## Lesson3（最小生成树，二分图）

> **大纲**
>
> **最小生成树（无向图）**
>
> 两种算法
>
> - **Prim算法**
>   - **朴素版Prim算法（稠密图）**O(n^2)
>   - 堆优化版Prim算法（稀疏图，不常用）O(mlogn)
> - **Kruskal算法（稀疏图）**
>   - 时间复杂度O(mlogm)，和O(mlogn)一个级别
>
> **二分图（和最大流相似）**
>
> - 如何判别是否为二分图（染色法DFS）  O(n+m)
> - 匈牙利算法（求二分图最大匹配）最坏O(nm)，实际运行时间一般远小于O(nm)

> **朴素版Prim算法**
>
> 步骤（和dijkstra算法相似）

```
S表示当前已经在连通块中的点集
dist[i] <- +∞
for (i=0; i<n; i++)
    t <- S外距离最近的点   （初始时都为+∞，随便选一点）
    用t更新其它点到集合的距离
    将t加入集合S，st[t] = true;
```

具体实现：O(n^2) 存储方式为**邻接矩阵**

```
static int N = 520, INF = 0x3f3f3f3f;

static int n, m;
static int[][] g = new int[N][N];
static int[] dist = new int[N];
static boolean[] st = new boolean[N];

static int prim() {
    Arrays.fill(dist, INF);

    int res = 0;
    for (int i=0; i<n; i++) {
        int t = -1;

        for (int j=1; j<=n; j++)
            if (!st[j] && (t == -1 || dist[j] < dist[t]))
                t = j;

        if (i != 0 && dist[t] == INF) return INF;
        if (i != 0) res += dist[t];
        st[t] = true;   // 将该点加入集合

        for (int j=1; j<=n; j++)
            dist[j] = Math.min(dist[j], g[t][j]);
    }

    return res;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=1; i<=n; i++) Arrays.fill(g[i], INF);

    while (m-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        ins.nextToken(); int c = (int)ins.nval;
        g[a][b] = g[b][a] = Math.min(g[a][b], c);
    }

    int t = prim();
```

```
    out.println((t == INF ? "impossible": t));

    out.flush();
}
```

堆优化思路与堆优化Dijkstra一致

**Kruskal算法（稀疏图，常数很小）**

步骤

```
将所有边按照权从小到大排序    //O(mlogm) 算法瓶颈，但排序常数小
按顺序枚举每条边 a<-w->b          //时间复杂度O(m)
 if a, b不连通       //并查集应用，近乎O(1)
     将该边加入连通块边集合
```

具体实现：O(mlogm) 并查集  只需存储每条边

```
static int N = 100010, M = 2*N, INF = 0x3f3f3f3f;

static int n, m;
static Edge[] edges = new Edge[M];
static int[] p = new int[N];

static int find(int x) {
    if (x != p[x]) p[x] = find(p[x]);
    return p[x];
}

static int kruskal() {
    Arrays.sort(edges, 0, m, (o1, o2) -> o1.w-o2.w);

    int res = 0, cnt = 0;
    for (int i=0; i<m; i++) {
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;

        if (find(a) != find(b)) {
            res += w;
            cnt++;
            p[find(a)] = find(b);
        }
    }

    return cnt < n-1? INF: res;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    for (int i=1; i<=n; i++) p[i] = i;  // 初始化并查集

    for (int i=0; i<m; i++) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        ins.nextToken(); int c = (int)ins.nval;
        edges[i] = new Edge(a, b, c);
    }

    int t = kruskal();
    out.println((t == INF? "impossible": t));

    out.flush();
}

static class Edge {
    int a, b, w;

    Edge(int aa, int bb, int ww) {
        a = aa; b = bb; w = ww;
    }
}
```

**二分图判别：染色法（DFS）**

重要性质：一个图是二分图，当且仅当图中不含奇数环（环的边数为奇数）

由于图中不含奇数环，所以染色过程中一定没有矛盾

步骤

```
for (i=1; i<=n; i++)
    if i未染色
        dfs(i, 1)
```

具体实现：**邻接表**

```
static int N = 100010, M = 2*N;

static int n, m;
static int idx;
static int[] h = new int[N], e = new int[M], ne = new int[M];
static int[] color = new int[N];

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

static boolean 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] == 0) {    // 此处括号不能省略!
            if (!dfs(j, 3-c)) return false;
        }
        else if (color[j] == c) return false;
    }

    return true;
}

public static void main(String[] args) throws Exception {
    ins.nextToken(); n = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    Arrays.fill(h, -1);

    while (m-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        add(a, b); add(b, a);
    }

    int flag = 1;
    for (int i=1; i<=n; i++)
        if (color[i] == 0) {
            if (!dfs(i, 1)) {
                flag = 0;
                break;
            }
        }

    out.println((flag == 1? "Yes": "No"));

    out.flush();
}
```

**匈牙利算法**：最坏O(nm)，实际运行时间一般远小于O(nm)

作用：给定一个二分图，求其**最大匹配**（成功匹配：不存在两条边共用一个顶点）

具体实现：使用**邻接表**存储

```
static int N = 510, M = 100010;

static int n1, n2, m;
static int idx;
static int[] h = new int[N], e = new int[M], ne = new int[M];
static int[] match = new int[N];
static boolean[] st = new boolean[N];

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

```
}

static boolean 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;
}


public static void main(String[] args) throws Exception {
    ins.nextToken(); n1 = (int)ins.nval;
    ins.nextToken(); n2 = (int)ins.nval;
    ins.nextToken(); m = (int)ins.nval;

    Arrays.fill(h, -1);

    while (m-- > 0) {
        ins.nextToken(); int a = (int)ins.nval;
        ins.nextToken(); int b = (int)ins.nval;
        add(a, b);
    }

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

    out.println(res);

    out.flush();
}
```