# 图

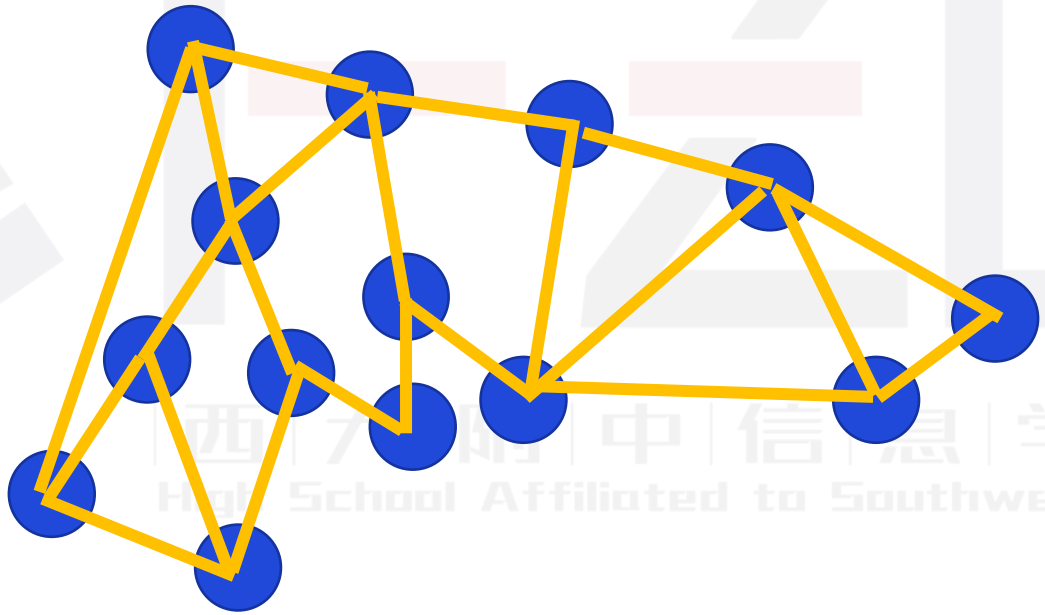
## 中国重庆·西南大学附属中学校园平面图

# 什么是图



图几乎可以用来表现所有类型的结构或系统，从交通网络到通信网络
从下棋游戏到最优流程，从任务分配到人际交互网络，图都有广阔的用武之地。

图 = 顶点+边 组成的网络




信息学暑期

# 简单图论入门

# 相关概念



**无向图**

**有向图**

- 度
- 入度
- 出度

**加权图**

**见讲义**

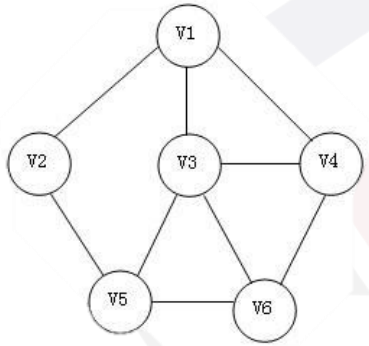
# 图的存储 邻接矩阵



如何让计算机存储图的关系？

存什么？

- 顶点
- 顶点关系




| | v1 | v2 | v3 | v4 | v5 | v6 |
|---|---|---|---|---|---|---|
| v1 | 0 | 1 | 1 | 1 | 0 | 0 |
| v2 | 1 | 0 | 0 | 0 | 1 | 0 |
| v3 | 1 | 0 | 0 | 1 | 1 | 1 |
| v4 | 1 | 0 | 1 | 0 | 0 | 1 |
| v5 | 0 | 1 | 1 | 0 | 0 | 1 |
| v6 | 0 | 0 | 1 | 1 | 1 | 0 |

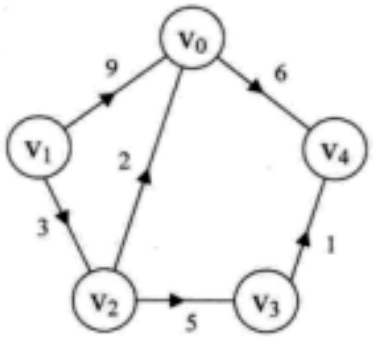
二维数组(mp[maxn][maxn])存储

对于无向图：

mp[i][j]=0，表示i和j之间没有联通

mp[i][j]=1，联通

并且二维数组是**对称**的




$$\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & \infty & 6 \\ 9 & 0 & 3 & \infty & \infty \\ 2 & \infty & 0 & 5 & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

对于有向图：

mp[i][j]=t，表示i到j的权值为t

mp[i][j]=∞，表示i到j未直接联通

二维数组是**不对称**的

# 邻接矩阵代码实现



**无向图：**

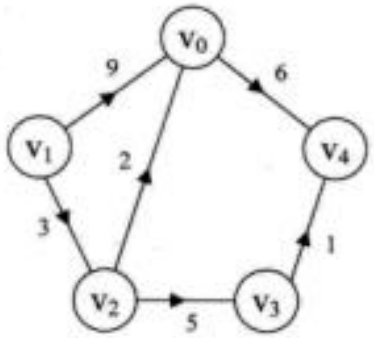
```
int g[123][123];
int main(){
    memset(mp,0,sizeof(mp));
    int e,x,y,w;
    cin>>e;
    for(int i=1; i<=e; i++){
        cin>>x>>y>>w;
        mp[x][y]=w;
        mp[y][x]=w;
    }
    //... do other things
    return 0;
}
```

**有向图：**

```
int g[123][123];
int main(){
    memset(mp,127,sizeof(mp));
    int e,x,y,w;
    cin>>e;
    for(int i=1; i<=e; i++){
        cin>>x>>y>>w;
        mp[x][y]=w;
    }
    //... do other things
    return 0;
}
```

# 邻接矩阵






顶点数组：

| $v_0$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|

边数组：

| | $v_0$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ |
|---|---|---|---|---|---|
| $v_0$ | 0 | ∞ | ∞ | ∞ | 6 |
| $v_1$ | 9 | 0 | 3 | ∞ | ∞ |
| $v_2$ | 2 | ∞ | 0 | 5 | ∞ |
| $v_3$ | ∞ | ∞ | ∞ | 0 | 1 |
| $v_4$ | ∞ | ∞ | ∞ | ∞ | 0 |

# 它有什么缺陷？

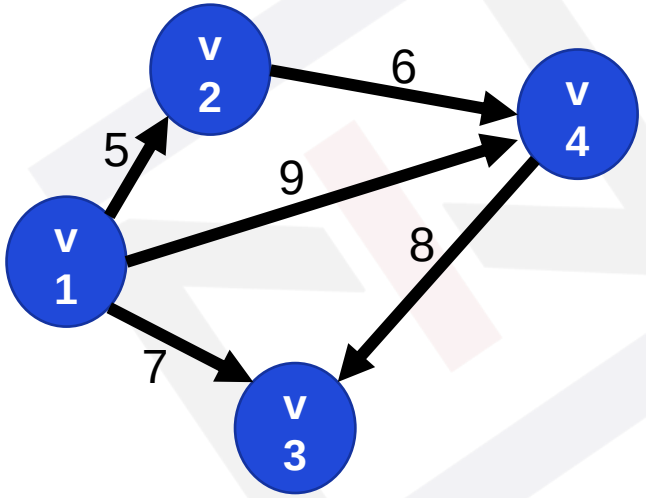
不难发现有很多没有连接的顶点

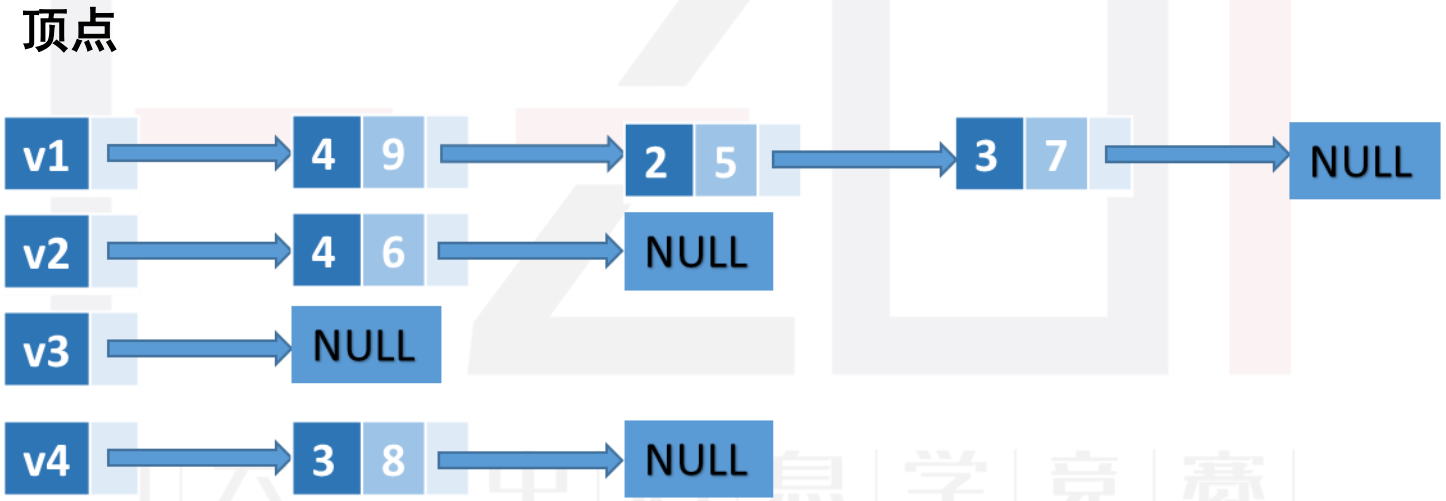
是非常浪费空间

# 图的存储



于是诞生了另外一种存储结构——邻接表



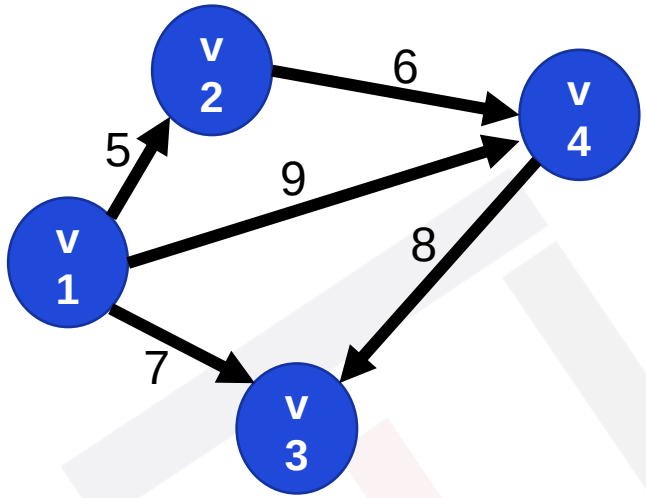



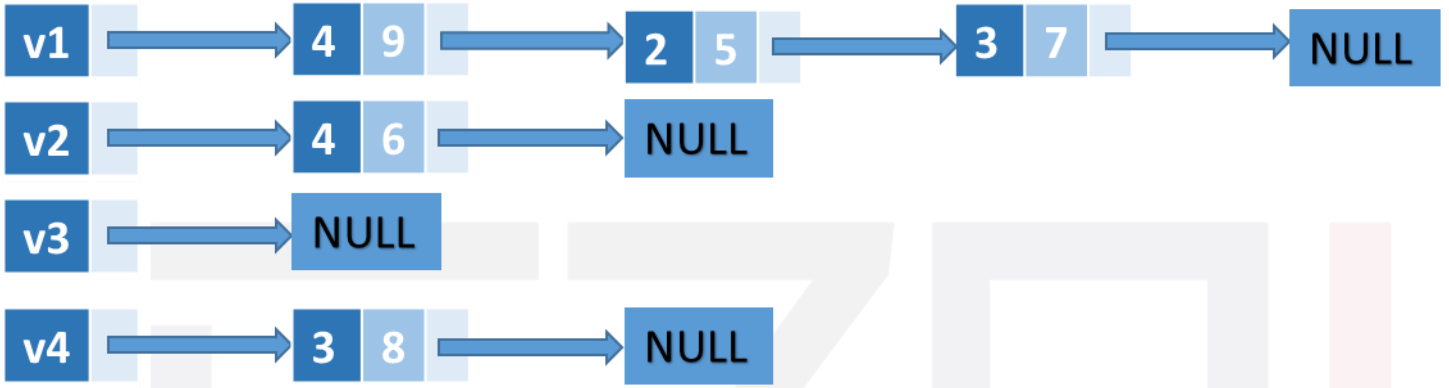

本质：多条**链表**

用**数组**来实现链条（静态链表）

# 图的存储






要用到哪些数组？

对于**顶点**：

1、与它相连的第一条边的id：first[]

对于**边**：

1、边权值:w[]

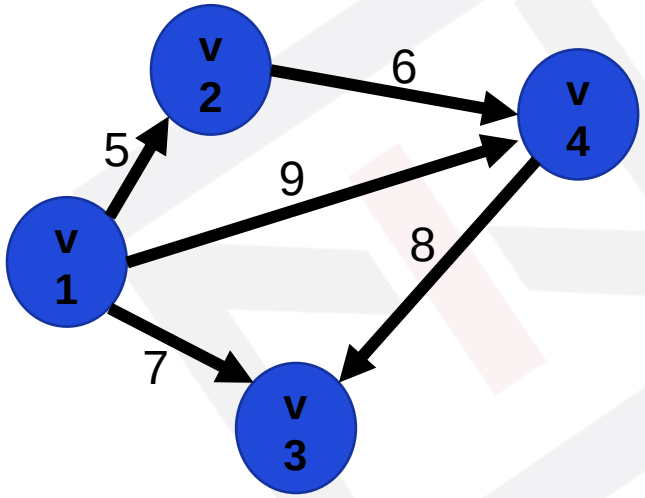
2、边的出发顶点：u[]

3、边到达的顶点：v[]

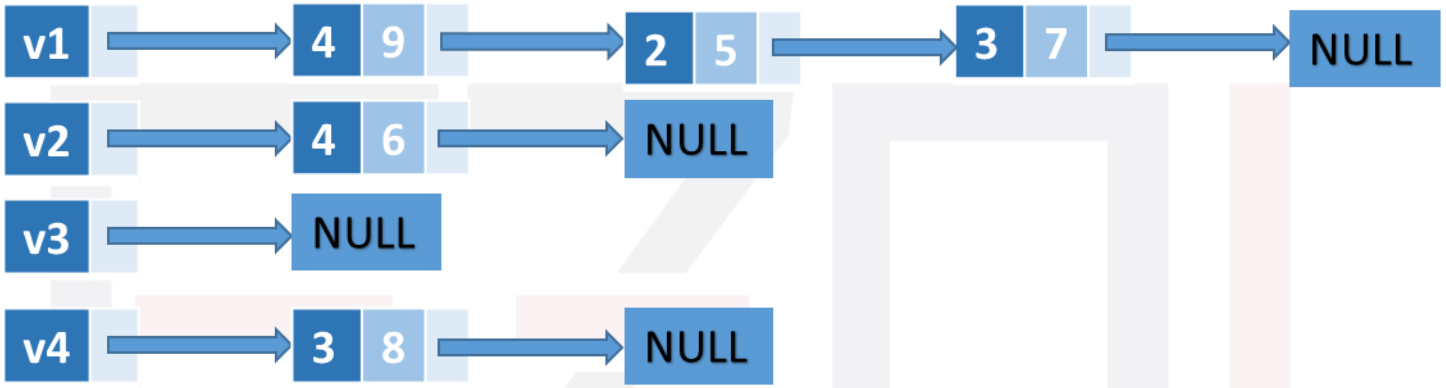
4、下一条与当前处理节点相连边的编号：next[]

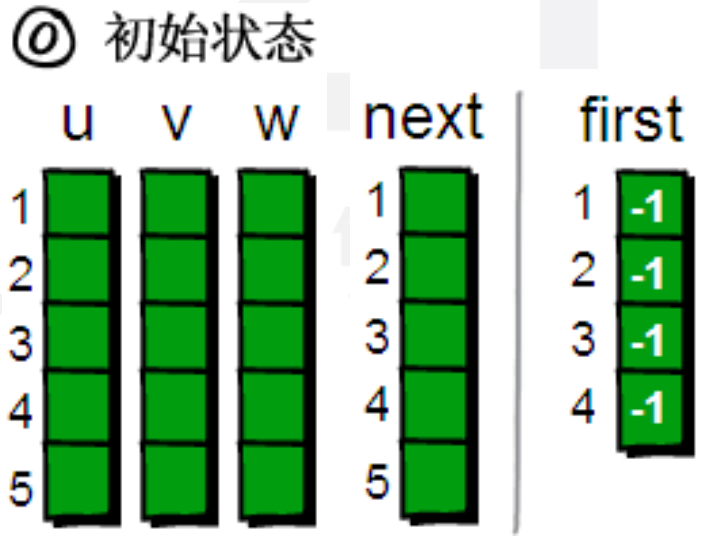
# 领接表储存




v 2
6
v 4
5
9
v 1
8
7
v 3

顶点


v1
4
9
2
5
3
7
NULL
v2
4
6
NULL
v3
NULL
v4
3
8
NULL


初始状态
u
v
w
next
first
1
2
3
4
5
-1
-1
-1
-1

# 邻接表储存(前向星建图)

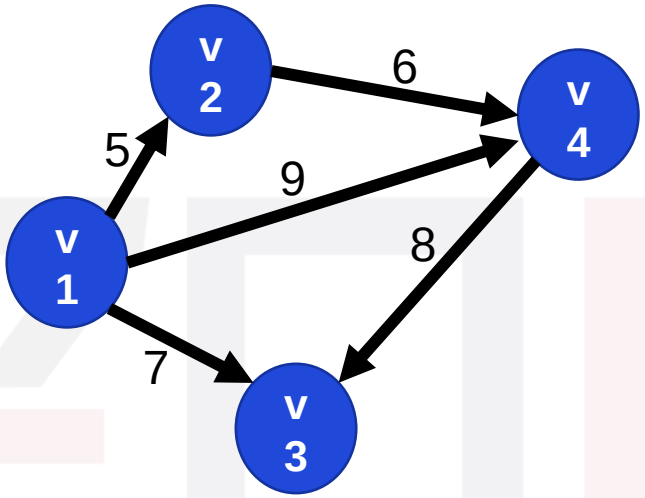


以v1顶点为例：

从v1出去的边有

1 4 9
1 2 5
1 3 7

边的编号分别为1, 3, 5




| | u | v | w |
|---|---|---|---|
| 1 | 1 | 4 | 9 |
| 2 | 4 | 3 | 8 |
| 3 | 1 | 2 | 5 |
| 4 | 2 | 4 | 6 |
| 5 | 1 | 3 | 7 |

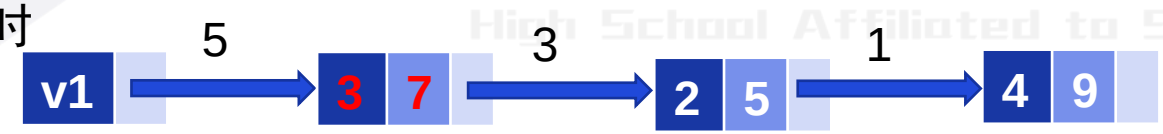
当读入1号边时


next[1]=-1, first[1]=1

当读入3号边时

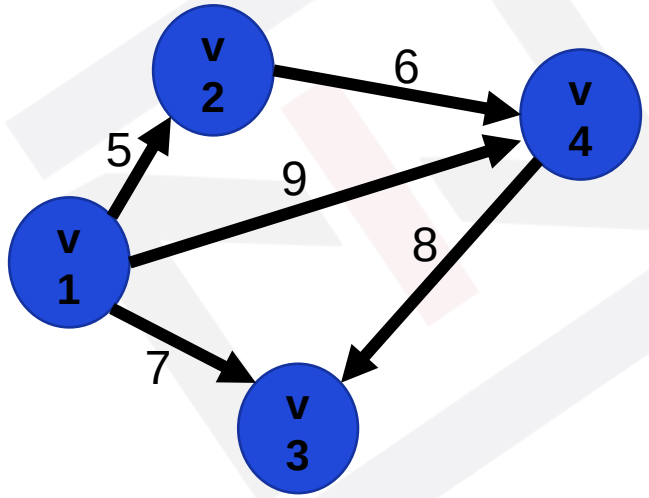

next[3]=1, first[1]=3

当读入5号边时

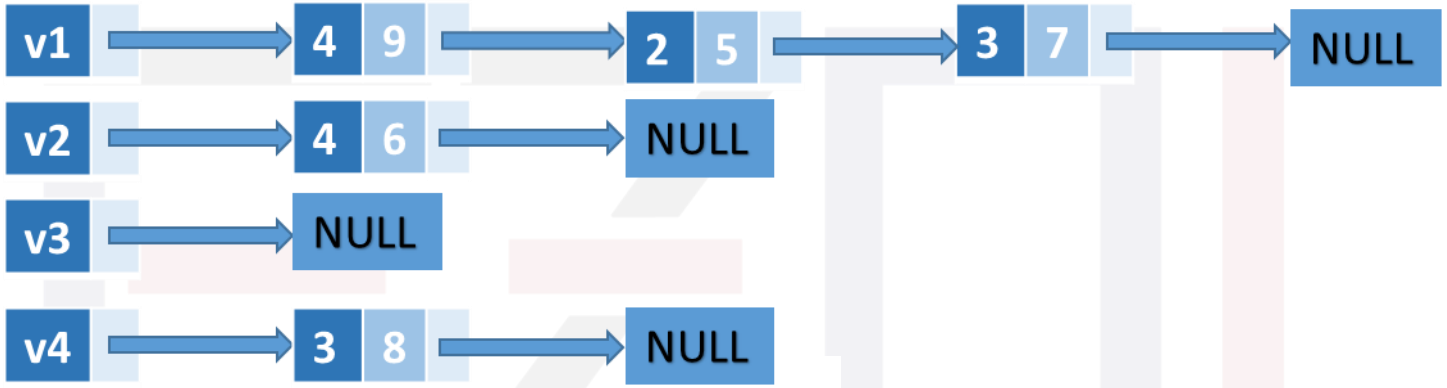



next[5]=3, first[1]=5

# 进一步简化




顶点




| | v | w | next |
|---|---|---|---|
| 1 | 4 | 9 | -1 |
| 2 | 3 | 8 | -1 |
| 3 | 2 | 5 | 1 |
| 4 | 4 | 6 | -1 |
| 5 | 3 | 7 | 3 |

| | first |
|---|---|
| 1 | 5 |
| 2 | 4 |
| 3 | -1 |
| 4 | 2 |

# 邻接表储存-代码实现



```
struct node
{
        int v,w,next;
}e[maxn];
int first[maxm];
int cnt=0;
void add(int u1,int v1,int w1){
        e[++cnt].v=v1; //cnt表示读入的边的编号
        e[cnt].w=w1;
        e[cnt].next=first[u1];
        first[u1]=cnt;
}
```

遍历一号顶点相连的边：

```
k=first[1];// 1号顶点其中的一条边的编号（其实也是最后读入的边）
for(int i=k;i;i=next[i]) {//其余的边都可以在next数组中依次找到
        printf("%d %d\n",e[i].v,e[i].w);
}
```

# vector：变长数组

```
#include<vector>
```

**定义：**

```
vector<typename> name;
1.vector<int> name;//int整数类型
2.vector<node> name;//node结构体类型
```

**访问：**

```
vector<int> a;
```

a[0],a[1];

a.size() 数组当前大小

**添加：**

**push_back()**

push_back(x)就是在vector后面添加一个元素x，时间复杂度为O(1)。

# vector储存

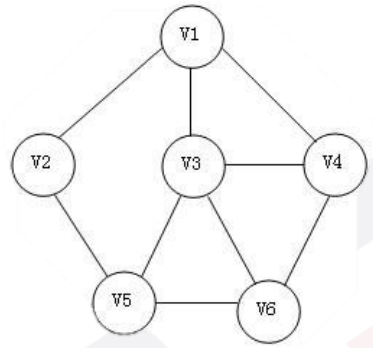


```
struct edge{
        int to,w;
};
vector<edge> v[maxn];

void addedge(int x,int y,int z){
        edge e;
        e.to=y;
        e.w=z;
        v[x].push_back(e);
}
//遍历与x相连的边
for(int i=0;i<v[x].size();i++){
        cout<<v[x][i].to<<" "<<v[x][i].w;
}
```

# 无向图的储存






邻接矩阵

```
mp[x][y]=w;
mp[y][x]=w; //无向图
```

邻接表(前向星建图)

```
add(x,y);
add(y,x) //无向图
```

vector存储

```
v[x].push_back(y);
v[y].push_back(x);
```

# 练习:20min



321. 「模板」图的存储

590. 图的存储练习

# 图的两种遍历方式

西南大学附属中学 High School Affiliated to Southwest University

## 深度优先遍历

```
bool vis[maxn];
void dfs(int x){
        cout<<x<<endl;
        vis[x]=1;
        for(int i=0;i<v[x].size();i++)
        {
                if(!vis[v[x][i].to])
                        dfs(v[x][i].to);
        }
}
for(int i=1;i<=n;i++)
        if(!vis[i])//防止图不连通
          dfs(i);
```

## 广度优先遍历

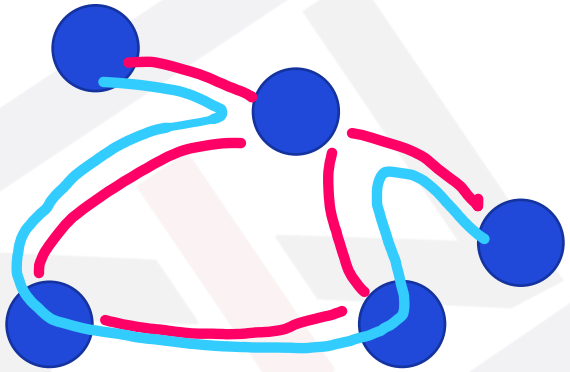
```
queue<int> q;
void bfs(int start){
        q.push(start);
        vis[start]=1;
        while(!q.empty()){
                int x=q.front();
                q.pop();
                cout<<x<<endl;
                for(int i=0;i<v[x].size();i++){
                  if(!vis[v[x][i].to]){
                    vis[start]=1;
                    q.push(v[x][i].to);
                }
        }
      }
}
```

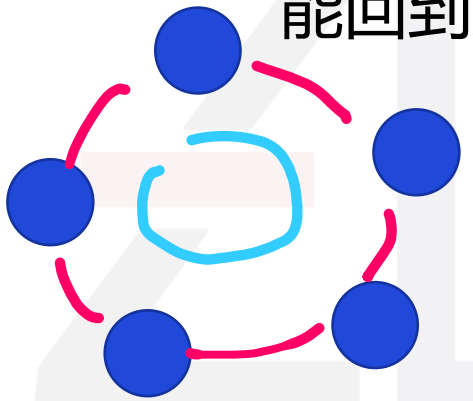
# 欧拉路问题



对于一个图，如果用一条线经过每条边各一次（点可以重复经过）
那么其实就是用一笔可以画出这个图（简称一笔画），也称为**欧拉路**，
如果起点和终点相同，称为**欧拉回路**

## 无向图





能回到起点

欧拉路存在：
图联通，且**奇点**有且仅有2个
与这个点相联的边为奇数

欧拉回路：
图联通，无奇点

## 无向图代码实现

1 使用邻接矩阵存储。
2 单独开数组du[i] 记录结点i连边数
3 单独开数组circuitpos 记录欧拉路上的点

```
int main(){
        建立无向图g[x][y]，注意更新du[i]
        for循环寻找奇点，如果有奇点就从它开始，否则从1开始（随机）
        dfs(start) //递归进行遍历，并记忆。（每次遍历完成后点值=0，删边）
}
```

## 无向图代码实现

```
void find_circuit(int i)              //这个点深度优先遍历过程寻找欧拉路
{
  int j;
  for (j=1; j <=n; j++)
    if (g[i][j]==1)                   //从任意一个与它相连的点出发
    {
        g[i][j]=g[j][i]=0;            //删去这条边,避免下一次重复走过
        find_circuit(j);
    }
  circuit[++circuitpos]=i;            //记录下路径
}
```

```
int main()
{
    memset(g,0,sizeof(g));
    cin >> n >> e;
    for (i=1; i <=e; i++)
    {
      cin >> x >> y;
      g[x][y]=g[y][x]=1;              //说明 x 和 y 间有连边
      du[x]++;                        //统计每个点的度
      du[y]++;
    }
    start=1;                          //如果有奇点,就从奇点开始寻找,这样找到的就是
    for (i=1; i <=n; i++)             //欧拉路。没有奇点就从任意点开始,
      if (du[i]%2==1)                 //这样找到的就是欧拉回路。(因为每一个点都是偶点)
          start=i;
    circuitpos=0;
    find_circuit(start);
    for (i=1; i <=circuitpos; i++)
      cout << circuit[i] << " ";
    cout << endl;
    return 0;
}
```

# 欧拉路问题



对于一个图，如果用一条线经过每条边各一次（点可以重复经过）
那么其实就是用一笔可以画出这个图（简称一笔画），也称为**欧拉路**，
如果起点和终点相同，称为**欧拉回路**

## 有向图

把一个点的出度**m**记为1，入度**n**记为-1，这个点的度数k=m+n

欧拉路存在：
只有一个点k=-1，一个点k=1
其中k=1是起点，k=-1的是终点

欧拉回路：
所有点k=0

# Thanks

## For Your Watching