hw-4

秦啸涵　**521021910604**

0330

1. 对于给定的二叉树，求其最小深度，即从根节点到最近的叶子的距离

Solution：

使用广度优先遍历**BFS**，最先到达的叶子节点具有最小的深度，在遍历过程中记录深度即可，最差情况下遍历了所有节点，时间复杂度为$O(N)$，其中N为二叉树节点数。

伪代码如下：

```
Input: root             #二叉树根节点
Output: depth           #最小深度
int getdepth()
{
    depth = 0;          #此处假定根节点深度为0
    queue<node*,int> q;
    q.push(<root,0>);
    while (!q.empty())
    {
        cur,depth = q.front();
        q.pop();
        if (cur.isleaf())
            return depth;
        else
        {
            if (cur->left)
                q.push(<cur->left, depth+1>);
            if (cur->right)
                q.push(<cur->right, depth+1>);
```

```
20          }
21      }
22  }
```

2. 设G是有向非循环图，其所有路径最多含k条边，设计线性时间算法，将所有顶点分为k+1组，每一组中任意两个点之间不存在路径

Solution :

有向无环图DAG一定存在入度为0的顶点和出度为0的顶点(否则由于是有限图，一定可以找到环)，而有向图中的最长路径一定是从入度为0的顶点开始，到出度为0的顶点结束(否则可以继续扩充获得更长的路径，无环保证了扩充的节点一定不会出现在之前的路径上)。

按照拓扑排序的顺序，删除所有入度为0的节点后，上述最长路径的长度一定减少1，且这些入度为0的节点两两之间一定不会存在边(否则入度不为0)。因此可以按照拓扑排序中删除节点的批次将节点分为k+1组，且每一组中两个节点之间不存在路径，其时间复杂度与拓扑排序相同，都是$O(|V|+|E|)$，是线性时间复杂度。

3. 给定一个有向图G(V,E)，其中边的权重可以是 x,$2x$,或者$3x$(x是一个正整数），O(E+V)时间计算从源s到其它各个顶点的最小成本路径。

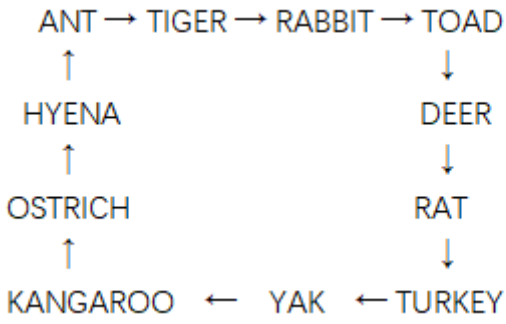
Solution :

基本思路：拆边。若v到w之间有边：

- 若边权值为x,无需改动
- 若边权值为2x,则删去原有权值为2x的边，在v,w之间增加一个虚拟节点tmp,并连边$v \to tmp \to w$,使这里每条边权值均为x
- 若边权值为3x,则删去原有权值为3x的边，在v,w之间增加两个虚拟节点tmp1和tmp2,并连边$v \to tmp1 \to tmp2 \to w$，使这里每条边权值均为x

在增加虚拟节点的过程中，我们需要维护一个布尔数组，如果某个节点是真实存在的，那么这个节点对应的值设置为**true**,反之如果是增加的虚拟节点我们将其设置为**false**。由此我们得到了一个新图$G'(V',E')$,在G'中，所有边权值均为x，此时从源点s出发的最小成本路径即为总长度最短的路径(可仅考虑长度不考虑权值)。为了求得s到固定点的最短路径，我们仅需进行一次**BFS**，在**BFS**过程中同时记录第一次入队时的距离(即为最短距离)，就得到了所有s可达的点与s的最短距离。此时遍历布尔数组筛选出所有的真实节点，我们就得到了从源点s到其他各个顶点的最小成本路径。对图更改的过程中我们需要遍历所有的边，时间复杂度为$O(E)$，**BFS**时，更改过后的图的规模最多不超过原有图的3倍(权值最大为3x)，因此**BFS**遍历的时间复杂度为$O(V+E)$，总的时间复杂度为$O(V+E)$。

4. 给定一组英文单词，检查这些单词是否可以按照如下规则重新排列形成一个圆圈，该规则为：两个单词X 和 Y，如果X的最后一个字符与Y第一个字符相同，则Y可以接在X后面。例如，考虑如下单词[ANT, OSTRICH, DEER, TURKEY, KANGAROO, TIGER, RABBIT, RAT, TOAD, YAK, HYENA]，则重排的结果为：



Solution :

初始的思路:

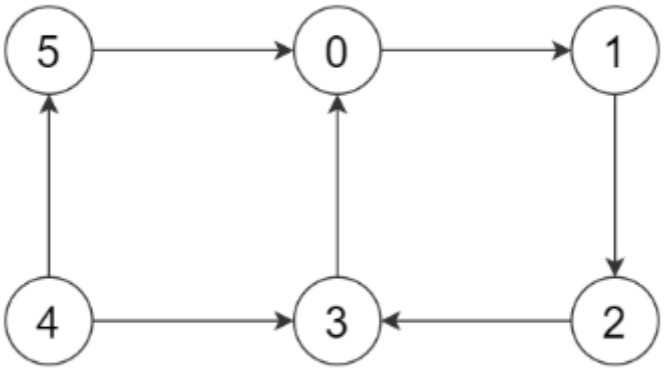
将每个单词看作一个节点，两个节点v和w之间有边当且仅当v表示的单词的最后一个字符与w表示的单词的第一个字符相同(此处边有向，为v指向w)。遍历所有单词，我们即可获得一个有向图，原题目等价于在这个有向图中找到哈密顿回路。

有向图的哈密顿回路问题为**NP完全问题**，为求得哈密顿回路，可使用dfs不断搜索、回溯枚举所有可能的解。(直接枚举全排列为阶乘级时间复杂度，dfs回溯优化可到指数级时间复杂度，但没有多项式时间复杂度的算法)

改进的思路:

由于将单词看作节点带来的时间复杂度过于高昂，我们换一种方式来考虑：将26个字母当作节点，将每个单词看作边，比如DEER这个单词，它将在图中增加一条由D指向R的边。则问题转化为在原图中寻找一条欧拉回路，即经过每条边各一次最终回到原点的路径。那么存在性等价于对于图中的每个点，判断其出度是否等于入度；若存在，可以使用DFS遍历整个图即可求出欧拉回路(上一次作业已经实现)，其时间复杂度为线性时间复杂度

5. 有向图的根顶点是指图中的所有其它顶点都可以从根顶点到达。一个图可以有多个根顶点(此时只要找到一个即可)，也可能没有根顶点。下图根顶点是4:



Solution :

为找到根节点，我们只需从每个节点出发进行一次dfs搜索，如果所有节点都被访问到，那么这个节点就是该有向图的根节点，否则对其他节点重复dfs即可，若遍历完全部节点依然没有找到这样的节点，则该有向图没有根节点。该算法时间复杂度为$O(V \times (V + E))$。

0406

1. 对于给定的二叉树，求其最小深度，即从根节点到最近的叶子的距离

与0330的第一题一样，不再重复。

2. 设G是有向非循环图，其所有路径最多含k条边，设计线性时间算法，将所有顶点分为k+1组，每一组中任意两个点之间不存在路径

与0330的第二题一样，不再重复。

3. 求n*m棋盘上任意两点之间马能够走的最短路径长度

Solution :

构造图G=(V,E)

- V为棋盘上的所有顶点
- 若马能一步从某个节点v走到w，则连边(v,w)(按无向图处理)

认为所有边权值一样均为1，由于是求出任意两点之间的最短距离长度，我们可以对每个源点使用`dijkstra`算法求出到其他点的最短路，也可以利用`floyd`算法直接求出所有点对之间的最短距离。

4. 给定连通无向图G，以及3条边a,b,c，在线性时间内判断G中是否存在一个包含a和b但不含c的闭链。

Solution :

闭链一定包含于某个双连通分支中，因此可进行如下操作：

1. 对图G划分双连通分支，在划分过程中标记a,b,c的归属，此过程时间复杂度为$O(V+E)$
2. 分情况讨论：
    - 若a,b在同一个双连通分支而c不在此分支，则一定存在一个包含a和b但不含c的闭链
    - 若a,b不在同一个双连通分支，则一定不存在这样的闭链
    - 若a,b,c均在同一个双连通分支，考虑在这个双连通分支中删去c，由于该分支原来是双连通的，因此删去c后一定还是连通的，此时对这个连通分支再做双连通划分，若a,b又被分到同一个双连通分支，则存在这样的闭链，反之不存在，由于是原图的子图，该操作的时间复杂度不超过$O(V+E)$

故总的时间复杂度为$O(V+E)$,是线性时间复杂度。

对连通图划分双连通分支的伪代码：

```
Input: G(V,E), v(DFS树顶点), n(节点数量)
Output: biconnected components and High values
main:
    for v in G:
        v.DFS_Number=0;
    DFS_N=n;
    BC(v);

BC(Node v):
    v.DFS_Number = DFS_N;
    DFS_N--;
    stack.push(v);
    v.High = v.DFS_Number;
    for all (v,w) in E:
        stack.push((v,w));
        if (w is not parent of v)
        {   if (w.DFS_Number == 0)
                BC(W);
                if (w.High < v.DFS_Number)      //v与w不能通过图的其余部分连通
                    repeat remove all edges and vertices from stack until v is reached
                    send them as a biconnected component
                    stack.push(v);
                v.High = max(v.High, w.High);
            else                                //w已经被访问过
                v.High = max(v.High, w.DFS_Number);
         }
```

5. 设计线性时间算法求树的最大匹配

Solution：

树中一个非叶子节点连向其子节点的若干条边及连向其父节点的边中，至多只有一条边属于匹配，因此可以考虑进行树形dp。

记$f(v,b)$为以v为根节点的子树在b状态下的最大匹配个数，其中b为bool值可取0,1

- b=1时，代表最大匹配包含v这个根节点
- b=0时，代表最大匹配不包含这个根节点

不难看出$f(v,1) \geq f(v,0)$(由于是树，若匹配不含根，从根节点出发要么通过交替得到的新匹配与原来数量相同，要么是一条增广路径可以比原匹配多1），则原问题可通过dp转化为若干个子问题

以根节点v有三个子节点w_1,w_2,w_3为例，列写状态转移方程：

$$f(v,0)=f(w_1,1)+f(w_2,1)+f(w_3,1)$$

$$f(v,1)=max\begin{cases}f(w_1,0)+f(w_2,1)+f(w_3,1)+1\\f(w_1,1)+f(w_2,0)+f(w_3,1)+1\\f(w_1,1)+f(w_2,1)+f(w_3,0)+1\end{cases}$$

当v为叶子节点时，$f(v,0)=f(v,1)=0$

因此我们只需要对树进行一次后序遍历，就可以得到所有的$f(v,b)$，最终以v为根的树的最大匹配即为$max[f(v,0),f(v,1)]$(实际就是$f(v,1)$)

时间复杂度与后序遍历相同，为$O(V+E)$，是线性时间复杂度。(实际上对于树，一定有$E=V-1$，因此写做$O(V)$或者$O(E)$亦可)

6. 无向图G的顶点覆盖是指顶点集合U，G中每条边都至少有一个顶点在此集合中。设计线性时间算法为树寻找一个顶点覆盖，并且使该点集的规模尽量小。

Solution :

依然使用树形dp，记$f(v,b)$为以v为根节点的子树在b状态下的最小点覆盖的点的个数，其中b为bool值可取0,1

- b=1时，代表最小点覆盖包含v这个根节点
- b=0时，代表最小点覆盖不包含这个根节点

原问题可通过dp转换为若干个子问题，

依然以根节点v有三个子节点w_1,w_2,w_3为例，列写状态转移方程：

$$f(v,0)=f(w_1,1)+f(w_2,1)+f(w_3,1) \qquad \text{不选择}v\text{,则必须选择}w_1,w_2,w_3$$

$$f(v,1)=min(f(w_1,0),f(w_1,1))+min(f(w_2,0),f(w_2,1))+min(f(w_3,0),f(w_3,1))+1$$

选择了v,则w_1,w_2,w_3可以选也可以不选，找较小的点集即可

更一般的，可以写为：

$$f(v,0)=\sum_{son} f(son,1)$$

$$f(v,1)=\sum_{son} min[f(son,0),f(son,1)]+1$$

当v为叶子节点时，$f(v,0)=0,f(v,1)=1$

依然对树进行后序遍历以得到所有的$f(v,b)$，最终以v为根的树的最小点覆盖的点的数量为$min[f(v,0),f(v,1)]$，由于题目要求找到对应的顶点，因此我们在后序遍历求$f(v,b)$的过程中，需要开辟额外的空间维护当前的最优解对应的顶点。总的时间复杂度与后序遍历相同，是$O(V+E)$，是线性时间复杂度。