3.- Descripción y análisis del algoritmo

A continuación se describirán el problema, el análisis del algoritmo, sus propiedades y los resultados al aplicarlo a las matrices de entrada de los casos de uso. Particularmente hablando en la segmentación de imágenes de texto antiguo. También se implementarán ejemplos sobre la segmentación de los espacios de configuración para planeación de movimiento.

3.1 Descripción del problema abstracto

Tenemos una matriz de tamaño de mxn, donde se representa un laberinto por ser resuelto. El laberinto tiene para representación de paso libre valores de ceros y para el caso de pared, unos. En el caso base, la posición *Xij* solamente tiene movimientos en cuatro direcciones; norte (Xi-1j), sur (Xi+1j), este (Xij+1) y oeste (Xij-1). Donde i representa nuestros renglones que van desde 1 hasta m y j la columna dentro de la matriz que va desde 1 hasta n. El laberinto puede tener varias salidas o sólo una y se encuentra del lado contrario a donde se comenzará la búsqueda. En caso de tener varias soluciones, se puede buscar al menos una, o todas.

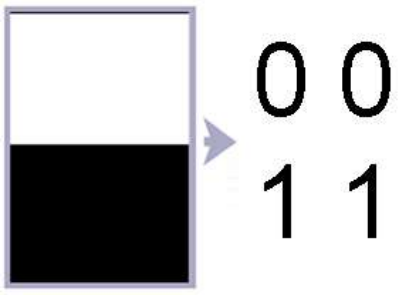

Figura 6 Representación.

La solución típica a este problema es, mediante una heurística llamada *Siempre Izquierda.* *Siempre izquierda* se comporta de la siguiente manera; tenemos un punto inicial Xij, de aquí comenzamos a avanzar dentro de la matriz hacia la siguiente posición libre buscando inicialmente por la izquierda, luego por el centro y finalizando por la derecha (este orden varía de acuerdo con la implementación del programador). La salida es una posición del lado contrario de la matriz. En caso de quedar encerrado, regresa a la última decisión tomada y recorre por la siguiente opción. Así sucesivamente hasta llegar a la solución que se busca.

3.2 Razonamiento

El razonamiento que ayudó a determinar el algoritmo fue el siguiente. Una persona al ver un laberinto para resolverse, tiene cierta noción de la dirección a tomar, para ilustrarlo mejor tomemos el siguiente ejemplo.

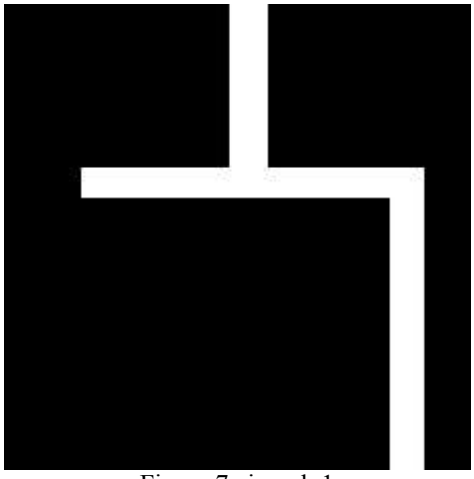

Figura 7 ejemplo1

En la figura tomándola como ejemplo simple, si se comienza por la parte superior, una persona no intentaría meterse en el callejón formado por la primera bifurcación de camino.

La persona sabe que ese camino nos llevaría al encierro, ó que sencillamente, no es parte de la solución. En cambio la heurística de *Primero Izquierda* si entra buscando una solución. Llegando al final del callejón y verificar que no es la salida se regresa a su última decisión y explora la siguiente ruta distinta a la tomada. De modo que el algoritmo busca primero los encierros de cada callejón dentro de la matriz. Estos encierros se deben a que eventualmente cada una de las posiciones al final de un callejón no son solución. En caso de que no tuviera solución, *primero izquierda* habría recorrido la matriz completa sin reportar error hasta terminar. Por el contrario, si hubiera varias soluciones, también tendría que recorrer todas las decisiones para ir formando los caminos correctos.

3.3 Descripción del algoritmo

El algoritmo funciona buscando primero las posiciones donde no se pueda avanzar más. Busca cada posición de Xij que no tienen más opción para continuar. Solamente avanza sobre cada cero de la matriz, verificando si alrededor tiene más de dos posiciones adyacentes con valores uno, es decir, se encuentra encerrado. Si la suma de los cuatro adyacentes; norte (Xi-1j), sur (Xi+1j), este (Xij+1) y oeste (Xij-1), es mayor a dos, entonces convierte esa posición en 1 también. Si esto se cumple entonces el algoritmo retrocede en i-1 por que es la última posición de la matriz que se ve afectada. La verificación sólo se hace hasta una vez por posición cerrada. Es decir, como sólo entra en las posiciones con valores igual a cero al ser cambiadas a uno, no vuelven a evaluarse. Al final, el resultado son todas las posiciones Xij que se conectan entre ellas formando un camino que llevan a las orillas de la matriz. Esta solución funciona en el caso de que nos encontremos resolviendo un problema donde solo nos podemos mover en esas cuatro direcciones.

En el caso de que existan las ocho posiciones alrededor de cada posición como opción a ser una movida, se hace un ligero cambio. Antes, sólo existían norte, sur, este y oeste. En este nuevo planteamiento también tenemos noreste, noroeste, sureste y suroeste. Se verifica cada posición Xij y se suman sus ocho adyacentes. Si la suma es mayor a 3 se verifica que la suma de los unos adyacentes sean consecutivos. Es decir, si la suma es igual a 4 verificamos que las cuatro posiciones con 1 estén conectadas entre sí. Si las posiciones se encuentran separadas, significa que existe un camino que pasa entre ellos y es posible seguir moviéndose. No necesariamente la suma debería ser mayor a 3, también puede variar el algoritmo y tener una *sensibilidad* mayor a 2. Los resultados varían dependiendo de esta sensibilidad. La secuencia de posiciones con valores 1 está rota, como se muestra en la siguiente figura.

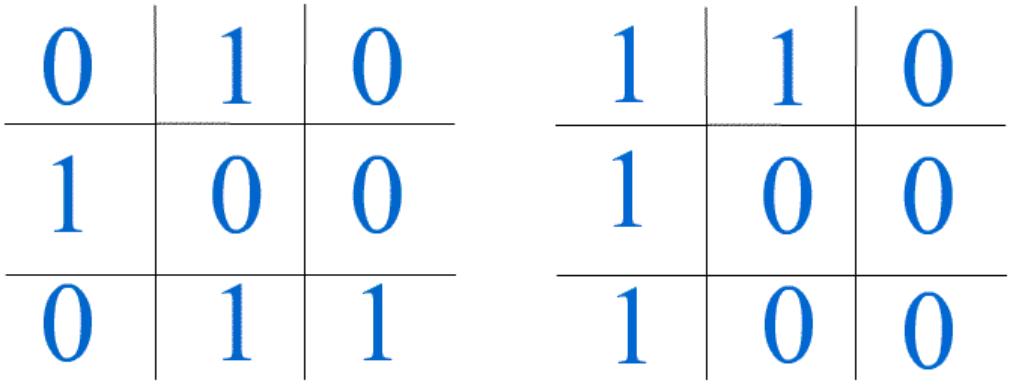

Figura 8 secuencia de 1s rota (izquierda), secuencia continua (derecha)

En la figura 8(izquierda), se muestra como existe un paso vacío entre los unos que representan pared. La suma de los adyacentes de la posición central con valor 0 es 4. A pesar de que la suma es mayor a 3, al verificar que la secuencia alrededor de Xij sea continua se marca falla, y permanece como camino. Es decir, comparamos la suma de los adyacentes con la suma de los números uno continuos y al ser menor a la suma total no se cambia la posición Xij de cero a uno. Esto se cumple para los valores mayores a 3 hasta 6,

si la suma es mayor a 6 entonces automáticamente la posición Xij se convierte en 1, es decir:

Si $\sum_{i-1}^{i+1} \sum_{j-1}^{j+1}$ Xij >= 3 y el valor de Xij = 0, entonces verificamos que sea una secuencia continua. Y si la secuencia es continua entonces el valor de Xij = 1. Entonces, el algoritmo en seudo código queda de la siguiente manera;

Si tenemos una matriz de tamaño mxn.

```
For ( i = 1 => m){                    //  Entra en el ciclo para recorrer la matriz
        For (j = 1 => n){
                If(Xij = 0){          //  al encontrar una posición Xij con valor cero
                  If (verifica(Xij)){ //manda a llamar al método verifica
                        Xij = 1               //Si el resultado es verdadero
                        i--           //el valor en la posición cambia a 1
                        j--           // y se decrementa en un renglón y una columna
                        }             // por que esa es la última posición afectada y que
                }                     //podría también cambiar su valor.
        }
}
```

Donde el método *verifica* se encargará de sumar los valores adyacentes y después verificar si el total de números mayor o igual a 3. En caso de que sea mayor o igual el método verifica que las posiciones que tuvieron valor 1 se encuentren conectados uno después del otro. Esto lo lleva a cabo mediante una serie de If´s anidados, el método se encuentra en el anexo al final del documento.

3.4 Análisis de rendimiento y espacio

Suponiendo que analizamos una matriz cuadrada de tamaño nxn, tenemos que el algoritmo pasa por lo menos n^2. Podemos asegurar que al verificar cada una de las posiciones con 0 y que además fueron cambiadas por el análisis a 1 no se vuelven a verificar. Entonces, el tiempo de corrida se encuentra acotado a On^3, lo cual es polinomial.

Para corroborar lo anterior, se corrieron matrices cuadradas y se cuantificaron las veces que se recorrieron las posiciones de esta. Siendo estas matrices con la forma de peor caso para el algoritmo. El peor caso, se encuentra en las matrices que contienen en su última fila y su última columna valores 1, el resto de las posiciones de la matriz que tienen valor 0. Para analizar el tiempo de corrida del algoritmo se usa el peor caso. Utilizando el método del *truco de contabilidad* donde se inicia un contador *C* con valor inicial cero y cada vez que el bucle superior hace el llamado a la función *p*, en este caso específico *verifica,* se incrementa el contador *C* en uno.[15] Esto con el fin de determinar una cota superior en las corridas del algoritmo.

0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1
1	1	1	1	1	1	1	1

Figura 9 peor caso

Al llegar a la última posición y verificar, tienen que cambiar el valor de la posición. Por esto, al regresar a la esquina izquierda superior y realizar nuevamente la verificación tiene que cambiar también ese valor, y así sucesisavemente.

Figura 10 comportamiento peor caso.

En la figura del comportamiento del peor caso, se muestra como la esquina se va moviendo hasta terminar con la columna. Este mismo proceso funciona para la siguiente columna hasta terminar con la matriz completa. El siguiente cuadro, muestra las veces que entró la posición para verificación.

veces que entró en la posición	tamaño n	n^3
119812	50	125000
208572	60	216000
332932	70	343000
498892	80	512000

Figura 11 Cuadro comparativo

Como podemos ver, las veces que se verifica cada posición en los peores casos, tienen como cota superior la n^3 la cuál no alcanza. De igual manera el algoritmo solamente procesa los resultados en la misma matriz. El espacio en memoria que utiliza depende directamente del tamaño de la matriz. En el mejor caso, una matriz llena de unos, no necesita entrar a verificar en ningún momento.

3.4.1 Heurísticas para evitar el peor caso

El caso peor para el algoritmo podría evitarse. Para esto comenzamos obteniendo las proyecciones verticales y horizontales de la matriz a analizar. Es decir, sumamos los valores de los renglones y columnas. Buscamos después la intersección de los valores mayores encontrados y comenzamos por esa posición a analizar la matriz. Incluso se puede usar de manera recursiva, es decir utilizar primero el valor más grande de intersecciones, después el siguiente valor mayor encontrado y continuar así hasta terminar.

3.4.2 Evaluación empírica del algoritmo

Se realizó un programa generador de matrices aleatorias. En este programa se desligan matrices de tamaño cuadrado 600. En el cuál se implementó el algoritmo y se ejecutó 100 veces. Para cada ejecución se desplegó el resultado de manera gráfica para visualizar el comportamiento y verificar el rendimiento. El algoritmo mostró constancia en los siguientes resultados. En ninguna de las corridas el algoritmo quedó ciclado en alguno de los bucles. En todas las corridas el algoritmo terminó.

3.5 Propiedades del algoritmo

El algoritmo anteriormente presentado, tiene variantes de comportamiento. Por lo tanto se explicará en la siguiente sección cada uno de los pasos que sigue. Así como su comportamiento, su representación y sus resultados de la variante que tiene. Esta variante de comportamiento es debido a la propiedad de sensibilidad que posee. Dependiendo de la sensibilidad, habrá más caminos que cambien ó se obstruyan. Excepto que, si existe un camino de ceros que pueda atravesar la matriz de una orilla a otra o las demás orillas de la matriz, seguirá existiendo ese camino. En la siguiente sección de resultados se demuestran estas propiedades, dependiendo de la sensibilidad utilizada y su comportamiento.

3.5.1 Sensibilidad del algoritmo

Como antes se mencionó el algoritmo para verificar una posición recurre a la suma de las posiciones adyacentes a sí misma. Si la suma de sus vecinos es superior a dos o tres,

dependiendo del uso del algoritmo, verifica para cambiar el valor en la posición o no hacerlo. Que sea superior a tres la suma de los adyacentes, alrededor de una posición con valor cero se analizará la secuencia. Es decir, verifica si las posiciones están conectadas una después de otra y la suma es igual a la suma de los adyacentes contados. A esto se le llamó sensibilidad y dependiendo de ella las propiedades del algoritmo cambian.

3.5.1.1 Mayor sensibilidad

Si se fija la sensibilidad a verificar las posiciones a partir de tres en adelante, mayor a dos, el comportamiento es el siguiente. El algoritmo busca todas las posiciones que tengan alrededor de cada cero en la matriz tres posiciones con valor igual a uno. Si la suma de los valores adyacentes se encuentran conectadas una después de otra, el valor de la posición Xij cambia de cero a uno. Esto es debido a que tiene la sensibilidad más alta que puede tener el algoritmo al verificar una posición. El resultado es un camino de ceros que rodean a las figuras dentro de la matriz, y continúan hasta tocar las orillas de la matriz. Conectando de esa manera, con una secuencia de ceros, desde los extremos de la matriz y pasando entre los obstáculos. En la sección de resultados se ejemplifica este caso.

3.5.1.2 Menor sensibilidad

Al fijar la sensibilidad del algoritmo en verificar secuencias a partir de una suma mayor a tres, el comportamiento es el siguiente. El algoritmo busca las esquinas en la matriz que rodean la posición analizada. Teniendo esta sensibilidad lo que ocurre con las figuras, en el espacio de trabajo, es que son engrosadas hasta convertirse en figuras cóncavas. También si existen caminos entre las figuras y embonan, deja un camino fino entre ellas después de haber engrosado las figuras. En la sección de resultados se ejemplifica este caso.