Homework 5

1. 是否存在一个有向无环图，它的某个拓扑排序是用基于深度优先搜索的算法得不到的？请证明你的结论。

   - **Solution**

     不存在，一个有向无环图的所有拓扑排序均可由深度优先搜索算法得到。

     对于任意的一个拓扑排序的逆拓扑排序，均可通过dfs得到。每次寻找出度为0的点，压入栈中，因为没有后继节点，因此只压入自身。循环往复，最后出栈得到所有拓扑排序。

     例：A->B->C的拓扑排序中，逆拓扑排序为C <- B <- A ，从C开始使用DFS算法，由于C的出度为0，因此算法只能得到C一个点，压入栈中，继续寻找下一个出度为0的点，即B，然后A，压栈，最后出栈，便得到了预期的拓扑排序A->B->C。因此所有的拓扑排序均可用基于DFS算法得到。

2. 一个带有边权的有向无环图中（边权表示距离），点s到点t的关键路径是从s到t长度最长的路径。给定一个图，图的描述在文件 data.in 中，其中第一行为一个数字10，表示共有十个顶点（标号为0到9）；第二行为一个数字22，表示接下来有22条边；此后的22行，每行为三个数字u, v,和w，表示从顶点u到顶点v有一条权重为w的边。

   请编写C++程序，计算图中从顶点1到顶点7的关键路径。

   提示：有向无环图中所有边反向之后，依然是有向无环图；关键路径上的每个点x，s到x的最大距离和x到t的最大距离之和始终相等。

   1）请用一串数字表示求出的关键路径。（如04819表示从顶点0出发，依次通过顶点4、8、1最后到达顶点9）

   2）请给出计算关键路径的基本算法原理和关键代码的C++实现（请用注释解释你的算法的每一步）。

```
1->7
the longest length: 175
the longest path: 1 9 2 4 6 7
```

```cpp
//得到该顶点的拓扑排序,压入Stack中
void Graph::topologicalSortUtil(int v, bool visited[], stack<int> &Stack) {
    visited[v] = true;
    list<AdjListNode>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); i++) {
        AdjListNode node = *i;
        if (!visited[node.getV()]) {
            topologicalSortUtil(node.getV(), visited, Stack);
        }
    }

    Stack.push(v);
}
```

```cpp
void Graph::longestPath(int u, int v) {
    stack<int> Stack;  // 用于存储拓扑排序
    int *dist = new int[num + 1];
    bool *visited = new bool[num];

    //是否访问过 初始化
    for (int i = 0; i < num; i++) visited[i] = false;

    topologicalSortUtil(u, visited, Stack);  //得到拓扑排序，存进Stack

    /*到点u的距离初始化为0，到其他点的距离初始化为负无穷大*/
    for (int i = 0; i < num; i++) dist[i] = -100;
    dist[u] = 0;

    //根据得到的拓扑排序，进行longest的查找
    while (Stack.empty() == false) {
        int tmp = Stack.top();  //按照顺序取出拓扑排序中的值
        Stack.pop();

        // 更新到所有邻接点的距离
        list<AdjListNode>::iterator i;
        if (dist[tmp] != -100) {  //当u到该点的距离不为-100时 (-100代表负无穷远)
            for (i = adj[tmp].begin(); i != adj[tmp].end(); ++i) {
                int next = i->getV();  //取出点tmp的指向的下一个顶点next

                //若(u->next)的距离小于( u->tmp + tmp->next )的距离,则更新u到next的距离和路径
                if (dist[next] < dist[tmp] + i->getWeight()) {
                    dist[next] = dist[tmp] + i->getWeight();
                    Path[next].clear();
                    Path[next] = Path[tmp];
                    Path[next].push_back(next);
                }
            }
        }
    }
    //输出结果
    cout << u << "->" << v << endl;
    cout << "the longest length: " << dist[v] << endl;
```