```
// SPOJ ARDA1
// 2D KMP, Hashing, brute force

unordered_map<string, int>patt;                     // Clear after each Kmp2D call
int flag = 0;                                       // Set to zero before calling PrefixTable

// r : Pattern row
// c : Pattern column
// table : prefix table (1D array)
// s : Pattern String (C++ string)

vector<int> PrefixTable2D(int r, int c, int table[], string s[]) {
    vector<int>Row;                                 // Contains Row mapped string index
    for(int i = 0; i < r; ++i) {
        if(patt.find(s[i]) == patt.end()) {
            patt[s[i]] = ++flag;
            Row.push_back(flag);
        }
        else
            Row.push_back(patt[s[i]]);
    }

    table[0] = -1;
    int i = 0, j = -1;
    while(i < r) {
        while(j >= 0 && Row[i] != Row[j])
            j = table[j];
        ++i, ++j;
        table[i] = j;
    }
    return Row;
}

vector<pair<int, int> > Kmp2D(int StrR, int StrC, int PattR, int PattC, string Str[], string Patt[], int table[]) {
    int mat[StrR][StrC];
    int limC = StrC - PattC;
    vector<int>PattRow = PrefixTable2D(PattR, PattC, table, Patt);

    for(int i = 0; i < StrR; ++i)
        for(int j = 0; j <= limC; ++j) {
            string tmp = Str[i].substr(j, PattC);
            if(patt.find(tmp) == patt.end()) {          // Generating String Mapped using same mapping values
                patt[tmp] = ++flag;                     // Stored in matrix
                mat[i][j] = flag;
            }
            else
                mat[i][j] = patt[tmp];
        }

    vector<pair<int, int> >match;               // This will contain the starting Row & Column of matched string
    for(int c = 0; c <= limC; ++c) {                // Scan columnwise
        int i = 0, j = 0;
        while(i < StrR) {
            while(j >= 0 && mat[i][c] != PattRow[j])
                j = table[j];
            ++i, ++j;
            if(j == PattR)
                match.push_back(make_pair(i-j,c));
        }
    }
    return match;
}

int main() {
    //freopen("in", "r", stdin);

    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int r, c;
    string Patt[310], Str[2003];
    cin >> r >> c;                          // Pattern row and column
    for(int i = 0; i < r; ++i)              // Pattern line by line input
        cin >> Patt[i];
    int table[310];
    vector<int>PattRow = PrefixTable2D(r, c, table, Patt);      // Making PatternPrefixTable

    int R, C;
    cin >> R >> C;                          // String row and column
    for(int i = 0; i < R; ++i)              // String line by line input
        cin >> Str[i];

    vector<pair<int, int> > ans = Kmp2D(R, C, r, c, Str, Patt, table);      // returns the matching points
    sort(ans.begin(), ans.end());
    for(auto it : ans)
        cout << "(" <<it.first+1 << "," << it.second+1 << ")" << endl;
    if(ans.empty())
        cout << "NO MATCH FOUND..." << endl;
    return 0;
}
```

// Prefix Table Automaton

```
void prefixTable(int n, char pat[], int table[]) {
    int len = 0, i = 1;                     // length of the previous longest prefix suffix
    table[0] = 0;                           // table[0] is always 0

    while (i < n) {
        if (pat[i] == pat[len]) {
            len++;
            table[i] = len;
            i++;
        }
        else {                              // pat[i] != pat[len]
            if (len != 0)                   // find previous match
                len = table[len-1];
            else                            // if (len == 0) and mismatch
                table[i] = 0, i++;          // set table[i] = 0, and go to next index
        }
    }
}
```