A. 化学方程式

```cpp
#include<iostream>
#include<string>
#include<vector>
#include<sstream>
#include<map>
using namespace std;
using ll = long long;

// 原子操作：读取一个整数
ll readInt(string &s, ll &p) {
    ll res = 0;
    bool flag = false;

    while (p != s.size() && isdigit(s[p])) {
        flag = true;
        res = res * 10 + (s[p] - '0');
        p++;
    }

    if (flag) return res;
    return 1;
}

// 根据符号对字符串进行分隔
vector<string> split(string &s, char c) {
    stringstream ss(s);
    vector<string> res;
    string t;

    while (getline(ss, t, c)) {
        res.push_back(t);
    }
    return res;
}

// 合并两个答案字典
void merge(map<string, ll> &res, map<string, ll> const & a, ll coef) {
    for (auto &i : a) {
        res[i.first] += i.second * coef;
    }
}

map<string, ll> readItem(string &s, ll &p);

// 读取化学式
map<string, ll> readFormula(string &s, ll &p) {
    map<string, ll> res, a;

    while (s[p] != ')') {
        a = readItem(s, p);
        // ll c = readInt(s, p);
        merge(res, a, 1);   // 逐项合并至答案
    }

    return res;
}

// 读取元素
string readElement(string &s, ll &p) {
    string res;
    res = s[p++];   // 读取首个字母
    while (p != s.size() && islower(s[p])) {    // 若是小写字母则和前面字母构成一个元素
        res += s[p];
        p++;
    }
    return res;
}

// 读取一项
map<string, ll> readItem(string &s, ll &p) {
    map<string, ll> res, a;

    if (s[p] == '(') {
        p++;
        a = readFormula(s, p);
        p++;    // ')'
        ll c = readInt(s, p);   // 像Ca(OH)2,要读取括号后参数乘至答案
        merge(res, a, c);
    }
    else {
        string b = readElement(s, p);   // 读一项:读一个元素，读一个系数
        ll c = readInt(s, p);
        res[b] += c;
    }

    return res;
}

map<string, ll> J(string s) {   // 求表达式的元素统计表
    vector<string> f = split(s, '+');
    map<string, ll> res, a;

    for (int i = 0; i < f.size(); i++) {
        a.clear();
        ll p = 0;
        ll coef = readInt(f[i], p);     // 读整体系数，像4H2O中的‘4’
        while (p != f[i].size()) {      // 逐项读取化学式，直至化学式结束
            a = readItem(f[i], p);
            merge(res, a, coef);
        }
    }
    return res;
}

void solve() {
    int n; cin >> n;

    string s;
    for (int i = 1; i <= n; i++) {
        cin >> s;
        vector<string> res = split(s, '=');     // 以‘=’分隔得到左右表达式

        if (J(res[0]) == J(res[1])) {
            cout << "Y\n";
        }
        else cout << "N\n";
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    solve();

    return 0;
}
```

B. 带配额的文件系统

```cpp
#include<iostream>
#include<map>
#include<vector>
#include<sstream>
using namespace std;
using ll = long long;

struct file
{
    bool D;          // 是否是目录文件
    ll ld, lr;       // 目录配额 后代配额
    ll sd, sr;       // 实际孩子文件大小 实际后代文件大小
    map<string, file> children;

    ll fileSize;

    file() {
        D = true;
        ld = lr = 1e18;
        sd = sr = 0;
        fileSize = 0;
    }
};

// 解析文件路径： /a/b/c/1 -> {"", "a", "b", "c", "1"}
vector<string> parsePath(string &path) {
    stringstream ss(path);
    string fileName;
    vector<string> res;

    while (getline(ss, fileName, '/')) {
        res.push_back(fileName);
    }
    return res;
}

// 根目录文件
file root;

// 查找文件，返回文件指针集合，代表目录树和查找文件的最大重合路径，支持对目录树的修改
vector<file *> findPath(vector<string> &fileNames) {
    vector<file *> res;
    res.push_back(&root);

    for (int i = 1; i < fileNames.size(); i++) {
        // 普通文件没有后代
        if (res.back()->D == false) {
            break;
        }
        // 后代中没有重合路径
```

```
            if (res.back()->children.count(fileNames[i]) == 0) {
                break;
            }
            res.push_back(&res.back()->children[fileNames[i]]);
        }
        return res;
    }

    bool create() {
        string path;
        ll fileSize;
        cin >> path >> fileSize;

        auto fileNames = parsePath(path);
        auto filePtrs = findPath(fileNames);
        ll inc;
        // 若最大重合路径和查找文件的路径长度相同，代表找到文件
        if (fileNames.size() == filePtrs.size()) {
            // 目录文件不支持修改
            if (filePtrs.back()->D) {
                return false;
            }
            else {
                // inc = 和原来文件大小的差量
                inc = fileSize - filePtrs.back()->fileSize;
            }
        }
        else {
            if (filePtrs.back()->D == false) {
                return false;
            }   // 与孩子文件重名
            else inc = fileSize;
        }

        // 检查配额
        for (int i = 0; i < filePtrs.size(); i++) {
            if (filePtrs[i]->sr + inc > filePtrs[i]->lr) {
                return false;
            }
            // 双亲目录同时检查目录配额
            if (i == fileNames.size() - 2) {
                if (filePtrs[i]->sd + inc > filePtrs[i]->ld) {
                    return false;
                }
            }
        }

        // 创建文件
        for (int i = filePtrs.size(); i < fileNames.size(); i++) {
            filePtrs.push_back(&filePtrs.back()->children[fileNames[i]]);
        }
        filePtrs.back()->D = false;
        filePtrs.back()->fileSize = fileSize;

        // 更新实际文件大小
        for (auto &i : filePtrs) {
            i->sr += inc;
        }
        filePtrs.end()[-2]->sd += inc;
        return true;
    }

    bool remove() {
        string path;
        cin >> path;

        auto fileNames = parsePath(path);
        auto filePtrs = findPath(fileNames);
        // 没有找到文件则不用移除，直接返回成功
        if (fileNames.size() != filePtrs.size()) {
            return true;
        }
        // 移除目录文件
        if (filePtrs.back()->D) {
            for (auto &i : filePtrs) {
                i->sr -= filePtrs.back()->sr;
            }
        }
        else {      // 移除普通文件
            for (auto &i : filePtrs) {
                i->sr -= filePtrs.back()->fileSize;
            }
            filePtrs.end()[-2]->sd -= filePtrs.back()->fileSize;
        }
        filePtrs.end()[-2]->children.erase(fileNames.back());
        return true;
    }

    bool modify() {
        string path;
        ll d, r;
        cin >> path >> d >> r;

        auto fileNames = parsePath(path);
        auto filePtrs = findPath(fileNames);

        if (fileNames.size() != filePtrs.size()) {
            return false;
        }
        if (filePtrs.back()->D == false) {
            return false;
        }

        // 0代表不限额，设置为1e18
        if (d && filePtrs.back()->sd > d) return false;
        if (r && filePtrs.back()->sr > r) return false;

        if (!d) filePtrs.back()->ld = 1e18;
        else filePtrs.back()->ld = d;

        if (!r) filePtrs.back()->lr = 1e18;
        else filePtrs.back()->lr = r;

        return true;
    }

    void solve() {
        int n; cin >> n;
        string cmd;
        for (int i = 1; i <= n; i++) {
            cin >> cmd;
            if (cmd == "C") {
                if (create()) cout << "Y\n";
                else cout << "N\n";
            }
            else if (cmd == "R") {
                if (remove()) cout << "Y\n";
                else cout << "N\n";
            }
            else {
                if (modify()) cout << "Y\n";
                else cout << "N\n";
            }
        }
    }

    int main() {
        ios::sync_with_stdio(false);
        cin.tie(nullptr);

        solve();

        return 0;
    }
```