

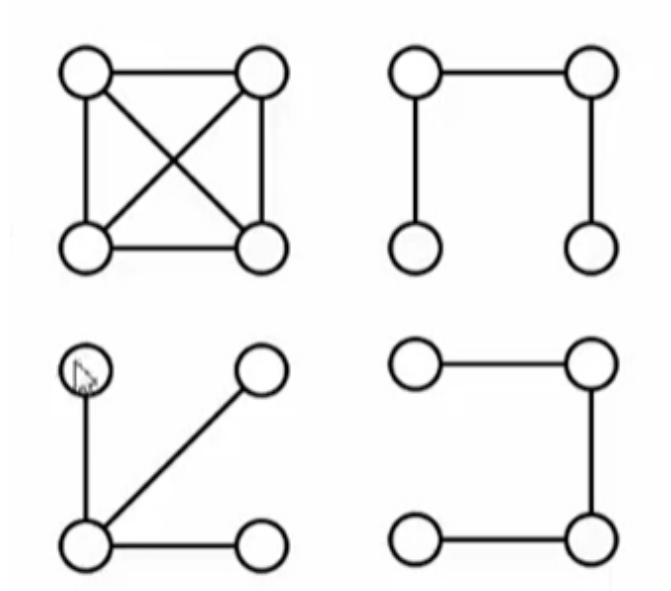
22 第八讲 图（下）

最小生成树问题

什么是最小生成树 (minimum spanning tree)

连通图 和 最小生成树存在 是 充要条件

- 是一棵树
 - 没有回路
 - V 个顶点一定有 $V-1$ 条边
- 是生成树
 - 包含全部顶点
 - $v-1$ 条边都在图里



- 边的权重和最小

贪心算法

- 什么是贪： 每一步都要最好的
- 什么是好： 权重最小的边
- 需要约束

- 只能用图里有的边
- 只能正好用掉 $V - 1$ 条边
- 不能有回路

Prim算法 —— 让一棵小树长大

1957 年，美国计算机科学家罗伯特·普里姆（Robert C. Prim）在独立研究时发现了这个算法，并将其应用于计算机科学领域，尤其是在解决图的最小生成树问题上展现出了很高的效率。由于普里姆的工作，该算法开始在计算机科学和相关工程领域得到广泛传播和应用，因此这个算法被命名为“普里姆算法”。有趣的是，1930 年，捷克数学家沃伊捷赫·亚尔尼克、1959 年，艾兹格·迪科斯彻（Edsger Dijkstra）都在独立研究中发现这一算法。

Prim算法 — 让一棵小树长大

```
void Dijkstra( Vertex s )
{ while (1) {
    V = 未收录顶点中dist最小者;
    if ( 这样的V不存在 )
        break;
    collected[V] = true;
    for ( V 的每个邻接点 W )
        if ( collected[W] == false )
            if ( dist[V]+E<V,W> < dist[W] ){
                dist[W] = dist[V] + E<V,W> ;
                path[W] = V;
            }
    }
}
```

$\text{dist}[V] = E_{(s,V)}$ 或 正无穷
 $\text{parent}[s] = -1$

```
void Prim()
{ MST = {s};
  while (1) {
    V = 未收录顶点中dist最小者;
    if ( 这样的V不存在 )
        break;
    将V收录进MST: dist[V] = 0;
    for ( V 的每个邻接点 W )
        if ( dist[W] != 0 )
            if ( E(V,W) < dist[W] ){
                dist[W] = E(V,W) ;
                parent[W] = V;
            }
    }
    if ( MST中收的顶点不到|V|个 )
        Error ( “生成树不存在” );
}
```

$$T = O(|V|^2)$$

稠密图合算



Copyright © 2014, 浙江大学计算机科学与技术学院
 All Rights Reserved

Dijkstra 算法和 Prim 算法虽然在实现结构上高度相似，但它们的命名差异源于以下三方面的本质区别：

1. 算法目标的根本差异

1. Dijkstra 算法的核心目标是单源最短路径问题
2. Prim 算法的目标是最小生成树问题

2. 历史发展的独立路径

1. Dijkstra 算法：由荷兰计算机科学家 Edsger Dijkstra 于 1956 年提出，最初用于解决荷兰国家银行的电信网络路由问题

2. Prim 算法：最早由捷克数学家 Vojtěch Jarník 于 1930 年发现，但未被广泛关注。1957 年，美国计算机科学家 Robert C. Prim 在独立研究中重新发现并推广了该算法，用于生成最小生成树
3. 数学模型的本质区别
 1. Dijkstra 的松弛操作：在更新节点距离时，需累加源点到当前节点的路径长度（如 $\text{dis}[v] = \min(\text{dis}[v], \text{dis}[u] + \text{weight}(u, v))$ ）
 2. Prim 的边权选择：每次选择的边权仅需比较当前节点到生成树的最短边（如 $\min(\text{weight}(u, v))$ ），**无需考虑路径累加**

Kruskal算法 —— 将森林合并成树

Kruskal 算法是由美国数学家、统计学家、计算机科学家和心理测量学家约瑟夫·伯纳德·克鲁斯卡尔（Joseph Bernard Kruskal）在 1956 年提出的。当时，克鲁斯卡尔在研究图论相关问题时，为了解决最小生成树问题，提出了这一经典算法。在 Kruskal 算法提出后，它在计算机科学、运筹学、电信网络设计等领域得到了广泛应用。例如，在电信网络中，该算法可用于确定如何以最小的成本连接各个基站，以实现网络覆盖。

Kruskal算法 — 将森林合并成树

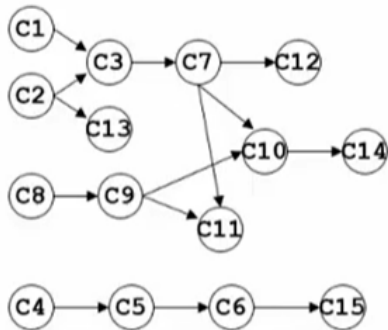
```
void Kruskal ( Graph G )
{
    MST = { } ;
    while ( MST 中不到 |V| -1 条边 && E 中还有边 ) {
        从 E 中取一条权重最小的边  $E_{(v,w)}$  ; /* 最小堆 */
        将  $E_{(v,w)}$  从 E 中删除;
        if (  $E_{(v,w)}$  不在 MST 中构成回路 ) /* 并查集 */
            将  $E_{(v,w)}$  加入 MST;
        else
            彻底无视  $E_{(v,w)}$  ;
    }
    if ( MST 中不到 |V| -1 条边 )
        Error ( “生成树不存在” );
}
```

$$T = O(|E| \log |E|)$$

拓扑排序 Topological Sorting

把 “Topos” 和 “-logia” 组合起来，“Topology” 的含义就是对物体位置和空间关系的研究。在数学领域，拓扑学是一门专门研究几何图形或空间在连续变形（如拉伸、弯曲，但不包括撕裂或黏合）下保持不变性质的学科

课程号	课程名称	预修课程
C1	程序设计基础	无
C2	离散数学	无
C3	数据结构	C1, C2
C4	微积分（一）	无
C5	微积分（二）	C4
C6	线性代数	C5
C7	算法分析与设计	C3
C8	逻辑与计算机设计基础	无
C9	计算机组成	C8
C10	操作系统	C7, C9
C11	编译原理	C7, C9
C12	数据库	C7
C13	计算理论	C2
C14	计算机网络	C10
C15	数值分析	C6



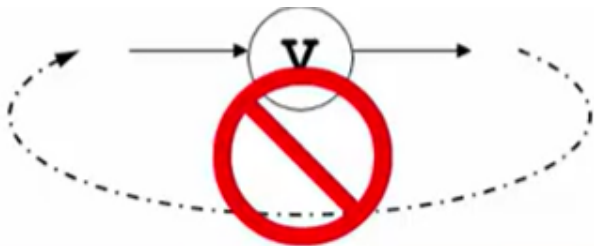
AOV (Activity On Vertex)
网络

定义

拓扑序：如果图中从V到W有一条有向路径，则V一定排在W之前。满足此条件的顶点序列称为一个拓扑序

拓扑排序：获得一个拓扑序的过程就是拓扑排序

AOV如果有合理的拓扑序，则必定是有向无环图（Directed Acyclic Graph, DAG）



v必须在v开始之前结束

算法

在TopSort函数中，如果外循环还没结束，就已经找不到“未输出的入度为0的顶点”，则说明


- ☐ A. 图中必定存在回路
- ☐ B. 图不连通
- ☐ C. 图中可能有回路
- ☐ D. 程序写错啦

拓扑排序的核心是每次选取入度为 0 的顶点输出并删除相关边，适用于有向无环图（DAG）。若外循环未结束就找不到入度为 0 的未输出顶点，说明图中存在回路（环）。因为回路中的顶点相互依赖，入度始终无法变为 0，导致拓扑排序无法正常完成。

- **A. 图中必定存在回路**：正确。回路的存在会使其中顶点入度始终不为 0，导致此情况。
- **B. 图不连通**：错误。图不连通与入度为 0 的顶点是否存在无必然联系，不连通图的各连通分量若为 DAG，仍可拓扑排序。
- **C. 图中可能有回路**：错误。不是“可能”，而是“必定”存在回路。
- **D. 程序写错啦**：错误。这是拓扑排序对有环图的正常表现，非程序错误。

算法

```
void TopSort()
{
    for ( cnt = 0; cnt < |V|; cnt++ ) {
        V = 未输出的入度为0的顶点; /* O(|V|) */
        if ( 这样的V不存在 ) {
            Error ( “图中有回路” );
            break;
        }
        输出V, 或者记录V的输出序号;
        for ( V 的每个邻接点 W )
            Indegree[W]--;
    }
}
```

 $T = O(|V|^2)$

算法优化：随时将入度变为0的顶点放到一个容器里，这样下次执行循环时不用去找哪些顶点入度为0

```

void TopSort()
{ for ( 图中每个顶点 V )
    if ( Indegree[V]==0 )
        Enqueue( V, Q );
    while ( !IsEmpty(Q) ) {
        V = Dequeue( Q );
        输出V, 或者记录V的输出序号; cnt++;
        for ( V 的每个邻接点 W )
            if ( --Indegree[W]==0 )
                Enqueue( W, Q );
    }
    if ( cnt != |V| )
        Error( "图中有回路" );
}

```

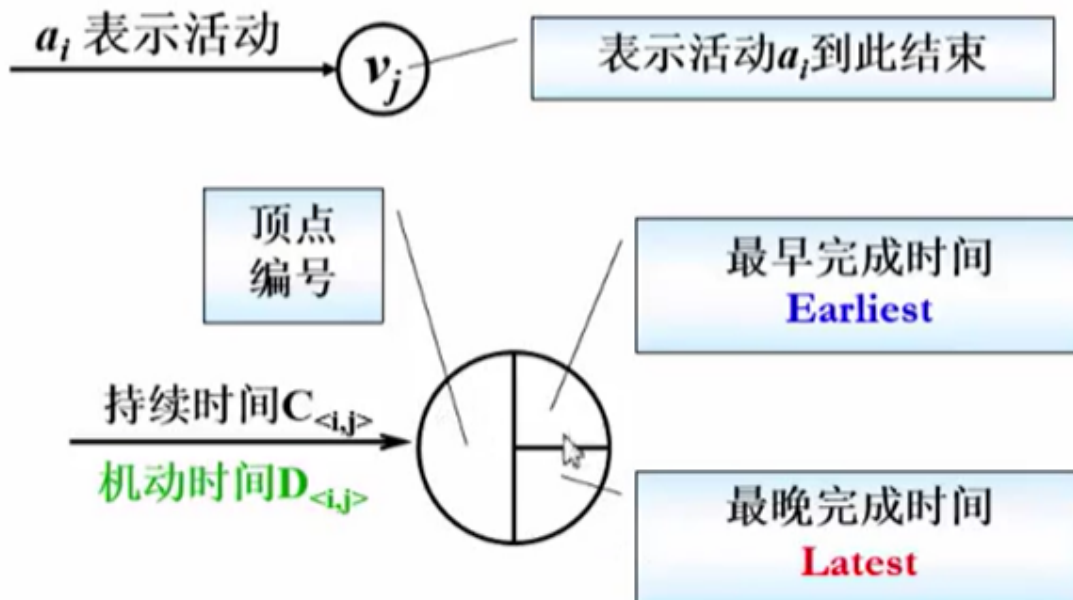
👍 $T = O(|V| + |E|)$

此算法可以用来
检测有向图是否
DAG

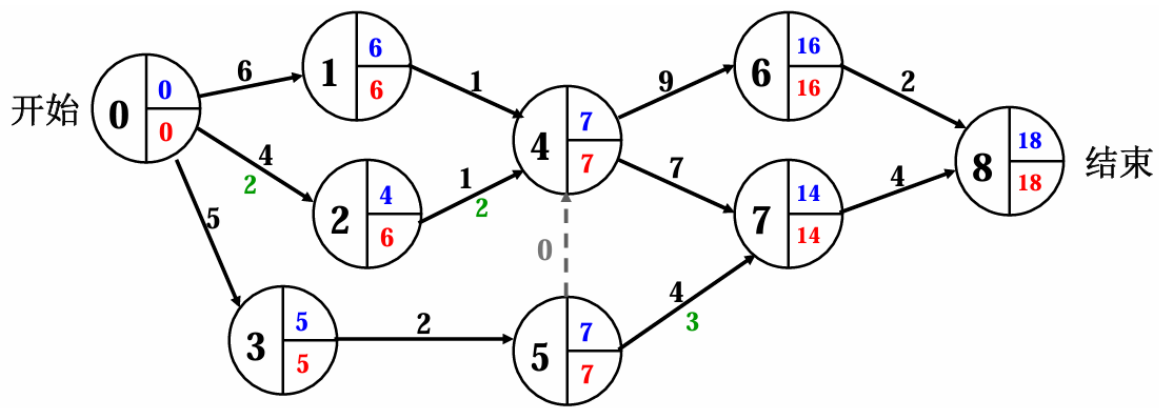
关键路径问题

AOE (Activity On Edge)网络 —— 一般用于安排项目的工序

抽象结构



例子



关键路径：由**绝对不允许延误**的活动组成的路径

在 AOE (Activity On Edge) 网中，关键路径是指从起始顶点到完成顶点的最长路径。关键路径上的活动称为关键活动，这些活动的持续时间直接决定了整个项目的最早完工时间

整个工期有多长：Earliest[8] = 18

哪几个组有机动时间：2、4、7

相关公式

- Earliest
 - Earliest[0] = 0
 -

$$\text{Earliest}[j] = \max_{\langle i,j \rangle \in E} \{ \text{Earliest}[i] + C_{\langle i,j \rangle} \};$$

- Latest
 -

$$\text{Latest}[i] = \min_{\langle i,j \rangle \in E} \{ \text{Latest}[j] - C_{\langle i,j \rangle} \};$$

- 判断是否有延迟
 -

$$D_{\langle i,j \rangle} = \text{Latest}[j] - \text{Earliest}[i] - C_{\langle i,j \rangle}$$

这构成了求关键路径的另外一种算法：

- 若 $D_{\langle i,j \rangle} > 0$ ，说明活动 $\langle i,j \rangle$ 有空闲时间，该活动对应的结点（活动）属于“有空闲的结点”。
- 若 $D_{\langle i,j \rangle} = 0$ ，说明活动 $\langle i,j \rangle$ 无松弛时间，是关键活动。

关键路径由所有 $D_{\langle i,j \rangle} = 0$ 的关键活动组成。由于关键活动没有空闲时间，因此关键路径中不包含上述通过 $D_{\langle i,j \rangle} > 0$ 判定的“有空闲的结点”（活动）。