Erdos Number

Program:

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

using namespace std;

#define N 5000

char publications [100000];
char authors [N] [50];
map <string, int> authorIndex;
int erdosNumber [N];
bool matrix [N] [N];
bool vis [N];
vector <string> output;

#define ERDOS "Erdos, P."

int currPos;

string parser() {
    if (publications [currPos] == ':') return "";

    int length = (int) strlen(publications);

    string ret = "";

    for (int i = currPos; i < length; i++) {
        ret += publications [i];
        if (publications [i] == '.' && publications [i + 1] == ',') {
            currPos = i + 3;
            return ret;
        } else if (publications [i] == '.' && publications [i + 1] == ':') {
            currPos = i + 1;
            return ret;
        }

    }

    return "";

}
```

```
void reset() {
    memset(matrix, false, sizeof matrix);
    memset(vis, false, sizeof vis);
    memset(erdosNumber, -1, sizeof erdosNumber);
    output.clear();
    authorIndex.clear();
}

int main() {

    int testCases;
    scanf("%d", &testCases);

    int cases = 0;

    while (testCases--) {
        int p, n;
        scanf("%d %d", &p, &n);

        getchar();
        reset();

        int index = 1;

        for (int i = 0; i < p; i++) {
            gets(publications);
            currPos = 0;
            int authorNameLength = 0;
            vector <string> authorList;
            do {
                string authorName = parser();
                authorNameLength = (int) authorName.size();

                if (authorNameLength == 0) break;

                authorList.push_back(authorName);

                if (!authorIndex [authorName]) authorIndex [authorName] = index++
;


            } while (authorNameLength != 0);

            for (int i = 0; i < authorList.size(); i++) {
```

```
                for (int j = i + 1; j < authorList.size(); j++) {
                    matrix [authorIndex[authorList [i]]] [authorIndex[authorList
[j]]] = matrix [authorIndex[authorList [j]]] [authorIndex[authorList [i]]] = true
;
                }
            }
        }

        for (int i = 0; i < n; i++) {
            gets(authors [i]);
            output.push_back(authors [i]);
            if (!authorIndex [authors [i]]) authorIndex [authors [i]] = index++;
        }

        queue< pair<int, int> > q;

        map <string, int>::iterator it;

        for (it = authorIndex.begin(); it != authorIndex.end(); it++) {
            if ((*it).first == ERDOS) {
                q.push(make_pair((*it).second, 0));
                erdosNumber [(*it).second] = 0;
                break;
            }
        }

        while (!q.empty()) {
            pair <int, int> pop = q.front();
            q.pop();

            erdosNumber [pop.first] = pop.second;

            for (int i = 1; i <= index; i++) {
                if (matrix [pop.first] [i] && !vis [i]) {
                    vis [i] = true;
                    q.push(make_pair(i, pop.second + 1));
                }
            }
        }

        printf("\nScenario %d\n", ++cases);

        for (int i = 0; i < output.size(); i++) {
            if (erdosNumber [authorIndex [output [i]]] == -1) {
                printf("%s infinity\n", output [i].c_str());
```

```
            } else {
                printf("%s %d\n", output [i].c_str(), erdosNumber [authorIndex [o
utput [i]]]);
            }

        }

    }

    return 0;
}
```

Output:

```
1
4 3
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors
Erdos, P., Reisig, W.: Stuttering in petri nets
Smith, M.N., Chen, X.: First order derivates in structured programming
Jablonski, T., Hsueh, Z.: Selfstabilizing data structures
Smith, M.N.
Hsueh, Z.
Chen, X.

Scenario 1
Smith, M.N. 1
Hsueh, Z. infinity
Chen, X. 2
```