**算法实验**

陈小龙 20140031 计科2班

实验1 算法基础与递归

# 一. 实验要求

1. 理解递归算法原理及一般应用。

# 二. 实验内容

## 1 设计算法和程序解决“百鸡问题”

## 2 分别用递归与非递归实现

1. 求取两数最大公约数（gcd）：gcd(m,n)=gcd(n,m%n)

2. 计算斐波那契数列；



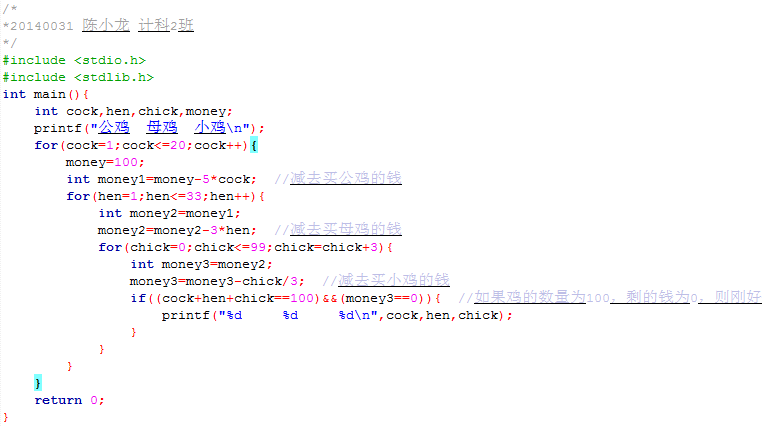
## 3 用递归算法实现汉诺塔问题。



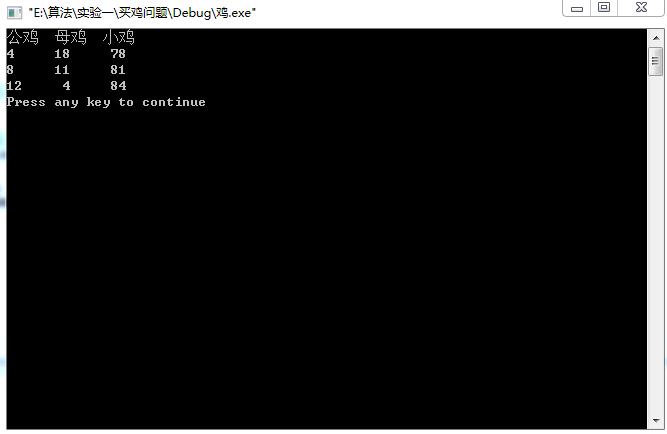
# 三. 实验结果

# 1 百钱买百鸡问题

## C代码

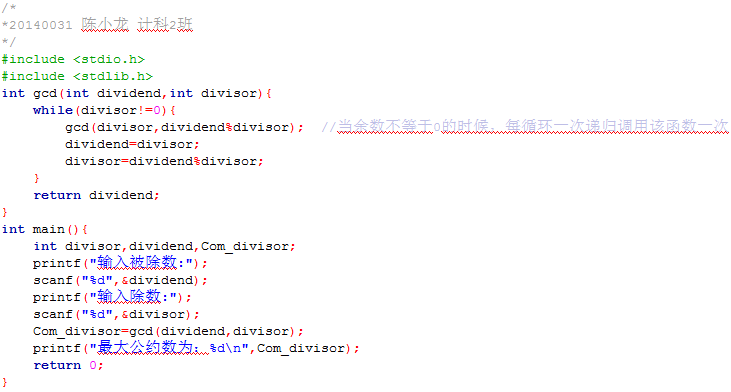


## 运行结果截图



# 2 最大公约数

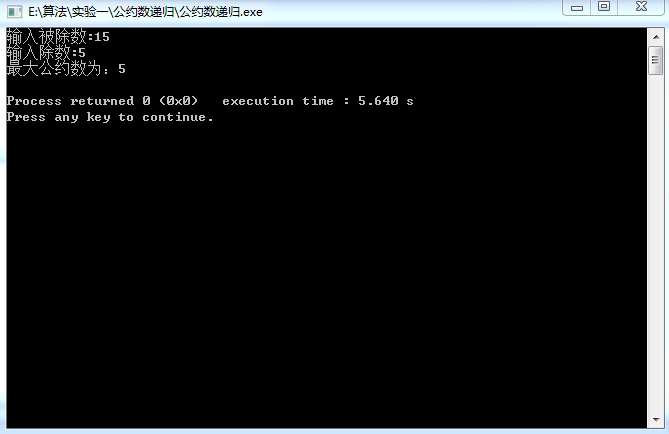
## C递归代码

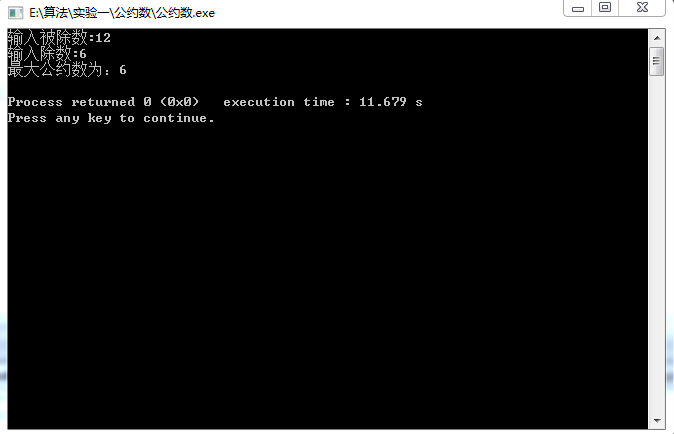


## C非递归代码

## C:\Users\chenxiaolong\AppData\Roaming\Tencent\Users\1061892125\QQ\WinTemp\RichOle\LOT0L_XQY18AEA98}VGRLBW.png

## 运行结果截图





## 算法分析

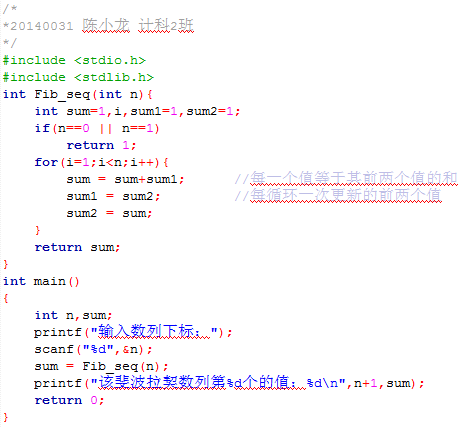
递归实现的总体时间效率要差一些，因为递归调用过程中会进行一些不必要的重复计算，好处就是代码实现简洁方便；而非递归则是依次循环计算出相应的值。

# 3 斐波那契数列

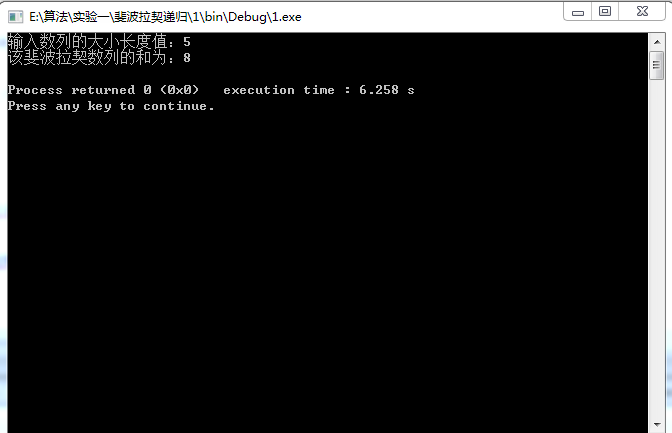
## C递归代码

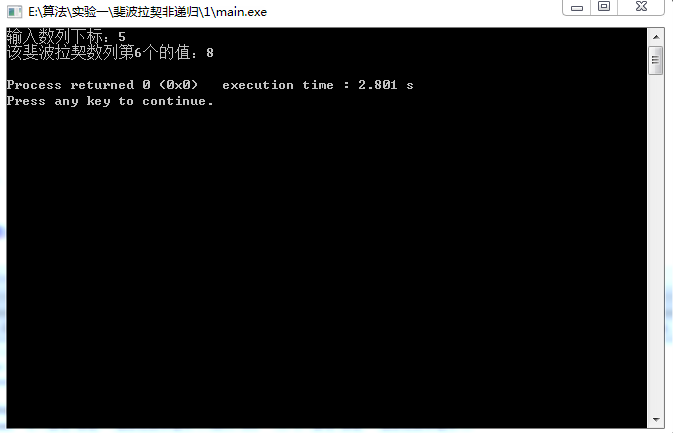
## C:\Users\chenxiaolong\AppData\Roaming\Tencent\Users\1061892125\QQ\WinTemp\RichOle\MGKCJ5WJ6FM@]OXXPWMDTWO.png

## C非递归代码



## 运行结果截图



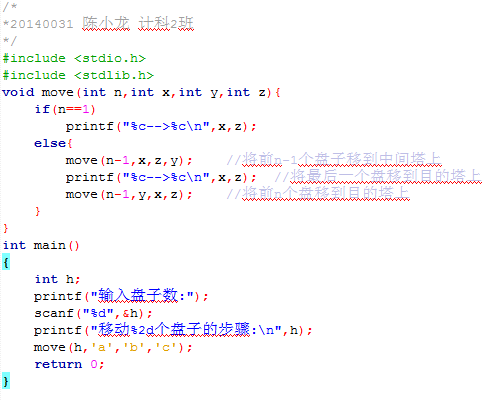


## 算法分析

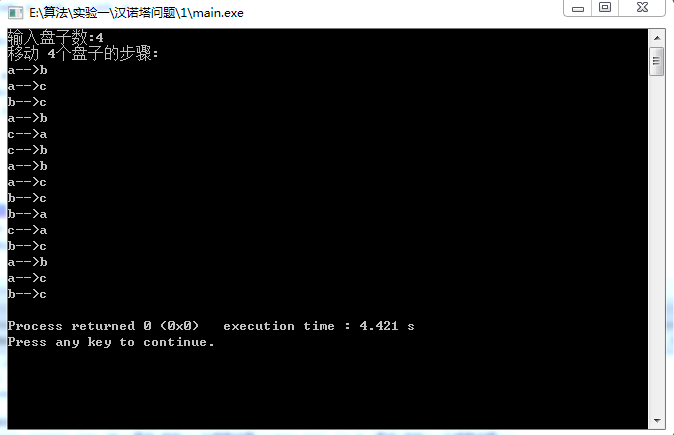
递归实现的总体时间效率要差一些，因为递归调用过程中会进行一些不必要的重复计算，好处就是代码实现简洁方便；而非递归算法则是借助中间变量循环着计算出前面相应的值，从而计算出结果。

# 4 汉诺塔

## C代码



## 运行结果截图



实验2 分治法排序查找

# 一. 实验要求

1. 了解用分治法求解的问题的特点和类型；

2. 编程实现典型的分治算法，对算法进行验证分析。

# 二. 实验内容

## 1.准备知识

### （1） 分治法实现二分查找

给定已按升序排好序的n个元素a[0:n-1]，要在这n个元素中找出一特定元素x。



### （2） 分治法快速排序

**分解**

以*aq*为基准元素将*ap:r*划分成3段*ap:q-1*、*aq*和*aq+1:r*，使得*ap:q-1*中任何元素小于*aq* ，*aq+1:r*中任何元素大于*aq* ；

下标*q*在划分过程中确定。

**递归求解**

递归调用快速排序算法对*ap:q-1*和*aq+1:r*进行排序；

**合并**

对*ap:q-1*和*aq+1:r*的排序是就地进行的，*ap:q-1*和*aq+1:r*排好序后不需要执行任何计算，*ap:r*就已排好序。

快速排序划分算法伪码：

## 2. 实验内容

对乱序整数序列，先用快速排序按非减序排列，再进行二分查找，查找某个元素是否存在，若存在返回匹配的第一个下标位置（从0开始），不存在返回-1。

**示例输入**（第一行表示要查找的数，第二行以后表示数组元素，为编程方便，可限定为20个数，也可不限，自动分配空间）：

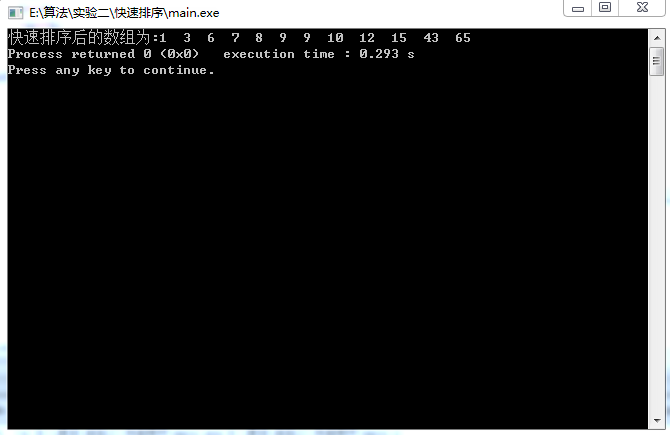
95 56 87 789 54 12 35 11 102 456 7821 2 44 245 56 879 45 82 57 62 30

**示例输出**（只输出数值即可，不许其他文字）：

-1

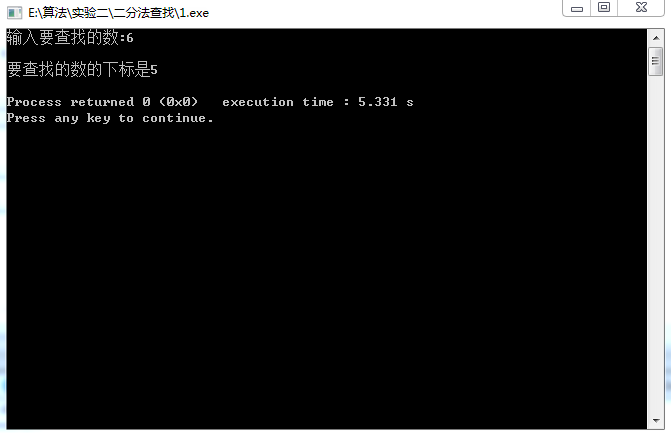
快速排序：





二分查找：





排序查找：

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

int swap**(**int **\***a**,**int **\***b**){**

int c**;**

c**=\***a**;**

**\***a**=\***b**;**

**\***b**=**c**;**

**}**

//这里进行快速排序

int **\***Partition**(**int **\***array**,**int l**,**int r**){**

**if(**l**>=**r**)**

**return** array**;**

int p**=**array**[**l**],**i**=**l**,**j**=**r**;**

**while(**i**<**j**){**

**while(**i**<**j **&&** array**[**j**]>=**p**)**

j**--;**

swap**(&**array**[**i**],&**array**[**j**]);**

**while(**i**<**j **&&** array**[**i**]<=**p**)**

i**++;**

swap**(&**array**[**j**],&**array**[**i**]);**

**}**

Partition**(**array**,**l**,**i**-**1**);**

Partition**(**array**,**i**+**1**,**r**);**

**return** array**;**

**}**

//这里进行折半查找

int ersearch**(**int array**[],**int n**,**int k**){**

int low**=**0**,**high**=**n**-**1**,**m**;**

**while(**low**<=**high**){**

m**=(**low**+**high**)/**2**;**

**if(**k**==**array**[**m**])**

**return** m**;**

**else** **if(**k**>**array**[**m**])**

low**=**m**+**1**;**

**else**

high**=**m**-**1**;**

**}**

**return** **-**1**;**

**}**

int main**()**

**{**

int i**=**0**,**a**[]={**56**,**87**,**789**,**54**,**12**,**35**,**11**,**102**,**456**,**7821**,**2**,**44**,**245**,**56**,**879**,**45**,**82**,**57**,**62**,**30**},**k**;**

int **\***b**;**

b**=**Partition**(**a**,**0**,**19**);**

printf**(**"快速排序后的数组为:"**);**

**for(**i**=**0**;**i**<**20**;**i**++){**

printf**(**"%d "**,**b**[**i**]);**

**}**

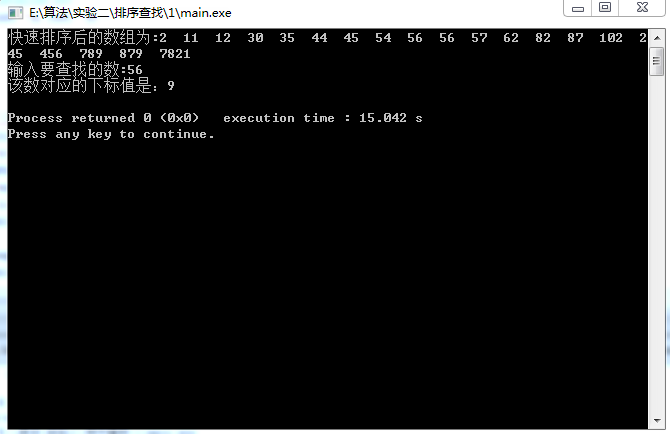
printf**(**"\n输入要查找的数:"**);**

scanf**(**"%d"**,&**k**);**

printf**(**"该数对应的下标值是：%d\n"**,**ersearch**(**a**,**20**,**k**));**

**return** 0**;**

**}**



实验3 堆排序

# 一. 实验要求

1. 理解堆排序算法的原理及一般应用。

3. 编程实现典型的堆排序算法，并对算法进行验证分析。

# 二. 实验内容

## 1 最大堆实现堆排序（从大到小输出）

**示例输入（11个数，第一个数表示元素个数）：**

**10 0 1 2 3 4 5 6 7 8 9**

**示例输出：**

**9 8 7 6 5 4 3 2 1 0**

**堆和优先队列**

在计算机科学中，堆是一种经过排序的树形数据结构，每个结点都有一个值。通常我们所说的堆的数据结构，是指二叉堆。堆的特点是根结点的值最小（或最大），且根结点的两个子树也是一个堆。由于堆的这个特性，常用来实现优先队列，也用于排序算法，如堆排序。

优先队列(priority queue)，不同于先进先出队列的另一种队列。每次从队列中取出的是具有最高优先权的元素。优先队列是0个或多个元素的集合，每个元素都有一个优先权或值。

**最大堆MaxHeap**

在起始索引为 0 的堆中

1)堆的根节点将存放在位置 0

2)节点i的左子节点在位置2\*i+1

3)节点i的右子节点在位置2\*i+2

4)节点i的父节点在位置floor((i-1)/2)

**SiftUp(int i)：**第i个元素上移，最多执行logn次元素比较；对堆中某个关键字进行修改，大于父结点键值时，执行上移操作，恢复堆的性质。

**SiftDown(int i)：**第i个元素下移；对堆中某个关键字进行修改，与其子结点中值较大的一个进行比较，小于子结点键值时，执行下移操作，恢复堆的性质。

**Push(Type x):**在队列中追加元素；堆大小增加1，元素放在堆的末端，执行向上筛选。

**Pop():**弹出元素；弹出最大（最小）堆的首元素，满足优先性质。

**Top():**取得首元素元素值；取得最大（最小）堆的首元素，满足优先性质。

**堆排序**

可利用堆性质，对序列进行排序。假定数组A元素个数为n，数组构建为堆后，根据最大堆性质，根结点A[0]即为最大元素；

交换A[0]、A[n-1]，A[n-1]为最大元素；相当于删掉A[n-1]，堆元素个数减1；交换到A[0]的元素破坏了堆的性质，需对A[0]作下筛操作，恢复堆的结构；此时元素个数为n-1；继续，直至元素个数为1。

**实验结果**

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

//构建最大堆

void HeapAdjust**(**int a**[],**int i**,**int size**){**

int lchild**=**2**\***i**+**1**,**rchild**=**2**\***i**+**2**,**max**=**i**,**temp**;**

**if(**i**<=(**size**)/**2**){**

//获取当前左右孩子较大值的下标

**if(**rchild**<=**size**-**1 **&&** a**[**rchild**]>**a**[**max**])**

max**=**rchild**;**

**if(**lchild**<=**size**-**1 **&&** a**[**lchild**]>**a**[**max**])**

max**=**lchild**;**

**if(**max**!=**i**){**

//更新当前最大值

temp**=**a**[**i**];**

a**[**i**]=**a**[**max**];**

a**[**max**]=**temp**;**

//继续比较构建堆，最终构建最大堆

HeapAdjust**(**a**,**max**,**size**);**

**}**

**}**

**}**

void HeapSort**(**int **\***a**,**int size**)**

**{**

int i**,**temp**;**

**for(**i**=**size**/**2**-**1**;**i**>=**0**;--**i**)**

HeapAdjust**(**a**,**i**,**size**);** //从数组中间到后面的元素进行堆排序构建堆

**for(**i**=**size**-**1**;**i**>=**1**;**i**--)**

**{**

temp**=**a**[**0**];**

a**[**0**]=**a**[**i**];**

a**[**i**]=**temp**;**

HeapAdjust**(**a**,**0**,**i**);** //从数组前一半元素堆排序构建

**}**

**}**

int main**()**

**{**

int i**;**

int a**[]={**9**,**8**,**7**,**6**,**5**,**4**,**3**,**2**,**1**,**0**};**

HeapSort**(**a**,sizeof(**a**)/sizeof(**int**));**

**for(**i**=**0**;**i**<sizeof(**a**)/sizeof(**int**);**i**++)**

**{**

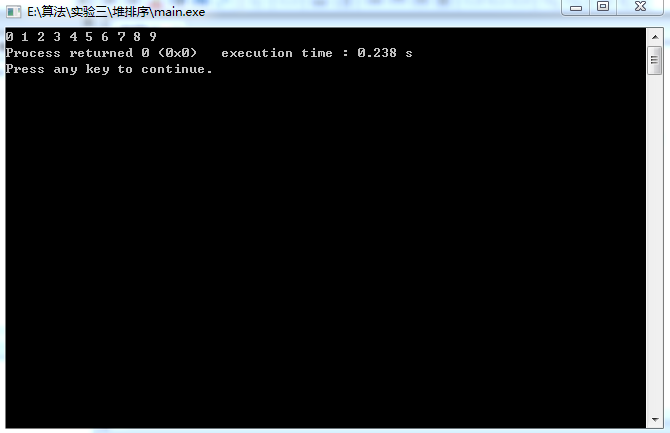
printf**(**"%d "**,**a**[**i**]);**

**}**

//printf("\n\n堆排序后数组为:\n");

**return** 0**;**

**}**

****

**实验4：动态规划法**

**一、实验目的**

掌握动态规划方法求解问题的思想，学会利用动态规划原理求解实际问题。

二、实验内容及要求

1. 有一个箱子的容量为v（正整数，0<=v<=20000）,同时有n个物品（0<n<=30），每个物品有一个体积（正整数）。要求从n个物品中，任取若干个装入箱内，使得箱子的剩余空间为最小。

输入为 ：

箱子的容量

物品的种类

物品的体积

输出为： 箱子的剩余空间

输入输出样例：

输入：

24（一个整数，表示箱子的容量）

6（一个整数，表示有n个物品）

8（接下来n行，分别表示这n个物品的体积）

3

12

7

9

7

输出：0（一个整数，表示箱子的剩余空间）

2. 对于币值为1，3，5且n=9的找零问题，利用动态规划算法求其所有解。

3. 编程实现教材229页习题8.2第1题a问题。

**实验步骤**

箱子问题：

/\*

\*20140031 陈小龙 计科2班

\*/

#include<stdio.h>

#define N 30

int xiangzi**(**int n **,**int V **,**int a**[])**

**{**

int minv**,**t**,**m**=**V**;**

**if(**n**==**0**)**

**{**

**if(**a**[**n**]<=**V**)**

minv**=**V**-**a**[**n**];**

**else**

minv**=**V**;**

**}**

**else**

**{**

t**=**xiangzi**(**n**-**1**,**V**,**a**);**//当前物品不入箱子的决策值

**if(**a**[**n**]<=**V**)**

m**=**xiangzi**(**n**-**1**,**V**-**a**[**n**],**a**);**//当前物品放入箱子的决策值

**if(**t**<**m**)** //比较当前物品放入与不放入的策略效果，选择最后剩余空间最小的一个

minv**=**t**;**

**else**

minv**=**m**;**

**}**

**return** minv**;**

**}**

void main**()**

**{**

int V**;**

int n**,**i**,**m**,**min**;**

int Vo**[**N**];**

printf**(**"箱子的容量V为："**);**

scanf**(**"%d"**,&**V**);**

printf**(**"物品的种类数为："**);**

scanf**(**"%d"**,&**n**);**

printf**(**"物品的体积分别为：\n"**);**

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

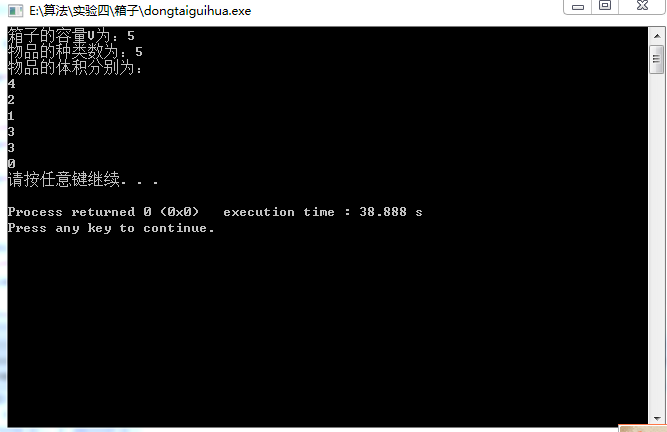
scanf**(**"%d"**,&**Vo**[**i**]);**

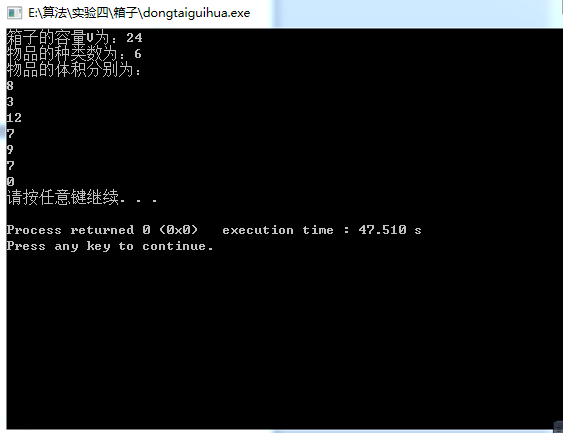
min**=**xiangzi**(**n**-**1**,**V**,**Vo**);**

printf**(**"%d\n"**,**min**);**

system**(**"pause"**);**

**}**





硬币找零问题

动态规划的一个最优解

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

void makeChange**(**int values**[],** int co\_types**,** int total**,** int coinUsed**[],** int cointTrack**[]){**

coinUsed**[**0**]** **=** 0**;**

cointTrack**[**0**]** **=** 0**;**

int i**,**j**;**

**for(**i **=** 1**;** i**<=** total**;** i**++){**

int mincoin **=** i**;** //初始化最小硬币数

int lastin **=** 0**;**

**for(**j **=** 0**;** j **<** co\_types**;** j**++){**

**if(**values**[**j**]** **<=** i**){**

**if(**coinUsed**[**i **-** values**[**j**]]** **+** 1 **<** mincoin**){**//如果当前决策选择得到的最小硬币数小于前一策略的硬币数

mincoin **=** coinUsed**[**i **-** values**[**j**]]** **+** 1**;**//更新当前的策略的最小硬币数

lastin **=** values**[**j**];** //把当前选择的硬币放入硬币选择组中

**}**

**}**

**}**

coinUsed**[**i**]** **=** mincoin**;**

cointTrack**[**i**]** **=** lastin**;**

**}**

**while(**total**>**0**){**

printf**(**"%d "**,**cointTrack**[**total**]);**

total **-=** cointTrack**[**total**];**

**}**

**}**

int main**(**int argc**,** char **\***argv**[])**

**{**

//硬币面值

int value**[]={**1**,**3**,**5**};**

//硬币数组大小

size\_t size**=sizeof(**value**)/sizeof(\***value**);**

//要找零的数

int money**=**9**;**

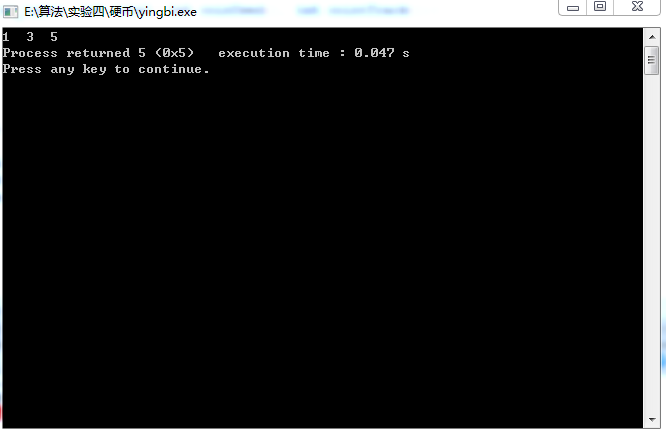
//保存每个面值对应最小值，因为0号位置舍弃，故加1

int coinsUsed**[**money**+**1**];**

int coinsTrack**[**money**+**1**];**

makeChange**(**value**,**size**,**money**,**coinsUsed**,**coinsTrack**);**

**}**



找出所有的解

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_VALUE 9

int array**[]** **=** **{**1**,**3**,**5**,**MAX\_VALUE**};**

int next**[**MAX\_VALUE**]** **=** **{**0**};**

void SegNum**(**int nSum**,** int**\*** pData**,** int nDepth**)**

**{**

**if(**nSum **<** 0**)**

**return;**

**if(**nSum **==** 0**)**

**{**

**for(**int j **=** 0**;** j **<** nDepth**;** j**++){**

printf**(**"%d "**,**pData**[**j**]);**

**}**

printf**(**"\n"**);**

**return;**

**}**

int i **=** **(**nDepth **==** 0 **?** next**[**0**]** **:** pData**[**nDepth**-**1**]);**

**for(;** i **<=** nSum**;)** //当当前的硬币不大于剩下还差待找的钱时

**{**

pData**[**nDepth**++]** **=** i**;** //使用该硬币，添加到使用的硬币集中

SegNum**(**nSum**-**i**,**pData**,**nDepth**);**//待找的钱数减去该硬币面值

nDepth**--;** //每一种策略(决策)选择完成输出后，保存的已选硬币记录便删去，用作下一个策略选择记录

i **=** next**[**i**];**

**}**

**}**

void ShowResult**(**int array**[],**int nLen**)**

**{**

next**[**0**]** **=** array**[**0**];**

int i **=** 0**;**

**for(;** i **<** nLen**-**1**;** i**++)**

next**[**array**[**i**]]** **=** array**[**i**+**1**];**

next**[**array**[**i**]]** **=** array**[**i**]+**1**;**

int pData**[**MAX\_VALUE**];**

SegNum**(**MAX\_VALUE**,**pData**,**0**);**

**delete** pData**;**

**}**

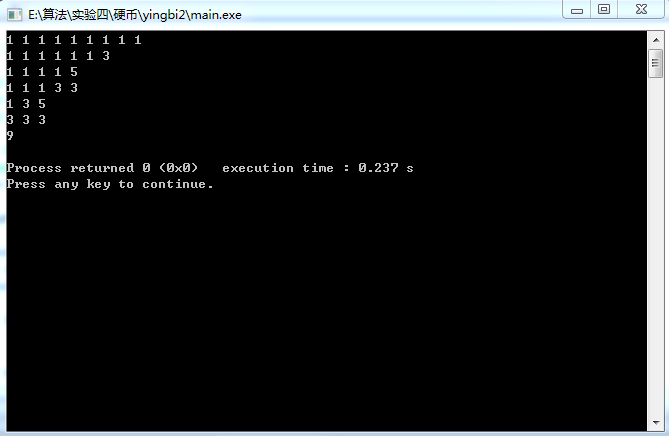
int main**()**

**{**

ShowResult**(**array**,sizeof(**array**)/sizeof(**int**));**

**return** 0**;**

**}**



教材习题8.2

/\*

\*20140031 陈小龙

\*/

#include<stdio.h>

#include<stdlib.h>

int c**[**6**][**7**];** //定义状态表

void knapsack**(**int m**,**int n**)**

**{**

int i**,**j**,**w**[**6**],**v**[**6**];** //这里之所以定义物品数组长度为6是为了方便后面写程序而把第0个物品也放进去了

**for(**i**=**1**;**i**<=**n**;**i**++){**

printf**(**"\n请输入第%d物品重量和对应价值："**,**i**);**

scanf**(**"\n%d,%d"**,&**w**[**i**],&**v**[**i**]);**

**}**

**for(**i**=**0**;**i**<**6**;**i**++)**

**for(**j**=**0**;**j**<**7**;**j**++)**

c**[**i**][**j**]=**0**;** //把物品价值重量表（状态表）全部初始化为0

**for(**i**=**1**;**i**<=**n**;**i**++)**

**for(**j**=**1**;**j**<=**m**;**j**++){**

**if(**w**[**i**]<=**j**){** //若当前剩余载重不小于当前物品重量

**if(**v**[**i**]+**c**[**i**-**1**][**j**-**w**[**i**]]>**c**[**i**-**1**][**j**])**//状态转移方程

c**[**i**][**j**]=**v**[**i**]+**c**[**i**-**1**][**j**-**w**[**i**]];** //若放入当前物品后的总价值大于不放入的情况，那就放入(添加到表中)

**else**

c**[**i**][**j**]=**c**[**i**-**1**][**j**];** //否则就不放入，把不放入放入情况(此时的价值)放入表中

**}**

**else** c**[**i**][**j**]=**c**[**i**-**1**][**j**];** //若当前剩余载重小于当前物品重量，不放入该物品，并把相应价值写入表中

**}**

**}**

int main**()**

**{**

int m**,**n**;**

int i**,**j**;**

printf**(**"陈小龙 20140031 计科2班\n\n"**);**

printf**(**"请输入箱子重量和物品数："**);**

scanf**(**"%d,%d"**,&**m**,&**n**);**

knapsack**(**m**,**n**);**

printf**(**"\n"**);**

printf**(**"动态规划的表为：\n"**);**

**for(**i**=**0**;**i**<**6**;**i**++)**

**for(**j**=**0**;**j**<**7**;**j**++){**

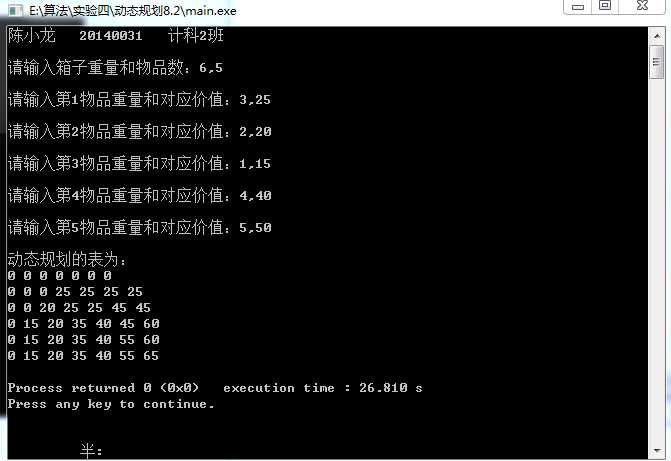
printf**(**"%d "**,**c**[**i**][**j**]);** //显示表中的信息

**if(**j**==**6**)**

printf**(**"\n"**);** //每行显示剩余载重为0-6的情况

**}**

**}**



实验5 贪心算法——Prim最小生成树

# 一. 实验要求

1. 理解贪心算法的原理及一般应用。

2. 思考贪心算法和动态规划法的区别。

3. 编程实现典型的贪心算法，并对算法进行验证分析。

# 二. 实验内容

## 2. 最小生成树算法

**Prim算法**

设G=(V,E)是连通带权图，V={1,2,…,n}。构造G的最小生成树的Prim算法的基本思想是：

(1)置S={1}

(2)只要S是V的真子集，就作如下的贪心选择

选取满足条件i ∈ S，j ∈ V-S，且c[i][j]最小的边，将顶点j添加到S中。

一直到S=V时为止。

(2)选取到的所有边恰好构成G的一棵最小生成树。

# 三. 实验结果

## 1 Prim

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

#define MAXVEX 50 //图中最大节点数

#define MAXIMUN 65535 //定义为最大数

**typedef** struct**{** //定义邻接矩阵的数据结构

char vexs**[**MAXVEX**];**

int edges**[**MAXVEX**][**MAXVEX**];**

int vexNum**,** adgeNum**;** //分别表示图中节点个数和边的条数

**}**AGraph**;**

//Prim算法求解最小生成树

void prim**(**AGraph **\***g**){**

int U**[**MAXVEX**],** S**[**MAXVEX**],** n **=**0**;** //U存放以加入的顶点,S存放剩余的顶点,已加入U中的顶点个数

int j**,**min **=** 0**;** //用于保存生成树的总权值

**for(**int i **=** 0**;** i **<** g**->**vexNum**;++**i**){** //将图中的顶点存入S中，只存储顶点的编号.

S**[**i**]** **=** i**;**

**}**

U**[**0**]** **=** 0**;** //将任意一个顶点放入集合U中，这里假设放入0号顶点

**++**n**;**

S**[**0**]** **=** **-**1**;** //同时从S中去掉0号顶点，这里赋值为-1

printf**(**"\n最小生成树由这些边构成：\n"**);**

**for(**j **=** 1**;** j **<** g**->**vexNum**;** **++**j**){**

int mintemp **=** MAXIMUN**;**

int indexK**,** indexP**;**

//贪心法寻找最优加入顶点并用index记录其位置

**for(**int k **=** 0**;** k **<** n**;** **++**k**){**

**for(**int p **=** 0**;** p **<** g**->**vexNum**;** **++**p**)**

**if((**S**[**p**]** **!=** **-**1**)** **&&** **(**mintemp **>**g**->**edges**[**U**[**k**]][**p**])){**

indexK **=** U**[**k**];**

indexP **=** p**;**

mintemp **=** g**->**edges**[**indexK**][**indexP**];**

**}**

**}**

U**[**n**++]** **=** indexP**;** //将顶点加入U中

S**[**indexP**]** **=** **-**1**;** //将该顶点从S中去掉

min **+=** mintemp**;**

printf**(**"%d->%d "**,**g**->**vexs**[**indexK**],**g**->**vexs**[**indexP**]);**

**}**

printf**(**"\n最小生成树的总权重为：%d\n"**,**min**);**

**}**

void createAG**(**AGraph **\***g**)** //创建无向网

**{**

int vNum**,** aNum**;** //分别代表要创建的图的节点数和边数

int start**,** end**;** //start->end表示节点start和end之间有一条边

int weight**;** //边的权重

printf**(**"请输入图中节点的个数和边的条数:"**);**

scanf**(**"%d %d"**,&**vNum**,&**aNum**);**

printf**(**"请输入%d个节点的信息："**,**vNum**);**

//创建节点

**for(**int i **=** 0**;** i **<** vNum**;** **++**i**){**

scanf**(**"%d"**,&**g**->**vexs**[**i**]);**

**}**

printf**(**"这里输出节点编号及其存储的节点信息\n"**);**

**for(**int k **=** 0**;** k **<** vNum**;** **++**k**){**

printf**(**"%d:%d "**,**k**,**g**->**vexs**[**k**]);**

**}**

printf**(**"\n"**);**

//初始化无向图的边信息

**for(**int m **=** 0**;** m **<** vNum**;** **++**m**)**

**for(**int n **=** 0**;** n **<** vNum**;** **++**n**){**

g**->**edges**[**m**][**n**]** **=** MAXIMUN**;**

**}**

//创建无向图的边信息

**for(**int j **=** 0**;** j **<** aNum**;** **++**j**){**

printf**(**"\n请输入第%d条边的start和end节点和边的权重weight:"**,**j**+**1**);**

scanf**(**"%d %d %d"**,&**start**,&**end**,&**weight**);**

g**->**edges**[**start**][**end**]** **=** weight**;**

g**->**edges**[**end**][**start**]** **=** weight**;**

**}**

//初始化vexNum和adgeNum

g**->**vexNum **=** vNum**;**

g**->**adgeNum **=** aNum**;**

**}**

int main**(){**

void createAG**(**AGraph **\*);**

void prim**(**AGraph**\*);**

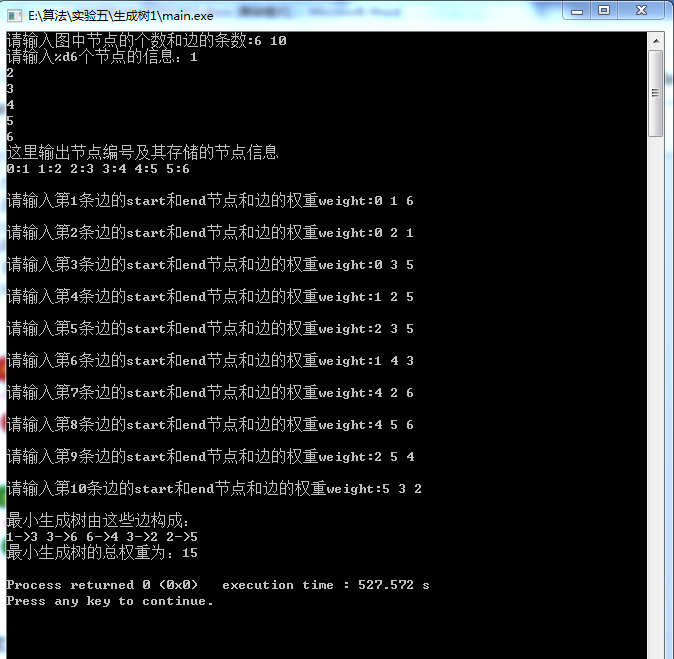
AGraph g**;**

createAG**(&**g**);**

prim**(&**g**);**

**return** 0**;**

**}**



实验6 回溯法与分支限界法

# 实验要求

1 掌握回溯法求解问题的思想，学会利用其原理求解相关问题。

2 理解分支限界算法的优先搜寻原理及一般应用。

3 编程实现典型分支限界算法，理解“分支”、“限界”思想，并对算法进行验证分析。

# 二. 实验内容

1. **八皇后问题**

八皇后问题是一个古老而著名的问题，是回溯算法的典型例题。该问题是十九世纪著名的数学家高斯1850年提出：在8×8格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。

高斯认为有76种方案。1854年在柏林的象棋杂志上不同的作者发表了40种不同的解，后来有人用图论的方法解出92种结果。

要求编程实现回溯法，并进行验证，并使其能扩展到任意的皇后数的情况，同时对源程序给出详细的注释。

**2. 分支限界法0-1背包问题**

示例输入（规定物品数量为10，背包容量为50，输入为20个数，**前十个为物品重量，后十个数为物品价值**）：

**12 3 11 5 6 8 9 4 7 10**6 2 7 3 2 9 8 10 4 5  
示例输出（**最大价值**）：

44

**三、实验步骤**

分支限界法0-1背包问题

/\*

\*20140031 陈小龙 计科2班

\*/

#include<iostream>

#include<queue>

**using** **namespace** std**;**

**typedef** struct treenode**{**

int weight**;**

int value**;**

int level**;**

int flag**;**

**}**treenode**;**

queue**<**struct treenode**>** que**;**

int enQueue**(**int w**,**int v**,**int level**,**int flag**,**int n**,**int**\*** bestvalue**)**

**{**

treenode node**;**

node**.**weight **=** w**;**

node**.**value **=** v**;**

node**.**level **=** level**;**

node**.**flag **=** flag**;**

**if** **(**level **==** n**)**

**{**

**if** **(**node**.**value **>** **\***bestvalue**)**

**{**

**\***bestvalue **=** node**.**value**;**

**}**

**return** 0**;**

**}else**

**{**

que**.**push**(**node**);**

**}**

**}**

//w为重量数组，v为价值数组，n为物品个数，c为背包容量，bestValue为全局最大价值

int bbfifoknap**(**int w**[],**int v**[],**int n**,**int c**,**int**\*** bestValue**)**

**{**

//初始化结点

int i**=**1**;**

treenode tag**,** livenode**;**

livenode**.**weight **=** 0**;**

livenode**.**value **=** 0**;**

livenode**.**level **=** 1**;**

livenode**.**flag **=** 0**;**//初始为0

tag**.**weight **=** **-**1**;**

tag**.**value **=** 0**;**

tag**.**level **=** 0**;**

tag**.**flag **=** 0**;**//初始为0

que**.**push**(**tag**);**

**while** **(**1**)**

**{**

**if** **(**livenode**.**weight **+** w**[**i**-**1**]** **<=** c**)**

**{**

enQueue**(**livenode**.**weight **+** w**[**i**-**1**],** livenode**.**value **+** v**[**i**-**1**],** i**,** 1**,**n**,**bestValue**);**

**}**

enQueue**(**livenode**.**weight**,**livenode**.**value**,** i**,** 0**,**n**,**bestValue**);**

livenode **=** que**.**front**();**

que**.**pop**();**

**if** **(**livenode**.**weight **==** **-**1**)**

**{**

**if** **(**que**.**empty**()** **==** 1**)**

**{**

**break;**

**}**

livenode **=** que**.**front**();**

que**.**pop**();**

que**.**push**(**tag**);**

i**++;**

**}**

**}**

**return** 0**;**

**}**

int main**()**

**{**

int w**[]** **=** **{**12**,**3**,**11**,**5**,**6**,**8**,**9**,**4**,**7**,**10**};**

int v**[]** **=** **{**6**,**2**,**7**,**3**,**2**,**9**,**8**,**10**,**4**,**5**};**

int n **=** 10**;**

int c **=** 50**;**

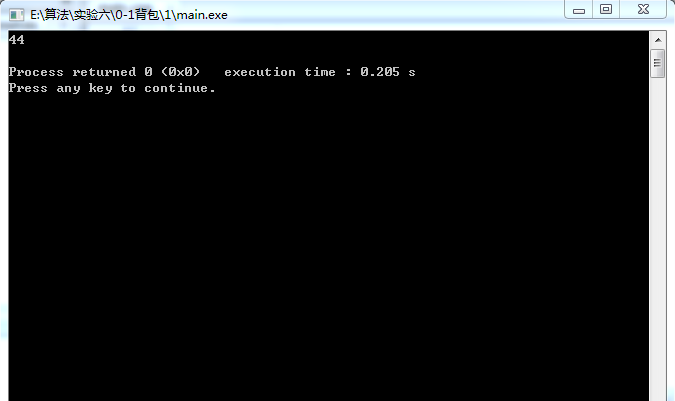
int bestValue**=**0**;**

bbfifoknap**(**w**,**v**,**n**,**c**,&**bestValue**);**

cout **<<** bestValue**<<**endl**;**

**return** 0**;**

**}**



八皇后问题

/\*

\*20140031 陈小龙 计科2班

\*/

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int x**[**100**];**

bool place**(**int k**)** //检测皇后k放置在x[k]列是否发生冲突

**{**

int i**;**

**for(**i**=**1**;**i**<**k**;**i**++)**

**if(**x**[**k**]==**x**[**i**]||**abs**(**k**-**i**)==**abs**(**x**[**k**]-**x**[**i**]))**

**return** **false;**

**return** **true;**

**}**

void queue**(**int n**)**

**{**

int i**,**k**;**

**for(**i**=**1**;**i**<=**n**;**i**++)**

x**[**i**]=**0**;**

k**=**1**;**

**while(**k**>=**1**)**

**{**

x**[**k**]=**x**[**k**]+**1**;** //在下一列放置第k个皇后

**while(**x**[**k**]<=**n**&&!**place**(**k**))**

x**[**k**]=**x**[**k**]+**1**;** //搜索下一列

**if(**x**[**k**]<=**n**&&**k**==**n**)** //得到一个输出

**{**

**for(**i**=**1**;**i**<=**n**;**i**++)**

printf**(**"%d "**,**x**[**i**]);**

printf**(**"\n"**);**

//若return则只求出其中一种解，若不return则可以继续回溯，求出全部的可能的解

//return;

**}**

**else** **if(**x**[**k**]<=**n**&&**k**<**n**)**

k**=**k**+**1**;** //放置下一个皇后

**else**

**{**

x**[**k**]=**0**;** //重置x[k],回溯

k**=**k**-**1**;**

**}**

**}**

**}**

int main**()**

**{**

int n**;**

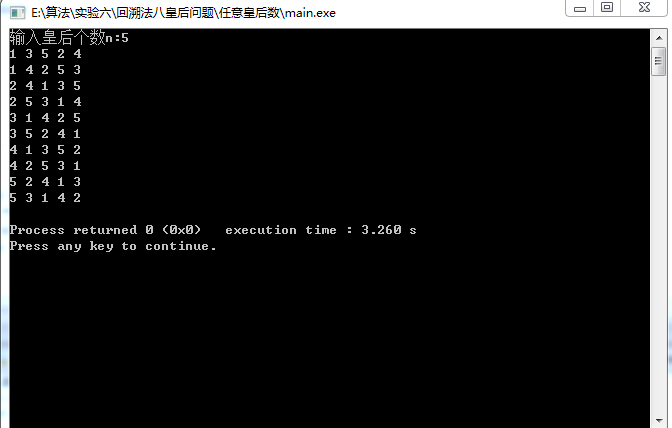
printf**(**"输入皇后个数n:"**);**

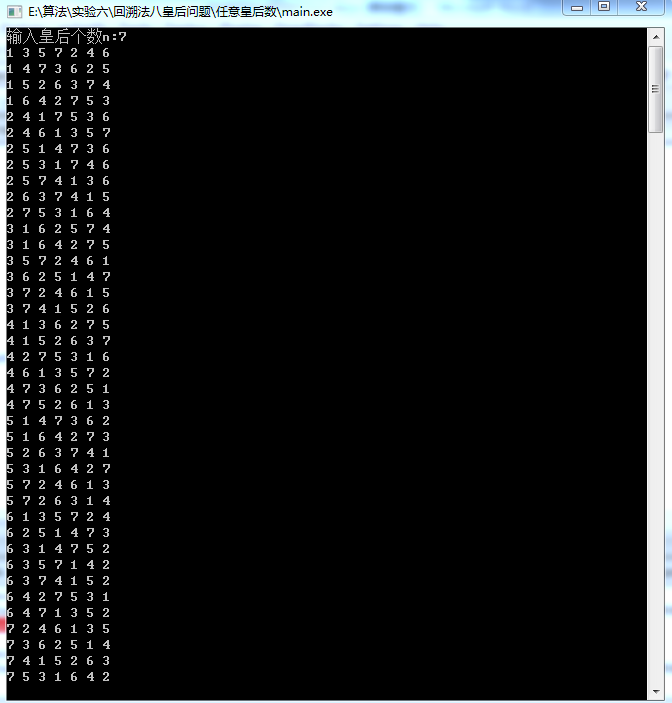
scanf**(**"%d"**,&**n**);**

queue**(**n**);**

**return** 0**;**

**}**





实验7 解非线性方程的算法

# 一. 实验要求

1. 理解平分法求解非线性方程的思想；

2. 理解试位法求解非线性方程的思想；

3. 理解牛顿法求解非线性方程的思想。

# 二. 实验内容

1. 用平分法求方程x3+x-1=0的根，而且绝对误差要小于10-2。

2. 用试位法求方程x3+x-1=0的根，而且绝对误差要小于10-2。

3. 用牛顿法求方程x3+x-1=0的根，而且绝对误差要小于10-2。

# 三. 实验结果

平分法：

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

#include <float.h>

#include <math.h>

int Bisection**(**float a**,**float b**,**float eps**,**int N**){**

float x**,**fval**;**

int n**;**

**while(**n **<=** N**){**

x **=** **(**a **+** b**)/**2**;** //计算中间点

fval **=** x**\***x**\***x**+**x**-**1**;** //计算对应y值

float fa**;**

fa **=** a**\***a**\***a**+**a**-**1**;** //计算左边点的y值

**if(**fabs**(**x**-**a**)** **<** fabs**(**eps**)){** //如果满足精度要求，则返回

printf**(**"解的区间为："**);**

printf**(**"(%f,%f)\n"**,**a**,**x**);**

printf**(**"解的结果为：%f"**,**fval**);**

**return** n**;**

**}**

**if(**fval **==** 0.0**){** //如果刚好等于，则返回

printf**(**"解为：%f"**,**x**);**

printf**(**"解的结果为：%f"**,**fval**);**

**return** n**;**

**}**

**if(**fval**\***fa **<** 0.0**)** //如果中间点与左边点异号，则以中间点更新右边的点

b **=** x**;**

**else**

a **=** x**;** //否则更新左边的点

n**++;**

**}**

**return** n**;**

**}**

int main**(){**

float a **=** 0.0**,**b **=** 1.0**,**eps **=** 0.01**,**i**;**

int N**,**n**;**

i **=** **(**b**-**a**)/**eps**;**

//N = log(i)/log(2)+1;

//printf("N=%d",N);

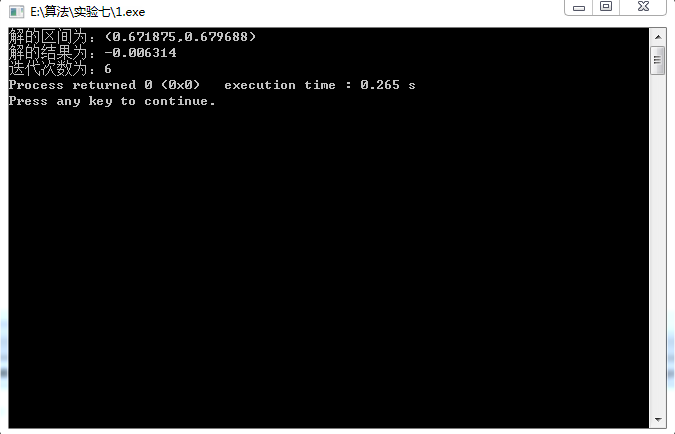
N **=** 50**;**

n **=** Bisection**(**a**,**b**,**eps**,**N**);**

printf**(**"\n迭代次数为：%d"**,**n**);**

**return** 0**;**

**}**



试位法：

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

#include <float.h>

#include <math.h>

struct point**{**

float x**;**

float y**;**

**}**a**,**b**;**

int Bisection**(**struct point a**,**struct point b**,**float eps**,**int N**){**

float x**,**fval**;**

int n**;**

**while(**n **<=** N**){**

x **=** **((**b**.**y**-**a**.**y**)\***a**.**x**/(**b**.**x**-**a**.**x**)-**a**.**y**)\*((**b**.**x**-**a**.**x**)/(**b**.**y**-**a**.**y**));** //计算截点的横坐标

fval **=** x**\***x**\***x**+**x**-**1**;** //计算以改横坐标为x值的方程的y值

float fa **=** a**.**x**\***a**.**x**\***a**.**x**+**a**.**x**-**1**;** //计算根的左边的另一点的纵坐标

**if(**fabs**(**x**-**a**.**x**)** **<** fabs**(**eps**)){** //满足精度要求则返回

printf**(**"解的区间为："**);**

printf**(**"(%f,%f)\n"**,**a**.**x**,**b**.**x**);**

printf**(**"解的结果为：%f"**,**fval**);**

**return** n**;**

**}**

**if(**fval **==** 0.0**){**

printf**(**"解的横坐标为：%f"**,**x**);**

printf**(**"解的结果为：%f"**,**fval**);**

**return** n**;**

**}**

**if(**fval**\***fa **<** 0.0**){**

a**.**x **=** x**;** //如果新的点与左边点异号，则用截点更新左边点

a**.**y **=** fval**;**

**}**

**else{**

b**.**x **=** x**;** //否则用截点更新右边点

b**.**y **=** fval**;**

**}**

n**++;**

**}**

**return** n**;**

**}**

int main**(){**

float eps **=** 0.01**;**

a**.**x **=** 0.0**,**a**.**y **=** **-**1.0**,**b**.**x **=** 1.0**,**b**.**y **=** 1.0**;**

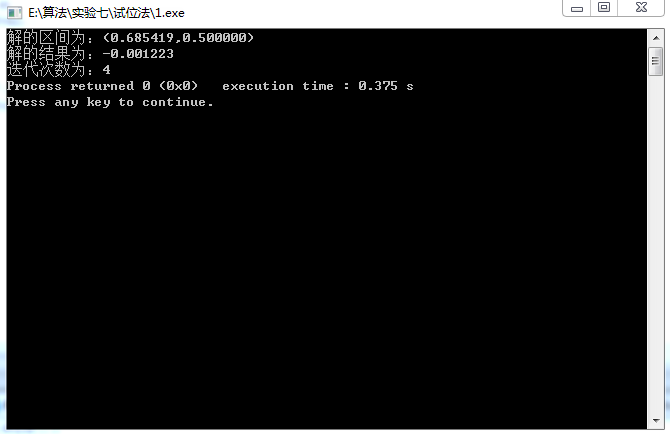
int N**,**n**;**

N **=** 50**;**

n **=** Bisection**(**a**,**b**,**eps**,**N**);**

printf**(**"\n迭代次数为：%d"**,**n**);**

**return** 0**;**

**}**

牛顿法：

/\*

\*20140031 陈小龙 计科2班

\*/

#include <stdio.h>

#include <stdlib.h>

#include <float.h>

#include <math.h>

struct point**{**

float x**;**

float y**;**

**}**a**;**

int niudun**(**struct point a**,**float eps**,**int N**){**

float fval**,**x**;**

int n **=** 1**;**

**while(**n **<=** N**){**

x **=** a**.**x**-**a**.**y**/(**3**\***a**.**x**\***a**.**x**+**1**);** //求截点的横坐标

fval **=** x**\***x**\***x**+**x**-**1**;**

**if(**fabs**(**x**-**a**.**x**)** **<** fabs**(**eps**)** **||** fval **==** 0.0**){**//刚好找到或满足精度要求就停止迭代

printf**(**"解的横坐标为：%f\n"**,**x**);**

printf**(**"解的结果为：%f"**,**fval**);**

**return** n**;**

**}**

**else{**

a**.**y **=** fval**;** //以截点的横坐标新的切点的横坐标，纵坐标

a**.**x **=** x**;**

**}**

n**++;**

**}**

**return** n**;**

**}**

int main**(){**

float eps **=** 0.01**;**

a**.**x **=** 1.0**,**a**.**y **=** 1.0**;**

int N**,**n**;**

N **=** 50**;**

n **=** niudun**(**a**,**eps**,**N**);**

printf**(**"\n迭代次数为：%d"**,**n**);**

**return** 0**;**

**}**

