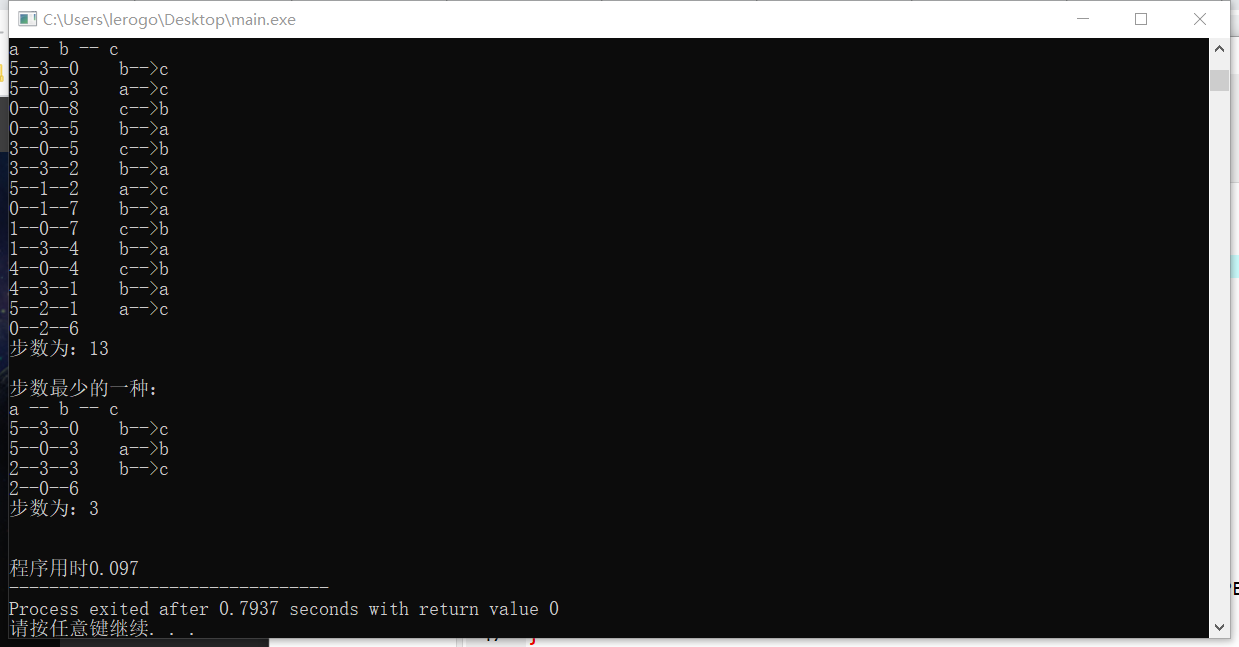
3：

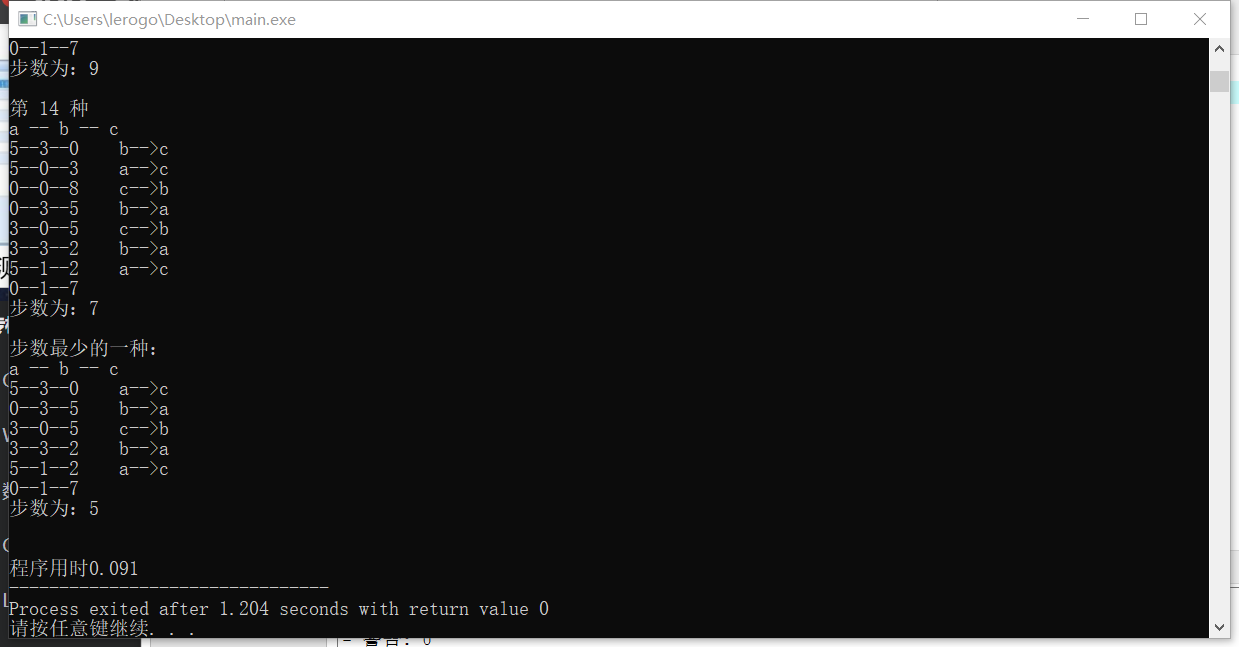
4：

5：



6：



7：

8：

