0323

1. G = (V, E) 是一个无向图，每个顶点的度数都为偶数。设计线性时间算法，给 G 中每条边一个方向，使每个顶点的入度等于出度。（请先简单说明算法思想，再给出伪代码，然后证明其时间复杂性符合要求）

因为无向图每个顶点的度数都为偶数，所以该图的每个连通分支都是欧拉图，即一定存在欧拉回路能经过每一条边且每条边仅经过一次。应用**DFS**，沿欧拉回路标记每一条边的方向，即可保证每个顶点的入度等于出度。所以本题的本质即为求出欧拉回路，这也可以通过**DFS**实现。其思想为，随机选取一个节点进行深度优先遍历直到没有新的节点可以被遍历到(一定形成了一个环)，此时找到路径中第一个还有未被访问的边的顶点，以它为起点执行另外一次深度优先遍历，这会得到另一个环，将这个新环拼接到原来的路径中。之后重复上述操作直到所有边都被遍历到即可。

```
//核心代码：
vector<int> dfs_euler_circuit(vector<vector<int>>& graph, int start_node)
{
    stack<pair<int,int>> output; //存储结果
    stack<int> stk;    // 初始化栈，将起始节点压入栈中
    stk.push(start_node);
    while (!stk.empty()) {
        int curr_node = stk.top();    // 获取栈顶元素
        if (!graph[curr_node].empty()) {
            // 如果当前节点还有未访问的邻居节点，随机选择一个进行DFS遍历
            int next_node = graph[curr_node].back();    // 从当前节点的邻居节点中随机选择一个
            graph[curr_node].pop_back();    // 在邻居节点中删除当前节点
            graph[next_node].erase(find(graph[next_node].begin(), graph[next_node].end(), curr_node));
            // 删除另一个节点中与当前节点相邻的边
            stk.push(next_node);    // 将选中的邻居节点压入栈中
            ouput.push((curr_node, next_node));  //有向边由curr_node指向next_node
```

```
        } else {
            // 如果当前节点没有未访问的邻居节点，则回溯到上一个节点
            stk.pop();    // 弹出栈顶元素,回溯
        }
    }
    return output;
}
```

以上代码实现的是在一个连通分支中找到欧拉回路，将上述代码在所有连通分支中实现，就对整个图G完成了标记(当**stk**内元素全部弹出仍有点未被边历到，则说明图不连通，从未被遍历到的点开始**DFS**即可)。我们使用邻接表而非邻接矩阵来存储图中的边，使得遍历更加高效，在整个过程中我们每遍历到一条边，就会将这条边加入**outout**并在原图中删除这条边，保证了不会重复遍历，因此总的时间复杂度是$O(|E|)$的(此处假设边数比点数多，否则写成$O(|V|+|E|)$不影响结论)，因此时间复杂度是线性的，符合要求。

2. 连连看游戏中用户可以把两个相同的图用线连到一起，如果连线拐的弯小于等于两个则表示可以消去。设计算法，判断指定的两个图形能否消去。

若能将所有与图片A经过不多于2个转角的路径相连的图片找出来，加入一个集合S中。那么判断B与A能否相连只需判断B是否存在于集合S中即可。采用广度优先搜索算法可以方便的实现这一构想。算法的思路如下：

1. 定义空集S与T，将A加入集合S
2. 找出所有与A能直接相连的点，将其加入集合S
3. 找出与集合S中的点能直接相连的点，加入集合T，然后将T中所有元素加入到集合S中，清空集合T
4. 找出与集合S中的点能直接相连的点，加入集合T，然后将T中所有元素加入到集合S中
5. 若B在集合S中，则A，B可以相连。否则A,B不能相连

实际上就是进行三次广度优先搜索，满足拐角数小于等于2

以下是伪代码实现

```
Input: G(V,E)  //直接可通过直线连接的点在图中有边
Output: S
void BFS(S)
```

```
{
    unmark S;
    for v in S:
        mark v;
        for (v,w) in E:
            S.insert(w);    //这里简化操作，直接将符合条件的放入集合S，不再生成一个集合T(重复的insert是无效的)
            mark w;         //防止以w为起点进行第二次遍历
}
S={}
S.insert(A)
BFS(S)  //与A直接相连的
BFS(S)  //拐角不超过1的
BFS(S)  //拐角不超过2的
if B in S:
    return true;
return false;
```

3. 证明任意连通无向图中必然存在一个点，删除该点不影响图的连通性。用线性时间找到这个点

以图中任意一节点为根节点做深度优先搜索，由于图是连通图，一定存在搜索到某个节点时，该节点的所有相邻节点都已经被标记了(有限图，一定存在这样的节点，不然DFS无法终止)。即该节点的所有相邻节点都可以从根节点通过不经过该节点的路径到达。 所以删除该节点一定不影响图的连通性。

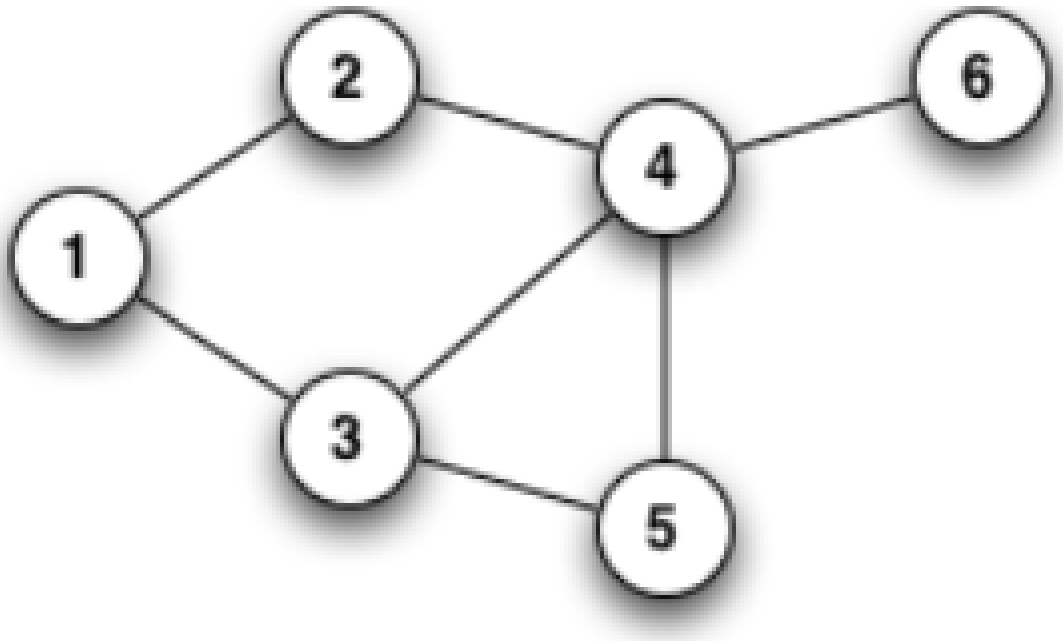
进行深度优先搜索的时间复杂度为 $O(|V|+|E|)$。因为该算法不需要完成对整个图的遍历，所以该算法的时间复杂度不超过 $O(|V|+|E|)$

以下是伪代码实现:

```
Input: G(V,E)
Output: v
v = random(V)   //随机选取一个节点
con = true
while(con)
{
    con = false;    //如果所有邻居节点都被遍历到，con将维持false
    mark v;
```

```
    for (v,w) in E:
        if (w is unmarked):
            mark w;
            v = w;
            con = true;
}
return v;
```

4. 给定一个连通的无向图和图中的一个顶点v，构造一个有向图，使得有向图中的任何路径都通向该特定顶点。考虑下面的连通无向图。设输入顶点为 1。



思路如下：

从目标节点v开始，进行广度优先搜索，找到与v相连的所有节点s，将s与v之间的边标记为指向v，之后删除v与这些边，将S中的节点作为新的目标节点重复操作，由于图是连通的，经过有限次操作一定能删除完所有的节点与边，此时图便被标记成了一张有向图，且从任意节点开始顺着路径都会通向最开始的目标节点。

伪代码实现如下：

```
Input: G(V,E), target;
Output: E1      //有向边集
void BFS(G,v)
{
    S = {}
    for (v,w) in E:
        E1.push((w,v));       //由w指向v
        E.delete((w,v));
        S.push(w);
    V.delete(v);
    for s in S:
        BFS(G,s);
}

G_copy = G    //保留原图
BFS(G_copy, target);
return E1
```

以上图为例，输入节点为1