最小生成树算法

Kruskal **加边法**（并查集思想）　复杂度 $O(m\log m)$

（1）边权从小到大排序.

（2）把每个点看作是独立的集合.

（3）按边权从小到大选边，只要边两点不在同一集合就合并.

（4）重复（3），直到选出 $n-1$ 条边.

```cpp
struct edge {
    int u,v,w;
}e[];

int getf(int x) {
    return x==f[x]?x:f[x]=getf(f[x]);
}

void merge(int a,int b) {
    int tx=getf(a),ty=getf(b);
    if (tx!=ty) {
        f[tx]=ty;
    }
}

bool cmp(edge x,edge y) {
    return x.w<y.w;
}

main() {
    for (int i=1;i<=m;i++) {
        cin>>e[i].u>>e[i].v>>e[i].w;
        f[i]=i;
    }
    sort(e+1,e+m+1,cmp);
    for (int i=1;i<=m;i++) {
        if (getf(e[i].u)!=getf(e[i].v)) {
            merge(e[i].u,e[i].v);
            ans+=e[i].w;
            cnt++;
        }
        if (cnt==n-1) {
            break;
        }
    }
}
```

Prim **加点法**

（1）从图上任意一点开始.

　　`book[]` 标记是否加入生成树，`dis[]` 标记 `i` 点离生成树最小的距离.

（2）找距离生成树最近的点（不在生成树上）.

（3）标记这个点在生成树中.

（4）用这个点更新 `dis[]`.

（5）重复（3）（4），直到所有点都被标记.

邻接矩阵：

```
main() {
    for (int i=1;i<=n;i++) {
        dis[i]=e[1][i];
    }
    while (cnt<n) {
        mn=0x7f; // infinity
        for (int i=1;i<=n;i++) {
            if (book[i]==0&&dis[i]<mn) {
                mn=dis[i];
                j=i;
            }
        }
        book[j]=1;
        for (int i=1;i<=n;i++) {
            if (book[i]==0&&dis[i]<e[j][i]) {
                dis[i]=e[j][i];
            }
        }
    }
}
```

最短路径算法

Floyd **算法**　复杂度 $O(n^3)$

```
void floyd() {
    for (int i=1;i<=n;i++) {
        for (int j=1;j<=n;j++) {
            for (int k=1;k<=n;k++) {
                a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
            }
        }
    }
}
```

Dijkstra **算法**　复杂度 $O(n^2)$

```
void dijkstra(int s) {
    int mn,n;
    for (int i=1;i<=n;i++) {
        vis[i]=0;
        dis[i]=way[s][i];
    }
    vis[s]=1;
    for (int i=1;i<=n;i++) {
        int u;
        mn=0x3f3f3f3f; // infinity
        u=-1;
        for (int j=1;j<=n;j++) {
            if (vis[j]==0&&dis[j]<mn) {
                u=j;
                mn=dis[j];
            }
        }
        if (u==-1) {
            break;
        }
        vis[u]=1;
        for (int j=1;j<=n;j++) {
            if (vis[j]==0) {
                if (dis[u]+way[u][j]<dis[j]) {
                    dis[j]=dis[u]+way[u][j];
                }
            }
        }
    }
}
```

Bellman-Ford **算法**　复杂度 $O(mn)$

对每一条边进行 $n-1$ 轮标记，会重复进队且有意义.

```
void bf() {
    for (int k=1;k<n;k++) {
        bool flag=false;
        for (int i=1;i<=m;i++) {
            if (dis[v[i]]>dis[u[i]]+w[i]) {
                dis[v[i]]=dis[u[i]]+w[i];
                flag=true;
            }
        }
        if (!flag) {
            break;
        }
    }
    bool flag=false;
    for (int i=1;i<=m;i++) {
        if (dis[v[i]]>dis[u[i]]+w[i]) {
            dis[v[i]]=dis[u[i]]+w[i];
            flag=true;
        }
    }
    if (!flag) {
        break;
    }
}
```

SPFA 算法　复杂度 $O(km)$，其中 k 可能为 2，n、m 约为 10^5 时不适用.

(1) 取队首 u，对它的出边松弛.

(2) 如果 $u \to v$ 使 `dis[v]` 更小，并且 v 不在队列，把 v 入队.

(3) 重复 (1)(2)，直到队空.

需要重复进队.

```cpp
int in[],dis[],vis[];
bool SPFA(int s) {
    for (int i=1;i<=3*n;i++) {
        dis[i]=1e10;
    }
    queue<int> Q;
    Q.push(s);
    vis[s]=true;
    dis[s]=0;
    while (!Q.empty()) {
        int now=Q.front();
        Q.pop();
        vis[now]=false;
        for (int i=head[now];i!=0;i=des[i].next) {
            int to=eds[i].to;
            if (dis[to]>dis[now]+eds[i].cost) {
                dis[to]=dis[now]+eds[i].cost;
                if (!vis[to]) {
                    vis[to]=true;
                    Q.push(to);
                    if (++in[to]>n) {
                        return false;
                    }
                }
            }
        }
    }
    return true;
}
```

查找强连通分量

连通图：图上任意两点可达.

强连通：有向图上两点互相可达，这两点强连通.

强连通图：有向图上任意两点互相可达，这个图是强连通图.

弱连通图：有向图看作无向图，任意两点互相可达，这个图是弱连通图.

强连通分量（Strongly Connected Components，SCC）：极大的强连通子图.

树边：每次搜索找到一个还没有访问过的结点的时候就形成了一条树边。

返祖边：也叫回边，即指向祖先结点的边（用栈来判断）。

横叉边：主要是在搜索的时候遇到了一个已经访问过且不在栈中的结点。

前向边：在搜索的时候遇到子树中的结点（`dfn[u]<dfn[v]`）的时候形成的。

Tarjan **算法**

`dfn[]` 时间戳　`low[]` 追溯值（最早的时间戳）

（1）任选一点 u 开始 DFS，记录下当前的 `dfn[u]` 和 `low[u]` 的值为 `num`，把 u 入栈.

（2）遍历 u 的子节点 v. $\begin{cases} 1.\ v\text{没有被搜索过} \to \texttt{dfs(v) low[u]=min(low[u],low[v])} \\ 2.\ v\text{被搜索过，判断}v\text{在不在栈里，}u \to v\text{返祖为}\texttt{low[u]-min(low[u],dfn[v])} \end{cases}$

（3）`low[u]` 确定，`if (low[u]==dfn[u])`，以 u 为根的树是一个强连通分量，栈里 u 及其后入栈的点全部出栈.

```cpp
vector<int> e[maxn];
int dfn[],low[],tot;
int stk[],instk[],top;
int scc[],siz[].cnt;

void tarjan(int x) {
    dfn[x]=low[x]=++tot;
    stk[++top]=x;
    instk[x]=1;
    for (int y:e[x]) {
        if (!dfn[y]) {
            tarjan(y);
            low[x]=min(low[x],low[y]
        } else if {
            low[x]=min(low[x],dfn[y])
        }
    }
    if (dfn[x]==low[x]) {
        ++cnt;
        while (stk[top+1]!=x) {
            int v=stk[top--];
            instk[v]=0;
            siz[cnt]++;
        }
    }
}
```

二分图匹配

匈牙利算法（二分图最大匹配）

参考：二分图匹配——通俗易懂 - cnblog

板子题 - HDU2063 **过山车**

RPG girls 今天和大家一起去游乐场玩，终于可以坐上梦寐以求的过山车了。可是，过山车的每一排只有两个座位，而且还有条不成文的规矩，就是每个女生必须找个个男生做 partner 和她同坐。但是，每个女孩都有各自的想法，举个例子把，Rabbit 只愿意和 XHD 或 PQK 做 partner，Grass 只愿意和 linle 或 LL 做 partner，PrincessSnow 愿意和水域浪子或伪酷儿做 partner。考虑到经费问题，boss 刘决定只让找到 partner 的人去坐过山车，其他的人，嘿嘿，就站在下面看着吧。聪明的 Acmer，你可以帮忙算算最多有多少对组合可以坐上过山车吗？

Input

输入数据的第一行是三个整数 K，M，N，分别表示可能的组合数目，女生的人数，男生的人数。0<K⩽1000，1⩽N 和 M⩽500. 接下来的 K 行，每行有两个数，分别表示女生 A_i 愿意和男生 B_j 做 partner。最后一个 0 结束输入。

Output

对于每组数据，输出一个整数，表示可以坐上过山车的最多组合数。

```cpp
const int N=505;
int line[N][N];
int girl[N],used[N];
int k,m,n;
bool found(int x) {
    for (int i=1; i<=n; i++) {
        if (line[x][i]&&!used[i]) {
            used[i]=1;
            if (girl[i]==0||found(girl[i])) {
                girl[i]=x;
                return 1;
            }
        }
    }
    return 0;
}

int main() {
    int x,y;
    while (scanf("%d",&k)&&k) {
        scanf("%d %d",&m,&n);
        memset(line,0,sizeof(line));
        memset(girl,0,sizeof(girl));
        for (int i=0; i<k; i++) {
            scanf("%d %d",&x,&y);
            line[x][y]=1;
        }
        int sum=0;
        for (int i=1; i<=m; i++) {
            memset(used,0,sizeof(used));
            if (found(i)) sum++;
        }
        printf("%d\n",sum);
    }
    return 0;
}
```

二分图最小点覆盖

最小点覆盖：选最少的点，满足每条边至少有一个端点被选。

König **定理**：二分图中，最小点覆盖 $=$ 最大匹配。

二分图最大独立集

最大独立集：选最多的点，满足两两之间没有边相连。

二分图中，最大独立集 $= n -$ 最小点覆盖。