Laboratorio 3 - Goloso

JUAN RETAMALES

Profesora: Mónica Villanueva

Ayudante: Patricio Vargas

Compiled June 4, 2017

En este documento se da a conocer conceptos básicos del algoritmo Goloso y el lenguaje de programación C, ya que, mediante un problema planteado se pretende comprender el funcionamiento del algoritmo además de mostrar las conclusiones realizadas.

1. INTRODUCCIÓN

En el presente documento se pretende mostrar los conceptos como Método goloso, el cual mediante el objetivo que sería comprender su uso, se utilizara el lenguaje de programación C y analizarlo para el aprendizaje de estos de manera más práctica que conceptual.

2. DESCRIPCIÓN DEL PROBLEMA

Se debe resolver el enunciado de la subsección A, antes del plazo de entrega informado en la subsección D.

A. Enunciado

El banco regional de Concepción tiene problemas de atención al público y sus clientes se quejan constantemente de la lentitud con que avanzan las filas de las cajas. Al parecer los cajeros no son lo suficientemente rápidos para calcular el vuelto que deben entregar después de un pago y además no siempre lo hacen correctamente, es decir, muchas veces devuelven una cantidad errónea de dinero, lo cual provoca peleas y retrasos en la atención. Para mitigar esta situación, el banco lo contrata a usted para que diseñe e implemente un programa que le indique a los cajeros el vuelto que deben entre- gar y la cantidad mínima de efectivo para hacerlo. El sistema monetario con el que se rige la sucursal es el chileno, por lo tanto, se deben considerar los billetes de $20.000 , $10.000, $5.000, $2.000 y $1.000 pesos, además de las monedas de $500, $100, $50, $10, $5 y $1 peso. La implementación de la solución debe ser en el **Lenguaje de Programación C** , y utilizando la técnica de resolución de problemas **Goloso** [1].

B. Entrada

En un archivo de texto titulado entrada.in se deben listar según tipo las cantidades de todos los billetes y todas las monedas que se mantienen en una caja al principio del turno. Además por línea de comando, el programa debe solicitar el monto a cancelar por cada cliente y el dinero que estos entregan como pago.

C. Salida

Considerando el total a cancelar y el dinero pagado por cada cliente, el programa debe mostrar por consola el vuelto a entregar y la cantidad mínima de efectivo con el que se puede realizar dicha devolución a partir del cambio que hay disponible en la caja. En un archivo de texto titulado salida.out se debe mantener un registro de el total en caja después de cualquier transacción ejercida durante el día, esto para que los cajeros tengan una referencia al momento de balancear sus cuentas al final del turno. Recuerde que cuando un cliente hace un pago, se debe sumar ese monto al total de dinero en la caja, y que la cantidad de efectivo entregado también es la mínima posible, por ejemplo, si cancela $35.000 pesos, se asume que entregó 1 billete de $20.000, otro de $10.000 y uno de $5.000 pesos. La solución debe funcionar para una cantidad variable de clientes. Si la totalidad de sencillo que el cajero posee en caja no alcanza para dar vuelto a una transacción, se debe indicar por pantalla y terminar la ejecución del programa.

D. Entrega

Domingo 28 de mayo del 2017 hasta las 23:55, se descontará 1 punto por cada hora de retraso, al no entregar se reprueba el laboratorio.

3. MARCO TEORICO

Se pretende mediante el enunciado anterior, comprender sobre el funcionamiento del algoritmo goloso usando el lenguaje de programación C.

A. Método Goloso

Hay muchos problemas en los que se pretende obtener un subconjunto de n elementos que satisfaga ciertas restricciones y que optimice alguna medida. Un problema de esta clase tiene al menos una solución. Puede haber varias soluciones óptimas, en cuyo caso no importa cual se elija. Por ejemplo, el problema de encontrar un subconjunto de los arcos de un grafo que formen el

Table 1. Ejemplo de entrada y salida

Entrada.in	**Salida.out**
20 20000	Inicial: 1315460
40 10000	Pago 1: 1333960
50 5000	.....
40 2000	
100 1000	
45 500	
1000 100	
750 50	
500 10	
90 5	
10 1	
Consola	**Consola**
Total a pagar: 18500	Vuelto: 1500
Cliente entrega: 20000	1 billete de 1000
....	1 moneda de 500
	....

camino más corto entre dos vértices dados del grafo. Otro ejemplo se da cuando, dados unos archivos almacenados en una cinta de recorrido secuencial, se quiere encontrar un modo de almacenarlos que garantice que el tiempo promedio de recuperación de un archivo cualquiera sea mínimo. A menudo, el problema incluye restricciones adicionales que limitan el número posible de soluciones.

Normalmente, estos problemas no se intentan resolver "de golpe", encontrando de un paso una solución óptima. Es más frecuente que el subconjunto de la solución se vaya formando paso a paso, analizando durante cada etapa el elemento que conviene añadir a la solución parcial ya existente.

Nunca se retiran elementos del conjunto construido. Siempre se obtiene una solución pero –en general- no se garantiza que sea óptima. Solo en algunos problemas particulares el método goloso obtiene siempre una solución óptima.

[2].

B. Lenguaje de programación C

El lenguaje C es un lenguaje estructurado, en el mismo sentido que lo son otros lenguajes de programación tales como el lenguaje Pascal, el Ada o el Modula-2, pero no es estructurado por bloques, o sea, no es posible declarar subrutinas (pequeños trozos de programa) dentro de otras subrutinas, a diferencia de como sucede con otros lenguajes estructurados tales como el Pascal. Además, el lenguaje C no es rígido en la comprobación de tipos de datos, permitiendo fácilmente la conversión entre diferentes tipos de datos y la asignación entre tipos de datos diferentes [3].

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int main(int argc, char** argv) {
    ...
    return (EXIT_SUCCESS);
}
```

C. Análisis de Algoritmos: Complejidad

A efectos prácticos o ingenieriles, nos deben preocupar los recursos físicos necesarios para que un programa se ejecute. Aunque puede haber muchos parametros, los mas usuales son el tiempo de ejecución y la cantidad de memoria (espacio). Ocurre con frecuencia que ambos parametros están fijados por otras razones y se plantea la pregunta inversa: ¿cual es el tamano del mayor problema que puedo resolver en T segundos y/o con M bytes de memoria? En lo que sigue nos centraremos casi siempre en el parametro tiempo de ejecución, si bien las ideas desarrolladas son fácilmente aplicables a otro tipo de recursos.

Para cada problema determinaremos un medida N de su tamaño (por número de datos) e intentaremos hallar respuestas en función de dicho N. El concepto exacto que mide N depende de la naturaleza del problema. Así, para un vector se suele utizar como N su longitud; para una matriz, el número de elementos que la componen; para un grafo, puede ser el número de nodos (a veces es mas importante considerar el número de arcos, dependiendo del tipo de problema a resolver); en un fichero se suele usar el número de registros, etc. Es imposible dar una regla general, pues cada problema tiene su propia lógica de coste [4].

4. DESCRIPCIÓN DE LA SOLUCIÓN

A. Etorno

El entorno de trabajo se realiza en el editor Netbeans IDE 8.2 en su versión de 64 bits el cual funciona con el software Cygwin que proporciona un compilador del lenguaje C ademáss de otras herramientas usadas por el editor. Todas las pruebas se realizan en un equipo AMD de 4 núcleos de 2.9 Ghz con 16Gb de memoria ram el cual tiene instalado el sistema operativo Windows 7 en su versión Ultimate de 64bits y más de 500Gb de espacio libre en el disco duro principal.

De los archivos del laboratorio se destacan los siguientes:

-**main.c**: Guarda el código principal del algoritmo así como su función principal (main).

-**entrada.in**: Es el utilizado para que el software pueda leer los números para generar la matriz inicial con el dinero, cabe destacar que para no incidir a un error se utilizara el archivo en formato UTF sin BOM.

-**salida.out**: Es el archivo el cual el software guarda los cambio del dinero de la matriz.

-**funciones.h**: Contiene las declaraciones y una breve explicación de las funciones creadas usadas.

-**funciones.c**: Este archivo contiene el código de las funciones creadas usadas declaradas en el archivo “funciones.h” las cual se encargan de ver cuando dinero hay en la matriz, y leer desde el teclado un número y validarlo.

B. Lógica

Para generar la solución, primero debemos leer el archivo entrada.in, para ello lo lee como textos e intenta transformarlo a números, de ser posible se considera válido. Luego recorre un arreglo previamente inicializado llamado “entrada” con todos los “valores de moneda” previamente fijados según el software para agregarlos a la cantidad cero que tiene al inicio. Además entrada.in debe estar en formato “{cantidad de monedas} {valor moneda} ” y cada uno de estos separados por saltos de línea

Table 2. Tabla de tiempo y orden

Función	Tiempo	Orden
dinero	n+1	n
leer	5	c

como se muestra en el ejemplo, luego de validar la transformación de los números los asigna en pares como "cantidad de monedas" y "valor de moneda" respectivamente. Esos dos valores de cada casilla del arreglo corresponderán a nuestra caja, separada por cantidad y valor de monedas.

Después crea o reemplaza de existir, el archivo salida.out el cual guarda el dinero actual de la caja después de todas sus transacciones.

Finalmente inicia un ciclo para realizar las operaciones que ingresa un usuario por teclado, primero pidiendo el "Total a pagar" y luego "Cliente entrega". Con estos valores genera el valor del vuelto a entregar, de tener un vuelto recorre el arreglo entrada de los valores más altos a los valores más bajos, revisa si la aproximación por corte de la división de este valor moneda sea mayor a cero y contenga ese valor en el arreglo "entrada", de tenerla o no, la guarda en una nueva variable llamada "operación" con cantidad en negativo indicando que se restara en caso de suma. Si logra generar el vuelto despliega el mensaje del vuelto a entregar y agrega a la variable operación, el dinero del pago. Además, realiza las operaciones con entrada sumando a la "cantidad de monedas". De no tener vuelto o si el cliente no entrega suficiente dinero despliega sus respectivos mensajes. Para detener el ciclo de operaciones, se debe ingresar "0" o algún texto, ya que, para complementar, cada ingreso de valores está debidamente validado.

Al terminar de ejecutar el programa, despliega el tiempo transcurrido en el software según el reloj del ordenador.

C. Complejidad y Orden

Para calcular el orden empezamos calculando el orden de funciones del archivo "funciones.c" como se ve en la siguiente tabla:

Finalmente luego de hacer los cálculos correspondientes para calcular tiempo, obtenemos que su valor seria Tiempo$((104n^2 + 8n + 25)$ y teniendo un orden final de $O(n^2)$

D. Analizando algoritmo

Para analizar el algoritmo se realizaran y responderán las siguientes preguntas:

¿Se detiene? Si, cuando el archivo.in no está correctamente ingresados o tiene problemas para leerlo. Y cuando no puede generar algún vuelto o, cuando ingresa pago o vuelto con valores errores o "0".

¿Tiempo de ejecución y orden? Como se puede apreciar en la subsección anterior, el tiempo seria $(132n^2 + 99n + 24)$ y un orden de $(132n^2)$.

¿Se puede mejorar? Se podría mejorar el tiempo, mas no el orden.

¿Otro método podría ser mejor? No, ya que algoritmo goloso para este tipo de ejercicio siempre entrega el óptimo.

5. TRAZA DE LA SOLUCIÓN

Primero lee el archivo entrada.in y llena el arreglo "entrada" como se describió en la subsección **B. Lógica**, y crea el archivo salida.out, luego entra al bucle que se muestra superficialmente a continuación.

```
while(pago!=0 || entrega!=0)
{
        int operacion[12][2]={{0, 20000},{0, 10000},{0,
             5000},{0, 2000},{0, 1000},{0, 500},{0,
             100},{0, 50},{0, 10},{0, 5},{0, 1}};
        printf("\n(Introduce cero para terminar)");
        pago=leer("\nTotal a pagar: ");
        if(pago!=0)
        {
                printf("\n(Introduce cero para terminar)");
                entrega=leer("\nTotal a pagar: ");
        }
        else
        {
                entrega=0;
        }
        vuelto=entrega-pago;
        if(vuelto>0)
        {
                int temp=0;
                actual=vuelto;/*Dinero actual que falta
                     para hacer el vuelto, no se trabaja
                     con vuelto por que se vuelve a
                     utilizar.*/
                for(int i=0;actual!=0 && i<largo;i++)
                {
                        /* Reviso si dar vuelto de esa
                             moneda es posible */
                        /*reviso si tengo suficientes de
                             esas monedas para dar vuelto
                             */
                        /*Agrego a operacion para despues
                             quitar a caja el total si
                             finaliza exitosamente.*/
                        /*Actualizo el vuelto a dar.*/
                }
                if(actual==0)
                {
                        /*Asigno vuelto a actual para
                             agregar el pago a caja*/
                        /*Agrego a operacion el dinero
                             recibido*/
                        for(int i=0;i<largo;i++)
                        {
                                /*Por goloso transformo
                                     dinero recibido a
                                     cantidad de dinero y
                                     dinero*/
                        }
                        /*Muestro el mejor vuelto*/
                        for(int i=0;i<largo;i++)
                        {
                                /* Muestra el vuelto a dar
                                     en consola */
                                /*Realiza los cambios en el
                                     arreglo*/
                        }
                        cantPagos++;
                        fp = fopen ( "salida.out", "a" );
                        int din = dinero(entrada,largo);
                        fprintf(fp, "Pago %d: %d \n",
                             cantPagos, din);
                        fclose(fp);
                }
                else
                {
                        printf("\n No puede generar un
                             vuelto con el saldo disponible
                             , cerrando aplicacion , falto
                             [%d]}",actual);
                        break;
                }
        }
        else
        {
                ...
        }
}
```

Ingreso por consola el pago, si no es válido o es cero termina la aplicación. En caso contrario como en este ejemplo se usara

"8960", Luego te pide el dinero del cliente entregado, si no es válido o es cero termina la aplicación. En caso contrario calcula el vuelto y genera la mejor combinación en términos de que ocupe menos dinero disponible para desplegarlos por consola. Para este ejemplo se ingresó "10000", y por consola muestra los siguientes mensajes.

```
Iniciando...ok
Entrada.in...
(Introduce cero para terminar)
Total a pagar: 8960

(Introduce cero para terminar)
Cliente entrega: 10000

 Vuelto: 1040
 1 billete de 1000
 4 moneda de 10
(Introduce cero para terminar)
Total a pagar:
```

A la vez, el software actualiza entrada y agrega el registro en salida.out con el pago con su número, y el valor en caja que sería la cantidad de dinero del arreglo "entrada".

```
Inicial: 1395460
Pago 1: 1404420
```

Finalmente, para terminar se puede ingresar un cero en el "Total a pagar" o en "Cliente entrega", en este ejemplo usaremos la primera opción como se muestra a continuación.

```
(Introduce cero para terminar)
Total a pagar: 0
Finalizando aplicacion
Tiempo transcurrido: 3 segundos
```

6. CONCLUSIÓN

El método Goloso es un método que se empleó para realizar el mejor vuelto de dinero según el problema propuesto (el óptimo), y consume mucho menos recursos que otros métodos como Backtracking y Fuerza Bruta y allí radica la gran importancia de intentar usarlo especialmente en este tipo de problemas. Y en la actividad propuesta se resolvió un problema en el lenguaje C, para demostrar de manera práctica que se logró cumplir los objetivos planteados al inicio del documento.

REFERENCES

1. M. Villanueva, “Algoritmos avanzados 1.2017 - laboratorio ii,” (2017).
2. M. Villanueva, “Algoritmos: Teoria y aplicaciones,” (2002).
3. E. V. B. Esteban, “Lenguaje c,” (2009).
4. J. A. Mañas, “Análisis de algoritmos: Complejidad,” (1997).

FULL REFERENCES

1. M. Villanueva, “Algoritmos avanzados 1.2017 - laboratorio ii,” (2017).
2. M. Villanueva, “Algoritmos: Teoria y aplicaciones,” (2002).
3. E. V. B. Esteban, “Lenguaje c,” (2009).
4. J. A. Mañas, “Análisis de algoritmos: Complejidad,” (1997).