第三章 图目录

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

```
/*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];
int ans[V];
int pre[V];
int 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*/
 }
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
			}
```

```
    }
```

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.*/
#include <bits/stdc++.h>
using namespace std;
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;
                }
            }
        }
    }
}
```