Algoritmos Avanzados: Insertar publicidad en un diario mediante *Método Goloso*

FELIPE IGNACIO MATURANA GUERRA[1]

[1] *Universidad de Santiago de Chile, Santiago, 2017.*
[*] *Correspondencia del autor: felipe.maturana.g@usach.cl*

Compilado May 8, 2017

El periódico local *La Sexta* en una nueva alternativa para generar ingresos mediante anuncios publicitarios. Como nunca antes realizaron esto, no saben cómo posicionar los anuncios dentro del diario para así optimizar el espacio compartido entre noticias y publicidad entro de una misma página. Es decir, dado una lista de n publicidades con un valor asociado al ser incluida en una de las páginas del diario y sus respectivas dimensiones de largo y ancho. Por otro lado, las páginas poseen dimensiones de largo y ancho asociadas, cabe destacar que la publicidad debe cumplir con la restricción de tamaño disponible en la página en cuestión.Es por esta razón que la empresa solicita un programa en lenguaje de programación *C* y utilizando el método *Goloso*, con criterio a elección nuestra, de modo que les permita a los editores generar la mayor cantidad de ganancia bajo esta modalidad.El código fuente está implementado en Lenguaje C desarrollado en ambiente Linux, respetando el estándar ANSI C.

URL Github: El código fuente de la publicación y los códigos del programa se encuentran alojados en Git-Hub.

https://github.com/Felipez-Maturana/AlgoritmosAvanzados_L3_Goloso-Greedy

1. INTRODUCCIÓN

En el mundo empresarial, es sumamente importante para los dueños la inserción de publicidad dentro de sus páginas ya que es mediante éstas que principalmente sus proyectos y editoriales son financiadas, por lo tanto, optimizar éste proceso de decidir cuál publicidad debe ir y en qué orden cobra mayor sentido ya que debe existir un equilibrio entre el contenido propio del diario/revista frente a la cantidad de publicidad para no perder los seguidores del respectivo diario, es por este motivo, que la cantidad de espacio disponible para la publicidad es *limitado*. La variedad de publicidades es ilimitada, poseen distinto *largo*, *ancho* y *valor* el cual se paga si se llega a incluir la publicidad. La condición que limita nuestra solución es la cantidad de páginas disponibles, donde cada una de ellas posee un ancho A, un largo L y un valor total V, el cual corresponde a la suma de todas las publicidades presentes en la página. Por otro lado, la publicidad posee un ancho Ap, un largo Lp, un valor Vp correspondiente al valor a pagar por incluir la publicidad y un un identificador Ip. Tanto el valor total de cada página como el valor asociado a cada publicidad está en pesos chilenos y las dimensiones (largo y ancho), tanto de la publicidad y la página, está en cms.

2. DESCRIPCIÓN DEL PROBLEMA

El periódico local La Sexta desea implementar la inclusión de publicidad en las páginas de su diario por lo cual ha dispuesto de p páginas, cada una con dimensiones de largo y ancho distintas. En estas páginas se desea incluir una cantidad determinada de las n publicidades dispuestas de modo a optimizar las ganancias obtenidas por la inclusión las publicidades en las páginas disponibles. La publicidad posee largo, ancho y un valor asociado el cual los patrocinadores están dispuestos a pagar si ésta es incluida en las páginas del periódico. Para incluir una publicidad debe incluir ciertas características.

1. La publicidad debe ser publicada de forma íntegra.

2. Se debe optimizar la cantidad de dinero obtenido por la inserción de publicidad, sin embargo, una de las características del método goloso es que no siempre nuestra solución es el óptimo.

3. Una publicidad sólo puede ser utilizada una vez. Además, se debe considerar que el enunciado sugiere que la cantidad de páginas no supere las 10. Cabe señalar que tanto las

dimensiones como el valor a trabajar en esta experiencias son positivos (mayor a 0). Para resolver este problema se debe utilizar el **lenguaje de programación C**, mediante la utilización de la técnica **Goloso**.

A. Entrada

En un archivo de texto titulado "Entrada.in" se encuentran listados todas las publicidades posibles a integrar dentro de nuestras páginas, cada línea posee el identificador, seguido de las dimensiones (ancho y largo) y finalmente el valor de la publicidad, así como se muestra a continuación. Además, se ingresa la cantidad

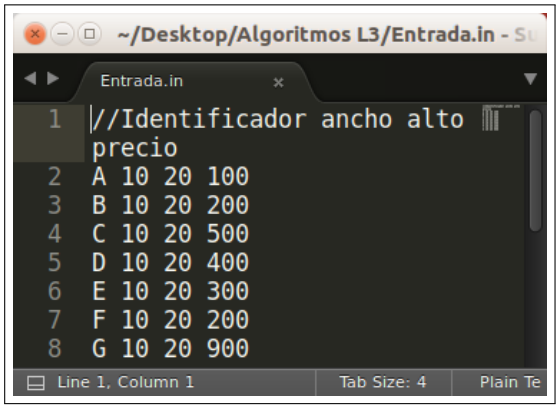

Fig. 1. Ejemplo de la publicidad ingresada en el archivo de entrada.

de páginas y las dimensiones de éstas por consola.

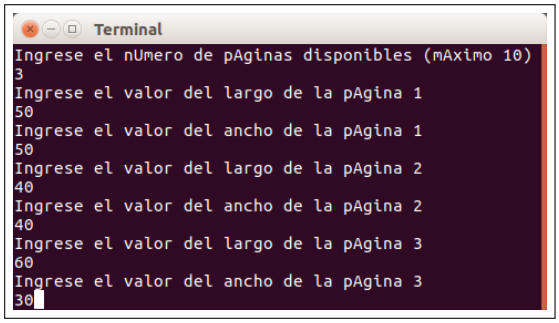

Fig. 2. Entrada por consola de las páginas disponibles para publicidad.

B. Salida

Al compilar el programa y ejecutarlo se debe generar un archivo de salida "Salida.out", este archivo posee de forma detallada, por cada página, los identificadores de los anuncios que posee la página, así como el valor de cada página obtenido a través de la suma de los valores asociados a las publicidades presentes en la página. Además, por consola, se imprime los resultados generales de las ganancias obtenidas y el número de publicidades insertadas.

3. MARCO TEÓRICO

A. Método Goloso

Goloso o también conocida como **Voraz o Devorador**. El método consiste en resolver un determinado problema, siguiendo un criterio previamente dado, consiste en intentar elegir

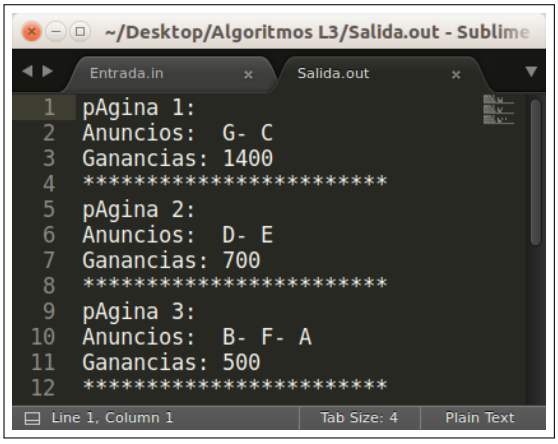

Fig. 3. Ejemplo de archivo de Salida.

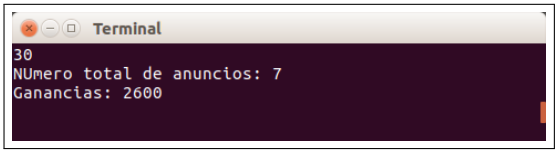

Fig. 4. Ejemplo de la salida por consola.

la solución óptima (según nuestro criterio) en cada paso con la esperanza de conseguir llegar a una solución óptima general. Es decir, Dado un conjunto finito de entradas **C**, el algoritmo devuelve un conjunto **S** de seleccionados que además cumple con los criterios de restricción del problema. A cada conjunto **S** se le llama prometedor, y según logre el objetivo de minimizar o maximizar (según el caso) diremos que **S** es solución óptima.

A.1. Funcionamiento

En cada paso el algoritmo escoge al mejor elemento **x**, el que mejor cumple con la condición escogida (menor tamaño, mayor valor, etc), que pertenezca a **C**, verificando que cumpla con las condiciones dadas por el problema, de modo de formar una solución factible.

- Analizar el tiempo de ejecución de algoritmos codiciosos generalmente es mucho más fácil que para otras técnicas (como Divide y conquistar). Para la técnica Divide y conquistar, no está claro si la técnica es rápida o lenta. Esto se debe a que en cada nivel de recursión el tamaño de se hace más pequeño y el número de sub-problemas aumenta.

- La solución obtenida con éste método no siempre asegura el óptimo.

4. DESCRIPCIÓN DE LA SOLUCIÓN

La solución del problema se puede enumerar de la siguiente manera.

1. Se lee el archivo de entrada: ***Entrada.in*** y se obtienen las **n** publicidades con sus respectivos atributos (largo, ancho, valor, identificador) y son guardados en **lista Publicidades**.

2. Se ordenan mediante ***Quicksort*** las publicidades según el criterio seleccionado, en este caso, el criterio de selección es el de mayor valor. Por lo tanto, se ordenan de forma descendiente respecto al valor de las mismas, siguiendo el

ejemplo de la figura 1 el resultado del ordenamiento es el de la figura 5. Este ordenamiento previo favorece disminuir el orden de complejidad de la solución.

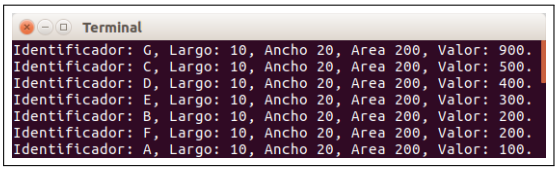

Fig. 5. Ejemplo de ordenamiento según criterio de valor.

3. Se obtienen la cantidad de páginas disponibles para la publicidad y las respectivas dimensiones de ancho y largo de ellas. Se verifica que la cantidad de páginas sea menor o igual a diez y las dimensiones mayor a 0.

4. Se crea lista ***usados*** del largo de la cantidad de publicidades inicializada con ceros en cada elemento, cada índice representa el número de la publicidad.

5. Se recorre cada una de las páginas disponibles para insertar publicidad y se busca la siguiente publicidad en la lista previamente ordenada que no esté ocupada y que cumpla con la restricción de que sus dimensiones sean menor o igual a las dimensiones disponibles de la página en cuestión. En caso de que la publicidad se pueda insertar se reduce el largo disponible de la página mencionada.

6. El proceso se repite hasta que en la página no quepa ninguna publicidad, cuando esto sucede se analiza la siguiente página, hasta que no queden **páginas** o **publicidades** disponibles.

7. Se escribe en el archivo de salida las publicidades presentes en cada una de las páginas insertadas en ella junto con el valor asociado a cada página, mientras que, en consola por consola se escribe los resultados generales como el número de página y el valor total obtenido por incluir todas las publicidades mencionadas en el archivo de saldia. Estos resultados se obtienen recorriendo cada una de las páginas, contando las publicidades y sumando el valor de éstas.

A. Diseño de la solución

B. TDA Utilizado: Listas

El primer problema que surge es poder manejar la cantidad dinámica tanto de páginas como de publicidad que obtendremos en la ejecución de nuestro programa. Así como generar la lista **utilizados** que representa (según el índice) si la publicidad ya fue utilizada o no, es por esto que se debe utilizar un TDA (Tipo de Dato Abstracto) adecuado en la implementación del código en C, cabe destacar que la implementación del código se respeta el estándar ANSI C y no se utiliza ninguna librería distinta a *stdio.h* y *stdlib.h* correspondientes al manejo de entrada-salida y a la librería de un correcto manejo de memoria respectivamente, ésta última cobra una gran importancia ya que es gracias a ella que es posible implementar una asignación dinámica de memoria ya que resulta imposible conocer a *priori* la cantidad de páginas que dispone el periódico para la publicidad, y éstas deben ser resueltas en tiempo de ejecución, por lo cual es necesario re-asignar memoria antes de agregar una nueva página a nuestra *listas utilizado* de esta forma se optimiza el uso de memoria. Se utiliza una lista doblemente enlazada implementada mediante cursor, la implementación del TDA se encuentra en el archivo de código fuente *2EnlazadasCursor.c* adjunto a este informe (también disponible en github) ya que éste TDA permite un manejo adecuado para los *strings*, los cuales son manejados como una lista de caracteres, donde cada elemento contenido en la lista representa a un carácter partiendo desde la posición 0 hasta la última posición correspondiente a la número k.

C. Lectura de datos de entrada

Los caracteres a utilizar se encuentran enlistados uno por cada línea como se puede apreciar en 1 lo cual facilita captar cada uno de los caracteres, en primer lugar se lee y se analiza si éste es una letra minúscula o si es un dígito, de no ser ninguno de los casos anteriormente mencionados el programa finaliza e imprime en el archivo de salida (Salida.out) que no se puede utilizar el alfabeto ingresado en el archivo de entrada, en caso de que sí sea un carácter permitido se comprueba que éste no se haya agregado previamente a la lista con los caracteres a utilizar, si ya se encuentra el programa finaliza e imprime el mismo mensaje anteriormente mencionado. Si cumple con las dos condiciones anteriormente mencionadas (que sea un carácter permitido y que éste no se encuentre ya en la lista de caracteres a utilizar) es agregado a la lista de caracteres a utilizar y procede a preguntar por el siguiente carácter en la lista.

D. Algoritmo Ordenamiento: Quicksort

Es un algoritmo de ordenamiento basado en la técnica de divide y vencerás, en la que en cada recursión, el problema se divide en subproblemas de menor tamaño y éstos se resuelven por separado (aplicando la misma técnica) donde finalmente se une la solución de los sub-problemas. es uno de los más rápidos conocidos. Su tiempo de ejecución promedio es de **O(n)=n*log(n)**. Una de las claves para que éste método sea exitoso y realmente presente mejora en la elección de pivote. La cual se selecciona según el siguiente algoritmo.

Algorithm 1. Pivote para Quicksort

```
1:  procedure PIVOTE(lista numeros, entero izquierdo,
    entero derecho)                          ▷ largolista = n
2:      pivote = izq
3:      valor_pivote = numeros[pivote]
4:      for i = izq + 1; i <= der; i++ do
5:          if numeros[i] > valor_pivote then
6:              pivote++
7:              aux=numeros[i]
8:              numeros[i]=numeros[pivote]
9:              numeros[pivote] = aux
10:     aux = numeros[izq]
11:     numeros[izq]=numeros[pivote]
12:     numeros[pivote]=aux
13:     return pivote
```

E. Algoritmo Goloso

El algoritmo Goloso implementado en este caso busca insertar las publicidades de mayor valor, es decir, les da prioridad. En caso de que la publicidad candidata no cumpla con los requisitos de inserción (tamaño) se repite el proceso con la siguiente imagen de la lista, como la lista es previamente ordenada con

Algorithm 2. Algoritmo Quicksort

```
1: procedure QUICKSORT(lista numeros, entero izquierdo, entero derecho)  ▷ largolista = n
2:     if izq<der then
3:         pivote = pivot(numeros,izq,der)
4:         Quicksort(numeros,izq,pivote-1)
5:         Quicksort(numeros,pivote+1,der)
```

Algorithm 3. Algoritmo Goloso Publicidad

```
1: procedure GOLOSO(lista Publicidades, lista Pginas)  ▷ numeroPublicidades = numPub, numeroPginas = numPag
2:     LargoActual=paginas[pag]
3:     pag=0
4:     ListaUsados  ▷ Lista de largo n, inicializada en 0
5:     while pag<numPag and todosUsados(Usados)==0 do
6:         pub=0
7:         largoActual=paginas[pag].largo
8:         while pag<numPag do
9:             if usados[pub]==0&publicidades[pub].largo<=largo Actual & publicidades[pub].ancho<=paginas[pag].ancho then
10:                Se inserta la publicidad en esa página.
11:                largoActual -= publicidades[pub].largo
12:            else
13:                pub+=1
14:            if pub>=numPub then
15:                pag+=1
16:        Siguiente página a analizar.
```

Quicksort el menor índice representa a la publicidad con mayor valor monetario.

El algoritmo recorre cada una de las páginas disponibles para publicidad y dentro de ellas pregunta si el mejor candidato (el de mayor valor y que no esté previamente ocupado) cumple con las condiciones para ser insertado, si lo es, se procede a:

1. Registrar el identificador de la publicidad en la página correspondiente y aumentar la cantidad de identificadores que posee.

2. Aumentar el valor total de la página en el valor de la publicidad.

3. Se debe actualizar el largo y ancho disponible para la página utilizada.

Orden(n) = n*log(n) + n*p.
Donde n es la cantidad de publicidades disponibles y p es la cantidad de páginas disponibles, en el peor de los casos el orden de complejidad de la solución es O(n)=n^2. Sin embargo, la lista de publicidades se encuentra ordenada, entonces, el orden de ejecución promedio es mucho menor ya que se reduce el tiempo en que el algoritmo debe buscar el mejor candidato.

5. ANÁLISIS DE RESULTADOS

En el siguiente capítulo se pondrá a prueba si efectivamente el código funciona, además de la robustez del código respecto a la respuesta a ***eventos inesperados***, como que existan una mayor cantidad de páginas disponibles para publicidad, de modo que no se inserte ninguna en ésta, además, de realizar pruebas en el caso contrario, es decir, que existan publicidades que sencillamente sean utilizadas. En cada subcapítulo se mostrarán las imágenes correspondientes a la consola luego de compilar y ejecutar el programa además de la entrada y salida de los archivos.

A. Camino Feliz

Se realiza una prueba con 7 publicidades, las cuales se presentan a continuación, además se dispone de 3 páginas para publicidad, todas de dimensiones de 50x50.

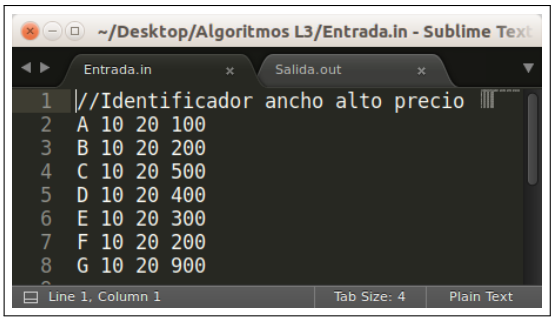

Fig. 6. Archivo de Entrada para el Camino feliz

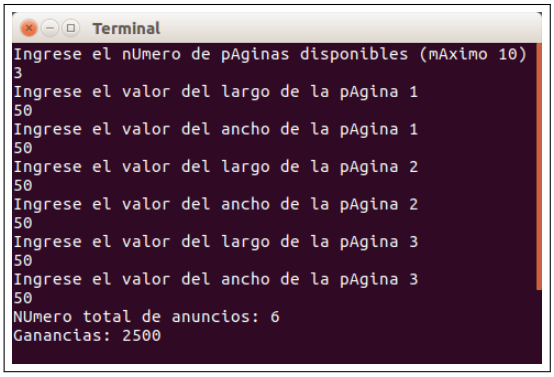

Fig. 7. Consola para el Camino Feliz

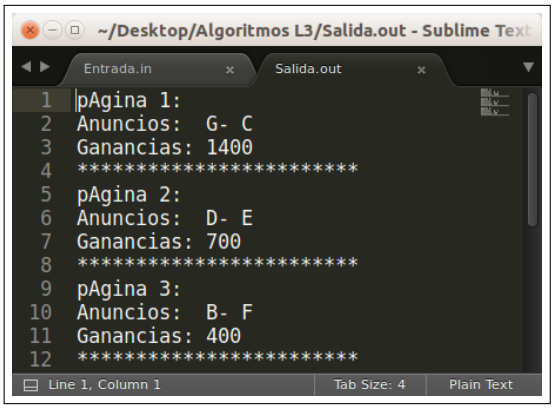

Fig. 8. Archivo de salida para el caso del Camino feliz

La prueba finaliza con éxito, en este caso se aprecia que 6 de las 7 publicidades disponibles se encuentra insertada en las páginas otorgadas para publicidad, en este caso se puede decir

que se ha encontrado la solución ***óptima***, dejando fuera la publicidad con identificador **A** con valor *100*. Además, mediante una simple inspección visual en la figura 8 de *Salida.out* y figura 7 se puede apreciar que la salida por pantalla de las ganancias total corresponde a la suma de las ganancias de las 3 páginas: 2500=1400+700+400.

B. Gran número de páginas.

En esta sub-sección se mostrará el comportamiento del programa cuando encuentra un mayor número de páginas de modo que queden páginas disponibles. La prueba finaliza con éxito, en

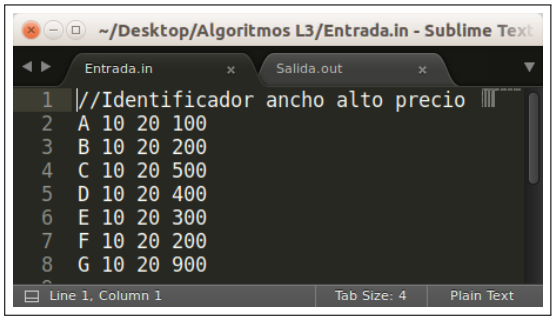

Fig. 9. Archivo de entrada caso Gran número de Páginas

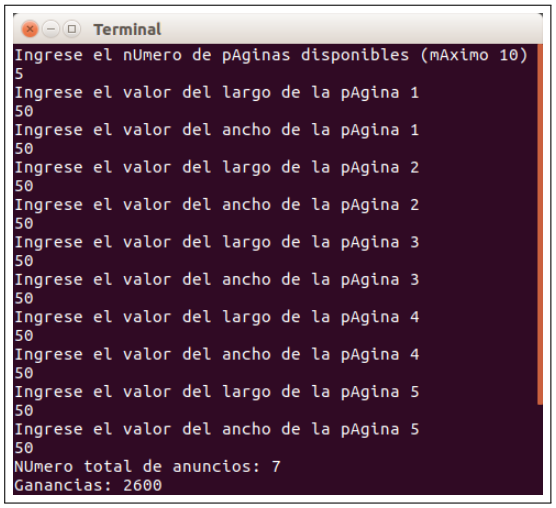

Fig. 10. Consola caso Gran número de Páginas

este caso se aprecia que todas y cada una de las publicidades disponibles se encuentra insertada en las páginas otorgadas para publicidad, en este caso se puede decir que se ha encontrado la solución ***óptima*** ya que no existe forma de mejorar la solución, pues, se encuentran todas las publicidades insertadas. Además, mediante una simple inspección visual en la figura 10 de *Salida.out* y figura 11 se puede apreciar que la salida por pantalla de las ganancias total corresponde a la suma de las ganancias de las 3 páginas: 2500=1400+700+400+100+0.

C. Publicidades quedarán fuera.

A continuación se probará la calidad de nuestra solución y verificar si se obtiene el óptimo, ya que en los 2 casos anteriores se ha cumplido. De ésta forma se disponen 4 publicidades, 1 grande que cubre toda la página y un resto de páginas que combinadas poseen un mayor valor y cubren de igual forma la página.

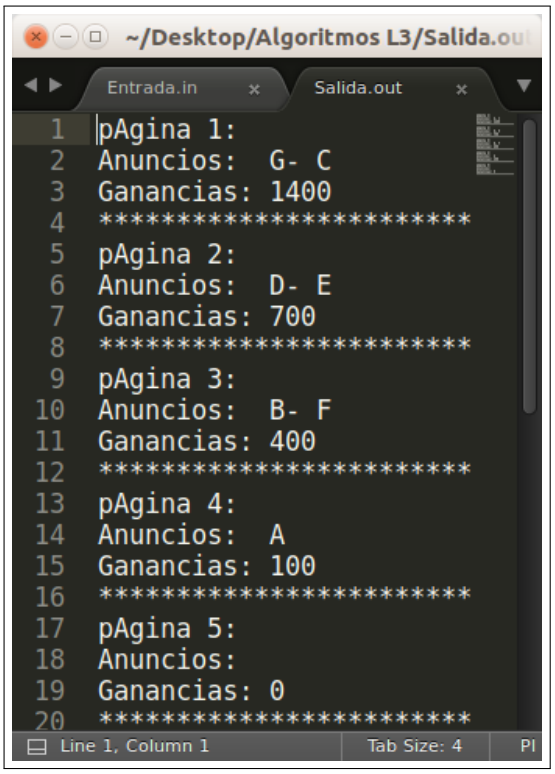

Fig. 11. Archivo de salida caso Gran número de Páginas

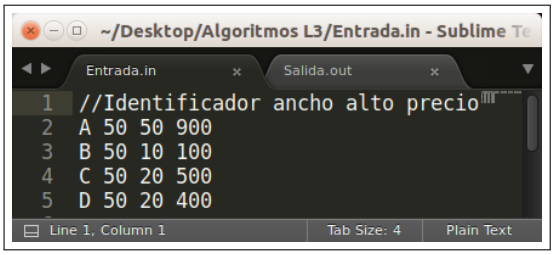

Fig. 12. Archivo de Entrada publicidades fuera.

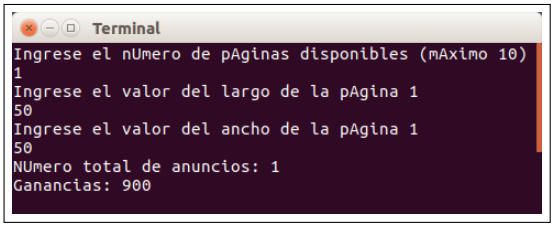

Fig. 13. Consola publicidades fuera.

Fig. 14. Archivo de salida publicidades fuera.

Esta prueba es clave para la experiencia, es posible notar que nuestro algoritmo no garantiza la solución **óptima** ya que obtenemos como resultado la inserción del elemento que mayor valor tiene "A" con $900 (nuestro criterio) pero además es el más grande, por lo tanto es la única que puede ser insertada, sin embargo existe una combinación que entrega un mayor valor que el obtenido y es "B-C-D" la cual entrega un valor equivalente a $1000, es por esto que se sugiere para una posterior implementación considerar el área, además del valor, para el criterio de selección de la publicidad.

6. TRAZA DE LA SOLUCIÓN

En esta sección se mostrará la traza de la solución generada por el archivo de entrada que se muestra en la figura 15, es necesario acotar el problema de forma que se logre apreciar el funcionamiento de nuestra solución. De acuerdo al algoritmo

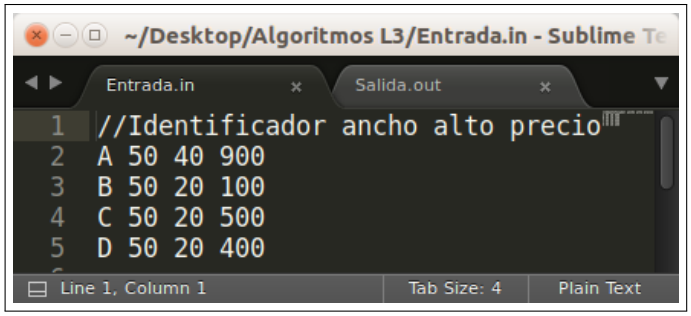

Fig. 15. Archivo de Entrada para análisis de traza.

3 (Goloso) en primer lugar se realiza un ordenamiento de las publicidades según el valor de forma descendiente A-C-D-B.

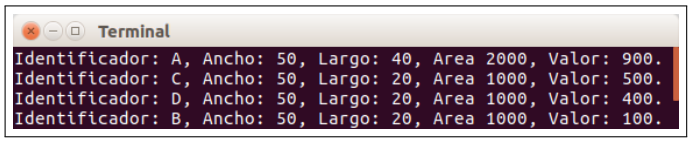

Fig. 16. Publicidades ordenadas.

Además, para realizar la prueba se ingresarán 2 páginas con dimensiones de 50x50 cms cada una.

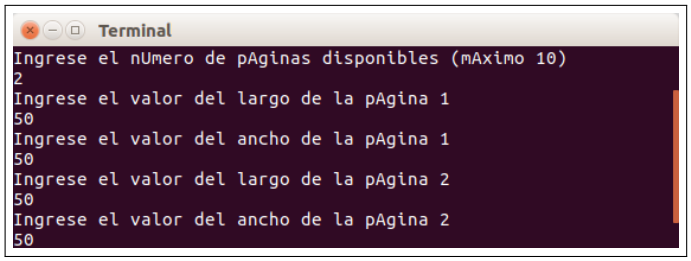

Fig. 17. Consola Traza.

De ésta forma la secuencia de inserción es la siguiente

1. Se intentará insertar la publicidad de mayor valor disponible **A**, cumple con las restricciones dadas de dimensiones. Tras esto quedan 10 cms disponibles de alto, por lo cual se intentará posicionar el resto de publicidades.
2. Se recorren cada una de las publicidades restantes, sin embargo, ninguna es posible insertarla.
3. Se procede a analizar una nueva página, partiendo por el nuevo mejor candidato, en este caso la publicidad C (ya que la publicidad A se encuentra ocupada), se verifica que C cumpla con las condiciones para ser insertada, se inserta y se actualizan los valores de restricción: El alto actual es de **30**.
4. Se analiza el siguiente mejor candidato D, el cual cumple con las condiciones ya que su largo es de 20 y el el largo disponible es de 30 (20<30) por lo tanto es insertado y se actualizan los valores de restricción alto actual: **10**.
5. Se analiza la última publicidad disponible **B**, la cual no cumple con la restricción dada, como la publicidad debe ser publicada de forma íntegra se deja de lado.

Finalmente la solución dada por nuestro algoritmo es:

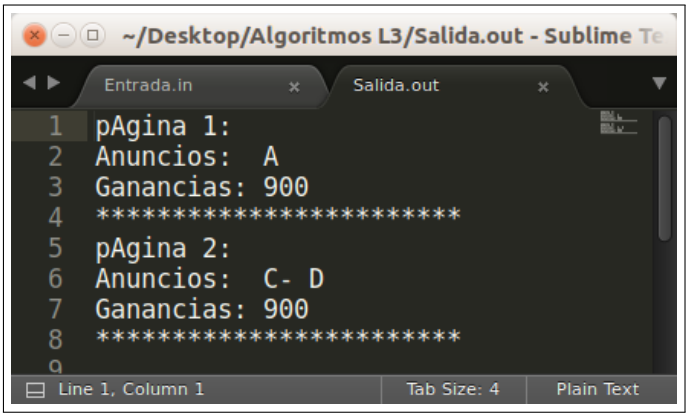

Fig. 18. Salida Traza.

En este caso la solución dada es la óptima, pero como se vio en la prueba C, el algoritmo no asegura el óptimo.

7. CONCLUSIONES

Durante la experiencia notamos la importancia que poseen los algoritmos de ordenamiento y el momento de la solución en la que éstos son aplicados ya que cobra importancia en la reducción del orden de complejidad de nuestra solución, pues, si los ordenamos según nuestro criterio previamente a recorrerlos resulta más rápido encontrar el próximo mejor candidato a ser analizado, por el contrario, si las publicidades no son ordenadas, en cada iteración se debe buscar el mejor candidato lo cual resulta más costoso computacionalmente ya que cada búsqueda es de orden O(n)=n lo cual perjudica significativamente la calidad de nuestra solución.

Cabe destacar que dado nuestro criterio de selección del mejor candidato, nuestra solución presentada de la versión del **algoritmo voraz** con el valor más alto no asegura el óptimo en todos los casos, por lo cual se recomienda que para la posterior implementación y mejora del algoritmo, al momento de seleccionar el criterio para nuestro algoritmo se consideren las dimensiones además del valor. Se sugiere utilizar el criterio de valor/área donde el área está en metros cuadrados dado que resulta del producto entre el largo y el ancho de la publicidad. A diferencia de la implementación de ***Enumeración Explícita*** y

Backtracking es que éstos dos algoritmos aseguran el óptimo, sin embargo, sus tiempos de ejecución son mucho mayores, respecto a la implementación de la solución bajo el algoritmo voraz el cual en el caso promedio posee un orden $O(n) = n * log(n)$ y en el peor de los casos $O(n) = n^2$.

REFERENCES

1. P. Gutiérrez, "¿Cuanto tardaría un hacker en reventar nuestra contraseña?" Opt. Express **22**, (2012).
2. J. Mañas, "Análisis de Algoritmos: Complejidad" Departamento de Ingeniería en Sistemas Telemáticos **1**, (1997).
3. M. Villanueva, "Algoritmos: Teoría y aplicaciones" Departamento de Ingeniería Informática, Universidad de Santiago de Chile.**42-43**, (2012).