



**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**

**CEUNES**

**CIÊNCIA DA COMPUTAÇÃO**

**ANA CLARA SESANA MOREIRA, MIKAELE RIBEIRO  
VITORINO, WINNIE BARROS**

**RELATÓRIO**

**JOGOS VORAZES**

**SÃO MATEU - ES**

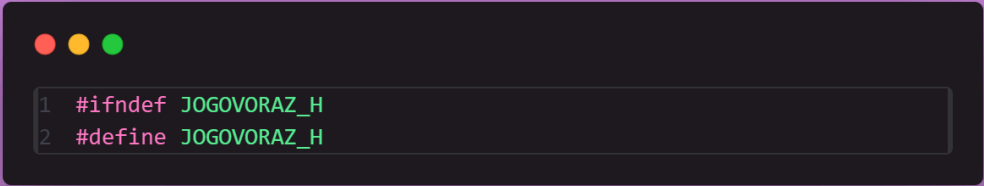
2024

# INTRODUÇÃO

Este relatório apresenta detalhadamente as estruturas implementadas e suas inter-relações para a resolução do problema dos Jogos Vorazes. O código foi organizado em quatro arquivos principais: **'jogovoraz.c'**, **'jogovoraz.h'**, **'main.c'** e **'Makefile'**. Para cada um desses arquivos, será fornecida uma descrição detalhada, acompanhada de trechos do código e explicações claras sobre suas respectivas funções. A divisão do código em múltiplos arquivos foi essencial para a utilização do Makefile, que automatiza o processo de compilação e construção do projeto, tornando-o mais eficiente e organizado. Ao final do relatório, incluímos uma seção dedicada aos casos de teste, onde serão testadas diferentes entradas e as respectivas saídas geradas pelo programa.

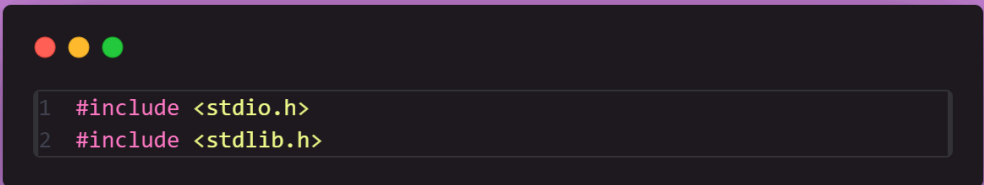
## 1. jogovoraz.h

Este é um arquivo de cabeçalho que contém todas as estruturas e declarações de funções necessárias para trabalhar no labirinto e implementar uma busca em largura, para encontrar o caminho ou verificar condições dentro do labirinto.



```
1 #ifndef JOGOVORAZ_H
2 #define JOGOVORAZ_H
```

Esse trecho do código é conhecido como "inclusion guards", ele serve como uma garantia que o conteúdo do arquivo **jogovoraz.h** será incluído apenas uma vez, evitando conflitos durante a compilação do código



```
1 #include <stdio.h>
2 #include <stdlib.h>
```

Inclusão das bibliotecas padrões para gerenciar a entrada e saída de dados e manipulação de memória

```
1 #define true 1
2 #define false 0
```

Definindo as macros para valores booleanos true e false

```
1 typedef int booleano;
```

Define booleano como tipo int para facilitar o retorno de valores em funções

```
1 typedef struct{
2     int altura;
3     int largura;
4     char **matriz;
5     int tributoX, tributoY;
6 }Labirinto;
```

Estrutura definida para representar o labirinto com altura, largura, matriz (representação do labirinto), e posições do tributo.

```
1 typedef struct auxNo{  
2     int x, y;  
3     struct auxNo* prox;  
4 }No;
```

Estrutura para os nós da fila, que armazena coordenadas (x, y) e um ponteiro para o próximo nó.

```
1 typedef struct{  
2     No *inicio;  
3     No *fim;  
4 }Fila;
```

Estrutura para a fila com ponteiros para o início e o fim.

```
1 typedef struct {  
2     int x, y;  
3     char dir;  
4 } Direcao;
```

Estrutura para armazenar direções com coordenadas (x, y) e um caractere que representa a direção.

```
1 extern Direcao movimentos[4];
```

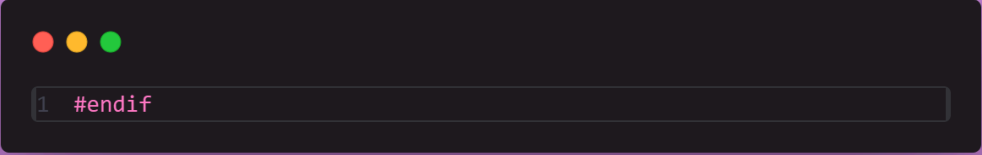
Declaração de um vetor global de direções, para uso em outros arquivos sem defini-los novamente aqui.

```
1 void inicializaFila(Fila *f);  
2 booleano filaVazia(Fila *f);  
3 void insereFila(Fila* f, int x, int y);  
4 No *excluiFila(Fila* f);
```

Funções para manipulação da fila

```
1 booleano posicaoValida(Labirinto *Labirinto, int x, int y);  
2 booleano isEscape(Labirinto *Labirinto, int x, int y);  
3 booleano buscaEmLargura(Labirinto *Labirinto);
```

Funções para manipulação do labirinto e realizar a busca em largura.

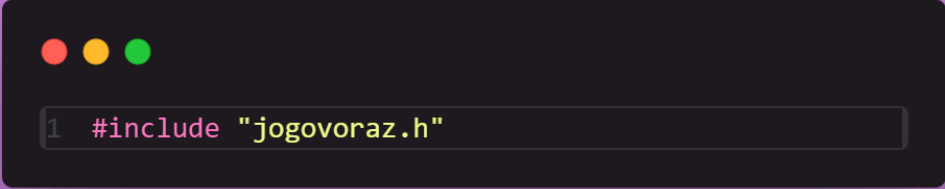
A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `#endif` is written on the first line, preceded by a line number `1`.

```
1 #endif
```

Marca o fim do bloco de código protegido pelas guardas de inclusão.

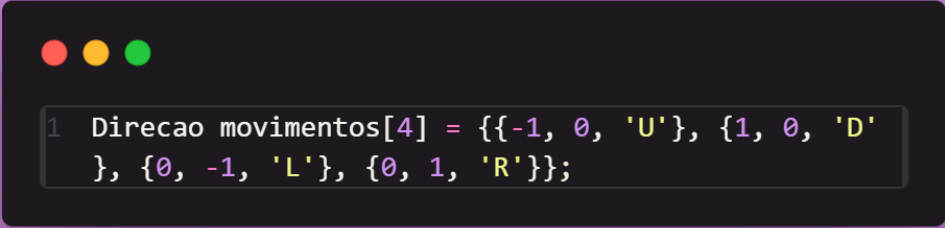
## 2. jogovoraz.c

Neste arquivo estão implementadas todas as funções necessárias para resolução do desafio dos jogos vorazes. Utilizamos uma busca em largura para explorar o labirinto e encontrar um caminho válido que permite o tributo escapar.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `#include "jogovoraz.h"` is written on the first line, preceded by a line number `1`.

```
1 #include "jogovoraz.h"
```

Para a inclusão do arquivo jogovoraz.h neste arquivo .c

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text defines an array of movement directions. The first line is `Direcao movimentos[4] = {{-1, 0, 'U'}, {1, 0, 'D'}}` and the second line is `}, {0, -1, 'L'}, {0, 1, 'R'}}};`. The first line is preceded by a line number `1`.

```
1 Direcao movimentos[4] = {{-1, 0, 'U'}, {1, 0, 'D'}  
    }, {0, -1, 'L'}, {0, 1, 'R'}}};
```

Define um array da estrutura direção para representar os quatro movimentos possíveis: **cima** ('U'), **baixo** ('D'), **esquerda** ('L'), e **direita** ('R'). Cada direção é representada por um par de deslocamentos (x, y) e um caractere indicando a direção.

```
1 void inicializaFila(Fila *f) {  
2     f->inicio = NULL;  
3     f->fim = NULL;  
4 }
```

Função que inicializa a fila. Define os ponteiros de início e fim como NULL, indicando que a fila está vazia.

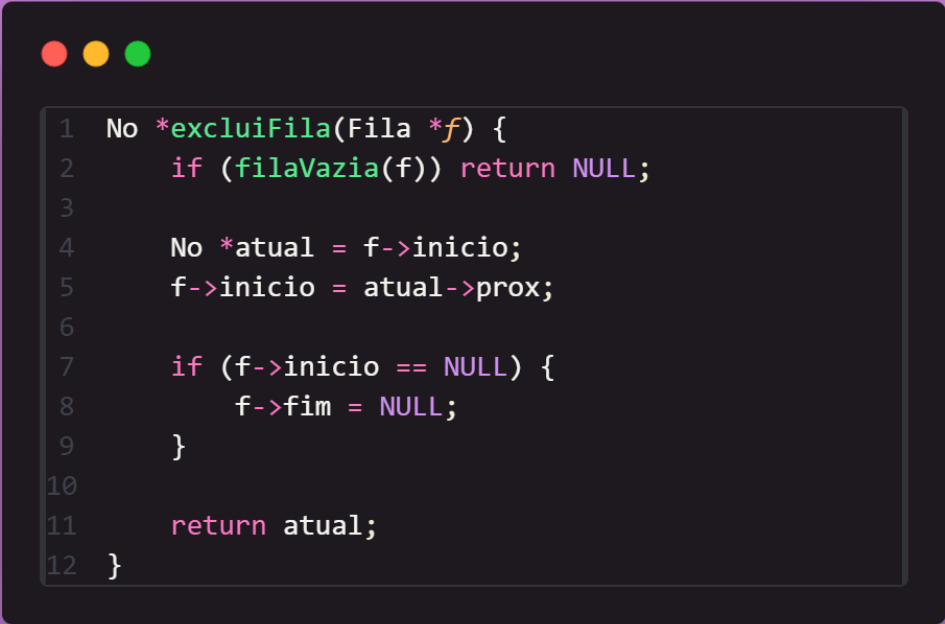
```
1 booleano filaVazia(Fila *f) {  
2     return (f->inicio == NULL);  
3 }
```

Função que verifica se a fila está vazia.



```
1 void insereFila(Fila *f, int x, int y) {
2     No *novo = (No *)malloc(sizeof(No));
3     novo->x = x;
4     novo->y = y;
5     novo->prox = NULL;
6
7     if (f->fim == NULL) {
8         f->inicio = novo;
9         f->fim = novo;
10    } else {
11        f->fim->prox = novo;
12        f->fim = novo;
13    }
14 }
```

Função para inserir um novo nó (posição (x, y)) na fila. É utilizada para inserir posições que serão exploradas durante a busca.



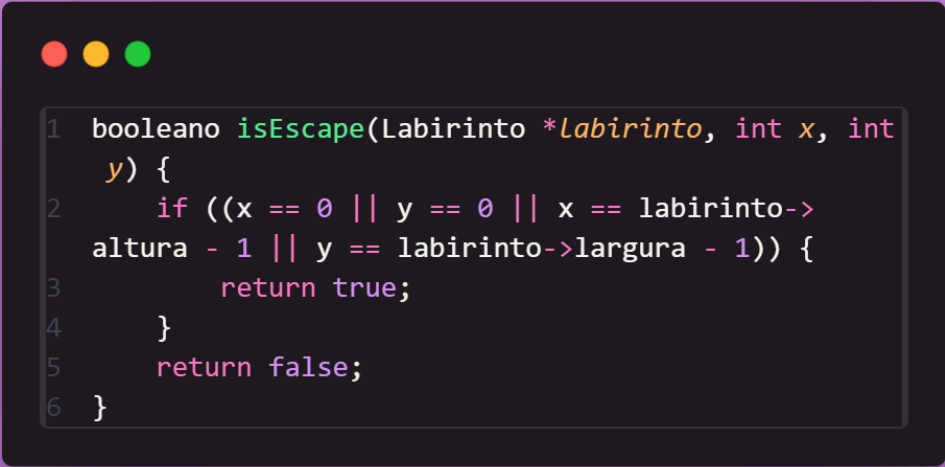
```
1 No *excluiFila(Fila *f) {  
2     if (filaVazia(f)) return NULL;  
3  
4     No *atual = f->inicio;  
5     f->inicio = atual->prox;  
6  
7     if (f->inicio == NULL) {  
8         f->fim = NULL;  
9     }  
10  
11     return atual;  
12 }
```

Função para remover e retornar o nó do início da fila. É utilizada para explorar as próximas posições durante a busca.

```
1 booleano posicaoValida(Labirinto *labirinto, int
  x, int y) {
2     if (x < 0 || y < 0 || x >= labirinto->altura
    || y >= labirinto->largura) return false;
3     if (labirinto->matriz[x][y] == '#') return
    false;
4
5     if (labirinto->matriz[x][y] == 'M') return
    false;
6
7     if (labirinto->matriz[x][y] == 'V') return
    false;
8     if (labirinto->matriz[x][y] == 'M') return
    false;
9
10    return true;
11 }
```

Verifica se uma posição (x, y) no labirinto é válida:

- Não pode estar fora dos limites.
- Não pode ser uma parede (#).
- Não pode ser ocupada por um monstro (M).
- Não pode ser uma posição já visitada (V).



```
1  booleano isEscape(Labirinto *labirinto, int x, int
   y) {
2      if ((x == 0 || y == 0 || x == labirinto->
   altura - 1 || y == labirinto->largura - 1)) {
3          return true;
4      }
5      return false;
6  }
```

Verifica se uma posição (x, y) é uma borda do labirinto, indicando uma possível saída.

- `x == 0`: Verifica se a posição está na borda superior do labirinto.
- `y == 0`: Verifica se a posição está na borda esquerda.
- `x == labirinto->altura - 1`: Verifica se a posição está na borda inferior.
- `y == labirinto->largura - 1`: Verifica se a posição está na borda direita.

```
1  booleano buscaEmLargura(Labirinto *labirinto) {
2      Fila fila;
3      inicializaFila(&fila);
4
5
6      char **direcoes = (char **)malloc(labirinto->altura * sizeof
(char *));
7      for (int i = 0; i < labirinto->altura; i++) {
8          direcoes[i] = (char *)malloc(labirinto->largura * sizeof
(char));
9          for (int j = 0; j < labirinto->largura; j++) {
10             direcoes[i][j] = '\\0';
11         }
12     }
13
14
15     for (int i = 0; i < labirinto->altura; i++) {
16         for (int j = 0; j < labirinto->largura; j++) {
17             if (labirinto->matriz[i][j] == 'M') {
18                 insereFila(&fila, i, j);
19             }
20         }
21     }
22
23
24     insereFila(&fila, labirinto->tributoX, labirinto->tributoY);
25     labirinto->matriz[labirinto->tributoX][labirinto->tributoY]
= 'V';
26
27     while (!filaVazia(&fila)) {
28         No *atual = excluiFila(&fila);
29         int x = atual->x;
30         int y = atual->y;
31         free(atual);
32
33
34         if (labirinto->matriz[x][y] == 'M') {
35             for (int i = 0; i < 4; i++) {
36                 int novoX = x + movimentos[i].x;
37                 int novoY = y + movimentos[i].y;
38
39                 if (posicaoValida(labirinto, novoX, novoY)) {
40                     labirinto->matriz[novoX][novoY] = 'M';
41                     insereFila(&fila, novoX, novoY);
42                 }
43             }
44         }
```

```

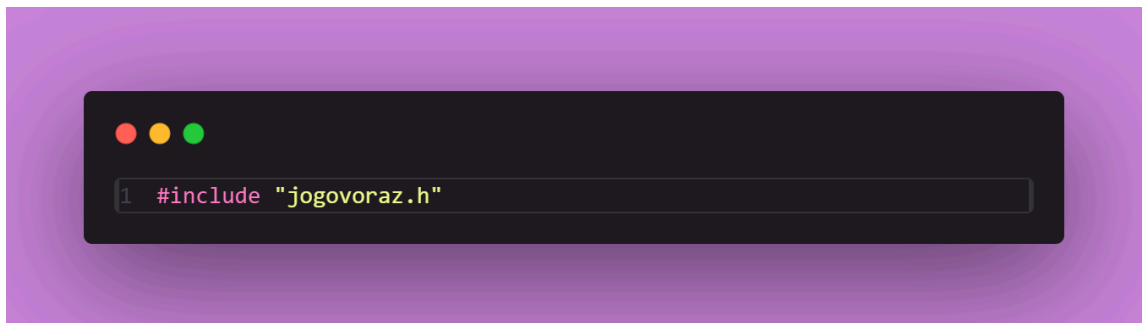
1  else if (labirinto->matriz[x][y] == 'V') {
2      for (int i = 0; i < 4; i++) {
3          int novoX = x + movimentos[i].x;
4          int novoY = y + movimentos[i].y;
5
6          if (posicaoValida(labirinto, novoX, novoY)) {
7              direcoes[novoX][novoY] = movimentos[i].dir;
8
9              labirinto->matriz[novoX][novoY] = 'V';
10             insereFila(&fila, novoX, novoY);
11
12             o
13             if (isEscape(labirinto, novoX, novoY)) {
14                 printf("YES\n");
15
16                 char caminho[labirinto->altura *
17                     labirinto->largura];
18                 int passos = 0;
19                 int caminhoX = novoX, caminhoY = novoY;
20
21                 while (!(caminhoX == labirinto->
22                     tributoX && caminhoY == labirinto->tributoY)) {
23                     caminho[passos++] =
24                     direcoes[caminhoX][caminhoY];
25                     switch (caminho[passos - 1]) {
26                         case 'U':
27                             caminhoX++;
28                             break;
29                         case 'D':
30                             caminhoX--;
31                             break;
32                         case 'L':
33                             caminhoY++;
34                             break;
35                         case 'R':
36                             caminhoY--;
37                             break;
38                     }
39                 }
40                 printf("%d\n", passos);
41                 for (int j = passos - 1; j >= 0; j--) {
42                     printf("%c", caminho[j]);
43                 }
44                 printf("\n");
45                 return true;
46             }
47         }
48     }
49
50     printf("NO\n");
51     return false;
52 }

```

Implementa a busca em largura para encontrar um caminho seguro para o tributo. Primeiro, insere na fila todas as posições dos monstros e a posição inicial do tributo. Depois, movimenta os monstros e, em seguida, tenta mover o tributo. Se o tributo encontrar uma saída, imprime "YES", o número de passos e o caminho seguido. Se nenhum caminho for encontrado, imprime "NO".

### 3. main.c

Neste arquivo está a função main, que serve para a preparação e organização da simulação do labirinto. Ela começa lendo as dimensões e os elementos do labirinto, e determina a posição inicial do tributo. Antes de executar a busca pelo caminho de escape, verifica se o tributo já está em uma borda do labirinto, o que representaria uma saída imediata. Caso contrário, chama a função buscaEmLargura para encontrar uma rota de escape. Após a execução, o código libera a memória alocada dinamicamente para a matriz do labirinto.



Para a inclusão do arquivo jogovoraz.h neste arquivo .c

```
1 int main() {  
2     int n, m;  
3     scanf("%d %d", &n, &m);
```

Define as variáveis que armazenam as dimensões do labirinto: n (altura) e m (largura) e realiza a leitura delas a partir da entrada dos dados

```
1     Labirinto labirinto;  
2     labirinto.altura = n;  
3     labirinto.largura = m;  
4     labirinto.matriz = (char **)malloc(n * sizeof(char *));  
5     for (int i = 0; i < n; i++) {  
6         labirinto.matriz[i] = (char *)malloc(m * sizeof(char))  
7     ;  
    }
```

Inicializa a estrutura do labirinto e define sua altura e largura com os valores lidos. Em seguida, aloca memória para a matriz que representará o labirinto. Essa matriz é um array de arrays, onde cada célula armazena um caractere indicando um elemento do labirinto: chão (.), parede (#), tributo (A), ou monstro (M).



```

1  for (int i = 0; i < n; i++) {
2      for (int j = 0; j < m; j++) {
3          char c;
4          scanf("%c", &c);
5          labirinto.matriz[i][j] = c;
6          if (c == 'A') {
7              labirinto.tributoX = i;
8              labirinto.tributoY = j;
9          }
10     }
11 }

```

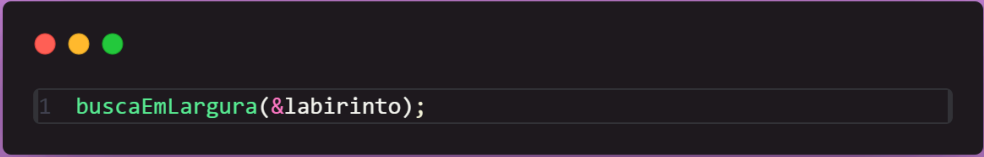
Lê a configuração do labirinto célula por célula e armazena na matriz. Em seguida, procura a posição inicial do tributo (A) e salva suas coordenadas (`tributoX` e `tributoY`) na estrutura do labirinto.

```

1  if (labirinto.tributoX == 0 || labirinto.tributoY == 0 ||
2      labirinto.tributoX == n - 1 || labirinto.tributoY == m - 1
3  ) {
4      printf("YES\n0\n");
5      return 0;
6  }

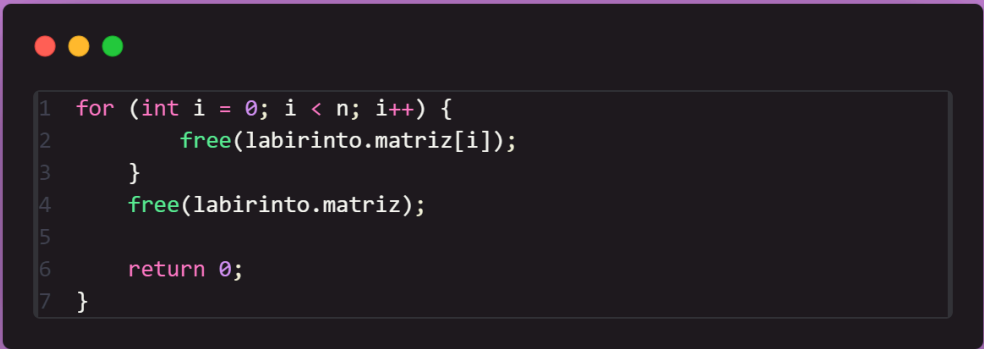
```

Verifica se a posição inicial do tributo já está em uma borda do labirinto (ou seja, se o tributo já está em uma posição de escape). Se sim, imprime "YES" e "0" (indicando que não são necessários passos adicionais) e termina o programa, pois o tributo já está seguro.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a single line of C code: `buscaEmLargura(&labirinto);`.

```
1 buscaEmLargura(&labirinto);
```

Chama a função `buscaEmLargura`, que tentará encontrar um caminho válido para o tributo escapar do labirinto, evitando os monstros. A função imprime a saída apropriada, que será "YES" com o caminho encontrado ou "NO" se não houver um caminho seguro.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains a block of C code for freeing memory: a loop to free each row of the matrix, followed by freeing the matrix itself, and a return statement.


```
1 for (int i = 0; i < n; i++) {  
2     free(labirinto.matriz[i]);  
3 }  
4 free(labirinto.matriz);  
5  
6 return 0;  
7 }
```

Libera a memória alocada para a matriz do labirinto. Primeiro, libera cada linha individual da matriz, e então libera a matriz. A última linha indica que o programa foi executado com sucesso.

#### 4. Makefile

Este é um arquivo Makefile, utilizado para automatizar a compilação do código do labirinto. Esse arquivo define como gerar o executável, compilar os arquivos objetos

(.o) e contém comandos para limpar arquivos intermediários.



```
1 all: executavel
2
3 executavel: main.o jogovoraz.o
4     gcc -o executavel main.o jogovoraz.o
5
6 main.o: main.c jogovoraz.h
7     gcc -o main.o -c main.c -Wall -Werror
8
9 jogovoraz.o: jogovoraz.c jogovoraz.h
10    gcc -o jogovoraz.o -c jogovoraz.c -Wall -Werror
11
12 clean:
13     rm -rf *.o executavel
14
15 run:
16     ./executavel
```

Ao executar **make**:

- Compila os arquivos **main.c** e **jogovoraz.c** em arquivos objeto (**.o**).
- Gera o executável **executavel** a partir dos arquivos objeto.

Alvos auxiliares:

- **clean**: Remove arquivos objeto e o executável, limpando o diretório.
- **run**: Executa o programa recém-compilado.

# CASOS DE TESTE

## 1. Caso 1

entrada:

```
1  3 3
2  .##
3  MA#
4  ###
```

codesnap.dev

saída:



**2. caso 2**

**entrada:**

```
1  14 7
2  #####
3  #.....#
4  #.####.#
5  #.####.#
6  #.....#
7  #A###..
8  #####.#
9  #####.#
10 #####.#
11 #####.#
12 #####.#
13 #####.#
14 #####.#
15 #M.....#
```

saída:

```
1  YES
2  7
3  URRRRDR
```

codesnap.dev

### 3. caso 3

entrada:

```
1  4  4
2  #.#M
3  #.##
4  #A..
5  #####
```

[codesnap.dev](https://codesnap.dev)

saída:



1 YES

2 2

3 UU