组卷作业 #47 韩柳彤

选择题：

1. 一个栈的入站元素序列是 1，2，3，4，5 若允许出栈操作可在任意可能的时刻进行，则下面的序列中。不可能出现的出栈序列是（b）
   a) 3，4，2，5，1
   b) 2，5，4，1，3
   c) 2，3，1，5，4
   d) 3，5，4，2，1
2. 一颗最大深度为 d 的满二叉树包含的节点总数为（c）。
   a) 2^{d+1}
   b) 2^{d}
   c) 2^{d+1}-1
   d) 2^{d}-1
3. 如果采用邻接矩阵作为图的存储结构，则求最小生成树的 prim 算法的时间复杂度为（a）。
   a) $O(n^2)$
   b) O(n)
   c) O(n*logn)
   d) O(1)

填空题：

1. 假设一个顺序表长度为 N，在任何一个有效位置上删除一个数据元素时，需要移动元素的平均个数为____。
   答案： (N-1)/2
2. 一颗高度为 4 的二叉树含有 4 个度为 2 的结点和 5 个叶子节点，那么这个二叉树可能有____个度为 1 的结点（写出所有可能性）。
   答案： 0 或 1
3. 设一个有 n 个顶点和 e 条弧的有向图用邻接表表示，则删除与某顶点 vi 相关的所有弧的时间复杂度____。
   答案： O(n+e)

算法设计题：

注：除特殊要求外，您只需要添加注释就可以未经引用直接调用 STL 标准库中的任意函数。

一、 用队列实现栈

1. 只使用队列实现栈的下列操作：

void push(x) --元素 x 入栈
int pop() -- 移除栈顶元素，并返回
bool empty() --返回栈是否为空

注意，只能使用队列（STL 库中的<queue>）的基本操作。
即，只有 push(x),pop(),front(),size(),empty()等函数是合法的，back()不合法。
除此以外，假设所有操作都是有效的 (例如，对一个空的栈不会调用 pop 操作)。

2. 分析 pop 函数的时间复杂度。

代码模板：

```
class MyStack {
private:

public:
    MyStack() {}

    void push(x) {}

    int pop() {}

    bool empty() {}
};
```

答案：

```
class MyStack {
private:
        queue<int> Q1; //初始化队列，由标准库
        queue<int> Q2; //初始化队列，由标准库
public:
    MyStack() { }

    void push(int x) {
        if (Q1.empty())
        {
```

```
                Q2.push(x);
        }else{
                Q1.push(x);
        }
    }

    int pop() {//时间复杂度O(n)
        if (Q1.empty())
        {       while(!Q2.empty()){
                if (Q2.size() == 1)
                {
                        int temp = Q2.front();
                        Q2.pop();
                        return temp;
                }
                Q1.push(Q2.front());
                Q2.pop();
                }
        }else{
                while(!Q1.empty()){
                if (Q1.size() == 1)
                {
                        int temp = Q1.front();
                        Q1.pop();
                        return temp;
                }
                Q2.push(Q1.front());
                Q1.pop();
                }
        }
    }

    bool empty() {
        if (Q1.empty()&&Q2.empty())
             return 1;
        return 0;
    }

};
```

二、 寻找二叉树的最近公共祖先

给定一棵二叉树，找到该树中两个指定节点的最近公共祖先（结点可以是自己的祖先）。具体来说，设计一个函数，传入根节点指针和两个指定结点的指针，返回其最近公共祖先的指针。注意，默认传入的节点都在树中，即不会传入非法节点。

代码模板：

```
/*struct TreeNode {
*       int val;
*       TreeNode* left;
*       TreeNode* right;
*       TreeNode(int x) : val(x), left(NULL), right(NULL) {}
*};
*/
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p,
TreeNode* q)
 {

 };
```

答案：

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p,
TreeNode* q) {
    TreeNode* pointer = p;
        while(1){
                if (Search(pointer, q))
                        return pointer;
                pointer = Parent(root,pointer);
        }
}

TreeNode* Parent(TreeNode* root,TreeNode* current){
        stack<TreeNode*> aStack;
        TreeNode* pointer = root;
        aStack.push(NULL); //初始化栈，由标准库
        while(pointer){
      if (pointer -> left == current || pointer ->right == current)
                {return pointer;}
                else{
                        if (pointer -> right != NULL)
                                aStack.push(pointer->right);
                        if (pointer -> left != NULL)
```

```
                                pointer = pointer -> left;
                        else{
                                pointer = aStack.top();
                aStack.pop();
                                }
                        }
        }
        return root;
}

bool Search(TreeNode* root ,  TreeNode* tar){
        bool r = 0;
        bool l = 0;
        if (root  == tar)       return 1;
        else{
                if (root -> right){
                        r = Search(root -> right, tar);
                }
                if (root -> left){
                        l = Search(root -> left, tar);
                }
                return (r||l);
        }
}

};
```

三、 课程表

现在你总共有 *n* 门课需要选，记为 0 到 n - 1。一些课程在修之前需要先修另外的一些课程，课程的先修关系存储在一个由邻接表示的有向图中。比如要学习课程 0 你需要先学习课程 1，则存在一条由 0 指向 1 的有向边。给定 *n* 门课以及他们的**关系邻接表**，判断是否可能完成所有课程？

代码模板

```
/*邻接表类：Graph
*如果你需要用到一个新的邻接表类中的域，则默认其已经合理初始化
*对于邻接表类，下列成员函数已经可用
* int VerticesNum()//返回图中节点个数
* Edge FirstEdge( int oneVertex )//给出顶点，返回其第一条边
* Edge NextEdge( Edge preEdge)//给出一条边，返回下一条边
```

```
* int ToVertex(Edge e)//给出一条边，返回弧尾节点
*/

bool canFinish(Graph& G) {}
```

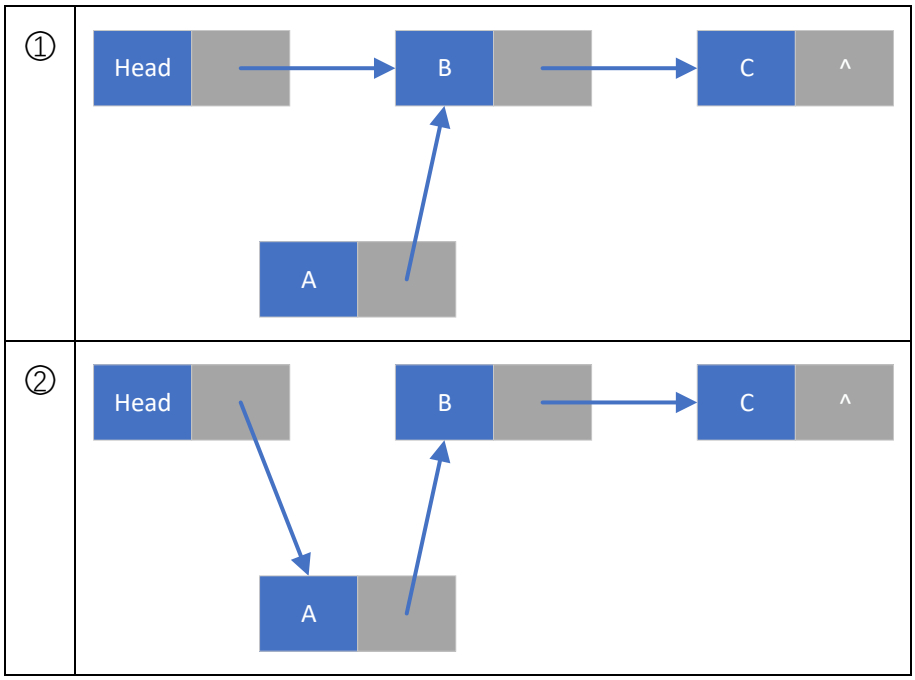
答案：

```
//思路：通过DSF进行拓扑排序，如果存在拓扑排序，则无环，即可行
bool canFinish (Graph& G){
        int vNum = G.VerticesNum();
        for(int i=0; i<vNum; i++){
                G.Mark[i]=UNVISITED;}//初始化标记数组（G的域）
        queue<int> Q;      //初始化队列，由标准库
        for(int i=0; i<vNum; i++)  {
        if(G.Indegree[i]==0) Q.push(i);
        }
    while(!Q.empty()){
            int V=Q.front();
            Q.pop();
         G.Mark[V]=VISITED;
        for(Edge e=G.FirstEdge(V);G.IsEdge(e);e=G.NextEdge(e)){
                G.Indegree[G.ToVertex(e)]--;
                 if(G.Indegree[G.ToVertex(e)]==0)
                        Q.push(G.ToVertex(e));
                }
        }
        for(int i=0; i<vNum; i++){
                if(G.Mark[i]==UNVISITED)
                return 0;
        }
        return 1;
}
```

解答题：

一、　画出在给定链表中头节点后插入一个值为 a 的新节点的过程示意图

答案：



二、　已知一个森林的先序遍历序列和中序遍历序列如下：

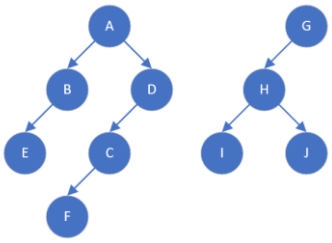
先序遍历序列：ABEDCFGHIJ

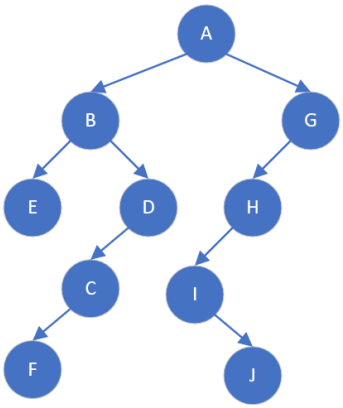
中序遍历序列：EBFCDAGIJH

完成下列问题：

1. 画出上述序列对应的森林
2. 画出这个森林对应的的二叉树
3. 写出这个二叉树的高度

答案：

1. 

2. 

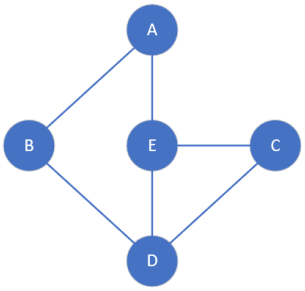
3. 高度：5

三、 已知一个无向图的顶点集为｛A, B, C, D, E｝，其邻接矩阵如下所示：

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

1. 画出这个图
2. 由定点 c 出发，写出任意一个广度优先遍历的顶点序列

答案：

1. 

2. CEDAB