A时间限制：C/C++ 1秒，其他语言 2秒  
空间限制：C/C++ 262144K，其他语言 524288K  
64bit IO Format: %lld

**题目描述**

    小美想要在电视上看电影，我们知道在电视上搜索电影可以通过搜索电影名字首字母缩写得到，通过首字母搜索电影的界面由一个九宫格组成，如下图：

|  |  |  |
| --- | --- | --- |
| @!: | ABC | DEF |
| GHI | JKL | MNO |
| PQRS | TUV | WXYZ |

    光标初始在这个九宫格的左上方，也就是在 “@!:”的位置，每次小美想要输入一个字母，需要通过不断地按上下左右四个方向键(并且只能按方向键），把光标从当前所在的格子移动到目标的格子(也就是待输入的字母所在的格子)，然后在目标的格子上通过其他的按键来输入字母。小美觉得频繁地按方向键是十分烦人的事情，所以她想设计一种移动光标方案使得方向键按的次数最少。问最少要几次?  
    小美想看 T 部电影，所以她会问你 T 个电影名字的缩写分别需要多少次输入。  
    注意在一个电影名字输入完以后，光标会回到左上角，期间按的方向键不会计入答案。

**输入描述:**

第一行一个T(T ≤ 10)，表示小美想看的电影数。  
接下来 T 行，每行一个长度不超过100,000的字符串，表示一部电影名字的缩写，保证缩写的每个字符都是大写英文字母。

**输出描述:**

对于每个电影名字缩写，输出输入这个名字的最小按方向键的次数。

**示例1**

输入

2

AA

AT

输出

1

3

#include <bits/stdc++.h>

**using** **namespace** std;

**int** T,len;

**char** s[1000005];

**struct** po{

**int** x,y;

**int** **operator** -(**const** po &a)**const**{

**return** abs(x-a.x)+abs(y-a.y);

}

};

po getpo(**char** w){

**if**('A'<=w&&w<='C') **return** {1,2};

**if**('D'<=w&&w<='F') **return** {1,3};

**if**('G'<=w&&w<='I') **return** {2,1};

**if**('J'<=w&&w<='L') **return** {2,2};

**if**('M'<=w&&w<='O') **return** {2,3};

**if**('P'<=w&&w<='S') **return** {3,1};

**if**('T'<=w&&w<='V') **return** {3,2};

**if**('W'<=w&&w<='Z') **return** {3,3};

}

**void** Main(){

scanf("%s",s+1);

len=strlen(s+1);

po cur={1,1};

**int** ans=0;

**for**(**int** i=1;i<=len;i++){

**char** w=s[i];

po now=getpo(w);

ans+=cur-now;

cur=now;

}

printf("%d**\n**",ans);

}

**int** main(){

**int** T;

scanf("%d",&T);

**while**(T--) Main();

}

B

时间限制：C/C++ 1秒，其他语言 2秒  
空间限制：C/C++ 262144K，其他语言 524288K  
64bit IO Format: %lld

**题目描述**

    有一个1\*n的棋盘，上面有若干个棋子，一个格子上可能有多个棋子。

    你每次操作是先选择一个棋子，然后选择以下两个操作中的一个：

    (1) 若该棋子不在 (1,1)，让这个棋子往左走一格，即从 (1,x) 走到 (1,x-1)；

    (2) 若该棋子不在 (1,n)，且这个棋子曾经到达过(1,1)，让这个格子往右走一格，即从 (1,x) 走到 (1,x+1)。

    给定一开始每个格子上有几个棋子，再给定目标局面每个格子上需要几个棋子，求最少需要多少次操作。

**输入描述:**

第一行一个正整数n表示棋盘大小。  
第二行n个非负整数a\_1, a\_2, …, a\_n 表示一开始 (1,i) 上有几个棋子。  
第三行n个非负整数b\_1, b\_2, …, b\_n 表示目标局面里 (1,i) 上有几个棋子。

保证 1 ≤ n ≤ 100,000，

**输出描述:**

输出一个非负整数，表示最少需要几次操作。

**示例1**

输入

5

0 0 1 1 0

1 0 0 0 1

输出

9

说明

先把(1,3)上的棋子走到(1,1)，花费了2次操作。  
然后把(1,4)上的棋子走到(1,1)，再往右走到(1,5)，花费了3+4=7次操作。  
所以一共花了9次操作。

#include<stdio.h>

#include<cstring>

#include<cstdlib>

#include<algorithm>

#include<vector>

#include<map>

#include<set>

#include<cmath>

#include<iostream>

#include<assert.h>

#include<queue>

#include<string>

#define rep(i,j,k) for(int i=(int)j;i<=(int)k;i++)

#define per(i,j,k) for(int i=(int)j;i>=(int)k;i--)

#define pii pair<int,int>

#define fi first

#define se second

#define pb push\_back

**using** **namespace** std;

**typedef** **long** **long** LL;

**const** **int** N=110000;

**int** a[N],b[N];

**int** n;

**int** main(){

scanf("%d",&n);

assert(1<=n&&n<=100000);

rep(i,1,n)scanf("%d",&a[i]);

rep(i,1,n)scanf("%d",&b[i]);

LL suma=0;

LL sumb=0;

rep(i,1,n)suma+=a[i];

rep(i,1,n)sumb+=b[i];

assert(suma==sumb);

assert(0<=suma&&suma<=1000000000);

**int** aft=0;

LL ans=0;

rep(i,1,n)ans+=(a[i]+b[i])\*1ll\*(i-1);

per(i,n,1){

aft+=a[i];

ans-=2\*(i-1)\*1ll\*min(b[i],aft);

aft-=min(b[i],aft);

}

printf("%lld**\n**",ans);

**return** 0;

}

# C题目描述

    在点点和评评的世界里有一个城市，城市是一个树形结构，由n个节点组成，有n-1条双向边把这些节点连在了一起（即在任意两个节点AB之间都存在从A到B的路径）。每条边都有一个边权，表示通过这条边需要的时间。  
    现在点点和评评想要从他们所在的S点到T点参加会议，他们想要在过程中体会旅途的乐趣，即好好欣赏每条边上的风景。详细地说，对于每条边，都有一个值l，l表示在从S点到T点的过程中，最少需要经过这条边的次数。因为会期将至，所以点点和评评想要在看够风景的情况下，花费尽可能少的时间。  
    点点和评评一共想询问m组这样的S和T，每组询问单独考虑，对于某一组特定的询问，他们想知道最少需要花费的时间。

**输入描述:**

第一行一个整数n(1 ≤ n ≤ 100,000)表示点数。  
接下来n-1行，每行四个整数x, y, z, l(1 ≤ x, y ≤ n, 1 ≤ z, l ≤ 1,000,000,000)，表示从x到y之间有一条边权为z的边，并且这条边至少要经过l次。  
数据保证没有重边和自环。  
接下来一个整数m(1 ≤ m ≤ 100,000)表示询问组数。  
接下来m行，每行两个整数S, T(1 ≤ S, T ≤ n)表示一组询问。

**输出描述:**

输出m行，每行一个整数表示答案，因为答案可能很大，请输出答案 mod 1,000,000,007的值。

**示例1**

输入

3

1 2 10 1

2 3 20 2

2

1 3

3 2

输出

70

80

说明

路径为：  
1->2->3->2->3  
3->2->1->2->3->2

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <ctime>

#include <cmath>

#include <iostream>

#include <algorithm>

**using** **namespace** std;

**const** **int** P = 1000000007;

**struct** node {

node \*next;

**int** where, cost, bound;

} \*first[100001], a[200001];

**int** n, m, dist[100001], l, c[100001], f[100001][21], g[100001], v[100001][21], pa[100001][21];

**inline** **void** makelist(**int** x, **int** y, **int** z, **int** k) {

a[++l].where = y;

a[l].cost = z;

a[l].bound = k;

a[l].next = first[x];

first[x] = &a[l];

}

**int** lca(**int** x, **int** y) {

**if** (dist[x] < dist[y])

swap(x, y);

**int** jump = dist[x] - dist[y];

**for** (**int** i = 0; jump; jump >>= 1, ++i)

**if** (jump & 1)

x = pa[x][i];

**if** (x == y)

**return** x;

**for** (**int** i = 20; i >= 0; --i)

**if** (pa[x][i] != pa[y][i])

x = pa[x][i], y = pa[y][i];

**return** pa[x][0];

}

**int** calc(**int** x, **int** y) {

**int** jump = dist[x] - dist[y], will = 0;

**for** (**int** i = 20; i >= 0; --i)

**if** (jump >= (1 << i))

jump -= 1 << i,

will += v[x][i], will %= P,

will -= f[x][i], will += P, will %= P,

x = pa[x][i];

**return** will;

}

**int** main() {

scanf("%d", &n);

memset(first, 0, **sizeof**(first));

l = 0;

**int** base = 0;

**for** (**int** i = 1; i < n; i++) {

**int** x, y, z, k;

scanf("%d%d%d%d", &x, &y, &z, &k);

makelist(x, y, z, k);

makelist(y, x, z, k);

base += 1LL \* (k + (k & 1)) \* z % P;

base %= P;

}

memset(pa, 0, **sizeof**(pa));

memset(dist, 0, **sizeof**(dist));

memset(v, 0, **sizeof**(v));

memset(f, 0, **sizeof**(f));

c[1] = 1; dist[1] = 1;

**for** (**int** k = 1, l = 1; l <= k; l++) {

**int** m = c[l];

**for** (node \*x = first[m]; x; x = x->next)

**if** (!dist[x->where])

dist[x->where] = dist[m] + 1, c[++k] = x->where,

pa[x->where][0] = m;

}

**for** (**int** i = n; i >= 1; i--) {

**int** m = c[i]; f[m][0] = 0;

**for** (node \*x = first[m]; x; x = x->next)

**if** (dist[x->where] == dist[m] + 1)

f[x->where][0] = 1LL \* ((x->bound) + (x->bound & 1)) \* x->cost % P,

v[x->where][0] = 1LL \* ((x->bound) + ((x->bound & 1) ^ 1)) \* x->cost % P;

}

**for** (**int** i = 1; i <= 20; i++)

**for** (**int** j = 1; j <= n; j++)

**if** (pa[j][i - 1])

pa[j][i] = pa[pa[j][i - 1]][i - 1],

f[j][i] = (f[j][i - 1] + f[pa[j][i - 1]][i - 1]) % P,

v[j][i] = (v[j][i - 1] + v[pa[j][i - 1]][i - 1]) % P;

scanf("%d", &m);

**for** (**int** i = 1; i <= m; i++) {

**int** x, y;

scanf("%d%d", &x, &y);

**int** z = lca(x, y), ans = base;

ans += calc(x, z); ans %= P;

ans += calc(y, z); ans %= P;

printf("%d**\n**", ans);

}

}

D

**题目描述**

    给定一个n\*m的迷宫，其中任意两个相邻单元之间可能存在墙，定义从单元S到单元T的路径为从S出发，通过向上、下、左、右四个方向上的移动组合，在不通过墙的前提下，最终到达T的轨迹。数据保证迷宫中的任意两点之间存在合法的路径。现在考虑一张有k个顶点的完全图G, 其中每个顶点对应迷宫中的某个单元，并且任意两个顶点之间连边的长度为其所对应迷宫单元之间的最短路径的长度。求G的最小生成树的边权和。

**输入描述:**

第一行三个正整数n, m, k (2 ≤ n, m ≤ 2,000, k ≤ 100,000)。  
接下来n行，每行m-1个 0/1。其中i行第j个数描述单元(i, j)和(i, j+1)之间是否存在墙。(1表示有墙，0表示没墙)  
接下来n-1行，每行m个 0/1。其中i行第j个数描述单元(i, j)和(i+1, j)之间是否存在墙。(1表示有墙，0表示没墙)  
接下来k行，每行两个正整数x, y，描述G中一个顶点所对应的单元，其中x代表行号，y代表列号。保证这k个顶点所对应的单元两两不同。

**输出描述:**

输出一行一个整数，描述最小生成树的边权和。

**示例1**

输入

5 5 3

0111

0010

0101

0110

0111

10000

10000

10001

10000

1 4

4 3

4 2

输出

9

#include <cstdio>

#include <cstdlib>

#include <iostream>

#include <algorithm>

**using** **namespace** std;

**const** **int** b13 = 1<<13;

**char** hmove[**int**(2e7) + 10];

**char** vmove[**int**(2e7) + 10];

**int** vis[**int**(2e7) + 10];

**int** dep[**int**(2e7) + 10];

**int** qidx[2000 \* 2000 + 10];

**int** cnt;

**int** n, m, k;

**int** ans;

**int** temp[2000 \* 2000 \* 2 + 10];

**int** \*temp\_a = temp;

**int** \*temp\_b = temp + 2000 \* 2000 + 5;

**inline** **int** check(**const** **int** &idx, **const** **int** &dir) {

**if** (dir == 0)

**return** hmove[idx] ? idx+1 : 0;

**else** **if** (dir == 1)

**return** hmove[idx-1] ? idx-1 : 0;

**else** **if** (dir == 2)

**return** vmove[idx] ? idx+b13 : 0;

**else** **if** (dir == 3)

**return** vmove[idx-b13] ? idx-b13 : 0;

}

**int** root(**const** **int** &x) {

**return** vis[x] == x ? x : vis[x] = root(vis[x]);

}

**int** main() {

**char** s[10010];

cin >> n >> m >> k;

**for** (**int** i = 0; i < n; ++i) {

scanf("%s", s);

**for** (**int** j = 0; j < m-1; ++j) {

**int** is\_wall = s[j] == '1';

hmove[(i+1)<<13|(j+1)] = !is\_wall;

}

}

**for** (**int** i = 0; i < n-1; ++i) {

scanf("%s", s);

**for** (**int** j = 0; j < m; ++j) {

**int** is\_wall = s[j] == '1';

vmove[(i+1)<<13|(j+1)] = !is\_wall;

}

}

**for** (**int** i = 1; i <= k; ++i) {

**int** idx, x, y;

scanf("%d %d", &x, &y);

idx = x<<13|y;

vis[idx] = idx;

qidx[cnt++] = idx;

}

**int** hp = 0, tp = cnt, curd = 0;

**while** (hp < tp) {

curd++;

**for** (**int** i = hp; i < tp; ++i) {

**int** idx = qidx[i];

**for** (**int** j = 0; j < 4; ++j) {

**int** nidx = check(idx, j);

**if** (nidx == 0)

**continue**;

**if** (vis[nidx]) {

**if** (root(nidx) != root(idx))

**if** (dep[nidx] == dep[idx]) {

ans += curd;

vis[root(nidx)] = root(idx);

}

**else** {

\*(temp\_a++) = idx;

\*(temp\_b++) = nidx;

}

}

**else** {

vis[nidx] = root(idx);

dep[nidx] = dep[idx] + 1;

qidx[cnt++] = nidx;

}

}

}

curd++;

**while** (temp\_a != temp) {

--temp\_a, --temp\_b;

**if** (root(\*temp\_a) != root(\*temp\_b)) {

vis[root(\*temp\_b)] = root(\*temp\_a);

ans += curd;

}

}

hp = tp;

tp = cnt;

}

cout << ans <

# E题目描述

    袋鼠先生最近迷上了玩塔防游戏。塔防游戏中有若干座防御塔。一些防御塔能覆盖的范围为这里面任意三个防御塔形成的三角形区域的并集。注意这里的三角形包括边界，且如果三角形退化那么就相当于一条线段。换句话说，三个或以上的防御塔所能覆盖的范围为这些点形成的凸包，小于三个的防御塔没有任何功能。  
    袋鼠先生发现有些防御塔和其他防御塔不同，如果拆除这个防御塔会导致防御的区域缩小那么我们称这个防御塔为关键的防御塔。这里区域缩小包括区域的面积减小，线段变短，或者从有到无。  
    袋鼠先生在建造某些防御塔上面氪了很多金币，所以袋鼠先生希望能通过拆除其他的防御塔使得他氪过金币的防御塔全部成为关键的防御塔。还有一点，袋鼠先生希望剩下的防御塔能覆盖的面积是正的。  
    袋鼠先生想知道有多少种这样的方案，答案可能很大，所以要对1,000,000,007取模。

**输入描述:**

第一行一个整数n (1 ≤ n ≤ 200)。  
  
接下来n行每行三个整数X\_i, Y\_i, W\_i(0 ≤ |X\_i|, |Y\_i| ≤ 1,000,000,000, W\_i ∈ {0,1})。(X\_i, Y\_i)表示每个塔的坐标。W\_i表示每个塔的属性，如果W\_i = 0表示这个塔可以拆除，否则表示这个塔不可拆除。  
  
数据保证没有重点。

**输出描述:**

输出一行表示答案，即有多少种方案。

**示例1**

输入

5

0 0 0

0 2 0

2 0 0

2 2 0

1 1 1

输出

4

**示例2**

输入

8

0 0 0

0 1 1

0 2 0

1 2 1

2 2 0

2 1 1

2 0 0

1 0 1

输出

7

**示例3**

输入

6

0 0 0

0 4 0

4 0 0

4 4 0

2 2 1

2 1 0

输出

11

# F题目描述

    小团在美团旅行的安全部门实习研究加密算法，他提出了一个加密算法。现在小团把解密算法告诉你，希望你对数据进行解密。  
    加密数据包有两部分构成，一对数字a, b和一个字符串S。解密的方式S串最早在a/b的小数部分出现的位置k，k即为解密信息。小数点后从1开始计数。如果a/b不是一个无限循环小数，则在认为后面有无穷多个0，如5/4=1.2500000000…。  
    如果S不在a/b的小数部分出现，则无法解密输出-1。

**输入描述:**

第一行两个正整数a, b (1 ≤ a, b ≤ 1,000,000,000)。第二行一个数字串S(1 ≤ |S| ≤100,000)。

**输出描述:**

输出一行k表示答案。

**示例1**

输入

1 2

500000000000

输出

1

**示例2**

输入

1234 5678

4579

输出

8

**示例3**

输入

233 999

333

输出

-1