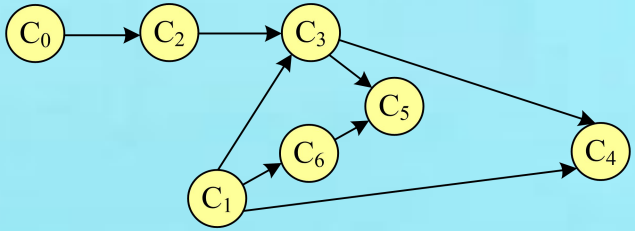
本节主题：

拓扑排序

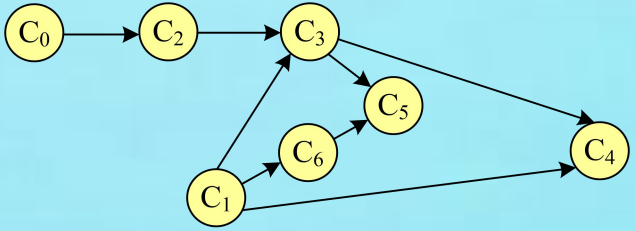
问题描述

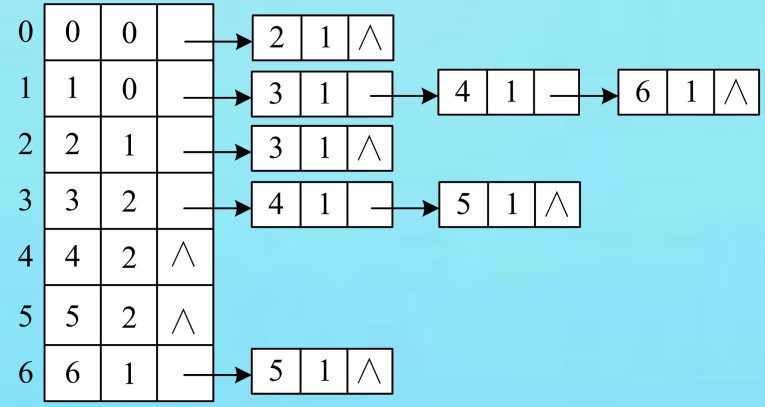
- 定义
  - 设G=(V，E)是一个具有n个顶点的**有向图**
  - V中顶点序列v_1，v_2，…，v_n称为一个**拓扑序列**，当且仅当序列满足：若<v_i,v_j>是图中的边(即从顶点v_i到v_j有一条路径)，则在拓扑序列中顶点v_i必须排在顶点v_j之前
- 例：右图的拓扑序列
  - C_0-C_2-C_1-C_3-C_6-C_5-C_4
  - C_1-C_6-C_0-C_2-C_3-C_4-C_5
  - ……
- 在一个有向图中找一个拓扑序列的过程称为拓扑排序



课程代号	课程名称	先修课程
C_0	高等数学	无
C_1	程序设计	无
C_2	离散数学	C_0
C_3	数据结构	C_1，C_2
C_4	编译原理	C_1，C_3
C_5	操作系统	C_3，C_6
C_6	计算机组成	C_1

拓扑排序中数据的存储结构






```
typedef struct ANode
{
  int adjvex;
  InfoType info;
  struct ANode *nextarc;
} ArcNode; //边表节点类型
typedef struct Vnode
{
  Vertex data;
  int count;     //存放顶点入度
  ArcNode *firstarc;
} VNode;       //表头节点类型
typedef VNode AdjList[MAXV];
typedef struct
{
  AdjList adjlist;
  int n,e;
} ALGraph; //完整的图邻接表类型
```

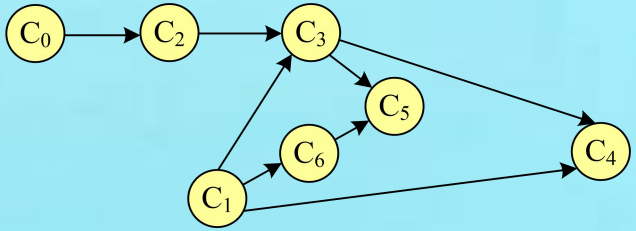
拓扑排序步骤及数据结构

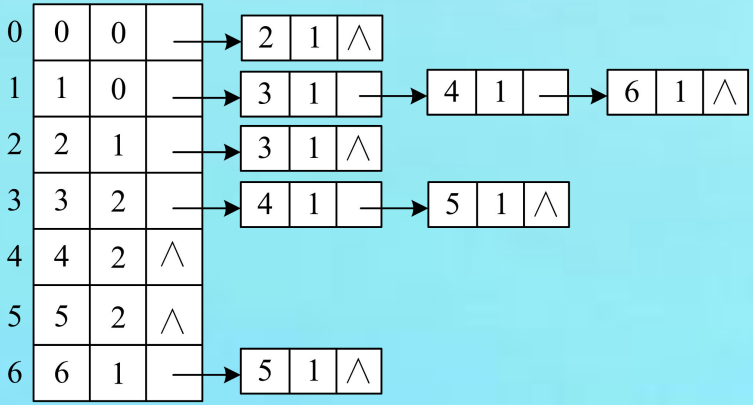
（1）从有向图中选择一个没有前驱（即入度为0）的顶点并且输出它。

（2）从图中删去该顶点，并且删去从该顶点发出的全部有向边。

（3）重复上述两步，直到剩余的网中不再存在没有前驱的顶点为止。

```
int St[MAXV],top=-1;
//定义栈St，记录入度为0的顶点
```





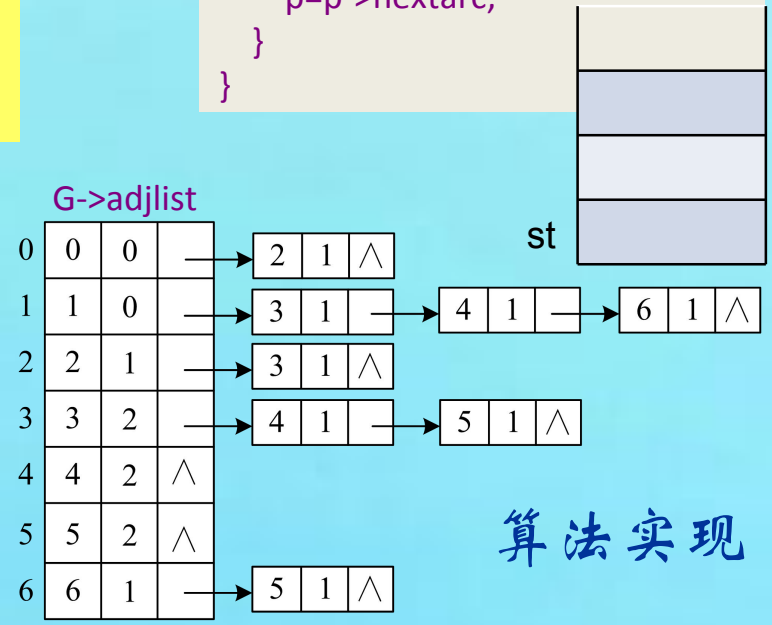
```
void TopSort(ALGraph *G)
{
   int i,j;
   ArcNode *p;
  //求所有顶点的入度
  //入栈度为0的顶点
   while (top>-1) //栈不为空时循环
   {
     //出栈并输出
     //"删除"顶点，即相邻顶点入度均减1
   }
}
```

```
int St[MAXV],top=-1;
for (i=0; i<G->n; i++)
    if (G->adjlist[i].count==0){ top++; St[top]=i; }
```

```
i=St[top];
top--;
printf("%d ",i);
```

```
p=G->adjlist[i].firstarc;
while (p!=NULL)
{
   j=p->adjvex;
   G->adjlist[j].count--;
   if (G->adjlist[j].count==0)  { top++;  St[top]=j;  }
   p=p->nextarc;
}
```

```
for (i=0; i<G->n; i++)
   G->adjlist[i].count=0;
for (i=0; i<G->n; i++)
{
   p=G->adjlist[i].firstarc;
   while (p!=NULL)
   {
      G->adjlist[p->adjvex].count++;
      p=p->nextarc;
   }
}
```



算法实现