1277：方格取数

题目描述

设有N×N的方格图，我们在其中的某些方格中填入正整数，而其它的方格中则放入数字0。如下图所示：

A

0	0	0	0	0	0	0	0
0	0	13	0	0	6	0	0
0	0	0	0	7	0	0	0
0	0	0	14	0	0	0	0
0	21	0	0	0	4	0	0
0	0	15	0	0	0	0	0
0	14	0	0	0	0	0	0
0	0	0	0	0	0	0	0

B

某人从图中的左上角A出发，可以向下行走，也可以向右行走，直到到达右下角的B点。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字0）。

此人从A点到B点共走了两次，试找出两条这样的路径，使得取得的数字和为最大。

输入

第一行为一个整数N（N≤10），表示N×N的方格图。

接下来的每行有三个整数，第一个为行号数，第二个为列号数，第三个为在该行、该列上所放的数。一行“0 0 0”表示结束。

输出

第一个整数，表示两条路径上取得的最大的和。

输入样例

```
8
2 3 13
2 6 6
3 5 7
4 4 14
5 2 21
5 6 4
6 3 15
7 2 14
0 0 0
```

输出样例

67

解析

读完这个题，大家的第一反应一定是来回走两遍，每一次都取最大的，很遗憾，这种思路是错误的，因为你会漏掉一部分数据。

如下所示：

原始数据

1	1	2	1
1	1	3	1

最大行走路径

1	2	3	4
2	3	6	7

黄色为第一次行走的最大值路径，结果为19。第二次的最大值是5，两次加起来只有24。但是我们一眼可以看出，两次行走的最大值应该是全部数字之和：28！

换句话说，本题求解的是两个人同时移动的最大值，是一道双人DP问题，那么该如何求解呢？

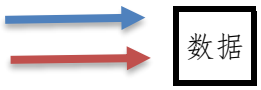
先来思考双人DP和单人DP有什么区别？

答案是：某个点的归属权。

我们设dp[x1][y1][x2][y2]这样一个四维数组来表示两个人分别在(x1,y1)和(x2,y2)时的最大值，设value为当前目标点的数据，用map[x][y]记录原始的数组。

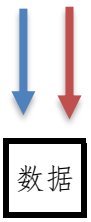
当两个人没有走到相同点时，那么value就等于，map[x1][y1] + map[x2][y2];

但是，当两个人行走到同一个点时，那么value就等于map[x1][y1];



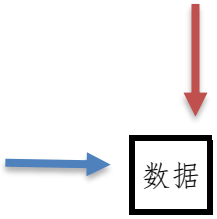
情况1：两人都从右边来。

```
dp[x1][y1][x2][y2] =
   max(dp[x1-1][y1][x2-1][y2] + value, dp[x1][y1][x2][y2])
```



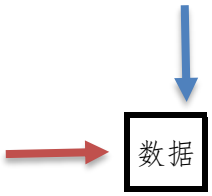
情况2：两人都从上边来。

```
dp[x1][y1][x2][y2] =
   max(dp[x1][y1-1][x2][y2-1] + value, dp[x1][y1][x2][y2])
```



情况3：A从左，B从上

```
dp[x1][y1][x2][y2] =
   max(dp[x1-1][y1][x2][y2-1] + value, dp[x1][y1][x2][y2])
```



情况4：A从上，B从左

```
dp[x1][y1][x2][y2] =
   max(dp[x1][y1-1][x2-1][y2] + value, dp[x1][y1][x2][y2])
```

编码

通过上面的分析，我们可以很容易的写出暴力代码。

```
#include <bits/stdc++.h>

using namespace std;

int mapData[11][11];  //最大行列值为10
int dp[11][11][11][11];  //两个单位在某个点的最大和

int main(int argc, char **argv) {
    int n;
    int x, y, v;  //临时列行和数值变量
   cin >> n;
    while (true) {
        //注意行列顺序
      cin >> y >> x >> v;
        //说明当前位置有数值
```

```
            if (x || y || v) {
                mapData[x][y] = v;
            } else {
                //不存在数据了，跳出循环
                break;
            }
        }

        int value;  //当前点的数值;

        for (int x1 = 1; x1 <= n; ++x1) {
            for (int y1 = 1; y1 <= n; ++y1) {
                for (int x2 = 1; x2 <= n; ++x2) {
                    for (int y2 = 1; y2 <= n; ++y2) {
                        //两个人站在了同一个格子上
                        if (x1 == x2 && y1 == y2) {
                            value = mapData[x1][y1];
                        } else {
                            value = mapData[x1][y1] + mapData[x2][y2];
                        }
                        //将数据进行取地址，进行简化
                        int &num = dp[x1][y1][x2][y2];
                        //模拟两个人都来自左边
                        num = max(dp[x1 - 1][y1][x2 - 1][y2] + value, num);
                        //模拟两个人都来自上边
                        num = max(dp[x1][y1 - 1][x2][y2 - 1] + value, num);
                        //模拟A来自左边，B来自上边
                        num = max(dp[x1 - 1][y1][x2][y2 - 1] + value, num);
                        //模拟B来自左边，A来自上边
                        num = max(dp[x1][y1 - 1][x2 - 1][y2] + value, num);
                    }
                }
            }
        }

        cout << dp[n][n][n][n];

        return 0;
    }
```

优化

对于本题，是否可以进行进一步的优化呢？答案是可以的。

我们将原本的四维数组修改成为三维数组，新的数组为dp[k][x1][x2]，其中k代表的是两个人的y坐标之和。那么则有以下变化：

两人同时来自上边：num = max(dp[k - 1][x1][x2] + value, num);

这里详细解释一下，因为k=x1+y1=x2+y2，又因为x1,x2不变，则意味着y1和y2分别进行了-1操作。

继续推导，A来自左，B来自上：num = max(dp[k - 1][x1 - 1][x2] + value, num);

A来自上，B来自左：num = max(dp[k - 1][x1][x2 - 1] + value, num);

两人同时来自左：num = max(dp[k - 1][x1-1][x2 - 1] + value, num);

编码

```
#include <bits/stdc++.h>

using namespace std;

int mapData[11][11];  //最大行列值为10
int dp[21][11][11];  //两个单位在某个点的最大和

int main(int argc, char **argv) {
    int n;
    int x, y, v;  //临时列行和数值变量
   cin >> n;
    while (true) {
        //注意行列顺序
      cin >> y >> x >> v;
        //说明当前位置有数值
      if (x || y || v) {
            mapData[x][y] = v;
        } else {
            //不存在数据了，跳出循环
          break;
        }
    }

    int value;  //当前点的数值;

    //遍历全部的k值，最小的k值为2
    for (int k = 2; k <= 2 * n; ++k) {
        for (int x1 = 1; x1 <= n; ++x1) {
            for (int x2 = 1; x2 <= n; ++x2) {
                int y1 = k - x1, y2 = k - x2;
                if (y1 >= 1 && y1 <= n && y2 >= 1 && y2 <= n) {
```

```
                    if (x1 == x2) {
                        value = mapData[x1][y1];
                    } else {
                        value = mapData[x1][y1] + mapData[x2][y2];
                    }

                    //将数据进行取地址，进行简化
                int &num = dp[k][x1][x2];
                    //模拟两个人都来自左边
                num = max(dp[k - 1][x1 - 1][x2 - 1] + value, num);
                    //模拟两个人都来自上边
                num = max(dp[k - 1][x1][x2] + value, num);
                    //模拟A来自左边，B来自上边
                num = max(dp[k - 1][x1 - 1][x2] + value, num);
                    //模拟B来自左边，A来自上边
                num = max(dp[k - 1][x1][x2 - 1] + value, num);
                }
            }
        }
    }
    cout << dp[2 * n][n][n];

    return 0;
}
```