```cpp
#include<iostream>
#include<vector>
#include<string>
#include<fstream>
#include<unordered_map>
#include<algorithm>
using namespace std;

const string IN_FILE = "d_difficult.in.txt";

class Customer {  // node
public:
    int id;
    vector<string> like;
    vector<string> dislike;

    Customer(int my_id):id(my_id){};
};

class Graph {
public:
    vector<Customer> vertex;
    vector<vector<bool>> arc;  // 邻接矩阵
    int vertex_num;  // 节点数

    Graph(int n) {
        vertex_num = 0;
        arc = vector<vector<bool>>(n, vector<bool>(n, false));
        for (int i = 0; i < n; i++) arc[i][i] = true;
    }
};

void update(Graph& g, Customer cus, unordered_map<string, vector<int>>& like_map, unordered_map<string, vector<int>>& dislike_map);
void maxclique(vector<int>& vis, vector<int>& cnt, Graph& g);
bool dfs(vector<int>& vis, vector<int>& cnt, Graph& g, int cur, int num, int& ans);

int main() {
    int n;  // number of customers
    unordered_map<string, vector<int>> like_map;
    unordered_map<string, vector<int>> dislike_map;

    // ifstream ("IN_FILE");  // input file
    ifstream infile(IN_FILE);

    if (!infile.is_open()) {
        cout << "Failed to open the input file." << endl;
        return 0;
    }

    infile >> n;  // reading data

    Graph g(n);  // 图

    for (int i = 0; i < n; i++) {  // 读取所有客户资料
        Customer cus(i);

        int n_like, n_dislike;
```

```cpp
            string temp;

            infile >> n_like;  // 喜欢的
            for (int j = 0; j < n_like; j++) {
                infile >> temp;
                cus.like.push_back(temp);
                like_map[temp].push_back(i);  // 喜欢蔬菜temp的有客户i
            }

            infile >> n_dislike;  // 不喜欢的
            for (int j = 0; j < n_dislike; j++) {
                infile >> temp;
                cus.dislike.push_back(temp);
                dislike_map[temp].push_back(i);  // 不喜欢蔬菜temp的有客户i
            }

            // 更新图
            update(g, cus, like_map, dislike_map);
        }

        // 用最大团方法（while）
        // step1.找到一个最大团
        // step2.把这个最大团从原图中抠出来
        // step3.重复直到g中没有节点

        int ret = 0;  // 最终能满足几个人

        while (g.vertex_num > 0) {
            // 最大团
            vector<int> vis(n, -1);  // 存放已选择的点
            vector<int> cnt(n, 0);  // cnt[i]表示编号>=i的点所能组成的最大团的点数
            maxclique(vis, cnt, g);

            // 挖掉最大团
            // 修改顶点数和邻接矩阵
            ret++;
            vector<int> t = vis;

            auto iter1 = g.arc.begin();  // 删除行
            for (int i = 0; i < cnt[0]; i++) {
                g.vertex_num--;
                g.arc.erase(iter1+t[i]);
                for (int j = i+1; j < cnt[0]; j++) t[j]--;
            }

            t = vis;
            for (int j = 0; j < cnt[0]; j++) {  // 删除列
                // g.arc[0].erase(g.arc[0].begin()+t[j]);
                std::for_each(g.arc.begin(), g.arc.end(), [&](std::vector<bool>& row) {
                    row.erase(std::next(row.begin(), t[j]));
                });
                for (int k = j+1; k < cnt[0]; k++) t[k]--;
            }
        }
        cout << ret;

        return 0;
    }
```

```cpp

// 更新图
void update(Graph& g, Customer cus, unordered_map<string, vector<int>>& like_map, unordered_map<string, vector<int>>& dislike_map) {
    g.vertex.push_back(cus);
    g.vertex_num++;
    // 如果两个客户之间有矛盾，就连起来
    // eg1.A喜欢但B不喜欢
    for (int i = 0; i < cus.like.size(); i++) {
        string name = cus.like[i];
        if (dislike_map.count(name)) {  // 有客户不喜欢蔬菜name
            for (int j = 0; j < dislike_map[name].size(); j++) {
                if (g.arc[cus.id][dislike_map[name][j]] == false) {
                    g.arc[cus.id][dislike_map[name][j]] = true;  // 连起来
                    g.arc[dislike_map[name][j]][cus.id] = true;
                }
            }
        }
    }
    // eg2.A不喜欢但C喜欢
    for (int i = 0; i < cus.dislike.size(); i++) {
        string name = cus.dislike[i];
        if (like_map.count(name)) {  // 有客户喜欢name
            for (int j = 0; j < like_map[name].size(); j++) {
                if (g.arc[cus.id][like_map[name][j]] == false) {
                    g.arc[cus.id][like_map[name][j]] = true;  // 连起来
                    g.arc[like_map[name][j]][cus.id] = true;
                }

            }
        }
    }
}

// 新的 max
void maxclique(vector<int>& vis, vector<int>& cnt, Graph& g) {
    int ans = 0;
    int max = 0;
    vector<int> temp(vis);
    for (int i = g.vertex_num - 1; i >= 0; i--) {
        temp[0] = i;
        dfs(temp, cnt, g, i, 1, ans);
        cnt[i] = ans;
        if (ans > max) {
            max = ans;
            vis = temp;
        }
    }
}

// dfs
bool dfs(vector<int>& vis, vector<int>& cnt, Graph& g, int cur, int num, int& ans) {
    // 从第cur个节点向后添加，当前点是第num个
    for (int i = cur+1; i < g.vertex_num; i++) {
        if (cnt[i] + num <= ans) return 0;
        if (g.arc[cur][i]) {  // 两点相邻
            int ok = 1;
            for (int j = 0; j < num; j++) {
```

```
173                     if (!g.arc[i][vis[j]]) {
174                         ok = 0;
175                         break;
176                     }
177                 }
178                 if (ok) {  // 可以加入团
179                     vis[num] = i;
180                     if (dfs(vis, cnt, g, i, num+1, ans)) return 1;
181                 }
182             }
183         }
184         ans = max(ans, num);
185         return ans == max(num, ans) ? 0 : 1;
186     }
```