T1点名

使用`ch=getchar()`判断下一个字符是空格还是回车，以此来判断一行的结束。

```
#include <bits/stdc++.h>
#define A 100010

using namespace std;
string s, ans = "11111111111111111111111111111111111111111";

int main(int argc, char const *argv[]) {
    while (cin >> s) {
        if (s.length() < ans.length()) ans = s;
        char ch = getchar();
        if (ch == '\n') {
            cout << ans << endl;
            ans = "11111111111111111111111111111111111111111";
        }
    }
}
```

T2石板密码

直接使用*string*自带的搜索方法`s.find(t)`，`s.npos`可以表示字符串s的结束位置，如果`s.find(t)`方法无法在字符串s中找到字符串t，那么会返回字符串s的结束位置，也就是`s.npos`。时间复杂度被字符串t在s中出现的位置所影响，可以拿到90分。

```
#include <bits/stdc++.h>
#define A 10010

using namespace std;
int L, n, m;
string s[A], S;

int main(int argc, char const *argv[]) {
    cin >> L >> n >> m;
    for (int i = 1; i <= n; i++) cin >> s[i];
    cin >> S;
    int ans = INT_MAX;
    for (int i = 1; i <= n; i++) {
        int x = S.find(s[i]);
```

```
            if (x == S.npos) continue;
            ans = min(ans, x);
        }
        if (ans == INT_MAX) puts("no");
        else cout << ans + 1 << endl;
    }
```

正解为字符串哈希，为了保证正确性，可以使用双哈希。
字符串哈希（hash）就是一个字符串到整数的映射，主要用处是快速比较两个字符串是否相等。
hash值相等两字符串大概率相等，hash值不等两字符串一定不等。
常用算法是 `RK-hash` ,就是把一个字符串看作一个 $base$ 进制的大整数，然后对一个素数 p 取模。

$$hash[i] = (hash[i-1] * base + s[i])$$

base 可以取 31、131、13131 等，大于m即可。
p 一定要是 `long long` 范围的一个素数，比如 23333333333333333、$10^{18}+9$。
`unsigned long long` 自然溢出可以看作是对 2^{64} 取模，但是可以被特殊构造卡掉。

```
#include <bits/stdc++.h>
#define A 100010

using namespace std;
typedef unsigned long long ull;
const ull mod = 1000000000 + 7;
const ull mod2 = 1000000000 + 9;
const ull base = 131;

int L, n, m;
ull h[A], h2[A];
char s[A];
ull gethash(char s[]) {
    ull ans = 0;
    for (int i = 0; s[i]; i++) {
        ans = (ans * base + s[i]) % mod;
    }
    return ans;
}
ull gethash2(char s[]) {
    ull ans = 0;
    for (int i = 0; s[i]; i++) {
        ans = (ans * base + s[i]) % mod2;
    }
    return ans;
}
set<pair<ull, ull> > st;
```

```
int main(int argc, char const *argv[]) {
    cin >> L >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> (s + 1);
        h[i] = gethash(s + 1);
        h2[i] = gethash2(s + 1);
        st.insert(make_pair(h[i], h2[i]));
    }
    cin >> (s + 1);
    ull now = 0, now2 = 0, basem = 1, basem2 = 1;
    for (int i = 1; i <= m; i++) {
        basem = basem * base % mod;
        basem2 = basem2 * base % mod2;
    }
    bool flag = 0;
    for (int i = 1; i <= L; i++) {
        now = (now * base + s[i]) % mod;
        now2 = (now2 * base + s[i]) % mod2;
        if (i > m) {
            now = ((now - s[i - m] * basem % mod) + mod) % mod;
            now2 = ((now2 - s[i - m] * basem2 % mod2) + mod2) % mod2;
        }
        if (st.count(make_pair(now, now2)) != 0) {
            printf("%d\n", i - m + 1);
            flag = 1;
            break;
        }
    }
    if (!flag) puts("no");
}
```

T3素数

枚举A，判断是否存在对应的整数B，使得$A^2+B^2=pq$。时间复杂度$O(T\sqrt{10^{18}})$，能拿到60分。

```
#include <bits/stdc++.h>
#define A 100010

using namespace std;
typedef long long ll;
ll p, q;
```

```
int main(int argc, char const *argv[]) {
    int T; cin >> T;
    while (T--) {
        cin >> p >> q;
        ll pq = p * q;
        int ans = 0;
        for (ll i = 1; i <= sqrt(pq); i++) {
            ll x = sqrt(pq - i * i);
            if (x * x + i * i != pq) continue;
            ans += 4;
        }
        cout << ans << endl;
    }
}
```

通过打表找规律，可以发现答案和p,q模4的余数有关系。
如果$p=q$:

1. $p=2$，$ans=4$
2. $p\%4=1$，$ans=12$
3. $p\%4=3$，$ans=4$

如果p不等于q:

1. $p=2$，$q\%4=1$，$ans=8$
2. $p=2$，$q\%4=3$，$ans=0$
3. $p\%4=1$，$q\%4=1$，$ans=16$
4. $p\%4=1$，$q\%4=3$，$ans=0$
5. $p\%4=3$，$q\%4=3$，$ans=0$

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

using namespace std;
int p, q, ans;

int main(int argc, char const *argv[]) {
    int T; cin >> T;
    while (T--) {
        cin >> p >> q;
        if (p == q) {
            if (p == 2) ans = 4;
            if (p % 4 == 1) ans = 12;
            if (p % 4 == 3) ans = 4;
        }
        else {
            if (p > q) swap(p , q);
```

```
            if (p % 4 == 3 || q % 4 == 3) {
                ans = 0;
            }
            else {
                if (p == 2) ans = 8;
                else ans = 16;
            }
        }
        cout << ans << endl;
    }
}
```

T4蚂蚁繁衍

把最外面一层的点都放入队列中做广搜，围起来的点是搜不到的，其中没被访问过的点的数量就是答案。

```
#include <bits/stdc++.h>
#define A 2010

using namespace std;
const int dx[] = {1, 0, -1, 0};
const int dy[] = {0, 1, 0, -1};
struct node {
    int x, y;
};
int k, a, x = 1001, y = 1001, ans;
char c;
bool vis[A][A];
void bfs() {
    queue<node> q;
    for (int i = 1; i < 2002; i++) {
        q.push(node{i, 0});
        q.push(node{i, 2002});
        q.push(node{0, i});
        q.push(node{2002, i});
    }
    while (!q.empty()) {
        node fr = q.front(); q.pop();
        for (int i = 0; i < 4; i++) {
            int fx = fr.x + dx[i], fy = fr.y + dy[i];
            if (fx < 1 or fx > 2001 or fy < 1 or fy > 2001 or vis[fx]
[fy]) continue;
            vis[fx][fy] = 1;
```

```
            q.push(node{fx, fy});
        }
    }
}

int main(int argc, char const *argv[]) {
    cin >> k;
    vis[x][y] = 1; ans = 1;
    while (k--) {
        cin >> c >> a;
        if (c == 'L') while (a--) if (!vis[x][--y]) vis[x][y] = 1, an
s++;
        if (c == 'R') while (a--) if (!vis[x][++y]) vis[x][y] = 1, an
s++;
        if (c == 'U') while (a--) if (!vis[--x][y]) vis[x][y] = 1, an
s++;
        if (c == 'D') while (a--) if (!vis[++x][y]) vis[x][y] = 1, an
s++;
    }
    bfs();
    for (int i = 1; i <= 2001; i++)
        for (int j = 1; j <= 2001; j++)
            if (!vis[i][j]) ans++;
    cout << ans << endl;
}
```

加一层离散化。

```
#include <bits/stdc++.h>
#define A 1010
#define li long long

using namespace std;
const int dx[4] = {-1, 0, 1, 0};
const int dy[4] = {0, 1, 0, -1};
li s1 = 19260817, s2 = 23333333, s3 = 998244853, srd;

int n, a[A], b[A];
li ans;
int nw[A], mn[A];
struct node {
    int l, r, p;
} p1[A], p2[A], q1[A], q2[A];
int tot1, tot2, t1, t2;
inline bool operator < (node q,node w){
    return q.p < w.p;
}
```

```
int bj[4010][4010], q[17000010][2], h, t;
void bf1() {
    for (int i = 1; i <= t1; i++)
        for (int j = q1[i].l; j <= q1[i].r; j++)
            bj[j + 1005][q1[i].p + 1005] = 1;
    for (int i = 1; i <= t2; i++)
        for (int j = q2[i].l; j <= q2[i].r; j++)
            bj[q2[i].p + 1005][j + 1005] = 1;
    q[++t][0] = 1; q[t][1] = 1; bj[1][1] = 2;
    while (h < t) {
        int nx = q[++h][0], ny = q[h][1];
        for (int i = 0; i < 4; i++) {
            int tx = nx + dx[i], ty = ny + dy[i];
            if (tx >= 1 and tx <= 2008 and ty >= 1 and ty <= 2008 and
!bj[tx][ty]){
                bj[tx][ty] = 2;
                q[++t][0] = tx;
                q[t][1] = ty;
            }
        }
    }
    for (int i = 1; i <= 2008; i++)
        for (int j = 1; j <= 2008; j++)
            if (bj[i][j] != 2)
                ans++;
    cout << ans << endl;
}
struct lsh {
    int x, id;
    bool fg, wz;
} ll[10010];
inline bool operator < (lsh q, lsh w) {
    return q.x < w.x;
}
int tot, len1[10010], len2[10010], nw1 = 1, nw2 = 1;
void work() {
    ++nw1; ++nw2; h = t = ans = 0;
    memset(bj, 0, sizeof bj);
    for (int i = 1; i <= t1; i++)
        for(int j = q1[i].l; j <= q1[i].r; j++)
            bj[j][q1[i].p] = 1;
    for (int i = 1; i <= t2; i++)
        for (int j = q2[i].l;j <= q2[i].r; j++)
            bj[q2[i].p][j] = 1;
    q[++t][0] = 1; q[t][1] = 1; bj[1][1] = 2;
    while (h < t) {
        int nx = q[++h][0], ny = q[h][1];
        for (int i = 0; i < 4; i++) {
```

```
            int tx = nx + dx[i], ty = ny + dy[i];
            if (tx >= 1 and tx <= nw1 and ty >= 1 and ty <= nw2 and !
bj[tx][ty]) {
                bj[tx][ty] = 2;
                q[++t][0] = tx;
                q[t][1] = ty;
            }
        }
    }
    for (int i = 1; i <= nw1; i++)
        for (int j = 1; j <= nw2; j++)
            if (bj[i][j] != 2)
                ans += 1ll * len1[i] * len2[j];
    cout << ans << endl;
}
int main(int argc, char const *argv[]) {
    srand(time(0));
    srd = (srd * s1 + s2 + rand()) % s3;
    char c;
    cin >> n;
    int nwx = 0, nwy = 0;
    for (int i = 1; i <= n; i++) {
        c = getchar();
        while (c < 'A' or c > 'Z') c = getchar();
        if (c == 'L') a[i] = 0;
        else if (c == 'U') a[i] = 1;
        else if (c == 'R') a[i] = 2;
        else a[i] = 3;
        cin >> b[i];
        int newx = nwx + dx[a[i]] * b[i], newy = nwy + dy[a[i]] * b
[i];
        if (nwx == newx) {
            p2[++tot2].l = min(newy, nwy);
            p2[tot2].r = max(newy, nwy);
            p2[tot2].p = newx;
        }
        else {
            p1[++tot1].l = min(newx, nwx);
            p1[tot1].r = max(newx, nwx);
            p1[tot1].p = newy;
        }
        nwx = newx; nwy = newy;
    }
    sort(p1 + 1, p1 + tot1 + 1);
    sort(p2 + 1, p2 + tot2 + 1);
    q1[0].p = q2[0].p = -INT_MAX;
    for (int i = 1; i <= tot1; i++) {
        if (q1[t1].p == p1[i].p) {
```

```
            q1[t1].l = min(q1[t1].l, p1[i].l);
            q1[t1].r = max(q1[t1].r, p1[i].r);
        }
        else q1[++t1] = p1[i];
    }
    for (int i = 1; i <= tot2; i++) {
        if (q2[t2].p == p2[i].p) {
            q2[t2].l = min(q2[t2].l, p2[i].l);
            q2[t2].r = max(q2[t2].r, p2[i].r);
        }
        else q2[++t2] = p2[i];
    }
    if (!t1) {
        cout << q2[1].r - q2[1].l + 1 << endl;
        return 0;
    }
    if (!t2) {
        cout << q1[1].r - q1[1].l + 1 << endl;
        return 0;
    }

    bool fg = 1;
    for (int i = 1; i <= t1; i++)
        if (q1[i].l < -1000 or q1[i].r > 1000 or q1[i].p < -1000 or q
1[i].p > 1000)
            fg = 0;
    for (int i = 1; i <= t2; i++)
        if (q2[i].l < -1000 or q2[i].r > 1000 or q2[i].p < -1000 or q
2[i].p > 1000)
            fg = 0;
    if (fg) {
        bf1();
        return 0;
    }
    for (int i = 1; i <= t1; i++) {
        ll[++tot].x = q1[i].l; ll[tot].id = i; ll[tot].wz = 0; ll[to
t].fg = 0;
        ll[++tot].x = q1[i].r; ll[tot].id = i; ll[tot].wz = 1; ll[to
t].fg = 0;
    }
    for (int i = 1; i <= t2; i++) {
        ll[++tot].x = q2[i].p;
        ll[tot].id = i;
        ll[tot].wz = 0;
        ll[tot].fg = 1;
    }
    sort(ll + 1, ll + tot + 1);
    for (int i = 1; i <= tot; i++) {
```

```
        if (i == 1) {
            ++nw1;
            len1[nw1] = 1;
        }
        else if (ll[i].x - ll[i - 1].x == 1){
            ++nw1;
            len1[nw1] = 1;
        }
        else if (ll[i].x - ll[i - 1].x > 1){
            ++nw1;len1[nw1] = ll[i].x - ll[i - 1].x - 1;
            ++nw1;len1[nw1] = 1;
        }
        if (ll[i].fg) q2[ll[i].id].p = nw1;
        else if (ll[i].wz) q1[ll[i].id].r = nw1;
        else q1[ll[i].id].l = nw1;
    }

    tot = 0;
    for (int i = 1; i <= t2; i++) {
        ll[++tot].x = q2[i].l; ll[tot].id = i; ll[tot].wz = 0; ll[to
t].fg = 0;
        ll[++tot].x = q2[i].r; ll[tot].id = i; ll[tot].wz = 1; ll[to
t].fg = 0;
    }
    for (int i = 1; i <= t1; i++) {
        ll[++tot].x = q1[i].p; ll[tot].id = i; ll[tot].wz = 0; ll[to
t].fg = 1;
    }
    sort(ll + 1, ll + tot + 1);
    for (int i = 1; i <= tot; i++) {
        if (i == 1) {
            ++nw2;
            len2[nw2] = 1;
        }
        else if (ll[i].x - ll[i - 1].x == 1) {
            ++nw2;
            len2[nw2] = 1;
        }
        else if (ll[i].x - ll[i - 1].x > 1) {
            ++nw2; len2[nw2] = ll[i].x - ll[i - 1].x - 1;
            ++nw2; len2[nw2] = 1;
        }
        if (ll[i].fg) q1[ll[i].id].p = nw2;
        else if (ll[i].wz) q2[ll[i].id].r = nw2;
        else q2[ll[i].id].l = nw2;
    }
    work();
```

```
}
```

T5连通颜色块

*bfs*遍历两列中间有多少颜色的块，有20分。

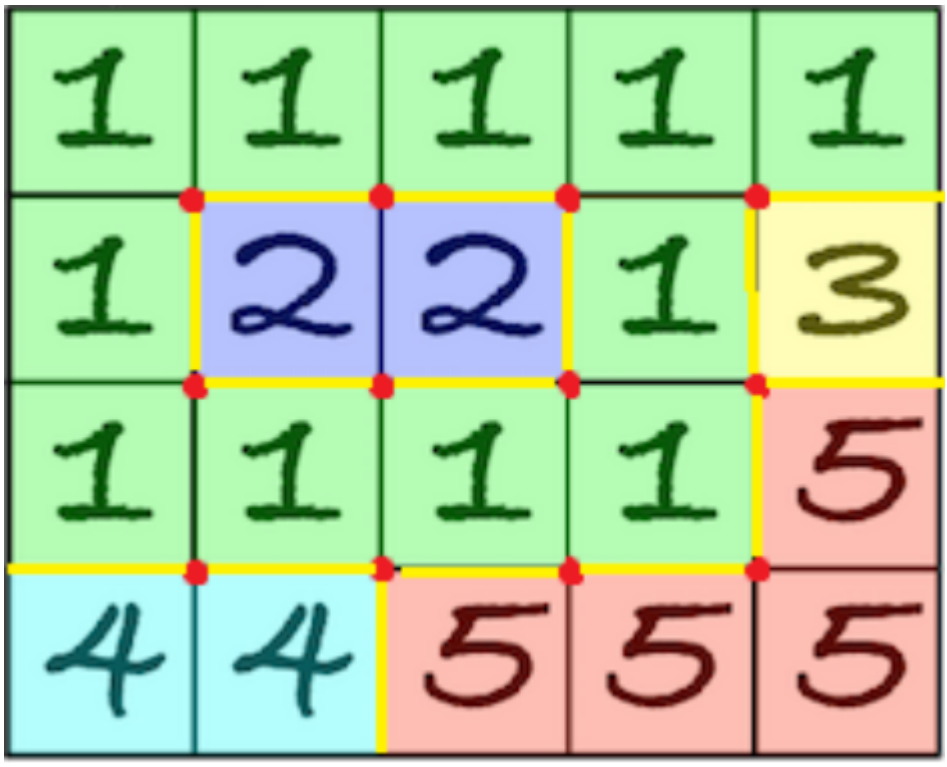
```
#include <bits/stdc++.h>
#define A 1010

using namespace std;
const int dx[] = {1, 0, -1, 0};
const int dy[] = {0, 1, 0, -1};
int n, m, q, c[A][A], l, r, ans;
bool vis[A][A];
void dfs(int x, int y, int color, int l, int r) {
    for (int i = 0; i < 4; i++) {
        int fx = x + dx[i], fy = y + dy[i];
        if (fx < 1 or fx > n or fy < 1 or fy > m) continue;
        if (fy < l or fy > r) continue;
        if (c[fx][fy] != color) continue;
        if (vis[fx][fy]) continue;
        vis[fx][fy] = 1;
        dfs(fx, fy, color, l, r);
    }
    return;
}

int main(int argc, char const *argv[]) {
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            cin >> c[i][j];
    while (q--) {
        memset(vis, 0, sizeof vis);
        ans = 0;
        cin >> l >> r;
        for (int i = 1; i <= n; i++)
            for (int j = l; j <= r; j++)
                if (!vis[i][j]) {
                    vis[i][j] = 1;
                    ans++;
                    dfs(i, j, c[i][j], l, r);
                }
        cout << ans << endl;
    }
```

```
}
```

建线段树，节点$[l,r]$统计$l\sim r$列颜色块数及第l列、第r列连通性，有40分。

将每个格子视作点。将同色相邻点连边。
先考虑没有环时（原图田字形也算一个环），记$a[i]$为单看第i列时的颜色段数，$b[i]$为第i列与第$i-1$列有多少个位置颜色相同。$l\sim r$列答案即为(
$a[l]+a[l+1]+\ldots+a[r])-(b[l+1]+b[l+2]+\ldots+b[r])$。
考虑每新出现一个环，上述方法计算的答案就会比正确答案小1，于是我们只需计算$l\sim r$列完整的环的个数，也就是计算每个极小环（只内部不再嵌套其他环）的左右端点。
知道每个极小环的左右端点后，可以使用二维前缀和或者区间dp的方式，每次$O(1)$计算$l\sim r$列完整的环的个数。
区间DP：$f[i][j]=f[i+1][j]+f[i][j-1]-f[i+1][j-1]+a[i][j]$

以原图内部格点为点（下图红色点），在相邻格数字不同时连边（每条黄色边两侧数字均不同），新图中每一个不与外边界联通的独立连通块均表示一个极小环。使用并查集维护每个连通块左右最值即可求出每个极小环左右端点。时间复杂度$O(mn+m^2+q)$，100分。




```cpp
#include <bits/stdc++.h>
#define A 1010

using namespace std;

int n, m, q, a[A][A];
int ps[A], ps1[A];
int ring[A][A], mp[A][A], flag, ll, rr;
void dfs(int i, int j) {
    if (mp[i][j]) return ;
```

```
    if (i == 0 or i == n or j == 0 or j == m) {
        flag = 0;
        return;
    }
    ll = min(ll , j); rr = max(rr, j);
    mp[i][j] = 1;
    if (a[i][j] != a[i + 1][j]) dfs(i , j - 1);
    if (a[i][j] != a[i][j + 1]) dfs(i - 1 , j);
    if (a[i][j + 1] != a[i + 1][j + 1]) dfs(i , j + 1);
    if (a[i + 1][j] != a[i + 1][j + 1]) dfs(i + 1 , j);
}
int main(int argc, char const *argv[]) {
    cin >> n >> m >> q;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
            if (a[i][j] == a[i - 1][j]) ps1[j]--;
            if (a[i][j] == a[i][j - 1]) ps[j]--;
        }
    for (int i = 1; i <= m; i++) {
        ps[i] += ps[i - 1];
        ps1[i] += ps1[i - 1];
    }
    for (int i = 1; i <= n - 1; i++)
        for (int j = 1; j <= m - 1; j++)
            if (!mp[i][j]) {
                flag = 1; ll = m + 1; rr = 0;
                dfs(i, j);
                if (flag == 1) {
                    rr++;
                    ring[ll][rr]++;
                }
            }
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= m; j++)
            ring[i][j] = ring[i - 1][j] + ring[i][j - 1] - ring[i -
1][j - 1] + ring[i][j];
    int u, v;
    for (int i = 1; i <= q; i++) {
        cin >> u >> v;
        cout << (v - u + 1) * n + ps[v] - ps[u] + ps1[v] - ps1[u - 1]
+ ring[v][v] - ring[u - 1][v] - ring[v][u - 1] + ring[u - 1][u - 1] <
< endl;
    }
}
```