Maze escape

locked

Problem | Submissions | Leaderboard | Discussions

Submissions: 72
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Вие се намирате в лабиринт, от който трябва да се измъкнете възможно най-бързо. Лабиринтът е разделен на клетки (може да си го представим като матрица с N реда и M колони), всяка от които е или празна ('.') или е стена ('#'). Все още не можете да минавате през стени и се налага да се придвижвате само през празните клетки, като преминаването от една клетка в друга отнема, точно 1 секунда.

Тъй като, това да излезете най-бързо от въпросния лабиринт, не би било твърде голямо предизвикателство, вие разполагате с отварачка на портали. Възпросната отварачка ви позволява да отворите портал, от клетката в която се намирате, до някоя от стените на лабиринта, до която имате права видимост. Тъй като това е все още ранен прототип, отварачката може да отваря портали само по права линия от текущото ви местоположение и самите портали стоят отворени само 1 секунда (т.е ако не минете през портала, а отидете в друга празна клетка, порталът се затваря).

Намерете най-краткото време за което можете да излезете от лабиринта при дадени начална клетка и изход.

Input Format

На първият ред на входа се въвеждат N и M - броят редове и броят колони на матрицата. Следват N реда с по M символа от азбуката ('.', '#', 'S', 'F'), където 'S' е началната клетка, а 'F' - изхода от лабиринта.

Constraints

0 < N, M <= 1 000

В 50% от тестовете 0 <= N, M <= 100.

Output Format

Изведете едно число - минималното време за достигане на изхода от лабиринта. Ако изходът е недостижим, изведете -1.

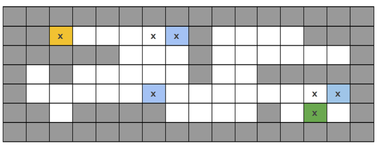
Sample Input 0

```
7 16
################
##S.....#....###
#####...#.....#
#.#.....#.#####
#.............#
##.####...#.F.#
################
```

Sample Output 0

```
6
```

Explanation 0



Начало
Отворени портали
Изход
x Клетки включени в пътя

Sample Input 1

```
5 5
#####
#S#.#
#.#.#
#.##F
#####
```

Sample Output 1

```
-1
```

Current Buffer (saved locally, editable) C++

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <list>
#include <string>
#include <queue>
using namespace std;
struct Node {
  list<int> neighbours;

  bool hasNeighbour(int index) {
    for (auto i : neighbours) {
      if (i == index) {
        return true;
      }
    }
    return false;
  }

  void addNeighbour(int index) {
    neighbours.push_back(index);
  }
};

class Graph {
private:
  vector<Node> nodes;

public:
  Graph(int nodeCount = 0) {
    nodes.resize(nodeCount);
  }

  void _BFS(int start, vector<bool> & visited,vector<int> &distance) {

    queue<int> nextToProcess;
    nextToProcess.push(start);
    visited[start] = true;
    distance[start]=0;
    while(!nextToProcess.empty()) {
      int current = nextToProcess.front();
      nextToProcess.pop();

      for (auto neighbour : nodes[current].neighbours) {
        if (!visited[neighbour]) {
          nextToProcess.push(neighbour);
          distance[neighbour]=distance[current]+1;
          visited[neighbour] = true;
        }
      }
    }
  }
  void BFS(int start,vector<int> &distance) {
    vector<bool> visited;
    visited.resize(nodes.size(), false);

    _BFS(start, visited,distance);
  }
  void print() const {
    for (int node = 0; node < nodes.size(); node++) {
      cout << node << ": ";
      for (auto neighbour : nodes[node].neighbours) {
        cout << neighbour << ", ";
      }
      cout << "\n";
    }
  }

  void connect(int from, int to) {
    if (!nodes[from].hasNeighbour(to) && from!=to) {
      nodes[from].addNeighbour(to);
      nodes[to].addNeighbour(from);
    }
  }
  void connectOne(int from, int to) {
    if (!nodes[from].hasNeighbour(to) && from!=to) {
      nodes[from].addNeighbour(to);

    }
  }
};

int main() {
  int heigth;
  int width;
  cin>>heigth;
  cin>>width;
  Graph g(heigth*width);
  char **matrix=new char*[heigth];
  for(int i=0;i<heigth;i++){
    matrix[i]=new char[width];
  }
  for(int i=0;i<heigth;i++){
    string row;
    cin>>row;
    for(int k=0;k<width;k++){
      matrix[i][k]=row[k];
    }
  }
  int start;
  int end;
    for(int i=0;i<heigth;i++){
    for(int k=0;k<width;k++){
      if(matrix[i][k]=='S'){
        start=width*i+k;
      }
      if(matrix[i][k]=='F'){
        end=width*i+k;
      }
      if(matrix[i][k]=='.'||matrix[i][k]=='S'||matrix[i][k]=='F'){
        int line=1;
        while(matrix[i][k+line]!='#'){
          line++;
        }
        g.connectOne(width*i+k,width*i+k+line-1);
        line=1;

        while(matrix[i][k-line]!='#'){
          line++;
        }
        g.connectOne(width*i+k,width*i+k-line+1);

        line=1;
        while(matrix[i+line][k]!='#'){
          line++;
        }
        g.connectOne(width*i+k,width*(i+line-1)+k);

        line=1;
        while(matrix[i-line][k]!='#'){
          line++;
        }
        g.connectOne(width*i+k,width*(i-line+1)+k);
        if(matrix[i][k+1]=='.'){
          g.connect(width*i+k,width*i+k+1);
        }
        if(matrix[i][k-1]=='.'){
          g.connect(width*i+k,width*i+k-1);
        }
        if(matrix[i+1][k]=='.'){
          g.connect(width*i+k,width*(i+1)+k);
        }
        if(matrix[i-1][k]=='.'){
          g.connect(width*i+k,width*(i-1)+k);
        }
      }
    }
  }
  vector<int> distance(width*heigth,-1);
  g.BFS(start,distance);
  cout<<distance[end];
    return 0;
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code