UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN.

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS.

Inteligencia Artificial.

PIA: Mundo de la aspiradora.

Profesor: M.C. Juan Pablo Rosas Baldazo

Alumno	Matricula
Elí Israel Delgado Escárcega.	1821213
Francisco Javier Martínez Palomar	1937837
Kevin Alexis Nájera Abrego	1798194

San Nicolás de los Garza, Nuevo León.

Mundo de la aspiradora.

Descripción general del problema.

Tenemos un agente aspiradora posicionado en un mundo de nxn losetas, y cuyo objetivo es recoger toda la suciedad del mapa. Cada una de las losetas puede ser de distintos tipos:

- Loseta transitable (representada con un símbolo «_»).
- Loseta con muro (representada con un símbolo «#»).
- Loseta con basura (representada con el símbolo «B»).
- Loseta donde se encuentra o expande la aspiradora (representada con el símbolo «A»).

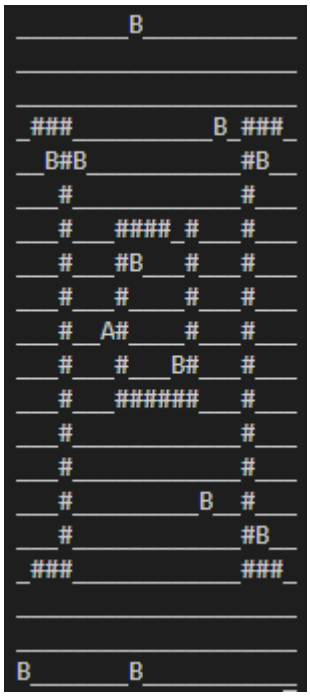


Ilustración 1:Ejemplo de mapa inicial.

Estado inicial.

El estado inicial de nuestra aspiradora puede ser cualquier loseta de nuestra malla de nxn losetas, mientras esta no sea un muro. Y con una cantidad determinada de muros y basura esparcida de manera aleatoria.

Acciones, reglas de transición y estado meta.

Nuestra aspiradora puede moverse en 4 direcciones, norte, sur, este y oeste, con la restricción de que no puede salirse de los límites del mapa, y, además, no está permitido que avance a una loseta en la que se encuentra un muro.

La aspiradora desconoce el tamaño del mapa, pero reconoce cuando llega al límite del mapa, debido a la naturaleza del algoritmo usado y la forma de utilizar los índices.

La aspiradora desconoce la cantidad de basura que hay en el mapa, su objetivo es recoger toda la basura en el menor numero de movimientos, para ello, se optó por dividir el problema en mini subproblemas, de la siguiente manera:

- Como la aspiradora desconoce la cantidad de basura que hay en el mapa, debe realizar una búsqueda hasta encontrar una basura.
- La búsqueda que realiza es un bfs. Una vez que encuentra esta basura, busca en su matriz de estados previos el camino mas corto para llegar a ella, considerando que estos cambios de estado no tienen un costo.
- Luego que realiza el camino mas corto hasta llegar a esa basura, el mapa se actualiza con la aspiradora en la loseta donde se encontraba la basura, y realiza otra búsqueda bfs como en el punto anterior.
- Estas búsquedas bfs se iteran tantas veces como basura haya en el mapa. Si la aspiradora realiza una búsqueda bfs en un mapa que ya no tiene basura, se da cuenta que ha recogido toda la basura y termina su tarea.

Tamaños de las instancias y parámetros.

La idea del programa es que este pueda resolver 20 mapas distintos dados por el programador. Estos 20 mapas son de dimensión nxn, empezando desde 20 filas/columnas y hasta 30 filas/columnas, dos mapas de cada dimensión.

Para que programa pueda resolver el problema se le debe dar un mundo o mapa con las características ya mencionadas. El nodo inicial es necesario, pero este se encuentra dentro del código al iterar hasta encontrar la loseta con el carácter ‘A’.

El código utilizado, bfs.

Como ya se mencionó, se optó por utilizar un algoritmo bfs, donde se resuelve «x» mini problemas de encontrar la ruta mas corta a la basura más cercana, así hasta no encontrar más basuras. Se deja aquí un enlace a un video ejemplificando el algoritmo con una instancia.

Pseudocódigo.

Procedimiento resolver(mundo):

Variables usadas

contador_de_pasos guarda el numero de pasos requeridos para hacer un bfs
contador_de_basura cuenta la basura del mapa
prev es una matriz nxn (nxn siendo las dimensiones del mapa) con las coordenas previas a cada loseta
t_r es la fila en la que está la basura encontrada
t_c es la columna en la que está la basura encontrada

Mientras siga habiendo basura:

contador_de_pasos, prev, t_r, t_c = resolver_bfs(mundo)

Esta función solo retorna el camino reconstruido de la aspiradora, no se considera necesario explicar más a fondo la función debido a que es más para realizar impresiones

camino = reconstruir_camino(prev)

Procedimiento resolver_bfs(mundo):

Esta función solo retorna la ubicación de la aspiradora, no se considera necesario explicar más a fondo la función

Aspiradora_fila, aspiradora_columna = aspiradora_index(mapa)

Variables usadas

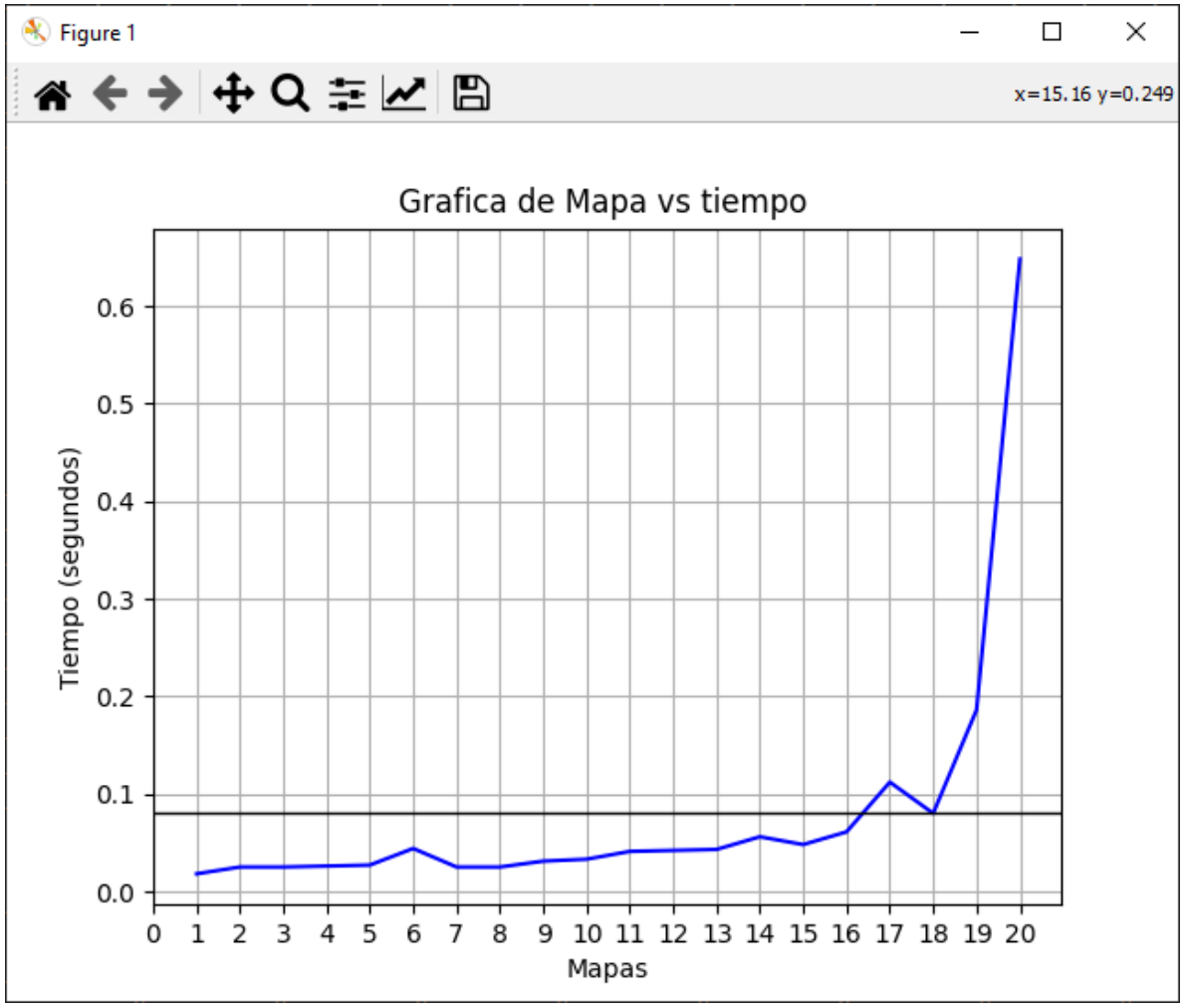
Filas es el número de filas de la matriz
Columnas es el número de columnas de la matriz
Filas_cola guarda los índices de las filas exploradas en el algoritmo
Columnas_cola guarda los índices de las columnas exploradas en el algoritmo
Visitado es una matriz nxn de booleanos que indica si un nodo ha sido visitado
//Estas dos variables son arreglos auxiliares para obtener las direcciones a las que puede ir la aspiradora
Direccion_fila = [-1, 1, 0, 0]
Direccion_columa = [0, 0, 1, -1]
Final_encontrado es un booleano que indica si se encontró una basura
Contador_de_pasos lleva un registro de los pasos necesarios para llegar a la basura más cercana
Nodos_restantes_en_la_capa es la cantidad de elementos que hay que sacar de la cola antes de hacer un paso
Nodos_en_siguiente_capa es la cantidad de nodos que agregamos en la expansion del BFS

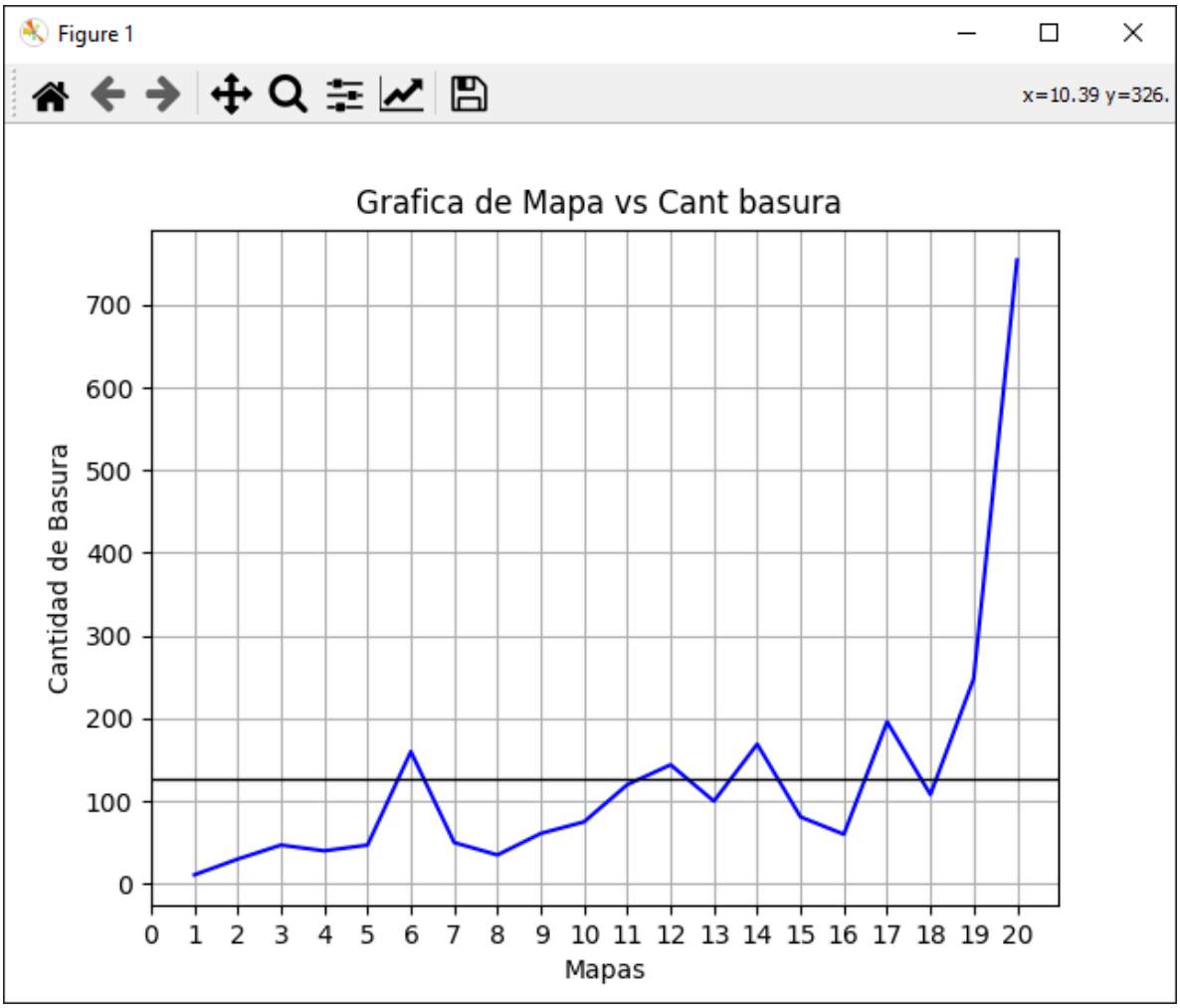
prev es una matriz nxn con las coordenas previas a cada loseta

```
Filas_cola.agregar(aspiradora_fila)
Columnas_cola.agregar(aspiradora_columna)
Visitado[Aspiradora_fila][Aspiradora_columna] = Verdadero

Mientras haya elementos en la cola:
    Fila_actual = fila_cola.sacar()
    Columna_actual = columna_cola.sacar()
    Si mundo[Fila_actual][ Columna_actual] = 'B' //B es el nodo basura
        Final_encontrado = verdadero
        Break;
    //En este for obtengo los vecinos
    Para i = 0 hasta i < 4:
        Fila_siguiente = Fila_actual + Direccion_fila[i]
        Columna_siguiente = Columna _actual + Direccion_ columna [i]
        //Verifico si la dirección siguiente está fuera del mapa
        Si Fila_siguiente < 0 or Columna_siguiente < 0:
            Continue
        Si Fila_siguiente >= Filas or Columna_siguiente >= Columnas:
            Continue

        Si Visitado[fila_siguiente][columna_siguiente] == Verdadero:
            Continue
        Si mundo[fila_siguiente][columna_siguiente] == '#' //'#' es un muro
            Continue

        Visitado[fila_siguiente][columna_siguiente] = Verdadero

        prev[fila_siguiente][columna_siguiente]                    =
        [Fila_actual,Columna_actual]

        Nodos_en_siguiente_capa += 1

    Nodos_restantes_en_la_capa -=1
    Si Nodos_restantes_en_la_capa == 0:
        Nodos_restantes_en_la_capa = Nodos_en_siguiente_capa
        Nodos_en_siguiente_capa = 0
        Contador_de_pasos += 1
Si Final_encontrado:
    Return Contador_de_pasos, Prev, Fila_actual, Columna_actual
Restun -1, -1, -1, -1 //El -1 indica cuando ya no hay más basura
```

Graficas.

Como se mencionó, el programa se corrió con 20 instancias de distintos tamaños y recogió la totalidad de la basura en cada ejecución. A continuación, se muestra la gráfica mapa vs tiempo y mapa vs cantidad de basura a manera de comparación. Nótese que para estas gráficas no se imprimieron los pasos, solo se realizó el algoritmo. Esto último con el objetivo de ver el tiempo real de ejecución.





Puede notarse que, para parejas con dimensiones iguales, pero cantidad de basura distintas, los valores se mantenían más o menos constates.

Por ejemplo 1 y 2 son mapas de 20x20, y el tiempo que tardaron fue cercano a pesar de tener diferentes disposiciones, debido a que su cantidad de basura era muy cercana.

Por otro lado, la pareja 19 y 20 del mapa de 30x30 difieren mucho, esto debido a que la cantidad de basura de uno a otro se llega a triplicar.

Referencias consultadas.

Videos de ayuda:

Fiset, William. [WilliamFiset]. (2018, abril, 20). Breadth First Search Algorithm | Shortest Path | Graph Theory. [Archivo de video]. Breadth First Search Algorithm | Shortest Path | Graph Theory - YouTube

Fiset, William. [WilliamFiset]. (2018, abril, 2). readth First Search grid shortest path | Graph Theory. [Archivo de video]. Breadth First Search grid shortest path | Graph Theory - YouTube