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**
>
> ```cpp
> #include <iostream>
> #include <cstring>
> #include <algorithm>
>
> using namespace std;
>
> const int N = 510, INF = 0x3f3f3f3f;
>
> int n, m;
> int g[N][N];
> int dist[N];    //dist[i]表示当前结点i到连通块的最短距离
> bool st[N];
>
> int prim() {
>     memset(dist, 0x3f, sizeof dist);
>
>     int res = 0;    //记录最小生成树权重
>     for (int i=0; i<n; i++) {  //迭代n次
>         int t = -1;
>
>         //找出距离集合最近的点
>         for (int j=1; j<=n; j++)
>             if (!st[j] && (t==-1 || dist[j]<dist[t]))
>                 t = j;
>
>         //第一个点特判
>         if (i && dist[t] == INF) return INF;
>         if (i) res += dist[t];
>         st[t] = true;   //加入连通块
>
>         //用t更新其他邻边到集合的距离
>         for (int j=1; j<=n; j++) dist[j] = min(dist[j], g[t][j]);
>     }
>
>     return res;
> }
>
> int main(void) {
>     scanf("%d%d", &n, &m);
>
>     memset(g, 0x3f, sizeof g);
>
>     for (int i=0; i<m; i++) {
> ```

```
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        g[a][b] = g[b][a] = min(g[a][b], c);
    }

    int t = prim();

    if (t == INF) puts("impossible");
    else printf("%d\n", t);

    return 0;
}
```

堆优化思路与堆优化Dijkstra一致

Kruskal算法（稠密图，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

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

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

struct Edge {
    int a, b, w;

    bool operator< (const Edge &W) const {
        return w < W.w;
    }
} edges[M];

int n, m;
int p[N];

//并查集
int find(int x) {
    if (p[x] != x) p[x] = find(p[x]);
    return p[x];
}

int kruskal() {
    //对所有边进行排序，算法瓶颈
    sort(edges, edges+m);

    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
    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);   //将两个连通块合并
        }
    }

    if (cnt < n-1) return INF;
    return res;
}

int main(void) {
    scanf("%d%d", &n, &m);

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

    for (int i=0; i<m; i++) {
        int a, b, c;
```

```
        scanf("%d%d%d", &a, &b, &c);
        edges[i] = {a, b, c};
    }

    int t = kruskal();

    if (t == INF) puts("impossible");
    else printf("%d", t);

    return 0;
}
```

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**

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 100010, M = 2*N;

int n, m;
int h[N], e[M], ne[M], idx;
int color[N];   //第i个结点的颜色

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

bool 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]) {    //未染色
            if (!dfs(j, 3-c)) return false;
        }
        else if (color[j] == c) return false;   //和父结点一个颜色
    }

    return true;
}

int main(void) {
    scanf("%d%d", &n, &m);

    memset(h, -1, sizeof h);

    for (int i=0; i<m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }


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

```
    
    if (flag) puts("Yes");
    else puts("No");

    return 0;
}
```

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

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;

const int N = 510, M = 100010;

int n1, n2, m;
int h[N], e[M], ne[M], idx;
int match[N];   //match[i]表示当前右半部中的点i匹配的左半部点
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

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

bool 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;
}

int main(void) {
    scanf("%d%d%d", &n1, &n2, &m);

    memset(h, -1, sizeof h);

    for (int i=0; i<m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);  //只需存储从左向右的边
    }

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

    cout << res;

    return 0;
}
```