IA – 1° SEMESTRE DE 2022

3. EXERCÍCIO TEÓRICO – BUSCA SEM INFORMAÇÃO

Nome: Luiz Gustavo Alves Assis da Silva
RA: 149115

1. Mostre um passo-a-passo da ideia que vc explorou no exercício do URI. Você conseguiu aplicar busca (BFS, DFS), ou usou outra estratégia. Explane as ideias que vc explorou para resolver o exercício, pode incluir imagens, parte do código para explicar. Incluir também um print da porcentagem de aceite do URI.

```
/* - - - - - - - - - - - - - - - - - - - - - - - -
 *    Problema 1
 *    Robôs em uma grade
 *    Tempo Limite: 1 s (C++)
 *
 *    NOME: LUIZ GUSTAVO ALVES ASSIS DA SILVA
 *    RA: 149115
 */

#include <iostream>
#include <string>
#include <queue>

#define INT_MAX 2147483647

using namespace std;

const long long mod = INT_MAX;
long long dp[1004][1004];

string grade[1004];
```

Podemos separar a resolução do exercício principal em dois problemas:

a) Contar o número de caminhos únicos de *s* até *t* podendo se movimentar apenas para direita e baixo.

b) Verificar se existe uma única solução de *s* até *t* podendo se movimentar para direita, esquerda, cima e baixo.

Para o caso a) podemos aplicar o método de programação dinâmica para encontrar a resolução desse problema de forma otimizada. Sabendo que o limite de linhas varia entre: $1 \leq n \leq 1000$, definimos uma matriz global "long long dp[1004][1004]" para armazenar o resultado dos subproblemas e encontrar a solução do problema principal por meio da memorização (bottom-up).

Além disso, devido a limitação física do computador, o exercício informou que o programa não é capaz de lidar com números muito grandes e, portanto, a resposta para o problema a) deve ser dada em módulo de $2^{31} - 1$, o valor máximo que um inteiro de 32-bits pode assumir, portanto, definimos uma variável global "const long long mod" no qual será atribuído o valor máximo de um inteiro de 32-bits. Dessa forma, temos que o algoritmo para obter a solução de forma eficiente do problema a) é dado por:

```
int countCaminhos(int num_linhas)
{
    dp[0][0] = 1;

    for (int i = 0; i < num_linhas; i++)
    {
        for (int j = 0; j < num_linhas; j++)
        {
            if (i == 0 && j == 0)
                continue;

            dp[i][j] = 0;

            if (grade[i][j] == '.')
            {
                if (j - 1 >= 0)
                    dp[i][j] = (dp[i][j] + dp[i][j - 1]) % mod;

                if (i - 1 >= 0)
                    dp[i][j] = (dp[i][j] + dp[i - 1][j]) % mod;
            }
        }
    }
    return dp[num_linhas - 1][num_linhas - 1];
}
```

O algoritmo inicialmente atribuirá o valor 1 para a posição inicial da matriz de memorização e percorrerá um loop aninhado atribuindo o valor 0 na posição atual da matriz. Posteriormente, é feito 3 checagens para verificar se a posição atual da grade é válida e se os valores das variáveis *i* e *j* são positivos (evitando, assim, um acesso indevido na memória da matriz). Em seguida, é feito o cálculo dos subproblemas e o resultado será extraído em módulo de $2^{31} - 1$.

Ao fim, será retornado o valor da última posição da matriz de memorização, caso o valor retornado for maior que 0, o resultado da contagem é impresso e o programa é encerrado, do contrário, é necessário verificar se existe uma única solução de s até t.

```
int num_caminhos = countCaminhos(num_linhas);
if (num_caminhos > 0)
{
    cout << num_caminhos;
}
else
{
    bool flag = BFS(num_linhas);

    if (flag)
        cout << "THE GAME IS A LIE";

    else
        cout << "INCONCEIVABLE";
}
return 0;
```

Para o caso b) podemos aplicar o método de busca em largura (BFS) para encontrar o caminho mínimo de *s* até *t* e resolver o problema. O algoritmo testará todas as posições possíveis, legais e que ainda não foram visitadas pela busca para cada movimento (direita, esquerda, baixo e cima, respectivamente). Dessa forma, temos que o algoritmo para obter a solução do problema b) é dado por:

```
bool BFS(int num_linhas)
{
    queue<pair<int, int> > fila;
    fila.push(make_pair(0, 0));

    /* FLAG PARA VERIFICAR POSIÇÕES JÁ VISITADAS */
    bool visitado[num_linhas][num_linhas] = { {false} };
    visitado[0][0] = true;

    /* VALORES PARA DIREITA, ESQUERDA, BAIXO, CIMA (RESPECTIVAMENTE) */
    int coluna[] = {1, -1, 0, 0};
    int linha[] = {0, 0, 1, -1};

    while (!fila.empty())
    {
        pair<int, int> coordenada = fila.front();
        fila.pop();

        /* VERIFICAR SOLUÇÃO */
        if (coordenada.first == num_linhas - 1 && coordenada.second == num_linhas - 1)
            return true;

        /* ITERARANDO EM CADA DIREÇÃO */
        for (int i = 0; i < 4; i++)
        {
            int x = coordenada.first + coluna[i];
            int y = coordenada.second + linha[i];

            if (x >= 0 && x < num_linhas && y >= 0 && y < num_linhas && !visitado[x][y] && grade[x][y] == '.')
            {
                visitado[x][y] = true;
                fila.push(make_pair(x, y));
            }
        }
    }
    return false;
}
```

Primeiramente, é utilizado uma estrutura de dados FIFO (first-in-first-out), a exemplo da fila, para armazenar as posições legais durante a busca e, ainda, criar uma matriz booleana (flags) para verificar e evitar repetição de caminhos. Em seguida, é criado dois vetores que armazenam todos os valores de movimentação do robô.

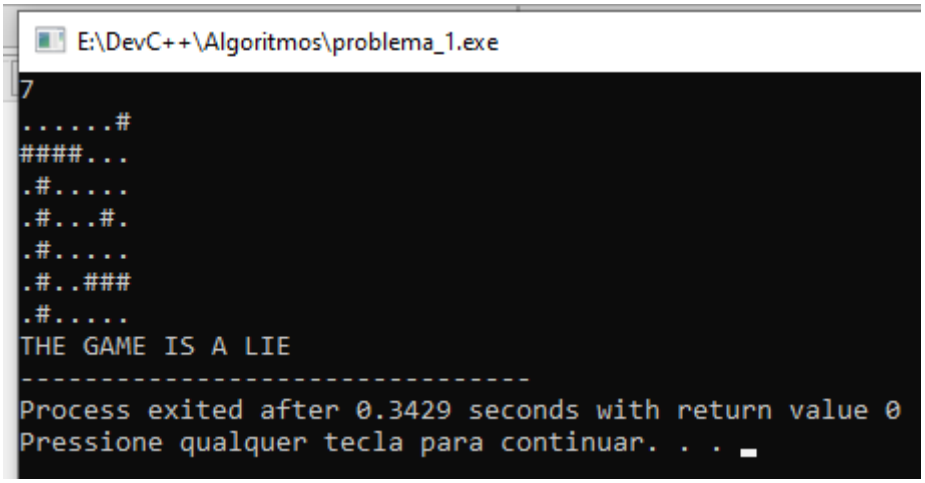
O algoritmo entra em um loop em que a condição de parada é satisfeita quando a fila estiver vazia – ou seja, todas as posições da grade foram visitadas e a solução não foi encontrada, portanto, a função retorna o valor *false* e o programa é encerrado imprimindo a mensagem: "INCONCEIVABLE".

Dentro do loop é feito uma checagem para verificar se foi encontrado um caminho de *s* até *t* e, caso verdadeiro, é retornado o valor *true* e o programa é encerrado imprimindo a mensagem: "THE GAME IS A LIE".

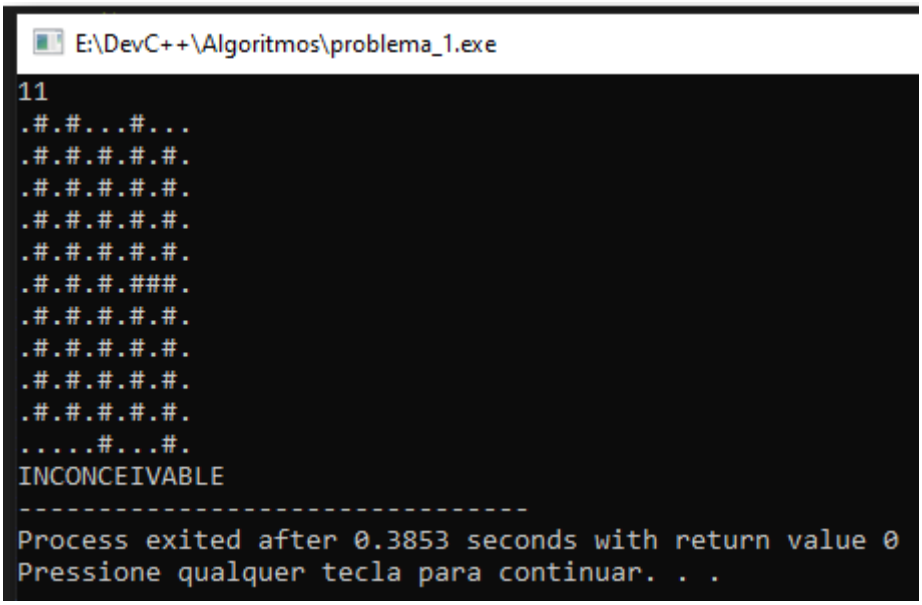
Caso a condição acima não é satisfeita, o algoritmo entra em outro loop iterando em cada direção de movimento checando se o movimento é valido, legal e que ainda não foi visitado pela busca. Por fim, a posição é armazenada simultaneamente na fila e na matriz booleana em que será atribuída o valor *true* para ela.

Acessando o site: https://www.udebug.com/URI/3247 é possível verificar 4 exemplos de entrada e saída para o exercício proposto, portanto, utilizando o programa listado temos que as saídas desses 4 exemplos são dadas, respectivamente por:

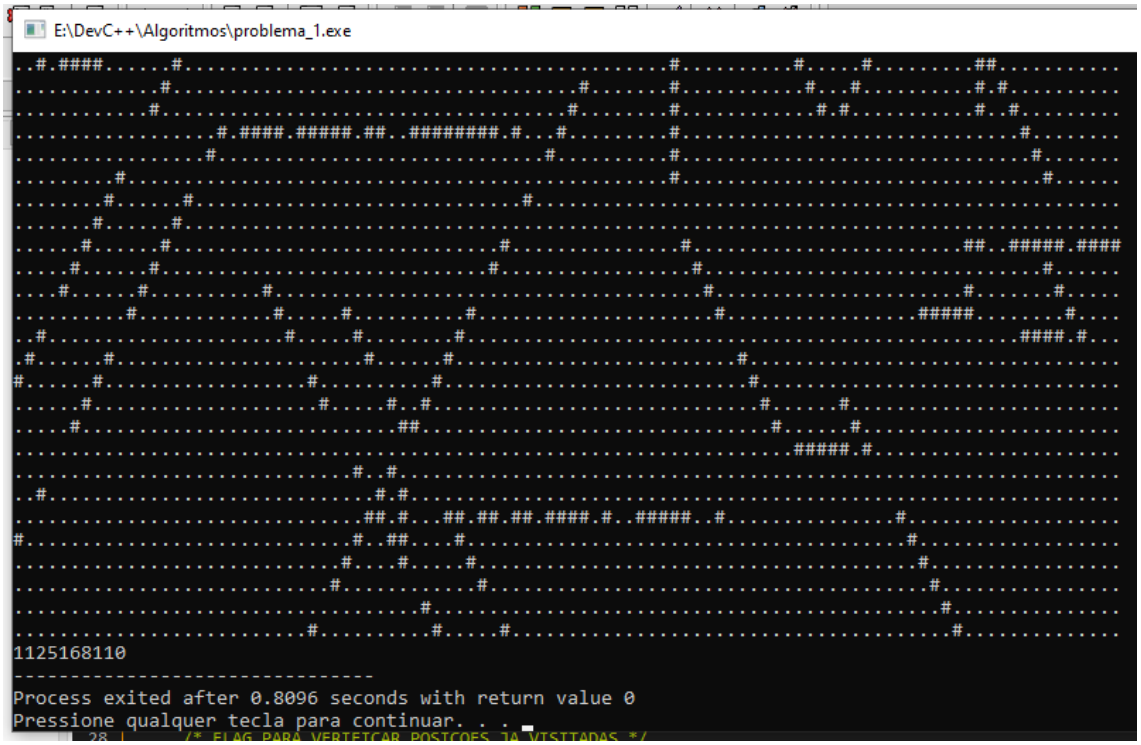
1)



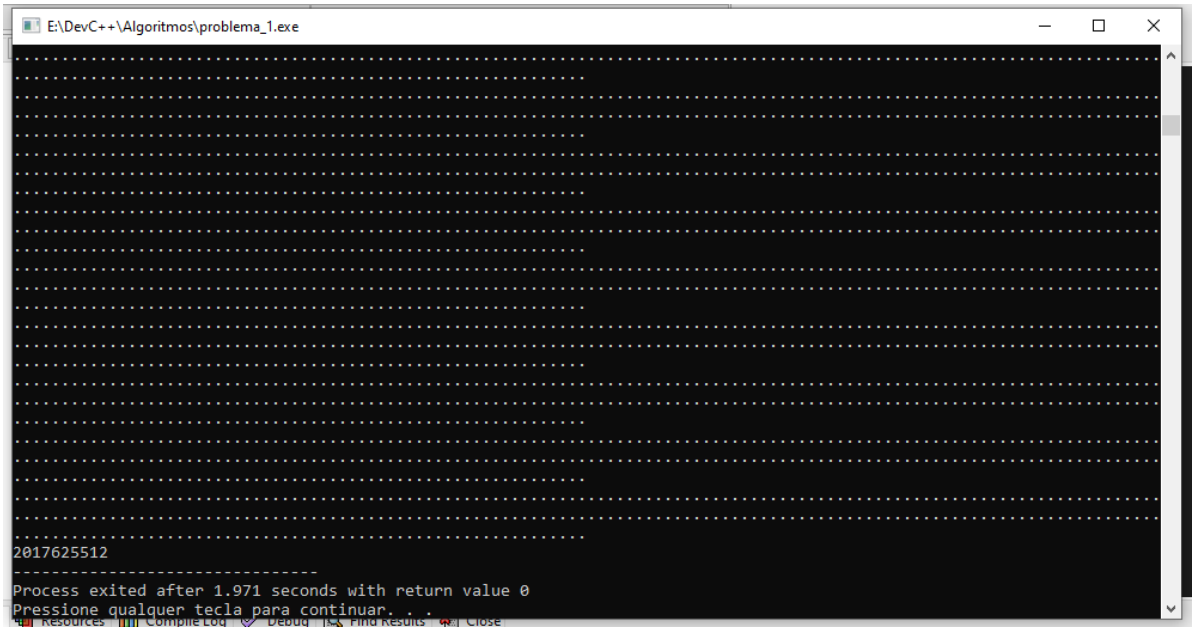
2)



3)



4)



RESULTADO FINAL URI JUDGE:

#		PROBLEMA	RESPOSTA	LINGUAGEM	HORA	DATA
27536662	3247	Robôs em uma Grade	Wrong answer (10%)	C++17	0.096	24/04/2022 18:22:15