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

```cpp
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);
}
```

```
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了度娘~~

> 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;
    }
}
```

dijkstra 算法

适用范围

适用于求**单源最短路径。**

其实在一些问题中，也可以

Bellman-ford & SPFA

参考资料

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

算法竞赛进阶指南 - 0x60 图论（纸质资料）

（以上参考资料按总结顺序排序。）

To be continued......