程序设计实践（一）

哈尔滨工业大学 计算机学院
任课教师：孙大烈教授
助教：付万增

图论

一、图论基础
二、最短路
三、最小生成树

一.图的基本概念

1. 图的定义

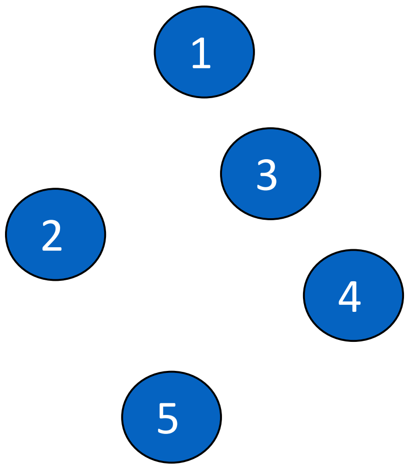
图是由一个顶点的集合V和一个顶点间关系的集合E组成：

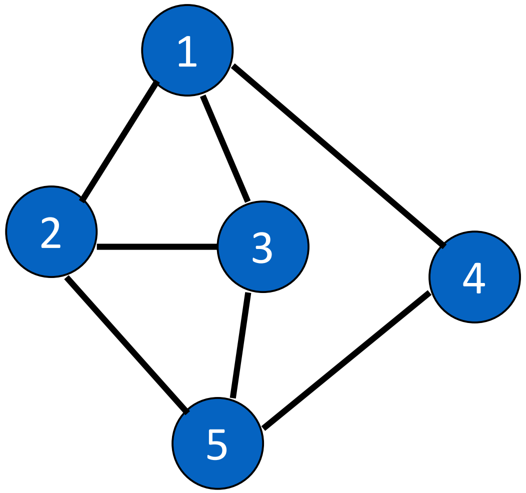
记　G=（V，E）

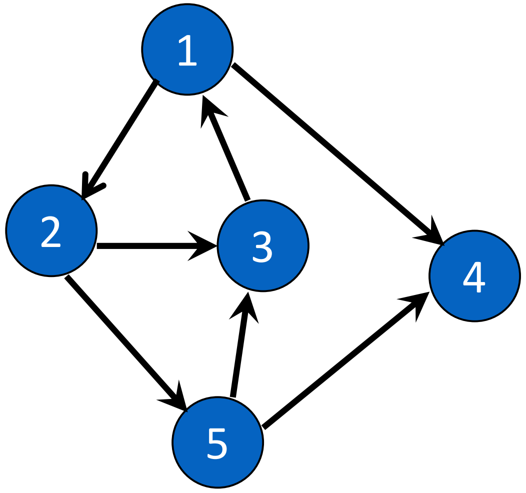
其中，　V：顶点的有限非空集合。

E：顶点间关系的有限集合（边集）。

存在一个结点v，可能含有多个前驱结点和后继结点。







2、无向图和有向图

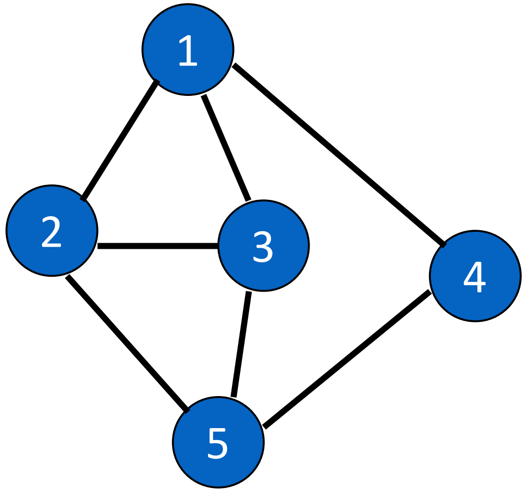
无向图：

在图G=（V，E）中，如果对于任意的顶点a，b∈V，

当(a，b)∈E时，必有（b，a）∈E（即关系R对称），此图称为无向图。

无向图中用不带箭头的边表示顶点的关系

V={1， 2， 3， 4， 5}
E={(1， 2),(1， 3),(1， 4),(2,3),(2,5),(3， 5),(4,5)}



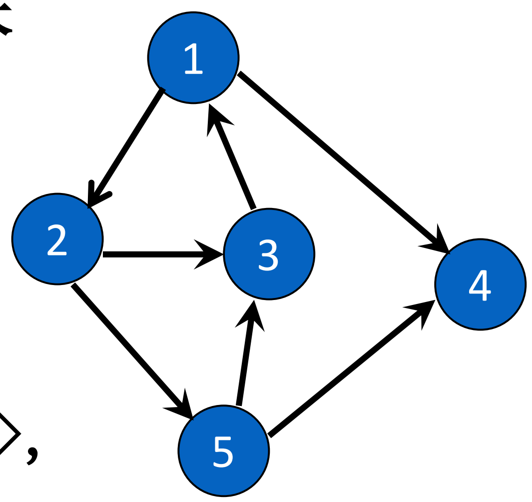
简而言之，每条边都是双向的。

2、无向图和有向图

有向图：

如果对于任意的顶点a，b∈V，当(a，b)∈E时 ，(b，a)∈E未必成立，则称此图为有向图。

在有向图中，通常用带箭头的边连接两个有关联的结点。



V={1，2，3，4,5}

E={<1，2>，<1，4>，<2，3>，<2，5>，<3，1>,<5，3>,<5，4>}

简而言之，每条边不一定是双向的。

3、顶点的度、入度和出度

无向图：顶点v的度是指与顶点v相连的边的数目D(v)。D(2)=3

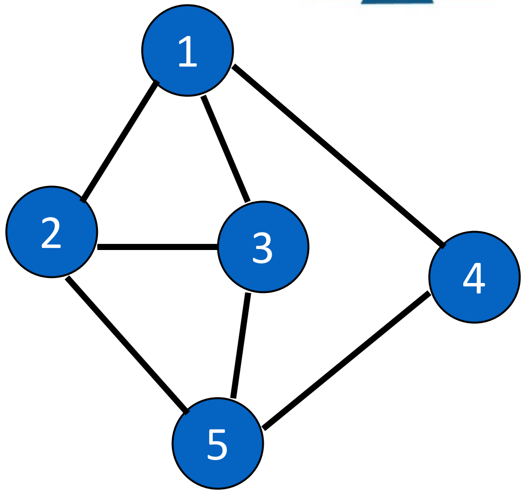
有向图：

入度——以该顶点为终点的边的数目和 . ID(3)=2

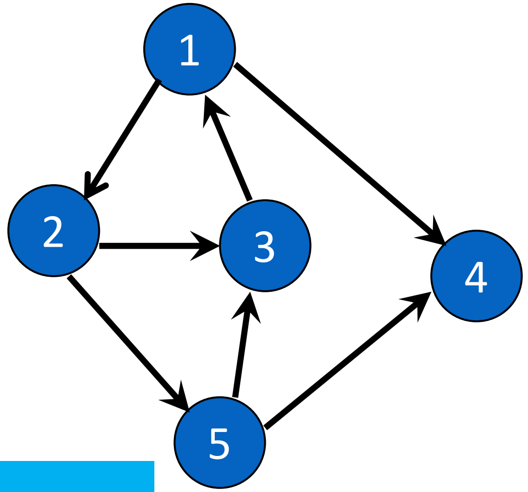
出度——以该顶点为起点的边的数目和 . OD(3)=1

度数为奇数的顶点叫做奇点，度数为偶数的点叫做偶点。

度：等于该顶点的入度与出度之和。 D(5)=ID(5)+OD(5)=1+2=3



结论：图中所有顶点的度 = 边数的两倍(不论有向图或无向图)



$$\sum_{i=1}^{n} D(v_i) = 2*e$$

4.其他

路径：起点a到终点b的顶点序列，相邻两个点间必须存在路径

简单路径：除起点a和终点b可以相同外，其与点均不相同的路径

回路（环）：a = b的简单路径成为回路

连通图：图中任意两个顶点均存在至少一条路径

连通分量：无向图中的极大连通子图

在有向图中分别对应着强连通、强连通图、强连通分量

更严谨的定义请看PDF!

带权图：图中边上挂有权值的图

二.图的存储结构

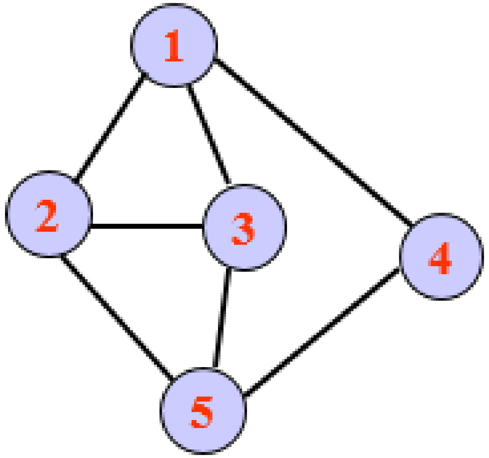
1. 邻接矩阵

邻接矩阵是表示结点间相邻关系的矩阵。若G=（V，E）是一个具有n个结点的图，则G的邻接矩阵是如下定义的二维数组 a[1..n,1..n]。

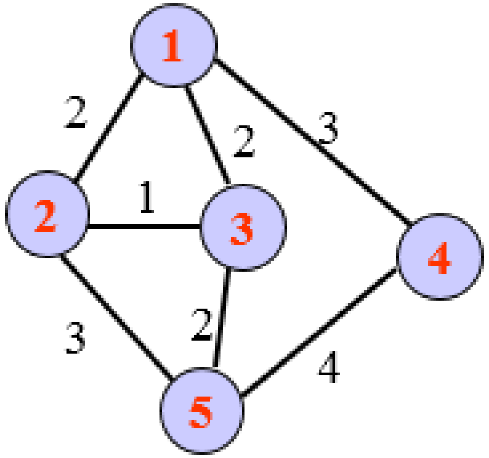
$$a[i,j]=\begin{cases}1\ (\text{或权值})\text{：无向图：有边}(i,j)\text{和边}(j,i) \\ \qquad\qquad\text{有向图：有边}<i,j> \\ 0:\qquad i\ \text{到}\ j\ \text{无边}\end{cases}$$

注意：当图有重边时，邻接矩阵法不能用

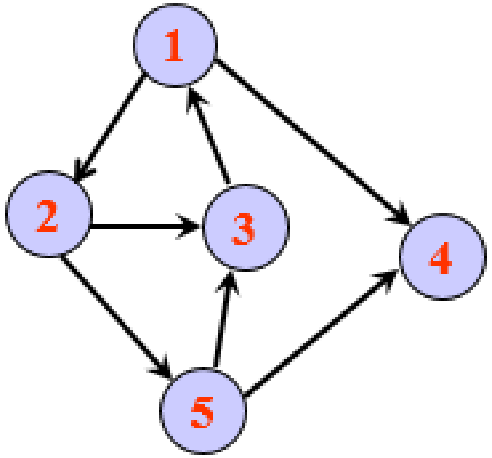
1. 邻接矩阵实例



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



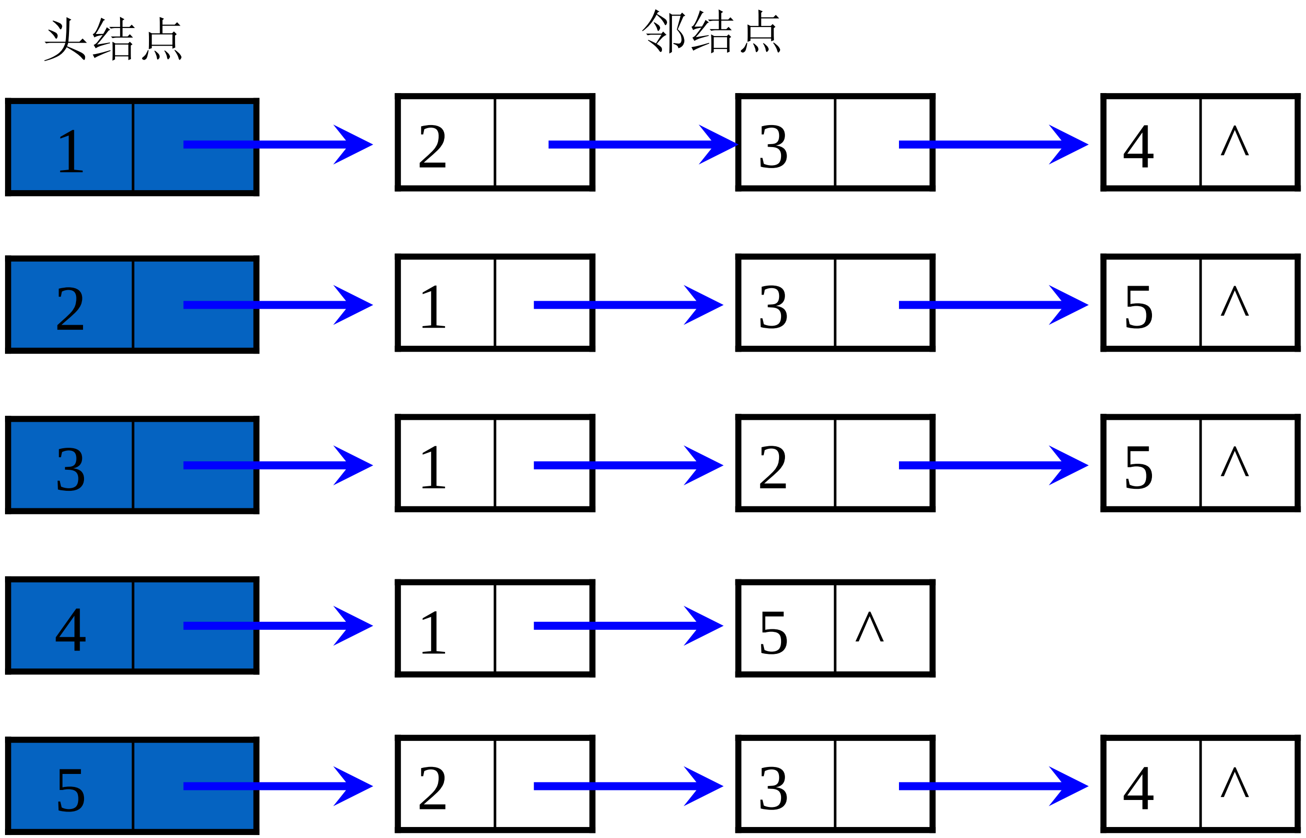
	1	2	3	4	5
1	0	2	2	3	0
2	2	0	1	0	3
3	2	1	0	0	2
4	3	0	0	0	4
5	0	3	2	4	0

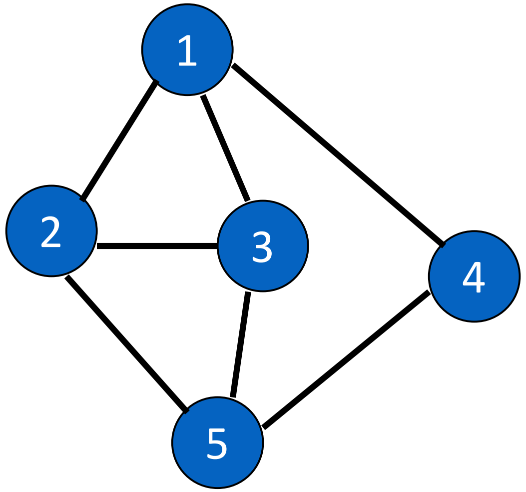


	1	2	3	4	5
1	0	1	0	1	0
2	0	0	1	0	1
3	1	0	0	0	0
4	0	0	0	0	0
5	0	0	1	1	0

2. 邻接表

结点	邻接点指针





思想：把与一个点的相邻的所有点集中在一起存储

2.邻接表实现方式

（1） 指针　　（2） 边集数组　　（3） 直接使用STL中的Vector

(1)指针存储：使用C++的指针，最直接的想法，但动态申请内存可能会耗时，不推荐；

(2)边集数组：用一维数组存储图中所有边，通过Head数组寻找起点为A的所有边（可以直接调用模板）；

(3)直接使用Vector：C++自带，最常用最方便，必须会；但是STL库中的方法比较慢，有可能会超时（如果超时可以考虑方法2）。

2、邻接表 ：

边集数组使用方法：

定义：

```
struct Edge{
    int x, y, w, next;
}e[M];
int head[N];
```

M：大于图中边的总数

N：大于顶点的总数

加边(相当于将新的边节点插在链的前端)：

```
inline void addEdge(int x,int y,int w)
{
    tot++; e[tot].x = x; e[tot].y = y; e[tot].w = w;
    e[tot].next = head[x]; head[x] = tot;
}
```

2、邻接表 ：

边集数组使用方法：

初始化：memset(head, -1, sizeof(head)), tot = 0

利用Head数组遍历与x相邻的所有点：

```
k = head[x]
while(k != -1)
        {
                做你想做的 //起点：e[k].x 终点：e[k].y 边权：e[k].w
                k = e[k].next;
        }
```

2、邻接表 ：

Vector使用方法：

程序开始引入库文件 ：#include <vector>

声明： vector<int> a[N]; //int也可以是其他数据类型或自定义类型

加边方式：a[x].push_back(y); //相当于与a[x]相连的点存成一个链

遍历方式：

```
for(int i = 0; i < a[x].size(); i++){
        a[x][i]即为与a[x]相连的第i个点
}
```

优点：动态开辟内存，1,2,4,8… ，从而不必担心会爆内存

这只是最基本的用法，Vector功能还有很多，具体的自行上网百度。

三.图的遍历方式

图的遍历

给出一个图G，从某一个初始点出发，按照一定的搜索方法对图中的每一个结点访问仅且访问一次的过程。

访问结点：处理结点的过程。如输出、查找结点的信息。

按照搜索方法的不同，通常有两种遍历方法：

1、深度优先搜索dfs

2、广度优先搜索bfs

1、深度优先搜索（DFS）：

遍历算法（递归过程）：

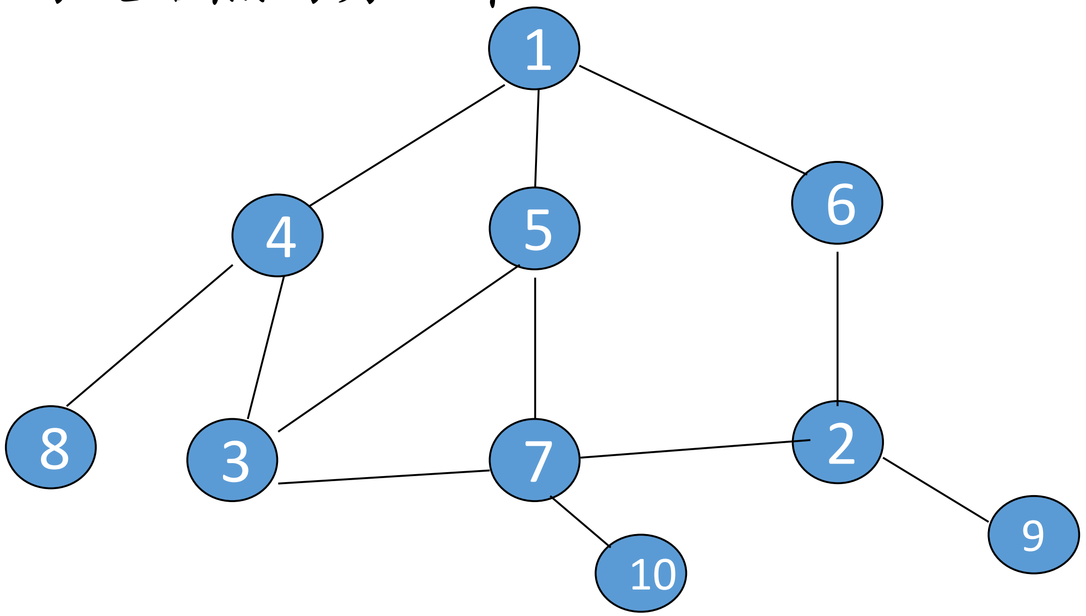
1)从某一初始出发点i开始访问：输出该点编号；并对该点作被访问标志（以免被重复访问）。

2)再从i的其中一个未被访问的邻接点j作为初始点出发继续深搜。

当i的所有邻接点都被访问完，则退回到i的父结点的另一个邻接点k再继续深搜。

直到全部结点访问完毕

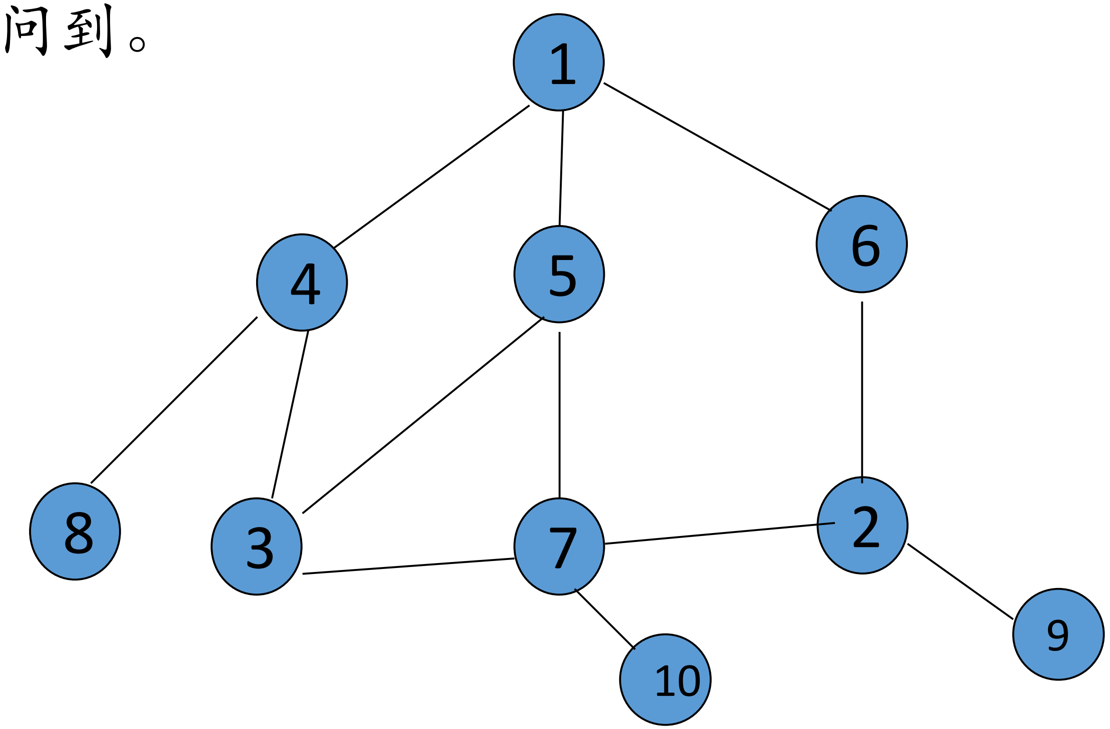
遍历序列不唯一，跟存边的顺序有关



2、广度优先搜索（BFS）：

按层次遍历：

从图中某结点i出发，在访问了i之后依次访问i的各个未曾访问的邻接点，然后分别从这些邻接点出发按广度优先搜索的顺序遍历图，直至图中所有可被访问的结点都被访问到。



遍历序列也不唯一，跟存边的顺序有关

需要使用**队列**实现！

四.欧拉路径

1.欧拉路的判断（一笔画问题）：

若图G中存在这样一条路径，使得它恰通过G中每条边一次，则称该路径为欧拉路径。若该路径是一个圈，则称为欧拉回路。

欧拉路径的判定：

(1)一个无向图存在欧拉路径，当且仅当该图仅存在两个或零个奇数度数的顶点，且该图是连通图。

(2)一个有向图存在欧拉路径，当且仅当该图的所有顶点度数为0或仅存在一个度为1和一个度为-1的点，其余点度数均为0，且该图是连通图。

欧拉回路的判定：

一个无向图存在欧拉回路，当且仅当该图不存在奇数度数的顶点,且该图是连通图。

一个有向图存在欧拉回路，当且仅当所有顶点的入度等于出度且该图是连通图。

理解的核心：每一个顶点，进来一条边就必须出去一条边（起点和终点除外，欧拉回路中起点终点也必须满足这个条件）

2.欧拉路径的求解方法：

通过DFS实现,伪代码如下： 算法效率：O(n + m)

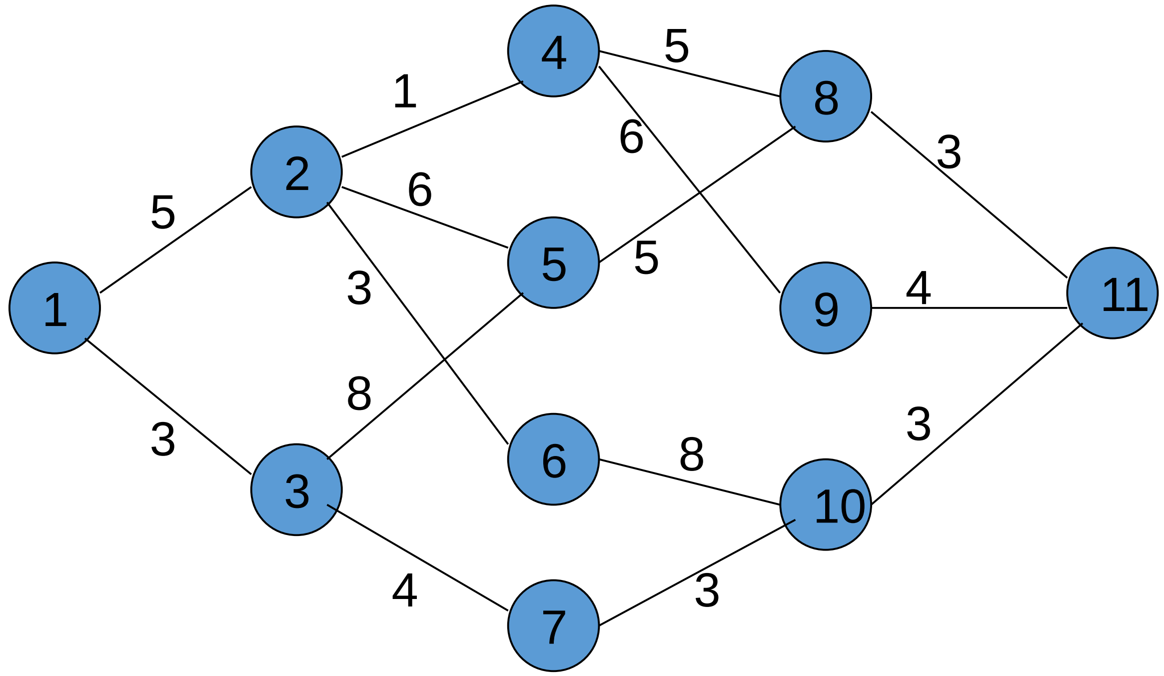
```
DFS(u):
    While (u存在未被删除的边e(u,v))
        删除边e(u,v)   //如果是无向图，反向边也要删除
        DFS(v)
    End
    PathSize ← PathSize + 1
    Path[ PathSize ] ← u   //把u加到欧拉路径中
End DFS;
```

可能会爆栈，此时需要采用非递归形式

五.最短路问题

1. 最短路

已知各个城市之间的道路情况如下：



现在，我们想从城市A到达城市E。

怎样走才能使得路径最短，最短路径的长度是多少？

1. 最短路

两类问题：

1、图中每对顶点（任意两点）之间的最短路径

（弗洛伊德算法：f loyed）。

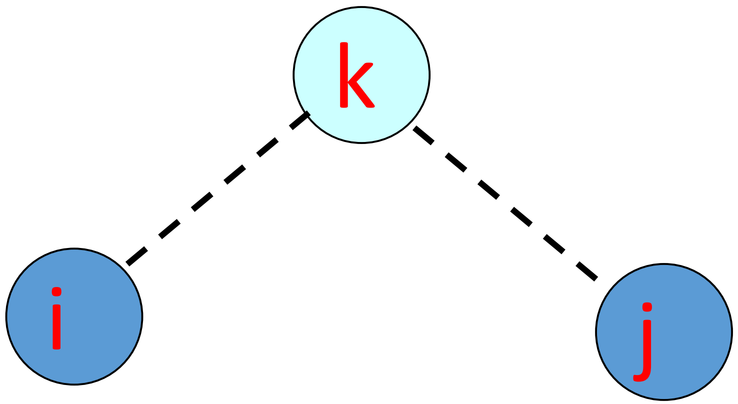
2、图中一个顶点到其他顶点的最短路径

（迪杰斯特拉算法：Dijkstra、Bellman-ford、Spfa）。

Floyd算法

目标：把图中**任意两点**i与j之间的最短距离都求出来 d[i,j]。

原理：根据图的传递闭包思想：



if d[i,k]+d[k,j]<d[i,j] then d[i,j]=d[i,k]+d[k,j]

Floyd算法

时间复杂度:O(N*N*N)

关键源码：

```
for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (f[i][j] > f[i][k] + f[k][j])
                f[i][j] = f[i][k] + f[k][j];
```

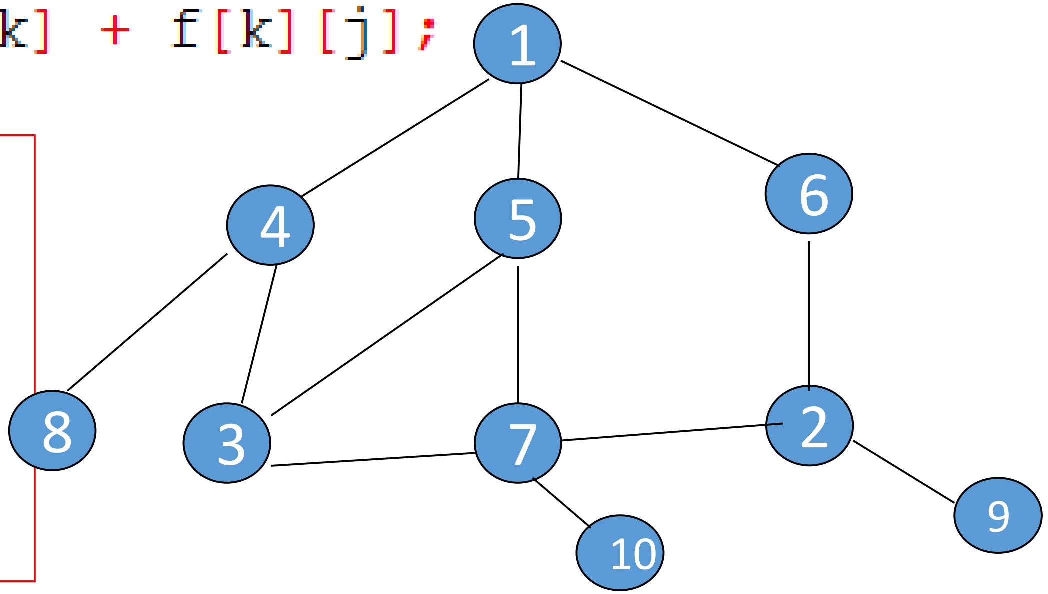
K层循环必须最外层！！

初始化条件：

d[i, i]=0 //自己到自己为0；对角线为0；

d[i,j]=边权，i与j有直接相连的边

d[i,j]= +∞，i与j无直接相连的边。；



Floyd输出最短路径：

再用一个数组存起来即可，输出时从终点倒推即可：

```
if (d[i,k] + d[k,j] < d[i,j]){
    d[i,j] = d[i,k] + d[k,j];
    path[i,j] =path[k,j];
}
```

Dijkstra 算法：

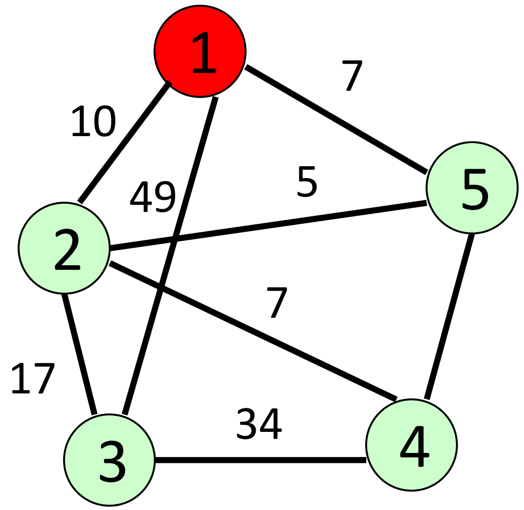
开始点（源点）： start

D[i]:顶点i到start的最短距离。

初始：

D[start]=0；

D[i]=a[start,i] （无边设为maxint）



Dijkstra 算法：

时间复杂度:O(N*N)

集合1：已求点　　集合2：未求点

1、在集合2中找一个到start距离最近的顶点k，距离=d[k]

2、把顶点k加到集合1中，同时修改集合2 中的剩余顶点j的d[j]是否经过k后变短。如果变短修改d[j]

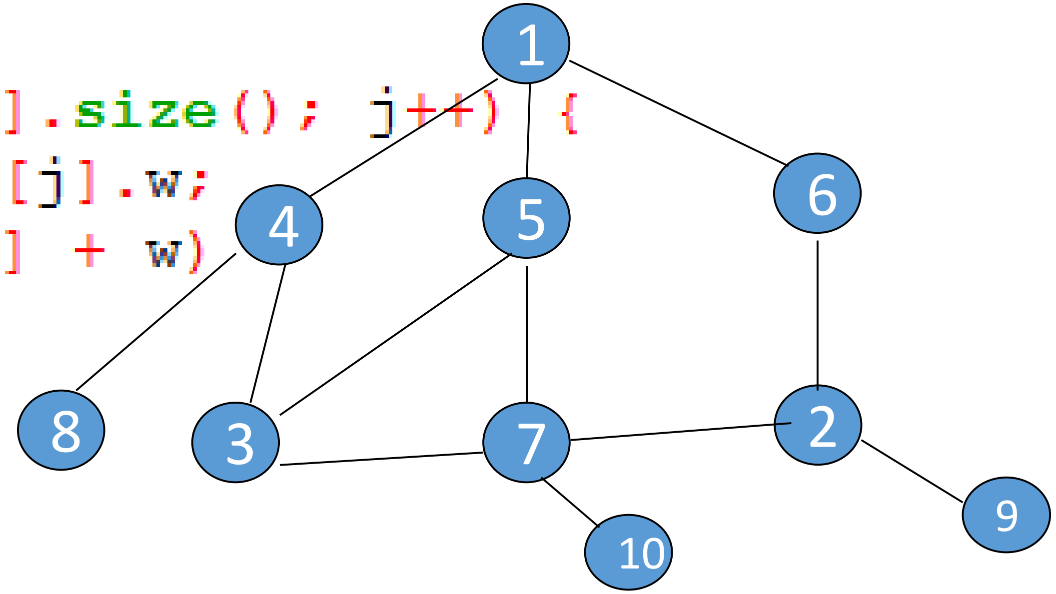
If d[k]+a[k,j]<d[j] then d[j]=d[k]+a[k,j]

3、重复1，直至集合2空为止。

Dijkstra 算法：

时间复杂度:O(N*N)

```
for (int i = 0; i < n; i++) {
    int k = 0, mint = inf;
    for (int j = 1; j <= n; j++)    //找一个到起点距离最短的点
        if (!vis[j] && d[j] < mint)
            mint = d[j], k = j;
    if (k == 0) break; //没找到，说明已经更新完了
    vis[k] = true; //找到的点为k
    for (int j = 0, ed, w; j < st[k].size(); j++) {
        ed = st[k][j].ed, w = st[k][j].w;
        if (!vis[ed] && d[ed] > d[k] + w)
            d[ed] = d[k] + w;
    }
}
```



Dijkstra +堆优化：

大概时间复杂度:O((N+M)*logN)

在第二个集合中找距离源点最近的点时，使用堆优化

（1）将起点放入优先队列中

（2）如果优先队列非空，则取出优先队列首节点，将其加入第一个集合；再用此节点去更新第二个集合中的其他节点，如果有节点的最短路被更新，则将此节点加入到优先队列中。

（3）直到集合为空。

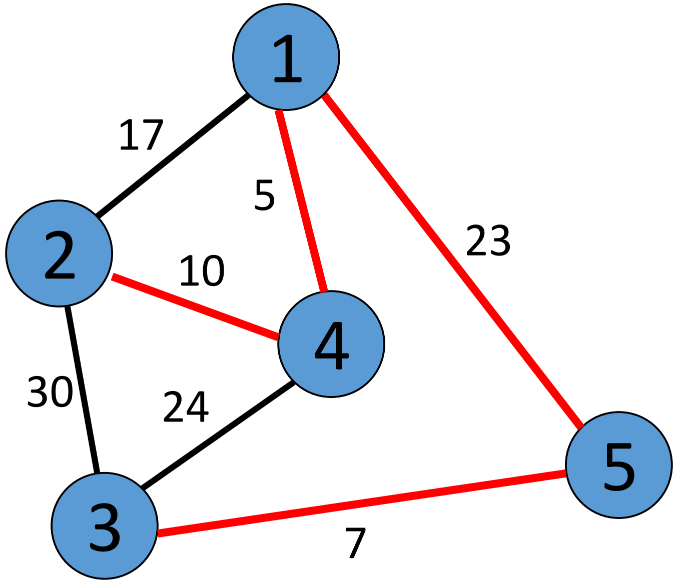
缺点：不能处理有负权的图

六.最小生成树问题

基本概念

最小生成树：

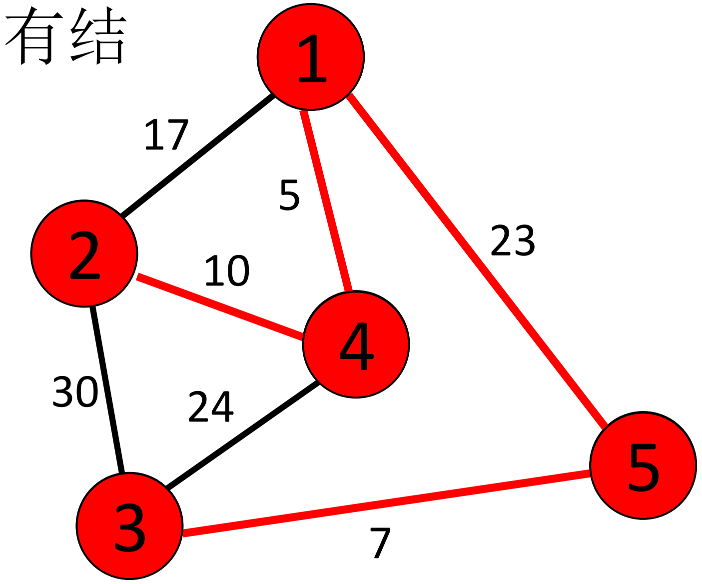
含有n个结点的图，从中选n-1条边，保持n-1个点中任意两点是连通的，并且n-1条边的和最小。这n个点和这n-1条边就成为原图的最小生成树。



Prim算法：

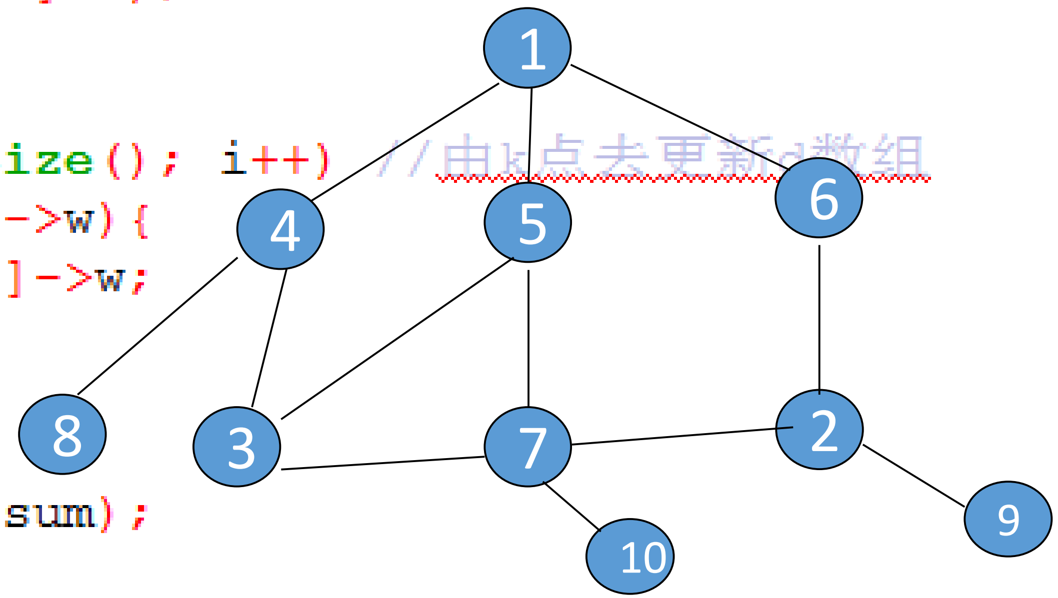
任意结点开始（不妨设为v1）构造最小生成树：

首先把这个结点包括进生成树里，然后在那些其一个端点已在生成树里、另一端点还未在生成树里的所有边中找出权最小的一条边，并把这条边、包括不在生成树的另一端点包括进生成树，…。依次类推，直至将所有结点都包括进生成树为止。



Prim算法：

```
while(tot--){//首先找d数组中未被染色的点到已染色点的最小距离点
    int k, mint = oo;
    for(int i = 1; i <= n; i++)
        if(!flag[i] && d[i].w < mint)
            k = i, mint = d[i].w;
    printf("%d %d %d\n", d[k].st, k, d[k].w);
    sum += d[k].w;
    flag[k] = true;//对k点染色
    for(unsigned int i = 0; i < st[k].size(); i++)  //由k点去更新d数组
        if(d[st[k][i]->ed].w > st[k][i]->w){
            d[st[k][i]->ed].w = st[k][i]->w;
            d[st[k][i]->ed].st = k;
        }
}
printf("最小生成树的权值大小为：%d\n", sum);
```



作业

重点掌握

0. 图论基础知识
1. 图的存储结构 - 至少掌握邻接矩阵
2. 图的遍历 - DFS & BFS
3. 图的一边画问题 - 基于图的遍历
4. 最短路算法 - Floyd算法 & Dijkstra算法
5. 最小生成树算法 - Prim算法

共同进步

- 一分耕耘，一分收获！
- 希望在今后的学习生活中共同进步！

谢谢！