Algorithm HW2

1.

（1）无向图：若每个顶点的度 ≥ 2，则图中一定含有圈

设无向图 $G = (V, E)$ 为有限图。任取一个顶点 v_0，从它出发构造一条**不重复顶点的简单路径**：

$$v_0, v_1, v_2, \ldots$$

每一步都从当前顶点选择一个尚未访问过的相邻顶点继续前进。由于图是有限的，该过程不可能无限进行，因此在某一步 v_k 处，所有相邻顶点都已经在路径中出现过。

注意：

- v_k 的度 $\deg(v_k) \geq 2$；
- 路径中与 v_k 相邻的顶点至少有一个不是前驱 v_{k-1}，否则度至多为 1，矛盾。

因此，v_k 必然与路径中某个更早的顶点 v_i $(i < k-1)$ 相邻，于是

$$v_i \to v_{i+1} \to \cdots \to v_k \to v_i$$

构成一个圈。

（2）有向图：若每个顶点的出度 ≥ 1，则图中一定含有有向圈

设有向图 $D = (V, A)$，对任意顶点 v_0，构造一条**沿有向边前进的路径**：

$$v_0 \to v_1 \to v_2 \to \cdots$$

由于每个顶点出度 ≥ 1，该路径始终可以继续延伸。

因为图是有限的，顶点数有限，根据抽屉原理，路径中必然出现重复顶点：存在 $i < j$，使得

$$v_i = v_j$$

于是路径

$$v_i \to v_{i+1} \to \cdots \to v_{j-1} \to v_j (= v_i)$$

构成一个有向圈。

2.

先看**有向图 D**。设存在一条有向 Euler 环游，它从某顶点出发，沿有向边行走，每条有向边恰好经过一次，且最终回到起点。对任意顶点 v 考察：每进入 v 一次，必然也要从 v 离开一次。因此：

$$\deg^+(v) = \deg^-(v)$$

从任意顶点出发，沿尚未使用的有向边前进。由于每个顶点满足

$$\deg^+(v) = \deg^-(v),$$

在未用完所有边之前，不可能在非起点处“卡死”。因此必然回到起点，形成一个有向闭游走。若该闭游走未覆盖全部边，则利用图的连通性，从已有回路上的某个顶点引出尚未使用的边，构造新的闭游走，并将其“拼接”进原回路。反复进行，最终得到覆盖全部边的有向 Euler 环游。

接下来是**无向图 G**。G 含 Euler 环游当且仅当 G 连通且每个顶点的度都是偶数。

3. 设无向图 $G = (V, E)$，其中 $|V| = n$，$|E| = m = n - 1$，且 G 连通。假设 G 不是树。由于 G 连通而不是树，则 G 必含有一个圈。设该圈包含 $k \geq 3$ 条边。去掉该圈中的任意一条边 e，得到新的连通图 $G' = (V, E \setminus e)$。有 $|E(G')| = (n-1) - 1 = n - 2$。但 G' 连通且顶点数为 n，这与“连通无向图至少需要 $n - 1$ 条边”矛盾。矛盾说明假设错误，因此 G 不含圈。G 连通且无圈，所以 G 是树。

4. 设用一个 $n \times n$ 的邻接矩阵 A 描述有向图，其中 $A[i][j] = 1$ 表示存在有向边 $i \to j$，否则为 0。

(1) 顶点 i 的出度等于第 i 行中 1 的个数。顺序扫描该行即可：

$$\mathrm{outdeg}(i) = \sum_{j=1}^{n} A[i][j]$$

(2) 对整个邻接矩阵逐行逐列扫描，统计所有为 1 的元素：

$$m = \sum_{i=1}^{n} \sum_{j=1}^{n} A[i][j]$$

(3) 若要删除有向边 $i \to j$，只需令

$$A[i][j] = 0$$

时间复杂度为 $\Theta(1)$。

5. 邻接表为

A	B, E
B	A, C
C	B, D, E
D	C
E	A, C, F, G
F	E, G
G	E, F

DFNL 执行过程

1. 访问 A，DFNL(A)=1 → 第一个未访问邻接点 B
2. 访问 B，DFNL(B)=2 → 第一个未访问邻接点 C
3. 访问 C，DFNL(C)=3 → 第一个未访问邻接点 D
4. 访问 D，DFNL(D)=4 → 无未访问邻接点，回溯到 C
5. C 的下一个未访问邻接点是 E
6. 访问 E，DFNL(E)=5 → 第一个未访问邻接点 F
7. 访问 F，DFNL(F)=6 → 第一个未访问邻接点 G
8. 访问 G，DFNL(G)=7 → 无未访问邻接点，回溯

回溯过程中其余邻接点均已访问，算法结束。

6.

(1) 假设图用**邻接表**表示，顶点编号为 `0..n-1`。

```
void DSearch(int v, const vector<vector<int>>& adj, vector<
bool>& visited) {
    stack<int> S;
    visited[v] = true;
    S.push(v);

    while (!S.empty()) {
```

```
        int u = S.top();
        S.pop();

        for (int w : adj[u]) {
            if (!visited[w]) {
                visited[w] = true;
                S.push(w);
            }
        }
    }
}
```

(2) 设顶点 u 从起点 v 可达，则存在路径：

$$v = v_0, v_1, \ldots, v_k = u$$

用归纳法证明路径上的顶点都会被访问。

- **基础情形**：$v_0 = v$ 在算法开始时被标记并入栈，必被访问。
- **归纳假设**：假设 v_i 已被访问并曾入栈。
- **归纳步骤**：当 v_i 被弹出栈时，算法遍历其邻接点。因为 (v_i, v_{i+1}) 是一条边且 v_{i+1} 尚未访问，所以 v_{i+1} 会被标记为已访问并压入栈。

由归纳法，路径终点 $u = v_k$ 必被访问。

(3) 设图有 n 个顶点，m 条边。

- **时间复杂度：**每个顶点最多入栈、出栈一次；每条边最多被检查常数次：$\Theta(n + m)$。
- **空间复杂度：**栈最多存放 n 个顶点，加上访问数组：$\Theta(n)$。