```
#include <stdlib.h>
#include <iostream>
#include <vector>
#include <climits>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>
#include <algorithm>
using namespace std;

typedef boost::adjacency_list_traits<boost::vecS, boost::vecS,
boost::directedS> traits;
typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS,
boost::no_property,
                              boost::property<boost::edge_capacity_t, long,

boost::property<boost::edge_residual_capacity_t, long,

boost::property<boost::edge_reverse_t, traits::edge_descriptor>>>>
    graph;

typedef traits::vertex_descriptor vertex_desc;
typedef traits::edge_descriptor edge_desc;

class edge_adder
{
    graph &G;

public:
    explicit edge_adder(graph &G) : G(G) {}

    void add_edge(int from, int to, long capacity)
    {
        auto c_map = boost::get(boost::edge_capacity, G);
        auto r_map = boost::get(boost::edge_reverse, G);
        const auto e = boost::add_edge(from, to, G).first;
        const auto rev_e = boost::add_edge(to, from, G).first;
        c_map[e] = capacity;
        c_map[rev_e] = 0;
        r_map[e] = rev_e;
        r_map[rev_e] = e;
    }
};

int main()
{
    std::ios_base::sync_with_stdio(false);
    int t;
    cin >> t;
    while (t-- > 0)
    {
        int lines, letters_per_line;
        string note;
        cin >> lines >> letters_per_line >> note;
        //read input and set the pair correctly
        vector<pair<char, char>> paper(lines * letters_per_line);
        // front page
        for (int i = 0; i < lines * letters_per_line; i++)
        {
            char val;
            cin >> val;
```

```
57              paper[i].first = val;
58          }
59          // back page
60          for (int i = 0; i < lines; i++)
61          {
62              for (int j = letters_per_line - 1; j >= 0; j--)
63              {
64                  char val;
65                  cin >> val;
66                  paper[i * letters_per_line + j].second = val;
67              }
68          }
69
70          if (letters_per_line * lines < note.length())
71          {
72              cout << "No\n";
73          }
74          else
75          {
76              map<pair<char, char>, int> possible_couples; // [char couple,
    occurrency]
77              for (int i = 0; i < lines * letters_per_line; i++)
78              {
79                  if (possible_couples.count({paper[i].second, paper[i].first})
    != 0)
80                      possible_couples[{paper[i].second, paper[i].first}]++;
81                  else
82                      possible_couples[paper[i]]++;
83              }
84
85              graph G(26);
86              edge_adder adder(G);
87              const vertex_desc v_source = boost::add_vertex(G);
88              const vertex_desc v_target = boost::add_vertex(G);
89
90              vector<int> alph(26, 0);
91              //node id, # occurrency in string note
92              for (int i = 0; i < note.length(); i++) // O(length(note))
93                  alph[note[i] - 65]++;
94
95              for (int i = 0; i < 26; i++)
96                  adder.add_edge(i, v_target, alph[i]);
97
98              for (auto it = possible_couples.begin(); it !=
    possible_couples.end(); it++)
99              {
100                 if(alph[it->first.second - 65] != 0 || alph[it->first.first -
    65] != 0){
101                     vertex_desc new_node = boost::add_vertex(G);
102                     adder.add_edge(v_source, new_node, it->second);
103                     if(alph[it->first.first - 65] != 0)
104                         adder.add_edge(new_node, it->first.first - 65,
    it->second);
105                     if(alph[it->first.second - 65] != 0)
106                         adder.add_edge(new_node, it->first.second - 65,
    it->second);
107                 }
108             }
109
110             int flow = boost::push_relabel_max_flow(G, v_source, v_target);
```

```
    // O(V^2 * E)
111             if (flow != note.length())
112                 cout << "No\n";
113             else
114                 cout << "Yes\n";
115         }
116     }
117     return 0;
118 }
```