算法分析与设计 E5-D

题目描述

题目描述

为了让金人巷再次伟大，你被指派了一个新的物流运输分配问题，现在有 n 个物流的寄送点和 n 个物流的目的地，你需要为每个目的地分配一个寄送点，且每个寄送点只能被分配给唯一一个目的地，即寄送点和目的地是一一对应的关系。

现在将 n 个寄送点的坐标记为 $(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$，$n$ 个目的地的坐标记为 $(x'_1,y'_1),(x'_2,y'_2),\cdots,(x'_n,y'_n)$。若指定寄送点 (x_i,y_i) 分配给目的地 (x'_j,y'_j)，则这条物流线的花费为两点间的曼哈顿距离 $|x_i-x'_j|+|y_i-y'_j|$。

你需要将寄送点和目的地两两匹配，得到 n 条物流线，并且让 n 条物流线的**总花费最小**，请你求出最小的总花费。

形式化地说，你需要得出 $1,2,\cdots,n$ 的一个排序 $p_1,p_2,\cdots,p_n$，使得

$$S=\sum_{i=1}^{n}|x_i-x'_{p_i}|+|y_i-y'_{p_i}|$$

最小，并计算出 S 的值。

分析题目发现这是一个带权二分图匹配问题，不能直接用匈牙利算法，这里介绍一个求解二分图最大权完美匹配的算法——KM算法。

算法原理

前备知识

可行顶标

给每个节点 i 分配一个权值 $l(i)$，对于所有边 (u,v) 满足 $w(u,v)\le l(u)+l(v)$。

相等子图

在一组可行顶标下原图的生成子图，包含所有点但只包含满足 $w(u,v)=l(u)+l(v)$ 的边 (u,v)。

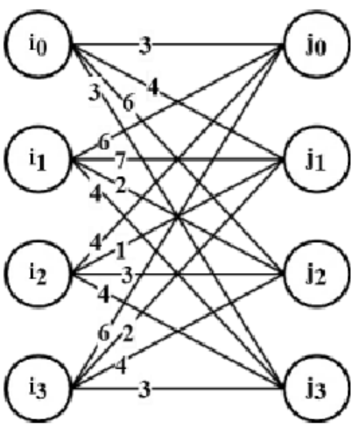


Figure 1. Example of a graph representation of an assignment problem

定理 1：对于某组可行顶标，如果其相等子图存在完美匹配，那么，该匹配就是原二分图的最大权完美匹配。

证明 1.

考虑原二分图任意一组完美匹配 M，其边权和为

$val(M) = \sum_{(u,v)\in M} w(u,v) \le \sum_{(u,v)\in M} l(u) + l(v) \le \sum_{i=1}^{n} l(i)$

任意一组可行顶标的相等子图的完美匹配 M' 的边权和

$val(M') = \sum_{(u,v)\in M} l(u) + l(v) = \sum_{i=1}^{n} l(i)$

即任意一组完美匹配的边权和都不会大于 $val(M')$，那个 M' 就是最大权匹配。

题目分析

有了定理 1，我们的目标就是透过不断的调整可行顶标，使得相等子图是完美匹配。

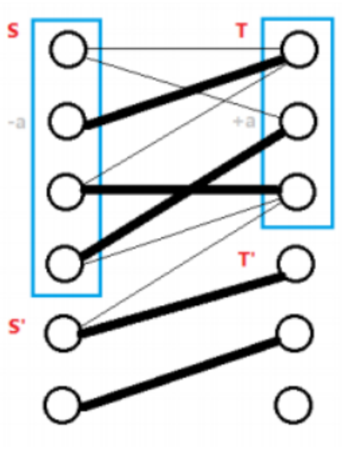
因为两边点数相等，假设点数为 n，$lx(i)$ 表示左边第 i 个点的顶标，$ly(i)$ 表示右边第 i 个点的顶标，$w(u,v)$ 表示左边第 u 个点和右边第 v 个点之间的权重。

首先初始化一组可行顶标，例如

$lx(i) = \max_{1\le j\le n}\{w(i,j)\},\ ly(i) = 0$

然后选一个未匹配点，如同最大匹配一样求增广路。找到增广路就增广，否则，会得到一个交错树。

令 S，T 表示二分图左边右边在交错树中的点，S'，T' 表示不在交错树中的点。



在相等子图中：

- $S-T'$ 的边不存在，否则交错树会增长。
- $S'-T$ 一定是非匹配边，否则他就属于 S。

假设给 S 中的顶标 $-a$，给 T 中的顶标 $+a$，可以发现

- $S-T$ 边依然存在相等子图中。
- $S'-T'$ 没变化。
- $S-T'$ 中的 $lx+ly$ 有所减少，可能加入相等子图。
- $S'-T$ 中的 $lx+ly$ 会增加，所以不可能加入相等子图。

所以这个 a 值的选择，显然得是 $S-T'$ 当中最小的边权，

$a=\min\{lx(u)+ly(v)-w(u,v)|u\in S,v\in T'\}$。

时间复杂度

交错树新增一个点进入S的时候需要$O(n)$更新$slack(v)$。修改顶标需要$O(n)$给每个$slack(v)$减去a。只要交错树找到一个未匹配点，就找到增广路。

一开始枚举n个点找增广路，为了找增广路需要延伸n次交错树，每次延伸需要n次维护，共$O(n^3)$。

参考代码

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstring>
#define ll long long
#define ull unsigned long long
#define pr pair<ll, ll>
#define MAX 250
#define INF 0x3f3f3f3f3f

using namespace std;

//KM算法模板
int T, n;//n为点的数量
ll x, y;

pr from[MAX];//记录左点的坐标
pr to[MAX];//记录右点的坐标
```

```
int match[MAX];//记录右点匹配的左点
int va[MAX], vb[MAX];//标记点是否在交替路中
ll w[MAX][MAX];//权值矩阵
ll la[MAX], lb[MAX];//左顶标和右顶标
ll slack[MAX];//松弛数组

void init() {
    //初始化权值矩阵
    memset(w, 0xbf, sizeof(w));
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++) {
            w[i][j] = -abs(from[i].first - to[j].first) -
abs(from[i].second - to[j].second);
        }
    }
}

bool dfs(int x) {
    va[x] = 1;//标记x在交替路中
    for (int y = 1; y <= n; y++)
    {
        if (vb[y]) continue;
        ll t = la[x] + lb[y] - w[x][y];
        if (t == 0) {//相等子图
            vb[y] = 1;//标记y在交替路中
            if (match[y] == -1 || dfs(match[y])) {
                match[y] = x;//找到增广路，更新匹配
                return true;
            }
        }
        else slack[y] = min(slack[y], t);//不在相等子图中，更新松弛数组
    }
    return false;
}

ll KM() {
    memset(match, -1, sizeof(match));//初始化匹配数组
    memset(lb, 0, sizeof(lb));//初始化右顶标为0
    for (int i = 1; i <= n; i++) //初始化左顶标为与之相连的最大权值
    {
        la[i] = -INF;
        for (int j = 1; j <= n; j++)
            la[i] = max(la[i], w[i][j]);
    }
    for (int i = 1; i <= n; i++)
    {
        memset(slack, 0x3f, sizeof(slack));//初始化松弛数组
```

```
        while (true)
        {
            memset(va, 0, sizeof(va));
            memset(vb, 0, sizeof(vb));
            if (dfs(i)) break;//找到增广路，退出
            ll d = INF;
            for (int j = 1; j <= n; j++)
                if (!vb[j]) d = min(d, slack[j]);
            for (int j = 1; j <= n; j++) //更新顶标
            {
                if (va[j]) la[j] -= d;//S中的点，左顶标减d
                if (vb[j]) lb[j] += d;//T中的点，右顶标加d
            }
        }
    }
    ll ans = 0;
    for (int i = 1; i <= n; i++)
        ans += w[match[i]][i];
    return ans;
}

int main()
{
    cin >> T;
    while (T--)
    {
        cin >> n;
        for (int i = 1; i <= n; i++)
        {
            cin >> x >> y;
            from[i] = make_pair(x, y);
        }
        for (int i = 1; i <= n; i++)
        {
            cin >> x >> y;
            to[i] = make_pair(x, y);
        }
        init();
        cout << -KM() << endl;
    }
    return 0;
}
```