UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE CIENCIAS

SISTEMA INTERACTIVO PARA PROGRAMACION ENTERA UTILIZANDO METODOS DE CORTADURAS

T E S I S

QUE PARA OBTENER EL TITULO DE:

A C T U A R I O

P R E S E N T A :

ENRIQUE MORALES SILVA

México, D. F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

INTRODUCCION

La Programación Entera es uno de los campos de la Investigación de Operaciones que cada vez toma una mayor importancia en la práctica; a pesar de no ser tan valorada y estudiada como la Programación Lineal, resulta de gran interés en aquellos casos donde la Programación Lineal sólo sirve para dar una aproximación al problema; además, en algunos casos, el óptimo continuo llega a estar "muy lejos" del óptimo entero lo cual provoca que los valores continuos obtenidos para las variables de desición pierdan su significado al redondearse a valores discretos. Además de los casos anteriores, la Programación Entera es de gran ayuda al tratar con funciones no lineales separables y en quellos problemas donde se manejan variables dicotómicas (1,0).

Si bien existe una gran cantidad de paquetes computacionales sobre la Programación Lineal, no existen tantos sobre la Programación Entera, además de que en ambos casos, a menudo se trata de sistemas bastante especializados y complejos. En el presente trabajo se implementa un sistema con varias características como son las de implementar un "laboratorio" para poder experimentar con los problemas durante su desarrollo, ya sea en el algoritmo mismo o simplemente observando paso a paso como alcanzan su óptimo (en caso de existir); también es posible el resolver problemas sin necesidad de enterarse qué es lo que hace el programa, esto cuando se sólo se necesiten resultados o cuando no se tenga una buena idea de la forma en que funcionan los algoritmos; esto se logra aprovechando los valores estándar que adopta el sistema para poder resolver un problema simplemente editándolo y escogiendo la opción de "Resolver" lo cual también permite encontrar óptimos continuos. Todo lo anterior se integra en un sistema que trata ser de fácil manejo al usarse a base de

menús donde se incluyen las diversas opciones del sistema; para que éste sea lo más autosuficiente posible, cuenta con un menú de ayuda y un programa de presentación el cual introduce al usuario de manera breve a lo que es el sistema en sí y a la Programación Entera con especial atención a los métodos que implementa el sistema.

El sistema implementa dos algoritmos de Cortaduras; esta clase de algoritmos reciben este nombre por generar restricciones adicionales a las originales con el fin de "cercar" al óptimo; este tipo de algoritmos es de fácil implementación debido a que no consumen gran cantidad de memoria y no necesitan de estructuras computacionales para su manejo; con el fin de brindar un sistema completo, se han implementado diversos algoritmos y métodos de la Programación Lineal y si bien el alcance del sistema no es grande en cuanto al manejo de un gran número de variables y restricciones, se pretende sea un punto de partida para futuras mejoras a este tipo de sistemas.

Además de los objetivos referentes a las características del sistema, existe un objetivo de gran importancia que es el de proporcionar una herramienta extra en la enseñanza de esta materia, ya que en la práctica es imposible aplicar cualquiera de estos métodos sin la ayuda de una computadora; por lo mismo, el familiarizar al estudiante con este tipo de apoyos didácticos, brindará una mejor formación académica al poder complementar lo aprendido en el salón de clase.

Otro de los objetivos primordiales del sistema respecto al punto anterior, es el de crear un ambiente agradable para el usuario donde no sea necesario el tener conocimiento alguno de programación y donde pueda intervenir lo más posible dentro del flujo del sistema, ya sea editando, modificando o resolviendo una gran diversidad de problemas con la mayor libertad posible.

Cabe comentar que para la total comprensión del presente trabajo, es necesario tener conocimientos medios sobre Programación Lineal y computación.

CAPITULO 1

INTRODUCCION A LA PROGRAMACION ENTERA

1.1 QUE ES LA PROGRAMACION ENTERA

Cualquier problema de desición (con una función objetivo a optimizar) en el cual las variables de desición deben tener valores no fraccionales o discretos, puede clasificarse como un problema de Programación Entera. En general, estos pueden ser restringidos o no y las funciones objetivo y restricciones pueden ser lineales o no; en sentido estricto, todo problema de Programación Entera debe ser visto como un problema no lineal, ya que sus funciones se definen sólo en valores discretos; sin embargo, este detalle técnico puede ser pasado por alto para establecer una clasificación más significativa para el punto de vista de los métodos de resolución; de lo anterior, un problema se considera como lineal si una vez que se elimina la restricción de que las variables tengan valores enteros, las funciones resultantes sean estrictamente lineales; en caso contrario, se considerarán como no lineales.

La Programación Entera en el sentido señalado anteriormente, no es nueva, pero hasta fines de los 40's cuando la Investigación de Operaciones comenzó a tomar un gran impulso, la mayoría de los problemas estudiados fueron de interés meramente matemático.

La importancia de esta disciplina para resolver problemas prácticos, fué el resultado de los grandes avances dentro de la Investigación de Operaciones y en particular de la Programación Lineal; fué hasta entonces que se reconoció la necesidad de resolver modelos de programación donde algunas o todas las variables de desición deberían asumir valores enteros. Aunque varios problemas en algunas áreas importantes se formularon como

problemas enteros, fué en 1958 que Gomory desarrolló el primer algoritmo de Programación Entera *finito* para resolver este tipo de problemas; a partir de entonces se han desarrollado diferentes algoritmos especializados.

1.2 DEFINICION MATEMATICA DEL PROBLEMA

El problema puede definirse de forma general como sigue:

Maximizar (o Minimizar) $Z = \sigma_0(X_1, X_2, \ldots, X_n)$

Sujeto a:

$$\sigma_i(X_1, X_2, \ldots, X_n) \; (\leq, =, \geq) \; b_i \qquad i \in \overline{1,m}$$

$$x_j \geq 0 \qquad j \in \overline{1,n}$$

$$x_j \in \mathbb{Z} \qquad j \in I \subseteq N$$

Si $I = N$, es decir, todas las variables x_j estan restringidas a tomar valores enteros, el problema se llama de Programación Entera pura (o Programación Entera); en caso de que $I \subset N$, se tiene un problema de Programación Entera Mixta (o Programación Mixta) donde algunas de las variables podrán aceptar valores fraccionales; la mayoría de los algoritmos desarrollados, incluyendo los que aquí se tratan, consideran a las funciones σ_i como lineales; así se puede plantear en forma matricial un problema de este tipo:

$$\text{Max (o Min)} \; Z = C\,x$$

sujeto a:

$$A\,x = b$$
$$x \geq 0$$
$$x \in \mathbb{Z}$$

Donde A es una matriz de $m \times n$.

Como se puede ver, al no existir la condición de que las variables tengan valores discretos, el problema se convierte en un problema común de Programación Lineal, por lo que los métodos de Programación Entera buscan determinar el punto óptimo de entre todos los puntos discretos incluidos dentro del espacio de solución.

Aparentemente, puede suponerse que el aumentar la condición de que las variables sean enteras no sea un gran problema porque de hecho, el espacio de soluciones está mejor definido ahora, ya que no se necesitará buscar de entre un número infinito de puntos como en el caso continuo (supóngase por simplicidad un área acotada); desafortunadamente, a pesar de que el espacio de soluciones está mejor definido (en el sentido anteriormente expuesto), en muchas ocasiones resulta ser muy complicado, esto porque a menudo la condición de tener valores discretos destruye las "buenas" propiedades del espacio de soluciones; un ejemplo común es el problema entero *lineal* donde al quitar esta condición, el espacio de soluciones deja de ser convexa que es la propiedad que aprovecha el método Simplex en los problemas de Programación Lineal

1.3 APLICACIONES DE LA PROGRAMACION ENTERA

La Programación Entera tiene aplicaciones de gran importancia; en algunos casos la condición de variables enteras surge como una restricción natural y no como una restricción extra; esto se debe a la condición de *unimodularidad* de la matriz *A*; uno de estos problemas de importancia es el problema del Transporte donde se trata de minimizar el costo de llevar ciertos artículos de un lugar a otro cumpliendo con condiciones de disponibilidad y demanda; este problema se plantea de la forma siguiente:

$$\text{Min } Z = \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ij} X_{ij}$$

s.c.[1]

$$\sum_{i=1}^{m} X_{ij} \leq a_i$$

$$\sum_{j=1}^{n} X_{ij} \geq d_j$$

$$X_{ij} \geq 0$$

donde se tienen:

m orígenes ⟶ a_i disponibilidad

n destinos ⟶ d_j demanda

C_{ij} : Costo unitario de transporte de i a j.

Este problema es de gran importancia pués es el caso general de muchos otros como lo es el problema de asignación donde las variables son binarias:

Suponiendo que se tienen m individuos y n trabajos

$$X_{ij} = \begin{cases} 1 \text{ si se asigna el trabajo } i \text{ al individuo } j \\ 0 \text{ en otro caso} \end{cases}$$

Además este problema tiene la particularidad de que $m = n$ por lo que su planteamiento será:

$$\text{Min } Z = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij}$$

s.c.

---

[1] s.c. = " sujeto a las condiciones "

$$\sum_{i=1}^{n} X_{ij} = 1, \qquad j \in \overline{1,n}$$

$$\sum_{j=1}^{n} X_{ij} = 1, \qquad i \in \overline{1,n}$$

$$X_{ij} \geq 0$$

Si bien puede parecer que este problema no es muy complejo, en realidad puede llegar a tener dimensiones impresionantes; por ejemplo:

n	Posibles asignaciones
4	4! = 24
10	10! = 3,628,800
20	20! = 2,342,900,000,000,000,000

Existen otros problemas donde la condición sobre las variables enteras ya no es implícita; dentro de estos problemas uno de los de mayor importancia y aplicación es el del "Agente Viajero" donde se desea cubrir cierta ruta pasando por todos los puntos de ésta minimizando el recorrido; este problema se plantea de la forma siguiente:

$$X_{ij} = \begin{cases} 1 \text{ si considera el trayecto de } i \text{ a } j \\ 0 \text{ en otro caso} \end{cases}$$

$$\text{Min} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} X_{ij}$$

s.c.

$$\sum_{i=1}^{n} X_{ij} = 1$$

(Para asegurar que se salga de todas las ciudades)

$$\sum_{j=1}^{n} X_{ij} = 1$$

(Para asegurar que se llegue a todas las ciudades)

$$\sum_{\substack{i \in Q \\ j \notin Q}} X_{ij} \geq 1$$

(Para evitar ciclos)

$$X_{ij} \geq 0, \qquad X_{ij} \in Z$$

donde d_{ij} representa la distancia de la ciudad i a la j, cuya suma global desea minimizarse.

Otro problema de importancia es la *Secuenciación de Actividades* donde se desea minimizar el tiempo que se tardará en realizar una serie de actividades basándose en la prioridad de una sobre otra; el modelo para este problema es análogo al anterior.

El conocido *problema de la Mochila* también entra dentro de los problemas de Programación Entera; aquí se desea maximizar la utilidad de ciertos artículos tomando en cuenta la capacidad de almacenamiento; el planteamiento del problema es el siguiente:

Dados n artículos disponibles:

$$\text{Max} \quad Z = \sum_{j=1}^{n} U_j X_j$$

s.c.

$$\sum_{j=1}^{n} V_j X_j \leq V$$

$$X_j \geq 0 \quad , \quad X_j \in Z$$

donde:

V : Capacidad

X_j : Cantidad de artículos j que se deben llevar, $j \in \overline{1,n}$

U_j : Utilidad del artículo j

V_j : Volúmen del artículo j

Otros dos problemas bastante parecidos entre sí, son el problema de *Recubrimiento Mínimo* y el de *Acoplamiento Máximo*; ambos se pueden plantear en términos de gráficas no orientadas donde se desea cubrir ciertos arcos en el primer caso, y unir ciertos nodos en el segundo; el planteamiento de estos problemas es el siguiente:

(Problema de Recubrimiento Mínimo):

$$\text{Min} \quad Z = 1_n \, X$$

s.c.

$$Ax \geq 1^m$$

$$x \geq 0 \quad , \quad x \in Z^n$$

(Problema de Acoplamiento Máximo):

$$\text{Max} \quad Z = 1_n \, X$$

s.c.

$$Ax \leq 1^m$$

$$x \geq 0 \quad , \quad x \in Z^n$$

Estos son entre otras, algunas de las aplicaciones de la Programación Entera; existen diversas aplicaciones más especializadas donde se pide la dicotomía de variables (para aproximar funciones separables, por ejemplo) lo cual es aprovechado por otros algoritmos de solución como lo es el algoritmo de Balas para problemas con variables 0,1 o bien métodos de naturaleza distinta como el Método de la casilla Noroeste para el problema de Transporte, etc. Los algoritmos implementados en el sistema son algoritmos de Cortaduras los cuales veremos más adelante y que pueden ser utilizados para resolver cualquiera de los problemas anteriores.

1.4 METODOS DE PROGRAMACION ENTERA

Los métodos de Programación Entera a menudo se clasifican en dos grandes grupos: (1) Métodos de Búsqueda y (2) Métodos de Cortaduras; el primer grupo surge por el hecho de que el espacio de solución Entero puede ser considerado como un número *finito* de puntos[2]; en su forma más sencilla, los métodos de búsqueda buscan enumerar "todos" esos puntos, lo cual sería equivalente a una simple enumeración exhaustiva; sin embargo, estos algoritmos pueden desarrollarse de forma que se enumere solamente a una porción de todas las soluciones posibles eliminando automáticamente a todos aquellos puntos que sean peores que el que se tiene; claramente, la eficiencia del método de búsqueda dependerá de la forma en la cual se eliminen aquellos puntos cuya solución no es tan buena como la obtenida hasta cierto punto del algoritmo.

Los métodos de búsqueda incluyen principalmente técnicas de enumeración implícitas y técnicas de "Branch and Bound"[3]; el primer tipo se enfoca de manera más eficiente para problemas 0,1 y de hecho puede ser considerado como un caso especial de los métodos de Branch and Bound.

Los métodos de Cortaduras se desarrollaron principalmente para los problemas de Programación Entera (Pura o Mixta) Lineal; estos métodos aprovechan el hecho de que la solución a un problema continuo está en un punto extremo del espacio de soluciones; es entonces que se generan restricciones (o Cortaduras) adicionales las cuales van eliminando puntos asociados a soluciones no enteras sin eliminar soluciones enteras factibles; la aplicación sucesiva de tal procedimiento, eventualmente debe guiar a un nuevo espacio

---

[2] Lo cual puede ser ambiguo en el caso de la Programacion Mixta

[3] Los metodos de Branch and Bound como su nombre lo indican, incluyen busquedas que se hacen en las "ramas" (branch) de arboles tomando cotas (bound) para decidir si se continua por la rama actual o se regresa para analizar otra. Dependiendo del algoritmo los arboles pueden ser binarios o tener varias ramas en cada nivel.

de soluciones donde uno de los puntos extremos corresponda a una solución entera; El nombre de "Cortaduras" proviene de la idea de "cortar" o quitar partes no factibles del espacio de soluciones continuo.

Existe otro método para los problemas de Programación Entera basado en las cortaduras el cual busca identificar la *Envolvente Convexa* de todos los puntos enteros factibles construyendo un conjunto de desigualdades lineales; una vez que esto se ha realizado, la aplicación normal del método Simplex deberá llevar a la solución deseada.

Existen métodos especializados para problemas bien estructurados como el caso del Problema de La Mochila, El Agente Viajero y otros ya mencionados en la sección anterior; a pesar de que estos métodos están diseñados para aprovechar la estructura especial del problema para el que fueron hechos, están inspirados por uno de los dos grupos anteriores (de Búsqueda o de Cortaduras) o una combinación de ambos.

Una propiedad común que caratériza a la mayoría de los métodos de Cortaduras es que no se tiene una solución factible (Entera) hasta el final del algoritmo; en otras palabras, no existe información disponible sobre la solución entera en caso de que no se termine el algoritmo; tales métodos se conocen como "Técnicas Duales" y esta propiedad representa una gran desventaja; se ha intentado desarrollar algoritmos primales (ie., manteniendo factibilidad entera durante todo el algoritmo) los cuales han sido poco satisfactorios desde el punto de vista computacional.

Los algoritmos de Búsqueda a menudo se denominan como "Casi Duales"; aunque no se garantiza factibilidad primal durante todo el algoritmo, ocasionalmente se puede obtener una buena solución entera factible, lo cual es una gran ventaja comparados con los métodos de Cortaduras[4]; sin embargo, los métodos de Búsqueda

---

[4] De hecho, en algunos algoritmos de Branch and Bound primero se hace una busqueda exhaustiva por una rama hasta encontrar una solucion factible entera que sirve de cota para despues buscar una mejor solucion; esta busqueda se detiene hasta encontrar la solucion optima o bien despues de un numero preestablecido de

(especialmente los de Branch and Bound) a menudo ocupan una gran cantidad de memoria en la computadora lo cual implica el uso de algoritmos computacionalmente más complicados para poder evitar este problema, lo cual es una dificultad inexistente en los métodos de Cortaduras.

Cabe comentar que los métodos de Búsqueda y de Cortaduras poseen propiedades exclusivas que, si se combinan, pudieran dar como resultado un método mejor; por decir, un algoritmo que obtenga soluciones factibles y que a la vez no ocupe demasiada memoria de computadora; tal idea ha guiado a el desarrollo de algoritmos interesantes.

Hasta aquí se ha hecho una breve introducción a la Programación Entera buscando dar una idea de lo importante que puede ser en las diversas aplicaciones en las que se utiliza; como se verá en el capitulo siguiente, el presente sistema implementa dos métodos de Cortaduras (Fraccional o Cíclico y Discreto) que serán presentados a continuación.

---

busquedas dando como resultado la mejor solucion entera hasta entonces lo cual resulta de gran utilidad en la practica donde se tienen problemas de grandes dimensiones.

CAPITULO 2

ALGORITMOS DISCRETO Y CICLICO

2.1 TABLA DE COEFICIENTES

La presente discusión se basará tomando como formato una nueva tabla; para identificar los elementos de ésta, se comparará con la tabla simplex que seguramente ya es familiar para el lector (apéndice A), para después trabajar la parte teórica de los algoritmos en el nuevo formato.

Para fijar ideas, se planteará al problema estándar de maximización en forma matricial:

$$\begin{aligned} &\text{Max } Z = cx \\ &\text{s.c.} \\ &\quad A\,x = b \\ &\quad x \geq 0 \\ &\quad x \in Z^n \end{aligned}$$

Donde:

- A es la matriz de coeficientes (de $m \times n$); $\text{rango}(A) = m < n$.
- c es un vector de constantes.
- x es el vector de variables.
- b es el vector de términos independientes.

El nuevo formato de la tabla será entonces:

$$x = \alpha + \beta\, x_J$$

donde el vector x incluirá a la función objetivo, el vector α

contendrá los valores actuales de todas las variables, β será el vector de los coeficientes de la tabla ($-y_i^j$ respecto a la tabla simplex), y x_J serán las variables no básicas.

Gráficamente se tiene:

	x^j			
	Z		x_k	
x_i				
	x_1		β_1^k	

De lo anterior, suponiéndo como pivote a β_l^k el sistema se transforma de la siguiente manera:

$$\alpha' = \alpha - \alpha_l / \beta_l^k \, \beta^k$$

$$(\beta')^j = \beta^j - \beta_l^j / \beta_l^k \, \beta^k \qquad j \neq k$$

Replanteando los algoritmos ya estudiados en la Programación Lineal en términos de este nuevo formato, se tendrá la siguiente tabla para el caso de maximizar.

ALGORITMO	CONDICIONES INICIALES	CONDICIONES OPTIMALIDAD
Simplex	$\alpha_i \geq 0, \; i \in \overline{1,n}$	$\beta_c^j \leq 0 \; \forall \; j \in J$
Dual Simplex	$\beta_c^j \leq 0 \; \forall \; j \in J$	$\alpha_i \geq 0, \; i \in \overline{1,n}$

Dual Lexicográfico	$\beta^j <_{LEX} 0,\ j \in J$	$\alpha_i \geq 0,\ i \in \overline{1,n}$
Discreto	$\alpha_i,\ \beta_i^j \in \mathbb{Z},\ i \in \overline{0,n}$ $\beta^j <_{LEX} 0,\ j \in J$	$\alpha_i \geq 0,\ i \in \overline{1,n}$
Cíclico	$\alpha_i \geq 0,\ i \in \overline{1,n}$ $\beta^j < 0,\ j \in J$	$\alpha_i \in \mathbb{Z},\ i \in \overline{1,n}$

2.2 ALGORITMO DISCRETO

El algoritmo Discreto es un algoritmo de cortaduras donde se inicia a partir de una solución óptima entera y se intenta llegar a una solución factible; al igual que en el algoritmo Fraccional o Cíclico que se verá más adelante, la cortadura que generan ambos cumplen dos reglas:

- Se excluye el punto asociado a la tabla anterior.
- Ninguna solución entera factible se elimina.

2.2.1 DESCRIPCION DEL ALGORITMO

Sea P el problema a resolver.

Inicialmente se tiene:

$$\alpha_i,\ \beta_i^j \in \mathbb{Z},\ i \in \overline{0,n}$$
$$\beta^j <_{LEX} 0,\ j \in J$$

1.- Si $\alpha_i \geq 0,\ \forall\ i \in \overline{1,n}$ Se tiene una solución factible óptima entera; sino, $\exists\ i$ t.q. $\alpha_i < 0$

a. Si $\beta_i^j \leq 0,\ \forall\ j \in J \Rightarrow$ P no tiene soluciones enteras factibles

---

[6] $<_{LEX}$ significa "mayor lexicográficamente"

b. $\exists\ \beta_i^j > 0$.

Como las soluciones en este algoritmo siempre son enteras, se desea encontrar un pivote con valor 1.

Sea $\lambda > 0$, "arbitrario"

$$x_i = \alpha_i + \sum_{j \in J} \beta_i^j x_j$$

dividiendo entre λ se tiene

$$\alpha_i/\lambda + \sum_{j \in J} \beta_i^j/\lambda\ x_j \geq 0$$

se debe cumplir

$$\sum_{j \in J} (\beta_i^j/\lambda)\ x_j \geq -\alpha_i/\lambda$$

tomando

$$[\beta_i^j/\lambda] = (\beta_i^j/\lambda) + r_j/\lambda, \quad 0 \leq r_j \leq \lambda$$

Los corchetes [ ] indican "el mínimo entero mayor a:"

tomando a éste último se cumple:

$$\sum_{j \in J} [\beta_i^j/\lambda]\ x_j \geq (-\alpha_i/\lambda) + \sum_{j \in J} (r_j/\lambda)\ x_j$$

y como

$$\sum_{j \in J} (r_j/\lambda)\ x_j \geq 0$$

para toda solución factible se tendrá

$$\sum_{j \in J} [\beta_i^j/\lambda]\ x_j \geq -\alpha_i/\lambda$$

y para toda solución *Entera* factible:

$$\sum_{j \in J} [\beta_i^j/\lambda]\ x_j \geq [-\alpha_i/\lambda] > 0$$

de lo anterior se tiene la cortadura o nueva restricción:

$$Y = -[-\alpha L/\lambda] + \sum_{j \in J} [\beta_L^j/\lambda] \; x_j$$

Ahora se ha llegado al punto de analizar el valor deseable para λ. Se tiene la desigualdad

$$\sum_{j \in J} [\beta_L^j/\lambda] \; x_j \geq [-\alpha L/\lambda]$$

Para simplificar la notación, sean

$$P_j = \sum_{j \in J} [\beta_L^j/\lambda]$$

$$q = [-\alpha L/\lambda]$$

entonces se tiene

$$\sum_{j \in J} P_j \; x_j \geq q$$

$$Y = -q + \sum_{j \in J} P_j \; x_j$$

Por una parte, se desea que λ sea lo suficientemente grande para que P_j sea 1;pero como

$$\alpha' = \alpha + q \; \beta^k$$

$$\alpha' - \alpha = q \; \beta^k$$

Se quiere que q sea grande para que exista el mayor cambio posible hacia el óptimo; de lo anterior, se busca la λ más pequeña tal que el elemento pivote sea 1.

Considerar que se tiene $k \in J$ y $P_k = 1$; entonces

$$(\beta')^j = \beta^j - P_j \; \beta^k$$

de aquí, se escogerá a k tal que:

$$\beta^j - P_j \beta^k <_{LEX}$$

Si $P_j \leq 0$, se cumple la condición; sino:

$$\beta^j < P_j \beta^k$$

De lo anterior, se necesita encontrar k tal que sea el máximo lexicográfico de estas columnas

$$1/P_j \beta^j < 1/P_k \beta^k \Rightarrow \beta^k >_{LEX} \beta^j$$

Por lo tanto, la regla es elegir $k \in J$ tal que

$$\beta^k = \underset{j/P_j>0}{MaxLex} \{ \beta^j \} \quad \Leftrightarrow \quad \beta^k = \underset{j/\beta_{ij}>0}{MaxLex} \{ \beta^j \}^2$$

Concluyendo, la columna que se eligirá será:

$$\beta^k = \underset{j/\beta_{ij} > 0}{MaxLex} \{ \beta^j \}$$

Sea μ_j el mayor entero tal que

$$\mu_j \beta^k >_{LEX} \beta^j , \quad \forall\ j \neq k$$

Como $\beta^k >_{LEX} \beta^j \Rightarrow \mu_j \geq 1$; cabe hacer notar que si sucede que

$$\beta^j \qquad \beta^k$$

$$\begin{bmatrix} -1 \\ . \\ . \\ . \\ . \end{bmatrix} \qquad \begin{bmatrix} 0 \\ -1 \\ . \\ . \\ . \end{bmatrix}$$

---

[2] MaxLex = Maximo Lexicografico

entonces μ_j no está acotado. Como

$$\beta_i^j/\lambda \leq [\beta_i^j/\lambda] = P_j \leq \mu_j \quad \text{entonces} \quad \mu_j \geq \beta_i^j/\lambda$$

De aquí, $\lambda \geq \beta_i^j/\mu_j$; y por tanto, el valor óptimo para λ será:

$$\lambda = \text{Max } \beta_i^j/\mu_j > 0, \ \forall \ j \text{ t.q. } \beta_i^j > 0$$

Con $\mu_k = 1$, $\lambda \geq \beta_i^k$ y por tanto $P_k = [\beta_i^k/\lambda] = 1$.

2.2.2 CONVERGENCIA DEL ALGORITMO

Se supone que se conoce $\underline{Zo}$, cota inferior para los valores de Z en las tablas; como

$$x \leq_{LEX} \alpha \qquad \alpha' <_{LEX} \alpha$$

entonces

$$Zo \geq Zo' \geq Zo'' \geq \ldots \geq \underline{Zo}$$ [a]

Por lo cual no puede decrecer indefinidamente. Los demás componentes α_i, $i \in \overline{i,n}$ pueden :

a. Decrecer indefinidamente.
b. Permanecer fijos a partir de cierta iteración.

b) Si permanecen fijos, entonces se debe tener $\alpha_i \geq 0, i \in \overline{i,n}$ Suponer que existe $l \in \overline{i,n}$ tal que

$$\alpha_l < 0 \Rightarrow \alpha_l' = \alpha_l + q\,\beta_l^k > \alpha_l \Rightarrow \text{no permanece fija}$$ ∎

a) Suponer que las primeras r componentes quedan fijas a

[a] Si la desigualdad es estricta, se decrece en enteros

partir de cierta iteración y que la componente $r + i$ decrece indefinidamente:
Tomemos α_r;

$$\alpha_r' = \alpha_r + q\,\beta_r^k > \alpha_r \quad \#$$
$$\Rightarrow \alpha \text{ no decrece (de hecho creció)}$$

De a) y b) el algoritmo converge.

De lo anterior, una regla heurística será el elegir el primer renglón l tal que α_l sea negativo; esta filosofía es la que se aplica en el sistema aunque de hecho, el mejor candidato sería:

$$l \text{ tal que } \quad \alpha_l < 0,$$

$$k(l) \text{ tal que } \quad \beta^{k(l)} = \underset{j\,|\beta_{lj} > 0}{\text{MaxLex}} \left(\beta^j \right)$$

$$\lambda(l) \text{ tal que } \quad \lambda(l) = \underset{j\,|\beta_{lj} > 0}{\text{Max}} \left(\beta_l^j / \mu_j \right)$$

$$q(l) = [-\alpha_l / \lambda(l)]$$

De lo anterior, el mejor candidato será

$$l \text{ tal que } \quad q(l)\beta^{k(l)} = \underset{i\,|\alpha_i < 0}{\text{MinLex}} \left(q(l)\,\beta^{k(l)} \right)$$

2.2.3 EJEMPLO NUMERICO

Resolver el siguiente problema:

$$\begin{aligned} \text{Max } Z &= -2x_1 - 6x_2 \\ \text{s.c.} & \\ & 2x_1 + 2x_2 \geq 9 \\ & 3x_1 + x_2 \geq 11 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

La tabla inicial será:

		X_1	X_2
Z	0	-2	-5
X1	0	1	0
X2	0	0	1
X3	-9	2	2
X4	-11	1	3

De aquí, $\mu_1 = 1$ y $\mu_2 = 2$;

$$\lambda = \text{Max} \{ 2/1 , 2/2 \} \rightarrow \lambda = 2$$

El renglón $l = 3$ puesto que $a_3 = -9 < 0$ y la nueva restricción será:

$$X_5 = -[9/2] + [2/2]X_1 + [2/2]X_2$$

que en la tabla será:

		X_1	X_2
Z	0	-2	-5
X1	0	1	0
X2	0	0	1
X3	-9	2	2
X4	-11	1	3
X5	-5	1	1

Pivoteando sobre la nueva restricción, se tendrá la siguiente tabla:

		X_5	X_2
Z	-10	-2	-3
X1	8	1	-1
X2	0	0	1
X3	1	2	0
X4	-8	1	2
X5	-3	1	1

Para obtener la cortadura se tiene que $\mu_1 = 1$, $\mu_2 = 1$;

$$\lambda = \text{Max} \langle 1/1 \ , \ 2/1 \rangle \Rightarrow \lambda = 2$$

$$X_6 = -[8/2] + [1/2]X_1 + [2/2]X_2$$

La tabla obtenida después de pivotear sobre la nueva restricción es:

		X_6	X_2
Z	-18	-2	-1
X1	8	1	-2
X2	0	0	1
X3	7	2	-2
X4	-3	1	1

En este caso se puede proceder igual que en las iteraciones anteriores, pero es conveniente el notar que se generaría una cortadura idéntica a X_4 ya que $\lambda = 1$; por lo tanto se pivotea sobre ésta restricción obteniéndose la tabla final del problema:

		X_5	X_4
Z	-19	-1	-1
X1	2	3	-2
X2	3	-1	1
X3	1	4	-2
X4	0	1	0

Como se puede ver las nuevas restricciones no se continúan escribiendo ya que carecen de interés al ser variables no básicas con valor cero pués se van eliminándo al generarse la siguiente cortadura y en el caso de la última (X_5 en este caso), es una variable no básica con valor cero.

2.3 ALGORITMO FRACCIONAL O CICLICO

El algoritmo Fraccional o Ciclico es un algoritmo de cortaduras que trabaja de forma distinta al algoritmo Discreto, ya que aquí se comienza con una solución factible óptima continua y se trata de llegar a una solución entera; al igual que el algoritmo anterior, cumple con las dos caracteristicas de eliminar al punto asociado a la tabla anterior y el de no eliminar a ninguna solución entera factible.

2.3.1 DESCRIPCION DEL ALGORITMO

Inicilamente se tiene:

$$\alpha_i \geq 0, \quad i \in \overline{1,n}$$
$$\beta \prec_{LEX} 0, \quad j \in J$$

1.- Si $\alpha_i \in Z$, $i \in \overline{1,n}$ entonces se tiene una solución entera óptima.

2.- Si $\exists\ i \in \overline{1,n}$ tal que $\alpha_i \notin Z$:

Sea $\{\alpha_i\}$[4]

$$\alpha_i = \{\alpha_i\} + f_i, \quad 0 < f_i < 1$$

De aquí se tiene que:

$$x_i = \alpha_i + \sum_{j \in J} \beta_i^j x_j$$

$$x_i - \{\alpha_i\} = f_i + \sum_{j \in J} \beta_i^j x_j$$

De lo anterior, para toda solución entera se tendrá

---

[4] $\{\}$ = "El maximo entero menor a"

$$f_i + \sum_{j \in J} \beta_i^j \, x_j \in Z$$

donde

$$\beta_i^j = [\beta_i^j] + f_i^j \, , \; 0 \leq f_i^j \leq 1$$

Por tanto

$$f_i + \sum_{j \in J} [\beta_i^j] \, x_j + \sum_{j \in J} f_i^j \, x_j \in Z$$

$$-f_i + \sum_{j \in J} f_i^j \, x_j \in Z$$

Por lo tanto para toda solución factible se tendrá:

$$\sum_{j \in J} f_i^j \, x_j \geq f_i$$

además

$$\sum_{j \in J} f_i^j \, x_j \equiv f_i \pmod 1$$

Por lo tanto la restricción generada será:

$$Y = -f_i + \sum_{j \in J} f_i^j \, x_j$$

Con $Y \in Z$. Las fórmulas de transformación serán las siguientes:

$$\alpha' = \alpha + (f_i / f_i^k) \, \beta^k$$

$$(\beta')^j = \beta^j - (f_i^j / f_i^k) \, \beta^k \qquad j \in J - (k)$$

$$(\beta')^Y = (1 / f_i^k) \, \beta^k$$

Como

$$\beta^j - (f_l^j / f_l^k)\,\beta^k \leq_{LEX} 0, \quad (1/f_l^j)\,\beta^j \leq (1/f_l^k)\,\beta^k \quad j \text{ t.q. } f_l^j > 0$$

Entonces se buscará k tal que

$$(1/f_l^k)\,\beta^k = \underset{j \mid f_l^j > 0}{\text{MaxLex}} \; \langle (1/f_l^j)\,\beta^j \rangle$$

Cabe aclarar que esto garantiza que $\beta^j \leq_{LEX} 0$, pero no que $\alpha_i \geq 0$ por lo que es posible que alguna $\alpha_i < 0$; en este caso, habrá que reoptimizar con el algoritmo Dual Simplex sin agregar nuevas restricciones.

Para elegir el renglón candidato una buena regla (que se implementa en el sistema) es la de elegir el primer α_i no entero, aunque el mejor sería el siguiente:

Elegir i tal que:

$$(f_i / f_l^{k(l)})\,\beta^{k(l)} = \underset{i \mid \alpha_i \notin \mathbb{Z}}{\text{MinLex}} \; \langle (f_i / f_l^{k(l)})\,\beta^{k(l)} \rangle$$

2.3.2 CONVERGENCIA DEL ALGORITMO

Sea $\underline{Z}$ cota inferior

Siguiendo el ciclo de añadir una cortadura y reoptimizar se tiene:

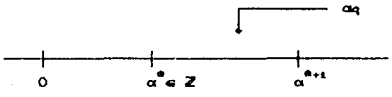
$$\alpha^0 >_{LEX} \alpha^1 >_{LEX} \alpha^2 >_{LEX} \ldots >_{LEX} \alpha^i >_{LEX} \ldots \geq_{LEX} \underline{Z}$$

- Si a partir de cierta iteración todos los componentes de α quedan fijos, entonces son enteros.

Suponer que las primeras q componentes quedan fijas y α_q decrece sobre un número infinito de ciclos; entonces:

$$\alpha_q^i \geq \alpha_q^{i+1}$$

Para visualizarlo, se plantea el siguiente diagrama:



Si se toma el renglón de α_q, la cortadura sería:

$$\sum_{j \in J} f_i^j x_j \geq f_i$$

$$\alpha_q' = \alpha_q + (f_i / f_q^k)\ \beta_q^k \qquad f_q = \alpha_q - \alpha^*$$

donde $\alpha^* \in Z$. Se debe tener que $\beta_q^k < 0$ por lo que se tienen dos casos:

a) $\beta_q^k < -1, \qquad 0 < f_q^k < 1, \qquad \beta_q^k / f_q^k < -1$

de donde

$$(f_q / f_q^k)\ \beta_q^k < -f_q$$

$$\alpha_q + (f_q / f_q^k)\ \beta_q^k < \alpha_q - f_q$$

$$\alpha_q' < \alpha^* \quad !!\ (\text{Pués se sale del intervalo})$$

b) $0 > \beta_q^k > -1, \qquad \beta_q^k / f_q^k = -1$

de aquí se tiene que

$\alpha_q' = \alpha_q - f_q = \alpha^*$!! (Pués se consideró que estaba en el intervalo abierto)

2.3.3 EJEMPLO NUMERICO

Resolver el siguiente problema:

$$\text{Max } Z = -5x_1 - 2x_2$$

s.c.

$$2x_1 + 2x_2 \geq 9$$
$$3x_1 + x_2 \geq 11$$
$$x_1, x_2 \geq 0$$
$$x_1, x_2 \in Z$$

La tabla inicial factible óptima continua será:

		X_4	X_5
Z	-75/4	-3/2	-1/4
X1	13/4	1/2	-1/4
X2	5/4	-1/2	3/4
X3	0	0	1
X4	0	1	0
X5	-1/4	1/2	1/4

La cortadura (X5) se obtuvo de la siguiente manera:

El renglón i será 1 ya que $\alpha_i \notin Z$.

$$f_i = 13/4 - \{13/4\}; \quad f_i = 13/4 - 3 \Rightarrow f_i = 1/4$$

$$f_j^i = [1/2] - 1/2; \quad f_j^i = 1 - 1/2$$

$$f_j^i = 1/2$$

$$f_4^2 = [-1/4] + 1/4; \quad f_4^2 = 0 + 1/4$$

$$f_4^2 = 1/4$$

Pivoteando sobre la restricción región generada se tiene la siguiente tabla que es la óptima

		X_4	X_5
Z	-19	-1	-1
X1	3	1	-1
X2	2	-2	3
X3	1	-2	4
X4	0	1	0

Al igual que en el algoritmo anterior, no se incluye el renglón de X_5 puesto que es una variable no básica con valor cero que no se encontraba en el problema original.

Así, de manera breve, se ha bosquejado la teoría en la cual se basa la implementación del presente sistema; ésta se encuentra en el capítulo 5 en las unidades "Ciclico" para el algoritmo Cíclico y "Discreto" para el algoritmo Discreto.

CAPITULO 3

DESCRIPCION DEL SISTEMA

3.1 QUE ES UN MODULO

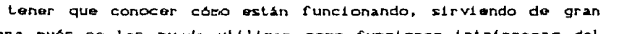
El sistema está organizado tomando la filosofía de la modularización; ésta consiste en la posibilidad que ofrecen algunos lenguajes de poder dividir un programa en módulos o unidades independientes lo cual permite mejorar la estructura de un sistema en los siguientes aspectos:

- Permite el identificar errores más fácilmente
- Dichos módulos pueden ser utilizados por varios programas
- El sistema puede ser modificado con mayor facilidad

Una unidad consta de dos partes; la pública y la interna; en la primera se encuentran los procedimientos y variables que "exporta", es decir, aquellos procedimientos que podrán ser utilizados por otros programas con sólo "importarlos"; en la parte interna, se implementan estos procedimientos y otros que sólo son de utilidad dentro de la unidad; estos procedimientos no pueden ser usados por otras unidades pués no han sido declarados como "públicos". Para que un programa o unidad pueda usar algún procedimiento o variable exportada por otra unidad, basta indicar al principio que se usará dicha unidad "heredándose" así todos los procedimientos y variables declarados como públicos en las

unidades que importa.[1]
La modularización permite también el separar tareas específicas en módulos distintos teniéndose así un manejo de memoria y espacio más eficiente; la idea de modularización no es nueva y existen algunos lenguajes que la tomaron como base para su estructura; con esto se permite el reutilizar procedimientos que de otra forma tendrían que ser incluidos en el programa, pero gracias a la modularización sólo es necesario el declarar que se usan. Otro punto de la utilidad de los módulos es el poderlos usar como "cajas negras" a manera de componentes electrónicos, donde se usan sin tener que conocer cómo están funcionando, sirviendo de gran manera pués se les puede utilizar como funciones intrínsecas del lenguaje.

La estructura de los módulos o unidades en el compilador usado[2] es la siguiente:



Parte Pública:

Interface ⟶ [**Uses**] ⟶ [*Declaración de Constantes, Tipos, Variables y Procedimientos que se exporta.*]

Implementación:

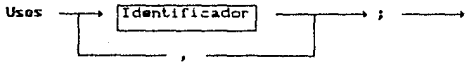
Implementation ⟶ [**Uses**] ⟶ [*Declaración de Constantes, Tipos, Variables y Procedimientos que se utilizan dentro del programa incluyendo aquellos que se exportan.*]

---

[1] En algunos lenguajes como Modula-2, es posible importar solamente aquellos procedimientos que necesitamos, siendo esto de utilidad pues no siempre deseamos importar todos los procedimientos de una unidad.

[2] TurboPascal Ver. 5.0

Para poder usar los procedimientos, variables, constantes y funciones que exporta una unidad basta especificar que se está utilizando esta unidad:



3.2 OBJETIVOS DEL SISTEMA

El crear un sistema de Programación Entera puede no ser por sí sólo algo nuevo; sin embargo, el presente sistema pretende reunir ciertos aspectos de importancia que no siempre se reconocen como tales; dentro del aspecto teórico y técnico, el sistema está enfocado a la resolución de problemas de Programación Entera de tamaño pequeño (No más de 30 variables y 15 restricciones), enfocándose más al análisis de problemas que a la obtención de soluciones sirviéndo como un "laboratorio de problemas"; en cuánto a los demás objetivos, los podemos enumerar de la siguiente forma:

- Interacción usuario - máquina.
- Ambiente de programación agradable.
- Completez del sistema.

A continuación se detallan los objetivos anteriores:

INTERACCION USUARIO - MAQUINA

El sistema pretende que el usuario pueda utilizarlo para resolver problemas sin necesidad siquiera de conocer qué algoritmos se usan, con que parámetros y en que orden, pero esto no siempre es deseable cuando se quiere analizar detalladamente ciertos aspectos de un problema; por ello se permite al usuario el modificar la estrategia seguida de forma estándar por el sistema

de forma que pueda observar los diferentes cambios que sufren las variables del problema; esta facilidad se encuentra dentro de la opción de *Opciones* que altera algunos parámetros de los procedimientos que ejecutan las operaciones principales de los algoritmos implementados; específicamente, las opciones posibles son las siguientes:

- Mostrar la restricción o cortadura generada.
- Elegir el renglón sobre el cual se pivoteará.
- Mostrar al elemento pivote.
- Elegir el factor λ para generar la restricción en el algoritmo Discreto.

Para sacar provecho de estas opciones es conveniente el resolver los problemas mediante la opción *Pivotear* la cual ejecuta una sola iteración del algoritmo usado. De esta forma se logra una interacción usuario - máquina que además de ser interesante es útil en los casos que se conoce de antemano algún dato que ayude a resolver el problema con mayor rapidez.

AMBIENTE DE PROGRAMACION

El ambiente de programación resulta a menudo un factor determinante para que un usuario se sienta agusto dentro del sistema; en ocasiones no es suficiente el que un programa realice sus tareas con eficacia, sino que lo haga de una forma fácil.

Para lograr este objetivo el sistema se maneja a base de menús de fácil acceso tratando que las opciones de éstos sean lo más claras posibles pudiéndo elegirlas de varias formas como se comenta más adelante; además de esto se cuenta con la posibilidad de pedir ayuda escogiéndo esta opción del menú principal; ésta se divide en tres para encontrar de forma más rápida el punto deseado siendo estas divisiones:

- Ayuda General (sobre las opciones del menú).
- Edición de problemas.
- Características del Sistema.

En cuanto a la edición de problemas, se da gran libertad al usuario para que no tenga que modificar de forma alguna el problema que tenga en papel pudiéndo elegir nombres de variables de hasta cinco caracteres; cabe comentar que el dar este tipo de libertad puede significar un gran problema para el sistema pero suele ser altamente estimado por quien lo usa.

COMPLETEZ DEL SISTEMA

Si bien el objetivo principal del sistema es el de implementar métodos para Programación Entera, ésta es sólo una parte importante del sistema ya que además implementan algoritmos de Programación Lineal para poder hacer las transformaciones necesarias a los problemas; estos son : Algoritmo Simplex (Apéndice B), Algoritmo Dual Lexicográfico (Apéndice C), Método de las dos Fases (Apéndice D) y Restricciones Artificiales; haciendo a un lado la parte técnica del sistema y para mayor manejabilidad del programa, se incluyen opciones de impresión de la tabla actual del problema a pantalla y a papel, opción de resolver el problema poco a poco o de una sola vez, información de la fecha y de la hora del día en que se está trabajando además de las características del sistema operativo en el que se está (Versión) y del espacio utilizado y disponible tanto en disco como en memoria RAM.

3.2 ORGANIZACIÓN DEL SISTEMA

El presente sistema consta de ocho módulos o unidades con diversos fines; a continuación se presenta una sinópsis de las funciones de cada unidad junto con un diagrama de dependencia entre éstas.

Unidad Global:

En esta unidad se encuentran declarados todos

los tipos no básicos introducidos por el programa; también se declaran variables que ya sea por su importancia o por su uso frecuente serán reconocidas en todo el sistema;entre éstas podemos mencionar a variables booleanas de control y aquellas que determinan el número de renglones y columnas de la matriz de coeficientes; además se declaran todos los códigos necesarios para el reconocimiento del teclado. En esta unidad se encuentran también procedimientos de salida y el procedimiento de errores el cual señala que se ha cometido un error además de indicar en que consiste este; también se implementa un procedimiento que proporciona información sobre el sistema, por lo que en resumen ésta es una unidad declarativa.

Unidad Detecta:

Al referirse a la libertad de edición del problema, esta unidad es el alma del sistema, ya que es un pequeño editor; está diseñado para brindar al usuario una libertad casi total, permitiéndole pasar su problema del papel al sistema sin necesidad de hacer modificaciones; las funciones que realiza esta unidad incluyen entre otras:

- El añadir variables de holgura y artificiales en caso de ser necesario.
- El detectar e identificar la función objetivo.
- El reconocer las variables del problema sin necesidad de indicarle cuantas o cuales son, pudiendo éstas tener nombres de hasta cinco caracteres.
- El revisar las restricciones del problema.
- El identificar el número de renglones y columnas de la matriz de coeficientes así como de ubicar en ésta los coeficientes del problema.

De lo anterior, es fácil ver que es una parte medular del sistema en cuanto al objetivo de la libertad de edición, lo cual es uno de los aspectos que hacen que un programa sea "usable" o no.

Unidad Discreto:

Como su nombre lo indica esta unidad implementa al algoritmo Discreto teniendo procedimientos auxiliares como lo es el procedimiento "Imprime" que se encarga de imprimir en pantalla la matriz de coeficientes, el vector de términos independientes, las variables del problema y cuales de éstas son no básicas; todo esto en el formato presentado en el capítulo anterior; cabe aclarar que gracias a la modularización, este procedimiento podrá ser usado por otras unidades con el sólo hecho de incluir en sus declaraciones que "usan a la unidad Discreto". Algunos de los procedimientos implementados en esta unidad, al igual que otros de la unidad *Cíclico*, tienen la opción de ser alterados por el usuario; ya sea para elegir un renglón candidato para encontrar al elemento pivote, mostrar la nueva cortadura recién generada o en el caso del algoritmo Discreto, el elegir una λ distinta a la calculada por el sistema.

Unidad Cíclico:

Esta unidad se encarga de implementar al algoritmo Cíclico; la descripción de esta unidad es simplemente la implementación del algoritmo presentado en el capítulo anterior con algunas opciones similares a las comentadas en la unidad anterior por lo cual no comentaremos aquí la función de los procedimientos que son análogos a los de la unidad *Discreto*.

Unidad Informa:

Dentro del objetivo de tener un ambiente de programación agradable y de fácil manejo, siempre es deseable el tener procedimientos de ayuda; ya sea para la edición de un problema, el encontrar una solución inicial o el conocer las características del sistema; por esta razón se implementa en esta unidad algunas notas que serán de utilidad al usuario. Se implementan tres procedimientos; uno dedicado a la ayuda general que indica la función que realiza cada una de las opciones del menú; otro sobre un ejemplo de la edición de problemas con algunos comentarios y finalmente, un tercero que resuma algunas de las características del sistema en cuanto a capacidad, algoritmos que implementa y algunas notas importantes; la idea de implementar una

unidad de ayuda es de que no sea necesario el usar un manual o este escrito como apoyo sino que sea autosuficiente.

Unidad Impresor:

Dentro del objetivo de completez del programa, es importante el poder imprimir a papel la tabla actual del problema con el cual se está trabajando; si bien es posible el imprimir los datos de pantalla en cualquier momento (presionando la tecla [PrintScreen]), es deseable el tener la opción de impresión a papel por dos motivos; uno, para poder analizar posteriormente el problema o revisar su desarrollo a través de otras tablas, y dos, porque debido a la intención del sistema de brindar una presentación agradable y objetiva de los datos, en algunos problemas de gran tamaño la tabla mostrada en pantalla puede desdibujarse perdiéndose la correcta ubicación de los coeficientes de ésta; este problema se soluciona imprimiendo a papel la tabla actual del problema donde se cuenta con mayor espacio.

Unidad Ajusta:

Esta unidad juega un papel de gran importancia dentro del sistema ya que se encarga de poner el problema editado en el formato adecuado para aplicar uno de los dos algoritmos de Programación Entera (Discreto y Cíclico) implementados.

Al realizar un sistema en alguna área específica como lo es la Programación Entera, existen dos opciones; una, el restringir la edición de problemas a un formato estricto preestablecido lo cual es molesto en la mayoría de los casos ya que implica el tener que preprocesar al problema que es por sí sólo un proceso que abarca una serie de algoritmos de Programación Lineal; la otra opción es facilitarle esta tarea al usuario realizándola dentro del sistema; siguiendo esta filosofía, en esta unidad se implementa el algoritmo Dual Lexicográfico, así como el método de las dos fases y el manejo de restricciones artificiales para obtener soluciones duales factibles; por lo anterior, el programa podría ser usado para resolver y analizar problemas de Programación Lineal, pero como este no es el principal objetivo, el sistema ha limitado esta posibilidad a obtener la solución

óptima finita de un problema de Programación Lineal (si existe) sin permitir el analizar el procedimiento que se sigue para llegar a ésta; esto se logra eligiendo la opción de resolver al problema mediante el algoritmo Cíclico y viendo su tabla inicial ya que ésta, como se vió en el capítulo anterior, es factible y óptima dentro de la Programación Lineal.

Unidad Tesis:

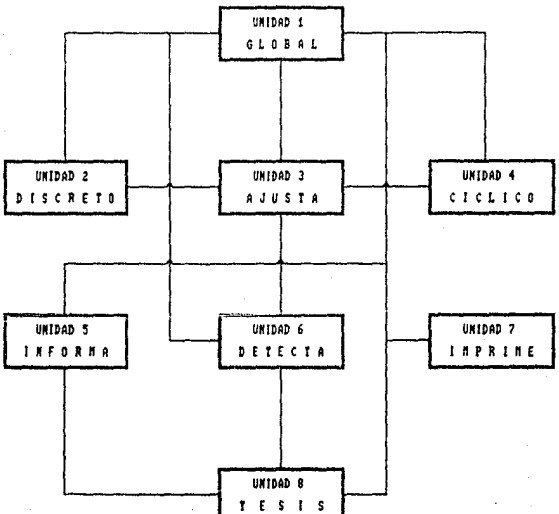
Esta es la unidad principal del sistema ya que es la que coordina a todas las demás; si bien carece de interés técnico, resulta de importancia dentro de los objetivos del sistema, ya que aquí se implementan las presentaciones a las diversas opciones del sistema.

Más detalladamente, el programa implementa la elección de opciones mediante el procedimiento *Espera*, el cual detecta la tecla presionada y de acuerdo a ésta efectúa cierta operación; al igual que algunos paquetes comerciales, se permite escoger una opción mediante dos formas; una moviéndose con las flechas hasta posicionarse sobre la opción elegida y presionar la tecla [*Enter*], y la otra simplemente presionando la primer letra de la opción la cual se encuentra en un color distinto a las demás letras de la opción.

A partir del menú principal se puede entrar a submenús como es el caso de la ayuda y de la edición manejándose en todos los casos la facilidad de elegir las opciones oprimiendo la primer letra de sus respectivos nombres.

ORGANIZACION DEL PROGRAMA

DEPENDENCIA DE MODULOS O UNIDADES



DESCRIPCION DE UNIDADES.

UNIDAD 1 : Declaraciones Globales del programa

UNIDAD 2 : Algoritmo Discreto

UNIDAD 3 : Ajusta la tabla leida al formato necesario para poder aplicar alguno de los dos algoritmos.

UNIDAD 4 : Algoritmo Ciclico; incluye tambien a los algoritmos Dual Lexicografico y Primal.

UNIDAD 5 : Implementacion de los procedimientos de ayuda del sistema.

UNIDAD 6 : A partir de la entrada por pantalla detecta variables y restricciones creando la tabla correspondiente al problema.

UNIDAD 7 : Imprime en papel la tabla actual del problema.

UNIDAD 8 : Presenta al programa y coordina todas las unidades anteriores.

3.4 UN RECORRIDO POR EL SISTEMA

Con el objetivo de dar una idea al lector del formato, presentación y funcionamiento del sistema, se resolverá el problema presentado en el capitulo anterior.

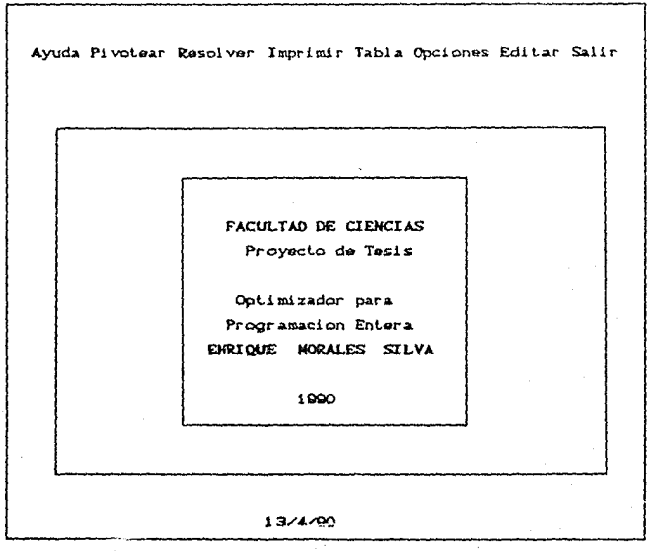


Fig. 3.1 Presentacion del Sistema

El problema se resolverá mediante el algoritmo Cíclico iteración por iteración para poder apreciar las diferentes opciones que ofrece el sistema.

$$\text{Max } Z = -5x_1 - 2x_2$$

s.c.

$$2x_1 + x_2 \geq 9$$
$$3x_1 + x_2 \geq 11$$
$$x_1, x_2 \geq 0$$

Se introduciría tecleandolo de la manera siguiente, después de haber elegido la opción *Editar* en el menú principal y las opciones *Algoritmo Cíclico* y *Editar Problema* en el siguiente menú (Notar que el sistema asume la no negatividad de las variables).

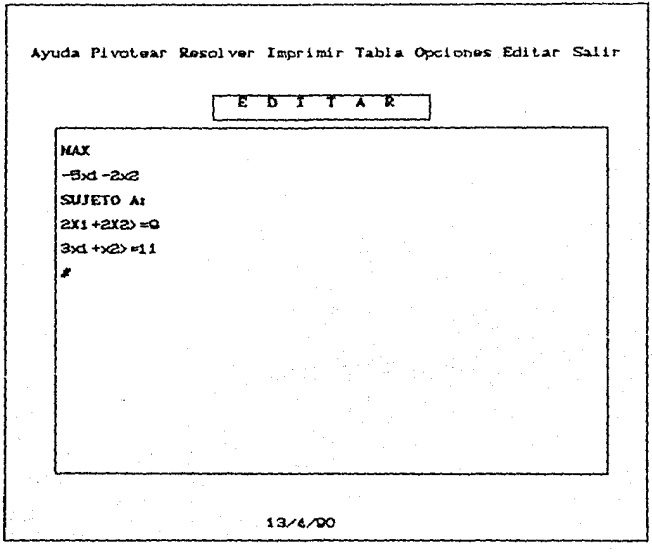


Fig. 3.3 Edicion de un Problema.

La frase "SUJETO A:" es puesta automáticamente por el sistema y el símbolo "#" es necesario para indicar que se ha terminado la edición del problema.

Otra de las facetas del programa, como ya se mencionó, es la de tener una pantalla de opciones donde se elijan las diferentes formas posibles de resolver un problema; aquí se elegirá para este problema la opción de escoger el renglón sobre el cual se pivoteará.

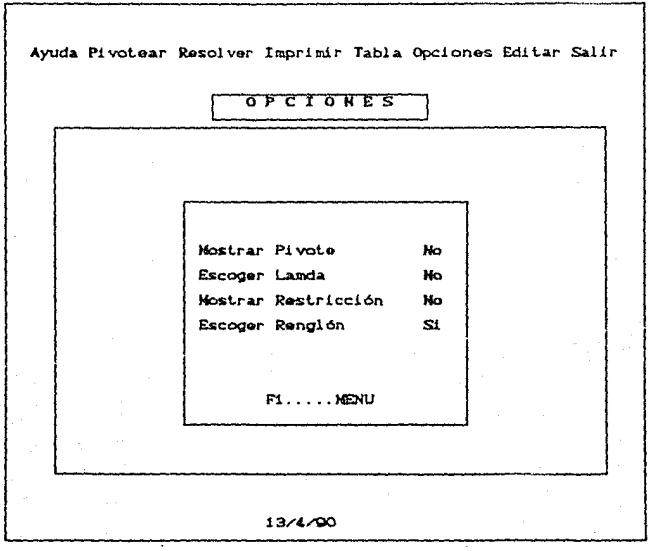


Fig 3.4 Pantalla de Opciones

Una vez editado el problema y elegido las opciones para su resolución, se puede escoger la opción *Tabla* que permite ver la tabla actual del problema; en este caso, la solución inicial, la cual resulta ser, debido al algoritmo elegido, la solución óptima continua al problema.

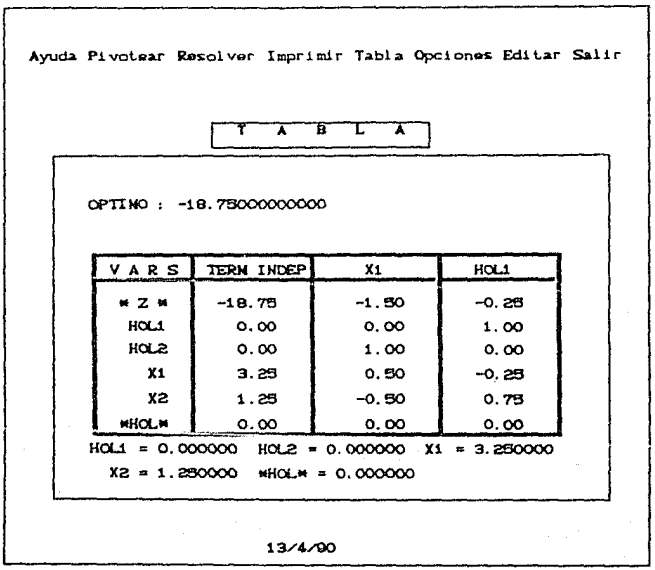


Fig. 3.5 Tabla de datos

Aquí se ven los datos en el formato presentado en el capítulo II; las variables no básicas se encuentran en el encabezado de la tabla, en el caso de usarse restricción artificial y por tanto, un valor "grande" para M, éste aparece arriba de la tabla; los términos independientes y el valor de la función objetivo aparecen

en dos lugares, en la tabla para visualizarlos, y en un lugar aparte donde se presentan con mayor presición.

Para resolver el problema, pueden utilizarse 2 opciones; una resolverlo totalmente y otra efectuar una sola iteración del algoritmo; como se escogió la opción de elegir el renglón sobre el cual se pivoteará, se resolverá el problema con la opción *Pivotear*; automáticamente se presenta la siguiente pantalla para elegir el renglón:

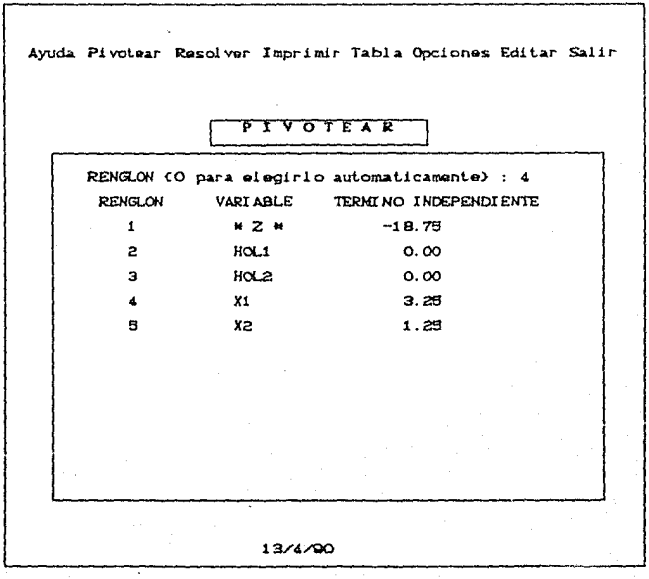


Fig. 2.6 Eleccion de renglon pivote

Una vez elegido el renglón pivote, se efectúa la iteración después de la cual se obtiene la siguiente tabla que en este caso

resulta ser la última.

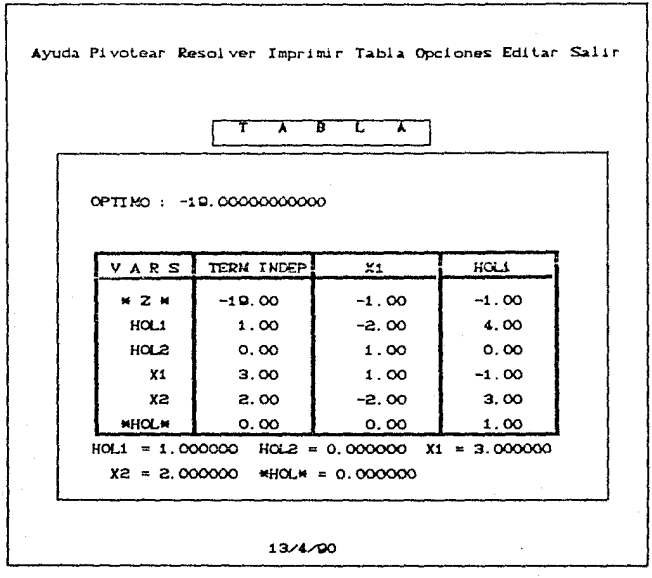


Fig. 2.7 Tabla Optima

Una vez resuelto el problema, se puede regresar a la opción *Editar*, la cual permite editar un nuevo problema, o bien modificar el problema actual, ya sea añadiendo, quitando o modificando restricciones, variables y/o función objetivo; dicha pantalla es la siguiente:

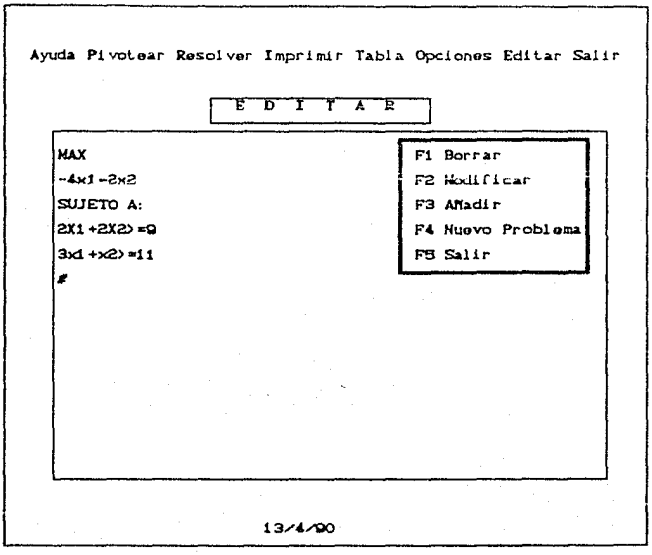


Fig. 3.8 Edicion de un Problema.

Como se ve, en este caso se modificó la función objetivo; pudiéndo resolver este problema de la misma forma que el anterior sin necesidad de volver a teclearlo por completo.

3.5 PROGRAMA DE PRESENTACION

Si bien la Programación Entera es de gran importancia en la práctica, no todos conocemos sobre ella; por está razón y a manera de presentación, se incluye el archivo "LEEME.EXE" el cual sirve de presentación al sistema indicando el lenguaje y compilador usado para su desarrollo así como una breve introducción a la Programación Entera y a los métodos de cortaduras (Apéndice E) respectivamente; a continuación se presenta al programa junto con un listado del mismo.

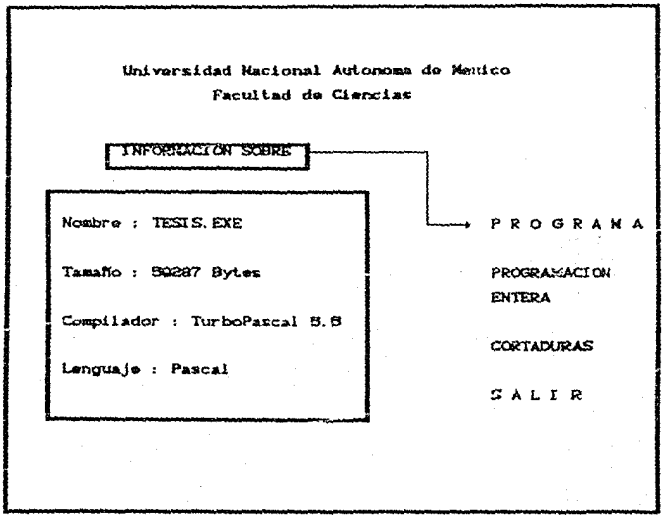


Fig. 3.9 Programa de Presentacion "LEEME.EXE"

Para elegir una opción basta con moverse con las flechas hasta que el indicador señale la opción deseada y presionar la tecla [Enter] apareciendo entonces en la ventana pequeña una breve introducción sobre el tema elegido; los archivos que lee este programa son:

TEMA	ARCHIVO
Programa	PRESENTA.DOC
Programación Entera	ENTERA.DOC
Cortaduras	CORTADUR.DOC

Por esta razón es de importancia que estos archivos también esten presentes al cargar el archivo "LEEME.EXE".

3.5.1 LISTADO DEL PROGRAMA DE PRESENTACION

```
Uses Crt;


  Var

    Pos,i:Byte;
    Ch:Char;

{ ********************** ESCRIBE ********************************)
 { Procedimiento que escribe el texto en "cad" en las coordenadas
x,y }

   Procedure Escribe(x,y:byte;cad:string);

    Begin
       Gotoxy(x,y);
       Write(Cad);
    End;


{********************************** MARCO ************************)
 (Procedimiento que pinta un marco de línea doble tomando como
puntos extremos las cuatro variables que recibe como parámetros)

   Procedure Marco(Uno,Dos,Tres,Cuatro:Byte);

   Var
    i,j:Byte;

   Begin
    Window(1,1,80,25);
    Gotoxy(Uno+1,Dos);Write('╔');
    Gotoxy(Uno+1,Cuatro);Write('╗');
    Gotoxy(Tres-1,Dos);Write('╚');
    Gotoxy(Tres-1,Cuatro);Write('╝');
    For i:=(Uno+2) to (Tres-2) do
     Begin
      Gotoxy(i,Dos);Write('═');
      Gotoxy(i,Cuatro);Write('═');
     End;
    For i:=Dos+1 to  (Cuatro-1) do
     Begin
      Gotoxy(Uno+1,i);Write('║');
      Gotoxy(Tres-1,i);Write('║');
     End;
     Window(Uno+2,Dos+1,Tres-2,Cuatro-1);

   End;
```

```
{************************** ERROR **************************}
{Procedimiento que reporta un mensaje de error en caso de que
alguno de los archivos que se leen no está presente}

  Procedure Error;

    Begin
      GotoXY(14,15);
      Write('ARCHIVO NO PRESENTE');
      Delay(1000);
      GotoXY(14,15);
      Write(' ':20);
    End;


{************************** LEE ARCHIVO **************************}
{Procedimiento que lee el archivo que se encuentra en la variable
"Nombre"}

  Procedure LeeArchivo(Nombre : String);
  Var
    Archivo : Text;
    Car : Char;

  Begin
      Assign(Archivo,Nombre);
      {$I-} Reset( Archivo ); {$I+}
      If ( IoResult <> 0 ) then
        Begin
          Error;
          Exit;
        End;
    Window(7,8,44,23);
    ClrScr;
    While not EOF( Archivo ) do
     Begin
       Read( Archivo,Car );
       Write(Car);
     End;
    Close( Archivo );
    Ch:=ReadKey;
    ClrScr;
    Window(3,2,78,24);
  End;


{************************** PON OPCIONES **************************}
{Procedimiento que pone en pantalla las opciones que se tendrán
en el programa}

  Procedure PonOpciones;
   Var
    i:Byte;
```

```
Begin
  ClrScr;
  TextColor( Yellow );
  Escribe(19,1,'Universidad Nacional Autónoma de México');
  Escribe(27,2,'Facultad de Ciencias');
  TextColor( LightCyan );
  GotoXY(9,4);Write('                              ');
  GotoXY(9,5);Write('  INFORMACION SOBRE           ');
  GotoXY(9,6);Write('                              ');
  GotoXY(30,5);
  For i:=1 to 18 do
    Write('-');
  Escribe(53,8,'P R O G R A M A');
  Escribe(53,12,'C O R T A D U R A S');
  Escribe(53,16,'P R O G R A M A C I O N');
  Escribe(53,17,'E N T E R A');
  Escribe(53,20,'S A L I R');
  Marco(4,8,46,24);
  Window(3,2,78,24);
End;
```

```
{**************************** ESCOGE ****************************}
{Procedimiento que pinta al indicador en el lugar especificado
por la variable "Renglon" en el color especificado por la variable
"Color"}

  Procedure  Escoge(Renglon,Color: Byte);

   Const

   Fle = chr(16);
   Ver = chr(179);
   Hor = chr(196);
   EsqSup = chr(191);
   EsqInf = chr(192);
     Ren = 8;
     Col = 43;

   Var
     i,j:Byte;

 Begin
    TextColor(Color);
    GotoXY(Col,Ren);
    For i:=1 to 3 do
      Write(Hor);
    Write(EsqSup);
    For i:=(Ren + 1) to Renglon do
      Escribe(Col+3,i,Ver);
    Escribe(Col+3,Renglon,EsqInf);
    For i:=1 to 5 do
      Write(Hor);
    Write(Fle);
 End;
```

```
{************************** MOVER **************************}
 {Procedimiento que detecta la tecla presionado y de acuerdo a
ésta efectúa la opción deseada, ya sea moverse hacia arriba o
hacia abajo, o bien ejecutar al procedimiento "LeeArchivo"}

  Procedure Mover;

    Const

     ENTER = #13;
     ARRIBA = #72;
     ABAJO = #80;

   Begin
   Repeat
    Ch:=ReadKey;
    If ( Ch = #0 ) then
      Begin
       Ch:=ReadKey;
        Case Ch of
          ARRIBA : Begin
                    If (Pos > 8) then
                      Begin
                        Escoge(Pos,1);
                        Pos:=Pos-4;
                        Escoge(Pos,11);
                      End
                   End;
          ABAJO :  Begin
                    If (Pos < 19) then
                      Begin
                        Escoge(Pos,1);
                        Pos:=Pos+4;
                        Escoge(Pos,11);
                      End
                   End;
             End;   { Case }
      End
    Else
      Begin
       If (Ch = ENTER) then
         Begin
          Case Pos of
            8 : LeeArchivo('Presenta.doc');
           12 : LeeArchivo('Cortadur.doc');
           16 : LeeArchivo('Entera.doc');
           20 : Begin
                  Window(1,1,80,25);
                  TextColor(White);
                  TextBackground(Black);
                  ClrScr;
                  Halt;
                End;
          End; { Case }
         End;
```

```
    End;
  Until Ch='/';
End;


{---------------------- PROGRAMA PRINCIPAL ----------------------}

Begin

 Repeat
   ClrScr;
   TextColor( LightCyan );
   TextBackground( Blue );
   Marco(1,1,80,25);
   PonOpciones;
   Pos:=0;
   Escoge(Pos,11);
   Mover;
 Until ( Ch = 'M' );
End.
```

CAPITULO 4

ANALISIS COMPUTACIONAL DEL SISTEMA

El presente capítulo, más que un análisis pretende ser un bosquejo más detallado de la implementación del sistema; si bien es importante que el programa realice las funciones que se propone, también lo es la forma en que las realiza. En la práctica, al manejar programas de gran tamaño, este es un punto crítico ya que la eficiencia con que se haga determinado procedimiento puede influir directamente en el tiempo y costo empleado por el sistema.

El sistema implementado no presenta el problema de escasez de memoria; en parte por el tipo de algoritmos implementados (que no requieren de almacenar gran cantidad de valores o datos) y en parte por el objetivo del programa (que no pretende resolver problemas de gran tamaño), además de que se considera que su aplicación se hará en microcomputadoras con por lo menos 256 Kilobytes de memoria RAM[1]; sin embargo, es deseable el optimizar la forma en que se emplean los recursos de la máquina dentro del sistema.

Antes de comenzar, es necesario el comentar que el programa está diseñado de forma tal que no aprovecha facilidades particulares del lenguaje (Pascal), o del compilador (TurboPascal Ver.5.0) salvo el caso de la modularización del programa[2]; en los demás casos se usan funciones que de no ser

---

[1] RAM = Random Access Memory

[2] Lenguajes como C y Modula 2, además de otros compiladores también

intrínsecas, se pueden simular fácilmente en otro lenguaje, razón por la cual es factible la implementación del sistema en otros lenguajes de programación sin necesidad de grandes cambios.

4.1 VARIABLES Y CONSTANTES

A pesar de que el sistema no es "muy grande", un programa con sus dimensiones y con la cantidad de procedimientos que implementa, utiliza un número considerable de variables y constantes que deben manejarse eficientemente; uno de los puntos de importancia aquí, es el de declarar a una variable *local* o *global*; una variable local se reconoce sólo dentro del procedimiento donde se declara mientras que una variable global se reconoce en todo el sistema; al tener la facilidad de la modularización se tiene una subdivisión dentro de las variables globales, ya que habrá algunas que serán globales dentro de la unidad donde se usan y declaran, y habrá otras que serán reconocidas en todo el sistema; además de esta diferencia que afecta la estructura del programa, existe otra ligada a ésta; si una variable se declara localmente, cada vez que se entre al procedimiento donde se utiliza se le creará espacio en memoria, para después eliminar este espacio al salir del procedimiento; por esta razón no es deseable el declarar globalmente variables que se usarán poco y que ocuparán espacio en memoria durante todo el tiempo que el sistema esté activo; por ende, es conveniente que aquellas variables que sean importantes o que se utilicen mucho se declaren como globales y aquellas de poco uso o importancia se declaren como locales.

Haciendo a un lado las variables globales dentro de las unidades donde se usan, todo el sistema gira alrededor de las siguientes variables que se declaran globalmente en la unidad "Global":

Max_Reng :Número total de renglones utilizados por el sistema

---

ofrecen esta facilidad

Max_Col : Número total de columnas utilizadas por el sistema

Term_Indep : Vector de términos independientes

Matriz : Matriz de coeficientes incluyendo a la función objetivo

Con el fin de ahorrar memoria, todo el sistema trabaja sobre la misma variable Matriz, desde que se leen los datos de entrada, hasta que se presentan los resultados; las variables *Max_Reng* y *Max_Col* son de importancia ya que todos los recorridos efectuados dentro de la matriz se harán hasta estos límites y no hasta los límites máximos de la variable; cabe recordar que en el lenguaje Pascal, no se pueden declarar arreglos dinámicos, por lo que al declararlos, se les tiene que asignar el número máximo de lugares que podran utilizar; al ser la variable Matriz un arreglo bidimensional, es necesario el asignarle sus valores máximos establecidos por las constantes *TOTREN* y *TOTCOL* con valores de 20 cada uno, lo que implica que se tendrá una matriz de 20 X 20 (400 casillas); esto es sin duda una pérdida de espacio; suponiendo que se resuelva un problema de 3 restricciones y 2 variables, se utilizarían a lo mas 10 lugares incluyendo posibles variables de holgura y artificiales, lo cual implica que se están desperdiciando 390 lugares, pero en caso de reducir las dimensiones de la matriz, se reduciría también el tamaño de los problemas que podrían ser resueltos. Además de estas variables globales, existen otras cuyo fin es el de manejar la presentación de pantallas y menús y algunas otras booleanas que sirven para controlar el desarrollo de los algoritmos.

El resto de las variables se declaran como locales tratando de optimizar código en algunos casos como en el de los contadores; en lugar de declararlos de tipo "Integer" con un rango de -32768 a 32767 ocupando 16 bits, se declaran de tipo "byte" con un rango de 0 a 255 y ocupando sólo 8 bits; si bien esto puede paracer un detalle sin importancia, es innecesario el declarar una variable de tipo Integer cuando los valores que tomarán serán en un rango de 0 a 20.

En lo referente a las constantes, se usan de forma que al cambiar el programa sólo se modifiquen los valores asignados a éstas y se puedan realizar cambios sin restructurar todo el programa; además de las constantes usadas para las dimensiones de la variable Matriz comentadas anteriormente, se utilizan constantes para controlar la entrada de datos y detectar las teclas presionadas al elegir opciones en los menús y una de particular importancia que se comentará a continuación.

4.2 PRESICION

Debido al tipo de sistema que se implementa, la presición resulta ser un aspecto de importancia suprema; además como los algoritmos implementados funcionan de forma iterativa, esta importancia aumenta al tomar en cuenta que un leve error de presición puede irse "acarreando" y agrandando a medida que se resuelve un problema. La principal dificultad en este aspecto es el de truncamiento de dígitos; ya que sin importar el tipo de computadora con la que se trabaje, siempre será un obstáculo en mayor o menor grado; para ejemplificar esta dificultad, considérese el siguiente ejemplo:

Efectuar la siguiente operación

$$(1/3 + 2/3) - 1$$

resolviendo ...

$$3/3 - 1 = 1 - 1 = 0$$

Pero para la máquina, suponiendo una presición de 2 decimales, esta operación sería:

$$(0.33 + 0.66) - 1$$

resolviendo ...

$$0.99 - 1 = -0.01$$

Si bien la diferencia parece ser pequeña, al preguntar el algoritmo si el elemento resultante es cero, la respuesta será negativa.

Para evitar el problema anterior, se trabajará considerando a todos los elementos de tipo "Real" (que es el tipo estándar para elementos reales en Pascal) que tiene un rango de 2.9×10^{-39} a 1.7×10^{38} con 12 cifras decimales significativas y ocupando 6 bytes[3] y añadiendo una constante denominada "Tolerancia" con un valor de 0.00005; utilizando esta tolerancia, se implementa un procedimiento llamado "Redondea" el cual se ejecuta despues de cada iteración; dicho procedimiento reajusta los coeficientes de la matriz y del vector de términos independientes en el caso de que se considere que el valor que tiene no es un valor correcto sino "basura" provocada por la imprecisión de los cálculos de la máquina; el procedimiento funciona de la forma siguiente:

- Se redondea el número que se está analizando y se le asigna este valor a una variable auxiliar.

- Se realiza la resta entre el número original y el valor redondeado.

- Se compara el valor absoluto de la diferencia obtenida contra el valor de la tolerancia; si la diferencia es mayor a la tolerancia, se considera que el valor es válido; en caso contrario se considera que el valor es "basura" y se reajusta el número asignando a su lugar el valor redondeado.

Este procedimiento tiene una desventaja evidente que es el de que sólo corrige los errores cerca de valores enteros, pero como los problemas mayores de los algoritmos implementados surgen al tratar

---

[3] TurboPascal tambien ofrece otros tipos como el "extended" con un rango de 3.4×10^{-4932} a 1.1×10^{4932} con 20 cifras decimales significativas y ocupando 10 bytes

de identificar valores Q o enteros, este procedimiento los evita confiándose en la presición de la máquina al tratar con valores fraccionales; por lo anterior, este procedimiento resulta bastante eficiente y "barato", ya que además de cumplir con su cometido, sólo revisa la tabla una vez y sólo en caso de ser necesario efectúa el ajuste requerido evitando pérdida de espacio y de tiempo.

Una parte medular del procedimiento es la constante Tolerancia la cual se definió empíricamente; al definir su valor se pasó por varios puntos de interés que a continuación se comentan:

- El valor de la Tolerancia va de acuerdo a la presición que se tenga en la máquina y al tipo de algoritmos que se implementen.

- Una cota máxima para la Tolerancia no debería ser mayor a 0.005 ya que estos valores se pueden encontrar en problemas ordinarios.

- Una Tolerancia demasiado pequeña podría dejar pasar "basura" como valores válidos.

- El valor fijado para la Tolerancia debe estar fuera del control del usuario, a menos que esta sea de interés dentro de las opciones del sistema y no sólo se utilice como un elemento de control.

- En este caso particular, la Tolerancia con valor de 0.00005 permite el tener valores de hasta 1/20000; valores más pequeños no son comunes en el tipo de problemas que se intenta resolver por lo cual se considerarán como "Basura".

Como se comentó anteriormente, esta no es la única ni la mejor forma de mantener la presición de los datos, pero resulta ventajosa considerando aspectos tanto matemáticos como computacionales; por ejemplo, considérese el siguiente ajuste posible:

Manejando a todos los elementos de la tabla de la forma m/n, se puede tener una presición total presentando los resultados en este formato o haciendo la división hasta el final del proceso lo cual disminuiría grandemente el error por truncamiento; sin embargo, tal procedimiento sería ineficiente por los siguientes puntos:

- El manejar dos valores, m y n, implica el manejo de dos tablas con el consiguiente aumento en lugares de memoria (de 400 a 800 en el caso particular del sistema).

- Se podría llegar a tener valores del tipo de 3267/3287 lo cual sería innecesario y en un caso extremo, podría hacer que los valores quedaran fuera del rango de la máquina.

- En caso de querer evitar el punto anterior, se tendrían que reducir valores, proceso que se tendría que realizar después de cada iteración además de que implicaría la necesidad de implementar un procedimiento que obtuviera el máximo común divisor de los dos elementos.

- De lo anterior, si bien el procedimiento usado no brinda una presición total, resulta ser bastante eficiente además de proporcionar una presición satisfactoria.

4.3 MANEJO DE INFORMACION

Una parte no menos importante pero a menudo menos cuidada, es la de que el programa tenga una buena secuencia de ejecución y que no "brinque" de un lado a otro en busca del siguiente procedimiento a ejecutar; este aspecto, ya en un nivel más alto, es de gran importancia ya que es necesario el optimizar el espacio de memoria; esto entre otras cosas, permite el uso de *Overlays* los cuales funcionan estructuralmente igual que una unidad (En TurboPascal) con la diferencia de que sólo se cargan en memoria en el momento en el cual se estan ejecutando, de forma que al no usarse no estan en memoria, ahorrandose consecuentemente espacio

que puede ser aprovechado para otros fines; como en el presente sistema no es necesario, se implementan unidades en lugar de overlays, pero aún así, es conveniente el tener un programa bien estructurado.

En este respecto, el sistema aprovecha la facilidad de modularización para separar en unidades procedimientos relacionados entre sí, con el objeto de no usar todas las unidades al mismo tiempo; para aclarar lo anterior, considérese el dezarrollo de un problema dentro del sistema. De forma general, un problema pasa por 3 fases:

- Edición

- Resolución

- Presentación de Resultados

Si bien el sistema permite que los dos últimos puntos se intercalen de forma que se pueda observar los cambios que va sufriendo el problema, no existe una interacción directa entre los dos puntos por lo cual no se efectúan al mismo tiempo.

Las 3 fases usan las siguientes unidades:

Fase	Unidad
Edición	Global, Detecta
Resolución	Global, Ajusta, Ciclico, Discreto
Presentación de Resultados	Global, Discreto, Impresor

En el caso de la resolución del problema, la unidad *Ajusta* se utiliza antes de comenzar con alguno de los dos algoritmos de Programación Entera por lo que al terminar el ajuste, deja de operar; además, solo una de las 2 unidades (*Discreto* o *Ciclico*) se usa; esto dependiendo del algoritmo previamente seleccionado. En la presentación de resultados, en caso de que ésta sea a pantalla, se usa un procedimiento de la unidad *Discreto*, y en caso de que sea a papel, se utiliza la unidad *Impresor*.

Como se puede observar, la unidad *Global* es la única unidad que está cargada todo el tiempo mientras que tanto la unidad *Tesis*, que coordina a todas las demás, como la unidad *Informa*, se cargan en los intermedios entre fase y fase; siendo esto opcional para la unidad *Informa* (al elegir la opción "Ayuda" del menú principal).

CAPITULO 5

LISTADO DEL PROGRAMA

5.1 UNIDAD GLOBAL

```
Unit Global;

Interface

 Const

    TOTREN      = 20;
    TOTCOL      = 20;
    F1          = #59;
    F2          = #60;
    F3          = #61;
    F4          = #62;
    F5          = #63;
    ENTER       = #13;
    DER         = #77;
    IZQ         = #75;
    ABAJO       = #80;
    ARRIBA      = #72;
    Tolerancia = 0.00005;

 Type

    Elemento  = String [5];
    Tabla     = Array [1..TOTREN,1..TOTCOL] of Real;
    Vector    = Array [1..TOTREN] of Real;
    Linea     = String [80];
    Letrero   = String [28];
    Respuesta = String [4];
    Registro  = RECORD
                  linea : Byte;
                  multiplo : Real;
                  End;

 Var

    Matriz              : Tabla;
    Term_Indep          : Vector;
    Total, Max_Reng     : Byte;
    Extra, Max_Col      : Byte;
    Func_Obj, NumRestr  : Byte;
```

```
    ContArt,ContHol     : Byte;
    Eme                 : Real;
    Slack, NoBas, Vars  : Array [1..TOTCOL] of Elemento;
    Continuo            : Array [1..TOTREN] of Boolean;
    Opcion              : Array[1..8] of Letrero;
    Default             : Array[1..8]of Boolean;
    Arr                 : Array[1..8]of Respuesta;
    Artificial          : Array [1..TOTCOL] of Byte;
    Cuadro              : Array [1..TOTREN] of Registro;
    Respaldo            : Array [1..TOTREN] of Linea;
    SinSolucion         : Boolean;
    UsandoEme           : Boolean;
    HayArtificial       : Boolean;
    Nuevo               : Boolean;
```

{

EXPLICACION DE VARIABLES GLOBALES

```
    Matriz              : Matriz de Coeficientes
    Term_Indep          : Vector de Terminos Independientes
    Total, Max_Reng     : Maximo de Variables, Maximo de Renglones
    Extra, Max_Col      : Variables de holgura, Maximo de Columnas
    Func_Obj, NumRestr  : Funcion objetivo, Numero de Restricciones
    ContArt,ContHol     : Contadores de variables de holgura y
                          artificiales
    Eme                 : Valor de " M "
    Slack, NoBas, Vars  : Tablas para registro de variables;
    Opcion              : Menu de opciones del sistema
    Default             : Vector de opciones para la resolucion de
                          problemas
    Arr                 : Opciones para la resolucion de problemas
    Artificial          : Vector de variables artificiales
    Cuadro              : Vector para acualizar la funcion objetivo
                          en el metodo de las dos fases
    Respaldo            : Arreglo para guardar las restricciones
    SinSolucion         : Determina si un problema tiene solucion
    UsandoEme           : Determina si se esta usando " M "
    HayArtificial       : Determina si hay variables artificiales
    Nuevo               : Determina si el problema es nuevo
```

}

```
Procedure ErrorCi:integer);
Procedure MarcaCs:string);
Procedure NormalCs:string);
Procedure IluminaCs:string);
Procedure EncabezadoCi,j:integer);
Procedure PonHora;
Procedure Sistema;
Procedure Tecla;
Function Optima  : Boolean;
Function Factible: Boolean;
Function Entera  : Boolean;


Implementation
```

```
Uses Crt,Dos;


{************************ PON HORA ********************************}
{ Procedimiento que pone la hora del sistema}

Procedure PonHora;
Var
   Hora,Min,Seg,Mic : Word;
   HoraSist,Aux : String[5];

Begin
   HoraSist[0]:=#0;
   Aux[0]:=#0;
   GetTime(Hora,Min,Seg,Mic);
   TextBackground(Cyan);
   TextColor(Black);
   If  (Hora < 10) then
     HoraSist:=HoraSist + '0';
   Str(Hora,Aux);
   HoraSist:=HoraSist + Aux;
   HoraSist:=HoraSist + ':';
   If (Min < 10) then
     HoraSist:=HoraSist + '0';
   Str(Min,Aux);
   HoraSist:=HoraSist + Aux;
   Gotoxy(70,4);
   Write(HoraSist);
   Delay(1500);
   Gotoxy(70,4);
   Write('      ');
End;


{************************ SISTEMA *********************************}
{  Procedimiento que informa sobre la version del sistema
 operativo y sobre la memoria RAM disponible}

Procedure Sistema;
Var
   Ver : Word;
   Ch : Char;

Begin
   Ver:= DosVersion;
   TextBackground(Black);
   TextColor(Black);
   Window(4,6,76,23);
   ClrScr;
   Window(20,10,60,19);
   TextBackground(Green);
   ClrScr;
   Gotoxy(14,2);
   Write('S I S T E M A');
   Gotoxy(4,4);
```

```
   WriteC'Versi"n del Sistema Operativo : ',LoCVer),'.',HiCVer));
   GotoXY(8,5);
   WriteCCDiskSizeC0)):8,' Bytes totales en disco');
   GotoXY(8,6);
   WriteCCDiskFreeC0)):8,' Bytes libres en disco');
   GotoXY(8,7);
   WriteCCMemAvail):8,' Bytes libres en Memoria');
   Ch:=ReadKey;
   TextBackground(Black);
   ClrScr;
End;


{########################## TECLA ##############################}
{ Procedimiento que espera a que se presione una tecla para
poder continuar con le ejecucion del programa }

Procedure Tecla;

Var

   Ch : Char;

Begin

   Ch:= ReadKey;
   If ( Ch = #0 ) then
     Ch:= ReadKey;

End;


{########################## ERROR ##############################}

Procedure Error;

Begin
  Writeln;
  WritelnC'                                    ');
  WritelnC'          E R R O R                 ');
  WritelnC'                                    ');
  WritelnC'                                    ');
  Case i of

   2: WritelnC'     Expresion invalida         ');
   3: WritelnC'     Restriccion Invalida       ');
   4: WritelnC'  Funcion Objetivo Invalida     ');
   5: WritelnC'    Valor de llamda Invalido    ');
   6: WritelnC'    Elemento Pivote Invalido    ');
   7: WritelnC'  Renglon Candidato Invalido    ');
   8: WritelnC'  Formato Erroneo A.Discreto    ');
   9: WritelnC'  Soluci"n Inicial no "ptima    ');
  10: WritelnC'  Factores demasiado grandes    ');
  11: WritelnC'       Error en Factores        ');
```

```
    12: WritelnC' |  Demasiadas Restricciones  |');
  End;
  WritelnC' ------------------------------------');
  Tecla;
End;


{************************** NORMAL ********************************}
{ Imprime en pantalla una cadena de caracteres iluminando de
 color blanco la primera letra y de color negro las demas}

Procedure NormalCs:string);

Var
 i : Byte;

Begin
  TextbackgroundCCyan);
  TextcolorCwhite);
  WriteCs[1]);
  TextcolorCblack);
  For i:=2 to lengthCs) do
    WriteCs[i]);
End;


{****************************** MARCA *****************************}
{ Imprime en pantalla una cadena de caracteres en color rojo
  sobre fondo negro}

Procedure MarcaCs:string);

Begin
  TextcolorCred);
  TextbackgroundCblack);
  WriteCs);
  TextcolorCblack);
  TextbackgroundCCyan);
End;


{************************** ILUMINA *******************************}
{ Imprime en pantalla una cadena de caracteres en color blanco
  sobre fondo cyan}

Procedure IluminaCs:string);

Begin
  TextcolorCWhite);
  TextbackgroundCCyan);
  WriteCs);
  TextcolorCWhite);
  TextbackgroundCBlack);
End;
```

```
{**********************  ENCABEZADO  **************************}
{ Registra como variable no basica la que acaba de salir de la
  base}
 Procedure Encabezado(i,j:integer);

 Begin
    NoBasi[j]:= Vars[i];
 End;


{**************************  FACTIBLE  ***************************}
{ Funcion que se encarga de detectar si la base actual es
factible regresando verdadero si lo es y falso en caso contrario}

Function Factible : Boolean;

Var
  i : Byte;

Begin

  Factible:= True;
  For i:=2 to ( Max_Reng - 1 ) do
    Begin
      If ( Term_Indep[i] < 0 ) then
       Begin
         Factible:= False;
         Exit;
       End;
    End;
End;


{**************************  ENTERA  ***************************}
{ Funcion que se encarga de detectar si la base actual es entera
regresando verdadero si lo es y falso en caso contrario}

Function Entera : Boolean;

Var
  i,j:Byte;

Begin

  Entera:= True;
  For i:=1 to Max_Reng do
    Begin
    For j:=1 to Max_Col do
      Begin
       If ( Frac( Matriz[i,j] ) <> 0 ) then
        Begin
```

```
            Entera:= False;
            Exit;
          End;
      End;
    End;
End;


 {********************************* OPTIMA ********************************}
{ Funcion que se encarga de detectar si la base actual es optima
regresando verdadero si lo es y falso en caso contrario}

Function Optima : Boolean;

Var
  i : Byte;

Begin

  Optima:= True;
  For i:=1 to Max_Col do
    Begin
      If ( Matriz[1,i] > 0 ) then
       Begin
         Optima:= False;
         Exit;
       End;
    End;
End;


 End.
```

5.2 UNIDAD DETECTA

```
 Unit Detecta;

Interface

Uses Global;

Procedure  Principal(Var Columnas : Byte;Var Correcto:Boolean);

Implementation

Uses Crt;

{********************* D E C L A R A C I O N E S *********************}
```

```
Var
 j,Max_Col,Cont : Integer;
 Code,Extra     : Integer;
 Valor          : Real;
 Cadena         : String[80];
 Variable       : Elemento;
 Numero         : String[5];
 Des,cad        : String[2];
 Obj            : String[10];
 c,Ch           : Char;
 Incorrecto,Ya  : Boolean;
 SeModifico     : Boolean;
 SalirEditor    : Boolean;


(********************* I N I C I A L I Z A **************************)
(Procedimiento que se encarga de Inicializar todas las variables
y contadores del programa incluyendo la matriz de coeficientes y
el vector de terminos independientes; esto se realiza cada vez
que se entra al editor del sistema)

Procedure Inicializa;

Var
 i,j : Byte;

Begin
   If Nuevo then
       NumRestr :=1;
   Incorrecto :=False;
   HayArtificial :=False;
   UsandoEme:=False;
   ContArt:=0;
   ContHol:=0;
   Eme:=0;
   Cadena[0]:=#1;
   variable[0]:=#0;
   Numero[0]:=#0;
   Des[0]:=#0;
   max_reng:=1;
   Total:=0;
   Cont:=1;
   Max_Col:=0;
   Extra:=0;
   For i:=1 to TOTREN do
    Continuo[i]:=False;
   For i:=1 to TOTREN do
     For j:=1 to TOTCOL do
        matriz[i,j]:=0;
   For i:=1 to TOTREN do
    Begin
        Term_Indep[i]:=0;
    End;
   For i:=1 to TOTCOL do
       Begin
```

```
          vars[i,j]:=' ';
          nobas[i,j]:=' ';
          slack[i,j]:=' ';
        End;

End;


{************************** C O P I A S L K **************}
{Detectada una variable de holgura se registra en la tabla de
 estas variables}

Procedure CopiaSlk(ind:Integer;st:elemento);

Begin
   slack[ind]:=st;
End;


{************************** C O P I A E N C **************}
{Registra como variable no basica la variable que recibe como
parametro}

Procedure CopiaEnc(ind:Integer;st:Elemento);

Begin
   Nobas[ind]:=st;
End;


{************************** R E N O M B R A **************}
{Para una mejor visualizacion de las variables para el usuario
se renombran estas como ART para artificiales, HOL para holgura
y Z para la funcion objetivo a optimizar}

Procedure Renombra;

Var
  i   : Byte;
  Cad : String[1],

Begin
 For i:=1 to (Total + 1) do
  Begin
   If Vars[i,1] = '@' then
      Begin
       HayArtificial:=True;
       ContArt:=ContArt + 1;
       Artificial[ContArt]:=i;
       Str(ContArt,Cad);
       Vars[i]:='ART'+ Cad;
      End
   Else
```

```
    If Vars[i,1] = '$' then
       Begin
        ContHol:=ContHol + 1;
        Str(ContHol,Cad);
        Vars[i]:='HOL'+ Cad;
       End
    Else
    If Vars[i,1] = 'f' then
       Vars[i]:='M Z M';
  End;
End;


{*********************** C  O  P  I  A **********************}
{Registra en la tabla de variables la variable que recibe}

Procedure Copia(ind:Integer;st:Elemento);

Begin
   Vars[ind]:=st;
End;


{********************* I N S E R T A ************************}
{Anade a la variable Numero el siguiente caracter de la cadena}

Procedure Inserta;

 Begin
   Numero:=Numero+Cadena[Cont];
 End;


 {********************* A G R E G A ************************}
 {Agrega a la matriz de coeficientes el numero que se acaba de
 recibir}

 Procedure AgregaCren,j:Integer);

 Begin
    variable[0]:=#0;
    val(Numero,valor,code);
    Matriz[ren,j]:=valor;
    Numero[0]:=#0;
 End;


{********************* B U S C A ************************}
{Busca en la tabla de variables la nueva variable y regresa en
que posicion de la tabla la encontro o 0 en caso negativo}

Procedure BuscaCvar j:Integer);
```

```
Var
  h : Integer;

Begin
       h:=1;
       ya:=False;
        Repeat
        If Nobas[h]=variable then
         Begin
            j:=h;
            ya:=true;
         End
        Else
          h:= h + 1;
        Until (ya=true)or(h)(Total - Extra));
       If  h > (Total - Extra) then
         Begin
            Total:=Total+1;
            Max_Col:= Max_Col + 1;
            j:=Max_Col;
            Copiaenc(j,variable);
            Copia(Total,Variable);
         End;
End;


{************************** L L E N A **************************}
{Llena la matriz de coeficientes}

Procedure llena(ren:Integer);

Var
  j : Integer;

  Begin
    If variable[0]<>'0' then
     Begin
       Busca(j);
       AgregaCren,j);
        If Cadena[Cont]='-' then
        Inserta;
     End
    Else
     Begin
      If Cadena[Cont]='-' then
      Inserta
     End;
  End;


 {************************** M E T E **************************}
 {En caso de detectar un caracter numerico lo agrega a "Variable"
 si es parte de una variable o llama a "Inserta" si se trata de
```

```
un coeficiente)

Procedure Mete;

Begin
   If Variable[0]>#0 then
      Variable:=Variable+Upcase(Cadena[Cont])
   Else
     Inserta;
End;


(##########  L A D O   D E R E C H O  ##########################)
(Revisa el lado derecho de la desigualdad anadiendo las
variables necesarias y acomodando los coeficientes en la matriz)

Procedure Lado_Derecho(ren:Integer;var invierte:boolean);

Var
 con,j : Integer;

 Begin
    invierte:=False;
    Str(Extra,cad);
    If Des[0]=#0 then
      Begin
         Busca(j);
         Agrega(ren,j);
         Des:=Des+Cadena[Cont];
         Total:= Total + 1;
         Extra:= Extra + 1;
         If Des='=' then
           Begin
             variable:='@'+cad;
             Copiaslk(extra,variable);
             Copia(Total,variable);
             variable[0]:=#0;
             Invierte:=True;
           End;
      End
    Else
       Begin
            If Des='<' then              { Desigualdad (<= )
              Begin
                 invierte:=True;
              End;
            Variable:='$'+cad;
            Copiaslk(extra,variable);
            Copia(Total,variable);
            Variable[0]:=#0;
       End;
            If Invierte = True then
              Begin
                For  con:=1 to Max_Col do
                  Matriz[ren,con]:=-Matriz[ren,con];
```

```
            End;
  End;


{********************  A U M E N T A  ***************************}
{En caso de detectar un caracter entre A y Z lo aumenta al nombre
de la variable}

Procedure Aumenta;

  Begin
   If (Des[0]<>#0) then
     Begin
      Incorrecto:=True;
      Exit;
     End;
   If (variable[0]=#0)and((Numero[0]=#0)or(Numero='-')) then
    Numero:=Numero+'1';
   variable:=variable+Upcase(Cadena[Cont]);
  End;


{********************  U L T I M O   C A R  *********************}
{Revisa al ultimo caracter de la cadena de entrada y dependiendo
de este da por terminada la edicion o bien analiza la desigualdad}

 Procedure Ultimo_car(ren:Integer);

  Var
    j : Integer;

     Begin
       If Des[0]=#0 then
        Begin
          variable:=variable+Upcase(Cadena[Cont]);
          Busca(j);
          AgregaCren,j);
        End
       Else
        Begin
         Inserta;
         val(Numero,valor,code);
         If (Des='>') then
           Term_Indep[ren]:=-valor
         Else
           Term_Indep[ren]:=valor;
         Numero[0]:=#0;
         Des[0]:=#0;
        End;
     End;


{********************  R E V I S A  *****************************}
```

```
{Revisa la cadena de entrada, ya sea en la funcion objetivo o
en las restricciones detectando cualquier error que impida
funcionar adecuadamente al sistema reportandolo e interrumpiendo
la edicion del problema}

Procedure Revisa(i:Integer;var correcto:Boolean);

 Var
  Operador   : Boolean;
  Igual      : Integer;
  Subcadena  : String[10];
  Cont,j     : Byte;
  NoVacia    : Boolean;


 Begin
  Correcto:=True;
  Operador:=False;
  NoVacia:=False;
  For j:=1 to Length(Cadena) do
    Begin
     If Cadena[j] <>' ' then
       NoVacia:=True;
    End;
    If NoVacia = False then
     Begin
        Correcto:=False;
        Error(2);
        Exit;
     End;
  If i=1 then
     Begin
        For j:=1 to Length(Cadena) do
          Begin
            If Cadena[j] in ['=','>','<','!','@','$','*'] then
              Begin
               Error(4);
               Correcto:=False;
               Exit;
              End;
          End;
     End
  Else
     Begin
        If (Cadena = '#') then
         Exit;
        For j:=1 to Length(Cadena) do
          Begin
            If Cadena[j] in ['!','@','$','*'] then
              Begin
               Error(2);
               Correcto:=False;
               Exit;
              End
            Else
            If Cadena[j] in ['<','>','='] then
```

```
{Revisa la cadena de entrada, ya sea en la funcion objetivo o
en las restricciones detectando cualquier error que impida
funcionar adecuadamente al sistema reportandolo e interrumpiendo
la edicion del problema}

Procedure Revisa(i:Integer;var correcto:Boolean);

 Var
  Operador   : Boolean;
  Igual      : Integer;
  Subcadena  : String[10];
  Cont,j     : Byte;
  NoVacia    : Boolean;


 Begin
  Correcto:=True;
  Operador:=False;
  NoVacia:=False;
  For j:=1 to Length(Cadena) do
    Begin
     If Cadena[j] <>' ' then
       NoVacia:=True;
    End;
    If NoVacia = False then
     Begin
        Correcto:=False;
        Error(2);
        Exit;
     End;
 If i=1 then
     Begin
        For j:=1 to Length(Cadena) do
          Begin
            If Cadena[j] in ['=','>','<','!','@','$','#'] then
               Begin
                Error(4);
                Correcto:=False;
                Exit;
               End;
          End;
     End
 Else
     Begin
        If (Cadena = '#') then
         Exit;
        For j:=1 to Length(Cadena) do
          Begin
            If Cadena[j] in ['!','@','$','#'] then
               Begin
                Error(2);
                Correcto:=False;
                Exit;
               End
            Else
            If Cadena[j] in ['<','>','='] then
```

```
            Begin
              Operador:=True;
              If ( Cadena[j] = '=' ) then
                Igual:=j;
            End;
        End;
      If ( Operador = False ) then
        Begin
          Error(3);
          Correcto:=False;
        End
      Else
        Begin
          Subcadena[0]:=#0;
          For Cont:=(Igual + 1) to Length(Cadena) do
          Subcadena:=Subcadena+Cadena[Cont];
          Val(Subcadena,Valor,Code);
          If ( Code <> 0 ) then
            Begin
              Error(3);
              Correcto:=False;
            End;
        End;
    End;
End;


{¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤  A J U S T A  ¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤¤}
{Ajusta las variables ordenandolas y llamando al procedimiento
"Renombra"}

Procedure Ajusta;

Var
  c,d : Integer;

Begin

    For d:=1 to (Total-extra) do
      Begin
        Matriz[Max_Reng,d]:=1;
        inc (Max_Reng);
      End;
    c:=1;
    Vars[1]:='1';
    For d:=1 to Total do
     Begin
      If d <= Extra then
        Vars[d+1]:=Slack[d]
      Else
        Begin
          vars[d+1]:=Nobas[c];
          c:= c + 1;
        End;
     End;
```

```
    Renombra;
    If ( Func_Obj = 2 ) then
      Begin
        For c:=1 to Max_Col do
          Matriz[1,c]:= -Matriz[1,c];
      End;
 End;


{************************** ERRMOD **************************}
{Procedimiento que reporta error al intentar borrar la
funcion objetivo}

Procedure ErrMod ( Tipo : Byte);

Begin
  TextColor(White);
  TextBackground(Red);
  Window(4,8,76,23);
  Gotoxy(31,12);Write('┌──────── ERROR ────────┐');
  Gotoxy(31,13);Write('│                       │');
  If Tipo = 1 then
    Begin
       Gotoxy(31,14);Write('│ No se puede borrar    │');
       Gotoxy(31,15);Write('│ la funcion objetivo   │');
    End
  Else
    Begin
       Gotoxy(31,14);Write('│ No se puede borrar    │');
       Gotoxy(31,15);Write('│ el caracter  " # "    │');
    End;
  Gotoxy(31,16);Write('│                       │');
  Gotoxy(31,17);Write('└───────────────────────┘');
  Tecla;
  TextBackground(Black);
  Window(34,15,76,23);
  ClrScr;
  Window(4,8,53,23);
  TextColor(White);
End;


{************************ M O D I F I C A ************************}
{Procedimiento que permite el poder hacer modificaciones al
problema original, ya sea a$adiendo, quitando o modificando
restricciones}

Procedure Modifica;

Var

  LineaActual, i : Byte;
  Cadena         : String [80];

Begin
```

```
    SeModifico :=False;
    LineaActual :=1;
    ClrScr;
    Ilumina (Respaldo[1]);
    Writeln;
    For i:=2 to NumRestr do
      Writeln(Respaldo[i]);
    TextColor(Black);
    TextBackground(Green);
    GotoXY(51,2);Write('┌─── Comandos ──┐');
    GotoXY(51,3);Write('│ Borra Linea   F1 │');
    GotoXY(51,4);Write('│ Modifica Linea F2 │');
    GotoXY(51,5);Write('│ Anade Linea   F3 │');
    GotoXY(51,6);Write('│ Nuevo Problema F4 │');
    GotoXY(51,7);Write('│ Salir         F5 │');
    GotoXY(51,8);Write('└──────────────┘');
    Window(4,6,53,23);
    TextColor(White);
    TextBackground(Black);
    Repeat
      Ch :=ReadKey;
      If Ch = #0 then
        Begin
          Ch :=ReadKey;
          Case Ch of

        ARRIBA : Begin
                   GotoXY(1,LineaActual);
                   Write(Respaldo [LineaActual]);
                   If LineaActual = 1 then
                       LineaActual :=NumRestr
                   Else
                       Dec (LineaActual);
                   GotoXY(1,LineaActual);
                   Ilumina(Respaldo [LineaActual]);
                 End;

        ABAJO : Begin
                   GotoXY(1,LineaActual);
                   Write(Respaldo [LineaActual]);
                   If LineaActual = NumRestr then
                       LineaActual :=1
                   Else
                       Inc (LineaActual);
                   GotoXY(1,LineaActual);
                   Ilumina(Respaldo [LineaActual]);
                 End;

( Borra ) F1 : Begin
                   If LineaActual = 1 then
                     ErrMod ( 1 )
                   Else if LineaActual = NumRestr then
                     ErrMod ( 2 )
                   Else
                    Begin
                      DelLine;
```

```
                    Dec(NumRestr);
                    For i:=LineaActual to NumRestr  do
                        Respaldo [i] :=Respaldo [i + 1];
                    SeModifico :=True;
                    GotoxyC1,LineaActual);
                    IluminaCRespaldo [LineaActual]);
                  End;
                End;

(Modifica)F2 : Begin
                 If LineaActual = NumRestr then
                   ErrMod ( 2 )
                 Else
                  Begin
                    DelLine;
                    InsLine;
                    GotoxyC1,LineaActual);
                    ReadlnCCadena);
                    Respaldo[LineaActual] := Cadena;
                    GotoxyC1,LineaActual);
                    IluminaCRespaldo [LineaActual]);
                    SeModifico :=True;
                  End;
               End;

( Añade ) F3 : Begin
                 GotoxyC1,LineaActual);
                 WriteCRespaldo [LineaActual]);
                 LineaActual:=NumRestr;
                 GotoxyC1,LineaActual);
                 InsLine;
                 ReadlnCCadena);
                 Respaldo[NumRestr] := Cadena;
                 IncCNumRestr);
                 Respaldo[NumRestr] := '#';
                 GotoxyC1,LineaActual);
                 IluminaCRespaldo [LineaActual]);
                 SeModifico :=True;
               End;

( Nuevo ) F4 : Begin
                 ClrScr;
                 NumRestr :=1;
                 Nuevo:= True;
                 SeModifico :=False;
                 SalirEditor:=False;
                 Exit;
               End;

( Salir ) F8 : Begin
                 ClrScr;
                 If SeModifico then
                   Begin
                     GotoxyC20,7);
                     WriteC'ANALIZANDO EL PROBLEMA . . .');
                   End;
```

```
                SalirEditor :=True;
                Exit;
              End;
        End; { Case }
      End;
  Until Ch = '?';
End;


{**********************  D E T E C T O R  **********************}
{Es el Programa que controla toda la edicion del problema
leyendo del teclado una cadena de caracteres que despues
analiza con ayuda de los Procedimientos anteriores}

Procedure Detector(Var Correcto:Boolean);

Var

  invierte : Boolean;
  i        : Integer;

Begin

For i:=1 to TOTREN do
Begin
 If Nuevo then
   Begin
     ReadlnCCadena);
     Inc(NumRestr);
     Respaldo [NumRestr]:=Cadena;
   End
 Else
   Cadena:= Respaldo [i + 1];
 Revisa(i,Correcto);
 If (Correcto = False) then
     Exit;
 For Cont:=1 to length(Cadena) do
  Begin
    If Cont=length(Cadena) then
     Begin
       If Cadena[Cont]='#' then
        Begin
         If i < 3 then
           Begin
             Correcto := False;
             Error (12);
             Exit;
           End;
         Ajusta;
         Exit;
        End
       Else
        Begin
          Ultimo_car(max_reng);
          Max_reng:=max_reng + 1;
```

```
        End;
      End
     Else
      Begin
      Case Cadena[Cont] of

      '0','1','2','3','4','5','6','7','8','9','.':Mete;

      '-','+':llena(max_reng);
            ' ':      ;
      '>','<','=':Begin
                    Lado_derecho(Max_reng,Invierte);
                  End
       Else
        Aumenta;
     End; { case }
    End; { If }
   End; { For }
  If ( (i = 1) and (Nuevo) ) then
   Writeln('SUJETO A :');
  End; { For }
 End; { Detector }


{@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ PRINCIPAL @@@@@@@@@@@@@@@@@@@@@@@@@@@@}
{Procedimiento que se exporta y que llama al editor}

  Procedure Principal(Var Columnas : Byte;Var Correcto:Boolean);

 Var

   i : Byte;

 Begin
  Correcto:=True;
  Obj[0]:=#0;
  Textcolor(white);Textbackground(black);
  Clrscr;
  If Nuevo = False then
    Begin
      Modifica;
      If ( (SeModifico = False) and (SalirEditor) ) then
        Exit;
    End;
  Inicializa;
  If ( (SeModifico = False) or (Nuevo = True) ) then
    Begin
      Writeln('DAME EL PROBLEMA A RESOLVER :');
      Readln(Cadena);
      Respaldo [1]:= Cadena;
    End
  Else
    Cadena:= Respaldo [1];
   i:=1;
   Repeat
```

```
    Begin
     If Cadena[i]<>' ' then
        Obj:= Obj + UpCase(Cadena[i]);
     i:=i+i;
    End;
    Until i>(Length(Cadena));
    If   ((Obj='MAX')or(Obj='MIN')) then
     Begin
      If Obj='MAX' then
        Func_Obj:= 1
      Else
        Func_Obj:= 2;
     End
    Else
     Begin
      Error(4);
      Correcto:=False;
      Exit;
     End;
    Detector(Correcto);
    If Incorrecto then
      Begin
       Error(3);
       Correcto:=False;
      End;
    Columnas:=Max_Col;
    If Correcto then
      Nuevo:= False;
    End;
END.
```

5.3 UNIDAD DISCRETO

```
Unit Discreto;

Interface

Uses Global,Detecta;

Procedure AlgDiscreto(Veces:Integer;Si,Escoger,Muestra,
                      NvaRest:boolean);
Procedure Escoge_Columna( ren: Integer; Var indice:
                      Integer;Muestra:Boolean);
Procedure Escoge_Renglon(Var i: Integer);
Procedure PivoteaCr,c:Integer);
Procedure Imprime(Largo:Integer);
Procedure Redondea;
Procedure Primal;
Function HayEmeEnTabla : Boolean;

Implementation
```

```
Uses Crt;


Var
  w,x : Integer;
  Ch  : Char;



{************************** MAY LEX ****************************}
{Funcion que determina si la primer columna es mayor
lexicograficamente a la segunda}

Function May_Lex(d,e:Integer):Boolean;

Var

 c : Integer;

Begin

   May_Lex:=False;
   c:=1;
   Repeat
    If  Matriz[c,d]=Matriz[c,e] then
     Begin
      If Matriz[c,d]> Matriz[c,e] then
       Begin
         May_Lex:=True;
         Exit;
       End
      Else
       inc(c);
     End
    Else
     Exit;
   Until ( c = TOTREN );
End;



{************************** PIVOTEA ****************************}
{Pivotea sobre el elemento en la posicion r,c                    }

Procedure Pivotea(r,c: Integer);

 Var

   i, j : Integer;
   Aux  : Real;

Begin
   Aux:= Matriz[r,c];
   For i:=1 to Max_Reng do
     Matriz[i,c]:= Matriz[i,c]/Aux;
```

```
    For j:=1 to Max_Col do
     Begin
          Aux:=Matriz[r,j]/Matriz[r,c];
          If j<>c then
            Begin
              For i:=1 to Max_Reng do
                Matriz[i,j]:=Matriz[i,j]-(Aux*Matriz[i,c]);
            End;
     End;
         Aux:= Term_Indep[r]/Matriz[r,c];
         For i:=1 to Max_Reng do
         Term_Indep[i]:=Term_Indep[i]-(Aux*Matriz[i,c]);
End;


{********************** ESCOGE RENGLON **********************}
{Escoge el renglon candidato para pivotear                    }


 Procedure Escoge_Renglon(Var i : Integer);

 Var

  c : Integer;

 Begin

   For c:=2 to (Max_Reng - 1) do
    Begin
     If  Term_Indep[c] < 0 then
      Begin
       i:=c;
       Exit;
      End;
    End;
   i:=0;
 End;


{********************** NUEVA RESTR **********************}
{Genera la cortadura en base al elemento elegido          }

 Procedure Nueva_Rest(ren,col: Integer; Op,NvaRest: Boolean);

 Const

   Infinito = maxint;

 Var

   Mu        : Array[1..TOTCOL] of Integer;
   c,r,s,i   : Byte;
   Aux,Lamda : Real;
   Tem       : Real;
```

```
  Ya          : Boolean;

Begin
  For c:=1 to Max_Col do
   Begin
    Ya:=False;
    If c=col then
     Mul[c]:=1
    Else
     Begin
      r:=1;
      Repeat
       If Matriz[r,c]<>0 then
        Begin
         If Matriz[r,col]=0 then
          Aux:=Infinito
         Else
          Begin
           If Matriz[r,c]=Matriz[r,col] then
            Aux:=2
           Else
            Aux:=Matriz[r,c]/Matriz[r,col];
          End;
         Ya:=True;
        End
       Else
        r:=r+1;
      Until  Ya = True;
     If Frac( Aux ) = 0 then
       Begin
          For i:=1 to Max_Reng do
            Matriz[i,col]:= Matriz[i,col] * Aux;
          If May_Lex(col,c) then
            Mul[c]:=Trunc( Aux )
          Else
            Mul[c]:=Trunc( Aux )-1;
          For i:=1 to Max_Reng do
            Matriz[i,col]:= Matriz[i,col] / Aux;
       End
      Else
       Mul[c]:=Trunc(Aux0);
     End;
   End;
   Lamda:=0;
   For c:=i to Max_Col do
    Begin
     If Matriz[ren,c]>0 then
      Begin
        Tem:=Matriz[ren,c]/Mul[c];
        If Lamda<Tem then
        Lamda:=Tem;
      End;
    End;
  If Op = True then
   Begin
    Repeat
```

```
      Gotoxy(1,1);
      Write('DAME LA LAMDA A USAR < 0 para elegirlo automaticamente >: ')
      {$I-} Readln(Tem); {$I+}
      If (Tem = 0) then
      Else
      If ((Tem > 0) and (Tem<Lamda)) then
        Error(8);
    Until ( (((Tem >= Lamda) or (Tem = 0))) and (IoResult = 0) );
    ClrScr;
    If (Tem > Lamda) then
      Lamda:=Tem;
  End;
 For c:=1 to Max_Col do
  Begin
   If  Frac(Matriz[ren,c]/Lamda)=0 then
       Matriz[Max_Reng,c]:= Trunc(Matriz[ren,c]/Lamda)
   Else
      Begin
        If ( ((Matriz[ren,c]/Lamda) < 0) and
                  ((Matriz[ren,c]/Lamda) >-1)) then
           Matriz[Max_Reng,c]:=0
        Else
           Matriz[Max_Reng,c]:= Trunc(Matriz[ren,c]/Lamda)+1;
      End;
  End;

  If  Frac(Term_Indep[ren]/Lamda)=0 then
    Term_Indep[Max_Reng]:= Trunc(Term_Indep[ren]/Lamda)
  Else
    Term_Indep[Max_Reng]:=-((Trunc(Abs(Term_Indep[ren])/Lamda))+1);
   If ( NvaRest = True ) then
    Begin
     Writeln;
     Writeln(' NUEVA RESTRICCION : ');
     Writeln;
     Write(' ',Term_Indep[Max_Reng]:8:3,' = ');
     Write(Matriz[Max_Reng,1]:8:3,NoBas[1]);
     For c:=2 to Max_Col do
      Begin
       If ( Matriz[Max_Reng,c] >= 0 ) then
          Write(' + ')
       Else
          Write(' - ');
       Write(Abs( Matriz[Max_Reng,c] ):8:3,NoBas[c]);
      End;
      Tecla;
      ClrScr;
    End;
End;


{**********************  ESCOGE COLUMNA **********************}
{Escoge la columna candidata para pivotear                    }

 Procedure Escoge_Columna( ren: Integer; Var indice: Integer;
```

```
                        Muestra: Boolean);

Var

j,Cont    : Integer;
Primero   : Integer;
Segundo   : Integer;
Candidato: Set of Byte;
Tem       : Real;
Comparado: Boolean;

Begin

    Indice:=0;
    Candidato:=[];
    For  j:=1 to Max_Col  do
     Begin
       If Matriz[ren,j]>0 then
            Candidato:=Candidato + [j];
     End;
     If ( Candidato = [] ) then
        Exit;
     Cont:=1;
     While not (Candidato = [])  do
      Begin
        If Cont in Candidato then
          Begin
            Primero:=Cont;
            Candidato:=Candidato - [Cont];
            If (Candidato = []) then
              Begin
                If ( Indice = 0 ) then
                  Begin
                    Indice:= Cont;
                    Exit;
                  End
                Else
                  Begin
                    If May_Lex(Cont,Indice) then
                      Indice:= Cont;
                    Exit;
                  End;
              End;
            If (Indice <> 0) then
              Begin
                If May_Lex(Primero,Indice) then
                    Indice:= Primero
                Else
                    Primero:= Indice;
              End;
            Segundo:=Cont + 1;
            Repeat
              Comparado:=False;
              If ( Segundo in Candidato) then
                Begin
                  Comparado:=True;
```

```
                    Candidato:=Candidato - (Segundo);
                    If May_Lex(Primero,Segundo) then
                       Indice:=Primero
                    Else
                       Indice:=Segundo;
                  End
                Else
                  Segundo:=Segundo + 1;
              Until ((Segundo > Max_Col) or (Comparado = True));
              Cont:=Segundo + 1;
            End
          Else
            Cont:=Cont + 1;
        End;
  End;


{**********************  HAY EME EN TABLA  **********************}
{Detecta si se necesita usar " M ", ie. restriccion artificial}

Function HayEmeenTabla:Boolean;

 Var

  i : Byte;

 Begin

   HayEmeenTabla:=False;
   For i:=1 to (Max_Reng-1) do
     Begin
       If(( Eme <> 0 )and( Abs(Term_Indep[i]) > Eme/2 )) then
         Begin
           HayEmeenTabla:=True;
           Exit;
         End;
     End;
 End;


{**************************  I M P R I M E  **************************}
{Imprime en pantalla la matriz de coeficientes                        }

Procedure Imprime(Largo:Integer);

 Var

  i,j       : Byte;
  Modificado: Real;

 Begin
      If ( Func_Obj = 1 ) then
        Modificado:= Term_Indep[1]
      Else
```

```
  Modificado:= -Term_Indep[1];
WritelnC' ':10,' O P T I M O : ',Modificado:1:12);
If (UsandoEme and HayEmeenTabla) then
  WritelnC' Valor usado de M = ',Eme:3:0)
Else
  Writeln;
WriteC'┌');
For i:=1 to 7 do
  WriteC'─');
WriteC'┬');
For i:=1 to (Max_Col + 1)  do
  Begin
    For j:=1 to 9 do
      WriteC'─');
    If (i = 1) then
      WriteC'┬')
    Else If (i = (Max_Col + 1)) then
      WriteC'┐')
    Else
      WriteC'┬');
  End;
Writeln;
WriteC'│ ',' ','VARS',' ','│','TER.IND.',' │ ');
  For x:=1 to Max_Col do
    WriteC' ',NoBas[x]:5,'  ','│ ');
   Writeln;
WriteC'├');
For i:=1 to 7 do
   WriteC'─');
WriteC'┼');
For i:=1 to (Max_Col + 1)  do
  Begin
    For j:=1 to 9 do
      WriteC'─');
    If (i = 1) then
      WriteC'┼')
    Else If (i = (Max_Col + 1)) then
      WriteC'┤')
    Else
      WriteC'┼');
  End;
Writeln;
For w:=1 to Largo do
 Begin
    If (w = 1) then
      WriteC'│ ',Vars[1]:5,' │',Modificado:8:2,' : ')
    Else
      WriteC'│ ',Vars[w]:5,' │',Term_Indep[w]:8:2,' : ');
  For x:=1 to Max_Col do
    WriteC(Matriz[w,x]:8:2,'│ ');
   Writeln;
 End;
WriteC'└');
For i:=1 to 7 do
   WriteC'─');
WriteC'┴');
```

```
      For i:=1 to (Max_Col + 1)  do
        Begin
          For j:=1 to 9 do
             Write('=');
          If (i = 1) then
             Write('+')
          Else If (i = (Max_Col + 1)) then
             Write('+')
          Else
             Write('+');
        End;
      If Max_Reng > 8 then
        Tecla;
      Writeln;
      For i:=2 to Largo do
        Begin
          Write(' ',Vars[i]:8,' = ',Term_Indep[i]:14:9);
          If ( ( i - 1 ) mod 3 = 0 ) then
           Writeln;
        End;
      Tecla;
      ClrScr;
  End;


{*************************** I T E R A ***************************}
{Efectua una iteracion del algoritmo                              }

   Procedure Itera(Op,Escoger,Muestra,NvaRest:boolean;
                   Var termine:boolean);

   Var

     ren,col,tot : Integer;
     i : Byte;

   Begin
    If (Escoger = True) then
     Begin
      ClrScr;
      Gotoxy(1,2);
      Write(' RENGLON       VARIABLE       TERMINO INDEPENDIENTE');
      For i:=1 to (Max_Reng - 1) do
        Begin
          Gotoxy(8,i+2);
          Write(i:2,' ':9,Vars[i]:8,' ':11,Term_Indep[i]:10:3);
        End;
      Repeat
        Gotoxy(1,1);
        Write('RENGLON ( 0 para elegirlo automaticamente ): ');
        {$I-} Readln(ren); {$I+}
        If (((Term_Indep[ren] >= 0)and(ren<>0)) or
              (ren > Max_Reng)) then
         Error(7);
      Until (((((Term_Indep[ren]<0)and((ren>1)and
```

```
            Cren<Max_Reng)))or(ren=0)))
     and (IoResult = 0));
     ClrScr;
     If (ren = 0)  then
        Escoge_Renglon(ren);
    End
   Else
    Escoge_Renglon(ren);
   If ren=0 then
    Begin
     Termine:=True;
     If (UsandoEme and HayEmeenTabla) then
          Writeln(' ':20,'*** SOLUCION NO ACOTADA ***')
     Else
          Writeln(' ':25,'SOLUCION ENTERA OPTIMA');
     Exit;
    End;
   Escoge_Columna(ren,col,Muestra);
   If ( ( Muestra = True ) and ( Col > 0 ) ) then
     Begin
       Writeln('EL PIVOTE ES ',ren,',',Col,':',Matriz[ren,Col]);
       Tecla;
       ClrScr;
     End;
   If col=0 then
    Begin
     termine:=True;
     Writeln('SOLUCION NO ACOTADA:');
     Exit;
    End;
   Nueva_Rest(ren,col,op,NvaRest);
   Encabezado(Max_Reng,col);
   Pivotea(Max_Reng,col);
  End;


{********************** REDONDEA ******************************}
{Redondea los numeros de la matriz y los terminos independientes
 para corregir posibles errores por truncamiento e imprecision
 en los calculos; esto lo hace en base a una tolerancia}

Procedure Redondea;

Var
 i,j : Integer;

Begin
   For i:=1 to Max_Reng do
    Begin
     If ((Abs(Round(Term_Indep[i])-Term_Indep[i]))
         < Tolerancia ) then
        Term_Indep[i]:= Round(Term_Indep[i]);
     For j:=1 to Max_Col do
      Begin
          If ((Abs(Round(Matriz[i,j]))-Matriz[i,j]))
```

```
                < Tolerancia ) then
              Matriz[i,j]:= Round(Matriz[i,j]);
        End;
      End;
End;


{********************* ESCCOLPRIMAL *********************************}
{Escoge la columna candidato para pivotear en el algoritmo
Primal}

Procedure EscColPrimal( var indice:integer);

Var
  i : integer;

Begin
   Indice:=0;
   For i:=1 to Max_Col do
     Begin
       If (matriz[1,i] > 0) then
         Begin
           indice:=i;
           Exit;
         End;
     End;
End;


{********************* ESCRENPRIMAL *********************************}
{ Escoge el renglon candidato para pivotear en el algoritmo
 Primal}

Procedure EscRenPrimal(col:integer; var indice:integer);

Var
  i,j    : Integer;
  Minimo : Real;

Begin
   Minimo:=MaxInt;
   Indice:=0;
   For i:=2 to Max_Reng do
     Begin
       If (matriz[i,col] < 0) then
         Begin
           If (Minimo > Abs((Term_Indep[i]/Matriz[i,col]))) then
             Begin
               Indice:=i;
               Minimo:=Abs(Term_Indep[i]/Matriz[i,col]);
             End;
         End;
     End;
```

```
End;


{************************** PRIMAL **************************}
{ Implementacion del algoritmo Primal}

Procedure Primal;

 Var
   col,ren : Integer;
   Ya      : Boolean;

 Begin
   Vars[Max_Reng]:='#HOLA';
   ya:=False;
   Repeat
     EscColPrimal( col );
     If (col = 0) then
       Begin
         ClrScr;
     GotoXY(10,6);Write('                                                  ');
     GotoXY(10,7);Write(' SE ENCONTRO SOLUCION OPTIMA CONTINUA             ');
     GotoXY(10,8);Write('                                                  ');
         Tecla;
         Ya:=True;
         ClrScr;
         Exit;
       End;
     EscRenPrimal(col,ren);
     If (ren = 0) then
       Begin
         ClrScr;
     GotoXY(10,6);Write('                                                  ');
     GotoXY(10,7);Write('          NO EXISTE SOLUCION FINITA               ');
     GotoXY(10,8);Write('                                                  ');
         SinSolucion:=True;
         Nuevo:=True;
         Tecla;
         ClrScr;
         Ya:=True;
         Exit;
       End;
     EncabezadoCren,col);
     PivoteaCren,col);
     Redondea;
   Until Ya = True;
 End;


{************************** ALG DISCRETO **************************}
{Implementacion del algoritmo Discreto                             }

 Procedure AlgDiscreto(Veces:Integer;Si,Escoger,Muestra,
                       NvaRest:boolean);
```

```
Var
 Cont,Tem,i : Integer;
 fin        : Boolean;
 c          : Char;

Begin
 fin:=False;
 Cont:=0;
 Vars[Max_Reng]:='HOLA';
 For i:=1 to veces do
  Begin
      IteraCsi,Escoger,Muestra,NvaRest,Fin);
      Cont:=Cont+1;
      If ( Veces = 1 ) then
        ImprimeCMax_Reng-1);
      If ( CFin = True) and (Veces > 1) ) then
       Begin
         WriteC'  NUMERO DE ITERACIONES : ',Cont-1,' ':5);
         ImprimeCMax_Reng-1);
         Exit;
       End;
      If  (Optima = False)  then
        Begin
          ClrScr;
          Writeln;
          WritelnC'     SE PERDIO OPTIMALIDAD ');
          WritelnC'  SE PROCEDERA A REOPTIMIZAR ');
          Writeln;
          WritelnC' Cualquier tecla para seguir . . .');
          Tecla;
          Primal;
          If (Entera = False) then
            Begin
              SinSolucion:=True;
              ClrScr;
              Writeln;
              WritelnC'                                    ');
              WritelnC'    EXISTEN FACTORES NO ENTEROS     ');
              WritelnC'                                    ');
              WritelnC'  Volver a resolver el problema     ');
              WritelnC'  plantendolo de otra forma o       ');
              WritelnC'  eligiendo otro elemento pivote    ');
              WritelnC'                                    ');
              Writeln;
              WritelnC'   Cualquier tecla para seguir . . .');
              Tecla;
              Exit;
            End;
          End;
  End;

End;

END.
```

5.4 UNIDAD CICLICO

```
Unit Ciclico;

 Interface

Uses Crt,Global,Discreto;

   Procedure DualLex;
   Procedure AlgCiclico(Veces:Integer;Escoger,Muestra,
                        NvaRest:Boolean);

Implementation



{************************ ESCCOLDUAL ******************************}
{ Escoge la columna candidato para pivotear en el algoritmo Dual}

Procedure EscColDual(ren:integer; var indice:integer);

Var
  i,j    : Byte;
  Minimo : Real;

Begin
    Minimo:=MaxInt;
    Indice:=0;
    For i:=1 to Max_Col do
      Begin
        If (matriz[ren,i] > 0) then
          Begin
            j:=0;
            If (Minimo >= abs(matriz[1,i]/matriz[ren,i])) then
              Begin
                If (Minimo > abs(matriz[1,i]/matriz[ren,i])) then
                  Begin
                     Minimo:= abs(matriz[1,i]/matriz[ren,i]);
                     Indice:=i;
                  End
                Else
                 Begin
                  Repeat
                  j:=j+1;
                  If ((Matriz[1,indice]/Matriz[ren+j,indice]) <
                     (Matriz[1,i]/matriz[ren+j,i])) then
                       Begin
                        Minimo:= abs(matriz[1,i]/matriz[ren,i]);
                        Indice:=i;
                       End;
                  Until ((Matriz[1,indice]/Matriz[ren+j,indice])
                         <> (Matriz[1,i]/matriz[ren+j,i]));
```

```
                End;
              End;
          End;
      End;
End;


{************************** NUEVA RESTRICCION ***********************}
{ Genera la restriccion que sera la cortadura a emplear}

Procedure Nueva_Restriccion(ren:integer;NvaRest:Boolean);

 Var
   i : Byte;
  Fl : Real;

 Begin

   Fl:= Frac(Term_Indep[ren]);
   Term_Indep[Max_Reng]:= -Fl;
   For i:=1 to Max_Col do
     Begin
       If (Frac(Matriz[ren,i]) = 0) then
         Matriz[Max_Reng,i]:= 0
       Else
        Begin
         If (Matriz[ren,i] > 0) then
           Matriz[Max_Reng,i]:= ((Int(Matriz[ren,i]) + 1) -
              Matriz[ren,i])
         Else
           Matriz[Max_Reng,i]:= (Int(Matriz[ren,i]) -
              Matriz[ren,i])
        End;
     End;
     If ( NvaRest = True ) then
      Begin
       Writeln;
       Writeln(' NUEVA RESTRICCION : ');
       Writeln;
       Write(Term_Indep[Max_Reng]:8:3,' = ');
       Write(Matriz[Max_Reng,1]:8:3,NoBas[1]);
       For i:=2 to Max_Col do
        Begin
         If ( Matriz[Max_Reng,i] >= 0 ) then
            Write(' + ')
         Else
            Write(' - ');
         Write(Abs(Matriz[Max_Reng,i]):8:3,NoBas[i]);
        End;
       Tecla;
       ClrScr;
      End;
End;


{************************** DUALLEX ***********************}
```

```
{ Implementacion del algoritmo Dual Lexicografico}

Procedure DualLex;

 Var
   col,ren : Integer;
   Ya      : Boolean;

 Begin
   Vars[Max_Reng]:='HOLA';
   ya:=False;
   Repeat
     Escoge_Renglon( ren );
     If (ren = 0) then
       Begin
         ClrScr;
    GotoXY(10,8);Write('┌──────────────────────────────────────┐');
      GotoXY(10,7);
         If UsandoElse then
            Begin
              Write('│         SOLUCION NO ACOTADA          │');
              SinSolucion:=True;
            End
         Else
            Write('│SE ENCONTRO SOLUCION OPTIMA CONTINUA│');
    GotoXY(10,8);Write('└──────────────────────────────────────┘');
         Tecla;
         ClrScr;
         Ya:=True;
         Exit;
       End;
     EscColDual(ren,col);
     If (col = 0) then
       Begin
         ClrScr;
    GotoXY(10,8);Write('┌──────────────────────────────┐');
    GotoXY(10,7);Write('│ NO EXISTE SOLUCION FACTIBLE  │');
    GotoXY(10,8);Write('└──────────────────────────────┘');
         SinSolucion:=True;
         Nuevo:=True;
         Tecla;
         ClrScr;
         Ya:=True;
         Exit;
       End;
     EncabezadoCren,col);
     PivoteaCren,col);
     Redondea;
   Until Ya = True;
 End;


{▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒ ESCRENCICLICO ▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒}
{ Escoge al renglon candidato en el algoritmo Ciclico}
 Procedure EscRenCiclico( var indice:integer);
```

```
Var
  i : Byte;

Begin
   Indice:=0;
   For i:=2 to Max_Reng do
     Begin
       If (Term_Indep[i] < 0) then
         Begin
            Indice:=MaxInt;
            Exit;
         End;
     End;
   For i:=2 to Max_Reng do
     Begin
       If (Term_Indep[i] <> 0) then
         Begin
            If (Frac(Term_Indep[i]) <> 0) then
              Begin
                 Indice:=i;
                 Exit;
              End;
         End;
     End;
End;


{************************** CICLICO **************************}
{ Implementacion del algoritmo Ciclico o Fraccional }

Procedure AlgCiclico(Veces:integer;Escoger,Muestra,
              NvaRest:Boolean);
 Var
   i,j,ren,col : Integer;
   con         : Byte;

 Begin
     Vars[Max_Reng]:='=HOLA';
     For i:=1 to Veces do
       Begin
          If Escoger = True then
            Begin
             ClrScr;
             Gotoxy(1,2);
             Write(' RENGLON       VARIABLE
                    TERMINO INDEPENDIENTE');
             For con:=1 to (Max_Reng - 1) do
              Begin
               Gotoxy(8,con+2);
               Write(con:2,' ':9,Vars[con]:8,' ':11,
                        Term_Indep[con]:10:3);
              End;
             Repeat
              Gotoxy(1,1);
             Write('RENGLON ( 0 para elegirlo
                              automaticamente ): ');
```

```
            {$I-} Read(j){$I+}
          Until ( ((Frac(Term_Indep[j]) <> 0) or (j = 0 )) and
             (IoResult = 0) ) ;
            If ( j = 0 ) then
               EscRenCiclico(ren);
          End
        Else
        EscRenCiclico(ren);
        If (ren = MaxInt) then
           DualLex
        Else
          Begin
            If (ren = 0) then
               Begin
                 If Optima then
                   Begin
                     Writeln(' ':25,'SOLUCION ENTERA OPTIMA');
                     Write(' NUMERO DE ITERACIONES : ',i - 1);
                     Imprima(Max_Reng-1);
                     Exit;
                   End
                 Else
                   Begin
                     Primal;
                     If Entera then
                       Begin
                     Writeln(' ':25,'SOLUCION ENTERA OPTIMA');
                     Write(' NUMERO DE ITERACIONES : ',i - 1);
                         Imprima(Max_Reng-1);
                         Exit;
                       End;
                   End;
               End;
            Nueva_Restriccion(ren,NvaRest);
            Escoge_Columna(Max_Reng,col,Muestra);
            If (col = 0) then
               Begin
               Writeln(' NO EXISTE SOLUCION ENTERA FACTIBLE');
                 Imprima(Max_Reng-1);
                 Exit;
               End;
            Encabezado(Max_Reng,col);
            Pivotea(Max_Reng,col);
            Redondea;
            If Veces = 1 then
               Imprima(Max_Reng-1);
          End;
      End;
End;


    END.
```

5.6 UNIDAD AJUSTA

```
Unit Ajusta;

Interface

Uses Crt,Global,Discreto,Ciclico;

  Procedure SolucionaCVeces:Integer;UsaDiscreto,EscLam,
                       EscRen,Muestra,NvaRest:Boolean);
  Procedure PreparaCUsaDiscreto:Boolean);

Implementation



{~~~~~~~~~~~~~~~~~~~~~~ ENCUENTRA EME ~~~~~~~~~~~~~~~~~~~~~~~~~~~~}
{En caso de usarse restriccion artificial decide el valor
apropiado para M}

Procedure EncuentraEme;

 Var
   Maximo : Real;
   i      : Byte;

 Begin
   Maximo:=0;
   For i:=1 to Max_Rang do
    Begin
     If CAbsCTerm_Indep[i]) > Maximo) then
       Maximo:=CAbsCTerm_Indep[i]));
    End;
   If CMaximo <= 10) then
     Begin
          Eme:=100;
          Exit;
     End
   Else If CMaximo <= 80) then
     Begin
          Eme:=1000;
          Exit;
     End
   Else If CMaximo <= 100) then
     Begin
          Eme:=10000;
          Exit;
     End
   Else If CMaximo <= 1000) then
     Begin
          Eme:=100000;
          Exit;
```

```
    End
  Else If (Maximo < 10000) then
    Begin
         Eme:=1000000;
         Exit;
    End
   Else
    Error(10);

End;


{************************** DIVIDE ****************************}
{Si los coeficientes son tan grandes que ningun valor de M es
suficiente propone al usuario dividir todos los coeficientes del
problema para poder seguir trabajando o bien suspender el
algoritmo}

Procedure Divide;

  Var
    i,j,Factor : Byte;

  Begin
    Writeln('Factor entre el cual se dividir
     la tabla, 2-10 ? : ');
    Repeat
     Read(Factor);
    Until ((Factor > 1)and(Factor < 11));
    For i:=1 to Max_Reng do
      Begin
        For j:=1 to Max_Col do
           Matriz[i,j]:=Matriz[i,j] / Factor;
        Term_Indep[i]:=Term_Indep[i] / Factor;
      End;
  End;


{************************ ENCUENTRA COLUMNA ***********************}
{Encuentra una columna candidato sobre la cual se pivoteara}

Procedure EncuentraColumna(Var index:Byte);

 Var
  j   : Byte;
  Max : Real;

 Begin
  Max:=0;
  For j:=1 to Max_Col do
    Begin
      If ((Matriz[Max_Reng,j]=1)and(Matriz[1,j]>Max)) then
        Begin
          Max:=Matriz[1,j];
          Index:=j;
```

```
        End;
    End;
End;


{********************** AGREGA RESTRICCION **********************}
{Una vez elegida M se agrega la restriccion artificial para
obtener una base dual factible}

Procedure Agrega_Restriccion(Var NoHaySol : Boolean);

Var
 i,j,Index : Byte;
 Ch        : Char;

Begin
  NoHaySol:= False;
  For j:=1 to Max_Col do
    Begin
      If (Matriz[i,j] > 0) then
          Matriz[Max_Reng,j]:=-1;
    End;
  EncuentraEme;
  If (Eme = 0) then
   Begin
    Writeln('Puedes continuar dividiendo toda la tabla ');
    Writeln('entre algún factor , CONTINUAR ?  S/N : ');
    Repeat
     Ch:=ReadKey;
    Until (Upcase(Ch) in ['S','N']);
    If (Upcase(Ch)='S') then
        Begin
          Divide;
          EncuentraEme;
          If (Eme = 0) then
            Begin
              Error(12);
              NoHaySol:=True;
              Exit;
            End;
        End
    Else
        Begin
          NoHaySol:=True;
          Exit;
        End;
    End;
    UsandoEme:=True;
    Term_Indep[Max_Reng]:=Eme;
    EncuentraColumna(Index);
    Pivotea(Max_Reng,Index);

End;
```

```
{************************ AJUSTA CICLICO ************************}
{Procedimiento que ajusta los datos introducidos por el usuario
al formato necesario para iniciar el algoritmo Ciclico}

Procedure Ajusta_Ciclico;

Var
 i         : Byte;
 NoHaySol  : Boolean;

Begin

  If Factible then
   Begin
     If Optima then
       Exit
     Else
       Primal;
   End
  Else
   Begin
    If Optima then
     DualLex
    Else
     Begin
       Agrega_Restriccion(NoHaySol);
       If NoHaySol then
         Begin
           SinSolucion:=True;
           Exit;
         End;
        DualLex;
        For i:=i to (Max_Reng-1) do
         Begin
          if (Abs(Term_Indep[i]) > (Eps/2)) then
            Begin
              SinSolucion:=True;
              Exit;
            End;
         End;
     End;
   End;
End;


{************************ AJUSTA DISCRETO ************************}
{Procedimiento que ajusta los datos introducidos por el usuario
al formato necesario para iniciar el algoritmo Discreto}

Procedure Ajusta_Discreto;

Var
 NoHaySol : Boolean;

Begin
```

```
  If Entera then
   Begin
     If Optima then
       Exit
     Else
      Begin
       Agrega_Restriccion(NoHaySol);
       If NoHaySol then
         Begin
           SinSolucion:=True;
           Exit;
         End;
       If Entera then
       Else
         Begin
           Error ( 8 );
           SinSolucion:=True;
           Exit;
         End;
      End;
   End
  Else
   Begin
    ClrScr;
    Gotoxy(27,7);
    Error ( 8 );
    SinSolucion:=True;
   End;
End;


{######################################## BORRA ########################################}
{Borra la funcion objetivo actual para sustituirla despues por
alguna otra, se utiliza antes y despues del metodo de las dos
fases}

Procedure Borra;

 Var
  j : Byte;

 Begin
   For j:=1 to Max_Col do
     Matriz[i,j]:=0;
   Term_Indep[i]:=0;
 End;


{######################################## RESTAURA ########################################}
{Una vez terminada la primera de las dos fases, se restaura la
funcion objetivo actualizafda del problema original}

 Procedure Restaura;

  Var
```

```
    i,j : Byte;

  Begin
   Borra;
   For i:=1 to Max_Col do
     Begin
       For j:=1 to Max_Col do
         Begin
            Matriz[i,j]:=Matriz[i,j]+
            (Matriz[Cuadro[i].linea,j] * Cuadro[i].Multiplo);
         End;
       Term_Indep[i]:=Term_Indep[i] +
       (Term_Indep[Cuadro[i].linea] * Cuadro[i].Multiplo);
     End;
  End;


{**********************************  BUSCA  **************************}
{Busca la Cadena que recibe en la tabla de variables}

Procedure Busca(Cad:String;Var Ind:Byte);

 Var
  i : Byte;

 Begin
   Ind:=0;
   For i:=1 to Max_Reng do
     Begin
      If ( Cad = Vars[i]) then
        Begin
         Ind:=i;
         Exit;
        End;
     End;
 End;


{**********************************  DOS FASES  **********************}
{Implementacion del metodo de las dos fases que utiliza algunos
de los Procedimientos anteriores}

Procedure DosFases(UsaDiscreto : Boolean);

 Var
  Ch        : Char;
  i,j,Ind   : Byte;
  NoHaySol  : Boolean;


 Begin
   ClrScr;
   Gotoxy(10,7);Write('SE INICIA METODO DE LAS DOS FASES');
   Delay(1000);
   For j:=1 to Max_Col do
```

```
    Begin
      Busca(NoBas[j],Ind);
      Cuadro[j].linea:=Ind;
      Cuadro[j].multiplo:=Matriz[i,j];
    End;
   Borra;
   For i:=1 to ContArt do
    Begin
      For j:=1 to Max_Col do
          Matriz[i,j]:=Matriz[i,j] + Matriz[Artificial[i],j];
      Term_Indep[i]:=Term_Indep[i] + Term_Indep[Artificial[i]];
    End;
   Term_Indep[1]:= -Term_Indep[1];
   For j:=1 to Max_Col do
     Matriz[1,j]:= -Matriz[1,j];
   Ajusta_Ciclico;
   If (Optima and (Term_Indep[1] = 0)) then
     Begin
       Gotoxy(4,7);Write('SE TERMINO CON EXITO
                          METODO DE LAS DOS FASES');
       Delay(1000);
       Restaura;
       If UsaDiscreto then
          Ajusta_Discreto
       Else
          Ajusta_Ciclico;
     End
   Else
         Begin
           Error ( 8 );
           SinSolucion:=True;
           Exit;
         End;
End;


{########################## PREPARA ##########################}
{Procedimiento que prepara la tabla inicial del problema}

Procedure Prepara(UsaDiscreto:Boolean);

Begin
  If HayArtificial then
    DosFases(UsaDiscreto)
  Else
    Begin
      If UsaDiscreto then
        Ajusta_Discreto
      Else
        Ajusta_Ciclico;
    End;
End;


{########################## SOLUCIONA ##########################}
```

```
(Procedimiento que coordina la ejecucion de los algoritmos
implementados en unidades anteriores)

Procedure Soluciona;

Begin

 If ( UsaDiscreto = True ) then
     AlgDiscreto(Veces,EscLam,EscRen,Muestra,NvaRest)
 Else
     AlgCiclico(Veces,EscRen,Muestra,NvaRest);
End;


END.
```

5.6 UNIDAD INFORMA

```
Unit Informa;

Interface

Procedure Ayuda_Editor;
Procedure Notas;
Procedure General;
Procedure Advierte;

Implementation

Uses Crt,Global;


Procedure Ayuda_Editor;

Begin

 ClrScr;
 Writeln;
 Writeln('      EJEMPLO DE PLANTEAMIENTO :');
 Writeln;
 Writeln('       MAX');
 Writeln('       3x1-5x2  ( funcion objetivo )');
   Write('       ');Marca('SUJETO A:');Writeln;
 Writeln('       2x1-5x3+x4=6');
 Writeln('       x1+7x2<=7');
 Writeln('       4x1-6x4+8x3>=12');
 Writeln('       #');
 Writeln;
 Writeln('  El simbolo # indica el final de las restricciones');
 Writeln('   y de la edicion del problema.');
 Tecla;
 ClrScr;
 Writeln;
```

```
    WriteLnC'    La funcion objetivo debe ocupar un solo renglon');
    WriteLnC'    reconociendose solamente " MAX " y " MIN " como');
    WriteLnC'    funciones objetivo.');
    WriteLn;
    WriteLnC'    El renglon " SUJETO A: " es puesto automaticamente');
    WriteLnC'    por el sistema despues de haber tecleado la funcion');
    WriteLnC'      objetivo;recuerda teclear ENTER despues de cada
restriccion.');
    WriteLn;
    WriteLnC'    Es indistinto el uso de MAYUSCULAS o minusculas o su');
    WriteLnC'    combinacion.');
    Tecla;
    ClrScr;
    WriteLn;
    WriteLnC'
-------------------------------------------------------------------------');
WriteLnC'  |                  **** A D V E R T E N C I A ****
          |');
WriteLnC'  |
          |');
WriteLnC'  | Al momento de editar cualquier problema, el sistema
          |');
WriteLnC'  | "prepara" a este y obtiene la tabla inicial para el
          |');
WriteLnC'  | algoritmo elegido;por esta razon, es NECESARIO elegir
          |');
WriteLnC'  | al algoritmo primero y despues editar el problema, ya
          |');
WriteLnC'  | que en caso de no hacerlo asi, es posible que se
          inicie          |');
WriteLnC'  | un algoritmo con la tabla equivocada. Es importante
          |');
WriteLnC'  | considerar esto cuando se desee cambiar el algoritmo
          |');
WriteLnC'  | con el cual se esta trabajando, ya que al resolver el
          |');
WriteLnC'  | problema, se asumir  una tabla inicial y lida.
          |');
WriteLnC'  |
          |');
WriteC'
-------------------------------------------------------------------------');
    Tecla;

    End;


  Procedure Notas;

  Begin

    ClrScr;
    WriteLn;
    WriteLnC'    ALGUNAS ESPECIFICACIONES :');
    WriteLn;
    WriteLnC'     Maximo de variables     : 20
```

```
      ( Incluyendo De Holgura )');
  WriteLnC'      Maximo de restricciones : 10');
  WriteLnC'      Funciones Objetivo      : MAX , MIN');
  WriteLnC'      Algoritmos a Escoger    : Discreto,Ciclico');
  WriteLn;
  WriteLnC'    Los nombres de variables pueden tener
       un maximo de');
  WriteLnC'    5 caracteres,no pudiendose usar los
        simbolos @,$,-,+');
  WriteLnC'    >,<,=,%,!  en los nombres pues son operadores
       o son');
  WriteLnC'    utilizados por el sistema.');
  Tecla;
  ClrScr;
  WriteLn;
  WriteLnC'    Como el sistema implementa los algoritmos
        Simplex y ');
  WriteLnC'    Dual Lexicografico,se puede utilizar el sistema
       para');
  WriteLnC'    Programacion Lineal escogiendo el algoritmo
        Ciclico y');
  WriteLnC'    revisando su tabla inicial.');
  WriteLn;
  WriteLnC'    En caso de minimizar la funcion objetivo,
       se utiliza el');
  WriteLnC'    hecho de que : Min ( Z ) = - Max ( -Z )
       resolviendose');
  WriteLnC'    el problema para este ultimo caso;
       el sistema hace este');
  WriteLnC'    cambio automaticamente por lo que no es
       necesario hacer');
  WriteC'    ningun ajuste.');
  Tecla;

End;


Procedure General;

Begin

  ClrScr;
  WriteLn;
  WriteLnC'                    A Y U D A   G E N E R A L ');
  WriteLn;
  WriteLn;
  WriteLnC'    Ayuda    : Menu de Ayuda');
  WriteLnC'    Editar   : Edicion del problema y eleccion
       del algoritmo');
  WriteLnC'    Resolver : Resuelve totalmente el problema');
  WriteLnC'    Pivotear : Efectua una sola iteracion del
       algoritmo');
  WriteLnC'    Imprimir : Imprime en papel la tabla actual
       del problema');
  WriteLnC'    Tabla    : Muestra en pantalla la tabla actual
       del problema');
  WriteLnC'    Opciones : Opciones para la resolucion del
```

```
        problema');
    WriteLnC'     Salir     : Terminar y salir al sistema
        operativo');
    WriteLnC'               F2 : Hora del Sistema');
    WriteLnC'               F3 : Informacion del Sistema');
    Tecla;

  End;


  Procedure Advierte;

  Begin

    ClrScr;
    WriteLn;
    WriteLn;
    WriteLn;
    WriteLnC'
═══════════════════════════════════════════════════════════════════════════');
    WriteLnC'                          ### I M P O R T A N T E ###
                 ');
    WriteLnC'
                 ');
    WriteLnC'          En esta opcion se proceder  a resolver el
        problema                                  ');
    WriteLnC'          totalmente; si se desea analizar el
        desarrollo se                             ');
    WriteLnC'          recomienda usar la opcion "Pivotear" dentro
del menu          ');
    WriteLnC'          principal.
                 ');
    WriteLnC'          Presionar cualquier tecla para resolver el
        problema                                  ');
    WriteLnC'          TOTALMENTE o la barra espaciadora para
        regresar al                               ');
    WriteLnC'          menu principal.
                 ');
    WriteLnC'
═══════════════════════════════════════════════════════════════════════════');
  End;

  End.
```

5.7 UNIDAD IMPRESOR

```
Unit Impresor;

Interface
```

```
Uses Crt,Printer,Global,Discreto;


Procedure ImpresionCLargo:Integer);

Implementation


Var

 Ch     : Char;
 Seguir : Boolean;


{************************** CHECA IMPRESORA **********************}
{Procedimiento que revisa si esta prendida la impresora }

Procedure ChecaImpresora;

Var
   Salir : Boolean;

 Begin
   Salir := False;
   Seguir:= False;
   Repeat
     GotoxyC11,7);
     Write C'▒▒▒▒▒▒▒▒▒▒     I M P R I M I E N D O     ▒▒▒▒▒▒▒▒▒▒');
     {$I-}
     WritelnCLst);
     {I+}
     If IoResult  <> 0 then
       Begin
         Gotoxy C11,7);
         Write C'▒▒ IMPRESORA NO EN LINEA, REINTENTAR S/N ? ▒▒');
         Repeat
           Ch:= ReadKey;
         Until Upcase (Ch) in ['N','S'];
         If Upcase (Ch) = 'N' then
           Exit;
       End
      Else
        Begin
         Seguir:= True;
         Salir := True;
        End;
   Until Salir = True;
 End;


{************************** IMPRIME **************************}
{procedimiento que imprime en papel la matriz de coeficientes,
 los terminos independientes y los valores de las variables}

Procedure ImprimeCLargo:Integer);
```

```
Var
 i,j         : Byte;
 Modificado : Real;

 Begin
      If ( Func_Obj = 1 ) then
        Modificado:= Term_Indep[1]
      Else
        Modificado:=-Term_Indep[1];
      ClrScr;
      ChecaImpresora ;
      If Seguir = False then
        Begin
          ClrScr;
          Exit;
        End;

      WritelnCLst);
      WritelnCLst);
      WritelnCLst,'-------------------------------------------');
      WritelnCLst,' ':5,'Optimizador para Programacion Entera');
      WritelnCLst,' ':5,'      Facultad de Ciencias, 1990');
      WritelnCLst,'-------------------------------------------');
      WritelnCLst);
      WritelnCLst,' ':10,' O P T I M O :  ',Modificado:1:12);
      WritelnCLst);
      If (UsandoEme and HayEmeenTabla) then
        WritelnCLst,' Valor usado de M = ',Eme:3:0)
      Else
        WritelnCLst);
      WritelnCLst);
      WriteCLst,'|');
      For i:=1 to 7 do
         WriteCLst,'=');
      WriteCLst,'|');
      For i:=1 to (Max_Col + 1)  do
        Begin
          For j:=1 to 9 do
             WriteCLst,'=');
          WriteCLst,'|')
        End;
      WritelnCLst);
      WriteCLst,'| ',' ','VARS',' ','|','TER.IND.',' | ');
        For i:=1 to Max_Col do
          WriteCLst,' ',NoBas[i]:5,'  ','| ');
        WritelnCLst);
      WriteCLst,'|');
      For i:=1 to 7 do
         WriteCLst,'=');
      WriteCLst,'|');
      For i:=1 to (Max_Col + 1)  do
        Begin
          For j:=1 to 9 do
             WriteCLst,'=');
          If (i = 1) then
             WriteCLst,'|')
```

```
          Else if (i = (Max_Col + 1)) then
             Write(Lst,'|')
          Else
             Write(Lst,'|');
        End;
      Writeln(Lst);
      For i:=1 to Largo do
       Begin
          If ( i = 1 ) then
           Write(Lst,'| ',Vars[i]:5,' |',Modificado:8:2,'|')
          Else
           Write(Lst,'| ',Vars[i]:5,' |',Term_Indep[i]:8:2,'|');
        For j:=1 to Max_Col do
          Write(Lst,Matriz[i,j]:8:2,'| ');
         Writeln(Lst);
       End;
      Write(Lst,'|');
      For i:=1 to 7 do
         Write(Lst,'=');
      Write(Lst,'|');
      For i:=1 to (Max_Col + 1)  do
        Begin
          For j:=1 to 8 do
             Write(Lst,'=');
          Write(Lst,'|')
        End;
      Writeln(Lst);
      Writeln(Lst);
      For i:=2 to Largo do
       Writeln(Lst,Vars[i]:5,' = ',Term_Indep[i]:4:12);
      For i:=1 to 5 do
       Writeln(Lst);
End;


{************************ IMPRESION ********************************}
{Procedimiento que se exporta que presenta en pantalla la opcion
de imprimir en pantalla llamando al procedimiento anterior}

 Procedure Impresion(Largo: Integer);

   Begin

     GotoXY(17,4);Marca('ESTA POR IMPRIMIRSE LA TABLA DE DATOS');
     GotoXY(21,6);Normal('Imprimir');
     GotoXY(42,6);Normal('Menu');
     GotoXY(31,8);Marca('OPCION : ');
     Repeat
      Ch:= ReadKey;
     Until ( Upcase (Ch) ) in ['I','M'];
     If ( Upcase (Ch) = 'I') then
      Begin
        ClrScr;
        TextColor(Black + Blink);
        GotoXY(14,6);Write('ASEGURATE DE QUE LA IMPRESORA
                            ESTE PRENDIDA');
```

```
        Gotoxy(23,8);Marca('PRESIONA CUALQUIER TECLA');
        Tecla;
        Imprime (Largo);
      End;
  End;

END.
```

5.8 UNIDAD TESIS

```
Uses Crt,Dos,Global,Detecta,Informa,Ajusta,Discreto,
              Ciclico,Impresor;

Const

  NumVueltas = 500;  { Numero maximo de iteraciones }

Type

 Elemento = String[28];
      Res = String[4];

Var

 cad          : String[20];
 c,Ch         : Char;
 let          : String[2];
 Opcion       : Array[1..8] of Letrero;
 lugar,Pos    : Integer;
 UsaDiscreto  : Boolean;
 TablaVacia   : Boolean;
 Correcto     : Boolean;
 Mes,Dia,Ano  : Word;
 DiaSemana    : Word;


{########################## INICIA ##################################}
{Procedimiento de inicializacion de variables locales}

 Procedure Inicia;

 Begin
  GetDate(Ano,Mes,Dia,DiaSemana);
  Opcion[1]:='Ayuda';
  Opcion[2]:='Pivotear';
  Opcion[3]:='Editar';
  Opcion[4]:='Resolver';
  Opcion[5]:='Opciones';
  Opcion[6]:='Imprimir';
  Opcion[7]:='Tabla';
  Opcion[8]:='Salir';
```

```
 Nuevo :=True;
 lugar:=i;
 Pos:=i;
 Default[1]:=False;
 Default[2]:=False;
 Default[3]:=False;
 Default[4]:=False;
 Arr[1]:=' No ';
 Arr[2]:=' No ';
 Arr[3]:=' No ';
 Arr[4]:=' No ';
 UsaDiscreto:= True;
 TablaVacia:= True;
End;


{========================== POSICION ==========================}
{Controla las posiciones de los letreros del menu}

Procedure Posicion(i:Integer);

Begin
  Case i of
   1:Gotoxy(3,2);
   2:Gotoxy(11,2);
   3:Gotoxy(22,2);
   4:Gotoxy(31,2);
   5:Gotoxy(42,2);
   6:Gotoxy(53,2);
   7:Gotoxy(64,2);
   8:Gotoxy(72,2);
  End;
End;


{========================== PON FECHA ==========================}

Procedure PonFecha;

 Begin
  Gotoxy(35,25);
  Write(Dia,'/',Mes,'/',Ano);
 End;


{========================== OPCIONES ==========================}
{Procedimiento de presentacion y eleccion de las distintas
opciones que ofrece el sistema}

Procedure Opciones;

Var
   i  : Integer;
   Ch : Char;
```

```
Begin

 GotoxyC27,4);
 MarcaC' O P C I O N E S ');
 WindowC24,7,54,19);
 ClrScr;
 GotoxyC10,2); WriteC' OPCIONES:');
 GotoxyC4,4);WriteC'Especificar Lamda  ');
 GotoxyC4,5);WriteC'Escoger Renglon');
 GotoxyC4,6);WriteC'Mostrar Pivote');
 GotoxyC4,7);WriteC'Mostrar Restriccion');
 GotoxyC10,9);WriteC'F1.......MENU');
 For i:=1 to 4 do
  Begin
    If i<>Pos then
     Begin
      GotoxyC25,i+3);WriteCArr[i]);
     End;
  End;
 GotoxyC25,Pos+3);MarcaCArr[Pos]);
 Repeat

  Ch := ReadKey;

 If Ch = #0 then
   Begin
   Ch:=Readkey;
     Case Ch of

 ARRIBA: Begin
          GotoxyC25,Pos+3);WriteCArr[Pos]);
          If Pos > 1  then
            Pos:=Pos - 1
          Else
            Pos:= 4;
          GotoxyC25,Pos+3);MarcaCArr[Pos]);
         End;
 ABAJO: Begin
         GotoxyC25,Pos+3);WriteCArr[Pos]);
          If Pos < 4  then
            Pos:=Pos + 1
          Else
            Pos:= 1;
          GotoxyC25,Pos+3);MarcaCArr[Pos]);
           End;
     F1:Begin
          TextbackgroundCBlack);
          ClrScr;
          Exit;
        End;

     End; < Case >
   End < If >
  Else
   Begin
     If Ch=ENTER then
```

```
      Begin
         If Default[Pos]=True then
           Begin
             Arr[Pos]:=' No ';
             GotoxyC28,Pos+3);MarcaCArr[Pos]);
             Default[Pos]:=False;
           End
         Else
           Begin
             Arr[Pos]:=' Si ';
             GotoxyC28,Pos+3);MarcaCArr[Pos]);
             Default[Pos]:=True;
           End;
      End;
   End;
 Until c = ' ';
 TextBackgroundCBlack);
 ClrScr;
End;


(************************** AYUDA ****************************)
(Procedimiento que presenta y coordina los
Procedimientos de ayuda)

Procedure Ayuda;

Const

 Espacio = 11;
 Linea   = 8;

Var
  Ch     : Char;
  Ops    : Array [1..4] of letrero;
  i,Cont : Byte;

Begin
Ops[1]:='Editor';
Ops[2]:='General';
Ops[3]:='Regresar';
Ops[4]:='Caracteristicas';
GotoxyC27,4);
MarcaC'    A   Y   U   D   A   ');
TextBackgroundCblack);
TextColorCred);
WindowC4,6,76,23);
ClrScr;
  Cont:=1;
  WindowC6,7,74,22);
  TextBackgroundCCyan);
  ClrScr;
  GotoxyC Espacio , Linea);MarcaCOps[1]);
  For i:=2 to 4 do
   Begin
    GotoxyCEspacio * i ,Linea); NormalCOps[i]);
```

```
  End;
 Repeat
      Ch:=ReadKey;
    If Ch = #0 then
   Begin
     Ch:=ReadKey;
     Case Ch of
        DER : Begin
              GotoXy (Espacio * Cont, Linea);
              Normal (Ops[Cont]);
              If Cont > 3 then
                Cont := 1
              Else
                Cont := Cont + 1;
              GotoXy ( Espacio * Cont, Linea);
              Marca (Ops[Cont]);
             End;
        IZQ : Begin
              GotoXy (Espacio * Cont, Linea);
              Normal (Ops[Cont]);
              If Cont < 2 then
                Cont := 4
              Else
                Cont := Cont - 1;
              GotoXy (Espacio * Cont, Linea);
              Marca (Ops[Cont]);
             End;
  End; ( Case )
End  ( If )
Else
 Begin
 Case Upcase(Ch) of
 ENTER : Begin
          Case Cont of
           1 : Ayuda_Editor;
           2 : General;
           3 : Begin
                 TextbackgroundC(Black);
                 ClrScr;
                 Exit;
               End;
           4 : Notas;
           End; ( Case)
          End;
   'E' : Ayuda_Editor;
   'C' : Notas;
   'G' : General;
   'R' : Begin
          TextbackgroundC(Black);
          ClrScr;
          Exit;
         End;
 End; ( Case )
Window(8,7,74,22);
TextBackground(Cyan);
ClrScr;
For i:=1 to 4 do
```

```
   Begin
    GotoxyCEspacio + i ,Linea); NormalCOps[i]);
   End;
  GotoxyC Espacio + Cont, Linea);MarcaCOps[Cont]);
 End;

Until UpcaseCCh) = 'R';
End;


(+++++++++++++++++++++++++++++ EDITAR ++++++++++++++++++++++++++++++++)
(Procedimiento que presenta al editor del sistema y donde tambien
puede elegirse el algoritmo a usar)

Procedure Editar;

Const

  Espacio = 4;
  Columna =  28;

Var

 Ch     :Char;
 Ops    : Array [1..3] of Letrero;
 i,Cont : Byte;

Begin

Ops[1]:='Editar Problema';
If UsaDiscreto then
  Ops[2]:='Algoritmo a usar:   Discreto'
Else
  Ops[2]:='Algoritmo a usar:    Ciclico';
Ops[3]:='Regresar';
GotoxyC27,4);
MarcaC'    E  D  I  T  A  R    ');
TextBackgroundCblack);
TextColorCredD;
WindowC4,6,76,23);
ClrScr;
Cont:=i;
WindowC6,7,74,22);
TextBackgroundCCyan);
ClrScr;
GotoxyC Columna , Espacio);MarcaCOps[1]);
For i:=2 to 3 do
   Begin
    GotoxyCColumna ,Espacio + i); NormalCOps[i]);
   End;
  Repeat
      Ch:=ReadKey;
    If Ch = #0 then
   Begin
     Ch:=ReadKey;
     Case Ch of
```

```
      ABAJO : Begin
               Gotoxy (Columna, Espacio * Cont);
               Normal (Ops[Cont]);
               If Cont > 2 then
                 Cont := 1
               Else
                 Cont := Cont + 1;
               GotoXy ( Columna, Espacio * Cont);
               Marca (Ops[Cont]);
              End;
     ARRIBA : Begin
               Gotoxy (Columna, Espacio * Cont);
               Normal (Ops[Cont]);
               If Cont < 2 then
                 Cont := 3
               Else
                 Cont := Cont - 1;
               Gotoxy (Columna, Espacio * Cont);
               Marca (Ops[Cont]);
              End;
  End; ( Case )
End  ( If )
Else
 Begin
 Case Upcase(Ch) of
 ENTER : Begin
         Case Cont of
          1 : Begin
                TextBackground(black);
                TextColor(White);
                Window(4,8,76,23);
                ClrScr;
                SinSolucion:=False;
                Principal(Max_Col,Correcto);
                Vars[Max_Reng]:='*HOL*';
                If ( ( Correcto = True ) and
                       (Nuevo = False) ) then
                 Begin
                  TablaVacia:= False;
                  Prepara(UsaDiscreto);
                 End;
                ClrScr;
               End;
          2 : Begin
               Gotoxy (Columna ,Espacio * 2);
               If UsaDiscreto then
                  Begin
                    Ops[2]:='Algoritmo a usar:     Ciclico';
                    UsaDiscreto:=False;
                  End
               Else
                  Begin
                    Ops[2]:='Algoritmo a usar:    Discreto';
                    UsaDiscreto:=True;
                  End;
               Marca (Ops[2]);
              End;
```

```
            3 : Begin
                  Textbackground(Black);
                  ClrScr;
                  Exit;
                End;
              End; { Case}
          End;
    'E' : Begin
            TextBackground(black);
            TextColor(White);
            Window(4,6,76,23);
            ClrScr;
            SinSolucion:=False;
            Principal(Max_Col,Correcto);
            Vars[Max_Reng]:='WHOLA';
            If ( Correcto = True ) then
              Begin
               TablaVacia:= False;
               Prepara(UsaDiscreto);
              End;
            ClrScr;
          End;
    'A' :  Begin
             Gotoxy (Columna, Espacio * 2);
             If UsaDiscreto then
                Begin
                  Ops[2]:='Algoritmo a usar:    Ciclico';
                  UsaDiscreto:=False;
                End
             Else
                Begin
                  Ops[2]:='Algoritmo a usar:    Discreto';
                  UsaDiscreto:=True;
                End;
             Marca (Ops[2]);
           End;
    'R' : Begin
            Textbackground(Black);
            ClrScr;
            Exit;
          End;
  End; { Case }
  Window(6,7,74,22);
  TextBackground(Cyan);
  ClrScr;
  For i:=1 to 3 do
   Begin
    Gotoxy( Columna, Espacio * i );
    Normal(Ops[i]);
   End;
  Gotoxy( Columna, Espacio * Cont);
  Marca(Ops[Cont]);
 End;

Until Upcase(Ch) = 'R';
End;
```

```
{**************************** RESOLVER ****************************}
{Procedimiento que resuelve al problema en su totalidad
realizando todas las iteraciones necesarias}

Procedure Resolver;

Begin

GotoXY(27,4);
Marca(' R  E  S  O  L  V  E  R ');
TextBackground(black);
TextColor(white);
Window(4,6,76,23);
ClrScr;
If ( ( TablaVacia = True )or( SinSolucion = True ) ) then
  Begin
    GotoXY(25,8);
    Write('N O   H A Y   D A T O S');
    Tecla;
  End
Else
 Begin
  Advierte;
  Ch:=ReadKey;
  ClrScr;
  If Ch = ' ' then
     Exit;

SolucionaC(NumVueltas,UsaDiscreto,Default[1],Default[2],
        Default[3],Default[4]);
 End;
ClrScr;
End;



{**************************** PIVOTEAR ****************************}
{Procedimiento que resuelve el problema realizando una sola
iteracion de este para poder analizar de que forma va cambiando
el problema}

Procedure Pivotear;

Begin

GotoXY(27,4);
Marca(' P  I  V  O  T  E  A  R ');
TextBackground(black);
TextColor(white);
Window(4,6,76,23);
ClrScr;
If ( ( TablaVacia = True )or( SinSolucion = True ) ) then
  Begin
    GotoXY(25,8);
    Write('N O   H A Y   D A T O S');
```

```
    Tecla;
  End
Else
Soluciona(1,UsaDiscreto,Default[1],Default[2],Default[3],
        Default[4]);
ClrScr;
End;


{######################## MUESTRA TABLA ##########################}
{Procedimiento que muestra la tabla actual del problema}

Procedure Muestra_Tabla;

Begin

Gotoxy(27,4);
Marca('      T  A  B  L  A     ');
TextBackground(black);
TextColor(white);
Window(4,6,76,23);
ClrScr;
If ( TablaVacia = True ) then
  Begin
    Gotoxy(25,8);
    Write('N O   H A Y   D A T O S');
    Tecla;
  End
Else
Imprime(Max_Reng-1);
ClrScr;
End;


{######################## MUESTRA TABLA ##########################}
{Procedimiento que imprime en papel la tabla actual del problema}

 Procedure Imprimir;

Begin

  Gotoxy(27,4);
  Marca('    I M P R I M I R      ');
  TextBackground(black);
  TextColor(white);
  Window(4,6,76,23);
  ClrScr;
  Window(6,7,74,22);
  TextBackground(Cyan);
  ClrScr;
 If ( TablaVacia = True ) then
  Begin
    Gotoxy(24,8);
    Write('N O   H A Y   D A T O S');
    Tecla;
```

```
  End
Else
 Impresion(Max_Reng);
Textbackground(Black);
ClrScr;
End;


{************************** SALIR ******************************}
{Procedimiento que Termina la ejecucion del sistema}

Procedure Salir;

  Begin
  Gotoxy(27,4);
  Marca('   S   A   L   I   R   ');
  TextBackground(Black);
  TextColor(White);
  Window(4,8,76,23);
  ClrScr;
  Window(20,12,60,18);
  TextBackground(Red);
  ClrScr;
     Gotoxy(9,4);
     Write('SALIR DEL SISTEMA  S/N ? : ');
     Repeat
       Ch:=ReadKey;
     Until ( Upcase(Ch) in ['S','N'] );
     TextBackground(Black);
     ClrScr;
     If ( Upcase (Ch) = 'N' ) then
        Exit
     Else
      Begin
       Window(1,1,80,25);
       ClrScr;
       Halt;
      End;
  End;


{************************** EJECUTA ******************************}
{Procedimiento que ejecuta la opcion indicada de acuerdo a la
posicion actual del cursor}

Procedure Ejecuta;

Begin

   Case lugar of

       1: Ayuda;
       2: Pivotear;
       3: Editar;
       4: Resolver;
```

```
      5: Opciones;
      6: Imprimir;
      7: Muestra_Tabla;
      8: Salir;
    End;  { Case }
End;


{**************************** ESPERA ******************************}
{Procedimiento que espera que se presione una tecla para realizar
los procedimientos necesarios}

Procedure Espera;

Var
 Ch : Char;

Begin


 Repeat Until KeyPressed;
 Ch := ReadKey;
 If Ch = #0 then
 Begin
    Ch := ReadKey;
   Case Ch of

     DER: Begin
           Posicion(lugar);
           Normal(Opcion[lugar]);
           If lugar<8 then
             lugar:=lugar + 1
           Else
             lugar:= 1;
           Posicion(lugar);
           Marca(Opcion[lugar]);
          End;
     IZQ: Begin
            Posicion(lugar);
            Normal(Opcion[lugar]);
            If lugar>1 then
              lugar:=lugar - 1
            Else
               lugar:= 8;
            Posicion(lugar);
            Marca(Opcion[lugar]);
            End;
      F2: PonHora;
      F3: Sistema;
  End;  { Case }
 End
Else
 Begin
   Case Upcase(Ch) of

       'S': Salir;
```

```
        'A': Ayuda;
        'T': Muestra_Tabla;
        'E': Editar;
        'R': Resolver;
      ENTER: Ejecuta;
        'O': Opciones;
        'I': Imprimir;
        'P': Pivotear;
      End;  { Case }
  End;
End; { Espera }


{************************** PON MENU ****************************}
{Pone en pantalla todas las opciones del menu}

Procedure PonMenu;

Var
  i : Integer;

Begin
 Posicion(1);
 Marca(Opcion[1]);
 For i:=2 to 8 do
  Begin
    Posicion(i);
    Normal(Opcion[i]);
  End;
End;


{************************** MENU ****************************}
{Coordina todos los procedimientos del sistema incluyendolos en
un menu}

Procedure Menu;

Var
 c : Char;

 Begin

   Window(1,1,80,25);
   TextBackground(Cyan);
   TextColor(black);
   ClrScr;
   PonFecha;
   PonMenu;
   TextBackground(black);
   Window(4,8,76,23);
   ClrScr;
   TextBackground(Cyan);
   TextColor(black);
   Window(25,9,55,20);
```

```
  ClrScr;
  Gotoxy(6,3);MarcaC'FACULTAD DE  CIENCIAS');
  TextBackground(Cyan);
  TextColor(black);
  Gotoxy(8,5);WriteC'Proyecto de Tesis');
  Gotoxy(2,7);WriteC'Optimizador para Programacion');
  Gotoxy(13,8);WriteC'Entera');
  Gotoxy(6,10);WriteC'ENRIQUE MORALES SILVA');
  Gotoxy(14,11);WriteC'1990');
  Tecla;
  TextBackground(black);
  WindowC4,6,76,23);
  ClrScr;
  TextBackground(Cyan);
  Repeat
  WindowC1,1,80,25);
  Espera;
  Until c='0';
End;


(######################## PROGRAMA PRINCIPAL ########################)
(Programa Principal del sistema)

Begin
 Inicia;
 Menu;
End.
```

BIBLIOGRAFIA

BAZARAA, Mokhtar. Programacion Lineal y Fluio en Redes. México, Ed. Limusa, 1981. 539 p.

BORLAND International. Turbo Pascal User's Guide. CA, 1987. 350 p.

BORLAND International. Turbo Pascal Reference Guide. CA, 1987. 493 p.

CANO, Agustin. Programacion Entera (Notas de clase). 1989.

ORCHARD-HAYS, William. Advanced Linear Programming Computing Techniques. Ed. McGraw Hill, 1968. 395 p.

SALKIN, Harvey. Integer Programming. Reading. Massachusettes, Ed. Addison - Wesley, 1975. 537 p.

TAHA, Hamdy. Integer Programming : Theory, Applications and Computations. Ed. New York Academic Press, 1975. 380 p.

APENDICE A

TABLA SIMPLEX

La tabla Simplex es un formato donde se acomodan a los datos de un problema de forma que se pueda trabajar iterativamente con ellos; fué diseñada por Dantzig y su estructura es la siguiente:

$X_1 \ldots X_k \ldots X_n$

X_I	Y_{ij}	$\overline{X_I}$
	$c^k - z^k$	$-Z$

Donde:

- X_i son todas las variables del problema
- X_I son las variables básicas
- X_I son los valores asociados a las variables básicas
- Y_{ij} son los coeficientes de la matriz A
- $c^k - z^k$ son los coeficientes de costo reducido

APENDICE B

METODO SIMPLEX

A continuación se presenta la forma explicita de un problema para que las explicaciones posteriores sean de fácil comprensión.

Forma Explicita:

$$X_i + \sum_{j \in J'} [\, Y_{ij} - (Y_{ik}/Y_{ik})\, Y_{ij} \,]\, X_j$$

$$X_k + \sum_{j \in J'} (\, Y_{ij}/Y_{ik}\,)\, X_j \;=\; \overline{X_i}/Y_{ik}$$

$$-Z + \sum_{j \in J'} [(C^j - Z^j) - (C^k - Z^k)/Y_{ik})\; Y_{ij}]\; X_j$$

$$= \; -Z_o \; - \; (C^k - Z^k)/Y_{ik})\; \overline{X_i}$$

Algoritmo Simplex (Caso de Maximizar)

(0) Problema escrito en forma explicita respecto a una base A^k (factible). Ir a (1).

(1)

- Si $C^j - Z^j \leq 0$, $\forall\, j \in J$, Terminar.

- Si no, Sea $J_1 = \{\, j \in J \mid C^j - Z^j > 0 \,\}$ $(\neq \emptyset)$
  Si existe $j \in J_1$ tal que $Y^j \leq 0$, terminar ya que no

hay solución óptima finita.
Si no, Ir a (2).

(2) Seleccionar $k \in J_s$ tal que:

$$C^k - Z^k = \max_{j \in J_s} \{ C^j - Z^j \}$$

Determinar $i \in I$ tal que

$$\overline{X_i / Y_{ik}} = \min_{i | Y_{ik} > 0} \{ \overline{X_i / Y_{ik}} \}$$

Ir a (3).

(3) Transformar la tabla utilizando a Y_{ik} como "elemento pivote"

Ir a (1).

APENDICE C

METODO DUAL LEXICOGRAFICO

Este método equivale a aplicar el algoritmo Simplex al problema Dual

Dado $\overline{X_I} = (A^I)^{-1} b$

- Si $\overline{X_I} \geq 0$, terminar pués se tiene una solución factible óptima

- Si no, $\exists\, i \in I$ tal que $\overline{X_i} < 0$

$$Y = [C^I (A^I)^{-1}]_i^t$$

$$Y' = Y + \theta [(A^I)^{-1}]_i^t \quad , \quad \theta \geq 0$$

$$W' = b^t Y' = b^t Y + \theta\, b^t [(A^I)^{-1}]_i^t$$

$$= W + \theta \overline{X_i}$$

De lo anterior, conviene incrementar θ lo más que se pueda, pero de manera que se sigan cumpliendo las restricciones del problema dual (Caso de Maximizar) :

$$(A^j)^t Y \geq C^j$$

$$(A^j)^t Y' = \begin{cases} (A^j)^t Y + \theta\, Y_{ij} & j \in J \\ (A^j)^t Y & j \in I,\ j \neq i \\ (A^j)^t Y + \theta & j = i \end{cases}$$

APENDICE D

METODO DE LAS DOS FASES

En el caso de que una vez planteado el problema en la tabla simplex, no se tenga una solución factible básica inicial, se procede a agregar variables artificiales a las restricciones con el fin de obtener variables básicas despejadas, planteándose una nueva función objetivo que es la siguiente:

$$\text{Min } Z = \sum a_i$$

donde ai son las variables artificiales.

De aquí se tiene el problema auxiliar

$$\text{Min } Z = \sum a_i$$

s.c.

$$Ax + Ia = b$$

$$x \geq 0 \quad , \quad a \geq 0$$

Una vez resuelto el problema auxiliar:

- Si Wmin > 0 ⇒ ai > 0 para algún i lo que implica que el problema original no tiene soluciones factibles.

- Wmin = 0

  a. Ninguna variable artificial es básica

Reemplazando la función objetivo original, se tiene una solución factible básica inicial para el problema original iniciándose la segunda fase.

  b. Al menos una variable artificial es básica

- Si existe un pivote en el renglón asociado a α, entonces pivotear e ir a a.

- Si en el renglón asociado a α sólo existen elementos cero, entonces la restricción correspondiente es redundante por lo que se procede a eliminarse y a seguir al paso a.

APENDICE E

TEXTOS DEL PROGRAMA DE PRESENTACION

ARCHIVO "PRESENTA.DOC"

```
Nombre      : TESIS.EXE
Tamaño      : 50287 Bytes
Lenguaje    : Pascal
Compilador  : TurboPascal Ver.5.0


Otros Archivos del Sistema ...

 LEEME.EXE      : Este Archivo
 PRESENTA.DOC   : Archivo Datos
 ENTERA.DOC     : Archivo Datos
 CORTADURA.DOC  : Archivo Datos

Caracteristicas Técnicas:

 Algoritmos
 Implementados : Simplex
                 Dual Lexicográfico
                 Cíclico
                 Discreto


 Número máximo de ...

  Variables por Problema    : 20
```

Restricciones por Problema: 20
Letras en los nombres de
variables : 5
Problemas a Resolver :

Programación Lineal
(Resolución)

Programación Entera
(Análisis y Resolución)

ARCHIVO "ENTERA.DOC"

La Programación Entera es una parte de la Programación Matemática que se encarga de resolver aquellos problemas de optimización donde algunas o todas las variables de desición deben tener valores discretos.

Basicamente existen dos enfoques para resolver este tipo de problemas que son:

- Métodos de Búsqueda
- Métodos de Cortaduras

Los métodos de búsqueda basan su funcionamiento encontrando cotas y después buscando soluciones enteras mejores a la ya obtenida; estas búsquedas a menudo se realizan en árboles variando tanto el número de ramas como la forma en que se obtienen las cotas y en que forma se realizan las búsquedas para este tipo de algoritmos. En el presente sistema se implementan métodos de cortaduras; dichos métodos, que basan su desarrollo al generar restricciones extra (cortaduras), se explican más detalladamente en la opción "Cortaduras" del menú de opciones de este programa.

ARCHIVO "CORTADUR.DOC"

Los métodos de cortaduras son métodos que localizan al óptimo entero (en caso de existir) generando restricciones adicionales al problema original de forma tal que se "cerca" al óptimo; existen varios algoritmos que emplean este enfoque de los que se implementan dos en el sistema y son :

- Algoritmo Discreto
- Algoritmo Ciclico

El algoritmo Ciclico toma como solución inicial una solución óptima continua, como sabemos que ésta corresponde a un punto extremo, se tiene que el algoritmo irá generando restricciones de forma que reducir la región factible hasta determinar el óptimo entero en caso de que éste exista.

El algoritmo Discreto por su parte, comienza con una solución entera factible, es decir, dentro de la región de factibilidad, para generar entonces restricciones adicionales que "cortaran" y reducirán la región factible hasta localizar al óptimo en caso de existir.