# 图论-最小生成树的典型应用

## 一、AcWing 1140. 最短网络

【题目描述】

农夫约翰被选为他们镇的镇长！

他其中一个竞选承诺就是在镇上建立起互联网，并连接到所有的农场。

约翰已经给他的农场安排了一条高速的网络线路，他想把这条线路共享给其他农场。

约翰的农场的编号是$1$，其他农场的编号是$2 \sim n$。

为了使花费最少，他希望用于连接所有的农场的光纤总长度尽可能短。

你将得到一份各农场之间连接距离的列表，你必须找出能连接所有农场并使所用光纤最短的方案。

【输入格式】

第一行包含一个整数$n$，表示农场个数。

接下来$n$行，每行包含$n$个整数，输入一个对角线上全是$0$的对称矩阵。

其中第$x+1$行$y$列的整数表示连接农场$x$和农场$y$所需要的光纤长度。

【输出格式】

输出一个整数，表示所需的最小光纤长度。

【数据范围】

$3 \leq n \leq 100$

每两个农场间的距离均是非负整数且不超过$100000$。

【输入样例】

```
4
0  4  9  21
4  0  8  17
9  8  0  16
21 17 16  0
```

【输出样例】

```
28
```

【分析】

模板题，当做复习一下Prim算法~

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 110;
int g[N][N];
int st[N], dis[N];
int n;

int prim()
{
    memset(dis, 0x3f, sizeof dis);
    dis[1] = 0;
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!st[j] && (!~t || dis[j] < dis[t])) t = j;
        res += dis[t];
        st[t] = 1;
        for (int j = 1; j <= n; j++) dis[j] = min(dis[j], g[t][j]);
    }
    return res;
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> g[i][j];
    cout << prim() << endl;
```

```
35      return 0;
36  }
```

# 二、AcWing 1141. 局域网

【题目描述】

某个局域网内有$n$台计算机和$k$条双向网线，计算机的编号是$1 \sim n$。由于搭建局域网时工作人员的疏忽，现在局域网内的连接形成了回路，我们知道如果局域网形成回路那么数据将不停的在回路内传输，造成网络卡的现象。

注意：

- 对于某一个连接，虽然它是双向的，但我们不将其当做回路。本题中所描述的回路至少要包含两条不同的连接。
- 两台计算机之间最多只会存在一条连接。
- 不存在一条连接，它所连接的两端是同一台计算机。

因为连接计算机的网线本身不同，所以有一些连线不是很畅通，我们用$f(i,j)$表示$i,j$之间连接的畅通程度，$f(i,j)$值越小表示$i,j$之间连接越通畅。

现在我们需要解决回路问题，我们将除去一些连线，使得网络中没有回路且不影响连通性（即如果之前某两个点是连通的，去完之后也必须是连通的），并且被除去网线的$\Sigma f(i,j)$最大，请求出这个最大值。

【输入格式】

第一行两个正整数$n,k$。

接下来的$k$行每行三个正整数$i,j,m$表示$i,j$两台计算机之间有网线联通，通畅程度为$m$。

【输出格式】

一个正整数，表示被除去网线的$\Sigma f(i,j)$的最大值。

【数据范围】

$1 \leq n \leq 100$

$0 \leq k \leq 200$

$1 \leq f(i,j) \leq 1000$

【输入样例】

```
5 5
1 2 8
1 3 1
1 5 3
2 4 5
3 4 2
```

【输出样例】

```
8
```

【分析】

---

本题可能存在多个连通块，需要我们去掉权值尽可能大的边，且不影响每个连通块内各点的连通性。即需要求出最小的“生成森林”。我们在做Kruskal算法时，当某条边的两个端点已经被连通时，说明该边需要去除，将***res***加上这条边的权值即可。

---

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 110, M = 210;
int pre[N];
int n, m;

struct Edge
{
    int x, y, w;
    bool operator< (const Edge& t) const
    {
        return w < t.w;
    }
}e[M];

int find(int k)
{
    if (pre[k] == k) return k;

    return pre[k] = find(pre[k]);
```

```
}

int kruskal()
{
    sort(e, e + m);
    int res = 0;
    for (int i = 0; i < m; i++)
    {
        int px = find(e[i].x), py = find(e[i].y);
        if (px != py) pre[px] = py;//x与y不在一个连通块中则连接
        else res += e[i].w;//x与y已经连通了说明这条边需要去掉
    }
    return res;
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) pre[i] = i;
    for (int i = 0; i < m; i++) cin >> e[i].x >> e[i].y >> e[i].w;
    cout << kruskal() << endl;
    return 0;
}
```

# 三、AcWing 1142. 繁忙的都市

【题目描述】

城市$C$是一个非常繁忙的大都市，城市中的道路十分的拥挤，于是市长决定对其中的道路进行改造。

城市$C$的道路是这样分布的：

城市中有$n$个交叉路口，编号是$1 \sim n$，有些交叉路口之间有道路相连，两个交叉路口之间最多有一条道路相连接。

这些道路是**双向**的，且把所有的交叉路口直接或间接的连接起来了。

每条道路都有一个分值，分值越小表示这个道路越繁忙，越需要进行改造。

但是市政府的资金有限，市长希望进行改造的道路越少越好，于是他提出下面的要求：

- 改造的那些道路能够把所有的交叉路口直接或间接的连通起来。
- 在满足要求$1$的情况下，改造的道路尽量少。
- 在满足要求$1,2$的情况下，改造的那些道路中分值**最大值**尽量小。

作为市规划局的你，应当作出最佳的决策，选择哪些道路应当被修建。

【输入格式】

第一行有两个整数$n, m$表示城市有$n$个交叉路口，$m$条道路。

接下来$m$行是对每条道路的描述，每行包含三个整数$u, v, c$表示交叉路口$u$和$v$之间有道路相连，分值为$c$。

【输出格式】

两个整数$s, max$，表示你选出了几条道路，分值最大的那条道路的分值是多少。

【数据范围】

$1 \leq n \leq 300$

$1 \leq m \leq 8000$

$1 \leq c \leq 10000$

【输入样例】

```
4 5
1 2 3
1 4 5
2 4 7
2 3 6
3 4 8
```

【输出样例】

```
3 6
```

【分析】

---

首先如果要连通所有点且边数最少，则边的数量一定是$n-1$条。要想求出连通所有点且边权最大的边最小，则只需在做Kruskal算法时不是对边权之和进行统计，而是对边权的最大值进行统计，由于边权是从小到大进行枚举的，因此当连接了$n-1$条边将所有点连通后，最后连接的一条边也就是第$n-1$条的权值一定是最大的，返回答案即可。

---

【代码】

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

```cpp
#include <algorithm>
using namespace std;

const int N = 310, M = 8010;
int pre[N];
int n, m;

struct Edge
{
    int x, y, w;
    bool operator< (const Edge& t) const
    {
        return w < t.w;
    }
}e[M];

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

int kruskal()
{
    sort(e, e + m);
    int cnt = 0;//当前已连接的边数
    for (int i = 0; i < m; i++)
    {
        int px = find(e[i].x), py = find(e[i].y);
        if (px != py) pre[px] = py, cnt++;
        if (cnt == n - 1) return e[i].w;//已产生最小生成树,最后连接的边即为最大权值的边
    }
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) pre[i] = i;
    for (int i = 0; i < m; i++) cin >> e[i].x >> e[i].y >> e[i].w;
    cout << n - 1 << ' ' << kruskal() << endl;
    return 0;

}
```

# 四、AcWing 1143. 联络员

【题目描述】

Tyvj已经一岁了，网站也由最初的几个用户增加到了上万个用户，随着Tyvj网站的逐步壮大，管理员的数目也越来越多，现在你身为Tyvj管理层的联络员，希望你找到一些通信渠道，使得管理员两两都可以联络（直接或者是间接都可以）。本题中所涉及的通信渠道都是双向的。

Tyvj是一个公益性的网站，没有过多的利润，所以你要尽可能的使费用少才可以。

目前你已经知道，Tyvj的通信渠道分为两大类，一类是必选通信渠道，无论价格多少，你都需要把所有的都选择上；还有一类是选择性的通信渠道，你可以从中挑选一些作为最终管理员联络的通信渠道。

数据保证给出的通信渠道可以让所有的管理员联通。

注意：对于某两个管理员$u,v$，他们之间可能存在多条通信渠道，你的程序应该累加所有$u,v$之间的必选通信渠道。

【输入格式】

第一行两个整数$n,m$表示Tyvj一共有$n$个管理员，有$m$个通信渠道；

第二行到$m+1$行，每行四个非负整数，$p,u,v,w(1\leq u,v\leq n)$。当$p=1$时，表示这个通信渠道为必选通信渠道；当$p=2$时，表示这个通信渠道为选择性通信渠道；$u,v,w$表示本条信息描述的是$u,v$管理员之间的通信渠道，$u$可以收到$v$的信息，$v$也可以收到$u$的信息，$w$表示费用。

【输出格式】

一个整数，表示最小的通信费用。

【数据范围】

$1\leq n\leq 2000$

$1\leq m\leq 10000$

【输入样例】

```
5 6
1 1 2 1
1 2 3 1
1 3 4 1
1 4 1 1
2 2 5 10
2 2 5 5
```

【输出样例】

```
9
```

【分析】

---

先将所有必选的边连起来，然后对非必选的边跑一遍Kruskal即可。

---

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 2010, M = 10010;
int pre[N];
int n, m, k, res;//k为非必选边的数量

struct Edge
{
    int x, y, w;
    bool operator< (const Edge& t) const
    {
        return w < t.w;
    }
}e[M];

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);

}
```

```cpp

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) pre[i] = i;
    while (m--)
    {
        int p, u, v, w;
        cin >> p >> u >> v >> w;
        if (p == 1)
        {
            pre[find(u)] = find(v);
            res += w;
        }
        else e[k++] = { u, v, w };
    }
    sort(e, e + k);
    for (int i = 0; i < k; i++)
    {
        int px = find(e[i].x), py = find(e[i].y);
        if (px != py) pre[px] = py, res += e[i].w;
    }
    cout << res << endl;
    return 0;
}
```

# 五、AcWing 1144. 连接格点

【题目描述】

有一个$m$行$n$列的点阵，相邻两点可以相连。

一条纵向的连线花费一个单位，一条横向的连线花费两个单位。

某些点之间已经有连线了，试问至少还需要花费多少个单位才能使所有的点全部连通。

【输入格式】

第一行输入两个正整数$m$和$n$。

以下若干行每行四个正整数$x_1, y_1, x_2, y_2$，表示第$x_1$行第$y_1$列的点和第$x_2$行第$y_2$列的点已经有连线。

输入保证$|x_1 - x_2| + |y_1 - y_2| = 1$。

【输出格式】

输出使得连通所有点还需要的最小花费。

【数据范围】

$1 \leq m, n \leq 1000$

$0 \leq$ 已经存在的连线数 $\leq 10000$

【输入样例】

```
2 2
1 1 2 1
```

【输出样例】

```
3
```

【分析】

---

根据Kruskal算法建立最小生成树的思想，我们需要优先使用纵向的边连接，其次再使用横向的边，这样才能使花费最小。

先将$n \times m$的二维数组的每个元素从左至右从上至下分别映射为$1, 2, \ldots, n \times m$作为每个点的标号。先将已存在的连线两点进行连接。

假设纵向的边统一为从上往下的，即从点$(i, j)$连到点$(i+1, j)$的，横向的边统一为从左往右的，即从点$(i, j)$连到点$(i, j+1)$。因此我们后续有以下两个步骤：

- 先枚举所有纵向的边，即枚举$1 \sim n-1$行的每个点，如果$(i, j)$与$(i+1, j)$不连通，则将其连接，花费$+1$；
- 再枚举所有横向的边，即枚举$1 \sim m-1$列的每个点，如果$(i, j)$与$(i, j+1)$不连通，则将其连接，花费$+2$。

---

【代码】

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;

const int N = 1010;
int pre[N * N];
int g[N][N];
int n, m, res;
```

```cpp

int find(int k)
{
    if (pre[k] == k) return k;
    return pre[k] = find(pre[k]);
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i < N * N; i++) pre[i] = i;
    //将二维下标映射成一维
    for (int i = 1, t = 1; i <= n; i++)
        for (int j = 1; j <= m; j++, t++)
            g[i][j] = t;
    int x1, y1, x2, y2;
    while (cin >> x1 >> y1 >> x2 >> y2) pre[find(g[x1][y1])] = find(g[x2][y2]);
    //先枚举第1至n-1行每一条向下的边
    for (int i = 1; i < n; i++)
        for (int j = 1; j <= m; j++)
        {
            int px = find(g[i][j]), py = find(g[i + 1][j]);
            if (px != py) pre[px] = py, res++;
        }
    //再枚举第1至m-1列每一条向右的边
    for (int i = 1; i <= n; i++)
        for (int j = 1; j < m; j++)
        {
            int px = find(g[i][j]), py = find(g[i][j + 1]);
            if (px != py) pre[px] = py, res += 2;
        }
    cout << res << endl;
    return 0;
}
```