图论

1. Floyd

```cpp
#include<iostream>
using namespace std;

const int INF = 1e4;
const int N = 101;

int dis[N][N];

int main() {
    int n, m;
    cin >> n >> m;
    int u, v, w;

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            if(i != j) {
                dis[i][j] = INF;
            }
        }
    }

    while(m--) {
        cin >> u >> v >> w;
        if(dis[u][v] == INF || dis[u][v] > w) {
            dis[u][v] = w;
            dis[v][u] = w;//无向图
        }
    }

    for(int k = 1; k <= n; k++) {
        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n; j++) {
                if(dis[i][k] != INF && dis[k][j] != INF && dis[i][k] + dis[k][j] < dis[i][j]) {
                    dis[i][j] = dis[i][k] + dis[k][j];
                }
            }
        }
    }

    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            cout << dis[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

2.Dijsktra

```
#include<bits/stdc++.h>

const int MaxN = 100010, MaxM = 500010;

struct edge{
    int to, dis, next;
};

edge e[MaxM];
int head[MaxN], dis[MaxN], cnt;
bool vis[MaxN];
int n, m, s;

void add_edge( int u, int v, int d ){
    cnt++;
    e[cnt].dis = d;
    e[cnt].to = v;
    e[cnt].next = head[u];
    head[u] = cnt;
}

struct node{
    int dis;
    int pos;
    bool operator <( const node &x )const{
        return x.dis < dis;
    }
};

std::priority_queue<node> q;

inline void dijkstra(){
    dis[s] = 0;
    q.push( ( node ){0, s} );
    while( !q.empty() ){
        node tmp = q.top();
        q.pop();
        int x = tmp.pos, d = tmp.dis;
        if( vis[x] )
            continue;
        vis[x] = 1;
        for( int i = head[x]; i; i = e[i].next ){
            int y = e[i].to;
            if( dis[y] > dis[x] + e[i].dis ){
                dis[y] = dis[x] + e[i].dis;
                if( !vis[y] ){
                    q.push( ( node ){dis[y], y} );
                }
            }
        }
    }
}

int main(){
    scanf( "%d%d%d", &n, &m, &s );
    for(int i = 1; i <= n; ++i)dis[i] = 0x7fffffff;
    for(int i = 0; i < m; ++i ){
        int u, v, d;
        scanf( "%d%d%d", &u, &v, &d );
        add_edge( u, v, d );
    }
    dijkstra();
    for( int i = 1; i <= n; i++ )
        printf( "%d ", dis[i] );
    return 0;
}
```

2.Prim

```
#include<bits/stdc++.h>
using namespace std;
int read(){
    int x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-') f=-1;c=getchar();}
    while(c>='0'&&c<='9') x=(x<<3)+(x<<1)+(c^48),c=getchar();
    return x*f;
}//快读
#define inf 123456789
#define maxn 5005
#define maxm 200005
struct edge{
        int v,w,next;
}e[maxm<<1];
//注意是无向图，开两倍数组
int head[maxn],dis[maxn],cnt,n,m,tot,now=1,ans;
//已经加入最小生成树的的点到没有加入的点的最短距离，比如说1和2号节点已经加入了最小生成
```

```
树，那么dis[3]就等于min(1->3,2->3)
bool vis[maxn];
//链式前向星加边
void add(int u,int v,int w){
        e[++cnt].v=v;
        e[cnt].w=w;
        e[cnt].next=head[u];
        head[u]=cnt;
}
//读入数据
void init(){
    n=read(),m=read();
    for(int i=1,u,v,w;i<=m;++i){
        u=read(),v=read(),w=read();
        add(u,v,w),add(v,u,w);
    }
}
int prim(){
        //先把dis数组附为极大值
        for(re int i=2;i<=n;++i){
                dis[i]=inf;
        }
    //这里要注意重边，所以要用到min
        for(re int i=head[1];i;i=e[i].next){
                dis[e[i].v]=min(dis[e[i].v],e[i].w);
        }
    while(++tot<n){//最小生成树边数等于点数-1
        int minn=inf;//把minn置为极大值
        vis[now]=1;//标记点已经走过
        //枚举每一个没有使用的点
        //找出最小值作为新边
        //注意这里不是枚举now点的所有连边，而是1~n
        for(int i=1;i<=n;++i){
            if(!vis[i]&&minn>dis[i]){
                minn=dis[i];
                                now=i;
            }
        }
        ans+=minn;
        //枚举now的所有连边，更新dis数组
        for(int i=head[now];i;i=e[i].next){
                int v=e[i].v;
                if(dis[v]>e[i].w&&!vis[v]){
                        dis[v]=e[i].w;
                }
                }
    }
    return ans;
}
int main(){
    init();
    printf("%d",prim());
    return 0;
}
```

3.Kruskal

```
#include<bits/stdc++.h>
using namespace std;
int read(){
    int x=0,f=1;char c=getchar();
    while(c<'0'||c>'9'){if(c=='-') f=-1;c=getchar();}
    while(c>='0'&&c<='9') x=(x<<3)+(x<<1)+(c^48),c=getchar();
    return x*f;
}
struct Edge{
        int u,v,w;
}edge[200005];
int fa[5005],n,m,ans,eu,ev,cnt;
bool cmp(Edge a,Edge b){
    return a.w<b.w;
}
//快排的依据（按边权排序）
int find(int x){
    while(x!=fa[x]) x=fa[x]=fa[fa[x]];
    return x;
}
//并查集循环实现模板，及路径压缩
void kruskal(){
    sort(edge,edge+m,cmp);
    //将边的权值排序
    for(int i=0;i<m;i++){
        eu=find(edge[i].u), ev=find(edge[i].v);
        if(eu==ev){
            continue;
        }
        //若出现两个点已经联通了，则说明这一条边不需要了
        ans+=edge[i].w;
        //将此边权计入答案
        fa[ev]=eu;
        //将eu、ev合并
        if(++cnt==n-1){
            break;
        }
        //循环结束条件，及边数为点数减一时
    }
}
int main(){
    n=read(),m=read();
    for(int i=1;i<=n;i++){
        fa[i]=i;
    }
    //初始化并查集
    for(int i=0;i<m;i++){
        edge[i].u=read(),edge[i].v=read(),edge[i].w=read();
    }
    kruskal();
    printf("%d",ans);
    return 0;
}
```

4.SPFA判负环

```
#include<cstdio>
#include<cstring>
#include<queue>
#define inf 0x3f3f3f3f
using namespace std;
const int MAXN=2010;
const int MAXM=3010;
int n,m;

int en=-1,eh[MAXN];
struct edge{
        int u,v,w,next;
        edge(int U=0,int V=0,int W=0,int N=0):u(U),v(V),w(W),next(N){}
};
edge e[MAXM<<1];
void add_edge(int u,int v,int w){
        e[++en]=edge(u,v,w,eh[u]);eh[u]=en;
}
void input(){
        scanf("%d %d",&n,&m);
        en=-1;
        memset(eh,-1,sizeof(eh));
        int u,v,w;
        for(int i=1;i<=m;++i){
                scanf("%d %d %d",&u,&v,&w);
                add_edge(u,v,w);
                if(w>=0)add_edge(v,u,w);
        }
}

int dis[MAXN],cnt[MAXN];
bool vis[MAXN];
queue<int> q;
```

```
void spfa(){
        fill(dis+1,dis+n+1,inf);
        memset(cnt,0,sizeof(cnt));
        memset(vis,0,sizeof(vis));

        while(!q.empty())q.pop();
        dis[1]=0;vis[1]=1;q.push(1);

        int u,v,w;
        while(!q.empty()){
                u=q.front();vis[u]=0;q.pop();
                for(int i=eh[u];i!=-1;i=e[i].next){
                        v=e[i].v;w=e[i].w;
                        if(dis[u]+w<dis[v]){
                                dis[v]=dis[u]+w;
                                cnt[v]=cnt[u]+1;//记录最短路径的边数
                                if(cnt[v]>=n){//最短路径边数≥n，即存在被重复遍历的点，也就是存在负环
                                        printf("YES\n");
                                        return;
                                }
                                if(!vis[v]){
                                        vis[v]=1;q.push(v);
                                }
                        }
                }
        }
        printf("NO\n");
}

int main(){
        int t;
        scanf("%d",&t);
        for(int i=1;i<=t;++i){
                input();
                spfa();
        }
        return 0;
}
```

5.拓扑排序

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

using namespace std;

```

```
const int N = 100010;

int n, m;
int h[N], e[N], ne[N], idx;
int d[N];
int q[N];

void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

bool topsort(){
    int hh = 0, tt = -1;

    for (int i = 1; i <= n; i ++ ){
        if (!d[i]){
            q[ ++ tt] = i;
        }
    }
    while (hh <= tt){
        int t = q[hh ++ ];

        for (int i = h[t]; i != -1; i = ne[i]){
            int j = e[i];
            if (-- d[j] == 0)
                q[ ++ tt] = j;
        }
    }

    return tt == n - 1;
}

int main(){
    scanf("%d%d", &n, &m);

    memset(h, -1, sizeof h);

    for (int i = 0; i < m; i ++ ){
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);

        d[b] ++ ;
    }

    if (!topsort()){
            puts("-1");
    }else{
        for (int i = 0; i < n; i ++ ) printf("%d ", q[i]);
        puts("");
    }
    return 0;
}
```

6.二分图判断(染色法)

```
#include <cstdio>
#include <cstring>

using namespace std;

const int N = 100010, M = 200010;
int h[N], e[M], ne[M], idx;
int color[N];

void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

bool dfs(int u, int c){
    color[u] = c;
    for (int i = h[u]; i != -1; i = ne[i]){
        int j = e[i];

        if(!color[j]){
            if (!dfs(j, 3 - c)) return false;
        }
        else if (color[j] == c) return false;
    }

    return true;
}

int main(){
    int n, m;
    scanf("%d%d", &n, &m);

    memset(h, -1, sizeof h);

    while (m -- ){
        int u, v;
        scanf("%d%d", &u, &v);
        add(u, v), add(v, u);
    }

    bool flag = true;
    for (int i = 1; i <= n; i ++ ){
        if (!color[i]){
            if (!dfs(i, 1)){
                flag = false;
                break;
            }
        }
    }
    if (flag){
            puts("Yes");
    }else{
            puts("No");
    }
    return 0;
}
```

7.二分图最大匹配(匈牙利)

```
#include <cstring>
#include <cstdio>
#include <algorithm>

using namespace std;

const int N = 510, M = 100010;

int n1, n2, m;
int h[N], e[M], ne[M], idx;
int match[N];
bool st[N];

void add(int a, int b){
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++ ;
}

bool find(int x){
    for (int i = h[x]; i != -1; i = ne[i]){
        int j = e[i];
        if (!st[j]){
            st[j] = true;
            if (match[j] == 0 || find(match[j])){
                match[j] = x;
                return true;
            }
        }
    }

    return false;
}
```

```
int main(){
    scanf("%d%d%d", &n1, &n2, &m);

    memset(h, -1, sizeof h);

    while (m -- ){
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }

    int res = 0;
    for (int i = 1; i <= n1; i ++ ){
        memset(st, false, sizeof st);
        if (find(i)) res ++ ;
    }

    printf("%d\n", res);

    return 0;
}
```

8.差分约束

如果一个不等式组由 n 个变量和 m 个约束条件组成，
形成 m 个形如

$$x_j - x_i <= k(i, j \in [1, n]且k为常数)$$

的不等式，则称其为**差分约束系统**。换句话说，差分约束系统就是求解一组变量的不等式组的算法。

$$\begin{cases} x1 - x2 \leq 3 \\ x2 - x3 \leq -2 \\ x1 - x3 \leq 1 \end{cases}$$

```
#include <cstring>
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;
struct edge {
        int v, w, next;
} e[10005];
int head[5005], tot[5005], dis[5005], vis[5005], cnt, n, m;
void addedge(int u, int v, int w) {
        e[++cnt].v = v;
```

```cpp
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt;
}
bool spfa(int s) {
    queue<int> q;
    memset(dis, 63, sizeof(dis));
    dis[s] = 0, vis[s] = 1;
    q.push(s);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = head[u]; i; i = e[i].next) {
            int v = e[i].v;
            if (dis[v] > dis[u] + e[i].w) {
                dis[v] = dis[u] + e[i].w;
                if (!vis[v]) {
                    vis[v] = 1, tot[v]++;
                    if (tot[v] == n + 1)return false; // 注意添加了一个超级源点
                    q.push(v);
                }
            }
        }
    }
    return true;
}
int main() {
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
        addedge(0, i, 0);
    for (int i = 1; i <= m; i++) {
        int v, u, w;
        cin >> v >> u >> w;
        addedge(u, v, w);
    }
    if (!spfa(0))cout << "NO" << endl;
    else
        for (int i = 1; i <= n; i++)
            cout << dis[i] << ' ';
    return 0;
}
```

扩展

1.如果有

$$x_i - x_j \geq c_k$$

则可以两边同时乘 −1，将不等号反转过来。

2.如果有

$$x_i - x_j = c_k$$

则可以把这个等式拆分为

$$x_i - x_j \leq c_k$$

和

$$x_i - x_j \geq ck$$

两个约束条件。