循环赛日程表（用来说明算法导论上的题目！！）

2016年5月4日　　19:16

设有$n=2^k$个选手参加比赛，要求设计一个满足一下要求的比赛日程表:

（1）每个选手必须与其他的n-1个选手个比赛一次；

（2）每个选手每天只能赛一次 。

按此要求可以把比赛日程表设计成一个n行n-1列的二维表，其中第i行第j列表示第i个选手在 第j天比赛的选手。

代码：（分治策略）

```
#include<stdio.h>
#include<math.h>

void gametable(int k)
{
    int a[100][100];
    int n,temp,i,j,p,t;
    n=2;//k=0两个参赛选手日程可以直接求得
    a[1][1]=1;a[1][2]=2;
    a[2][1]=2;a[2][2]=1;
    for(t=1;t<k;t++)//迭代处理，依次处理2^n....2^k个选手的比赛日程
    {
        temp=n;n=n*2;//填左下角元素
        for(i=temp+1;i<=n;i++)
            for(j=1;j<=temp;j++)
                a[i][j]=a[i-temp][j]+temp;//左下角和左上角元素的对应关系
        for(i=1;i<=temp;i++)//将左下角元素抄到右上角
            for(j=temp+1;j<=n;j++)
                a[i][j]=a[i+temp][(j+temp)%n];
        for(i=temp+1;i<=n;i++)//将左上角元素抄到右下角
            for(j=temp+1;j<=n;j++)
                a[i][j]=a[i-temp][j-temp];

    }
    printf("参赛人数为:%d\n(第i行第j列表示和第i个选手在第j天比赛的选手序号)\n",n);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            printf("%d ",a[i][j]);
                if(j==n)
                printf("\n");

        }
}

int main()
{
    int k;
    printf("比赛选手个数为n(n=2^k)，请输入参数K(K>0):\n");
    scanf("%d",&k);
    if(k!=0)
    gametable(k);

}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5090287>

五子棋

2016年5月4日　　19:17

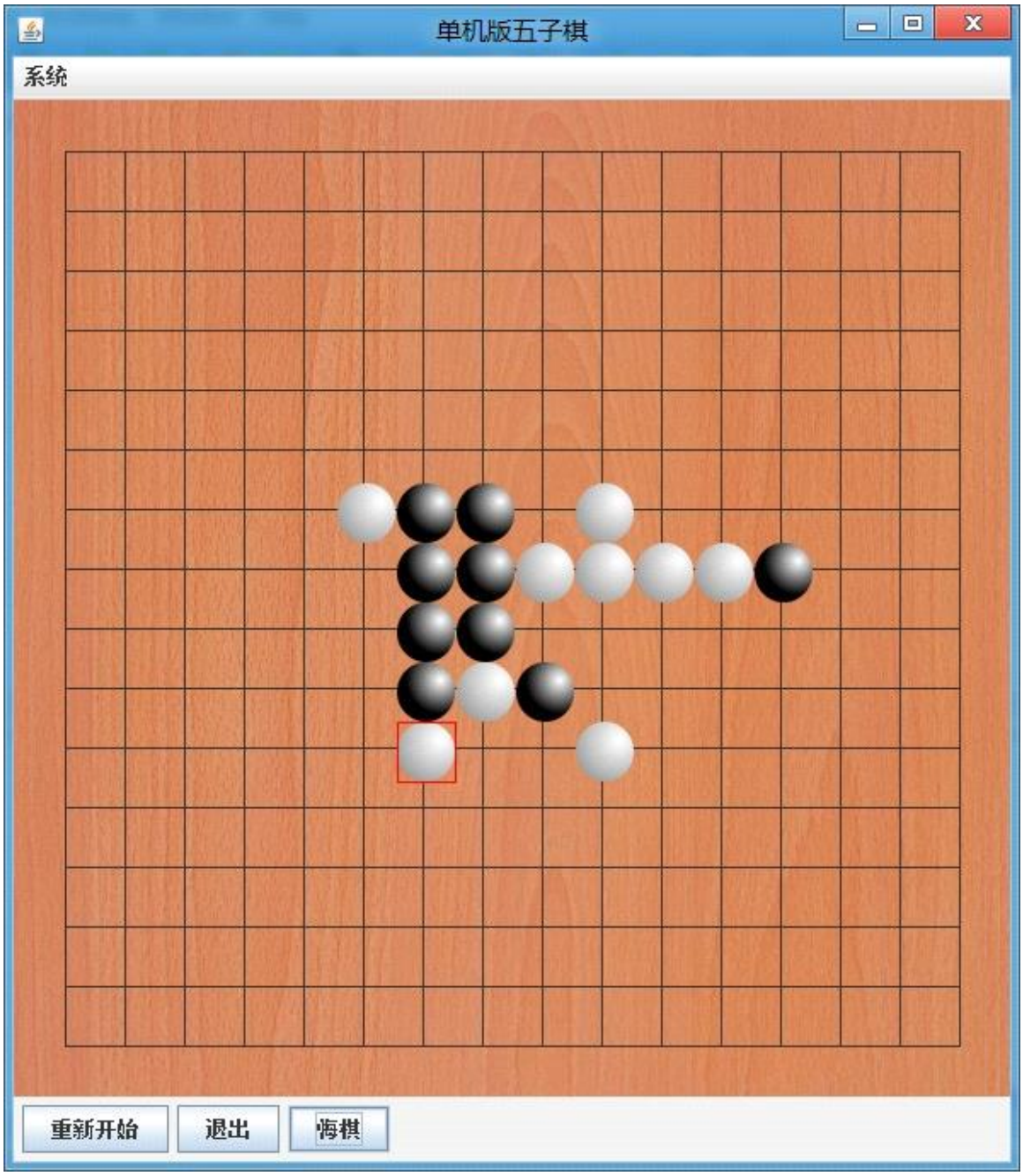
一、实践目标：

1.掌握JavaGUI界面设计

2.掌握鼠标事件的监听（MouseListener，MouseMotionListener）

二、实践内容：

设计一个简单的五子棋程序，能够实现五子棋下棋过程。如下图所示



五子棋运行界面

```
1 package cn.edu.ouc.fiveChess;
2
```

```
 3 import java.awt.Color;
 4 import java.awt.Cursor;
 5 import java.awt.Dimension;
 6 import java.awt.Graphics;
 7 import java.awt.Graphics2D;
 8 import java.awt.Image;
 9 import java.awt.RadialGradientPaint;
10 import java.awt.RenderingHints;
11 import java.awt.Toolkit;
12 import java.awt.event.MouseEvent;
13 import java.awt.event.MouseListener;
14 import java.awt.event.MouseMotionListener;
15 import java.awt.geom.Ellipse2D;
16
17 import javax.swing.*;
18 /**
19  * 五子棋--棋盘类
20  */
21
22 public class ChessBoard extends JPanel implements MouseListener {
23    public static final int MARGIN=30;//边距
24    public static final int GRID_SPAN=35;//网格间距
25    public static final int ROWS=15;//棋盘行数
26    public static final int COLS=15;//棋盘列数
27
28    Point[] chessList=new Point[(ROWS+1)*(COLS+1)];//初始每个数组元素为null
29    boolean isBlack=true;//默认开始是黑棋先
30    boolean gameOver=false;//游戏是否结束
31    int chessCount;//当前棋盘棋子的个数
32    int xIndex,yIndex;//当前刚下棋子的索引
33
34    Image img;
35    Image shadows;
36    Color colortemp;
37    public ChessBoard(){
38
39       // setBackground(Color.blue);//设置背景色为橘黄色
40        img=Toolkit.getDefaultToolkit().getImage("board.jpg");
41        shadows=Toolkit.getDefaultToolkit().getImage("shadows.jpg");
42        addMouseListener(this);
43        addMouseMotionListener(new MouseMotionListener(){
44            public void mouseDragged(MouseEvent e){
45
46            }
47
48            public void mouseMoved(MouseEvent e){
49              int x1=(e.getX()-MARGIN+GRID_SPAN/2)/GRID_SPAN;
50              //将鼠标点击的坐标位置转成网格索引
51              int y1=(e.getY()-MARGIN+GRID_SPAN/2)/GRID_SPAN;
52              //游戏已经结束不能下
53              //落在棋盘外不能下
54              //x，y位置已经有棋子存在，不能下
55              if(x1<0||x1>ROWS||y1<0||y1>COLS||gameOver||findChess(x1,y1))
56                  setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
57              //设置成默认状态
58              else setCursor(new Cursor(Cursor.HAND_CURSOR));
59
60            }
61        });
62    }
63
64
65
66 //绘制
67    public void paintComponent(Graphics g){
68
69        super.paintComponent(g);//画棋盘
70
71        int imgWidth= img.getWidth(this);
72        int imgHeight=img.getHeight(this);//获得图片的宽度与高度
73        int FWidth=getWidth();
74        int FHeight=getHeight();//获得窗口的宽度与高度
```

```
 75         int x=(FWidth-imgWidth)/2;
 76         int y=(FHeight-imgHeight)/2;
 77         g.drawImage(img, x, y, null);
 78 
 79 
 80         for(int i=0;i<=ROWS;i++){//画横线
 81             g.drawLine(MARGIN, MARGIN+i*GRID_SPAN, MARGIN+COLS*GRID_SPAN,
MARGIN+i*GRID_SPAN);
 82         }
 83         for(int i=0;i<=COLS;i++){//画竖线
 84             g.drawLine(MARGIN+i*GRID_SPAN, MARGIN, MARGIN+i*GRID_SPAN,
MARGIN+ROWS*GRID_SPAN);
 85 
 86         }
 87 
 88         //画棋子
 89         for(int i=0;i<chessCount;i++){
 90             //网格交叉点x，y坐标
 91             int xPos=chessList[i].getX()*GRID_SPAN+MARGIN;
 92             int yPos=chessList[i].getY()*GRID_SPAN+MARGIN;
 93             g.setColor(chessList[i].getColor());//设置颜色
 94            // g.fillOval(xPos-Point.DIAMETER/2, yPos-Point.DIAMETER/2,
 95                         //Point.DIAMETER, Point.DIAMETER);
 96             //g.drawImage(shadows, xPos-Point.DIAMETER/2, yPos-Point.DIAMETER/2,
Point.DIAMETER, Point.DIAMETER, null);
 97             colortemp=chessList[i].getColor();
 98             if(colortemp==Color.black){
 99                 RadialGradientPaint paint = new RadialGradientPaint(xPos-
Point.DIAMETER/2+25, yPos-Point.DIAMETER/2+10, 20, new float[]{0f, 1f}
100                 , new Color[]{Color.WHITE, Color.BLACK});
101                 ((Graphics2D) g).setPaint(paint);
102                 ((Graphics2D)
g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
103                 ((Graphics2D)
g).setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_DEFAULT);
104 
105             }
106             else if(colortemp==Color.white){
107                 RadialGradientPaint paint = new RadialGradientPaint(xPos-
Point.DIAMETER/2+25, yPos-Point.DIAMETER/2+10, 70, new float[]{0f, 1f}
108                 , new Color[]{Color.WHITE, Color.BLACK});
109                 ((Graphics2D) g).setPaint(paint);
110                 ((Graphics2D)
g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
111                 ((Graphics2D)
g).setRenderingHint(RenderingHints.KEY_ALPHA_INTERPOLATION,
RenderingHints.VALUE_ALPHA_INTERPOLATION_DEFAULT);
112 
113             }
114 
115             Ellipse2D e = new Ellipse2D.Float(xPos-Point.DIAMETER/2, yPos-
Point.DIAMETER/2, 34, 35);
116             ((Graphics2D) g).fill(e);
117             //标记最后一个棋子的红矩形框
118 
119             if(i==chessCount-1){//如果是最后一个棋子
120                 g.setColor(Color.red);
121                 g.drawRect(xPos-Point.DIAMETER/2, yPos-Point.DIAMETER/2,
122                             34, 35);
123             }
124         }
125     }
126 
127     public void mousePressed(MouseEvent e){//鼠标在组件上按下时调用
128 
129         //游戏结束时，不再能下
130         if(gameOver) return;
131 
132         String colorName=isBlack?"黑棋":"白棋";
133 
```

```
        //将鼠标点击的坐标位置转换成网格索引
        xIndex=(e.getX()-MARGIN+GRID_SPAN/2)/GRID_SPAN;
        yIndex=(e.getY()-MARGIN+GRID_SPAN/2)/GRID_SPAN;

        //落在棋盘外不能下
        if(xIndex<0||xIndex>ROWS||yIndex<0||yIndex>COLS)
            return;

        //如果x，y位置已经有棋子存在，不能下
        if(findChess(xIndex,yIndex))return;

        //可以进行时的处理
        Point ch=new Point(xIndex,yIndex,isBlack?Color.black:Color.white);
        chessList[chessCount++]=ch;
         repaint();//通知系统重新绘制


        //如果胜出则给出提示信息，不能继续下棋

        if(isWin()){
            String msg=String.format("恭喜，%s赢了！", colorName);
            JOptionPane.showMessageDialog(this, msg);
            gameOver=true;
        }
        isBlack=!isBlack;
      }
    //覆盖mouseListener的方法
    public void mouseClicked(MouseEvent e){
        //鼠标按键在组件上单击时调用
    }

    public void mouseEntered(MouseEvent e){
        //鼠标进入到组件上时调用
    }
    public void mouseExited(MouseEvent e){
        //鼠标离开组件时调用
    }
    public void mouseReleased(MouseEvent e){
        //鼠标按钮在组件上释放时调用
    }
    //在棋子数组中查找是否有索引为x，y的棋子存在
    private boolean findChess(int x,int y){
        for(Point c:chessList){
            if(c!=null&&c.getX()==x&&c.getY()==y)
                return true;
        }
        return false;
    }


    private boolean isWin(){
        int continueCount=1;//连续棋子的个数

        //横向向西寻找
        for(int x=xIndex-1;x>=0;x--){
            Color c=isBlack?Color.black:Color.white;
            if(getChess(x,yIndex,c)!=null){
                continueCount++;
            }else
                break;
        }
       //横向向东寻找
        for(int x=xIndex+1;x<=COLS;x++){
           Color c=isBlack?Color.black:Color.white;
           if(getChess(x,yIndex,c)!=null){
              continueCount++;
           }else
              break;
        }
        if(continueCount>=5){
              return true;
        }else
        continueCount=1;
```

```
207
208         //继续另一种搜索纵向
209         //向上搜索
210         for(int y=yIndex-1;y>=0;y--){
211             Color c=isBlack?Color.black:Color.white;
212             if(getChess(xIndex,y,c)!=null){
213                 continueCount++;
214             }else
215                 break;
216         }
217         //纵向向下寻找
218         for(int y=yIndex+1;y<=ROWS;y++){
219             Color c=isBlack?Color.black:Color.white;
220             if(getChess(xIndex,y,c)!=null)
221                 continueCount++;
222             else
223                break;
224
225         }
226         if(continueCount>=5)
227             return true;
228         else
229             continueCount=1;
230
231
232         //继续另一种情况的搜索：斜向
233         //东北寻找
234         for(int x=xIndex+1,y=yIndex-1;y>=0&&x<=COLS;x++,y--){
235             Color c=isBlack?Color.black:Color.white;
236             if(getChess(x,y,c)!=null){
237                 continueCount++;
238             }
239             else break;
240         }
241         //西南寻找
242         for(int x=xIndex-1,y=yIndex+1;x>=0&&y<=ROWS;x--,y++){
243             Color c=isBlack?Color.black:Color.white;
244             if(getChess(x,y,c)!=null){
245                 continueCount++;
246             }
247             else break;
248         }
249         if(continueCount>=5)
250             return true;
251         else continueCount=1;
252
253
254         //继续另一种情况的搜索：斜向
255         //西北寻找
256         for(int x=xIndex-1,y=yIndex-1;x>=0&&y>=0;x--,y--){
257             Color c=isBlack?Color.black:Color.white;
258             if(getChess(x,y,c)!=null)
259                 continueCount++;
260             else break;
261         }
262         //东南寻找
263         for(int x=xIndex+1,y=yIndex+1;x<=COLS&&y<=ROWS;x++,y++){
264             Color c=isBlack?Color.black:Color.white;
265             if(getChess(x,y,c)!=null)
266                 continueCount++;
267             else break;
268         }
269         if(continueCount>=5)
270             return true;
271         else continueCount=1;
272
273         return false;
274       }
275
276
277     private Point getChess(int xIndex,int yIndex,Color color){
278         for(Point p:chessList){
279             if(p!=null&&p.getX()==xIndex&&p.getY()==yIndex
280                     &&p.getColor()==color)
```

```
                return p;
            }
            return null;
        }


        public void restartGame(){
            //清除棋子
            for(int i=0;i<chessList.length;i++){
                chessList[i]=null;
            }
            //恢复游戏相关的变量值
            isBlack=true;
            gameOver=false; //游戏是否结束
            chessCount =0; //当前棋盘棋子个数
            repaint();
        }

        //悔棋
        public void goback(){
            if(chessCount==0)
                return ;
            chessList[chessCount-1]=null;
            chessCount--;
            if(chessCount>0){
                xIndex=chessList[chessCount-1].getX();
                yIndex=chessList[chessCount-1].getY();
            }
            isBlack=!isBlack;
            repaint();
        }

        //矩形Dimension

        public Dimension getPreferredSize(){
            return new Dimension(MARGIN*2+GRID_SPAN*COLS,MARGIN*2
                                 +GRID_SPAN*ROWS);
        }


    }
```

2.棋子类

```
package cn.edu.ouc.fiveChess;

import java.awt.Color;
/**
 * 棋子类
 */
public class Point {
  private int x;//棋盘中的x索引
  private int y;//棋盘中的y索引
  private Color color;//颜色
  public static final int DIAMETER=30;//直径

  public Point(int x,int y,Color color){
      this.x=x;
      this.y=y;
      this.color=color;
  }

  public int getX(){//拿到棋盘中x的索引
      return x;
  }
  public int getY(){
      return y;
  }
  public Color getColor(){//获得棋子的颜色
      return color;
  }
}
```

3.五子棋主框架类

```
1 package cn.edu.ouc.fiveChess;
2 import java.awt.event.*;
3 import java.awt.*;
4
5 import javax.swing.*;
6 /*
7  五子棋主框架類，程序啟動類
8  */
9 public class StartChessJFrame extends JFrame {
10   private ChessBoard chessBoard;
11   private JPanel toolbar;
12   private JButton startButton,backButton,exitButton;
13
14   private JMenuBar menuBar;
15   private JMenu sysMenu;
16   private JMenuItem startMenuItem,exitMenuItem,backMenuItem;
17   //重新开始，退出，和悔棋菜单项
18   public StartChessJFrame(){
19       setTitle("单机版五子棋");//设置标题
20       chessBoard=new ChessBoard();
21
22
23       Container contentPane=getContentPane();
24       contentPane.add(chessBoard);
25       chessBoard.setOpaque(true);
26
27
28       //创建和添加菜单
29       menuBar =new JMenuBar();//初始化菜单栏
30       sysMenu=new JMenu("系统");//初始化菜单
31       //初始化菜单项
32       startMenuItem=new JMenuItem("重新开始");
33       exitMenuItem =new JMenuItem("退出");
34       backMenuItem =new JMenuItem("悔棋");
35       //将三个菜单项添加到菜单上
36       sysMenu.add(startMenuItem);
37       sysMenu.add(exitMenuItem);
38       sysMenu.add(backMenuItem);
39       //初始化按钮事件监听器内部类
40       MyItemListener lis=new MyItemListener();
41       //将三个菜单注册到事件监听器上
42       this.startMenuItem.addActionListener(lis);
43       backMenuItem.addActionListener(lis);
44       exitMenuItem.addActionListener(lis);
45       menuBar.add(sysMenu);//将系统菜单添加到菜单栏上
46       setJMenuBar(menuBar);//将menuBar设置为菜单栏
47
48       toolbar=new JPanel();//工具面板实例化
49       //三个按钮初始化
50       startButton=new JButton("重新开始");
51       exitButton=new JButton("退出");
52       backButton=new JButton("悔棋");
53       //将工具面板按钮用FlowLayout布局
54       toolbar.setLayout(new FlowLayout(FlowLayout.LEFT));
55       //将三个按钮添加到工具面板
56       toolbar.add(startButton);
57       toolbar.add(exitButton);
58       toolbar.add(backButton);
59       //将三个按钮注册监听事件
60       startButton.addActionListener(lis);
61       exitButton.addActionListener(lis);
62       backButton.addActionListener(lis);
63       //将工具面板布局到界面”南方“也就是下方
64       add(toolbar,BorderLayout.SOUTH);
65       add(chessBoard);//将面板对象添加到窗体上
66       //设置界面关闭事件
67       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
68       //setSize(800,800);
```

```
69          pack();//自适应大小
70
71      }
72
73      private class MyItemListener implements ActionListener{
74          public void actionPerformed(ActionEvent e){
75              Object obj=e.getSource();//获得事件源
76              if(obj==StartChessJFrame.this.startMenuItem||obj==startButton){
77                  //重新开始
78                  //JFiveFrame.this内部类引用外部类
79                  System.out.println("重新开始");
80                  chessBoard.restartGame();
81              }
82              else if (obj==exitMenuItem||obj==exitButton)
83                  System.exit(0);
84              else if (obj==backMenuItem||obj==backButton){
85                  System.out.println("悔棋...");
86                  chessBoard.goback();
87              }
88          }
89      }
90
91
92
93      public static void main(String[] args){
94          StartChessJFrame f=new StartChessJFrame();//创建主框架
95          f.setVisible(true);//显示主框架
96
97      }
98  }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5087531>

算法期末考试练习题！！！

2016年5月4日　　19:17

一、选择题

1.算法分析中，记号O表示（B），记号Ω标售（A）,记号Θ表示（D）

A 渐进下界 B 渐进上界 C 非紧上界 D 紧渐进界 E 非紧下界

2.以下关于渐进记号的性质是正确的有：（A）

A f(n) =Θ(g(n)),g(n) =Θ(h(n)) ⇒f(n) =Θ(h(n))

B f(n) =O(g(n)),g(n) =O(h(n)) ⇒h(n) =O(f(n))

C O(f(n))+O(g(n)) = O(min{f(n),g(n)})

D f(n) = O(g(n)) ⇔g(n) = O(f(n))

3. 记号O的定义正确的是（A）。

A O(g(n)) = { f(n) | 存在正常数c和n0使得对所有n≥n_0有：0≤ f(n) ≤ cg(n) }；

B O(g(n)) = { f(n) | 存在正常数c和n_0使得对所有n≥n_0有：0≤ cg(n) ≤ f(n) }；

C O(g(n)) = { f(n) | 对于任何正常数c>0，存在正数和n0 >0使得对所有n≥n_0有：0 ≤f(n) <cg(n) }；

D O(g(n)) = { f(n) | 对于任何正常数c>0，存在正数和n_0 >0使得对所有n≥n_0有：0 ≤cg(n) < f(n) }；

4. 记号Ω的定义正确的是（B）。

A O(g(n)) = { f(n) | 存在正常数c和n_0使得对所有n≥n_0有：0≤ f(n) ≤ cg(n) }；

B O(g(n)) = { f(n) | 存在正常数c和n_0使得对所有n≥n_0有：0≤ cg(n) ≤ f(n) }；

C (g(n)) = { f(n) | 对于任何正常数c>0，存在正数和n_0 >0使得对所有n≥n_0有：0 ≤f(n) <cg(n) }；

D (g(n)) = { f(n) | 对于任何正常数c>0，存在正数和n_0 >0使得对所有n≥n_0有：0 ≤cg(n) < f(n) }；

5. T（n）表示当输入规模为n时的算法效率，以下算法效率最优的是(C)

A T（n）= T（n － 1）+1，T（1）=1

B T（n）= $2n^2$

C T（n）= T（n/2）+1，T（1）=1

D T（n）= 3nlog2n

6. 动态规划算法的基本要素为（C）

A 最优子结构性质与贪心选择性质

B 重叠子问题性质与贪心选择性质

C 最优子结构性质与重叠子问题性质

D 预排序与递归调用

7.下列不是动态规划算法基本步骤的是（ A ）。

A 找出最优解的性质　　B 构造最优解

C 算出最优解　　D 定义最优解

8.能采用贪心算法求最优解的问题，一般具有的重要性质为：（A）

A 最优子结构性质与贪心选择性质

B 重叠子问题性质与贪心选择性质

C 最优子结构性质与重叠子问题性质

D 预排序与递归调用

9.下面是贪心算法的基本要素的是（ C ）。

A 重叠子问题 B 构造最优解 C 贪心选择性质 D 定义最优解

10（ D ）是贪心算法与动态规划算法的共同点。

A 重叠子问题　　B 构造最优解

C 贪心选择性质　　D 最优子结构性质

11.使用分治法求解不需要满足的条件是（A ）。

A 子问题必须是一样的　　B 子问题不能够重复

C 子问题的解可以合并　　D 原问题和子问题使用相同的方法解

12.实现循环赛日程表利用的算法是（ A ）。

A 分治策略　　B 动态规划法

C 贪心法　　D 回溯法

13.使用分治法求解不需要满足的条件是（A ）。

A 子问题必须是一样的　　B 子问题不能够重复

C 子问题的解可以合并　　D 原问题和子问题使用相同的方法解

14.下列算法中不能解决0/1背包问题的是（A ）

A 贪心法 B 动态规划 C 回溯法 D 分支限界法

15.以下(C)不能不能在线性时间完成排序

A 计数排序　　B 基数排序　　C 堆排序　　D 桶排序

16.下列哪一种算法是随机化算法（ D ）

A 贪心算法　　B 回溯法　　C 动态规划算法　　D 舍伍德算法

17. 下列算法中通常以自底向上的方式求解最优解的（ B ）。 A 分治法　　B 动态规划法　　C 贪心法　　D 回溯法

18. n个人拎着水桶在一个水龙头前面排队打水，水桶有大有小，水桶必须打满水，水流恒定。如下 (A) 说法不正确？A

A 让水桶大的人先打水，可以使得每个人排队时间之和最小

B 让水桶小的人先打水，可以使得每个人排队时间之和最小

C 让水桶小的人先打水，在某个确定的时间t内，可以让尽可能多的人打上水

D 若要在尽可能短的时间内，n个人都打完水，按照什么顺序其实都一样

19.哈弗曼编码的贪心算法所需的计算时间为 (■ B)。

A O(n2^n) B O(nlogn) C O(2n) D O(n)

20.下面关于NP问题说法正确的是(■B ┌┐)

A NP问题都是不可能解决的问题

B P类问题包含在NP类问题中

C NP完全问题是P类问题的子集

D NP类问题包含在P类问题中

21.背包问题贪心算法所需的计算时间为(B)

A O(n2^n) B O(nlogn) C O(2n) D O(n)

22.背包问题的回溯算法所需的计算时间为（ A ）

A O(n2^n) B O(nlogn) C O(2n) D O(n)

23.采用最大效益优先搜索方式的算法是（ A ）

A 分支界限发 B 动态规划法 C 贪心法 D 回溯法

24. 在棋盘覆盖问题中，对于2k×2k的特殊棋盘（有一个特殊方块），所需的L型骨牌的个数是（ A ）

A (4k － 1）/3 B 2k /3 C 4k D 2k

25. 下列随机算法中运行时有时候成功有时候失败的是（C ）

A 数值概率算法 B 舍伍德算法 C 拉斯维加斯算法 D 蒙特卡罗算法

26. 实现大整数的乘法是利用的算法（ C ）。

A 贪心法 B 动态规划法 C 分治策略 D 回溯法

二、填空题

1.算法的复杂性有 时间 复杂性和 空间 复杂性之分。

2.矩阵连乘问题的算法可由 动态规划 设计实现。

3.从分治法的一般设计模式可以看出，用它设计出的程序一般是 递归算法 。

4.算法是由若干条指令组成的有穷序列，且要满足输入、输出、确定性和有限性四条性质。

5.快速排序算法的性能取决于 划分的对称性 。

6.使用二分搜索算法在n个有序元素表中搜索一个特定元素，在最佳情况下，搜索的时间复杂性为O（ 1 ），在最坏情况下，搜索的时间复杂性为O（ logn ）。

三、解答题

1. 给定已按升序排好序的n个元素a[0:n-1]，现要在这n个元素中找出一特定元素x，返回其在数组中的位置，如果未找到返回-1。 写出二分搜索的算法，并分析其时间复杂度

2. 分析最有搜索二叉树和0/1背包的时间复杂度

3. 试用动态规划算法实现最大子矩阵和问题：求n×m矩阵A的一个子矩阵，使其各元素之各为最大

4已知输入向量a=（1,3,4，-2），给出用FFT的蝶形操作求输出向量y的结果，并分析出FFT的计算时间复杂度。

1.采用递归算法求递归方程F(n)=F(n-1(+F(n-2)，算法描述如下（就是递归fibonacci！）

问：算法共需要进行多少次递归调用（算法中没引用F(i)一次称为一次递归调用）。有没有更好的算法来计算F(n)?若有请描述算法并分析复杂度。

2.如下算法是否产生平均分布的随即置换？为什么？

```
Permute-With-All(A)
  n←length(A)
  for i←1 to n
     swap(A[i],A[RANDOM(1,n)])
```

注：RANDOM(1,n)随机、等可能地返回整数k，1≤k≤n

3. 能否在给定的s[n]中判断是否存在两个数的和为x，并且时间复杂度是nlgn，如果可以写出程序的伪代码，否则给出理由。

6. 活动选择问题

（1）递归的时间复杂度分析

（2）动态规划的时间复杂度分析

（3）写出一个更优的算法，并且对其进行时间复杂度分析

7. 多项式乘法问题

描述一个高效的算法来解决复数之间的多项式乘法，多项式的系数为复数，未知数也为复数。

并且对此进行时间复杂度分析。

郑55　　算法2013考试题

填空：

1. 给一系列的函数，根据渐进性，从大到小排列n^3/2,nlgn,lgn,e^n,n!,n^2
2. 动归的两个性质
3. 贪心的两个性质
4. 1）NP问题：　2）P问题
5. 面试问题 雇一个人的概率：　，雇n个人的概率：
6. 复杂度分为：　和　；动归是牺牲　复杂度换取　复杂度
7. 使用二分搜索算法在n个有序元素表中搜索一个特定元素，在最佳情况下，搜索的时间复杂性为O（ 1 ），在最坏情况下，搜索的时间复杂性为O（ logn　）。

选择：

1摊还排序是（）情况下的（）代价

A最优B最坏C平均D最好

2动态规划的设计思想是a

(a)自底向上　(b)自上而下　(c)从左向右　(d)从右向左

3贪心算法的设计思想是b

(a)自底向上　(b)自上而下　(c)从左向右　(d)从右向左

4以下哪一个更可能描述实际一个有效的算法d

(a)　(b)　(c)　(d)

5蝶形操作的设计思想是a

(a)分治法　(b)贪心算法　(c)动态规划　(d)回溯

6以下哪一个不是NPC问题

(a)单源最短路径　(b)巡回售货员问题　(c)布尔可满足性问题　(d)大数分解

7以下哪个不是几何研究中的基本操作

(a)+　(b)—　(c)×　(d)/

8哪个问题不能用贪心算法解决

(a)活动安排　(b)哈夫曼编码　(c)分数背包　(d)0-1背包

9哪个问题不能用动态规划解决c

(a)分数背包　(b)0-1背包　(c)最长简单路径　(d)最短简单路径

10插入排序的最好时间复杂度是a

(a)n　(b)n^2　(c)nlgn　(d)lgn

11归并排序的时间复杂度是c

(a)n　(b)n^2　(c)nlgn　(d)lgn

12算法分析中，记号O表示（B），记号Ω标售（A）,记号Θ表示（D）

A 渐进下界 B 渐进上界 C 非紧上界 D 紧渐进界 E 非紧下界

13.n个人拎着水桶在一个水龙头前面排队打水，水桶有大有小，水桶必须打满水，水流恒定。如下（A）说法不正确？A

A 让水桶大的人先打水，可以使得每个人排队时间之和最小

B 让水桶小的人先打水，可以使得每个人排队时间之和最小

C 让水桶小的人先打水，在某个确定的时间t内，可以让尽可能多的人打上水

D 若要在尽可能短的时间内，n个人都打完水，按照什么顺序其实都一样

14.哈弗曼编码的贪心算法所需的计算时间为 (■ B　)。

A O(n2n)┌┐ B O(nlogn)　C O(2n)　D O(n)

15.下面关于NP问题说法正确的是(■B ┌┐)

A NP问题都是不可能解决的问题

B P类问题包含在NP类问题中

C NP完全问题是P类问题的子集

D NP类问题包含在P类问题中

大题

1. 四路归并排序：分解时分成四个，合并等其他步骤与二路归并相似，请写出核心算法，并分析时间复杂度。

2. 矩阵链乘积的递归算法

T（n）= 1　n=1

n>=2

请详细分析该算法的时间复杂度

1. 给出最优二叉搜索树的程序代码和p，q概率

根据概率1）求e,w,root，2）画出二叉树的结构3）请说出root[i,j]有什么意义

（见3621大班试卷影印版的第五题）

1. FFt蝶形操作 见3621大班试卷的影印版第九题
2. 字符串自动机构造，见书

2006-2007学年第二学期《计算机算法设计与分析》试题

院系：软件学院　　专业：软件工程　　年级：2004级

一．简答题（共10分）

略

二．计算题（35分）

1．(6分) 对下列各组函数f(n)和g(n)，确定f(n)=O(g(n))或f(n)=Ω(g(n))或f(n)=θ(g(n))。

(1)　f(n)=3^n，g(n)=2^n

(2)　f(n)=log n + 5，g(n)=log n^2

(3)　f(n)=log n，g(n)=

答：

(1)　　f(n) = Ω(g(n))　　(2分)

(2)　　f(n) = θ(g(n))　　(2分)

(3)　　f(n) = O(g(n))　　(2分)

2．（8分）采用动态规划策略，计算a=｛5,-3,7,-4,-5,9,-2,10,-3,2｝的最大子段和，并给出这个最大子段和的起始下标和终止下标。［设数组a中的元素下标从1开始。］要求给出过程。

答：

b[1]=5；

b[2]=max{b[1]+a[2]，a[2]}=max{2,-3}=2

b[3]=max{b[2]+a[3]，a[3]}=max{9,7}=9

b[4]=max{b[3]+a[4]，a[4]}=max{5,-4}=5

b[5]=max{b[4]+a[5]，a[5]}=max{0,-5}=0

b[6]=max{b[5]+a[6]，a[6]}=max{9,9}=9

b[7]=max{b[6]+a[7]，a[7]}=max{7,-2}=7

b[8]=max{b[7]+a[8]，a[8]}=max{17,10}=17

b[9]=max{b[8]+a[9]，a[9]}=max{14,-3}=14

b[10]=max{b[9]+a[10]，a[10]}=max{16,2}=16

(上述每两行 1 分，共 5 分)

最大子段和为17（1分）

（若数组下标从 1 开始）起始下标： 6 （ 1 分），终止下标： 8 （ 1 分）

（若数组下标从 0 开始）起始下标： 5 （0.5分），终止下标：7（0.5分）

3．（11分）设有 3 件工作分配给 3 个人，将工作i分配给第j个人所花的费用为C_{ij}，现将为每一个人都分配 1 件不同的工作，并使总费用达到最小。设：

要求：

（1）画出该问题的解空间树；（ 6 分）

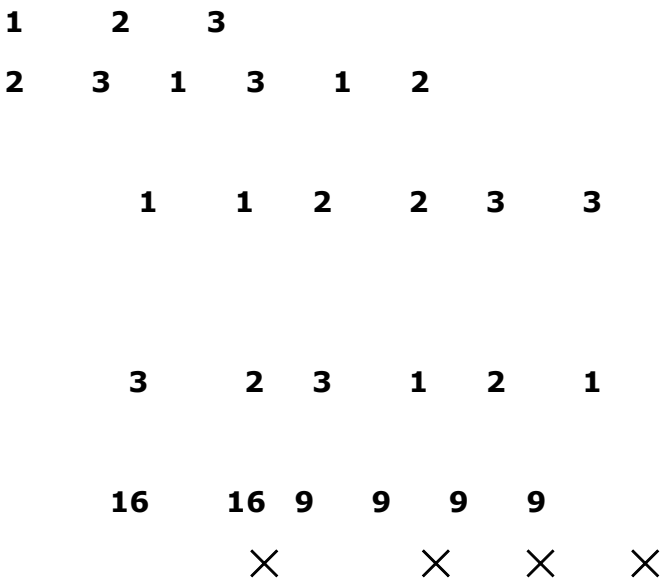
（2）写出该问题的剪枝策略(限界条件)；（1分）

（3）按回溯法搜索解空间树，并利用剪枝策略对应该剪掉的子树打´；（ 2 分）

（4）最终给出该问题的最优值和最优解。（ 2 分）

答：

（1）



（每个分枝1分，或者是每层2分，共6分）

（2）当耗费大于等于当前最优值时，剪枝。（1分）

（3）见上图。（每2个╳ 1分，共2分）

（4）最优值： 9 （1分）

最优解：2，1，3

4．（8 分）给定两个序列X=<A,B,C,B,A>，Y=<B,D,C,A,B>，请采用动态规划策略求出其最长公共子序列。要求给出过程。

答：

j	**0**	**1**	**2**	**3**	**4**	**5**
i		y_i **B**	**D**	**C**	**A**	**B**
0	x_i **0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3**
5	A	**0**	**1**	**2**	**2**	**2**	**3**

（上述表内每一行一分，共 6 分）

最长公共子序列为<B，C，B>（ 2 分）

5. （2分）考虑n=3时的0-1背包问题的实例：W={15,10,10}，P={50,30,30}，c=20。当x[1]=1，x[2]=0时，请计算Bound(2)。

答：

Bound(3)=50+5/10 * 20 = 65　(2分)

三、**算法填空题（共10分，每空2分）**

给定n种物品和一个背包，物品i的重量是w[i], 其价值是p[i], 背包的容量为c。设物品已按单位重量价值递减的次序排序。每种物品不可以装入背包多次，但可以装入部分的物品i。求解背包问题的贪心算法如下：

```
float Knapsack (float x[ ], float w[ ], float p[ ],float c, int n)
{ float maxsum= _____ ;  // maxsum表示装进包的物品价值总和
for ( int i=1;i<=n; i++)  x[i]=0 // x[i]=0表示第i个物品未装进包
for ( _____ )
    if( _____ )
{ x[i] =1;
      maxsum+=_____ ;
      c- = w[i];
  }
```

else break;

if (c>0) {x[i]=c/w[i]; _____ ;}

return maxsum;

}

答：(共 5 个空，每空2分，总计10分)

0

int i=1；i<=n； i ++

w[i]<=C

p[i]

maxsum+ = p[i] * c / w[i]

四、**算法设计及分析题（共45分）**

1．（20分）请用分治策略设计递归的二分检索算法，并分析其时间复杂性（要求给出每个阶段所花的时间，在此基础上列出递归方程，并求出其解的渐进阶）。

答：（此题解法不唯一）

算法：

```
int bsearch(A[],K,L,H)                                    （1分）
{ if (H<L)  return(-1);                                   (2分)
 else
{ mid=(L+H) /2;                                          (2分)
        if (A[mid] == K)                                  (1分)
           return(mid)；                                  (1分)
    else if （A[mid]>K）                                  (2分)
return(bsearch(A,K,L, mid-1)) ;                           (2分)
       else  return(bsearch(K, mid+1,H));                          (2分)
    }
}
```

算法时间复杂性分析：

∵ **Divide**阶段所花时间为**O(1)**　　**(1分)**

Conquer阶段所花时间为**T(n/2)**　　**(1分)**

merge阶段没有花时间（或者说，**merge**阶段所花时间为**O(1)**）　**(1分)**

∴ **算法时间复杂性递归方程如下：**

T(n)=O(1)　　当n≤0

T(n)= T(n/2)+O（1）　当n>1　　(2分)

用套用公式法（master method）求解递归方程：

∵ a=1,b=2,

f(n)=O(1), ∴ **与f(n)同阶**

∴ **T(n)= O(*log n)*　　(2分)**

2. （15分）请用回溯法设计算法，用四种颜色给地图着色（若在调用了其它算法，也需将该算法写出）。请在每个循环语句和条件判断语句后加注释。

答：（此题解法不唯一）

算法：

```
boolColor::Ok(int k)
{ for (int j=1; j<k; j++) //检查前k-1个点与当前点（第k点）颜色是否冲突 (2分)
if ((a[k][j]==1)&&(x[j]==x[k]))
//判断第j点与当前点（第k点）颜色是否冲突 (2分)
return false;　　(1分)
else  return true;　　(1分)
}

void Color::Backtrack(int t)
{ if (t > n)  // 判断是否到叶节点　　(1分)
{ sum++;
for (int i=1;i<=n; i++) //输出每个点的色号
cout << x[i]<< ' ';
cout << endl;　　（2分）
}
else
for (int i=1; i<=4; i++) // 依次检查当前点（第t点）是否可着第i(1≤i≤4)种颜色 (2分)
{ x[t]=i;　　(1分)
if (Ok(t))  Backtrack(t+1); //若当前点（第t点）可着第i种颜色，则递归调用 (3分)
}
}
```

3. （10分）请设计一个算法，实现优先队列的出队操作。要求：

（1）指出用什么数据结构实现优先队列；

（2）用C语言描述算法。

答：（此题解法不唯一）

（1）用堆实现优先队列。　　（2分）

（2）算法

```
SIFT(k,i,m)
{temp = k[i];
 j=2*i;                    (1分)
 while (j<=m)              (1分)
      { if（（j<m）&&（R[j].key<R[j+1].key））j++;  (1分)
if （temp.key <R[j].key）              (1分)
   { R[i] = R[j];
     i=j; j=2*i;                    （1分）
   }
else break;                          (0.5分)
   R[i]=temp;                              (0.5分)
  }

 Delete（n）
    { temp = R[1]; R[1]=R[i]; R[i]=temp;       (1分)
SIFT(k,1,n-1);                      （1分）
 }
```

算法导论试题

题型：选择+解答

一，选择

（1）　　动态规划的设计思想是a

(a)自底向上　(b)自上而下　(c)从左向右　(d)从右向左

（2）　　贪心算法的设计思想是b

(a)自底向上　(b)自上而下　(c)从左向右　(d)从右向左

（3）　　以下哪一个更可能描述实际一个有效的算法d

(a)　(b)　(c)　(d)

（4）　　蝶形操作的设计思想是a

(a)分治法　(b)贪心算法　(c)动态规划　(d)回溯

（5）　　以下哪一个不是NPC问题

(a)单源最短路径　(b)巡回售货员问题　(c)布尔可满足性问题　(d)大数分解

（6）　　以下哪个不是几何研究中的基本操作

(a)+　(b)—　(c)×　(d)/

（7）　　哪个问题不能用贪心算法解决

(a)活动安排　(b)哈夫曼编码　(c)分数背包　(d)0-1背包

（8）　　哪个问题不能用动态规划解决c

(a)分数背包　(b)0-1背包　(c)最长简单路径　(d)最短简单路径

（9）　　插入排序的最好时间复杂度是a

(a)n　(b)n^2　(c)nlgn　(d)lgn

（10）　归并排序的时间复杂度是c

(a)n　(b)n^2　(c)nlgn　(d)lgn

二，解答题（共6题，此处少1题）

（1）　　编写伪代码并分析时间复杂度，需要设计的程序是：运用递归算法设计插入排序，并且用到折半查找。提示：要排序a[1……n]，先排a[1……n-1]，插入a[n]用折半查找。

（2）　　能否在给定的s[n]中判断是否存在两个数的和为x，并且时间复杂度是nlgn，如果可以写出程序的伪代码，否则给出理由。

（3）　　两个n维矩阵乘法A×B=C，将A，B分解为4个n/2维的矩阵，分析以下算法的时间复杂度。

A=,B=,C=

定义P1=（A11+A22）（B11+B22），P2=（A11+A22）B11，P3=A11(B11-B22)，P4=A22（-B11+B22），P5=（A12+A11）B22，P6=（-A11+A21）（B11+B12），P7=（A12-A22）（B11+B22）

则C11=P1+P4-P5+P7，C12=P3+P5，C21=P2+P4，C22=P1+P3-P2+P6

（4）　　递归斐波纳切数列的时间复杂度

（5）　　矩阵链乘法计算最优代价，需要画出和书本上p200类似的三角形图（动态规划）

一。选择题

1、二分搜索算法是利用（　A　　）实现的算法。

A、分治策略　B、动态规划法　C、贪心法　D、回溯法

2、下列不是动态规划算法基本步骤的是（　A　）。

A、找出最优解的性质　B、构造最优解　C、算出最优解　D、定义最优解

3、最大效益优先是（　A　　）的一搜索方式。

A、分支界限法　　B、动态规划法　C、贪心法　D、回溯法

4、在下列算法中有时找不到问题解的是（ B　　）。

A、蒙特卡罗算法　B、拉斯维加斯算法　C、舍伍德算法　D、数值概率算法

5. 回溯法解旅行售货员问题时的解空间树是（ A　　）。

A、子集树　　B、排列树　C、深度优先生成树　D、广度优先生成树

6．下列算法中通常以自底向上的方式求解最优解的是（　B　　）。

A、备忘录法　　B、动态规划法　C、贪心法　　D、回溯法

7、衡量一个算法好坏的标准是（C ）。

A 运行速度快 B 占用空间少 C 时间复杂度低 D 代码短

8、以下不可以使用分治法求解的是（D ）。

A 棋盘覆盖问题 B 选择问题 C 归并排序 D 0/1背包问题

9. 实现循环赛日程表利用的算法是（　A　　）。

A、分治策略　　B、动态规划法　　C、贪心法　　D、回溯法

10、下列随机算法中运行时有时候成功有时候失败的是（C ）

A 数值概率算法 B 舍伍德算法 C 拉斯维加斯算法 D 蒙特卡罗算法

11．下面不是分支界限法搜索方式的是（　D　）。

A、广度优先　　B、最小耗费优先　C、最大效益优先　　D、深度优先

12．下列算法中通常以深度优先方式系统搜索问题解的是（　　D　）。

A、备忘录法　　B、动态规划法　C、贪心法　　D、回溯法

13.备忘录方法是那种算法的变形。（　B ）

A、分治法　　B、动态规划法　C、贪心法　　D、回溯法

14．哈弗曼编码的贪心算法所需的计算时间为（　B　）。

A、O（$n2^n$）　　B、O（nlogn）　C、O（2^n）　　D、O（n）

15．分支限界法解最大团问题时，活结点表的组织形式是（　B　）。

A、最小堆　　B、最大堆　　C、栈　　D、数组

16．最长公共子序列算法利用的算法是（　B　　）。

A、分支界限法　B、动态规划法　　C、贪心法　　D、回溯法

17．实现棋盘覆盖算法利用的算法是（　A　　）。

A、分治法　　B、动态规划法　C、贪心法　　D、回溯法

18.下面是贪心算法的基本要素的是（　　C　　）。

A、重叠子问题　B、构造最优解　C、贪心选择性质　　D、定义最优解

19.回溯法的效率不依赖于下列哪些因素（　D　）

A.满足显约束的值的个数　　B. 计算约束函数的时间

C. 计算限界函数的时间　　D. 确定解空间的时间

20.下面哪种函数是回溯法中为避免无效搜索采取的策略（ B ）

A. 递归函数 B.剪枝函数 C。随机数函数 D.搜索函数

21、下面关于NP问题说法正确的是（B ）

A NP问题都是不可能解决的问题

B P类问题包含在NP类问题中

C NP完全问题是P类问题的子集

D NP类问题包含在P类问题中

22、蒙特卡罗算法是（ B ）的一种。

A、分支界限算法 B、概率算法 C、贪心算法 D、回溯算法

23.下列哪一种算法不是随机化算法（ C ）

A. 蒙特卡罗算法B. 拉斯维加斯算法C.动态规划算法D.舍伍德算法

24. （ D ）是贪心算法与动态规划算法的共同点。

A、重叠子问题 B、构造最优解 C、贪心选择性质 D、最优子结构性质

25. 矩阵连乘问题的算法可由（ B）设计实现。

A、分支界限算法 B、动态规划算法 C、贪心算法 D、回溯算法

26. 分支限界法解旅行售货员问题时，活结点表的组织形式是（ A ）。

A、最小堆 B、最大堆 C、栈 D、数组

27、Strassen矩阵乘法是利用（ A ）实现的算法。

A、分治策略 B、动态规划法 C、贪心法 D、回溯法

29、使用分治法求解不需要满足的条件是（A ）。

A 子问题必须是一样的

B 子问题不能够重复

C 子问题的解可以合并

D 原问题和子问题使用相同的方法解

30、下面问题（B ）不能使用贪心法解决。

A 单源最短路径问题 B N皇后问题

C 最小花费生成树问题 D 背包问题

31、下列算法中不能解决0/1背包问题的是（A ）

A 贪心法 B 动态规划 C 回溯法 D 分支限界法

32、回溯法搜索状态空间树是按照（C ）的顺序。

A 中序遍历 B 广度优先遍历 C 深度优先遍历 D 层次优先遍历

33、下列随机算法中运行时有时候成功有时候失败的是（C ）

A 数值概率算法 B 舍伍德算法 C 拉斯维加斯算法 D 蒙特卡罗算法

34. 实现合并排序利用的算法是（ A ）。

A、分治策略　　B、动态规划法　　C、贪心法　　D、回溯法

35. 下列是动态规划算法基本要素的是（ D ）。

A、定义最优解　　B、构造最优解　　C、算出最优解　　D、子问题重叠性质

36. 下列算法中通常以自底向下的方式求解最优解的是（ B ）。

A、分治法　　B、动态规划法　　C、贪心法　　D、回溯法

37. 采用广度优先策略搜索的算法是（ A ）。

A、分支界限法　　B、动态规划法　　C、贪心法　　D、回溯法

38、合并排序算法是利用（ A ）实现的算法。

A、分治策略　B、动态规划法　C、贪心法　D、回溯法

39、在下列算法中得到的解未必正确的是（ B ）。

A、蒙特卡罗算法　B、拉斯维加斯算法　C、舍伍德算法　D、数值概率算法

40、背包问题的贪心算法所需的计算时间为（ B ）

A、O（$n2^n$）　B、O（nlogn）　C、O（2^n）　D、O（n）

41. 实现大整数的乘法是利用的算法（ C ）。

A、贪心法　　B、动态规划法　　C、分治策略　　D、回溯法

42. 0-1背包问题的回溯算法所需的计算时间为（ A ）

A、O（n2n）　　B、O（nlogn）　　C、O（2n）　　D、O（n）

43. 采用最大效益优先搜索方式的算法是（ A ）。

A、分支界限法　　B、动态规划法　　C、贪心法　　D、回溯法

44. 贪心算法与动态规划算法的主要区别是（ B ）。

A、最优子结构　　B、贪心选择性质　　C、构造最优解　　D、定义最优解

45. 实现最大子段和利用的算法是（ B ）。

A、分治策略　　B、动态规划法　　C、贪心法　　D、回溯法

46.优先队列式分支限界法选取扩展结点的原则是（ C ）。

A、先进先出　　B、后进先出　　C、结点的优先级　D、随机

47.背包问题的贪心算法所需的计算时间为（ B ）。

A、O（$n2^n$）　　B、O（nlogn）　C、O（2^n）　　D、O（n）

48、广度优先是（ A ）的一搜索方式。

A、分支界限法　　B、动态规划法　C、贪心法　D、回溯法

49、舍伍德算法是（ B ）的一种。

A、分支界限算法　　B、概率算法　C、贪心算法　D、回溯算法

50、在下列算法中有时找不到问题解的是（ B ）。

A、蒙特卡罗算法　B、拉斯维加斯算法　C、舍伍德算法　D、数值概率算法

51下列哪一种算法是随机化算法（ D ）

A. 贪心算法B. 回溯法C.动态规划算法D.舍伍德算法

52. 一个问题可用动态规划算法或贪心算法求解的关键特征是问题的（ B ）。

A、重叠子问题 B、最优子结构性质 C、贪心选择性质 D、定义最优解

53. 采用贪心算法的最优装载问题的主要计算量在于将集装箱依其重量从小到大排序，故算法的时间复杂度为 (B)。

A、O（$n2^n$） B、O（nlogn） C、O（2^n） D、O（n）

54. 以深度优先方式系统搜索问题解的算法称为 (D) 。

A、分支界限算法 B、概率算法 C、贪心算法 D、回溯算法

55. 实现最长公共子序列利用的算法是（ B ）。

A、分治策略 B、动态规划法 C、贪心法 D、回溯法

二、 填空题

1.算法的复杂性有 时间 复杂性和 空间 复杂性之分。

2、程序是 算法 用某种程序设计语言的具体实现。

3、算法的“确定性”指的是组成算法的每条 指令 是清晰的，无歧义的。

4.矩阵连乘问题的算法可由 动态规划 设计实现。

5、拉斯维加斯算法找到的解一定是 正确解。

6、算法是指解决问题的 一种方法 或 一个过程 。

7、从分治法的一般设计模式可以看出，用它设计出的程序一般是 递归算法 。

8、问题的 最优子结构性质 是该问题可用动态规划算法或贪心算法求解的关键特征。

9、以深度优先方式系统搜索问题解的算法称为 回溯法 。

10、数值概率算法常用于 数值问题 的求解。

11、计算一个算法时间复杂度通常可以计算 循环次数 、 基本操作的频率 或计算步。

12、利用概率的性质计算近似值的随机算法是 数值概率算法，运行时以一定的概率得到正确解的随机算法是 蒙特卡罗算法 。

14、解决0/1背包问题可以使用动态规划、回溯法和分支限界法，其中不需要排序的是 动态规划 ，需要排序的是 回溯法 ，分支限界法 。

15、使用回溯法进行状态空间树裁剪分支时一般有两个标准：约束条件和目标函数的界，N皇后问题和0/1背包问题正好是两种不同的类型，其中同时使用约束条件和目标函数的界进行裁剪的是 0/1背包问题 ，只使用约束条件进行裁剪的是 N皇后问题 。

16、 贪心选择性质 是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

17、矩阵连乘问题的算法可由 动态规划 设计实现。

18、拉斯维加斯算法找到的解一定是 正确解。

19.贪心算法的基本要素是 贪心选择 质和 最优子结构 性质 。

21. 动态规划算法的基本思想是将待求解问题分解成若干 子问题 ，先求解 子问

题，然后从这些 子问题 的解得到原问题的解。

22.算法是由若干条指令组成的有穷序列，且要满足输入、 输出 、确定性和 有限性 四条性质。

23、大整数乘积算法是用 分治法 来设计的。

24、以广度优先或以最小耗费方式搜索问题解的算法称为 分支限界法 。

25、舍伍德算法总能求得问题的 一个解 。

26、 贪心选择性质 是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

27.快速排序算法是基于 分治策略 的一种排序算法。

28.动态规划算法的两个基本要素是. 最优子结构 性质和 重叠子问题 性质 。

30.回溯法是一种既带有 系统性 又带有 跳跃性 的搜索算法。

31.分支限界法主要有 队列式（FIFO） 分支限界法和 优先队列式 分支限界法。

32. 分支限界法是一种既带有 系统性 又带有 跳跃性 的搜索算法。

33. 回溯法搜索解空间树时，常用的两种剪枝函数为 约束函数 和 限界函数 。

34.任何可用计算机求解的问题所需的时间都与其 规模 有关。

35.快速排序算法的性能取决于 划分的对称性 。

三、算法填空

1.背包问题的贪心算法

```
void Knapsack(int n,float M,float v[],float w[],float x[])
{
    Sort(n,v,w);
    int i;
    for (i=1;i<=n;i++) x[i]=0;
    float c=M;
    for (i=1;i<=n;i++) {
      if (w[i]>c) break;
      x[i]=1;
      c - =w[i];
      }
    if (i<=n) x[i]=c/w[i];
}
```

2.最大子段和: 动态规划算法

```
int MaxSum(int n, int a[])
```

```
{
   int sum=0, b=0；    //sum存储当前最大的b[j], b存储b[j]
   for(int j=1； j<=n； j++) {
      if (b>0)  b+= a[j] ；
      else  b=a[i];    ；     //一旦某个区段和为负，则从下一个位置累和
 if(b>sum) sum=b;

    }
    return sum；
 }
```

3.贪心算法求装载问题

```
template<class Type>
void Loading(int x[],  Type w[], Type c, int n)
{
      int *t = new int [n+1];
      _____ ;
      for (int i = 1; i <= n; i++) x[i] = 0;
      for (int i = 1; i <= n && w[t[i]] <= c; i++)
 {x[t[i]] = 1;
_____;}
}
```

4.贪心算法求活动安排问题

```
template<class Type>
void GreedySelector(int n, Type s[], Type f[], bool A[])
{
     A[1]=true;
     int j=1;
     for (int i=2;i<=n;i++) {
       if (s[i]>=f[j])
{  A[i]=true;
j=i;
 }
       else A[i]=false;
```

```
        }
}
```

5.快速排序

```
template<class Type>
void QuickSort (Type a[], int p, int r)
{
     if (p<r) {
       int q=Partition(a,p,r);
       QuickSort (a,p,q-1); //对左半段排序
       QuickSort (a,q+1,r); //对右半段排序
       }
}
```

6.排列问题

```
Template <class Type>
void perm(Type list[],  int k, int m )
{ //产生[list[k:m]的所有排列
   if(k==m)
    {  //只剩下一个元素
       for (int i=0;i<=m;i++)  cout<<list[i];
       cout<<endl;
   }
   else  //还有多个元素待排列，递归产生排列
     for (int i=k; i<=m; i++)
      {
        swap(list[k]，list[i]);
        perm(list,k+1;m);
        swap(list[k],list[i]);
    }
 }
```

四、问答题

1.分治法的基本思想时将一个规模为n的问题分解为k个规模较小的子问题，这些子问题互相独立

且与原问题相同。递归地解这些子问题，然后将各个子问题的解合并得到原问题的解。

2设计动态规划算法的主要步骤为：

（1）找出最优解的性质，并刻划其结构特征。（2）递归地定义最优值。（3）以自底向上的方式计算出最优值。（4）根据计算最优值时得到的信息，构造最优解。

3. 分治法与动态规划法的相同点是：将待求解的问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。

两者的不同点是：适合于用动态规划法求解的问题，经分解得到的子问题往往不是互相独立的。而用分治法求解的问题，经分解得到的子问题往往是互相独立的。

4. 分支限界法与回溯法的相同点是：都是一种在问题的解空间树T中搜索问题解的算法。

不同点：（1）求解目标不同；

（2）搜索方式不同；

（3）对扩展结点的扩展方式不同；

（4）存储空间的要求不同。

5用回溯法搜索子集树的算法为：

```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=0;i<=1;i++) {
        x[t]=i;
        if (constraint(t)&&bound(t)) backtrack(t+1);
      }
}
```

6. 分治法所能解决的问题一般具有的几个特征是：

（1）该问题的规模缩小到一定的程度就可以容易地解决；

（2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质;

（3）利用该问题分解出的子问题的解可以合并为该问题的解；

（4）原问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

7. 用分支限界法设计算法的步骤是：

(1)针对所给问题，定义问题的解空间（对解进行编码）；分

(2)确定易于搜索的解空间结构（按树或图组织解）；

(3)以广度优先或以最小耗费（最大收益）优先的方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

8. 常见的两种分支限界法的算法框架

（1）队列式(FIFO)分支限界法：按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。
（2）优先队列式分支限界法：按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

9. 回溯法中常见的两类典型的解空间树是子集树和排列树。

当所给的问题是从n个元素的集合S中找出满足某种性质的子集时，相应的解空间树称为子集树。这类子集树通常有2n个叶结点，遍历子集树需O(2n)计算时间 。

当所给的问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树。这类排列树通常有n!个叶结点。遍历排列树需要O(n!)计算时间。

10. 分支限界法的搜索策略是：

在扩展结点处，先生成其所有的儿子结点（分支），然后再从当前的活结点表中选择下一个扩展结点。为了有效地选择下一扩展结点，加速搜索的进程，在每一个活结点处，计算一个函数值（限界），并根据函数值，从当前活结点表中选择一个最有利的结点作为扩展结点，使搜索朝着解空间上有最优解的分支推进，以便尽快地找出一个最优解。

五、算法题

1. 给定已按升序排好序的n个元素a[0:n-1]，现要在这n个元素中找出一特定元素x，返回其在数组中的位置，如果未找到返回-1。

写出二分搜索的算法，并分析其时间复杂度。

```
1. template<class Type>
int BinarySearch(Type a[], const Type& x, int n)
{//在a[0:n]中搜索x，找到x时返回其在数组中的位置，否则返回-1
    Int left=0;  int  right=n-1;
    While (left<=right){
      int middle=(left+right)/2;
      if (x==a[middle]) return middle;
      if (x>a[middle]) left=middle+1;
      else right=middle-1;
     }
     Return -1;
}
```

时间复杂性为O(logn)

2. 利用分治算法写出合并排序的算法，并分析其时间复杂度

```
1. void MergeSort(Type a[], int left, int right)
  {
    if (left<right) {//至少有2个元素
```

```
      int i=(left+right)/2;  //取中点
      mergeSort(a, left, i);
      mergeSort(a, i+1, right);
      merge(a, b, left, i, right);  //合并到数组b
      copy(a, b, left, right);    //复制回数组a
      }
   }
```

算法在最坏情况下的时间复杂度为O(nlogn)。

3.N皇后回溯法

```
bool Queen::Place(int k)
{ //检查x[k]位置是否合法

  for (int j=1;j<k;j++)
    if ((abs(k-j)==abs(x[j]-x[k]))||(x[j]==x[k])) return false;
  return true;
}

void Queen::Backtrack(int t)
{
  if (t>n) sum++;
  else    for (int i=1;i<=n;i++) {
      x[t]=i;
      if (   约束函数  ) Backtrack(t+1);
    }
}
```

4.最大团问题

```
void Clique::Backtrack(int i) // 计算最大团
{  if (i > n) { // 到达叶结点
      for (int j = 1; j <= n; j++) bestx[j] = x[j];
      bestn = cn;   return;}
 // 检查顶点 i 与当前团的连接
   int OK = 1;
```

```
  for (int j = 1; j < i; j++)
    if (x[j] && a[i][j] == 0) {          // i与j不相连
      OK = 0;  break;}

  if (OK ) { // 进入左子树
    x[i] = 1;  cn++;
    Backtrack(i+1);
    x[i] = 0; cn--;  }

 if (     cn + n - i > bestn       ) { // 进入右子树
    x[i] = 0;
    Backtrack(i+1);  }
}
```

5.最长公共子序列问题

6．哈弗曼编码算法

算法设计与分析试题

一、概念题

1．队列　2．完全二叉树　3．堆　　4．P类问题　　5.NP问题

二、程序填空题

1．宽度优先图周游算法

```
procedure bft(g，n)
    / / g的宽度优先周游 / /
      declare visited(n)
      for iß1 to n do  / / 将所有结点标记为未访问 / /
          ____(1)____
      repeat
      for i<--1 to n do  / / 反复调用bfs / /
        if visited(i)＝0 then____(2)____ endif
      repeat
     end bft
```

2．找一个图的所有m—着色方案

procedure mcoloring(k)

/ / 这是图着色的一个递归回溯算法。图g用它的布尔邻接矩阵graPh(1：n，1：n)表示。它计算并打印出符合以下要求的全部解，把整数1，2，…，m分配给图中各个结点且使相邻近的结点的有不同的整数。k是下一个要着色结点的下标。 / /

global integer m，n，x(1：n)boolean graPh(1；n，1：n)

integer k

loop / / 产生对x(k)所有的合法赋值。 / /

call nextvalue(k)。//将一种合法的颜色分配给x(k) / /

if (1) then exit endif / / 没有可用的颜色了 / /

if (2)

then print(x) / / 至多用了m种颜色分配给n个结点 / /

else call mcoloring<k+1) / / 所有m—着色方案均在此反复递归调用中产生 / /

endif

repeat

end mcoloring

三、问答题

1.二分查找的思想是什么？

2．请用递归方法写出归并排序法的主要思想和算法。

3.已知如下多段图，请用动态规划方法的向后处理法写出求解此问题的递推公式并完成对各结点的计算。

4. 最小自然数：求具有下列两个性质的最小自然数n：

(1)n的个位数是6；

(2)若将n的个位数移到其余各位数字之前，所得的新数是n的4倍。

提示：仍用穷举法寻找，当找到一个符合条件者便停止。“找到便停止”的重复，宜采用repeat—until循环。

5. 以二叉链表为存储结构，分别写出求二叉树结点总数及叶子总数的算法。

博客建议(Suggestions)

2016年5月4日 19:17

I don't know if you will like the music. But I am sure there are some songs which are really wonderful and beautiful.

I hope everyone can enjoy the music and learn somethings from my blog.

Maybe I am not the best , but I will try my best to be better. I hope you can be aside with me, and with your words and suggestions , I believe I could do better. Whoever you are, wherever you are from, whenever you back, whatever you do , you are the special one. Not only because your IP address, but also the attitude to life and to stduy makes you be the only one. So no matter what happen to you, just do it and think up a way to solve it, maybe it isn't the best solution to problem, but at least you have you own idea, and your own thoughts. Keep in mind, the idea which firstly comes to you would the special one to you, don't mind other people's words, after all you have a chance to be yourself, and also you have a chance to make you be betteer. Therefore , not be afraid of making mistakes, not be afraid of changing. What you have done are all to get you a new height, and give you a new way to see the things around you.

You may think I can't do it, I can't finish it on my own, I can't solve it. Do not give up! The problem which you are facing is a chance to improve your ability. Just think the NP and P, no one can solve it. So your problem is just more easy when compared to this question. But if you realize that the problem you are facing is not just so easy, maybe you find a question which can have equal ranking with problem of P&NP. Life is so interseting, we always have to do something which really doesn't have a polynomial time to solve it. That why we are more clever than computer. Believe that each choose you made is the best choose, so that you can use greedy algorithm to get a solution, maybe it is not the best, but who knows.

Maybe you face a problem which is so huge that you can't solve it. Why not think to divide it into some small questions, that why we have Divide and Conquer Algorithm.

Something may seem no law at all.But if you be more careful, you can find the law contained in it. That the Regular algorithm application in our life.

You know that there are millions and thousands algorithm in computer science, that means there are millions and thousands solutions to solve the problems in our life.

So it is very important to study algorithm so that you will get many keys to success.

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5084441>

傅里叶：有关FFT,DFT与蝴蝶操作（转 重要！！！！重要！！！！真的很重要！！！！）

2016年5月4日　　19:18

转载地址：http://blog.renren.com/share/408963653/15068964503（作者： 徐可扬）

有没有！！！

其实我感觉这个学期算法最难最搞不懂的绝对不是动态规划啊！绝对是快速傅里叶变换啊！最近才弄懂有木有。

有不少人问我，于是干脆就写成日志吧。

首先明确一下基本概念吧，就三点，DFT,FFT,蝴蝶操作。

DFT（离散傅里叶变换）：书上写的最清楚的一句话叫做，向量y=(y0,y1,……yn-1)是系数向量a=(a0,a1,,……,an-1)的离散傅里叶变换，也写作y=DFTn(a)。说白了，就是求n个y值，但是n个自变量x的取值很特殊

是w_n^k(k=0,1,2,3……).

好，肯定很多人想知道什么是w_4^k，其实就是 n 次单位复根的 k 次方。

w_n^k=cos(2π*k/n)+sin(2π*k/n)*i (i^2=-1)。

FFT(快速傅里叶变换)：这个名词好理解，就是很快地算出这n个y值。一般我们计算n个n次的多项式值需要O(n^2)的时间。现在用FFT可以减少到O(n*logn)。具体原理一两句话说不完……要考的话也太理论性了…

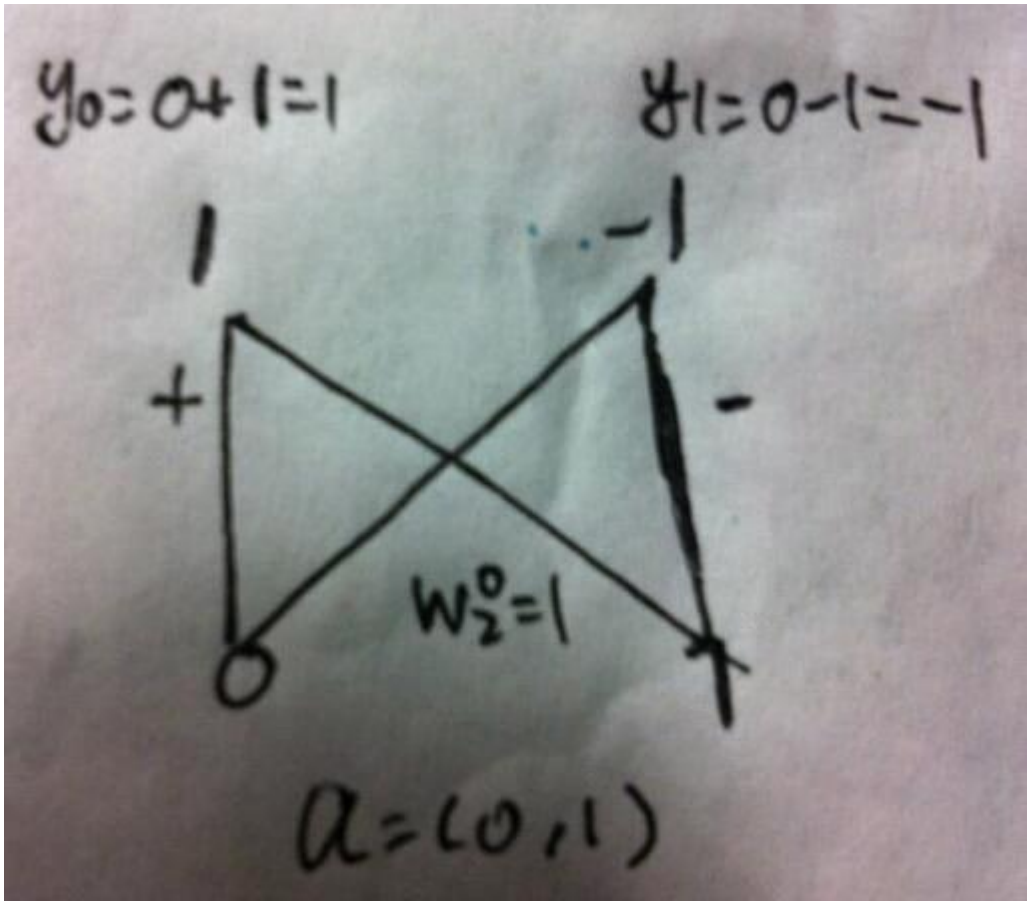
蝴蝶操作:这个应该是让大家最费解的…他就是一个FFT得实际应用…因为这是实践的东西所以要掌握。书上有一个图，虽然感觉看书不一定看的怎么懂，但是这个图还是要记住。不过我觉得从下往上写看起来更舒服。

转过来以后，向左边是加，向右边是减。谨记啊！

突然发现好难解释，先把宋老师课件里面上次坑爹的没有答案的课后练习拿来看看。

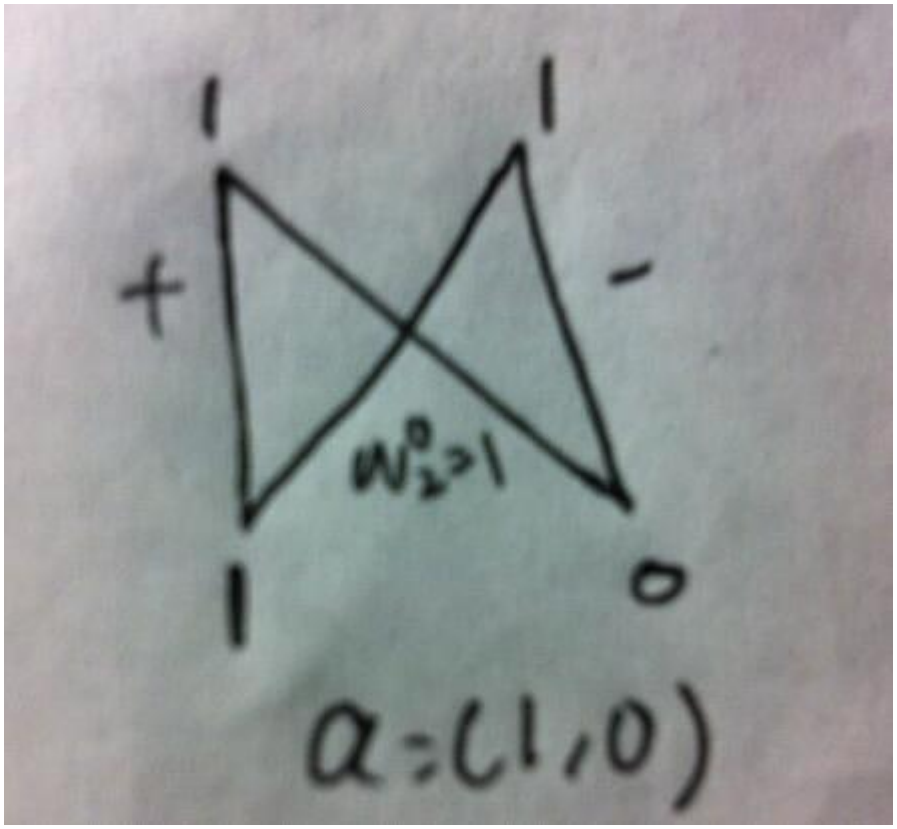
我这里a0,a1,a2……就不调换顺序了，其实是一样的，格式不同罢了。

a=(0,1)



经过FFT，Y=（1，-1），这个就是书上最简单的应用了。

再来一个简单的 a=(1,0).

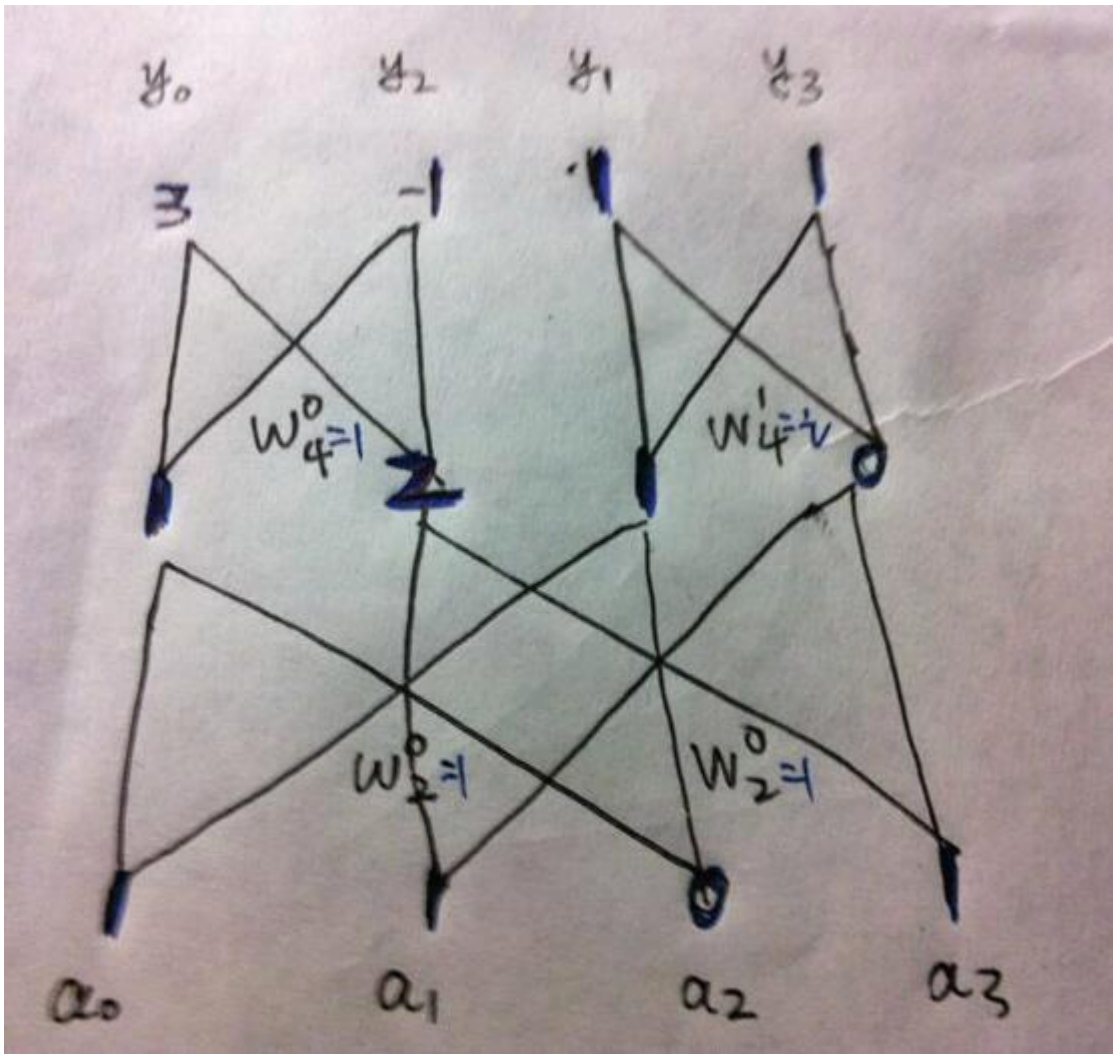


y=(1,1).

我觉得要考试一般就是考四个的吧，两个太简单了，8个的太复杂了（我后面有写）

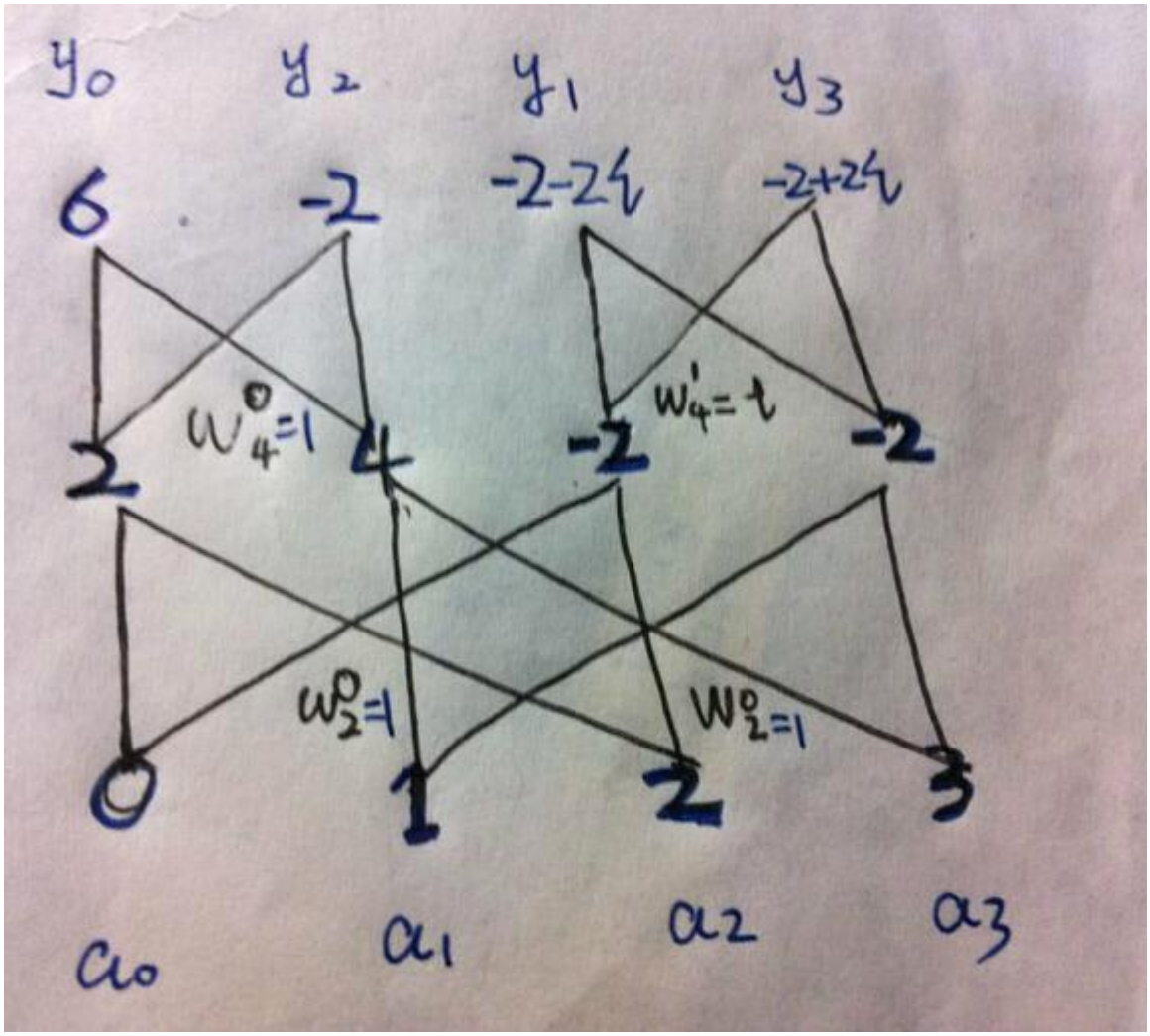
例子

a=(1,1,0,1)



y=(3,1,-1,1) 注意算出来是 y0,y2,y1,y3的顺序。

a=(0,1,2,3)

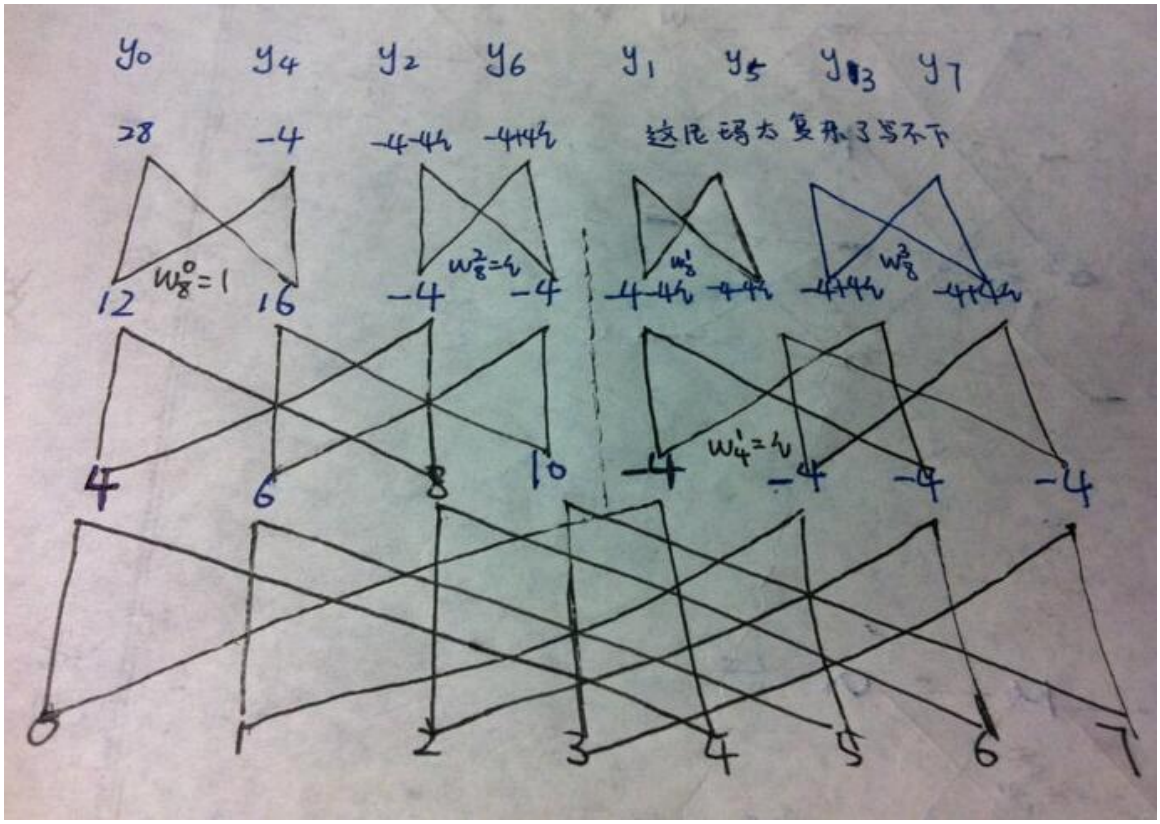


y=（6，-2-2i，-2 , -2+2i）.

注意右上角的蝶形运算中的旋转因子变成了(恩！？怎么不能插公式啊！)w(1/4）=i。。。那是因为在a0,a2的蝶形运算中我们多乘了(n/2)，这里n=2.（这个我解释不来…不过这个数是固定的，背就行了）。

那我上个八个的，有点复杂，估计可能不会考

a=(0,1,2,3,4,5,6,7).



y的顺序如图

告诉大家一个比较好的办法判断是否算对了，直接人肉使用O(n^2)算法就好了，举四个的例子。就是把x得值带入多项式y=a3*x^3+a2*x^2+a1*x+a0算出y值什么的。x0=1,x1=i，x2=-1,x3=-i带入。这样的话，其实考察算没算对完全是个伪命题吗……

再提醒一次，左加右减哦！

最后再说一下这玩意有什么用，什么信号学的废话就不说了。无非是FFT用N*LogN时间算出点值方便用点值法算出多项式乘法的系数结果，这个大家看书上510的图我觉得就能理解了。插值神马的，背公式吧……虽然我也不记得了。

时间有点紧迫，感觉写的巨烂无比，大神们如果发现写错了留言给我立马改，关键不要误人子弟……

这有帮助的请让我知道。感谢大家!

If it helps you , let me know. Thank you !

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5081155>

钢管切割（动态规划）

2016年5月4日　　19:18

钢管价格pi以及对应长度按照下面规则：

长度i	1	2	3	4	5	6	7	8	9	10
价格pi	1	5	8	9	10	17	17	20	24	30

动态规划的自底向上：

```
#include<bits/stdc++.h>//用多了,不知道具体该用哪些头文件
#define MAX 1000010 //因为之前的一道题少了一个0,所以数组挺大的
#define INF -100010//去掉负号就是我之前经常用的
using namespace std;
int p[MAX],r[MAX],s[MAX]//p记录钢管价格，r记录子问题的长度,s记录左端钢管长度
int cut_rod(int *p,int *r,int *s,int n)
{
    r[0]=0;
    for(int j=1;j<=n;j++)
    {
        int q=INF;
        for(int i=1;i<=j;i++)
        {
            if(q<p[i]+r[j-i])
            {
                q=p[i]+r[j-i];
                s[j]=i;
            }
        }
        r[j]=q;
    }
    return r[n];
 }
 void print(int *p,int *s,int n)
 {
     while(n>0)
     {
         cout<<s[n]<<" ";
         n-=s[n];
     }
 }
int main()
{
    for(int i=1;i<=10;i++)
    {
        cin>>p[i];
    }
    int n;
    while(cin>>n)
    {
    cout<<cut_rod(p,r,s,n)<<endl;
    print(p,s,n);
    cout<<endl;
    }
 }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5079064>

LeetCode 8 String to Integer (string转int)

2016年5月4日 19:18

题目来源：https://leetcode.com/problems/string-to-integer-atoi/

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

Update (2015-02-10):

The signature of the C++ function had been updated. If you still see your function signature accepts a const char * argument, please click the reload button to reset your code definition.

解题思路：

照着要求写代码，可以总结如下：

1. 字串为空或者全是空格，返回0；

2. 字串的前缀空格需要忽略掉；

3. 忽略掉前缀空格后，遇到的第一个字符，如果是‘+’或‘—’号，继续往后读；如果是数字，则开始处理数字；如果不是前面的2种，返回0；

4. 处理数字的过程中，如果之后的字符非数字，就停止转换，返回当前值；

5. 在上述处理过程中，如果转换出的值超出了int型的范围，就返回int的最大值或最小值。

Java代码：

```
 1 public class Solution {
 2     public int myAtoi(String str) {
 3         int max = Integer.MAX_VALUE;
 4         int min = -Integer.MIN_VALUE;
 5         long result = 0;
 6         str = str.trim();
 7         int len = str.length();
 8         if (len < 1)
 9             return 0;
10         int start = 0;
11         boolean neg = false;
12 
13         if (str.charAt(start) == '-' || str.charAt(start) == '+') {
14             if (str.charAt(start) == '-')
15                 neg = true;
16             start++;
17         }
18 
19         for (int i = start; i < len; i++) {
20             char ch = str.charAt(i);
21 
22             if (ch < '0' || ch > '9')
```

```
23                  break;
24              result = 10 * result + (ch - '0');
25              if (!neg && result > max)
26                  return max;
27              if (neg && -result < min)
28                  return min;
29
30          }
31          if (neg)
32              result = -result;
33
34          return (int) result;
35      }
36
37  };
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5076830>

LeetCode 7 Reverse Integer（反转数字）

2016年5月4日 19:18

题目来源：https://leetcode.com/problems/reverse-integer/

Reverse digits of an integer.

Example1: x = 123, return 321

Example2: x = -123, return -321

解题思路：

其实这道题看起来非常简单，要实现也是几行代码的事。但是有个小问题容易被忽略,就是边界问题。什么意思呢？如果我们输入的整数超出了int的表达范围，这个问题要怎么解决呢？
用比int更大的数据类型存储我们转换后的结果，然后与int的边界比较，超出了边界则返回0。
Java实现：

```
public class Solution {
    public int reverse(int x) {
        long reverse = 0;

        while(x != 0){
            reverse = reverse * 10 + x % 10;
            if(reverse > Integer.MAX_VALUE || reverse < Integer.MIN_VALUE)
                return 0;
            x = x / 10;
        }
        return (int)reverse;
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5076819>

LeetCode 6 ZigZag Conversion（规律）

2016年5月4日　　19:19

题目来源：https://leetcode.com/problems/zigzag-conversion/

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string text, int nRows);
```

convert("PAYPALISHIRING", 3) should return "PAHNAPLSIIGYIR".

解题思路：

Zigzag:即循环对角线结构（

```
0                           8                           16
1                     7     9                     15    17
2               6           10              14          18
3       5                   11        13                19
4                           12                          20
```

）

给出代码：

```
 1 class Solution {
 2 public:
 3     string convert(string s, int nRows){
 4     if(nRows == 1) return s;
 5     string res[nRows];
 6     int i = 0, j, gap = nRows-2;
 7     while(i < s.size()){
 8         for(j = 0; i < s.size() && j < nRows; ++j) res[j] += s[i++];
 9         for(j = gap; i < s.size() && j > 0; --j) res[j] += s[i++];
10     }
11     string str = "";
12     for(i = 0; i < nRows; ++i)
13         str += res[i];
```

```
14      return str;
15      }
16  };
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5076802>

LeetCode 5 Longest Palindromic Substring（最长子序列）

2016年5月4日 19:19

题目来源：https://leetcode.com/problems/longest-palindromic-substring/

Given a string *S*, find the longest palindromic substring in *S*. You may assume that the maximum length of *S* is 1000, and there exists one unique longest palindromic substring.

动态规划，类似于lcs的解法，数组flag[i][j]记录s从i到j是不是回文

首先初始化，i>=j时，flag[i][j]=true，这是因为s[i][i]是单字符的回文，当i>j时，为true，是因为有可能出现flag[2][1]这种情况，比如bcaa，当计算s从2到3的时候，s[2]==s[3]，这时就要计算s[2+1] ?= s[3-1]，总的来说，当i>j时置为true，就是为了考虑j=i+1这种情况。

接着比较s[i] ?= s[j]，如果成立，那么flag[i][j] = flag[i+1][j-1]，否则直接flag[i][j]=false

c++代码：

```
class Solution {
public:
    string longestPalindrome(string s) {
        int len = s.length(), max = 1, ss = 0, tt = 0;
        bool flag[len][len];
        for (int i = 0; i < len; i++)
            for (int j = 0; j < len; j++)
                if (i >= j)
                    flag[i][j] = true;
                else flag[i][j] = false;
        for (int j = 1; j < len; j++)
            for (int i = 0; i < j; i++)
            {
                if (s[i] == s[j])
                {
                    flag[i][j] = flag[i+1][j-1];
                    if (flag[i][j] == true && j - i + 1 > max)
                    {
                        max = j - i + 1;
                        ss = i;
                        tt = j;
                    }
                }
                else flag[i][j] = false;
            }
        return s.substr(ss, max);
    }
};
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5076791>

LeetCode 4 Median of Two Sorted Arrays (两个数组的mid值)

2016年5月4日 19:19

题目来源：https://leetcode.com/problems/median-of-two-sorted-arrays/

There are two sorted arrays nums1 and nums2 of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

解题思路：

题目是这样的：给定两个已经排序好的数组（可能为空），找到两者所有元素中第k大的元素。另外一种更加具体的形式是，找到所有元素的中位数。本篇文章我们只讨论更加一般性的问题：如何找到两个数组中第k大的元素？不过，测试是用的两个数组的中位数的题目，Leetcode第4题 Median of Two Sorted Arrays

方案1：假设两个数组总共有n个元素，那么显然我们有用O(n)时间和O(n)空间的方法：用merge sort的思路排序，排序好的数组取出下标为k-1的元素就是我们需要的答案。

这个方法比较容易想到，但是有没有更好的方法呢？

方案2：我们可以发现，现在我们是不需要“排序”这么复杂的操作的，因为我们仅仅需要第k大的元素。我们可以用一个计数器，记录当前已经找到第m大的元素了。同时我们使用两个指针pA和pB，分别指向A和B数组的第一个元素。使用类似于merge sort的原理，如果数组A当前元素小，那么pA++，同时m++。如果数组B当前元素小，那么pB++，同时m++。最终当m等于k的时候，就得到了我们的答案——O(k)时间，O(1)空间。

但是，当k很接近于n的时候，这个方法还是很费时间的。当然，我们可以判断一下，如果k比n/2大的话，我们可以从最大的元素开始找。但是如果我们要找所有元素的中位数呢？时间还是O(n/2)=O(n)的。有没有更好的方案呢？

我们可以考虑从k入手。如果我们每次都能够剔除一个一定在第k大元素之前的元素，那么我们需要进行k次。但是如果每次我们都剔除一半呢？所以用这种类似于二分的思想，我们可以这样考虑：

Assume that the number of elements in A and B are both larger than k/2, and if we compare the k/2-th smallest element in A(i.e. A[k/2-1]) and the k-th smallest element in B(i.e. B[k/2 - 1]), there are three results:

(Becasue k can be odd or even number, so we assume k is even number here for simplicy. The following is also true when k is an odd number.)

A[k/2-1] = B[k/2-1]

A[k/2-1] > B[k/2-1]

A[k/2-1] < B[k/2-1]

if A[k/2-1] < B[k/2-1], that means all the elements from A[0] to A[k/2-1](i.e. the k/2 smallest elements in A) are in the range of k smallest elements in the union of A and B. Or, in the other word, A[k/2 - 1] can never be larger than the k-th smalleset element in the union of A and B.

Why?

We can use a proof by contradiction. Since A[k/2 - 1] is larger than the k-th smallest element in the union of A and B, then we assume it is the (k+1)-th smallest one. Since it is smaller than B[k/2 - 1], then B[k/2 - 1] should be at least the (k+2)-th smallest one. So there are at most (k/2-1) elements smaller than A[k/2-1] in A, and at most (k/2 - 1) elements smaller than A[k/2-1] in B.So the total number is k/2+k/2-2, which, no matter when k is odd or even, is surly smaller than k(since A[k/2-1] is the (k+1)-th smallest element). So A[k/2-1] can never larger than the k-th smallest element in the union of A and B if A[k/2-1]<B[k/2-1];

Since there is such an important conclusion, we can safely drop the first k/2 element in A, which are definitaly smaller than k-th element in the union of A and B. This is also true for the A[k/2-1] > B[k/2-1] condition, which we should drop the elements in B.

When A[k/2-1] = B[k/2-1], then we have found the k-th smallest element, that is the equal element, we can call it m. There are each (k/2-1) numbers smaller than m in A and B, so m must be the k-th smallest number. So we can call a function recursively, when A[k/2-1] < B[k/2-1], we drop the elements in A, else we drop the elements in B.

We should also consider the edge case, that is, when should we stop?

1. When A or B is empty, we return B[k-1](or A[k-1]), respectively;

2. When k is 1(when A and B are both not empty), we return the smaller one of A[0] and B[0]

3. When A[k/2-1] = B[k/2-1], we should return one of them

In the code, we check if m is larger than n to garentee that the we always know the smaller array, for coding simplicy.

Java实现：

```
 1 public class Solution {
 2     public double findMedianSortedArrays(int[] nums1, int[] nums2) {
 3         int m = nums1.length, n = nums2.length;
 4         int k = (m + n) / 2;
 5         if((m+n)%2==0){
 6             return
(findKth(nums1,nums2,0,0,m,n,k)+findKth(nums1,nums2,0,0,m,n,k+1))/2;
 7         }   else {
 8             return findKth(nums1,nums2,0,0,m,n,k+1);
 9         }
10
11     }
12
13     private double findKth(int[] arr1, int[] arr2, int start1, int start2, int
len1, int len2, int k){
14         if(len1>len2){
```

```
15              return findKth(arr2,arr1,start2,start1,len2,len1,k);
16          }
17          if(len1==0){
18              return arr2[start2 + k - 1];
19          }
20          if(k==1){
21              return Math.min(arr1[start1],arr2[start2]);
22          }
23          int p1 = Math.min(k/2,len1) ;
24          int p2 = k - p1;
25          if(arr1[start1 + p1-1]<arr2[start2 + p2-1]){
26              return findKth(arr1,arr2,start1 + p1,start2,len1-p1,len2,k-p1);
27          } else if(arr1[start1 + p1-1]>arr2[start2 + p2-1]){
28              return findKth(arr1,arr2,start1,start2 + p2,len1,len2-p2,k-p2);
29          } else {
30              return arr1[start1 + p1-1];
31          }
32      }
33  }
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5076786>

听说你会打地鼠（动态规划dp）

2016年5月4日　　19:19

题目来源：https://biancheng.love/contest-ng/index.html#/41/problems

G 听说你会打地鼠

时间限制：300ms　内存限制：65536kb

题目描述

打地鼠是个很简单的游戏，不过你知道怎么打地鼠才能最省力吗？

每时刻，都有N只地鼠在pi(xi,yi)位置出现，打掉一只，该时刻其他地鼠会消失。

打掉第一只地鼠不消耗能量，之后每只地鼠消耗的能量约与鼠标移动距离成正比，即cost = dis(pi,pi+1)（平面上两点距离怎么求不多说了）

现在认为打第一只地鼠不消耗能量，那么打完所有地鼠消耗的最小能量是多少？

输入

多组测试数据

每组测试数据第一行两个整数为时长K和每时刻地鼠数量N

接下来N行每行2N个整数表示N只地鼠坐标

N<=100,K<40

输出

对于每组数据，输出一行，为最小消耗,结果保留3位小数

输入样例

```
2 2
1 1 3 4
2 2 5 3
```

输出样例

```
1.414
```

解题思路：

状态转移方程：

```
dp[k][i]=dp[k-1][j]+cost(p[k][i],p[k-1][j]);
```

说明：dp[k][j] 表示打第k层第j个的时候所能得到的最小值

给出代码：

```
#include <bits/stdc++.h>
#define INF 9999999999
```

```
 3
 4 using namespace std;
 5 struct Point{
 6     double x;
 7     double y;
 8 };
 9
10 double cost(Point a,Point b)
11 {
12     return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
13 }
14
15 Point p[110][110];
16 double dp[110][110];
17 int k,n,i,j,f;
18 double ans;
19 int main()
20 {
21     while(~scanf("%d%d",&k,&n))
22     {
23         ans=INF;
24         for(i=0;i<k;i++)//k层
25         {
26             for(j=0;j<n;j++)//n个
27             {
28                 scanf("%lf%lf",&p[i][j].x,&p[i][j].y);//x表示层数，y表示第几个，因此
p[x][y]可以确定所选择的地鼠层数以及在该层的位置
29             }
30         }
31         for(f=0;f<k;f++)
32         {
33             for(i=0;i<n;i++)
34             {
35                 dp[f][i]=INF;
36             }
37         }
38        // memset(find_min,INF,sizeof(find_min));
39         for(f=0;f<n;f++)
40         {
41             dp[0][f]=0;//第一只不需要消耗能量
42         }
43         for(f=1;f<k;f++)//第f层
44         {
45             for(i=0;i<n;i++)
46             {
47                 for(j=0;j<n;j++)
48                 {
49                     if(dp[f][i]>(dp[f-1][j]+cost(p[f][i],p[f-1][j])))//状态转移方
程
50                         dp[f][i]=dp[f-1][j]+cost(p[f][i],p[f-1][j]);//在第f层最小
消耗能量等于在第f-1层打第j个地鼠+两点之间距离
51                 }
52             }
53         }
54         for(i=0;i<n;i++)
55         {
56             if(ans>dp[k-1][i])
57                 ans=dp[k-1][i];
58         }
59         printf("%.3lf\n",ans);
60     }
61 }
```

推荐博客：http://www.tuicool.com/articles/QV7rQjZ

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5075713>

似乎该博弈了！（动态规划）

2016年5月4日　　19:19

题目来源：https://biancheng.love/contest-ng/index.html#/41/problems

F 似乎该博弈了！

题目描述

nova君陷入了困境，因为他无法在PS4游戏上凭借操作战胜对手。机智如他，只好和对手博弈了！

nova君拿出的方案和以往有些不一样，他说：“你可以决定石子数量和石子的取法，我来决定先后手，这样非常公平。”他的对手觉得nova君说的有道理，于是不仅决定了每局的石子数量，还给出了k个数字，表示每次可以任意取走数量等同于这k个数字中一个的石子，k中一定有一个为1。先取完者为胜。

现在很急很关键，快帮nova君看看他到底应该先手还是后手才能战胜对手。

输入

每组测试数据两行。

第一行两个整数n和k，第二行k个整数，意义如题目描述。

N<=100000,k<=15

输出

对于每组数据，输出一行，为nova应该采取的先后手 sente \ gote

输入样例

```
4 3
1 2 3
1 1
1
```

输出样例

```
gote
sente
```

解题思路：给定n个石头，和k种去除石头的方式，每种方式可以去除一定量的石头， 现在Sente(简称S),gote(简称O)，S先手，O后手，每次每个人能选择一种去除石头的方式，谁去除最后一堆谁就赢了。要求出必胜之人是谁。

分析：

1.用一个dp数组记录，对于先手者能取到的记录为1，后手者为0.

2.初始dp数组都为0，遍历1到n，如果dp[i]为0，说明上一手是后手取得，这样先手就能取，把dp[i]变为1，由于是从1到n，这样每个状态记录时，前面的都已经记录好了，所以是可行的.

3.这样最后只需要判断dp[n]是1，还是0，就可以判断是先手胜还是后手胜

了。

状态转移方程为：if (i - mjmj[j] >= 0 && !dp[i - mjmj[j]]) dp[i] = 1.

给出代码：

```
#include <bits/stdc++.h>
#define MAX 1000010
int dp[MAX],mjmj[15];

int n,m,i,j;
int main()
{
    while(~scanf("%d",&n))
    {
        memset(dp,0,sizeof(dp));
        scanf("%d",&m);
        for (i=0; i<m; i++)
        {
            scanf("%d",&mjmj[i]);
        }
        for (i=1;i<=n;++i)
        {
            for (j=0;j<m;++j)
            {
                if (i-mjmj[j]>=0&&!dp[i-mjmj[j]])
                {
                    dp[i]=1;
                    break;
                }
            }
        }
        if (dp[n])
            printf("sente\n");
        else
            printf("gote\n");
    }
    return 0;
}
```

推荐博客：http://blog.csdn.net/kjc19920531/article/details/8120989

推荐博客：http://blog.csdn.net/wangtaoking1/article/details/7308117

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5075680>

网络传输（最大重叠次数）

2016年5月4日　　19:20

题目来源：https://biancheng.love/contest-ng/index.html#/41/problems

E 网络传输

题目描述

网络上的数据传输是共用数据线的，当一台主机发送传输请求时网络上有其他主机在发送数据，就不能成功发送只能等待。如果网络经常处于占用情况，平均等待时间过长，可能说明网络负载过重、拓扑结构不好或延时算法有问题。为了检测是否有这些问题，可以在服务器之间发送一些数据查看响应时间。

不过现在，我们姑且简化问题，将网络看做是一维的线段，各个主机看做线段上的点。当主机i向主机j发送数据时，i->j窗口所有的主机此时都不能发送数据。每次数据传输为1单位时间（由于网线上数据传输速度很快，不计距离影响）。

下面给出一组测试，求完成所有传输请求需要使用的最少的时间。

输入

多组测试数据，每组N+1行。

每组第一行数为题目描述中一组测试的请求数n

接下来n行，每行2个整数i,j，表示一次数据传输i->j。

1<=i,j<=N<=10000

输出

对于每组数据，输出一行，为最小用时。

输入样例

```
3
1 3
3 4
5 1
```

输出样例

```
3
```

解题思路：找到网络传输中最拥堵的地方，由于是单位时间的传输，因此拥堵的程度也就是最短的传输时间。

网络拓扑、AOV网络、AOE网络.

AOV网络：通常把计划、施工过程、生产流程和程序流程等都当作成一个工程。除了很小的工程外，一般都把工程划分为若干个成为“活动”的子工程。完成了这些活动，这个工程也就完成了。简称为AOV网络（Activity On Vertices）

AOE网络：如果在无有向环的有向带权图中用有向边表示一个工程中的各项活动，用有向边上的权重表示活动的持续时间，用顶点表示事件，则这样的

有向图叫做用边表示活动的网络，简称为AOE网络（Activity On Edges）

在本题中，需要求出最少时间.本网络一共包括了n个操作,我们为了求出最短时间只需要确定操作的顺序.但是如果暴力的话，一共有n!种操作顺序，因此这种思路不可以。

分析：

1.数轴上一共有n条线段,需要找到相交处最多的地方。

2.由于网络传输数据本一定是按照从小到大的顺序，可以在每次输入数据之后进行判断i,j的大小,然后将i->j上的每个点（整数点）拥堵频率都递加1.

3.找到最拥堵的地方,采用sort时间复杂度为O(nlgn),但是由于只需要找到最大的拥堵，也就是顺序统计量的查找.最大值查找的时间复杂度为线性时间.

给出代码：

```
#include <bits/stdc++.h>
#define MAX 10010

using namespace std;
int a[MAX];
int n,t,i,j,f,l,r,num;//l,r减小循环区间长度

int main()
{
    while(~scanf("%d",&n))
    {
        num=0;
        l=10010;//左值
        r=-1;//右值
        memset(a,0,sizeof(a));
        for(t=1; t<=n; t++)
        {
            scanf("%d %d",&i,&j);
            if(i>j)//完成交换
            {
                i^=j;
                j^=i;
                i^=j;
            }
            if(l>i)
                l=i;
            if(r<j)
                r=j;
            for(f=i; f<=j; f++)
            {
                a[f]++;
            }
        }
        for(f=l; f<=r; f++)//找到最大值
        {
            if(num<a[f])
                num=a[f];
        }
        printf("%d\n",num);
    }
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5075662>

jhljx跑跑跑（找规律）

2016年5月4日　　19:20

[题目来源：https://biancheng.love/contest/41/problem/D/index](https://biancheng.love/contest/41/problem/D/index)

jhljx跑跑跑

题目描述

数学不好的jhljx又在和别人打牌，他们一共m人每人n张牌，牌只有点数没有花色，从1点到k点一共n*m张。每轮每个人出一张牌，只要jhljx出的牌比其余m-1个人都大，jhljx这轮就能跑掉。现在jhljx手里的牌是什么全都可以告诉你，那么他至少能跑掉几次呢？

没时间解释了，jhljx快跑了，现在很急很关键！

输入

多组测试数据，每组测试数据两行。

第一行为m,n两个整数，第二行为n个整数，表示jhljx手牌点数。

n*m<=10000

输出

对于每组测试数据输出一行为jhljx能赢的次数。

输入样例

```
2 3
1 3 6
```

输出样例

1

解题思路：其实就是一共有m个人每个人手里拿着n张牌。(牌的大小从1到n*m即total)，题目要求找到获胜的最少次数，也就是至少获胜多少次。现在分析怎样才能达到至少的获胜次数。

分析：

1、已经知道自己拿了什么牌，因为牌是已知的，所以也知道剩下的牌是什么了；

2、将自己手中的牌进行排序，整理出合适的牌面；

3、其他人作弊把手里的牌都合在一起了，所以一共两个人打牌，（其实是一个人在打牌），这样的话我每次都出自己手里最大的牌，对面的人只要能大得过就输掉，直到手里的牌出完。

4、在这里需要说明一下，其他人的牌合在一起只是为了方便理解，但是在具体实现中还是要注意自己的牌是n个

给出代码：

```
#include <bits/stdc++.h>

using namespace std;
int a[10010];
int b[10010];
int m,n,total,tn,i,j,num,counter;
int main()
{
    while(~scanf("%d%d",&m,&n))
    {
        total=n*m;
        tn=n;
        counter=num=0;
        for(i=1;i<=n;i++)
            scanf("%d",&a[i]);
        for(i=1;i<=total;i++)
            b[i]=i;
        sort(a+1,a+n+1);
//        for(i=1;i<=n;i++)
//            printf("%d ",a[i]);
//        printf("\n");
        for(i=total;i>=1&&tn>0;i--)
        {
            if(a[tn]!=b[i])
                counter++;
            else
            {
                if(0==counter)
                    num++;
                else
                    counter--;
                tn--;
            }
        }
        printf("%d\n",num);
    }
    return 0;
}
```

来自 <http://i.cnblogs.com/EditPosts.aspx?postid=5074762>