5. 图的遍历

Problem 5.1

伪代码：

```
初始化栈
对图中每个节点 v 进行如下操作：
    如果节点 v 未访问过，则：
        将节点 v 置为 DISCOVER 状态并入栈
        重复下列操作直到栈为空：
            取栈顶元素（不出栈）
            如果栈顶元素存在未被访问过的邻接节点 w，则：
                将节点 w 置为 DISCOVER 状态
                将节点 w 进栈
            否则，当前节点退栈，并置为 FINISH 状态
```

在具体实现时，栈中每个元素可以存储两个值，一个是当前入栈的节点，另一个是当前节点的下一个未被访问的邻接节点，这样，在访问栈顶元素 x 的下一个邻接节点时，就可以不用循环查找 x 的所有邻接节点了。

Problem 5.2

满足这样条件的边是 **Cross Edge**。在图的 DFS 算法中，每个节点 v 满足$v.\mathrm{discoverTime} < v.\mathrm{finishTime}$，因此上述的边$(u,v)$的$(u.\mathrm{discoverTime}, u.\mathrm{finishTime})$和$(v.\mathrm{discoverTime}, v.\mathrm{finishTime})$是不相交的，这是 CE 的特征。

Problem 5.3

如果一个有向图的收缩图中有环，那么在这个环上的所有点（及该点所代表的内部所有点点）都是互相可达的，因此这个环可以收缩成一个点，产生矛盾。因此一个有向图的收缩图是无环的。

Problem 5.4 ((必选))

第一次深度优先遍历不可以替换为广度优先遍历，因为第一次的深度优先遍历产生的 finishTime 决定了第二次遍历的顺序，在 SCC 算法的证明过程中，第二次遍历能够产生 SCC 的性质是由第一次深度优先遍历决定的，所以不能换成广度优先遍历。

第二次深度优先遍历可以替换为广度优先遍历，因为第二次遍历只是为了找到连通分量，此时用 DFS 还是 BFS 都可以。

Problem 5.5 ((必选))

无向连通图的深度优先搜索树的**根节点** v 是割点的充要条件是，v 在该深度优先搜索树中有两个以上的子树。

充分性" $\Leftarrow$ "：

若 v 有两个以上子树，不妨设 x，y 是 v 的某两个不同子树中的节点，无向连通图的深度优先搜索树中不存在 cross edge，因此从 x 到 y 的所有路径上都要经过根节点 v，因此 v 是割点。

必要性" $\Rightarrow$ "：

若根节点 v 是割点，则存在两个节点 x，y（不同于 v），从 x 到 y 的每一条路径上都有 v。

反证法：假设根节点 v 只有一个子树，不妨设 v 的唯一子节点是 w，则这会产生两种情况：

1. w 是上述 x，y 节点中的一个，不妨设 w 是 x，则从 w（x）到 y 存在一条不经过 v 的路径（即沿着 w 的子树向下一直到 y 的路径），与假设矛盾。
2. w 与 x，y 都不同，则从 x 到 y 存在一条不经过 v 的路径$x \ldots w \ldots y$，与假设矛盾。

因此，根节点 v 至少有两个子树。

Problem 5.6 ((必选))

算法仍然正确。

要证明该算法仍然正确，我们只要证明在遍历过程中，每个节点的 back 值仍然能正确计算。

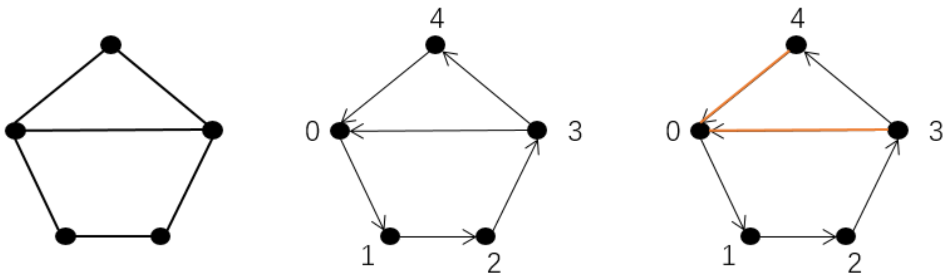
注意到在无向图的 DFS 过程中，每个节点至少有一个 back edge 是指向其父节点的，因此在访问该节点的 back edge 时，通过 `back=min(discoverTime[w], back)`可以得到 back 的值至多为 `discoverTime[w]`，这与将节点的 back 值直接初始化为 `discoverTime` 结果是一样的。

Problem 5.7

1. 若 G 中存在度为 1 的点，则删除该点后 G 仍连通；若 G 中不存在度为 1 的点，则 G 中所有的点都在某个环中，因此删除任意一点 G 都连通。
2. 环。
3. 如图：

Problem 5.8

该算法错误在于，所记录的 level 值应该是从根节点到该节点的最短路径的长度，而不是 DFS 树中的路径长度。而且，通过 DFS 不能正确求出每个节点的到根节点的最短路径。考虑如下反例：



图一 图二 图三

按图二所示的 DFS 顺序，我们在图三中的两次红色线上遇到了 back edge，并计算得到两次的环路长度分别为 5 和 4。但是图中的最小环长度为 3，因此该算法出错。

Problem 5.9

只要无向图中每个顶点都在某个环中，我就可以为这些环指定一个方向（顺着环形的方向），使得每个顶点的入度为 1。因此我们只要通过算法判断每个点是否都在某个环中即可。

应用无向图的 DFS 算法，我们发现对于 DFS 树中的任意节点 v（非叶节点），v 位于某个环中的充要条件是存在 v 的某个子树节点 w，且 w 有连接到 v 的祖先的一条 back edge。而对于 DFS 树中的叶节点 v，v 位于某个环中的充要条件是 v 存在一条连接到 v 的祖先的 back egde。

在 DFS 过程中，DFS 遍历的方向就可以作为指定的边的方向。

Problem 5.11

用邻接表来表示图，这样寻找一个点相连的所有边的时间正比于边的个数，最后一共删除了 E 条边，因此删除边的总的时间复杂度为$\mathrm{O(E)}$，我们在删除边的过程中可以顺带找到下一个需要删除的入度为 0 的顶点。删除 V 个顶点需要的时间复杂度为$\mathrm{O(V)}$。两个综合起来，总的时间复杂度为$O(V+E)$。

如果 G 中包含回路的话，算法运行时会发现，图 G 中找不到入度为 0 的顶点了，但是 G 中仍有顶点。

Problem 5.12 (顶点间的"one-to-all"可达性问题(必选))

1. 以顶点 s 为根节点进行深度优先遍历，若遍历完发现图中所有顶点都被访问过，则 s 可达所有其它顶点，否则不可达。深度优先遍历的时间复杂度为$\mathrm{O(m+n)}$，遍历完判断是否所有顶点都被访问过需要$\mathrm{O(n)}$，因此该算法的时间复杂度为$\mathrm{O(m+n)}$。
2. 对图进行一次 SCC 算法，得到一个收缩图，时间复杂度为$\mathrm{O(m+n)}$，然后判断这个收缩图中入度为 0 的顶点的数目，若存在一个入度为 0 的顶点，则该点就可达图中其它所有顶点；若存在大于一个入度为 0 的顶点，则不存在可达图中所有其他顶点的点。判断的时间复杂度最多为$\mathrm{O(n)}$，因此算法的时间复杂度为$\mathrm{O(m+n)}$。

Problem 5.13 ((必选))

对图进行一次 SCC 算法，得到一个收缩图，时间复杂度为$\mathrm{O(m+n)}$。在收缩图中，每个点都对应了$\mathrm{k_i}$个原图中的顶点。在收缩图中，每个顶点的影响力值为该点可以到达的其它顶点的$\mathrm{k_i}$值之和，即，出度为 0 的顶点的影响力值为该点的$\mathrm{k_i}$值，连接到出度为 0 顶点的点影响力值为该点的$\mathrm{k_i}$值加上出度为 0 的点的$\mathrm{k_{i+1}}$值，以此重复。因此我们通过反向遍历计算出每个点的影响力值。两小题就都解决了。

Problem 5.14

对 G 进行拓扑排序，找出 critical 路径，这条路径上的顶点数就是答案。

Problem 5.15

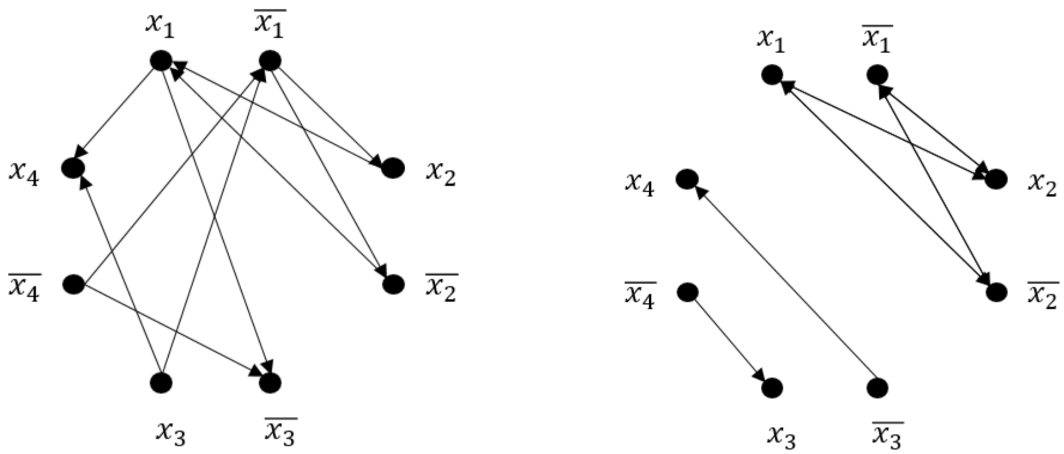
1. 对该问题建模，该市的所有十字路口为图 G 中的顶点，所有街道为连接顶点的有向边，因此求解的问题就变成，图 G 中是否每个顶点都可达所有其它顶点，也就是说是否所有顶点处于一个强连通图中，因此只要用 SCC 算法求出所有强连通片，判断强连通片的数目是否为 1 即可，时间复杂度为$\mathrm{O(V+E)}$。
3. 图的建模和上题一样，该问题可以描述为，对于图 G 中的某个顶点 v，v 可达的所有顶点都可达 v。算法可以这样设计，以 v 为起始节点进行深度优先遍历，在 DFS 树中的所有子节点都应该存在一条通向根节点的路径。在遍历过程中，若发现某个顶点有一条指向根节点的 back edge，则遍历过程中该节点的所有祖先节点都可达根节点，因此可以进行回溯更新。在算法结束后，判读一下是否所有节点都可以通向根节点。DFS 过程的时间复杂度为$\mathrm{O}(V+E)$，回溯的顶点数最多为 V 个，因此该算法的时间复杂度为$\mathrm{O(m+n)}$。

Problem 5.16 (小孩排队问题(必选))

1. 每个小孩看成图 G 中的一个顶点，若小孩 i 恨 j，则存在一条有向边从$\mathrm{v_i}$指向$\mathrm{v_j}$。对该图进行拓扑排序，若存在拓扑排序，则可以将所有小孩排成一队，排队的顺序就是逆拓扑序。若不存在拓扑排序，则解不存在。算法的时间复杂度为$\mathrm{O(m+n)}$。
2. 建模过程同上一小题，先进行拓扑排序，若不存在拓扑排序则解不存在，否则找出 critical 路径，这条路径上的顶点数就是答案。

Problem 5.17 (2-SAT 问题)

1. TRUE，TRUE，FLASE，TRUE
2. $(\mathrm{x_1} \vee \mathrm{x_2}) \wedge (x_1 \vee \overline{\mathrm{x_2}}) \wedge (\overline{x_1} \vee \mathrm{x_2}) \wedge (\overline{x_1} \vee \overline{\mathrm{x_2}}) \wedge (x_3 \vee x_4)$
3. 如下图：



4. 若图$\mathrm{G_1}$中有一个同时包含x和$\bar{\mathrm{x}}$的强连通片，意味着存在一条$\mathrm{x} \to \cdots \to \bar{x} \to \cdots \to x$的回路，这产生了一堆矛盾的蕴含关系$\mathrm{x} \to \bar{x}$和$\bar{x} \to x$，因此不存在可以满足的赋值方法。

~~5. TODO~~

Problem 5.18

做个不严谨的证明：

1. 假设无向图的 BFS 树中存在 BE，则在 BFS 过程中，该节点一定会在它 BE 指向的顶点的下一次遍历过程中访问

到，也就不存在 BE 了。
2. 假设无向图的 BFS 树中存在 DE，则在 DFS 过程中，当遍历到该节点时，会将该节点连接的所有其它顶点放到队列中，此时 DE 指向的顶也就放到队列中，因此最后得到的 BFS 树中该点就是 TE，不可能是 DE。

Problem 5.19 (二部图(必选))

可以用 DFS 来判断二部图，方法是在 DFS 过程中，记录每个顶点在 DFS 树中的层数（层数从 1 开始），若在遍历过程中遇到 BE，则判断 BE 指向的点的层数和该点的层数奇偶性是否不同，即奇数层可以指向偶数层，偶数层可以指向奇数层，但是若遇到 BE 的奇偶性相同，则可以判断该图不是二部图。

BFS 也可以判断二部图，方法和 DFS 一样，两者在时间复杂度方面都一样，不过可能 BFS 更符合直觉一点。

Problem 5.20

若图中存在环，则从环上去掉一条边图仍然连通。若图中不存在环，则不可以从 G 中移除一条边，使 G 仍然保持连通。

因此只要用 DFS 或者 BFS 来找图中的环即可，时间复杂度为$O(V+E)$。

Problem 5.21

1. 递归

```
findHeight(root):
    if root == NULL:
        return 0
    else:
        return max(findHeight(root.leftChild), findHeight(root.rightChild)) + 1
```

2. 调用上述过程：

```
findDiameter(root):
    return findHeight(root.leftChild) + findHeight(root.rightChild) + 2
```

Problem 5.22 ((必选))

用递归算法，基于这样的判断：一个二叉树是完美二叉树，当且仅当它的两个子树是完美二叉树，且两个子树的高度相同。

```
maxHeight = 0
result = []
findPerfectBinaryTree(root):
    if root == NULL:
        return (0, true)
    (l_Height, l_isPBT) = findPerfectBinaryTree(root.leftChild)
    (r_Height, r_isPBT) = findPerfectBinaryTree(root.rightChild)
    if l_isPBT && r_isPBT && l_Height == r_Height:
        if l_Height + 1 > maxHeight:
            result.clear()
            result.add(root)
            maxHeight = l_Height + 1
        else if l_Height + 1 == maxHeight:
            result.add(root)
        return (l_Height + 1, true)
    else:
        return (max(l_Height, r_isPBT) + 1, false)
```

时间复杂度为$O(n)$。

Problem 5.23

对图 G 进行预处理：进行一次 DFS 遍历，记录每个节点的 discoverTime 和 finishTime。

查询过程基于这样的推断：节点 u 是 v 的祖先，当且仅当$u.\mathrm{discoverTime} < v.\mathrm{discoverTime}\ \ \&\&\ \ u.\mathrm{finishTime} > v.\mathrm{finishTime}$。

Problem 5.24 ((必选))

1. 在有向图和无向图的 DFS 过程中，若遇到 BE，则图中存在环。
   在有向图的 BFS 过程中，若遇到 BE，则图中存在环；在无向图的 BFS 过程中，若遇到 CS，则图中存在环。
2. 采用对手策略，我们可以设计一个有$\Omega(n^2)$条边的图，该算法以$O(n)$的时间不可能遍历所有的边，因此一定存在该算法没有访问的边，那么我们就可以设计这样一张邻接表，使得以老师的算法过程，判断出该图中不存在环（因为该算法没有考虑所有边），但是其实该图中是有环的（只要在该算法遍历过的任意两点间加上两条边即可）。

Problem 5.25 ((必选))

1. 若边的权值为 1，则 G 的最小生成树为 n-1，以任一节点做 DFS 或 BFS 得到的 DFS 树或 BFS 树都是最小生成树。
2. 判断图是否连通需要$O(V+E)$，对于$m=n+10$的图，我们循环尝试删除图中的权值最大的边，即，每次判断删除图中权值最大的边是否满足连通性，若不满足则取权值次大的边，以此类推，删除 11 条边后算法结束。时间复杂度为$O(V+E)$。
3. 对图进行一次 DFS 遍历，在遍历过程中，每次尽可能先取当前节点的权值为 1 的出边进行遍历，这样最终得到的 DFS 树就是 MST。

Problem 5.26

1. 在标准的 BFS 过程中，每次遍历节点 v 的邻居 w，若 w 为白色则将其加入队列，并且设置 v 是 w 的父节点，以此来构造 BFS 树。我们对其进行适当修改，对每个节点记录其在 BFS 树中的高度，然后，对于 v 的邻居 w，若 w 不是白色，但是 w 的高度是 v 的高度+1，则不将 w 加入队列，但是仍然将 v 设置为 w 的父节点。这样就构成了图G'。

~~2. TODO~~

Problem 5.27

用图建模，图中有 n 个点，若 A 认识 B，则有一条从n_A到n_B有向边。题述条件即可描述为，求图 G 的一个子图G'，其中G'中每个节点的出度d_v的取值范围是$5 \le d_v \le n(G')-5$。$n(G')$表示G'中顶点个数。

算法过程为，每次从图中去掉一个不满足条件（$5 \le d_v \le n(G')-5$）的顶点，然后判断子图是否满足条件，循环直到找到满足条件的子图，或者不存在。

判断一个子图是否满足条件需要$O(V+E)$的时间，每次去掉一个顶点需要更新图，更新的时间为该顶点连接的边的个数，整个算法最多去掉所有点和所有边，因此去掉顶点的总的时间复杂度为$O(V+E)$，因此该算法的时间复杂度为$O(V+E)$，在此题中，$0 \le E \le n^2$，因此时间复杂度在$O(n)$和$O(n^2)$之间。

姓名：陆依鸣

学号：151220066

邮箱：luyimingchn@gmail.com