SPOJ

Herding

Prof. Edson Alves – UnB/FGA

Problema

Oh no! A number of stray cats have been let loose in the city, and as the City Cat Catcher, you have been assigned the vital task of retrieving all of the cats. This is an ideal opportunity to test your latest invention, a cat trap which is guaranteed to retrieve every cat which walks into a square-shaped subsection of the city.

Fortunately, you have the assistance of one of the world's foremost cat psychologists, who has the amazing ability of predicting, given a square subsection of the city, exactly which of the four cardinal directions (north, east, south or west) the cat will head. While this information is handy, you still don't know where all the cats currently are.

In order to prove the cost-effectiveness of your method to the City it would, of course, be important to minimize the number of traps used.

Entrada e saída

Input

The input will begin with a line consisting of two numbers n and m, separated by a space ($1 \leq n, m \leq 1000$). The city will be an $n \times m$ grid of square subsections. The next n lines will each consist of a string of length m, consisting of the letters ‘`N`’, ‘`E`’, ‘`S`’, or ‘`W`’, representing north, east, south and west, respectively. (The first character of the first line will be the northwesternmost point.) The direction in the square is the direction which cats will head if they are in that square. The cat psychologist assures you that cats have no interest in leaving the city.

Output

Output the minimum number of traps needed.

Exemplo de entradas e saídas

Sample Input

```
3 4
SWWW
SEWN
EEEN
```

Sample Output

```
2
```

Solução

- As informações dadas na entrada permitem determinar, a partir de uma célula (x, y) dada, quais são todas as demais células que um gato pode chegar tendo passado por ela
- Interpretando cada célula como um vértice de um grafo, e as arestas as ligações entre a célula atual e o próximo destino do gato, o problema consiste em determinar o número de componentes conectados do grafo
- Isto pode ser feito por meio de uma DFS ou de uma UFDS
- No caso da UFDS, é preciso adicionar um membro extra, que mantenha a contagem do número de componentes conectados distintos
- Esta solução terá complexidade $O(MN \log MN)$

Solução AC com complexidade $O(MN \log MN)$

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

using namespace std;

const int MAX { 1010 };
string A[MAX];

class UFDS {
    std::vector<int> size, ps;
    int count;

public:
    UFDS(int N) : size(N + 1, 1), ps(N + 1), count(N) {
        std::iota(ps.begin(), ps.end(), 0);
    }

    int find_set(int x)
    {
        return x == ps[x] ? x : (ps[x] = find_set(ps[x]));
    }
```

Solução AC com complexidade $O(MN \log MN)$

```
    bool same_set(int x, int y)
    {
        return find_set(x) == find_set(y);
    }

    void union_set(int x, int y)
    {
        if (same_set(x, y))
            return;

        int p = find_set(x);
        int q = find_set(y);

        if (size[p] < size[q])
            std::swap(p, q);

        ps[q] = p;
        size[p] += size[q];
        --count;
    }
```

Solução AC com complexidade $O(MN \log MN)$

```
    int get_count() const { return count; }
};

int solve(int n, int m)
{
    UFDS ufds(n*m);

    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < m; ++j)
        {
            int x = i, y = j;

            switch (A[i][j]) {
            case 'N':
                --x;
                break;

            case 'S':
                ++x;
```

Solução AC com complexidade $O(MN \log MN)$

```
            case 'E':
                ++y;
                break;

            case 'W':
                --y;
                break;
            }

            auto u = j + i*m;
            auto v = y + x*m;
            ufds.union_set(u, v);
        }
    }

    return ufds.get_count();
}
```

Solução AC com complexidade $O(MN \log MN)$

```cpp
int main()
{
    ios::sync_with_stdio(false);

    int n, m;
    cin >> n >> m;

    for (int i = 0; i < n; ++i)
        cin >> A[i];

    cout << solve(n, m) << '\n';

    return 0;
}
```