研究生算法课课堂笔记

上课日期：2016/11/21　　　　　　　　第(2)节课

组长学号及姓名：1601214750 周东

组员学号及姓名：1601214753 王帅　　1601214732 马力

一、内容概要

第二节课主要讲授的内容总结如下：

1. 二部图(Bipartite Graph)的性质
2. 有向图(Directed Graph)的性质
3. 有向无环图(Directed Acyclic Graph, DAG)的性质和拓扑排序(Topological Sorting)

二、详细内容

1. 二部图(Bipartite Graph)的性质

1.1 定理：无向图 G 为二部图的充要条件是 G 不存在长度为奇数的环。

① 必要性：

设 G 为二部图 $<V_1, E, V_2>$，由于 V_1、V_2 非空，故 G 至少有两个顶点。

因为 G 为二部图，所以必然存在一条分界线 L 将属于 V_1 和 V_2 的顶点分别划分到分界线两侧。假设存在一个长度为奇数的环，环的每条边连接的两个顶点必然是分别属于 V_1 和 V_2，所以环的每条边必然经过分界线 L。选择 V_1 中的一个点作为环的起点，经过分界线 L 跳跃奇数次后无法回到同一侧，也即无法成环。所以 G 不存在长度为奇数的环。

② 充分性：

以下讨论假设图为连通图，否则可以对各个子图同样进行如下讨论。

若环的长度为 0，即不成环，无向图 G 变为一棵树，对树分层交替染色必然是二部图。

若环的长度为大于 0 的偶数，取 $v_0 \in X$，全部顶点集合为 V，将所有的顶点划分为两类：

　　$X = \{v \mid v = v_0$ 或 v 到 v_0 有偶数长度的通路$\}$

　　$Y = V - X$

显然 X、Y 均非空。接下来我们证明图中所有边上的两点分别属于 X 和 Y。

采用反证法，假设有边(u, v)，使 $u \in X, v \in X$。那么，v_0 到 u 有偶数长度的通路，或 $u = v_0$；v_0 到 v 有偶数长度的通路，或 $v = v_0$。无论何种情况，均有一条从 v_0 到 v_0 的奇数长度的闭路径，因而有从 v_0 到 v_0 的奇数长度的回路，与题设矛盾。故不可能有边(u, v)使 u, v 均在 X 中。

所以所有的边上两点都分别属于两个集合，也即此图为二部图。

1.2 二部图判定的算法

可以使用广度优先搜索(以下简称 BFS)或者深度优先搜索(以下简称 DFS)对二部图进行遍历，下面是基于 DFS 遍历进行判定的算法：

(1) 初始化所有顶点的值为 0

(2) 随机选择一个顶点 v_0 作为起始点，值 $P(v_0)=1$;

(3) 访问与当前顶点连接的下一个顶点 v_{i+1}：

若 $P(v_{i+1}) == 0$，则 $P(v_{i+1}) = -P(v_i)$，若仍然有未遍历顶点则继续(2)，否则输出成功；

若 $P(v_{i+1}) == -P(v_i)$，若仍然有未遍历顶点则继续(2)，否则输出成功；

若 $P(v_{i+1}) == P(v_i)$，终止遍历，输出失败。

2. 有向图(Directed Graph)的性质

2.1 无权有向图的最短路径问题

对于无权有向图，因为**边的数目即路径的长度**，可以通过 BFS 遍历得到最短路径。因为 BFS 遍历的生成树不存在跨层的边，所以得到的路径必然最短。

补充：对于带权有向图，其最短路径问题较为复杂，可以参考算法设计(英文版)P137 (Dijkstra)、P292 (Bellman-Ford)，以及算法导论(英文版)P651 (Bellman-Ford)、P658 (Dijkstra)、P693(Floyd-Warshall)进行学习。

2.2 对于无向图使用 BFS 遍历得到的生成树不存在的边可能连接在树的同一层或者下一层，那么对于有向图是否相同？如果是使用 DFS 遍历得到的生成树呢？

对于 BFS 遍历得到的生成树除了可能存在同一层或者下一层(图 1 a, b)，还可能存在子孙指向祖先的边(图 1 c)，以及一个分支指向另一个分支跨很多层向上的边(图 1 d)。示意图如下：

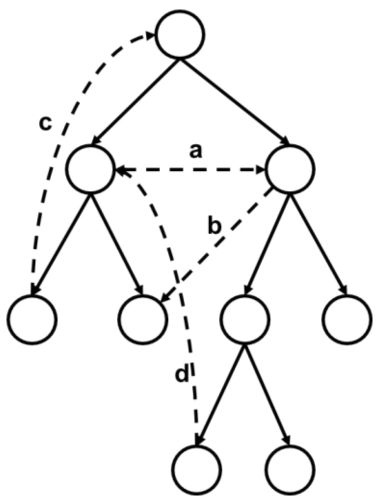

图 1 有向图的 BFS 遍历

对于 DFS 遍历得到的生成树除了可能存在子孙指向祖先的边(图 2 a)以外，也可能存在祖先指向子孙的边(图 2 b)，或者跨分支的横向边(图 2 c)、纵向向下(图 2 d)或向上(图 2 e)的边，并且层数没有限制。示意图如下：

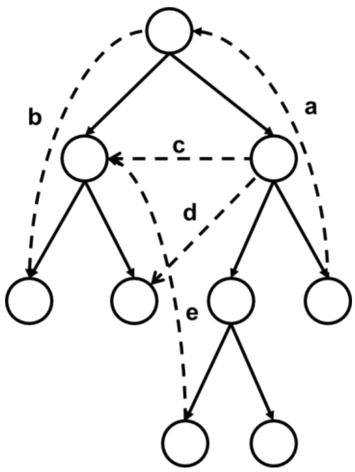

图 2 有向图的 DFS 遍历

2.3 (算法设计 *Algorithm Design* P108 第 6 题)对一个无向图使用 BFS 和使用 DFS 遍历得到的生成树相同，证明这个图就是树本身。

根据第一节课内容，对于无向图使用 BFS 遍历得到的生成树不存在的边可能连接在树的同一层或者下一层(不同分支)，对于无向图使用 DFS 遍历得到的生成树不存在的边只能是由子孙指向祖先的边(相同分支)，所以二者相互否定，故不存在其他的边。

2.4 对于 2.3 的问题，教材并没有限制无向图，那么对于有向图这一结论是否成立呢？

不成立。我们举一个反例见下图：

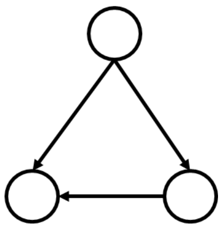

图 3 题 2.3 对于有向图的反例

2.5 (算法设计 *Algorithm Design* P110 第 9 题)对于 n 个节点的无向图 G，存在两个节点 s 和 t，如果 s 和 t 两个节点之间的距离严格大于 n/2，那么在二者中间必然有一个节点 v，删除 v 之后 s 和 t 之间无法连通(单点失效)。

我们可以对无向图 G 从 s 开始做 BFS 遍历得到一棵生成树(见图 4)。

因为 s 到 t 之间的距离严格大于 n/2，所以在生成树上 s 和 t 中间至少有 n/2 层节点；又因为通过 BFS 遍历得到的生成树不存在跨层的边，总的节点数为 n，如果每层节点个数大于等于 2，总节点数大于 n。所以则必然有一层节点只有 1 个，去掉这一层的这个节点之后，s 到 t 则无法连通。

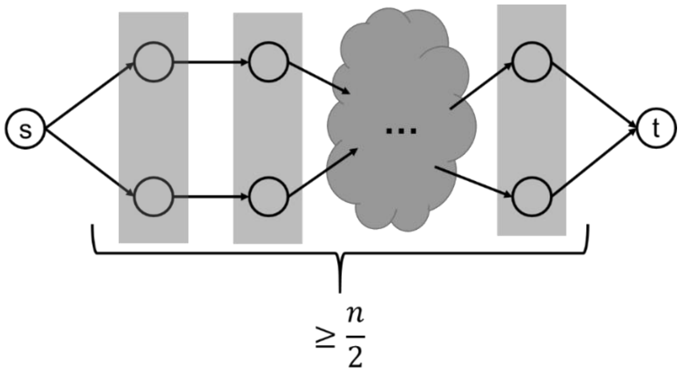

图 4 单点失效示意图

2.6 如何判断有向图的强连通？

定理：若任意选择一个顶点 S 开始做 BFS(或者 DFS)可以遍历所有的节点，并且有向图中所有的边都反向以后通过 S 仍然可以遍历到所有的节点，那么图是强连通的。

证明：可以把节点 S 当做中继节点，对于原始方向进行遍历的结果表明 S 可以到达任意节点，反向之后仍然可以遍历所有节点表明所有节点都可达 S，所以任意两个节点都可以通过 S 连接到对方。

2.7 如何用程序实现 2.6 的算法？

使用邻接链表来存储图，同时保存出边表和入边表，反向遍历时从此顶点的入边表遍历即可。

伪代码：

```
//使用邻接链表保存图 G=< V, LI, LO>, LI 为入边表, LO 为出边表。
// DFS 遍历
Function DFS (i, side):
    depth = 0
    If G[i] is not visited:
        For v in G[i][side]:
            depth += DFS(v, side)
    depth += 1
    Return depth
//  判断图是否为强连通图 SCC
Function SCC ( ):
    If DFS(0, LO) == len(G) && DFS(0, LI) == len(G):
        Return True
    Else:
        Return False
```

2.8 如果有向图不是强连通的，那么可以分成若干子图，每个子图内部都是强连通的。
这些子图称为有向图的强连通分量。

最大强连通分量：给定一个顶点 s，找到这个顶点 s 所在的强连通分量可以先遍历 s 所有正向可以连通的节点集合 T1，然后再反向遍历找到 s 可以连通的节点集合 T2，对 T1 对 T2 取交集得到的就是 s 所属的最大强连通分量。

对剩余的节点仍然执行同样的操作可以得到若干强连通分量。

3. 有向无环图(Directed Acyclic Graph, DAG)的性质和拓扑排序(Topological Sorting)

3.1 如果一个无向图没有环最多能有多少条边？有向图呢？

结论：无向图最多 N-1 条边，有向图最多 $O(N^2)$条边。

证明：

对于无向图只有 N 个节点，不存在环则只能是树的结构，只有 N-1 条边。

对于有向图，可以将其进行拓扑排序，在拓扑排序中任意节点都可以和排序在其后的节点构成一条边，构成一个等差数列{N-1, N-2, ..., 1}，所以共有 N*(N-1)/2 条边，近似看作 $O(N^2)$条边。

3.2 DAG 不可能是强连通的。

强连通的有向图指该有向图中任意两个节点之间至少有一条路径。DAG 中的任何两个节点之间最多只有一个方向是可达的（否则就会构成环），因此不可能是强连通的。

3.3 如何实现一个拓扑排序？

方法一：时间复杂度 $O(N^2)$

(1) 遍历图，找到一个入度为 0 的顶点，作为拓扑排序的一个点输出，并将其指向的所有顶点的入度均减 1；
(2) 循环(1)直至全部顶点均输出。

该算法每次都需要扫描所有的顶点以判断其入度，所以复杂度为 $O(N^2)$。

方法二：时间复杂度 O(m+n)

(1) 将图全部扫描一遍，记录所有节点的入度到集合 D，并把所有入度为 0 的顶点放到队列 Q 里；
(2) 取出一个顶点并输出，将其指向的所有顶点的入度均减 1 更新 D，如果减 1 之后该顶点的入度变为 0，则放入队列 Q；
(3) 循环(2)直到队列为空。

每个有向边均被访问一次，若初始所有顶点入度总和为 m，则算法复杂度为 o(m+n)。

三、思考问题

Q1. 如果不确定一个有向图是否无环如何对其进行拓扑排序？

使用 BFS 对图进行遍历，将当前所有入度为 0 的顶点置入队列。如果在遍历过程中发现队列为空但是仍然有顶点没有输出，则表明剩余的节点入度均不为 0，即出现了环，则此图不是 DAG，无法进行拓扑排序。

Q2. 对于 Q1 如何结合 DFS 实现？

对于 DFS，使用栈保存拓扑排序结果。

对每个节点，首先遍历其所有子节点，遍历完成后将当前节点放入保存拓扑排序的栈内，最后将栈内的元素按照 FILO 输出即可得到拓扑排序。

如果不确定中间是否有环，可以将每个节点标记为三种状态：已访问、未访问、正在访问。如果 DFS 过程中遇到了正在访问的顶点，说明形成了回路，则无法进行拓扑排序。

伪代码（假设从第 0 个顶点可达其它所有顶点）：

```
// 使用 DFS 对有向图进行拓扑排序
// 使用变量：S(栈)保存排序结果, G 表示有向图, V 表示访问标记
// V 的三种状态：2(已访问), 1(正在访问), 0(未访问)
Function DFS ( i ):
    If V[i] == 2:
        Return 0                //本节点已经执行完毕
    If V[i] == 1:
        Return -1               //出现回路
    If V[i] == 0:
        V[i] = 1                //标记当前节点为正在访问
        For v in G[i].Childs：
            If DFS(v) == -1:
                Return -1       //如果有环提前终止遍历
        S.push(i)               //将当前节点入栈
        V[i] = 2                //标记此节点为已访问
        return 0

If DFS(0) != -1:
    While S is not empty:
        Print S.pop()
Else:
    Print "Not a DAG"
```

Q3. (算法设计 P107 第 4 题) 蝴蝶分类问题：有 n 个节点的集合 G，以及有 m 个判断，每个判断表明了集合中某两个节点是同色的或者异色的。已知节点只可能有两种颜色，试

实现一个复杂度为O(m+n)的算法判断这m个已知的判断信息是否一致(是否相互冲突)。

首先根据n个节点和m个判断来构建无向图G=<V, E>，当V_i和V_j之间存在一个判断时则图G中对应存在一条边，所以共有m条边。构建完成的图不一定是连通图，可能存在多个子图。

之后从图G任意选择一个顶点s标记为1，并开始进行BFS遍历，遍历的每一条边都对应一个判断，所以BFS的结果可以实现对图中所有节点的标记。

标记完成后可以根据m个判断逐个检查是否正确。

复杂度分析：

构建无向图G过程包括n个节点和m条边，复杂度为O(m+n)；

BFS遍历的复杂度为O(m+n)；

检查m个判断的复杂度为O(m);

所以总的时间复杂度为O(m+n)。