**Documentación de solución**

**Sergio Franco Pineda (202116614), Lina María Ojeda (202112324)**

**1. Algoritmo de solución**

**1.1 Explicación del algoritmo elegido**

El algoritmo se implementó bajo las siguientes anotaciones:

*-* Un componente del grafo está compuesto por vértices y arcos entre cofres que se abren con la misma llave.

*-* El área que cubre cada componente en la matriz se calcula como para nodos que forman parte del componente.

*-* Hay dos condiciones que deciden cómo se abrirán los cofres en cierta iteración, esto se explicará en el punto *e*.

En ese sentido, el algoritmo es el siguiente:

*a.* Formar el grafo. En este grafo cada vértice se conecta con sus adyacentes de la matriz, es decir, el elemento en la matriz se conecta con los elementos de la matriz , , y siempre y cuando existan en la matriz, donde y . El identificador de un cofre es su posición en la matriz. Esto se hace por medio de y el resultado de este paso es el siguiente:

|  |  |
| --- | --- |
|  |  |
|  |  |

*b.* Formar los componentes del grafo, este paso se realiza en el mismo del paso *a* y el resultado es el siguiente:

*c.* Calcular las áreas de cada componente en la matriz. Esto se hace por medio de un recorrido y el resultado en este caso sería la priority queue .

Los eleméntenos son de la forma donde es mayor a y corresponde al valor calculado para cada componente en el paso *b* y es el identificador de cada componente que es para el componente que contiene los cofres que se abren con la llave .

Junto la anterior estructura, este paso también da como resultado el siguiente diccionario para poder encontrar el área de un componente en tiempo constate .

*d*. Escoger el componente que más área ocupa, esto es hacer heappop a la queue resultado de *c*.

*e.* Desde cualquier cofre del componente seleccionado en *d*, se realiza DFS para identificar todos los nodos que son alcanzables desde ese nodo y que además están dentro del área cubierta por componente seleccionado. Al mismo tiempo se van abriendo los cofres que se encuentren y hagan parte del componente seleccionado, pero si se encuentra un cofre que ya está abierto en esa área se sabe que no se puede descifrar el mensaje y el programa termina. Abrir un cofre significa cambiar el valor de a del elemento en una matriz de .

La explicación anterior aplica para cuando el componente que se está abriendo en cierta iteración comparte área con otros componentes (en caso de que lo haga) con un componente de área menor o igual a la del componente , para cuando esto no pasa (el área de es mayor que la de ), se procede de la siguiente manera.

Algorítmicamente se vuelven a cerrar los cofres del componente y se continúan abriendo los del componente , pero la información de área del componente se vuelve a pushear a la priority queue por lo que en la siguiente iteración será el primer componente en abrirse. Un caso donde esto pasa es el siguiente:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
| 1 | 1 | 3 | 3 | 3 | 3 | 3 | 2 |
| 1 | 1 | 3 | 3 | 3 | 3 | 3 | 2 |
| 4 | 4 | 3 | 3 | 3 | 3 | 3 | 5 |
| 4 | 4 | 3 | 3 | 3 | 3 | 3 | 5 |

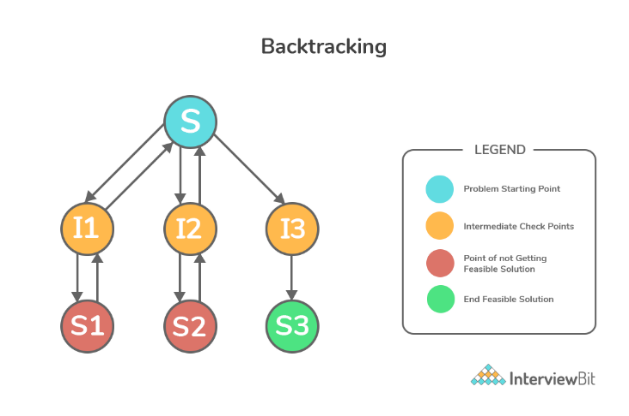
En este caso los componentes 1 y 2 comparten área con el componente 3 que tiene mayor área que ellos, abrir primero el 3 es un error por lo que aquí es donde se usa el caso explicado en el algoritmo anterior. Un posible orden en el que se abren estos cofres usando el algoritmo anterior es 1,2,3,4,5.

*f.* Si no se encontró ningún cofre ya abierto en el área cubierta de otro componente con menor o igual área, añadir el paso a una lista de pasos.

*g.* Repetir los pasos *d*, *e* y *f* hasta que la lista de áreas quede vacía.

**1.2 Alternativas de implementación**

En principio, el primer subproblema de este problema es saber el orden en el que se usaran las llaves por lo que lo empezamos a pensar como un problema de decisión donde el dilema es elegir si usar o no cierta llave en cierta iteración. La alternativa sería entonces una solución usando programación dinámica junto con una estrategia de backtracking, donde la idea sería experimentar las posibles permutaciones de orden en cuanto al uso de las llaves, si se encuentra que un camino del árbol de decisión en un nodo dado no es conveniente para la solución nos tendríamos que devolver al nodo anterior para evaluar otra posible decisión desde ahí; cada nodo sería un tipo de llave.



**1.3 ¿Por qué se escogió la alternativa que se implementó?**

Al observar los resultados obtenidos por la alternativa de programación dinámica fue cuando nos dimos cuenta de que el orden en el que se abrían los componentes depende de los dos casos explicados anteriormente (cuando un componente A comparte área con uno B de menor o igual área y cuando se comparte con uno B de mayor área), por eso decidimos implementar este orden de mayor a menor y condiciones de intersección de áreas en la alternativa implementada ya que reduce a las permutaciones que evaluaba la alternativa en el peor de los casos y por ende también en favor de reducir la complejidad temporal de la solución.

**1.4 ¿Por qué el algoritmo implementado resuelve perfectamente el problema?**

Observar los resultados de la alternativa explicada anteriormente y ver que el orden en el que se abrían los componentes dependía siempre del orden de mayor a menor área y de los casos de intersección de áreas explicados anteriormente, nos permite estar seguros de que implementar este orden y condiciones para cualquier caso permite llegar a la solución correcta.

**2. Análisis de complejidad**

en el peor de lo casos

**2.1 Complejidad temporal**

La función tiene una complejidad de , la función tiene una complejidad de y la función tiene una complejidad de , por lo tanto, la complejidad del algoritmo es .

**2.2 Complejidad espacial**

La función tiene una complejidad de , la función tiene una complejidad de y la función tiene una complejidad de , por lo tanto, la complejidad del algoritmo es

**3. Escenarios de comprensión de problemas algorítmicos**

**3.1 Se puede aplicar la misma llave hasta dos veces**

**3.1.1 ¿Qué nuevos retos presupone este escenario?**

En este escenario, se debe identificar si al intentar abrir el componente completo como se hace en el problema original el mensaje de otro tipo de cofre se daña, en este caso, deberíamos dividir el área cubierta por el componente en dos para poder hacer el segundo uso de la llave y evaluar si con estas dos áreas no se daña ningún cofre, si se daña, es porque no se puede descifrar el mensaje. El reto aquí es saber cuál es la mejor manera de dividir el área completa del componente.

**3.1.2 ¿Qué cambios le tendría que realizar a su solución para que se adapte a este nuevo escenario?**

En la función se deberían retornar aparte del área completa del componente también las subáreas del área completa de un componente, es decir, para el componente en el siguiente ejemplo, se debería retornar la siguiente variable :

|  |  |  |
| --- | --- | --- |
|  |  |  |

Cada elemento de la lista es una tupla donde el primer elemento es el área máxima encontrada para el componente; el segundo elemento es una lista de tuplas que representan las áreas encontradas donde en cada tupla el valor absoluto del primer elemento es el valor del área, el segundo la fila inicial del área, el tercero es la fila final del área, el cuarto es la columna inicial del área y el quinto es la columna final del área; y el último elemento es el componente al que pertenece toda la información anterior.

En la función se debería implementar lo dicho en 3.1.1: evaluar si es posible usar el área de la primea tupla de lista de áreas del componente, si no, evaluar si es posible con alguna otra combinación de dos entre las áreas de la lista de áreas.

**3.2 Se pueden diseñar estructuras en forma de ele (L).**

**3.2.1 ¿Qué nuevos retos presupone este escenario?**

En este escenario, se debe identificar si al intentar abrir el componente completo como se hace en el problema original el mensaje de otro tipo de cofre se daña, en este caso, deberíamos formar las áreas en formas de ele que se pueden formar dentro de esta área y evaluar si con alguna de estas nuevas áreas no se daña ningún cofre. El reto aquí es, en caso de que exista, saber cuál es la ele que puede abrir los cofres del componente sin dañar a ningún otro cofre.

**3.2.2 ¿Qué cambios le tendría que realizar a su solución para que se adapte a este nuevo escenario?**

En la función se deberían retornar, en una manera similar al escenario anterior, aparte del área completa del componente también las eles que se pueden formar dentro de esta área.

En la función se debería evaluar si es posible usar el área completa del componente, si no, abrir los cofres con la ele que puede abrir los cofres del componente sin dañar el mensaje de otro cofre en otro componente.

Por ejemplo, para la siguiente matriz se podría usar la llave de la fila a la 3 y de la columna a la , luego, la llave no se podrá usar con un área rectangular por lo que se debe usar el área en forma de ele que pueda abrir los cofres correctamente, esta es la ele que cubre las celdas , y .

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |