第 8 章

图应用

李荣华

北京理工大学

提纲

8.1 拓扑排序：问题引入

问题：

- 针对规模大且难度高的任务，通常会把大任务分解成多个规模小且简单的子任务，然后按照一定的顺序逐个解决所有的子任务（分治法思想、序列化方法）。
- 在所有子任务中，有些子任务之间是独立的关系，即没有先后之别，但有些子任务之间却存在依赖关系。
- 因此对所有子任务序列化时，如果子任务**b**依赖于子任务**a**，则**a**必须安排在**b**的前面（可以不连续！），如何合理安排所有子任务的次序?

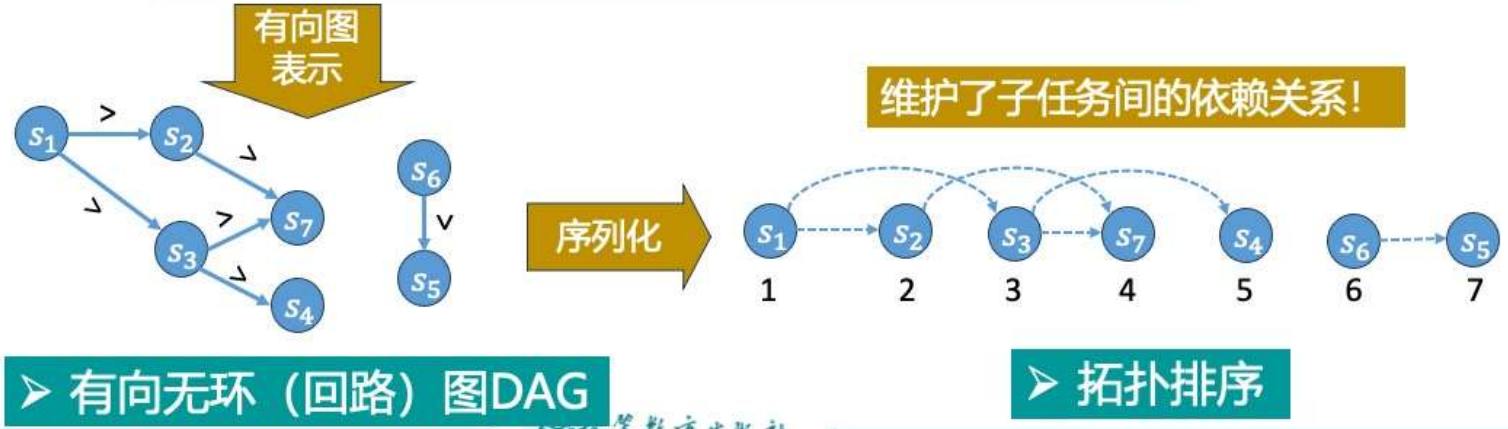
- 用 a>b 表示 b依赖于a
- 依赖关系满足传递性：即 如果 a>b 并且 b>c, 则 a>c
- 依赖关系无“矛盾”：即 a>b 和 b>a 不能同时成立!

8.1 拓扑排序：问题引入

问题：

- 对大规模且难度高的任务，通常会把大任务分解成众多规模小且简单的子任务，然后按照一定的顺序逐个解决所有的子任务（分治法思想、序列化方法）。
- 在所有子任务中，有些子任务之间是独立的，即没有先后之别，但有些子任务之间却存在依赖关系。
- 因此对所有子任务序列化时，如果一个子任务依赖于另一个任务，则后者必须安排在前者的前面（可以不连续！），如何合理安排所有子任务的次序?

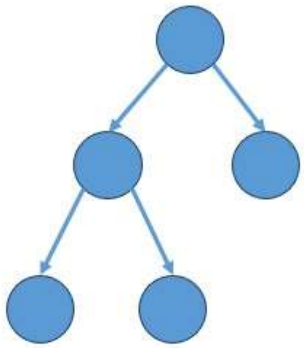
例： 任务 $S=\{s_1,s_2,s_3,s_4,s_5,s_6,s_7\}$
依赖关系： $s_1>s_3, s_2>s_7, s_1>s_2, s_3>s_4, s_6>s_5, s_3>s_7$



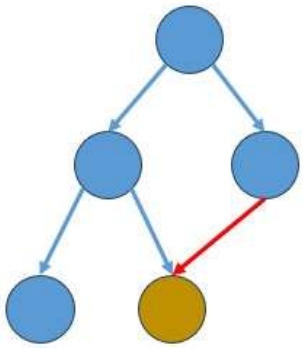
➢ 有向无环（回路）图DAG

➢ 拓扑排序

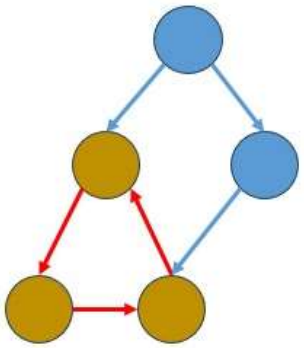
树、有向无环图与有向图



（有向）树



有向无环图DAG



有向图

多选题 1分

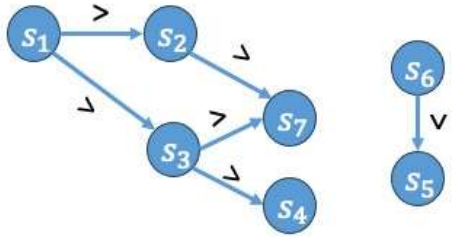
对右边的DAG序列化（拓扑排序），下面哪些序列是正确的排序结果

A <s_1, s_3, s_2, s_4, s_7, s_6, s_5>

B <s_1, s_3, s_6, s_4, s_2, s_7, s_5>

C <s_1, s_2, s_6, s_7, s_5, s_3, s_4>

D <s_1, s_6, s_2, s_3, s_7, s_5, s_4>



8.1 拓扑排序示例：课程安排

问题：普通高等学校计算机专业本科生的课程安排，某些课程存在先修课，如何确定一个合理的课程学习的顺序？

关键：课程学习顺序需要遵循课程之间的先修关系

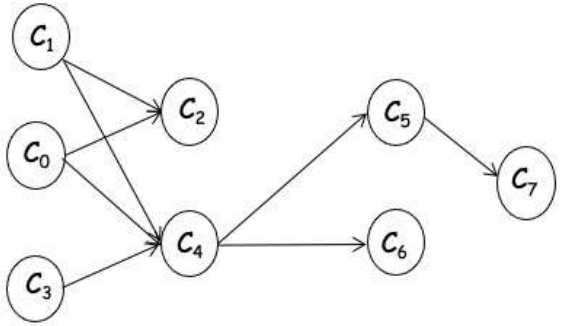
课程先修关系：

课程名称	课程代码	先修课程
微积分	C_0	无
线性代数	C_1	无
概率论与数理统计	C_2	C_0, C_1
程序设计基础	C_3	无
数据结构	C_4	C_0, C_1, C_3
计算机组成原理	C_5	C_4
操作系统	C_6	C_4
计算机体系结构	C_7	C_5

问题建模

使用有向图表示课程关系：将课程视为顶点，将课程间的先修关系视为有向边

课程名称	微积分	线性代数	概率论与数理统计	程序设计基础	数据结构	计算机组成原理	操作系统	计算机体系结构
课程代码	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
先修课程	无	无	C_0, C_1	无	C_0, C_1, C_3	C_4	C_4	C_5

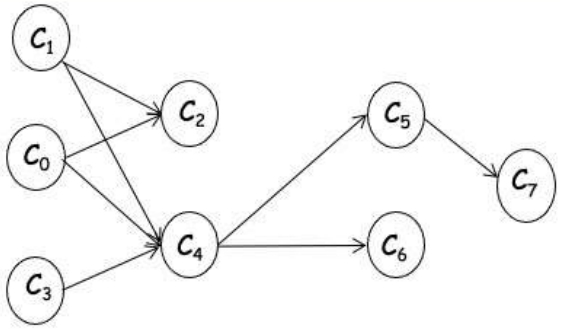


课程先修关系图

问题建模

有向无环图 不存在环路的有向图称为有向无环图（Directed Acyclic Graph，DAG）。由于课程先修关系不会出现闭环，因此该图一定是有向无环图。

AOV网络 在一个有向图中，如果图中的顶点表示活动，图的有向边表示活动之间的优先关系，则称这样的有向图为AOV网络（Activity On Vertex Network）。

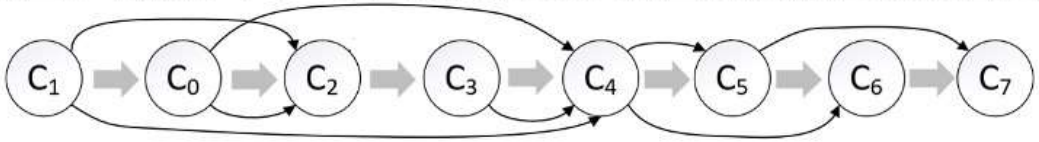


课程先修关系图

问题求解：拓扑排序

问题分析

在一个合法的课程学习顺序中，每个课程的先修课程必须排在该课程之前。



一个合法的课程学习顺序

拓扑序列&拓扑排序

在DAG中，将满足下列条件的顶点序列称为**拓扑序列**：

（1）每个顶点在序列中出现且仅出现一次；

（2）序列中的每个顶点都不存在边指向其前面的结点。

定理：DAG中一定存在拓扑序列，存在拓扑序列的AOV网络一定是DAG。

将求解拓扑序列的算法称为**拓扑排序**。

问题求解：拓扑排序

拓扑序列&拓扑排序

在DAG中，将满足下列条件的顶点序列称为拓扑序列：

（1）每个顶点在序列中出现且仅出现一次；

（2）序列中的每个顶点都不存在边指向其前面的节点。

DAG的重要性质

① DAG中一定有一个或多个入度为0的结点（思考），拓扑序列的先头结点一定入度为0，并且任何入度为0的结点都可以安排在拓扑序列的先头

→ BFS算法

② 同样，DAG必含一个或多个出度为0的结点（思考），拓扑序列末尾的结点一定出度为0，并且任何度为0的结点都可以出现在拓扑序列的末尾

→ DFS算法

问题求解：拓扑排序的BFS算法

思路：当一个结点所依赖的所有结点都已被访问并放入拓扑序列中时，**随时可以**将该结点添加至拓扑序列

关键步骤：每处理完一个结点，将其所有邻接结点的入度减1，因此入度为0表示该结点无依赖结点或者所有依赖结点已处理完毕！

关键数据结构：用队列存放入度为0的结点

问题求解：拓扑排序的BFS算法

BFS算法：

(1) 计算DAG所有结点的入度，将所有入度为0的结点放入队列

(2) 从队列先头取出结点，添加至拓扑序列，从图中删除结点及其所有出边

(option)

(3) 将上述结点的所有邻接结点的入度减1，如果出现入度为0的结点，则入队

(4) 如果队列非空，重复(2)的操作；否则结束排序

时间复杂度： $O(|V|+|E|)$

若执行操作(4)后所有顶点均被删除，则输出的是合法的拓扑序列；否则说明原图存在环（回路？），无合法拓扑序列。

问题求解：拓扑排序的BFS算法

算法: 计算有向图中各结点的入度 CountInDegree(*graph*, *n*, *indegree*)
输入： 图*graph*，结点数*n*，顺序表*indegree*[1..*n*], 记录结点入度
输出： 图*graph*中各结点的入度

```
1. for i ← 1 to n do
2. |  indegree[i] ← 0   //入度的初始值
3. end
4. for i ← 1 to n do
5. |  p ← graph.ver_list[i].adj
6. |  while p ≠ NIL do
7. |  |  indegree[p.dest] ← indegree[p.dest] + 1  //邻接结点的入度加1
8. |  end
9. end
```

时间复杂度：O(|V|+|E|)

问题求解：拓扑排序的BFS算法

算法： TopSort-BFS(*dag, n, top_list*)
输入： 有向无环图*dag*，结点数*n*
输出： 图*dag*的拓扑序列*top_list*

时间复杂度：O(|V|+|E|)

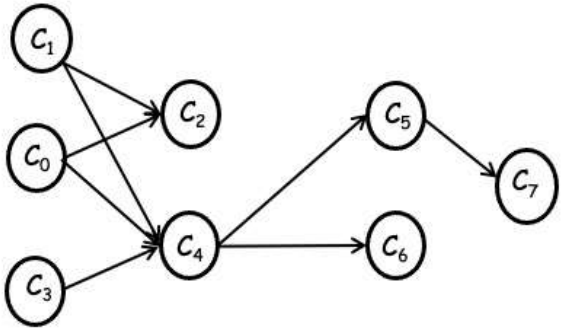
思考：算法中没有删除结点与边，因此如何判断有向图中是否有环（回路）？

```
1.  InitList(indegree)  //记录各结点入度的顺序表
2.  CountInDegree(dag, n, indegree)
3.  InitQueue(queue)
4.  for i ← 1 to n do
5.  |  if indegree[i] = 0 then
6.  |  |  EnQueue(queue, i)        //入度为0的结点入队
7.  |  end
8.  end
9.  while IsEmpty(queue) = false do
10. |  u ← DeQueue(queue)
11. |  Append(top_list, u)          //结点添加至拓扑序列
12. |  p ← dag.ver_list[u].adj
13. |  while p ≠ NIL do            //遍历邻接结点
14. |  |  indegree[p.dest] ← indegree[p.dest] – 1 //邻接结点入度减1
15. |  |  if indegree[p.dest] = 0 then  //如果邻接结点入度降为0
16. |  |  |  EnQueue(queue, p.dest)    //邻接结点入队
17. |  |  end
18. |  end
19. end
```

问题求解：拓扑排序的BFS算法

queue:

课程先修关系图



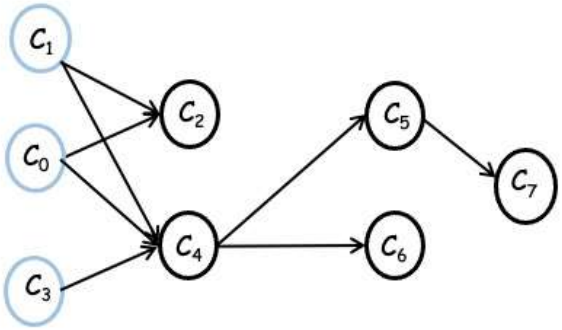
indegree	0	0	0	0	0	0	0	0
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list:

问题求解：拓扑排序的BFS算法

queue: C_0, C_1, C_3

课程先修关系图



indegree	0	0	2	0	3	1	1	1
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

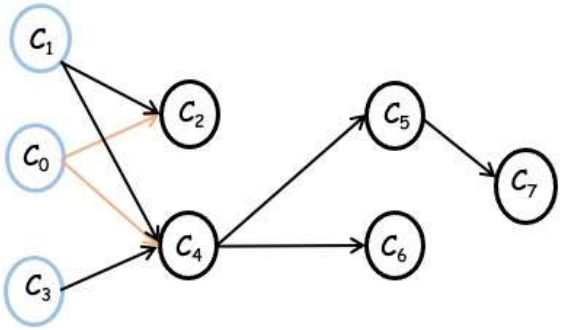
计算各结点入度

top_list:

问题求解：拓扑排序的BFS算法

queue: C_1, C_3

课程先修关系图



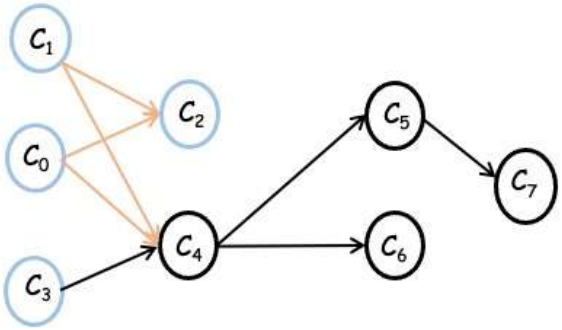
indegree	0	0	1	0	2	1	1	1
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0

问题求解：拓扑排序的BFS算法

queue: C_3, C_2

课程先修关系图



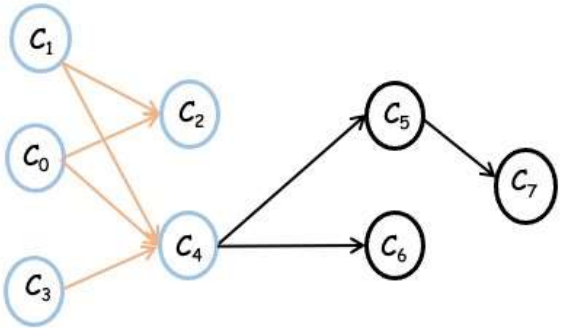
indegree	0	0	0	0	1	1	1	1
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_1

问题求解：拓扑排序的BFS算法

queue: C_2, C_4

课程先修关系图



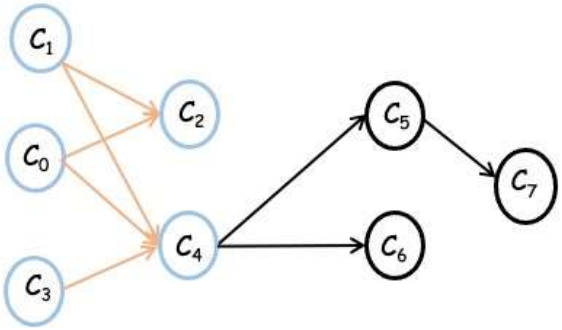
indegree	0	0	0	0	0	1	1	1
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_1 -> C_3

问题求解：拓扑排序的BFS算法

queue: C_4

课程先修关系图



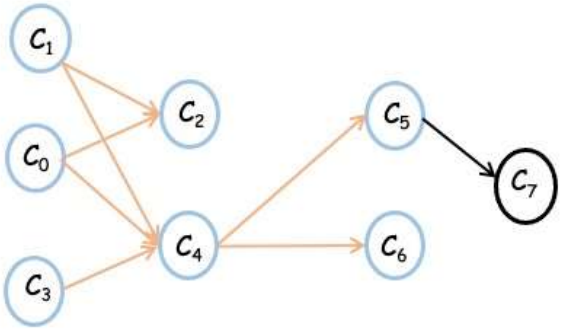
indegree	0	0	0	0	0	1	1	1
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_1 -> C_3 -> C_2

问题求解：拓扑排序的BFS算法

queue: C_5, C_6

课程先修关系图



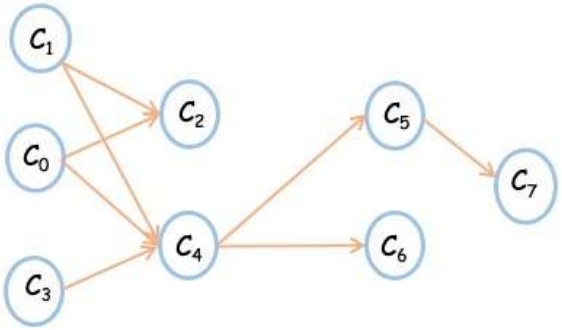
indegree	0	0	0	0	0	0	0	1
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_1 -> C_3 -> C_2 -> C_4

问题求解：拓扑排序的BFS算法

queue: C_6, C_7

课程先修关系图



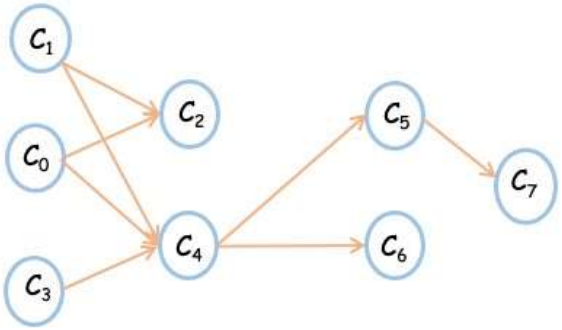
indegree	0	0	0	0	0	0	0	0
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_1 -> C_3 -> C_2 -> C_4 -> C_5

问题求解：拓扑排序的BFS算法

queue:

课程先修关系图



indegree	0	0	0	0	0	0	0	0
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_1 -> C_3 -> C_2 -> C_4 -> C_5 -> C_6 -> C_7

问题求解：拓扑排序的BFS算法

拓扑序列：C_0 -> C_1 -> C_3 -> C_2 -> C_4 -> C_5 -> C_6 -> C_7

课程名称	课程代码	先修课程	课程学习顺序
微积分	C_0	无	1
线性代数	C_1	无	2
概率论与数理统计	C_2	C_0, C_1	4
程序设计基础	C_3	无	3
数据结构	C_4	C_0, C_1, C_3	5
计算机组成原理	C_5	C_4	6
操作系统	C_6	C_4	7
计算机体系结构	C_7	C_5	8

问题求解：拓扑排序的DFS算法

DFS算法：

(1) 从DAG中的某一个未访问过的结点出发，访问并对该结点加已访问标志

(2) 如果该顶点无邻接结点（出度为0）或者所有邻接结点访问完毕，输出结点并返回；否则找到未被访问过的邻接结点，执行步骤(1)

(3) 如果还有结点未被访问过，选中其中一个结点作为起始顶点，再次转向(1)。如果所有的结点都被访问到，遍历结束

时间复杂度： $O(|V|+|E|)$

- 在基于BFS的算法中，如果图中有环（回路），则构成环的所有结点无法输出！
- **思考**：如果有向图有环，DFS算法能否输出环上的结点?

问题求解：拓扑排序的DFS算法

算法： TopSort-DFS(*dag, n, top_list*)
输入： 有向无环图*dag*，结点数*n*
输出： 图*dag*的拓扑序列top_list
关键数据结构： 顺序表visited

```
1. for v ← 1 to n do //初始化各顶点的已访问标志为未访问
2. |   visited[v] ← false
3. end
4. for v ← 1 to n do
5. |   if visited[v] = false then
6. |   |   DFS-Sort(dag, v, visited, top_list)
7. |   end
8. end
```

问题求解：拓扑排序的DFS算法

算法： DFS-Sort(*dag, v, visited, top_list*)

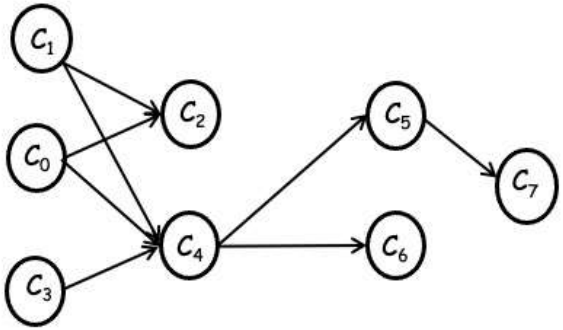
```
1.  visited[v] ← true   //标记结点
2.  p ← dag.ver_list[v].adj
3.  while p ≠ NIL do
4.  |  if visited[p.dest] = false then          //邻接结点未访问
5.  |  |  DFS-Sort(graph, p.dest, visited, top_list)  //递归排序邻接结点
6.  |  end
7.  |  p ← p.next   //继续遍历下一个邻接结点
8.  end          //无邻接结点（出度0）或所有邻接结点已标记
9.  Insert(top_list, 1, v)  //结点插入拓扑序列的头位置（？）
```

时间复杂度：O(|V|+|E|)

思考：如何判断有向图中是否有环（回路）？

问题求解：拓扑排序的DFS算法

课程先修关系图

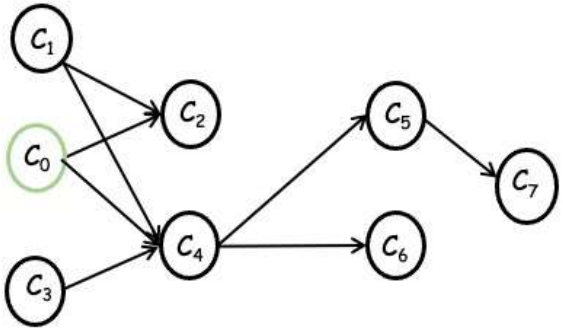


visited	F	F	F	F	F	F	F	F
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list:

问题求解：拓扑排序的DFS算法

课程先修关系图

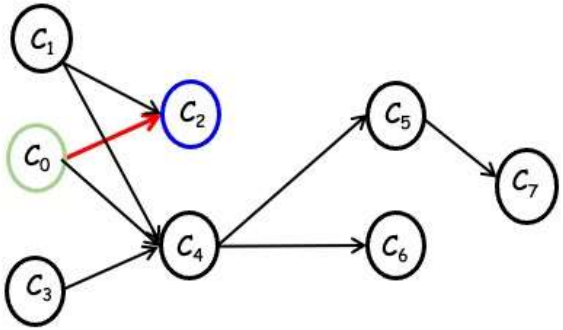


visited	T	F	F	F	F	F	F	F
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list:

问题求解：拓扑排序的DFS算法

课程先修关系图

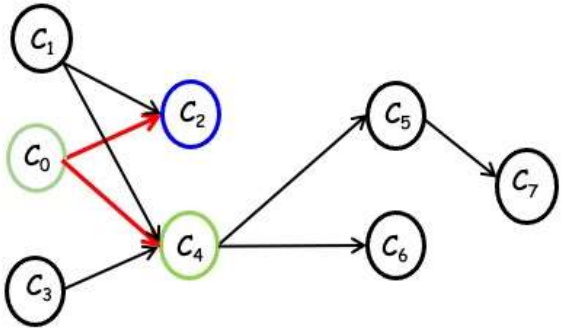


visited	T	F	T	F	F	F	F	F
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list：C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

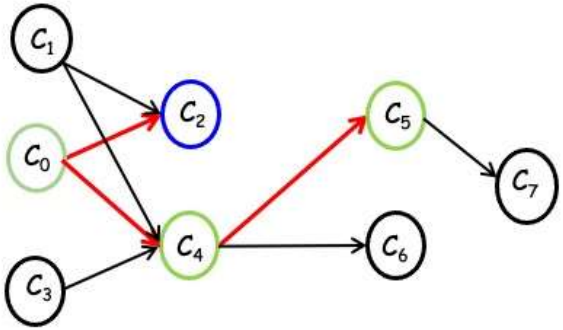


visited	T	F	T	F	T	F	F	F
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list：C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

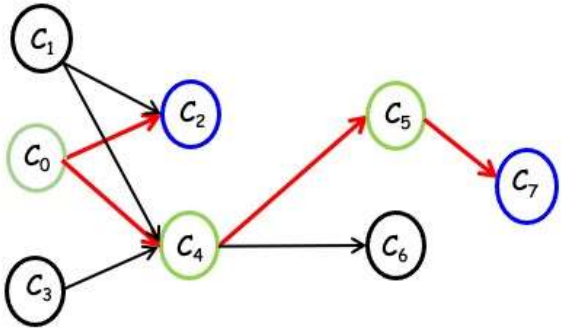


visited	T	F	T	F	T	T	F	F
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

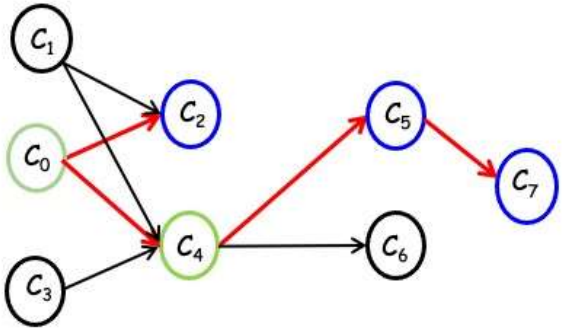


visited	T	F	T	F	T	T	F	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

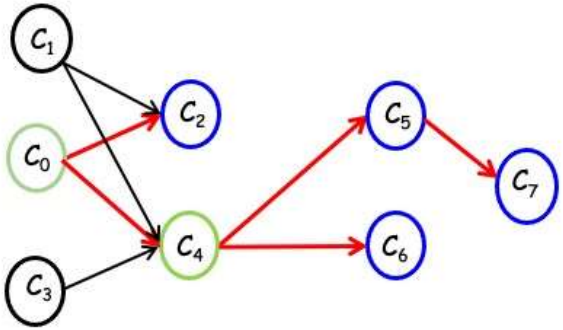


visited	T	F	T	F	T	T	F	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_5 -> C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

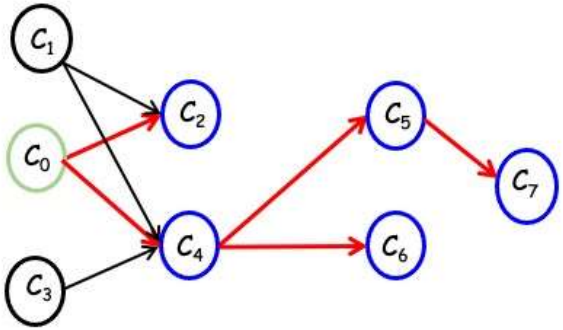


visited	T	F	T	F	T	T	T	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_6 -> C_5 -> C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

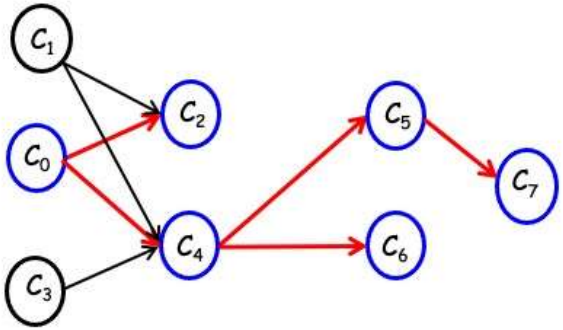


visited	T	F	T	F	T	T	T	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_4 -> C_6 -> C_5 -> C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

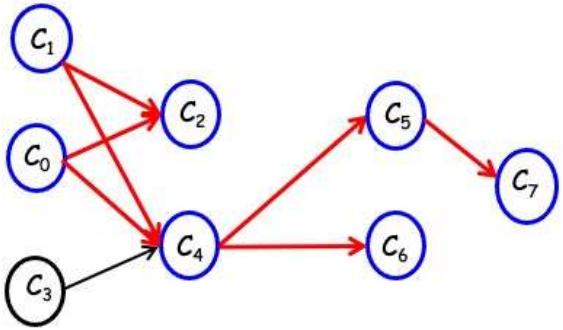


visited	T	F	T	F	T	T	T	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_0 -> C_4 -> C_6 -> C_5 -> C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图

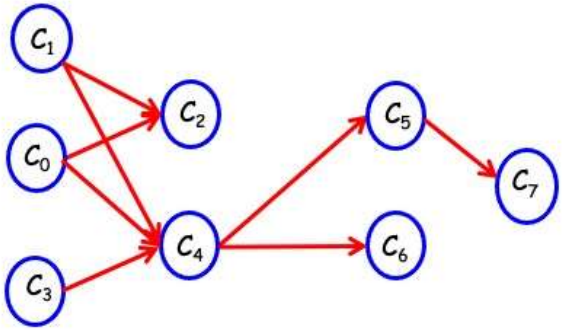


visited	T	T	T	F	T	T	T	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_1 -> C_0 -> C_4 -> C_6 -> C_5 -> C_7 ->C_2

问题求解：拓扑排序的DFS算法

课程先修关系图



visited	T	T	T	T	T	T	T	T
	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7

top_list: C_3 -> C_1 -> C_0 -> C_4 -> C_6 -> C_5 -> C_7 ->C_2

问题求解：拓扑排序的BFS算法

DFS拓扑序列： C_3 -> C_1 -> C_0 -> C_4 -> C_6 -> C_5 -> C_7 ->C_2

课程名称	课程代码	先修课程	课程学习顺序
微积分	C_0	无	3
线性代数	C_1	无	2
概率论与数理统计	C_2	C_0, C_1	8
程序设计基础	C_3	无	1
数据结构	C_4	C_0, C_1, C_3	4
计算机组成原理	C_5	C_4	6
操作系统	C_6	C_4	5
计算机体系结构	C_7	C_5	7

BFS拓扑序列：C_0 -> C_1 -> C_3 -> C_2 -> C_4 -> C_5 -> C_6 -> C_7

多选题 1分

如果把二叉树结点间的父子关系表示成从父结点指向子结点的有向边，则二叉树属于有向无环图（思考）。对二叉树拓扑排序，可用下面哪些遍历法实现?

A 前序遍历

B 中序遍历

C 后序遍历

D 层序遍历

谷歌面试题（LeetCode1136): 并行课程

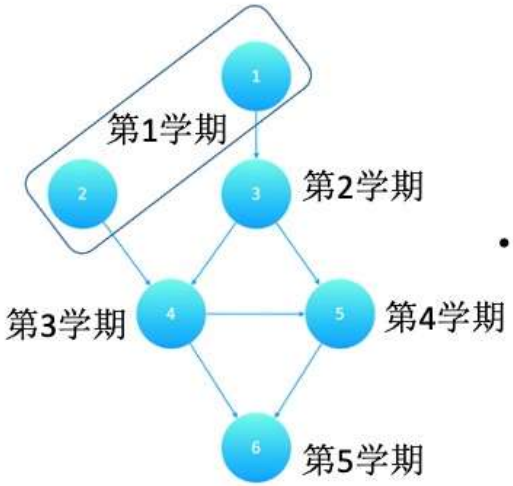
问题描述：给你一个整数 n ，表示编号从 1 到 n 的 n 门课程。另给你一个数组 relations，其中 relations[i] = [prevCourse, nextCourse] ，表示课程 prevCourse 和课程 nextCourse 之间存在先修关系：课程 prevCourse 必须在 nextCourse 之前修读完成。

- 在一个学期内，你可以学习 **任意数量** 的课程，但前提是你已经在上一学期修读完待学习课程的所有先修课程。
- 请你返回学完全部课程所需的 **最少** 学期数。如果没有办法做到学完全部这些课程的话，就返回 -1。

谷歌面试题（LeetCode1136): 并行课程

问题描述：给你一个整数 n ，表示编号从 1 到 n 的 n 门课程。另给你一个数组 relations，其中 relations[i] = [prevCourse, nextCourse] ，表示课程 prevCourse 和课程 nextCourse 之间存在先修关系：课程 prevCourse 必须在 nextCourse 之前修读完成。

- 在一个学期内，你可以学习 **任意数量** 的课程，但前提是你已经在上一学期修读完待学习课程的所有先修课程。
- 请你返回学完全部课程所需的 **最少** 学期数。如果没有办法做到学完全部这些课程的话，就返回 -1。



- **思路**：使用BFS算法拓扑排序（DAG层序遍历）
- **关键数据结构**：顺序表sem[1..n], 每个课程最早可学完的学期
- **关键步骤**：用每门出队的课程更新其后继课程的学习学期

谷歌面试题（LeetCode1136): 并行课程

时间复杂度：O(|V|+|E|)

思考：可否用DFS算法解决问题?

```
1.  for i ← 1 to n do                    //计算入度等操作省略
2.  | if indegree[i] = 0 then
3.  | | EnQueue(queue, i)                //无先修课程的课程入队
4.  | | sem[i] ← 1                       //全部安排在第一学期
5.  | end
6.  end
7.  count ← 0    //计数可安排（拓扑排序）的课程数
8.  while IsEmpty(queue) = false do
9.  | u ← DeQueue(queue)
10. | count ← count + 1                  //可安排（排序）的课程数加1
11. | last_sem ← sem[u]                  //按出队序记录每门课程学完的学期
12. | p ← graph.ver_list[u].adj
13. | while p ≠ NIL do                   //遍历后续课程
14. | | term[p.dest] ← term[u] + 1        //更新后续课程开始学习的学期
15. | | indegree[p.dest] ← indegree[p.dest] – 1
16. | | if indegree[p.dest] = 0 then
17. | | | EnQueue(queue, p.dest)
18. | | end
19. | end
20. end
21. if count < n then    //图中有回路，部分课程无法安排
22. | last_sem ← -1
23. end
24. return last_sem                      //返回最后一门课程最早学完的学期
```

8.1 最短路径：问题引入

问题：在一个3阶魔方中，魔方的每一个面都由 9 个方块（3×3）构成。魔方处在初始形态时，每个面的方块都是同一种颜色，魔方的六个面一共包含蓝、红、橙、绿、黄、白六种颜色。将魔方的各面随机旋转几次，如何它通过最少的旋转还原到魔方的初始形态，即各个面的方块颜色一样呢?

关键：如何将该问题转化为一个图问题?

核心思想：将魔方的任意一个形态建模为图中的一个顶点。如果魔方的两个形态可以通过魔方一个面的一次旋转相互转化，那么就将相应的两个顶点连成一条边。这样可以得到一个魔方形态的转移图。从一个打乱的魔方还原至初始形态可以转化为在该转移图上寻找一条从打乱状态所对应顶点到初始形态所对应顶点的路径问题。而这两个顶点之间的最短路径即为最优的旋转方案。