

## RESEÑA DEL PROBLEMA

<b>Problema</b>	:	Encajar
<b>Complejidad</b>	:	Media
<b>Tópico</b>	:	Estructuras de datos
<b>Conocimientos</b>	:	Estructuras de datos alternativas
<b>Tamaño del Código</b>	:	Medio
<b>Codificación</b>	:	Media
<b>Estructuras de datos</b>	:	Media
<b>Propuesto por</b>	:	Mario Cruz, Colombia

## ANTECEDENTES

El problema de *Encajar* busca que el estudiante identifique la necesidad de implementar estructuras de datos alternativas que permita una consulta eficiente a información recurrente cuando la velocidad de proceso es crítica.

## APROXIMACIONES

Esta es una tarea donde la obtención de “algunos” puntos es extremadamente fácil, pues se limita a un algoritmo correcto que permita rotar la herramienta a sus diferentes posiciones y la inclusión de cinco (5) *fors* anidados de la siguiente forma:

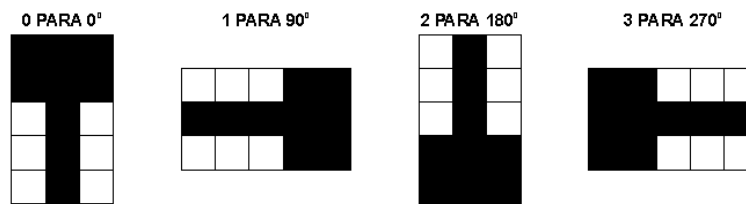
```
for (posibles filas en tablero(ft))
  for (posibles columnas en tablero(ct))
    for (posibles rotaciones de herramienta(4))
      Valida=Verdadero
      for (posibles filas en herramienta(fh-ch))
        for (posibles columnas en herramienta(ch-fh))
          si (no es posible colocar dada posición de la herramienta
              en dada posición del tablero)
            entonces Valida=falso
      si (Valida)
        entonces "Imprimir solucion y terminar programa"
```

Si el algoritmo termina sin encontrar una posición válida se deduce que “No hay solucion “. La facilidad de implementación del algoritmo se ve contrarrestada con su complejidad que es (para el peor de los casos) de:  $O(ft*ct*4*fh*ch)$  lo que equivale a:  $O(200*200*4*100*100) = 1.600.000.000$  (ó un algo menos, optimizando los límites de los ciclos), que es demasiado para el límite de tiempo de 2 s.

## SOLUCION

La solución propuesta se basa en el hecho de que para cada posición posible de la herramienta en el tablero, se revisan reiterativamente cada una de las cuadrículas involucradas, cuando podemos describir este proceso por un rango de celdas libres en tablero (vertical y horizontalmente) que al menos sea igual a las necesarias para una fila o columna dada en la herramienta. Esta estructura se precalcula antes de hacer el proceso de verificación asignando a cada posición *tablero[i][j].f* o *tablero[i][j].c* el número de celdas libres hacia abajo o hacia la derecha respectivamente de la posición *[i][j]*. Después de llenar esta estructura almacenamos en la estructura *tamano[i]* para cada rotación de la herramienta, la información correspondiente a la coordenada con respecto a la herramienta del rango adyacente horizontal y vertical de mayor tamaño y su tamaño. La razón de almacenar estas zonas de mayor tamaño es para verificar en primera instancia las condiciones que más posiciones nos puede hacer descartar, aunque es posible mejorar esta simple verificación (ver mejoras al algoritmo).

Por ejemplo para la herramienta del ejemplo almacenaríamos la siguiente información en *tamano[i]* (teniendo en cuenta que *[i]* corresponde al código de rotación de la herramienta):



	Tamaño zona horizontal adyacente de mayor área	Coordenada zona horizontal adyacente de mayor área	Tamaño zona vertical adyacente de mayor área	Coordenada zona vertical adyacente de mayor área
<b>tamano[0]</b>	3	(0,0)	5	(0,1)
<b>tamano[1]</b>	5	(1,0)	3	(0,3)
<b>tamano[2]</b>	3	(3,0)	5	(0,1)
<b>tamano[3]</b>	5	(1,0)	3	(0,0)

Con lo cual el bloque principal del programa luciría de la siguiente forma:

```

for (posibles filas en tablero(ft))
  for (posibles columnas en tablero(ct))
    for (posibles rotaciones j de herramienta(4))
      si (están libres las áreas correspondientes a tamano[j])
        entonces
          Valida=Verdadero
          for (posibles filas en herramienta(fh-ch))
            for (posibles columnas en herramienta(ch-fh))
              si (no es posible colocar dada posición de la
                herramienta en dada posición del tablero)
                entonces Valida=falso
          si (Valida)
            entonces "Imprimir solución y terminar programa"

```

La complejidad de este algoritmo es difícil de medir pues depende exclusivamente de la entrada, así que puede ir desde un máximo de  $O(f_t * c_t)$  en el mejor de los casos hasta un máximo para el peor de los casos de  $O(f_t * c_t * 4 * f_h * c_h / 4)$ , pero en cualquier caso tendrá un desempeño muy superior para juegos convencionales de datos, a medida que el tablero esta más lleno.

### CODIGO

La solución propuesta esta compuesta de cinco (5) funciones, que en su orden son:

#### leer()

Se inicializan las matrices que almacenaran el tablero (*tablero[i][j].marca*) y la herramienta (*herramienta[0][i][j]*) con valores binarios de 1 o 0 respectivamente para una cuadrícula ocupada o no ocupada. El primer índice en la matriz *herramienta* indica el código de rotación.

**rotar(unsigned char origen, unsigned char destino)**

Esta función realiza una rotación hacia la derecha de *herramienta[origen]* y la almacena en *herramienta[destino]*, aplicando la transpuesta a la matriz de entrada (logrado con una inversión de coordenadas). Es suficiente una única función para rotar hacia la derecha, pues, un rotación hacia la izquierda es equivalente a 3 hacia la derecha.

**llenar( )**

Esta función se encarga de precalcular los valores de *tablero[i][j].f* y *tablero[i][j].c* que corresponden al número de celdas vacías adyacentes verticalmente hacia abajo y horizontalmente hacia la derecha a partir de la cuadrícula *tablero[i][j]*. Para evitar tener que acumular la cantidad de espacios libres en las celdas anteriores cuando se encuentra una nueva celda libre, se hace un recorrido de la matriz en orden inverso.

**llenarh( )**

Esta función llena la información correspondiente a *tamano* donde se almacena para cada rotación de la herramienta, la información correspondiente a las zonas horizontales y verticales adyacentes de mayor tamaño de acuerdo a lo explicado anteriormente.

**proceso( )**

En primera instancia se almacena las posibles rotaciones de la herramienta usando para ello la función *rotar()*, luego se llama la función *llenarh()* y por último se realiza el ciclo principal de acuerdo a lo expuesto en la solución propuesta. Para la verificación de una zona ocupada se usa simplemente una operación “and” lógica entre las dos posiciones en prueba.

**EVALUACIÓN**

A continuación se discriminan para cada entrada de prueba (N), los criterios de evaluación previstos tales como: Tamaño del tablero y herramienta, criterios, salida, tiempo de ejecución para solución ineficiente y tiempo de ejecución para solución eficiente:

N	TAM.	CRITERIOS	SALIDA	T. LEN.	T. RAP.
1	20 20 1 5	-Muy Fácil (Cualquier algoritmo lo soluciona). - Prueba aspectos básicos del algoritmo como rotaciones, Verif. de posición, lectura y escritura en disco, etc. -Herramienta de forma básica y simetría vertical. -Herramienta Rectangular -Figura encaja en zona rectangular libre.	16 16 1	0.00	0.00
2	35 35 4 4	-Fácil (Cualquier algoritmo lo soluciona) -Herram. de forma semicompleja -Herramienta Cuadrada. -Figura encaja en zona rectangular parcialmente ocupada.	30 26 2	0.00	0.00
3	150 150 10 10	-Complejidad media (algoritmo medianamente programado lo resuelve). -Herramienta de forma semicompleja con simetria vertical. -Herramienta Cuadrada. -Tablero disperso. -Prueba No solución	No hay solucion	0.49	0.11

4	200 200 35 20	-Complejidad alta (algoritmo "inteligente" lo resuelve). -Herramienta de forma compleja. -Sin simetria. -Herramienta rectangular. -Tablero disperso.	176 125 3	2.80	0.33
5	200 200 50 50	-Complejidad muy alta (algoritmo "inteligente" lo resuelve). -Caso de prueba semi-extremo. -Herramienta básica. -Con simetria vertical y horizontal. -Herramienta cuadrada. -Tablero lineal.	151 145 0	22.91	0.11

### **MEJORAS AL ALGORITMO**

Para la solución propuesta se han tenido en cuenta únicamente las áreas horizontales y verticales de mayor tamaño, pero ¿porqué no considerar todas las áreas posibles en la herramienta?. Para esto cambiaríamos la descripción de una herramienta de su forma real a una descripción de coordenadas de zonas adyacentes horizontales o verticales de un tamaño dado, de igual forma a como lo hacíamos con sólo las zonas más grandes. Teniendo esta descripción no revisaríamos todas y cada una de las cuadrículas involucradas sino que lo haríamos únicamente contra estas zonas aprovechando la información calculada por *llenar()*. Por ejemplo para la herramienta del ejemplo tendríamos la siguiente descripción:

N	Tipo	Coordenada	Longitud	N	Tipo	Coordenada	Longitud
1	H	(0,0)	3	5	H	(4,1)	1
2	H	(1,0)	3	6	V	(0,0)	2
3	H	(2,1)	1	7	V	(0,1)	5
4	H	(3,1)	1	8	V	(0,2)	2

En general tendría un desempeño muy superior que los algoritmos antes expuestos pero existirían casos extremos donde la mejora sería únicamente de la mitad, por ejemplo en herramientas que contengan muchas áreas de sólo 1 celda adyacente (como en el caso de un tablero de ajedrez), en cuyo caso conviene combinarlo con la optimización hecha en el programa propuesto.

*Raúl Andrés Duque M.*  
*rduque@venus.uanarino.edu.co*