> 课后思考：请在学习时间与空间复杂度后，判断这里每道题的解题时间空间复杂度

I/O练习 1

注意要点：

1. 开 long long，因为 $10^5 \times 580^5 \approx 6.5 \times 10^{19}$
2. 注意读整行时，因为第一行的缘故，可能需要判走第一行的换行

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll; //要开long long
ll x, ans;
ll read_and_calc()
{
    ll sum = 0, v;
    string s;
    getline(cin, s);
    stringstream ss(s);
    while (ss >> v)
    {
        ll vp = 1;
        for (ll i = 0; i < x; ++i)
        {
            vp *= v;
        }
        sum += vp;
    }
    return sum;
}
signed main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> x;
    cin.ignore(); //注意
    ans += read_and_calc();
    ans -= read_and_calc();
    cout << ans;
    return 0;
}
```

I/O练习 2

一种解决思路是先把每行看成若干个单词。根据五种运算序列的不同单词进行区分。那么就需要能够识别出每行的各单词，使用读整行得到一个字符串，再对该字符串进行一次读入即可。

比较单词时，可以比较第一个不一样的字母位置即可，不需要全部字母都比较，能一定程度简化代码编写。

记得开 long long，因为 $10^9 \times 10^9$ 爆 int。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll n, x, y;
string s, t, t1;
signed main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    cin >> n, cin.ignore();
    while (n--)
    {
        getline(cin, s);
        stringstream ss(s);
        ss >> t >> t1 >> t;
        if (t[0] == 'p' || t[0] == 'm')
        {
            x = stol(t1);
            if (t[0] == 'p') // plus
            {
                ss >> y;
                cout << x + y << '\n';
            }
            else if (t[1] == 'i') // minus
            {
                ss >> y;
                cout << x - y << '\n';
            }
            else // multiplied
            {
                ss >> t >> y;
                cout << x * y << '\n';
            }
        }
        else if (t[0] == 'q') // quotient
        {
            ss >> t >> x >> t >> y;
            cout << x / y << '\n';
        }
        else // remainder
        {
            ss >> t >> x >> t >> t >> y;
            cout << x % y << '\n';
        }
    }
    return 0;
}
```

I/O练习 3

因为单个维度长度可以长达 10^6，所以需要开动态数组。

可以不用 resize，那么每次都 push_back 输入即可；实现时下标可以从 0 开始算，没有影响。

读取 `/` 是本题的核心考点。`scanf` 读字符会读空格而不是 `/`，当然有很多方法可以处理，最直接的是将 `/` 看成字符串。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
vector<vector<vector<ll>>> s;
ll a, b, c, ans;
char tmp[3];
ll sgn(ll x) { return x > 0 ? 1 : (x < 0 ? -1 : 0); }
signed main()
{
    sc(a), sc(b), sc(c);
    s.resize(a + 1, vector<vector<ll>>(b + 1, vector<ll>(c + 1)));
    for (ll i = 1; i <= a; ++i)
    {
        for (ll j = 1; j <= b; ++j)
        {
            for (ll k = 1; k <= c; ++k)
            {
                sc(s[i][j][k]);
            }
            if (j != b)
            {
                scanf("%s", tmp);
            }
        }
    }
    for (ll i = 1; i <= a; ++i)
    {
        for (ll j = 1; j <= b; ++j)
        {
            for (ll k = 1; k <= c; ++k)
            {
                ans += sgn(s[i][j][k] + s[a - i + 1][b - j + 1][c - k + 1]);
            }
        }
    }
    printf("%lld", ans);
    return 0;
}
```

I/O练习 4

这题主要考察字符串输入，包括：单个字符的读入、整行的读入。以及处理行末需要特判。

对于操作 1，可以把 `a` 先全部换成一个不可能出现的字符如 `#`，再把 `b` 换成 `a`，在把 `#` 换成 `b`，有点像 swap 的思想；或者干脆用 `if-else if` 循环一次即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
```

```cpp
typedef long long ll;
ll m;
string x, y, c;
signed main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    while (cin >> m >> x >> y)
    {
        cin.ignore();
        getline(cin, c);
        if (x == "Y")
        {
            replace(c.begin(), c.end(), 'a', '#');
            replace(c.begin(), c.end(), 'b', 'a');
            replace(c.begin(), c.end(), '#', 'b');
        }
        if (y == "Y")
        {
            for (auto &i : c)
            {
                i = toupper(i);
            }
        }
        for (ll i = 0; i < m; ++i)
        {
            cout << c << '\n';
        }
    }
    return 0;
}
```

vector

模板题。注意 `size` 方法返回的算 `unsigned`，如果不强转就比较可能会 warning，当然也可以不管。插入和删除越界会运行错误，所以需要手动特判。vector 下标是从 0 开始算的，那么要么对输入的 i 直接减一，要么就像题解这样插入一个无用首元素垫一下。

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
ll n, m;
vector<ll> a;
signed main()
{
    sc(n), sc(m);
    a.push_back(0); //一种方便做法，加入一个无关首元素使得下标从1开始
    for (ll i = 1, v; i <= n; ++i)
    {
        sc(v);
        a.push_back(v);
    }
    for (ll c, i, x; m; --m)
```

```
    {
        sc(c), sc(i);
        if (c == 1 || c == 3)
        {
            sc(x);
        }
        if (i > (ll)a.size() - 1) //多了一个首元素0,故-1
        {
            continue;
        }
        if (c == 1)
        {
            a.insert(a.begin() + i, x);
        }
        else if (c == 2)
        {
            a.erase(a.begin() + i);
        }
        else if (c == 3)
        {
            a[i] = x;
        }
        else if (c == 4)
        {
            printf("%lld\n", a[i]);
        }
    }
    return 0;
}
```

priority_queue

priority_queue 默认是大根堆，题意要求需要使用小根堆，所以使用 pair 的话，默认重载了运算符，直接调用重载即可。

pair 版本：

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
typedef pair<ll, ll> pll;
priority_queue<pll, vector<pll>, greater<pll>> q;
ll m, cmd, x, y;
signed main()
{
    for (sc(m); m; --m)
    {
        sc(cmd);
        if (cmd == 1)
        {
            sc(x), sc(y);
            q.push({x, y});
```

```
        }
        else if (cmd == 2 && !q.empty())
        {
            q.pop();
        }
        else if (cmd == 3 && !q.empty())
        {
            auto pr = q.top();
            printf("%lld %lld\n", pr.first, pr.second);
        }
    }
    return 0;
}
```

自定义结构体版本：

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
struct node
{
    ll x, y;
    bool operator<(const node &r) const
    {
        return x != r.x ? x > r.x : y > r.y;
    }
};
priority_queue<node> q;
ll m, cmd, x, y;
signed main()
{
    for (sc(m); m; --m)
    {
        sc(cmd);
        if (cmd == 1)
        {
            sc(x), sc(y);
            q.push({x, y});
        }
        else if (cmd == 2 && !q.empty())
        {
            q.pop();
        }
        else if (cmd == 3 && !q.empty())
        {
            auto pr = q.top();
            printf("%lld %lld\n", pr.x, pr.y);
        }
    }
    return 0;
}
```

set

因为 set 结构是有序的且能够自动实现去重，所以将输入全部用 set 存储并顺序输出即可。

也可以使用 `std::sort` 和 `std::unique` 先排序再对排序好的数组进行去重。

两种方法都比较重要，都建议熟练掌握。作为对比的话，前者能够在线(这里表现为可以随时插入新的值得到新状态下的解)，而后者只能离线(全部输入后才能求解)。前者空间代价和时间代价稍大于后者。

set 参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
ll n;
set<ll> s;
signed main()
{
    sc(n);
    for (ll i = 1, v; i <= n; ++i)
    {
        sc(v);
        s.insert(v);
    }
    printf("%lld\n", s.size());
    for (auto v : s)
    {
        printf("%lld ", v);
    }
    return 0;
}
```

排序参考代码：(也可以用 vector，可自行尝试)

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
ll n, a[100010];
signed main()
{
    sc(n);
    for (ll i = 0; i < n; ++i)
    {
        sc(a[i]);
    }
    sort(a, a + n);
    n = unique(a, a + n) - a;
    printf("%lld\n", n);
    for (ll i = 0; i < n; ++i)
    {
        printf("%lld ", a[i]);
    }
    return 0;
```

```
21 }
```

set2

斜率为 $k=-\dfrac{A}{B}$，直接计算是浮点数，存在精度误差。虽然本题可过(double或以上精度，因为本题误差约为 10^{-9}，而 double 允许更小的误差)，但强烈不建议

用浮点数斜率的方法，因为通常很多题目不会允许浮点数误差能过题。

先令 B 取相反数，那么斜率可以表示成数值对 $(A,-B)$。存在整数 $k\in Z^*$，则所有数值对 $(kA,-kB)$ 与 $(A,-B)$ 斜率相等。 换言之，若对两个数值对 $(A_1,-B_1),(A_2,-B_2)$，设最大公因数为 $g_1=\gcd(A_1,-B_1),g_2=\gcd(A_2,-B_2)$，则 $\dfrac{A_1}{g_1}=\dfrac{A_2}{g_2}$ 且 $\dfrac{B_1}{g_1}=\dfrac{B_2}{g_2}$ 时斜率相等。

或者，只需要判断 $-\dfrac{A_1}{B_1}=-\dfrac{A_2}{B_2}$ 即 $A_1B_2=A_2B_1$ 即可判断斜率相等，从而避免了浮点数误差。

同理，设 B 取相反数后，比较 $\dfrac{A_1}{B_1}<\dfrac{A_2}{B_2}$ 即可判断两直线斜率的大小关系。

> 注意 A_1B_2 这样的乘式会爆 int，需要用 long long

特别地，思考这个例子：

$$\begin{cases}\dfrac{-1}{1}<\frac{1}{2} & \Rightarrow(-1)\times 2<1\times 1\\ \dfrac{1}{-1}<\frac{1}{2} & \Rightarrow 1\times 2<1\times(-1)\end{cases}$$

可以发现分母为负数时会导致错误，所以约定分母非负数。(也就是说，两边同乘分母，不等号不变向的充要条件时，分母均不为零)

特别地，当 $x=0$ 或 $y=0$ 时，需要把另一方设为 1 。否则容易发现，在该情况下，多个零斜率或无穷斜率与其他斜率的比较会出现偏差。如果做了 $\gcd$ 处理，即 A,B 同除以 $\gcd(A,B)$，那么可以顺利地做到这一点。如果不做 $\gcd$ 处理，也可以用 if 特判。

> 对于多数题目，一般会要求输出经过 $\gcd$ 化简的式子，所以建议还是学一下如何做 $\gcd$ 处理

那么，对第一个问题，即斜率是否相等，只需要调用 set 的 find 方法即可。

对第二个问题，调用 set 的 upper_bound 方法即可。

对第三个问题，调用 set 的 lower_bound 方法可以找到第一个大于等于 l 斜率的直线，然后使用指针往前移位，即可找到第一个不满足大于等于，即第一个小于的直线。

> 注意，切不可暴力遍历 set 数组，或使用 `std::lower_bound(set.begin(),set.end(), )` 这样的方法，会超时。复杂度是不正确的，为 $O(n)$。而用 set 内置的上述三个方法为 $O(\log n)$。

参考程序：(未使用 $\gcd$)

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
struct line
{
    ll a, b, c;
    line(ll aa, ll bb, ll cc)
```

```
    { //构造函数,无返回值(不是void),功能是创建时自动执行
        a = aa, b = -bb, c = cc;
        if (b < 0) //分母为零处理
        {
            a *= -1, b *= -1;
        }
        if (a == 0)
        {
            b = 1;
        }
        if (b == 0)
        {
            a = 1;
        }
    }
};
bool operator<(const line &l, const line &r)
{ //-a1/b1 < -a2/b2 => a1*b2>a2*b1
    return l.a * r.b < r.a * l.b;
}
set<line> s;
ll n, m, a, b, c;
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= n; ++i)
    {
        sc(a), sc(b), sc(c);
        s.insert(line(a, b, c)); //调用构造函数
    };
    while (m--)
    {
        sc(a), sc(b), sc(c);
        line l = line(a, b, c); //调用构造函数
        auto p = s.find(l);
        if (p == s.end())
        {
            printf("no ");
        }
        else
        {
            printf("%lld ", p->c);
        }

        p = s.upper_bound(l);
        if (p == s.end())
        {
            printf("no ");
        }
        else
        {
            printf("%lld ", p->c);
        }

        p = s.lower_bound(l);
```

```
        if (p == s.begin())
        {
            printf("no ");
        }
        else
        {
            --p;
            printf("%lld ", p->c);
        }
        printf("\n");
    }
    return 0;
}
```

若要使用 $\gcd$，将构造函数修改为：

```
line(ll aa, ll bb, ll cc)
{
    bb *= -1;
    ll s = __gcd(aa, bb);
    a = aa / s, b = bb / s, c = cc;
    if (b < 0)
    {
        a *= -1, b *= -1;
    }
}
```

map

可以先用一个 map 计数每个字符串的出现频次，然后再设一个 map，记录每个不同的频次出现的所有字符串(可以用 vector)，遍历第一个 map，此时键按字典序排序，所以存入第二个 map 时字符串一定是按字典序的。为了实现频次逆序，可以把频次的相反数存入第二个 map，输出时再取一次相反数。

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
map<string, ll> m;
ll n;
string s;
map<ll, vector<string>> m2;
signed main()
{
    ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
    for (cin >> n; n; --n)
    {
        cin >> s;
        m[s]++;
    }
    for (auto pr : m)
    {
        m2[-pr.second].push_back(pr.first);
    }
```

```
    for (auto i : m2)
    {
        for (auto j : i.second)
        {
            cout << j << ' ' << -i.first << '\n';
        }
    }
    return 0;
}
```

bitset

把每个集合用一个 bitset 存储，初始设第 i 个 bitset 的第 i 位为 1，其他位都是 0。每次合并即直接进行或运算。最后输出所有含 n 个 1 的 bitset 编号。

```
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
const ll mn = 5e4 + 5;
bitset<mn> s[mn];
ll n, m, cnt;
signed main()
{
    sc(n), sc(m);
    for (ll i = 1; i <= n; ++i)
    {
        s[i][i] = 1;
    }
    for (ll u, v; m; --m)
    {
        sc(u), sc(v);
        s[u] = s[v] = s[u] | s[v];
    }
    for (ll i = 1; i <= n; ++i)
    {
        if (s[i].count() == n)
        {
            ++cnt;
            printf("%lld ", i);
        }
    }
    if (!cnt)
    {
        printf("-1");
    }
    return 0;
}
```

梅花易数

签到小模拟题。按题意模拟即可。

将输入时的 12 种地支串转化为 $1,2,3,\cdots,12$ 的对应数字。

而观察三爻卦象的排位与其阴阳爻的关系。不难发现对于一个除以 8 的余数 i，$(i-1)$ 的二进制最后一位为 $0/1$ 分别对应了卦象最上一爻是阴爻还是阳爻；同理，$(i-1)$ 的二进制倒数第二位与倒数第三位，分别对应卦象的中间爻和最下爻分别是阴爻还是阳爻。

因此，我们不妨用一个 $[0,2^6)$ 内的二进制数表示一个六爻的卦象。其低三位为 $y+m+d-1 \bmod 8$，高三位为 $y+m+d+h-1 \bmod 8$。输出时从高往低遍历，遇到二进制位为 0 时输出“短线-短线-短线”，遇到为 1 时输出“短线-空格-短线”。

而对于变卦，其变化的位置为该二进制数的从低向高第 $6-(y+m+d+h) \bmod 6$ 位，直接将结果异或上 $2^{6-(y+m+d+h) \bmod 6}$ 即可。同样采用上述方法输出，即可得到正确答案。

```cpp
#include <bits/stdc++.h>
using ll = long long;
using namespace std;
map<string,ll> h1;
string ys,hs;
ll y,h,m,d,ans[10];
ll a,b,c;
void print()
{
    for(ll i=0;i<=5;++i)
    {
        printf("-%c-\n","- "[ans[i]]);
    }
}
signed main()
{
    h1["Zi"]=1;
    h1["Chou"]=2;
    h1["Yin"]=3;
    h1["Mao"]=4;
    h1["Chen"]=5;
    h1["Si"]=6;
    h1["Wu"]=7;
    h1["Wei"]=8;
    h1["Shen"]=9;
    h1["You"]=10;
    h1["Xu"]=11;
    h1["Hai"]=12;
    cin>>ys>>m>>d>>hs;
    y=h1[ys];
    h=h1[hs];
    a=(y+m+d-1)%8;
    ans[0]=a%2,ans[1]=a/2%2,ans[2]=a/4%2;
    b=(y+m+d+h-1)%8;
    ans[3]=b%2,ans[4]=b/2%2,ans[5]=b/4%2;
    print();
    printf("\n");
    c=(6-(y+m+d+h)%6)%6;//注意取模两次而不是一次
    ans[c]^=1;
    print();
    return 0;
```

```
}
/*
Zi 1 9 Si
Chen 12 17 Wei
*/
```

星月学语

参见 这里，在C++模拟里给出了 C 风格字符串和 C++ string 两种参考代码，希望你至少掌握其中一种。其他解法可以暂时不必理会。

望舒客栈的每日委托

小模拟题。

显然，最多需要 n 张桌子。开 5 个 set s_i 表示当前坐了 i 个人的所有桌子集合。初始时 $s_0 = (1, 2, \cdots, n), s_i (i \neq 0) = \varnothing$。执行下列操作时记录所操作的最大桌子编号 m，即为答案。预处理复杂度为 $O(n \log n)$。

按时间遍历：

每来一个社恐，就从 s_0 里找到第一张桌子并删除，然后往 s_{x_i} 里插入对应编号的桌子，复杂度为 $O(2 \log n)$。

每来一个社牛，就从所有满足 $0 \leq j \leq 4 - x_i$ 的 s_j 里找到它们的第一张桌子，然后取这些桌子里编号最小的那张，将其删除，然后往 s_{j+x_i} 里插入对应桌子。复杂度为 $O(4 + 2 \log n)$。

插入桌子时在这桌人要走的时刻对应存一下桌子编号。

每走一个人，读取存的桌子编号，找一下哪个 s_j 有这个编号，把 s_j 里对应删掉，然后把 s_{j-x_i} 插入这个桌子。复杂度为 $O(4 \log n + 2 \log n)$。

注意不能直接存 j，因为一张桌子可能有很多批客人拼桌，当一批客人走了之后，对应的 j 会改变，需要更新，比较麻烦。如果要这么实现，还需要开多一个数组，动态记录每张桌子当前有多少个人。这样的话复杂度优化为 $O(1 + 2 \log n)$。下面代码没有这么做，感兴趣自行实现。

因此，总时间复杂度为 $O(n \log n + 2n \log n + 6n \log n) \approx 1.8 \times 10^8$，可以过题。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define sc(x) scanf("%lld", &x)
typedef long long ll;
const ll mn = 2e6 + 10;
set<ll> s[5]; // s[i]表示坐了i个人的所有桌子
ll x[mn], t[mn], isJoin[mn], n, m, alloc[mn], nx[mn];
signed main()
{
    sc(n);
    for (ll i = 1; i <= n; ++i)
    {
        s[0].insert(i);
```

```
    }
    for (ll i = 1, xv, a, d, tv; i <= n; ++i)
    {
        sc(xv), sc(a), sc(d), sc(tv);
        x[a] = x[d] = xv;
        t[a] = t[d] = tv;
        isJoin[a] = 1, isJoin[d] = 0;
        nx[a] = d;
    }
    for (ll i = 1; i <= 2 * n; ++i)
    {
        if (isJoin[i])
        {
            if (!t[i])
            { //社恐
                auto j = s[0].begin();
                m = max(m, *j);
                alloc[nx[i]] = *j;
                s[x[i]].insert(*j);
                s[0].erase(j);
            }
            else
            {
                ll mi = n + 1, del;
                decltype(s[0].begin()) p; //类似auto p = s[0].begin();(感兴趣自行百度)
                for (ll j = 0; j <= 4 - x[i]; ++j)
                {
                    auto k = s[j].begin();
                    if (k == s[j].end())
                    {
                        continue;
                    }
                    if (*k < mi)
                    {
                        mi = *k;
                        p = k;
                        del = j;
                    }
                }
                m = max(m, mi);
                alloc[nx[i]] = mi;
                s[del + x[i]].insert(mi);
                s[del].erase(p);
            }
        }
        else
        {
            decltype(s[0].begin()) p;
            ll del;
            for (ll j = 0; j <= 4; ++j)
            {
                auto k = s[j].find(alloc[i]);
                if (k != s[j].end())
                {
```

```
                    p = k;
                    del = j;
                    break;
                }
            }
            s[del - x[i]].insert(*p);
            s[del].erase(p);
        }
    }
    printf("%lld", m);
    return 0;
}
```