UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

INTELIGENCIA ARTIFICIAL

CONTENIDO:

TAREA-1. EL PROBLEMA DEL LABERINTO BÁSICO.

PORCENTAJE TERMINADO: 100%

INTEGRANTES	DT	HG	HI	EVAL
Garcia Taboada Brayan Albaro	3	1	3	100

Fecha de presentación : Jueves, 9 de Mayo de 2024

Fecha Presentada : : Jueves, 9 de Mayo de 2024

Días de Atraso : 0

EL PROBLEMA DEL LABERINTO.

TRABAJO INDIVIDUAL.

1. Dado una matriz de n x m, inicialmente todas las posiciones con valores de cero, avanzar las casillas en sentido horario con movimientos de izquierda, arriba, derecha y abajo. Hacer Algoritmos para los siguientes:

a) Algoritmo para mostrar todos los caminos posibles desde una posición inicial a una posición final. Además, mostrar la cantidad de soluciones posibles (Cantidad de caminos posibles de la posición inicial a la posición final).

```
public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {

    if (!posValida(m, i1, j1)) {

       return;

    }

    m[i1][j1] = paso;

    if (i1 == i2 && j1 == j2) {

       mostrar(m);

       soluciones++;

    }

    laberinto(m, i1, j1 - 1, i2, j2, paso + 1);

    laberinto(m, i1 - 1, j1, i2, j2, paso + 1);

    laberinto(m, i1, j1 + 1, i2, j2, paso + 1);

    laberinto(m, i1 + 1, j1, i2, j2, paso + 1);

    m[i1][j1] = 0;

  }
```

b) Algoritmo para mostrar todos los caminos posibles desde una posición inicial a una posición final tal que se visiten todas las casillas de la matriz. Además, mostrar la cantidad de soluciones posibles.

```
  public static void laberintoLleno(int m[][], int i1, int j1, int i2, int j2, int paso) {

    if (!posValida(m, i1, j1)) {
```

```
        return;
    }
    m[i1][j1] = paso;
    if (i1 == i2 && j1 == j2 && todosMarcados(m)) {
        mostrar(m);
        soluciones++;
    }
    laberintoLleno(m, i1, j1 - 1, i2, j2, paso + 1);
    laberintoLleno(m, i1 - 1, j1, i2, j2, paso + 1);
    laberintoLleno(m, i1, j1 + 1, i2, j2, paso + 1);
    laberintoLleno(m, i1 + 1, j1, i2, j2, paso + 1);
    m[i1][j1] = 0;
  }
```

c) Algoritmo para mostrar todos los caminos posibles desde una posición inicial a una posición final tal que NO se visiten todas las casillas de la matriz. Además, mostrar la cantidad de soluciones posibles.

```
    if (!posValida(m, i1, j1)) {
        return;
    }
    m[i1][j1] = paso;
    if (i1 == i2 && j1 == j2 && !todosMarcados(m)) {
        mostrar(m);
        soluciones++;
    }
    laberintoNoLleno(m, i1, j1 - 1, i2, j2, paso + 1);
    laberintoNoLleno(m, i1 - 1, j1, i2, j2, paso + 1);
    laberintoNoLleno(m, i1, j1 + 1, i2, j2, paso + 1);
```

```
        laberintoNoLleno(m, i1 + 1, j1, i2, j2, paso + 1);

        m[i1][j1] = 0;

    }
```

d) Algoritmo para mostrar todos los caminos posibles de máxima longitud desde una posición inicial a una posición final.. Además, mostrar la cantidad de soluciones posibles.

```
    public static void laberintoMaxpasos(int m[][], int i1, int j1, int i2, int j2, int paso, int n)
{

        if (!posValida(m, i1, j1)) {

            return;

        }

        m[i1][j1] = paso;

        if (i1 == i2 && j1 == j2 && paso < n) {

            mostrar(m);

            soluciones++;

        }

        laberintoMaxpasos(m, i1, j1 - 1, i2, j2, paso + 1, n);

        laberintoMaxpasos(m, i1 - 1, j1, i2, j2, paso + 1, n);

        laberintoMaxpasos(m, i1, j1 + 1, i2, j2, paso + 1, n);

        laberintoMaxpasos(m, i1 + 1, j1, i2, j2, paso + 1, n);

        m[i1][j1] = 0;

    }
```

e) Algoritmo para mostrar todos los caminos posibles de mínima longitud desde una posición inicial a una posición final.. Además, mostrar la cantidad de soluciones posibles.

```
    public static void laberintoMinpasos(int m[][], int i1, int j1, int i2, int j2, int paso, int n) {

        if (!posValida(m, i1, j1)) {

            return;
```

```
    }

    m[i1][j1] = paso;

    if (i1 == i2 && j1 == j2 && paso >= n) {

       mostrar(m);

       soluciones++;

    }

    laberintoMinpasos(m, i1, j1 - 1, i2, j2, paso + 1, n);

    laberintoMinpasos(m, i1 - 1, j1, i2, j2, paso + 1, n);

    laberintoMinpasos(m, i1, j1 + 1, i2, j2, paso + 1, n);

    laberintoMinpasos(m, i1 + 1, j1, i2, j2, paso + 1, n);

    m[i1][j1] = 0;

  }
```

2. Ejecutar para todos los incisos del Ejercicio 1, inicialmente con posiciones con valor de cero (paso libre), valor de -1 (atajo o pared). Analizar las salidas y escribir conclusiones.

Para valores de 0 funciona correctamente pero para valores de -1 no funciona porque están en una posición no valida así que como su primer posición es en una no valida o pared o vacio, pues el algoritmo termina en ese momento

3. Implementar los ejercicios 2 y 3. Redefiniendo el moviento en el Laberinto, también se puede mover una casilla por las diagonales.

Con respecto al ejercicio uno con un valor inicial de cero el algoritmo aumenta en gran medida su cantidad de soluciones al haber mas pasos disponibles

De igual manera que con un valor inicial de menos 1 no funciona, la manera en la que podría funcionar seria haciendo el movimiento primero y luego preguntando si es una posición valida, de esta manera recorrerá las posiciones hasta hallar una valida en la cual iniciar nuevamente.