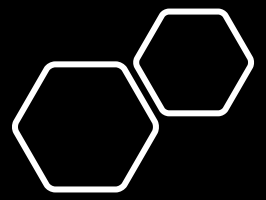


马上开始

35tang-C++竞赛系列三阶课程

三五堂

《35tang-C++竞赛系列三阶课程》

回顾这道题

给定n（不超过100）个不同数字如何选出若干个（1到n）数字，使得他们的和是m。列出所有的可能。每种可能一行输出，数字之间用空格分开。输出数字按照从小到大顺序排列。题目保证所有输入数据的和不超过10的8次方。

示例输入

5 7

5 2 3 4 1

示例输出:

1 2 4

2 5

3 4

示例说明：这个例子中要求选择的和是7，显然有3种可能，都列出来。

一起来模拟一下dfs搜索的过程

- 5 2 3 4 1
- 由于输出排序，所以输入数据先排序1 2 3 4 5
- dfs每一次选择一个数，然后往下走。
- 每一次从当前数开始选择后面的每一个数字进行尝试。

要点

- DFS函数要处理的当前状态就是已经选择哪些数字，一旦和为m就输出。相邻节点就是下一步，也就是可以选择上一次选择之后的任意一个数。所以：两个参数来传递状态，一个是当前枚举到了输入数组的第几个数字，一个是当前路径的和，另外，已经选择的数放在输出数组，可以是参数也可以是全局变量。
- 数组如果用静态数组，需要维护一个当前数组长度，由于每一个数字可能被不同的路径选择，所以需要回溯这个长度。也可以用动态数组直接回溯数组最后一个元素，就不需要保存长度的参数了。
- 为了保证输出按照从小到大的顺序，先把数组排序。

DFS模板函数
{
处理当前节点
进入下一个合法节点：
递归调用相邻合法节点
}

迷宫DFS函数
{
当前行，列位置是否终点
选择下一个位置：递归调用（当前位置的四个没有走过的相邻位置）
}

DFS模板函数
{
处理当前节点
进入下一个合法节点：
递归调用相邻合法节点
}

选数DFS函数
{
当前输出数组是否选够了（看他们的和）
选择下一个数：递归调用（下一个数分别选择当前已经选的数的后面的任意一个可选的数）
}

vector

- Vector的push_back() 和pop_up()方法特别适合做回溯，相比较静态数组，我们不需要回溯数组的长度

写程序

```
vector <int> outputnum;
void dfs(int i, int sum)
//i表示源数组处理到哪里了，sum表示当前选择的和
{
    if(sum>m) return; //剪枝
    if (sum==m)
    {
        //for (int l=1;l<=outlen;l++)
        for (int l=0;l<outputnum.size();l++)
            cout<<outputnum[l]<<" ";
        cout<<endl;
        return;
    }
    for (int l=i+1;l<n;l++)
    {
        //outputnum[++outlen]=inputnum[l];
        outputnum.push_back(inputnum[l]);
        dfs(l,sum+inputnum[l]);
        outputnum.pop_back();
    }
    return;
}
```

现在题目做个修改：输入的数据有可能重复。

- 例如：输入

5 5

1 2 2 2 5

- 这个时侯应该输出

1 2 2

5

- 如果按照前面的程序，就会输出

1 2 2

1 2 2

1 2 2

5

为什么？

- 1 2 2 2 5

- 还是把输入数组先排序。
- 我们第一个数字选1，第二个数字可以选第1个2，也可以是第2个2，这就重复了。怎么办？选数的时侯能不能只选重复的几个里面的最后一个或者第一个？

解决办法：如何让重复的数只选一次？

选数的时侯加个判断，只有前面没有重复的才选，这样就保证只会选择第三个2了。前面两个2都不会选。

在选择下一个数字的时侯前面，加一句

```
for (int l=i+1;l<n;l++)
{
    if(l>i+1 && inputnum[l]==inputnum[l-1]) continue;
    //上面这句去重复的
    outputnum.push_back(inputnum[l]);
    dfs(l,sum+inputnum[l]);
    outputnum.pop_back();
}
```

字符串搜索

给出一个m行n列的字母组成的矩阵，判断一个给定的单词在这个矩阵里面是否存在。

单词可以由矩阵的相邻元素构成，每一个相邻元素是指上下左右四个位置，每一个位置只能用一次。

1<=m,n<=200

给定单词的长度不超过1000

示例输入

3 4

ABCE

SFCS

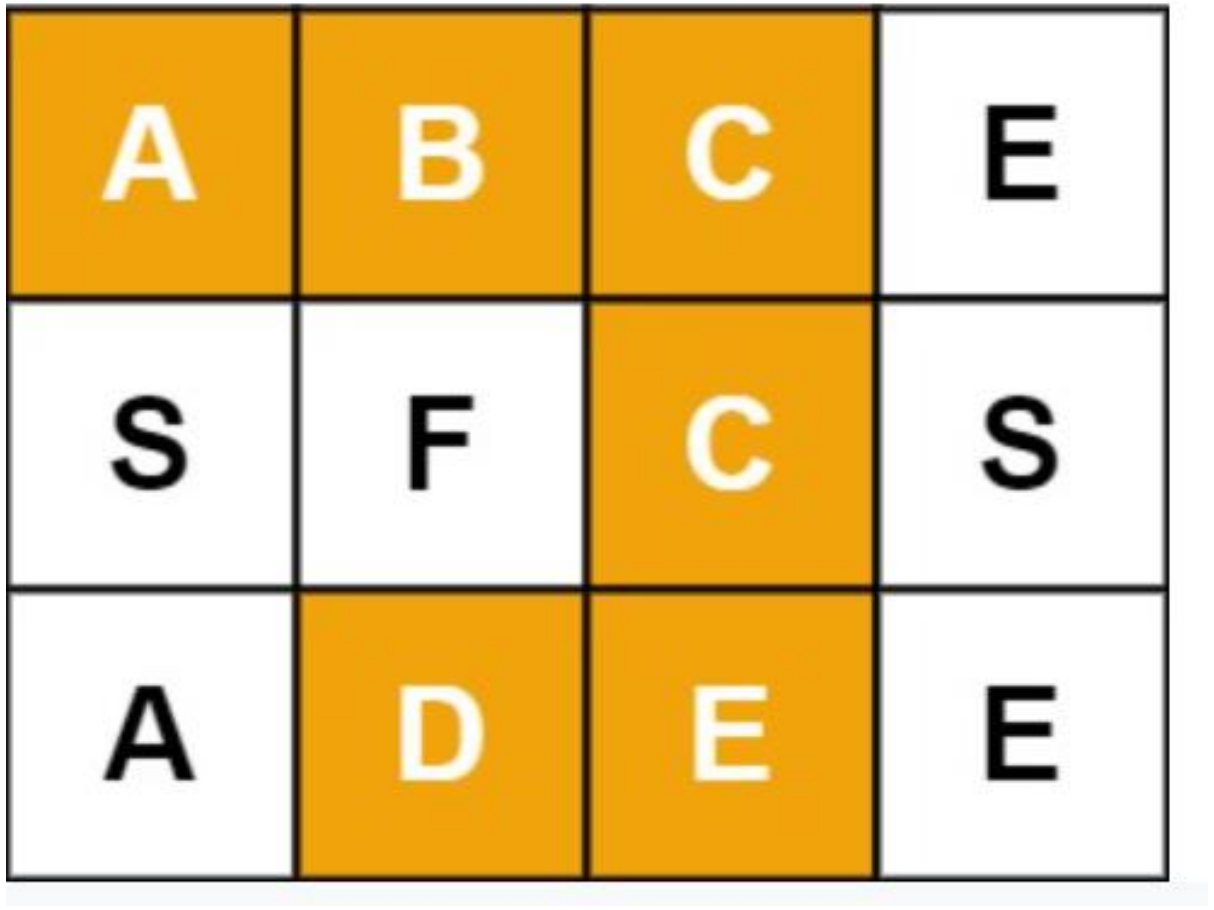
ADEE

ABCCED

示例输出

true

示例分析



DFS模板函数
{
处理当前节点
进入下一个合法节点：
递归调用相邻合法节点
}

迷宫DFS函数
{
当前行，列位置是否终点
选择下一个位置：递归调用（当前位置的四个没有走过的相邻位置）
}

本题的DFS函数
{
当前行，列位置的字母加到搜索到的字符串并判断是否完成匹配

选择下一个位置：递归调用（当前位置的四个没有走过的相邻位置）
}

算法

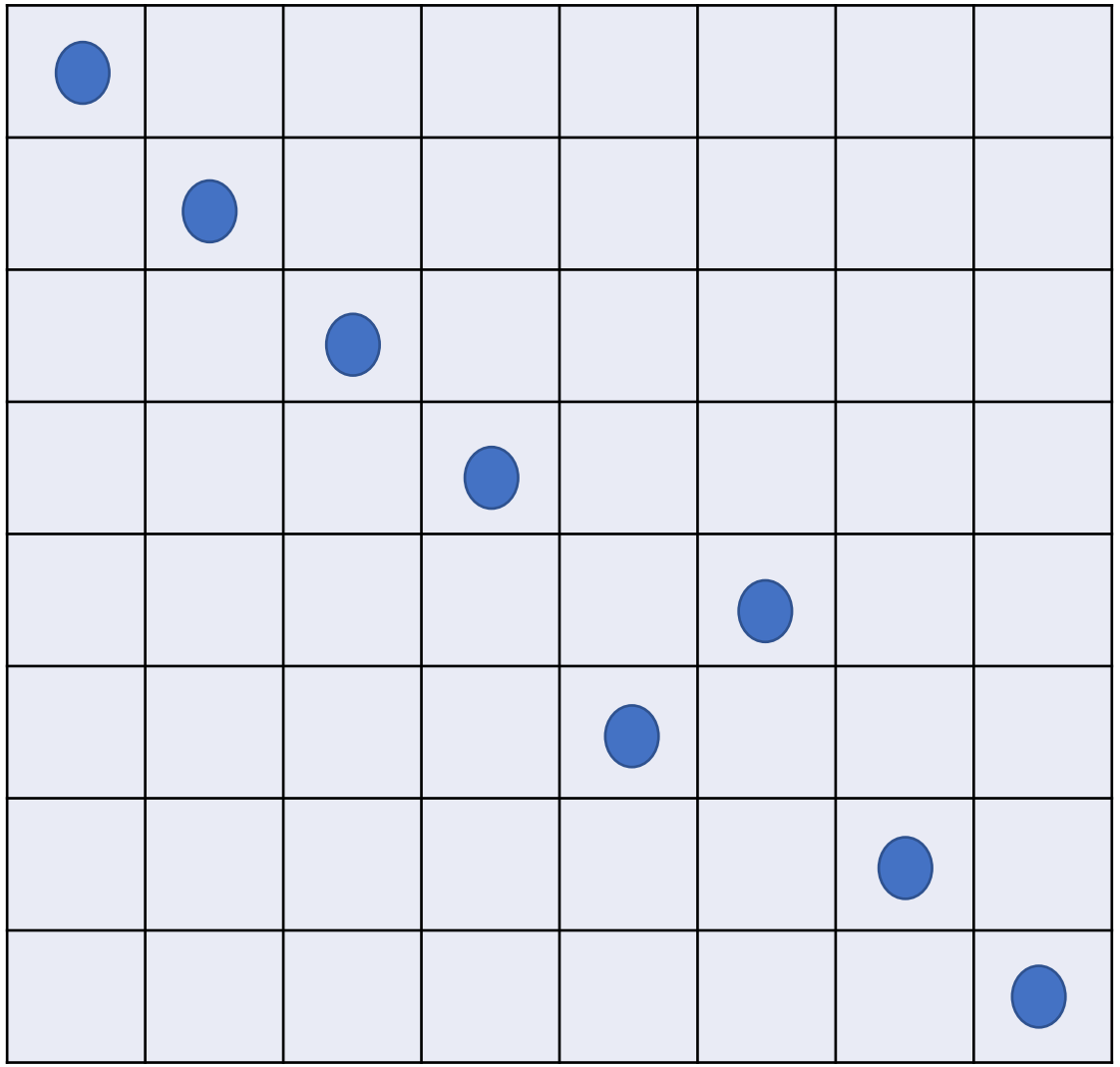
- 肯定是类似迷宫回溯搜索（手工回溯visited数组，因为一个位置可能被多个路线访问），相当于找不同路线，哪个路线经过的字符可以组成这个单词。如果从一个位置开始，这个位置是单词的第一个字符，然后找相邻元素，如果相邻的字符==单词的第2个字符，那么这个就是合法的，继续找单词的第3个元素。
- 区别迷宫几个细节：

1. 开始位置不一定是左上角，有可能是任意位置，或者限定一步，是任意一个==word第一个字符的位置。
2. 结束位置不是右下角，而是当前这条路线长度到达了word的长度
3. 中间寻找合法的相邻元素，不是说1还是0，而是看这个相邻元素是否等于单词当前位置的字符
4. 重复元素不能访问，可以用visited数组，这里也可以用一个矩阵经常用的方法，就用读入的字符二维数组，访问过了设置为特殊字符，比如‘’

一起写

经典回溯：n皇后问题

按照国际象棋规则，皇后可以攻击与之处在同一行，或者同一列，或者同一斜线上的其它棋子。现在有n个皇后放置在n*n格的棋盘上，如何摆放n个皇后而使它们都不能互相吃子？有多少种摆法？



第一点需要明确的n*n的棋盘放n个，一定是每行1个。所以我们其实是在找每一行可能放在哪一列。

如果是8个，可以用八重循环，去枚举尝试每一行的8列，每一重循环判断当前行放在哪一列的时侯肯定要看前面的行都放在哪一列了，不能互相冲突。

超过8个呢？ 用DFS来枚举状态。DFS每次尝试当前行放在什么位置然后递归去尝试下一行。

注意：在DFS过程种，当我们到一行的时侯必须知道前面每一行放的列，这样才能判断是否冲突。左图选择的就不行。

- 上节课作业全排列：1到n选择各种排列？一共选择n个数，每次都是选择1到n里面没有选择过的数。
- N皇后（我们用二维数组表示矩阵，从0行0列开始放）：0到n-1行，每行选择一个列。实际上也是选择n个数，每次选择0到n-1内的一个数。重复的肯定不能选，但是还要注意的是，冲突的也不能选。比如：第一个数选择0，表示第0行的皇后在第0列，第2行按照全排列，可以选择1，表示第1行的皇后在第1列，但是在N皇后这里，第1行不能选择1，因为第0行第0列和第1行第1列是在一个斜线上，冲突了。
- 所以和全排列最大的区别：全排列每一次0到n-1里面哪些可选只要直接判断visited状态就可以了。但是这里，我们要判断的是“是否冲突”。

DFS模板函数
{
处理当前节点
进入下一个合法节点：
递归调用相邻合法节点
}

全排列举选数DFS函数（每次选择一个数）
{
当前输出数组是否选够了
选择下一个数：递归调用（所有没有选择过的数）
}

DFS模板函数
{
处理当前节点
进入下一个合法节点：
递归调用相邻合法节点
}

N皇后DFS函数（每次选择一列）
{
当前是不是n行都放完了，或者说选择了n个列
选择下一行：递归调用（所有不冲突的列）
}

算法

```
void findquene(int index)
//index表示现在尝试第几行的皇后，也就当前处理第几个皇后的位置
   if (index==n)找完所有行，输出
   else {
      尝试当前行放在每一列（也就是n个可能的相邻节点），去和前面的比较，没有重复，当前行的皇后就 放在这一列（合法的相邻节点）然后往下走，也就是下一行
   }
  return;
}
```

细化

- 首先每一行只有一个，所以只需要记录每一行放在第几列，用一个数组col[]表示输出结果就够了（同时col数组还记录了当前已经选好的行的状态，用来判断下一行怎么放）。这里我们用col[]数组而不是visited数组，因为我们不能直接通过visited[i]的状态来判断i是不是可选，每一次判断i是不是可选，需要根据前面每一行选择了什么来判断，这样才能判断出斜线是否冲突。
- 搜索的状态就有了：当前节点就是当前在处理哪一行，而相邻节点就只有一个，就是去处理下一行。当然，处理的过程中为了判断冲突，还需要传递当前路线已经选择的行是放在哪些列的也就是col数组（col数组可以作为全局变量也可以作为参数传递）。
- 每放一个皇后，我们需要判断这个皇后是否和前面行已经放好的皇后冲突，冲突就说明这条路走不通，不需要再往下。
- 判断列冲突很容易，看看前面每一行的列（col[i]就是第i行皇后放的列）有没有和当前列冲突的（同一列，两个斜线），有就不能放了。

	列0	列1	列2	列3	列4	列5	列6	列7
行0	0，0							0，7
行1		1，1					1，6	
行2			2，2			2，5		
行3				3，3	3，4			
行4				4，3	4，4			
行5			5，2			5，5		
行6		6，1					6，6	
行7	7，0							7，7

关于对角线，也可以总结规律；

- 黄色对角线：

行-列=0；

- 绿色对角线：

行+列=n-1

	列0	列1	列2	列3	列4	列5	列6	列7
行0							0，6	
行1	1，0					1，5		
行2		2，1			2，4			
行3			3，2	3，3				
行4			4，2	4，3				
行5		5，1			5，4			
行6	6，0					6，5		
行7							7，6	

关于对角线，也可以总结规律；

- 黄色对角线：

行-列=0；

- 绿色对角线：

行+列=n-1

扩展到普通的斜线？

关于对角线，也可以总结规律；

- 一条黄色左斜线的所有位置：

行-列是一样的；

- 一条绿色左斜线的所有位置：

行+列是一样的

```
bool checkpos(int currow,int
curcol)
{
for (int i=0;i<currow;i++)
{
      if (col[i]==curcol)
            return false;//同列
      if (curcol-currow==col[i]-i)
            return false;//左斜线
      if (curcol+currow==col[i]+i)
            return false;//右斜线
}
return true;
}

//这里的逻辑就是去检查前面每一行放好
的皇后和新的准备放在(currow,curcol)
这个位置的皇后是否冲突。
//注意col[i]就是第i行的皇后放的列
```

```
void findquene(int index)//index表示现在尝试第几行的皇后
{
   if (index==n) {//找完所有行，输出
            countnum++;
            for (int i=0;i<n;i++)        cout<<"["<<i<<","<<col[i]<<"]\t";
            cout<<endl;
            }
    else{//每一列去和前面的比较，没有重复，就放在这一列找下面的
              for (int j=0;j<n;j++)     {
                        if (checkpos(index,j)) {//检查j是不是和前面冲突，用函数
                                  col[index]=j;
                                  findquene(index+1);
              } } }
}
```

index为什么不回溯，比如这里的index，调用下一次传递index+1给下一次，所以下一次函数结束了index还是原来的值，而不是index+1。

如果这个时侯修改为：index=index+1;findquene(index);

那么，在findquene(index);后面就要加上：index--;

思考

- 这里每一次都要检查前面所有的皇后，每次检查时间复杂度都是O(n).
- 能不能像前面迷宫一样，改造一下visited数组标记状态，后面检查状态的时候时间函数就是线性的了,或者说检查冲突就变成O(1)了。

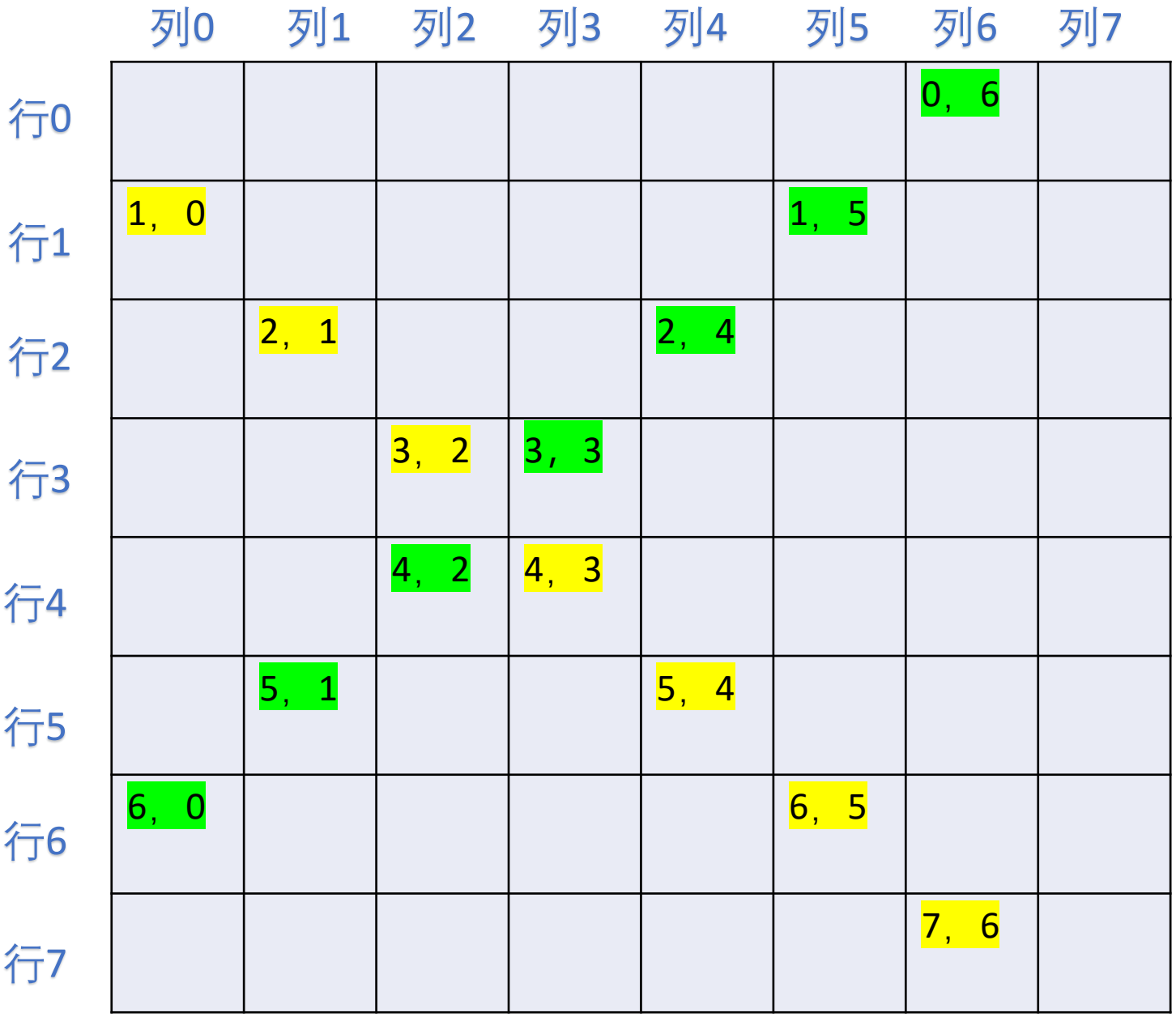
优化：加上visited标记检查冲突

bool vis[3][200];

优化算法，保存一个vis二维数组，vis[0]保存当前皇后占用的列（就是和前面的visited数组一样的意思，判断当前列是否用过），vis[1],vis[2]分别表示两个斜线，后面就不用遍历col数组去check了，而是直接查表判断，O(1)时间复杂度 。注意，vis[1]这个数组里面保存的是所有走过的“行+列”的状态，vis[2]这个数组里面保存的是所有走过的“行-列”的状态。

就像迷宫一样，但是现在visited数组不是标记一个点是否走过，而是标记一个状态是否走过，这个状态包括了列，对角线的信息

index还是不用手工回溯，但是新增加的vis数组需要手工回溯状态，因为换一个i（选择不同的列）就意味着前面已经选择的vis发生了变化



如果vis[1][6]为1，说明绿色斜线上放过一个皇后，那么如果当前选择第3行第3列或者其他所有图中绿色位置就不行了，因为vis[1][6]为1，绿色斜线不能用了。

优化：加上visited标记检查冲突

```
void findquene(int index){
//index表示当前尝试皇后在第几行
   if(index == n){
       countnum++;
      for (int i=0;i<n;i++)       fout<<"["<<i<<","<<col[i]<<"]\t";
      fout<<endl;
      return;
   }
    for(int i = 0; i < n; i++){
          if(!vis[0][i] && !vis[1][index+i] && !vis[2][index-i+n]){ //vis[0][i]表示当前列i是否出现过
         //vis[1][index+i]表示按当前行+当前列所代表的左斜线上有没有出现过。
        //vis[1][index-i]表示按当前行-当前列所代表的右斜线上有没有出现过。+n是为了防止出现负数
              col[index]=i;
              vis[0][i] = 1;  vis[1][index+i] =1; vis[2][index-i+n] = 1;
              findquene(index + 1);
              vis[0][i] = 0; vis[1][index+i] =0; vis[2][index-i+n] = 0;//手工回溯
          }
   }
   return;
}
```

作业1　选k个数selectk.cpp

给出n个数字（1=<n<=100），从里面选k个数字（1=<k<=80）请给出从大到小的所有排列组合。格式如下，第一行n和k空格隔开，第二行n个数字空格隔开：

5 2

1 2 3 4 5

输出

1 2

1 3

1 4

1 5

2 3

2 4

2 5

3 4

3 5

4 5

挑战作业2：k个数的和 ksum.cpp

给出n个数字（1=<n<=100），从里面选k个数字（1=<k<=80）请把这个k个数的可能的和按照从小到大的顺序列出来。每一个数字小于10的8次方。

输入格式：

第一行两个整数n和k，空格隔开

第二行n个数字空格隔开：

输出格式：

第一行1个整数，表示一共多少个不同的和

第二行空格分隔的若干整数，按照从小到大的顺序，表示所有的不同的和。

示例输入：

5 2

2 2 3 4 5

示例输出

6

4 5 6 7 8 9

由易到难，思维体系训练
实战结合，创新协作培养
兴趣导向，未来职业引领

https://www.35tang.com

https://www.三五堂.com

扫码关注公众号

添加辅导老师