

Lecture 23: 图的经典应用

23.1 最小生成树

23.1.1 基础概念

最小生成树

- 概念:
 - 对于一个带权连通无向图 $G = (V, E)$, 可以生成多种不同的生成树, 每棵树的权 (即树中所有边上的权值之和) 也可能不同;
 - 设 R 为 G 的所有生成树的集合, 若 T 为 R 中 **边权值之和最小** 的生成树, 则 T 称为 G 的 **最小生成树 (Minimum-Spanning-Tree, MST)**;
- 特点
 - 1. 最小生成树可能有多个, 但边的权值之和总是唯一且最小的;
 - 公式:
$$\text{最小生成树的边数} = \text{顶点数} + 1$$
 - 2. 砍掉条则不连通, 增加一条边则会出现回路;
- 注意:
 - 1. 如果本身就是一棵树, 则其最小生成树就是它本身
 - 2. 只有连通图才有生成树, 非连通图只有生成森林
- 图示:



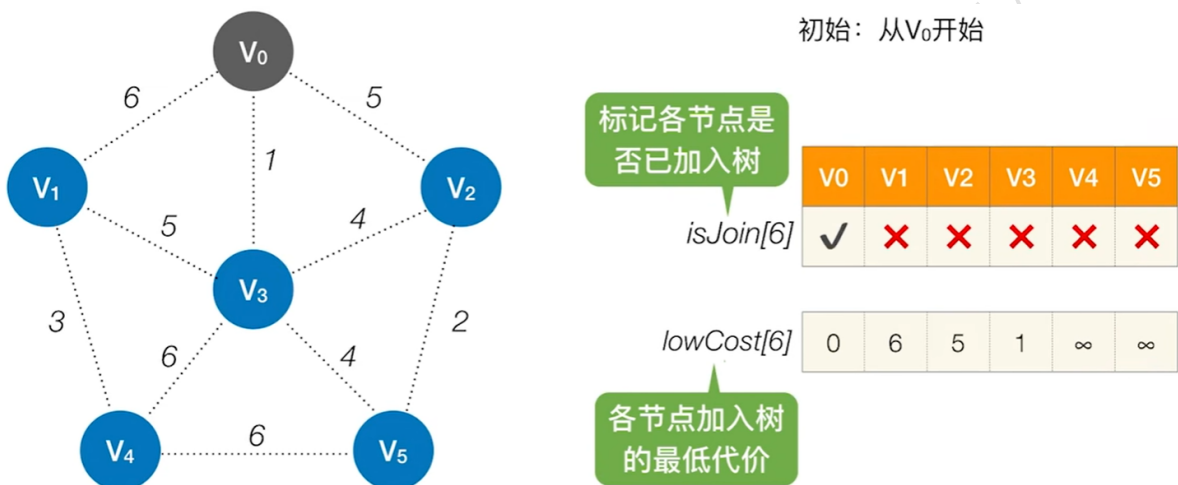
代码实现

```
//通用最小生成树算法
GENERIC_MST(G){
    T=NULL;
    while T 未形成一颗生成树;
        do 找到一条最小代价边(u,v)并且加入T后不会产生回路;
            T=TU(u,v)
}
//利用性质: 最小边一定在树中
//每次加入一条边以逐渐形成一颗生成树
```

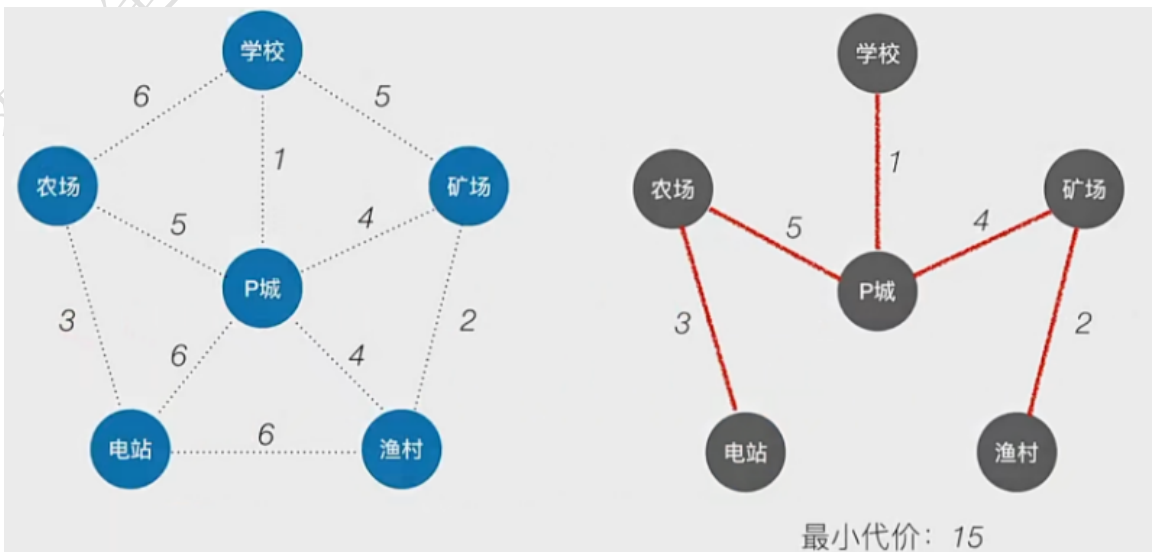
23.1.2 Prim 算法

Prim 算法 (普里姆)

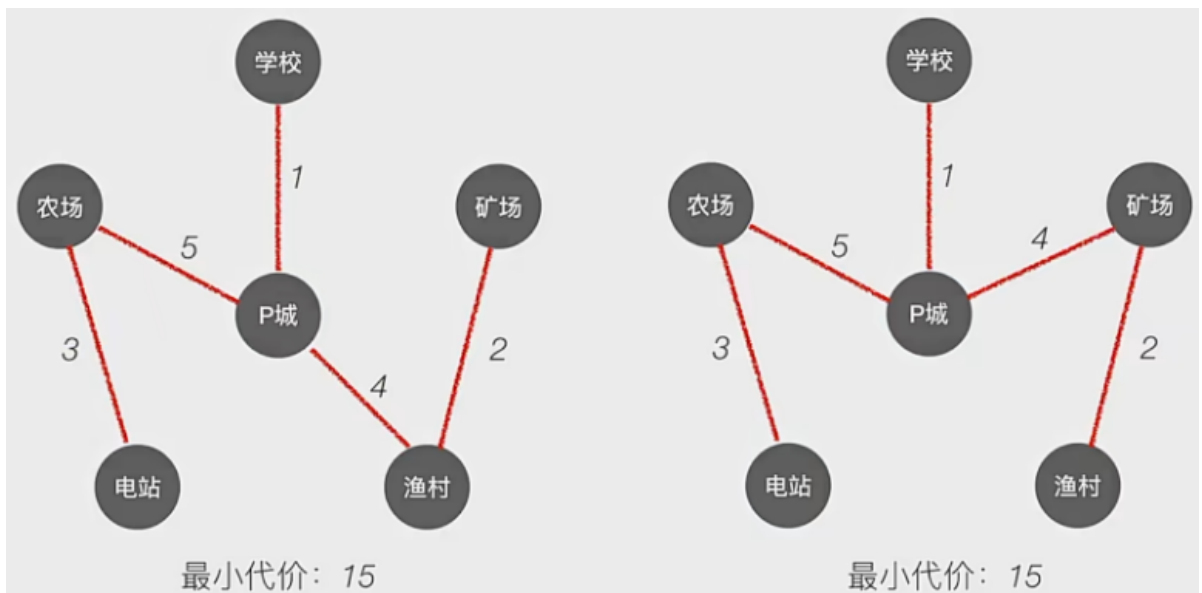
- 概念：
 - 从某个顶点开始构建生成树；
 - 每次将**代价最小的新顶点**纳入生成树，直到所有顶点都纳入为止；
- 性能：
 - 时间复杂度： $O(|V|^2)$
 - 每一轮循环需要 $O(2n)$ 时间复杂度
 - 一共要循环 $(n - 1)$ 次；
 - 只和顶点的数量有关；
 - 适合用于**边稠密图**；
- 图示：



- 过程：
 - 选择 **P城** 结点，其作为当前生成子树，与其相连最低代价为 1，因此将学校通过 1 纳入进来；
 - 和 **P城** 以及 **学校** 相连的节点中，权值最低的是 4，因此将 **矿场** 纳入进来；
 - 与 **P城**、**学校** 以及 **矿场** 相连的边中，2 的代码 最低，因此将渔村纳入进来；
 - 重复以上步骤，直至如下图所示：



- 性质：
 - 同一个图可能有多种最小生成树，但它们的权值相等；



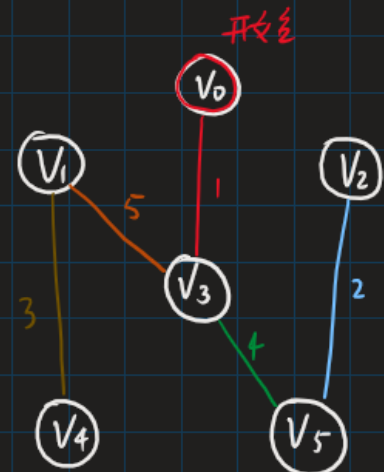
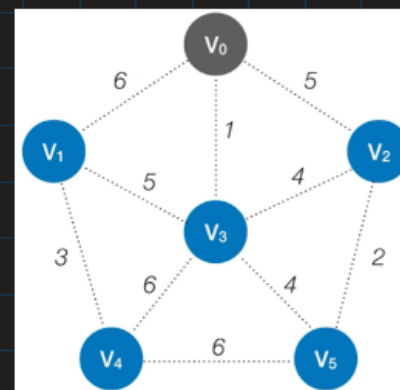
总结

- 假设从 V_0 开始:

方法 - 使用Prim算法获得最小生成树

步骤:

- 1 任选一个顶点作为起始点, 加入顶点集合T;
- 2 选择与顶点集合T相连的代价为 $O(|V|)$ 点当中、边权值最小的一条边, 通过此边和新结点相连;
- 3 重复此步骤, 直到所有顶点都加入T;

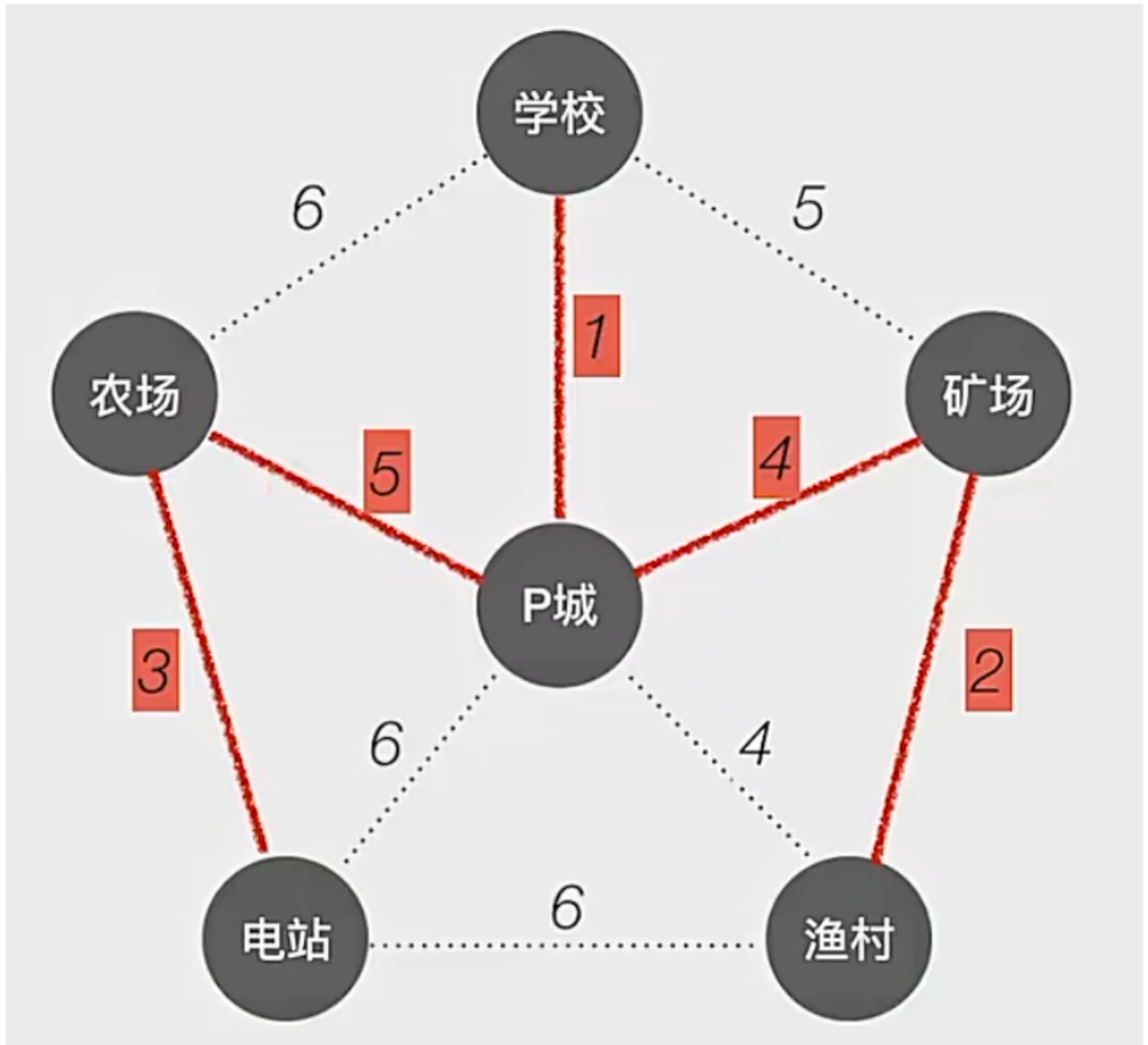


23.1.3 Kruskal 算法

Kruskal 算法 (克鲁斯卡尔)

- 概念:
 - 每次选择一条权值最小的边, 使这条边的两头顶点连通 (原本已经连通的就不选);

- 重复直到所有结点都连通
- 性能：
 - 时间复杂度: $O(|E|\log_2|E|)$
 - 只和边的数量有关;
 - 适合边稀疏的场景;
- 图示:

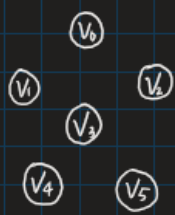


总结

最多要对 $|E|$ 条边作判断
 $O(|E|) \cdot O(\log_2 |E|)$

方法 - 使用 Kruskal 算法构建最小生成树

第一步：画出对应的五个独立的点，作为五个独立的连通分量；



第二步：对边权值按从小到大进行排序，保证有 $|V|$ 个（也就是顶点数）排序；

边权值排序

V_0V_3	V_2V_5	V_1V_4	V_2V_4 或 V_3V_5	V_0V_2 或 V_1V_3
1	2	3	4	5

判断哪一条边落在两个连通分量上
 这里两种都可以

第三步：按照排序，从小到大逐个连接；
 每次连接的时候，判断当前的边是否落在两个连通分量上，只有落在两个连通分量上的才可以；



5个独立的连通分量

23.2 BFS 相关算法

23.2.1 最短路径问题

单源最短路径问题

- 概念：
 - 两对顶点之间最短路径问题；
- 算法：
 - BFS 算法
 - Dijkstra 算法

多源最短路径问题

- 概念：
 - 每对顶点间的最短路径；

- 算法：
 - Floyd 算法

23.2.2 BFS 算法实现

BSF 算法修改

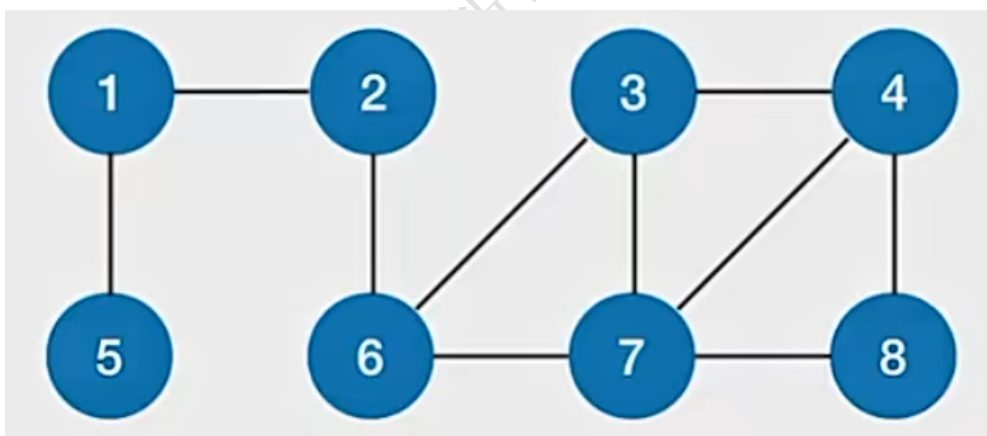
- 概念：
 - 就是对 BFS 的小修改，在 visit 一个顶点时，修改其最短路径长度 $d[]$ 并在 $path[]$ 记录前驱结点
- 问题：
 - 求从 2 顶点到其他顶点的最短路径；
- 几个辅助数组：默认值
 - 1. $d \rightarrow$ 表示距离； $path \rightarrow$ 表示此节点的直接前驱；

	1	2	3	4	5	6	7	8
$d[]$	∞	∞	∞	∞	∞	∞	∞	∞
$path[]$	-1	-1	-1	-1	-1	-1	-1	-1

- 2. Visited

	1	2	3	4	5	6	7	8
visited	false	false	false	false	false	false	false	false

- 图示：



代码实现

```

//下面是改造的广度优先算法
void BFS_MIN_Distance(Graph G, int u){
    //d[i]表示从u到i结点的最短路径
    for(i=0; i<G.vexnum; ++i){
        d[i]=无穷;           //初始化路径长度
        path[i]=-1;          //最短路径从哪个顶点过来
    }
    d[u]=0;
    visited[u]=TRUE;
    EnQueue(Q, u);
  
```

```

while(!isEmpty(Q)){ //BFS算法主过程
    DeQueue(Q,u);    //对头元素u出队
    for(w=FirstNeighbor(G,u);w>=0;w=NextNeighbor(G,u,w))
        if(!visited[w]){ //w为u的尚未访问的邻接点
            d[w]=d[u]+1;    //路径长度加1
            path[w]=u;      //最短路径应从u到w
            visited[w]=TRUE; //设已访问标记
            EnQueue(Q,w);   //顶点w入队
        } //if
    } //while
}

```

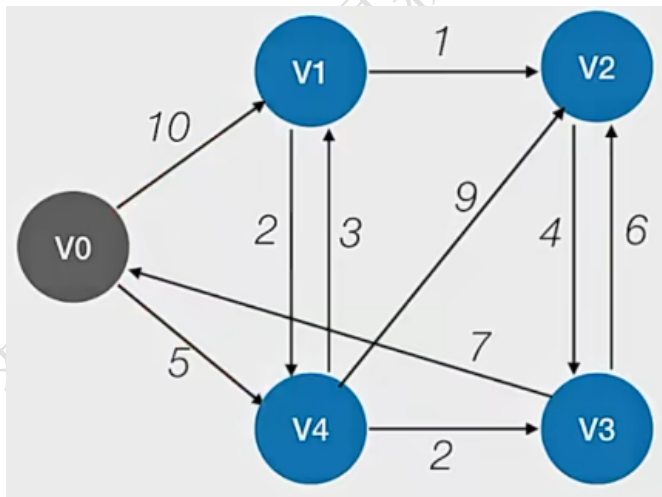
23.3 Dijkstra 算法

带权路径长度

- 概念：
 - 当图是带权图时，一条路径上所有边的权值之和，称为该路径的带权路径长度；

Dijkstra 算法

- 概念：
 - BFS** 算法求单源最短路径只适用于无权图，或所有边的权值都相同的图；
- 性能：
 - $O(n^2)$ 即 $O(|V|^2)$
- 过程：
 - 0. 问题：



- 1. 初始；
 - 从 V_0 开始，初始化三个数组信息；

标记各顶点是否已找到最短路径	V0	V1	V2	V3	V4
$final[5]$	✓	✗	✗	✗	✗
最短路径长度	0	10	∞	∞	5
路径上的前驱	-1	0	-1	-1	0

2. 第二轮

- 循环遍历所有结点，找到还没确定最短路径，且 $dist$ 最小的顶点 V ，令

$final[i]=ture$

标记各顶点是否已找到最短路径	V0	V1	V2	V3	V4
$final[5]$	✓	✗	✗	✗	✓
最短路径长度	0	8	14	∞	5
路径上的前驱	-1	0	-1	-1	0

检查所有邻接自 V_i 的顶点，若其 $final$ 值为false，则更新 $dist$ 和 $path$ 信息

...

- V_0 到 V_2 的最短 (带权) 路径长度为: $dist[2]=9$ 通过 $path[]$ 可知, V_0 到 V_2 的最短 (带权) 路径: $V_2 \leftarrow V_1 \leftarrow V_4 \leftarrow V_0$

代码思路:

初始化:

- 若从 V 开始, 令 $final[0]=ture; dist[0]=0; path[0]=-1$
- 其余顶点 $final[k]=false; dist[k]=arcs[0][k]; path[k]=(arcs[0][k]==\infty)?-1:0$

$n-1$ 轮处理:

- 循环遍历所有顶点，找到还没确定最短路径，且 $dist$ 最小的顶点 V ，令 $final[i]=ture$;
- 并检查所有邻接自 V 的顶点，对于邻接自 V 的顶点 V ，若 $final[j]==false$ 且 $dist[i]+arcs[i][j]<dist[j]$ ，则:
 - 令: $dist[j]=dist[i]+arcs[i][j]; path[j]=i$

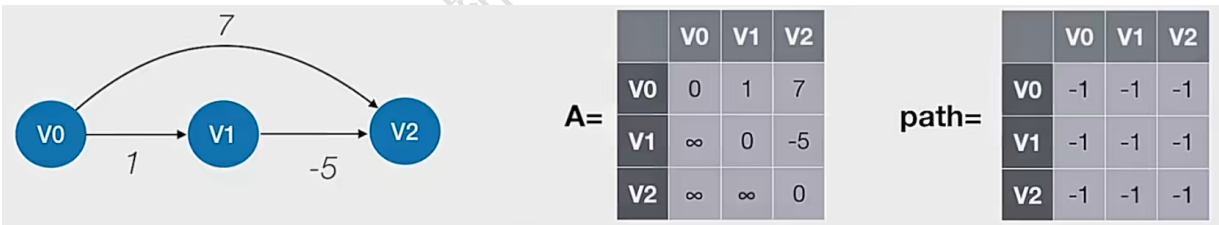
用于负权值的带权图

- Dijkstra 算法不适用于带负权值的带权图;

23.4 Floyd 算法

Floyd 算法

- 概念：
 - 求出每一对顶点之间的最短路径
 - 使用动态规划思想，将问题的求解分为多个阶段
 - 过程
 - 对于 n 个顶点的图，求任意一对顶点 $V_i \rightarrow V_j$ 之间的最短路径可分为如下几个阶段：
 - 初始：不允许在其他顶点中转，最短路径是？
 - 0：若允许在 V_0 中转，最短路径是？
 - 1：若允许在 V_0, V_1 中转，最短路径是？
 - 2：若允许在 V_0, V_1, V_2 中转，最短路径是？
 - ...
 - $n-1$ ：若允许在 $V_0, V_1, V_2, \dots, V_{n-1}$ 中转，最短路径是？
- 性能：
 - 时间复杂度 $O(|V|^3)$
 - 空间复杂度 $O(|V|^2)$
- 注意
 - Floyd 算法不能解决带有负权回路的图 (有负权值的边组成回路)，这种图有可能没有最短路径 (因为会无限轮转)
 - Floyd 算法可以用于解决带有负权的图 (不是回路)；
- 问题：

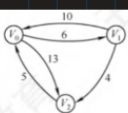


代码实现

```
//代码实现
//...准备工作，根据图的信息初始化矩阵A和path
for(int k=0;k<n;k++){ //考虑以Vk作为中转点
    for(int i=0;i<n;i++){ //编译整个矩阵,i为行号,j为列号
        for(int j=0;j<n;j++){
            if(A[i][j]>A[i][k]+A[k][j]){ //以Vk为中转点的路径更短
                A[i][j]=A[i][k]+A[k][j]; //更新最短路径长度
                path[i][j]=k; //中转点
            }
        }
    }
}
```

总结

Floyd 算法总结



P^{-1}

	0	1	2
0	0	6	13
1	10	0	4
2	5	∞	0

P^0

以0为中间点更新距离

	0	1	2
0	0	6	13
1	10	0	4
2	5	11	0

P^1

以1为中间点

	0	1	2
0	0	6	13
1	10	0	4
2	5	11	0

P^2

最短路径

	0	1	2
0	0	6	10
1	9	0	4
2	5	11	0

方法:

- 第一步: 写第一个矩阵, 其中填入每个结点之间的距离;
- 第二步: 如果当前有三个结点, 依次将它们作为中间结点 Mid, 更新其他结点的值 (如果以此中间节点为路径时, 值是否更少, 更少的话更新)

- 划掉对角线和当前 第i行、第i列 (当前i结点)
- 逐个计算未被划掉的结点的值:
 - if (从 $a \rightarrow b$ 的距离 $<$ 从 $a \rightarrow$ 中间结点 + 中间结点 $\rightarrow b$ 的距离)
 - { 距离更新值为 (a $\rightarrow$ 中间结点 + 中间结点 $\rightarrow b$ 的距离); }
 - else { 不变; }

分析:

	BFS 算法	Dijkstra 算法	Floyd 算法
无权图	✓	✓	✓
带权图	✗	✓	✓
带负权值的图	✗	✗	✓
带负权回路的图	✗	✗	✗
时间复杂度	$O(V ^2)$ 或 $O(V + E)$	$O(V ^2)$	$O(V ^3)$
通常用于	求无权图的单源最短路径	求带权图的单源最短路径	求带权图中各顶点间的最短路径

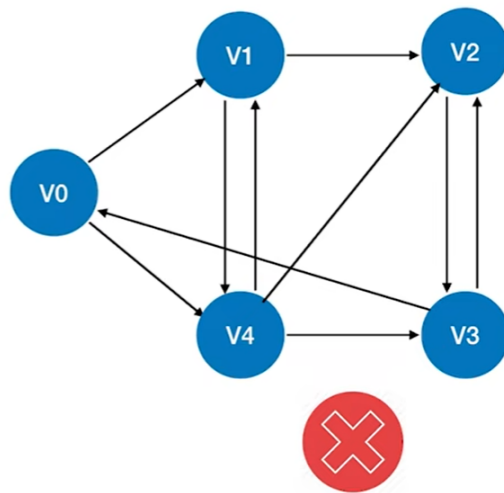
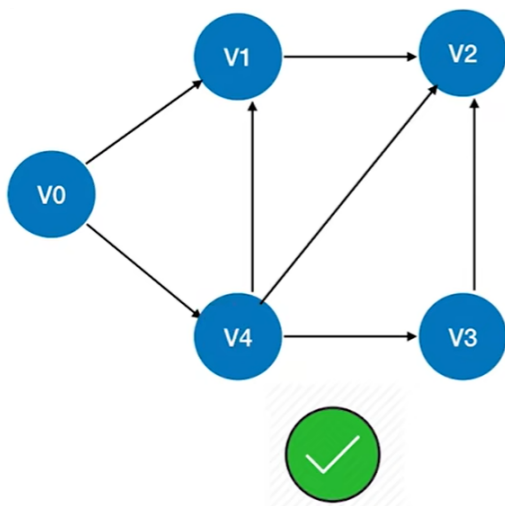
23.5 有向无环图描述表达式

有向无环图

概念:

- 若一个有向图中不存在环, 则称为有向无环图, 简称 **DAG** 图 (Directed Acyclic Graph);

图示:

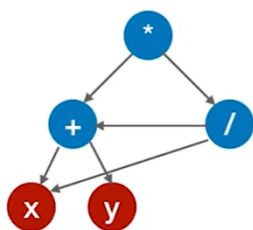


DAG 描述表达式

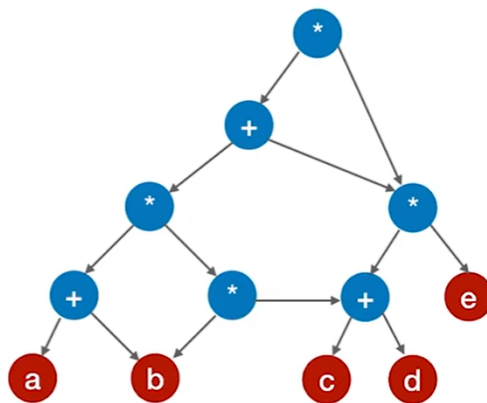
- 图示:

$$(x + y)((x + y) / x)$$

$$((a + b) * (b * (c + d)) + (c + d) * e) * ((c + d) * e)$$



顶点中不可能出现重复的操作数

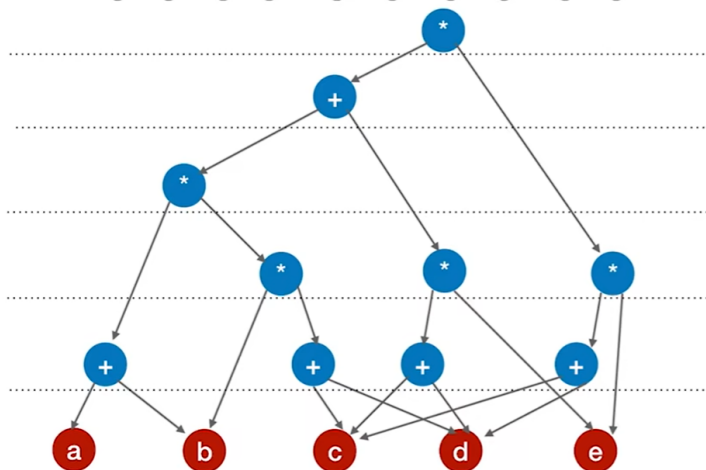


- 解题方法

- 1. 把各个操作数不重复地排成一排;
- 2. 标出各个运算符的生效顺序 (先后顺序有点出入无所谓);
- 3. 按顺序加入运算符, 注意“分层”
- 4. 从底向上逐层的检查, 同层的运算符是否可以合体;

$$((a + b) * (b * (c + d)) + (c + d) * e) * ((c + d) * e)$$

① ④ ③ ② ⑦ ⑤ ⑥ ⑩ ⑧ ⑨

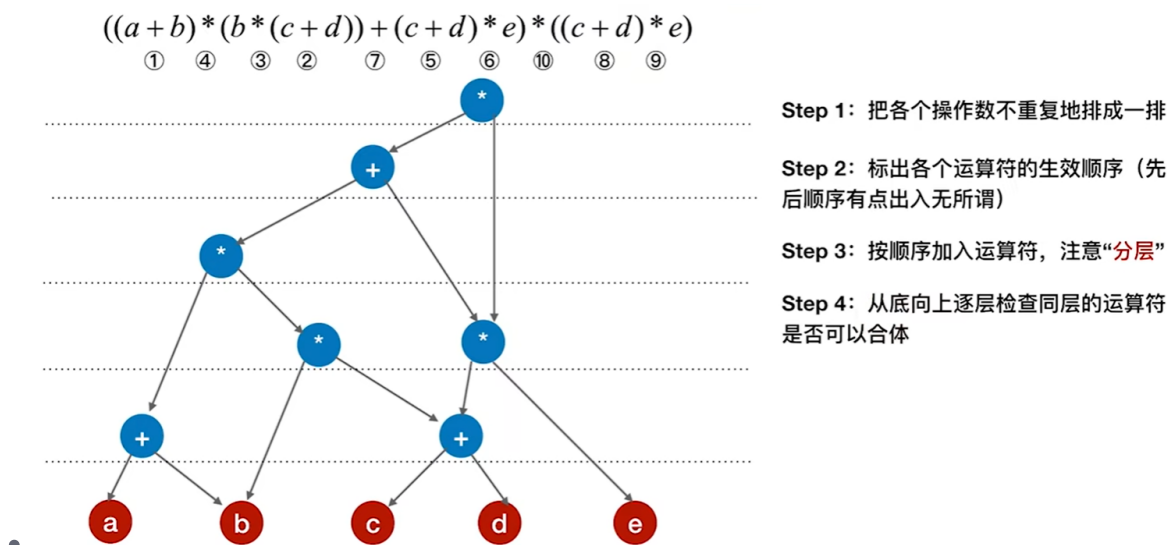


Step 1: 把各个操作数不重复地排成一排

Step 2: 标出各个运算符的生效顺序 (先后顺序有点出入无所谓)

Step 3: 按顺序加入运算符, 注意“分层”

- 将三个+号合并, 将两个乘号合并;



23.6 拓扑排序

23.6.1 AOV 网

AOV 网

- Activity on Vertex NeWork, 用顶点表示活动网；
- 用 DAG 图(有向无环图)表示一个工程。顶点表示活动，有向边 $\langle V_i, V_j \rangle$ 表示活动 V_i 必须先于活动 V_j 进行

23.6.2 拓扑排序

基础概念

- 概念：
 - 找到做事情的顺序；
- 拓扑排序：
 - 在图论中，由一个有向无环图的顶点组成的序列，当且仅当满足下列条件时，称为该图的一个拓扑排序：
 - 1. 每个顶点出现且只出现一次；
 - 2. 若顶点 A 在序列中排在顶点 B 的前面，则在图中不存在从顶点 B 到顶点 A 的路径；
 - 拓扑排序是对有向无环图的顶点的一种排序，它使得若存在一条从顶点 A 到顶点 B 的路径，则在排序中顶点 B 出现在顶点 A 的后面；
 - 每个 AOV 网都有一个或多个拓扑排序序列；
- 实现：
 - 1. 从 AOV 网中选择一个没有前驱(入度为 0)的顶点并输出。
 - 2. 从网中删除该顶点和没有以它为起点的有向边。
 - 3. 重复 (1)和 (2)直到当前的 AOV 网为空或当前网中不存在无前驱的顶点为止。
- 性能：
 - 时间复杂度： $O(|V| + |E|)$
 - 若采用邻接矩阵，则需要 $O(|V|^2)$

• 总结:

★

■ 方法 - 手算拓扑排序

- 1. 从 AOV 网中选择一个没有前驱(入度为 0)的顶点并输出。
- 2. 从网中删除该顶点和没有以它为起点的有向边。
- 3. 重复 (1)和 (2)直到当前的 AOV 网为空或当前网中不存在无前驱的顶点为止。

■ 性质 - 拓扑排序的唯一性

- 若各个顶点排在一个线性有序、每个顶点有唯一的前驱后继关系，则拓扑排序结果唯一；

■ 性质 - 拓扑排序的存在性

- ★ 有向无环图一定有拓扑序列，但不一定唯一

■ 性质 - 拓扑排序的邻接矩阵存储性质

已排好序的拓扑图:

```

    ① → ② → ③ → ④ → ⑤
          ↑
          原的4,重新编号
  
```

↓

	1	2	3	4	5
1	0	1	0	0	0
2		0	1	0	0
3			0	1	0
4				0	1
5					0

可用三角矩阵存储

代码实现

```

#define MaxVertexNum 100 //图中顶点数目的最大值
typedef struct ArcNode{ //边表结点
    int adjvex; //该弧所指向的顶点的位置
    struct ArcNode *nextarc; //指向下一条弧的指针
    //InfoType info; //网的边权值
}ArcNode;
typedef struct VNode{ //顶点表结点
    VertexType data; //顶点信息
    ArcNode *firstarc; //指向第一条依附该顶点的弧的指针
}VNode, AdjList[MaxVertexNum];
typedef struct{
    AdjList vertices; //邻接表
    int vexnum, arcnum; //图的顶点数和弧数
}Graph; //Graph是以邻接表存储的图类型
  
```

```

//拓扑排序算法的实现如下:
bool TopologicalSort(Graph G){
    InitStack(s); //初始化栈, 存储入度为0的顶点
    for(int i=0; i<G.vexnum; i++)
        if(indegree[i]==0)
            Push(S, i); //将所有入度为0的顶点进栈
  
```

```

int count=0;    //计数，记录当前已经输出的顶点数
while(!IsEmpty(S)){ //栈不空，则存在入度为0的顶点
    Pop(S,i);    //栈顶元素出栈
    print[count++]=i; //输出顶点i
    for(p=G.vertices[i].firstarc;p;p=p->nextarc){
        //将所有i指向的顶点的入度减1，并且将入度减为0的顶点压入栈S
        v=p->adjvex;
        if(!(--indegree[v]))
            Push(S,v); //入度为0，则入栈
    }
} //while
if(count<G.vexnum)
    return false;    //排序失败，有向图中有回路
else
    return true;    //拓扑排序成功
}

```

23.6.3 逆拓扑排序

逆拓扑排序

- 概念：
 - 对一个 AOV 网，如果采用下列步骤进行排序，则称之为逆拓扑排序：
- 实现：
 - 1. 从 AOV 网中选择一个没有后继(出度为 0)的顶点并输出
 - 2. 从网中删除该顶点和所有以它为终点的有向边。
 - 3. **重复**(1)和 (2)直到当前的 AOV 网为空
- 性质
 - 1. 拓扑排序，逆拓扑排序序列可能不唯一
 - 2. 若图中有环，则不存在拓扑排序序列/逆拓扑排序序列

代码实现

```

void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v)//本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}

void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE;    //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
}

```

```
print(v);           //输出顶点
}
```

23.7 关键路径

23.7.1 AOE 网

AOE 网基础概念

- 概念：
 - 在带权有向图中，以顶点表示事件，以有向边表示活动，以边上的权值表示完成该活动的开销(如完成活动所需的时间)，称之为用边表示活动的网络，简称 **AOE网** (Activity On Edge NetWork)
- 性质：
 - 1. 只有在某顶点所代表的事件发生后，从该顶点出发的各有向边所代表的活动才能开始。
 - 2. 只有在进入某顶点的各有向边所代表的活动都已结束时，该顶点所代表的事件才能发生。
 - **另外**，有些活动是可以并行进行的。
- 概念：
 - 在 AOE 网中仅有一个入度为 0 的顶点，称为开始顶点 (源点)，它表示整个工程的开始。
 - 也仅有一个出度为 0 的顶点，称为结束顶点 (汇点)，它表示整个工程的结束。
 - 从源点到汇点的有向路径可能有多条，所有路径中，具有最大路径长度的路径称为关键路径，而把关键路径上的活动称为关键活动。
 - **完成整个工程的最短时间**就是关键路径的长度，若关键活动不能按时完成，则整个工程的完成时间就会延长。
 - 时间 v_k 的最早发生时间 $ve(k)$ **决定了所有从 v_k 开始的活动能够开工的最早时间**。
 - 活动 a_i 的最早开始时间 $e(i)$ ：指该活动弧的起点所表示的事件的最早发生事件。
 - 事件的 v_k 最迟发生时间 $vl(k)$ ：
 - 它是指在不推迟整个工程完成的前提下，该事件最迟必须发生的事件。
 - 活动 a_i 的最迟开始时间 $l(i)$ ：
 - 它是指该活动弧的终点所表示事件的最迟发生时间与该活动所需时间之差。
 - 活动 a_i 的时间余量 $d(i) = l(i) - e(i)$
 - 表示在不增加完成整个工程所需总时间的情况下，活动 a_i 可以延迟的时间。
 - 若一个活动的时间余量为零，则说明该活动必须要如期完成， $d(i) = 0$ 即 $l(i) = e(i)$ 的活动 a_i 是关键活动。
 - 由**关键活动**组成的路径就是关键路径。

求关键路径的步骤

1. 求所有事件的最早发生时间 $ve(k)$
2. 求所有事件的最迟发生时间 $vl(k)$
3. 求所有活动的最早开始时间 $e(i)$
4. 求所有活动的最迟开始时间 $l(i)$

5. 求所有活动的时间余量 $d(i)$ ($d(i) = 0$ 的活动就是关键活动)

关键活动，关键路径的特性

1. 若关键活动耗时增加，则整个工程的工期将增长
2. 缩短关键活动的时间，可以缩短整个工程的工期
3. 当缩短到一定程度时，关键活动可能会变成非关键活动

获取笔记持续更新可加微信: WandD2277