南川算法笔记之详解多源bfs问题：以最优配餐问题为例

> 本文感谢大佬“醉卧”的思路启发，没有他我可能将耗费更多时间在一些对解题帮助不大的琐碎细节上。

题目

3205. 最优配餐

题目 | 提交记录 | 讨论 | 题解 | 视频讲解

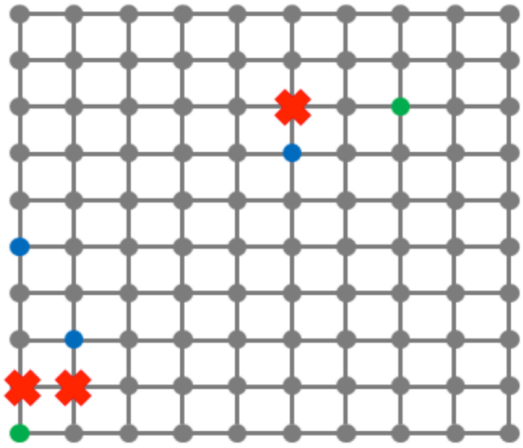
栋栋最近开了一家餐饮连锁店，提供外卖服务。

随着连锁店越来越多，怎么合理的给客户送餐成为了一个急需解决的问题。

栋栋的连锁店所在的区域可以看成是一个 $n \times n$ 的方格图（如下图所示），方格的格点上的位置上可能包含栋栋的分店（绿色标注）或者客户（蓝色标注），有一些格点是不能经过的（红色标注）。

方格图中的线表示可以行走的道路，相邻两个格点的距离为 1。

栋栋要送餐必须走可以行走的道路，而且不能经过红色标注的点。



送餐的主要成本体现在路上所花的时间，每一份餐每走一个单位的距离需要花费 1 块钱。

每个客户的需求都可以由栋栋的任意分店配送，每个分店没有配送总量的限制。

现在你得到了栋栋的客户的需求，请问在最优的送餐方式下，送这些餐需要花费多大的成本。

输入格式

输入的第一行包含四个整数 n, m, k, d，分别表示方格图的大小、栋栋的分店数量、客户的数量，以及不能经过的点的数量。

接下来 m 行，每行两个整数 x_i, y_i，表示栋栋的一个分店在方格图中的横坐标和纵坐标。

接下来 k 行，每行三个整数 x_i, y_i, c_i，分别表示每个客户在方格图中的横坐标、纵坐标和订餐的量。（注意，可能有多个客户在方格图中的同一个位置）

接下来 d 行，每行两个整数，分别表示每个不能经过的点的横坐标和纵坐标。

输出格式

输出一个整数，表示最优送餐方式下所需要花费的成本。

数据范围

前 30% 的评测用例满足：$1 \le n \le 20$。
前 60% 的评测用例满足：$1 \le n \le 100$。
所有评测用例都满足：$1 \le n \le 1000, 1 \le m, k, d \le n^2$。
可能有多个客户在同一个格点上。
每个客户的订餐量不超过 1000，每个客户所需要的餐都能被送到。

输入样例：

```
10 2 3 3
1 1
8 8
1 5 1
2 3 3
6 7 2
1 2
2 2
6 8
```

输出样例：

```
29
```

解读

障碍的意义

由于障碍的存在，两点之间的曼哈顿距离并不等价于其实际距离。

否则，本题可直接对客户坐标与分店坐标写两层for循环，得出答案。

因为障碍的存在，所以我们需要使用bfs等最短路算法，得到每个客户到最近分店的实际距离。

重复客户的处理

由于分店没有配送限制，所以问题可简化为对任一客户，找出与它最近的分店距离。

言外之意，即便题中陈述“可能有客户在同一个格点上”，也可以将它们视作同一处理，直接累加它们的需求量即可。

基于此，只需使用静态数组存储客户的需求，否则，最好需要使用列表，以区别同一格点的不同客户。

实际编码上， 可以基于原题中的 k 一趟遍历生成新的 k2，k2 是所有不重复的客户数，且基于 k2 的数组内存储的是该地址客户的需求总和。

这样，只需要一趟遍历k2，无需考虑重复问题，就可以得到最终答案。

算法设计（一）：以分店为中心，做m遍bfs

听起来很有道理，但其实是不可行的。

因为不是根据分店能不能到达某个客户而执行操作的，而是根据客户最近的分店。

所以“**客户才是中心**”。

算法设计（二）：以客户为中心，做k遍bfs

如果我们把求解每个客户到周围分店的最短距离，视作一个独立的单位，或者抽象成一个独立的函数，并记录该距离值。那么在接下来求解其它客户到周围分店的最短距离时，该值或许可被利用。

但是，其利用价值是有限的。

假设已经知道A客户最近的分店距离为5。

现在对B客户再次使用bfs，由于A、B客户的地址并不相同，这导致基于A、B客户为中心的周围单元格实际距离的信息，是没有直接关系的，除非我们像floyd算法那样计算出每两个单元格之间的最短距离。

因此，当遍历B客户时，假设最近的分店都很远，它bfs到了A客户的位置，这个时候，我们能够简单地将B与A的距离和A到目标分店的距离相加吗?

答案是显然不能，因为B到A的这条路径，只是B到A实际距离内的其中一条路径，我们不能保证是否存在另一个分店到B的距离更短，其路径上实际不通过A。

于是，bfs将继续下去，直到在从A的路径上抵达A的最近分店前B寻找到了真正最近的分店，或者最终确定B的最近分店就是A的最近分店，B的最近分店的距离恰好等于B到A的最近距离加上A到A的最近分店的距离。

所以，如果我们以客户为基本单位，就不得不做k遍bfs，算法的复杂度将达到 $k \times n^2$，非常高。

实际测试，只通过4个测例，第五个开始超时了。

算法设计（三）：以分店集体为中心，做一遍bfs

虽然以分店个体为中心做bfs的思路不可行，因为我们只得到了某个分店距离其他客户的距离，但是这个距离因为缺少相关的参照对象，所以无法作为选择派送的依据。

但是!

如果以分店的集体为中心，情况就不一样了！

回忆一下我们的prim算法，每次把一个新的点加入到集合 S 中，该算法每一步能够确定到达新加入的点的距离是当前也是全局最短的。这是因为已经考虑了所有可以到达该点最短距离的可能情形。

现在，把prim算法中，一个个加入的点集 S 初始化为本题中的所有店铺的集合。

接着这个集合，集体向周围扩展1步、2步……n步。

易知，只要我们使用一个 vis 数组记录已经发现的格点，由于格点与格点距离之间的等权性（即不存在绕路），所以到达每次扩展的 F(k) 个结点的最短距离就等于 k。

并且这个k的含义即是，从所有的店铺中任选一个店铺，能到达该地点的最短距离，这，正符合题中配送的定义。

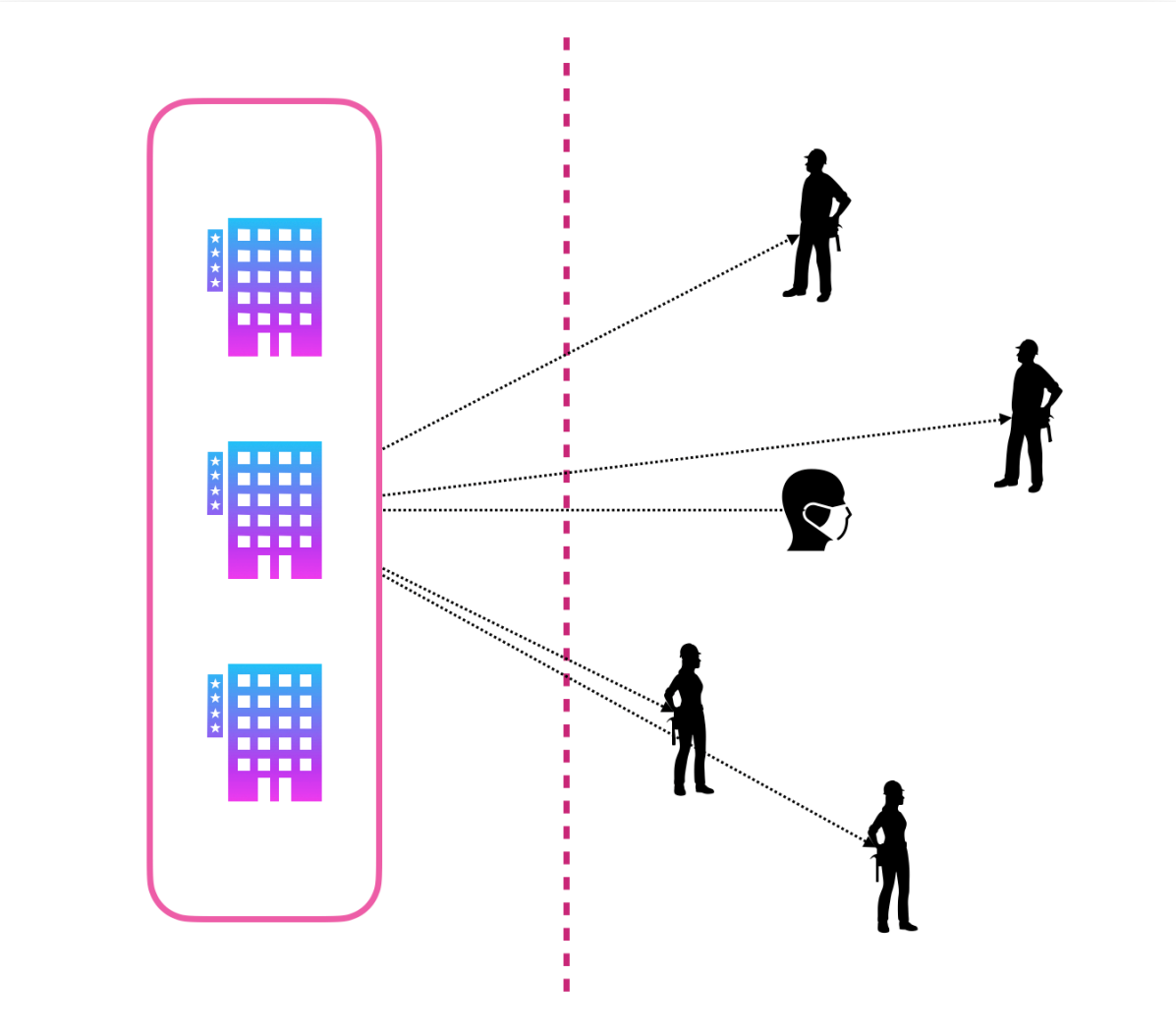
只要我们在扩展过程中，一一覆盖客户地址就可以。

我们可以使用一个计数变量 k2 表示所有客户的不重复地址，然后逐个减一，并累加到答案，直到为0。

算法设计（四）：以客户集体为中心，做一遍bfs

与之对应的，如果我们以客户集体为中心，则又不可行了。

说到底，是因为，这道题其实是一个 server-clients 的服务模型，所以可以把分店集体作为中心，但是客户个体却是独立的。如图。



代码实现（基于分店集体，向客户个体的一次bfs算法）

版本一，使用struct标记分店、客户、障碍与空格

```
//
// Created by 南川 on 2021/2/18.
//

#include "cstdio"
#include "queue"
#include "tuple"

using namespace std;


const int N = 1000;
int n, m, k2, d, r1, r2, r3;

enum Type{Default, Store, Customer, Obstacle};
```

```
struct Grid{
    int v = 0;
    Type t = Default;
} grid[N+2][N+2];
int stores[N * N + 1][2];

queue<pair<int, int>> Q;
int vis[N+2][N+2];
int offset[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

void readCase()
{
    int k;
    scanf("%d %d %d %d", &n, &m, &k, &d);
    for(int i=0; i<m; ++i)
    {
        scanf("%d %d", & r1, & r2);
        grid[r1][r2].t = Store;
        stores[i][0] = r1;
        stores[i][1] = r2;
    }

    for(int i=0; i<k; ++i)
    {
        scanf("%d %d %d", & r1, & r2, & r3);
        if(grid[r1][r2].t == Default) ++k2;
        grid[r1][r2].t = Customer;
        grid[r1][r2].v += r3;       // multi customers hook
    }

    for(int i=0; i<d; ++i)
    {
        scanf("%d %d", & r1, & r2);
        grid[r1][r2].t = Obstacle;
    }
}

long long bfsSolveCustomers()
{
    long long ans = 0;
    for(int i=0; i<m; ++i)
        Q.push({stores[i][0], stores[i][1]});
    int dis = 0;
    while (!Q.empty())
    {
        ++dis;
        const int size = Q.size();
        for(int i=0; i<size; ++i)
        {
            auto & cur = Q.front();
//            printf("cur: <%d, %d>\n", cur.first, cur.second);
            for(auto & j : offset)
```

```
            {
                int x = cur.first + j[0], y = cur.second + j[1];
                if(x < 1 || x > n || y < 1 || y > n) continue;
                if(vis[x][y] || grid[x][y].t == Obstacle || grid[x][y].t == Store)
continue;
                ++vis[x][y];
                Q.push({x, y});
                if(grid[x][y].t == Customer) {
                    ans += (long long)dis * grid[x][y].v;
//                    printf("dis: %d, x: %d, y: %d, v: %d, ans: %lld\n", dis, x, y,
grid[x][y].v, ans);
                    if(--k2 == 0) return ans;
                }
            }
            Q.pop();
        }
    }
}

int main()
{
    //freopen("/Users/mark/MarkLearningCPP/Acwing/3205. 最优配餐/case2.txt", "r",
stdin);
    readCase();
    printf("%lld", bfsSolveCustomers());
}
```

版本二，更简洁的优化版本，直接预处理分店，加入队列，非客户直接标记为已访问

```
//
// Created by 南川 on 2021/2/18.
//

#include "cstdio"
#include "queue"
#include "tuple"
using namespace std;

const int N = 1000;
int n, m, k, k2, d, r1, r2, r3;
long long ans;

int customers[N+2][N+2];
queue<pair<int, int>> Q;
int vis[N+2][N+2];
const int offset[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

void readCase()
```

```
{
    scanf("%d %d %d %d", &n, &m, &k, &d);
    for(int i=0; i<m; ++i) {
        scanf("%d %d", & r1, & r2);
        ++vis[r1][r2];
        Q.push({r1, r2});
    }
    for(int i=0; i<k; ++i) {
        scanf("%d %d %d", &r1, &r2, &r3);
        if (!customers[r1][r2]) ++k2;
        customers[r1][r2] += r3;
    }
    for(int i=0; i<d; ++i) {
        scanf("%d %d", & r1, & r2);
        ++vis[r1][r2];
    }
}

void bfsSolveCustomers(int depth)
{
    int size = Q.size();
    while (size--)
    {
        auto & cur = Q.front(); Q.pop();
        for(auto & j : offset) {
            int x = cur.first + j[0], y = cur.second + j[1];
            if(x < 1 || x > n || y < 1 || y > n || vis[x][y]) continue;
            ++vis[x][y]; Q.push({x, y});
            if(customers[x][y]) {
                ans += depth * customers[x][y];
                if(--k2 == 0) return;
            }
        }
    }
    bfsSolveCustomers(++depth);
}

int main()
{
    //freopen("/Users/mark/MarkLearningCPP/Acwing/3205. 最优配餐/case2.txt", "r",
stdin);
    readCase();
    bfsSolveCustomers(1);
    printf("%lld", ans);
}
```