图论-最小生成树的扩展应用

一、AcWing 1146. 新的开始（虚拟源点）

【题目描述】

发展采矿业当然首先得有矿井，小FF花了上次探险获得的千分之一的财富请人在岛上挖了n口矿井，但他似乎忘记了考虑矿井供电问题。

为了保证电力的供应，小FF想到了两种办法：

- 在矿井i上建立一个发电站，费用为v_i（发电站的输出功率可以供给任意多个矿井）。
- 将这口矿井i与另外的已经有电力供应的矿井j之间建立电网，费用为$p_{i,j}$。

小FF希望你帮他想出一个保证所有矿井电力供应的最小花费方案。

【输入格式】

第一行包含一个整数n，表示矿井总数。

接下来n行，每行一个整数，第i个数v_i表示在第i口矿井上建立发电站的费用。

接下来为一个$n \times n$的矩阵P，其中$p_{i,j}$表示在第i口矿井和第j口矿井之间建立电网的费用。

数据保证$p_{i,j} = p_{j,i}$，且$p_{i,i} = 0$。

【输出格式】

输出一个整数，表示让所有矿井获得充足电能的最小花费。

【数据范围】

$1 \leq n \leq 300$

$0 \leq v_i, p_{i,j} \leq 10^5$

【输入样例】

```
4
5
4
4
3
0 2 2 2
2 0 3 3
2 3 0 4
2 3 4 0
```

【输出样例】

```
9
```

【分析】

我们可以建一个“超级发电站”，对于第一种操作，在某个矿井i上建立发电站费用为v_i，可以转换为将矿井i与“超级发电站”连接所需的费用为v_i，设这个“超级发电站”为0号点，$g[i][j]$为邻接矩阵，即$g[0][i]=g[i][0]=v_i$。之后，我们只需在加上了虚拟源点的这$n+1$个点做一遍最小生成树算法即可，本文使用Prim算法。

【代码】

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

const int N = 310;
int g[N][N];
int dis[N], st[N];
int n;

int prim()
{
    memset(dis, 0x3f, sizeof dis);
    int res = 0;
    for (int i = 0; i <= n; i++)
    {
        int t = -1;

        for (int j = 0; j <= n; j++)
```

```
            if (!st[j] && (!~t || dis[j] < dis[t])) t = j;
        st[t] = true;
        if (i) res += dis[t];
        for (int j = 0; j <= n; j++) dis[j] = min(dis[j], g[t][j]);
    }
    return res;
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) { cin >> g[0][i]; g[i][0] = g[0][i]; }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            cin >> g[i][j];
    cout << prim() << endl;
    return 0;
}
```

二、AcWing 1145. 北极通讯网络

【题目描述】

北极的某区域共有n座村庄，每座村庄的坐标用一对整数(x,y)表示。

为了加强联系，决定在村庄之间建立通讯网络，使每两座村庄之间都可以直接或间接通讯。

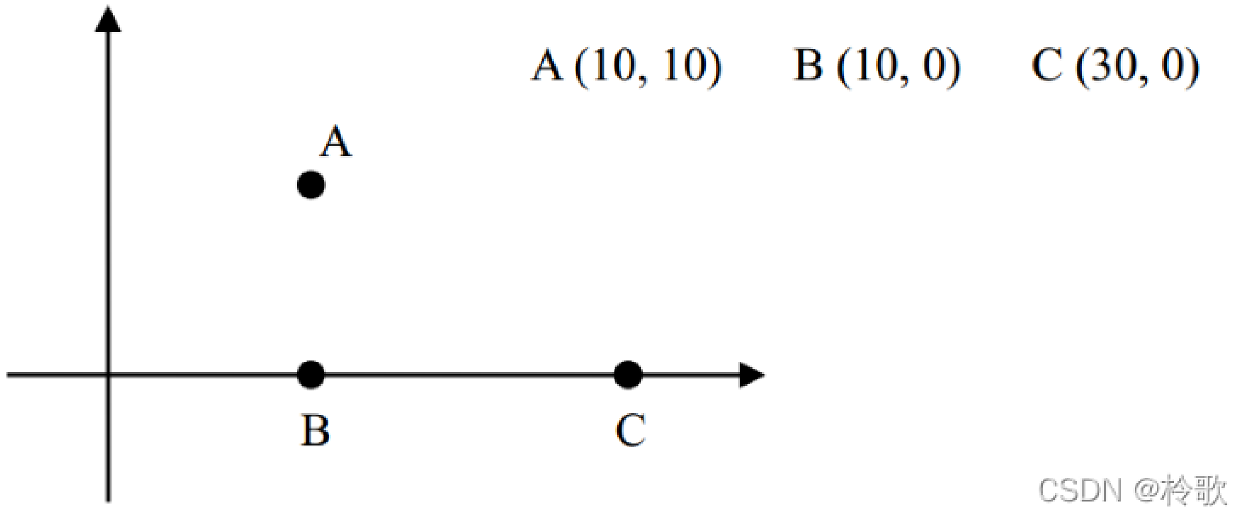
通讯工具可以是无线电收发机，也可以是卫星设备。

无线电收发机有多种不同型号，不同型号的无线电收发机有一个不同的参数d，两座村庄之间的距离如果不超过d，就可以用该型号的无线电收发机直接通讯，d值越大的型号价格越贵。现在要先选择某一种型号的无线电收发机，然后统一给所有村庄配备，**数量不限**，但型号都是**相同**的。

配备卫星设备的两座村庄无论相距多远都可以直接通讯，但卫星设备是**有限**的，只能给一部分村庄配备。

现在有k台卫星设备，请你编一个程序，计算出应该如何分配这k台卫星设备，才能使所配备的无线电收发机的d值最小。

例如，对于下面三座村庄：



其中，$|AB|=10,|BC|=20,|AC|=10\sqrt{5}\approx 22.36$。

如果没有任何卫星设备或只有1台卫星设备（$k=0$或$k=1$），则满足条件的最小的$d=20$，因为A和B，B和C可以用无线电直接通讯；而A和C可以用B中转实现间接通讯（即消息从A传到B，再从B传到C）。

如果有2台卫星设备（$k=2$），则可以把这两台设备分别分配给B和C，这样最小的d可取10，因为A和B之间可以用无线电直接通讯；B和C之间可以用卫星直接通讯；A和C可以用B中转实现间接通讯。

如果有3台卫星设备，则A,B,C两两之间都可以直接用卫星通讯，最小的d可取0。

【输入格式】

第一行为由空格隔开的两个整数n,k。

接下来n行，每行两个整数，第i行的x_i,y_i表示第i座村庄的坐标(x_i,y_i)。

【输出格式】

一个实数，表示最小的d值，结果保留2位小数。

【数据范围】

$1\le n\le 500$

$0\le x,y\le 10^4$

$0\le k\le 100$

【输入样例】

```
3 2
10 10
10 0
30 0
```

【输出样例】

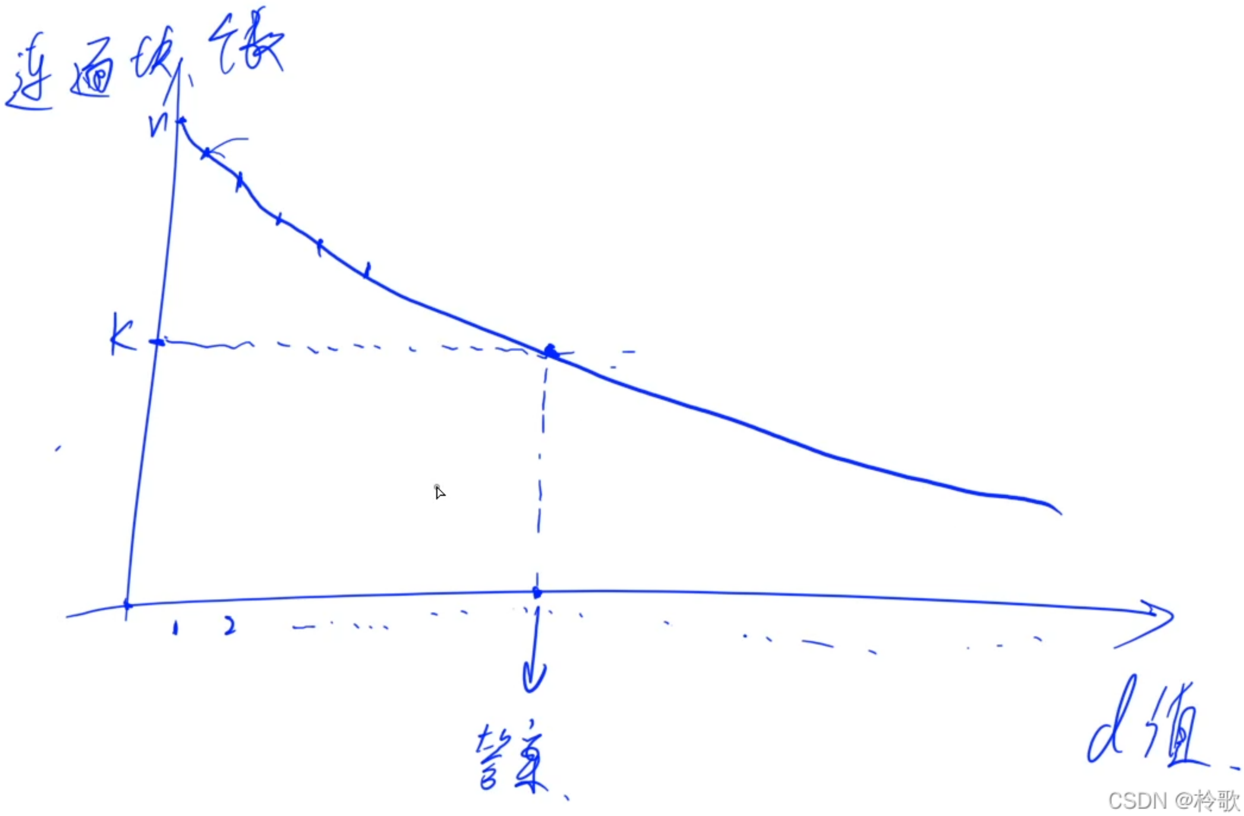
```
10.00
```

【分析】

题意为找一个最小的d值，使得将所有权值大于d的边删去之后整个图形的连通块个数不超过k。

假设Kruskal算法当前已经循环完第i条边，说明已经求出了由前i条边构成的所有连通块。

首先预处理出每条边即每两点间的距离。由于我们在做Kruskal算法的时候是从小到大枚举每条边的，因此我们可以在连边的同时记录连通块的数量，初始时连通块数量$cnt = n$，每连通一条边后cnt减一，当$cnt \le k$的时候，最后连接的那条边的权值即为使得$cnt \le k$的最小值也就是最小的d，如下图所示：



【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
#define X first
#define Y second
using namespace std;

typedef pair<int, int> PII;
const int N = 510, M = N * N;
PII p[N];
int pre[N];
int n, m, k;

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

double get_dis(PII a, PII b)
{
    return sqrt(pow(a.X - b.X, 2) + pow(a.Y - b.Y, 2));
}

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

double kruskal()
{
    sort(e, e + m);
    double res = 0;
    int cnt = n;
    for (int i = 0; i < m; i++)
    {
        if (cnt <= k) return res;
```

```
        int px = find(e[i].x), py = find(e[i].y);
        if (px != py) pre[px] = py, cnt--, res = e[i].w;
    }
}

int main()
{
    cin >> n >> k;
    for (int i = 0; i < n; i++) pre[i] = i;
    for (int i = 0; i < n; i++) cin >> p[i].X >> p[i].Y;
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            e[m++] = { i, j, get_dis(p[i], p[j]) };
    printf("%.2lf\n", kruskal());
    return 0;
}
```

三、AcWing 346. 走廊泼水节

【题目描述】

给定一棵N个节点的树，要求增加若干条边，把这棵树扩充为完全图，并满足图的唯一最小生成树仍然是这棵树。

求增加的边的权值总和最小是多少。

注意：树中的所有边权均为整数，且新加的所有边权也必须为整数。

【输入格式】

第一行包含整数t，表示共有t组测试数据。

对于每组测试数据，第一行包含整数N。

接下来$N-1$行，每行三个整数X,Y,Z，表示X节点与Y节点之间存在一条边，长度为Z。

【输出格式】

每组数据输出一个整数，表示权值总和最小值。

每个结果占一行。

【数据范围】

$1 \le N \le 6000$

$1 \le Z \le 100$

【输入样例】

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

【输出样例】

```
4
17
```

【分析】

这道题目说的很清楚，就是让我们将一个最小生成树的图，添加一些边，使得这张图成为一个完全图。但是我们这张图的最小生成树，必须还是原来那张图的最小生成树。也就是说两张图的最小生成树表示是一模一样的。

根据上面的信息，我们不难发现这道题目和最小生成树算法联系紧密，那么现在我们的主要问题就在于如何去构造最小生成树。我们可以考虑最小生成树算法中的`Kruskal`算法：

1. 首先将所有的边按照从小到大的顺序排序。此时我们保证了是最小生成树的完美生成法则。
2. 对于每一条边(x,y,w)而言，他们之间有某种关系：假如说x和y不在同一个连通块（集合）之中，也就是他们之间没有边相连，那么我们相连之后，现在这两个点各自所在的连通块（集合）都拥有了一个最短边，也就是(x,y,w)。

最小生成树是已经确定了，但是对于这原来两个连通块的其他点怎么办？

首先我们设S_x表示为x之前所在的连通块，那么S_y表示为y之前所在的连通块。因为我们不能破坏这个最小生成树，所以我们这原来的两个连通块中的点就必须有如下性质：

假如说点A属于S_x这个集合之中，点B属于S_y这个集合之中。那么点A与点B之间的距离必须要大于之前的w，否则就会破坏之前的最小生成树。所以说(A,B)之间的距离最小为$w+1$。

假如说我们知道S_x有p个元素，S_y有q个元素。那么将S_x与S_y连通块的所有点相连。显然这个两个连通块会增加$p*q-1$条边，然后每一条边的最小长度为$w+1$。所以我们会得出：$(w+1)\times(p*q-1)$为两个连通块成为完全图的最小代价。

【代码】

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

const int N = 6010;
int pre[N], cnt[N];
int n;

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

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

int main()
{
    int T;
    cin >> T;
    while (T--)
    {
        cin >> n;
        for (int i = 1; i <= n; i++) pre[i] = i, cnt[i] = 1;
        for (int i = 0; i < n - 1; i++) cin >> e[i].x >> e[i].y >> e[i].w;
        sort(e, e + n - 1);
        int res = 0;
        for (int i = 0; i < n - 1; i++)
        {
            int px = find(e[i].x), py = find(e[i].y);//由于本身就是一棵树，因此不需要判断是否成环
            res += (cnt[px] * cnt[py] - 1) * (e[i].w + 1);
```

```
            cnt[py] += cnt[px];
            pre[px] = py;
        }
        cout << res << endl;
    }
    return 0;
}
```

四、AcWing 1148. 秘密的牛奶运输（严格次小生成树）

【题目描述】

农夫约翰要把他的牛奶运输到各个销售点。

运输过程中，可以先把牛奶运输到一些销售点，再由这些销售点分别运输到其他销售点。

运输的总距离越小，运输的成本也就越低。

低成本的运输是农夫约翰所希望的。

不过，他并不想让他的竞争对手知道他具体的运输方案，所以他希望采用费用第二小的运输方案而不是最小的。

现在请你帮忙找到该运输方案。

注意：

- 如果两个方案至少有一条边不同，则我们认为是不同方案；
- 费用第二小的方案在数值上一定要**严格大于**费用最小的方案；
- 答案保证一定有解。

【输入格式】

第一行是两个整数N, M，表示销售点数和交通线路数；

接下来M行每行3个整数x, y, z，表示销售点x和销售点y之间存在线路，长度为z。

【输出格式】

输出费用第二小的运输方案的运输总距离。

【数据范围】

$1 \le N \le 500$

$1 \le M \le 10^4$

$1 \leq z \leq 10^9$

数据中可能包含重边。

【输入样例】

```
4 4
1 2 100
2 4 200
2 3 250
3 4 100
```

【输出样例】

```
450
```

【分析】

求次小生成树有两种方法：

次小生成树
定义：给一个带权的图，把图的所有生成树按权值从小到大排序，第二小的称为次小生成树。

方法1：先求最小生成树，再枚举删去最小生成树中的边求解。时间复杂度。 O(mlogm + nm)
方法2：先求最小生成树，然后依次枚举非树边，然后将该边加入树中，同时从树中去掉一条边，使得最终的图仍是一棵树。则一定可以求出次小生成树。

我们采用第二种方法，对于次小生成树，我们有如下定理：

设T为图G的一棵生成树，对于非树边a和树边b，插入边a，并删除边b的操作记为(+a, -b)。
如果T+a-b之后，仍然是一棵生成树，称(+a,-b)是T的一个可行交换。
称由T进行一次可行变换所得到的新的生成树集合称为T的邻集。
定理：次小生成树一定在最小生成树的邻集中。

因此我们能够归纳次小生成树的性质以及求解本题的思路：

- 次小生成树与最小生成树只有一条边不同。
- 我们可以先通过`Kruskal`求出最小生成树，建图并记录树的权值sum，时间复杂度为$O(mlogm)$，并标记在最小生成树中的边。
- 通过这个树预处理出某点u到其他点v所经过的路径中的最大边$dis1[u][v]$、严格次大边$dis2[u][v]$，时间复杂度为$O(n^2)$。
- 枚举所有未在树中的边(u, v, w)，看是否大于u到v的边中的最大值$dis1[u][v]$，若大于则可以替换这条边，替换后的结果为$sum - dis1[u][v] + w$；若不大于，根据最小生成树的定义可知该边一定等于$dis1[u][v]$，那么就替换严格次大边$dis2[u][v]$，替换后的结果

为$sum - dis2[u][v] + w$。我们初始化最大边与次大边为负无穷，如果w等于最大边且不存在次大边，$dis2[u][v]$为负无穷，因此式子计算后结果是正无穷，说明u, v间的路径长度全相等且等于w，无法更新，不会对结果造成影响。

【代码】

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

typedef long long LL;
const int N = 510, M = 10010;
int e[N << 1], ne[N << 1], d[N << 1], h[N], idx;
int dis1[N][N], dis2[N][N];//dis1/2表示最小生成树中两点间经过的最长/严格次长路径
int pre[N];
int n, m;

struct Edge
{
    int x, y, w;
    bool flag;//标记是否为最小生成树中的边
    bool operator< (const Edge& t) const
    {
        return w < t.w;
    }
}s[M];

int add(int u, int v, int w)
{
    e[idx] = v, d[idx] = w, ne[idx] = h[u], h[u] = idx++;
}

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


//当前节点为u,父节点为fa,走到该点的所有路径中的最大值maxd,最长距离数组dis1[]和
```

```
次长距离数组dis2[]
void dfs(int u, int fa, int maxd1, int maxd2, int dis1[], int dis2[])
{
    dis1[u] = maxd1, dis2[u] = maxd2;
    for (int i = h[u]; ~i; i = ne[i])
        if (e[i] != fa)
        {
            int t1 = maxd1, t2 = maxd2;//注意为了不破坏现场所以得开临时变量修改maxd1和maxd2
            if (d[i] > t1) t2 = t1, t1 = d[i];
            else if (d[i] < t1 && d[i] > t2) t2 = d[i];
            dfs(e[i], u, t1, t2, dis1, dis2);
        }
}

int main()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i++) pre[i] = i;
    memset(h, -1, sizeof h);
    for (int i = 0; i < m; i++) cin >> s[i].x >> s[i].y >> s[i].w;
    sort(s, s + m);
    LL sum = 0, res = 1e18;//sum表示最小生成树的总权值
    for (int i = 0; i < m; i++)
    {
        int px = find(s[i].x), py = find(s[i].y);
        if (px != py)
        {
            pre[px] = py;
            sum += s[i].w;
            add(s[i].x, s[i].y, s[i].w), add(s[i].y, s[i].x, s[i].w);
            s[i].flag = true;
        }
    }
    //枚举每个起点,求出在最小生成树中该点到其他点经过的最长路径和严格次长路径
    for (int i = 1; i <= n; i++) dfs(i, -1, -1e9, -1e9, dis1[i], dis2[i]);
    for (int i = 0; i < m; i++)
        if (!s[i].flag)//枚举每条非树边
        {
            int x = s[i].x, y = s[i].y, w = s[i].w;

            //如果该边严格大于最小生成树中这两点间的最长路径,则用这条边替换最长
```

```
路径
                if (w > dis1[x][y]) res = min(res, sum - dis1[x][y] + w);
                //如果该边等于最长路径,则用这条边替换严格次长路径
                else res = min(res, sum - dis2[x][y] + w);
            }
    cout << res << endl;
    return 0;
}
```