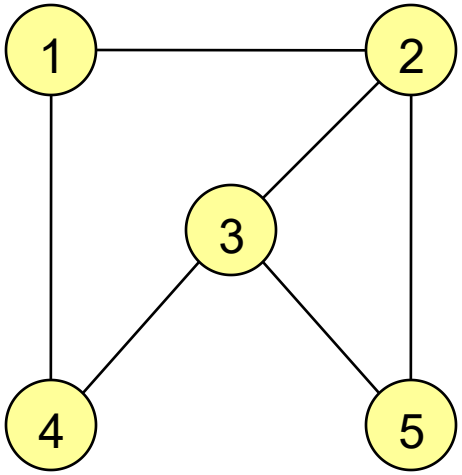
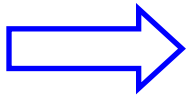
四 、图的搜索算法

4.1 图的存储表示

- 邻接矩阵表示法

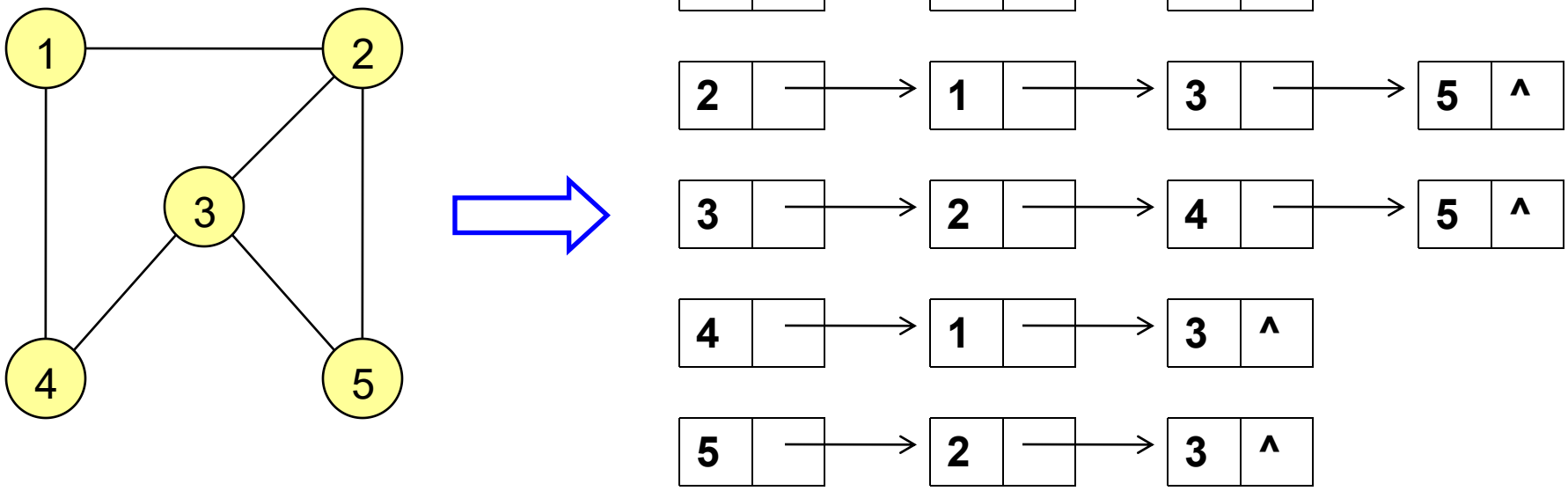




	①	②	③	④	⑤
①	0	1	0	1	0
②	1	0	1	0	1
③	0	1	0	1	1
④	1	0	1	0	0
⑤	0	1	1	0	0

4.1 图的存储表示

- 邻接表表示法



4.2 图的遍历

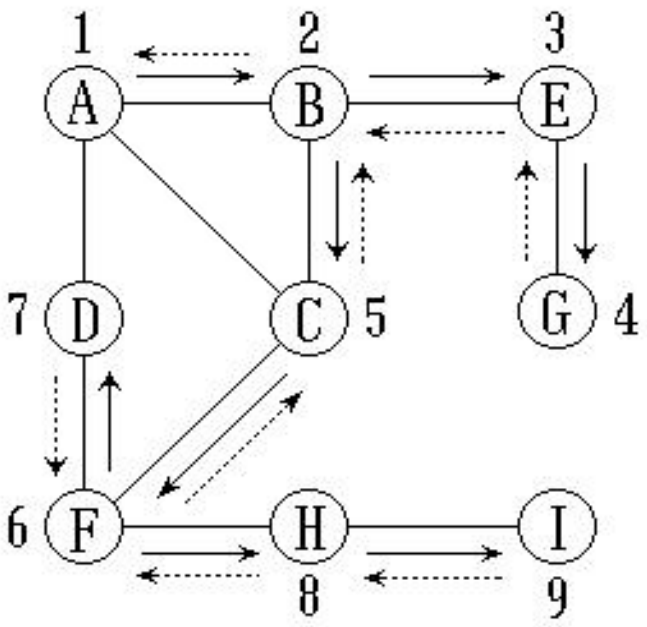
- 图的遍历：从图中某一顶点出发访遍图中其余顶点，且使每个顶点仅被访问一次。
- 两条遍历图的路径：深度优先搜索、广度优先搜索

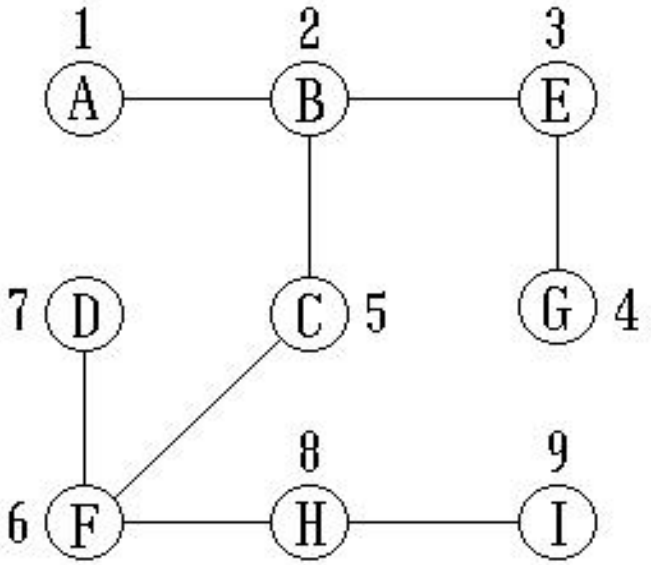
4.2 图的遍历

- 深度优先搜索DFS (Depth First Search)
  - 访问指定的某顶点V，将V作为当前顶点
  - 访问当前顶点的下一未被访问过的邻接点，并以该邻接点作为当前顶点
  - 重复2，直到所有和当前顶点有路径相通的顶点都被访问到
  - 沿搜索路径回退，退到尚有邻接点未被访问过的某结点
  - 将该结点作为当前结点，重复以上步骤，直到所有顶点都被访问过的为止

4.2 图的遍历

- 深度优先搜索DFS (Depth First Search)





4.3 图的搜索算法

- 深度优先搜索算法

```
int g[100][100]={……};     //初始化邻接矩阵
int n;        //图中顶点个数
dfs(int k)    //从邻接矩阵中顶点k的那行开始
{
   int j;
   printf("搜索到%d\n",k);
   visited[k]=1;        //k顶点设置为已访问过
   for(j=1;j<=n;j++)       //依次搜索k的邻接点
        if(g[k][j]==1&&visited[j]==0)  //判断1——n顶点与k顶点是否邻接边，
                                       如果是且未被访问过
              dfs(j);    //则从邻接矩阵中该顶点那行出发继续深度优先搜索
}
```

4.3 图的搜索算法

- 例：走迷宫问题（DFS）
  - 迷宫是许多小方格构成的矩形，在每个小方格中有的是墙（1）有的是路（0）。设迷宫的入口是在左上角(1,1)，出口是右下角(8,8)。根据给定的迷宫，找出一条从入口到出口的路径。

0	0	0	0	0	0	0	0
0	1	1	1	1	0	1	0
0	0	0	0	1	0	1	0
0	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0
0	1	0	0	0	0	1	1
0	1	0	0	1	0	0	0
0	1	1	1	1	1	1	0

4.3 图的搜索算法

- 算法设计：
  - 深度优先搜索一直向着可通行的下一个方格行进，直到搜索到出口就找到一个解。若行不通时，则返回上一个方格，继续搜索其它方向。
  - 迷宫中任意方格A（X，Y），有四个搜索方向：

    向左A（X-1，Y）　　向右A（X+1，Y）

    向上A（X，Y-1）　　向下A（X，Y+1）
  - 当对应方格值为0时可通行
  - 把已经访问过的方格值改为2，不另外开辟空间记录其访问的情况

4.3 图的搜索算法

- 算法设计：
  - 用数组fx={-1,1,0,0},fy={0,0,-1,1}模拟左右上下搜索时的下标的变化过程。
  - 当一个方格四个方向都搜索完还没有走到出口，说明该方格或无路可走或走入了“死胡同”，方格值改为-1

4.3 图的搜索算法

■算法实现：

```
int
a[9][9]={{0},{0,0,0,0,0,0,0,0,0},{0,0,1,1,1,1,0,1,0},{0,0,0,0,0,1,0,1,0},
{0,0,1,0,0,0,0,1,0},{0,0,1,0,1,1,0,1,0},{0,0,1,0,0,0,0,1,1},{0,0,1,0,0,1,
0,0,0},{0,0,1,1,1,1,1,1,0}};
int fx[5]={0,-1,1,0,0},fy[5]={0,0,0,-1,1},i,j,k,total=0;
void  Out()
{       printf("\n路径示意图：\n");
        for( i=1;i<=8;i++)
        {       for(j=1;j<=8;j++)
                        if(a[i][j]==2)
                        {       printf("0 ");
                                total++;
                        }
                        else
                                printf("* ");
                printf("\n");
        }
}
```

4.3 图的搜索算法

```
int check(int i,int j,int k)
{
        int flag=1;
        i= i+fx[k];
        j= j+fy[k];
        if(i<1||i>8||j<1||j>8) //若坐标移动超出了迷宫范围
                flag=0;
        else
                if(a[i][j]!=0)  //若不可通行，1（墙）、2（已走过）、-1（死胡同
                        flag=0;
        return(flag);
 }
```

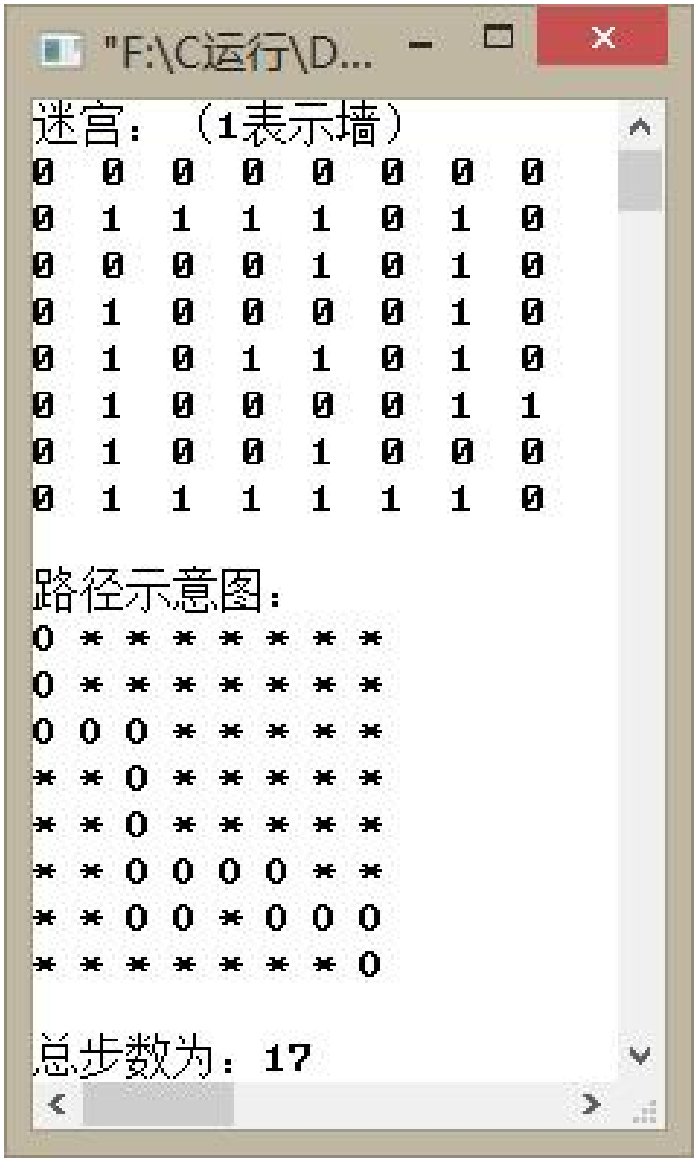
4.3 图的搜索算法

```
void search(int i, int j)
{
        int k,newi,newj;
        for(k=1;k<=4;k++)        //分四个方向搜索可通行的方格
                if(check(i,j,k)==1)     //如果某一方向可通行
                {
                        newi=i+fx[k];
                        newj=j+fy[k];
                        a[newi][newj]=2;       //把该方向的方格设置为已走过
                        if(newi==8&&newj==8)     //如果到了出口则输出路径
                                Out( );
                        else
                                search(newi,newj);  //否则从该方向方格继续搜索
                }
        a[i][j]=-1;     //分四个方向搜索都不可通行，则该方格走入死胡同
}
```

4.3 图的搜索算法

```
int main()
{
        printf("迷宫：（1表示墙）\n");
        for(i=1;i<=8;i++)
                {
                for(j=1;j<=8;j++)
                        printf("%-3d",a[i][j]);
                printf("\n");
                }
        a[1][1]=2;  //入口设置已走标志2
        search(1,1);
    printf("\n总步数为：%d\n",total);
    return 0;
}
```

4.3 图的搜索算法


"F:\C运行\D...
迷宫：（1表示墙）
0 0 0 0 0 0 0 0
0 1 1 1 1 0 1 0
0 0 0 0 1 0 1 0
0 1 0 0 0 0 1 0
0 1 0 1 1 0 1 0
0 1 0 0 0 0 1 1
0 1 0 0 1 0 0 0
0 1 1 1 1 1 1 0
路径示意图：
0 * * * * * * *
0 * * * * * * *
0 0 0 * * * * *
* * 0 * * * * *
* * 0 * * * * *
* * 0 0 0 0 * *
* * 0 0 * 0 0 0
* * * * * * * 0
总步数为：17

例2:

设计一个考试日程安排表，使在尽可能短的时间内安排完考试，要求同一个学生选修的几门课程不能安排在同一个时间内。

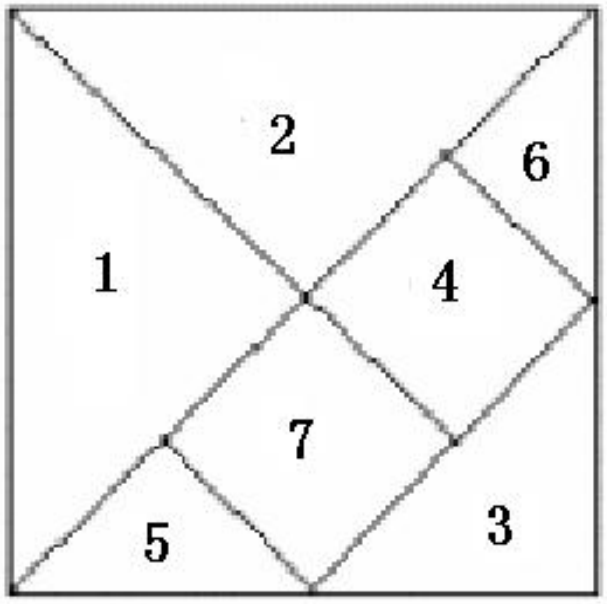
姓名	选修1	选修2	选修3
张	A课程	B课程	E课程
王	C课程	D课程	
赵	C课程	E课程	F课程
李	D课程	F课程	A课程
陈	B课程	F课程	

- 解决该问题，用无向图数据结构，图中的顶点表示课程，不能同时考试的课程之间连上一条边。用尽可能少的颜色去给图中每个顶点着色，使得任意两个相邻的顶点着不同的颜色。同一种颜色表示一个考试时间。

- 答案：1：A,C　2:B,D　3:E　4: F

4.3 图的搜索算法

- 例：有如图所示的七巧板，使用至多四种不同颜色对七巧板进行涂色(每块涂一种颜色)，要求相邻区域的颜色互不相同，打印输出所有可能的涂色方案。



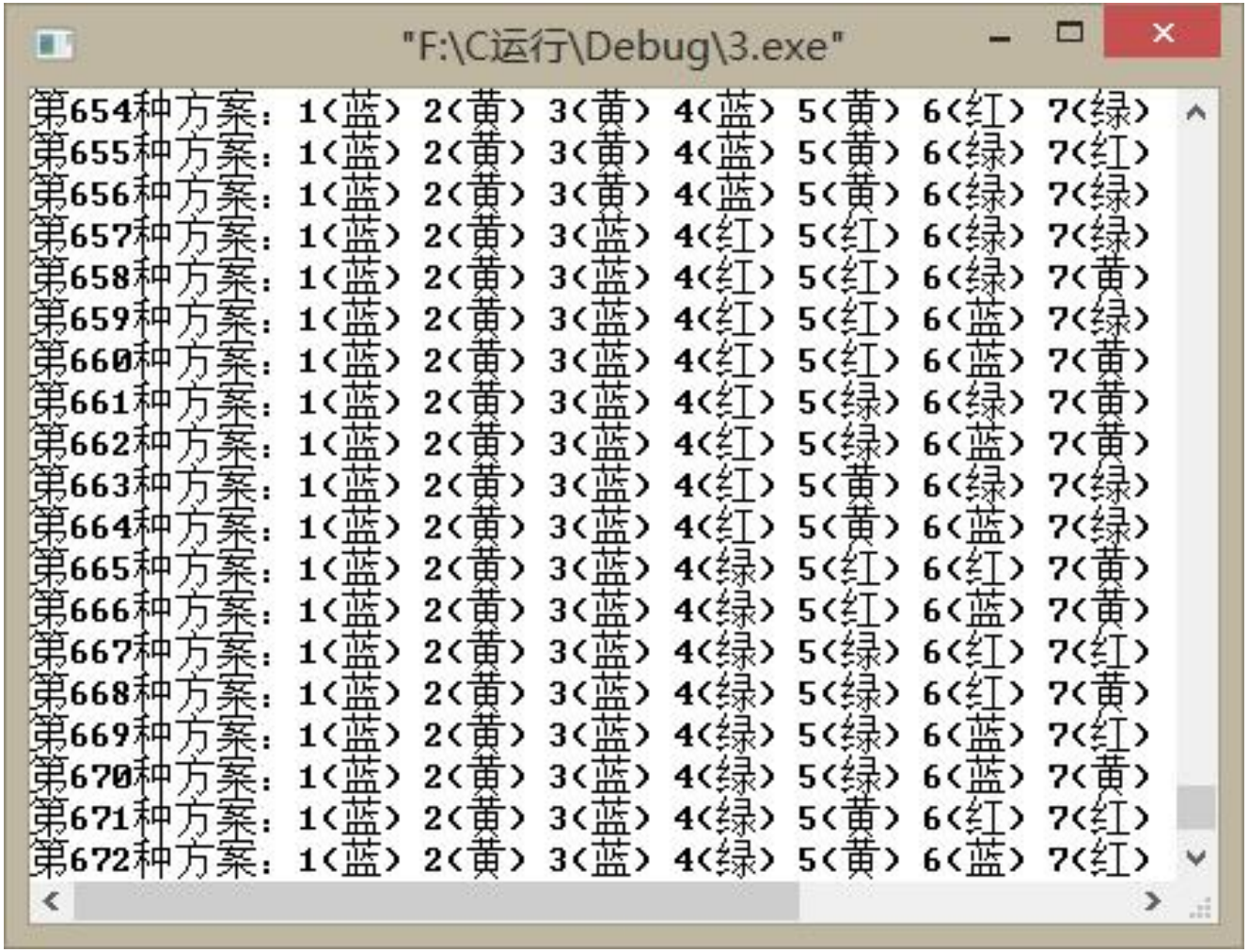
4.3 图的搜索算法

- 问题分析:
  - 为了能识别不同区域间的相邻关 系，把七巧板上每一个区域看成一个顶点，若两个区域相邻，则相应的顶点间用一条边相连，则该问题就转化为图的搜索问题。

4.3 图的搜索算法

- 算法设计：
  - 按顺序分别对 1 号、 2 号、.....、 7 号区域进行试探性涂色，用 1 、 2 、 3 、 4 号代表 4 种颜色。
  - 涂色过程如下：
    - 对某一区域涂上与其相邻区域不同的颜色。
    - 若使用 4 种颜色进行涂色均不能满足要求，则回溯一步，更改前一区域的颜色。
    - 转步骤 1 继续涂色，直到所有区域全部涂色为止，输出结果。

4.3 图的搜索算法


"F:\C运行\Debug\3.exe"
第654种方案: 1(蓝) 2(黄) 3(黄) 4(蓝) 5(黄) 6(红) 7(绿)
第655种方案: 1(蓝) 2(黄) 3(黄) 4(蓝) 5(黄) 6(绿) 7(红)
第656种方案: 1(蓝) 2(黄) 3(黄) 4(蓝) 5(黄) 6(绿) 7(绿)
第657种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(红) 6(绿) 7(绿)
第658种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(红) 6(绿) 7(黄)
第659种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(红) 6(蓝) 7(绿)
第660种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(红) 6(蓝) 7(黄)
第661种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(绿) 6(绿) 7(黄)
第662种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(绿) 6(蓝) 7(黄)
第663种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(黄) 6(绿) 7(绿)
第664种方案: 1(蓝) 2(黄) 3(蓝) 4(红) 5(黄) 6(蓝) 7(绿)
第665种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(红) 6(红) 7(黄)
第666种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(红) 6(蓝) 7(黄)
第667种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(绿) 6(红) 7(红)
第668种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(绿) 6(红) 7(黄)
第669种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(绿) 6(蓝) 7(红)
第670种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(绿) 6(蓝) 7(黄)
第671种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(黄) 6(红) 7(红)
第672种方案: 1(蓝) 2(黄) 3(蓝) 4(绿) 5(黄) 6(蓝) 7(红)

4.3 图的搜索算法

- 算法框架：

```
int a[8][8]={……};  //定义邻接矩阵
int color[8];  //存放每种方案中七个块颜色的数组

output()  //输出着色方案
{
        for(……)    //依次输出七个块的颜色
        {
                ……1对应输出红色
                ……2对应输出蓝色
                ……3对应输出黄色
                ……4对应输出绿色
        }
}
```

4.3 图的搜索算法

```
try(……)
{
        if (……)  //当第7块着色完成
                output();     //输出着色方案
        else
                for(……)  //对当前块（第i块）分别尝试4种颜色
                {
                      ……  //给当前块（第i块）着色
                      for(……)    //判断前i-1块中与第i块相邻的块是否有同色
                         ……//如果有同色则重新用下一种颜色尝试当前块（第i块）
                      ……  //如果当前块（第i块）跟任何一块相邻的都没有同色
                           try(……);  //则递归地给下一个块（第i+1块）着色
                }
}
main()
{
        try(……);  //从1块开始着色
}
```

4.3 图的搜索算法

■算法实现：

```
int
a[8][8]={{0},{0,0,1,0,0,1,0,1},{0,1,0,0,1,0,1,0},{0,0,0,0,1,0,0,1},{0,0,1,
1,0,0,1,1},{0,1,0,0,0,0,0,1},{0,0,1,0,1,0,0,0},{0,1,0,1,1,1,0,0}};
int color[8]={0},total=1;
output()
{       int i;
        printf("第%d种方案：",total);
        for(i=1;i<=7;i++)
        {       if(color[i]==1)
                        printf("%d(红) ",i);
                else    if(color[i]==2)
                                printf("%d(绿) ",i);
                        else    if(color[i]==3)
                                        printf("%d(黄) ",i);
                                else
                                        printf("%d(蓝) ",i);
        }
        printf("\n");
        total=total+1;
}
```

4.3 图的搜索算法

```
try(int s)
{
        int i,j,flag;
        if (s>7)  //当7块着色完成
                output();
        else
                for(i=1;i<=4;i++)  //对当前块分别尝试4种颜色
                {
                        color[s]=i;  //给当前块着色
                        flag=0;
                for(j=1;j<=s-1;j++)   //判断前s-1块中与s块相邻的块是否有同色
                                if(a[s][j]==1&&color[j]==color[s])
                                        flag=1;  //有同色
                        if(flag==0)
                                try(s+1);  //给下一个块着色

                }
}
main()
{       try(1);  //从1块开始着色
}
```