Grafos

Breath-First Search: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

Codeforces Round #470 (rated, Div. 2, based on VK Cup 2018 (Round 1)

Problema

Bob is a farmer. He has a large pasture with many sheep. Recently, he has lost some of them due to wolf attacks. He thus decided to place some shepherd dogs in such a way that all his sheep are protected.

The pasture is a rectangle consisting of $R \times C$ cells. Each cell is either empty, contains a sheep, a wolf or a dog. Sheep and dogs always stay in place, but wolves can roam freely around the pasture, by repeatedly moving to the left, right, up or down to a neighboring cell. When a wolf enters a cell with a sheep, it consumes it. However, no wolf can enter a cell with a dog.

Initially there are no dogs. Place dogs onto the pasture in such a way that no wolf can reach any sheep, or determine that it is impossible. Note that since you have many dogs, you do not need to minimize their number.

Entrada e saída

Input

First line contains two integers R $(1 \leq R \leq 500)$ and C $(1 \leq C \leq 500)$, denoting the number of rows and the numbers of columns respectively.

Each of the following R lines is a string consisting of exactly C characters, representing one row of the pasture. Here, 'S' means a sheep, 'W' a wolf and '.' an empty cell.

Entrada e saída

Output

If it is impossible to protect all sheep, output a single line with the word "No".

Otherwise, output a line with the word "Yes". Then print R lines, representing the pasture after placing dogs. Again, 'S' means a sheep, 'W' a wolf, 'D' is a dog and '.' an empty space. You are not allowed to move, remove or add a sheep or a wolf.

If there are multiple solutions, you may print any of them. You don't have to minimize the number of dogs.

Exemplo de entradas e saídas

Sample Input

```
6 6
..S...
..S.W.
.S....
..W...
...W..
......

1 2
SW
```

Sample Output

```
Yes
..SD..
..SDW.
.SD...
.DW...
DD.W..
......

No
```

Solução com complexidade $O(RC)$

- O problema pode ser modelado como um grafo
- Os vértices são as células da malha, de cada vértice partem até quatro arestas, para cada um dos vizinhos posicionados nas direções cardeais, se existirem
- A travessia adequada é a BFS, uma vez que é necessário avaliar apenas os vizinhos diretos de um nó
- O problema não terá solução de uma ovelha tiver como um vizinho um lobo
- Caso contrário, basta posicionar um cachorro nos vizinhos vazios de uma ovelha
- De fato, como não é necessário minimizar o número de cachorros, basta colocar cachorros em todas as células vazias

Solução AC com complexidade $O(RC)$

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

using namespace std;

struct Point { int x, y; };

const int MAX { 510 };

char M[MAX][MAX];

bool solve(int R, int C)
{
    for (int r = 1; r <= R; ++r)
    {
        for (int c = 1; c <= C; ++c)
        {
            if (M[r][c] != 'S')
                continue;

            vector<Point> ps { { r - 1, c },  { r + 1, c },
                { r, c - 1 }, { r, c + 1 } };
```

Solução AC com complexidade $O(RC)$

```
23              for (const auto& p : ps)
24              {
25                  if (M[p.x][p.y] == 'W')
26                      return false;
27
28                  if (M[p.x][p.y] == '.')
29                      M[p.x][p.y] = 'D';
30              }
31          }
32      }
33
34      return true;
35  }
36
37  int main()
38  {
39      int R, C;
40      scanf("%d %d", &R, &C);
41
42      for (int r = 1; r <= R; ++r)
43          scanf("%s", &M[r][1]);
```

Solução AC com complexidade $O(RC)$

```
44
45      auto ans = solve(R, C);
46
47      if (ans)
48      {
49          printf("Yes\n");
50
51          for (int r = 1; r <= R; ++r)
52              printf("%s\n", &M[r][1]);
53      } else
54          printf("No\n");
55
56      return 0;
57  }
```

UVA 10687 – Monitoring the Amazon

Problema

A network of autonomous, battery-powered, data acquisition stations has been installed to monitor the climate in the region of Amazon. An order-dispatch station can initiate transmission of instructions to the control stations so that they change their current parameters. To avoid overloading the battery, each station (including the order-dispatch station) can only transmit to two other stations. The destinataries of a station are the two closest stations. In case of draw, the first criterion is to chose the westernmost (leftmost on the map), and the second criterion is to chose the southernmost (lowest on the map).

You are commissioned by Amazon State Government to write a program that decides if, given the localization of each station, messages can reach all stations.

Entrada e saída

Input

The input consists of an integer N, followed by N pairs of integers X_i, Y_i, indicating the localization coordinates of each station. The first pair of coordinates determines the position of the order-dispatch station, while the remaining $N-1$ pairs are the coordinates of the other stations. The following constraints are imposed: $-20 \leq X_i, Y_i \leq 20$, and $1 \leq N \leq 1000$. The input is terminated with $N = 0$.

Output

For each given expression, the output will echo a line with the indicating if all stations can be reached or not (see sample output for the exact format).

Exemplo de entradas e saídas

Sample Input

```
4
1 0 0 1 -1 0 0 -1
8
1 0 1 1 0 1 -1 1 -1 0 -1 -1 0 -1 1 -1
6
0 3 0 4 1 3 -1 3 -1 -4 -2 -5
0
```

Sample Output

```
All stations are reachable.
All stations are reachable.
There are stations that are unreachable.
```

Solução com complexidade $O(TN^2)$

- Observe que a transmissão se inicia na estação 1, e vai se propagando para as demais estações
- Também é importante notar a restrição de retransmissão para, no máximo, duas outras estações, sendo estas as mais próximas possíveis
- Estas condições do problema fazem com que a travessia mais apropriada seja a BFS
- Como o grafo é complexo, cada travessia teria complexidade $O(N^2)$ no pior caso
- Porém, a restrição para apenas duas retransmissões reduz a complexidade da travessia para $O(N)$
- A complexidade $O(N^2)$ se dá pelo processo de identificação das estações mais próximas

Solução com complexidade $O(TN^2)$

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

using namespace std;

struct Point { int x, y; };

struct Edge
{
    int d, x, y, i;

    bool operator<(const Edge& e) const
    {
        if (d != e.d)
            return d < e.d;

        if (x != e.x)
            return x < e.x;

        return y < e.y;
    }
};
```

Solução com complexidade $O(TN^2)$

```
22
23 const int MAX { 1010 };
24
25 vector<Edge> adj[MAX];
26 bitset<MAX> visited;
27
28 void bfs(size_t u)
29 {
30     queue<int> q;
31
32     q.push(u);
33     visited[u] = true;
34
35     while (not q.empty())
36     {
37         auto v = q.front();
38         q.pop();
39
40         for (int k = 0; k < 2; ++k)
41         {
42             auto w = adj[v][k].i;
```

Solução com complexidade $O(TN^2)$

```
43
44              if (not visited[w])
45              {
46                  visited[w] = true;
47                  q.push(w);
48              }
49          }
50      }
51  }
52
53  bool solve(int N, const vector<Point>& ps)
54  {
55      for (int p = 1; p <= N; ++p)
56          adj[p].clear();
57
58      for (int p = 1; p <= N; ++p)
59      {
60          auto P = ps[p];
61
```

Solução com complexidade $O(TN^2)$

```
62          for (int q = p + 1; q <= N; ++q)
63          {
64              auto Q = ps[q];
65
66              auto d2 = (P.x - Q.x)*(P.x - Q.x) + (P.y - Q.y)*(P.y - Q.y);
67              adj[p].push_back(Edge { d2, Q.x, Q.y, q });
68              adj[q].push_back(Edge { d2, P.x, P.y, p });
69          }
70      }
71
72      for (int P = 1; P <= N; ++P)
73          sort(adj[P].begin(), adj[P].end());
74
75      visited.reset();
76      bfs(1);
77
78      return (int) visited.count() == N;
79  }
80
```

Solução com complexidade $O(TN^2)$

```
81 int main()
82 {
83     ios::sync_with_stdio(false);
84
85     int N;
86
87     while (cin >> N, N)
88     {
89         vector<Point> ps(N + 1);
90
91         for (int i = 1; i <= N; ++i)
92             cin >> ps[i].x >> ps[i].y;
93
94         auto ans = solve(N, ps);
95
96         cout << (ans ? "All stations are reachable." :
97             "There are stations that are unreachable.") << '\n';
98     }
99
100    return 0;
101 }
```

Referências

1. UVA 10687 – Monitoring the Amazon
2. Codeforces Round #470 (rated, Div. 2, based on VK Cup 2018 (Round 1)