PROBLEMA DE LOS TRABAJADORES Y TAREAS

1. Descripción de la solución propuesta:

Para empezar, se recuerda el planteamiento del problema a resolver. Suponemos n trabajadores, n tareas y $c_{i,j}$ (> 0) como el coste de asignar la tarea j al trabajador i. Una tarea puede ser resuelta por un único trabajador y el trabajador puede realizar una única tarea. Tras esto, se suma cual es el coste total de las asignaciones y ese es resultado que se pide, saber qué coste tendría el proceso de asignación entero. Sea S un vector que contiene las soluciones ordenadas de tal forma que s[0] contenga el índice de la tarea que realiza el primer trabajador, s[1] la tarea que realiza el segundo y así hasta s[n-1] siendo la tarea que realiza el último.

Podemos plantear el problema con un procedimiento que asigne la mejor tarea de cada caso al trabajador correspondiente, así si el coste de su asignación es la más baja o es la única opción por ser el último emparejamiento, se selecciona como tarea definitiva para ese trabajador concreto. De esta forma, se obtiene la mejor tarea para cada trabajador en cada caso. Por ejemplo, para el siguiente caso en el que hay 5 tareas y 5 trabajadores se generan tales costes:

```
TABLA DE COSTES:
              Tarea 1    Tarea 2    Tarea 3    Tarea 4    Tarea 5
Trabajador 1:     1          8          9          4          8
Trabajador 2:     3          8          9          4          1
Trabajador 3:     2          4          3          1          6
Trabajador 4:     3          4          4          9          8
Trabajador 5:     9          4          8          2          2
```

Este procedimiento empezaría comprobando qué tarea cuesta menos para el trabajador 1 (en este caso la Tarea 1), luego para el trabajador 2 teniendo en cuenta que la tarea 1 no está disponible (le correspondería la Tarea 5) y así de forma consecutiva. El resultado de esta operación resultaría en algo así, con un coste total de 15:

```
Tareas que hace cada trabajador:
El trabajador 1 hace la tarea 1 que tiene un coste de: 1
El trabajador 2 hace la tarea 5 que tiene un coste de: 1
El trabajador 3 hace la tarea 4 que tiene un coste de: 1
El trabajador 4 hace la tarea 2 que tiene un coste de: 4
El trabajador 5 hace la tarea 3 que tiene un coste de: 8
El coste total asignando tareas es: 15
```

De esta forma, se observa que el coste de asignarle tareas a los últimos trabajadores es muy grande, por lo que se plantea asignar por cada tarea, qué trabajador es el que produce menos coste en la realización. Se empieza comparando para la tarea 1 los trabajadores posibles, siendo el mejor candidato el trabajador 1, luego para la tarea 2 se comprueban todos los trabajadores menos el primero que ya tiene tarea (en ese caso, la tarea 2 sería realizada por el trabajador 3, que es, de todos los que tienen el coste mínimo para esa tarea, el primero que encuentra). Siguiendo este planteamiento resulta un coste de 12 y un reparto de:

```
Trabajador para cada tarea:
La tarea 1 es hecha por el trabajador 1 que tiene un coste de: 1
La tarea 2 es hecha por el trabajador 3 que tiene un coste de: 4
La tarea 3 es hecha por el trabajador 4 que tiene un coste de: 4
La tarea 4 es hecha por el trabajador 5 que tiene un coste de: 2
La tarea 5 es hecha por el trabajador 2 que tiene un coste de: 1
El coste total asignando trabajadores es: 12
```

En este caso, el resultado parece ser más óptimo porque obtiene con los mismos datos, un coste menor por el simple hecho de aplicar otro criterio de reparto.

Pero, ¿y qué ocurre con los casos en los que hay una igualdad de costes como hemos visto en el caso anterior? El algoritmo planteado es básico. Realiza las operaciones más simples de un algoritmo voraz, que es escoger el mejor que encuentre en ese momento, sin pensar en lo que ha hecho o en lo que hará. A pesar de existir infinidad de soluciones a este problema, se plantea una algo más compleja que las dos comprobadas, pero de eficacia relativa.

Si nos situamos en el caso en el que las tareas son las comprobadas y asignadas a cada trabajador, se propone sumar el coste que supone cada tarea con cada uno de los trabajadores. En la tabla de costes vista previamente, quedaría un resultado como este, para hacerlo más visual:

	Tarea 1	**Tarea 2**	**Tarea 3**	**Tarea 4**	**Tarea 5**
Coste total	18	28	33	20	25

Así, cuando nos encontremos una igualdad de costes, seleccionamos aquella tarea que tenga un valor mayor en la suma de la columna. Esto es porque, el tener una suma mayor, indica que **por lo general** el coste de esa misma tarea en otros trabajadores es mayor. Si en este caso hay un empate y selecciono la que por lo general tiene costes mayores, elimino la posibilidad de que esa cifra alta sea la que le toque a la última asignación (que inevitablemente se queda con la tarea sobrante, tenga el coste que tenga). Como hay igualdad, da igual cual cojas porque sumas el mismo coste, pero de quitar valores grandes del medio, es posible que más adelante las opciones que queden sean menos costosas. Reitero que esta solución solo es efectiva en casos realmente concretos, pero aplicándolo al ejemplo que traemos desde el principio de la observación, vemos que:

```
Tareas que hace cada trabajador teniendo en cuenta el coste del resto:
El trabajador 1 hace la tarea 1 que tiene un coste de: 1
El trabajador 2 hace la tarea 5 que tiene un coste de: 1
El trabajador 3 hace la tarea 4 que tiene un coste de: 1
El trabajador 4 hace la tarea 3 que tiene un coste de: 4
El trabajador 5 hace la tarea 2 que tiene un coste de: 4
El coste total asignando tareas es: 11
```

Efectivamente cumple con la restricción de que un trabajador implica una única tarea y viceversa:

	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5
Trabajador 1	X				
Trabajador 2					X
Trabajador 3				X	
Trabajador 4			X		
Trabajador 5		X			

Y como se observa, el coste es 11, menor que ambos casos anteriores.

Aplicando el mismo razonamiento al caso de asignar trabajadores a tareas (sumando en este caso el contenido de las filas y no de las columnas), se obtendría tal reparto:

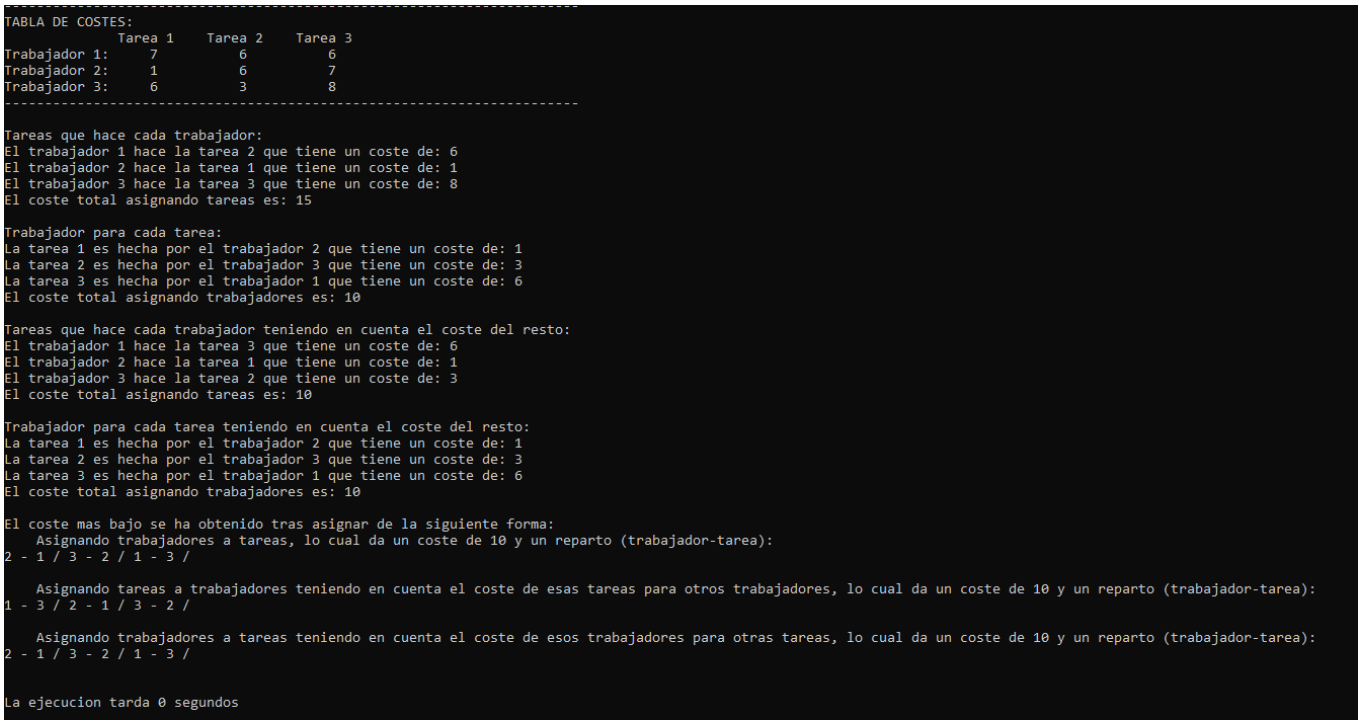
```
Trabajador para cada tarea teniendo en cuenta el coste del resto:
La tarea 1 es hecha por el trabajador 1 que tiene un coste de: 1
La tarea 2 es hecha por el trabajador 4 que tiene un coste de: 4
La tarea 3 es hecha por el trabajador 3 que tiene un coste de: 3
La tarea 4 es hecha por el trabajador 5 que tiene un coste de: 2
La tarea 5 es hecha por el trabajador 2 que tiene un coste de: 1
El coste total asignando trabajadores es: 11
```

Que también cumple con la asignación única:

	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5
Trabajador 1	X				
Trabajador 2				X	
Trabajador 3			X		
Trabajador 4					X
Trabajador 5		X			

Y su coste es 11, también menor que los dos algoritmos básicos originales.

Tras comprobar que existen casos en los que el razonamiento del empate es mejor, solo queda decidirse por uno en cada caso. Como la generación de coste, aunque limitada en el intervalo [1, 9], es aleatoria, las posibilidades son infinitas; así que la solución propuesta es la implementación de estas 4 versiones y tras la asignación de cada una, coger aquellas que devuelvan el coste mínimo y asumirlo como solución óptima al problema. La solución proporcionaría información sobre cada algoritmo, sobre cada emparejamiento y sobre cada coste. Para verlo, se proporciona otra ejecución aleatoria con sus resultados:



Para el caso en el que se introducen más tareas que trabajadores o al revés, se rellena de la dimensión que sea menor hasta que sea una matriz cuadrada. El resto de funcionamiento es idéntico.

2. *Descripción del método voraz:*

El algoritmo voraz consiste en la elección de la mejor opción de entre un conjunto de candidatos en cada momento, sin tener en cuenta lo ya hecho previamente, hasta obtener la solución al problema. Inicialmente debe partir de un conjunto de candidatos seleccionados vacío. De esta forma, partimos de un contexto en el que ninguna tarea ha sido asignada a ningún trabajador y ningún trabajador ha sido seleccionado como mejor opción para una tarea. Por lo general, se aplica a problemas si tenemos:

1. *Candidatos y conjunto de candidatos ya usados*:

En nuestro caso los candidatos son las tareas o trabajadores que se quieran asignar y los ya usados se van almacenando en S.

2. *Un criterio que diga cuándo un conjunto de candidatos es una solución:*

Aquí sería el haber repartido todas las tareas y todos los trabajadores de tal forma que ninguno se repita.

3. *Una función que indique cuándo un conjunto de candidatos es posible o no para obtener una solución:*

$$\textbf{Min } \{\sum_{i=0...n-1} s[i] \;/\; s[j] > 0;\ s[j] \in 0 <= s[j] < 9,\ j = 0...n-1\}$$

4. *Una función de selección que da en cada etapa el candidato más prometedor de los no usados:*

La comparación de todos los costes de las tareas/trabajadores restantes y la selección del valor más bajo.

5. *Función objetivo que da una solución:*

Minimizar los costes de las asignaciones.

No se garantiza la obtención del mínimo global como sería deseable, pero sí da siempre soluciones óptimas locales.

3. *Pseudocódigo del problema:*

1º Algoritmo (Tareas asignadas a trabajadores):

ALGORITMO AsignarTareasATrabajadores(matriz[][], trabajadores, tareas, tareas_asignadas[], mayor_fila_columna)

```
VAR
        ENTERO menor, coste_total = 0
        BOOL ignorar
INICIO
        FOR i DESDE 0 HASTA mayor_fila_columna – 1
                Menor  <- MAX  //Mayor valor que acepta un entero. Es equivalente a inicializarlo a infinito porque sabes que cualquier número es más pequeño. Variable global
                FOR j DESDE 0 HASTA mayor_fila_columna – 1
                        Ignorar <- FALSE
                        FOR k DESDE 0 HASTA mayor_fila_columna – 1
                                Si (j = tareas_asignadas[k]) entonces
                                        Ignorar <- TRUE
                                FinSi
                        FIN FOR
                        Si (NO ignorar) entonces
                                Si (matriz[i, j] < menor) entonces
                                        Tareas_asignadas[i] <- j
                                        Menor <- matriz[i][j]
                                FinSi
                        FinSi
                FIN FOR
                Coste_total <- coste_total + matriz[i, tareas_asignadas[i]]
        FIN FOR
        DEVUELVE coste_total
FINAL
```

2º Algoritmo (Trabajadores a tareas):

ALGORITMO AsignarTrabajadoresATareas (matriz[][], trabajadores, tareas, trabajadores_asignado[], mayor_fila_columna)

```
VAR
        ENTERO menor, coste_total = 0
        BOOL ignorar
INICIO
        FOR i DESDE 0 HASTA mayor_fila_columna – 1
                Menor  <- MAX //Mayor valor que acepta un entero. Es equivalente a inicializarlo a infinito porque sabes que cualquier número es más pequeño. Variable global
                FOR j DESDE 0 HASTA mayor_fila_columna – 1
                        Ignorar <- FALSE
                        FOR k DESDE 0 HASTA mayor_fila_columna – 1
                                Si (j = trabajadores_asignados[k]) entonces
                                        Ignorar <- TRUE
                                FinSi
                        FIN FOR
                        Si (NO ignorar) entonces
                                Si (matriz[j, i] < menor) entonces
                                        trabajadores_asignados [i] <- j
                                        Menor <- matriz[j][i]
                                FinSi
                        FinSi
                FIN FOR
                Coste_total <- coste_total+matriz[trabajadores_asignados[i],i]
        FIN FOR
        DEVUELVE coste_total
FINAL
```

3º Algoritmo (Tareas a trabajadores con suma):

ALGORITMO AsignarTareasATrabajadoresConSuma (matriz[][], trabajadores, tareas, tareas_asignadas_suma [], mayor_fila_columna)

```
VAR
        ENTERO menor, coste_total = 0, suma
        BOOL ignorar
        VECTOR DINAMICO ENTERO aux[TAM <- mayor_fila_columna]
INICIO
        FOR i DESDE 0 HASTA mayor_fila_columna – 1
                Suma <- 0
                FOR j DESDE 0 HASTA mayor_fila_columna – 1
                        Suma <- suma + matriz[j,i]
                FIN FOR
        FIN FOR
        FOR i DESDE 0 HASTA mayor_fila_columna – 1
                Menor  <- MAX //Mayor valor que acepta un entero. Es equivalente a inicializarlo a infinito porque sabes que cualquier número es más pequeño. Variable global
                FOR j DESDE 0 HASTA mayor_fila_columna – 1
                        Ignorar <- FALSE
                        FOR k DESDE 0 HASTA mayor_fila_columna – 1
                                Si (j = tareas_asignadas_suma[k]) entonces
                                        Ignorar <- TRUE
                                FinSi
                        FIN FOR
                        Si (NO ignorar) entonces
                                Si (matriz[i, j] < menor) entonces
                                        trabajadores_asignados [i] <- j
                                        Menor <- matriz[i][j]
                                Si no pero (matriz[i][j] = menor) entonces
                                        Si (aux[j] > aux[tareas_asignadas_suma[i]]) entonces
                                                trabajadores_asignados [i] <- j
                                                Menor <- matriz[i][j]
```

```
                                        Fin Si
                                    FinSi
                                FinSi
                            FIN FOR
                            Coste_total <- coste_total+matriz[i,tareas_asignadas_suma[i]]
                        FIN FOR
                        DEVUELVE coste_total
FINAL
```

4º Algoritmo (Trabajadores a tareas con suma):

ALGORITMO AsignarTrabajadoresATareas (matriz[][], trabajadores, tareas, trabajadores_asignados_suma [], mayor_fila_columna)

```
VAR
        ENTERO menor, coste_total = 0, suma
        BOOL ignorar
        VECTOR DINAMICO ENTERO aux[TAM <- mayor_fila_columna]
INICIO
        FOR i DESDE 0 HASTA mayor_fila_columna – 1
                Suma <- 0
                FOR j DESDE 0 HASTA mayor_fila_columna – 1
                        Suma <- suma + matriz[i,j]
                FIN FOR
        FIN FOR
        FOR i DESDE 0 HASTA mayor_fila_columna – 1
                Menor  <- MAX //Mayor valor que acepta un entero. Es equivalente a inicializarlo a infinito porque sabes que cualquier número es más pequeño. Variable global
                FOR j DESDE 0 HASTA mayor_fila_columna – 1
                        Ignorar <- FALSE
                        FOR k DESDE 0 HASTA mayor_fila_columna – 1
                                Si (j = trabajadores_asignados_suma [k]) entonces
                                        Ignorar <- TRUE
                                FinSi
                        FIN FOR
                        Si (NO ignorar) entonces
                                Si (matriz[j, i] < menor) entonces
                                        trabajadores_asignados_suma [i] <- j
                                        Menor <- matriz[j][i]
                                Si no pero (matriz[j][i] = menor) entonces
                                        Si (aux[j] > aux[trabajadores_asignados_suma[i]]) entonces
                                                trabajadores_asignados_suma[i] <- j
```

```
                                        Menor <- matriz[i][j]
                                    Fin Si
                                FinSi
                            FinSi
                    FIN FOR
                    Coste_total <- coste_total + matriz[trabajadores_asignados_suma[i],i]
            FIN FOR
            DEVUELVE coste_total
FINAL
```

EJEMPLOS Y CASOS DE EJECUCION CON RESULTADOS Y EXPLICACION DEL PROBLEMA ASIGNADO 3.3: TRABAJADORES Y TAREAS

Existe una carpeta "*CODIGOS Y EJECUTABLES*" en la que se encuentra un .cpp llamado "*trabajadores_tareas_greedy_por_pasos.cpp*" que no es un código precisamente eficiente, es una versión adaptada a que la salida por pantalla muestre paso por paso las comprobaciones y asignaciones que hace para facilitarlo al usuario que haga uso de él con el único fin de hacerlo comprensible. Una forma de probarlo sería compilarlo con el make o con la siguiente línea:

```
g++ trabajadores_tareas_greedy_por_pasos.cpp -o ejecutable
```

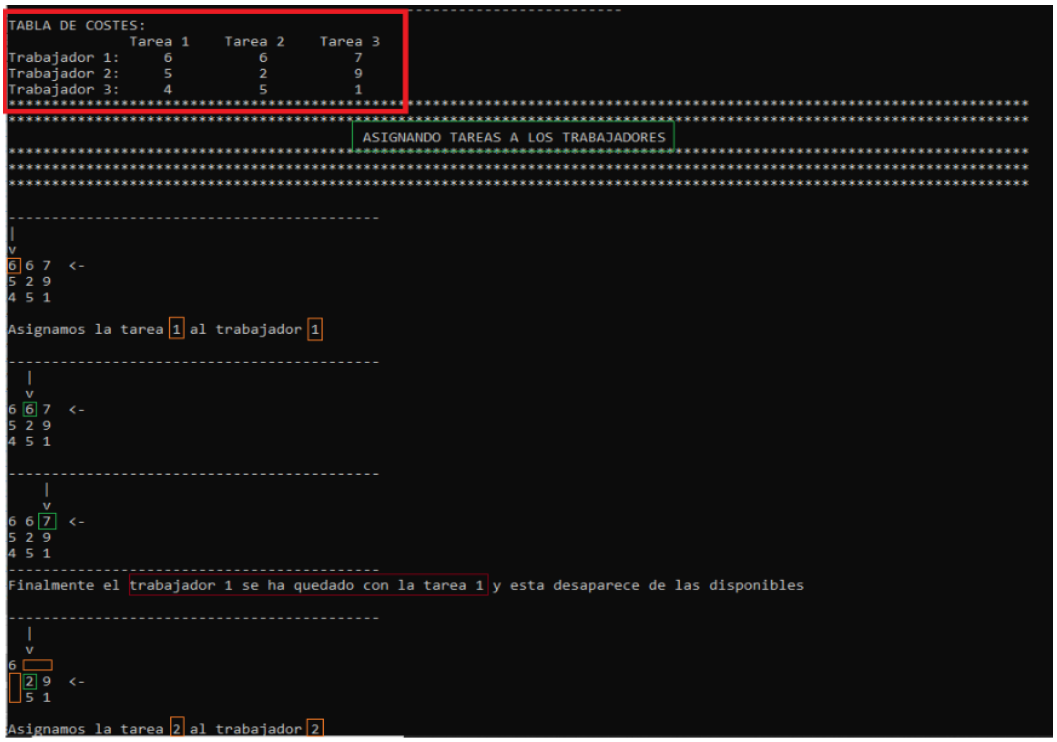
Y probarlo con la ejecución tal que:

```
./ejecutable [nº_tareas] [nº_trabajadores]
```

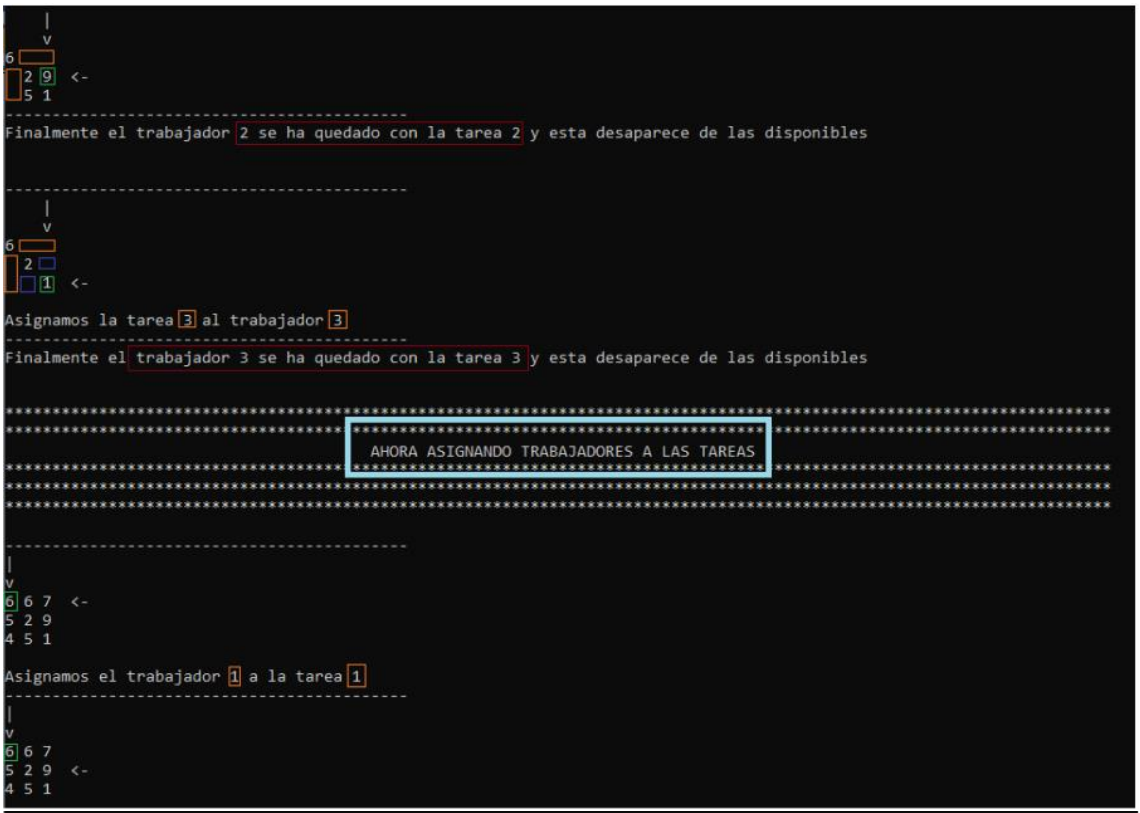
Para poder ver el ejemplo de cualquier caso que se considere oportuno. Para facilitar la comprobación, se adjuntan varios ejemplos que cubran las diferentes opciones de ejecución que pueden darse con el código y una breve explicación sobre estos.

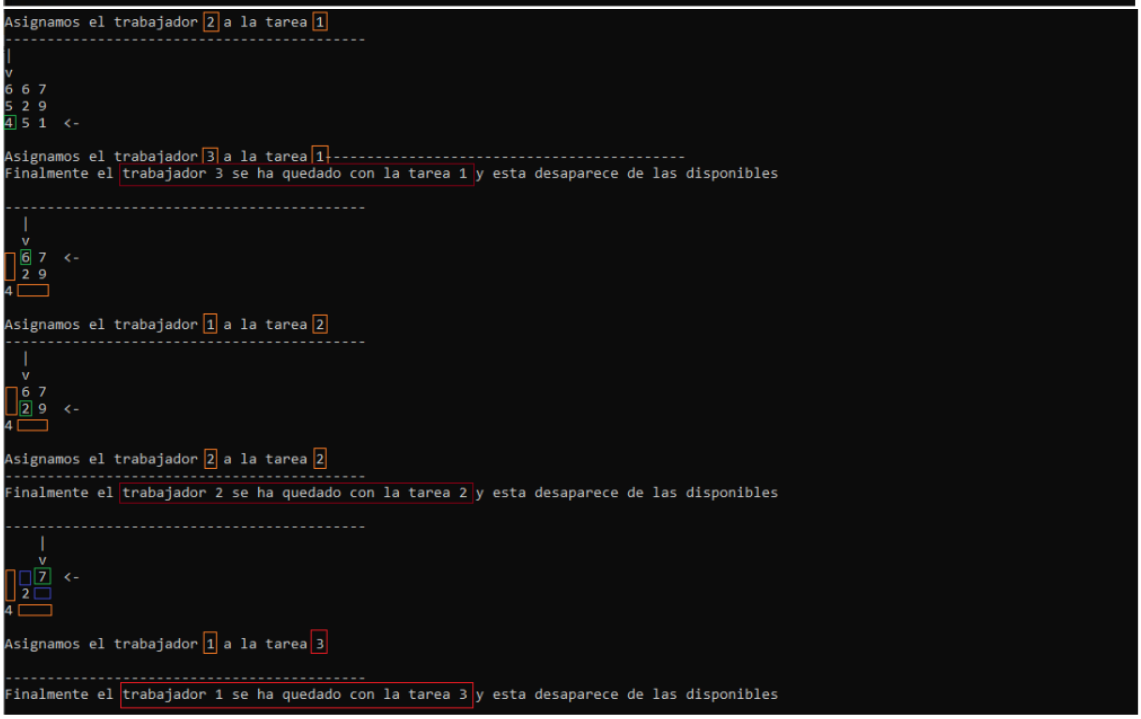
Ejemplo 1: Nº de trabajadores = 3 / Nº de tareas = 3. Ejecución estándar.

En este caso vamos a probar a introducir 3 trabajadores y 3 tareas. En el recuadro rojo se genera una tabla aleatoria de datos comprendidos en [1, 9] que representan los costes de la realización de las diferentes tareas (columnas) con los diferentes trabajadores (filas). Lo primero que se lleva a cabo es una asignación de tareas a trabajadores. Los recuadros verdes muestran qué elemento de la matriz se está estudiando. Estudiar un elemento consiste en comprobar si en esa fila o columna hay otra tarea o trabajador que cumpla con los requisitos necesarios y tenga un costo menor. Por defecto, la primera tarea disponible se asigna automáticamente al trabajador que se está evaluando y a raíz de eso ya depende del costo. Lo mismo ocurre con la asignación automática del primer trabajador libre para la tarea que se evalúe, facilitando las siguientes comprobaciones. Los cuadros naranjas son las asignaciones que se hacen en cada momento y los huecos que quedan en blando cuando se pasa de trabajador o tarea. Ocurre porque obligatoriamente una tarea es realizada por un trabajador solamente y un trabajador realiza una tarea solamente. Por tanto, al asignar un elemento, tanto la fila (el trabajador) como la columna (la tarea) dejan de estar disponibles para futuras asignaciones.

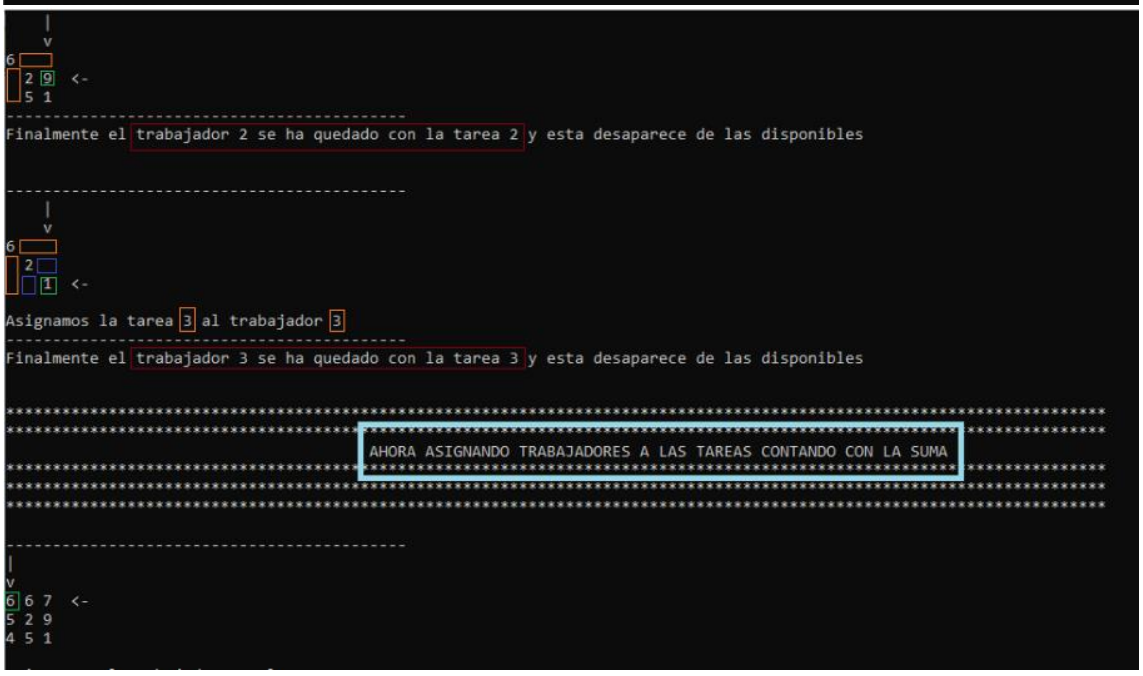


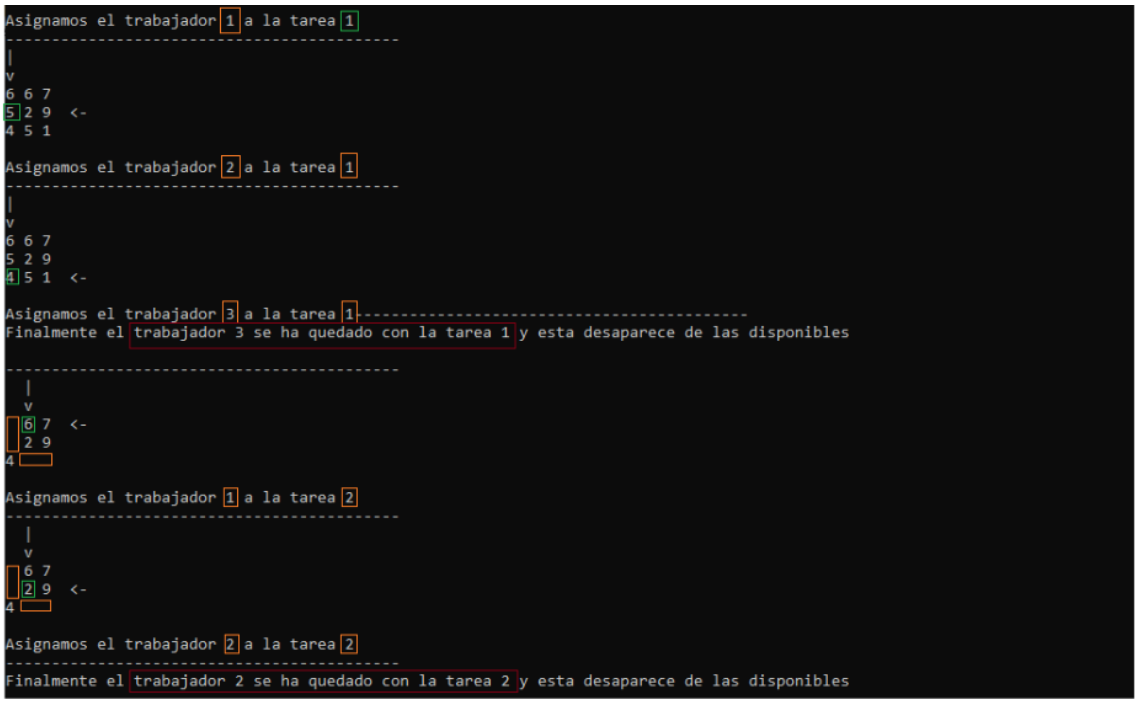
Las flechas que iteran indican qué elemento se va evaluando. La explicación del algoritmo se encuentra en un lugar externo a los ejemplos como este, pero básicamente comprueba si la tarea que se estudia tiene menos coste que otra asignada. De ser así, se le asigna esta y si no, es ignorada. Conociendo el funcionamiento básico, podemos ver a simple vista cómo avanzan los números a lo largo de las filas y columnas para hacer todas las comprobaciones. Si la relación trabajador-tarea que evalúa es mejor que la anterior, se muestra un texto bajo la matriz especificándolo y, al terminar la fila completa (haber evaluado para un trabajador i concreto todas las tareas j posibles), se indica finalmente qué trabajador se ha quedado qué tarea y se pasa a evaluar al siguiente (recuadro corinto).

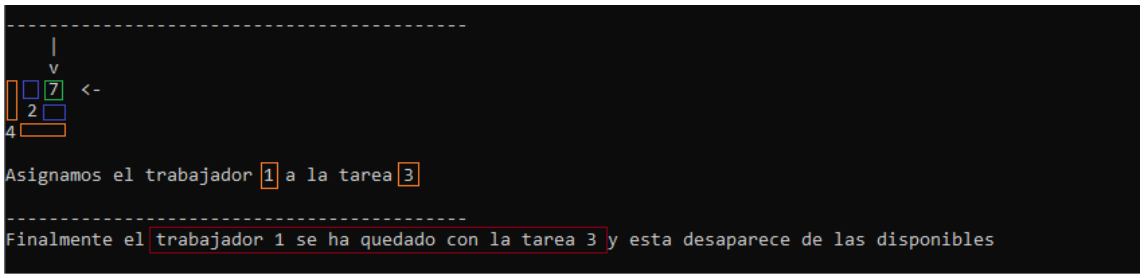

|
v
6
2 9 <-
5 1
Finalmente el trabajador 2 se ha quedado con la tarea 2 y esta desaparece de las disponibles
|
v
6
2
1 <-
Asignamos la tarea 3 al trabajador 3
Finalmente el trabajador 3 se ha quedado con la tarea 3 y esta desaparece de las disponibles
AHORA ASIGNANDO TRABAJADORES A LAS TAREAS
|
v
6 6 7 <-
5 2 9
4 5 1
Asignamos el trabajador 1 a la tarea 1
|
v
6 6 7
5 2 9 <-
4 5 1


Asignamos el trabajador 2 a la tarea 1
|
v
6 6 7
5 2 9
4 5 1 <-
Asignamos el trabajador 3 a la tarea 1
Finalmente el trabajador 3 se ha quedado con la tarea 1 y esta desaparece de las disponibles
|
v
6 7 <-
2 9
4
Asignamos el trabajador 1 a la tarea 2
|
v
6 7
2 9 <-
4
Asignamos el trabajador 2 a la tarea 2
Finalmente el trabajador 2 se ha quedado con la tarea 2 y esta desaparece de las disponibles
|
v
7 <-
2
4
Asignamos el trabajador 1 a la tarea 3
Finalmente el trabajador 1 se ha quedado con la tarea 3 y esta desaparece de las disponibles

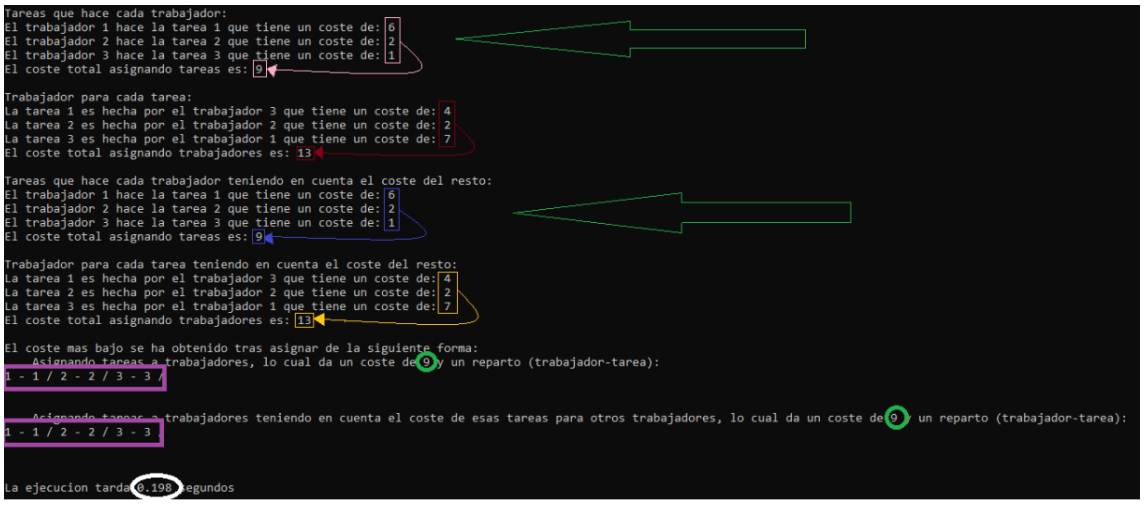

AHORA ASIGNANDO TAREAS A LOS TRABAJADORES CONTANDO CON LA SUMA
6 6 7 <-
5 2 9
4 5 1
Asignamos la tarea 1 al trabajador 1
6 6 7 <-
5 2 9
4 5 1
6 6 7 <-
5 2 9
4 5 1
Finalmente el trabajador 1 se ha quedado con la tarea 1 y esta desaparece de las disponibles
6
2 9 <-
5 1
Asignamos la tarea 2 al trabajador 2


6
2 9 <-
5 1
Finalmente el trabajador 2 se ha quedado con la tarea 2 y esta desaparece de las disponibles
6
2
1 <-
Asignamos la tarea 3 al trabajador 3
Finalmente el trabajador 3 se ha quedado con la tarea 3 y esta desaparece de las disponibles
AHORA ASIGNANDO TRABAJADORES A LAS TAREAS CONTANDO CON LA SUMA
6 6 7 <-
5 2 9
4 5 1





Los huecos vacíos tras cada comprobación resaltan la ausencia de valores en la fila y columna que acaban de ser emparejadas como una asignación definitiva. De esta forma, y con el objetivo de limpiar la salida para el usuario, se elimina la muestra de estos valores sucesivamente con cada línea que evaluamos.

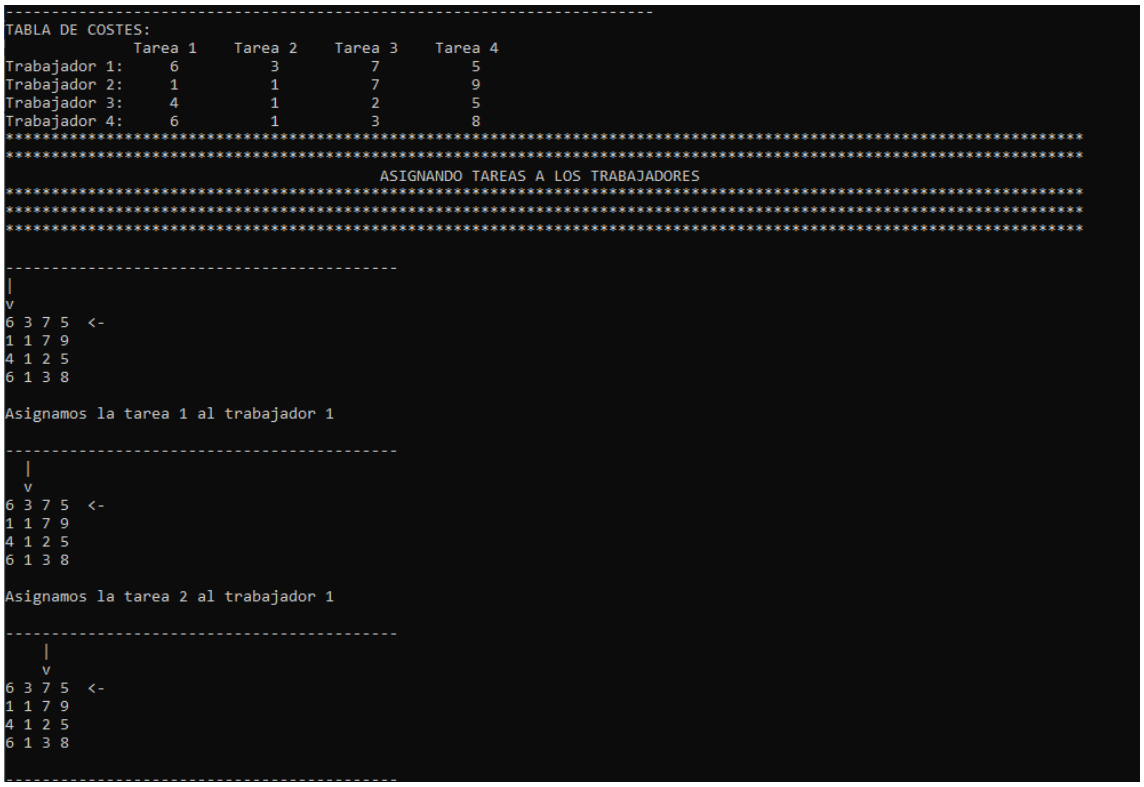


En la anterior imagen se muestra por pantalla el resultado de ambos estudios realizados. Un pequeño resumen que recopila qué tareas son realizadas por qué trabajadores en cada caso y una suma de sus costes (colores rosa, rojo, azul y amarillo). A partir de la obtención de datos,

se muestra una respuesta coherente con los valores obtenidos. En este caso concreto, como se ve con las flechas verdes, sale mejor resultado asignando tareas a trabajadores y tareas a trabajadores con un razonamiento algo más específico para el caso de lo empates de coste. Como el desarrollo de la asignación acaba eliminando toda situación de empate, el 3º algoritmo y el primero realizan el mismo procedimiento y obtienen las mismas asignaciones, pero más adelante veremos que no siempre es así y existen casos para los que razonar los empates es mejor opción. Finalmente, muestra la relación trabajadores-tareas que se produce como mejor. Otro dato observable es el tiempo que tarda **solamente el algoritmo que estudia las asignaciones**, no el tiempo de muestra por pantalla o recepción de datos. En este caso, al ser un código alterado, el tiempo no es totalmente válido, por lo que es mejor medir este en la ejecución estándar del fichero o en el fichero adjunto "trabajadores_tareas_greedy_tiempos.cpp".

Tras esta explicación, el resto de ejemplos serán expuestos como una sucesión de imágenes que muestran el mismo procedimiento ya observado, pero para diferentes casos, haciéndonos ver diferentes alternativas de ejecución y resultados.

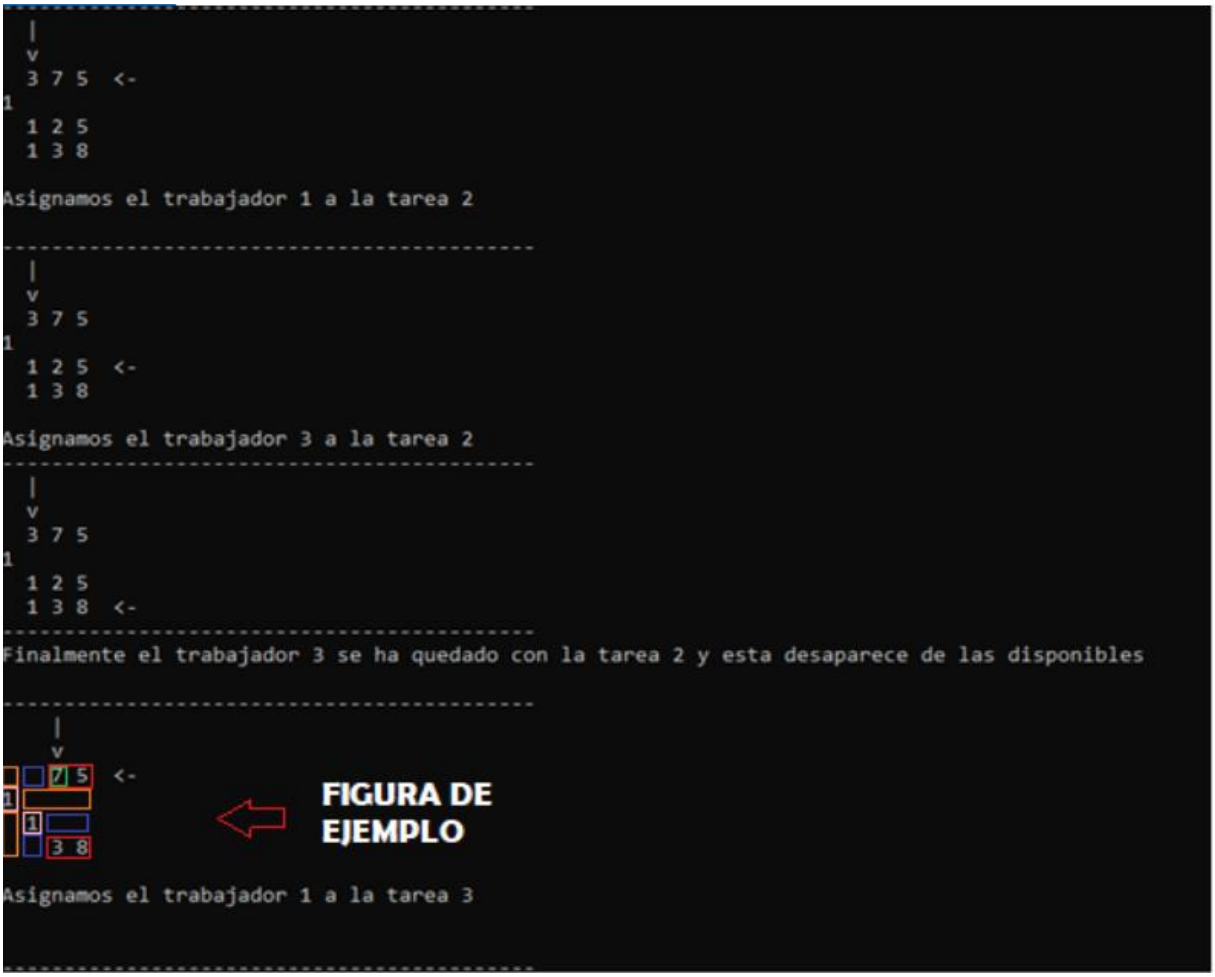
Ejemplo 2: Nº de trabajadores = 4 / Nº de tareas = 4. Igual al ejemplo anterior, pero con otras dimensiones.



```
      |
      v
6 3 7 5  <-
1 1 7 9
4 1 2 5
6 1 3 8
-----------------------------------------
Finalmente el trabajador 1 se ha quedado con la tarea 2 y esta desaparece de las disponibles
-----------------------------------------
|
v
  3
1   7 9  <-
4   2 5
6   3 8

Asignamos la tarea 1 al trabajador 2


-----------------------------------------
    |
    v
  3
1   7 9  <-
4   2 5
6   3 8

-----------------------------------------
      |
      v
  3
1   7 9  <-
4   2 5
6   3 8
-----------------------------------------
Finalmente el trabajador 2 se ha quedado con la tarea 1 y esta desaparece de las disponibles


-----------------------------------------
-----------------------------------------
    |
    v
  3
1
    2 5  <-
    3 8

Asignamos la tarea 3 al trabajador 3

-----------------------------------------
      |
      v
  3
1
    2 5  <-
    3 8
-----------------------------------------
Finalmente el trabajador 3 se ha quedado con la tarea 3 y esta desaparece de las disponibles



-----------------------------------------
      |
      v
  3
1
    2
      8  <-

Asignamos la tarea 4 al trabajador 4
-----------------------------------------
Finalmente el trabajador 4 se ha quedado con la tarea 4 y esta desaparece de las disponibles


**********************************************************************************************************************************
**********************************************************************************************************************************
                                    AHORA ASIGNANDO TRABAJADORES A LAS TAREAS
**********************************************************************************************************************************
**********************************************************************************************************************************
**********************************************************************************************************************************

-----------------------------------------
```

En la siguiente imagen se han añadido algunos colores a un caso concreto para facilitar el entendimiento de lo que ocurre con los huecos en blanco, los números que quedan y las comprobaciones por realizar. Para hacerlo más visual:

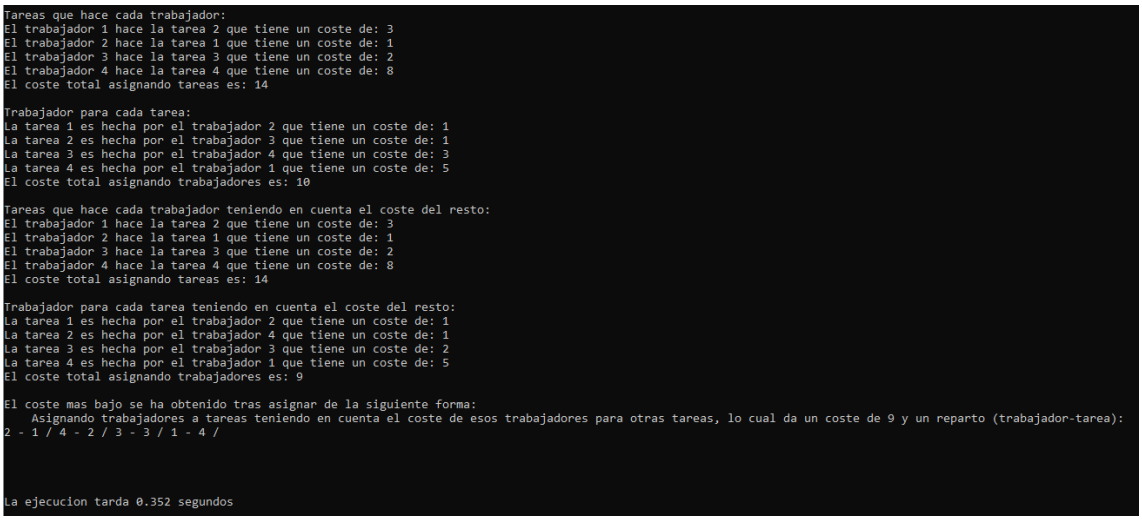
- Cuadros rosas: Asignaciones ya hechas. El coste de los emparejamientos que van hasta ahora, colocados en la fila y columna que corresponde con su trabajador y tarea correspondiente. Obviamente, con la restricción de que una trabajador solo realice una tarea y que una tarea sea únicamente realizada por un trabajador, cuando se muestran datos como los recuadrados en rosa, se observa que no coinciden ni en fila ni en columna con ningún otro dato.
- Huecos naranjas: Números que se han eliminado tras la primera asignación (la tarea 1 con el trabajador 2 y coste 1)
- Huecos azules: Números que se han eliminado tras la segunda asignación (tarea 2, trabajador 3 y coste 1)
- Números en rojo: Coste de tareas aún asignables en las filas de trabajadores que aún pueden tener asignaciones nuevas.
- Número en verde: Evaluado en ese momento (se ve por las flechas de alrededor de la matriz)



```
----------------------------------------
    |
    v
    7 5
1
  1
    3 8  <-

Asignamos el trabajador 4 a la tarea 3----------------------------------------
Finalmente el trabajador 4 se ha quedado con la tarea 3 y esta desaparece de las disponibles

----------------------------------------
     |
     v
     5  <-
1
  1
    3

Asignamos el trabajador 1 a la tarea 4


----------------------------------------
Finalmente el trabajador 1 se ha quedado con la tarea 4 y esta desaparece de las disponibles



****************************************************************************************************************************************
****************************************************************************************************************************************
                                    AHORA ASIGNANDO TAREAS A LOS TRABAJADORES CONTANDO CON LA SUMA
****************************************************************************************************************************************
****************************************************************************************************************************************
****************************************************************************************************************************************

----------------------------------------
|
v
6 3 7 5  <-
1 1 7 9
4 1 2 5
6 1 3 8

Asignamos la tarea 1 al trabajador 1

----------------------------------------
  |
  v
6 3 7 5  <-
1 1 7 9
4 1 2 5
6 1 3 8

Asignamos la tarea 2 al trabajador 1

----------------------------------------
    |
    v
6 3 7 5  <-
1 1 7 9
4 1 2 5
6 1 3 8

----------------------------------------
     |
     v
6 3 7 5  <-
1 1 7 9
4 1 2 5
6 1 3 8
----------------------------------------
Finalmente el trabajador 1 se ha quedado con la tarea 2 y esta desaparece de las disponibles
----------------------------------------
|
v
  3
1   7 9  <-
4   2 5
6   3 8

Asignamos la tarea 1 al trabajador 2


----------------------------------------
```

```
-------------------------------------------
    |
    v
  3
1   7 9  <-
4   2 5
6   3 8

-------------------------------------------
      |
      v
  3
1   7 9  <-
4   2 5
6   3 8
-------------------------------------------
Finalmente el trabajador 2 se ha quedado con la tarea 1 y esta desaparece de las disponibles


-------------------------------------------
    |
    v
  3
1
    2 5  <-
    3 8

Asignamos la tarea 3 al trabajador 3

-------------------------------------------
      |
      v
  3
1
    2 5  <-
    3 8
-------------------------------------------
Finalmente el trabajador 3 se ha quedado con la tarea 3 y esta desaparece de las disponibles
-------------------------------------------
     |
     v
  3
1
   2
     8  <-

Asignamos la tarea 4 al trabajador 4
-------------------------------------------
Finalmente el trabajador 4 se ha quedado con la tarea 4 y esta desaparece de las disponibles


**********************************************************************************************************************
**********************************************************************************************************************
                                    AHORA ASIGNANDO TRABAJADORES A LAS TAREAS CONTANDO CON LA SUMA
**********************************************************************************************************************
**********************************************************************************************************************
**********************************************************************************************************************

-------------------------------------------
|
v
6 3 7 5  <-
1 1 7 9
4 1 2 5
6 1 3 8

Asignamos el trabajador 1 a la tarea 1
-------------------------------------------
|
v
6 3 7 5
1 1 7 9  <-
4 1 2 5
6 1 3 8

Asignamos el trabajador 2 a la tarea 1
-------------------------------------------
```

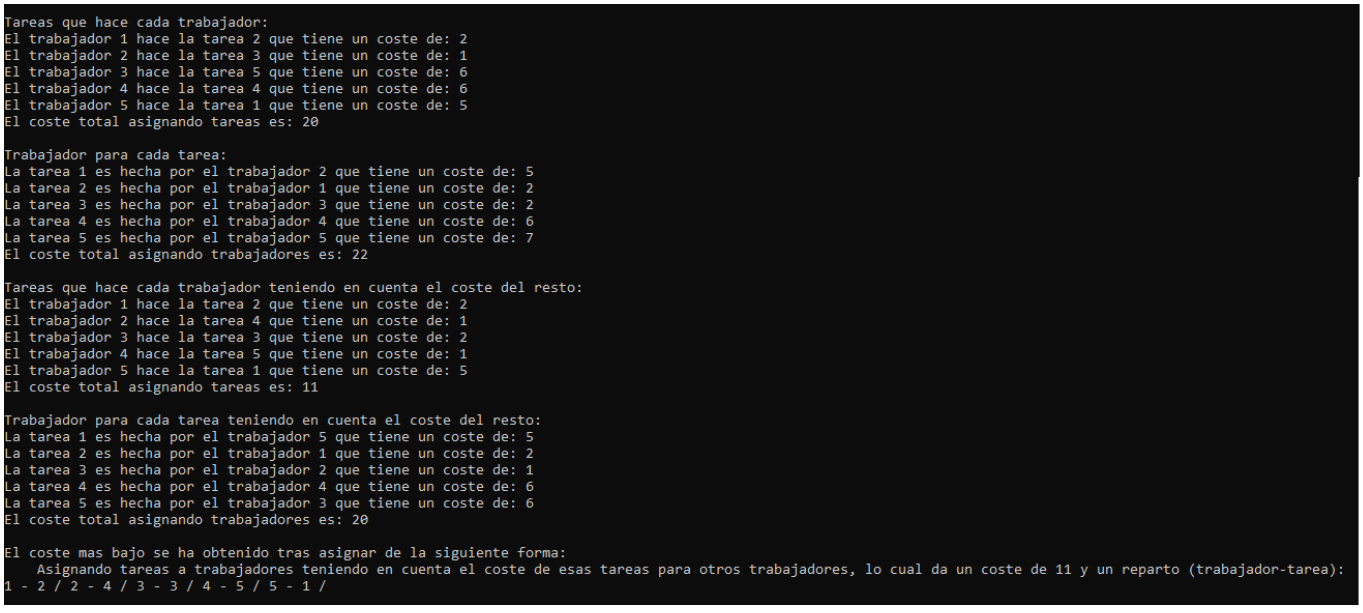
```
----------------------------------------
|
v
6 3 7 5
1 1 7 9
4 1 2 5  <-
6 1 3 8

----------------------------------------
|
v
6 3 7 5
1 1 7 9
4 1 2 5
6 1 3 8  <-
----------------------------------------
Finalmente el trabajador 2 se ha quedado con la tarea 1 y esta desaparece de las disponibles

----------------------------------------
  |
  v
  3 7 5  <-
1
  1 2 5
  1 3 8

Asignamos el trabajador 1 a la tarea 2

----------------------------------------
  |
  v
  3 7 5
1
  1 2 5  <-
  1 3 8

Asignamos el trabajador 3 a la tarea 2
----------------------------------------
  |
  v
  3 7 5
1
  1 2 5
  1 3 8  <-
----------------------------------------
----------------------------------------
Finalmente el trabajador 4 se ha quedado con la tarea 2 y esta desaparece de las disponibles

----------------------------------------
    |
    v
    7 5  <-
1
    2 5
  1

Asignamos el trabajador 1 a la tarea 3

----------------------------------------
    |
    v
    7 5
1
    2 5  <-
  1

Asignamos el trabajador 3 a la tarea 3
----------------------------------------
Finalmente el trabajador 3 se ha quedado con la tarea 3 y esta desaparece de las disponibles

----------------------------------------
      |
      v
      5  <-
1
    2
  1

Asignamos el trabajador 1 a la tarea 4


----------------------------------------
Finalmente el trabajador 1 se ha quedado con la tarea 4 y esta desaparece de las disponibles
```



Este es uno de esos casos concretos en los que, haber añadido un algoritmo que en caso de empate evalúa más opciones que la de quedarse con el primero que se encuentra, es útil y genera una asignación que cumple con las restricciones y da menos coste que el resto.

Ejemplo 3: Nº de trabajadores = 3 / Nº de tareas = 5. Ejemplo de funcionamiento de los algoritmos extras.

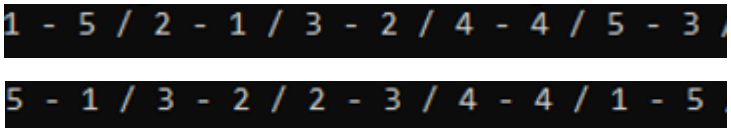
Otro ejemplo que sirve como muestra de que en algunos casos es mejor el algoritmo que tiene en cuenta la suma que el que asigna por orden de encuentro.



Ejemplo 4: Nº de trabajadores = 5 / Nº de tareas = 5. Empate con diferentes asignaciones.

```
Tareas que hace cada trabajador:
El trabajador 1 hace la tarea 5 que tiene un coste de: 2
El trabajador 2 hace la tarea 1 que tiene un coste de: 4
El trabajador 3 hace la tarea 2 que tiene un coste de: 2
El trabajador 4 hace la tarea 4 que tiene un coste de: 5
El trabajador 5 hace la tarea 3 que tiene un coste de: 4
El coste total asignando tareas es: 17

Trabajador para cada tarea:
La tarea 1 es hecha por el trabajador 5 que tiene un coste de: 2
La tarea 2 es hecha por el trabajador 3 que tiene un coste de: 2
La tarea 3 es hecha por el trabajador 2 que tiene un coste de: 6
La tarea 4 es hecha por el trabajador 4 que tiene un coste de: 5
La tarea 5 es hecha por el trabajador 1 que tiene un coste de: 2
El coste total asignando trabajadores es: 17

Tareas que hace cada trabajador teniendo en cuenta el coste del resto:
El trabajador 1 hace la tarea 5 que tiene un coste de: 2
El trabajador 2 hace la tarea 1 que tiene un coste de: 4
El trabajador 3 hace la tarea 2 que tiene un coste de: 2
El trabajador 4 hace la tarea 4 que tiene un coste de: 5
El trabajador 5 hace la tarea 3 que tiene un coste de: 4
El coste total asignando tareas es: 17

Trabajador para cada tarea teniendo en cuenta el coste del resto:
La tarea 1 es hecha por el trabajador 5 que tiene un coste de: 2
La tarea 2 es hecha por el trabajador 3 que tiene un coste de: 2
La tarea 3 es hecha por el trabajador 2 que tiene un coste de: 6
La tarea 4 es hecha por el trabajador 4 que tiene un coste de: 5
La tarea 5 es hecha por el trabajador 1 que tiene un coste de: 2
El coste total asignando trabajadores es: 17

El coste mas bajo se ha obtenido tras asignar de la siguiente forma:
    Asignando tareas a trabajadores, lo cual da un coste de 17 y un reparto (trabajador-tarea):
1 - 5 / 2 - 1 / 3 - 2 / 4 - 4 / 5 - 3 /


    Asignando trabajadores a tareas, lo cual da un coste de 17 y un reparto (trabajador-tarea):
5 - 1 / 3 - 2 / 2 - 3 / 4 - 4 / 1 - 5 /

    Asignando tareas a trabajadores teniendo en cuenta el coste de esas tareas para otros trabajadores, lo cual da un coste de 17 y un reparto (trabajador-tarea):
1 - 5 / 2 - 1 / 3 - 2 / 4 - 4 / 5 - 3 /

    Asignando trabajadores a tareas teniendo en cuenta el coste de esos trabajadores para otras tareas, lo cual da un coste de 17 y un reparto (trabajador-tarea):
5 - 1 / 3 - 2 / 2 - 3 / 4 - 4 / 1 - 5 /
```

En este caso, el coste es el mismo en los 4 algoritmos, pero se llega a él asignando tareas diferentes a trabajadores diferentes. Muestra así, dos alternativas que plantean lo mismo.



También se comprueba que ocurre cuando se mete un número de trabajadores diferentes al número de tareas. En este caso, se añaden tareas extras para que haya una por cada trabajador, y en el caso de que sea al revés, se añadirán trabajadores que puedan realizar esas tareas.