图论总结

5.13 开始讲图论的，我咕咕咕到了 5.20。。。

图论相对于 DP 和数学期望那块，算是简单的了。感觉就是要多总结多找题感吧。

这里从基础概念开始，总结一些非常非常基础的东西。~~由于我太菜似乎也总结不出什么深奥的东西/kk。~~

概念

注：这里是总结了一部分我认为比较重要或者比较难的概念，可能并不全面，全面的图的概念见图论相关概念 - *OI Wiki*。

图

图（Gragh）是一个二元组 $G = (V(G), E(G))$。其中 $V(G)$ 是**点集**，$E(G)$ 是**边集**。

根据不同的分类标准可以把图分为不同的种类：

按照**边是否有边权**把图分为**无权图**和**有权图**；按照**图是否连通**分为**连通图**和**非连通图**；按照**边是否有向**分为**有向图**和**无向图**……

度数

与一个顶点 v 相连的边的条数称作该顶点的**度数（degree）**，记作 $d(v)$。

对于有向图，一个点的度数又可以分为**入度**和**出度**。对于一个点 v，以该点为**终点**的边的条数叫点 v 的入度，以该点为**起点**的边的条数叫点 v 的出度。

重边

若 E 中存在两个完全相同的元素（边）e_1, e_2，则它们被称作（一组）重边。

生成子图

在图 $G = (V(G), E(G))$ 中，选一些点和边，若这些点和边构成了**一棵树**，则这是这些点和边构成了一张**生成子图**。

连通图/强联通图

在一个**无向图**上的**任意**两个点可以互相到达，那么这张图叫做**连通图**。

在一个**有向图**上的**任意**两个点可以互相到达，那么这张图叫做**强连通图**。

稀疏图/稠密图

若一张图的边数远小于其点数的平方，那么它是一张**稀疏图**。

若一张图的边数接近其点数的平方，那么它是一张**稠密图**。

图的存储

图有三种存储方式。

邻接矩阵

方法

定义一个二维数组 $a_{u,v}$ 表示节点 u 到 v 之间是否有边；有边，则 $a_{u,v}=1$，反之，则 $a_{u,v}=0$。

对于**有权图**，$a_{u,v}$ 可以存储 u 到 v 的边权。

时间复杂度

查询两点间是否有边：$O(1)$。

遍历一个点所有出边：$O(n)$。

遍历整张图：$O(n^2)$。

空间复杂度：$O(n^2)$。

优点

可以在 $O(1)$ 的时间里查询两点间是否有边。

缺点

1. 只适用于图无重边的情况；
2. 对于点数较多的图，空间复杂度太大，无法接受；
3. 遍历一个点的所有出边和遍历整张图时间复杂度较大，难以接受。

代码实现

```
int a[maxn][maxn];
for(int i=1;i<=m;i++)//输入 m 条边
{
    int u,v,w;
    u=read();v=read();w=read();//无权图不需要输入 w
    a[u][v]=w;//无权图：a[u][v]=1;
    //无向图：a[v][u]=w;
}
```

邻接表

使用一个可以作为动态数组的数据结构（vector 或者 basic_string）来存边。

方法

定义 `basic_string<int>edge[maxn]`（或 vector），`edge[u]` 就表示点 u 所有出边信息。每次遇到一个 u,v,w，就连边，具体见下方实现代码。

时间复杂度

查询两点 u,v 之间是否有边：$O(d(u))$。

遍历一个点所有出边：$O(d(u))$。

遍历整张图：$O(n+m)$（n 是点数，m 是边数）。

空间复杂度：$O(m)$（**注意无向图需要开 2 倍空间**）。

优点

1. 遍历整张图和遍历一个所有出边的时间复杂度均较小；
2. 空间复杂度较小；
3. 尤其适用于需要对一个点的所有出边进行排序的场合。
4. 适用于稠密图。

缺点

判断两点间是否有边的时间复杂度较大。

代码实现（这里以有向无环图为例）

```
struct Node{int v,w;}//v 另一个端点，w 表示边权。

basic_string<Node>edge[maxn];

//存边
for(int i=1;i<=m;i++)
{
    int u,v,w;
    u=read();v=read();w=read();
    edge[u]+=Node{v,w};
}
//遍历一个点所有出边
for(Node y:edge[x])
{
    //y.v 即为终点
    //y.w 即为边权
}
```

链式前向星

~~由于这玩意写起来太麻烦我实在懒得用。不过为了方便以后复（重）习（开）还是总结一下吧。~~

方法

将邻接表换成**类链表**的形式即可。

对于一个点 u，定义 $head_u$ 表示以 u 为起点的第一条边编号，to_u 表示当前边的终点，nxt_i 表示 u 的第 i 条边的下一条边的编号，cnt 表示当前图中总共有多少边。

逆序存边。每次连一条边 u, v 时：

1. 边数 $cnt++$；
2. 将该边的 nxt 设为原 $head_u$；
3. 新的 $head_u$ 设为当前边数 cnt；
4. to_u 设为 v。

（这里说的比较简略，若想要更加深刻的了解链式前向星或没看懂的，可以移步链式前向星（详解）_Stephencurry's csdn的博客-CSDN博客_链式前向星）

时间复杂度：同邻接表时间复杂度。

空间复杂度：同邻接表空间复杂度。

优点

前两条同邻接表。

然而其实我也不知道它有什么其它的优点（至少现在未体会到）。先引（chao）用（xi） OIWiki 的一句话，等日后有所体会再回来补充吧：

> 优点是边是带编号的，有时会非常有用，而且如果 `cnt` 的初始值为奇数，存双向边时 `i ^ 1` 即是 `i` 的反边（常用于 网络流）。

缺点

1. 判断两点间是否有边的时间复杂度较大;
2. 不能方便地对一个点的出边进行排序;
3. 写起来显然比前两种麻烦的多（~~所以用这玩意干啥~~）。

代码实现

```
int head[maxn],to[maxn],nxt[maxn];
int cnt;
void add(int u,int v)//连一条从 u 到 v 的边
{
    nxt[++cnt]=head[u];
    head[u] = cnt;
    to[cnt] = v;
}
// 遍历 u 的出边
for(int i=head[u];i;i=nxt[i])
{
    int v = to[i];
}
```

最小生成树

定义

生成树：对于一个**无向连通图** $G=(V,E), n=|V|, m=|E|$，由 V 中的全部 n 个节点和 E 中的 $n-1$ 条边构成的无向联通子图叫做图 G 的一棵生成树。

最小生成树：对于一个**无向连通图**，其边权和最小的生成树就叫做这个无向连通图的**最小生成树（Minimum Spanning Tree）**，简称 MST。

注：最小生成树存在的前提是无向连通图，非无向连通图没有生成树。（非连通图只有最小生成森林）

Kruskal 算法

基本思想

利用**贪心**思想。

在任意时刻，都从**剩余的边**中选出一条权值最小，且该边两个端点不属于同一棵树（不连通），把该边加入 MST 中。

步骤

1. 对于每一个节点单独建立一个并查集；
2. 将图上所有的边从小到大排序；
3. 遍历每一条边。
4. 判断连接这条边的两个节点是否在同一个集合内。

   若不在，则将它们连边，同时加入到一个集合里。

   若在，则 `continue`。
5. 直到加入 $n-1$ 条边，即形成了一棵树，结束遍历。

时间复杂度

排序+并查集：$O(m\log m)$。

证明

~~咕咕咕~~

~~懒得写（之后补上吧，最近要总结的东西多了点）~~

代码实现

```
struct Node{int u,v,w;}e[maxm];//结构体存边
int fa[maxn];//并查集
int n,m,ans;

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

void add(int x,int y)
{
    x=find(x);y=find(y);
    fa[x]=y;
}

int cmp(Node a,Node b){return a.w<b.w;}//按照边权从小到大排序

void Kruskal()
{
    sort(e+1,e+m+1,cmp);
    for(int i=1;i<=m;i++)
    {
        if(find(x)!=find(y))//若不在一个集合则合并。
        {
            add(x,y);
            ans+=e[i].w;//ans 表示最小生成树边权和。
        }
     }
}

int main()
{
    n=read();m=read();
    for(int i=1;i<=m;i++)e[i].u=read(),e[i].v=read(),e[i].w=read();
    Kruskal();
    cout<<ans<<'\n';
}
```

例题

P3366【模板】最小生成树 - 洛谷

P1396 营救 - 洛谷：

P1967 [NOIP2013 提高组] 货车运输 - 洛谷：经典 LCA+Kruskal 练手题。

P2323 [HNOI2006]公路修建问题 - 洛谷：分别考虑两种公路，两边 Kruskal。0

P4047 [JSOI2010]部落划分 - 洛谷：可以加深对 Kruskal 本质的理解。

P2245 星际导航 - 洛谷（这题和货车运输本质上一样）

Prim 算法

基本思想

依然是**贪心**思想。

随便选择一个点作为起始点（加入到连通块中），然后每次从剩下的点中选择与当前连通块（最小生成树）距离最短的点加入到连通块中，直到所有的点都被加到这个连通块为止。那么这个连通块就是最小生成树。

步骤

1. 定义一个数组 dis_i 表示节点 i 到当前联通块的距离；
2. 随便选择一个点，加入到连通块（最小生成树）中；
3. 更新所有剩下的节点到 i 的距离；
4. 选择一个距离当前连通块距离最近的点 t 加入到连通块中；
5. 对于所有剩下的节点 i，判断 dis_i 是否大于 t 与 i 的距离；若大于，则 dis_i 更新为 t 到 i 的距离。
6. 重复步骤 2-5，直至所有的点加入到连通块中为止，则连通块即为最小生成树。

时间复杂度

$O(n^2)$，优先队列优化：$O(m \log n)$。

证明

适用范围

相对于 Kruskal：在稠密图尤其是完全图上，暴力 Prim 的复杂度比 Kruskal 优，但 **不一定** 实际跑得更快。

实现代码

```cpp
int calc(int a,int b){return (x[a]-x[b])*(x[a]-x[b])+(y[a]-y[b])*(y[a]-y[b]);}
//计算两点间距离
void add(int x)//将一个点加入连通块。
{
    for(int i=1;i<=n;i++)
    {
        if(vis[i])continue;
        if(i==x)continue;
        dis[i]=min(dis[i],calc(i,x));
    }
    vis[x]++;
}

void Prim()
{
    dis[1]=0;
    add(1);
    int T=n-1;
    while(T--)
    {
        int now=0;
        for(int i=1;i<=n;i++)
        {
            if(!vis[i]&&dis[i]<dis[now])now=i;
        }
        add(now);
        ans+=double(sqrt(dis[now]));
    }
}

signed main()
{
    n=read();
    for(int i=1;i<=n;i++)
    {
        x[i]=read();y[i]=read();
    }
    memset(dis,0x3f3f3f,sizeof(dis));//别忘了 dis 数组刚开始要赋值为 INF。
    Prim();
    cout<<ans<<endl;
}//懒得打一遍所以直接复制粘贴了我 公路修建 的代码。
```

例题

P1265 公路修建 - 洛谷：典型的 Prim 例题，用 Kruskal 会 MLE。

次小生成树

注：由于本人语文太差，所以以下内容部分借鉴了 *OIWiki* 上次小生成树讲解。

非严格次小生成树

定义

在一张无向图中，边权和最小的满足边权和 $\geq$ 最小生成树边权和的生成树。

求解步骤

1. 求出无向图的最小生成树，设其边权和为 val；
2. 遍历每条未被选中的边 $e = (u, v, w)$；
3. 找到最小生成树上 u 到 v 边权最大的一条边 $e' = (u', v', w')$：用在 货车运输 那道题里的思路，在倍增求 LCA 的过程中维护每个节点到其 2^i 级祖先的最大边权；
4. 用 e 替换 e'，可以得到一条边权和 $val' = val - e' + e$ 的生成树；
5. 由于求的是次小，所以只需要在上述所有求得的 val' 取**最小值**即可。

严格次小生成树

定义

在一张无向图中，边权和最小的满足边权和严格 $>$ 最小生成树边权和的生成树。

求解步骤

考虑在求严格次小生成树的过程中稍作改动。

在刚刚的求解过程中，之所以是**非严格大于**，是因为最小生成树保证生成树中 u 到 v 路径上的边权最大值一定**不大于**其它从 u 到 v 路径的边权最大值。即：我们在用 e 替换 e' 时，两者边权可能是相等的。

解决方法：在倍增求 LCA 的过程中维护每个节点到其 2^i 级祖先的最大边权的同时维护**严格次大边权**，当最大边权与原最小生成树上最大边权相等，用严格次大值替换；

时间复杂度

$O(m \log m)$。

代码实现

```
//摘抄我的 P4180 严格次小生成树
struct P{int u,v,w;}e[maxn<<1];
int f[maxn],vis[maxn],fa[maxn][25],tt[5];
int maxx[maxn][25],minn[maxn][25],dep[maxn];
//maxx 表示的是
int n,m,ans,ss;

struct Node{int v,w;};
basic_string<Node>edge[maxn<<1];

int cmp(P a,P b){return a.w<b.w;}

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

void add(int x,int y)
{
    x=find(x);y=find(y);
    f[x]=y;
}

void Kruskal()//求最小生成树
{
    sort(e+1,e+m+1,cmp);
    for(int i=1;i<=m;i++)
    {
        if(find(e[i].u)!=find(e[i].v))
        {
            add(e[i].u,e[i].v);
            ss+=e[i].w;
            vis[i]++;
            edge[e[i].u]+=Node{e[i].v,e[i].w};
            edge[e[i].v]+=Node{e[i].u,e[i].w};
        }
    }
}

void dfs(int x,int fath)//dfs 过程中倍增求出最大和次大边权
{
    dep[x]=dep[fath]+1;
    fa[x][0]=fath;
    minn[x][0]=-INF;
    for(int i=1;i<=20;i++)
    {
        fa[x][i]=fa[fa[x][i-1]][i-1];
        tt[1]=maxx[x][i-1];tt[2]=maxx[fa[x][i-1]][i-1];
        tt[3]=minn[x][i-1];tt[4]=minn[fa[x][i-1]][i-1];
```

```
        sort(tt,tt+4);
        maxx[x][i]=tt[3];
        int t=2;
        while(t>=0&&tt[t]==tt[3])t--;
        if(t<0)minn[x][i]=-INF;
        else minn[x][i]=tt[t];
    }
    for(Node y:edge[x])
    {
        if(y.v==fath)continue;
        maxx[y.v][0]=y.w;
        dfs(y.v,x);
    }
}

int lca(int u,int v)
{
    if(dep[u]<dep[v])swap(u,v);
    for(int i=0;i<=20;i++)
    {
        if((dep[u]-dep[v])&(1<<i))u=fa[u][i];
    }
    if(u==v)return u;
    for(int i=20;i>=0;i--)
    {
        if(fa[u][i]!=fa[v][i]){u=fa[u][i];v=fa[v][i];}
    }
    return fa[u][0];
}

int query(int x,int y,int val)
{
    int ret=-INF;
    for(int i=20;i>=0;i--)
    {
        if(dep[fa[x][i]]>=dep[y])
        {
            if(val!=maxx[x][i])ret=max(ret,maxx[x][i]);
            else ret=max(ret,minn[x][i]);
            x=fa[x][i];
        }
    }
    return ret;
}

void work()
{
    ans=INFLL;
    for(int i=1;i<=m;i++)
    {
        if(!vis[i])
```

```
        {
            int LCA=lca(e[i].u,e[i].v);
            int x=query(e[i].u,LCA,e[i].w);
            int y=query(e[i].v,LCA,e[i].w);
            int kk=max(x,y);
            if(kk!=-INF)ans=min(ans,ss-kk+e[i].w);
        }
    }
    if(ans==INFLL)cout<<-1<<endl;
    else cout<<ans<<endl;
}

signed main()
{
    n=read();m=read();
    for(int i=1;i<=n;i++)f[i]=i;
    for(int i=1;i<=m;i++){e[i].u=read();e[i].v=read();e[i].w=read();}
    Kruskal();
    dfs(1,0);
    work();
    return 0;
 }
```

Kruskal 重构树

定义/步骤

1. 前三步同 Kruskal 算法；
2. 判断连接这条边的两个节点是否在同一个集合内。

   若不在，则：

   - 将他们连边；
   - 新建一个点，点权为加入边的边权；
   - 将两个集合的根节点分别设为新建点的左儿子和右儿子
   - 将两个集合和新建点合并成一个集合。将新建点设为根。
3. 直到加入 $n-1$ 条边，即形成了一棵树，结束遍历。

那么形成的这个有 n 个叶子节点的**二叉树**，我们就叫做 Kruskal 重构树。

性质

1. 是一棵有根二叉树，根节点是最后新建节点；

2. 若原图联通，则 Kruskal 重构树会比原图多 $n-1$ 个节点（连了 $n-1$ 条边嘛）；

3. 上述定义下（即：边权**从小到大**排序），节点 u 到 v 路径上最大边权的最小值 = Kruskal 重构树上 $lca(u,v)$；

4. 边权**从大到小**排序，节点 u 到 v 路径上最小边权的最大值 = Kruskal 重构树上 $lca(u,v)$。

适用范围

求图上两点路径上最大边权最小值/最小边权最大值。

代码实现（以最小边权最大为例）

```
//复制粘贴我 货车运输 的代码。码风可能和现在有所区别。但~~懒得改了~~
int find(int x)
{
    if(x!=f[x]){f[x]=find(f[x]);}
    return f[x];
}

void dfs(int x,int fath,int v)
{
    dep[x]=dep[fath]+1;
    sum[x][0]=v;
    fa[x][0]=fath;
    for(int i=1;i<=20;i++)
    {
        fa[x][i]=fa[fa[x][i-1]][i-1];
        sum[x][i]=min(sum[x][i-1],sum[fa[x][i-1]][i-1]);
    }
    for(Node y:edge[x])
    {
        if(y.v==fath)continue;
        dfs(y.v,x,y.w);
    }
}

int work(int x,int y)
{
    if(dep[x]<dep[y])swap(x,y);
    int ans=100000;
    for(int i=0;i<=20;i++)
    {
        if((dep[x]-dep[y])&(1<<i))
        {
            ans=min(ans,sum[x][i]);
            x=fa[x][i];//注意这两句位置不能换！！！
        }
    }
    if(x==y)return ans;
    for(int i=20;i>=0;i--)
    {
        if(fa[x][i]!=fa[y][i])
        {
            ans=min(ans,min(sum[x][i],sum[y][i]));
            x=fa[x][i];
            y=fa[y][i];
        }
    }
    int t=sum[x][0];
    if(fa[x][0]!=y) t=min(t,sum[y][0]);
    return min(t,ans);
}
```

```cpp
int main()
{
    memset(sum,0x3f3f3f,sizeof(sum));
    n=read();m=read();
    for(int i=1;i<=m;i++)
    {
        e[i].x=read();e[i].y=read();e[i].z=read();
    }
    sort(e+1,e+m+1,cmp);
    for(int i=1;i<=n;i++)f[i]=i;
    for(int i=1;i<=m;i++)
    {
        int a=find(e[i].x),b=find(e[i].y);
        if(f[a]==b)continue;
        f[a]=b;
        edge[e[i].x]+=((Node){e[i].y,e[i].z});
        edge[e[i].y]+=((Node){e[i].x,e[i].z});
    }
    for(int i=1;i<=n;i++)
    {
        if(f[i]==i)dfs(i,i,100000);
     }
    int q=read();
    while(q--)
    {
        int x,y;// q 组询问，每次求 x 和 y 路径上最小边权最大值。
        x=read();y=read();
        int X=find(x),Y=find(y);
        if(X!=Y)cout<<-1<<'\n';
        else cout<<work(x,y)<<'\n';
    }
    return 0;
}
```

例题

P1967 [NOIP2013 提高组] 货车运输 - 洛谷：可以看做是生成树，也可以看做是 Kruskal 重构树求最小边权最大。

P2245 星际导航 - 洛谷：几乎和 货车运输 一样，只是求的是最大边权最小值。

P7834 [ONTAK2010] Peaks 加强版 - 洛谷：离散化+Kruskal 重构树+树上倍增+主席树。

拓扑排序

定义

~~感觉自己完全叙述不出来于是抄子度娘~~

> 对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边<u,v>∈E(G)，则u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑次序(Topological Order)的序列，简称拓扑序列。简单的说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为拓扑排序。 ——百度百科

可以看出，拓扑排序的目标是将所有的节点排序，使得排在前面的点不依赖排在后面的点。

其实这也就是 DP 求解的本质。

我们在 DP 中学到，一个 DP 问题，求解大的状态依赖于小的状态。 只有当小状态求解完成之后，才能获取大状态的解。 这些依赖关系形成了 DAG，即：

- 自顶向下 + 记忆化的求解，对应自顶向下的拓扑排序。
- 自底向上的 DP 求解，对应自底向上的拓扑排序。

（这一点会在我之后复（重）习（开）DP 后提到，~~如果不咕的话~~）

求解步骤

1. 建立一个空队列 q；
2. 在图上找到所有**入度**为 0 的点，将它们入队；
3. 对于所有在队列中的点：
    - 出队；
    - 遍历所有与它们连边的点 i，将它们出度减一；

      若此时 i 的入度为 0，则将其入队。
4. 重复步骤 2-3；

优化/拓展

对于求字典序最大/最小的拓扑排序：可将队列换成优先队列实现。

时间复杂度

设 DAG 有 n 个点 m 条边。

普通队列下：$O(n+m)$。

优先队列求字典序最大/最小：$O(n\log n+m)$。

代码实现

```
for(int i=1;i<=m;i++)
{
    int u,v;
    u=read();v=read();
    edge[u]+=v;
    in[v]++;//in[v] 表示 v 的入度。
}
for(int i=1;i<=n;i++)//将第一轮所有入度为 0 的点入队。
{
    if(!in[i])q.push(i);
}
while(!q.empty())
{
    int now=q.front();
    q.pop();
    for(int y:edge[now])
    {
        in[y]--;
        if(!in[y])q.push(y);
    }
}
```

例题

P1347 排序 - 洛谷：去年 9 月份写的，已经差不多忘了题意了，但似乎挺板的（？）

P7113 [NOIP2020] 排水系统 - 洛谷：比较板的题，注意最后一个点会爆 ull，所以请使用 int128 或者把一个数压成两个数的方法存储。

P3243 [HNOI2015]菜肴制作 - 洛谷：反向 topsort+优先队列求字典序。

P1983 [NOIP2013 普及组] 车站分级 - 洛谷

最短路

一些说明/概念

单源最短路径：图上一个点到其它所有点的最短路径；

多源最短路径：图上每个点分别作为起点和终点的最短路径；

下面要说明的几种算法都是针对**有权有环图**而言的。

在 DAG 上可以直接用 topsort 来求最短路径；

在**无权图**上可以直接 BFS。

Floyd 算法

适用范围

Floyd 算法用于求解**多源最短路**。

求解过程

定义 $dp_{k,x,y}$ 表示从 x 到 y 只经过编号 $\leq k$ 的节点的最短路径。

初始化：$dp_{0,x,y}=e(x,y)$。

特殊地，所有的 $dp_{0,x,x}=0(x=y)$；若 x,y 不连边，则 $dp_{0,x,y}=\infty$（设成 ∞ 是因为后面转移的时候要取最小值）。

考虑转移。有两种情况：

- 经过编号为 k 的点：$dp_{k,x,y}=dp_{k-1,x,k}+dp_{k-1,k,y}$。
- 不经过编号为 k 的点：$dp_{k,x,y}=dp_{k-1,x,y}$。

上述两种情况取最小值即可。

那么对于所有的 x,y，$dp_{n,x,y}$ 即为答案。

空间复杂度 $O(n^3)$，n 稍大就会 MLE，考虑优化。

可以发现第一维的 k 只与上一层的 $k-1$ 有关，所以可以省略。

那么有：

$$dp_{x,y}=\min(dp_{x,k}+dp_{k,y},dp_{x,y})$$

复杂度

时间复杂度：$O(n^3)$。

空间复杂度：$O(n^2)$。

代码实现

```
//注意 k 是最外层循环，i 是次外层，j 是内层。
//不能是以 i,j,k 的顺序循环。
memset(dp,0x3f,sizeof(dp));
for(int i=1;i<=m;i++)
{
    int u,v,w;
    u=read();v=read();w=read();
    dp[u][v]=min(dp[u][v],w);
}
for(int k=1;k<=n;k++)
{
    for(int x=1;x<=n;x++)
    {
        for(int y=1;y<=n;y++)
        {
            dp[x][y]=min(dp[x][y],dp[x][k]+dp[k][y]);
        }
        dp[x][x]=0;
    }
}
```

拓展：Floyd 寻找无向图最小环

考虑 Floyd 算法 的求解过程，当外层循环 k 开始时，$dp_{x,y}$ 表示的是经过节点**不超过 $k-1$ 的节点**从 x 到 y 的最短路长度。

所以，$\min\{dp_{i,j}+a_{j,k}+a_{k,i}\}$ 满足以下两个条件的最小环长度：

1. 由编号不超过 k 的节点构成；
2. 经过节点 k。

上述 i,j 实际上就是与 k 相邻的两个点。

对于 $k\in[1,n]$，利用上述进行计算，取最小值即可。

例题

P2935 [USACO09JAN]Best Spot S - 洛谷：比较板的一道基础题。

P6175 无向图的最小环问题 - 洛谷

P1119 灾后重建 - 洛谷：Floyd 变形，有助于加深对于 Floyd 算法的理解；

dijkstra 算法

适用范围

适用于求**单源最短路径**。但只适用于所有边长度都是**非负数**的图。

其实在一些要求**多源最短路径**的问题中，也可以以每个点为起点，跑 n 轮 dijkstra 求得多源最短路。

求解思路

定义一个数组 dis_i 表示从起点到 i 的最短路径的长度，vis_i 表示节点是否被标记。

不断利用类似 Prim 算法的贪心策略，选取一个最短路最小的点，并用这个点更新与这个点相连的所有点的最短路。

求解步骤

1. 初始化所有节点 i 的 $vis_i = 0$；
2. 初始化起点 $dis_s = 0$，其余 $dis_i = \infty$；
3. 找出一个未被标记的，dis_x 最小的节点 x，标记 x；
4. 遍历节点 x 的所有出边 (x, y, w)，若 $dis_y > dis_x + w$，则 $dis_y = dis_x + w$；
5. 重复步骤 2-3，直至所有节点被标记。

时间复杂度

普通 dij：$O(n^2)$。

优先队列优化：

对于一条边的更新：$O(\log n)$；

总时间复杂度：$O(m \log n)$。

代码实现

```cpp
struct Node
{
    int v,w;
    bool operator<(const Node &t)const
    {
        return w>t.w;
    }
};
basic_string<Node>edge[maxn<<1];
priority_queue<Node>q;
int dis[maxn],vis[maxn];
int n,m,s;

void dijkstra(int s)
{
    for(int i=1;i<=n;i++)dis[i]=(1ll<<31)-1;
    dis[s]=0;
    q.push(Node{s,dis[s]});
    while(!q.empty())
    {
        Node now=q.top();
        q.pop();
        if(vis[now.v])continue;
        vis[now.v]++;
        for(Node y:edge[now.v])
        {
            if(dis[y.v]>dis[now.v]+y.w)
            {
                dis[y.v]=dis[now.v]+y.w;
                q.push(Node{y.v,dis[y.v]});
            }
        }
    }
}
```

例题

P3371【模板】单源最短路径（弱化版） - 洛谷

P4779【模板】单源最短路径（标准版） - 洛谷

P1144 最短路计数 - 洛谷：比较简单的一道计数题。

P4673 [BalticOI 2005]Bus Trip - 洛谷：抽象化的图，比较典型的例题，之后会考虑写一篇题解专门来记录一下这个典型的转化。

P5468 [NOI2019] 回家路线 - 洛谷：虽然正解并非 dij，但可以 AC（雾），思想上和 Bus Trip 具有异曲同工之处，之后同样会写一篇题解加以记录。

P7473 [NOI Online 2021 入门组] 重力球 - 洛谷：可以姑且理解为 dij+BFS，同样很经典。

P3953 [NOIP2017 提高组] 逛公园 - 洛谷：dij+topsort，两边 dij 反向建图典型例题。

Bellman-ford & SPFA

适用范围

1. 也适用于**单源最短路径。**

   与 dijkstra 不同的是，可以用于有**负权**的边。

2. 可用于**判断负环。**

Bellman-ford 求解步骤

1. 初始化起点 $dis_i = 0$，其余 $dis_i = \infty$；
2. 扫描所有边 (x, y, w)，若 $dis_y > dis_x + w$，则 $dis_y = dis_x + w$；
3. 重复步骤 2，直至所有边被更新完。

SPFA 优化

1. 初始化一个队列 `queue`；
2. 将起点入队；
3. 取出队头元素 x，扫描其所有出边 (x, y, w)，若 $dis_y > dis_x + w$，则 $dis_y = dis_x + w$；
4. 将 y 入队；
5. 重复步骤 3-4，直至队列为空；

时间复杂度

Bellman-ford：$O(nm)$；

SPFA：虽然一般情况下较快，最坏情况下可能被卡成 $O(nm)$，**考场上一般若无负权不使用！而使用 dijkstra。**

代码实现

```cpp
//SPFA 优化
void SPFA(int s)
{
    for(int i=1;i<=n;i++)dis[i]=(1ll<<31)-1;
    dis[s]=0;
    q.push(s);
    while(!q.empty())
    {
        int now=q.top();
        q.pop();
        for(Node y:edge[now])
        {
            if(dis[y.v]>dis[now]+y.w)
            {
                dis[y.v]=dis[now]+y.w;
                q.push(y.v);
            }
        }
    }
}
```

例题

P1073 [NOIP2009 提高组] 最优贸易 - 洛谷：反向建图+两遍 SPFA；

P1462 通往奥格瑞玛的道路 - 洛谷：SPFA+二分答案；

P1979 [NOIP2013 提高组] 华容道 - 洛谷：正解是 SPFA+BFS。

差分约束系统

问题描述

给出一组包含 m 个不等式，有 n 个未知数的形如：

$$\begin{cases} x_{c_1} - x_{c_1'} \leq y_1 \\ x_{c_2} - x_{c_2'} \leq y_2 \\ \cdots \\ x_{c_m} - x_{c_m'} \leq y_m \end{cases}$$

的不等式组，求任意一组满足这个不等式组的解。

求解

首先，对于差分约束的每个约束条件 $x_{c_k} - x_{c_k'} \leq y_k$，可以变形为 $x_{c_k} \leq x_{c_k'} + y_k$。

观察会发现和 dijkstra 里面的转移式 $dis_y =$

咕咕咕……

post on 2022.6.6：由于最近艾教讲了图的连通性相关内容但是我感觉我没完全理解，而又恰好要去做连通性相关的题目，所以这部分先咕一段时间，等我总结完连通性相关内容之后再回过头来补上，包括这个部分下面的负环和另一个板块欧拉回路部分也会之后回来补上。

图的连通性

关于 Tarjan 算法的一些概念

搜索树相关

在一张无向连通图中，**任选**一个节点出发进行 DFS，每个点只访问一次，所有发生递归的边 (x, y) 构成了一棵树，这棵树就叫做**搜索树**。

对于一般的（非联通）无向图，DFS 一遍形成的是**搜索森林**。

时间戳

在 DFS 的过程中，按照每个节点第一次被访问的时间顺序，依次给图中的 n 个节点 $1-n$ 的整数标记，该标记被称为**时间戳**，记为 dfn_x。

回溯值

定义

设 $subtree(x)$ 表示无向图上搜索树中以 x 为根的子树。

x 的回溯值 low_x 定义为满足以下条件的节点的**最小时间戳**：

1. $subtree(x)$ 中的节点；
2. 通过一条不在搜索树上的边，能够达到的 $subtree(x)$ 的节点。

求解步骤

假设计算的是节点 x 的回溯值 low_x。

1. 初始化 $low_x = dfn_x$；
2. 扫描从 x 出发的每条边 (x, y)：

- 若 (x,y) 是搜索树上边，令 $low_x=\min(low_x,low_y)$；
- 若 (x,y) 不是搜索树上边，则 (x,y) 是**返祖边**，令 $low_x=\min(low_x,dfn_y)$。

割边/桥

定义

给定一张无向连通图 $G=(V,E)$，对于一条边 $e\in E$，若从图中删去边 e 后，G 分裂成两个**不相连**的子图，则称 e 为 G 的**割边**或**桥**。

求解

首先，割边一定是搜索树上的边，因为如果不是搜索树上的边，删边之后仍然存在生成树。

其次，对于一条搜索树上的边 (x,y)，x 是 y 的父亲。当且仅当 $low_y>dfn_x$ 时，该边是 G 的割边。因为 y 无法通过搜索树以外的边到达除了他子树以外（x 或比 x 更早访问）的任何点。

代码实现

```
void dfs(int x,int pre)
{
    dfn[x]=low[x]=++tot;
    int ret=0;
    for(auto nxt:edge[x])
    {
        int y=nxt.first,id=nxt.second;
        if(!dfn[y])
        {
            dfs(y,id);
            low[x]=min(low[x],low[y]);
            if(low[y]>dfn[x])cout<<x<<"->"<<y<<"是割边"<<'\n';
        }
        else if(id!=pre)low[x]=min(low[x],dfn[y]);
    }
    return ;
}
```

割点

模板题目链接

P3388【模板】割点（割顶） - 洛谷

定义

给定一张无向连通图 $G=(V,E)$，对于一个节点 $x \in V$，若从图中删去节点 x 以及所有与 x 相连的边后，G 分裂成两个或两个以上**不相连**的子图，则称 x 是 G 的**割点**。

求解

分类讨论。

- x **不是**搜索树根节点：当且仅当搜索树上存在 x 的一个子节点 y，满足：$low_y \geq dfn_x$；
- x **是**搜索树根节点：当且仅当搜索树上至少存在两个子节点 y_1, y_2 满足上述条件。

代码实现

```
int tot;
int dfn[maxn],low[maxn];
basic_string<pair<int,int> >edge[maxn];
basic_string<int>ans;
void dfs(int x,int pre)
{
    dfn[x]=low[x]=++tot;
    int ret=0;
    for(auto nxt:edge[x])
    {
        int y=nxt.first,id=nxt.second;
        if(!dfn[y])
        {
            dfs(y,id);
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x])ret++;
        }
        else if(id!=pre)low[x]=min(low[x],dfn[y]);
    }
    if(ret>=2||(ret==1&&pre!=0))ans+=x;
    return ;
}

int main()
{
    int n,m;
    n=read();m=read();
    for(int i=1;i<=m;i++)
    {
        int x,y;
        x=read();y=read();
        edge[x]+=mk(y,i);
        edge[y]+=mk(x,i);
    }
    for(int i=1;i<=n;i++)
    {
        if(!dfn[i])dfs(i,0);
    }
    sort(ans.begin(),ans.end());
    cout<<ans.size()<<'\n';
    for(int i:ans)cout<<i<<" ";
    cout<<'\n';
    return 0;
}
```

无向图的双连通分量

双连通图

定义

点双连通图：若一张无向连通图中不存在**割点**，则称之为点双联通图；

边双连通图：若一张无向连通图中不存在**桥**，则称之为边双联通图；

定理

1. 一张无向连通图是**点双连通图**，当且仅当满足下列两个条件之一：
    - 图的顶点数不超过 2；
    - 图中任意两点都同时包含在至少一个简单环（即：不自交的环）中。
2. 一张无向连通图是**边双连通图**，当且仅当满足任意一条边都包含在至少一个简单环中。

证明：

> 相比于结论，证明似乎就不是那么重要了（大雾），先咕着，其它东西整理完再说。。。

边双连通分量（e-DCC）

模板题链接

T103489【模板】边双连通分量 - 洛谷

定义

无向图的极大**边双**连通子图被称为**边双联通分量**，简称：e-DCC。

求解

求出无向图中**所有的桥**，把桥都删除后，无向图会分成若干个连通块，每一个连通块就是一个**边双连通分量**。

具体实现上，可以先用 Tarjan 算法标记出所有的桥边。然后对于整个无向图进行一次 DFS，遍历过程中不访问桥边，划分出每个连通块。

代码实现

```cpp
int dfn[maxn],low[maxn],vis[maxm],book[maxn],tot;

basic_string<pair<int,int> >edge[maxn];

void dfs(int x,int pre)
{
    dfn[x]=low[x]=++tot;
    int ret=0;
    for(auto nxt:edge[x])
    {
        int y=nxt.first,id=nxt.second;
        if(!dfn[y])
        {
            dfs(y,id);
            low[x]=min(low[x],low[y]);
            if(low[y]>dfn[x])vis[id]++;
        }
        else if(id!=pre)low[x]=min(low[x],dfn[y]);
    }
}

void dfs2(int x,int pre)
{
    book[x]++;
    for(auto nxt:edge[x])
    {
        int y=nxt.first,id=nxt.second;
        if(!vis[id]&&!book[y])dfs2(y,id);
    }
}

int main()
{
    int n,m;
    n=read();m=read();
    for(int i=1;i<=m;i++)
    {
        int x,y;
        x=read();y=read();
        edge[x]+=mk(y,i);
        edge[y]+=mk(x,i);
    }
    for(int i=1;i<=n;i++)
    {
        if(!dfn[i])dfs(i,0);//第一个 dfs 用来求所有的桥。见上文求割边。
    }
    int ans=0;
    for(int i=1;i<=n;i++)
    {
        if(!book[i])ans++,dfs2(i,0);//第二个 dfs 用来标记删掉桥边每个连通块上的点。
```

```
    }
    cout<<ans<<'\n';
    return 0;
}
```

缩点

可以把每个 e-DCC 看做一个节点，把桥边 (x, y) 看做连接 x 和 y 所在 e-DCC 的无向边，就会产生一棵树（在非连通无向图下，会产生森林）。

这就是 e-DCC 的缩点。

点双连通分量（v-DCC）

模板题链接

T103492【模板】点双连通分量 - 洛谷

定义

无向图的极大**点双**连通子图被称为**点双连通分量**。

误区

注意：点双连通分量不是单纯的“删除割点后图中剩余的连通块”。

求解

为了求出**点双连通分量**，需要在 Tarjan 算法的过程中维护一个栈，并按照如下的方法维护栈中元素：

1. 当一个节点第一次被访问时，把该节点入栈；
2. 当 $low_y \geq dfn_x$ 成立时：
    - 从栈顶不断弹出节点，直至节点 y 被弹出；
    - 刚才弹出的所有节点与节点 x 一起构成一个 v-DCC。

代码实现

参考资料

网络资料

图论相关概念 - OI Wiki

最小生成树 - OI Wiki

拓扑排序 - OI Wiki

拓扑排序_百度百科

纸质资料

算法竞赛进阶指南 - 0x60 图论

深入浅出程序设计竞赛进阶篇（书稿） - 图论

图论基础 - aqx.pptx

（以上参考资料按总结顺序排序。）

To be continued……