

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) $O(m\log n)$
- Kruskal算法 (稀疏图)
 - 时间复杂度 $O(m\log m)$, 和 $O(m\log n)$ 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)

```
1 | S表示当前已经在连通块中的点集
2 | dist[i] <- +∞
3 | for (i=0; i<n; i++)
4 |     t <- S外距离最近的点    (初始时都为+∞, 随便选一点)
5 |     用t更新其它点到集合的距离
6 |     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

```
1 | static int N = 520, INF = 0x3f3f3f3f;
2 |
3 | static int n, m;
4 | static int[][] g = new int[N][N];
5 | static int[] dist = new int[N];
6 | static boolean[] st = new boolean[N];
7 |
8 | static int prim() {
9 |     Arrays.fill(dist, INF);
10 |
11 |     int res = 0;
12 |     for (int i=0; i<n; i++) {
13 |         int t = -1;
14 |
15 |         for (int j=1; j<=n; j++)
16 |             if (!st[j] && (t == -1 || dist[j] < dist[t]))
17 |                 t = j;
18 |
19 |         if (i != 0 && dist[t] == INF) return INF;
20 |         if (i != 0) res += dist[t];
21 |         st[t] = true;    // 将该点加入集合
22 |
23 |         for (int j=1; j<=n; j++)
24 |             dist[j] = Math.min(dist[j], g[t][j]);
25 |     }
26 |
27 |     return res;
28 | }
29 |
30 | public static void main(String[] args) throws Exception {
31 |     ins.nextToken(); n = (int)ins.nval;
32 |     ins.nextToken(); m = (int)ins.nval;
33 |
34 |     for (int i=1; i<=n; i++) Arrays.fill(g[i], INF);
35 |
36 |     while (m-- > 0) {
37 |         ins.nextToken(); int a = (int)ins.nval;
38 |         ins.nextToken(); int b = (int)ins.nval;
39 |         ins.nextToken(); int c = (int)ins.nval;
40 |         g[a][b] = g[b][a] = Math.min(g[a][b], c);
41 |     }
42 |
43 |     int t = prim();
```

```
44     out.println((t == INF ? "impossible": t));
45
46     out.flush();
47 }
```

堆优化思路与堆优化Dijkstra一致

Kruskal算法（稀疏图，常数很小）

步骤

- 1
- 将所有边按照权从小到大排序 // $O(m\log m)$ 算法瓶颈，但排序常数小
- 2
- 按顺序枚举每条边 $a \leftarrow w \rightarrow b$ //时间复杂度 $O(m)$
- 3
- $\text{if } a, b \text{ 不连通}$ //并查集应用，近乎 $O(1)$
- 4
- 将该边加入连通块边集合

具体实现： $O(m\log m)$ 并查集 只需存储每条边

```
1  static int N = 100010, M = 2*N, INF = 0x3f3f3f3f;
2
3  static int n, m;
4  static Edge[] edges = new Edge[M];
5  static int[] p = new int[N];
6
7  static int find(int x) {
8      if (x != p[x]) p[x] = find(p[x]);
9      return p[x];
10 }
11
12 static int kruskal() {
13     Arrays.sort(edges, 0, m, (o1, o2) -> o1.w-o2.w);
14
15     int res = 0, cnt = 0;
16     for (int i=0; i<m; i++) {
17         int a = edges[i].a, b = edges[i].b, w = edges[i].w;
18
19         if (find(a) != find(b)) {
20             res += w;
21             cnt++;
22             p[find(a)] = find(b);
23         }
24     }
25
26     return cnt < n-1? INF: res;
27 }
28
29 public static void main(String[] args) throws Exception {
30     ins.nextToken(); n = (int)ins.nval;
31     ins.nextToken(); m = (int)ins.nval;
32
33     for (int i=1; i<=n; i++) p[i] = i; // 初始化并查集
34
35     for (int i=0; i<m; i++) {
36         ins.nextToken(); int a = (int)ins.nval;
37         ins.nextToken(); int b = (int)ins.nval;
38         ins.nextToken(); int c = (int)ins.nval;
39         edges[i] = new Edge(a, b, c);
40     }
41
42     int t = kruskal();
43     out.println((t == INF? "impossible": t));
44
45     out.flush();
46 }
47
48 static class Edge {
49     int a, b, w;
50
51     Edge(int aa, int bb, int ww) {
52         a = aa; b = bb; w = ww;
53     }
54 }
```

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

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

具体实现：邻接表

```
1  static int N = 100010, M = 2*N;
2
3  static int n, m;
4  static int idx;
5  static int[] h = new int[N], e = new int[M], ne = new int[M];
6  static int[] color = new int[N];
7
8  static void add(int a, int b) {
9      e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
10 }
11
12 static boolean dfs(int u, int c) {
13     color[u] = c;
14
15     for (int i=h[u]; i!=-1; i=ne[i]) {
16         int j = e[i];
17         if (color[j] == 0) { // 此处括号不能省略!
18             if (!dfs(j, 3-c)) return false;
19         }
20         else if (color[j] == c) return false;
21     }
22
23     return true;
24 }
25
26 public static void main(String[] args) throws Exception {
27     ins.nextToken(); n = (int)ins.nval;
28     ins.nextToken(); m = (int)ins.nval;
29
30     Arrays.fill(h, -1);
31
32     while (m-- > 0) {
33         ins.nextToken(); int a = (int)ins.nval;
34         ins.nextToken(); int b = (int)ins.nval;
35         add(a, b); add(b, a);
36     }
37
38     int flag = 1;
39     for (int i=1; i<=n; i++)
40         if (color[i] == 0) {
41             if (!dfs(i, 1)) {
42                 flag = 0;
43                 break;
44             }
45         }
46
47     out.println((flag == 1? "Yes": "No"));
48
49     out.flush();
50 }
```

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

```
1  static int N = 510, M = 100010;
2
3  static int n1, n2, m;
4  static int idx;
5  static int[] h = new int[N], e = new int[M], ne = new int[M];
6  static int[] match = new int[N];
7  static boolean[] st = new boolean[N];
8
9  static void add(int a, int b) {
10     e[idx] = b; ne[idx] = h[a]; h[a] = idx++;
```

```

11 }
12
13 static boolean find(int x) {
14     for (int i=h[x]; i!=-1; i=ne[i]) {
15         int j = e[i];
16         if (!st[j]) {
17             st[j] = true;
18
19             if (match[j] == 0 || find(match[j])) {
20                 match[j] = x;
21                 return true;
22             }
23         }
24     }
25
26     return false;
27 }
28
29
30 public static void main(String[] args) throws Exception {
31     ins.nextToken(); n1 = (int)ins.nval;
32     ins.nextToken(); n2 = (int)ins.nval;
33     ins.nextToken(); m = (int)ins.nval;
34
35     Arrays.fill(h, -1);
36
37     while (m-- > 0) {
38         ins.nextToken(); int a = (int)ins.nval;
39         ins.nextToken(); int b = (int)ins.nval;
40         add(a, b);
41     }
42
43     int res = 0;
44     for (int i=1; i<=n1; i++) {
45         Arrays.fill(st, false);
46         if (find(i)) res++;
47     }
48
49     out.println(res);
50
51     out.flush();
52 }

```