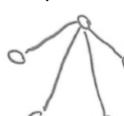
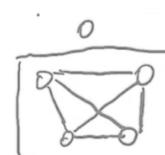


## 第五章、图

- 1、掌握图的定义及其它基本概念。
- 2、掌握图的存储结构——邻接矩阵、邻接表。
- 3、掌握图的遍历方法——深度优先搜索、广度优先搜索。
- 4、掌握最小生成树生成方法。
- 5、掌握图的最短路径算法。
- 6、了解拓扑排序概念，了解关键路径算法。

图的定义们																													
符号们	<table border="1"> <tr> <td><math>V(G)</math></td><td>图<math>G</math>中顶点的有限非空集</td></tr> <tr> <td><math> V </math></td><td>图<math>G</math>中顶点的个数=图<math>G</math>的阶</td></tr> <tr> <td><math>E(G)</math></td><td>图<math>G</math>中边的集合</td></tr> <tr> <td><math> E </math></td><td>图<math>G</math>中边的条数</td></tr> </table>	$V(G)$	图 $G$ 中顶点的有限非空集	$ V $	图 $G$ 中顶点的个数=图 $G$ 的阶	$E(G)$	图 $G$ 中边的集合	$ E $	图 $G$ 中边的条数																				
$V(G)$	图 $G$ 中顶点的有限非空集																												
$ V $	图 $G$ 中顶点的个数=图 $G$ 的阶																												
$E(G)$	图 $G$ 中边的集合																												
$ E $	图 $G$ 中边的条数																												
路径	<table border="1"> <tr> <td>定义</td><td><math>v</math> 到 <math>v'</math> 的 路径 是一个 顶点 序列 <math>v = v_{i,0}, v_{i,1}, \dots v_{i,m} = v'</math> , 其 中 , <math>(v_{i,j-1}, v_{i,j}) \in E, 1 \leq j \leq m</math></td></tr> <tr> <td>路径长度</td><td></td></tr> <tr> <td>距离</td><td>两点间的最短路径。若不存在路径，则距离为<math>\infty</math></td></tr> <tr> <td>简单路径</td><td>路径序列中，顶点不重复出现的路径</td></tr> <tr> <td rowspan="2">回路/环</td><td>定义</td></tr> <tr> <td>第一个顶点和最后一个顶点相同的路径</td></tr> <tr> <td rowspan="2">子图</td><td>简单回路</td></tr> <tr> <td>除第一个和最后一个顶点，其余顶点不重复出现的回路</td></tr> <tr> <td>稀疏图</td><td>定义</td></tr> <tr> <td>稠密图</td><td>生成子图 满足 <math>V(G') = V(G)</math> 的子图</td></tr> <tr> <td>简单图</td><td>有很少条边 or 弧的图，如 <math> E  &lt; n \log n</math></td></tr> <tr> <td>多重图</td><td>与稀疏图相反</td></tr> <tr> <td>度</td><td> <table border="1"> <tr> <td>出度</td><td></td></tr> <tr> <td>入度</td><td></td></tr> </table> </td></tr> </table>	定义	$v$ 到 $v'$ 的 路径 是一个 顶点 序列 $v = v_{i,0}, v_{i,1}, \dots v_{i,m} = v'$ , 其 中 , $(v_{i,j-1}, v_{i,j}) \in E, 1 \leq j \leq m$	路径长度		距离	两点间的最短路径。若不存在路径，则距离为 $\infty$	简单路径	路径序列中，顶点不重复出现的路径	回路/环	定义	第一个顶点和最后一个顶点相同的路径	子图	简单回路	除第一个和最后一个顶点，其余顶点不重复出现的回路	稀疏图	定义	稠密图	生成子图 满足 $V(G') = V(G)$ 的子图	简单图	有很少条边 or 弧的图，如 $ E  < n \log n$	多重图	与稀疏图相反	度	<table border="1"> <tr> <td>出度</td><td></td></tr> <tr> <td>入度</td><td></td></tr> </table>	出度		入度	
定义	$v$ 到 $v'$ 的 路径 是一个 顶点 序列 $v = v_{i,0}, v_{i,1}, \dots v_{i,m} = v'$ , 其 中 , $(v_{i,j-1}, v_{i,j}) \in E, 1 \leq j \leq m$																												
路径长度																													
距离	两点间的最短路径。若不存在路径，则距离为 $\infty$																												
简单路径	路径序列中，顶点不重复出现的路径																												
回路/环	定义																												
	第一个顶点和最后一个顶点相同的路径																												
子图	简单回路																												
	除第一个和最后一个顶点，其余顶点不重复出现的回路																												
稀疏图	定义																												
稠密图	生成子图 满足 $V(G') = V(G)$ 的子图																												
简单图	有很少条边 or 弧的图，如 $ E  < n \log n$																												
多重图	与稀疏图相反																												
度	<table border="1"> <tr> <td>出度</td><td></td></tr> <tr> <td>入度</td><td></td></tr> </table>	出度		入度																									
出度																													
入度																													

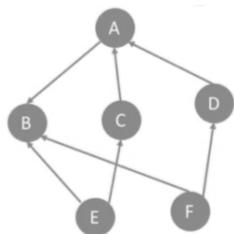
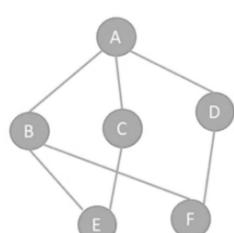
有向图	有向树	其中一个顶点入度为 0，其余顶点入度为 1	
	完全图	有 $n(n - 1)$ 条弧的有向图	
	强连通	定义	一对顶点双向有路径
		强连通图	图中任意一对顶点都是强连通的。强连通图最少有 $n$ (顶点数) 条边，此时刚好形成回路
	强连通分量	有向图中的极大强连通子图	
无向图	定义	弧尾：没箭头的那边 弧头：有箭头的那边	
	度		
	完全图	有 $\frac{n(n-1)}{2}$ 条边的无向图	
	连通	定义	两顶点有路径存在
		连通分量	无向图中的极大连通子图
		极大连通子图	无向图的连通分量，要求包含所有的边
		极小连通子图	保持图连通，且边数要最少
	连通图	定义	<ul style="list-style-type: none"> <li>图中任意两个顶点都是连通的。</li> <li>连通图最少有 <math>n - 1</math> (<math>n</math> 为顶点数) 条边，此时</li> </ul> 
		生成树	<ul style="list-style-type: none"> <li>包含连通图中全部顶点的一个极小连通子图</li> <li>生成树有 <math>n - 1</math> (<math>n</math> 为顶点数) 条边</li> <li>生成树不唯一</li> </ul>
	非连通图	定义	<ul style="list-style-type: none"> <li>图中有不连通的顶点。</li> <li>非连通图最多有 <math>C_{n-1}^2</math> (<math>n</math> 为顶点数) 条边，此时</li> </ul> 
		生成森林	非连通图中，由连通分量的生成树构成

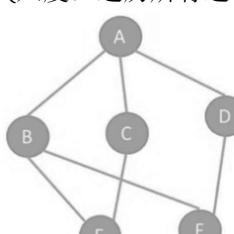
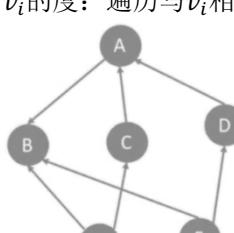
网和图的区别：网里面对应的边是有权值的

## 掌握图的存储结构——邻接矩阵、邻接表、十字链表、邻接多重表

### 速查速查

	邻接矩阵	邻接表	十字链表	邻接多重表
空间复杂度	$O( V ^2)$	无向图 $O( V  + 2 E )$ 有向图 $O( V  +  E )$	$O( V  +  E )$	$O( V  +  E )$
找相邻边	遍历对应行或列 时间复杂度为 $O( V )$	找有向图的入边必须遍历整个邻接表	很方便	很方便
删除边或顶点	删除边很方便，删除顶点需要大量移动数据	无向图中删除边或顶点都不方便	很方便	很方便
适用于	稠密图	稀疏图和其他	只能存有向图	只能存无向图
表示方式	唯一	不唯一	不唯一	不唯一

	对象	定义&性质	比较&特点																																										
邻接矩阵	有向图	$A[i][j] = \begin{cases} i \text{ 到 } j \text{ 有路径: 1} \\ i \text{ 到 } j \text{ 间无路径: 0} \end{cases}$  <table border="1" data-bbox="577 758 770 972"> <tr><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td></tr> <tr><td>A</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>B</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>C</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>D</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>E</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>F</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table> <p><math>v_i</math> 的度 <math>\begin{cases} \text{出度: 第 } i \text{ 行元素之和} \\ \text{入度: 第 } i \text{ 列元素之和} \end{cases}</math></p>	A	B	C	D	E	F	A	0	1	0	0	0	B	0	0	0	0	0	C	1	0	0	0	0	D	1	0	0	0	0	E	0	1	1	0	0	F	0	1	0	1	0	适用于存稠密图  空间复杂度只和顶点数有关，与边数无关， $O( V ^2)$  图的邻接表表示方式唯一
A	B	C	D	E	F																																								
A	0	1	0	0	0																																								
B	0	0	0	0	0																																								
C	1	0	0	0	0																																								
D	1	0	0	0	0																																								
E	0	1	1	0	0																																								
F	0	1	0	1	0																																								
无向图	$A[i][j] = \begin{cases} 两点间有路径: 1 \\ 两点间无路径: 0 \end{cases}$  <table border="1" data-bbox="577 1185 770 1399"> <tr><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td></tr> <tr><td>A</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr> <tr><td>B</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>C</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>D</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr> <tr><td>E</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>F</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr> </table> <p>邻接矩阵是唯一的对称矩阵，可压缩存储；  <math>v_i</math> 的度 = 第 <math>i</math> 行 / 列之和</p>	A	B	C	D	E	F	A	0	1	1	1	0	B	1	0	0	0	1	C	1	0	0	0	1	D	1	0	0	0	1	E	0	1	1	0	0	F	0	1	0	1	0	$A^n[i][j]$ : $i$ 到 $j$ 长度为 $n$ 的路径的数量  优点：容易确定图中任意两点间是否有边相联  缺点：不方便确定图中有多少条边 空间复杂度高	
A	B	C	D	E	F																																								
A	0	1	1	1	0																																								
B	1	0	0	0	1																																								
C	1	0	0	0	1																																								
D	1	0	0	0	1																																								
E	0	1	1	0	0																																								
F	0	1	0	1	0																																								
网	$A[i][j] = \begin{cases} 两点间有路径: 两点间的权重 \\ 两点间无路径: \infty \end{cases}$																																												
定义	头结点： <table border="1" data-bbox="247 1691 495 1749"><tr><td>data</td><td>链域</td></tr></table> data: 存储 $v_i$ 的名或其他有关信息	data	链域	适用于存储稀疏图 $ E  \ll \frac{n(n-1)}{2}$																																									
data	链域																																												

	<p>链域：指向链表中第一个结点</p> <p>表结点</p> <table border="1"> <thead> <tr> <th>邻接点域</th><th>链域</th><th>info</th></tr> </thead> </table> <p>邻接点域：与<math>v_i</math>邻接的点在图中的位置</p> <p>链域：下一条边/弧的结点</p> <p>info：边/弧的相关信息（如权值）</p>	邻接点域	链域	info	<p>图的邻接表表示方式不唯一 (边链表的顺序不唯一)</p> <p>优点：容易找到任一顶点的第一个邻接点和邻边</p>											
邻接点域	链域	info														
邻接表 — 顺序 + 链式	<p><math>v_i</math>的度</p> <p>出度：<math>v_i</math>相关边链表的结点数</p> <p>入度：遍历所有边链表找到指向该结点的边</p>  <table border="1"> <thead> <tr> <th>data</th> <th>*first</th> </tr> </thead> <tbody> <tr> <td>0 A</td> <td>→ [1] → [2] → [3] ^</td> </tr> <tr> <td>1 B</td> <td>→ [0] → [4] → [5] ^</td> </tr> <tr> <td>2 C</td> <td>→ [0] → [4] ^</td> </tr> <tr> <td>3 D</td> <td>→ [0] → [5] ^</td> </tr> <tr> <td>4 E</td> <td>→ [1] → [2] ^</td> </tr> <tr> <td>5 F</td> <td>→ [1] → [3] ^</td> </tr> </tbody> </table>	data	*first	0 A	→ [1] → [2] → [3] ^	1 B	→ [0] → [4] → [5] ^	2 C	→ [0] → [4] ^	3 D	→ [0] → [5] ^	4 E	→ [1] → [2] ^	5 F	→ [1] → [3] ^	<p>空间复杂度：<math>O( V  + 2 E )</math>：相比有向图，无向图的每条边在邻接表中出现两次</p> <p>缺点：</p> <p>每条边对应两份冗余信息，删除顶点、删除边等操作时间复杂度高</p>
data	*first															
0 A	→ [1] → [2] → [3] ^															
1 B	→ [0] → [4] → [5] ^															
2 C	→ [0] → [4] ^															
3 D	→ [0] → [5] ^															
4 E	→ [1] → [2] ^															
5 F	→ [1] → [3] ^															
有向图	<p><math>v_i</math>的度：遍历与<math>v_i</math>相关的边链表，结点个数=度</p>  <table border="1"> <thead> <tr> <th>data</th> <th>*first</th> </tr> </thead> <tbody> <tr> <td>0 A</td> <td>→ [1] ^</td> </tr> <tr> <td>1 B</td> <td>^</td> </tr> <tr> <td>2 C</td> <td>→ [0] ^</td> </tr> <tr> <td>3 D</td> <td>→ [0] ^</td> </tr> <tr> <td>4 E</td> <td>→ [1] → [2] ^</td> </tr> <tr> <td>5 F</td> <td>→ [1] → [3] ^</td> </tr> </tbody> </table>	data	*first	0 A	→ [1] ^	1 B	^	2 C	→ [0] ^	3 D	→ [0] ^	4 E	→ [1] → [2] ^	5 F	→ [1] → [3] ^	<p>空间复杂度：<math>O( V  +  E )</math></p> <p>缺点：</p> <p>计算度和入度不方便； 找入边不方便</p>
data	*first															
0 A	→ [1] ^															
1 B	^															
2 C	→ [0] ^															
3 D	→ [0] ^															
4 E	→ [1] → [2] ^															
5 F	→ [1] → [3] ^															

## 十字链表

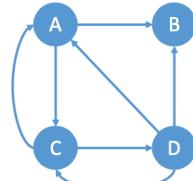
弧结点:

弧尾顶点编号	弧头顶点编号	权值
弧头相同的下一条弧		弧尾相同的下一条弧

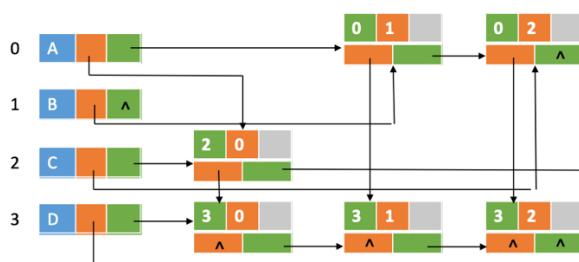
顶点结点:

data	该顶点为弧头 的第一条弧	该顶点为弧尾 的第一条弧
------	-----------------	-----------------

有向图



空间复杂度:  $O(|V| + |E|)$



边结点:

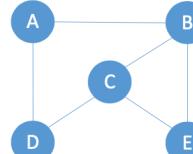
边的顶点 1	边的顶点 2	权值
顶点 1 下一条边	顶点 2 的下一条边	

顶点结点:

data	与该顶点相连的第一条边
------	-------------

## 邻接多重表顺序 + 链式

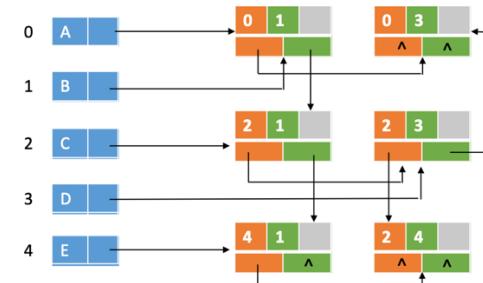
无向图



空间复杂度:  $O(|V| + |E|)$

优于邻接矩阵和邻接表

删除结点、边方便



```

//图的邻接矩阵存储表示

#define INFINITY          //最大值∞

#define MAX_VERTEX_NUM 20 //最大顶点个数

typedef enum

{

    DG,   //有向图

    DN,   //有向网

    UDG,  //无向图

    UDN  //无向网

} GraphKind;

typedef struct ArcCell

{

    VRTType adj;    // VRTType 是顶点关系类型。对于无权图，1 为相邻，0 为不相邻。对于带权图，为权值类型

    InfoType *info; //该弧相关信息的指针

} ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct

{

    VertexType vexts[MAX_VERTEX_NUM]; //顶点集合

    AdjMatrix arcs;                //邻接矩阵

    int vexnum, arcnum;           //图的当前顶点数和弧数

    GraphKind kind;               //图的种类标志

} MGraph;

//图的邻接表存储表示

#define MAX_VERTEX_NUM 20 //最大顶点个数

typedef struct ArcNode

{

    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;             //该弧相关信息的指针

} ArcNode;

typedef struct VNode

{

    VertexType data;      //顶点信息

    ArcNode *firsttarc; //指向第一条依附该顶点的弧的指针

} VNode, AdjList[MAX_VERTEX_NUM];

typedef struct

```

```

{
    AdjList vertices;

    int vexnum, arcnum; //图当前的顶点和弧数

    int kind;           //图的种类

} ALGraph;

```

## 掌握图的遍历方法——深度优先搜索 DFS、广度优先搜索 BFS

	要点	算法评价
BFS	<ul style="list-style-type: none"> <li>● =树的层序遍历</li> <li>● 需要一个辅助队列，空间复杂度来自他</li> <li>● <i>visited</i>数组阻止重复访问</li> <li>● 处理非连通图：遍历完图后遍历 <i>visited</i>[], 若存在 false 又开始遍历</li> <li>● 遍历序列 <math>\begin{cases} \text{邻接矩阵: 唯一} \\ \text{邻接表: 不唯一} \end{cases}</math></li> <li>● 广度优先生成树：只要第一次到的时候的路径</li> <li>● 广度优先生成森林：非连通图的广度优先生成树们</li> </ul>	$\begin{cases} \text{空间(最坏, 发散状): } O( V ) \\ \text{时间} \begin{cases} \text{邻接矩阵: } O( V ^2) \\ \text{邻接表: } O( V  +  E ) \end{cases} \end{cases}$
DFS	<ul style="list-style-type: none"> <li>● =树的先根遍历</li> <li>● 需要一个辅助栈，空间复杂度来自他</li> <li>● <i>visited</i>数组阻止重复访问</li> <li>● 处理非连通图：同 BFS</li> <li>● 遍历序列：同 BFS</li> <li>● 深度优先生成树&amp;森林：同 BFS</li> <li>● 无向图：有多少连通分量，就调用多少次遍历函数</li> <li>● 有向图：强连通图只调用一次</li> </ul>	$\begin{cases} \text{空间} \begin{cases} \text{最好: 一条线时 } O(1) \\ \text{最坏: 发散状 } O( V ) \end{cases} \\ \text{时间} \begin{cases} \text{邻接矩阵: } O( V ^2) \\ \text{邻接表: } O( V  +  E ) \end{cases} \end{cases}$

## 图的遍历和图的连通性

$$\begin{cases} \text{无向图: BFS/DFS 调用次数 = 连通分量数} \\ \text{有向图} \begin{cases} \text{起始点到其他顶点都有路径: 调用一次 DFS/BFS} \\ \text{强连通图: 调用一次 BFS/DFS} \end{cases} \end{cases}$$

## 掌握最小生成树生成方法

最小生成树特点：

- 1、可能有多个；
- 2、砍掉一条则不连通，增加一条则出现回路
- 3、若连通图本身就是树，最小生成树是他本身

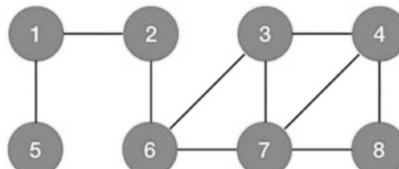
	<i>prim</i> 算法	<i>kruskal</i> 算法
描述	将代价最小的顶点纳入生成树，直到所有顶点都纳入为止 	每次挑选权值最小的边，使这条边两头连通，若出现回路就不选 直到所有节点连通（步骤和权值一样） 
评价	时间复杂度: $O( V ^2)$ (只和顶点有关系) 要进行 $n-1$ 轮的循环，每一轮要遍历两个数组 适用于边稠密图	时间复杂度只和边有关系: $O( E  \log_2  E )$ 适用于边稀疏图

求图的最短路径算法

速查速查

		无权	带权	回路	负权	负权回路	时间复杂度
单源最短路径	<i>BFS</i>						
	<i>Dijkstra</i>						
每对顶点间最短路径	<i>floyd</i>						

*BFS*求最短路径



比如，求 2 到其他顶点的最短路径

在*BFS*的基础上增加两个数组

1、各个顶点到原始顶点的路径  $d[]$

2、顶点在路径上的直接前驱  $path[]$

\*为方便表示，数组下标里直接是顶点的编号

*step1.*初始化：

	1	2	3	4	5	6	7	8
<b>d[]</b>	$\infty$							
<b>path[]</b>	-1	-1	-1	-1	-1	-1	-1	-1

step2.2 号到其实顶点的距离为 0，所以将  $d[2]$  设为 0

	1	2	3	4	5	6	7	8
<b>d[]</b>	$\infty$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
<b>path[]</b>	-1	-1	-1	-1	-1	-1	-1	-1

step3.  $visited[2] = \text{TRUE}$ , 将 2 号顶点放入队列中

step4. 队列非空，弹出队头元素 2

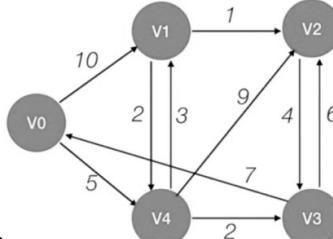
step5. 找到和 2 相邻的所有顶点，记录最短路径和直接前驱

	1	2	3	4	5	6	7	8
<b>d[]</b>	1	0	$\infty$	$\infty$	$\infty$	1	$\infty$	$\infty$
<b>path[]</b>	2	-1	-1	-1	-1	2	-1	-1

step6.  $visited[1] \& visited[6] = \text{TRUE}$ , 将 1 号和 6 号顶点放入队列

……以此类推

Dijkstra 求最短路径



比如这个图，求  $v_0$  的最短路径

初始化三个数组

1、标记各顶点是否已找到最短路径  $bool final[]$

2、最短路径长度  $dist[]$

3、路径上的直接前趋  $path[]$

step1. 初始化：

数组名	V0	V1	V2	V3	V4	备注
<b>final[]</b>	TRUE	FALSE	FALSE	FALSE	FALSE	
<b>dist[]</b>	0	10	$\infty$	$\infty$	5	没有直接路径 (V2) or 方向不对 (V3)，都是 $\infty$
<b>path[]</b>	-1	0	-1	-1	0	起点和没有直接前驱的顶点为 -1

step2. 遍历所有结点，找到  $final=False \&& dist$  最小的顶点，令  $final=True$

数组名	V0	V1	V2	V3	V4
<b>final[]</b>	TRUE	FALSE	FALSE	FALSE	TRUE
<b>dist[]</b>	0	10	$\infty$	$\infty$	5
<b>path[]</b>	-1	0	-1	-1	0

step3. 检查所有与 V4 相邻的顶点 (V1、V2、V3)，若存在比之前更短的 path 则进行覆盖

数组名	V0	V1	V2	V3	V4
<b>final[]</b>	TRUE	FALSE	FALSE	FALSE	TRUE
<b>dist[]</b>	0	8	14	7	5
<b>path[]</b>	-1	4	4	4	0
备注		V0→V4→V1 比 V0→V1 更优		找到了从 V0 过来的路径	

下一个 V3……以此类推

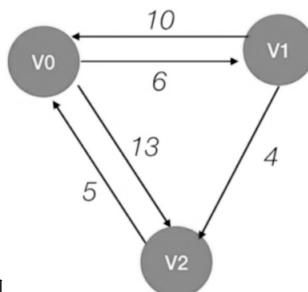
最后得到：

数组名	V0	V1	V2	V3	V4
<b>final[]</b>	TRUE	TRUE	TRUE	TRUE	TRUE
<b>dist[]</b>	0	8	9	7	5
<b>path[]</b>	-1	4	1	4	0

特点：

- 算法评价： $O(|V|^2)$
- 不适用于有负权值的带权图

floyd 求两点间最短路径



比如这个图

step1. 不允许在其他顶点中转，初始化两个矩阵

- 1、目前看来，各顶点间的最短路径长度  $A^{(-1)}$
- 2、两个顶点之间的中转点  $path^{(-1)}$ ，两点间有直接路径就设为 -1

	V0	V1	V2
V0	0	6	13
V1	10	0	4
V2	5	$\infty$	0

	V0	V1	V2
V0	-1	-1	-1
V1	-1	-1	-1
V2	-1	-1	-1

step2. 若允许在 V0 中转?

遍历  $A^{(-1)}$ ,

```
if ( $A^{(-1)}[i][j] > A^{(-1)}[i][0] + A^{(-1)}[0][j]$ )
```

```
{
```

```
//本例中只有 v2→v1>v2→v0→v1, 所以注释中用这个举例
```

```
 $A^{(-1)}[i][j] = A^{(-1)}[i][0] + A^{(-1)}[0][j];$  //将 v2→v1 的距离替换成 v2→v0→v1 的距离
```

```
 $path^{(-1)}[i][j] = 0;$  // v2→v1 的最短路径的中转点为 v0, 所以设成 0
```

```
}
```

	V0	V1	V2
V0	0	6	13
V1	10	0	4
V2	5	11	0

	V0	V1	V2
V0	-1	-1	-1
V1	-1	-1	-1
V2	-1	0	-1

step3. 若允许在 V0、V1 中转?

遍历  $A^{(0)}$ ,

```
if ( $A^{(0)}[i][j] > A^{(0)}[i][1] + A^{(0)}[1][j]$ )
```

```
{
```

```
//本例中只有只有 v0→v2>v0→v1→v2, 所以注释中用这个举例
```

```
 $A^{(0)}[i][j] = A^{(0)}[i][1] + A^{(0)}[1][j];$  //将 v0→v2 的距离替换成 v0→v1→v2 的距离
```

```
 $path^{(0)}[i][j] = 0;$  // v0→v2 的最短路径的中转点为 v1, 所以设成 1
```

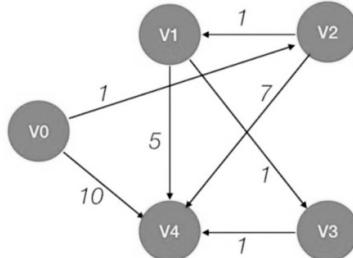
```
}
```

	V0	V1	V2
V0	0	6	10
V1	10	0	4
V2	5	11	0

	V0	V1	V2
V0	-1	-1	1
V1	-1	-1	-1
V2	-1	0	-1

下一个允许在 V0、V1、V2 中转……以此类推

如果中转点不唯一, like this



**A =**

	V0	V1	V2	V3	V4
V0	0	2	1	3	4
V1	$\infty$	0	$\infty$	1	2
V2	$\infty$	1	0	2	3
V3	$\infty$	$\infty$	$\infty$	0	1
V4	$\infty$	$\infty$	$\infty$	$\infty$	0

**path =**

	V0	V1	V2	V3	V4
V0	-1	2	-1	2	3
V1	-1	-1	-1	-1	3
V2	-1	-1	-1	1	3
V3	-1	-1	-1	-1	-1
V4	-1	-1	-1	-1	-1

通过 path 矩阵递归找到完整路径

解释

<b>step1</b>	V0			V3	V4	通过 path 矩阵, V0→V4 中间是 V3
<b>step2</b>	V0	V2		V3	V4	V0→V3 中间是 V2, V3→V4 中间无
<b>step3</b>	V0	V2	V1	V3	V4	V0→V2 无, V2→V3 中 V1。V2→V1 无, V1→V3 无, 完

特点:

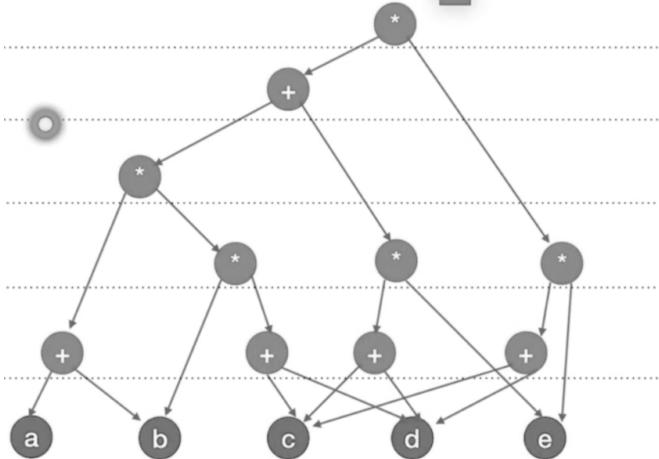
- 算法评价
 
$$\begin{cases} \text{时间复杂度: } O(|V|^3) \\ \text{空间复杂度: } O(|V|^2) \end{cases}$$
- 可以解决负权值的图
- 不可以解决带有负权回路的图

## 有向无环图 dag 的应用-描述表达式

- 表达式的有向无环图不唯一

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

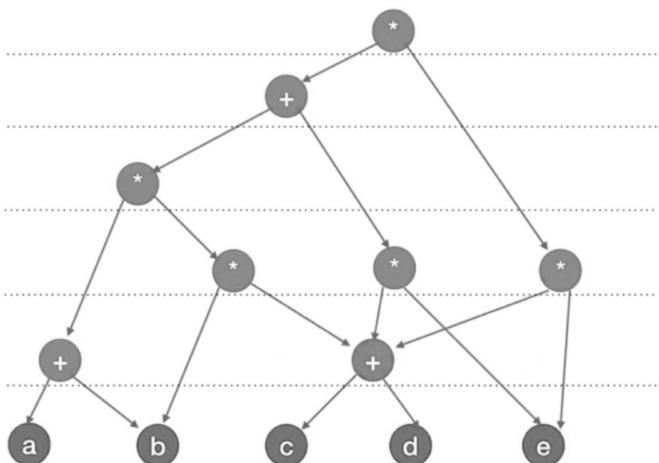
① ④ ③ ② ⑦ ⑤ ⑥ ⑩ ⑧ ⑨



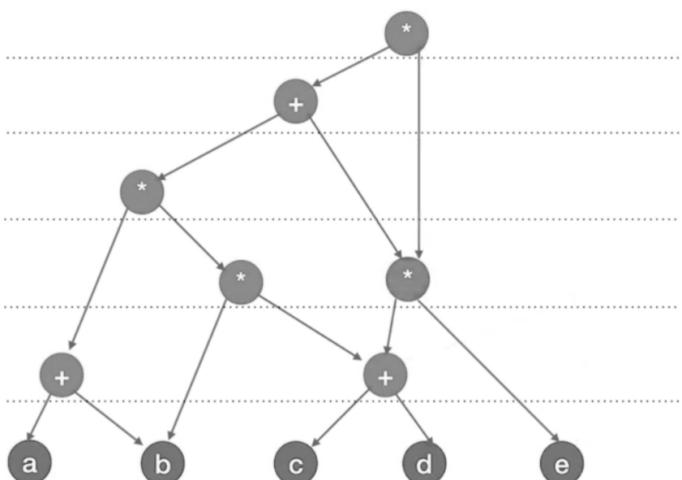
Step 1: 把各个操作数不重复地排成一排

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”



Step 4: 从底向上逐层检查同层的运算符是否可以合体  
这里合了三个+



最后长这样

## AOV 和 AOE

	解释	特点
AOV (vertex) 网	顶点代表活动，有向边代表活动执行的先后顺序，	<ul style="list-style-type: none"> <li>不能出现环路</li> </ul>
AOE (edge) 网	有向边表示活动，边上权值表示活动开销	<ul style="list-style-type: none"> <li>顶点的事件发生后，从顶点出发的有向边代表的活动才能开始</li> <li>进入顶点的有向边们代表的活动们都结束时，顶点代表的事件才算发生</li> <li>有的事件可以并行</li> </ul>

## 有向无环图 dag 的应用 2-拓扑排序

	解释	特点
拓扑排序	<p><i>step1.</i> 从 AOV 网选择入度为 0 的顶点输出  <i>step2.</i> 从网中删除该点和从该点出发的弧          重复直到网为空</p> <p>时间复杂度 <math>\begin{cases} \text{邻接表: } O( V  +  E ) \\ \text{邻接矩阵: } O( V ^2) \end{cases}</math></p>	<ul style="list-style-type: none"> <li>排序可用 DFS 实现</li> <li>序列不唯一</li> <li>若图中有环，不存在（逆）拓扑序列</li> </ul>
逆拓扑排序	<p><i>step1.</i> 从 AOV 网选择出度为 0 的顶点输出  <i>step2.</i> 从网中删除该点和以他为终点的弧          重复直到网为空</p> <p>● 用邻接矩阵比较好，因为邻接表不好找入边</p>	

## 了解关键路径算法

关键路径：AOE 中的最大路径。关键路径的长度是完成整个工程的最短时间

开始顶点/源点：工程的开始

结束顶点/汇点：工程的结束

关键活动：关键路径上的活动

事件的最早发生时间  $ve()$ ：

事件的最迟发生时间  $vl()$ ：

活动的最迟开始时间  $l(i)$ ：

活动的最早开始时间  $e(i)$ ：

时间余量:  $d(i) = l(i) - e(i)$  不增加整个工程时间的情况下，活动  $i$  可以拖延的时间。关键活动  $d(i) = 0$

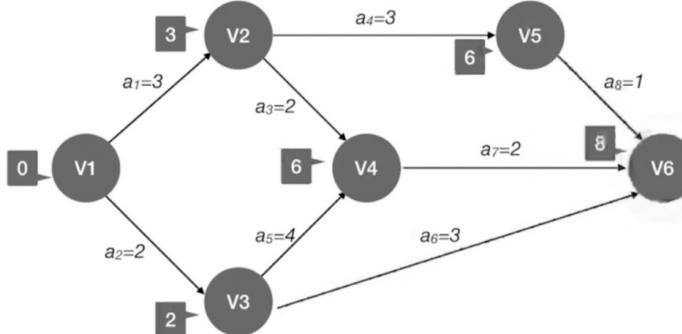
## 计算关键路径:

### step1.求所有事件的最早发生时间

按拓扑排序序列，依次求各个顶点的  $ve(k)$ :

$$ve(\text{源点}) = 0$$

$$ve(k) = \max\{ve(j) + \text{Weight}(v_j, v_k)\}, v_j \text{为} v_k \text{的任意前驱}$$



拓扑序列: **V1、V3、V2、V5、V4、V6**

$$ve(1)=0$$

拓扑序列只是为了让计算有条不紊的进行，用不同的拓扑序列也没关系

$$ve(5)=6$$

$$ve(4)=\max\{ve(2)+2, ve(3)+4\}=6$$

$$ve(6)=\max\{ve(5)+1, ve(4)+2, ve(3)+3\}=8$$

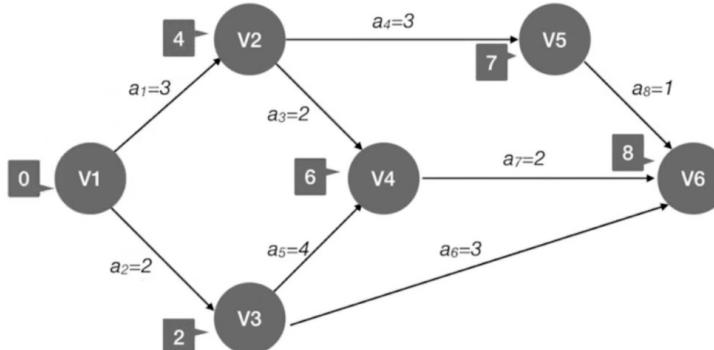
	V1	V2	V3	V4	V5	V6
ve(k)	0	3	2	6	6	8

### step2.求所有事件的最迟发生时间

按逆拓扑排序序列，依次求各个顶点的  $vl(k)$ :

$$vl(\text{汇点}) = ve(\text{汇点})$$

$$vl(k) = \min\{vl(j) - \text{Weight}(v_k, v_j)\}, v_j \text{为} v_k \text{的任意后继}$$



逆拓扑序列: **V6、V5、V4、V2、V3、V1**

$$vl(6)=8$$

$$vl(5)=7$$

$$vl(4)=6$$

$$vl(2)=\min\{vl(5)-1, vl(4)-2\}=4$$

$$vl(3)=\min\{vl(4)-4, vl(6)-3\}=2$$

$$vl(1)=0$$

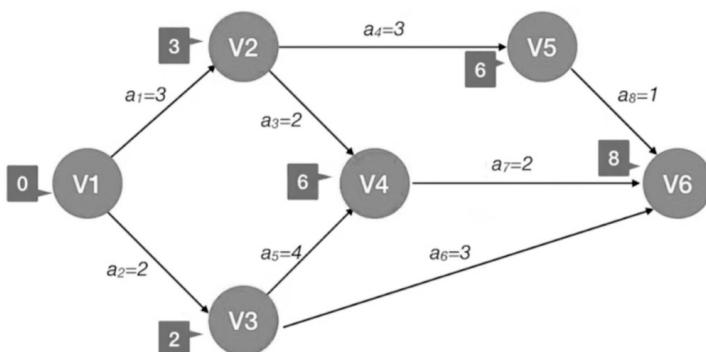
	V1	V2	V3	V4	V5	V6
ve(k)	0	3	2	6	6	8
vl(k)	0	4	2	6	7	8

### step3.求所有活动的最早发生时间

#### ③ 求所有活动的最早发生时间 $e(i)$

若边  $\langle v_k, v_j \rangle$  表示活动  $a_i$ ，则有  $e(i) = ve(k)$

	V1	V2	V3	V4	V5	V6
ve(k)	0	3	2	6	6	8
vl(k)	0	4	2	6	7	8



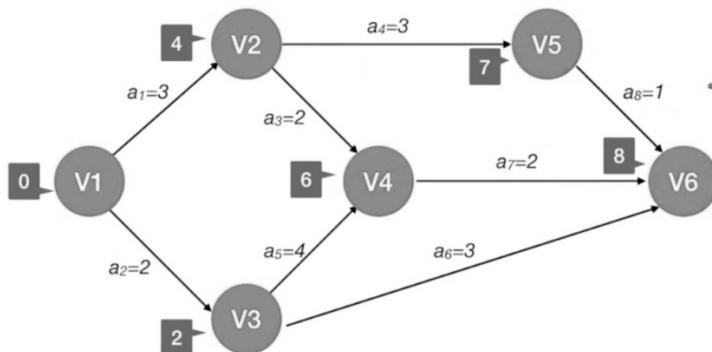
	a1	a2	a3	a4	a5	a6	a7	a8
e(k)	0	0	3	3	2	2	6	6

step4.求所有活动的最迟发生时间

④ 求所有活动的最迟发生时间  $l(i)$

若边  $\langle v_k, v_j \rangle$  表示活动  $a_i$ , 则有  $l(i) = v_l(j) - \text{Weight}(v_k, v_j)$

	V1	V2	V3	V4	V5	V6
$ve(k)$	0	3	2	6	6	8
$vl(k)$	0	4	2	6	7	8



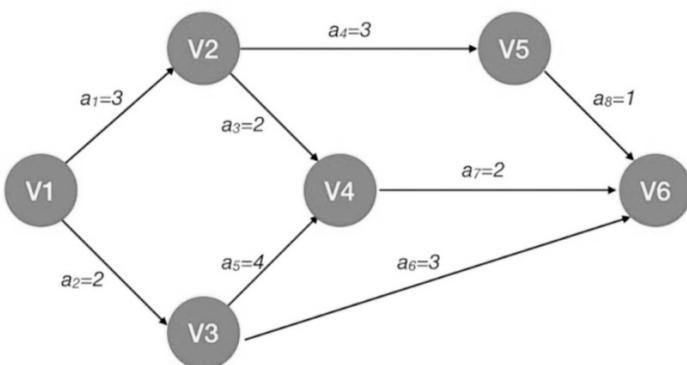
	a1	a2	a3	a4	a5	a6	a7	a8
$e(k)$	0	0	3	3	2	2	6	6
$l(k)$	1	0	4	4	2	5	6	7

step5.求所有活动的时间余量, 时间余量为 0 的为关键活动

⑤ 求所有活动的时间余量  $d(i)$

$$d(i) = l(i) - e(i)$$

	V1	V2	V3	V4	V5	V6
$ve(k)$	0	3	2	6	6	8
$vl(k)$	0	4	2	6	7	8



	a1	a2	a3	a4	a5	a6	a7	a8
$e(k)$	0	0	3	3	2	2	6	6
$l(k)$	1	0	4	4	2	5	6	7
$d(k)$	1	0	1	1	0	3	0	1