/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Topological Sorting

O(|V| + |E|)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

（涉及图片均来自Wikipedia）

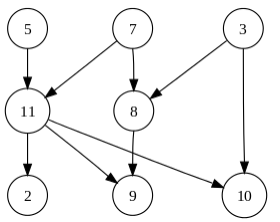
拓扑排序， 对有前置关系的任务序列进行排序， 得到可行的处理顺序。

流程模拟：

（1） 取出入度为0的点

（2）删除相关边

（3） 若仍有入度为0的点，回到（1）， 否则结束



对右图

1、取出入度为0（即作为边终点的次数为0）的点 5 7 3

2、删除相关边

3、取出入度为0的点11 8

4、删除相关边

5、取出入度为0的点2 9 10

6、结束

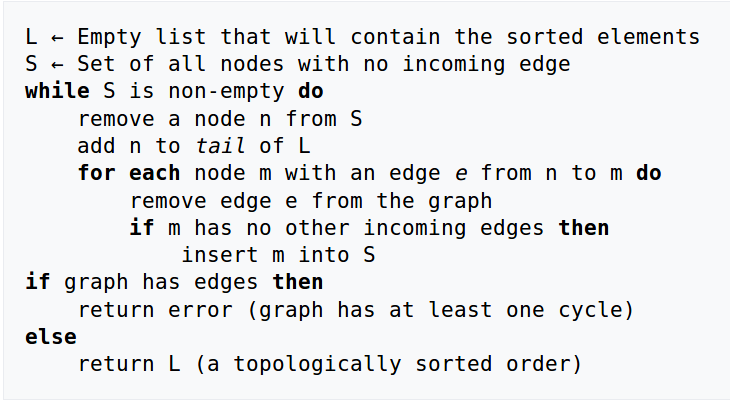
得到的序列为5，7，3，11，8，2，9，10

（序列非唯一， 可根据不同的条件得到不同的序列，如优先处理序号小的点可得：3, 5, 7, 8, 11, 2, 9, 10）

实现方式：

**Kahn‘s Algorithm:**

基于BFS， 伪代码如下：



Code：

int n;

vector<int> e[MAX]; // 邻接表存边

int deg[MAX]; // 入度

vector<int> List;

bool topological()

{

queue<int> q;

/\* 入度为0的点进队 \*/

for (int i = 1; i <= n; i++)

if (deg[i] == 0)

q.push(i);

while (!q.empty())

{

int u = q.front();

q.pop();

List.push\_back(u); // 记录序列

/\* 删边， 新增的入度为0的点进队 \*/

for (auto v : e[u])

{

deg[v]--;

if (!deg[v])

q.push(v);

}

}

/\* 扫描是否仍有入度不为0的点，若有则成环 \*/

for (int i = 1; i <= n; i++)

if (deg[v])

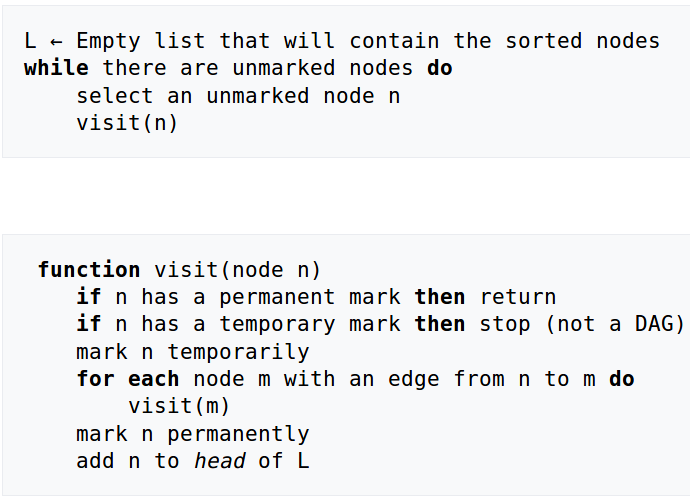
return false;

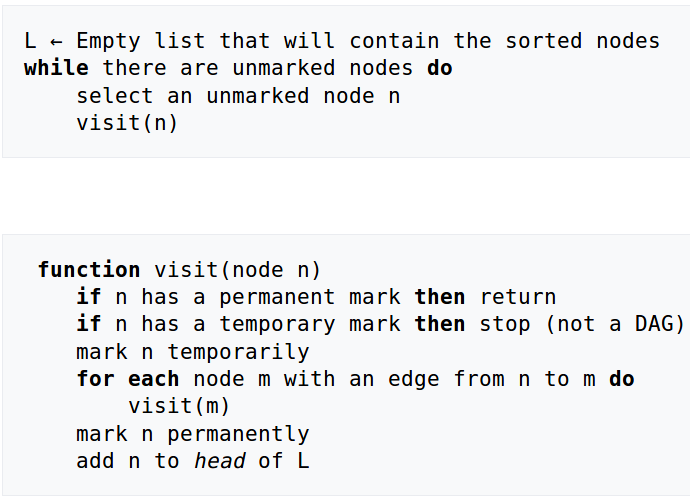
return true;

}

### Depth-first search:

伪代码：

 （visit函数在下一页=\_=)



Code：

int n;

vector<int> e[MAX];

int vis[MAX]; // 标记, 1为临时标记， 2为正式标记

bool topological(int u)

{

vis[u] = 1;

for (auto v : e[u])

if (vis[v] == 1 || (vis[v] == 0 && topological(v))

return false;

vis[u] = 2;

return true;

}