作业

- 首先我们找到入度为0的节点包括，1、2、4。
- 删除这些节点之后，发现3、5两个节点变成入度为0了！
- 再删掉3、5，然后发现7入度变成0了
- 删掉7，然后就是6入读为0了
- 删掉6，然后就是9、13入度为0了
- 删掉9、13，发现8、10入读为0了
- 然后显然11、12入读为0了
- 删掉11、12，只剩下14了
- 所以序列：1 2 4 3 5 7 6 9 8 13 10 11 12 14

为了第一题专门写了代码，本质就是第二题增加了读取Buffer和写入Buffer。是个什么含义呢？就是我每次从q里面，要把所有要删除的节点拿出来，然后统一操作，操作完成之后要push进入q的新的元素用一个缓存区域接受，然后排序之后再统一写入。

```
        // 不断从队列中取出节点，并将其加入结果序列中
        while (!q.empty()) {
            vector<int> willPopBuffer;
            vector<int> willPushBuffer;
            while (!q.empty()) {
                willPopBuffer.push_back(q.front());
                q.pop();
            }
            sort(willPopBuffer.begin(), willPopBuffer.end());
            for (int i = 0; i < willPopBuffer.size(); i++) {
                res.push_back(willPopBuffer[i]);
            }

            for (int i = 0; i < willPopBuffer.size(); i++) {
                int node = willPopBuffer[i];
                for (int j = 0; j < adj_list[node].size(); j++) {
                    // TODO:将当前节点的所有邻居节点的入度数减1
                    // 如果减1后邻居节点的入度数变为0，则将其加入队列
                    int neighbor = adj_list[node][j];
                    in_degree[neighbor]--;
                    if (in_degree[neighbor] == 0) {
                        willPushBuffer.push_back(neighbor);

                    }
                }
            }

            sort(willPushBuffer.begin(), willPushBuffer.end());
            for (int i = 0; i < willPushBuffer.size(); i++) {
                q.push(willPushBuffer[i]);
            }
```

```
        }

```

而至于第二题，需要的数据如下：

```
    graph.add_edge_1Base(1, 5);
    graph.add_edge_1Base(1, 6);
    graph.add_edge_1Base(1, 12);
    graph.add_edge_1Base(2, 5);
    graph.add_edge_1Base(2, 9);
    graph.add_edge_1Base(2, 3);
    graph.add_edge_1Base(3, 6);
    graph.add_edge_1Base(3, 7);
    graph.add_edge_1Base(3, 10);
    graph.add_edge_1Base(4, 3);
    graph.add_edge_1Base(4, 7);
    graph.add_edge_1Base(4, 14);
    graph.add_edge_1Base(5, 8);
    graph.add_edge_1Base(6, 9);
    graph.add_edge_1Base(6, 13);
    graph.add_edge_1Base(7, 6);
    graph.add_edge_1Base(9, 8);
    graph.add_edge_1Base(10, 11);
    graph.add_edge_1Base(10, 12);
    graph.add_edge_1Base(11, 14);
    graph.add_edge_1Base(13, 10);
```

这个是对于第一题、第二题分别输出的结果，可以看到第一题是严格的优先输出数字比较小的节点。但是第二题就宽松了一些