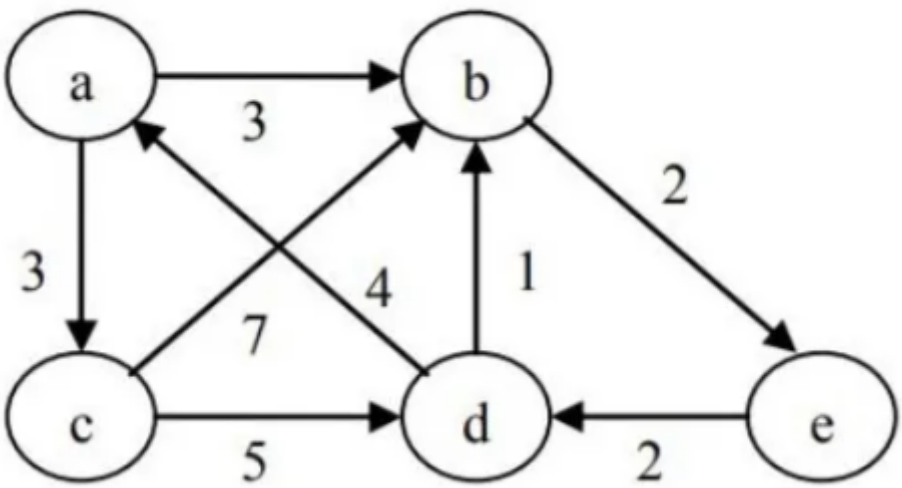
图论理论

理论

图：由点(node，或者vertex)和连接点的边(edge)组成。图是点和边构成的网。



树，即连通无环图

名词

图上名词顶点连边的数量就是顶点的**度数**。

路径：从起点依次沿着边移动到下一个点，直到终点所经过的点和/或边。

圈/环：从一个点出发到自己的路径。

两个顶点中间有不止一条边，这被称为重边。

针对每一条边的属性值被称为**边权**，顶点也可以有点权。

每条道路只能单向通行，即边是单向的，被称为**有向图**；

可以双向通行，被称为**无向图**。

对于有向图：

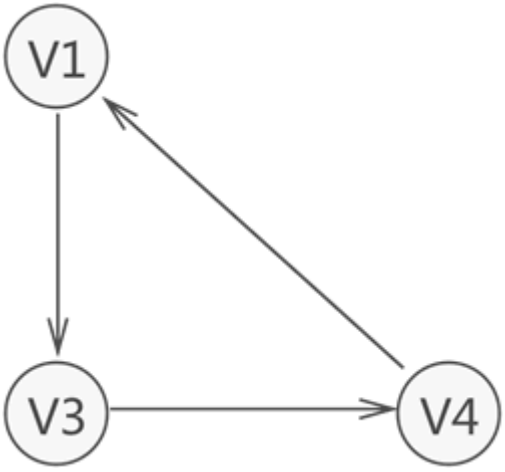
一个顶点向别的顶点连边的条数称作这个结点的**出度**，别的顶点连边到一个顶点的条数称作这个结点的**入度** 。

如果它没有环，则称为有向无环图。

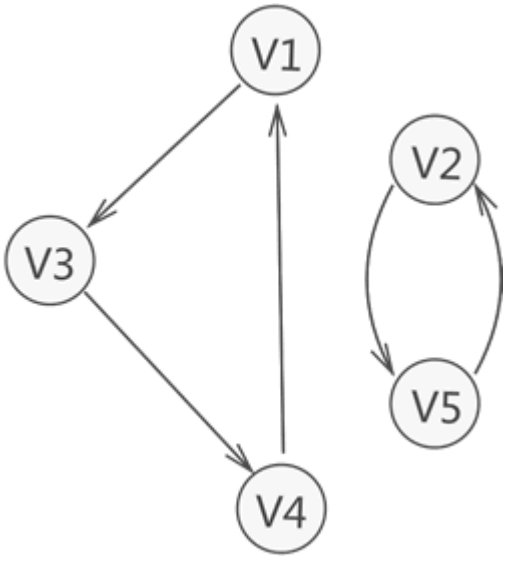
强连通：有向图中，两个顶点至少存在一条路径。

有向图中，**若任意两个顶点** Vi 和 Vj，满足从 Vi 到 Vj 以及从 Vj 到 Vi 都连通，也就是都含有**至少一条通路**，则称此有向图为强连通图。

强连通图：



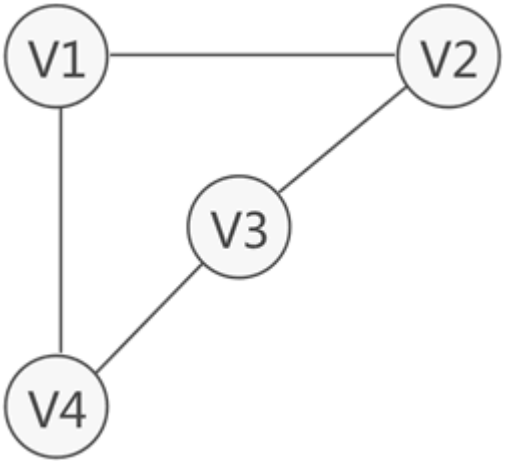
强连通分量：若有向图本身不是强连通图，但其包含的最大连通子图具有强连通图的性质，则称该子图为强连通分量。



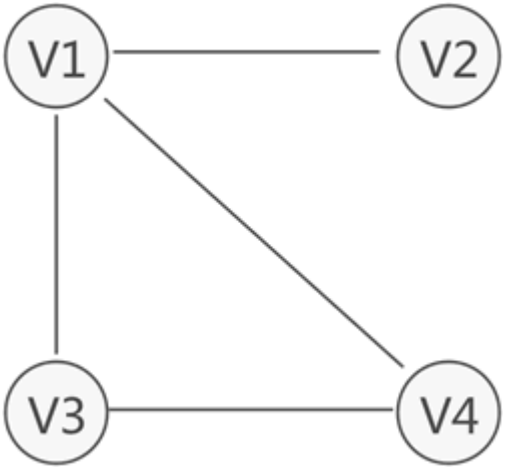
如图 5所示，整个有向图虽不是强连通图，但其含有两个强连通分量。

对于无向图：

连通：图中从一个顶点到达另一顶点，若存在至少一条路径，则称这两个顶点是连通着的。



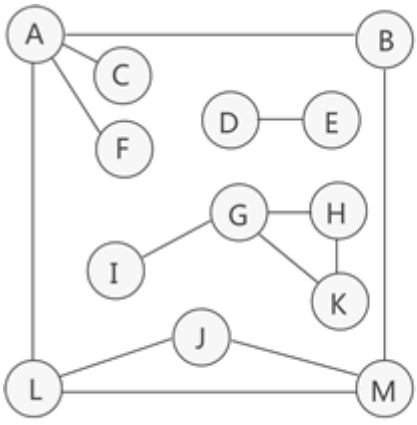
连通图：无向图中，如果任意两个顶点之间都能够连通，则称此无向图为连通图。



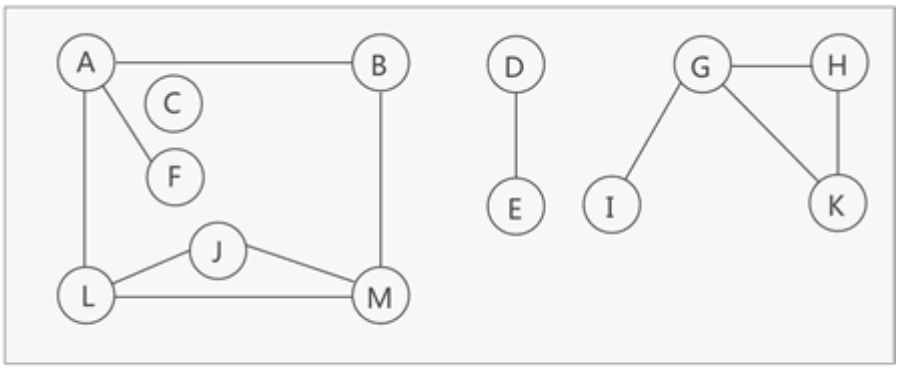
连通分量：若无向图不是连通图，但图中存储某个子图符合连通图的性质，则称该子图为连通分量。

由图中部分顶点和边构成的图为该图的一个子图，但这里的**子图指的是图中"最大"的连通子图（也称"极大连通子图"）**。

图 3a) 中的无向图不是连通图，但可以将其分解为 3 个"最大子图"（图 3b)），它们都满足连通图的性质，因此都是连通分量。



a) 非连通图



b) 连通分量

图的储存

邻接矩阵：

无向图：G[i][j] = G[j][i]。

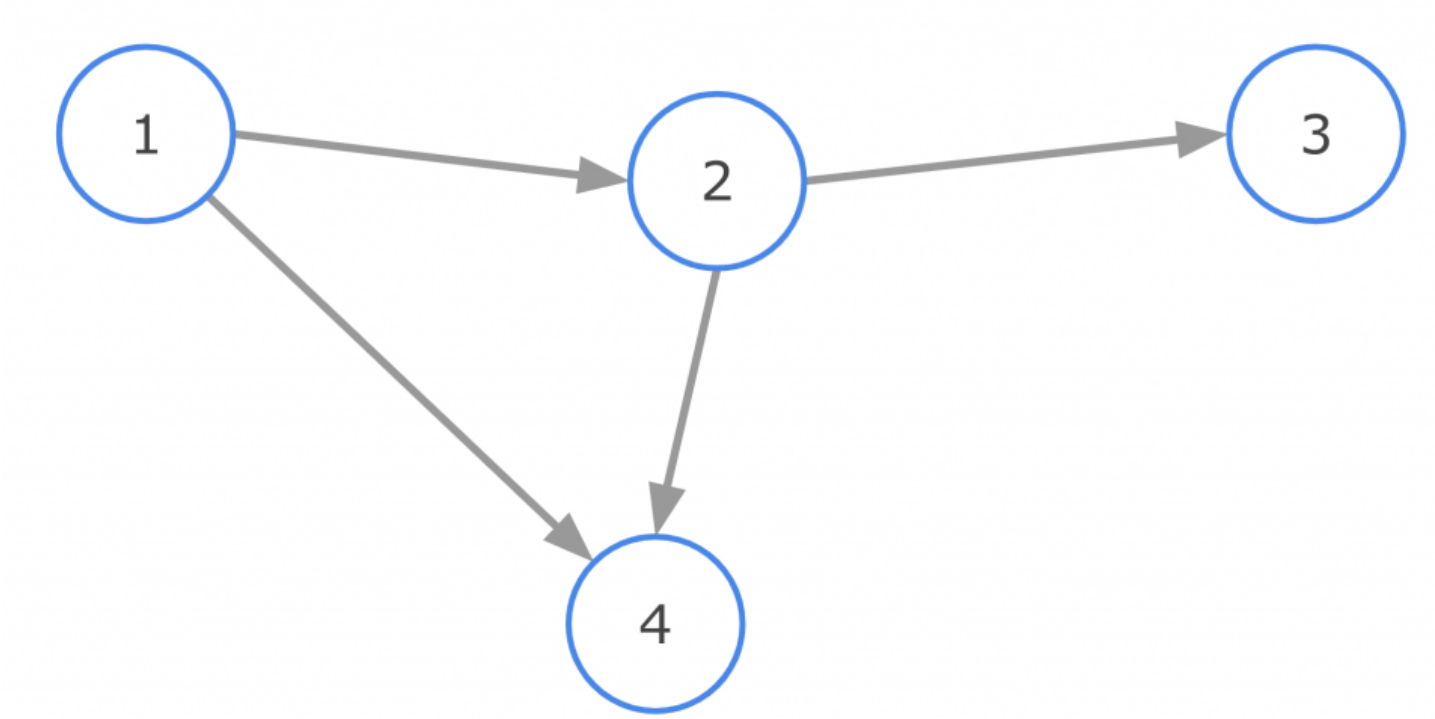
有向图：G[i][j] != G[j][i]。

权值：G[i][j]存结点i到j的边的权值。

例如 G[1][2] = 3，G[2][1] = 5等等。用 G[i][j] = INF表示 i，j 之间无边。

优点：适合稠密图；编码非常简短；对边的存储、查询、更新等操作又快又简单。

缺点：存储复杂度O(V2)太高。V=10000时，空间100M。不能存储重边。



```
#include<iostream>
using namespace std;

const int N = 105;
int G[N][N];

int main(){

    G[1][2] = 1;//1表示有边，或者一个具体的指表示边权
    G[1][4] = 1;
    G[2][4] = 1;
    G[2][3] = 1;

    return 0;
}
```

邻接表(vector)：

邻接表：vector e[N];

应用场景：大稀疏图。

优点：存储效率非常高，存储复杂度O(V+E)；

能存储重边。

```
#include<iostream>
#inlcude<vector>
using namespace std;

const int N = 105;
struct Node{
    int v;//表示与v有边
    //int v,w;//表示与v有边，边权为w
}

vector<Node>G[N];

int main(){

    //G[1][2] = 1;//1表示有边，或者一个具体的指表示边权
    //G[1][4] = 1;
    //G[2][4] = 1;
    //G[2][3] = 1;
    G[1].push_back({2});
    G[1].push_back({4});
    G[2].push_back({4});
    G[2].push_back({3});

    return 0;
}
```