第三章 图&网络流目录

1、AStar_K 短路

```
const int maxn=100010;
int n,m,dis[maxn];
int tot,head1[maxn],head2[maxn];
bool flag[maxn];
struct edge{
	int to;
	int w;
	int next;
}e[maxn*2],e2[maxn*2];
struct node{
	int f;
	int g;
	int from;
	bool operator < (node a)const{
		if(a.f==f)
			return g>a.g;
		return f>a.f;
	}
};
void add_edge(int u,int v,int w){
	tot++;
	e[tot].to=v;
	e[tot].w=w;
	e[tot].next=head1[u];
	head1[u]=tot;
	e2[tot].to=u;
	e2[tot].w=w;
	e2[tot].next=head2[v];
	head2[v]=tot;
}
void prepare(){
	for(int i=1;i<=n;i++)
		dis[i]=maxn;tot=0;
	memset(head1,0,sizeof(head1));
	memset(head2,0,sizeof(head2));
}
void spfa(int t){
	for(int i=1;i<=n;i++)
		dis[i]=maxn;
	dis[t]=0;
	queue<int> q;
	q.push(t);
```

```
    flag[t]=1;
    while(!q.empty()){
        int v=q.front();
        q.pop();flag[v]=0;
        for(int i=head2[v];i;i=e2[i].next)
            if(dis[e2[i].to]>dis[v]+e2[i].w){
                dis[e2[i].to]=dis[v]+e2[i].w;
                if(!flag[e2[i].to]){
                    q.push(e2[i].to);
                    flag[e2[i].to]=1;
                }
            }
    }
}
int a_star(int s,int t,int k){
    if(s==t) k++;
    if(dis[s]==maxn) return -1;
    priority_queue<node> q;
    int cnt=0;
    node tmp,to;
    tmp.from=s;
    tmp.g=0;
    tmp.f=tmp.g+dis[tmp.from];
    q.push(tmp);
    while(!q.empty()){
        tmp=q.top();
        q.pop();
        if(tmp.from==t) cnt++;
        if(cnt==k) return tmp.g;
        for(int i=head1[tmp.from];i;i=e[i].next){
            to.from=e[i].to;
            to.g=tmp.g+e[i].w;
            to.f=to.g+dis[to.from];
            q.push(to);
        }
    }
    return -1;
}
int main(){ //  该模板能处理带环图
    int x,y,z,s,t,k;
    while(cin>>n>>m) {//  输入 n 个点  m 条边
        prepare();
        cin>>s>>t>>k; //  输入起点  终点  第 k 短路
        for(int i=1;i<=m;i++) {//  输入边
```

```
            cin>>x>>y>>z;
            add_edge(x,y,z);
        }
        spfa(t);
        int ans=a_star(s,t,k); // ans 为第 k 短路的长度
    }
    return 0;
}
```

2、DAG 深度优先队列标记

```
/*DAG(有向无环图)的深度优先搜索标记
 * INIT:edge[][]邻接矩阵；pre[], post[], tag 全置 0
 CALL:dfsTag(i, n);     pre/post:开始/结束时间*/
const int V = 1010;
int edge[V][V];
int pre[V];
int post[V];
int tag;
void dfsTag(int cur, int n){
    //vertex:0 ~ n - 1
    pre[cur] = ++tag;
    for (int i = 0; i < n; i++){
        if (edge[cur][i]){
            if (0 == pre[i]){
                std::cout << "Three Edge!" << '\n';
                dfsTag(i, n);
            }
            else{
                if (0 == post[i])       std::cout << "Back Edge!" << '\n';
                else if (pre[i] > pre[cur])     std::cout << "Down Edge!" << '\n';
                else       std::cout << "Cross Edge!" << '\n';
            }}
    }
    post[cur] = ++tag;
    return ;
}
```

3、无向图找桥

```
 /* INIT: edge[][]邻接矩阵；vis[],pre[],ans[],bridge 置 0；
   CALL: dfs(0, -1, 1, n);*/
const int V = 1010;
int bridge; //桥
```

```
int edge[V][V], ans[V] ,pre[V] ,vis[V];
void dfs(int cur, int father, int dep, int n){
	//vertex: 0 ~ n - 1
	if (bridge)	return ;
	vis[cur] = 1;
	pre[cur] = ans[cur] = dep;
	for (int i = 0; i < n; i++){
		if (edge[cur][i]){
			if (i != father && 1 == vis[i]){
				if (pre[i] < ans[cur])		ans[cur] = pre[i];	//back edge
			}
			if (0 == vis[i]) {	//tree edge
				dfs(i, cur, dep + 1, n);
				if (bridge)		return ;
				if (ans[i] < ans[cur])		ans[cur] = ans[i];
				if (ans[i] > pre[cur]){bridge = 1;		return ;}
			}
		}
	}
	vis[cur] = 2;
}
int main(){
	// 在这里输入 n
	/*
	 * 在这里输入图
	 */
	// dfs(0,-1,1,n); 调用函数
}
```

4、无向图连通度(割点)

```
const int V = 1010;
int edge[V][V];
int anc[V];
int pre[V];
int vis[V];
int deg[V];
void dfs(int cur, int father, int dep, int n){
	//vertex:0 ~ n - 1
	int cnt = 0;
	vis[cur] = 1;
	pre[cur] = anc[cur] = dep;
	for (int i = 0; i < n; i++){
		if (edge[cur][i]){
```

```
            if (i != father && 1 == vis[i])    if (pre[i] < anc[cur])    anc[cur] = pre[i];    //back edge
                if (0 == vis[i]){                    //tree edge
                    dfs(i, cur, dep + 1, n);
                    cnt++;    //分支个数
                    if (anc[i] < anc[cur])        anc[cur] = anc[i];
                    if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur]))
                        deg[cur]++; //link degree of a vertex
                }
            }
        }
        vis[cur] = 2;
}
int main(){
/* INIT: edge[][]邻接矩阵；vis[],pre[],anc[],deg[]置为 0；
 * CALL: dfs(0, -1, 1, n);
 * k = deg[0], deg[i] + 1(i = 1...n - 1)为删除该节点后得到的连通图个数
  注意: 0 作为根比较特殊*/
 }
```

5、曼哈顿最小生成树

```
const int MAXN = 100010;
const int INF = 0x3f3f3f3f;
struct Point{
    int x;
    int y;
    int id;
}poi[MAXN];
bool cmp(Point a, Point b){
    if (a.x != b.x)     return a.x < b.x;
    else       return a.y < b.y;
}
//树状数组，找 y - x 大于当前的，但是 y + x 最小的
struct BIT{
    int minVal;
    int pos;
    void init(){
        minVal = INF;
        pos = -1;
    }
}bit[MAXN];
//所有有效边
struct Edge{
    int u;
```

```
    int v;
    int d;
}edge[MAXN << 2];
bool cmpEdge(Edge a, Edge b){    return a.d < b.d;}
int tot;
int n;
int F[MAXN];
int find(int x){
    if (F[x] == -1)      return x;
    else    return F[x] = find(F[x]);
}
void addEdge(int u, int v, int d){
    edge[tot].u = u;
    edge[tot].v = v;
    edge[tot++].d = d;
    return ;
}
int lowbit(int x){      return x & (-x);}
//更新 bit
void update(int i, int val, int pos){
    while (i > 0){
        if (val < bit[i].minVal){
            bit[i].minVal = val;
            bit[i].pos = pos;
        }
        i -= lowbit(i);
    }
    return ;
}
//查询[i, m]的最小值位置
int ask(int i, int m){
    int minVal = INF;
    int pos = -1;
    while (i <= m){
        if (bit[i].minVal < minVal){
            minVal = bit[i].minVal;
            pos = bit[i].pos;
        }
        i += lowbit(i);
    }
    return pos;
}
int dist(Point a, Point b){    return abs(a.x - b.x) + abs(a.y - b.y);}
void ManhattanMinimumSpanningTree(int n, Point p[]){
```

```
        int a[MAXN], b[MAXN];
        tot = 0;
        for (int dir = 0; dir < 4; dir++){
            //变换 4 种坐标
            if (dir == 1 || dir == 3){
                for (int i = 0; i < n; i++)        std::swap(p[i].x, p[i].y);
            }
            else if (dir == 2){
                for (int i = 0; i < n; i++)        p[i].x = -p[i].x;
            }
            std::sort(p, p + n, cmp);
            for (int i = 0; i < n; i++)        a[i] = b[i] = p[i].y - p[i].x;
            std::sort(b, b + n);
            int m = (int)(std::unique(b, b + n) - b);
            for (int i = 1; i <= m; i++)        bit[i].init();
            for (int i = n - 1; i >= 0; i--){
                int pos = (int)(std::lower_bound(b, b + m, a[i]) - b + 1);
                int ans = ask(pos, m);
                if (ans != -1)    addEdge(p[i].id, p[ans].id, dist(p[i], p[ans]));
                update(pos, p[i].x + p[i].y, i);
            }
        }
        return ;
    }
    int solve(int k){
        ManhattanMinimumSpanningTree(n, poi);
        memset(F, -1, sizeof(F));
        std::sort(edge, edge + tot, cmpEdge);
        for (int i = 0; i < tot; i++){
            int u = edge[i].u;
            int v = edge[i].v;
            int tOne = find(u);
            int tTwo = find(v);
            if (tOne != tTwo){
                F[tOne] = tTwo;
                k--;
                if (k == 0)    return edge[i].d;
            }
        }
        return -1;
    }
    int main(int argc, const char * argv[]){
        //freopen("in.txt", "r", stdin);
        //freopen("out.txt", "w", stdout);
```

```
    int k;
    while ((std::cin >> n >> k) && n){
        for (int i = 0; i < n; i++){
            std::cin >> poi[i].x >> poi[i].y;
            poi[i].id = i;
        }
        std::cout << solve(n - k) << std::endl;
    }
    return 0;
}
```

6、最小生成树(prim)

```
#define inf 0x3f3f3f3f
typedef struct {
    int point;
    int value;
}node;
int N;
int sum;
vector<node>point[1000];
int hashtable[1000] = {0};
int num[1000];
void init() {
    int i;
    for (i = 0; i < N; i++)    num[i] = inf;
    num[0] = 0;
    sum = 0;
}
void prim() {
    int n = N,i,k,min,min_num;
    while (n--) {
    min=-1;
while (n--) {
        min_num = inf;
        for (i = 0; i < N; i++) {
            if (hashtable[i] == 0 && num[i] != inf) {
                if (num[i] < min_num) {
                    Min=i;
                    min_num = num[i];
                }
             }
         }
        sum += num[min];
```

```
            hashtable[min] = 1;
            if (min == -1)return;
            for (k = 0; k < point[min].size(); k++) {
                int v = point[min][k].point;
                int value = point[min][k].value;
                if (num[v] > value && hashtable[v] == 0)num[v] = value;
            }
        }
}
int main() {
        int M;
        scanf("%d", &N);
        scanf("%d", &M);
        int x, y, i, check, value;
        for (i = 0; i < M; i++) {
            scanf("%d%d%d", &x, &y, &value);
            node new_node = { y,value };
            point[x].push_back(new_node);
            new_node.point = x;
            point[y].push_back(new_node);
        }
        init();
        prim();
        printf("%d",sum);
        scanf("%d", &check);
}
```

7、次小生成树

```
int g[M][M],path[M][M];//path 求的是 i 到 j 最大的边权
int dist[M],pre[M],vis[M];
bool used[M][M];//是否在最小生成树中
int n,m,mst;
void init(){
        for(int i=0;i<=n;i++)
            for(int j=i+1;j<=n;j++)      g[i][j]=g[j][i]=inf;
}
int prime() {
        int mst=0;
        memset(path,0,sizeof(path));
        memset(vis,0,sizeof(vis));
        memset(used,0,sizeof(used));
        vis[1]=1;
        for(int i=1;i<=n;i++){
```

```
            dist[i]=g[1][i];
            pre[i]=1;
        }
        for(int i=1;i<n;i++)    {
            int u=-1;
            for(int j=1;j<=n;j++){
                if(!vis[j])        if(u==-1||dist[j]<dist[u])        u=j;
            }
            used[u][pre[u]]=used[pre[u]][u]=true;//加入 mst
            mst+=g[pre[u]][u];
            vis[u]=1;
            for(int j=1;j<=n;j++)
            {
                if(vis[j]&&j!=u)//从 u 到 j 这条路径上最大边的权值
                    path[j][u]=path[u][j]=max(path[j][pre[u]],dist[u]);
                if(!vis[j])
                    if(dist[j]>g[u][j]){//更新相邻节点的距离
                        dist[j]=g[u][j];
                        pre[j]=u;//记录他的前驱
                    }
            }
        }
        return mst;
}
int second_tree(){//求次小生成树
        int res=inf;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                if(i!=j&&!used[i][j])
           res=min(res,mst-path[i][j]+g[i][j]);//删除树上权值最大的路径并且加上这条路径其它边
        return res;
}
int main()    {
        int t;
        scanf("%d",&t);
        while(t--)    {
            scanf("%d%d",&n,&m);
            init();
            mst=prime();//最小生成树
            int second_mst=second_tree();//次小生成树
        }
}
```

8、欧拉路径

```
/*SGU 101 */
struct Edge{
	int to;
	int next;
	int index;
	int dir;
	bool flag;
} edge[220];
int head[10];	//前驱
int tot;
void init(){
	memset(head, -1, sizeof((head)));
	tot = 0;
}
void addEdge(int u, int v, int index){
	edge[tot].to = v;
	edge[tot].next = head[u];
	edge[tot].index = index;
	edge[tot].dir = 0;
	edge[tot].flag = false;
	head[u] = tot++;
	edge[tot].to = u;
	edge[tot].next = head[v];
	edge[tot].index = index;
	edge[tot].dir = 1;
	edge[tot].flag = false;
	head[v] = tot++;
	return ;
}
int du[10];
std::vector<int>ans;
void dfs(int u){
	for (int i = head[u]; i != -1; i = edge[i].next){
		if (!edge[i].flag){
			edge[i].flag = true;
			edge[i ^ 1].flag = true;
			dfs(edge[i].to);
			ans.push_back(i);	//容器尾部插入 i
		}
	}
	return ;
}
```

```
int main(){
	//freopen("in.txt", "r", stdin);
	//freopen("out.txt", "w", stdout);
	int n;
	while (std::cin >> n)	{
		init();
		int u, v;
		memset(du, 0, sizeof(du));
		for (int i = 1; i <= n; i++){
			std::cin >> u >> v;
			addEdge(u, v, i);
			du[u]++;
			du[v]++;
		}
		int s = -1;
		int cnt = 0;
		for (int i = 0; i <= 6; i++){
			if (du[i] & 1){
				cnt++;
				s = i;
			}
			if (du[i] > 0 && s == -1)	s = i;
		}
		if (cnt != 0 && cnt != 2){
			std::cout << "No solution" << '\n';
			continue;
		}
		ans.clear();
		dfs(s);
		if (ans.size() != n){
			std::cout << "No solution" << '\n';
			continue;
		}
		for (int i = 0; i < ans.size(); i++){
			printf("%d ", edge[ans[i]].index);
			if (edge[ans[i]].dir == 0)	std::cout << "-" << '\n';
			else	std::cout << "+" << '\n';
		}
	}
	return 0;
}
```

9、迪杰斯特拉优化模板

```
typedef struct {
int point;//能够到达的点
int value;//第一尺度
int cost; //第二尺度
}node;
int N;
int num[1001];//第一尺度的最小值储存单位
int cost[1001];//第二尺度的最小值储存单位
int hashtable[2000] = { 0 };//哈希表，判断点是否访问过
vector<node>point[2000];//邻接表
void init(int start) {//初始化
	int i;
	for (i = 0; i < N; i++) {
		num[i] = Max;
		cost[i] = Max;
	}
	num[start] = 0;
	cost[start] = 0;
}
void djistra(int start) {
	init(start);
	int i;
	int min = 0;
	int min_num;
	int check;
	while (1) {
		min_num = Max;
		check = 0;
		for (i = 0; i < N; i++) {//找出当前离起点最近的且未访问过的节点
			if (hashtable[i] == 0 && num[i] != Max) {
			check = 1;
			if (num[i] < min_num) {
				min = i;
				min_num = num[i];
				}
			}
		}
		if (check == 0)
		return;//如果没有就说明优化距离结束
		hashtable[min] = 1;
		for (i = 0; i < point[min].size(); i++) {
			if (hashtable[point[min][i].point] == 0) {
```

```
                    if (num[point[min][i].point] > point[min][i].value + num[min]) {
                    //以第一尺度为标准，先计算出第一尺度的最小值下的第二尺度的值
                    num[point[min][i].point] = point[min][i].value + num[min];
                    cost[point[min][i].point] = point[min][i].cost + cost[min];
                    }
                    else if (num[point[min][i].point] == point[min][i].value + num[min]) {
                    //以计算出的第二尺度值为标准，计算出第二尺度的最小值
                  if (cost[point[min][i].point] > point[min][i].cost + cost[min])
                    cost[point[min][i].point] = point[min][i].cost + cost[min];
                    }
                }
            }
        }
    }
int main() {        //点标号 0 开头
    int M, start, end;
    int x, y, value,cost_value;
    while (scanf("%d%d", &N, &M) && (N != 0 || M != 0) ) {
        while (M--) {
        scanf("%d%d%d%d", &x, &y, &value, &cost_value);
        node new_node = { y,value,cost_value };
        point[x].push_back(new_node);//无向边
        new_node.point = x;
        point[y].push_back(new_node);
        }
    scanf("%d%d", &start, &end);
    djistra(start);
    printf("%d %d\n", num[end], cost[end]);
    }
}
```

10、最小树形图

```
const int INF = 0x3f3f3f3f;
const int MAXN = 1010;
const int MAXM = 1000010;
struct Edge{    int u, v, cost;};
Edge edge[MAXM];
int pre[MAXN], id[MAXN], visit[MAXN], in[MAXN];
int zhuliu(int root, int n, int m){
    int res = 0, v;
    while (true){
        memset(in, 0x3f, sizeof(in));
        for (int i = 0; i < m; i++){
```

```
            if (edge[i].u != edge[i].v && edge[i].cost < in[edge[i].v]){
                pre[edge[i].v] = edge[i].u;
                    in[edge[i].v] = edge[i].cost;
                }
        }
        for (int i = 0; i < n; i++){
            if (i != root && in[i] == INF)      return -1;    //   不存在最小树形图
    }
    int tn = 0;
        memset(id, -1, sizeof(id));
        memset(visit, -1, sizeof(visit));
        in[root] = 0;
        for (int i = 0; i < n; i++){
            res += in[i];
            v = i;
                while (visit[v] != i && id[v] == -1 && v != root){
                    visit[v] = i;
                    v = pre[v];
                    }
                    if (v != root && id[v] == -1){
                        for (int u = pre[v]; u != v ; u = pre[u])       id[u] = tn;
                            id[v] = tn++;
                        }
                    }
                if (tn == 0)       break;    //   没有有向环
            for (int i = 0; i < n; i++)     if (id[i] == -1)       id[i] = tn++;
            for (int i = 0; i < m; i++){
                v = edge[i].v;
                edge[i].u = id[edge[i].u];
                edge[i].v = id[edge[i].v];
                    if (edge[i].u != edge[i].v)       edge[i].cost -= in[v];
                }
                n = tn;
                root = id[root];
            }
            return res;
        }
int main(){
  /*最小树形图
    *   int 型
    *   复杂度 O(NM)
    *   点从 0 开始*/}
```

11、生成树计数

```
/*取模*/
//  求生成树计数部分代码,计数对 10007 取模
const int MOD = 10007;
int INV[MOD]; // 逆元打表数组
int g[MAXN][MAXN];
//  求 ax = 1(mod m)的 x 值,就是逆元(0<a<m)
long long inv(long long a, long long m){
    if (a == 1)    return 1;
    return inv(m % a, m) * (m - m / a) % m;
}
struct Matrix{
    int mat[330][330];
    void init()  {  memset(mat, 0, sizeof(mat));}
    int det(int n){  // 求行列式的值模上 MOD,需要使用逆元
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++)    mat[i][j] = (mat[i][j] % MOD + MOD) % MOD;
        }
        int res = 1;
        for (int i = 0; i < n; i++){
            for (int j = i; j < n; j++){
                if (mat[j][i] != 0){
                    for (int k = i; k < n; k++)      swap(mat[i][k], mat[j][k]);
                    if (i != j)    res = (-res + MOD) % MOD;
                    break;
                }
            }
            if (mat[i][i] == 0){
                res = -1;    // 不存在(也就是行列式值为 0)
                break;
            }
            for (int j = i + 1; j < n; j++){
                int mut = (mat[j][i]*INV[mat[i][i]])%MOD;//打表逆元
                int mut = (mat[j][i] * inv(mat[i][i], MOD)) % MOD;
                for (int k = i; k < n; k++){
                    mat[j][k] = (mat[j][k] - (mat[i][k] * mut) % MOD + MOD) % MOD;
                }
            }
            res = (res * mat[i][i]) % MOD;
        }
        return res;
    }
};
```

```
int main()
{
	Matrix ret;
	ret.init();
	int n;
	scanf("%d",&n);
	for(int i = 0;i < n;i++){
		int u,v;
		scanf("%d%d",&u,&v); //  输入数据
		u--;v--;
		g[u][v] = g[v][u] = 1;
	}
	for (int i = 0; i < n; i++){
		for (int j = 0; j < n; j++){
			if (i != j && g[i][j]){
				ret.mat[i][j] = -1;
				ret.mat[i][i]++;
			}
		}
	}
	printf("%d\n", ret.det(n - 1));
	return 0;
}

/*不取模*/
const double eps = 1e-8;
const int MAXN = 110;
int sgn(double x){
	if (fabs(x) < eps)	return 0;
	if (x < 0)	return -1;
	else	return 1;
}
double b[MAXN][MAXN];
double det(double a[][MAXN], int n){
	int i, j, k, sign = 0;
	double ret = 1;
	for (i = 0; i < n; i++){
		for (j = 0; j < n; j++)	b[i][j] = a[i][j];
	}
	for (i = 0; i < n; i++){
		if (sgn(b[i][i]) == 0){
			for (j = i + 1; j < n; j++){
				if (sgn(b[j][i]) != 0)	break;
			}
```

```
            if (j == n)      return 0;
            for (k = i; k < n; k++)        swap(b[i][k], b[j][k]);
            sign++;
        }
        ret *= b[i][i];
        for (k = i + 1; k < n; k++)        b[i][k] /= b[i][i];
        for (j = i+1; j < n; j++){
            for (k = i+1; k < n; k++)        b[j][k] -= b[j][i] * b[i][k];
        }
    }
    if (sign & 1)      ret = -ret;
    return ret;
}
double a[MAXN][MAXN];
int g[MAXN][MAXN];
int main(){
    int T,n,m,u,v;
    scanf("%d", &T);
    while (T--){
        scanf("%d%d", &n, &m);
        memset(g, 0, sizeof(g));
        while (m--){
            scanf("%d%d", &u, &v); //   输入数据
            u--;
            v--;
            g[u][v] = g[v][u] = 1;
        }
        memset(a, 0, sizeof(a));
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; j++){
                if (i != j && g[i][j]){
                    a[i][i]++;
                    a[i][j] = -1;
                }
            }
        }
        double ans = det(a, n - 1);
        printf("%.0lf\n", ans);
    }
    return 0;
}
```

12、一般图匹配带花树

```
const int maxn = 300;
int N;
bool G[maxn][maxn];
int match[maxn];
bool InQueue[maxn], InPath[maxn], InBlossom[maxn];
int head, tail;
int Queue[maxn];
int Start;
int finish;
int NewBase;
int father[maxn], Base[maxn];
int Count;
void CreateGraph(){
	int u, v;
	memset(G, 0, sizeof(G));
	scanf("%d", &N);
	while (scanf("%d%d",&u,&v) != EOF)	G[u][v] = G[v][u] = true;
}
void Push(int u){
	Queue[tail++] = u;
	InQueue[u] = true;
}
int Pop(){
	int res = Queue[head++];
	return res;
}
int FindCommonAncestor (int u, int v){
	memset(InPath, 0, sizeof(InPath));
	while (true){
		u = Base[u];
		InPath[u] = 1;
		if (u == Start)	break;
		u = father[match[u]];
	}
	while (true){
		v = Base[v];
		if (InPath[v])	break;
		v = father[match[v]];
	}
	return v;
}
void ResetTrace(int u){
```

```
    int v;
    while (Base[u] != NewBase){
        v = match[u];
        InBlossom[Base[u]] = InBlossom[Base[v]] = 1;
        u = father[v];
        if (Base[u] != NewBase)       father[u] = v;
    }
}
void BlossomContract(int u, int v){
    NewBase = FindCommonAncestor(u, v);
    memset(InBlossom, 0, sizeof(InBlossom));
    ResetTrace(u);
    ResetTrace(v);
    if (Base[u] != NewBase)       father[u]=v;
    if (Base[v] != NewBase)       father[v]=u;
    for (int tu=1; tu <= N; tu++){
        if (InBlossom[Base[tu]]){
            Base[tu] = NewBase;
            if (!InQueue[tu])     Push(tu);
        }
    }
}
void FindAugmentingPath(){
    memset(InQueue, 0, sizeof(InQueue));
    memset(father, 0, sizeof(father));
    for (int i = 1; i <= N; i++)    Base[i] = i;
    head = tail = 1;
    Push(Start);
    finish = 0;
    while (head < tail){
        int u = Pop();
        for (int v = 1; v <= N; v++){
            if (G[u][v] && (Base[u] != Base[v]) && match[u] != v){
                if ((v == Start) || ((match[v] > 0) && father[match[v]] > 0))    BlossomContract(u, v);
                else if (father[v] == 0){
                    father[v] = u;
                    if (match[v] > 0)     Push(match[v]);
                    else{
                        finish = v;
                        return ;
                    }}}}}
}
void AugmentPath(){
    int u, v, w;
```

```
    u = finish;
    while (u > 0){
        v = father[u];
        w = match[v];
        match[v] = u;
        match[u] = v;
        u = w;
    }
}
void Edmonds(){
    memset(match, 0, sizeof(match));
    for (int u = 1; u <= N; u++){
        if (match[u] == 0){
            Start = u;
            FindAugmentingPath();
            if (finish > 0)     AugmentPath();
        }
    }
}
void PrintMatch(){
    Count = 0;
    for (int u = 1; u <= N; u++){
        if (match[u] > 0)   Count++;
    }
    printf("%d\n", Count);
    for (int u = 1; u <= N; u++){
        if (u < match[u])   printf("%d %d\n", u, match[u]);
    }
}
int main(){
    CreateGraph();
    Edmonds();          //  进行匹配
    PrintMatch();   //  输出匹配
    return 0;
}
```

13、最大团

```
const int V = 10010;
int g[V][V];
int dp[V];
int stk[V][V];
int mx;
int dfs(int n, int ns, int dep){
```

```
    if (0 == ns){
        if (dep > mx)    mx = dep;
        return 1;
    }
    int i, j, k, p, cnt;
    for (i = 0; i < ns; i++){
        k = stk[dep][i];
        cnt = 0;
        if (dep + n - k <= mx)    return 0;
        if (dep + dp[k] <= mx)    return 0;
        for (j = i + 1; j < ns; j++){
            p = stk[dep][j];
            if (g[k][p])    stk[dep + 1][cnt++] = p;
        }
        dfs(n, cnt, dep + 1);
    }
    return 1;
}
int clique(int n){
    int i, j, ns;
    for (mx = 0, i = n - 1; i >= 0; i--){    //  vertex: 0 ~ n-1
        for (ns = 0, j = i + 1; j < n; j++){
            if (g[i][j])    stk[1][ns++] = j;
        }
        dfs(n, ns, 1);
        dp[i] = mx;
    }
    return mx;
}
int main(){
/*INIT: g[][]邻接矩阵
 *  CALL: res = clique(n);*/
    /*在这里输入 n
     * 在这里输入邻接矩阵 g[][] */
}
```

14、拓扑排序

```
/* 拓扑排序
 * INIT:edge[][]置为图的邻接矩阵;cnt[0...i...n-1]:顶点 i 的入度.*/
const int MAXV = 1010;
int edge[MAXV][MAXV];
int cnt[MAXV];
void TopoOrder(int n){
```

```
    int i,top = -1;
    for(i = 0;i < n;i++){
        if (cnt[i] == 0){
            cnt[i] = top;
            top = i;
          }
  }
for(i = 0;i < n;i++){
    if (top == -1){
        printf("存在回路\n");
        return;
    }
    else{
        int j = top;
        top = cnt[top];
        printf("%d",j);
        for(int k = 0;k < n;k++){
            if (edge[j][k] && (--cnt[k]) == 0){
                cnt[k] = top;
                top = k;
            }
        }
    }
}
}
```

15、2-SAT

```
/* 2-sat  问题
* N 个集团,每个集团 2 个人,现在要想选出尽量多的人,
*  且每个集团只能选出一个人。如果两人有矛盾,他们不能同时被选中
*  问最多能选出多少人*/
const int MAXN = 3010;
int n, m;
int g[3010][3010], ct[3010], f[3010];
int x[3010], y[3010];
int prev1[MAXN], low[MAXN], stk[MAXN], sc[MAXN];
int cnt[MAXN];
int cnt0, ptr, cnt1;
void dfs(int w) {
      int min(0);
      prev1[w] = cnt0++;
      low[w] = prev1[w];
      min = low[w];
```

```
    stk[ptr++] = w;
    for (int i = 0; i < ct[w]; ++i) {
        int t = g[w][i];
        if (prev1[t] == -1) { dfs(t); }
        if (low[t] < min) { min = low[t]; }
    }
    if (min < low[w]) {
        low[w] = min;
        return;
    }
    do {
        int v = stk[--ptr];
        sc[v] = cnt1;
        low[v] = MAXN;
        } while (stk[ptr] != w);
    ++cnt1;
    return;
}
void Tarjan(int N) { // 传入 N 为点数,结果保存在 sc 数组中,同一标号的点在同一个强连通分量内, // 强连通分量数为 cnt1
    cnt0 = cnt1 = ptr = 0;
    int i;
    for (i = 0; i < N; ++i) { prev1[i] = low[i] = -1; }
    for (i = 0; i < N; ++i) { if (prev1[i] == -1) { dfs(i); }}
    return;
}
int solve() {
    Tarjan(n);
    for (int i = 0; i < n; i++) { if (sc[i] == sc[f[i]]) { return 0; }}
    return 1;
}
int check(int Mid) {
    for (int i = 0; i < n; i++) { ct[i] = 0; }
    for (int i = 0; i < Mid; i++) {
        g[f[x[i]]][ct[f[x[i]]]++] = y[i];
        g[f[y[i]]][ct[f[y[i]]]++] = x[i];
    }
    return solve();
}
int main() {
    while (scanf("%d%d", &n, &m) != EOF && n + m) {
        for (int i = 0; i < n; i++) {
            int p, q;
            scanf("%d%d", &p, &q);
```

```
            f[p] = q, f[q] = p;
        }
        for (int i = 0; i < m; i++) { scanf("%d%d", &x[i], &y[i]); }
        n *= 2;
        int Min = 0, Max = m + 1;
        while (Min + 1 < Max) {
            int Mid = (Min + Max) / 2;
            if (check(Mid)) { Min = Mid; } else { Max = Mid; }
        }
        printf("%d\n", Min);
    }
    return 0;
}
```

16、DAG_DFS

```
/* DAG(有向无环图)的深度优先搜索标记
* INIT:edge[][]邻接矩阵；pre[], post[], tag 全置 0
* CALL:dfsTag(i, n); pre/post:开始/结束时间*/
const int V = 1010;
int edge[V][V], pre[V], post[V], tag;
void dfsTag(int cur, int n){
    //vertex:0 ~ n - 1
    pre[cur] = ++tag;
    for (int i = 0; i < n; i++){
        if (edge[cur][i]){
            if (0 == pre[i]){
                std::cout << "Three Edge!" << '\n';
                dfsTag(i, n);
            }
            else{
                if (0 == post[i])  std::cout << "Back Edge!" << '\n';
                else if (pre[i] > pre[cur])   std::cout << "Down Edge!" << '\n';
                      else   std::cout << "Cross Edge!" << '\n';
            }
        }
    }
    post[cur] = ++tag;
    return ;
}
```

17、Floyd 求最小环

```
const int INF = 0x3f3f3f3f;
const int MAXN = 110;
int n, m; //n:节点个数, m:边的个数
int g[MAXN][MAXN];//无向图
int dist[MAXN][MAXN];//最短路径
int r[MAXN][MAXN]; //r[i][j]:i 到 j 的最短路径的第一步
int out[MAXN], ct;//记录最小环
int solve(int i, int j, int k) {
	//记录最小环
	ct = 0;
	while (j != i) {
		out[ct++] = j;
		j = r[i][j];
	}
	out[ct++] = i;
	out[ct++] = k;
	return 0;
}
int main() {
	while (scanf("%d%d", &n, &m) != EOF) {
		int i, j, k;
		for (i = 0; i < n; i++) {
			for (j = 0; j < n; j++) {
				g[i][j] = INF;
				r[i][j] = i;
			}
		}
		for (i = 0; i < m; i++) {
			int x, y, l;
			scanf("%d%d%d", &x, &y, &l);
			--x;
			--y;
			if (l < g[x][y]) { g[x][y] = g[y][x] = l; }
		}
		memmove(dist, g, sizeof(dist));
		int Min = INF;
		//最小环
		for (k = 0; k < n; k++) {
		// Floyd
			for (i = 0; i < k; i++){ //一个环中的最大结点为 k(编号最大)
	  if (g[k][i] < INF) {
				for (j = i + 1; j < k; j++) {
```

```
                    if (dist[i][j] < INF && g[k][j] < INF && Min > dist[i][j] + g[k][i] + g[k][j]) {
                        Min = dist[i][j] + g[k][i] + g[k][j];
                        solve(i, j,k);            // 记录最小环
                    }
                }
            }
        }
        for (i = 0; i < n; i++) {
            if (dist[i][k] < INF) {
                for (j = 0; j < n; j++) {
                    if (dist[k][j] < INF && dist[i][j] > dist[i][k] + dist[k][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                        r[i][j] = r[k][j];
                    }
                }
            }
        }
        if (Min < INF) {
            for (ct--; ct >= 0; ct--) {
                printf("%d", out[ct] + 1);
                if (ct) { printf(" "); }
            }
        } else { printf("No solution."); }
        printf("\n");
    }
    return 0;
}
```

18、树的重心

```
const int INF = 0x3f3f3f3f;
const int MAXN = 100000 + 10;
/* 树的重心
* 初始化 vis[] son[] 为 0
* 初始化 sz 为 INF*/
int zx, sz, n;
int son[MAXN], vis[MAXN];
vector<pll> edge[MAXN];
void init() {
    for (int i = 1; i <= n; i++) { edge[i].clear(); }
    memset(vis, 0, sizeof(vis));
    sz = INF;
    zx = -1;
```

```
}
void dfs(int r) {
    vis[r] = 1;
    son[r] = 0;
    int tmp = 0;
    for (int i = 0; i < edge[r].size(); i++) {
        int v = edge[r][i].second;
        if (!vis[v]) {
            dfs(v);
            son[r] += son[v] + 1;
            tmp = max(tmp, son[v] + 1);
        }
    }
    tmp = max(tmp, n - son[r] - 1);
    if (tmp < sz) {
        zx = r;
        sz = tmp;
    }
}
```

19、无向图最小割

```
/* INIT: 初始化邻接矩阵 g[][]
* CALL: res = mincut(n);
* 注: Stoer-Wagner Minimum Cut;
* 找边的最小集合，若其被删去则图变得不连通（我们把这种形式称为最小割问题）*/
#define typec int //type of res
const typec inf = 0x3f3f3f3f; //max of res
const typec maxw = 1000; // maximum edge weight
const typec V = 10010;
typec g[V][V], w[V];
int a[V], v[V], na[V];
typec minCut(int n) {
    int i, j, pv, zj;
    typec best = maxw * n * n;
    for (i = 0; i < n; i++){v[i] = i;       // vertex:0~n - 1}
    while (n > 1) {
        for (a[v[0]] = 1, i = 1; i < n; i++) {
            a[v[i]] = 0;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }
    for (pv = v[0], i = 1; i < n; i++) {
        for (zj = -1, j = 1; j < n; j++) {
```

```
            if (!a[v[j]] && (zj < 0 || w[j] > w[zj])) { zj = j; }
        }
        a[v[zj]] = 1;
        if (i == n - 1) {
            if (best > w[zj]) { best = w[zj]; }
            for (i = 0; i < n; i++) { g[v[i]][pv] = g[pv][v[i]] += g[v[zj]][v[i]]; }
            v[zj] = v[--n];
            break;
        }
        pv = v[zj];
        for (j = 1; j < n; j++) { if (!a[v[j]]) { w[j] += g[v[zj]][v[j]]; }}
        }
    }
    return best;
}
```

20、最大流

```
/* Dinic  最大流  O(V^2 * E)
* INIT: ne=2; head[]置为 0; addedge()加入所有弧;
* CALL: flow(n, s, t);*/
#define typec int // type of cost
const typec inf = 0x3f3f3f3f; // max of cost
const typec E = 10010;
const typec N = 1010;
struct edge{
      int x, y, nxt;
      typec c;
} bf[E];
int ne, head[N], cur[N], ps[N], dep[N];
void addedge(int x, int y, typec c)
{ // add an arc(x->y, c); vertex:0~n-1;
      bf[ne].x = x;
      bf[ne].y = y;
      bf[ne].c = c;
      bf[ne].nxt = head[x];
      head[x] = ne++;
      bf[ne].x = y;
      bf[ne].y = x;
      bf[ne].c = 0;
      bf[ne].nxt = head[y];
      head[y] = ne++;
      return ;
}
```

```
typec flow(int n, int s, int t){
    typec tr, res = 0;
    int i, j, k, f, r, top;
    while (1){
        memset(dep, -1, n * sizeof(int));
        for (f = dep[ps[0] = s] = 0, r = 1; f != r;){
            for (i = ps[f++], j = head[i]; j; j = bf[j].nxt){
                if (bf[j].c && -1 == dep[k = bf[j].y]){
                    dep[k] = dep[i] + 1;
                    ps[r++] = k;
                    if (k == t){ f = r; break; }
                }
            }
        }
        if (-1 == dep[t]) break;
        memcpy(cur, head, n * sizeof(int));
        for (i = s, top = 0; ;){
            if (i == t){
            for (k = 0, tr = inf; k < top; ++k)    if (bf[ps[k]].c < tr)      tr = bf[ps[f = k]].c;
            for (k = 0; k < top; ++k)      bf[ps[k]].c -= tr, bf[ps[k]^1].c += tr;
            res += tr;
            i = bf[ps[top = f]].x;
            }
            for (j = cur[i]; cur[i]; j = cur[i] = bf[cur[i]].nxt){
                if (bf[j].c && dep[i] + 1 == dep[bf[j].y])    break;
            }
            if (cur[i]){
                ps[top++] = cur[i];
                i = bf[cur[i]].y;
            }
            else{
                if (0 == top)break;
                dep[i] = -1;
                i = bf[ps[--top]].x;
            }
        }
    }
    return res;
}
```

21、最小费用流

```
/*  最小费用流  O(V * E * f)
* INIT: network g; g.build(v, e);
* CALL: g.mincost(s, t); flow=g.flow; cost=g.cost;
*  注意: SPFA 增广, 实际复杂度远远小于 O(V * E);*/
#define typef int           // type of flow
#define typec int         // type of dis
const typef inff = 0x3f3f3f3f;       // max of flow
const typec infc = 0x3f3f3f3f;     // max of dis
const int E = 10010;
const int N = 1010;
struct network{
    int nv, ne, pnt[E], nxt[E];
    int vis[N], que[N], head[N], pv[N], pe[N];
    typef flow, cap[E];
    typec cost, dis[E], d[N];
    void addedge(int u, int v, typef c, typec w){
        pnt[ne] = v;
        cap[ne] = c;
        dis[ne] = +w;
        nxt[ne] = head[u];
        head[u] = (ne++);
        pnt[ne] = u;
        cap[ne] = 0;
        dis[ne] = -w;
        nxt[ne] = head[v];
        head[v] = (ne++);
    }
    int mincost(int src, int sink){
        int i, k, f, r;
        typef mxf;
        for (flow = 0, cost = 0; ;){
            memset(pv, -1, sizeof(pv));
            memset(vis, 0, sizeof(vis));
            for (i = 0; i < nv; ++i)    d[i] = infc;
            d[src] = 0;
            pv[src] = src;
            vis[src] = 1;
            for (f = 0, r = 1, que[0] = src; r != f;){
                i = que[f++];
                vis[i] = 0;
                if (N == f)    f = 0;
```

```
                for (k = head[i]; k != -1; k = nxt[k]){
                    if(cap[k] && dis[k]+d[i] < d[pnt[k]]){
                        d[pnt[k]] = dis[k] + d[i];
                        if (0 == vis[pnt[k]]){
                            vis[pnt[k]] = 1;
                            que[r++] = pnt[k];
                            if (N == r)    r=0;
                        }
                        pv[pnt[k]] = i;
                        pe[pnt[k]] = k;
                    }
                }
            }
            if (-1 == pv[sink]) break;
            for (k = sink, mxf = inff; k != src; k = pv[k])
                if (cap[pe[k]] < mxf)    mxf = cap[pe[k]];
            flow += mxf;
            cost += d[sink] * mxf;
            for (k = sink; k != src; k = pv[k]){
                cap[pe[k]] -= mxf;
                cap[pe[k] ^ 1] += mxf;
            }
        }
        return cost;
    }
    void build(int v, int e){
        nv = v;
        ne = 0;
        memset(head, -1, sizeof(head));
        int x, y;
        typef f;
        typec w;
        for (int i = 0; i < e; ++i){
            cin >> x >> y >> f >> w; // vertex: 0 ~ n-1
            addedge(x, y, f, w); // add arc (u->v, f, w)
        }
    }
} g;

/*  最小费用流  O(V^2 * f)
* INIT: network g; g.build(nv, ne);
* CALL: g.mincost(s, t); flow=g.flow; cost=g.cost;
*  注意: 网络中弧的 cost 需为非负. 若存在负权, 进行如下转化:
*  首先如果原图有负环, 则不存在最小费用流. 那么可以用 Johnson
```

```
* 重标号技术把所有边变成正权，以后每次增广后进行维护，算法如下：
* 1、用 bellman-ford 求 s 到各点的距离 phi[];
* 2、以后每求一次最短路，设 s 到各点的最短距离为 dis[];
* for i = 1 to v do
* phi[v] += dis[v];
* 下面的代码已经做了第二步，如果原图有负权，添加第一步即可。*/
const typef inff = 0x3f3f3f3f; // max of flow
const typec infc = 0x3f3f3f3f; // max of cost
const int E = 10010;
const int N = 1010;
struct edge{
    int u, v;
    typef cuv, cvu, flow;
    typec cost;
    edge (int x, int y, typef cu, typef cv, typec cc) :u(x), v(y), cuv(cu), cvu(cv), flow(0),
cost(cc){}
    int other(int p){return p == u ? v : u;}
    typef cap(int p){return p == u ? cuv-flow : cvu+flow;}
    typec ecost(int p){
        if (flow == 0) return cost;
        else if (flow > 0) return p == u ? cost : -cost;
            else return p == u ? -cost : cost;
        }
    void addFlow(int p, typef f){ flow += (p == u ? f : -f);}
};
struct network{
    vector<edge> eg;
    vector<edge*> net[N];
    edge *prev[N];
    int v, s, t, pre[N], vis[N];
    typef flow;
    typec cost, dis[N], phi[N];
    bool dijkstra();
    void build(int nv, int ne);
    typec mincost(int, int);
};
bool network::dijkstra(){
    // 使用 O(E * logV)的 Dij 可降低整体复杂度至 O(E * logV * f)
    int i, j, p, u = 0;
    typec md, cw;
    for (i = 0; i < v; i++)   dis[i] = infc;
    dis[s] = 0;
    prev[s] = 0;
    pre[s] = -1;
```

```
        memset(vis, 0, v * sizeof(int));
        for (i = 1; i < v; i++){
            for (md = infc, j = 0; j < v; j++){
                if (!vis[j] && md > dis[j]){
                    md = dis[j];
                    u = j;
                }
            }
            if (md == infc)    break;
            for (vis[u] = 1, j = (int)net[u].size() - 1; j >= 0; j--){
                edge *ce = net[u][j];
                if (ce->cap(u) > 0){
                    p = ce->other(u);
                    cw = ce->ecost(u) + phi[u] - phi[p];
                    // !! assert(cw >= 0);
                    if (dis[p] > dis[u] + cw){
                        dis[p] = dis[u] + cw;
                        prev[p] = ce;
                        pre[p] = u;
                    }
                }
            }
        }
        return infc != dis[t];
}
typec network::mincost(int ss, int tt){
        s = ss;
        t = tt;
        int i, c;
        typef ex;
        flow = cost = 0;
        memset(phi, 0, sizeof(phi));
        // !! 若原图含有负消费的边，在此处运行 Bellmanford
        //  将 phi[i](0 <= i <= n - 1)置为 mindist(s, i).
        for (i = 0; i < v; i++)    net[i].clear();
        for (i = (int)eg.size() - 1; i >= 0; i--){
            net[eg[i].u].push_back(&eg[i]);
            net[eg[i].v].push_back(&eg[i]);
        }
        while (dijkstra()){
            for (ex = inff, c = t; c != s; c = pre[c]){
                if (ex > prev[c]->cap(pre[c])) ex = prev[c]->cap(pre[c]);
            }
            for (c = t; c != s; c = pre[c])    prev[c]->addFlow(pre[c], ex);
```

```
            flow += ex;
            cost += ex * (dis[t] + phi[t]);
            for (i = 0; i < v; i++)    phi[i] += dis[i];
        }
        return cost;
}
void network::build(int nv, int ne){
        eg.clear();
        v = nv;
        int x, y;
        typef f;
        typec c;
        for (int i = 0; i < ne; ++i){
            cin >> x >> y >> f >> c;
            eg.push_back(edge(x, y, f, 0, c));
        }
        return ;
}
```