图

一、基本概念及术语：

图（Graph）是一种非线性结构。

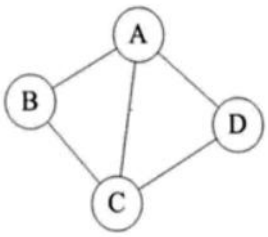
图的特点 { **顶点的前驱和后继个数无限制**；**顶点之间的关系是任意的**；**图中任意两个顶点之间都可能相关** } **多对多**

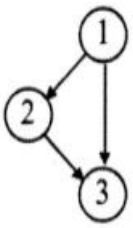
●图 G：由两个集合 V（G）和 E（G）所组成，记作 G=（V，E），其中 V（G）是图中顶点的非空有限集合，E（G）是图中边的有限集合。

注：图不能是空图，图的顶点集 V 一定非空，但边集 E 可以为空

●无向边、无向图：若顶点 vi 到 vj 之间没有方向,则称这条边为无向边，用无序对(vi,vj)来表示。如果图中任意两个顶点之间的边都是无向边，则称该图为无向图。

●有向边、有向图：若从顶点 vi 到 vj 的边有方向，则为有向边，也称为弧，用有序对< vi, vj> 来表示，vi 称为弧尾，vj 称为弧头。如果图中任意两个顶点之间的边都是有向边，则称该图为有向图。





●简单图：在图中,如果不存在顶点到其自身的边，且同一条边不重复出现，则称这样的图为简单图。数据结构仅讨论简单图。

●完全图（简单完全图）

无向图中边的取值范围：0≤e≤n(n-1)/2。（用 n 表示图中顶点数目，用 e 表示边的数目，且不考虑顶点到其自身的边。）

在无向图，如果任意两个顶点之间都存在边，则称该图为无向完全图。即共有 n（n-1）/2 条边。

有向图中弧的取值范围：0≤e≤n(n-1)。（用 n 表示图中顶点数目，用 e 表示弧的数目。且不考虑顶点到其自身的弧。）

在有向图中，如果任意两个顶点之间都存在方向相反的两条弧，则称该图为有向完全图。即有 n（n-1）条弧。

●稀疏图&稠密图：有很少的边或弧的图称为稀疏图，反之称为稠密图

●权:有些图的边或者弧具有与它相关的数字，这个相关的数称为权。

●带权图（又称网）：边上带有权值的图

●子图：如果图 G = (V, E) 和 G′= (V ′, E′)，满足：V ′⊆V 且 E′⊆E，则称 G′ 为 G 的子图。

注意：并非 V 和 E 的任何子集都能构成 G 的子图，因为这样的子集可能不是图。

●邻接点：若 (v, v′) 是一条边，则称顶点 v 和 v′互为邻接点， 或称 v 和 v′ 相邻接；称边 (v, v′) 依附于顶点 v 和 v′，或称 (v, v′) 与顶点 v 和 v′ 相关联。若 <v, v′> 是一条弧，则称顶点 v 邻接到 v′，顶点 v′ 邻接自顶点 v。并称弧 <v, v′> 与顶点 v 和 v′ 相关联。

●度、入度和出度

若（Vi，Vj）是 E（G）中的一条边，则称顶点 Vi 和 Vj 是邻接的.并称边（Vi，Vj）依附于顶点 Vi 和 Vj。所谓顶点的度，就是依附于该顶点的边数，记为 TD(V)。$\sum_{i=1}^{n} TD(Vi) = 2e$。在有向图中，顶点的入度为指向该顶点的边数，记为 ID(v)；顶点的出度为该顶点发出的边数,记为 OD(v)。$\sum_{i=1}^{n} ID(Vi)$）=$\sum_{i=1}^{n} OD(Vi) = e$。该顶点的入度和出度之和称为该顶点的度。

●路径：一个顶点序列 i1，i2.....ik 是图的一条路径，当且仅当边（i1，i2）（i2，i3）.....(ik-1,ik)都在图中。

●简单路径：在路径序列中，顶点不重复出现的路径称为简单路径。

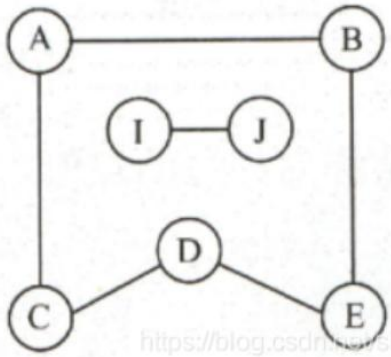
●回路（环）：第一个顶点和最后一个顶点相同的路径。

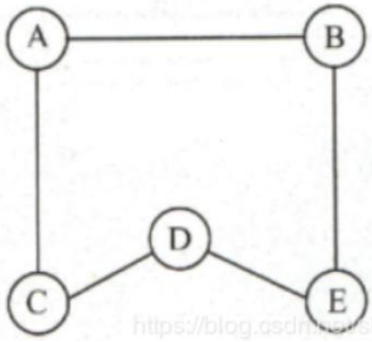
●简单回路：除第一个顶点和最后一个顶点外，其余顶点不重复出现的回路。

●连通、连通图、连通分量

在无向图中，两顶点有路径存在，就称为连通。若图中任意两个顶点都连通，则此图为连通图。无向图中的极大连通子图称为连通分量。

以下面两个图为例，右边的图是左边的图的连通分量，并且右边的图是连通图。左图中 I 与 J 也是连通的。

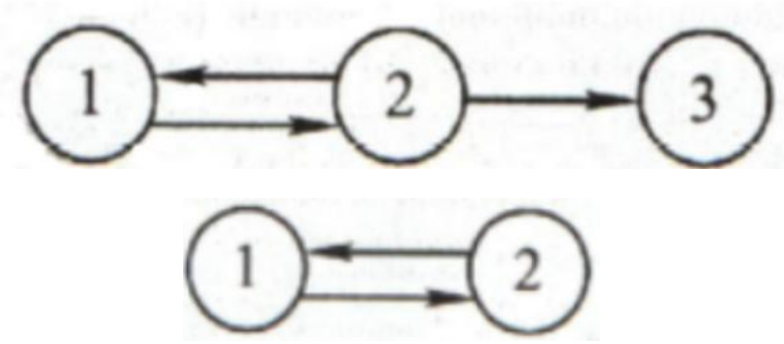




●强连通图、强连通分量

在有向图中，两个顶点两个方向都有路径，则这两个顶点是强连通的。若图中任意一对顶点都是强连通的，则此图称为强连通图。有向图中的极大强连通子图称为有向图的强连通分量。

以下面的图为例：下面的图就是一个强连通图，并且是上面的图的强连通分量。



▶对于 n 个顶点的无向图 G，若 G 是连通图，则最少有 n-1 条边；若 G 是非连通图，则最多可能有(n-1)(n-2)/2 条边。

对于 n 个顶点的有向图 G，若 G 是强连通图，则最少有 n 条边（形成回路）。

●生成树和生成森林

连通图的生成树是包含图中全部顶点的一个极小连通子图，若图中有 n 个顶点，则生成树有 n-1 条边。所以对于生成树而言，若砍去一条边，就会变成非连通图；若加上一条边则会形成一个回路。

在非连通图中，连通分量的生成树构成了非连通图的生成森林。

●距离：若两顶点存在路径，其中最短路径长度为距离。若不存在路径，则记距离为无穷。

●有向树：如果一个有向图恰有一个顶点的入度为 0 ，其余顶点的入度均为 1 ，则是一棵有向树。

二、图的存储结构（★重点★）

1.邻接矩阵

设 G = (V, E) 是具有 n 个顶点的图，顶点的顺序依次为 {v1, v2, …, vn}，则 G 的邻接矩阵是具有如下性质的 n 阶方阵：

$$A[i,j]=\begin{cases}1: & \text{若 } (v_i,v_j) \text{ 或} < v_i,v_j > \text{是 E(G)中的边或弧} \\ 0: & \text{若 } (v_i,v_j) \text{ 或} < v_i,v_j > \text{不是 } E(G) \text{中的边或弧}\end{cases}$$

带权图的邻接矩阵可定义为：

$$A[i,j]=\begin{cases}w_{ij}: & \text{若 } (v_i,v_j) \text{ 或} < v_i,v_j > \text{是 } E(G) \text{中的边或弧} \\ \infty: & \text{若 } (v_i,v_j) \text{ 或} < v_i,v_j > \text{不是 } E(G) \text{中的边或弧}\end{cases}$$

▶特点：

①无向图的邻接矩阵对称，可压缩存储；有 n 个顶点的无向图需存储空间为 n(n+1)/2。

②有向图邻接矩阵不一定对称；有 n 个顶点的有向图需存储空间为 n^2。

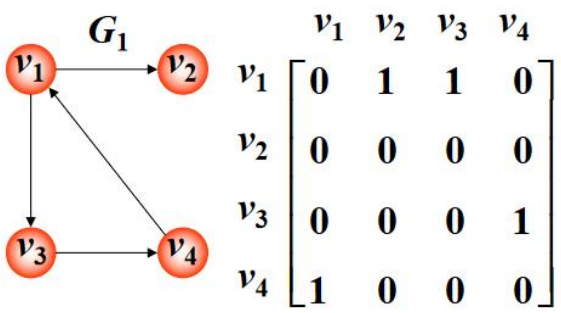
③无向图中顶点 vi 的度 TD(vi) 是邻接矩阵中第 i 行（或第 i 列）非零元素的个数。

④邻接矩阵的第 i 行非零元素的个数正好是顶点 i 的出度 OD（vi）；第 i 列非零元素的个数正好是顶点 i 的入度 ID（vi）。

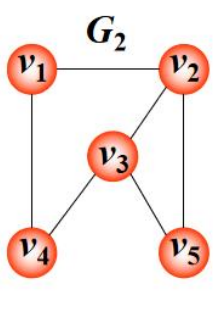
⑤稠密图适合使用邻接矩阵的存储表示。

⑥邻接矩阵的空间复杂度为 $O(n^2)$。

⑦一个图的邻接矩阵表示是唯一的。



	v_1	v_2	v_3	v_4
v_1	0	1	1	0
v_2	0	0	0	0
v_3	0	0	0	1
v_4	1	0	0	0

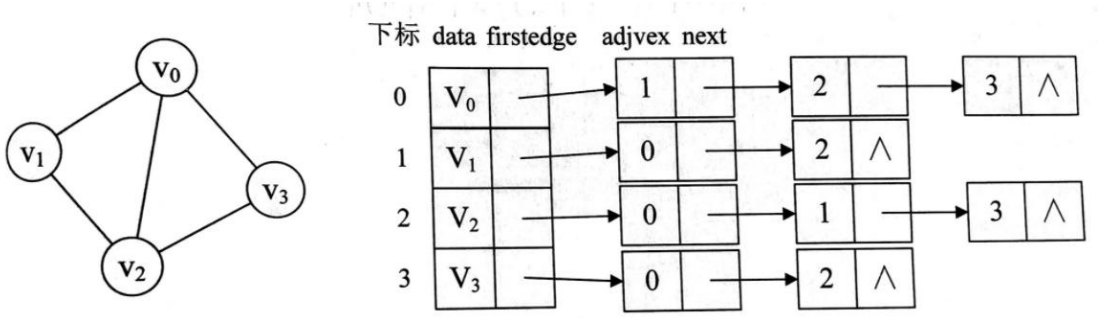


	v_1	v_2	v_3	v_4	v_5
v_1	0	1	0	1	0
v_2	1	0	1	0	1
v_3	0	1	0	1	1
v_4	1	0	1	0	0
v_5	0	1	1	0	0

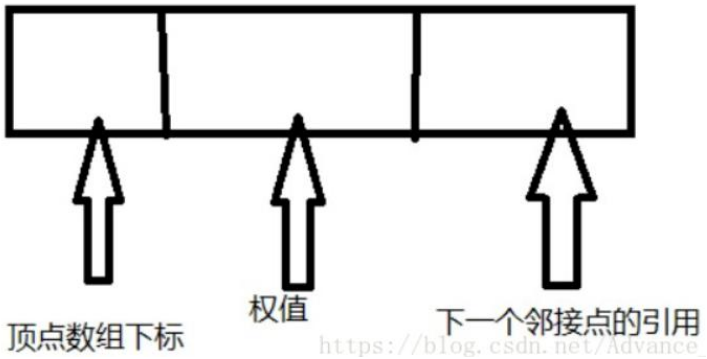
2.邻接表

图的邻接表存储结构是一种顺序分配和链式分配相结合的存储结构，包括两个部分：一部分是顶点表，另一部分是边表。

下图表示无向图邻接表的转换：



值得注意的是带权图需要多记录一项属性（权值），对边表结点类型加一个域，来存储权值，结构如下



▶特点：

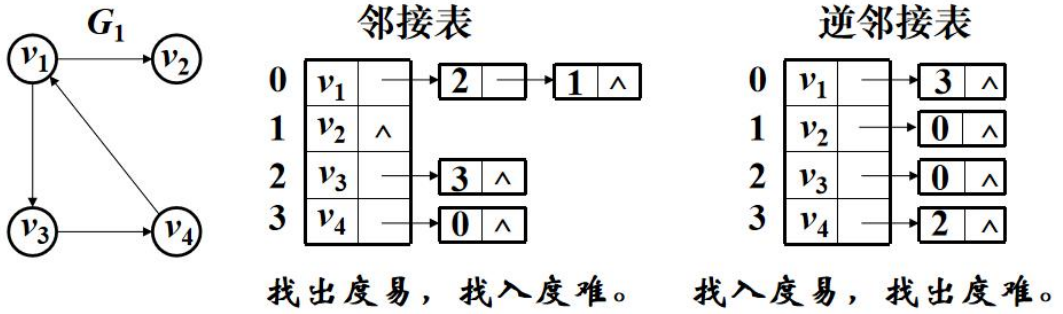
①若 G 为无向图，则所需的存储空间为 O（|V|+2|E|）；若 G 为有向图，则所需的存储空间 O(|V|+|E|)

②对于稀疏图，采用邻接表表示将极大地节省存储空间。

③在邻接表中，给定一顶点，能很容易地找出它的所有邻边，因为只需要读取它的邻接表。在邻接矩阵中，相同的操作则需要扫描一行，花费的时间为 O（n）。但是，若要确定给定的两个顶点间是否存在边，则在邻接矩阵中可以立刻查到，而在邻接表中则需要在相应结点对应的边表中查找另一结点，效率较低。

④在有向图的邻接表表示中，求一个给定顶点的出度只需计算其邻接表中的结点个数；但求其顶点的入度则需要遍历全部的邻接表。

⑤图的邻接表表示并不唯一，因为在每个顶点对应的单链表中，各边结点的链接次序可以是任意的，它取决于建立邻接表的算法及边的输入次序。



特点：

- 顶点 v_i 的**出度**为第 i 个单链表中的结点个数。
- 顶点 v_i 的**入度**为整个单链表中邻接点域值是 i -1 的结点个数。
- 顶点 v_i 的**入度**为第 i 个单链表中的结点个数。
- 顶点 v_i 的**出度**为整个单链表中邻接点域值是 i -1 的结点个数。

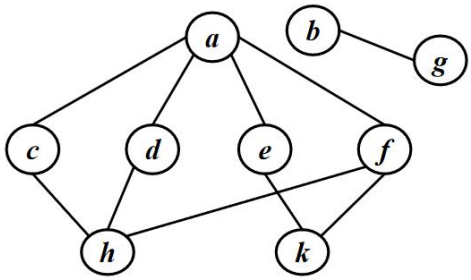
三、图的遍历（★重点★）

1.广度优先搜索（BFS）

广度优先搜索类似于二叉树的层序遍历算法。

☆方法：从图的某一顶点出发，首先依次访问该顶点的所有邻接顶点 Vi1, Vi2, …, Vin，再按这些顶点被访问的先后次序依次访问与它们相邻接的所有未被访问的顶点，重复此过程，直至所有顶点均被访问为止。若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作为始点，重复上述过程，直至图中所有顶点都被访问到为止。

例：



从顶点a出发的两个BFS访问次序(结果不唯一):

a c d e f h k b g

a c d e f h k g b

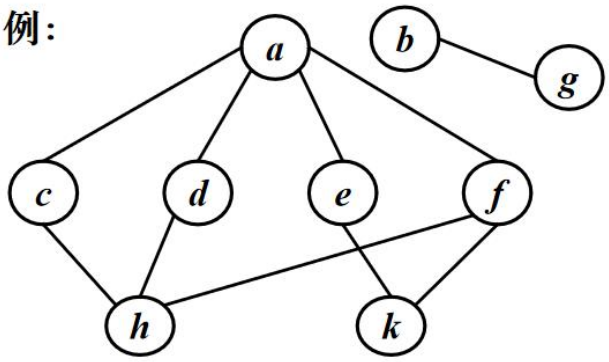
一给定图的邻接矩阵表示唯一，所以广度遍历序列、广度优先生成树也是唯一的；但邻接表表示不唯一，故广度遍历序列、广度优先生成树也不唯一。

2.深度优先搜索（DFS）

深度优先搜索类似于树的先序遍历。

☆方法：首先访问图中某一起始顶点 v，然后由 v 出发，访问与 v 邻接且未被访问的第一个顶点 w1，再访问与 w1 邻接且未被访问的第一个顶点 w2……重复上述过程。当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直至图中所有顶点均被访问过为止。若最后图中尚有顶点未被访问，则再选其中一个顶点作为起始顶点并访问之，重复上述过程，直到所有顶点都被访问。

例：



从顶点a出发的DFS访问次序（不唯一）：

a c h d f k e b g

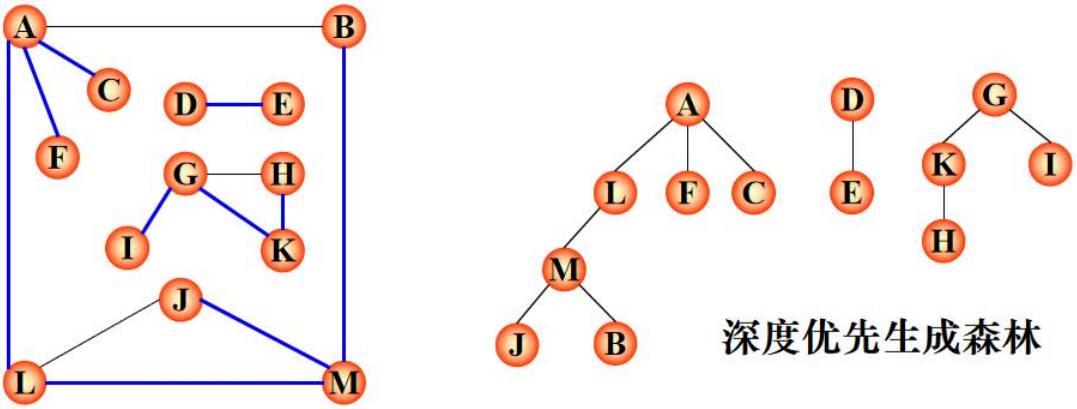
a c h d f k e g b

a c h f k e d b g

一给定图的邻接矩阵表示唯一，所以深度遍历序列、深度优先生成树也是唯一的；但邻接表表示不唯一，故深度遍历序列、深度优先生成树也不唯一。

对无向图进行 BFS/DFS 遍历：调用 BFS/DFS 的次数=连通分量数；对于**连通图**，只需**调用一次 BFS/DFS**。

对有向图进行 BFS/DFS 遍历：调用次数要具体分析；**强连通图**任一顶点出发都只需**调用一次 BFS/DFS**。



四、最小生成树（★重点★）

最小生成树：给定一个无向网络，在该网的所有生成树中，使得各边权数之和最小的那棵生成树称为该网的最小生成树，也叫最小代价生成树。

▶性质：

①最小生成树不是唯一的，即最小生成树的树形不唯一，R 中可能有多个最小生成树。当图 G 中的各边权值互不相等时，G 的最小生成树是唯一的；若无向连通图 G 的边数比顶点数少 1，即 G 本身是一棵树时，则 G 的最小生成树就是它本身。

②最小生成树的边的权值之和总是唯一的，虽然最小生成树不唯一，但其对应的边的权值之和总是唯一的，而且是最小的。

③最小生成树的边数为顶点数减 1。

1.普里姆（prim）算法（★重点★）

★算法步骤：

第一步：设图中所有顶点的集合为 v，u 代表已经加入最小生成树的顶点的集合，v 代表未加入最小生成树的顶点的集合，最小生成树从某点 s 开始，因此 u={s}，v={V-u}

第二步：在两个集合 u,v 中选择一条代价最小的边，将在 v 中连接这条边的那个顶点 x 加入到最小生成树顶点集合 u 中，并且更新与 x 相连的所有邻接点

第三步：重复上述步骤，直到最小生成树顶点集合 u 中有 n 个顶点为止

2.克鲁斯卡尔算法（★重点★）

★算法步骤：每次选择权值最小的边，使边的两端连通（若边的两端已连通则不选择），直到图中有 n-1 条边（图中有 n 个顶点）

五、最短路径（了解）

- Dijkstra 算法求单源最短路径问题（不适合带负权值的图）
- Floyd 算法求各顶点之间最短路径问题（可用于负权图，但不能解决带“负权回路”的图）

Dijkstra 算法步骤： 初始时令 S={v_0}，T={其余顶点}。

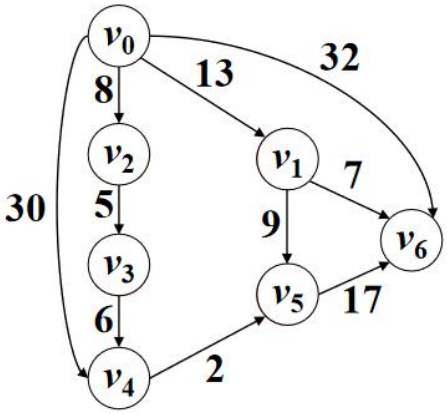
T 中顶点对应的距离值用辅助数组 D 存放。

$D[i]$ 初值： 若 $<v_0, v_i>$ 存在，则为其权值；否则为∞。

从 T 中选取一个其距离值最小的顶点 v_j，加入 S。 $D[j]=\underset{i}{\mathrm{Min}}\{D[i]\mid v_i\in T\}$

对 T 中顶点的距离值进行修改：若加进 v_j 作中间顶点，从 v_0 到 v_i 的距离值比不加 v_j 的路径要短，则修改此距离值。

重复上述步骤，直到 S = V 为止。

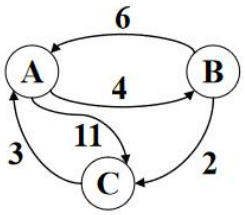


终点	从 v_0 到各终点的最短路径及长度					
	i=1	i=2	i=3	i=4	i=5	i=6
v_1	13	13				
v_2	8					
v_3	∞	13	13			
v_4	30	30	30	19		
v_5	∞	∞	22	22	21	21
v_6	32	32	20	20	20	
v_j	v_2	v_1	v_3	v_4	v_6	v_5
s	8	13	8+5	8+5+6	13+7	8+5 +6+2

Floyd求最短路径步骤：

初始时设置一个 n 阶方阵，令其对角线元素为 0，若存在弧$<v_i,v_j>$，则对应元素为权值；否则为∞。

逐步试着在原直接路径中增加中间顶点，若加入中间顶点后路径变短，则修改之；否则，维持原值。所有顶点试探完毕，算法结束。



初始：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$ 路径：

	AB	AC
BA		BC
CA		

加入A：$\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	AC
BA		BC
CA	CAB	

加入B：$\begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	ABC
BA		BC
CA	CAB	

加入C：$\begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$ 路径：

	AB	ABC
BCA		BC
CA	CAB	

六、有向无环图（DAG）描述表达式（了解）

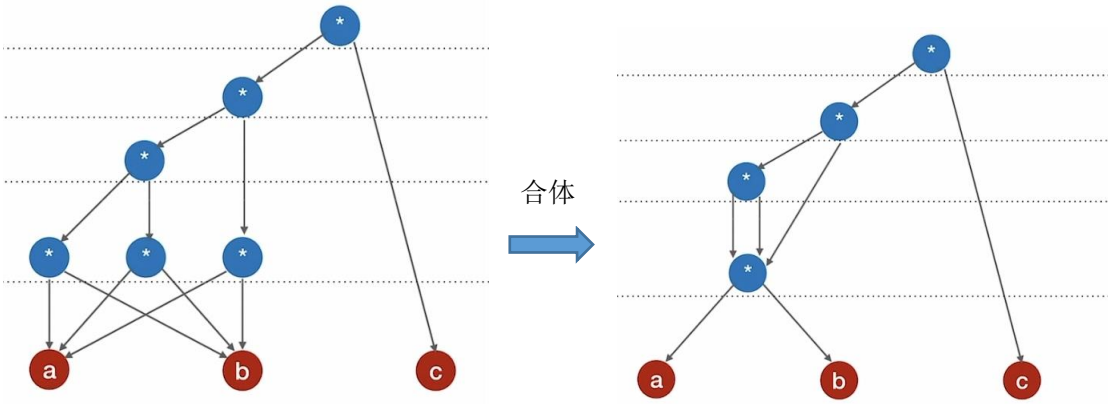
步骤：

1.把各个操作数不重复的排成一排

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”（左根右，运算符为根）

4.自底向上逐层检查同层的运算符是否可以合体



七、拓扑排序（★重点★）

1.AOV 网：在有向图中以顶点表示“活动”，用有向边表示“活动”之间的优先关系，这样的有向图称为以顶点表示“活动”的网（Activity On Vertex Network），简称 AOV 网。

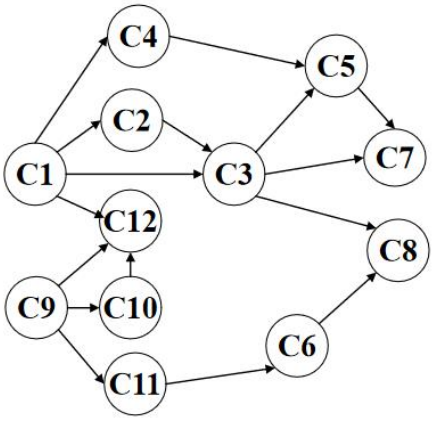
2.对 AOV 网进行拓扑排序的步骤：

①在网中选择一个没有前趋的顶点（入度为 0）且输出。

②在网中删去该顶点和从该顶点发出的全部有向边。

③重复上述两步，直到网中不存在没有前趋的顶点为止。

这样操作结果有两种：一种是网中全部顶点均被输出，说明网中不存在有向回路；另一种是网中顶点未被全部输出，剩余的顶点均有前趋顶点，说明网中存在有向回路。



拓扑序列：

C1, C2, C3, C4, C5, C7, C9, C10, C11, C6, C12, C8

C9, C10, C11, C6, C1, C12, C4, C2, C3, C5, C7, C8

一个AOV网的拓扑序列不是唯一的

逆拓扑排序：

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。

这样的操作结果与拓扑排序类似。

八、关键路径（了解）

在带权有向图 G 中，以顶点表示事件，边表示活动，权表示活动持续的时间，则此带权有向图称为用边表示活动的网（Activity On Edge network），简称 AOE 网。

在 AOE 网仅有一个入度为 0 的顶点，称为开始顶点（源点），它代表整个工程的开始；也仅存在一个出度为 0 的顶点，称为结束顶点（汇点），它代表整个工程的结束。

AOV 网和 AOE 网都是有向无环图（DAG），AOE 网的边有权值，AOV 网的边无权值。

关键路径——从源点到汇点的所有路径中，具有最大路径长度的路径

关键活动——关键路径上的活动

事件 vj 的最早发生时间 ve(j) ——决定所有从 vj 开始的活动能够开工的最早时间（多个入边取最大)

事件 vj 的最迟发生时间 vl(j) ——指在不推迟整个工程完成的前提下，事件最迟开始时间(多个出边取最小)

活动 ai 的最早开始时间 e(i) ——指该活动弧的起点所表示事件的最早发生时间

活动 ai 的最迟开始时间 l(i) ——指该活动弧的终点代表事件的最迟发生时间减去该活动所需的时间

▶算法步骤：

①从源点出发对 AOE 网中的顶点进行拓扑排序，如果得到的拓扑序列顶点个数小于网中顶点数，则说明网中有环存在，不能求关键路径，终止算法。否则，令 ve(源点)=0，按拓扑有序求出其余顶点的最早发生时间 ve(i)。

②从汇点出发，令 vl(汇点)=ve(汇点)，按照逆拓扑序列求其他顶点的最晚发生时间 vl(i)。

③由各顶点的最早发生时间 ve(i)和最晚发生时间 vl(i)，求出每个活动 ai 的最早开始时间 e(i)和最晚开始时间 l(i)。

④找出所有满足条件 e(i)=l(i)的活动 ai，构成关键路径。

▶对于关键路径，需要注意以下几点：

①关键路径上的所有活动都是关键活动，它是决定整个工程的关键因素，因此可通过加快关键活动来缩短整个工程的工期。但也不能任意缩短关键活动，因为一旦缩短到一定的程度，该关键活动就可能会变成非关键活动。

②网中的关键路径并不唯一，且对于有几条关键路径的网，只提高一条关键路径上的关键活动速度并不能缩短整个工程的工期，只有加快那些包括在所有关键路径上的关键活动才能达到缩短工期的目的

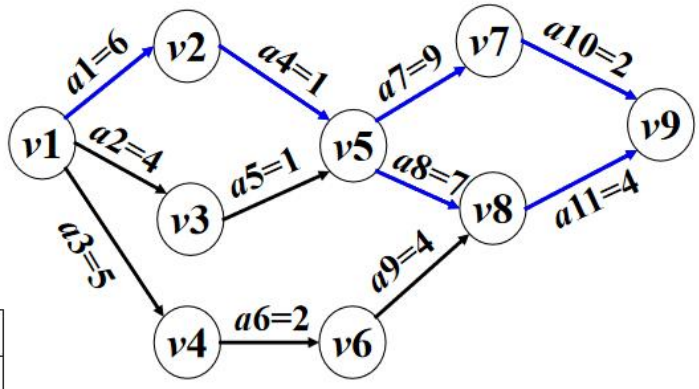
求关键路径步骤：

求 *ve*(*i*)、*vl*(*j*)

求 *e*(*i*)、*l*(*i*)

计算 *l*(*i*) - *e*(*i*)

拓扑排序：V1,V2,V3,V4,V5,V6,V7,V8,V9
逆拓扑排序:V9,V8,V7,V6,V5,V4,V3,V2,V1



活动	*e*	*l*	*l* – *e*
*a*1	0	0	0 ✓
*a*2	0	2	2
*a*3	0	3	3
*a*4	6	6	0 ✓
*a*5	4	6	2
*a*6	5	8	3
*a*7	7	7	0 ✓
*a*8	7	7	0 ✓
*a*9	7	10	3
*a*10	16	16	0 ✓
*a*11	14	14	0 ✓

顶点	*ve*	*vl*
*v*1	0	0
*v*2	6	6
*v*3	4	6
*v*4	5	8
*v*5	7	7
*v*6	7	10
*v*7	16	16
*v*8	14	14
*v*9	18	18