Union-Find Disjoint Sets

Definição e Implementação: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2020

Sumário

SPOJ – Herding

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;
                break;
```

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();
}

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

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

```

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

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

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

    return 0;
}
```

UVA 1197 – The Suspects

Problema

Severe acute respiratory syndrome (SARS), an atypical pneumonia of unknown aetiology, was recognized as a global threat in mid-March 2003. To minimize transmission to others, the best strategy is to separate the suspects from others.

In the Not-Spreading-Your-Sickness University (NSYSU), there are many student groups. Students in the same group intercommunicate with each other frequently, and a student may join several groups.

To prevent the possible transmissions of SARS, the NSYSU collects the member lists of all student groups, and makes the following rule in their standard operation procedure (SOP).

Once a member in a group is a suspect, all members in the group are suspects.

However, they find that it is not easy to identify all the suspects when a student is recognized as a suspect. Your job is to write a program which finds all the suspects.

Entrada e saída

Input

The input file contains several cases. Each test case begins with two integers n and m in a line, where n is the number of students, and m is the number of groups. You may assume that $0 < n \leq 30000$ and $0 \leq m \leq 500$. Every student is numbered by a unique integer between 0 and $n-1$, and initially student 0 is recognized as a suspect in all the cases. This line is followed by m member lists of the groups, one line per group. Each line begins with an integer k by itself representing the number of members in the group. Following the number of members, there are k integers representing the students in this group. All the integers in a line are separated by at least one space.

A case with $n = 0$ and $m = 0$ indicates the end of the input, and need not be processed.

Output

For each case, output the number of suspects in one line.

Exemplo de entradas e saídas

Sample Input

```
100 4
2 1 2
5 10 13 11 12 14
2 0 1
2 99 2
200 2
1 5
5 1 2 3 4 5
1 0
0 0
```

Sample Output

```
4
1
1
```

Solução com complexidade $O(NM \log NM)$

- A relação "teve contato com um suspeito" é uma relação de equivalência
  (a) O suspeito teve contato consigo mesmo (reflexiva)
  (b) Se o suspeito teve contato com uma pessoa, a pessoa teve contato com o suspeito (simétrica)
  (c) Se o suspeito teve contato com uma pessoa, e a pessoa teve contato com fulano, é considerado que fulano também teve contato com o suspeito
- Assim, para identificar o grupo de pessoas que teve contato com o suspeito, basta usar uma UFDS
- A resposta será o número de elementos do conjunto que contém o suspeito
- A complexidade da solução, no pior caso, é $O(NM \log NM)$

Solução com complexidade $O(NM \log NM)$

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

using namespace std;

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

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

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

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

```

Solução com complexidade $O(NM \log NM)$

```cpp
    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];
    }

    int get_size(int x) { return size[find_set(x)]; }
};

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

Solução com complexidade $O(NM \log NM)$

```
    int n, m;

    while (cin >> n >> m, n | m)
    {
        UFDS ufds(n);

        while (m--)
        {
            int k;
            cin >> k;

            if (k == 0) continue;

            int x;
            cin >> x;
            --k;

            while (k--)
            {
                int y;
                cin >> y;
```

Solução com complexidade $O(NM \log NM)$

```

                ufds.union_set(x, y);
            }
        }

        cout << ufds.get_size(0) << '\n';
    }

    return 0;
}
```

Referências

1. SPOJ – Herding
2. OJ 1197 – The Suspects