

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 | #include <iostream>
2 | #include <cstring>
3 | #include <algorithm>
4 |
5 | using namespace std;
6 |
7 | const int N = 510, INF = 0x3f3f3f3f;
8 |
9 | int n, m;
10 | int g[N][N];
11 | int dist[N];    //dist[i]表示当前结点i到连通块的最短距离
12 | bool st[N];
13 |
14 | int prim() {
15 |     memset(dist, 0x3f, sizeof dist);
16 |
17 |     int res = 0;    //记录最小生成树权重
18 |     for (int i=0; i<n; i++) {    //迭代n次
19 |         int t = -1;
20 |
21 |         //找出距离集合最近的点
22 |         for (int j=1; j<=n; j++)
23 |             if (!st[j] && (t==-1 || dist[j]<dist[t]))
24 |                 t = j;
25 |
26 |         //第一个点特判
27 |         if (i && dist[t] == INF) return INF;
28 |         if (i) res += dist[t];
29 |         st[t] = true;    //加入连通块
30 |
31 |         //用t更新其他邻边到集合的距离
32 |         for (int j=1; j<=n; j++) dist[j] = min(dist[j], g[t][j]);
33 |     }
34 |
35 |     return res;
36 | }
37 |
38 | int main(void) {
39 |     scanf("%d%d", &n, &m);
40 |
41 |     memset(g, 0x3f, sizeof g);
42 |
43 |     for (int i=0; i<m; i++) {
```

```
44     int a, b, c;
45     scanf("%d%d%d", &a, &b, &c);
46     g[a][b] = g[b][a] = min(g[a][b], c);
47 }
48
49 int t = prim();
50
51 if (t == INF) puts("impossible");
52 else printf("%d\n", t);
53
54 return 0;
55 }
```

堆优化思路与堆优化Dijkstra一致

Kruskal算法 (稠密图, 常数很小)

步骤

- 1
- 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
- 2
- 按顺序枚举每条边 a<-w->b //时间复杂度O(m)
- 3
- if a, b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(mlogm) 并查集 只需存储每条边

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int N = 100010, M = 2*N, INF = 0x3f3f3f3f;
8
9  struct Edge {
10     int a, b, w;
11
12     bool operator< (const Edge &w) const {
13         return w < w.w;
14     }
15 } edges[M];
16
17 int n, m;
18 int p[N];
19
20 //并查集
21 int find(int x) {
22     if (p[x] != x) p[x] = find(p[x]);
23     return p[x];
24 }
25
26 int kruskal() {
27     //对所有边进行排序, 算法瓶颈
28     sort(edges, edges+m);
29
30     int res = 0, cnt = 0; // res记录最小生成树权重, cnt记录当前最小生成树边数
31     for (int i=0; i<m; i++) {
32         int a = edges[i].a, b = edges[i].b, w = edges[i].w;
33
34         if (find(a) != find(b)) {
35             res += w;
36             cnt++;
37             p[find(a)] = find(b); //将两个连通块合并
38         }
39     }
40
41     if (cnt < n-1) return INF;
42     return res;
43 }
44
45 int main(void) {
46     scanf("%d%d", &n, &m);
47
48     //初始化并查集
49     for (int i=1; i<=n; i++) p[i] = i;
50
51     for (int i=0; i<m; i++) {
52         int a, b, c;
```

```
53     scanf("%d%d%d", &a, &b, &c);
54     edges[i] = {a, b, c};
55 }
56
57 int t = kruskal();
58
59 if (t == INF) puts("impossible");
60 else printf("%d", t);
61
62 return 0;
63 }
```

### 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  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int N = 100010, M = 2*N;
8
9  int n, m;
10 int h[N], e[M], ne[M], idx;
11 int color[N];    //第i个结点的颜色
12
13 void add(int a, int b) {
14     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
15 }
16
17 bool dfs(int u, int c) {
18     color[u] = c;
19
20     for (int i=h[u]; i!=-1; i=ne[i]) {
21         int j = e[i];
22         if (!color[j]) {    //未染色
23             if (!dfs(j, 3-c)) return false;
24         }
25         else if (color[j] == c) return false;    //和父结点一个颜色
26     }
27
28     return true;
29 }
30
31 int main(void) {
32     scanf("%d%d", &n, &m);
33
34     memset(h, -1, sizeof h);
35
36     for (int i=0; i<m; i++) {
37         int a, b;
38         scanf("%d%d", &a, &b);
39         add(a, b), add(b, a);
40     }
41
42
43     bool flag = true;
44     for (int i=1; i<=n; i++) {
45         if (!color[i]) {
46             if (!dfs(i, 1)) {
47                 flag = false;
48                 break;
49             }
50         }
51     }
```

```
52
53     if (flag) puts("Yes");
54     else puts("No");
55
56     return 0;
57 }
```

**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  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4
5  using namespace std;
6
7  const int N = 510, M = 100010;
8
9  int n1, n2, m;
10 int h[N], e[M], ne[M], idx;
11 int match[N];    //match[i]表示当前右半部中的点i匹配的左半部点
12 bool st[N];      //st[i]表示右半部点i是否已被某个特定的左半部点考虑过
13
14 void add(int a, int b) {
15     e[idx] = b, ne[idx] = h[a], h[a] = idx++;
16 }
17
18 bool find(int x) {
19     for (int i=h[x]; i!=-1; i=ne[i]) {
20         int j = e[i];
21         if (!st[j]) {
22             st[j] = true;
23
24             if (match[j] == 0 || find(match[j])) {
25                 match[j] = x;
26                 return true;
27             }
28         }
29     }
30
31     return false;
32 }
33
34 int main(void) {
35     scanf("%d%d%d", &n1, &n2, &m);
36
37     memset(h, -1, sizeof h);
38
39     for (int i=0; i<m; i++) {
40         int a, b;
41         scanf("%d%d", &a, &b);
42         add(a, b);    //只需存储从左向右的边
43     }
44
45     int res = 0;
46     for (int i=1; i<=n1; i++) {
47         memset(st, false, sizeof st);
48         if (find(i)) res++;
49     }
50
51     cout << res;
52
53     return 0;
54 }
```