Codeforces Round #516 (Div. 1)

Problem B – Labyrinth

Prof. Edson Alves

Faculdade UnB Gama

You are playing some computer game. One of its levels puts you in a maze consisting of n lines, each of which contains m cells. Each cell either is free or is occupied by an obstacle. The starting cell is in the row r and column c. In one step you can move one square up, left, down or right, if the target cell is not occupied by an obstacle. You can't move beyond the boundaries of the labyrinth.

Unfortunately, your keyboard is about to break, so you can move left no more than x times and move right no more than y times. There are no restrictions on the number of moves up and down since the keys used to move up and down are in perfect condition.

Now you would like to determine for each cell whether there exists a sequence of moves that will put you from the starting cell to this particular one. How many cells of the board have this property?

Você está jogando um jogo de computador. Um dos níveis do jogo o coloca em um labirinto composto por n linhas, cada uma delas contendo m celas. Cada cela ou está livre ou está ocupada por um obstáculo. A cela de partida está na linha r, coluna c. A cada passo você pode se mover um quadrado para cima, esquerda, baixo ou direita, se a cela alvo não estiver ocupada por um obstáculo. Você não pode ser mover além dos limites do labirinto.

Infelizmente, seu teclado está prestes a quebrar, de modo que você pode ser mover para a esquerda no máximo x vezes e se mover para a direita no máximo y vezes. Não há restrição quanto ao número de movimentos para cima ou para baixo, uma vez que as teclas utilizadas para se mover para cima ou para baixo estão em perfeitas condições.

Agora você gostaria de determinar, para cada cela, se existe uma sequência de movimentos que permitem atingir tal cela a partir da cela de partida. Quantas celas do tabuleiro tem esta propriedade?

Input

The first line contains two integers n, m $(1 \leq n, m \leq 2000)$ ***– the number of rows and the number columns in the labyrinth respectively.***

The second line contains two integers r, c $(1 \leq r \leq n, 1 \leq c \leq m)$ ***– index of the row and index of the column that define the starting cell.***

The third line contains two integers x, y $(0 \leq x, y \leq 10^9)$ ***– the maximum allowed number of movements to the left and to the right respectively.***

Entrada

A primeira linha contém dois inteiros n, m $(1 \leq n, m \leq 2000)$ – o número de linhas e o número de colunas no labirinto, respectivamente.

A segunda linha contém dois inteiros r, c $(1 \leq r \leq n, 1 \leq c \leq m)$ – os índices da linha e da coluna que definem a cela de partida.

A terceira linha contém dois inteiros x, y $(0 \leq x, y \leq 10^9)$ – o número máximo de movimentos permitidos para a esquerda e para a direita, respectivamente.

The next n lines describe the labyrinth. Each of them has length of m and consists only of symbols '$.$' and '$*$'. The j-th character of the i-th line corresponds to the cell of labyrinth at row i and column j. Symbol '$.$' denotes the free cell, while symbol '$*$' denotes the cell with an obstacle.

It is guaranteed, that the starting cell contains no obstacles.

Output

Print exactly one integer – the number of cells in the labyrinth, which are reachable from starting cell, including the starting cell itself.

As próximas n linhas descrevem o labirinto. Cada uma delas tem tamanho m e são compostas apenas pelos símbolos '.' e '*'. O j-ésimo caractere da i-ésima linha corresponde a cela do labirinto que ocupa a linha i e a coluna j. O símbolo '.' indica uma cela livre, enquanto que o símbolo '*' indica uma cela com um obstáculo.

É garantido que a cela de partida não contém obstáculos.

Saída

Imprima um único inteiro – o número de celas do labirinto que são alcancáveis a partir da cela de partida, inclusive a própria cela de partida.

Exemplo de entrada e saída

Exemplo de entrada e saída

```
4 5
```

Exemplo de entrada e saída

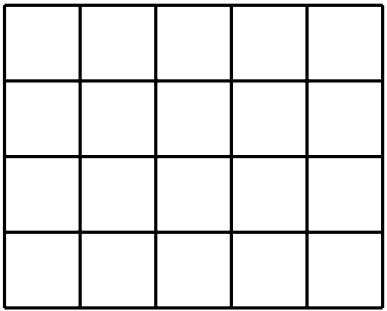
4 5

Exemplo de entrada e saída

4 5

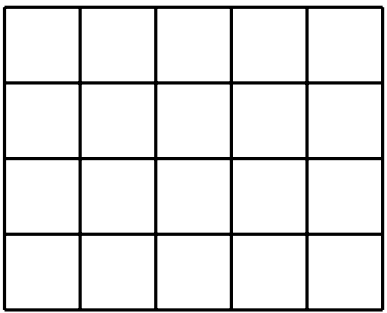
Exemplo de entrada e saída

```
4 5
```



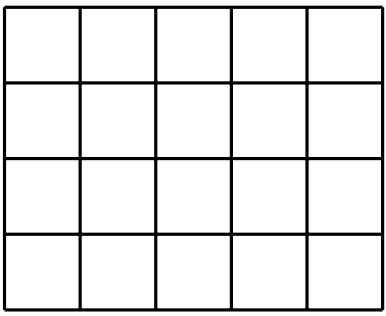
Exemplo de entrada e saída

```
4 5
3 2
```



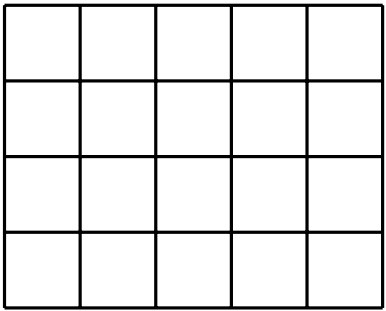
Exemplo de entrada e saída

4 5
3 2



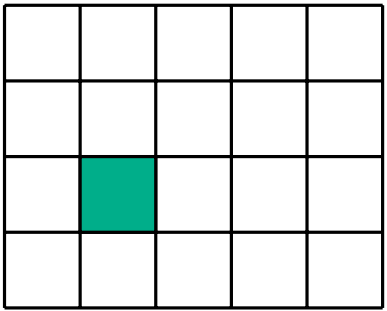
Exemplo de entrada e saída

4 5
3 2
c



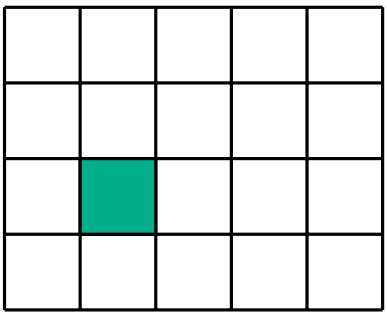
Exemplo de entrada e saída

4 5
3 2
c



Exemplo de entrada e saída

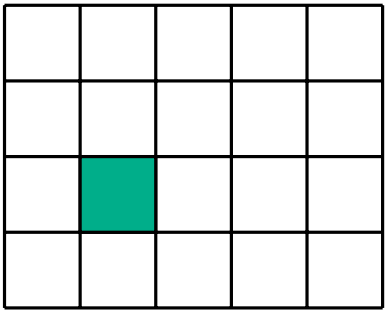
```
4 5
3 2
1 2
```



Exemplo de entrada e saída

```
4 5
3 2
1 2
```

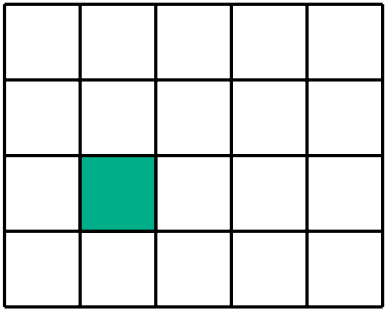
mov. para esquerda



Exemplo de entrada e saída

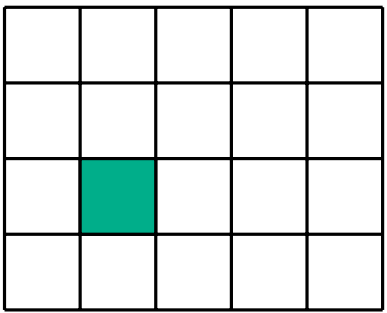
```
4 5
3 2
1 2
```

mov. para direita



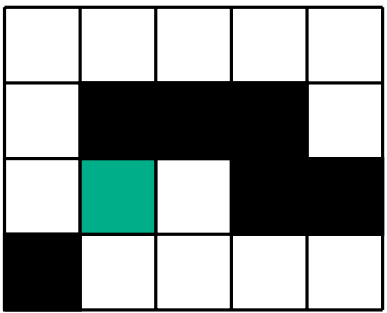
Exemplo de entrada e saída

```
4 5
3 2
1 2
.....
.***.
...**
*....
```



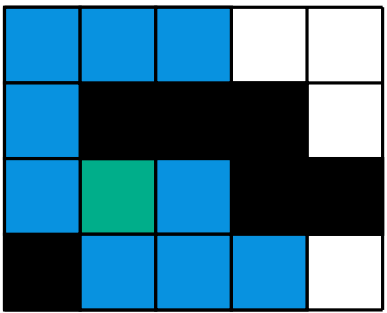
Exemplo de entrada e saída

```
4 5
3 2
1 2
.....
.***.
...**
*....
```



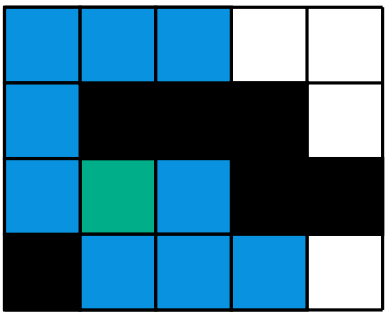
Exemplo de entrada e saída

```
4 5
3 2
1 2
.....
.***.
...**
*....
```



Exemplo de entrada e saída

```
4 5
3 2
1 2
.....
.***.
...**
*....
```



10

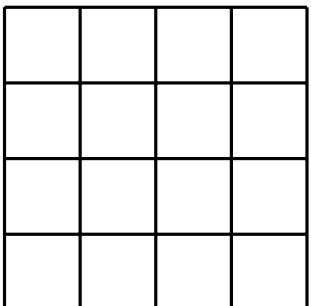
Exemplo de entrada e saída

Exemplo de entrada e saída

```
4 4
```

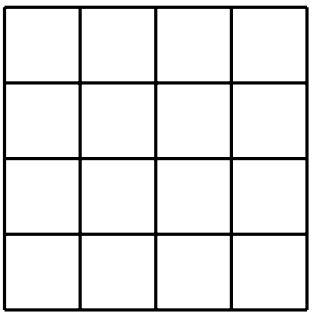
Exemplo de entrada e saída

```
4 4
```



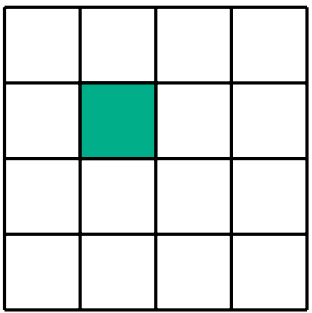
Exemplo de entrada e saída

```
4 4
2 2
```



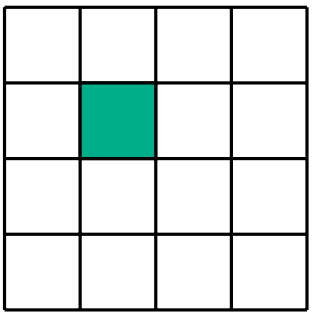
Exemplo de entrada e saída

```
4 4
2 2
```



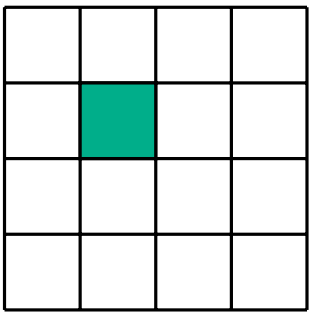
Exemplo de entrada e saída

```
4 4
2 2
0 1
```



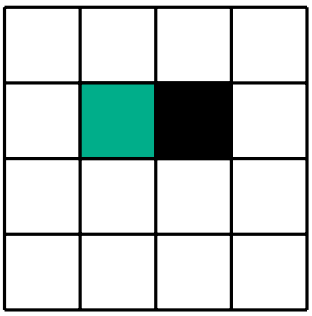
Exemplo de entrada e saída

```
4 4
2 2
0 1
.....
...*.
.....
.....
```



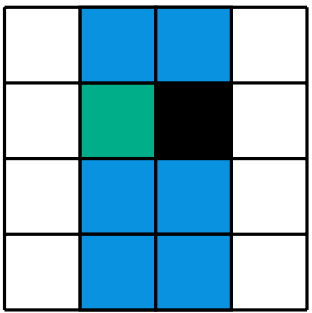
Exemplo de entrada e saída

```
4 4
2 2
0 1
.....
...*.
.....
.....
```



Exemplo de entrada e saída

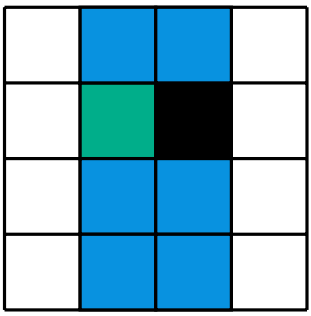
```
4 4
2 2
0 1
.....
...*.
.....
.....
```



Exemplo de entrada e saída

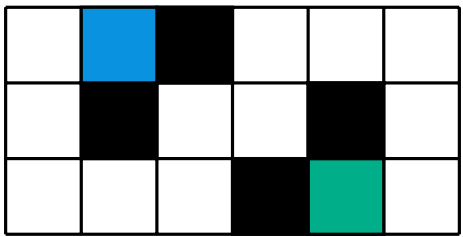
```
4 4
2 2
0 1
.....
...*.
.....
.....
```

7

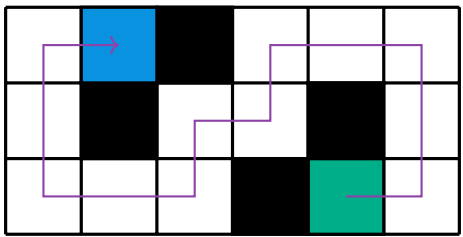


Solução

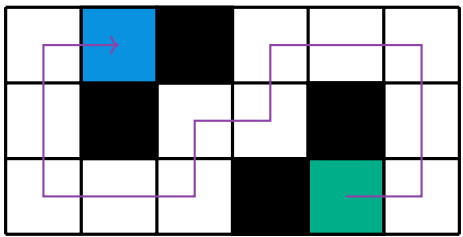
Solução



Solução

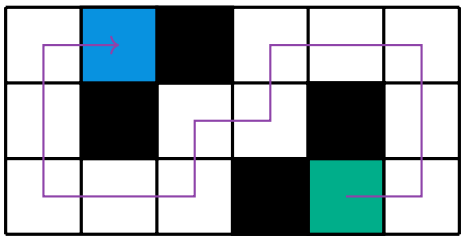


Solução



Considere (x_p, y_p) e (x_q, y_q) os pontos de partida e chegada, respectivamente.

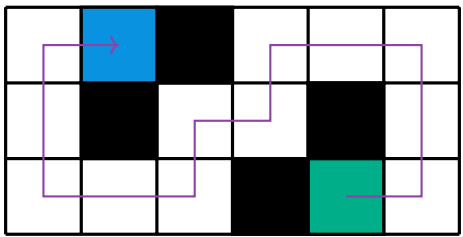
Solução



Se foram dados L passos para a esquerda e R passos a direita, então

$$x_q = x_p + R - L$$

Solução



Se L for minimizado, o valor de R pode ser obtido por meio da expressão

$$R = x_q - x_p + L$$

```cpp
int solve(int N, int M, int R, int C, int X, int Y, const vector<string>& A)
{
    vector<vector<int>> dist(N + 1, vector<int>(M + 1, oo));
    dist[R][C] = 0;

    deque<ii> q;
    q.emplace_back(R, C);

    while (not q.empty()) {
        auto [x, y] = q.front();
        q.pop_front();

        for (auto [dx, dy] : dirs) {
            auto u = x + dx, v = y + dy;

            if (u < 0 or u >= N or v < 0 or v >= M)
                continue;

            if (A[u][v] == '*')
                continue;
```

```
            auto w = dy == -1 ? 1 : 0;

            if (dist[u][v] > dist[x][y] + w) {
                dist[u][v] = dist[x][y] + w;
                w == 0 ? q.emplace_front(u, v) : q.emplace_back(u, v);
            }
        }
    }

    auto ans = 0, yi = C;

    for (int xj = 0; xj < N; ++xj)
        for (int yj = 0; yj < M; ++yj) {
            auto left = dist[xj][yj];
            auto right = left + yj - yi;
            ans += (left <= X and right <= Y ? 1 : 0);
        }

    return ans;
}
```