UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
ACATLAN

Analizador de Cadenas de Markov con una Base de Conocimiento

T E S I S

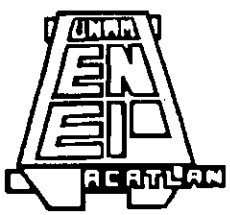
QUE PARA OBTENER EL TITULO DE
LIC. EN MATEMATICAS APLICADAS
Y COMPUTACION

PRESENTAN

DUARTE SILVA AMADA
CAMARGO TIBURCIO CESAR

ASESOR

ACT. MA. DEL CARMEN GONZALEZ VIDEGARAY



NAUCALPAN, EDO. DE MEXICO 1998

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Agradecimientos

A nuestros padres

Que sin su apoyo y comprensión, no se hubiera realizado este proyecto.

Carmen Silva Sánchez.
Juan Duarte Valencia (†).
María Tiburcio Cristino.
Esteban Camargo Alcalá.

A nuestros parientes y hermanos con gratitud por su valiosa paciencia y soportarnos en nuestros malos momentos.

A la Escuela Nacional de Estudios Profesionales Acatlán

Dedicamos este trabajo a la Institución Educativa donde culminamos un periodo de preparación decisivo para la vida.

Con gran afecto a todos nuestros maestros quienes pacientemente y con profesionalismo, vertieron en nosotros los conocimientos que conforman hoy, una disciplina.

En especial a nuestra maestra y asesora de esta tesis : Act. Ma. del Carmen González Videgaray.

Al Instituto Mexicano del Petróleo.

Por el apoyo y los medios que se nos brindó para la realización de esta tesis.

A nuestros amigos

Porque parte importante de este trabajo, ha sido cada una de aquellas personas que nos otorgaron un apoyo, una dirección, un conocimiento y experiencias sobre la marcha, por tanto, para no caer en omisiones involuntarias no mencionaremos los nombres de quienes hicieron tan valiosas aportaciones y solo hemos de expresar que donde fue necesario una ayuda ahí estuvo una persona a la que hoy dedicamos este trabajo.

Para ti con gratitud.

INDICE

Introducción

Los cálculos matemáticos han sido la base de muchos inventos para la humanidad. La observación de los fenómenos de nuestro mundo, aunado al razonamiento lógico, ha permitido modelar y crear herramientas para las actividades humanas, conduciéndonos a mejores condiciones de vida.

La curiosidad del ser humano ha ido más allá de sólo describir cálculos para procesos determinados por un método o una secuencia, ha llegado a desarrollar modelos y métodos para predecir el futuro de un proceso.

Esta tesis está enfocada a los procesos Markovianos, cuyo método proporciona las herramientas de análisis para un proceso estocástico. Se desarrolla alrededor de lo que significa un modelo matemático, los tipos de modelos y los tipos de procesos, sustentándose en ecuaciones obtenidas de el estudio de dichos procesos.

La sistematización de los procesos para el análisis de Cadenas de Markov se ha codificado de forma que la computadora sea capaz de ser una herramienta de análisis.

Se han vislumbrado, en este trabajo, dos vertientes: el planteamiento de modelos matemáticos y la implementación de un sistema en computadora. La primera parte se desarrolla en base a las ecuaciones que representan un proceso probabilístico característico de las cadenas de Markov; y la segunda, al desarrollo integral de un sistema que realice el papel de herramienta de análisis.

Combinar ambos aspectos nos lleva al desarrollo de algoritmos matemáticos, conocimiento del sistema operativo de acuerdo al tipo de equipo, y diversas herramientas que se utilizan en el desarrollo del software.

No es menos importante el planteamiento matemático que el desarrollo del software, pues uno no se explica sin el otro. Por esta razón se dedica el primer capítulo al planteamiento teórico de las Cadenas de Markov explicando los elementos de éstas.

El segundo capítulo se refiere a los elementos teóricos del desarrollo del software, cuyas explicaciones sólo se centran en las metodologías básicas, no por ello, escasas.

Para modelar en forma óptima los procesos de Markov implementados en computadora, se desarrolla el capítulo tres, con el propósito de explicar el tipo de sistema de hardware, software y lógica del programa que se han visto involucrados para el desarrollo de esta herramienta; así como la conformación de una base de conocimiento para servir de ayuda a quienes utilicen esta herramienta.

El análisis de resultados involucra utilizar el software diseñado para obtenerlos. Esta etapa tiene como fin hacer la prueba del software, la comprobación de que el algoritmo integra los elementos esenciales de la teoría de las Cadenas de Markov, para la evaluación de ejemplos prácticos. Estos se resuelven manualmente con lápiz y papel, y se comparan con los resultados proporcionados por la computadora. En el capítulo cuatro se realiza este análisis como parte de la implementación del software.

Con el desarrollo de ejemplos se vislumbra que el programa cumple con los objetivos planteados al inicio del trabajo. Considerando las limitaciones que tiene se establecen las bases para mejorarlo, proponiendo nuevas orientaciones o aplicaciones, así como futuras modificaciones explicadas brevemente en el capítulo cinco.

Capítulo 1

Cadenas de Markov

1.1 Antecedentes

En la evolución de algunas disciplinas científicas existen etapas donde se da el desarrollo de teorías matemáticas que consideran y explican las observaciones generadas por el fenómeno en cuestión. Durante estas etapas, las teorías verbales y cualitativas son reemplazadas por teorías matemáticas y cuantitativas, que se expresan en la forma de algún tipo de ecuación o ecuaciones, un mecanismo postulado, o un modelo que puede generar un conjunto de observaciones teóricas; en general todas estas formas son modelos matemáticos. Estas observaciones teóricas y experimentales pueden entonces ser comparadas para comprobar si el modelo es razonable, es decir, si es capaz de comprender las observaciones experimentales obtenidas bajo las condiciones estipuladas por el modelo.

Los modelos matemáticos se han desarrollado como una abstracción de un problema real. Algunos son tan casuales como el que desarrolló a principios de siglo el matemático ruso llamado Andrei Andreyevich Markov (1856-1922). Este matemático se encontraba investigando la alternancia de las vocales y consonantes que conformaban el poema de Onegin, cuyo autor fue Poeshkin. Para medir dicha alternancia construyó un modelo probabilístico que consistió en sucesivos ensayos para determinar un parámetro que permitiera establecer, para una letra, la secuencia en que volvería a repetirse. Llegó a la conclusión de que las posiciones anteriores de la letra en cuestión no afectaba a la posición futura, salvo que su posición anterior fuera el estado actual. Este modelo, que es la generalización más simple del modelo de probabilidad para ensayos independientes, apareció para darle una excelente descripción a la alternancia de vocales y consonantes, y habilitó a Markov para calcular una estimación muy precisa para las frecuencias en las cuales aparecían las consonantes en el poema de Poeshkin.

El modelo desarrollado por Markov no es la excepción a la regla de que los modelos más simples, frecuentemente son los modelos más útiles para analizar problemas prácticos. Un modelo como el planteado por Markov nos permite representar la incertidumbre en muchos sistemas del mundo real que están involucrados dinámicamente en el tiempo.

El modelo desarrollado por Markov es denominado **cadenas de Markov**.

1.2 Marco de Referencia.

El estudio de las cadenas de Markov se lleva a cabo en dos ámbitos principalmente : como una herramienta en la Toma de Decisiones; y dentro del desarrollo de modelos matemáticos. El enfoque principal aquí es exponer las características de las cadenas de Markov y desarrollar una herramienta que procese dichas cadenas.

Uno de los principales objetivos de las matemáticas es crear modelos que representen situaciones y circunstancias reales, modelos que nos permitan la solución de problemas en forma práctica.

El proceso de creación de los modelos matemáticos involucra una serie de etapas :

Observación del mundo real: El creador de un modelo realiza conjeturas acerca de un problema en el mundo real a través de observaciones como estudio inicial.

Modelo Real: Consiste en plasmar el problema tan preciso como sea posible. Esta etapa involucra hacer ciertas idealizaciones y aproximaciones, con el fin de intentar identificar y seleccionar aquellos conceptos que serán considerados como básicos en el estudio. También se elimina la información innecesaria.

Modelo Matemático: Esta etapa involucra un alto grado de creatividad, se debe revisar el modelo real e intentar identificar los procesos operativos del trabajo. El objetivo es expresar la situación completa del problema en términos simbólicos. Así, el modelo real se transforma en un modelo matemático que comprenderá las cantidades reales y procesos involucrados, como una expresión de símbolos y operaciones matemáticas. Esto no quiere decir que sólo se pueda construir un modelo único para un problema en el mundo real; se pueden tener más de un modelo para el mismo problema y el investigador o creador del modelo elige el que desee.

Conclusiones y predicciones: Se llega a esta etapa una vez que el sistema ha sido transformado en términos simbólicos, el sistema resultante será estudiado con ideas y técnicas matemáticas apropiadas. Como resultado de este proceso se obtendrán teoremas y predicciones que se comprobará con el problema en el mundo real. El fin de esta etapa es producir información acerca de la situación que se está estudiando, el valor agregado son las fórmulas y los teoremas.

Si estas etapas llegan a feliz término, todas las teorías matemáticas se cumplirán en el mundo real ; pero es común que algunas de estas teorías no se cumplan. En tal caso se examina cada etapa del proceso hasta que se afina y corrige, y finalmente se llega a la aceptación de lo postulado.

La fig. 1.1. se muestran gráficamente estas etapas.

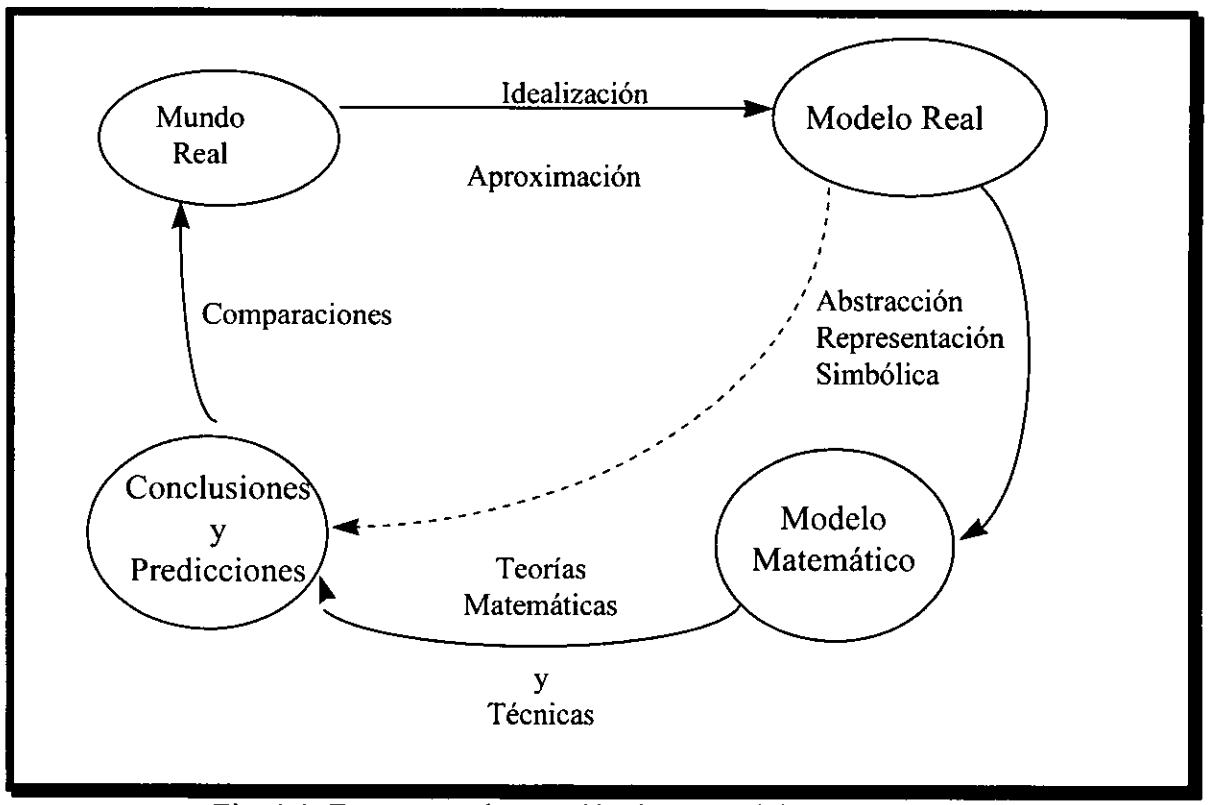


Fig. 1.1. Etapas para la creación de un modelo matemático.

La línea punteada indica una versión abreviada de este proceso y en la práctica es la más utilizada.

A pesar de que este tipo de método no es el único, representa una guía para elaborar un modelo matemático.

Para el estudio de las cadenas de Markov, como un modelo matemático, se mencionan los siguientes tipos de modelos :

> ***Modelo Determinístico*** **:** Un modelo se considera determinístico si dada una información suficiente en un instante en el tiempo o en un estado, pronostica que el futuro comportamiento del sistema de estudio puede ser exactamente determinado.
>
> ***Modelo Estocástico*** **:** Se dice que un modelo es estocástico si éste incorpora un comportamiento probabilístico. Para estos modelos las predicciones son tales que no importa cuánto sabemos acerca del sistema en un tiempo dado, es imposible determinar con absoluta certeza la naturaleza del sistema para tiempos futuros.

Las cadenas de Markov involucran comportamiento probabilístico, así que son modelos estocásticos.

1.3 Tipos de proceso

Un proceso engloba un conjunto de pasos o acciones que se integran para lograr un fin o conformar un procedimiento.

La clase de procesos a estudiar se encuentran dentro de los procesos de decisión, que se llevan a cabo bajo un entorno de incertidumbre y que pueden realizarse bajo [1]:

a) Completa certeza
b) Riesgo
c) Conflicto
d) Completa incertidumbre

A los procesos de decisión bajo completa certeza se les llama **determinísticos,** se caracterizan porque el grupo decisor conoce perfectamente cuál va a ser el estado de la naturaleza relativo a sus objetivos, y por lo tanto, selecciona aquella acción que, de acuerdo al criterio imperante, logrará acercarlos más rápido a la meta preestablecida.

En un **proceso estocástico** no se conoce con exactitud el estado que adoptará la naturaleza, pero se asocia a éste una distribución de probabilidad (continua o discreta). En función a esta última el grupo decisor selecciona aquella acción que optimiza la esperanza de acercarlos a la meta propuesta.

En forma matemática un proceso estocástico se define como una colección subindizada de variables aleatorias denotadas por $\{X_t\}$, donde :

$\{X_t\}$ representa el conjunto de todos los posibles estados en que puede estar el proceso a un tiempo t.

A este conjunto de estados se le conoce como **espacio de estados.** El valor asociado a X_t es el estado del proceso en el tiempo t.. Una **transición** se define como cualquier cambio de estado.

Cuando el espacio de estados contiene valores discretos el *proceso estocástico* se denomina *discreto*, de otra manera se denomina *continuo.*

[1] [Prawda 1991] pág. 27

Los tipos de procesos estocásticos pueden ser :

1) ***Recurrente.-*** Se dice que un proceso estocástico es recurrente si se repite a lo largo del tiempo, es decir, siempre existe la probabilidad de que el proceso sucederá.
Ejemplo : Un día con lluvia.

2) ***Transitorio.-*** Es aquel en el que existe la probabilidad de que no se repita jamás el proceso.
Ejemplo : La niñez.

3) ***Absorbente.-*** Una vez que se alcanza, el sistema permanece ahí para siempre, ya no existe la posibilidad de continuar.
Ejemplo : La muerte de un ser vivo.

4) ***Periódico.-*** Un proceso es periódico si se repite con una frecuencia más o menos regular.
Ejemplo : El clima.

5) ***Estacionario.-*** Son aquellos procesos que se estabilizan después de cierto tiempo.
Ejemplo : Las ventas en negocios.

6) ***Caminata aleatoria.-*** Se llama caminata aleatoria al proceso estocástico en el cual el sistema pueda trasladarse de un estado a otro de uno o varios estados adyacentes.

1.4 Definición de Cadenas de Markov Finitas

Sea un sistema que pueda estar en cualquier estado, siendo el número de estados finito o contable finito. Sea S quien denota este conjunto de estados. Podemos asumir que S es un subconjunto de los enteros. El conjunto S es llamado el espacio de estados del sistema. Sea el sistema observado en momentos discretos de tiempo $n = 0,1,2,...$, y sea X_n quien denote el estado del sistema en el tiempo n.

Tomando en cuenta que el sistema es *no determinístico*, nombremos a las X_n, donde $n \geq 0$, como las *variables aleatorias* definidas en un espacio de probabilidad común.

Hasta aquí poco se puede decir acerca de tales variables aleatorias, es por tanto necesario, definir un modelo sobre ellas.

El modelo más simple es el de *variables aleatorias independientes, el cual* es un buen modelo para los sistemas que muestran, que la probabilidad de los estados futuros son independientes de las probabilidades de los estados presentes y pasados.

Otro modelo, es el que muestran los sistemas cuyas probabilidades de los estados pasados y presentes tienen influencia en las probabilidades de los estados futuros, y en ocasiones, las determinan.

El modelo de nuestro interés es el de los sistemas que tienen la propiedad de que dada la probabilidad del estado presente, la probabilidad de los estados pasados no tienen influencia en la probabilidad de los estados futuros. A esta propiedad se le llama ***propiedad de Markov***, y los sistemas que tienen esta propiedad son llamados ***cadenas de Markov.*** La propiedad de Markov está definida precisamente por los requerimientos de la ecuación:

$$P(X_{n+1} = x_{n+1} \mid X_0 = x_0, \ldots, X_n = x_n) = P(X_{n+1} = x_{n+1} \mid X_n = x_n) \ldots (1.1)$$

Para toda selección de enteros no negativos ***n,*** los números $X_0, \ldots X_{n+1}$, están cada uno en ***S***. Las probabilidades condicionales $P(X_{n+1} = y \mid X_n = x)$ son llamadas ***probabilidades de transición*** de la cadena.

1.4.1 Cadenas de Markov con dos estados.[2]

Con objeto de obtener los resultados generales comenzaremos por considerar cadenas de Markov de dos estados.

Como ejemplo de una cadena de Markov con dos estados, se presenta el siguiente caso: Considere una máquina que al inicio de un día en particular puede presentar cualquiera de estas dos situaciones: estar descompuesta o en condiciones de operación. Asuma que si la máquina está descompuesta en el *n*-ésimo día, la probabilidad de que la máquina se repare con éxito y se encuentre en condiciones de operación al comienzo del siguiente día *(n+1 día)* es **p**.

Asuma también que si la máquina está en condiciones de operación al inicio del *n-ésimo* día, la probabilidad de que presente alguna una falla y se descomponga al inicio del siguiente día (*n+1 día*) es **q**. Finalmente sea $\boldsymbol{\pi}_0$ (0) el vector de probabilidad de que la máquina se descomponga desde el inicio, es decir, desde el 0-ésimo día.

[2] Hoel, pág. 1

Sea 0 el estado que corresponde a la máquina cuando está descompuesta y sea 1 el estado que corresponde a la máquina cuando está en condiciones de operación. Sea X_n la variable aleatoria que denota el estado de la máquina en el tiempo n.

De acuerdo con la descripción mencionada tenemos :

$P(X_{n+1} = 1 \mid X_n = 0) = p$ probabilidad de que la máquina descompuesta en el día n sea reparada y esté en condiciones de operación el día n+1.

$P(X_{n+1} = 0 \mid X_n = 1) = q$ probabilidad de que siendo el día n esté en condiciones de operación y falle el día $n+1$.

y

$P(X_0 = 0) = \pi_0(0)$ probabilidad de que el día inicial esté descompuesta.

Dado que sólo son dos estados: 0 y 1, se sigue inmediatamente que las situaciones restantes se expresan en las siguientes ecuaciones:

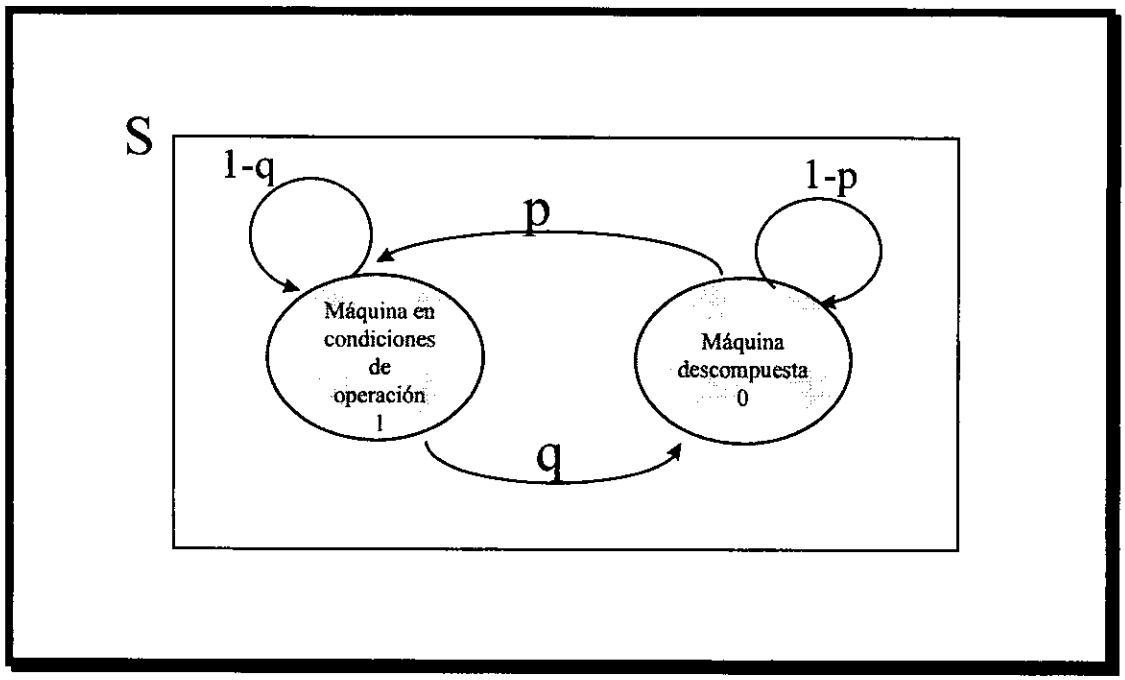
$P(X_{n+1} = 0 \mid X_n = 0) = 1 - p$ probabilidad de estar descompuesta en el día n y seguir descompuesta el día n+1.

$P(X_{n+1} = 1 \mid X_n = 1) = 1 - q$ probabilidad de estar en condiciones de operación en el día n y continúe así hasta el día n+1.

y la probabilidad $\pi_0(1)$ de estar inicialmente en el estado 1 o en condiciones de operación está dada por :

$$\pi_0(1) = P(X_0 = 1) = 1 - \pi_0(0)$$

En forma gráfica tendremos lo siguiente:



Con esta información podemos calcular $P(X_n=0)$ y $P(X_n=1)$

Observemos que

$$\begin{aligned}
P(X_{n+1}=0) &= P(X_n=0 \text{ y } X_{n+1}=0) + P(X_n=1 \text{ y } X_{n+1}=0) \\
&= P(X_n=0)P(X_{n+1}=0 \mid X_n=0) \\
&\quad + P(X_n=1)P(X_{n+1}=0 \mid X_n=1) \\
&= (1-p)P(X_n=0)+qP(X_n=1) \\
&= (1-p)P(X_n=0)+q(1-P(X_n=0)) \\
&= \mathbf{(1-p-q)P(X_n=0)+q}
\end{aligned}$$

Como $P(X_0=0)=\pi_0(0)$,

Entonces para n=0 calculamos

$$P(X_1=0) = (1-p-q)\pi_0(0)+q$$

y

para n = 1 $P(X_2 = 0) = (1 - p - q)\, P(X_1 = 0) + q$

$= (1 - p - q)\,[(1 - p - q)\,\pi_0(0) + q]$

$= (1 - p - q)^2\,\pi_0(0) + q\,[1 + (1 - p - q)]$

De aquí, fácilmente, vemos que repitiendo este procedimiento n veces tendremos

$$P(X_n = 0) = (1 - p - q)^n \pi_0(0) + q\sum_{j=0}^{n-1}(1 - p - q)^j \quad \text{.....(1.2)}$$

En el caso trivial de $p = q = 0$, es claro que para toda n

$$P(X_n = 0) = \pi_0(0) \;\; y \;\; P(X_n = 1) = \pi_0(1)$$

Suponga ahora que $p + q > 0$. Entonces por comparación con una progresión geométrica se obtiene

$$\sum_{j=0}^{n-1}(1 - p - q)^j = \frac{1 - (1 - p - q)^n}{p + q}$$

Concluimos de la Ec. (1.2) que

$$P(X_n = 0) = \frac{q}{p+q} + (1 - p - q)^n\left(\pi_0(0) - \frac{q}{p+q}\right) \quad \text{.....(1.3)}$$

y también

$$P(X_n = 1) = \frac{p}{p+q} + (1 - p - q)^n\left(\pi_0(1) - \frac{p}{p+q}\right) \quad \text{.....(1.4)}$$

Suponga que p y q no son ni igual a cero ni igual a uno. Entonces $0 < p+q < 2$, lo que implica que $|1-p-q| < 1$. En este caso podemos hacer que $n \to \infty$ en las ecuaciones (3) y en (4) y concluimos que :

$$limP(X_n = 0) = \frac{q}{p+q} \text{ y } limP(X_n = 1) = \frac{p}{p+q} \quad \text{.....(1.5)}$$

Podemos obtener las probabilidades $q/(p+q)$ y $p/(p+q)$ mediante una aproximación de diferencias. Suponga que elegimos a $\pi_0(0)$ y a $\pi_0(1)$ de manera que $P(X_n = 0)$ y $P(X_n = 1)$ sean independientes de n. Resulta claro de (1.3) y (1.4) que para hacer esto deberemos escoger las ecuaciones

$$\pi_0(0) = \frac{q}{p+q} \quad \text{y} \quad \pi_0(1) = \frac{q}{p+q}$$

Así vemos que si $X_n, n \geq 0$, comienza fuera de la distribución inicial

$$P(X_0 = 0) = \frac{q}{p+q} \text{ y } P(X_0 = 1) = \frac{p}{p+q}$$

Entonces para toda n

$$P(X_n = 0) = \frac{q}{p+q} \quad \text{y} \quad P(X_n = 1) = \frac{p}{p+q}$$

La descripción de la máquina es vaga debido a que no puede decirse si X_n, $n \geq 0$, satisface la propiedad de Markov. Supongamos, sin embargo, que el comportamiento de este sistema no tiene la propiedad de Markov. Usaremos la información obtenida para calcular la *distribución conjunta* de $x_0, x_1, \ldots, x_n$.

Por ejemplo sea $n = 2$ y sean x_0, x_1 y x_2 cada una igual a 0 o 1 entonces

$$P(X_0 = x_0, X_1 = x_1 \text{ y } X_2 = x_2) = P(X_0 = x_0 \text{ y } X_1 = x_1)\, P(X_2 = x_2 \mid X_0 = x_0 \text{ y } X_1 = x_1)$$
$$= P(X_0 = x_0) P(X_1 = x_1 \mid X_0 = x_0) P(X_2 = x_2 \mid X_0 = x_0 \text{ y } X_1 = x_1)$$

Ahora $P(X_0 = x_0)$ y $P(X_1 = x_1 \mid X_0 = x_0)$ están determinadas por p, q y $\pi_0(0)$; pero sin la propiedad de Markov no podemos evaluar $P(X_2 = x_2 \mid X_0 = x_0 \text{ y } X_1 = x_1)$ en términos de p, q y $\pi_0(0)$.

Si la propiedad de Markov se satisface, entonces

$$P(X_2 = x_2 \mid X_0 = x_0 \text{ y } X_1 = x_1) = P(X_2 = x_2 \mid X_1 = x_1),$$

la cual está determinada por p y q. En este caso

$$P(X_0 = x_0, X_1 = x_1 \text{ y } X_2 = x_2)$$
$$= P(X_0 = x_0)\, P(X_1 = x_1 \mid X_0 = x_0)\, P(X_2 = x_2 \mid X_1 = x_1)$$

Por ejemplo :

$$P(X_0 = 0, X_1 = 1 y X_2 = 0) = P(X_0 = 0) P(X_1 = 1 \mid X_0 = 0) P(X_2 = 0 \mid X_1 = 1)$$
$$= \pi_0(0) pq$$

La siguiente tabla muestra la distribución conjunta de X_0, X_1 y X_2

X_0	X_1	X_2	$P(X_0 = x_0, X_1 = x_1, \text{ y } X_2 = x_2)$
0	0	0	$\pi_0(0)(1-p)^2$
0	0	1	$\pi_0(0)(1-p)p$
0	1	0	$\pi_0(0)pq$
0	1	1	$\pi_0(0)p(1-q)$
1	0	0	$(1-\pi_0(0))q(1-p)$
1	0	1	$(1-\pi_0(0))qp$
1	1	0	$(1-\pi_0(0))(1-q)q$
1	1	1	$(1-\pi_0(0))(1-q)^2$

1.4.2 Función de Transición

Sea X_n, $n > 0$, una cadena de Markov con un espacio de estados $\boldsymbol{S}$. La función $P(x,y)$, $x \in \boldsymbol{S}$ y $y \in \boldsymbol{S}$, definida por

$$P(x, y) = P(X_1 = y \mid X_0 = x),\ x, y \in S \quad \text{.....(1.6)}$$

Es llamada **función de transición** de la cadena. Con las siguientes características

$$P(x, y) \geq 0,\ x, y \in S \quad \text{.....(1.7)}$$

y

$$\sum_y P(x, y) = 1,\ x \in S \quad \text{.....(1.8)}$$

Por las probabilidades de estacionariedad (más adelante veremos la estacionariedad), tenemos :

$$P(X_{n+1} = y \mid X_n = x) = P(x,y), n \geq 1 \quad \text{.....(1.9)}$$

Ahora sigue de la propiedad de Markov que

$$P(X_{n+1} = y \mid X_0 = x_0, \ldots, X_{n-1} = x_{n-1}) = P(x,y) \quad \text{.....(1.10)}$$

En otras palabras, si la cadena de Markov está en el estado x en el tiempo n, entonces no importa como llegó a x, ésta tiene probabilidad $P(x,y)$ de estar en el estado y en el próximo paso. Por esta razón los números $P(x,y)$ son llamadas ***probabilidades de transición de un paso*** de la cadena de Markov.

1.4.3 Distribución Inicial

La distribución inicial nos indica en qué estado principia el sistema en un momento dado, teniendo en cuenta que el estado en que principia es en el tiempo 0, así

La función $\pi_0(x), x \in S$, definida por :

$$\pi_0(x) = P(X_0 = x), x \in S \quad \text{.....(1. 11)}$$

Es llamada la distribución inicial de la cadena. Tal que

$$\pi_0(x) \geq 0, x \in S \quad \text{.....(1.12)}$$

y

$$\sum_x \pi_0(x) = 1 \quad \text{.....(1.13)}$$

1.5 Matriz de Transición

Supongamos que ahora el espacio de estados **S** es finito, sea **S** = {0,1,...,d}. En este caso P es la matriz de transición que tiene d+1 renglones y columnas dadas por

$$P = \begin{bmatrix} P(0,0) & P(0,d) \\ P(d,0) & P(d,d) \end{bmatrix}$$

o bien,

$$P = \begin{bmatrix} p_{00} & p_{01} & p_{02} & \cdots \\ p_{10} & p_{11} & p_{12} & \cdots \\ p_{21} & p_{22} & p_{23} & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

claramente P es una matriz cuadrada (que puede ser de orden infinito debido a que la cadena puede tener un número infinito de estados) con elementos no negativos. Con la suma de los renglones igual a uno, por que $\sum_{j=0}^{\infty} p_{ij} = 1$ para toda i. Una matriz que satisface estas condiciones es llamada ***estocástica*** o *matriz de Markov.*

Una cadena de Markov está completamente definida por una matriz de transición de probabilidades P y un vector columna, digamos $\pi_0 = (\pi_0(1),...\pi_0(d))$, el cual da una distribución de probabilidades para los estados x=0,1,2,3... en el tiempo cero.

1.6 Potencias de Matrices estocásticas

Podemos decir que P^n es una matriz de transición de *n-pasos*. Como las probabilidades de transición pueden ser definidas recursivamente por:[3]

$$p_{ij}^{1} = p_{ij} \qquad\qquad p_{ij}^{n+1} = \sum_{v=0}^{\infty} p_{iv}^{n} p_{vj}$$

Formalmente se define como $P_{ij}^{n} = \Pr\{X_{n+m} = j \mid X_m = i\}$

1.7 Clasificación de estados.

A continuación se nombran algunas propiedades básicas ya que la clasificación puede estar basada en las definiciones que siguen. Estas definiciones son dadas sin tomar en cuenta el número de estados, ya que son comunes a las cadenas de Markov con un número finito o infinito de estados.

Definición: El estado j se dice ser ***accesible*** desde el estado i si j puede ser alcanzado desde i en un número finito de pasos. Si dos estados i y j son accesibles cada uno a otro, se dice que son ***comunicados***. Probabilísticamente estas definiciones implican:

$$\text{i} \rightarrow \text{j (j accesible desde i) si para algún } n \geq 0,\ P_{ij}^{n} > 0$$
$$\text{j} \rightarrow \text{i (i accesible desde j) si para algún } n \geq 0,\ P_{ji}^{n} > 0$$

[3][Bharucha-Reid 1960] pág.12

$i \leftrightarrow j$ (i y j se comunican) si para algún $n \geq 0$, $P_{ij}^{n} > 0$

y para algún $m \geq 0$, $P_{ji}^{m} > 0$

consecuentemente

$i \longrightarrow\!\!\!\!\!\!+\; j$ (j no es accesible desde i) si para algún $n \geq 0$, $P_{ij}^{n} = 0$

$j \longrightarrow\!\!\!\!\!\!+\; i$ (i no es accesible desde j) si para algún $n \geq 0$, $P_{ji}^{n} = 0$

$i \longleftarrow\!\!+\!\!\longrightarrow j$ (i y j no se comunican) si $P_{ij}^{n} = 0$ para toda $n \geq 0$

o $P_{ji}^{n} = 0$ para toda $n \geq 0$

o $P_{ij}^{n} = 0$ y $P_{ji}^{n} = 0$ para toda $n \geq 0$

Como consecuencia de la definición tenemos las siguientes propiedades de esta relación de comunicación:

1.***Reflexividad:*** $i \leftrightarrow j$, para

$$P_{ij}^{0} = \delta_{ij} = \begin{cases} 1 & si \quad i = j \\ 0 & si \quad i \neq j \end{cases}$$

2.***Simétria:*** si $i \leftrightarrow j$, entonces $j \leftrightarrow i$

3.***Transitividad:*** si $i \leftrightarrow j$ y $j \leftrightarrow k$, entonces $i \leftrightarrow k$.

Cuando estas tres propiedades se cumplen en una relación de comunicación se le llama ***relación de equivalencia***.

Definición: Si una cadena de Markov tiene todos sus estados dentro de una clase de equivalencia, se le dice ***Irreducible***.

La clasificación de estados se basa en su naturaleza interna de acuerdo a la siguiente definición:

Sea

$$f_{ii}^{(n)} = P\left[X_n = i, X_r \neq i (r = 1,2,\ldots,n-1) \middle| X_0 = i\right]$$

y

$$f_{ii}^{*} = \sum_{n=1}^{\infty} f_{ii}^{(n)}$$

En otras palabras, $f_{ii}^{(n)}$ es la probabilidad que, comenzando en i, el proceso retorne a i para el primer tiempo en n pasos, y f_{ii}^{*} es la probabilidad que comenzando en i, el retorno al estado inicial ocurra en un tiempo finito.

Estado Recurrente. Un estado se dice que es recurrente si y sólo si, comenzando del estado *i*, el retorno eventual a este estado es seguro.

En términos de probabilidad f_{ii}^{*} implica que el estado i es recurrente si y sólo si $f_{ii}^{*} = 1$. Cuando el estado i es recurrente definimos :

$$\mu_i = \sum_{n=1}^{\infty} n f_{ii}^{(n)}$$

lo cual es una expectativa matemática del número de pasos requeridos para el primer retorno al estado i. El número de pasos requeridos para el primer retorno al mismo estado es llamado tiempo recurrente; μ_i puede entonces ser llamado *tiempo **recurrente principal del estado i**.*

Usando μ_i, un estado recurrente puede ser además clasificado ***nulo recurrente o positivo recurrente***:

a) Un estado recurrente i se dice que es nulo recurrente si y sólo si el tiempo recurrente es ∞ ; esto es, $\mu_i = \infty$.

b) Un estado i se dice que es positivo recurrente si y sólo si el tiempo recurrente es finito; esto es, $\mu_i < \infty$.

Para una cadena de Markov finita, μ_i, i $\notin$ S es siempre finita. Por consiguiente la recurrencia nula es posible únicamente cuando el espacio de estados es contablemente infinito.

Estado Transitorio. Un estado i se dice ser transitorio si y sólo si, empezando del estado i, existe una probabilidad positiva de que el proceso pueda no retornar eventualmente a este estado.

Esto implica $f_{ii}^{*} < 1$.

Expresado de otra manera, en términos de probabilidades $P_{ii}^{(n)}$, la probabilidad que el proceso ocupe el estado i después de n pasos, dado que el estado inicial fue también i (note que el retorno a este estado i no es necesariamente en el primer tiempo), considerando $f_{ii}^{(n)}$ se refiere a la probabilidad del primer retorno a i.

Estado Absorbente. Un estado i se dice ser un estado absorbente si y sólo si $P_{ii} = 1$.

Claramente cuando i es absorbente, $f_{ii}^{(1)} = P_{ii} = 1$, y por lo tanto $f_{ii}^{*} = 1$ y $\mu_i = 1$, mostrando que i es recurrente positivo.

Definición: El período de un estado *i* es definido como el máximo común divisor de todos los enteros $n \geq 1$, para los cuales $P_{ii}^{(n)} > 0$. Cuando el período es 1 el estado es referido como *aperiódico*.

Estado Ergódico. Por razones de conveniencia se llama un estado ***recurrente positivo aperiódico*** a un estado ergódico.

1.7.1 Clasificación de Estados Mediante Algoritmos Gráficos[4]

Las propiedades de los estados nos permiten organizar la matriz de Transición en forma canónica con el fin de simplificar el análisis de la cadena de Markov.

Se tienen tres tareas generales para representar la matriz de transición en su forma canónica:

1. La identificación de Clases de Equivalencia irreducibles.
2. Clasificación de las Clases de Equivalencia como transitorias o recurrentes
3. Finalmente, la jerarquización entre clases de equivalencia.

Para una cadena de Markov con un número pequeño de estados cuya matriz de transición también es pequeña, estas tres tareas pueden realizarse mediante observación o analizando las relaciones de comunicación entre los estados.

Cuando el número de estados es grande, el análisis requiere de un algoritmo que realice estas tareas de manera ordenada. Para establecer un algoritmo de esta naturaleza, se examinará a la matriz mediante la teoría de gráficas.

Definiciones Básicas de Teoría de Gráficas

Una cadena de Markov puede definirse como una gráfica siguiendo el siguiente procedimiento:

[4] [Bhat, U. Narayan (1984)]

Identificar los estados de una Cadena de Markov como el conjunto de vértices (o nodos) de una gráfica, y a las posibles transiciones de un-paso por arcos dirigidos. Así, si existe un arco (i,j) que va del estado i al estado j, la probabilidad de transición $i \rightarrow j$ es mayor que cero. La gráfica obtenida a través de este procedimiento es conocida como **gráfica de Markov** o **gráfica estocástica.**

Ahora se consideran las siguientes definiciones de la teoría de gráficas.

Sea x:{ x_1, x_2,..., x_m,} los vértices (o nodos), y sea $\boldsymbol{a}$ el conjunto de todos los arcos dirigidos entre estos vértices. La gráfica **G**={x,$\boldsymbol{a}$} es una gráfica dirigida (también llamada digráfica).

Un ***camino*** en una gráfica dirigida es cualquier secuencia de arcos donde el vértice final de un camino es el vértice inicial del próximo. Un *camino* a_1, a_2,..., a_k en donde el vértice inicial a_1, es el mismo que el vértice final a_k, es llamado ***circuito.***

El ***grado exterior de*** un vértice x_i está definido como el número de arcos que inician en x_i . El ***grado interno*** de un vértice x_i es el número de arcos que llegan a x_i.

Una **gráfica** se dice **fuertemente conectada** o **conectada** si para cualesquiera dos vértices distintos x_i y x_j existe al menos un camino que va de x_i a x_j. Una ***subgráfica máxima*** de **G** es una subgráfica conectada de **G** la cual no está contenida en cualquier otra subgráfica conectada de **G**. Tal subgráfica es llamada un **componente conectado** de **G**.

Suponga y = $\{y_1,y_2,...,y_k\}$ son componentes conectados de **G**, donde y_i es una subgráfica con vértices $\{x_{i1},x_{i2},...x_{ir_i}\}$, $i=1,2,...,k$, e identifique un vértice correspondiente a cada y_i. Sea $\boldsymbol{c}$ el conjunto de arcos entre estos vértices. La gráfica resultado **G***=(y,c) es llamada ***gráfica condensada*** de **G**.

Dada una gráfica **G**, la matriz de adyacencia A = $\| a_{ij} \|$ está determinada por

a_{ij} = 1 si el arco (x_i,x_j) existe en **G**
= 0 de otra manera

Así los elementos de la matriz adyacente indican la posibilidad de movimiento entre cualquiera dos vértices en un solo paso (movimiento a lo largo de un arco es llamado un paso).

Extendiendo esta definición a más de un paso, definimos la ***matriz de accesibilidad*** como $\boldsymbol{R}$ = $\|r_{ij}\|$, donde

r_{ij} = 1 si x_j puede ser alcanzada desde x_i sin restricciones en el número de pasos que tome

= 0 de otra manera.

Conceptualmente el *i*-ésimo renglón de la matriz de accesibilidad **R** (i.e., si uno empieza desde el vértice x_i) nos da el conjunto de vértices que pueden ser alcanzados en 1,2,3,...,*m*-1 pasos. Claramente, si el número total de vértices es *m*, el máximo numero de pasos a ser considerado es *m*-1. Simbólicamente queremos escribir esto como sigue.

Sea $R^k(x_i)$ el conjunto de vértices que pueden ser alcanzados desde x_i en exactamente *k* pasos y $R(x_i)$, el conjunto de vértices que pueden ser alcanzados eventualmente desde x_i,

Así

$$R(x_i) = \{x_i\} \cup R^1(x_i) \cup R^2(x_i) \cup ... \cup R^{n_i}(x_i)$$

donde n_i es el número más largo de pasos necesarios para esta operación. (La operación se detiene, cuando el total de conjunto de paradas incrementan en tamaño). Sea $n = max_i(n_i)$; usando operaciones boleaneas en la matriz de Adyacencia A (0+1=1, 1+1=1, 1•1=1, 0•1=0, ect), puede mostrarse que los elementos distintos de cero en la i-ésima columna de A^k identifica los vértices en el conjunto $R^k(x_i)$. Así escribimos

$$R = I + A + A^2 + ... + A^n \qquad (1.14)$$

Un algoritmo dado por Warshall(1962) simplifica este procedimiento considerablemente. El algoritmo determina la matriz de accesibilidad $R = R^{(n)}$ en una manera iterativa.
Sea $R^{(k)} = \| r_{ij}^{(k)} \|$. Se determina $R^{(k)}$, k=1,2,3,4,...,n iterativamente como

$$r_{ij}^{(1)} = a_{ij}$$
$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \vee [r_{ik}^{(k-1)} \wedge r_{kj}^{(k-1)}] \text{ para } k > 1 \qquad (1.15)$$

donde $\vee$ y $\wedge$ denotan las operaciones boleanas de adición y multiplicación, respectivamente. Para calcular a mano la segunda ecuación en (1.15) se puede implementar una matriz suplementaria $S^{(k)}$ que representa las operaciones boleanas ubicadas entre los corchetes cuadrados en el lado derecho. Si $r_{ik}^{(k-1)} = 0$, el i-ésimo renglón de $S^{(k)}$ tendrá todos los elementos en cero, y si $r_{ik}^{(k-1)} = 1$, entonces el i-ésimo renglón de $S^{(k)}$ será el mismo que el k-esimo renglón de $R^{(k-1)}$. Ahora $R^{(k-1)}$ es obtenido de la suma boleana de elemento a elemento de las matrices de $R^{(k-1)}$ y $S^{(k)}$. Para un algoritmo de computadora las matrices $S^{(k)}$ no necesitan estar determinadas. La matriz $R^{(k)}$ puede ser directamente obtenida de $R^{(k-1)}$ usando la relación.

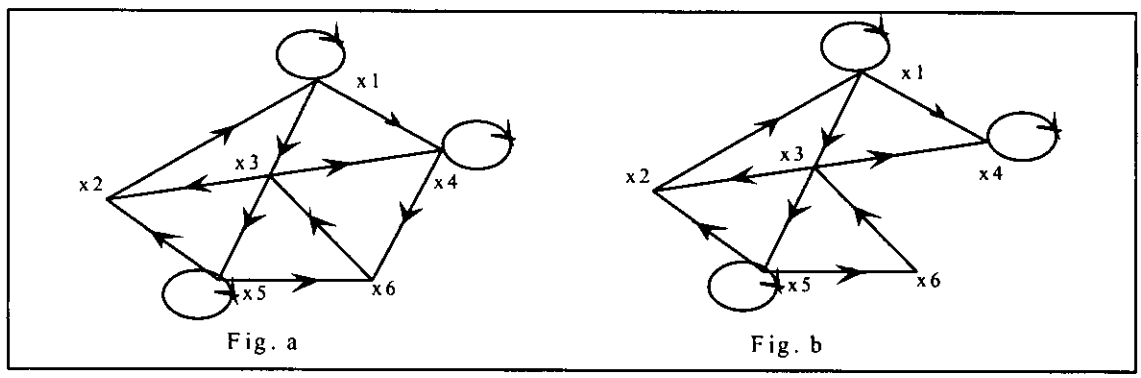
$$r_{ij}^{(k)} \begin{cases} r_{ij}^{(k-1)} & si\ r_{ik}^{(k-1)} = 0 \\ r_{ij}^{(k-1)} & si\ r_{ik}^{(k-1)} = 1 \end{cases} \qquad (1.16)$$

Hemos definido $\mathbf{R}(x_i)$ como el conjunto de vértices de la gráfica **G** que pueden ser alcanzado desde x_i. Ahora definimos $q(x_i)$ como el conjunto de vértices que desde cualesquier x_i puede ser alcanzado. La matriz $\mathbf{Q} = \|q_{ij}\|$, donde $q_{ij}=1$, si x_i puede ser

alcanzada desde x_j , y $q_{ii} = 0$ de otro modo, es llamada la matriz accesible. Claramente tenemos $\mathbf{Q} = \mathbf{R}^T$ (la cual es la transpuesta de R). Así, si uno está interesado en identificar si un par de vértices pueden ser mutuamente alcanzados de uno al otro (i.e., que los vértices que corresponden a los estados, pertenezcan a la misma clase de equivalencia), se debe realizar la multiplicación booleana de elemento a elemento de las matrices **R** y **Q**. Deberemos representar este producto por $\mathbf{R} \otimes \mathbf{Q}$. Los vértices correspondientes a los elementos distintos de cero en los diferentes renglones de esta matriz ahora proporciona los componentes conectados de la gráfica **G**.

Ejemplo

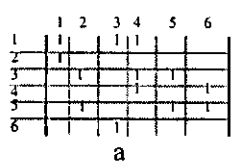
Considere las gráficas representadas en las Figuras a y b. Las propiedades de estas gráficas pueden identificarse como sigue:

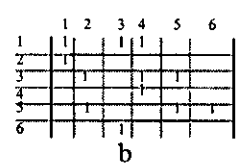


1.Grado exterior de $x_4 = 1$;
Grado interior de $x_4 = 2$
2.Fuertemente conectada

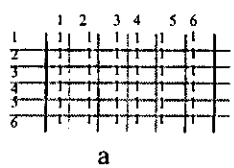
Grado Exterior de $x_4 = 0$
Grado interior de $x_4 = 2$
No es Fuertemente conectada
pero $\{x_1, x_2, x_3, x_5, x_6\}$ es un componente conectado

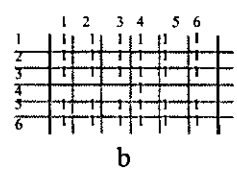
3. Matriz de Adyacencia


a


b

4. Matriz de Accesibilidad


a


b

Cadenas de Markov como Gráficas.

Como se mencionó al inicio, una cadena de Markov puede ser fácilmente representada por una gráfica de Markov donde los vértices representan los estados y los arcos dirigidos representan transiciones de un paso, cuyas probabilidades de transición son distintas de cero. Usando esta representación, las propiedades de equivalencia de la cadena de Markov y la Gráfica de Markov pueden identificarse como sigue:

1. *Cadena de Markov.* Los Estados x_i y x_j están comunicados.
   Gráfica de Markov. Los vértices x_i y x_j son mutuamente alcanzables.

2. *Cadena de Markov:* El conjunto de estados que están mutuamente comunicados con cualquier otro forman una clase de equivalencia.
   Gráfica de Markov: Una subgráfica máxima conectada es llamada una componente conectada.

3. *Cadena de Markov:* Una clase de equivalencia es recurrente si y sólo si no es posible una transición desde un estado que está dentro de la clase a un estado fuera de la clase. Si tal transición es posible ésta es transitoria.

   Gráfica de Markov: Los componentes conectados de una gráfica pueden ser tratados como vértices de una gráfica condensada. Los estados que corresponden a los vértices de un componente y_i pueden estar clasificados como recurrente o transitorio, de acuerdo a la siguiente propiedad: una clase de equivalencia de una cadena de Markov es recurrente si en la gráfica condensada de la gráfica de Markov, el grado exterior del vértice y_i, que representa el correspondiente componente conectado, es cero. La clase de equivalencia es transitoria si el grado exterior es mayor a cero. También las gráficas condensadas no pueden tener circuitos.

Basados en estas propiedades, los estados de una cadena de Markov pueden agruparse en clases de equivalencia al identificar los componentes conectados de la gráfica correspondiente de Markov. Los vértices, correspondientes a los elementos distintos de cero en los distintos renglones de la matriz Boleana $\mathbf{R} \otimes \mathbf{Q}$, donde $\mathbf{R}$ es la matriz de alccesibilidad y $\mathbf{Q}$ es la matriz acesible, nos proporciona los componentes conectados de la gráfica de Markov. Si la clase de equivalencia es recurrente, los vértices correspondientes en la gráfica condensada tienen grado exterior cero. De otra forma, la clase es transitoria, esto es, si x_i, x_j, x_k pertenece a la clase de equivalencia, el i-ésimo, el j-ésimo y la k-ésimo renglón de la matriz de accesibilidad deberá contener elementos distintos de cero en la i-ésima, j-ésima, y k-ésima columnas.

El próximo paso en nuestro análisis es arreglar la matriz de probabilidad de transición en la forma canónica, esto es, en la gráfica correspondiente de Markov establecemos una jerarquía para los componentes conectados.

Un algoritmo simple se obtiene considerando para este propósito, el nivel de jerarquía, como dependiente de la "distancia" de una clase transitoria desde otras clases de equivalencia recurrentes. La "distancia" considerada aquí es el número de pasos correspondientes al camino más largo (camino crítico) del vértice a los vértices que representan las clases recurrentes.

Se ha resaltado desde el principio que (1) la gráfica condensada, la cual tiene componentes conectados para los vértices, no tiene circuitos y (2) los vértices correspondientes a una clase recurrente tienen un grado exterior 0. En una forma canónica las clases de equivalencia recurrentes ocupan niveles más altos en la jerarquía que las clases transitorias. Basados en la información de grado exterior, los niveles pueden identificarse para cada vértice de la gráfica condensada (i.e., para cada clase de equivalencia en la cadena de Markov) como sigue.

Sea $L(y_i)$ el nivel del vértice y_i. Más adelante defininiremos $\lambda(y_i)$ como el conjunto de vértices y_j de la gráfica condensada, tal como el arco (y_i,y_j) que existe. En términos de la cadena de Markov, $\lambda(y_i)$ incluye el conjunto de clases de equivalencia que son accesibles desde la clase correspondiente a y_i. Ahora los niveles de los vértices de la gráfica condensada pueden estar determinadas en la siguiente manera:

1. Todos los vértices con grado exterior cero están a nivel cero; esto es

$$L(y_i) = 0 \text{ si } \lambda(y_i) = \Phi \quad (1.17)$$

2. Para los vértices con grado exterior mayor que cero, el nivel es obtenido como

$$L(y_i) = \left[\min_{y_j \in \lambda(y_i)} \{ L(y_j) \} \right] - 1 \qquad (1.18)$$

Esto deberá estar claro para llevar a cabo las operaciones en la ecuación (1.18.), así los niveles de y_j para $y_j \in \lambda(y_i)$ deberán ser conocidos. Es por tanto necesario comenzar con los vértices con grado exterior cero y entonces proceder con otros vértices vecinos en una forma ordenada. Si la información completa no está disponible para un vértice vecino, otro vértice debe elegirse. Al considerar todos los vértices, es posible llegar a una secuencia de y_i's que permita usar el algoritmo efectivamente.

3. Organizar los vértices de la gráfica condensada (las clases de equivalencia de las cadenas de Markov) en orden decreciente de sus niveles, aquellos con nivel cero se establecen en el tope de la jerarquía. Cuando existan más de una clase recurrente en una cadena de Markov, este procedimiento ayudará a organizar a los estados absorbentes en el tope de la jerarquía de clases; entre los vértices de la gráfica condensada, los correspondientes vértices deberán estar dados los de más alto rango hasta aquellos con nivel cero, (Un vértice y_i corresponde a un estado absorbente si en la matriz de accesibilidad solamente el elemento en la diagonal es diferente de cero en el i-ésimo renglón .)

4. Otros vértices con el mismo nivel pueden arreglarse dentro de cualquier orden entre sí

Es claro, del algoritmo anterior, que el nivel de un vértice es obtenido como el negativo del camino más largo de los vértices en el nivel cero.

1.8 Estacionaridad

Sea $X_n, n \geq 0,$ una cadena de Markov que tiene un espacio de estados S y una función de transición P. Si π (x), $x \in S$, y son número no negativos que suman 1 y si

$$\sum_x \pi(x)P(x,y) = \pi(y), \qquad y \in S, \ldots\ldots\ldots(1.19)$$

entonces π es llamado distribución estacionaria. Supóngase que la distribución estacionaria π existe y que

$$\lim_{n\to\infty} \quad P^n(x,y) \;=\; \pi(y), \qquad y \in S \ldots\ldots(1.20)$$

Entonces, como pronto veremos, que no importa la distribución inicial de la cadena, la distribución de X_n se aproxima a π cuando $n \to \infty$. En tales casos, π es algunas veces llamado distribución de estados permanente (steady state distribution).

Si X_0 tiene la distribución estacionaria π como su distribución inicial, entonces (1.19) implica que para toda n.

$$P(X_n = y) = \pi(y), \qquad y \in S \ldots\ldots(1.22)$$

y por lo tanto la distribución de X_n es independiente de n. Supóngase de manera inversa que la distribución de X_n es independiente de n. Entonces la distribución inicial π_0 es tal que

$$\pi_0(y) = P(X_0 = y) = P(X_1 = y) = \sum_x \pi_0(x)P(x,y) \quad \text{.....(1.23)}$$

Por consecuencia, $\boldsymbol{\pi_0}$ es una distribución estacionaria. En resumen, la distribución de X_n es independiente de n si y sólo si la distribución inicial es una distribución estacionaria.

Sea $\boldsymbol{\pi_0}$ una distribución inicial y $\boldsymbol{\pi}$, una distribución estacionaria. Entonces:

$$P(X_n = y) = \sum_x \pi_0(x)P^n(x,y), \qquad y \in S \quad \text{.....(1.24)}$$

Cuando evaluamos $n \to \infty$ en la fórmula anterior tenemos

$$\lim_{n\to\infty} P(X_n = y) = \sum_x \pi_0(x)\pi(y)$$

Porque $\sum_x \pi_0(x) = 1$, se concluye que

$$\lim_{n\to\infty} \quad P(X_n = y) \quad = \quad \pi(y), \qquad y \in S \quad \text{....(1.25)}$$

La fórmula anterior expresa que dada cualquier distribución inicial, para grandes valores de n la distribución de X_n es aproximadamente igual a la distribución estacionaria π. Esto implica que π es una distribución estacionaria única. Si hubiese otras distribuciones estacionarias podríamos usarla para la distribución inicial π_0. De (1.22) y (1.25) podemos concluir que

$$\pi_0(y) = \pi(y), \quad y \in S.$$

Consideremos un sistema descrito por una cadena de Markov que tiene una función de transición P y una distribución estacionaria única π. Supongamos que empezamos a observar el sistema después de que ha ido cambiado por algún tiempo, se dice que n_0 son unidades de tiempo bajo los enteros positivo n_0. En efecto observamos que $Y_n, n \geq 0$, donde

$$Y_n = X_{n+n_0}, \qquad n \geq 0$$

La variable aleatoria $Y_n, n \geq 0$, también forma una cadena de Markov con una función de transición P. En orden para determinar una probabilidad única para eventos definidos en

términos de una cadena Y_n, es necesario conocer una distribución inicial, la cual es la misma que la distribución de X_{n_0}. En la mayoría de aplicaciones prácticas es muy difícil determinar exactamente esta distribución. Podríamos no tener elección pero asumimos que $Y_n, n \geq 0$, tiene la distribución estacionaria π para esta distribución inicial.

Glosario

Cadenas de Markov : Un sistema que tiene la propiedad de Markov es una cadena de Markov.

Clase de Equivalencia: Son aquellos subconjuntos de estados del total de espacio de estados que forman una relación de equivalencia. Estos conjuntos de estados son irreducibles; es decir; no son subconjuntos de otros subconjuntos de estados

Distribución Estacionaria : Se define una distribución estacionaria cuando ésta es resultante del proceso de calcular la probabilidad en n-pasos de la cadena y ésta dejó de cambiar.

Distribución Conjunta : Es el conjunto de probabilidades posibles que puede presentar una cadena de Markov.

Espacio de Estados ; Es el conjunto de estados que representan todos los posibles estados en que puede hallarse al proceso a un tiempo t o instante dado.

Estado : Situación en que se encuentra el modelo en determinado momento; conjunto de características relevantes del sistema.

Estado Accesible: Un estado j es accesible desde un estado i si el estado j puede ser alcanzado desde i en un número finito de pasos.

Estado Comunicado: Dos estados i y j son accesibles, entonces se dice que son comunicados.

Estado Absorbente: Un estado es absorbente cuando la probabilidad de transición de ir de un estado i al mismo estado i es igual a 1.

Estado Recurrente: Un estado se dice recurrente, si y sólo si, comenzando en el estado i el retorno eventual a este estado es seguro.

Estado Transitorio : Un estado i se dice transitorio, si y sólo si, empezando en el estado i existe una probabilidad positiva de que el proceso pueda no retornar eventualmente a este estado.

Función de Transición : Es igual a la probabilidad condicional que existe de que un estado esté en X_n y pase al estado X_{n+1} en un paso.

Modelo : es una representación cuantitativa o cualitativa de un sistema. Esta representación debe mostrar las relaciones entre los diversos factores que son de interés para el análisis que se está llevando a cabo.

Parámetro : Un parámetro es una constante que puede ser fijada a voluntad ; también puede ser una variable auxiliar (t) que aparece en una expresión paramétrica ; son las constantes del modelo. Los parámetros influyen en las variables de decisión y de estado.

Probabilidades de Transición : Están definidas por las probabilidades condicionales $P(X_{n+1}=y/ X_n=x)$ donde X_{n+1} y X_n pertenecen al mismo espacio de estados.

Proceso: Es un conjunto de pasos o accones que se integran para lograr un fin o conformar un procedimiento.

Propiedad de Markov : En un sistema donde el estado futuro depende sólo del estado actual se dice que tiene la propiedad de Markov.

Relación de Equivalencia: Son aquellos conjuntos de estados que cumplen con las propiedades de una relación de comunicación que son Reflexiva, Simétrica y Transitiva.

Sistema : Un sistema es simplemente un conjunto de componentes que interactúan para alcanzar algún objetivo ; los sistemas son, de hecho, todo lo que rodea al ser humano.

Transición : Es el cambio de un estado a otro, en la matriz estocástica tiene un valor distinto de cero o menor o igual a uno.

Variable de decisión : Son las condiciones que varían o cambian en un proceso.

Variable de estado : Es aquel conjunto de variables, cuyo valor es necesario conocer en un determinado instante de tiempo.

Bibliografía.

Bharucha-Reid A.T. (1960): ELEMENTS OF THEORY OF MARKOV PROCESSES AND THEIR APPLICATION; Ed. McGraw-Hill; New York, USA, Cap. 1.

Bhat, J. Narayan (1984): ELEMENTS OF APPLIED STOCHASTIC PROCESSES; 2ª edición; Ed. Jhon Wiley and Sons; New York; USA

Chover, Josua (1967): MARKOV PROCESS AND POTENTIAL THEORY; Ed. Jhon Wiley and Sons Inc.; New York; USA.

Coleman, Rodney (1976); PROCESOS ESTOCASTICOS; Ed. Limusa; México

Cox, D.R. y Miller, H.D.(1990): THE THEORY OF STOCHASTIC PROCESSES: Ed. Chapman and Hall, Gran Bretaña.

Hoel, Paul G.; Port , Sidney C. y Stone, Charles J. (1972): INTRODUCTION TO STOCHASTIC PROCESSES; Ed. Waveland Press Inc.; Los Angeles, USA. Cap. 1 y 2

Karlin, Samuel y Taylor Howard M.(1975): A FIRST COURSE IN STOCHASTIC PROCESSES; 2ª Edición, Ed. Academic Press; New York, USA.

Philips, Don T. (1969): A MARKOVIAN ANALISYS OF THE CONVEYOR SERVICE ORDERED; University Microfilms, Michigan, USA.

Prawda, Juan (1991): METODOS Y MODELOS DE INVESTIGACION DE OPERACIONES VOL. 2, MODELOS ESTOCASTICOS, Ed. Limusa;México.

Referencias de Internet.

Strang, Gilbert (1980): LINEAR ALGEBRA AND ITS APPLICATIONS; 2ª edición; Ed. Academic Press Inc., New York, USA.

Capítulo 2

Modelo y Diseño de Cadenas de Markov

2.1 Campos de Aplicación.

¿Por qué desarrollar un programa (software) para el análisis de cadenas de Markov ?

En la actualidad el uso de una computadora está siendo tan cotidiano como el teléfono, llegando a ser una poderosa herramienta para solucionar problemas científicos y administrativos, es por eso que ha entrado en todos los ámbitos de la sociedad como : educación, medios de comunicación, automatización de procesos administrativos y contables, para el manejo de negocios, control de productos y el mejor control de grandes volúmenes de información. El Software a utilizar depende del tipo de aplicación a diseñar, para que en la computadora se desarrolle un óptimo trabajo en la actividad que fuese asignada.

La diferencia entre información automatizada y no automatizada, es la precisión y oportunidad de los informes generados por el software (y bases de datos), para tomar las mejores decisiones. Un programa (software) que es diseñado con interfaces amigables para el usuario marca la diferencia entre productos competidores, aún cuando ambos tengan funciones similares. Por esa razón se desarrollan programas computacionales que integran las matemáticas a procesos cotidianos para poder tomar mejores decisiones.

El desarrollo del programa Analizador de cadenas de Markov se realiza tomando en cuenta la teoría de las cadenas de Markov finitas, así como, los conceptos que engloba este tema. Por lo tanto, se pueden adaptar a este modelo de programa, los procesos discretos independientemente de su aplicación. El tema pudiera ser muy extenso si se incluyen procesos continuos cuyo número de estados puede ser muy grande.

Los procesos de Markov discretos como también se les denomina a las cadenas de Markov finitas tienen aplicaciones en :

A)Redes computacionales, etc.

B) Investigación de operaciones

i) Teoría de colas

ii) Proceso de nacimiento y muerte

El desarrollo de cualquier modelo para ser implementado en la computadora involucra una metodología de seguimiento como la creación de algoritmos.

2.2 Algoritmos

En el principio el hombre resolvió sus problemas basándose en juicios intuitivos, en experimentos casuales y más adelante fue sistematizando su forma de realizar su trabajo, de tal forma que surgieron los oficios. La tecnificación del trabajo exige procedimientos adecuados para realizarlo, y similarmente, el surgimiento de las computadoras exigió que se llevara a cabo un cuidadoso y detallado procedimiento para hacer cálculos en ella. Esto nos lleva a la idea de lo que es un algoritmo.

Un algoritmo es una fórmula para resolver un problema, compuesto por un conjunto finito de instrucciones precisas y se ejecutan en un tiempo finito. Cada una de estas instrucciones tienen una secuencia de operación a realizar en un determinado tiempo para resolver un problema específico. Esta concepción de algoritmo es aplicable a todas las actividades que realizamos, pero en realidad tomó significado particular en el desarrollo de programas de computadoras.

2.2.1 Propiedades de un algoritmo

Todo algoritmo es un procedimiento o proceso, pero no todo proceso es un algoritmo, dado que el término proceso se emplea en un sentido muy general para describir una serie de pasos o tareas que no necesariamente están bien detallados; y un algoritmo es un método o secuencia de acciones muy precisas y determinantes para la consecución de un proceso.

Los procesos pueden ser : secuenciales o paralelos.

- Secuencial.- Las acciones que conforman un proceso se realizan una tras otra hasta que se ha ejecutado la última.

- Paralelo.- Dos o más acciones de un proceso se ejecutan en forma simultánea.

2.2.2 Características

Con el fin de precisar la definición de un algoritmo se enlistan sus características:

- ***Finito***.- Cada instrucción deberá ser ejecutada en una cantidad finita de tiempo.
- ***Ausencia de ambigüedad***. Cada instrucción deberá estar perfectamente definida, o bien, cada vez que se procese el mismo conjunto de datos se obtengan los mismos resultados.

- ***Definición de secuencia***.- Un algoritmo debe tener una instrucción inicial única y cada instrucción debe tener un sucesor único para un dato de entrada dado.
- ***Definición de E/S***.- Un proceso con cero o más datos de entrada obtiene uno o más datos de salida.
- ***Efectividad***.- Las instrucciones deben de ordenar tareas posibles de realizar.
- ***Alcance***.- Un algoritmo se aplica a un problema o a una clase de problemas específicos para solucionar.

2.2.3. Utilización en Computadoras

Para las primeras computadoras, el concepto matemático de algoritmo fue definido como una secuencia finita de operaciones, que eran ejecutadas por el procesador de la computadora. En la actualidad las computadoras contienen uno o más procesadores que son capaces de realizar procesamiento paralelo. La tecnología de los microchips ha permitido que además del procesador maestro (CPU) de la computadora, se cuente con procesadores para otros dispositivos como la tarjeta de video, impresoras, etc.

Dentro de los algoritmos tenemos aquellos que se utilizan para realizar procesos secuenciales y procesos paralelos. La forma secuencial de procesamiento consiste en realizar una tarea tras otra en un orden predeterminado; y los algoritmos paralelos realizan tareas, que por su característica de no tener un orden forzoso de ejecución, en paralelo con varios procesadores. Un ejemplo de procesamiento paralelo es el cálculo de votantes en los estados de la República. Se aplica un mismo algoritmo a diversos conjuntos de datos ubicados en distintas áreas geográficas y procesados por varias computadoras.

En la computación los algoritmos están compuestos por acciones que también se les denomina sentencia. El algoritmo es escrito en lenguaje natural y posteriormente son traducidas a un lenguaje de programación que es entendible por la computadora.

Cuando el algoritmo ha sido escrito en el lenguaje que entiende y ejecuta la computadora se le denomina programa.

Un programa en general consiste de dos partes: una descripción de los datos, que son manipulados por la descripción de acciones a ejecutar y el conjunto de acciones que manipularán los datos. Las acciones se describen mediante las **sentencias** y los datos mediante **declaraciones** y **definiciones.**

Estas sentencias se componen a su vez de instrucciones que son las acciones concretas que debe realizar la computadora.

En un algoritmo generalmente se tiene datos de entrada y datos de salida. Los datos de entrada son procesados en el orden del programa y resultan los datos de salida. Los datos de

entrada son proporcionados desde el teclado, de un archivo, por dispositivos de medición o generados por la propia computadora. Los datos de salida son producidos por el proceso descrito en el programa para satisfacer un fin. En un momento dado los datos de entrada provienen de otro programa; pero todo esto depende de la aplicación o resolución especificada en el o los algoritmos.

La representación de los datos para la computadora es binaria; pero para la, o las personas encargadas de desarrollar los algoritmos, deben clasificar los datos en tipos que estarán determinados por el lenguaje de programación.

Entre los tipos más comunes tenemos los numérico, alfanumérico, booleano, carácter, entero, flotante o real, etc.

El lenguaje de programación es la notación que describe las estructuras de datos y los algoritmos. Dentro de los elementos del lenguaje se permiten la declaración de variables.

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

La programación es considerada como un conjunto de actividades y operaciones que realiza el personal informático para implementar en la máquina programas que realicen las funciones previstas en el algoritmo, y siempre se basan en el flujo de información muchas veces representada en los diagramas de flujo de datos(DFD)..

2.3 Diagramas de flujo.

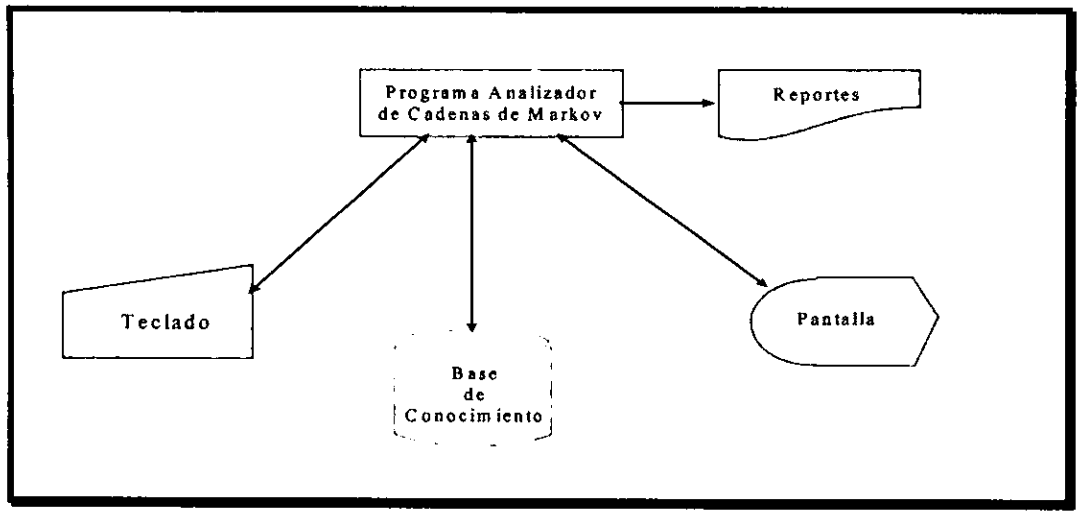
Existen diferentes formas de especificar los algoritmos, pero a menudo la forma más conveniente de especificarlos es en forma gráfica mediante un diagrama de flujo. Esta representación gráfica de la solución a un problema, consiste en los pasos para resolverlo expresados en dibujos determinados como cajas con instrucciones escritas en ellas; símbolos para expresar decisiones, símbolos para especificar salidas y entradas, y el orden en que estas instrucciones han de ejecutarse es indicado por flechas.

Existen tres tipos fundamentales de diagramas de flujo: Diagrama del sistema o de configuración, Diagrama de Macropocesos o Bloques; y Diagrama de Detalles.

A) Diagrama del sistema o de configuración.

Estos diagramas describen el flujo de información entre los distintos soportes físicos de un sistema informático, reflejando las operaciones normales para el desarrollo del proceso.

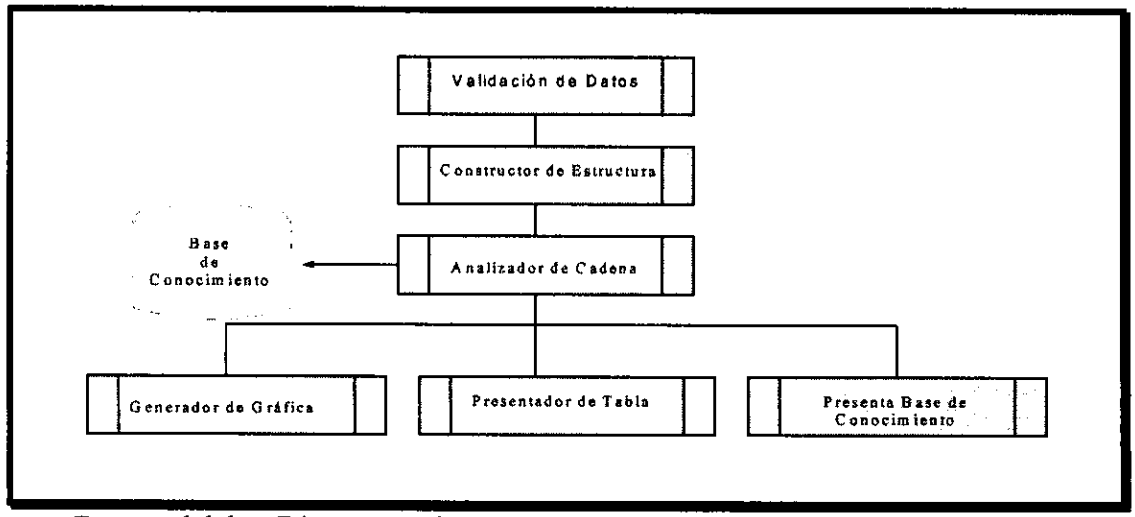
Ejemplo :



B) Diagrama de macroprocesos o bloques.

Representa la estructura en los módulos por bloques en que fue dividido el problema a resolver. También incluye el flujo de información entre los diversos módulos, así como el orden de ejecución de ellos.

Ejemplo :

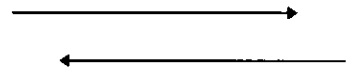


Dentro del los Diagramas de Macroprocesos o bloques también identificamos los Diagramas de Flujo de Datos.

Los diagramas de Flujo de Datos ayudan a ilustrar los componentes esenciales de un proceso y la forma en que interactúan. Sus componentes son: flujo de datos;

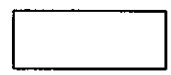
procesos; origen o destino de los datos; y datos almacenados. Cada uno se describe a continuación.

- ***Flujo de Datos:*** Los datos cambian en una dirección específica desde su origen hasta su destino, en forma de un documento, carta, llamada telefónica o en cualquier otro medio.
  El flujo de los datos es un "paquete" o "grupo" de datos y se representa por medio de flechas

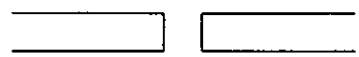


- ***Procesos:*** Los procedimientos o dispositivos, utilizan, producen o transforman datos. Se representan por medio de círculos o "burbujas".

- ***Origen o Destino de los Datos*** : El origen externo o destino de los datos interactúan con el sistema, pero están fuera de su límite; se representan por medio de un rectángulo.



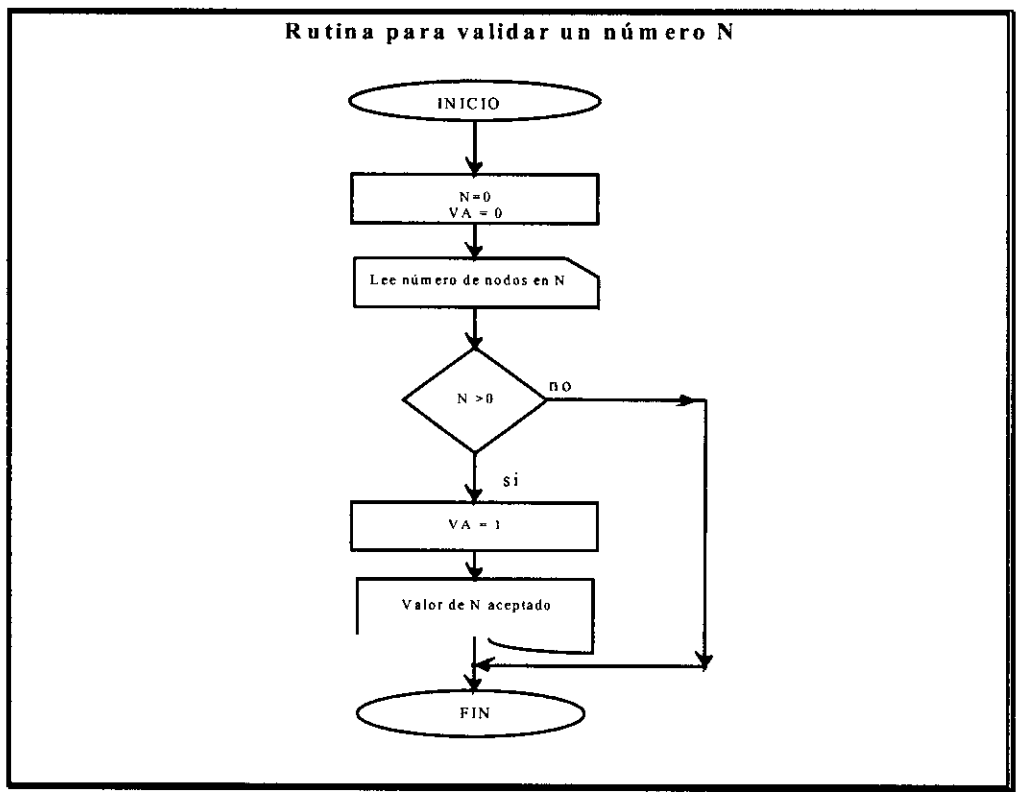
- ***Datos Almacenados***: Los datos se almacenan o se hace referencia a ellos a través de un proceso dentro del sistema. Puede o no representar dispositivos de computadora.Se representan por el símbolo:



C) Diagramas de detalles.

Son las órdenes en secuencia que se deben dar a la máquina para la resolución del problema.

Ejemplo :



2.3.1 Ventajas y desventajas de los diagramas de flujo.[1]

Ventajas :

a) Rápida comprensión de las relaciones.
b) Análisis fácil de las diferentes secciones del programa
c) Útiles como modelos de trabajo en el diseño de nuevos programas y sistemas.
d) Comunicación visual con el usuario
e) Codificación eficaz de programas
f) Depuración y pruebas ordenadas de programas.

[1] Luis Joyanes Aguilar,(1993) P.38.

Desventajas :

a) Los diagramas complejos y detallados suelen ser laboriosos en su planteamiento y dibujo
b) Las acciones que salen de un símbolo de decisión pueden ser difícil de seguir si hay diferentes caminos.
c) No existen normas fijas para la elaboración de diagramas que permiten incluir todos los detalles que el usuario desea introducir.

2.4 Programación estructurada.

La programación estructurada es la disciplina de escribir programas para computadora aplicando ciertos criterios metodológicos básicos para la solución de un problema en concreto.

Los principios metodológicos básicos consisten en dividir el problema en partes más pequeñas para su análisis y de esta forma agilizar el proceso para entender la solución del problema.

En la programación, estas subdivisiones o módulos tiene que cumplir lo siguiente :

1. Estar jerarquizados.
2. Ser pequeños y sencillos
3. Esconder los detalles poco importantes a módulos superiores en la jerarquía.
4. Utilizar módulos de más baja jerarquía como sea necesario para cumplir con 2.
5. Usar las estructuras de datos y control adecuadas para cumplir con el punto 2.
6. Deberán ser legibles, es decir, sean entendibles tanto para el autor como para otros programadores.

El procedimiento para pasar de la descripción de un problema a la generación de un programa estructurado para su solución es el siguiente:

1.- Proponer una solución global al problema en términos de una descripción en un algoritmo que describirá en una forma aproximada, el procedimiento para resolver el problema.

2.- Con base en la solución global se descompone en módulos que se comienzan a refinar progresivamente, tratando de traducir las instrucciones en órdenes que pueda entender la computadora y verificando al mismo tiempo que estén correctas.

3.- Dado que el modelo se desarrolló en varios módulos, se deberá traducir en cada uno de ellos las instrucciones para que la computadora las pueda ejecutar, una vez que todas hayan sido traducidos se prueban y realizan depuraciones.

Dependiendo del tipo de datos se deben desarrollar los procedimientos que permitan su control, y la organización jerárquica de los datos tiene una gran semejanza con la estructura del programa que usa los datos. La estructura de información que se procesa casi siempre predice la estructura del programa, es decir, tiene la misma forma en que fluye la información. Para la obtención de resultados se deben desarrollar los procesos en forma secuencial, así la organización de los mismos son estratificados de acuerdo a su importancia o necesidad.

Algo importante en esta forma de programación es producir una descripción procedimental ; la estructura modular del programa no es considerada explícitamente. Los módulos se consideran un subproducto del procedimiento y a la independencia de módulos se les da poca importancia. Porque al mismo tiempo que se hace la descripción se pueden agrupar los procesos en módulos, y si éstos son considerados en el procedimiento no importaría manejarlos por separado. Pero dependiendo del método a utilizar, la realización de un programa sigue teniendo su base en los datos que procesará.

Cada método de diseño da un conjunto de reglas que facilitan al diseñador la transformación de la estructura de datos en una representación del programa. Dependiendo del método son las reglas, sin embargo, casi siempre se deben cumplir las siguientes tareas :

1. Evaluar las características de la estructura de datos

2. Representar los datos en términos de formas elementales como secuencia, selección y repetición.

3. Transformar la representación de la estructura de datos en una jerarquía de control para el software.

4. Refinar la jerarquía del software usando los criterios definidos como parte del método.

5. Desarrollar una descripción procedimental del programa.

La programación estructurada está caracterizada por tener potentes posibilidades procedimentales y de estructuración de datos. Los lenguajes de programación estructurada proporcionan dentro de sus instrucciones herramientas que orientan al programador a escribir algoritmos en forma estructurada, uno de ellos es Pascal.

Pascal es un lenguaje orientado a procedimientos, mientras que C está orientado a funciones; la diferencia entre un procedimiento y una función no es mucha, ambas permiten expresar un que y un como. Así depende del diseño de la aplicación la elaboración de procedimientos y funciones.

El lenguaje C permite programar estructuradamente a través de funciones; la función indispensable para que un programa en C tenga sentido es la función *main*. Desde este punto de vista el lenguaje de programación C es un lenguaje que tiene los atributos de ser un lenguaje de programación estructurada.

Existen varios lenguajes de programación en el mercado, los aquí mencionados sólo son dos de los más conocidos. Cada lenguaje tiene sus fortalezas y debilidades, el elegir uno involucra varios factores como: disponibilidad del software; disponibilidad del hardware; que sus características permitan programar en forma óptima el algoritmo planteado, etc.

2.5 Programación orientada a objetos

La programación orientada al objeto es una nueva forma de pensar cómo resolver algunos de los problemas planteados en computación. Es decir, en lugar de adaptar un problema a algún aspecto familiar de la computadora, es adaptar a la computadora el problema.

En este tipo de programación, se examina el problema global como un conjunto de ***entidades*** independientes que se relacionan entre sí. Estas entidades se definen porque tienen algún límite físico o conceptual que los separa del resto del problema. Dichas entidades son representadas como objetos en el programa de computadora; es tener una correspondencia de uno a uno entre las entidades del problema físico y los objetos especificados en el programa.

Se puede tener dificultad en elegir objetos porque el proceso puede ser demasiado obvio. Cuando se utiliza un lenguaje de procedimientos tradicional, quien programa, a menudo crea funciones que no se adaptan fácilmente a nuevas situaciones. En estos casos es cuando surge otra forma de programar, porque cuando surgen nuevas necesidades el programa debe diseñarse nuevamente.

Para manejar entidades múltiples en un lenguaje de procedimiento tradicional deberá declararse una estructura de datos capaz de soportar toda la información necesaria para cada entidad .

En C++, la creación de entidades múltiples es muy sencilla : se declara una estructura de datos para cada entidad, y se llama a las funciones con la dirección de una estructura como parámetro.

Cuando se crea una estructura de datos en el lenguaje procedural tradicional se crean normalmente funciones para manipular la estructura, que puedan utilizarse únicamente en esta estructura, otras funciones no sabrían manipularla , ya que no es un tipo de dato incorporado. Es necesario vincular a una unidad la estructura de datos y las funciones que la manipularán. En C++ esta unidad se denomina ***class*** (clase). Las variables, o ***instancias*** de esa ***class*** se les denomina ***objetos.***

Los elementos de datos de un unidad o clase pueden ser privados (private), para que el usuario no pueda manipularlos directamente. Los elementos de datos **Private** pueden únicamente manipularse por medio de funciones especiales (llamadas funciones miembro) que son parte de la **class** o funciones no miembro (funciones amiga). Esto evita una modificación eventual de los datos, y facilita encontrar los errores.

Cada clase tiene dos tipos especiales de función miembro, denominadas : **constructor,** que se encarga de la inicialización cuando se crea un nuevo objeto; y **destructor**, que se encarga de eliminar el objeto de memoria para liberar el espacio utilizado; sucede cuando el objeto ha salido del alcance de la aplicación o la aplicación ha terminado.

Las clases también soportan un desglose lógico de los componentes del código. En un lenguaje orientado al objeto, se pueden heredar las características de un tipo definido por el usuario (**class**) a otro.

Cuando una clase derivada hereda una clase base, los objetos de la clase derivada retienen, el número de miembros en la clase base.

2.6 Lenguajes.

Los lenguajes de programación son la herramienta que nos permite estructurar y comunicar a la computadora un algoritmo. Nos permite producir software para resolver problemas en computadora. Generalmente tenemos

Software
Hardware

el software sobre el hardware. La programación a nivel hardware es a nivel físico, a nivel de programación de circuitos lógicos; y la programación a nivel software es totalmente a nivel lógico.

"El concepto de proceso computacional, también llamado proceso algorítmico o algoritmo es fundamental para la ciencia de la computación. Al ejecutar estos procesos, los

computadores son capaces de resolver problemas; por el contrario, un computador no puede resolver un problema que no tenga una solución algorítmica."[2]

El lenguaje de programación es la forma intermedia que es entendible para las personas que realizan programas como para las computadoras que los procesan. Tiene como características: un vocabulario limitado, una gramática definida explícitamente y reglas bien formadas de sintaxis y semántica.

Las clases de lenguajes que se utilizan en la actualidad son los lenguajes máquina, los lenguajes de alto nivel y los no procedurales . A los lenguajes se les suele clasificar en términos de niveles. El nivel de un lenguaje de programación es indirectamente proporcional al número de instrucciones necesarias para realizar una tarea específica. Los lenguajes de bajo nivel están más cerca de la máquina que los lenguajes de alto nivel que están más próximos al usuario.

Se tienen principalmente tres clases de lenguajes:

- Lenguaje de máquina.
- Lenguaje ensamblador
- Lenguajes de alto nivel.

2.6.1 Lenguaje máquina

Los lenguajes máquina son una representación simbólica de un conjunto de instrucciones de la unidad de procesamiento central de una computadora. Con el lenguaje de máquina se puede hacer uso extremadamente eficiente de la memoria y optimizar la velocidad en la ejecución del programa.

Un programa máquina es un conjunto de instrucciones de máquina escritas con ciertas reglas sintácticas que constituyen el lenguaje máquina, sólo es entendible por la computadora. Dichos programas son largos y difíciles de escribir.

Los programas escritos en lenguaje máquina fueron los primeros que aparecieron y se les denominó lenguajes de primera generación, son difíciles de interpretar y corregir. Estos lenguajes sólo sirven para ciertas máquinas o procesadores.

2.6.2 Lenguaje ensamblador

El lenguaje ensamblador utiliza símbolos reconocibles llamados mnemotécnicos para representar las instrucciones y son considerados como lenguajes de 2a. generación. Son

[2] J. G. Glenn BrookShear (1979) p.1

onsiderados en los lenguajes de bajo nivel. El uso de este tipo de lenguajes ha ido decreciendo notablemente con respecto a los de alto nivel.

Este tipo de lenguajes es distinto en su sintaxis para cada arquitectura de los microprocesadores como el **Z-80, 6502, 8088, 6800** etc., debido a que son producidos por diferentes compañías. Por ejemplo para un microprocesador con un bus de trasferencia de datos de 16 bits, una instrucción será : 1000110000110001.

2.6.3 Lenguajes de alto nivel

La ventaja de los lenguajes de alto nivel es permitir al programador y al programa independizarse de la máquina. Cuando se utiliza un traductor sofisticado, el vocabulario, la gramática, la sintaxis y la semántica de un lenguaje de alto nivel pueden ser mucho más elaborados que los lenguajes máquina. Los compiladores e intérpretes de los lenguajes de alto nivel producen lenguaje máquina como salida.

Estos lenguajes simplifican en gran medida el trabajo de programación, esto es, porque cada instrucción del lenguaje de alto nivel equivale a muchas instrucciones de los lenguajes de bajo nivel, y son más fáciles de entender por el programador.

Hoy en día se usan cientos de lenguajes de programación, pero más de una decena de lenguajes de alto nivel son aceptados por los que desarrollan software comercial. A treinta años de la aparición de los lenguajes como COBOL y FORTRAN siguen siendo vigentes en la actualidad. Hay otros lenguajes de programación más modernos como PASCAL, C y ADA que son utilizados más ampliamente. Los lenguajes orientados a objetos como C++, OBJET PASCAL, EIFFEL y otros , están ganando más seguidores en el desarrollo de sus proyectos.

Los lenguajes especializados como APL, LISP, OPS5 y lenguajes descriptivos para redes neuronales artificiales, aumentan su aceptación conforme las aplicaciones pasan del laboratorio a la práctica.

C++ es un lenguaje orientado a objetos es un C mejorado, que permite encubrir algunas complejidades del programa. Una forma de programar en Windows es usando el lenguaje C++ y el programa analizador de Cadenas de Markov está programado para funcionar en ambiente Windows.

2.7 La Base de Conocimiento.

La base de conocimiento es una base de datos que posee una información y unas reglas específicas sobre una materia determinada. Se puede considerar la base de conocimiento como una lista de objetos con sus atributos asociados.

Se define al **objeto** como la conclusión que es definida por sus reglas asociadas y el **atributo** comouna cualidad específica que, con su regla, ayuda a definir el objeto.

En el sentido más simple (y para muchas aplicaciones), la regla que se aplica a un atributo determina si un objeto "tiene" o "no tiene" dicho atributo. Así pues, se puede definir un objeto, usando una lista de atributos que el objeto posee o no.

La información, en los sistemas expertos y de inteligencia artificial, se representa como hechos que se expresan en forma lógica.[3]

"En general los sistemas de este tipo se forman de hechos o reglas que no aparecen de manera explícita en la base de datos y no son utilizados para agilizar una consulta. Pueden utilizarse las reglas para responder a consultas que no se pueden expresar en los lenguajes de consulta estándar (como SQL).Con los lenguajes de consulta de base de datos sólo es posible obtener información almacenada en la base de datos. En una base de conocimientos, puede hacerse una consulta para obtener metadatos, es decir, datos acerca de los datos."[4]

2.8 Metodología de la programación

La metodología de la programación comprende las siguientes etapas :

a) ***Toma de datos***.- El analista o programador tiene que tener la suficiente información o una descripción clara y detallada de la aplicación a desarrollar; con este fin en el primer capítulo de este trabajo se maneja la teoría de las cadenas de Markov para desarrollar un programa computacional.

b) ***Modularización*** .- Consiste en descomponer sucesivamente el problema en módulos o subprogramas cada vez más concretos y detallados. Los módulos se desarrollarán independientemente y después se unirán para formar el programa.

c) ***Representación gráfica de las operaciones a realizar***.- Esta etapa consiste en realizar una representación gráfica clara y detallada de la secuencia en que deben ser ejecutadas las diferentes operaciones en la máquina.

d) ***Codificación en un lenguaje de programación***.- Una vez que el diagrama de flujo o el algoritmo de resolución de problema está ya definido, se pasa a la fase de codificación del programa en el lenguaje escogido y la obtención del código fuente.

[3] Henry F.Korth (1992) P.452
[4] Ibidem P.460

e) ***Preparación de un conjunto de datos***.- Es necesario un conjunto de datos para probar el programa cuando se ejecute.

f) ***Ejecución y corrección de errores del programa***.- Una vez armados los algoritmos en lenguaje natural o diagrama de flujo, se realizarán las fases de la traducción del programa al lenguaje elegido y se realizan las pruebas de ejecución en la computadora.

g) ***Puesta a punto final del programa***.- El programa se considera terminado cuando se han realizado pruebas, y su fiabilidad con el conjunto de datos de prueba y otros conjuntos similares, y ya no se encuentren errores de ningún tipo.

h) ***Documentación del programa***.- Es todo aquel material escrito y que se fue produciendo simultáneamente en la elaboración del programa. La puesta a punto final del programa y la documentación se desarrolla al mismo tiempo. El tipo de documentación es: Documentación para el programador y documentación para el usuario.

La documentación para el programador debe incluir toda la información técnica del programa o conjunto de programas que conforman una aplicación computacional. Y debe incluir como mínimo: Algoritmos y diagramas de flujo de los diferentes módulos de la aplicación, listados del programa de la aplicación y definición de variables y ficheros de cada módulo.

La documentación para el usuario debe incluir una descripción general de las tareas que realiza el programa, así como una descripción detallada de todas las instrucciones que sean necesarias para su instalación, puesta en marcha y funcionamiento; así como consejos, recomendaciones de uso, explicación de los mensajes de errores y modo de solucionarlos

2.9 Paradigmas de la Ingeniería del Software[5]

La ingeniería del software surge de la ingeniería de sistemas y de hardware. Abarca tres elementos clave: métodos, herramientas y procedimientos.

Los métodos de la ingeniería del software indican "cómo" construir técnicamente el software, los métodos abarcan un amplio espectro de tareas que incluyen: planificación y estimación de proyectos; análisis de los requisitos del sistema y del software; diseño de estructuras de datos; arquitectura de programas y procedimientos algorítmicos; codificación, prueba y mantenimiento. Los métodos de la ingeniería del software

[5] Pressman Roger S.[1993] p.25 -39

introducen frecuentemente una notación especial orientada a un lenguaje o gráfica y un conjunto de criterios para la calidad del software.

Las herramientas de la ingeniería del software suministran un soporte automático o semiautomático para los métodos.

Los procedimientos de la ingeniería del software son el pegamento que une los métodos y las herramientas y facilita un desarrollo racional y oportuno del software de computadora. Los procedimientos definen la secuencia en que se aplican los métodos, las entregas que se requieren, los controles que ayuda a asegurar la calidad y coordinar los cambios, y las directrices que ayudan a los gestores del software a evaluar el progreso.

La ingeniería del software está compuesta por una serie de pasos que abarcan los métodos, las herramientas y los procedimientos antes mencionados. Estos pasos se denominan paradigmas de la ingeniería del software. La elección de un paradigma para la ingeniería del software se lleva a cabo de acuerdo a la naturaleza del proyecto y de la aplicación. Como lo señala Roger en su trabajo, los paradigmas ampliamente debatidos y tratados son:

- El ciclo de vida clásico.
- Construcción de prototipos.
- Modelo en espiral.
- Técnicas de cuarta generación.
- Así como la combinación de paradigmas.

El ciclo de vida clásico

Algunas veces llamado "modelo en cascada", el paradigma del ciclo de vida exige un enfoque sistemático y secuencial del desarrollo del software que comienza en el nivel del sistema y progresa a través del análisis, diseño, codificación, prueba y mantenimiento.

Abarca las siguientes etapas:

Ingeniería y análisis del sistema: se establecen los requisitos de todos los elementos del sistema.

Análisis de los requerimientos del software: A través de conocer el ámbito de la información el analista determina la naturaleza de los programas a realizar. La recopilación se intensifica en este punto.

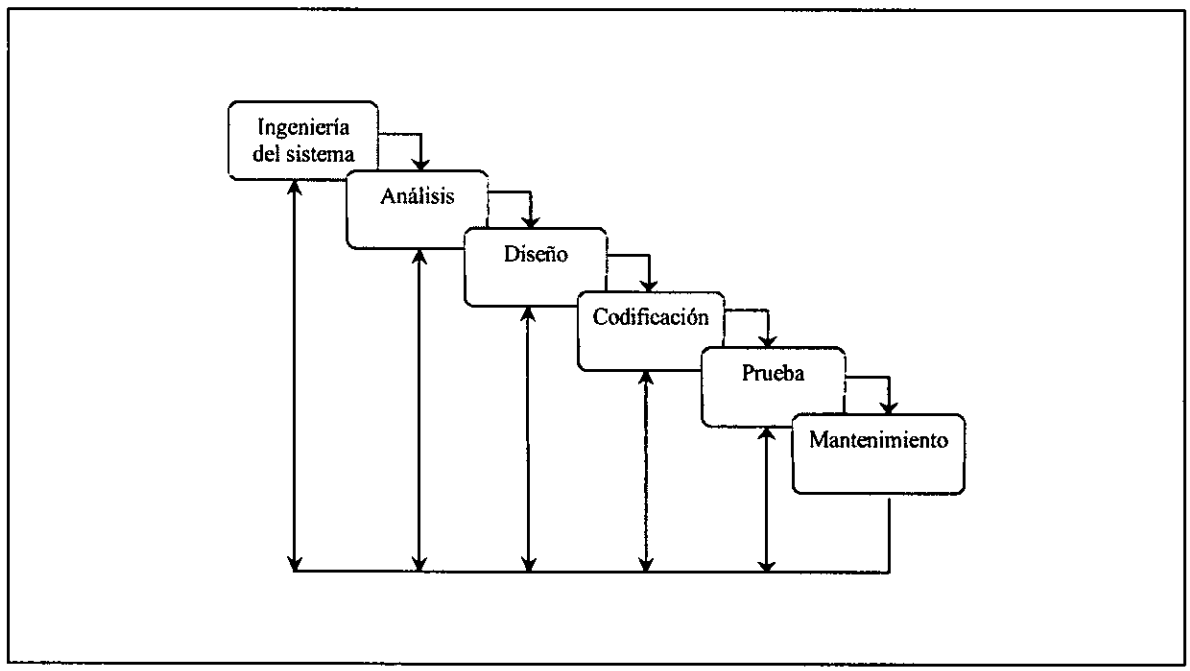
Diseño: En el proceso de diseño se traducen los requisitos en una representación del software que pueda ser establecida de forma que obtenga la calidad requerida antes de que comience la codificación.

Codificación. En esta etapa el diseño se traduce en forma legible para la máquina. Cuando más detallado sea el diseño más fácil será realizar la codificación.

Prueba. Una vez que se ha generado el código, comienza la prueba del programa. Esta se centra en la lógica interna del software, asegurando que todas las sentencias se han probado. En las funciones externas se realizan pruebas que aseguren que la entrada definida produzca los resultados que se requieren.

Mantenimiento. El software por lo general requerirá cambios después de que se entregue al cliente.

El ciclo de Vida Clásico

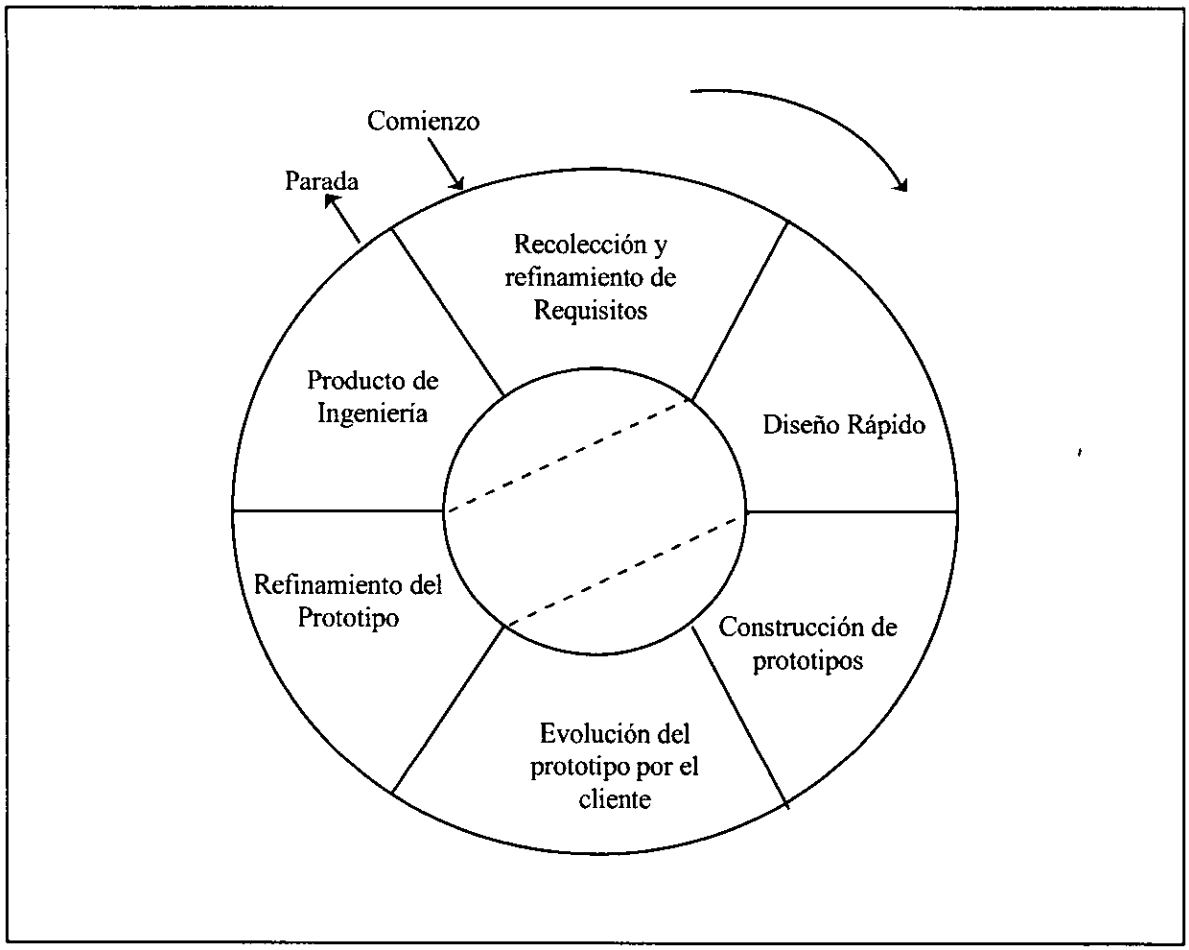


Construcción de prototipos

La construcción de prototipos es un proceso que facilita al programador la creación de un modelo de software a construir.

El modelo tomará una de las tres formas siguientes: (1) un prototipo en papel o un modelo basado en PC que describa la interacción hombre-máquina, de forma que facilite al usuario la comprensión de cómo se producirá tal interacción; (2) un prototipo que implemente algunos subconjuntos de la función requerida del programa deseado; o (3) un programa existente que ejecute parte o toda la función deseada, pero que tenga otras características que deban ser mejoradas en el nuevo trabajo de desarrollo.

La figura muestra las etapas para la elaboración de prototipos



El objetivo de los prototipos tiene la finalidad de identificar plenamente las necesidades del cliente, cuando éstas son poco claras y difíciles de explicar, y a partir de aquí elaborar el diseño adecuado a sus necesidades.

Modelo en Espiral

El modelo en espiral para la ingeniería del software ha sido desarrollado para cubrir las mejores características, tanto del ciclo de vida clásico, como de la creación de prototipos, añadiendo al mismo tiempo un nuevo elemento: el análisis de riesgo, que falta en aquellos paradigmas.

Las actividades principales que involucra son cuatro:

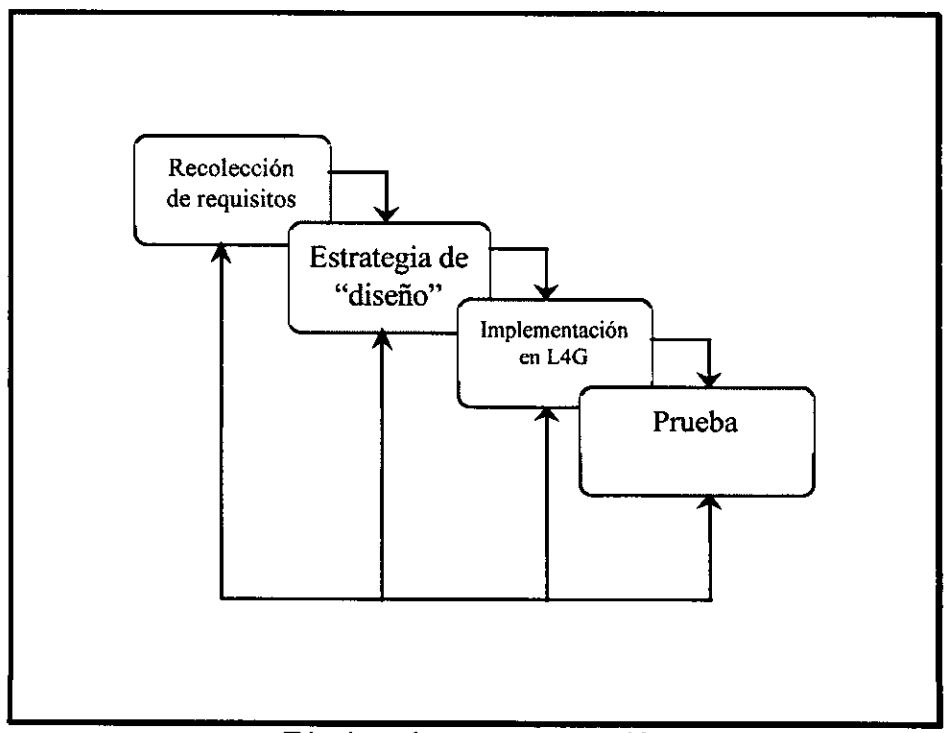
1.-Planificación. Determinación de Objetivos, alternativas y restricciones.
2.-Análisis de Riesgo. Análisis de alternativas e identificación/resolución de riesgos.
3.-Ingeniería. Desarrollo del producto de "Siguiente Nivel"
4.-Evaluación del cliente: valoración de los resultados de la ingeniería.

Técnicas de cuarta generación

Las "técnicas de cuarta generación" (T4G) abarca un amplio espectro de herramientas de software que tienen algo en común: todas facilitan, al que desarrolla el software, la especificación de algunas características del software a alto nivel. Entonces, la herramienta genera automáticamente el código fuente basándose en la especificación del técnico.

Esta técnica puede incluir algunas o todas las herramientas siguientes: lenguajes no procedimentales, manipulación de datos, interacción y definición de pantallas, generación de código, facilidades gráficas de alto nivel y facilidades de hoja de cálculo.

Es recomendable para aplicaciones pequeñas, dado que el cliente puede no estar seguro de lo que necesita; puede ser ambiguo en la especificación de hechos que le son conocidos y puede no desear o ser incapaz de especificar la información en la forma en que una herramienta T4G puede aceptarla.



Técnicas de cuarta generación

Combinación de Paradigmas

Frecuentemente, se describen los paradigmas de la ingeniería del software, tratados en las secciones anteriores, como métodos alternativos para la ingeniería del software en lugar de como métodos complementarios. En muchos casos, los paradigmas pueden y deben combinarse de forma que puedan utilizarse las ventajas de cada uno en un único proyecto.

Por ejemplo, cuando el usuario no tiene bien definido el tipo de sistema que necesita, se recurre a la creación de prototipos con la información recopilada, y con éste método obtener más datos y realizar una mejor especificación de requerimientos para el *análisis de requerimientos del software* en el *ciclo de vida clásico.*

Otro ejemplo, es cuando se aplican técnicas de cuarta generación. Se obtienen resultados más exitosos si se realizan las tres primeras etapas del *ciclo de vida clásico*, porque se tiene una especificación ordenada del sistema, y, a partir de este paso se pueden modelar en herramientas 4GL; de otra forma se estaría experimentando con diseños hasta llegar al ideal.

En resumen, la aplicación de los paradigmas dependen en gran medida de las herramientas con que se disponen, incluyendo la habilidad del analista, y el tipo de sistema a realizar.

2.10 Modelo del Sistema

El tema anterior abarca una forma de realizar una reingeniería del software por medio de paradigmas y muestran una guía de pasos para el desarrollo de nuestro software Analizador de cadenas de Markov, como paradigma base elegimos el ciclo de vida clásico. Comenzamos por definir el objetivo del sistema:

Objetivo: Crear un programa Analizador de cadenas de Markov con una Base de conocimiento.

Requerimientos Identificados:

1.-Representar, en estructuras de datos, una Cadena de Markov.

2.-Calcular la probabilidad en n pasos de ir de un estado i a un estado j.

3.-Conocer las características de la matriz y el vector inicial, indicar cuando es estacionaria la cadena, los distintos estados que la componen, i.e., estados recurrentes, transitorios, y absorbentes; y si la cadena es reducible o no.

4.-Representar gráficamente la cadena de Markov para el número de nodos que se puedan distribuir en la pantalla.

5.-Imprimir la matriz estocástica, vector inicial y vector final.

6.-Proporcionar información acerca del comportamiento de la cadena de Markov.

7.-El programa proporcione módulos para :

-Entrada de Datos
-Salida gráfica
-Salida Impresa
-Evaluación de la cadena
-Simulación de la Cadena conforme transcurre un tiempo.
-Base de conocimientos como sistema de ayuda

8.-El programa funcione en una PC con el sistema de Windows 3.11 o superior.

9.-Posibilidad de modificar los datos del vector inicial y la matriz estocástica en forma interactiva.

10.-Recuperar la cadena de un Archivo y poder guardarla

Procedimientos Identificados

Entrada de Datos: Consiste en recibir el número de nodos de la cadena, los elementos de la matriz estocástica y el vector inicial.

Cálculo de Probabilidades de n-pasos para la Matriz Estocástica: Realizando la multiplicación de matrices, cada elemento de la matriz resultante denota la probabilidad de ir de un estado a otro en un número finito de pasos, dependiendo de la potencia de la matriz.

Clasificación de Estados: Se realiza un examen de los estados, determinando con quien están conectados, si forman circuitos, para determinar su naturaleza.

Elaboración de Gráfica de Estados: En base al número de estados y a las probabilidades de transición se dibuja la gráfica de estados de la cadena.

Consulta a la Base de Conocimientos: Se realizan consultas en un diccionario de términos.

Análisis para determinar si la matriz es reducible o no: Se examinan los circuitos de la matriz estocástica, para determinar las clases de equivalencia.

Identificación de Relaciones de Equivalencia.

Alcances del Sistema

Los alcances son definidos en los siguientes enunciados:

- Por ser un programa académico se delimita al uso de éste en Computadora Personal.
- Se aprovecha la Interfaz gráfica que proporciona el ambiente Windows para la realización de gráficas.
- Los ejemplos que se evalúan son un número máximo de 20 estados, ya que el rendimiento decrece conforme aumenta el número e esados.

Controles

Los controles son los que nos permiten regular el acceso o las condiciones en que debe operar el sistema. Así tenemos que:

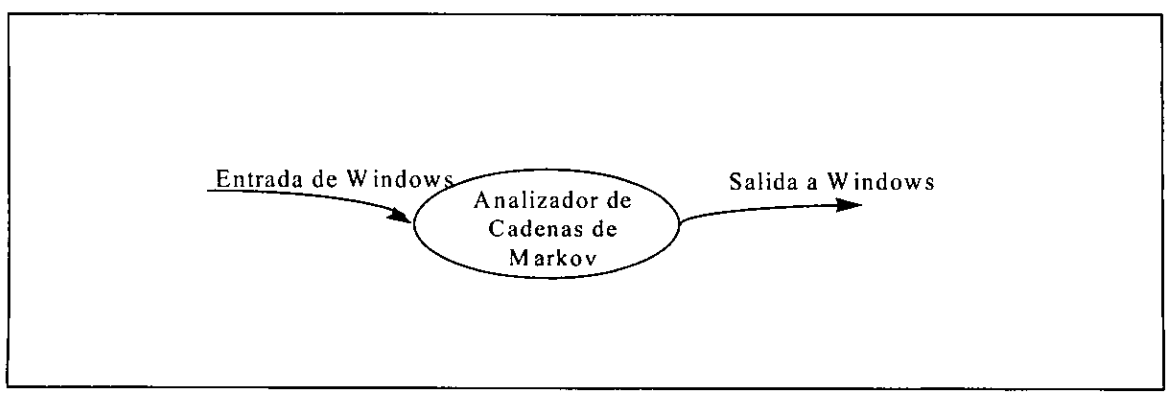
- Una cadena de Markov puede ser introducida mediante el teclado
- Recuperada de un Archivo que previamente se elaboró por este programa
- Las gráficas no exederán a más de 20 nodos

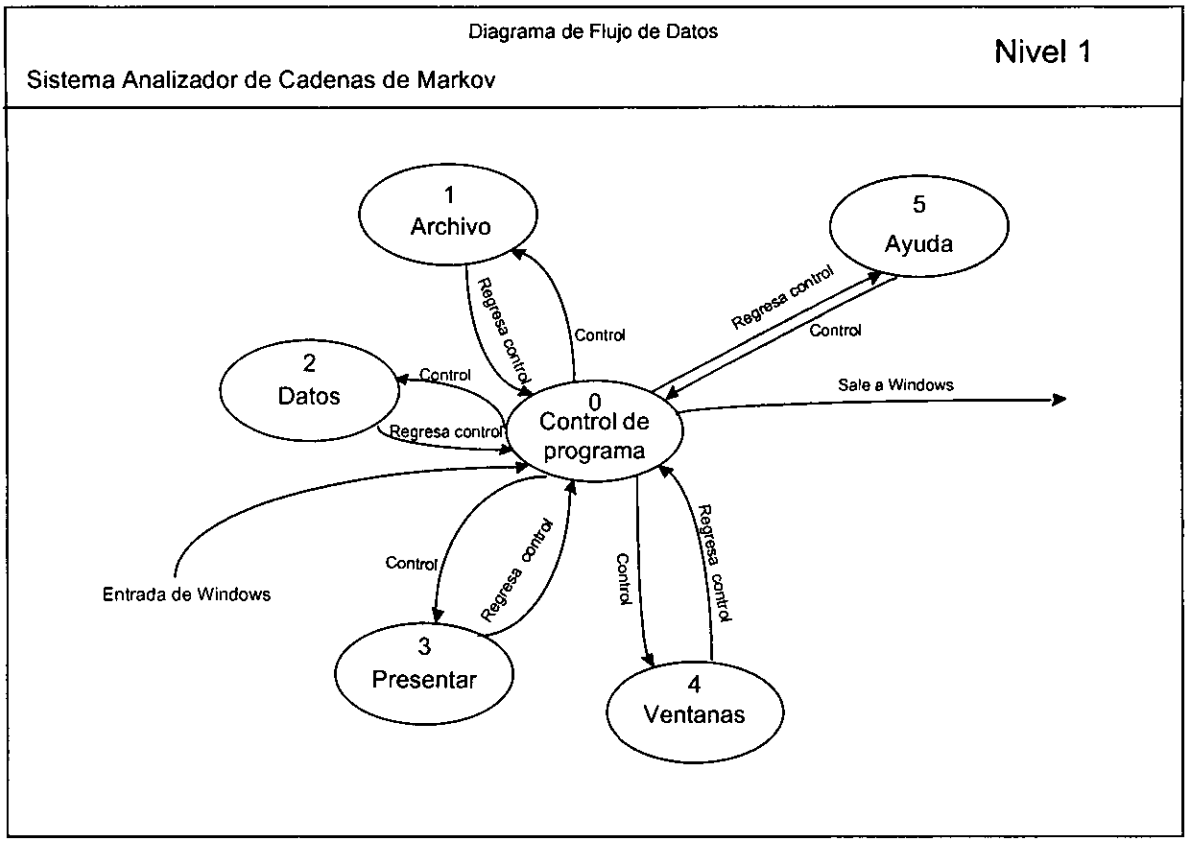
2.10.1 Diseño del Sistema

El Diseño se puede expresar con los siguientes DFD's:

DFD de Procesos Programa Principal
Analizador de Cadenas de Markov

Nivel 0





Un mayor detalle se puede apreciar en el Anexo A.

Glosario :

Constante.- Es la declaración de una variable que a lo largo de un programa el valor que representa es inamovible.

DFD .- Diagrama de Flujo de Datos.

Identificador .- Es el nombre con que se les designa a las variables para referenciar su valor o contenido. Asignación es la acción que consiste en atribuir un valor a una variable.

Proceso.- Es una acción que se puede descomponer en otras más simples, o también un conjunto de fenómenos organizados en el tiempo y concebidos como activos.

Procesador .- Es el elemento capaz de ejecutar un determinado proceso de trabajo.

Símbolos de Operación.- Son aquellos utilizados para denotar operaciones entre los datos como son sumas, restas, multiplicación, mayor que, menor que, igual, etc.

Variable .- Es el objeto de un programa que contiene los datos o a donde son asignados.

Bibliografía

Brookshear, J. Glenn:(1993)TEORIA DE LA COMPUTACION. LENGUAJES FORMALES, AUTOMATAS Y COMPLEJIDAD, Ed. Addison-Wesley Iberoamericana, S.A. ,USA.

Casper, Jhon(1994): OBJECT ORIENTED PROGRAMMING ANALYSIS, DESIGN AND IMPLEMENTATION, Computer Technology Research Corp.; South Carolina, USA, Cap.3.

Eckel, Bruce(1991):APLIQUE C++; Ed. McGraw-Hill, México.

Joyanes Aguilar , Luis (1993):METODOLOGIA DE LA PROGRAMACION, Ed. McGraw-Hill, México.

Korth, Henry S.F. (1988): FUNDAMENTOS DE BASE DE DATOS, Ed. McGraw-Hill, México.

Schild Herbert(1990):UTILIZACION DE C EN INTELIGENCIA ARTIFICIAL; Ed. Osborne/McGraw-Hill; Madrid, España.

Pressman, Roger S. (1993):INGENIERIA DEL SOFTWARE: UN ENFOQUE PRACTICO, 3ª Edición, Ed. McGraw-Hill/Interamericana de España, S.A., Madrid España.

Capítulo 3

Estructura del Sistema (Cadenas de Markov)

3.1. Módulos.

Los módulos se definen en base a la funcionalidad requerida por el sistema. Por ejemplo, un sistema por lo general recibe información desde fuera; sea por el teclado o por dispositivos que estén conectados a la computadora. Sin importar en detalle de que caso se trate, la función es recibir datos, por tanto tenemos un módulo de entrada de datos.

Con base en la explicación anterior se enlistan los módulos del sistema cadenas de Markov de acuerdo a su funcionalidad.

Entrada/Recuperación de Datos.- Se introducen los datos mediante el programa o se pueden recuperar desde un archivo

Simulación.- Se calculan las probabilidades de Transición de n-pasos iterativamente

Gráfica.- Se dibujan los estados de las cadenas de Markov como nodos y las transiciones como arcos.

Impresión.- Impresión de los elementos de la cadena de Markov como son cadena Inicial, cadena de la Simulación y Análisis de la cadena.

Consulta a la Base de Conocimientos.- Esta consulta se realiza a través del sistema de ayuda.

3.2. Interfases.

Las interfaces se refieren a qué medios usa el software para comunicarse con el resto del sistema. Por ejemplo, cuando un sistema está programado en MSDOS, la interfase es tipo texto, si se realizara una gráfica en este medio habría que llamar a un programa que configurara el video para modo gráfico, este programa establecería el medio para tener un ambiente gráfico en lugar de un ambiente tipo texto.

El sistema Analizador de Cadenas de Markov está programado para Windows, por tanto la interfaz que utiliza es una interfaz gráfica; a su vez aprovecha las opciones que proporciona Windows como sus drivers(manejadores) para impresión y sus librerías, aunque estas últimas no sean visibles para el usuario.

3.3. Funcionamiento.

El funcionamiento de un programa se basa en dos aspectos: Primero, cómo interactúa con el sistema operativo y Segundo cómo opera el programa.

Interacción del Programa con el Sistema

El sistema operativo Windows es un software muy complejo donde sus características están basadas en una interface gráfica (GUI por sus siglas en inglés Graphical User Interfase), y en su conducción de mensajes (también referidos como manejadores de eventos). Esto significa que los eventos que están fuera de una aplicación (tal como introducción de datos por teclado, eventos del sistema, o los efectos de algunas otras aplicaciones), controlan al programa. Para cada evento, un mensaje (de acuerdo al evento) es enviado a la aplicación, y ésta es libre de considerarlo o de ignorarlo.

La programación Windows es diferente de otras por tres razones básicas[1]:

1. Los programas son organizados alrededor de eventos.
2. Los recursos de la computadoras están compartidos entre las aplicaciones.
3. Los programas no manipulan directamente los periféricos (por ejemplo la pantalla).

Así como funciona un programa Windows, la aplicación que se desarrolle en él, necesita interactuar con el ambiente operativo de Microsoft. Básicamente, Windows es una colección de funciones que se pueden llamar para construir programas que realicen diferentes cosas, desde mostrar una ventana en la pantalla hasta el manejo de bloques de memoria.

Basados en esta información el programa desarrollado en C++ está compuesto por una serie de funciones regidas por una función principal llamada *WinMain.*

Esta función *WinMain* es el punto de entrada para la ejecución de un programa en Windows. Las tareas que realiza son de inicialización, entre otras, de las clases de ventanas, creación y visualización de una ventana. Una vez realizada esta tarea sigue la ejecución del *Message Loop,* o ciclo de mensajes, que nos permite captar los mensajes del sistema y así ejecutar las tareas asignadas a dichos mensajes. Cuando se envía un mensaje de que se debe terminar la aplicación, el ciclo de mensaje se termina.

Otra de las características de un programa Windows es que el código de la aplicación incluso después de múltiples ejecuciones sólo existe una vez en la memoria, esto se traduce a un ahorro de memoria, y los segmentos de datos de las aplicaciones existen tantas veces en memoria como veces se ejecutó el programa.

[1] Heiny Loren(1992) ,p 1

Funcionamiento de Operación

El funcionamiento de operación se desglosa en los siguientes pasos:

- Instalación del Programa
- Activación del Programa
- Manipulación del Programa mediante Menús tipo Windows

Instalación del Programa: La instalación del programa se puede realizar básicamente en cualquier directorio, pero por mejor conveniencia se instalará en un directorio llamado Markov, esto se realiza ejecutando un archivo de procesamiento por lotes, ubicado en el diskette del programa.

Una vez instalado se puede proceder a la ejecución.

Activación del Programa: Desde el menú del archivo ejecutar, se localiza el archivo Markov.exe y se activa. Apareciendo una ventana con los datos del programa de Markov.

Manipulación del Programa mediante Menús tipo Windows: Las siguientes operaciones concernientes con las cadenas de Markov se realizan desde la barra de Menú. La definición de los Menús se logró con la especificación de las opciones en un archivo de recursos.

Cargado de los Datos : Los datos pueden ser introducidos mediante el teclado, se recomienda antes abrir un archivo, sea nuevo o ya existente en el menú de Archivo; en caso de no elegir un nombre de archivo mediante la opción *guardar* se tiene un nombre por omision: *sinnom.dat.*

La introducción de datos se realiza en el Menú Pop-up Datos, con las opciones:

- *Estados y matriz de T.* Donde se introduce el número de nodos de la cadena y la matriz de transición de la cadena.
- *Vector Inicial.* Se introducen el vector que matemáticamente definimos como π, que indica en qué estado inicia la cadena.

Simulación de la Cadena: La simulación de la cadena se realiza en base a la matriz de transición y el vector inicial; para realizar esta simulación se debe contar con una cadena de Markov ya introducida y que se active la ventana de simulación en el Menú Presentar, seguido de presionar la tecla F6, se realiza la simulación paso a paso según se presione esta tecla.

Ver la Gráfica: Del Menú Presentar se selecciona la opción de gráfica para desplegar una ventana que nos muestra la gráfica con sus transiciones y colorea el nodo donde se encuentra la cadena en el tiempo *n*.

Ver la Evaluación de la Cadena: En el mismo menú de presentar, la opción de evaluación nos proporciona el análisis de la cadena realizado en base al análisis gráfico propuesto en la clasificación de Estados del capítulo 1. Muestra la matriz de adyacencia, la matriz de alcanzabilidad R, la matriz que nos permite encontrar los componentes conectados, la clasificación de los estados y la matriz canónica.

Cambiar los parámetros de los Datos: Básicamente por simplicidad se recomienda cambiar los datos del vector inicial, pues al cambiar datos en la matriz estocástica (cadena de Markov), es cambiar de cadena.

Consultar la Base de conocimiento par medio de la ayuda: La base de conocimiento está conformada mediante una jerarquización de archivos de conceptos con una estructura de árbol, donde se indizaron los temas y los descriptores temáticos. El resultado de esto nos dio por un lado un apoyo didáctico en base a un documento en hipertexto; y por el otro el poder buscar conceptos separados.

Explicación de los Menús

Menú Archivo

Abrir.- Un archivo que contiene una Cadena de Markov, entiendiéndose que está expresada como vector inicial y matriz estocástica.

Guardar.- La cadena activa se almacena en un archivo.

Imprimir.- Impresión de :

Datos Iniciales
Simulación de la Matriz Actual
Análisis de la Cadena

Salir.- Terminar el programa de Cadenas de Markov y volver al ambiente windows

Menú de Datos

Permite crear o modificar cadenas de Markov sus opciones:

Estados y Matriz de Transición.- Esta es la primera opción para introducir una cadena de Markov por primera vez, aquí se le solicita el No. de estados o nodos de que consta la cadena; y a continuación se introduce la matriz estocástica, elemento a elemento.

Vector Inicial.- Una vez registrada una Matriz Estocástica se procede a darle valores al vector inicial.

Iterar.- Esta opción se utiliza una vez que tenemos una cadena de Markov y su vector inicial, con el fin de ver su comportamiento paso a paso, mismo que podemos observar simultáneamente en la opción de simular y graficar, en el menú presentar.

Menú Presentar

Datos.- Presenta una ventana con la matriz estocástica de la cadena de Markov.

Simular.- Se despliegan los datos de la matriz y el vector en su n-ésimo paso.

Gráfica.- Dibuja la gráfica correspondiente a la cadena de Markov estrechamente ligada a la simulación de la misma, por tanto, muestra en qué estado se halla la cadena en el paso *n*.

Evaluar.- Se despliega el análisis de la Cadena a partir de una evaluación de sus transiciones, llegando a la clasificación de estados y su matriz canónica.

Nuevo Menú.- Esta opción cambia el contexto de la ventana principal, con el objeto de constrastar los resultados de las ventanas que se tienen activas.

Menú de ventanas

Esta opción del menú se programó tener organizar las varias vistas de la cadena de Markov

Las opciones se describen a continuación:

Mosaico.- Distribuye las ventanas en toda el área del administrador de programas.

Cascada.- Distribuir las ventanas en cascada

Organizar Iconos.- Distribuir las ventanas cuando están Iconizadas o Minimizadas.

Cerrar.- Se cierran todas las ventanas.

Menú de Ayuda

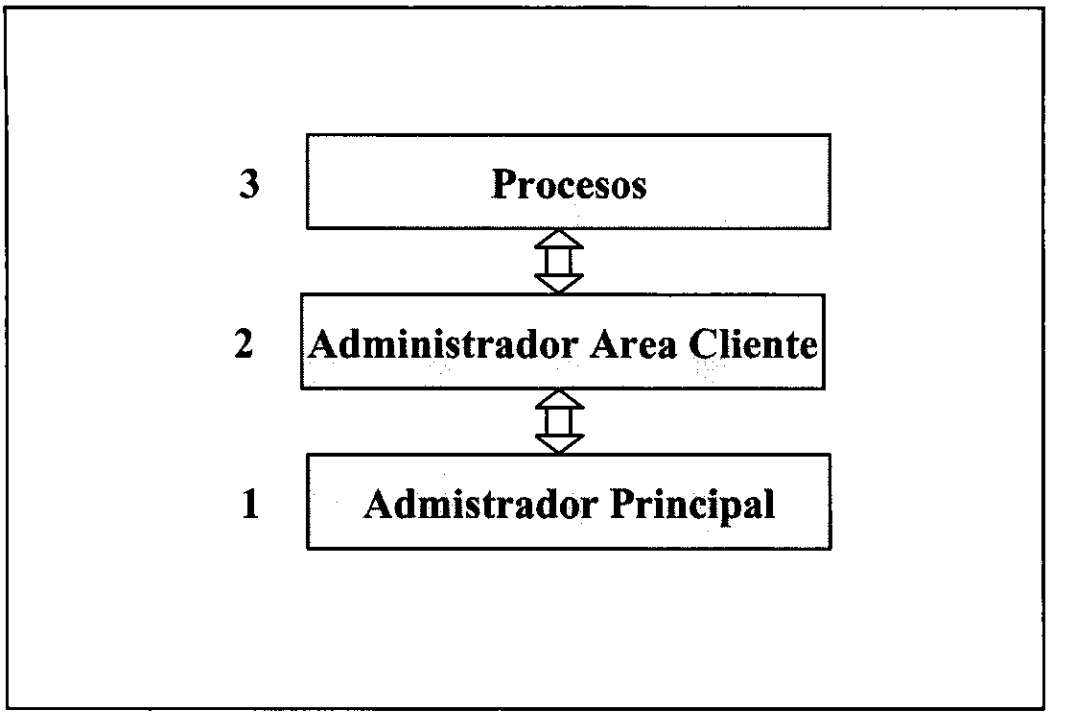
Concentra la base de conocimientos para auxiliar al usuario del programa, su efectividad está en relación con su estructura.

Introducción.- Se lee una breve introducción al programa.
Programa .- Las instrucciones de operación del programa son proporcionadas aquí
Cadenas de Markov.- El sustento teórico aquí se enlista en forma de apartados para un aprendizaje progresivo.
Buscar.- Nos permite un acceso rápido a conceptos a través de un índice.

3.4. Pseudocódigo

El pseudocódigo como se mencionó en el capítulo anterior son los algoritmos que se escriben entre lenguaje natural y lenguaje de alto nivel, y conforman el desarrollo del sistema, a continuación se lista el pseudocódigo a grandes rasgos.

El programa de cadenas de Markov, por conveniencia propia, se puede describir como una serie de niveles:



Le llamaremos al nivel 1 Administrador principal , al nivel 2 Administrador del Area Cliente y al nivel 3 Procesos.

Administrador Principal

En este nivel se encuentra la función WinMain. En esta función comienza la ejecución del programa y normalmente también en esta misma termina. La función WinMain es responsable de:

- Registrar el tipo de la clase de ventana de la aplicación.
- Realizar todas las inicializaciones requeridas.
- Crear e Iniciar el ciclo de procesamiento de mensajes de la aplicación (que accede a la cola de mensajes del programa).
- Terminar el programa, normalmente al recibir un mensaje WM_QUIT.

En base a esta descripción de la función WinMain se realizan las siguientes tareas:

-Registro de la clase Ventana Padre (Frame)
-Registro de la clase Ventana hijo Datos

-Registro de la clase Ventana hijo Simular
-Registro de la clase Ventana hijo Gráficos
-Registro de la clase Ventana hijo Evaluar
-Muestra la ventana padre
-Actualiza la Ventana padre
-Procesa Ciclo de Mensajes

Administrador Area Cliente:

Todas las aplicaciones tienen que incluir una función WinMain y una función de rellamada, esta es *FrameWndProc*.

La ventana que se encarga de la administración de los procesos del menú principal es el procedimiento *FrameWndProc*

Esta ventana FrameWndProc está basada en el diseño de aplicaciones MDI (Multiple Document Interface) que consiste en devolver una interfaz[2] estandarizada para la presentación y el procesamiento de diferentes documentos en un programa.

Una aplicación MDI se compone de una ventana principal, en la que el usuario puede abrir tantos documentos, como los que se quieren presentar o modificar. Cada documento aparece en una ventana propia, que es una ventana hijo con menú de sistema.

Las ventanas hijo se coordinan con ayuda de mensajes, que se envían a través de diferentes niveles de jerarquía (ventana marco ⇔ ventana cliente ⇔ ventana hijo). La ventana cliente, que se encuentra en la ventana marco en vez de la *Client Area* y que es controlada por Windows, se encarga de muchas tareas de administración.

A continuación se escribe el pseudocódigo de ventana Marco o *FrameWndProc*

Declaración de variables

```
CLIENTCREATESTRUCT      ccs;
MDICREATESTRUCT   mcs;
HWND                    hWndChild;
static HWND hEdit;
static   HANDLE      hInst;
static               int i = 1;
char                 titel[10];
static   HMENU       hMenu1, hMenu2;
char     chFileName[10] = "help.hlp";

// variables para procedimientos de  entrada de datos por teclado
     static FARPROC fpfnAboutDiaProc;
     static FARPROC fpfnMatDiaProc;
```

[2] Bär,J.; Bauder, I.(1995), p.537

```
static FARPROC fpfnInputNodoDiaProc;
static FARPROC fpfnVectorInicialDiaProc;

static HANDLE   hInst1,hInst2,hInst3,hInst4,hInst5;
int j,k;

En caso de que llegue mensaje igual a
{
        case WM_CREATE:
                Genera el apuntador a la instancia de la ventana hijo
                Carga los menús de la barra de menús
                Carga los submenús
                Se carga la estructura ccs de la estructura CLIENTCREATESTRUCT con el
                handle del menú Popup bajo el título del cual han de aparecer las ventanas-hijo, y
                con el valor ID de la primera ventana-hijo.
                Crea la Ventana Cliente MDI
                Se hace visible la ventana cliente
                Se crea el handle para abrir archivos hEdit
                Se llena la estructura ofn para el manejo de archivos
                Se generan las instancias y se asignan a cada una de las cajas de diálogo que
                serán necesarias en la aplicación.


case WM_COMMAND: // En ruta  los mensajes generados por los menús
        switch (wParam)  //en caso de que wParam sea igual a
{
        case ID_IMPRE
                Se despliega un menú de opciones de impresion donde puede imprimirse
                Datos Iniciales o  cadena de Markov
                Simulación de Matriz Actual
                Análisis de Cadena

        case ID_ITERA:
                Calcula la Matriz en la transición n+ 1 con el procedimiento Markov.Multiplica();
                Actualiza las ventana
                break;



//----------INICIAN OPCIONES DE LA BASE DE CONOCIMIENTO
        case ID_HELP_INT:
                muestra el texto de la introducción
                break;

        case ID_HELP_PROG:
                muestra ayuda para operación del programa
                break;

        case ID_HELP_CADMKV:
                Accesa a los temas sobre cadenas de Markov
                break;

        case ID_HELP_BUSCAR:
```

```
                    Accesa un índice por descriptores de todos los temas de cadenas de Markov y
                    operación del programa
                    break;
//----------------FIN  SISTEMA DE  BASE DE CONOCIMIENTOS

            case ID_DATOS:
                    Crea y muestra la ventana hijo para los datos y los presenta en pantalla
                    break;

            case ID_SIMUL:
                    Crea y muestra la ventana hijo para las Transiciones de la Matriz en n-pasos
                    break;
            case ID_GRAF:
                    Crea y muestra la ventana hijo para la gráfica  y los presenta en pantalla
                    break;

            case ID_EVAL:
                    Crea y muestra la ventana hijo para los datos Analizados y los presenta en
                    pantalla
                    break;
//*************************************************************************
//          Rutinas  para  abrir y guardar archivos
//*************************************************************************
            case ID_ABRIR:
                    Abre el archivo
//*************************************************************************
//      Rutinas para la introduccion de datos por teclado
//*************************************************************************
             case ID_NODOS:
                    Markov.Inicializa();   // Inicializa la matriz
                    Pide el no. de nodos
                    para (i=0; i<nodos; i++)
                    {
                            para (j=0; j<nodos;j++)
                            {
                                    Captura de pantalla el elemento(i,j)
                                    Si es un número válido
                                            Markov.ObtMatriz(i,j,atof(numstring1)); // Asignalo
                                    sino
                                    {
                                            Despliega el mensaje("Valor Incorrecto");
                                            j- -;
                                    }                                       }
                                    Actualiza ventana
                            }
                                     Si existe error en la suma decrementa i en 1 para que vuelva a
                            leer el renglón
                    }
                      Iguala Cadena de Markov a una temporal para realizar la multiplicación de
                      matrices
                     break;

             case ID_VECTOR:
                    //Lectura del Vector
```

```
                    para (j=0;j<2;j++)
                    {
                        Lee elemento(j)
                        si es correcto Asignar a la estructura de vector inicial
                        y copiarlos a un temporal para la simulación de la transicion en n-pasos
                    }

                case ID_ACERCA:
                    Muestra los créditos del programa

//**********************************************************************
//      Rutinas para manipular las ventanas
//**********************************************************************
                case ID_TILE:
                    Organiza las ventanas hijo Abiertas en forma de Mosaico
                    break;

                case ID_CASC:
                    Organiza las ventanas hijo Abiertas en forma de Cascada
                    break;

                case ID_ICON:
                    Organiza las ventanas hijo que se encuentran en forma de Iconos en hilera
                    break;

                case ID_MENU1:
                    En caso de que se seleccione cambio del menú2 al menú1, éste se activa
                    break;
                case ID_MENU2:
                    En caso de que se seleccione cambio del menú1 al menú2, éste se activa
                    break;

                case ID_ENDE:
                    Se ha activado el comando Cerrar
                    Oculta la ventan cliente
                    cierra y destruye las ventanas hijo
                    break;
                case ID_SALIR:
                    Se ha activado el comando Salir que implica salir de la aplicación
                    break;
        }fin del Case del Command
        break;

case WM_DESTROY:
                El mensaje generado es un mensaje de salir
                break;

default:
                Si no existe mensaje dentro de estas opciones se regresa el estado de la ventana cliente
                break;
}
return 0;
}
```

Procesos

En esta sección se presentarán los procesos principales utilizados por la aplicación y más adelante se mostrará un organigrama de la jerarquía de procesos.

Proceso para la ventana-hijo de Datos

```
LONG FAR PASCAL MDIChildWndProc (HWND hwnd,UINT message,UINT wParam,LONG lParam )
{

 Inicializaciones de Handles y variables
        switch (message)
        {
                case WM_CREATE:
                        Crea la ventana hijo y la actualiza
                        break;

                 case WM_PAINT:
                        Coloca la Matriz de Markov en la Ventana
                        break;
                default:
                        return DefMDIChildProc (hwnd, message, wParam, lParam);
        }
        return FALSE;
}


Pseudocódigo de Coloca la matriz de Markov en la Ventana
Markov.ImpreMatriz(hdc,1)
si (n = 1)
{
        despliega ,"Vector inicial"
        for (k=0;k<nodos;k++){
                imprime elemento vk[k]


       despliega "Matriz de Transición
        for(k=0;k<nodos;k++) {
                for(i=0;i<nodos;i++)
                        imprime elemento mi[k][i]
                Avance de linea
}
```

Proceso para la ventana-hijo de Simular

```
LONG FAR PASCAL MDISimularWndProc (HWND hwnd,UINT message,UINT wParam,LONG lParam )
{
        Inicialización de Handles y Variables
        switch (message)
        {
```

```
                case WM_CREATE:
                        Crea la ventana hijo y la actualiza
                        break;
                case WM_SIZE:
                        hwndList = GetWindowWord (hwnd, 0);
                        MoveWindow (hwndList, 0, 0, LOWORD(lParam), HIWORD(lParam), TRUE);
                        return DefMDIChildProc (hwnd, message, wParam, lParam);
                        break;

         case WM_PAINT:
                Presenta la Matriz de Markov en su transición n+1
                break;
        default:
                        return DefMDIChildProc (hwnd, message, wParam, lParam);
        }
        return FALSE;
}
```

Pseudocódigo del Proceso de la ventana Simular

```
Markov.ImpreMatriz(hdc,0);

{
        Despliega el mensaje,"Vector 'n'"
        for (k=0;k<nodos;k++)
                Imprime elemento vf[k]

        Despliega el mensaje,"Matriz de Transición a la 'n'"
        for(k=0;k<nodos;k++)
        {
                for(i=0;i<nodos;i++)
                        imprime ele elemento mf[k][i]
                avance de linea
        }
```

Proceso para la ventana-hijo de Graficar

```
LONG FAR PASCAL MDIGraficarWndProc (HWND hwnd,UINT message,UINT wParam,LONG lParam )
{

        Inicialización de Handles y Variables

        switch (message)
        {
                case WM_CREATE:
                        Crea la ventana hijo y la actualiza
                        break;

         case WM_PAINT:
```

```
            Dibuja la gráfica
            break;
        default:
                    return DefMDIChildProc (hwnd, message, wParam, lParam);
        }
        return FALSE;
}
```

Pseudocódigo para dibujar la gráfica
void Cadena::Vertices(HDC hdc,int t)

1.- Obtener de la variable nodos el valor de estados a dibujar
2.- Obtener de la variable mi la matriz de transición
3.- Calcular el ángulo para cada nodo, es decir, dividir 360° entre el número de nodos
Calcular el centro de la pantalla y determinar el radio para distribuir los nodos
4.- Para cada nodo i, i= 0...nodos-1
Si mi[i][i]>0
Crear un círculo, imprimir dicho valor de mi[i][i] de acuerdo a los grados y el radio del nodo i
Si i = Nodo con Mayor valor dibujar una X y calcular 4 puntos de la periferia del círculo si no crear un círculo de color normal e imprimir su no. correspondiente y calcular también 4 puntos de la periferia del círculo
incrementar i en 1
5. Para cada nodo i, i = 0,..,nodos-1
5.1 Para cada nodo j, j=0,...nodos-1
i=0, j=0
si mi[i][j] >0; es decir si existe una transición de i a j
si i=j-1 unir el nodo i con el nodo j con una flecha en direccion a j
si i=j+1 unir el nodo i con el nodo j con una flecha en direccion a j
si i < j-1 o i > j+1
Si i = 0 y j = nodos -1 unir el nodo i con el nodo j con una flecha en direccion a j
sino j = 0 e i = nodos-1 unir el nodo i con el nodo j con una flecha en dirección a i
sino si i<j-1 unir el nodo i con el nodo j con una flecha en dirección a j
sino si (i>j+1) unir el nodo i con el nodo j con una flecha en dirección a j
Incrementar j en 1
si j < nodos ir a 5.1
si no Terminar
incrementar i
si i < nodos ir a 5
si no terminar

Proceso para la ventana-hijo de Evaluar

```
LONG FAR PASCAL MDIEvaluarWndProc (HWND hwnd,UINT message,UINT wParam,LONG lParam )
{

        Inicialización de Handles y Variables
        switch (message)
        {
```

```
                case WM_CREATE:
                        Crea la ventana hijo y la actualiza
                        AnalizaCad(Cadena Markov, "result.txt")
                        Carga el documento en la ventana
                        break;

                case WM_SIZE:
                        MoveWindow(hRes, 0,0, LOWORD(lParam),  HIWORD(lParam), TRUE);
                        break;

                default:
                        return DefMDIChildProc (hwnd, message, wParam, lParam);
        }
        return FALSE;
}

Pseudocódigo de void AnalizaCad(Cadena Markov, char *Arc_resul)
{
 unsigned long int R[20],QR[20];
 BYTE Prior[20];
 FILE *arch;
 arch = fopen(Arc_resul,"w");
 if(Markov.nodos){
        ObtenAdy(R,Markov.nodos,Markov.mi,arch); //en R colocamos la Matriz de
                                                        adyacencia de la cadena
Esta matriz, tiene las mismas dimensiones que la matriz correspondiente a la cadena de Markov, donde para
cada elemento distinto de cero en la cadena de Markov  se coloca un 1 en la matriz R.

        ObtenR(R,QR,Markov.nodos,arch);                 //en R colocamos la Matriz de
                                                                alcanzabilidad
Se calcula QR a partir de R realizando la operación
QR = R * Q,  donde Q = R'
y esta matriz resultado nos proporciona en los renglones de la matriz las clases de Equivalencia.
El procedimiento ClasificaEdo(R,QR,Prior,Markov.nodos,arch); //
nos permite obtener las clases de equivalencia que conforman a la cadena.En este mismo procedimiento se
asigna la prioridad para cada conjunto de estados de tal forma de organizar a la matriz de transición
colocando primero a los estados recurrentes y en seguida a los estados transitorios

Este procedimiento imprime la matriz canónica con el orden que se da en el vector de prioridad
        Canonica(Markov.mi,Prior,Markov.nodos,arch);
 }
 fclose(arch);
}
```

Base de Conocimiento

El procedimiento para la creación de la base de conocimiento siguió los pasos siguientes:
-Definición Estructural de los temas.

- Programa
- Cadenas de Markov
- Buscar

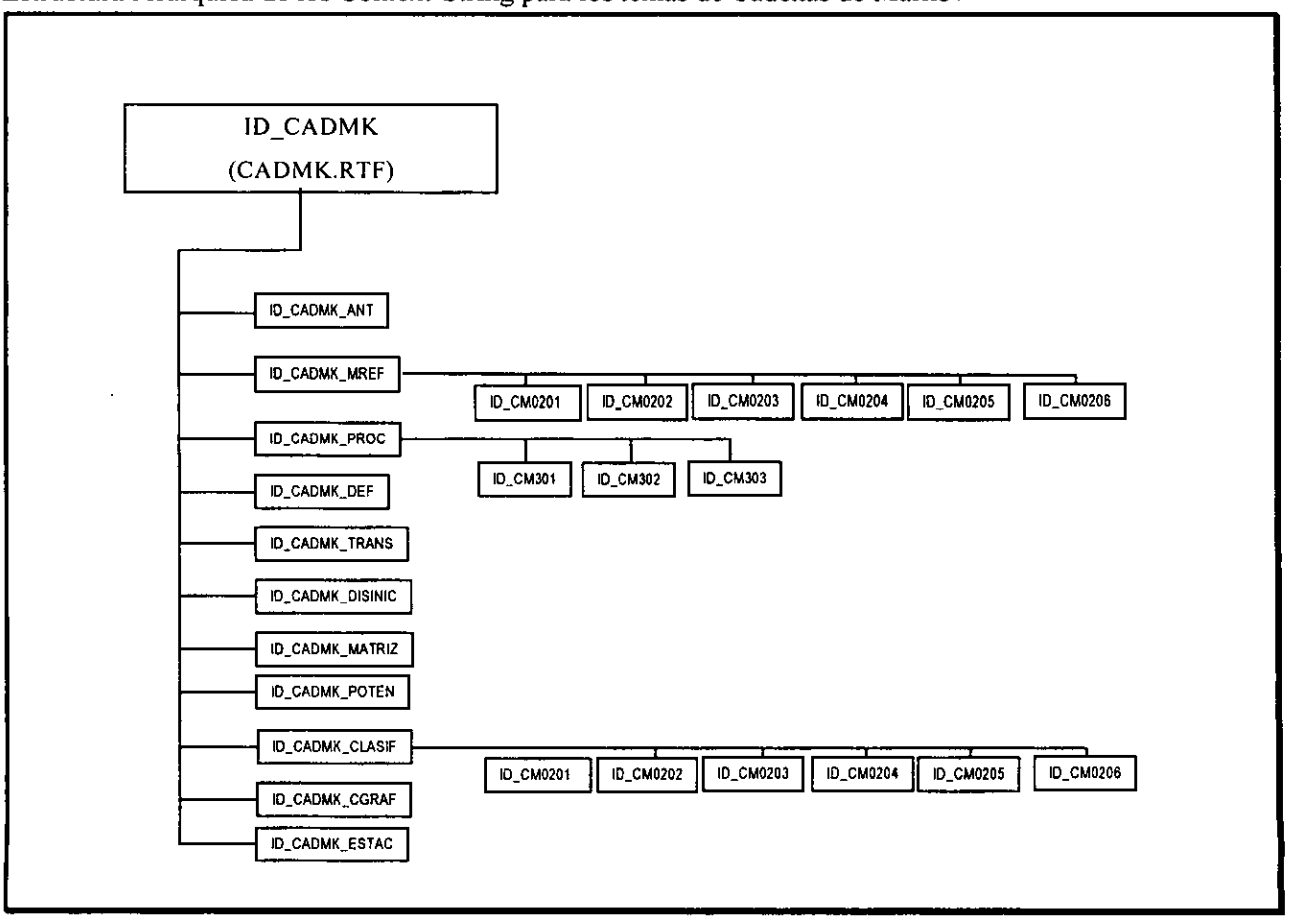
Donde en Programa se describe la operación del programa.

En Cadenas de Markov se describe una estructura jerarquizada de temas.

La base de conocimiento fue conformada en una estructura jerárquica de archivos de formato RTF (Rich Text Format). A continuación se describen:

ARCHIVO	**SECUENCIA**	
CADMK01.RTF	CADM:010	ANTECEDENTES
CADMK02.RTF	CADM:020	MARCO DE REFERENCIA
CADMK03.RTF	CADM:030	TIPOS DE PROCESO
CADMK04.RTF	CADM:040	DEFINICION DE CADENAS DE MARKOV
CADMK05.RTF	CADM:050	FUNCION DE TRANSICION
CADMK06.RTF	CADM:060	DISTRIBUCION INICIAL
CADMK07.RTF	CADM:070	MATRIZ DE TRANSICION
CADMK08.RTF	CADM:080	POTENCIAS DE MATRICES ESTOCASTICAS
CADMK09.RTF	CADM:090	CLASIFICACION DE ESTADOS
CADMK10.RTF	CADM:O100	ESTACIONARIEDAD
CADMK95.RTF	CADM:095	ANALISIS GRAFICO
CMK_02.RTF		DEFINICIONES

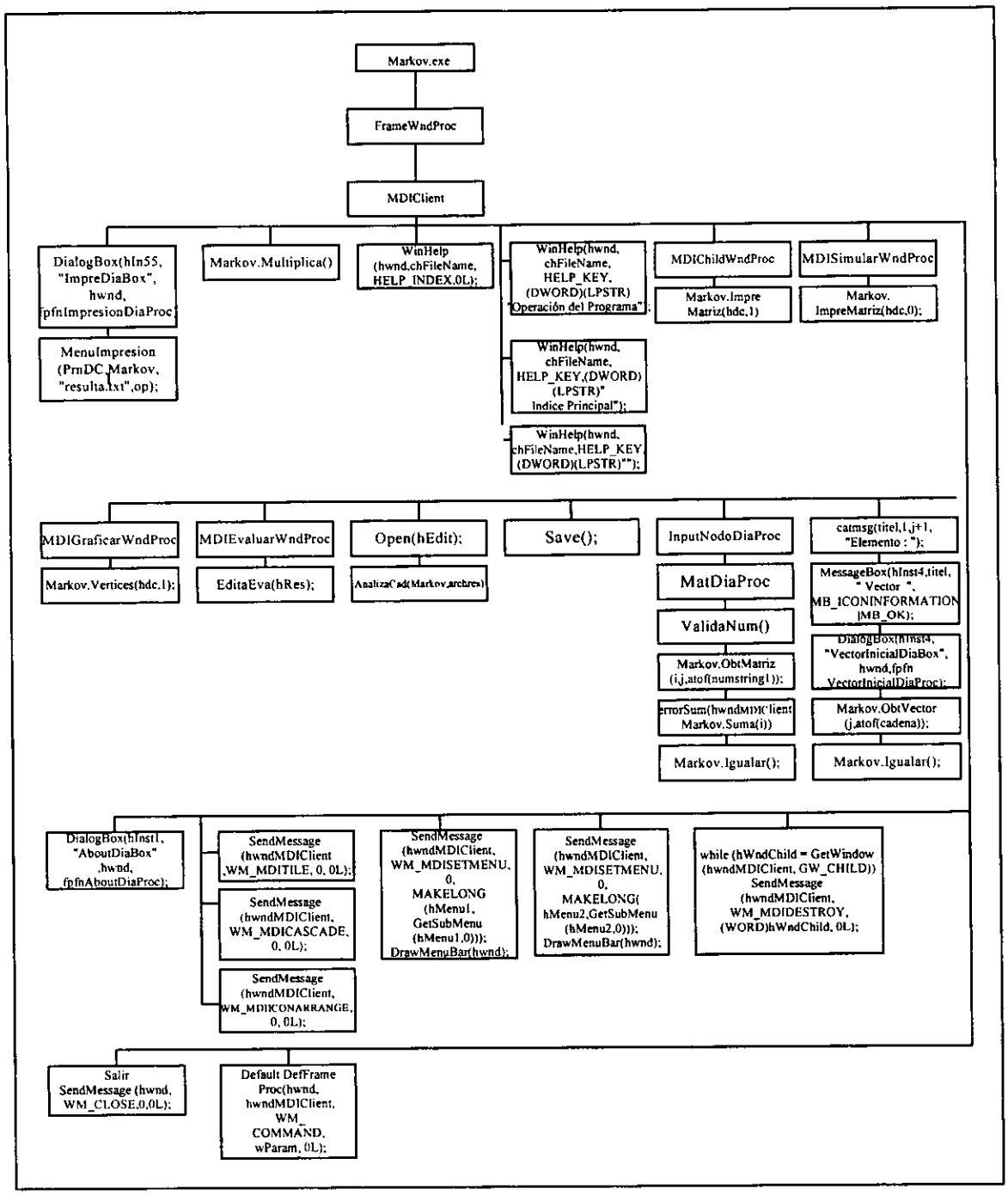
Estructura Jerárquica de los Context-String para los temas de Cadenas de Markov



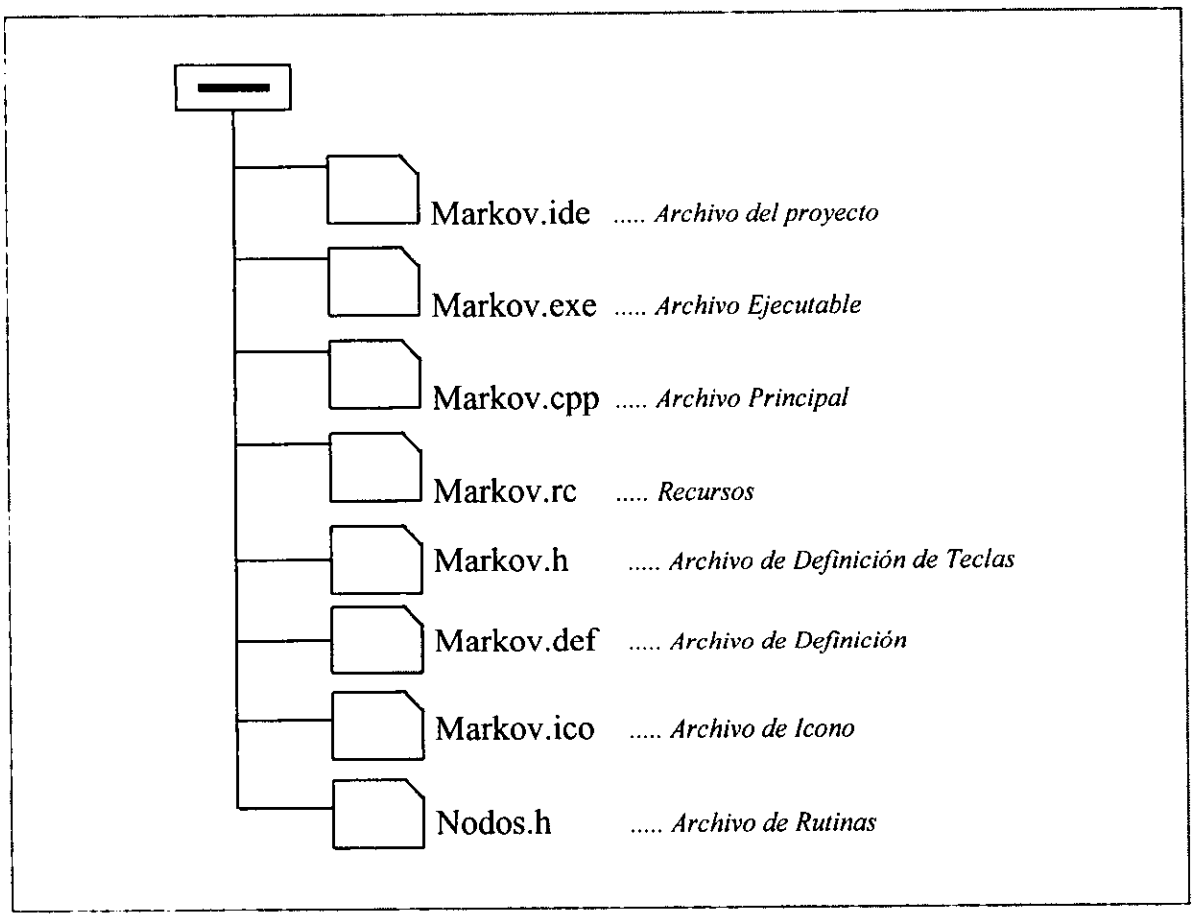
ID_CADMK	Indice de Temas de Cadenas de Markov
ID_CADMK_ANT	Antecedentes
ID_CADMK_MREF	Marco de Referencia
ID_CADMK_PROC	Tipos de Proceso
ID_CADMK_DEF	Definición de Cadenas de Markov
ID_CADMK_TRANS	Función de Transición
ID_CADMK_DISINIC	Distribución Inicial
ID_CADMK_MATRIZ	Matriz de Transición
ID_CADMK_POTEN	Potencias de Matrices Estocásticas
ID_CADMK_CLASIF	Clasificación de Estados
ID_CADM_ESTAC	Estacionariedad
ID_CM0201	Mundo Real
ID_CM0202	Modelo Real
ID_CM0203	Modelo Matemático
ID_CM0204	Conclusiones y Predicciones
ID_CM0205	Modelo Determinístico
ID_CM0206	Modelo Estocástico
ID_CM0301	Proceso
ID_CM0302	Proceso Determinístico
ID_CM0303	Proceso Estocástico
ID_CM0901	Estado Recurrente
ID_CM0902	Estado Transitorio
ID_CM0903	Estado Absorbente
ID_CM0904	Periódico
ID_CM0905	Estado Ergódico

Este programa fue compilado por Borland C++ ver. 4.

Diagrama Jerárquico de Funciones



El programa se compone de los siguientes módulos en cuestión de archivos.



Glosario.

MDI : Multiple Document Interface, Interface para múltiples documentos, es una técnica de programación para aplicaciones Windows.

RTF.- Rich Text Format

Bibliografía

Bär, J. Bauder I.:(1995) WINDOWS 3.1 INTERNO, AlfaOmega Grupo Editor, S.A. de C.V., México.

Heiny, Loren (1992): WINDOWS GRAPHICS PROGRAMMING WITH BORLAND C++, Ed. Jhon Wiley and Sons, New York, USA.

Murray, William H. y Pappas Chris H. :(1994); PROGRAMACION DE APLICACIONES PARA WINDOWS NT, Ed. Osborne/McGraw-Hill, México.

Perry Greg (1993): APRENDIENDO PROGRAMACION ORIENTADA A OBJETOS CON TURBO C++ EN 21 DIAS; Ed. Prentice Hall Hispanoamericana; México.

Schild, Herbert (1985): PROGRAMACION EN LENGUAJE C; Ed. McGraw-Hill, México.

Schild, Herbert (1992): TURBO C/C++, MANUAL DE REFERENCIA; Ed. McGraw-Hill, México.

Capítulo 4

Implementación

4.1 Análisis de Ejemplos Prácticos

La implementación o prueba del sistema de Cadenas de Markov se llevó al cabo en una computadora 80586(aunque también fue implementado en una computadora 80486), fue necesario añadir un archivo del compilador C++ de Borland en la computadora para su funcionamiento.

Los pasos de la implementación fueron los siguientes:

1.-Instalación del programa
2.-Activación del programa
3.-Introducción de las cadenas de Markov

Como se mencionó, fue necesario agregar un archivo que es bc40rtl.dll, una librería del C++ de borland.

Para la construcción de un modelo matemático se deben llevar al cabo cuatro distintas fases:

1) Identificación del problema.
2) Idealización.
3) Construcción del Modelo.
4) Implementación.

El analizador de cadenas de Markov solamente trata con la fase de implementación que consiste en ver su comportamiento (o descripción) a través de los resultados que produce dicho analizador.

Así la materia prima del analizador son las matrices estocásticas o de transición de probabilidades, y el vector de distribución inicial elementos que definen una cadena de Markov.

Problema 1.

Dos personas A y B empiezan a jugar con $ 3.00 cada uno como capital. Al término de un juego el perdedor paga $ 1.00 al ganador. En cada juego la probabilidad de que A gane es de 0.6 y de que B gane es de 0.4. Ellos terminan de jugar cuando uno de ellos pierde o gana todo. Sea C_n el capital de A después de *n* juegos. Entonces

$$C_{n+1} = \begin{cases} C_n - 1 \text{ con probabilidad } 0.4 \\ C_n + 1 \text{ con probabilidad } 0.6 \end{cases}$$

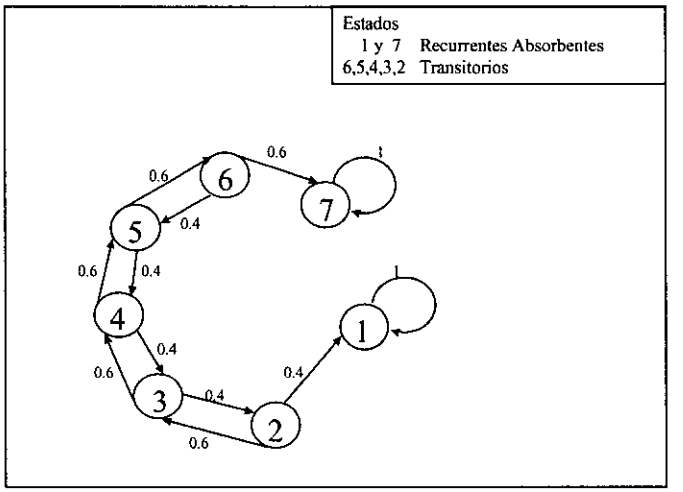
Claramente C_{n+1} depende únicamente de C_n ; así tenemos una cadena de Markov. Note que $C_{n+1} = C_n$ únicamente si ellos han detenido el juego.

Este es un ejemplo clásico de cadena de Markov conocida como la *ruina del jugador*. Las probabilidades de transición están dadas por :

$$P = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0.6 & 0 & 0 & 0 & 0 \\ 0 & 0.4 & 0 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ v_0 \end{array}$$

El vector v_0 indica que la cadena está iniciando en el estado 2.

En la correspondiente gráfica de estados se muestra, mediante el examen visual, la clasificación de los estados.



En cuatro pasos o después de cuatro jugadas el vector de distribución presenta los siguientes cambios

$$V_1 = \begin{bmatrix} 0.4 \\ 0 \\ 0.6 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad V_2 = \begin{bmatrix} 0.40 \\ 0.24 \\ 0 \\ 0.36 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad V_3 = \begin{bmatrix} 0.496 \\ 0 \\ 0.288 \\ 0 \\ 0.216 \\ 0 \\ 0 \end{bmatrix} \quad V_4 = \begin{bmatrix} 0.496 \\ 0.1152 \\ 0 \\ .2592 \\ 0 \\ 0.1296 \\ 0 \end{bmatrix}$$

La matriz canónica para este ejemplo es la siguiente:

$$P = \begin{array}{c} \begin{matrix} 1 & 7 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0 \end{bmatrix} \end{array}$$

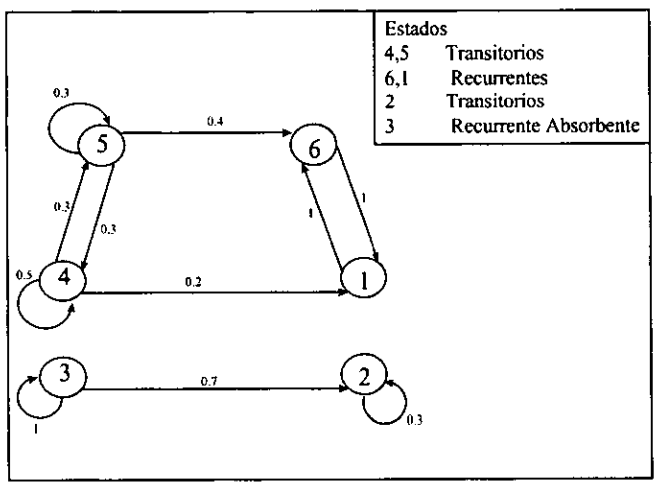
Problema 2.

Clasifique los estados de las cadena de Markov con la siguiente matriz de transición de probabilidades. También organice las matrices de transición de probabilidades en forma canónica.

$$P = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0.3 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0.2 & 0 & 0 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0 & 0.3 & 0.3 & 0.4 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

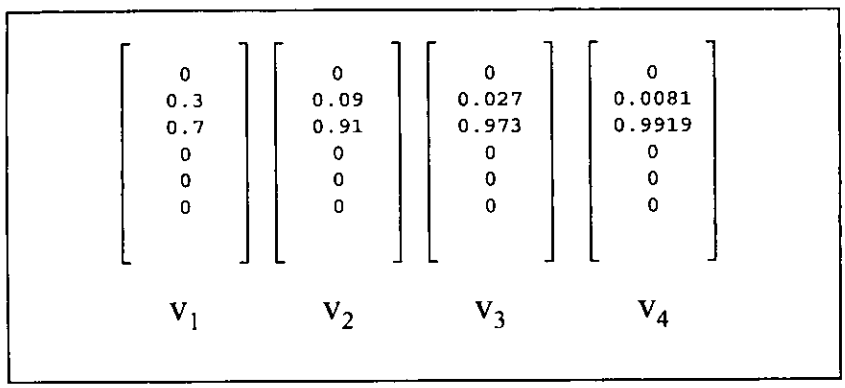
v_0

Su correspondiente gráfica se muestra en la siguiente figura



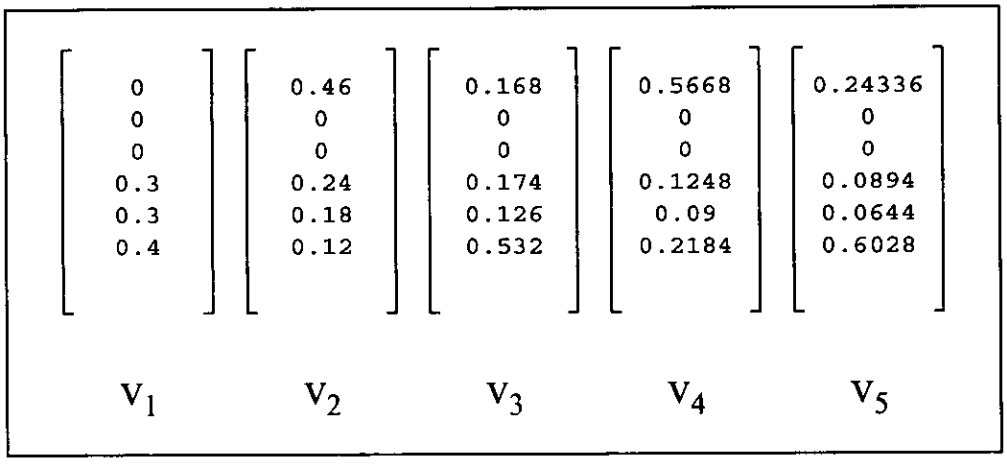
La clasificación de estados se ha realizado mediante la observación.

El siguiente recuadro indica al vector de distribución del paso 1 al 5, la cadena inició en el estado 2 y sucede lo mismo para el estado 3.



Aquí el estado absorbente es 3

Cuando inicia en 1, 4, 5, ó 6 los estados recurrentes son 1 y 6, y permanecen en cualquiera de ellos.



Forma Canónica de la matriz de probabilidad

$$P = \begin{array}{c} \begin{matrix} 3 & 1 & 6 & 4 & 5 & 2 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0.5 & 0.3 & 0 \\ 0 & 0 & 0.4 & 0.3 & 0.3 & 0 \\ 0.7 & 0 & 0 & 0 & 0 & 0.3 \end{bmatrix} \end{array}$$

4.2 Análisis de resultados

Estos mismos problemas se analizaron en la computadora, obteniéndose los siguientes resultados:

Problema 1.

Matriz de adyacencia

```
1 0 0 0 0 0 0
1 0 1 0 0 0 0
0 1 0 1 0 0 0
0 0 1 0 1 0 0
0 0 0 1 0 1 0
0 0 0 0 1 0 1
0 0 0 0 0 0 1
```

Matriz de Alcanzabilidad R

```
1 0 0 0 0 0 0
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
0 0 0 0 0 0 1
```

Matriz de los Componentes Fuertemente Conectados RQ

```
1 0 0 0 0 0 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 1 1 1 1 1 0
0 0 0 0 0 0 1
```

* * * Clasificación de Estados * * *

Clases
de Equivalencia

1 = { 1 } Recurrente(s) Absorbente(s) Grado Ext. Graf.Cond. = 0

2 = { 2 3 4 5 6 } Transitorio(s) Grado Ext. Graf.Cond. = 2

3 = { 7 } Recurrente(s) Absorbente(s) Grado Ext. Graf.Cond. = 0

Matriz de adyacencia de la Digráfica Condensada

	1	2	3
1	0	0	0
2	1	0	1
3	0	0	0

Matriz Canónica

	1	7	2	3	4	5	6
1	1.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	1.000	0.000	0.000	0.000	0.000	0.000
2	0.400	0.000	0.000	0.600	0.000	0.000	0.000
3	0.000	0.000	0.400	0.000	0.600	0.000	0.000
4	0.000	0.000	0.000	0.400	0.000	0.600	0.000
5	0.000	0.000	0.000	0.000	0.400	0.000	0.600
6	0.000	0.600	0.000	0.000	0.000	0.400	0.000

Iniciando en el estado 2, como en el problema tenemos la siguiente tabla de distribución de probabilidades

Estados	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Iteración 0	0	**1**	0	0	0	0	0
Iteración 1	0.4	0	**0.6**	0	0	0	0
Iteración 2	**0.4**	0.24	0	0.36	0	0	0
Iteración 3	**0.496**	0	0.288	0	0.216	0	0
Iteración 4	**0.496**	0.115	0	0.259	0	0.13	0
Iteración 5	**0.542**	0	0.173	0	0.207	0	0.077

De acuerdo a la tabla de distribución de probabilidades, se observa que iniciando en el estado 2, la cadena tiende a quedarse o estacionarse en el estado 1.

Revisando la tabla de distribución de probabilidades es un poco difícil observar el comportamiento de la cadena, por lo cual el programa cuenta con la gráfica.

Respecto a las variaciones de decimales que existen en la tabla conviene mencionar que es un error de redondeo al presentar los datos, pero no afecta en el cálculo interno. En caso de necesitar mayor precisión se ajustaría el programa al número de dígitos necesarios.

La matriz canónica para este ejemplo coincide puesto que ambos tipos de estado son recurrentes absorbentes.

Problema 2

Matriz de adyacencia	Matriz de Alcanzabilidad R	Matriz de los Componentes Fuertemente Conectados RQ
0 0 0 0 0 1	1 0 0 0 0 1	1 0 0 0 0 1
0 1 1 0 0 0	0 1 1 0 0 0	0 1 0 0 0 0
0 0 1 0 0 0	0 0 1 0 0 0	0 0 1 0 0 0
1 0 0 1 1 0	1 0 0 1 1 1	0 0 0 1 1 0
0 0 0 1 1 1	1 0 0 1 1 1	0 0 0 1 1 0
1 0 0 0 0 0	1 0 0 0 0 0	1 0 0 0 0 0

* * * Clasificación de Estados * * *

Clases
de Equivalencia

1 = { 1 6 } Recurrente(s) Grado Ext. Graf.Cond. = 0

2 = { 2 } Transitorio(s) Grado Ext. Graf.Cond. = 1

3 = { 3 } Recurrente(s) Absorbente(s) Grado Ext. Graf.Cond. = 0

4 = { 4 5 } Transitorio(s) Grado Ext. Graf.Cond. = 2

Matriz de adyacencia de la Digráfica Condensada

	1	2	3	4
1	0	0	0	0
2	0	0	1	0
3	0	0	0	0
4	1	0	0	0

Matriz Canónica

	1	6	3	2	4	5
1	0.000	1.000	0.000	0.000	0.000	0.000
6	1.000	0.000	0.000	0.000	0.000	0.000
3	0.000	0.000	1.000	0.000	0.000	0.000
2	0.000	0.000	0.700	0.300	0.000	0.000
4	0.200	0.000	0.000	0.000	0.500	0.300
5	0.000	0.400	0.000	0.000	0.300	0.300

Iniciando en el estado 1 los resultados son los siguientes:

Estados	*1*	*2*	*3*	*4*	*5*	*6*
Iteración 0	1	0	0	0	0	**0**
Iteración 1	0	0	0	0	0	**1**
Iteración 2	**1**	0	0	0	0	0
Iteración 3	0	0	0	0	0	**1**
Iteración 4	**1**	0	0	0	0	0

En el estado 2

Estados	*1*	*2*	*3*	*4*	*5*	*6*
Iteración 0	0	1	0	0	0	0
Iteración 1	0	0.3	**0.7**	0	0	0
Iteración 2	0	0.09	**0.91**	0	0	0
Iteración 3	0	0.27	**0.973**	0	0	0
Iteración 4	0	0.008	**0.992**	0	0	0

.
.
.

En el estado 5

Estados	*1*	*2*	*3*	*4*	*5*	*6*
Iteración 0	0	0	0	0	**1**	0
Iteración 1	0	0	0	0.3	0.3	**0.4**
Iteración 2	**0.46**	0	0	0.24	0.18	0.12
Iteración 3	0.168	0	0	0.174	0.126	**0.532**
Iteración 4	**0.567**	0	0	0.125	0.09	0.218

En el estado 6

Estados	*1*	*2*	*3*	*4*	*5*	*6*
Iteración 0	0	0	0	0	0	**1**
Iteración 1	**1**	0	0	0	0	0
Iteración 2	0	0	0	0	0	**1**
Iteración 3	**1**	0	0	0	0	0
Iteración 4	0	0	0	0	0	**1**

De acuerdo a las tablas de distribución iniciar en los estados 1, 4, 5, ó 6, independientemente de las probabilidades de transición que presenten, llegan a los estados transitorios 1 y 6

Iniciar en los estados 2 ó 3 nos llevan inmediatamente al estado absorbente 3

En general, analizar una cadena de Markov con un número pequeño de estados (digamos menor a 5) lleva una serie de operaciones que se pueden realizar con lápiz y papel, pero para cadenas mayores, este proceso es muy engorroso.

Hasta aquí el programa nos ha dado una gráfica de estados, un análisis de la cadena y un estado de la misma conforme se simulan el cambio de transición.

Al comparar los resultados obtenidos mediante lápiz y papel, y los que arroja el programa y constatar que son iguales, podemos decir que hasta aquí se cumplió con los objetivos esperados.

4.3 Beneficios

Los beneficios de un sistema en computadora son múltiples, pero por mencionar los más notorios:

- Contamos con software que nos permite introducir un esquema de vector de distribución y matriz de transición que nos muestra paso a paso las transiciones de *n* pasos.

- El esquema de la cadena de Markov puede ser grabada a un archivo y recuperada desde él.

- Podemos visualizar la gráfica (para cadenas menores a 21 estados) y comprobar visualmente la transición de un estado a otro, según, simulamos el siguiente paso con la tecla F6.

- Tenemos a disposición un Sistema de Ayuda que nos permite consultar conceptos acerca de las cadenas de Markov.

- Se cuenta con una evaluación de la cadena de Markov que nos permite vislumbrar por adelantado, las características principales de la cadena como son conectividad, forma canónica y clases de equivalencia; y,

- Opciones de Impresión para la cadena de Markov, la simulación en el paso *n* del esquema de la cadena de Markov, y el análisis de la misma.

La información que se obtiene a través de estas características proporcionan un panorama amplio de la cadena de Markov, y en los procesos de decisión esto permite elegir el camino más adecuado a los intereses de lo que se persigan.

4.4 Limitaciones

Dentro de las limitaciones encontramos las siguientes:

- En cierta forma es una limitación el que este programa trabaje bajo Windows 3.1 o superior, ya que tenemos computadoras que no cuentan con este software, pero la mayoría si lo tiene.

- En la introducción de cadenas de Markov, la limitante es el número de nodos por cadena, sólo podemos introducir esquemas hasta de 20 nodos.

- En la matriz canónica de la cadena, agrupa los estados por clases recurrentes en primer nivel, y después las clases transitorias; pero no distingue de las clases recurrentes absorbentes y las que sólo son recurrentes.

- Se tiene un máximo de 20 nodos para una gráfica de Markov.

- Para cambiar algún elemento del esquema de la cadena de Markov (conformado por la matriz de transicion de un paso y un vector de distribución inicial), es necesario proporcionar cada parte del esquema completo, es decir, si deseamos cambiar algún elemento de la matriz de transición, se deberá introducir completa; asímismo si se quisiera cambiar algún elemento del vector inicial se tendría que introducir completamente.

- Respecto a la impresión esto es sólo posible para el esquema de la cadena, la simulación del esquema cuando cambia de un estado a otro, y el análisis de la cadena.

- Aún cuando en la simulación del esquema de la cadena, en la presentación, los resultados están redondeados a tres dígitos significativos, en las operaciones internas se consideran con todos los necesarios.

- El programa presenta la información en cuatro ventanas que corresponden a datos, simulación, gráfica y evaluar; las primeras tres ventanas tienen las siguientes características : una barra de título , un menú de control, los botones de maximizar, minimizar y restaurar, una área cliente (en esta área se presenta la información) y su marco correspondiente. A falta de las barra de desplazamiento horizontal y vertical, el área cliente esta limitada al tamaño de la ventana, por tanto los datos que excedan la ventana no podrán visualizarse cuando esta sea menor que la pantalla o cuando se presente en forma de mosaico.

- No hay simulación automática.

Capítulo 5

Propuestas Futuras

5.1 Descripción de Detalles para Mejorar el Programa.

Con el análisis de resultados y el desglose de limitaciones desarrollados en el capítulo anterior se observa lo siguiente:

a) Sólo es posible manejar una Cadena de Markov con 20 estados, es decir, una matriz cuadrada de 20 x 20 con un tipo de datos real (float), junto con un vector de 20 registros de tipo real, por el tamaño que se ocupa en memoria y se explica en f).

b) Gráficamente se puede representar 20 estados en la pantalla igual al número de la matriz.

c) La forma de Simular es por medio de la tecla F6 y no hay una forma automática.

d) En la impresión sólo es posible para los datos iniciales, la simulación y el análisis de la cadena.

e) La aplicación está dividida en cuatro partes, cada parte se maneja en una ventana con diferentes limitaciones.

f) Para el análisis de una Cadena de Markov el tipo de datos que se maneja es a nivel de bits, usando para ello el tipo de variable *unsigned long int,* por lo tanto dicho análisis no tiene problemas con 20 estados.

g) No tiene una opción donde sólo modifique un renglón de la matriz o determinados elementos.

5.2 Alternativas para Futuras Modificaciones.

Una vez descritas brevemente las principales limitaciones se pueden proponer las siguientes alternativas para mejorar el programa.

En el análisis de una Cadena de Markov con un número de estados, mayor al planteado, se tendría que manejar dinámicamente la memoria con el fin de que los procesos que se realizan no minimicen la rapidez para la presentación gráfica, o en la presentación de datos o simplemente no saturen la memoria base. En el programa la asignación de memoria para las variables es estática, es decir, siempre que se ejecuta el programa, se reservan 20 espacios para el vector y 400 para la matriz, todos de tipo real, independientemente si son utilizados o no.

Manejando dinámicamente la memoria su asignación sería de tal forma que sólo reservaría los espacios de acuerdo al número de estados que se procesen.

En este caso, para el manejo de memoria se debe utilizar las funciones de C++ como ***GlobalAlloc, GlobalFree, GlobalReAlloc, GlobalSize, GlobalLock*** , así como sus respectivas instancias.

Por Ejemplo, para utilizar memoria dinámica con la Clase Cadena una forma sería de siguiente manera:

```
class Cadena{
        float mi[20][20],mf[20][20];
        float vi[20],vf[20];
        int edo[20];
        int estado;
public:                         // Funciones Miembro
        int nodos;
        Cadena(void);
        ~Cadena(void);
        void  Inicializa(void);
        void  ObtMatriz(int conta,int contb,float num);
        void  ObtNumNodos(int n);
        void  Igualar(float MI[20][20],float MF[20][20],float VI[20],float VF[20]);
        void  ImpreMatriz(HDC hdc,int n,int tam);
        void  ImpreMatriz(HDC hdc,int n);
        void  Igualar(void);
        void  Multiplica(void);
        int   ObtEstado(void);
        void  ObtVector(int cont,int num);
        float ValorMatriz(int cont,int cont1);
        float Suma(void);
        float Suma(int i);
        void  Vertices(HDC hdc,int t);
        void  guard(char * nombrearc);
        void  AbreCad(char * Archimk);
        friend void AnalizaCad(Cadena Markov,char *Arc_resul);
        friend BOOL MenuImpresion(HDC PrnDC,Cadena M,char *Arch, int op);
                };
Handle hMem;
 hMem  =  GlobalAlloc ( GMEM_MOVEABLE, 1024 );
Cadena =GlobalLock( hMem);
...
```

Donde 1024 es la cantidad de memoria asignada a la variable Cadena.

En la Gráfica se presentan como máximo 20 estados, para no exceder el tamaño de la matriz de trancisión, definida para operaciones a nivel de bits (se realizan en variables unsigned long int operaciones de hasta 32 bits), pero es posible presentar hasta 45 estados sin problemas. Un número mayor de estados, por ejemplo, mayor de 50, implicaría reducir el tamaño de las circunferencias que representan los estados; aumentar la distancia equidistante, de las mismas, al centro de la pantalla; y, disminuir el tamaño de la fuente del tipo de letra que se usa para etiquetar los valores de los nodos y sus probabilidades de transición.

En dónde se harían las correcciones es en la siguiente parte de código

```
void Cadena::Vertices(HDC hdc,int t)
        {
        int ceny=200,cenx=250,ancho=40,cx,cy;
        struct ubica {
                        int p1,p2,p3,p4,f1,f2,f3,f4;
                        }pos[100];
        float ang=0, pi= 3.141592;
        float  grados=0;
        int   n,x,x1,y,y1,k,dir[4];
        float val;
        char sal[3];
        grados=2*pi/nodos;n=0;
```

Y las variables a modificar sus valores son **ceny, cenx** y **ancho.**

Para la impresión de la gráfica se tendría que introducir una variable para escalar las figuras, con el propósito de representar tal como vemos la gráfica en la pantalla, en la impresión, ya que las coordenadas de la pantalla difieren de las coordenadas de la impresora. La variable *t* ya está definida, solamente faltaría multiplicar las variables que generan las figuras por ésta. De esta forma cuando se llame al procedimiento *Vértices* de la clase *Cadena*, se pasaría el valor como parámetro. Por ejemplo, definiendo una variable MKV del tipo Cadena el llamado sería de la siguiente forma:

Cadena MKV;
MKV.Vertices(hdc, 30);

en donde 30 es el valor de *t*, el cual da el tamaño o escala en la que presentarán las figuras de la gráfica en la página a imprimir.

Se podría también agregar una opción de configuración de impresoras, y así se podrían configurar las impresoras desde el programa, y no solamente desde el administrador de

impresión de Windows. El procedimiento sería similar al siguiente[1] en el procedimiento *FrameWndProc* y en la rutina del *WM_COMMAND* .

```
case IDM_DEVMODE:
 if (!DefaultDrucker())
  MessageBox(hWnd, "No se pudo localizar la impresora", NULL, MB_OK);
 lstrcpy(szDriverFile, lpTreiber);
 lstrcat(szDriverFile, ".DRV") ;
 hDriver = LoadLibrary(szDriverFile);
 if (hDriver < 32)
  return(FALSE);
 lpfnDevMode = (LPFNDEVMODE)GetProcAddress(hDriver, "EXTDEVICEMODE");
 if (lpfnDevMode != NULL)
 {
  memset(&dmIn,0x00, sizeof(DEVMODE));
  lstrcpy(dmIn.dmDeviceName,lpDrucker);
  dmIn.dmSpecVersion = DM_SPECVERSION;
  dmIn.dmDriverVersion = 0;
  dmIn.dmDriverExtra = 0;
  dmIn.dmFields = DM_ORIENTATION | DM_COPIES | DM_PAPERSIZE;
  dmIn.dmSize = sizeof(DEVMODE);
  dmIn.dmOrientation = dmOut.dmOrientation;
  dmIn.dmCopies = dmOut.dmCopies;
  dmIn.dmPaperSize = dmOut.dmPaperSize;

  wErgebnis = lpfnDevMode(hWnd, hDriver, (LPDEVMODE)&dmOut, lpDrucker, lpSchnittstelle,
   (LPDEVMODE)&dmIn, NULL, DM_IN_PROMPT| DM_IN_BUFFER | DM_OUT_BUFFER);
  if (wErgebnis == IDOK)
   bDevMode = TRUE;
  else
   bDevMode = FALSE;
  InvalidateRect(hWnd, NULL, TRUE);
 }
 FreeLibrary(hDriver);
```

Para realizar una simulación automática, se puede hacer ya sea oprimiendo una tecla para que entre en un ciclo con cierto numero de iteraciones o que entre en un ciclo infinito y se detenga con la opresión de otra tecla, agregando la opción de WM_CHAR en el procedimiento principal *FrameWinProc*, en el procesamiento de mensajes. Por ejemplo el siguiente código genera un ciclo multiplicando 500 veces.

```
case WM_CHAR:
        if (wParam = = "C" || wParam = = "c")
        for (i =0;i<500 ;i + +)
                {
                Markov.Multiplica( );
```

[1] Bär, J; Bauder, I..(1995) pag. 197.

```
        }
InvalidateRect(hWnd, NULL, TRUE);
break;
```

Para el caso de una simulación concurrente el procedimiento es similar al anterior, salvo que se tendría que agregar otra condición que permita tanto para parar el procedimiento como continuar donde se quedó.

El programa presenta la informacion en cuatro ventanas, tres de ellas (datos, simulación y gráfica) tiene la misma apariencia, contienen foco, marco, menú de ventana, y la sintaxis es igual en forma y muy similar en código; sólo la que presenta el analisis de la Cadena de Markov (evaluar) tiene, adicional a las otras tres, barras de desplazamiento tanto horizontal como vertical y su sintaxis de código es diferente. Cuando las tres primeras ventanas contengan información que exceda al area cliente de éstas, no se podrá visualizar su contenido completo; por lo que la falta de las barras de desplazamiento vertical y horizontal se convierte en un problema.

Para solucionar el problema existen dos formas: una, grabar los datos en un archivo y presentarlos en una ventana de edición que ya tiene estas las barras de desplazamiento, tal como se realiza en la ventana de *Evaluación* al presentar el análisis de la Cadena de Markov. La otra forma, consiste en mapear las coordenadas de la pantalla con las variables ligadas a las barras de desplazamiento. Esto involucra muchas variables y comandos por su complejidad, y en este trabajo, no se da una solución detallada, pero se mencionan los comandos que deberán intervenir en la mdificación como son:

WM_CREATE	aquí se crea su instancia
WM_SIZE	aquí se determina su tamaño
WM_VSCROLL	aquí se la ubicación vertical del cursor
WM_HSCROLL	aquí se la ubicación horizontal del cursor
WM_PAINT	aquí se debe presentar la información
ostrstream(xxx,160)	este representa una variable xxx que contiene la información hasta 160 caracteres (tanto xxx, como 160 se pueden alterar).

La estacionaridad se puede determinar si siempre se rige por una matriz de transición regular por la fórmula siguiente $(1\text{-}P) \bullet q = 0$. Para poder determinar esta estacionaridad tanto de la matriz como del vector, se estima debe aplicarse algún método de resolución de sistemas de ecuaciones, agregándose como una opción más, para que de esta manera se pudiese orientar a aplicaciones más especificas donde se requiera conocerla en forma rápida. Esta opción se insertaría en el procedimiento *FrameWinProc* y su formato será de una forma similar a la siguiente insertada en las rutinas de *WM_COMMAND:*

```
case ID_METODO :
        llamado del  procedimiento que permita la solución del sistema de ecuaciones.
        break;
```

El procedimiento para la solución de un sistema de ecuaciones podría ser el de Gauss, Gauss-Jordan, Crout, etc.

En el procedimiento del análisis de la Cadena de Markov se realizan operaciones a nivel de bit, porque es más rápido y se economiza memoria; como el lenguaje permite el manejo de bits, se utiliza el tipo de variable: entero largo sin signo (***unsigned long int***) de 32 bits (4 bytes), en este caso cada bit representa un estado, por tanto se manejan sin problemas 20 estados, si se utilizará una variable de tipo real se tendrían 64 bits de esta forma también 64 estados, pero para una variable de este tipo no es posible modificar los bits que puede representar, y aun cuando fuera posible nuevamente se restringe a 64 estados manejables, por lo tanto es mejor declarar un bloque de memoria para poder utilizar más de 32 bits y por consecuencia más estados.

Para modificar elementos de la matriz se tiene también que comprobar que el renglón a modificar sume 1. Cuando se llame a la opción puede ser ya sea llamando elemento a elemento y comprobar la suma o pedir todo el renglón comprobándola al mismo tiempo.
Para modificar un elemento se puede realizar por el siguiente procedimiento:

```
Modifica(int x, int y )
{
        catmsg(titel,x,y,"El elemento ");
        MessageBox(hwndMDIClient,titel,"Matriz",
                MB_ICONINFORMATION|MB_OK);
        DialogBox(hInst,"MatInputDiaBox",hwnd,fpfnMatDiaProc);
        if (ValidaNum())        Markov.ObtMatriz(x,y,atof(numstring1));
        else    MessageBox(hwndMDIClient,"Valor Incorrecto",
                        "Error",MB_ICONQUESTION|MB_OK);
        if (errorSum(hwndMDIClient,Markov.Suma( x ) ) )
        Markov.Igualar( );
}
```

Este procedimiento tiene que ser una función amiga de la clase *Cadena* para que tenga compatibilidad con la clase y puéda manipular los elementos. Donde ***x*** y ***y*** son los parámetros que determinan el elemento de la matriz a modificar.

Para el procedimiento de modificación por renglón deberá ser similar a la siguiente forma:

```
Modifica(int x )
{
int i;
```

```
for (i=0;i<Markov,nodos;i + +)
        {
        catmsg(titel,x,i,"El elemento ");
        MessageBox(hwndMDIClient,titel,"Matriz",
                MB_ICONINFORMATION|MB_OK);
        DialogBox(hInst,"MatInputDiaBox",hwnd,fpfnMatDiaProc);
        if (ValidaNum())        Markov.ObtMatriz(x,i,atof(numstring1));
        else    MessageBox(hwndMDIClient,"Valor Incorrecto",
                        "Error",MB_ICONQUESTION|MB_OK);
        }
if (errorSum(hwndMDIClient,Markov.Suma( x )))
Markov.Igualar( );
}
```

Al igual que la función anterior ésta debe ser función amiga de la clase *Cadena* para que tenga compatibilidad con la clase. La variable ***x*** representa el renglón a modificar.

Bibliografía

Bär, J. Bauder I.:(1995) WINDOWS 3.1 INTERNO, AlfaOmega Grupo Editor, S.A. de C.V., México.

Murray, William H. y Pappas Chris H. :(1994); PROGRAMACION DE APLICACIONES PARA WINDOWS NT, Ed. Osborne/McGraw-Hill, México.

Perry Greg (1993): APRENDIENDO PROGRAMACION ORIENTADA A OBJETOS CON TURBO C++ EN 21 DIAS; Ed. Prentice Hall Hispanoamericana; México.

Schild, Herbert (1985): PROGRAMACION EN LENGUAJE C; Ed. McGraw-Hill, México.

Schild, Herbert (1992): TURBO C/C++, MANUAL DE REFERENCIA; Ed. McGraw-Hill, México.

En el desarrollo de este trabajo, el objetivo que persigue es llegar a demostrar una teoría, mediante los conocimientos adquiridos en la licenciatura.

Como primer paso fué muy importante realizar la investigación acerca de las cadenas de Markov, detectar las características, los procesos en donde se involucran, que variaciones presentan las diversas aplicaciones, etc; con el fin de tener una amplia visión del tema y establecer hasta que punto era viable desarrollar el trabajo.

Existen muchos fenómenos que no dependen de cómo se comportaron anteriormente, solamente de su presente como una Cadena de Markov, o pueden describirse de una manera similar como los procesos de nacimiento y muerte; la predicción del clima; la estadística en accidentes; siniestros, etc., cuyos resultados son utilizados en las diferentes disciplinas que los estudian. Pero para obtener resultados fue necesario estudiar la forma en que se relacionan los estados, donde se especifica la clasificación de los mismos a partir de sus características en transitorios, recurrentes, etc.; y éstos se analizan a partir de la matriz de probabilidades de transición y el vector inicial. Otra forma de llegar a los mismos resultados, es simulando el comportamiento de la cadena por medio de la multiplicación de la matriz de transición en "n" veces.

Estos procesos o procedimientos se realizaron por separado, comprobando una vez más que, la división de un problema en sus partes más elementales para estudio nos lleva al modelado del sistema; y en este caso a su programación.

Con la implementación del programa, una Cadena de Markov puede ser analizada más rápidamente con resultados precisos, que a su vez son elementos que permitirán al grupo o persona decisor tomar decisiones con oportunidad y acertación.

Los pequeños inconvenientes se resolvieron, aunque se llevaron más tiempo de lo esperado; por ejemplo: en el desarrollo del procedimiento para graficar, fue necesario aplicar conocimiento básicos como la ecuación de la recta, la fórmula de la pendiente, y otros conceptos de trigonometría que permitieron representar equitativamente las figuras. El conjunto de éstos conceptos se sistematizó en forma de procedimientos estructurados relacionados entre sí.

La aplicación de las operaciones a nivel de bit, se implementó con el fin de eficientar los procesos de análisis de la cadena, y para optimizar la memoria

Finalmente una vez implementado el programa, los resultados obtenidos de él fueron puestos a juicio, al comparar ejercicios desarrollados manualmente y las soluciones provistas por la computadora. Las soluciones coincidieron y por lo tanto se cumplió con el objetivo.

En la programación de cualquier resolución de un problema existen variaciones y limitaciones, pero siempre se busca que los resultados sean concretos dentro de un límite o

de lo contrario no tendría fin. Es necesario mencionar, además, que todo proyecto es mejorable y por tanto se plantearon los puntos donde se podrían realizar estas mejoras.

1. Bharucha-Reid A.T. (1960): ELEMENTS OF THEORY OF MARKOV PROCESSES AND THEIR APPLICATION; Ed. McGraw-Hill; New York, USA, Cap. 1.

2. Bhat, J. Narayan (1984): ELEMENTS OF APPLIED STOCHASTIC PROCESSES; 2ª edición; Ed. Jhon Wiley and Sons; New York; USA

3. Chover, Josua (1967): MARKOV PROCESS AND POTENTIAL THEORY; Ed.Jhon Wiley and Sons Inc.; New York; USA.

4. Coleman, Rodney (1976); PROCESOS ESTOCASTICOS; Ed. Limusa; México

5. Cox, D.R. y Miller, H.D.(1990): THE THEORY OF STOCHASTIC PROCESSES: Ed. Chapman and Hall, Gran Bretaña.

6. Hoel, Paul G.; Port , Sidney C. y Stone, Charles J. (1972): INTRODUCTION TO STOCHASTIC PROCESSES; Ed. Waveland Press Inc.; Los Angeles, USA. Cap. 1 y 2

7. Karlin, Samuel y Taylor Howard M.(1975): A FIRST COURSE IN STOCHASTIC PROCESSES; 2ª Edición, Ed. Academic Press; New York, USA.

8. Kemeny, J. y Snell, J. (1976): FINITE MARKOV CHAINS; Springer-Verlag, New York, USA.

9. Philips, Don T. (1969): A MARKOVIAN ANALISYS OF THE CONVEYOR SERVICE ORDERED; University Microfilms, Michigan, USA.

10. Prawda, Juan (1991): METODOS Y MODELOS DE INVESTIGACION DE OPERACIONES VOL. 2, MODELOS ESTOCASTICOS, Ed. Limusa;México.

11. Referencias de Internet.

12. Strang, Gilbert (1980): LINEAR ALGEBRA AND ITS APPLICATIONS; 2ª edición; Ed. Academic Press Inc., New York, USA.

13. Brookshear, J. Glenn:(1993)TEORIA DE LA COMPUTACION. LENGUAJES FORMALES, AUTOMATAS Y COMPLEJIDAD, Ed. Addison-Wesley Iberoamericana, S.A. ,USA.

14. Casper, Jhon(1994): OBJECT ORIENTED PROGRAMMING ANALYSIS, DESIGN AND IMPLEMENTATION, Computer Technology Research Corp.; South Carolina, USA, Cap.3.

15. Eckel, Bruce(1991):APLIQUE C++; Ed. McGraw-Hill, México.

16. Joyanes Aguilar , Luis (1993):METODOLOGIA DE LA PROGRAMACION, Ed. McGraw-Hill, México.

17. Korth, Henry S.F. (1992): FUNDAMENTOS DE BASE DE DATOS, Ed. McGraw-Hill, México.

18. Schild Herbert(1990):UTILIZACION DE C EN INTELIGENCIA ARTIFICIAL; Ed. Osborne/McGraw-Hill; Madrid, España.

19. Pressman, Roger S. (1993):INGENIERIA DEL SOFTWARE: UN ENFOQUE PRACTICO, 3ª Edición, Ed. McGraw-Hill/Interamericana de España, S.A., Madrid España.

20. Bär, J. Bauder I.:(1995) WINDOWS 3.1 INTERNO, AlfaOmega Grupo Editor, S.A. de C.V., México.

21. Heiny, Loren (1992): WINDOWS GRAPHICS PROGRAMMING WITH BORLAND C++, Ed. Jhon Wiley and Sons, New York, USA.

22. Murray, William H. y Pappas Chris H. :(1994); PROGRAMACION DE APLICACIONES PARA WINDOWS NT, Ed. Osborne/McGraw-Hill, México.

23. Perry Greg (1993): APRENDIENDO PROGRAMACION ORIENTADA A OBJETOS CON TURBO C++ EN 21 DIAS; Ed. Prentice Hall Hispanoamericana; México.

24. Schild, Herbert (1985): PROGRAMACION EN LENGUAJE C; Ed. McGraw-Hill, México.

25. Schild, Herbert (1992): TURBO C/C++, MANUAL DE REFERENCIA; Ed. McGraw-Hill, México.

DFD de Procesos Programa Principal
Analizador de Cadenas de Markov

Nivel 0

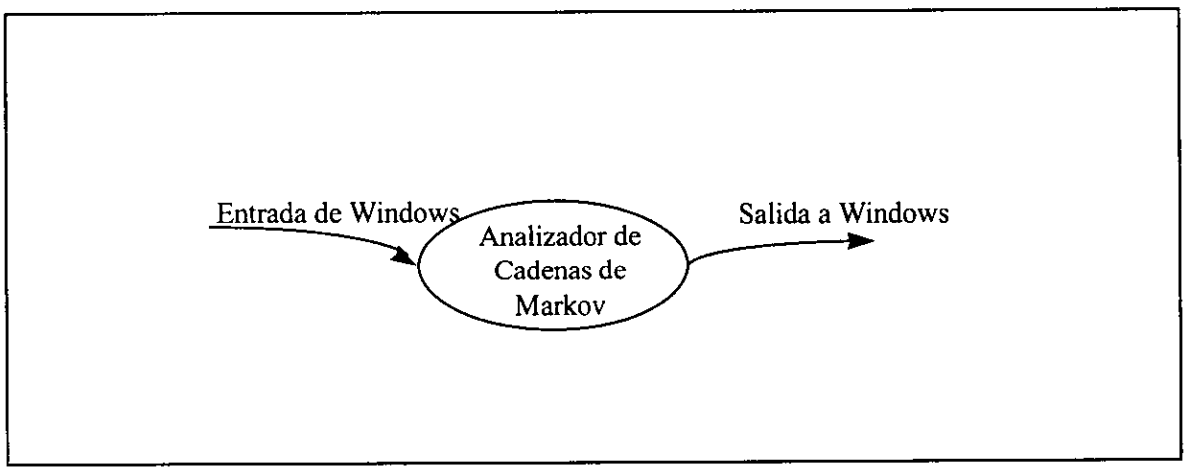


Diagrama de Flujo de Datos

Nivel 1

Sistema Analizador de Cadenas de Markov

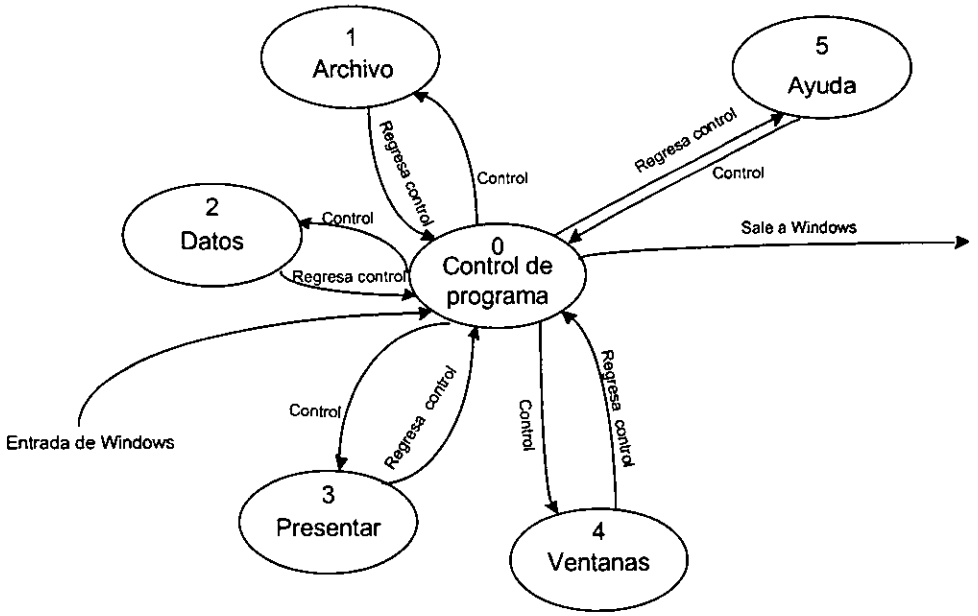


Diagrama de Flujo de Datos

Nivel 2

Sistema Analizador de Cadenas de

1 Archivo

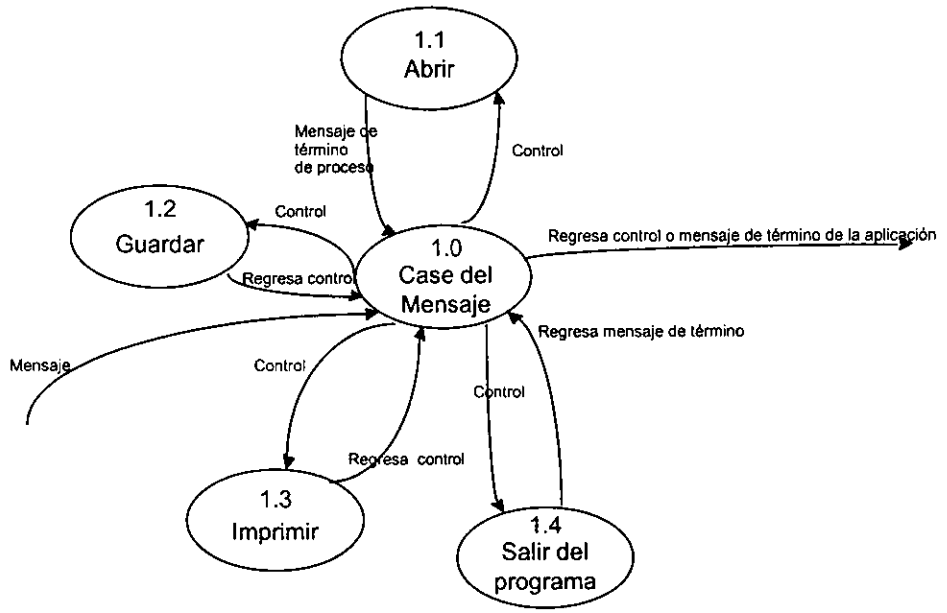


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

1.1 Abrir

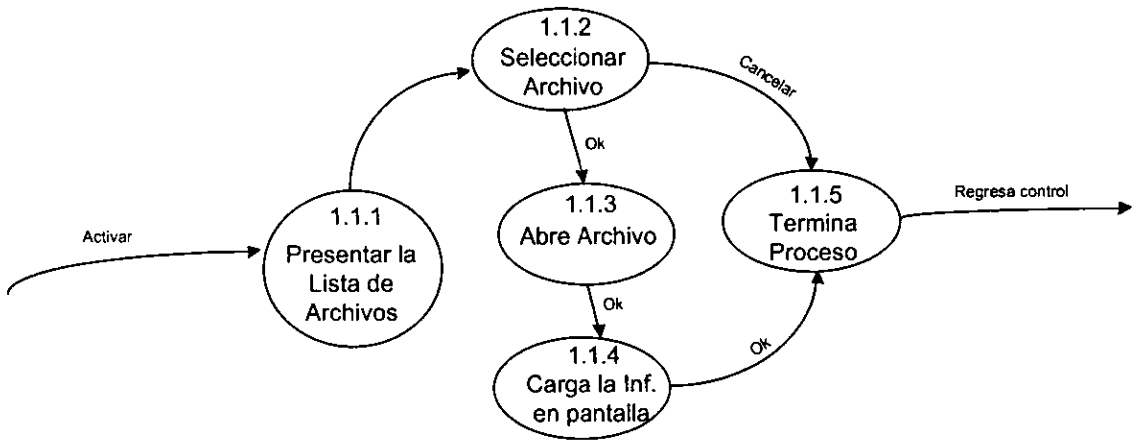


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

1.2 Guardar

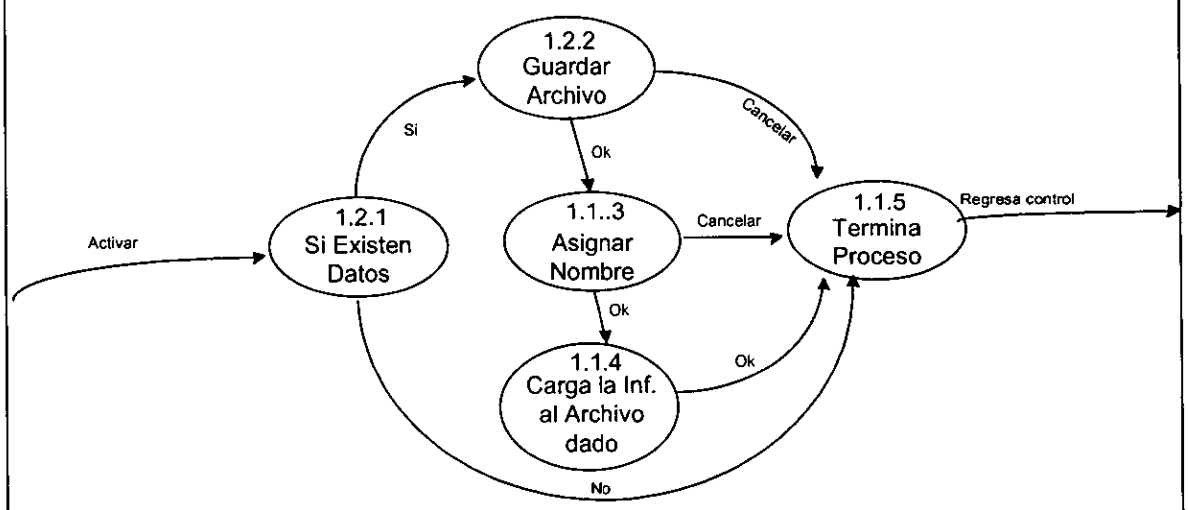


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

1.3 Imprimir

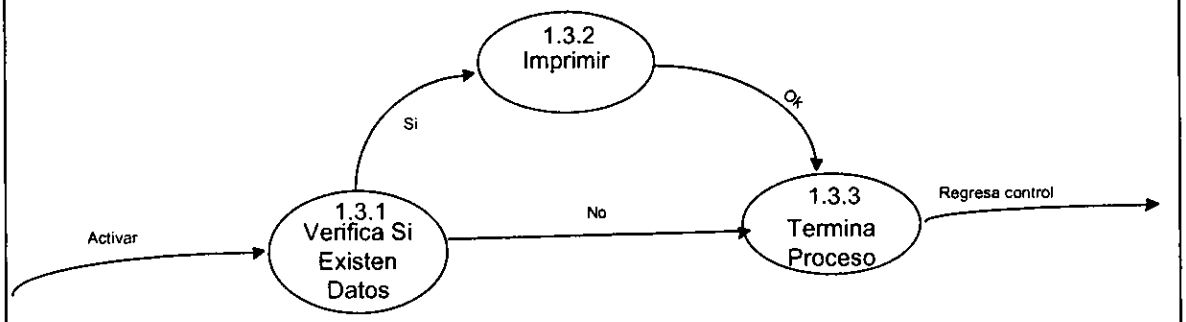


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

1.4 Salir

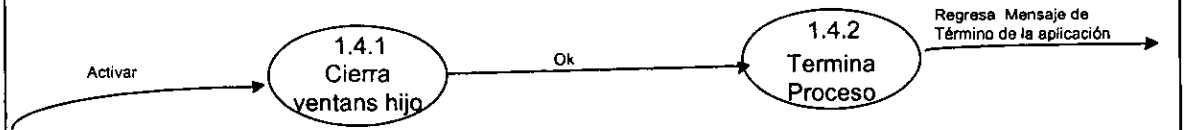


Diagrama de Flujo de Datos

Nivel 2

Sistema Analizador de Cadenas de Markov

2 Datos

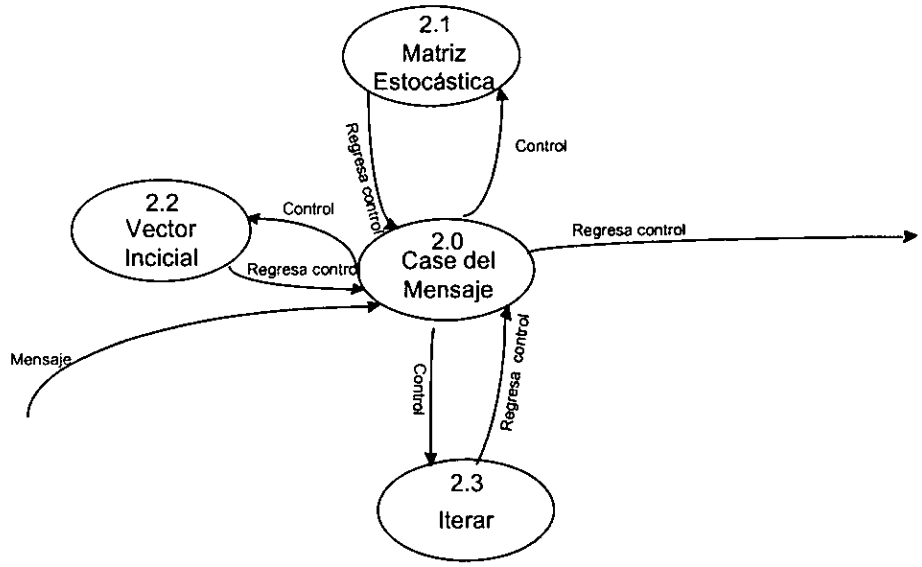


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

2.1 Matriz Estocástica

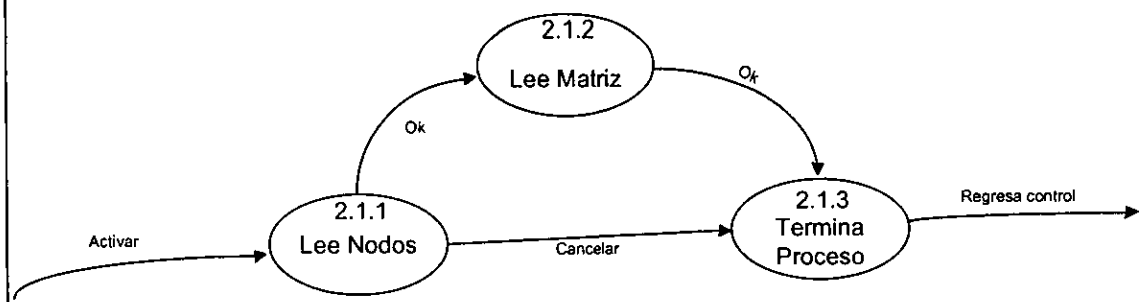


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

2.2 Vector Incial

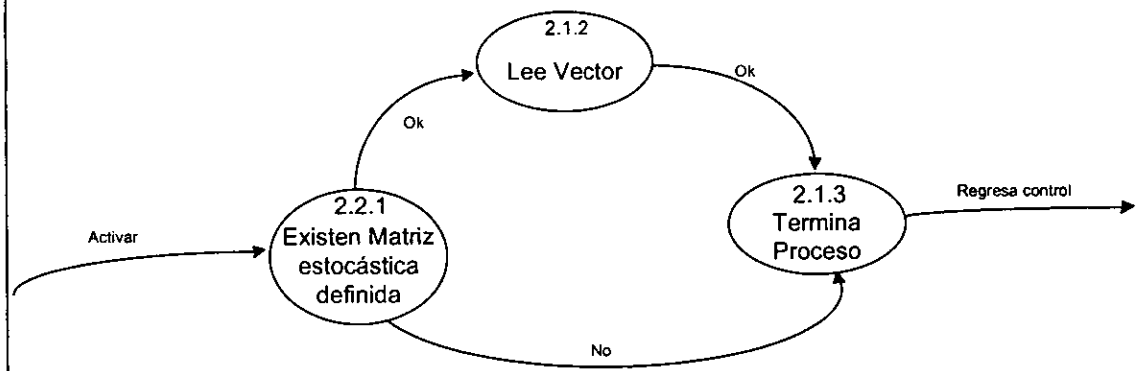


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

2.3 Iterar

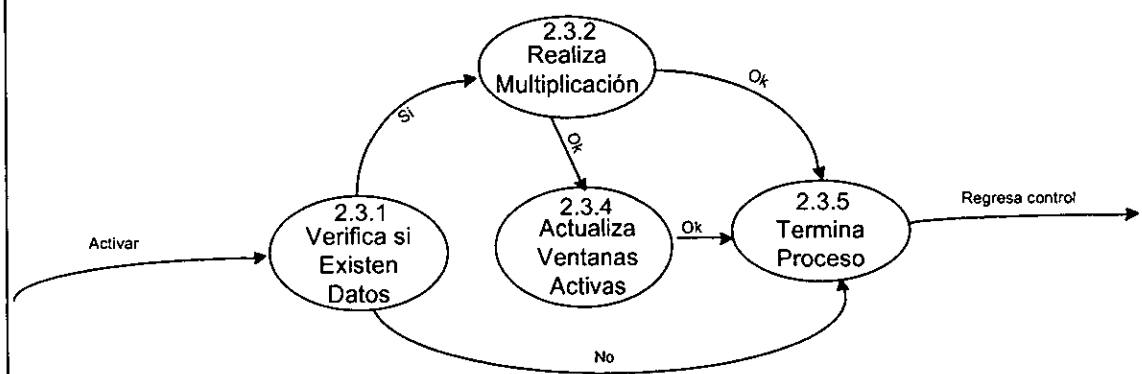


Diagrama de Flujo de Datos

Nivel 2

Sistema Analizador de Cadenas de Markov

3 Presentar

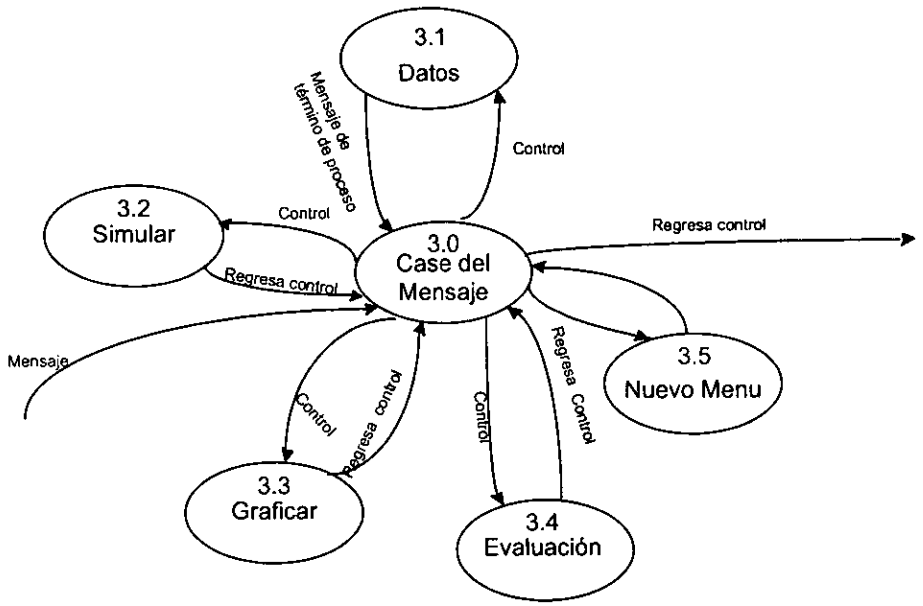


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

3.1 Datos

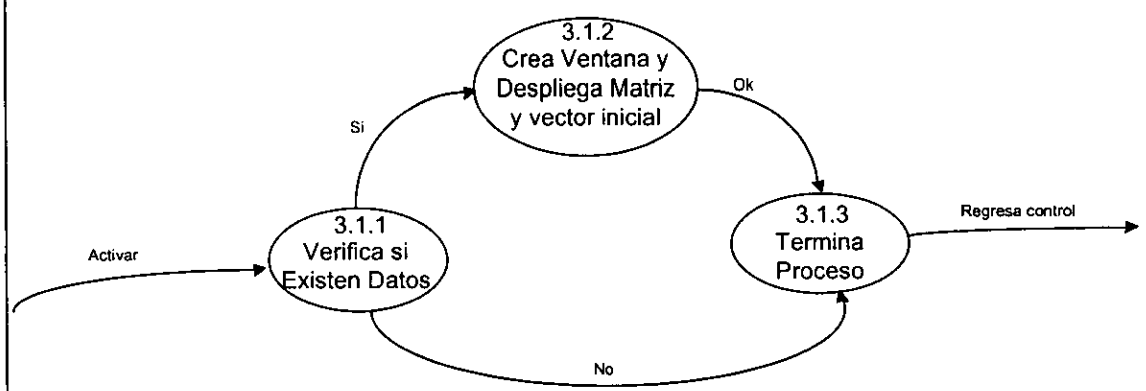


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

3.2 Simular

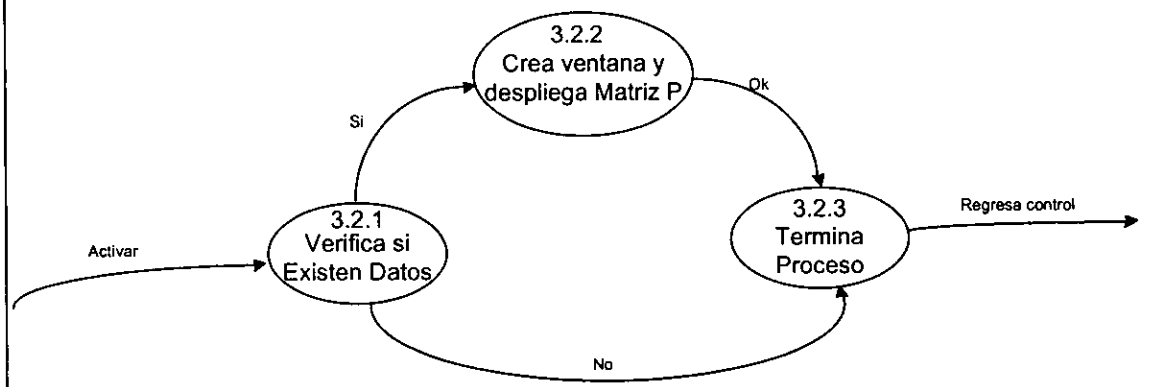


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

3.3 Gráfica

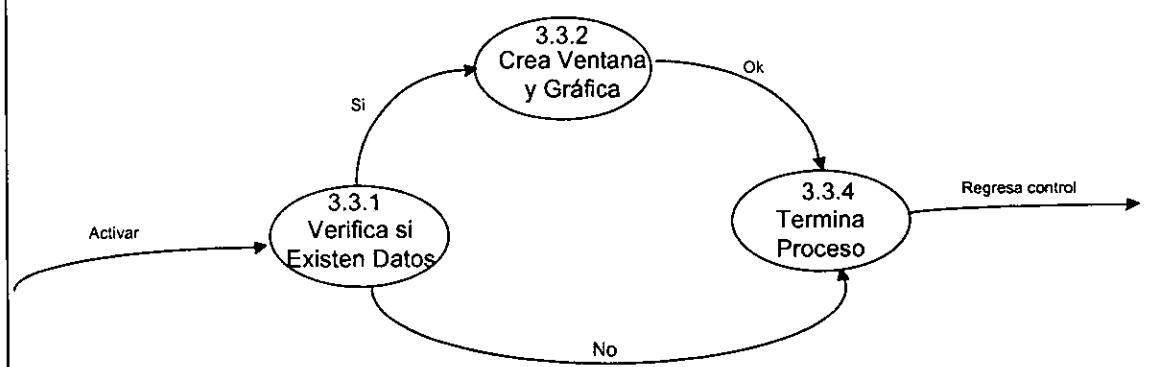


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

3.4 Evaluación

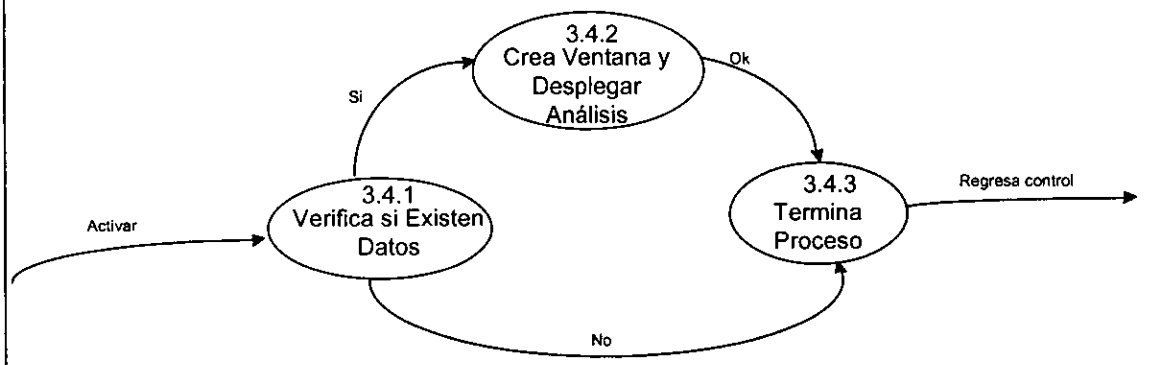


Diagrama de Flujo de Datos

Nivel 3

Sistema Analizador de Cadenas de Markov

3.5 Nuevo Menú

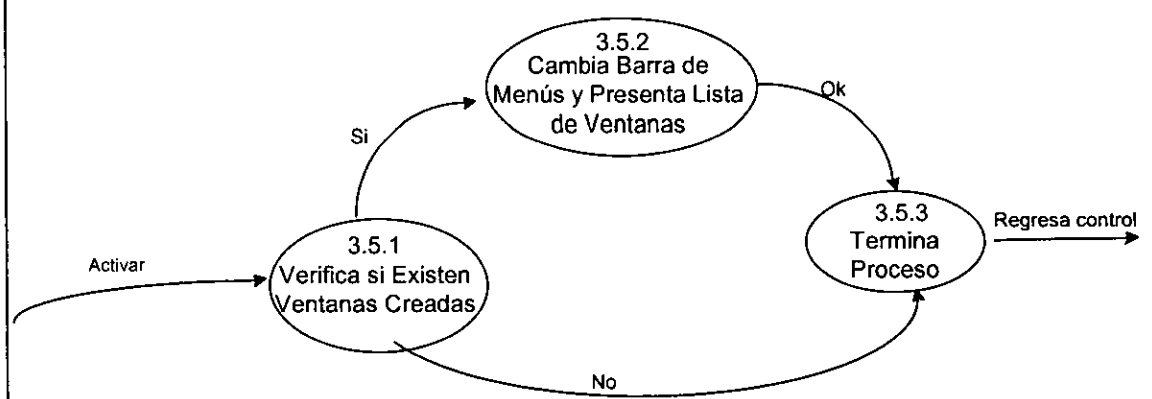


Diagrama de Flujo de Datos

Nivel 2

Sistema Analizador de Cadenas de Markov

4 Ventanas

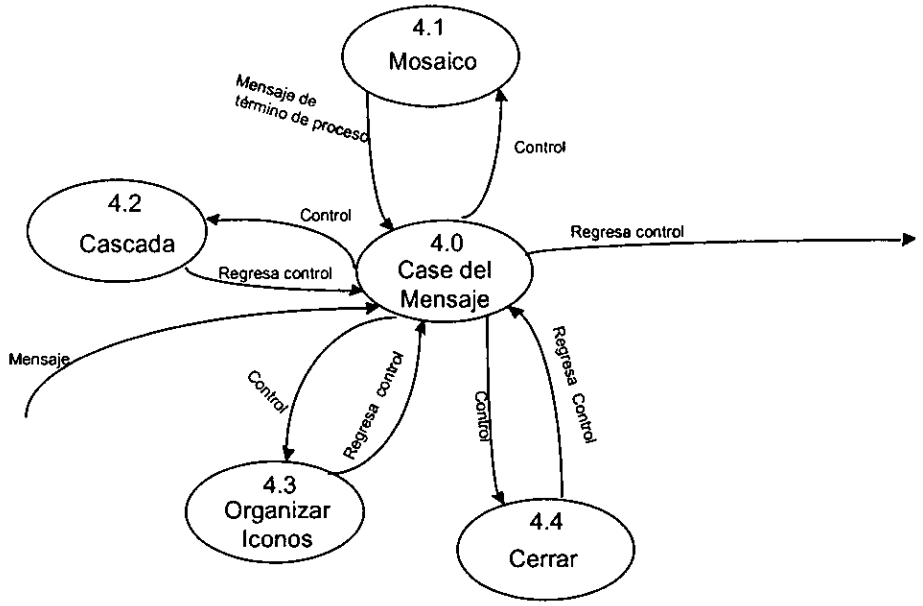


Diagrama de Flujo de Datos

Nivel 2

Sistema Analizador de Cadenas de Markov

5 Ayuda