图的基础

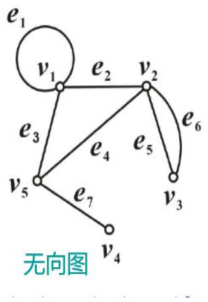
1.图的**基本概念**

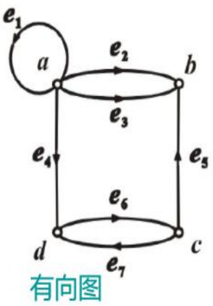
图：一个二元组 $G=<V(G),E(G)>$ 称为图(Graph)，其中 $V(G)$ 是非空集，称为点集(Vertex set)，对于 V 中的每一个元素，称其为顶点(Vertex)或结点(Node)，简称点；$E(G)$ 为 $V(G)$ 各结点之间边的集合，称为边集(Edge set).

简单图：没有环和重边的图.

图的分类：无向图，有向图

- 如果边都是双向的，则这个图为无向图；
- 如果边都是单向的，则这个图为有向图.





顶点的度：

- 在**无向图**中，一个顶点相连的边数称为该顶点的**度**。例：顶点 v_5 的度为 3(e_3,e_4,e_7)；
- 在**有向图**中，从一个顶点**出发**的边数称为该顶点的**出度**；**到达**该顶点的边数称为该顶点的**入度**。例：点 d 的入度为 2(e_4,e_7)，出度为 1(e_6)；
- 顶点的最大度数称为图的度数.

完全图、稠密图和稀疏图：

- ✓ 任何两个顶点之间都有边（弧）相连称为完全图；
- ✓ 边（弧）很少的图称为稀疏图，反之为稠密图.

连通图、强连通图、连通网：——方便引入最小生成树

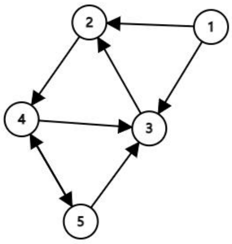
①连通图：在无向图中，若任意两个顶点 v_i 和 v_j 都有路径相通，则称该无向图为连通图；

②强连通图：在有向图中，若任意两个顶点 v_i 和 v_j 都有路径相通，则称该有向图为强连通图；

③连通网：在连通图中，若图的边具有一定的意义，每一条边都对应着一个数，称为**权**；权代表着连接两个顶点的代价，称这种连通图叫做连通网.

2. 图的表示方式

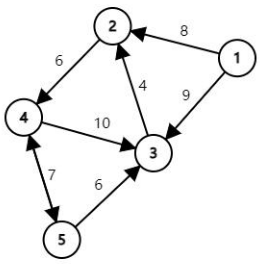
✧ 邻接矩阵（用于稠密图，常用二维数组存储） $A[i,j]=\begin{cases}1(\text{或权}),(v_i,v_j)\in E;\\0(\text{或特殊值}),(v_i,v_j)\notin E\end{cases}$

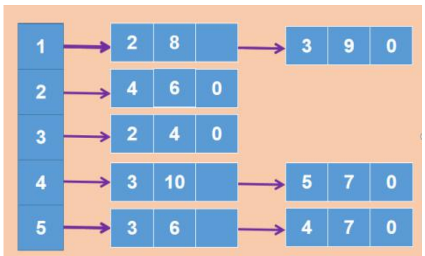


起点 \ 终点	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	0	1	1	0

$$\begin{bmatrix}0&1&1&0&0\\0&0&0&1&0\\0&1&0&0&0\\0&0&1&0&1\\0&0&1&1&0\end{bmatrix}$$

✧ 邻接表（用于稀疏图）实现代码链接

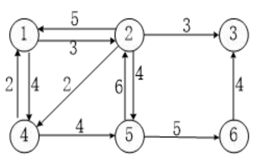




✧ 链式前向星（一般采用这种方法存图） **代码链接**

按照边来进行存储图，而邻接矩阵是以点出发进行存储图（如果点的数量庞大，会有爆内存的可能），链式前向星又叫边集数组，与邻接表的差异在于未使用链表，而是借助索引进行实现.

以同起点为一条链，数组$head[i]$存储起点i的最新录入的一条边的索引，每条边a以结构体的形式（也可以不用结构体）存储该边的信息（该边的终边$ed[a]$,该边的权值$w[a]$和同找出一条起点的边中的上一条边即上一次录入的边的位置$nex[a]$);
例如，找出一条$a\to b$的边，先根据$head[a]$找出从起点a出发的所有边里最新录入的那条边，然后根据该边找出它的上一条边，再根据上一条边找出它的上上一条边$\cdots$直到找到一条终边为b的边为止。整个过程是链式回溯，所以是链式前向星。



u	0	1	2	3	4	5	6
head[u]	-1	5	8	-1	9	10	3

i	0	1	2	3	4	5	6	7	8	9	10
ed[i]	2	1	2	3	3	4	4	1	5	5	6
nex[i]	-1	-1	-1	-1	1	0	4	-1	6	7	2
val[i]	3	5	6	4	3	4	2	2	4	4	5
	1->2	2->1	5->2	6->3	2->3	1->4	2->4	4->1	2->5	4->5	5->6

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<iostream>

#define MAXN 10000
#define MAXM 1000010
using namespace std;
int n,m;
/******************核心部分**************************/
int idx;
int head[MAXN],ed[MAXM],nex[MAXM],w[MAXM];
//head 存储序号，ed 表示这条边的终点，nex 表示上一个与起点相连的边，w 表示边权值（距离），
idx 计数
void add(int a,int b,int c){//起点,终点,边权值
    ed[idx]=b;//记录终边
    w[idx]=c;//记录边权值
    nex[idx]=head[a];//记录与上一个起点相连的序号,作为后序操作索引
    head[a]=idx++;
}
/******************核心部分**************************/
int main(){
    /******************核心部分**************************/
    idx=0;
    fill(head,head+MAXN,-1);//head 初始化为-1 且 idx=0
    //memset(head,-1,sizeof(head));或者用 menset 初始化为-1 也可以
    /******************核心部分**************************/
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int x,y,z;
        scanf("%d %d %d",&x,&y,&z);
        add(x,y,z);
        //无向图需反向增加一条边
        /*add(y,x,z);*/}
    return 0;
}
/******************核心部分**************************/
//访问从 x 出发的所有边
for(int i=head[x];i!=-1;i=nex[i]){
    int y=ed[i],z=w[i];
    ...你想进行的操作//找到一条有向边(x,y),且边权值为 z
}
/******************核心部分**************************/
```

优先队列

优先队列(priority_queue)一般用于解决贪心问题,其底层是用**堆**来实现,在优先队列中,任何时刻,队首元素一定是当前队列中优先级最高(优先值最大)的那一个(大根堆),也可以是最小的那一个(小根堆)。可以不断往优先队列中添加某个优先级的元素,也可以不断弹出优先级最高的那个元素,每次操作其会自动调整结构,始终保证队首元素的优先级最高。

1. 优先队列的头文件

```
#include<queue>//包括队列queue 和优先队列priority_queue 两个容器
using namespace std;
```

2. 优先队列的定义

```
    priority_queue <typename> name;//默认是大根堆
    //typename 可以是任何基本类型或者容器，name 为优先队列的名字
    priority_queue <int> q1;//生成一个int 类型的大根堆
    //priority_queue < int , vector <int> ,less <int> > q1;//与上方优先队列等价
    priority_queue < double, vector <double> ,greater <double> > q2;
    //生成一个double 类型的小根堆
```

3. 优先队列的使用

和queue不一样的是,priority_queue没有front()和back(),而只能通过top()或pop()访问队首元素(也称堆顶元素),也就是优先级最高的元素.

```
priority_queue <int> q;//优先队列的定义
```

①q.push(x);

//将x加入优先队列q中,时间复杂度为$O(\log_2 n)$,n为当前优先队列中的元素个数.加入后会自动调整priority_queue的内部结构,以保证堆顶元素的优先级最高.

②q.top();

//获得堆顶元素,时间复杂度为$O(1)$.

③q.pop();

//让堆顶元素出队,时间复杂度为$O(\log_2 n)$,n为当前优先队列中的元素个数.出队后会自动调整priority_queue的内部结构,以保证队首元素(堆顶元素)的优先级最高.

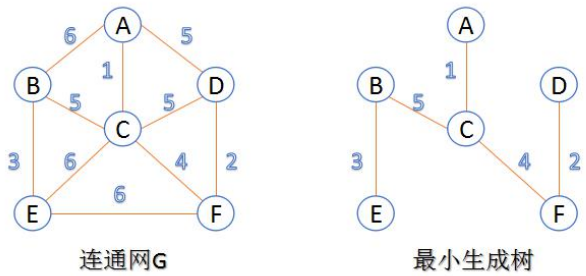
最小生成树

1．最小生成树的**概念**

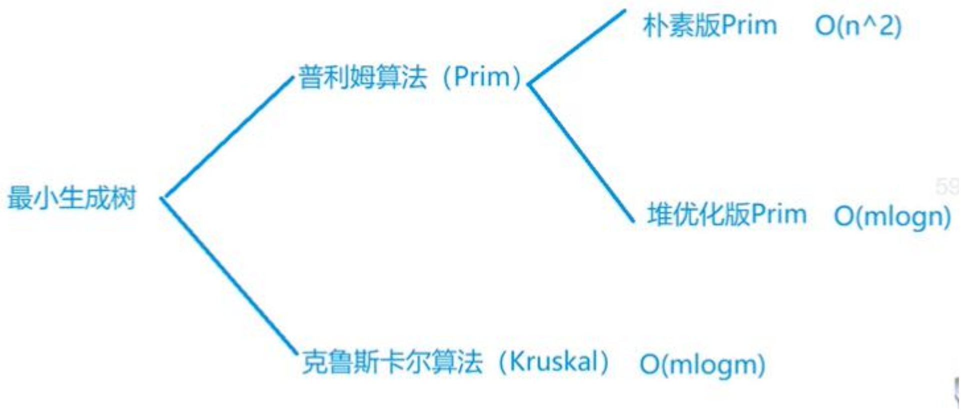
生成树：一个连通图的生成树是指一个连通子图，它含有**图中全部 n 个顶点**，但只有足以构成**一棵树的 n-1 条边**，一棵有 n 个顶点的生成树有且仅有 n-1 条边，如果生成树中再添加一条边，则必定成环.
最小生成树：在连通网的所有生成树中，所有边的代价和最小的生成树，称为最小生成树.（最小生成树可能不唯一）.

构成**最小生成树**的准则：

- ✓ 必须只使用该网络中的边来构造最小生成树；
- ✓ 必须使用有且仅使用 n-1 条边来连接网络中的 n 个顶点；
- ✓ 不能使用产生回路的边.



2. 最小生成树的**求法**

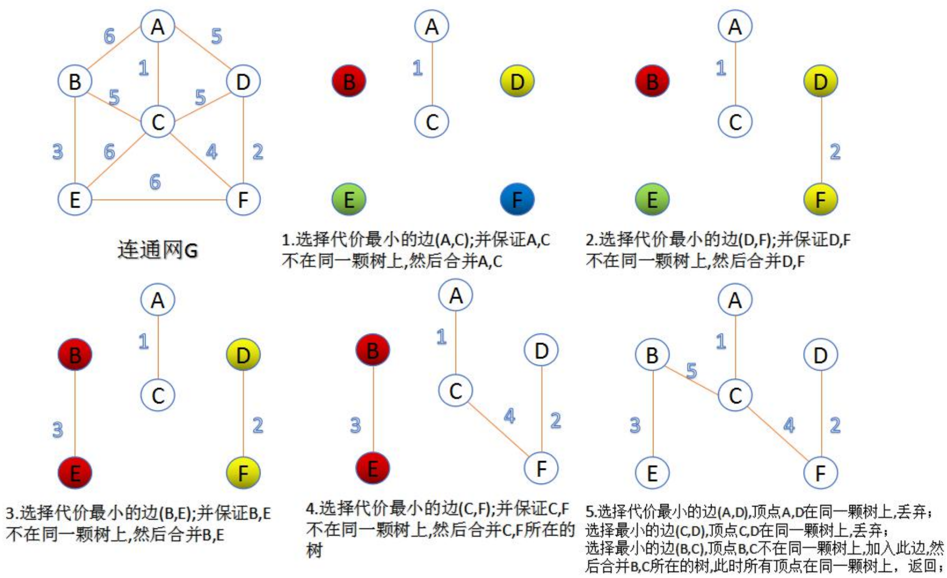


最小生成树题目链接

❖ 克鲁斯卡尔(Kruskal)算法(时间复杂度为 $O(m\log m)$)

该算法可以称为**加边法**,初始化最小生成树边数为 0,每迭代一次选择一条满足条件的最小代价边,加入到最小生成树的边集合中.

1. 把图中的所有边按代价从小到大排序;
2. 把图中的 n 个顶点看成独立的 n 棵树组成的森林;
3. 按权值从小到大选择边,所选的边连接的两个顶点 u_i, v_i 属于两棵不同的树,使之成为最小生成树的一条边,并将这两棵树合并作为一棵树(**此处用到上一讲的并查集**);
4. 重复(3),直到所有顶点都在同一棵树上或者有 $n-1$ 条边为止.



Kruskal 代码链接

代码如下：

```
#include <iostream>
#include <cstring>
#include <algorithm>

#define MAXN 100010

using namespace std;

int fa[MAXN];
struct node{
    int start_;
    int end_;
    int val;
}s[2*MAXN];

int cmp(struct node a,struct node b){
    return a.val<b.val;
}

void init(int n){
    for(int i=1;i<=n;i++)
        fa[i]=i;
}

int find(int x){
    if(fa[x]!=x)
        fa[x]=find(fa[x]);
    return fa[x];
}

int main(){
    int n,m,res=0,cnt=0;
    scanf("%d %d",&n,&m);
    init(n);
    for(int i=1;i<=m;i++)
        scanf("%d %d %d",&s[i].start_,&s[i].end_,&s[i].val);
    sort(s+1,s+m+1,cmp);
    for(int i=1;i<=m;i++){
        int f_start=find(s[i].start_),f_end=find(s[i].end_);
        if(f_start!=f_end){
            fa[f_end]=f_start;
            res+=s[i].val;
```

```
            cnt++;
        }
    }
    if(cnt<n-1)
        printf("orz\n");
    else printf("%d\n",res);
    return 0;
}
```

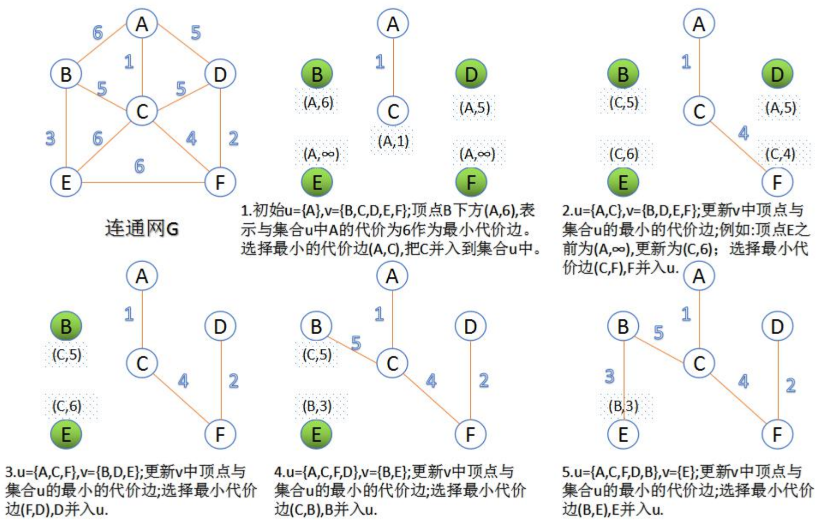
❖ 普利姆(prim)算法

该算法可以称为**加点法**,每次迭代选择代价最小的边对应的点,将满足条件的加入到最小生成树中.算法从某个顶点 s 出发,逐渐覆盖整个连通网的所有顶点.

1. 图的所有顶点集合为 V;初始令顶点的集合 $u=\{s\}$,剩余的顶点集合:$v=V-u$.

2. 在两个集合 u,v 能够组成的边中,选择一条代价最小的边 $e=(u_0,v_0)$,并把这条边加入到最小生成树中,把 v_0 加入到集合 u 中;

3. 重复上述步骤,直到最小生成树有 $n-1$ 条边或者有 n 个顶点为止.



朴素版,时间复杂度为 $O(n^2)$　　朴素版代码链接(此处也可用二维数组建图,代码略)

```
#include <iostream>
#include <cstring>
#include <algorithm>

#define MAXN 5010
#define MAXM 400010//由于是无向图,开两倍的空间存边
```

```
using namespace std;

const int inf=0x3f3f3f3f;
int n,m;
int dis[MAXN],vis[MAXN];
int head[MAXN],ed[MAXM],nex[MAXM],val[MAXM],idx;

void add(int a,int b,int c){
    ed[idx]=b;
    val[idx]=c;
    nex[idx]=head[a];
    head[a]=idx++;
}

int prim(){
    for(int i=1;i<=n;i++)//将每个点到生成树的距离初始化为无穷大
        dis[i]=inf;
    int res=0;
    dis[1]=0;//将1(也可选择其他的点)作为生成树的起点
    for(int i=1;i<=n;i++){
        int pos=-1;
        for(int j=1;j<=n;j++)//找到未被访问并且距离最小的点
            if(vis[j]==0&&(pos==-1||dis[pos]>dis[j]))
                pos=j;
        if(dis[pos]==inf)//所有点都未连通
            return -1;
        vis[pos]=1;//标记作为它已被使用
        res+=dis[pos];
        for(int j=head[pos];j!=-1;j=nex[j])//枚举从pos点出发的边并更新dis
            if(vis[ed[j]]==0&&dis[ed[j]]>val[j])
                dis[ed[j]]=val[j];
    }
    return res;
}

int main(){
    memset(head,-1,sizeof(head));
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int start_,end_,val_;
        scanf("%d %d %d",&start_,&end_,&val_);
        add(start_,end_,val_);
        add(end_,start_,val_);
    }
```

```
    int ans=prim();
    if(ans==-1)
        printf("orz\n");
    else printf("%d\n",ans);
    return 0;
}
```

堆(优先队列)优化版，时间复杂度为 $O(m\log n)$　堆优化版代码链接

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>

#define MAXN 5010
#define MAXM 400010//由于是无向图,开两倍的空间存边

using namespace std;

typedef pair<int,int> PII;
const int inf=0x3f3f3f3f;
int n,m;
int dis[MAXN],vis[MAXN];
int head[MAXN],ed[MAXM],nex[MAXM],val[MAXM],idx;

void add(int a,int b,int c){
    ed[idx]=b;
    val[idx]=c;
    nex[idx]=head[a];
    head[a]=idx++;
}

int prim(int start){
    int res=0,cnt=0;
    priority_queue<PII,vector<PII>,greater<PII> > q;
    for(int i=1;i<=n;i++)
        dis[i]=inf;
    dis[start]=0;
    q.push({0,start});
    while(q.size()>0){
        PII temp=q.top();
        q.pop();
        if(vis[temp.second]==1)//当前点访问过
            continue;
```

```
        vis[temp.second]=1;//标记该点访问过
        res+=temp.first;//生成树加入该边
        cnt++;//记录点的个数
        for(int i=head[temp.second];i!=-1;i=nex[i])
            if(vis[ed[i]]==0)
                q.push({val[i],ed[i]});
    }
    if(cnt<n)//点的个数小于n
        return -1;
    else return res;
}

int main(){
    memset(head,-1,sizeof(head));
    scanf("%d %d",&n,&m);
    for(int i=1;i<=m;i++){
        int start_,end_,val_;
        scanf("%d %d %d",&start_,&end_,&val_);
        add(start_,end_,val_);
        add(end_,start_,val_);
    }
    int ans=prim(1);
    if(ans==-1)
        printf("orz\n");
    else printf("%d\n",ans);
    return 0;
}
```