SPOJ UCI2009D

Digger Octaves

Prof. Edson Alves – UnB/FGA

Problema

After many years spent playing Digger, little Ivan realized he was not taking advantage of the octaves. Oops, sorry! Most of you were not born when Digger came to light!

Digger is a Canadian computer game, originally designed for the IBM personal computer, back in 1983. The aim of the game is to collect precious gold and emeralds buried deep in subterranean levels of and old abandoned mine.

We Digger gurus call a set of eight consecutive emeralds an octave. Notice that, by consecutive we mean that we can collect them one after another. Your Digger Mobile is able to move in the four directions: North, South, West and East.

In a simplified Digger version, consisting only of emeralds and empty spaces, you will have to count how many octaves are present for a given map.

Entrada e saída

Input

Input starts with an integer T, representing the number of test cases ($1 \leq T \leq 20$). Each test case consists of a map, described as follows:

An integer N ($1 \leq N \leq 8$), representing the side length of the square-shaped map. N lines follow, N characters each. A 'X' character represents an emerald, and a '.' represents an empty space.

Output

For each test case print the number of octaves on a single line.

Exemplo de entradas e saídas

Sample Input

```
2
3
XXX
X.X
XXX
3
XXX
XXX
XXX
```

Sample Output

```
1
5
```

Solução

- Cada cadeia de esmeraldas pode ser identificada por meio do uso do *backtracking*
- É preciso tomar cuidado, porém, para contar apenas cadeias únicas
- Duas cadeias são idênticas se elas passam pelas mesmas coordenadas, indepentemente da ordem de travessia
- Além disso, cada cadeia só pode começar em uma coordenada que contém uma esmeralda
- Também não pode haver repetições de uma mesma coordenada em uma cadeia
- Há, no pior caso, $8^2 = 64 = 2^6$ pontos iniciais possíveis
- Para cada ponto inicial, são $4^7 = 2^{14}$ possíveis cadeias
- Logo há, no máximo, $2^6 \times 2^{14} = 2^{20} \approx 10^6$ caminhos a serem verificados em cada caso de teste

Solução

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

using namespace std;
using ii = pair<int, int>;

set<set<ii>> solutions;
vector<string> A;

bool is_solution(const vector<ii>& xs)
{
    return xs.size() == 8ul;
}

void process_solution(const vector<ii>& xs)
{
    set<ii> s(xs.begin(), xs.end());
    solutions.emplace(s);
}
```

Solução

```
20 vector<ii> candidates(const vector<ii>& xs)
21 {
22     const vector<ii> dirs { ii(0, 1), ii(1, 0), ii(0, -1), ii(-1, 0) };
23     set<ii> used(xs.begin(), xs.end());
24     vector<ii> cs;
25
26     auto p = xs.back();
27
28     for (auto dir : dirs)
29     {
30         auto x = p.first + dir.first, y = p.second + dir.second;
31
32         if (A[x][y] == 'X' and used.count(ii(x, y)) == 0)
33             cs.push_back(ii(x, y));
34     }
35
36     return cs;
37 }
```

Solução

```
39 void backtracking(vector<ii>& xs)
40 {
41     if (is_solution(xs))
42     {
43         process_solution(xs);
44     } else
45     {
46         auto cs = candidates(xs);
47 
48         for (auto c : cs)
49         {
50             xs.push_back(c);
51             backtracking(xs);
52             xs.pop_back();
53         }
54     }
55 }
```

Solução

```
57 size_t solve(int N)
58 {
59     solutions.clear();
60     vector<ii> xs;
61
62     for (int x = 1; x <= N; ++x)
63         for (int y = 1; y <= N; ++y)
64         {
65             if (A[x][y] != 'X')
66                 continue;
67
68             xs.push_back(ii(x, y));
69             backtracking(xs);
70             xs.pop_back();
71         }
72
73     return solutions.size();
74 }
```

Solução

```
76 int main()
77 {
78     ios::sync_with_stdio(false);
79
80     int T;
81     cin >> T;
82
83     while (T--)
84     {
85         int N;
86         cin >> N;
87
88         A = vector<string>(N + 2, string(N + 2, ' '));
89
90         for (int i = 1; i <= N; ++i)
91         {
92             string line;
93             cin >> line;
```

Solução

```
95              for (int j = 1; j <= N; ++j)
96                  A[i][j] = line[j - 1];
97          }
98
99          cout << solve(N) << endl;
100     }
101
102     return 0;
103 }
```