**★★★★☆**

**題組：Problem Set Archive** **with Online Judge**

**題號：753: A plug for UNIX**

**解題者：劉育希**

**解題日期：2025年5月22日**

**題意：**

不同的電器設備需要接上不同類型的插頭才能接上電源，現在你要盡可能的讓更多的設備接上電源。首先你手上有 n $\left(1\leq n\leq 100\right)$ 個插頭以及插頭的類型，還有 m $\left(1\leq m\leq 100\right)$ 個設備以及他們所需要的插頭，另外還有 k $\left(1\leq k\leq 100\right)$ 個不同種類的轉接頭，轉接頭是單向的 A B 代表能把 B 插頭轉變為 A 插頭，轉接頭數量沒有限制。求最少有多少設備無法接上電源。

輸入第一行有一個正整數 t 表示一共有 t 筆測資。每筆測資的第一行包含一個正整數 n 表示有 n 個插頭，接下來的 n 行字串列出了每個插頭的類型；再來的一行包含一個正整數 m 表示有 m 個設備，接下來的 m 行列出了每個設備的名字與所需要的插頭類型；再來的一行包含一個正整數 k 表示有 k 個轉接頭，接下來的 k 行列出了每個轉接頭的前端與後端的插頭類型。插頭類型與設備名稱皆為英文字母組成，且長度不會超過 24 字元。

**題意範例：**

1 → **1**

5

A

B

C

D

E

5

laptop B

phone C

pager B

clock B

comb X

3

B X

X A

X D

**解法：**

Step 0：

觀察題目，n 種插頭需要盡可能接上 m 種設備，可以看做「最大流」問題。

Step 1：

將源點(source, 標記為 S)接上所有插頭，將所有的設備接上匯點(sink, 標記為 T)

Step 2：

利用Edmonds-Karp Algorithm 取得最大流 → $O\left(VE^{2}\right)$

Step 2-1： BFS 尋找擴增路徑

Step 2-2： 新擴增路徑上每一條邊的流量

Repeat ： 重複直到無法找到新的擴增路徑

Step 3：

匯集到匯點的流量表示可以接上的設備的最大數量，將總設備量減去最大流量即為答案

**解法範例：**

|  |  |  |
| --- | --- | --- |
| 初始化 | 1st Iteration | 2nd Iteration |
|  |  |  |
| 3rd Iteration | 4th Iteration | Complete |
|  |  |  |

**討論：**

* 1. 此報告僅使用 Edmonds-Karp Algorithm 解題，其餘的最大流演算法例如 Dinitz’s 演算法 ($O\left(V^{2}E\right)$) ，只要複雜度可以接受也可拿來解題。
	2. 範例測資並沒有使用到了逆流(退流)的邊，但是單純 BFS 的貪心版本也沒辦法取得正確答案，在更大且複雜的測資，貪心法會產生阻塞流，所有會有最大流演算法的必要。

**程式：**

#include <bits/stdc++.h>

#define BOOST ios\_base::sync\_with\_stdio(false);cin.tie(0);

using llong = long long;

using namespace std;

const int N = 1e3 + 5;

const int INF = 1e9;

const int S = 1000; // Starting Point

const int T = 1001; // Ending Point

struct edg{

 int s, t;

 int cap;

 int flow;

 edg(int s, int t, int c, int f):

 s(s), t(t), cap(c), flow(f) {}

};

int n, m, k;

vector<vector<int>> graph;

vector<edg> edges;

vector<int> path;

unordered\_map<string, int> nameMap;

void init(){

 edges.clear();

 graph.assign(N, vector<int>());

 path.assign(N, 0);

 decltype(nameMap)().swap(nameMap);

}

int getIdx(string s){

 if(nameMap.find(s) == nameMap.end()){

 int idx = nameMap.size();

 nameMap[s] = idx;

 }

 return nameMap[s];

}

void addEdge(int s, int t, int c){

 edges.push\_back(edg(s, t, c, 0));

 edges.push\_back(edg(t, s, 0, 0)); // backflow cap == 0

 int idx = edges.size();

 graph[s].push\_back(idx - 2);

 graph[t].push\_back(idx - 1);

}

int BFS(){

 vector<bool> vis(N, false);

 vector<int> flow(N, INF);

 queue<int> q;

 vis[S] = true;

 q.push(S);

 while(!q.empty()){

 int cur = q.front();

 q.pop();

 if(cur == T) return flow[T];

 for(int i=0;i<graph[cur].size();i++){

 edg e = edges[graph[cur][i]];

 if(!vis[e.t] && (e.cap > e.flow)){

 flow[e.t] = min(flow[cur], (e.cap - e.flow));

 path[e.t] = graph[cur][i];

 vis[e.t] = true;

 q.push(e.t);

 }

 }

 }

 return 0; // No path

}

int maxFlow(){ // Edmonds - Karp

 int f, df;

 for(f = 0;(df=BFS());f+=df){

 for(int i=T;i!=S;i=edges[path[i]].s){

 edges[path[i]].flow += df;

 edges[path[i] + 1].flow -= df;

 }

 }

 return f;

}

void solve(){

 cin >> n;

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

 string type;

 cin >> type;

 int plug = getIdx(type);

 addEdge(S, plug, 1);

 }

 cin >> m;

 for(int i=0;i<m;i++){

 string a, b;

 cin >> a >> b;

 int device = getIdx(a);

 int plug = getIdx(b);

 addEdge(device, T, 1);

 addEdge(plug, device, 1);

 }

 cin >> k;

 for(int i=0;i<k;i++){

 string a, b;

 cin >> a >> b;

 int plug1 = getIdx(a);

 int plug2 = getIdx(b);

 addEdge(plug2, plug1, INF);

 }

 cout << m - maxFlow() << "\n";

}

int main(){ BOOST

 int tc;

 cin >> tc;

 bool first = true;

 while(tc--){

 if(!first)

 cout << "\n";

 init();

 solve();

 first = false;

 }

 return 0;

}