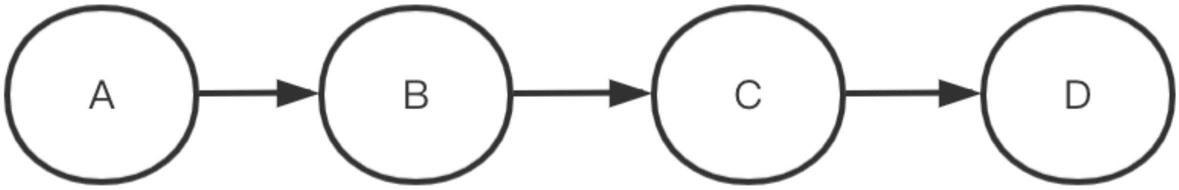
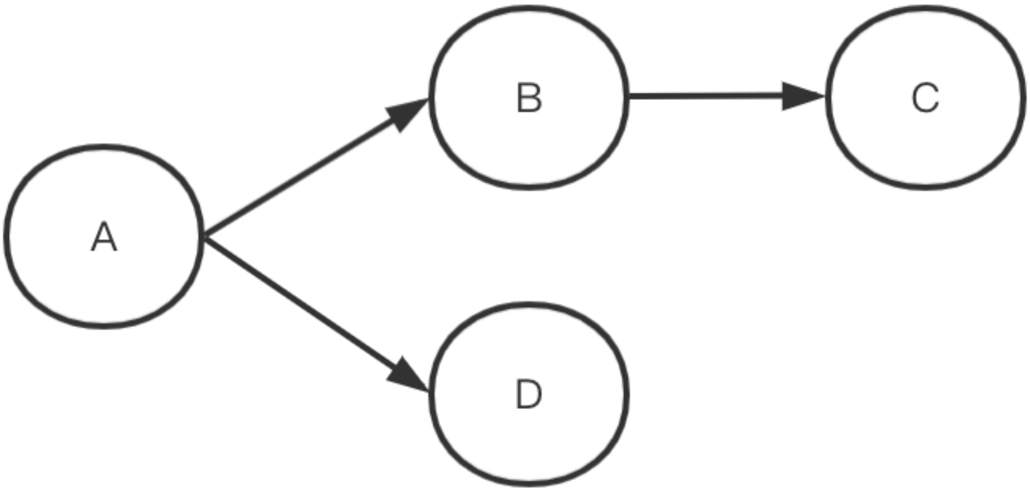
4.1



DFS的访问顺序:ABCD

BFS的访问顺序:ABCD



DFS的访问顺序:ABCD

BFS的访问顺序:ABDC

4.2

(a)

```
inOrderTraversal(node){
  if(node == null) return []
  res = []
  res.extends(inOrderTraversal(node.left))
  res.extends(node.key)
  res.extends(inOrderTraversal(node.right))
  return res
}
```

> inOrderTraversal函数接受一个结点的引用，先定义一个结果数组res,通过递归将左子树排序好的数组添加至res,再把node结点的值添进去，然后再通过递归将右子树排序好的数组添加至res，最后返回一个排序好的以node为根树上结点值的有序的数组。

(b)

因为该算法就是一次树上的递归中序遍历，每个节点只访问了一次，所以渐进时间为$\Theta(n)$

(c)

1.$a_1, a_2, ..., a_n$的值并不会影响算法的时间复杂度。

2.BST的结构也不会影响到算法饿时间复杂度,因为不管树的结构如何排列，每个节点都只访问一次。

4.3

```
dfs(node,arr,depth){
  if(node == null) return
  dfs(node.left,arr,depth+1)
  arr[depth].append(node.val)
  dfs(node.right,arr,depth+1)
  return
}

levelAverage(){
  arr = []
  dfs(root,arr,0)
  for (i = arr.length-1; i >=0; --i){
    level = arr[i]
    sum = 0
    for (j = 0; j < level.length; ++j){
      sum += level[j]
    }
    print("level:"+i+"avrage:"+sum/level.length)
  }
}
```

4.4

(a)c->d->e->a->b

(c)d->e->a->b->c

(e)a->b->e->c->d

4.5

该图不是二分图

因为存在一条奇数个点的回路$1 \rightarrow 3 \rightarrow 13 \rightarrow 12 \rightarrow 9 \rightarrow 7 \rightarrow 2 \rightarrow 1$,而奇数个顶点组成的环不可能为二分图

> 我们可以简单假设2k+1个顶点连成的2k条边的环图，前2k个顶点都已经着色，则第1个点和第2k个点的颜色必定相同，则此时第2k+1个点的颜色无论是什么颜色都会和第1个点或者第2k个点的颜色冲突。

4.6

(a)

我们设有影响力的人为p

(1)题目要求我们说明所有p都在同一个强连通图内，我们用反证法证明

假设p1所在的强连通图为g1,p2所在的强连通图为g2，且它们不在一个强连通图里。g1和g2是两个不同的强连通图，由于p1可以到达p2，且p2也可以到达p1。所以这两个连通图间的任意两个点都存在一条路径可以到达，因此这两个强连通图可以组成一个大的强连通图g3，p1,p2都在g3这个大的强连通图里。所以假设不成立，原命题为真。

(2)证明这个强连通图里的所有人都是有影响力的人

因为这个强连通图里的每个人都可以将消息传给图里有影响力的人p，而p又可以将消息传给所有G中的人，所以换个说法，强连通图里的每个人都可以将消息传给G中的每个人。所以根据定义强连通图里的每个人都是有影响力的人。

(b)

```
#拓扑排序
topological-sort(G){
  call DFS(G) to compute finising times v.f for each vertex
  as each vertex is finished, insert it onto the front of a   linked list
  return the linked list of vertices
}


#强连通
strongly-connected-components(G){
  call DFS(G) to compute finishing times u.f for each vertex u
  compute G^T
  call DFS(G^T),but in the main loop of DFS,consider the vertices in order of
decreasing u.f (as computed in line 1)
  output the vertices of each tree in the depth-first forest formed in line 3
as a separate strongly connected component
}


#寻找影响力的人
findinfluentialpeoples(G){
  g_scc = strongly-connected-components(G)#得到G的强连通图
  g_scc_list = topological-sort(g_scc)#将强连通图以每一个强连通分量为节点进行拓扑排序
  for v in g_scc_list[0]:#便利拓扑排序后的第一个强连通分量
    print("influential peple:"+v)
}
```

正确性:

因为第一问得知所有的有影响力的人都在同一个强连通分量中，所以我们可以先获取图中的所有强连通分量，其中肯定有一个就是我们的结果，那是哪一个呢？因为题目说了一定存在一个有影响力的人，也就是说一定存在一个强连通分量能达到其他的所有强连通分量。我们对得到的所有强连通分量进行一次拓扑排序，这样得到的排序后的首位的强连通分量就是我们的结果，这个强连通分量可以到任意其他的强连通分量。所以最后遍历首个强连通分量中的人就行了

时间复杂度分析:

首先获取G的强连通分量为O(n+m),然后拓扑排序的时间复杂度为O(n+m),所以总的时间复杂度为O(n+m)

(C)

```
dfs(node,num){
  if(node.out_list.length == 0){
    return 1
  } else{
    sum = 1
    for(i = 0; i < node.out_list.length; ++i){
      sum += dfs(node.out_list[i],0)
    }
    return sum
  }
}

findinfluentialpeoples(G){
  g_scc = strongly-connected-components(G)#得到G的强连通图
  g_scc_list = topological-sort(g_scc)#将强连通图以每一个强连通分量为节点进行拓扑排序
  num = dfs(g_scc_list[0],0)#以拓扑排序后的第一个节点dfs，查看是否遍历完整个图
  if(num < g_scc_list.length){
    print("no influential person")
  }else{
    for v in g_scc_list[0]:#便利拓扑排序后的第一个强连通分量
      print("influential peple:"+v)
  }
}
```

正确性:

如何判断有没有一个影响力的人呢？我们在(b)中得到了拓扑排序后的强连通分量节点数组。我们定以强连通分量为一个节点的图为G2,原一个人为一个节点的图为G，如过存在一个有影响力的人在G中，那么拓扑排序后数组的第一个强连通分量里的人都是有影响力的人，因为这个强连通分量在G2中通过一次DFS可以走完整个G2图。而如果通过一次DFS走不完G2图，那么肯定不存在有影响力的人在G中。

时间复杂度:

获取G的强连通分量为O(n+m)，拓扑排序的时间复杂度为O(n+m)，一次DFS判断为O(n+m),所以总的时间复杂度为O(n+m)