problem2 算法分析

对于G中的任两个顶点，检查它们之间是否有路径存在。如有，输出其中一条路径。如无，返回false。(20')

```
//findRoad 函数
//读入起点和终点、以及一个输出用的一定长度的数组
//返回bool值，表示是否有路径，并将路径存入数组
template <class TypeOfVer, class TypeOfEdge>
bool adjListGraph<TypeOfVer, TypeOfEdge>::findRoad(int start, int end, int
*road, int &length)
{
    bool *visited = new bool[Vers];
    for (int i=0; i < Vers; ++i)
        visited[i] = false;
    seqStack<verNode> myStack;
    //将初始点进栈
    myStack.push(verList[start]);

    //对栈顶的结点进行搜索，直到没得选，或者找到终点
    for(bool found = false;!myStack.isEmpty() and !found;)
    {
        edgeNode *p = myStack.top().head;//p为指向后继的指针

        //对栈顶结点的后继而言，对第一个可以访问的后继进栈，直到没有后继或者找到目标
        //如果该结点有后继但都不符合要求，p会重新指回该节点
        while(p!= nullptr and !found)
        {
            int succeed = p->end;
            //如果后继符合，将后继进栈，并且指针指向后继
            if(!visited[succeed]){
                myStack.push(verList[succeed]);
                p = myStack.top().head;
            }
            //如果当前后继不符合，找下一个后继
            else p = p->next;

             //找到目标就直接跳出
            found = (succeed == end);
        }

        //出栈,并对出栈结点进行标记，直到顶部结点有后继
        while(p == nullptr and !found and !myStack.isEmpty()){
            int now = myStack.pop().ver;
            visited[now] = true;
            p = myStack.top().head;
        }
    }

    //输出结果，逐个出栈直到栈为空
```

```
    if(!myStack.isEmpty()){
        int i = 0;
        while(!myStack.isEmpty()){
            road[i++] = myStack.pop().ver;
        }
        length = i;
        return true;
    }
    else return false;
}
//main.cpp 文件中的测试代码
if(myGraph.findRoad(v1, v2, road, length)){
    for(int i = length-1; i >= 0; --i){
        std::cout << road[i] << ' ';
    }
    std::cout << std::endl;
}
```

用深度优先搜索实现拓扑排序，判断该图是否为无环图，是则给出拓扑排序结果，否则返回loop。(20').

```
//hasloop函数，分别从每个结点向下深度优先搜索
template <class TypeOfVer, class TypeOfEdge>
bool adjListGraph<TypeOfVer, TypeOfEdge>::hasLoop()
{
    bool *visited = new bool[Vers];
    for (int i=0; i < Vers; ++i) visited[i] = false;
    for (int i = 0; i < Vers; ++i) {
        bool result = dfs2(i, visited);
        if (result) return true;
        else for (int i = 0; i < Vers; ++i) visited[i] = false;
    }
    return false;
}
//从某个结点开始向下深度优先搜索，对经过的路径进行标记，如果碰到已搜过结点就返回true
template <class TypeOfVer, class TypeOfEdge>
bool adjListGraph<TypeOfVer, TypeOfEdge>::dfs2(int start, bool visited[])
{
    edgeNode *p = verList[start].head;
    visited[start] = true;
    while (p != NULL){
        if (visited[p->end] == true) return true; //后继结点是已搜过结点
        else if(dfs2(p->end, visited)) return true; //递归，当后继结点碰到已搜
过结点
        else p = p->next;
    }
    visited[start] = false;
    return false;
}
```

时间复杂度分析

```
//findRoad 函数
template <class TypeOfVer, class TypeOfEdge>
bool adjListGraph<TypeOfVer, TypeOfEdge>::findRoad(int start, int end, int
*road, int &length)
{
    //栈操作，时间复杂度O(V(V+E))
    for(bool found = false;!myStack.isEmpty() and !found;)
    {
        //进栈，时间复杂度O(V+E)
        while(p!= nullptr and !found)
        {
            if(!visited[succeed]){
                myStack.push(verList[succeed]);
                p = myStack.top().head;
            }
            else p = p->next;
        }
        //出栈，时间复杂度O(1)
        while(p == nullptr and !found and !myStack.isEmpty()){
            int now = myStack.pop().ver;
            visited[now] = true;
            p = myStack.top().head;
        }
    }
    //输出，时间复杂度O(V)
    if(!myStack.isEmpty()){
        int i = 0;
        while(!myStack.isEmpty()){
            road[i++] = myStack.pop().ver;
        }
        length = i;
        return true;
    }
    else return false;
}
//对每个点进行栈操作，时间复杂度O((V+E)V^2)
//main.cpp 文件中的测试代码
if(myGraph.findRoad(v1, v2, road, length)){
    for(int i = length-1; i >= 0; --i){
        std::cout << road[i] << ' ';
    }
    std::cout << std::endl;
}
```

```
//时间复杂度O(V(V+E))
//hasloop函数
template <class TypeOfVer, class TypeOfEdge>
```

```
bool adjListGraph<TypeOfVer, TypeOfEdge>::hasLoop()
{
    //时间复杂度O(V)
    for (int i=0; i < Vers; ++i) visited[i] = false;
    //对每个点都进行深度优先搜索，并且可能重复搜索，时间复杂度O(V(V+E))
    for (int i = 0; i < Vers; ++i) {
        //时间复杂度O(V+E)
        bool result = dfs2(i, visited);
        if (result) return true;
        else for (int i = 0; i < Vers; ++i) visited[i] = false;
    }
    return false;
}
//递归函数部分
template <class TypeOfVer, class TypeOfEdge>
bool adjListGraph<TypeOfVer, TypeOfEdge>::dfs2(int start, bool visited[])
{
    while (p != NULL){
        if (visited[p->end] == true) return true;
        else if(dfs2(p->end, visited)) return true;
        else p = p->next;
    }
    return false;
}
```