程序运行，读入数据，把每个结点(结点的坐标与序号)存入init[n]这个结点类的数组中，同时利用邻接矩阵mat[ ][ ]存储图（若两个结点i,j相连，mat[i][j]= 1），利用邻接表e[ ]存入每个结点所邻接的结点。

功能函数 solve()

为了遍历到所有结点,函数运行在两重循环下

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

for(j = 0; j < n ; j++)

{

}

**S3:** (1) 以point[1]结点作为起点,进入到一个while(n - 2)的循环

(2)point[1]与point[0]组成向量**a** = (point[1],point[0])

(a)在 **a**向量顺时针旋转180范围内，检索有无point[1]相邻接的结点node，如果存在，通过分别计算比较向量**a**与向量(point[1],node)的夹角，选择夹角最小的结点。如果在顺时针180度的范围内没有检索到结点,进入步骤 (b),否则执行下一步

(b) 在 **a**向量逆时针旋转180范围内，检索有无point[1]相邻接的结点node，如果存在，通过分别计算比较向量**a**与向量(point[1],node)的夹角，选择夹角最小的结点。如果在逆时针180度的范围内没有检索到结点,进入步骤 (c),否则执行下一步

(c)因为在顺时针与逆时针180度的范围内都没有检索到结点，所以若point[1]除了point[0]还有邻接点的话，肯定是向量**a**与向量(point[1], node)夹角为180度

功能函数 solve()

在两重循环下

**S1:** 建立一个point[]结点类数组,用来接下来存储寻找到的符合条件的结点。

**S2:** 每次循环先把init[i]与init[j]这两个结点存入到point[0]与point[1]这两个位置上

**S5**:

两重循环执行完毕后，所有的结点都被遍历，符合条件的环被找出

**S4:** 通过**S3 ，**得到一个结点node，把node纳入到路径中,即存入到point[ ]数组中。

**a** 在返回去执行**S3**之前，需要进行判断

1. node与 point[] 数组的第一个结点是否相同，若相同，说明point[ ]数组中的结点构成了一个单环(对于简单情况)
2. 在满足(1)情况下, 利用check( )函数，判断point[ ]形成的环中是否有嵌套情况，如果没有，则该环为真正的单环，存入map中,否则舍弃。
3. (1),(2)执行完后，返回**S2**执行，重新创建一个point[]数组，探索以新的结点i 与 j作为初始路径的环

**b** 如果**a** 中的条件(1)不满足，说明当前的point[ ]中的结点尚不能构成一个环，那么返回到**S3**循环执行，并且以node结点作为新的起始点，检测与node相邻接的点

时间复杂度分析：两重循环时间复杂度为n2，在两重循环内部有一层while(n - 2)的循环，时间复杂度为最坏情况下为n,剩下的为在循环中的指令的执行，时间复杂度都是小于n的，所以总体为n2 + n, 即O(n2)。