2 Ej.

UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
" A C A T L A N "

FUNDAMENTOS DE PROGRAMACION HEURISTICA Y APLICACIONES

TESIS DE GRADO
QUE PARA OBTENER EL TITULO DE:
LIC. EN MATEMATICAS APLICADAS Y COMPUTACION
P R E S E N T A:

Fernando Delgado Ortega

Santa Cruz Acatlán, Edo. de México 1987

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ACATLAN"

COORDINACION DEL PROGRAMA DE ACTUARIA
Y MATEMATICAS APLICADAS Y COMPUTACION.

CAMAC-205/87.

SR. FERNANDO DELGADO ORTEGA
Alumno de la carrera de Matemáticas
Aplicadas y Computación.
P r e s e n t e.

De acuerdo a su solicitud presentada con fecha 9 de diciembre de 1986, me complace notificarle que esta Coordinación tuvo a bien asignarle el siguiente tema de tesis: "Fundamentos de -- Programación Heurística y Aplicaciones", el cual se desarro-- llará como sigue:

Introducción.
I.- Introducción a la Inteligencia Artificial (IA) y la Heurística.
II.- Fundamentación de la Programación Heurística.
III.- Introducción al Lenguaje LISP.
IV.- Lenguaje LISP como Herramienta a la Programación Heurística.
V.- Aplicaciones a Juegos.
Conclusiones y recomendaciones.

Asimismo fue designado como Asesor de Tesis el Sr. Fís. Mat.- Sergio V. Chapa Vergara.

Ruego a usted tomar nota que en cumplimiento de lo especifica do en la Ley de Profesiones, deberá prestar servicio social - durante un tiempo mínimo de seis meses como requisito básico- para sustentar examen profesional, así como de la disposición de la Coordinación de la Administración Escolar en el sentido de que se imprima en lugar visible de los ejemplares de la te sis el título del trabajo realizado. Esta comunicación deberá imprimirse en el interior de la tesis.

A t e n t a m e n t e
"POR MI RAZA HABLARA EL ESPIRITU"
Acatlán, Estado de México, 11 de agosto de 1987.

ACT. MAT. GERARDO MIGUEL CAIROS
Coordinador.

ENEP. ACATLAN
COORDINACION ACTUARIA Y
MATEMATICAS APLICADAS Y
COMPUTACION

I N D I C E

INTRODUCCION

Uno de los más apasionantes tópicos computacionales de los últimos tiempos es sin lugar a dudas, la Inteligencia Artificial, que se está convirtiendo en la piedra angular de los trabajos de desarrollo de la quinta generación de computadoras. El objetivo de este trabajo es presentar un estudio acerca de la Inteligencia Artificial y profundizar a detalle en una de sus más importantes áreas, la Programación Heurística.

Se ha seleccionado el tema de Programación Heurística por ser una rama importantísima de la Inteligencia Artificial y por ser una área hasta cierto punto poco desarrollada en México, que no debe ser descuidada en lo más mínimo para evitar que la dependencia tecnológica con el exterior sea mayor, por otra parte presenta importantes matices en lo que corresponde al entendimiento de la inteligencia humana y los procesos mentales del pensamiento.

Cabe mencionar que la aplicación a juegos, presentada en este trabajo, basa su justificación en la amplia gama de posibilidades de decisión, ya sea con conocimiento de la situación o bajo incertidumbre, que se presentan; lo cual en una aplicación particular presentaría sólo uno de los dos casos: decisión con conocimiento o con incertidumbre, además de que requeriría de un equipo interdisciplinario (una parte aportaría la programación y la otra los conocimientos de una área especifica) para que el proyecto a realizar se convirtiera en un Sistema Experto, y simulara lo más cercano a la realidad la experiencia los conocimientos y la inteligencia de un experto humano.

Por último las técnicas aplicadas en juegos constituyen la base primordial para la construcción de Sistemas Expertos, y aunque parezcan poco formales en realidad son tan formales como cualquier modelo matemático de tipo algorítmico, como se puede observar en la bibliografía especializada del tema (que es bastante por cierto) que constituye toda una teoría en la actualidad y ha sido por años área de aplicación de la IA.

El capítulo 1, ubica a la heurística en su exacta dimensión dentro de la IA y proporciona los conceptos básicos que permiten tener un soporte para el entendimiento de la filosofía misma de la heurística, pasando por su desarrollo a través de la historia.

En el capítulo 2 se presenta un amplio estudio de los fundamentos de la programación heurística que abarca temas específicos como su naturaleza, la cual una vez comprendida lleva perfectamente al entendimiento de sus objetivos y alcances.

INTRODUCCION

Uno de los más apasionantes tópicos computacionales de los últimos tiempos es sin lugar a dudas, la Inteligencia Artificial, que se está convirtiendo en la piedra angular de los trabajos de desarrollo de la quinta generación de computadoras. El objetivo de este trabajo es presentar un estudio acerca de la Inteligencia Artificial y profundizar a detalle en una de sus más importantes áreas, la Programación Heurística.

Se ha seleccionado el tema de Programación Heurística por ser una rama importantísima de la Inteligencia Artificial y por ser una área hasta cierto punto poco desarrollada en México, que no debe ser descuidada en lo más mínimo para evitar que la dependencia tecnológica con el exterior sea mayor, por otra parte presenta importantes matices en lo que corresponde al entendimiento de la inteligencia humana y los procesos mentales del pensamiento.

Cabe mencionar que la aplicación a juegos, presentada en este trabajo, basa su justificación en la amplia gama de posibilidades de decisión, ya sea con conocimiento de la situación o bajo incertidumbre, que se presentan; lo cual en una aplicación particular presentaría sólo uno de los dos casos: decisión con conocimiento o con incertidumbre, además de que requeriría de un equipo interdisciplinario (una parte aportaría la programación y la otra los conocimientos de una área especifica) para que el proyecto a realizar se convirtiera en un Sistema Experto, y simulara lo más cercano a la realidad la experiencia los conocimientos y la inteligencia de un experto humano.

Por último las técnicas aplicadas en juegos constituyen la base primordial para la construcción de Sistemas Expertos, y aunque parezcan poco formales en realidad son tan formales como cualquier modelo matemático de tipo algorítmico, como se puede observar en la bibliografía especializada del tema (que es bastante por cierto) que constituye toda una teoría en la actualidad y ha sido por años área de aplicación de la IA.

El capítulo 1, ubica a la heurística en su exacta dimensión dentro de la IA y proporciona los conceptos básicos que permiten tener un soporte para el entendimiento de la filosofía misma de la heurística, pasando por su desarrollo a través de la historia.

En el capítulo 2 se presenta un amplio estudio de los fundamentos de la programación heurística que abarca temas específicos como su naturaleza, la cual una vez comprendida lleva perfectamente al entendimiento de sus objetivos y alcances.

Se presenta un esbozo de enfoques de solución de problemas, tópico importantisimo para el desarrollo de los algoritmos de búsqueda.

Partiendo de lo anterior se presentan a continuación las estrategias que se utilizan en una búsqueda de tipo heurfstico, comenzando con las estrategias de búsqueda básicas, explicando los algoritmos de Búsqueda en Profundidad (BP) y Búsqueda Horizontal (BH) con sencillos ejemplos. De la misma manera se tratan las búsquedas de tipo mini-max, para finalmente tratar formalmente los algoritmos de poda alfa-beta y alfa-beta modificado.

El capítulo 3 comienza con una introducción al lenguaje LISP su diferencia con otros lenguajes de alto nivel y su historia.

A continuación se desarrollan temas específicos del lenguaje que representan el soporte básico para el tratamiento posterior de temas avanzados. Dentro de lo básico en este capítulo se encuentran las funciones primitivas, numéricas y de procesamiento de listas, para finalmente llegar a la recursión que es precisamente una de las potencialidades de éste lenguaje.

Por supuesto este capítulo no intenta abarcar todo el lenguaje LISP, sino solo el subconjunto necesario que conjuntamente con los temas del capftulo 4, proporcionan las herramientas para la buena realización del proyecto.

En el capftulo 4 se dan los lineamientos y fundamentos del LISP necesarios para la programación de los algoritmos tratados en el capftulo 2, comenzando con el tratamiento de registros y su manipulación, para continuar con nodos y árboles, que son parte fundamentalmente importante para la programación.

Después se da un ligero enfoque de como es que trabajan los algoritmos de búsqueda asi como un análisis de la complejidad del algoritmo alfa-beta. Finalmente se presentan sugerencias acerca de cómo programar los algoritmos de búsqueda, particularmente el alfa-beta.

En el capítulo 5, se tratan algunas aplicaciones a juegos donde la máquina adquiere un comportamiento se podría decir "inteligente". En realidad lo que sucede es que un programa maneja estupendamente las opciones que se presentan en un juego, precisamente por eficientes algoritmos de búsqueda que dan la impresión de que la máquina está pensando.

En la interacción con un usuario que sea un buen jugador podrá observarse hasta qué punto la inteligencia es programable, lo que finalmente se discutirá en las conclusiones de este proyecto.

CAPITULO 1

INTRODUCCION A LA INTELIGENCIA ARTIFICIAL (IA) Y LA HEURISTICA

1.1 INTRODUCCION

Hasta ahora nadie ha podido definir qué es la inteligencia. El ser humano es capaz de utilizarla de maneras tan diversas, muchas de ellas sin relación aparente, que cuando se intenta encontrar una definición que se aplique a todos los casos posibles, se llega a un laberinto sin salida. ¿ Cómo funciona la inteligencia para hacer una demostración matemática ? ¿ Cómo para hacer un poema ? Es muy diffcil encontrar rasgos comunes y a la vez esenciales entre ambos tipos de inteligencia.

Hay, sin embargo, un criterio que, aunque no puede aplicarse siempre, permite denotar una inmensa cantidad de actividades humanas atribuidas a la inteligencia. Según este criterio, la inteligencia es la capacidad para resolver problemas. Un hombre o una mujer inteligentes no sólo pueden resolver muchos problemas de los más diversos tipos, sino que pueden encontrar soluciones a problemas complicados. Mientras más problemas complicados es capaz de resolver una persona, se le considera más inteligente. Hay problemas tan complejos que sólo muy pocos pueden resolverlos. En el uso cotidiano del problema, los hay de tal complejidad que ni siquiera los más grandes genios los han podido resolver, como, por ejemplo, el problema del campo unificado, ante el cual fracasó nada menos que Einstein [1], y la conjetura de Goldbach, que, hecha hace casi tres siglos, no ha podido ser resuelta hasta el momento por ningún matemático [2].

Otro aspecto que debe tenerse en cuenta es el tipo de problemas que una persona puede resolver. Algunas son capaces de resolver diffciles problemas de ciencias exactas, pero fracasan lamentablemente cuando quieren encontrar una rima elegante; y viceversa, hay poetas extraordinarios que nunca pudieron aprender ni a sumar fracciones. Estas desconcertantes diferencias en el uso de la inteligencia muestran que, en muchos casos, no sólo es diffcil, sino imposible, saber si una persona es más o menos inteligente que otra.

Entre todos los problemas a los que puede enfrentarse la inteligencia humana, los lógicos y matemáticos destacan por ciertas caracterfsticas muy especiales. En primer lugar, se trata de problemas que se pueden plantear con toda precisión (por ejemplo, cuánto es la suma de dos números, cuál es la solución de determinada ecuación diferencial, o existe o no una relación deductiva entre la proposición A y la proposición B, etc); en segundo lugar, las soluciones de dichos problemas, cuando existen, son también precisas; en tercer lugar, tienen enorme importancia teórica, pues la ffsica, que es la principal ciencia natural se constituye utilizando teorías matemáticas, de manera que casi todos los problemas de dicha disciplina son

problemas matemáticos. En los últimos tiempos casi todas las ciencias están utilizando las matemáticas para alcanzar cada vez mayor claridad conceptual y métodos de descubrimiento más eficaces. De manera que muchos problemas cientfficos, incluso en el caso de las ciencias sociales, pueden plantearse, en nuestros dfas, y resolverse con métodos matemáticos. Por último, las matemáticas y la lógica tienen una incalculable utilidad práctica, puesto que, como dijo el gran filósofo inglés Francis Bacon, saber es poder. La ciencia natural permite crear tecnologfas industriales sumamente eficientes. Esto ha producido, en sólo dos siglos, un cambio revolucionario en nuestra manera de vivir. Lo mismo, aunque en menor medida está sucediendo en las ciencias sociales.

Se ve así que poder resolver la mayor cantidad de problemas matemáticos es del mayor interés para el ser humano, porque esta posibilidad le permite resolver cuestiones teóricas muy profundas, que satisfacen su insaciable afán de conocimiento, y porque, además, le permite resolver problemas prácticos de carácter social y económico.

Ahora bien, teóricamente, todo problema matemático puede reducirse a un problema de lógica. Basta saber que la matemática es una ciencia deductiva para comprender esta afirmación. En matemáticas, los teoremas se deducen de los axiomas. O sea, cuando se tiene el problema de querer probar la verdad de alguna afirmación matemática, hay que tener la capacidad de deducir lo que se quiere probar de los axiomas de la teorfa correspondiente. En este sentido, cualquier problema de matemáticas resulta un problema deductivo, pues todo problema puede reducirse a deducir un teorema partiendo de los axiomas de una teorfa.

Debido a que las operaciones necesarias para resolver problemas matemáticos y lógicos son, con frecuencia muy numerosas, un ser humano se ve imposibilitado de realizarlas. Aunque, en principio, puede hacerlas, en la práctica un conjunto de operaciones puede ser tan inmenso que una persona, por más capaz que fuera necesitaría miles de años para efectuarlas. Por eso ha sido imprescindible inventar máquinas que, debido a la rapidez con que realizan sus operaciones, tengan la capacidad de obtener resultados que, de otra manera, no prodrían alcanzarse. Como las operaciones matemáticas y lógicas han sido realizadas por seres inteligentes, la capacidad de las máquinas para hacerlas ha sido llamada inteligencia artificial.

1.2 DEFINICIONES Y CONCEPTOS

¿ Qué es en realidad la Inteligencia Artificial (IA) ?, hay varias respuestas admisibles a esta pregunta, como éstas:

a) Es la rama de la Computación que se dedica a la investigación y desarrollo de procesos que simulan un comportamiento inteligente.

b) Es el estudio de cómo la computadora puede hacer las cosas que en la actualidad la gente hace mejor [Rich 1983].

c) Es una tecnología que ha surgido en los últimos años que busca la creación de una máquina con inteligencia propia [Geva 1984].

De lo anterior se desprende una definición más formal:

definición de IA:

Es la parte de la Computación que se encarga de la investigación y elaboración de procedimientos que simulan la inteligencia humana. Usa técnicas desarrolladas a partir del estudio del sentido común y la experiencia en el campo del conocimiento humano.

Una vez que se ha entendido lo que es la IA, es importante señalar que ésta en sí se divide en partes, siendo una de ellas, precisamente la **Heurística**. Por ello es importante saber qué se entiende por Heurística.

definición de Heurística:

Heurística o heurética, o "ars inveniendi", era el nombre de una ciencia bastante mal definida y que se relacionaba tanto a la lógica, como a la filosofía o a la sicología. Se exponían con frecuencia las líneas generales, pero rara vez sus detalles. En nuestros días está prácticamente olvidada. Tenía por objeto el estudio de las reglas y de los métodos del descubrimiento y de la invención. Se pueden encontrar algunos esbozos de este estudio entre los comentadores de Euclides; un párrafo de Pappus es particularmente interesante sobre este tema. Los ensayos más conocidos sobre la construcción de un sistema heurístico son debidos a Descartes y Leibniz, ambos filósofos y matemáticos célebres. Igualmente se debe a Bernard Bolzano una exposición sobre heurística detallada y notable.

La palabra heurística proviene de la raiz griega **heuriskein** la cual significa literalmente "descubrir". Según el diccionario "Webster's New Collegiate" (1956), heurística es un adjetivo que significa "sirviendo para descubrir".

En la actualidad la heurística trata de comprender el método que conduce a la solución de problemas, en particular las operaciones mentales típicamente útiles en este proceso. Son diversas sus fuentes de información y no debe descuidarse ninguna. En el estudio de la Heurística debe tenerse en cuenta el trasfondo tanto lógico, como sicológico; pero debe apegarse más a la experiencia objetiva. Una experiencia que resulta a la vez de la solución de problemas y de la observación de los métodos de solución de la gente. Esto último es la base sobre la cual se construye la heurística.

Para fines de la IA, la heurística es la técnica que se encarga de mejorar la eficiencia de un algoritmo de búsqueda bajo ciertas condiciones. Otro enfoque dentro de la IA describe a la heurística como una regla empírica, estrategia, método o truco usado para mejorar la eficiencia de un sistema que trata de encontrar soluciones a problemas complejos.

La idea más clara de la heurística es sin lugar a dudas entenderla como **búsqueda con conocimiento**, que se refiere precisamente a la búsqueda de una solución a un problema específico.

1.3 UBICACION DE LA HEURISTICA DENTRO DE LA IA

Se ha mencionado que la heurística es una parte de la IA, en realidad la IA es dividida de acuerdo al criterio de algunos autores, llamando a estas partes, enfoques o elementos de la IA de la cual la heurística es uno de ellos. De esta forma la IA está dividida en tres enfoques principalmente: redes artificiales, evolución artificial y programación heurística.

Una red consiste en un gran número de elementos interconectados de algún modo. Una red artificial consiste esencialmente en lo mismo pero simulado con ayuda de la computadora. A menudo cada elemento de la red es una neurona artificial (célula nerviosa). Una ventaja de este enfoque es que la red es generalmente adaptable; esto significa que puede "aprender" por la experiencia. Los investigadores de este enfoque parten de la premisa de que la inteligencia natural se basa en la red de neuronas del cerebro. Y hasta ahora la mejor red artificial lo único que ha hecho es "aprender" a reconocer simples patrones visuales y auditivos. Una dificultad presentada con este enfoque es que no existe la posibilidad por el momento de crear una red artificial tan grande como la red de neuronas del cerebro humano debido a que, éste cuenta con aproximadamente 10 elevado a la 10 número de neuronas. Por si esto fuera poco, en nuestros días los fisiólogos de neuronas están bastante lejos de explicar cómo trabajan las neuronas y cómo están interconectadas.

En el enfoque de evolución artificial, la computadora trata de simular el proceso de evolución natural, consistente en la mutación y selección, por medio de un sistema; algunos sistemas artificiales han logrado resolver ecuaciones muy simples, utilizando éste enfoque.

Los investigadores que están de acuerdo con este enfoque, se basan en la teoría de que la inteligencia humana se ha desarrollado a través de un proceso de mutación y selección natural. La desventaja que se tiene con este enfoque es que la evolución natural aún no esta entendida completamente. considerando además, que la evolución artificial es práctica solo si ésta puede ir más lejos y más rápido que la misma evolución natural.

Finalmente el enfoque de la Programación Heurística, es precisamente el que corresponde a este estudio y el cual se explica con detalle en capítulos siguientes.

La Programación Heurística tiene dos principales propósitos:

1.- Entender la inteligencia humana (natural) y,

2.- Construir programas y máquinas que adquieran conocimiento y resuelvan intelectualmente problemas difíciles.

Un segundo criterio de clasificación de la IA se debe al modelo creado por Nilsson (1982). Su criterio de clasificación es conocido como el modelo cebolla (ver figura 1.1); y consiste en un anillo interno que encierra los elementos básicos y un anillo externo que considera las áreas de aplicación de dichos elementos.

figura 1.1

1.3.1 ELEMENTOS BASICOS DE LA IA

BUSQUEDA HEURISTICA

Muchos de los primeros trabajos de la IA, fueron enfocados a la elaboración de programas que pudieran "buscar" soluciones a problemas. Tomando en cuenta que al momento de hacer una decisión, la situación es cambiada abriendose una gama de nuevas oportunidades para más decisiones. Hay siempre un camino o rama por donde ir en la búsqueda de una solución. De hecho una de las maneras tradicionales de representar un problema en IA es por medio de árboles (ver figura 1.2).

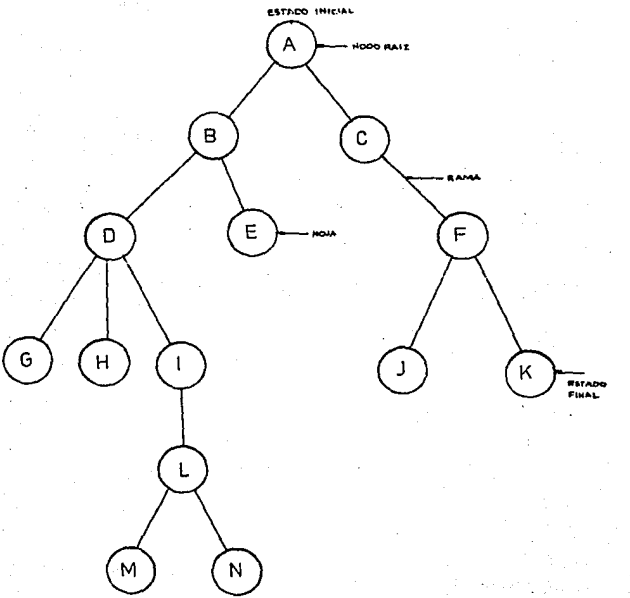


figura 1.2

Un árbol es un conjunto de nodos que están relacionados por medio de líneas. A cada una de las líneas se les llama **ramas**, y al nodo inicial **raíz**, como se observa en la figura 1.2. Una búsqueda en un árbol es un recorrido a través de sus ramas y nodos hasta encontrar un estado que cumpla una condición, que es precisamente la solución que se busca. La búsqueda comienza desde el nodo raíz, continuando por las ramas hacia abajo; este proceso se repite siempre que una decisión es hecha; al recorrido que se hace en un árbol en una búsqueda se le llama **trayectoria**. Sin embargo cuando los árboles son muy grandes las busquedas se vuelven lentas y tediosas, por lo que es necesario optimizar dichas búsquedas.

Inicialmente, los métodos de búsqueda utilizados no eran muy claros, razón por la cual hoy en día son conocidos como "ocultos". Estos métodos aseguraban que la misma trayectoria de solución no seria intentada más que una vez. Sin embargo para problemas más complejos que juegos simples y rompecabezas ese enfoque resultaba inadecuado. Luego entonces, reglas de thumb (traducido literalmente como reglas del pulgar, y que en realidad quiere expresar que se trata de reglas empíricas) referidas como **heurísticas**, tuvieron que ser añadidas para la selección de una probable rama en la trayectoria de solución, lo que reduce enormemente la búsqueda.

Un ejemplo de una heurística simple para ayudar a seleccionar un camino de regreso cuando se maneja al atardecer del puerto de Veracruz a la ciudad de México es: "vaya por donde se pone el sol". Definitivamente esto no produce instantáneamente la mejor trayectoria pero puede ayudar a encontrarla, estas reglas en realidad guían la búsqueda y la ventaja es que la reducen notablemente.

REPRESENTACION DE CONOCIMIENTO

Inicialmente, los investigadores de la IA descubrieron que el comportamiento inteligente, no es tanto debido a métodos de razonamiento, como al conocimiento con que se cuenta para razonar con él. Esto es, cuando el conocimiento sustancial ha sido adquirido para aclarar un problema, algunos métodos son necesarios para lograr un modelo que refleje eficientemente éste conocimiento, tomando en cuenta que el modelo debe estar expresado en una forma accesible para que pueda ser leido. El resultado de tanto énfasis en el conocimiento es precisamente la representación del conocimiento que es hoy por hoy, una de las más activas áreas de la IA.

SENTIDO COMUN RAZONAMIENTO Y LOGICA

Los investigadores de la IA encontraron que el sentido común (virtualmente dado por hecho en los humanos) es lo más difícil de representar como modelo en una computadora. Fué finalmente

concluido que el sentido común es un bajo nivel de razonamiento, basado en un cúmulo de experiencia. En la adquisición del sentido común aprendimos a esperar que, por ejemplo cuando hay una gota en un cristal, ésta resbalará hacia abajo y en general a anticipar cosas en los eventos cotidianos. Cómo representar el sentido común en una computadora, es una cuestión que es poco probable que se resuelva pronto.

Otra área que es muy importante en la IA, es la lógica. En esta área surgen dos importantes cuestiones (1) ¿Cómo deducir algo de un conjunto de hechos? y (2) ¿Cómo probar una conclusión a partir de un conjunto de premisas? La lógica computacional fué una de las primeras esperanzas doradas de la IA, para encontrar un método universal de solución de problemas. Sin embargo, la prueba de convergencia de la solución es muy diffcil con problemas complejos, con lo cual el interés en la lógica ha disminuido. La lógica está ahora renaciendo con un nuevo enfoque basado en nuevas formulaciones y combinándola con el uso de la heurfstica para guiar la búsqueda de la solución.

LENGUAJES DE IA Y HERRAMIENTAS

En la ciencia de la computación, han sido desarrollados lenguajes de alto nivel muy especificos de acuerdo a un propósito en alguna área del conocimiento. Esto también es válido para la IA. Actualmente, LISP y PROLOG son los principales lenguajes de programación para la IA. A la fecha, LISP (List Processing Language) ha sido el lenguaje utilizado para desarrollar paquetes como sistemas expertos y ayudas básicas de programación, así como herramienta de software.

Por otro lado, PROLOG (Programming Logic), iniciado y desarrollado ampliamente en Europa, ha sido utilizado para problemas de IA, tales como inferencias y deducciones.

1.3.2 ELEMENTOS DE APLICACION DE LA IA

PROCESAMIENTO DE LENGUAJE NATURAL (PLN)

PLN se refiere a que una computadora pueda entender, procesar o interpretar el lenguaje natural en forma de texto o en forma hablada. Un detallado informe de PLN es dado en [Geva 1983].

VISION POR COMPUTADORA

La visión por computadora fundamentalmente trata de lograr que la computadora pueda "ver", identificar o entender qué es lo que "ve", localizar qué es lo que está buscando, etc. Un estudio detallado sobre este tópico es dado en una buena forma

en [Geva 1982B].

SISTEMAS EXPERTOS

Sistemas expertos es quiza el tópico más tratado hoy por hoy dentro de la IA. Su finalidad es, cómo hacer que una computadora actúe como todo un experto humano dentro de un campo del conocimiento. Por ejemplo, ¿ Cómo obtener un diagnóstico médico por medio de una computadora ?, o ¿ un diseño arquitectónico ?. Un estudio muy completo sobre este tema es encontrado en [Geva 1982A].

SOLUCION DE PROBLEMAS Y PLANEACION

Existen muchos problemas para los cuales no hay expertos, y menos programas de computadora para solucionarlos. Sin embargo, hay sistemas de planeación que tienen que ver más con las técnicas de solución que con el conocimiento. Este trabajo (en el capítulo 2) trata el enfoque de solución de problemas y planeación.

1.4 ANTECEDENTES E HISTORIA DE LA IA Y LA HEURISTICA

1.4.1. ANTECEDENTES

Los primeros éxitos de la IA consistierón en hacer las operaciones elementales de la aritmética con mayor rapidez que la mente humana. Las primeras máquinas capaces de ello, comenzarón a construirse en la segunda mitad del siglo pasado y fueron utilizadas por las grandes empresas para hacer sus cálculos contables. A principios de este siglo comienzan a utilizarse máquinas de tarjetas perforadas ("punching-card machines") que permitían hacer operaciones más complicadas, como cálculos estadísticos, selección de personal, etc..

En 1925 se construye, bajo la dirección de Vanevar Bush, en el MIT (Massachusetts Institute of Tecnology), la primera máquina capaz de resolver ecuaciones diferenciales. En 1942, gracias a las revolucionarias concepciones de Norbert Weiner, se utilizan máquinas muy poderosas para dirigir la puntería de la artillería antiaérea. Fue gracias a estas máquinas que los barcos de guerra estadounidenses pudieron anular los tremendos ataques de los kamikase (pilotos suicidas japoneses). En 1944, el profesor Aiken, de la Universidad de Harvard puso en funcionamiento una máquina que contaba con "memoria", cualidad que le permitía resolver una amplia gama de problemas.

A partir de esta máquina los progresos son vertiginosos. Pero las primeras máquinas eran enormes, porque no existia una teoría matemática general para su construcción. Para construirlas, había que proceder por tanteos, utilizando la

experiencia previa en construcciones anteriores y resolviendo problemas parciales conforme se iban presentando. La mayor dificultad consistía en que los circuitos eléctricos necesarios para construirlas eran combinaciones de circuitos en serie y en paralelo, y no había una teoría matemática de estos circuitos. Por ello, estas máquinas contaban con kilometros de alambre.

En 1936, Shannon descubre que la teoría general de los circuitos en serie y en paralelo es la parte más elemental de la lógica matemática. Más o menos por la misma época, en forma independiente, el soviético Chostanov y el japonés Nakamura descubren lo mismo. Pero estos descubrimientos no comienzan a utilizarse sistemáticamente, sino a comienzos de la década de los cincuentas. Rápidamente la teoría general se desarrolla y perfecciona, y se transforma en lo que hoy se conoce como la teoría del diseño lógico de circuitos.

Con todo este auge, el optimismo creció en forma por demás fuera de la realidad como se comprobaría años más tarde. Y es en esta época precisamente cuando surge la IA como área de la computación, y se comienzan a hacer los primeros trabajos de investigación al respecto.

En 1956, diez científicos acordarón en una conferencia en el Dartmouth College los campos de la IA que tendrían auge en un futuro. Las predicciones hechas por estos científicos, fueron de que el hombre se dedicaría a actividades recreativas y artísticas y las computadoras se encargarian del trabajo. Curiosamente en 1981, en la conferencia internacional de IA en Vancouver, Canada, un grupo de cinco de los mismos científicos volvierón a expresar la misma (demasiado) optimista predicción, a veinticinco años de haberse formulado la primera.

En 1956, el comportamiento inteligente fué primeramente simulado, mediante cortas técnicas de razonamiento y se vió que gente brillante podía descubrir técnicas ad hoc, para la realización de programas computacionales.

En el inicio la mayor de las actividades fué la traducción por computadora, basándose en el uso de un diccionario bilingüe y un conocimiento de gramática. Sin embargo, este enfoque fracasó en forma por demás rotunda, una vez que palabras, oraciones y frases provocarón una serie de ambigüedades que hacian confusa la traducción o cambiaban totalmente el contexto general de lo que se quería expresar. De hecho hasta la actualidad es que se ha vuelto a retomar el problema con mejores resultados.

Weizenbaum en 1966 en el MIT (Massachusetts Institute of Tecnology) diseñó un programa que entendía el lenguaje natural humano y que simulaba una terapia sicológica (ver [Weiz 1966]). El programa (llamado ELIZA) entraba en interacción con los usuarios por medio de preguntas que selecciona de su memoria, sin embargo cuando el programa no reconocía alguna estructura u

oración, contestaba con una frase como: "por favor continúa" o "ya veo". Aunque Weizenbaum escribio en parte el programa para mostrar qué ridículo resultaba la idea de que una máquina entendiera el lenguaje natural, el programa llego a ser bastante popular y hoy en día se usan sus técnicas para algunos programas comerciales en esta área.

En 1961, Slagle en el MIT desarrolló un programa de tipo heurfstico para realizar integración simbólica. Este programa dio la pauta para la realización de una afortunada serie de programas matemáticos de tipo simbólico que culminarón con MACSYMA.

Una de las primeras áreas en donde se aplicó la heurfstica fué a los juegos. Samuel's en 1963 trabajando para IBM, "enseño" a la computadora a jugar, lo que constituyó uno de los primeros logros de las técnicas heurfsticas.

Otra de las primeras áreas trabajadas en los primeros años, fue la solución a laberintos y rompecabezas, desarrollando técnicas muy especfficas, que se basan principalmente en dos premisas: (1) búsqueda y, (2) reducir la dificultad del problema, dividiendolo en subproblemas más fáciles de resolver.

También, la lógica matemática, constituye una área importante, cuyo ffn era desarrollar un resolvedor de problemas de propósito general. Dentro de esta área Green en 1969 desarrolló un sistema llamado QA3, que resolvia problemas simples como: movimiento de un robot, rompecabezas y problemas de qufmica. Sin embargo, en ocasiones encontrar la solución provocaba todo un número de pasos intermedios que hacian lentisima la búsqueda de la solución, en este caso el problema fundamental es la explosión combinatoria.

Con otro enfoque Newell en 1960 ideá el GPS (General Problem Solver). La premisa es separar los mecanismos de solución del conocimiento, para así clasificar los estados intermedios antes de la meta final. Sin embargo, es precisamente la clasificación de los estados, lo que presenta grados de dificultad grandes y en ocasiones más diffciles que el problema a resolver.

Definitivamente las técnicas de búsqueda presentan en esta época, graves dificultades para su implantación, más de las esperadas. Para 1970 fuera de algunos juegos y problemas sencillos no habfa gran avance. En 1971 el gobierno británico pidió que se realizará un estudio sobre el campo de la IA, el cual se encargó de elaborar Sir Lightill de la Universidad de Cambridge. En 1972 Lightill acabó su reporte, que trajo como conclusión final que en ninguna de las áreas de la IA se había logrado el avance suficiente como para corresponder a los pronósticos hechos al final de la década de los cincuentas (ver [Ligh 1972]).

Al final de los sesentas se llegó a las siguientes conclusiones:

- La heurística es necesaria para evitar que las búsquedas crezcan combinatoriamente.

- Difícilmente se podrán representar características humanas en la computadora, tales como, sentido común e intuición.

1.4.2 LA DECADA DE LOS SETENTAS

A principios de esta década y después de haberse llegado a ciertos resultados, surgen nuevas técnicas de representación de conocimiento. En cuanto a las técnicas de búsqueda heurística, estas comienzan a ser pulidas y corregidas para hacerlas más eficientes y se comienza a aplicar los sistemas a campos como la medicina, la química y la electrónica, con lo cual finalmente se desarrollan importantes herramientas de apoyo para estas áreas.

SHRDLU un programa de lenguaje natural diseñado por Terry Winograd en 1972 en el MIT, contaba con una interface artificial de "bloques de palabras" y fué el primer programa que logró integrar el lenguaje natural combinándolo con un muy buen análisis semántico y sintáctico, y con un cuerpo de palabras que conformaban un conocimiento.

De 1971 a 1976 se desarrollo ARPA después de cinco años de investigaciones. HEARSAY II en la Universidad Carnie Mellon fué todo un éxito, ya que podía entender oraciones, con un 90 % de seguridad de forma continua basado en un vocabulario de 1000 palabras. Realmente la arquitectura del sistema con múltiples fuentes de conocimiento diseñado para el HEARSAY II, ha sido utilizado a menudo para otras aplicaciones dentro de la IA. Un sistema compilado de arquitectura para redes de trabajo llamado HARPY, el cual maneja el mismo vocabulario de HEARSAY II proporciona una seguridad de 95 % en su interacción con el usuario, lo que constituye una ganancia de 5 % más en comparación con su antecesor.

En los tres primeros años de los setentas otro fructífero evento alentó la investigación en el desarrollo de Sistemas Expertos (SE), se trata de DENDRAL, una variedad de programas prototipo, que se les dió el nombre de Sistemas Expertos, diseñados para capturar y utilizar la experiencia de un experto humano en algún campo del conocimiento. MYCIN, un sistema experto de diagnóstico médico y consulta, ha sido uno de los más utilizados y fue desarrollado por Shortliffe en 1976 en la Universidad de Stanford.

Así, las investigaciones y trabajos en los años setentas dejarón el desarrollo de herramientas y técnicas básicas en la construcción de Sistemas Expertos y abrían las puertas para el

desarrollo de sistemas más complejos en un futuro. El énfasis en el conocimiento, tan esencial para la inteligencia, creó un subcampo dentro de la representación del conocimiento que es conocido como "Ingenieria del Conocimiento" importante en el diseño de un sistema experto.

1.4.3 DE 1980 AL PRESENTE

La década de los setentas sentó las bases para los trabajos realizados en los ochentas, y se ha visto como es que los sistemas expertos se han perfeccionado y actualmente alcanzan bastantes áreas del conocimiento. Han sido fabricados "chips" que tienen implementado el principio de resolución y unificación. Pero la novedad en los ochentas lo constituye no sólo la investigación sino la penetración tecnológica. El hecho es que la IA se ha comercializado y diversas compañías se han dedicado a su explotación. En el área de los sistemas expertos, DEC reporta que el sistema RI desarrollado para la configuración de una computadora VAX, ahorra a la compañía veinte millones de dólares anuales. Computadoras personales han sido fabricadas especialmente para trabajar el lenguaje LISP y varios compiladores han sido puestos a la venta. La NASA y la NBS (National Bureau of Standards) han realizado una serie de reportes técnicos con los avances en el área. En 1982 los japoneses anuncian que en un plazo de diez años y con un presupuesto de un billón y medio de dólares crearán lo que será la quinta generación de computadoras. Las características de estas máquinas serán contar con: (1) Interfaces inteligentes, (2) Manejador de Base de Conocimiento y, (3) Resolvedor automático de problemas y capacidad de inferencia. Todas esas capacidades serán desarrolladas con el uso de técnicas de IA. La máquina en sí, no es visualizada como una máquina de Von Neumann, teniendo proceso en paralelo y con una capacidad de un billón de inferencias lógicas por segundo.

Los japoneses están considerando el lenguaje PROLOG desarrollado en Europa, para realizar sus aplicaciones y la arquitectura de sus máquinas. Usando PROLOG, para el desarrollo de sistemas resolvedores de problemas (con guía heurística) el campo de solución de problemas, incluyendo los más complejos ha resurgido de una manera sorprendente y se esperan grandes logros en esta decada. Con el advenimiento de la quinta generación por parte de los japoneses, algunos paises Europeos como Francia e Inglaterra en conjunción con los Estados Unidos, están poniendo especial énfasis en la investigación y desarrollo de la IA. Si se desea conocer algo más al respecto, puede verse [Warr 1982].

Se puede concluir que las herramientas de la IA y los sistemas han llegado a estar disponibles en la actualidad, y las técnicas heurísticas están suficientemente perfeccionadas ahora para algunas aplicaciones. Hoy en día la IA se ha convertido en una área de suma importancia en la cual se están invirtiendo grandes recursos en su desarrollo e investigación.

No es de extrañarse que en un futuro quiza no muy lejano, se conviertan en realidad las predicciones hechas a fines de los años cincuentas.

NOTAS DEL CAPITULO 1

[1] Einstein trató de encontrar una fórmula general que permitiese describir la estructura de los campos gravitatorios y electromagnéticos, pero sus esfuerzos fracasaron. Sólo en los últimos años, después de un arduo trabajo en equipo, se ha podido avanzar, aunque aun no en forma definitiva, en la solución del problema.

[2] La conjetura de Goldbach es la siguiente: Todo número par es igual a la suma de dos números primos. La mayoría de los matemáticos están convencidos de que esta conjetura, hecha por un oscuro profesor de matemáticas de comienzos de siglo XVIII, es cierta. Sin embargo, hasta el momento nadie ha podido demostrarla.

BIBLIOGRAFIA DEL CAPITULO 1

LIBROS

[Poly 1965] COMO PLANTEAR Y RESOLVER PROBLEMAS POLYA, G.
TRADUCCION POR: ZUBAZAGOITIA, J.
1965 EDITORIAL TRILLAS, MEXICO
DECIMOTERCERA REIMPRESION 1986
215 pp.

[Rich 1983] ARTIFICIAL INTELLIGENCE RICH, ELAINE
INTERNATIONAL STUDENT EDITION
1983 McGRAW HILL BOOK CO. SINGAPURE
436 pp.

[Slag 1971] ARTIFICIAL INTELLIGENCE: THE HEURISTIC
PROGRAMMING APPROACH SLAGLE, JAMES R.
1971 McGRAW HILL BOOK CO. NEW YORK
196 pp.

[Traj 1975] INTRODUCCION A LA TEORIA MATEMATICA DE LAS
COMPUTADORAS Y DE LA PROGRAMACION TRAJTENBROT, B.A
1975 SIGLO VEINTIUNO EDITORES MEXICO.

[Weiz 1978] LA FRONTERA ENTRE EL ORDENADOR Y LA MENTE
WEIZENBAUM, J.
1978 EDICIONES PIRAMIDE MADRID.

[Wins 1979] ARTIFICIAL INTELLIGENCE WINSTON, PATRICK
1979 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
444 pp.

REPORTES TECNICOS

[Geva 1982A] AN OVERVIEW OF COMPUTER VISION GEVARTER, W.B.
SEPTEMBER 1982
NATIONAL BUREAU OF STANDARDS WASHINGTON
55 pp.

[Geva 1982B] AN OVERVIEW OF EXPERT SYSTEMS GEVARTER, W.B.
OCTOBER 1982
NATIONAL BUREAU OF STANDARDS WASHINGTON
52 pp.

[Geva 1983] AN OVERVIEW OF COMPUTER-BASED NATURAL LANGUAGE
PROCESSING GEVARTER, W.B.
1983 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
(N.A.S.A.) TM WASHINGTON
60 pp.

[Geva 1984] AN OVERVIEW OF ARTIFICIAL INTELLIGENCE AND ROBOTICS
VOLUME 1 - ARTIFICIAL INTELLIGENCE
PART A - THE CORE INGREDIENTS
1984 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
(N.A.S.A.) WASHINGTON
67 pp.

[Ligh 1972] ARTIFICIAL INTELIGENCE: A GENERAL SURVEY
LIGHTHILL, J.
1972 SCIENTIFIC RESEARCH COUNCIL OF BRITAIN, SRC
72-72

ARTICULOS

[Warr 1982] A VIEW OF THE FIFTH GENERATION AND HIS IMPACT
WARREN, D.H.D.
1982 THE AI MAGAZINE, VOL. III No. 4
6 pp. 34-39

[Weiz 1966] ELIZA - A COMPUTER PROGRAM FOR THE STUDY OF NATURAL LANGUAGE COMUNICATION BETWEEN MAN AND MACHINE
WEIZENBAUM, J.
1966 CACM 9
10 pp. 36-45

CAPITULO 2

FUNDAMENTACION DE LA PROGRAMACION HEURISTICA

2.1 NATURALEZA DE LA HEURISTICA

2.1.1 PRELIMINARES

En el capítulo 1, se mencionó que los problemas de tipo lógico-matemático, son de fundamental importancia. Estos problemas son de tres tipos diferentes: problemas mecánicos, cuya solución exige un pequeño número de operaciones; problemas mecánicos, cuya solución exige un número muy grande de operaciones; y problemas no mecánicos.

Los primeros problemas de IA que podían resolver las computadoras eran del primer tipo. Desde luego, la expresión "pequeño número de operaciones" debe tomarse en relación a su significación cibernética (por ejemplo, seiscientas operaciones por minuto eran un número aceptable para las computadoras de la primera generación). Pronto se descubrió que problemas mecánicos aparentemente simples exigían, para ser resueltos, una pavorosa cantidad de operaciones. El juego de damas, tan simple de aprender por un humano, requiere, para ser analizado en todos los movimientos posibles de las fichas, de 10 a la 40 número de operaciones. En 1963, la capacidad para realizar hazaña semejante, con la más poderosa computadora, se habría demorado más de quinientos mil años en hacerlas. Y si de las damas se pasa al ajedrez, el número de operaciones que deben hacerse para analizar todos los movimientos posibles de las piezas (y, en consecuencia, para hacer una máquina invencible) es 35 elevado a la 100 número de operaciones. Las más poderosas computadoras actuales aún no cuentan con tal capacidad.

Por eso, para lograr que una computadora pueda jugar bien damas o ajedrez, es necesario utilizar métodos heurísticos. Estos son métodos que permiten cortar camino, aprovechando la experiencia obtenida (acumulada en la memoria de la máquina). Pero aunque estos métodos han permitido programar computadoras prácticamente invencibles en el juego de damas y capaces de enfrentarse a un buen jugador de ajedrez, no son absolutamente confiables. Si un jugador, por rara casualidad, hace un movimiento excepcional, el método heurístico puede fallar y desconcertar a la máquina, haciendola perder. Otra falla la constituye, la exagerada lentitud de respuesta cuando el juego toma altos niveles.

En cuanto al tercer tipo de problemas, son los más difíciles de resolver. Un problema es no mecánico cuando no hay ni puede haber un método fijo, siempre igual, para su solución. La solución sólo puede encontrarse mediante el talento. Naturalmente, los problemas más profundos de la lógica y de las matemáticas son de carácter no mecánico. ¿ Cómo han hecho los investigadores para resolver este tipo de problemas ? En estos

casos precisamente, los métodos heurísticos son de gran utilidad, pero esta utilidad es limitada por el hecho de que no hay reglas fijas para hacer las operaciones que permitan resolver el problema. La mejor manera de proceder, es tratar de descubrir casos particulares que sí pueden resolverse con reglas fijas. Por ejemplo, no hay reglas fijas para resolver problemas de matemáticas cuando las fórmulas tienen dos o más variables y, por lo menos, un cuantificador existencial. Pero sí las hay cuando las fórmulas sólo tienen una variable [1]. Por eso, cuando se presenta un problema lógico-matemático, se trata de expresarlo mediante fórmulas de una sola variable. Asimismo, si las fórmulas tienen ciertas propiedades especiales, pueden resolverse de manera mecánica aunque tengan varias variables [2]. De este modo se ha llegado a resolver algunos problemas que, hasta hace poco, parecían insolubles como, por ejemplo, el problema de los cuatro colores [3].

El impresionante desarrollo a últimas fechas de los sistemas nuevos en IA (por ejemplo Sistemas Expertos), ha dado origen a que se considere más intensamente el campo de la heurística. La tendencia es considerar: razonamiento, juicios, deducciones, inferencias, intuición y sentido común (para dotar a los programas con cualidades inherentes al humano).

La pregunta central que se establece para estas consideraciones es: ¿ Cuál es la fuente de poder de la heurística ? y aunada a ésta, la pregunta que constituye la gran incognita de los procesos mentales del razonamiento: ¿ Cómo reglas heurísticas nuevas pueden ser generadas por un programa ?. Precisamente lo que se pretende en este capítulo es tratar temas que ayuden a contestar las dos cuestiones anteriores, con lo que se podrá comprender la naturaleza de la heurística.

2.1.2 CUAL ES LA FUENTE DE PODER DE LA HEURISTICA

En el capítulo 1, cuando se hablaba de qué era la heurística, se presentó un ejemplo de una guía para encontrar el camino de regreso a la ciudad de México, cuando se venía de Veracruz. La regla "vaya por donde se pone el sol" no daba la trayectoria óptima, pero podía ayudar a encontrarla. En este tipo de situaciones cotidianas existen eventos, que de una u otra forma ayudan a que se establezca un "hecho" a partir de ellos, a estos eventos se les llama indicios de progreso.

Para ilustrar este concepto supóngase que una persona está jugando tenis, y se espera que por el esfuerzo físico sude en abundancia, un claro indicio de progreso, lo constituiría el hecho de que comience a sudar. De forma análoga en la resolución de problemas sucede lo mismo. Cuando se busca una solución un indicio de progreso constituye un avance hacia ella. Ahora bien, el enunciado anterior se puede expresar de la siguiente manera:

- Cuando una persona juega tenis con frecuencia suda
- Ahora está sudando
- Por lo tanto es probable que esté jugando tenis

Sin la palabra "probable" la conclusión sería completamente erronea, con la palabra "probable" la conclusión es razonable, sin ser una demostración concluyente; es solo una indicación, una sugerencia heurística. Esto último hay que enfatizarlo ya que si es un error tomar la conclusión como verdad absoluta, es aún más grave no tomarla en cuenta en absoluto.

Esta estructura de razonamiento se establece considerando premisas a las primeras proposiciones y a la tercera conclusión. Al conjunto del razonamiento se le puede denominar silogismo heurístico. El razonamiento sugerido por el ejemplo puede expresarse de la siguiente manera:

- Si A se da, con frecuencia ocurre B
- Ahora B se está dando
- Por consiguiente A es factible

En terminos computacionales puede verse asi:

- Si A, entonces B
- B cierto
- A más factible

Lo anterior constituye el origen del poder de la heurística, lo que en realidad la hace útil en los mecanismos de búsqueda de una solución. A continuación se exponen los principios de tipo formal en la creación de una heurística.

La fuente de poder de la heurística es considerada como un tipo de continuidad bidimensional. Es decir, si una heurística H fué (o ha sido usada) en una situación S, entonces es probable que una heurística similar a H, pueda ser útil en situaciones similares a S. En otras palabras si se tiene una función que trabaja con propiedades de acción y situación, esa función será continua en ambas variables.

Un útil ejercicio es considerar los valores de la gráfica de propiedades de una función para una acción fija, variandola sobre las situaciones en que debe ser aplicada. El lenguaje de las gráficas de funciones representa una atractiva metáfora, en la cual se puede discutir cada proceso como la especialización de una heurística, el uso de múltiples heurísticas, y la medición de los atributos durante la ejecución de la heurística.

Hay tres condiciones básicas que debe cumplir un algoritmo con búsqueda heurística en lo que se refiere a la fuente de su poder:

1) Continuidad

2) Estabilidad

3) Observalidad

Continuidad: Si el ambiente cambia de forma abrupta, las heurísticas podrían no ser válidas.

Estabilidad: Si los cambios son continuos y rápidos, las heurísticas deben tener un corto lapso de vida, antes de que se conviertan en obsoletas.

Observalidad: Si los datos no pueden ser reunidos, las heurísticas no pueden ser formadas y mucho menos evaluadas.

2.1.3 COMO NUEVAS HEURISTICAS PUEDEN SER GENERADAS

Estudios realizados en el campo de la IA y resultados obtenidos de la experiencia de observar el comportamiento de programas de este tipo, han llevado a los investigadores a establecer que las nuevas heurísticas se generan a partir de tres fuentes principales. Estas constituyen los métodos principales para la generación de las heurísticas: **Especialización, Generalización y Analogía.**

Especialización: Se aplica a la existencia de heurísticas generales. El propósito es producir heurísticas más especializadas y más eficientes para cada caso en particular; habiendo partido de la observación de hechos generales. La compilación y la programación estructurada son un claro ejemplo de este proceso. Es a menudo la forma utilizada para adaptar, encuadrar, probar y templar los datos observados dentro de un programa.

Generalización: Se aplica a partir de heurísticas especializadas, con la observación de resultados y datos. La generalización se establece generando una nueva heurística que constituirá un pronóstico de lo que va a ocurrir. Es como una demostración utilizando la inducción matemática.

Analogía: La analogía parte del principio de la fuente de poder de la heurística explicado en la sección 2.1.2. Se basa en la premisa de que si una heurística fué generada para un caso X, y se presenta un caso Y similar a X. Entonces una nueva heurística puede ser generada a partir del conocimiento de la anterior. Es necesario hacer los cambios y ajustes correspondientes para la nueva situación y guiar de forma eficiente la búsqueda.

Estos tres métodos han sido aplicados en diversas situaciones en programas de IA, llegandonse a la conclusión de que la analogía es la más eficiente pero la más difícil de implantar. Mientras que la especialización y la generalización dependen de los procesos y tipos de problemas que se vayan a presentar en una búsqueda.

2.1.4 OTROS ENFOQUES HEURISTICOS

En la sección 2.1.3 se exponen los métodos para generar nuevas heurísticas y habiéndose conocido el origen de su potencialidad, se tiene un panorama bastante claro al respecto de cómo es que podría trabajar una función de este tipo. Más sin embargo, existen otros enfoques que igualmente que los anteriores basan su existencia y directrices en preguntas.

Un primer enfoque se basa en la pregunta: ¿Cuál es el impacto de una heurística individual en una búsqueda ?, y un segundo señala: ¿ Cómo varias heurísticas podrían ser combinadas ?. Para ambos métodos el desarrollo aún no es suficiente, y se carece de trabajos al respecto, por lo que la máxima de Feigenbaum es válida "En el conocimiento la potencialidad de una función miente". Debido a que profundizar en estos enfoques es un tema de investigación que aparta del objetivo a este trabajo, si se desea profundizar puede verse [Davi 1982] y [Gasc 1977].

2.2 OBJETIVOS Y ALCANCES DE LA HEURISTICA

El objetivo principal de la heurística es el de comprender el método que conduce a la solución de problemas. En particular las operaciones mentales que típicamente son útiles en este proceso. Para lograr este objetivo se cuenta con dos enfoques: el enfoque sicológico y el enfoque lógico. El enfoque sicológico busca la construcción de un modelo de comportamiento, para tratar de comprender los mecanismos que se esconden en el proceso del pensamiento. Por otro lado el lógico busca aplicar el razonamiento que utiliza la gente en la solución de problemas incluyendo los cotidianos. Es decir, la aplicación de las reglas empíricas (en nuestro país un conjunto de sabiduría empírica lo forma sin lugar a dudas, los dichos y refranes populares).

Dentro de la computación el objetivo de la heurística es el de dotar de conocimiento a los programas de búsqueda, con el fin de guiar dicha búsqueda para encontrar la solución a un problema. En otras palabras a la heurística se le puede ver como una especie de tip o sugerencia.

En cuanto a sus alcances, aún no se puede vislumbrar hasta donde puede llegar, porque falta mucho por conocer acerca de los mecanismos y operaciones mentales propias del pensamiento

humano. Por supuesto cuando se descubran nuevas reglas, se tratará de dotar a la computadora con ellas, hasta que llegue el dia en que la máquina tenga "inteligencia" propia.

En estos momentos surgen grandes interrogantes al respecto, tres son las principales cuestiones: (1) ¿ Hasta dónde llegarán las computadoras ?, (2) ¿ Serán capaces de hacer cosas que el hombre no puede hacer ?, (3) ¿ Puede el hombre construir una máquina que tenga mayor capacidad intelectual que él mismo ?

La respuesta a estas interrogantes en la actualidad es bastante simple. Cuando se trata de hacer operaciones mecánicas, las computadoras son mucho más poderosas que el ser humano. La razón es obvia. Una computadora puede hacer en un segundo un número fantastico de operaciones. El hombre, en cambio, por más inteligencia matemática que posea, apenas puede hacer una, siempre y cuando sea muy simple.

Si no fuera por las computadoras, la tecnolgía, que caracteriza nuestra vida moderna, no existiría. En este sentido, una computadora tiene una "inteligencia" muchísimo más poderosa que la inteligencia humana. Pero se trata de una superioridad puramente cuantitativa. El ser humano, puede en principio, hacer todo lo que hace una computadora (puesto que él mismo la ha programado), pero en la práctica no puede por falta de velocidad y resistencia.

Cuando se trata de resolver problemas no mecánicos, el ser humano es muy superior. Una computadora no puede resolver problemas para los que no ha sido programada; en cambio, el ser humano, tiene talento y puede resolver problemas de increíble dificultad. Nadie sabe cómo lo hace, pero es capaz de encontrar el camino adecuado a pesar de que nadie antes que él sabía ni podía saber cómo era este camino. Una computadora puede compararse a un auto deportivo muy veloz que avanza a una velocidad vertiginosa sobre una autopista; un gran matemático puede compararse a un explorador que, conforme avanza va forjando su propio camino (caminante no hay camino, se hace camino al andar).

Para el que esto escribe, la siguiente frase resume con propiedad el verdadero alcance de la heurística:

EL LIMITE DEL INTELECTO HUMANO, ES EL ALCANCE DE LA HEURISTICA

2.3 ESTRATEGIAS DE BUSQUEDA HEURISTICA

2.3.1 DEFINICIONES BASICAS

Para comprender qué es una estrategia de búsqueda de tipo heurístico es primordial establecer antes algunas definiciones:

Un problema P es una quintuple

$$P = (D, R, C, S, F)$$

Donde **D** es un conjunto de estados, conocido como **espacio del problema**; **R** es el conjunto parcial de las funciones con dominio en **D** e imagen en **D** llamado **operadores** ($R: D \rightarrow D$); **C** es una función parcial ($C: D \times D \rightarrow R$), llamada **costo de la función** (**R** es el conjunto de los reales positivos); **S** es un subconjunto de **D**, llamado el **conjunto de estados iniciales**; y **F** es un subconjunto de **D**, denominado **conjunto de estados finales**.

Una **estrategia Sp** es una función (posiblemente parcial) la cual para un estado dado en algún dominio de un problema, retorna secuencias de estados sobre el dominio del problema. En otras palabras una estrategia, es un función que actúa sobre un estado o conjunto de estados y retorna un valor que constituye un secuencia de estados sobre el dominio del problema. Ya establecidas estas definiciones se veran a continuación enfoques de solución de problemas y estrategias de búsqueda bien específicas, presentándose el algoritmo aplicado a un problema, para ejemplificar su funcionamiento.

2.3.2 ENFOQUES DE SOLUCION DE PROBLEMAS

Todos los problemas tienen aspectos comunes: una situación inicial, una meta (situación deseada) y operadores (procedimientos o funciones) que pueden ser usados para cambiar situaciones. En la solución de problemas, una estrategia es usada para intentar llegar a la meta. Este último concepto es ilustrado en la figura 2.1, donde se observa que la estrategia opera con los procedimientos para generar una secuencia de acciones (llamadas un **plan**) para transformar las condiciones iniciales en el estado objetivo o meta. Normalmente existen también restricciones (especificando las condiciones necesarias para que un procedimiento pueda ser aplicado) las cuales deben ser cumplidas para generar la solución.

En el proceso de generación de un plan, el resolvedor de problemas debe mantener la pista sobre las acciones que ya han sido ejecutadas y su impacto sobre el sistema. La figura 2.2 ilustra un esquema de un resolvedor de problemas. Se puede observar claramente la relación directa de los operadores y sus efectos sobre el sistema.

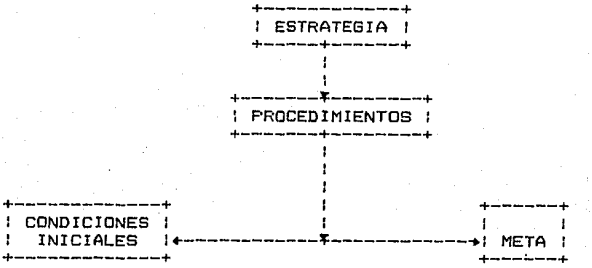


figura 2.1

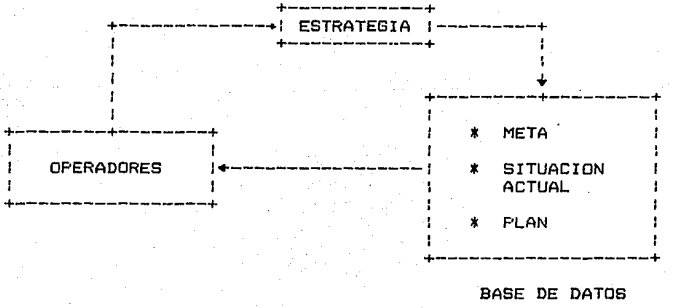


figura 2.2

REDUCCION DE PROBLEMAS

Una forma simple de resolver un problema es seguir la heurfstica: "divide y venceras". La reducción de problemas consiste en dividir el problema en subproblemas más fáciles de resolver y la solución de todos los subproblemas conforma la solución del problema original. Sin embargo, este enfoque a menudo falla si no se tiene un conocimiento específico del problema que se va a resolver.

REDUCCION DE DIFERENCIAS (Análisis "Means-Ends")

La reducción de diferencias fué introducida por el GPS (General Problem Solver). Este fué el primer programa en separar el método general de problemas del conocimiento especifico del problema.

El análisis "Means-Ends" determina primeramente la diferencia entre el estado inicial y el estado objetivo, para después seleccionar el operador que acorte más esa diferencia. En este punto se crea un estado intermedio producto de la aplicación del operador anterior. Ahora desde el estado intermedio se selecciona un nuevo operador que de la mínima diferencia con el estado objetivo. El proceso continua hasta que la diferencia es nula, consistiendo la solución del conjunto de operadores aplicados.

El enfoque de reducción de diferencias asume que la diferencia entre un estado actual y un estado deseado puede ser definida y que los operadores pueden ser clasificados de acuerdo a los tipos de diferencias que ellos puedan reducir.

TACTICAS MAS EFICIENTES PARA SOLUCION DE PROBLEMAS

Para mayor eficiencia en la solución de problemas es necesario consultar técnicas para guiar la búsqueda y poder usar adecuadamente el conocimiento inicial sobre el problema. Sacerdoti (ver [Sace 1979]) indica que la información relevante de la planeación puede ser aprendida durante el proceso de exploración y esto incluye:

a) El orden de relación entre acciones

b) Ligaduras jerárquicas entre acciones y varios niveles de abstracción

c) El propósito de las acciones en un plan

d) La dependencia entre objetos (o estados) que serán manipulados

Hay una buena cantidad de métodos para la solución de problemas entre los más importantes destacan: Planeación Jerárquica, Subplanes de Propósito Especial, Backtracking Relevante, Seudo-Reducción, y Regresión a la Meta. Si se desea conocerlos a detalle puede consultarse la obra de Sacerdoti ya referida.

2.3.3 ESTRATEGIAS DE BUSQUEDA BASICAS

Los procesos de búsqueda, desempeñan un papel fundamental en la solución de un problema, asi que una vez formulado el problema y habiendose escogido una representación del conocimiento de acuerdo a las características del mismo, es necesario atacar el problema con una estrategia de solución.

Un algoritmo de búsqueda debe contar tambien con una **estrategia de control**, la cual aplica reglas de manera repetida a descripciones de estado hasta que se produce una descripción de un estado final. Se lleva el registro de las reglas que se han aplicado de manera que pueda componerlas en la secuencia que representa la solución al problema.

Se tienen dos clases de estrategia de control **IRREVOCABLES Y TENTATIVAS**, en las irrevocables se selecciona una regla y se aplica sin prevenir una reconsideración posterior; y en las tentativas se selecciona una regla aplicable, pero previniendo que se pueda regresar a ese punto.

A continuación se presentan ejemplos de representación de problemas, así como estrategias para su resolución.

Un problema puede resolverse empezando en un **punto de partida o estado inicial** , y explorar los posibles caminos o alternativas que conduzcan a una **solución o estado final**. A esto último se le denomina **exploración de una gráfica de estados**, pues la manera más generalizada de representar un problema en IA es por medio de árboles, como ya se había visto en el capítulo 1.

Dos de los algoritmos de búsqueda más utilizados son el **Breadth-First Search o Búsqueda Horizontal** y el **Depth-First Search o Búsqueda en Profundidad.**

BUSQUEDA EN ARBOLES

Hay diversas formas de buscar en árboles, pero usualmente se da una gráfica de manera implfcita. Si se aplican operadores a un nodo dado, se pueden generar todos sus hijos; Se dice que se **expande** un nodo cuando se generan sus hijos.

CLASIFICACION DE NODOS

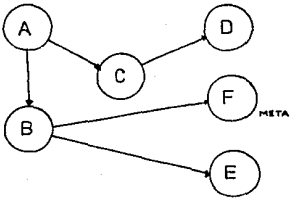
Los nodos de un árbol, se clasifican como **abiertos o cerrados**; Un nodo abierto aún no se ha examinado para posible expansión, en cambio un nodo cerrado ya se examinó, y tal vez no se expandió debido a dos razones: (1) era el nodo final u objetivo, o (2) era un estado absorbente (rama terminal o callejón sin salida).

Para los ejemplos que a continuación se presentaran, los nodos cerrados se representaran sombreados y los abiertos sin sombrear.

SOLUCION DE LABERINTOS

Una de las aplicaciones de los métodos de búsqueda horizontal y en profundidad, suele ser la solución a laberintos; que es precisamente la aplicación que aquí se verá como ejemplo.

Se sabe de antemano que un laberinto puede ser representado por una gráfica dirigida (ver figura 2.3).



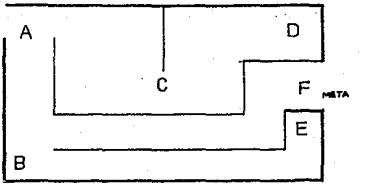


figura 2.3

Una vez que se ha visto el ejemplo, se trata de resolver el problema dando la gráfica de forma explícita, que ésta se conserve en memoria y proporcionar la información necesaria para que por medio de los algoritmos de búsqueda horizontal (BH) o búsqueda en profundidad (BP), se encuentre la trayectoria de solución.

2.3.4 BUSQUEDA HORIZONTAL (BH)

Esta búsqueda consiste en examinar los nodos del árbol nivel por nivel, es decir que no se examina un nodo hasta haberse examinado los del nivel anterior. Para programar ésta búsqueda es necesario utilizar dos listas (una que controle los nodos abiertos y otra los cerrados).

Una manera sencilla de programar ésta búsqueda es utilizar una matriz que simule un árbol, un vector auxiliar y dos listas. A continuación se presenta el algoritmo de ésta búsqueda y un ejemplo gráfico de como se llega a la solución del problema del laberinto planteado anteriormente (ver figura 2.4).

ALGORITMO PARA LA BH

1.- Poner el nodo inicial en abiertos.

2.- Si abiertos esta vacia, terminar con falla. En otro caso continuar.

3.- Tomar el primer nodo en abiertos y ponerlo en la lista de cerrados. Llamar n a éste nodo.

4.- Expander el nodo n generando todos sus sucesores;
- si no hay sucesores regresar a 2, sí si continuar
- Poner los sucesores al final de la lista de abiertos y marcar estos sucesores como descendientes de n.

5.- Si cualquiera de los sucesores es un nodo objetivo, terminar con la trayectoria de solución, la cual se obtiene avanzando en reversa hasta el nodo inicial. En otro caso regresar al punto 2.

EJEMPLIFICACION GRAFICA DE LA BUSQUEDA HORIZONTAL (BH) PARA EL PROBLEMA DEL LABERINTO

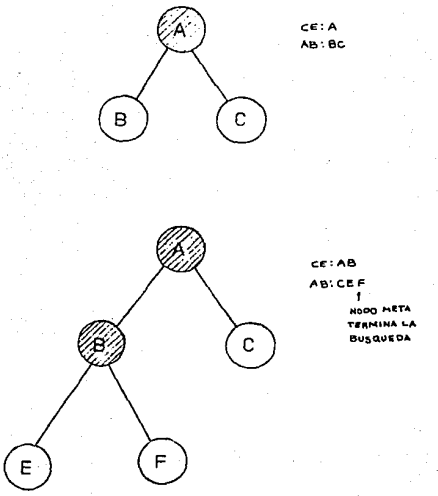


figura 2.4

2.3.5 BUSQUEDA EN PROFUNDIDAD (BP)

Esta búsqueda consiste en examinar los nodos del árbol e ir bajando a examinar otro nodo en el nivel que sigue, hasta que, o se haya agotado la cota de profundidad (cuando la hay) o el nodo es un nodo terminal. Cabe mencionar que en ésta búsqueda la lista de abiertos es en realidad un stack, por lo tanto para programar el algoritmo se necesitan una estructura de lista y otra de pila o "stack".

Igualmente que con la búsqueda anterior se presentan el algoritmo y el ejemplo gráfico (ver figura 2.5).

ALGORITMO PARA LA BP

1.- Poner el nodo inicial en abiertos.

2.- Si abiertos esta vacia terminar con falla. En otro caso continuar.

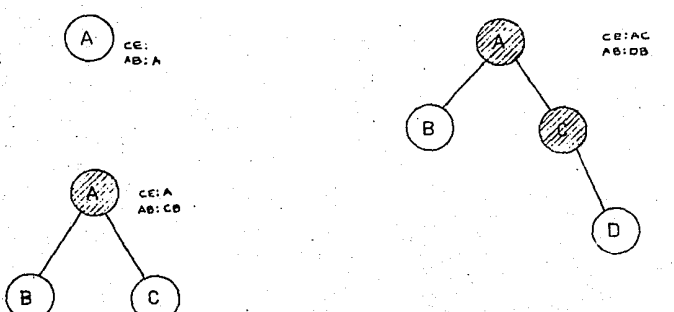
3.- Tomar el primer nodo de abiertos y ponerlo en la lista de cerrados. Llamar n a éste nodo.

4.- Si la profundidad de n es igual a la cota de profundidad. Regresar a 2. En otro caso continuar.

5.- Expander el nodo n generando todos sus sucesores. Poner estos sucesores al principio de la lista de abiertos y marcarlos como sucesores de n.

6.- Si cualquiera de los sucesores es la solución, salir con la trayectoria de solución que se obtiene al ir hacia atrás, en otro caso regresar a 2.

EJEMPLIFICACION GRAFICA DE LA BUSQUEDA EN PROFUNDIDAD (BP) PARA EL PROBLEMA DEL LABERINTO



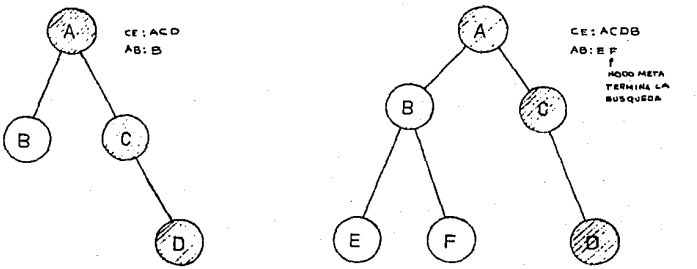


figura 2.5

2.4 BUSQUEDAS MINI-MAX

2.4.1 ARBOLES DE JUEGOS

Supóngase que dos personas están jugando el mismo juego, a uno de ellos se le denomina jugador y al otro adversario. Por razones de orden el juego se verá siempre desde el punto de vista del jugador; Por lo tanto se dirá que el juego se ha ganado cuando gane el jugador, y en caso contrario se perderá el juego (aunque cabe la posibilidad de un empate).

Cuando un juego se inicia el jugador tiene que hacer un primer movimiento ya sea que haya iniciado él el juego, o que lo haya hecho el adversario; En este momento el jugador tendrá varias opciones para realizar su jugada, al ejecutarla le toca mover al adversario, que tiene también las mismas opciones y así continúa el juego hasta que se gane o se pierda. Estas opciones bien pueden ser representadas por ramificaciones en un árbol, pero con ciertas caracterfsticas que lo diferenciaran de un árbol como los anteriormente vistos, el jugador tratará de escoger las opciones que lo lleven al éxito, mientras que el adversario las que lleven a fracaso, pudiendo existir la posibilidad de un empate.

La diferencia básica en un árbol de juegos es que el jugador selecciona solo la mitad de las opciones, pues la otra mitad corresponde al adversario, además de que por cada movimiento hecho para llegar al estado final, el adversario realizá uno que lo aleja de su objetivo, por lo tanto en un juego aparte de

buscar una buena opción hay que pelear contra las del adversario.

En un árbol de juegos los nodos corresponden a las posiciones en el tablero, y las ramas a los movimientos. La raíz corresponde a la posición inicial, y las hojas o terminales a posiciones en las que el juego ha terminado (el jugador gano, perdio o empato) o no existen jugadas legales por realizar. Por una posición, se entenderá toda la información que se pueda grabar. Un ejemplo de un árbol de juegos puede verse en la figura 2.6.

El árbol de juego es considerado un árbol **and/or** o **Y/O**. La razon de esto es que cuando el jugador le toca mover, la expansión del nodo es **disyuntiva** y en cambio es **conjuntiva** cuando le toca mover al adversario.

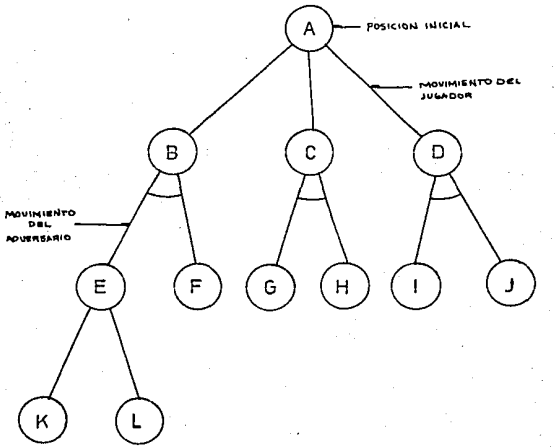


figura 2.6

2.4.2 ESTRATEGIAS DE JUEGOS

Para marcar un árbol de juegos se utiliza la notación g para gana, p para pierde y e para empate, y se siguen las siguientes reglas:

1.- Turno del jugador (nodo con expansión disyuntiva). Si al menos uno de los hijos del nodo está marcado con g, marcar el nodo con g, si todos los hijos están marcados con p, entonces marcar el nodo con p. En otro caso marcar el nodo con e.

2.- Turno del adversario (nodo con expansión conjuntiva). Si todos los hijos del nodo están marcados con g, entonces marcar el nodo con g. Si al menos uno de los hijos está marcado con p marcar el nodo con p. De otra forma marcar el nodo con e.

En la figura 2.7 se muestra el proceso mencionado.

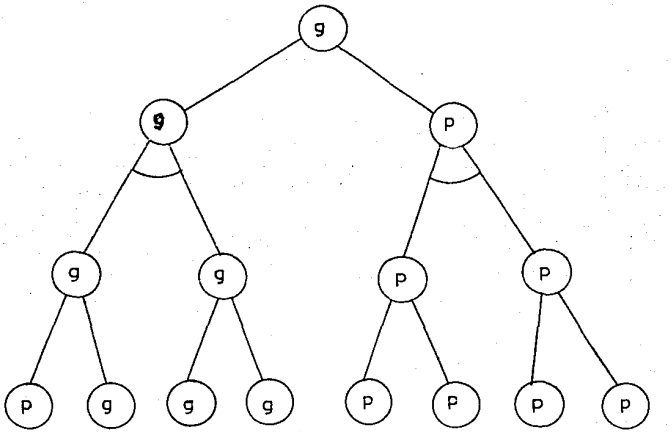


figura 2.7

2.4.3 EL METODO MINI-MAX

El método mini-max es un procedimiento para marcar un árbol de juegos y tiene la ventaja sobre el anterior, de que puede generalizarse a situaciones donde el juego total es muy grande y solo se tiene disponible parte del árbol del juego.

Para ejemplificar el método mini-max supóngase que se apuesta dinero en el juego, por ejemplo un peso. Si el jugador gana, obtiene un peso. Si pierde, pierde un peso. En caso de empate ni se gana ni se pierde dinero. A la cantidad que gana el jugador se le llama pago, si el jugador gana el pago es 1, si pierde el pago es -1; en caso de empate el pago es 0.

El pago del jugador está garantizado si sigue la estrategia descrita anteriormente. Por lo tanto, el pago es 1 para un nodo marcado con g, 0 para un nodo marcado con e y -1 para un nodo marcado con p; Estos números son lo que se conoce como valor de un nodo. Y no es necesario marcar los nodos primero y luego evaluarlos, sino que se evalúan directamente.

El proceso de evaluación sigue el orden de ir de las hojas a la rafz, evaluando el valor de cada nodo en términos de los valores de sus hijos. La estrategia de juego es que el jugador escoge siempre un movimiento que conduzca al pago más alto, mientras que el adversario escoge un movimiento que conduzca a una posición con pago menor. A continuación se presentan las reglas fundamentales de este proceso:

1.- Turno del jugador. El valor del nodo es el máximo de los valores de sus hijos.

2.- Turno del adversario. El valor del nodo es el mínimo de los valores de sus hijos.

En la figura 2.8 se ejemplifica la evaluación para nodos usando el método mini-max.

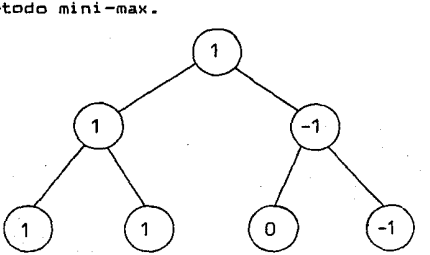


figura 2.8

2.4.4 EL METODO MINI-MAX EN PROFUNDIDAD

Suponiéndose que se tiene un procedimiento para generar todos los movimientos posibles en cualquier posición y también las posiciones que resultarán de hacer cualquiera de los movimientos. Se construye un árbol de juegos parcial como el de la figura 2.9, expandiéndolo y luego expandiendo a sus hijos y así sucesivamente.

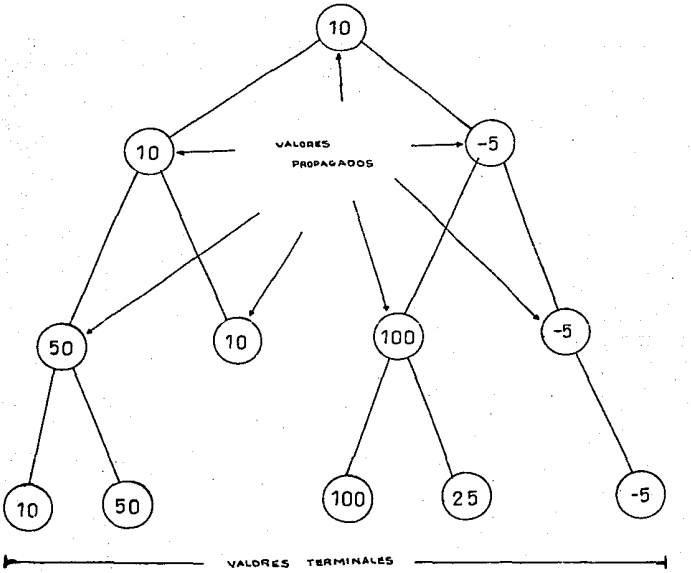


figura 2.9

Antes de expander cualquier nodo primero se verifica si es terminal. Si es terminal no se expande, sólo en caso contrario se expande. Hay que hacer notar que se especifica una cota de profundidad correspondiente al árbol cuya raíz es la posición actual. Continuando con este proceso hasta que no se puedan expander, terminando con un árbol cuya raíz corresponde a la posición actual y cuyas hojas corresponden a posiciones terminales. Aplicando la técnica mini-max se asigna un valor a cada nodo en el árbol. Como antes, el valor de un nodo en que le toca mover al jugador será el máximo de los valores de sus hijos y para el adversario el mínimo. La figura 2.10 muestra esa evaluación.

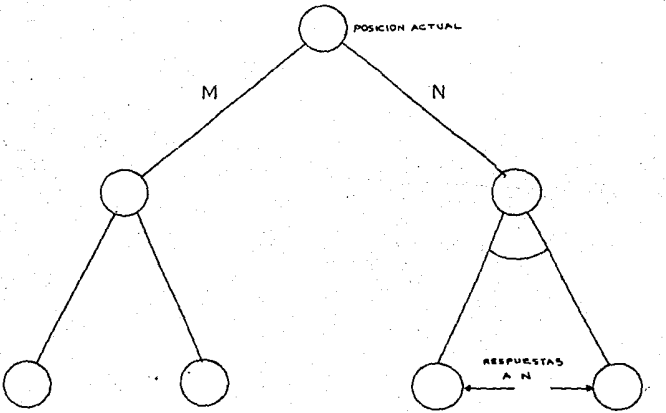


figura 2.10

Debido a la manera de como se calculan los valores asignados a nodos intermedios, se denominan valores propagados hacia atrás.

Siempre el mejor movimiento del jugador es el que conduce al máximo valor propagado. En este caso no se tiene que almacenar el árbol completamente. La técnica en profundidad consiste en la evaluación de los hijos de la raíz, al mismo tiempo que no se almacena en memoria mas que una parte del árbol, y se crean los nodos cuando se necesiten para la evaluación, se crea otro hijo cuando el procedimiento propaga y visita este nodo otra vez. Los únicos hijos que permanecen en memoria son los del camino que se esta explorando y los que tienen valores que no se han propagado.

La figura 2.11 ilustra la creación de nodos y la asignación de valores parcialmente propagados a cada nodo. Este valor se da como un valor inicial cuando se crea el nodo y se actualiza cada vez que se vuelve a visitar tal nodo, En cualquier momento, el valor parcialmente propagado para un nodo será el máximo o el mínimo (dependiendo de a quién le toque mover) de los valores de los hijos que se han evaluado hasta ese momento.

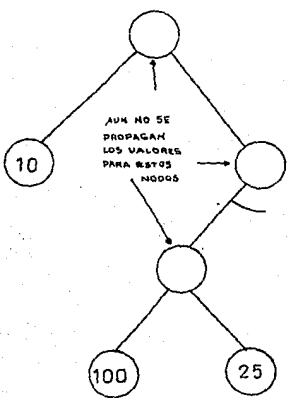


figura 2.11

Cuando se han evaluado todos los hijos de un nodo el valor parcialmente propagado será el máximo o el mínimo de los valores de todos sus hijos, y será justamente el valor del nodo. Para un nodo asociado al movimiento del jugador, se le da inicialmente al valor parcialmente propagado un valor negativo grande. Cuando se revisita un nodo, el valor del hijo que se acaba de evaluar se compara con el valor parcialmente propagado actual y el mayor de los dos se convierte en el valor actual parcialmente propagado.

Para un nodo correspondiente a un movimiento del adversario, el valor propagado tiene inicialmente un número positivo muy grande, cuando se revisita el nodo, se compara el valor del hijo que se evaluó recientemente con el valor propagado actual, y el menor de ellos pasa a ser el nuevo valor propagado. Por lo tanto, los únicos nodos que tienen que almacenarse en cualquier momento, son los estan en la trayectoria que va de la raíz al nodo que se procesa actualmente. Estos pueden almacenarse utilizando una estructura de pila o stack con la raíz en la base y el nodo que se examina en el tope.

2.5 EL ALGORITMO DE PODA ALFA-BETA

En una evaluación mini-max en profundidad se puede mostrar que ciertas partes del árbol no afectan los valores de interés y, por lo tanto no necesitan explorarse. La técnica para identificar las partes del árbol que pueden ignorarse se denomina algoritmo de poda alfa-beta (alfa-beta prunning).

Por una poda deberá entenderse que ya no se van a analizar ciertas ramas del árbol.

Supóngase un caso donde se tiene un valor propagado parcialmente de 10 para un nodo correspondiente a un movimiento del jugador. El valor propagado para el hijo que se evalua actualmente es 5. Ahora ya que el hijo es un nodo donde mueve el adversario, continuar explorando conduce a disminuir su valor. Esto se debe a que el adversario siempre se mueve hacia la posición que tiene el valor mínimo. Cualquiera que sea el valor del hijo, siempre será 5 o menos. Además siendo el valor propagado del nodo igual a 10, sólo los valores de hijos mayores a 10 pueden cambiar el valor propagado.

La conclusión es que no es necesario explorar el hijo o alguno de sus descendientes. Continuar explorando lo único que ocasiona es una reducción del valor propagado del nodo hijo, y el valor actual ya es demasiado pequeño para afectar el valor propagado del padre. Esta situación se denomina "corte alfa".

Concluyendo, se tiene que el principio se establece de la manera siguiente:

Considérese un padre en que mueve el jugador y un hijo en que mueve el adversario. Si el valor propagado para el hijo es menor o igual que el valor propagado para el padre, no es necesario procesar más en la dirección del hijo y sus sucesores, y la evaluación se propaga hasta el padre.

Si el nodo padre es uno en que mueve el adversario y el nodo hijo es uno en que mueve el jugador, se tiene lo siguiente: Considérese un nodo padre en que mueve el adversario y un nodo hijo en que mueve el jugador. Si el valor propagado para el hijo es mayor o igual al valor propagado para el padre, no es necesario procesar el hijo y sus sucesores, y la evaluación se propaga hasta el padre. Este caso se denomina "corte beta".

La técnica alfa-beta permite ignorar en ciertas ocasiones a algunos de los hijos de un nodo. Ya que cada hijo que no es terminal es la rafz de todo un subárbol de descendientes, se consigue ignorar al hijo y a todos sus descendientes. Por lo tanto, el procedimiento alfa-beta evita evaluar un gran número de nodos, produciendo ahorros sustanciales en el tiempo requerido para explorar el árbol.

A continuación se presentan reglas de bifurcación ("branching") y el algoritmo de poda alfa-beta.

2.5.1 REGLAS DE BIFURCACION PARA EL CORTE ALFA-BETA

- Si el adversario tiene una respuesta establecida de que su movimiento es malo éste no necesita cualquier otra alternativa. Si uno está derrotado hay ciertamente la necesidad de encontrar una salida en el peor de los casos. En este principio se fundamenta el procedimiento **Alfa-Beta**. Los principios del procedimeinto **Alfa - Beta** son:

> - Siempre que algo sea descubierto acerca de la mejor forma de ejecutar un nuevo movimiento se debe checar todo acerca del nodo antecesor. Esto puede dar una mejor idea acerca de la sensibilidad del nodo a generar.
>
> - Siempre que el valor exacto de un nodo ha sido calculado, se debe checar todo acerca de los nodos padre esto facilitará el encontrar el valor màs exacto del nodo a generar.

El parecido del prodedimiento alfa-beta con el de bifurcación y acotamiento estriba en que algunas rutas se demuestran que son malas aún cuando no son seguidas hasta el final.

El procedimiento se basa en el siguiente principio básico, si un adversario tiene una respuesta estableciendo que un movimiento potencial es malo, no hay necesidad de checar otras respuestas al movimiento potencial. Si una es mala, no hay necesidad de encontrar de cuántas maneras o que tan malo es el peor de los casos.

2.5.2 ALGORITMO ALFA-BETA

1) Determinar si el nivel es el nivel tope o si el límite de la búsqueda ha sido alcanzado o si el nivel es el nivel minimizante.

1a) Si el nivel es el tope, hacer que ALFA sea - y que BETA sea + .

1b) Si el límite de la búsqueda ha sido alcanzado, calcular el valor estático de la posicion actual relativa al jugador apropiado. Reportar el resultado.

1c) Si el nivel es un nivel minimizante :

1c1) Hasta que todos los hijos son examinados con minimax ALFA-BETA:

1c1.1) Hacer a BETA igual al menor de los valores dados de BETA.

1c1.2) Usar minimax en el siguiente hijo de la posicion, manejando esta nueva aplicación de minimax al actual ALFA-BETA.

1c2) Reportar BETA.

1d) Si el nivel es un nivel maximizante. Hasta que todos los hijos sean examinados por minimax o ALFA-BETA.

1d1.1) Hacer a ALFA igual al mayor de los valores dados de ALFA, hasta entonces reportado por minimax.

1d1.2) Usar minimax en el siguiente hijo de la posicion actual.

1e) Reportar ALFA.

NOTA: Es importante comprender que el algoritmo Alfa-Beta puede ser aplicado si se pregunta sobre el mejor y el peor de los casos.

2.6 ALGORITMO DE PODA ALFA-BETA MODIFICADO

En general, la desventaja principal del procedimiento alfa-beta es la explosión combinatoria. A menos que se controle la dimensión del árbol crecerá tanto que no se podrá almacenar en memoria ni explorar en una cantidad de tiempo razonable.

La dimensión del árbol puede controlarse por medio de métodos heurísticos que limiten el ancho y la profundidad de la búsqueda. El ancho se refiere al número de movimientos que se considera en cada posición. Si se consideran todos los movimientos legales para cada posición, se realiza una búsqueda en todo lo ancho. La profundidad se refiere al número de movimientos hacia adelante, hacia donde se explora una línea de juego particular. El alfa-beta modificado consiste precisamente en podar el árbol, y hacer reducciones en la generación de hijos a partir de técnicas heurísticas, es decir de tener conocimiento previo, para guiar la búsqueda y evitar la explosión combinatoria. Estas técnicas se discutirán en el capítulo 4.

NOTAS DEL CAPITULO 2

[1] Un cuantificador existencial es un símbolo lógico que se utiliza para expresar que hay algún objeto que tiene determinado atributo.

[2] El primer método se funda en algoritmos descubiertos por Quine, Kleene y otros, y el segundo en el teorema de Herbrand, en la teoría de las formas prenexas de Skolem.

[3] El problema de los cuatro colores, es el siguiente: Cuál es el mínimo de colores que se necesitan para pintar un mapa tal que ninguno de los países limítrofes (o provincias) tenga el mismo color ? Euler, genial matemático suizo de fines del siglo XVII y comienzos del XVIII, sostuvo que bastaban cuatro colores. Pero no pudo demostrarlo para todos los mapas posibles. El problema se presenta porque teóricamente, un mapa puede tener un número inmenso de países. Hace unos cinco o seis años, utilizando una computadora, se pudo resolver el problema y se vio que Euler tenía razón. La solución fué una perfecta cooperación entre el pensamiento creador del hombre y el "pensamiento" mecánico de la computadora. Los pasos no mecánicos los daba el hombre y la inmensa cantidad de operaciones mecánicas que debían hacerse entre paso y paso las hacía la computadora.

BIBLIOGRAFIA DEL CAPITULO 2

LIBROS

[Davi 1982] KNOWLEDGE BASED SYSTEMS IN ARTIFICIAL INTELLIGENCE
DAVIS, R.
LENAT, D.
1982 McGRAW HILL NEW YORK

[Pear 1984] HEURISTICS:INTELLIGENT SEARCH STRATEGIES FOR COMPUTER PROBLEM SOLVING PEAR, JUDEA
1984 ADDISON WESLEY PUBLISHING CO. INC. TEL AVIV
382 pp.

[Poly 1965] COMO PLANTERAR Y RESOLVER PROBLEMAS POLYA, G.
TRADUCCION POR: ZUGAZAGOITIA, JULIAN
1965 EDITORIAL TRILLAS MEXICO
DECIMOTERCERA REIMPRESION 1986
215 pp.

[Rich 1983] ARTIFICIAL INTELLIGENCE RICH, ELAINE
INTERNATIONAL STUDENT EDITION
1983 McGRAW HILL BOOK CO. SINGAPURE
436 pp.

[Wins 1979] ARTIFICIAL INTELLIGENCE WINSTON, PATRICK
1979 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
444 pp.

ARTICULOS

[Gasc 1977] EXACTLY HOW GOOD ARE HEURISTICS ? TOWARD A REALISTIC PREDICTIVE THEORY OF BEST-FIRST SEARCH
GASCHING, J.
1977 PROC. FIFTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE CAMBRIDGE MASSACHUSETTS

[Geor 1983] STRATEGIES IN HEURISTIC SEARCH GEORGEFF, M.P.
1983 ARTIFICIAL INTELLIGENCE 20 pag. 395-425
ELSEVIER SCIENCE PUBLISHER B. V. (NORTH-HOLLAND)

[Lena 1982] THE NATURE OF HEURISTICS LENAT, DOUGLAS B.
1982 ARTIFICIAL INTELLIGENCE 19 pag. 189-249
NORTH-HOLLAND

[Sace 1979] PROBLEM SOLVING TACTICS SACERDOTI, E.D.
1979 TN189, SRI INTERNATIONAL
MENLO PARK CALIFORNIA.

REPORTES TECNICOS

[Geva 1984] AN OVERVIEW OF ARTIFICIAL INTELLIGENCE AND ROBOTICS
VOLUME 1 - ARTIFICIAL INTELLIGENCE
PART A - THE CORE INGREDIENTS
1984 NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (N.A.S.A.) WASHINGTON
67 pp.

CAPITULO 3

INTRODUCCION AL LENGUAJE LISP

3.1 ANTECEDENTES E HISTORIA

Al final de los años cincuentas, surge un gran optimismo provocado por el auge de la computación, lo que lleva a los investigadores a la realización de los más diversos proyectos. Es precisamente en esta época cuando en el MIT (Massachusetts Institute of Tecnology), le es encargado a John McCarthy el proyecto "ADVICE TAKER" (literalmente "tomador de consejos", aunque en realidad se trataba de desarrollar un consejero). El "ADVICE TAKER" debía tener la capacidad de manejar oraciones en inglés. McCarthy pronto se dió cuenta que el proyecto no podría realizarse, porque se carecia de un lenguaje capaz de manejar exitosamente símbolos y oraciones. En ese momento se tenia fundamentalmente FORTRAN como el lenguaje más poderoso. Sin embargo, el problema principal es que el lenguaje está enfocado al manejo de fórmulas y expresiones numéricas. Es entonces cuando nace la idea de desarrollar un lenguaje que tuviera la capacidad de manejar expresiones simbólicas, tales como oraciones y textos.

McCarthy y un grupo de sus alumnos se dan a la tarea de crear el primer compilador de este lenguaje, al cual denominarón LISP (acrónimo de LISt Processing). En 1960 McCarthy publico un articulo titulado "Recursive Functions of Symbolic Expressions and their Computation by Machine" (Funciones Recursivas de Expresiones Simbólicas y su Procesamiento en Computadora, ver [McCa 1960]). El artículo establece las bases teóricas para la construcción del lenguaje.

En los primeros años de la decada de los sesentas es implementado el primer compilador, y desde entonces al presente, LISP se ha convertido en el lenguaje predilecto de la comunidad de IA, monopolizando todas sus áreas.

Por supuesto LISP ha tenido infinidad de mejoras, con lo que han surgido diversas versiones (a estas versiones se les conoce como **dialectos**), más sin embargo dos de ellas se han convertido en las más aceptadas: MACLISP e INTERLISP.

Con la aparición de las microcomputadoras comienzan a aparecer los primeros compiladores para este tipo de máquinas, ampliando su campo de aplicación, ya que el LISP estaba vedado para algunas máquinas incluso minicomputadoras, porque necesita gran cantidad de memoria para trabajar. Pero el avance tecnológico de los últimos años ha solucionado satisfactoriamente el problema, debido al desarrollo de las microcomputadoras con una formidable capacidad de memoria.

3.2 DIFERENCIAS ENTRE LISP Y OTROS LENGUAJES DE ALTO NIVEL

LISP como antes se mencionó, fué creado para el manejo de expresiones simbólicas, por lo tanto su enfoque es muy distinto al de otros lenguajes de alto nivel. Es sin lugar a dudas otra manera de pensar, aunque parece difícil de aprender en realidad es todo lo contrario, ya que su sintaxis es muy cercana al lenguaje natural cotidiano, y el manejo de sus funciones es básicamente el lenguaje matemático común.

LISP presenta dificultad cuando ya se ha aprendido algún otro lenguaje con propósitos diferentes como: BASIC, PASCAL O COBOL. Porque se puede decir sin temor a equivocarse que los ambientes de programación son diferentes. La tabla 3.A muestra las principales diferencias entre el LISP y otros lenguajes de alto nivel.

TABLA 3.A DIFERENCIAS ENTRE LISP Y OTROS LENGUAJES

LISP	LENGUAJES CONVENCIONALES (BASIC, PASCAL, FORTRAN, COBOL)
- Usado para procesamiento simbólico principalmente	- Usados para procesamiento numérico principalmente
- No necesariamente secuencial, búsqueda heurística (pasos de solución implícitos)	- Secuenciales de tipo algorítmico (pasos de solución explicitos)
- Estructuras de control generalmente separadas del dominio del conocimiento	- Información y control estan integrados conjuntamente
- Generalmente fácil de modificar, actualizar y expander	- Difícil de modificar la mayoría de las veces
- Algunas respuestas incorrectas son tolerables	- Respuestas correctas son requeridas
- Respuestas satisfactorias son aceptables	- La mejor solución posible es la única aceptada

Acerca del lenguaje LISP han surgido una serie de mitos en su contra, los siguientes son los principales:

1.- En general es lento.

2.- Los programas hechos en LISP son de gran tamaño, lo que dificulta su entendimiento.

3.- LISP es lentísimo en procesamiento numérico.

4.- LISP es difícil de aprender por el uso de los paréntesis.

Sin embargo, las conjeturas en su contra no son sino verdades a medias, las siguientes observaciones cuestionan las anteriores:

1.- Sí es lento, pero no tanto como el COBOL.

2.- Los programas sí son grandes, pero como los de cualquier otro lenguaje.

3.- LISP aunque puede hacer procesamiento numérico, fué construido con otros propósitos.

4.- Lo que sucede es que LISP es un lenguaje que exige ser ordenado y disciplinado al programar, sin embargo si se tiene experiencia matemática con el uso de los paréntesis, esto último más bien resulta una ventaja.

Ahora bien, el lenguaje cuenta con una serie de ventajas que lo hacen muy poderoso en su ámbito. Algunas ventajas importantes son:

1.- Facilidad para el manejo de listas.

2.- Fabricación por parte del usuario de sus propias funciones.

3.- Una gran cantidad de funciones y procedimientos disponibles para los más diversos propósitos.

4.- Funciona tanto a nivel intérprete, como a nivel compilador.

5.- El manejo dinámico de la memoria (el que la cantidad de memoria asignada al programa sea determinado durante la ejecución del mismo), mediante lo que se conoce como "recolección de basura".

El compilador huésped de este trabajo es el Microsoft muLISP-83(*) Artificial Intelligence Development System, para el sistema operativo MS-DOS(**) y microcomputadoras IBM-PC(***) y/o compatibles. Por lo tanto todos los ejemplos de programación y los programas de aplicación serán desarrollados utilizando el muLISP-83(*).

Se eligió este compilador porque no necesita de gran memoria para trabajar (basta una microcomputadora con 256K de memoria), y porque cuenta con dos bibliotecas de funciones que abarcan dos de los principales diálectos de LISP: INTERLISP Y MACLISP.

(*) muLISP es marca registrada de Soft Warehouse

(**) MS-DOS es marca registrada de Microsoft Corporation

(***) IBM-PC es marca registrada de International Business Machines Corporation

3.3 FUNCIONES PRIMITIVAS, NUMERICAS Y DE PROCESAMIENTO DE LISTAS

3.3.1 INTRODUCCION AL PROCESAMIENTO SIMBOLICO

Para poder entender la filosoffa de la programación en LISP, es necesario comprender el manejo de las oraciones; los siguientes ejemplos muestran el uso de los paréntesis y las oraciones que construyen. En el primer caso, información acerca de la UNAM y en el segundo una regla de identificación.

Ejemplo 1

```
(UNAM (UN_TIPO_DE_UNIVERSIDAD)
      (UBICACION (MEXICO DF)
      (CARRERAS_DE_COMPUTACION ( MAC
                                 ING_EN_COMPUTACION
                                 INFORMATICA ))
      (RECTOR (DR JORGE CARPIZO MCGREGOR )))
```

Ejemplo 2

```
(REGLA IDENTIFICA_ANIMAL
            (SI (ANIMAL TOMA LECHE)
                (ANIMAL ES FELINO)
                (ANIMAL ES RAYADO)
            (ENTONCES (ANIMAL ES TIGRE))))
```

Como se puede observar el uso de los paréntesis no es ningun problema, al contrario, separa adecuadamente los elementos en su categoria correspondiente. Estas sencillas reglas son las que se siguen con las instrucciones del lenguaje, a la hora de programar.

En LISP hay un término fundamentalmente importante, que se maneja todo el tiempo. Este término es **objeto**; objeto debe entenderse como el argumento o argumentos de una función.

En el manejo de LISP, existen dos tipos de objetos fundamentales: **átomos** y **listas**; un átomo generalmente esta formado de un conjunto de bits y puede ser desde un simple caracter hasta una cadena de caracteres, dependiendo la longitud de la cadena del compilador que se use, aunque el estándard es treinta caracteres. Una lista es un conjunto de átomos.

3.3.2 FUNCIONES PRIMITIVAS

En la explicación de las funciones, se seguirá el siguiente formato:

NOMBRE DE LA FUNCION [argumento(s)] explicación de lo que hace la función

Depués se presentarán dos ejemplos, llamando a la función con diferentes argumentos.

NOTA: Los ejemplos se harán en el muLISP, el signo $ es el promt del muLISP.

Muchas de las funciones son llamadas predicados, un predicado es una función que regresa una valor verdadero (T) o falso o nulo (NIL). Sin embargo, no todas las funciones son predicados.

En LISP se consideran funciones primitivas, a las funciones básicas, con las cuáles se pueden construir otras. Las principales funciones primitivas son:

ATOM [objeto] regresa T si <objeto> es un átomo, de otra forma regresa NIL.

```
$ (ATOM 'ALFA)
T

$ (ATOM '(A B C))
NIL
```

APPEND [lista1, lista2,..., listan] regresa una lista formada por los elementos de lista1 hasta los de listan en ese orden.

```
$ (APPEND '(A B) '(C D))
(A B C D)

$ (APPEND '(MATEMATICAS APLICADAS Y) '(COMPUTACION))
(MATEMATICAS APLICADAS Y COMPUTACION)
```

CAR [lista] regresa el primer elemento de la lista

```
$ (CAR '(MEXICO BRASIL ALEMANIA))
MEXICO
```

```
$ (CAR '(1 2 3))
1
```

CDR [lista] regresa toda la lista menos el primer elemento

```
$ (CDR '(A B C))
(B C)

$ (CDR '(X Y Z A B C))
(Y Z A B C)
```

EQ [objeto1 objeto2] regresa T si <objeto1> es identico a <objeto2>, de lo contrario regresa NIL.

```
$ (EQ 'A 'A)
T

$ (EQ 'CLASE 'CLASS)
NIL
```

En los ejemplos anteriores hay que hacer notar un detalle muy importante, este es, el uso del apóstrofe. El apóstrofe es necesario para que el compilador distinga en el momento de ejecución de la función, si el argumento es un átomo o el nombre de una lista, o si es una lista. En la sección 3.3.4 se verá con más detalle este concepto.

Dadas las funciones primitivas es posible combinarlas para dar respuesta a problemas más complicados. Por ejemplo supóngase que se quiere obtener el tercer elemento de la lista (MEXICO FRANCIA BRASIL PORTUGAL), ¿ Cómo se obtendría este resultado combinando CAR Y CDR ?, la respuesta a esta interrogante es bastante simple como se puede observar en el siguiente ejemplo:

```
$ (CAR (CDR (CDR '(MEXICO FRANCIA BRASIL PORTUGAL))))
BRASIL
```

El resultado es BRASIL, porque el primer CDR regresa la lista (FRANCIA BRASIL PORTUGAL), al actuar el segundo CDR se obtiene la lista (BRASIL PORTUGAL), y finalmente CAR se encarga de extraer BRASIL. Como se puede observar, la función más interna es la que se ejecuta primero, y esto hay que tenerlo muy en cuenta cuando se programa.

Tener la necesidad de escribir de forma explícita las funciones cada vez que se requiere, es un problema engorroso. Por tal razón LISP da la facilidad de que el programador construya sus funciones para la solución de algún problema.

El formato para la construcción de funciones es el siguiente:

```
(DEFUN NOMBRE_DE_LA_FUNCION (LAMBDA (<argumento(s)>)
          ( <cuerpo de la función>
             ----------------------
             ----------------------
             ----------------------
                                  ))...)))
```

Donde DEFUN es la instrucción que permite la definición de funciones, DEFUN = DEFINE FUNCTION y LAMBDA es la función identificadora de los argumentos.

Regresando al ejemplo, la aplicación de la definición DEFUN al formato da como resultado la función TERCERO. La función TERCERO dará como resultado el tercer elemento de cualquier lista.

```
$ (DEFUN TERCERO (LAMBDA (LISTA)
       (CAR (CDR (CDR LISTA)))))
TERCERO
```

y al aplicarla se tiene:

```
$ (TERCERO '(MEXICO ALEMANIA HOLANDA SUECIA)
HOLANDA
```

Como se aprecia, en la definición de una función no se utiliza el apóstrofe, ya que se está utilizando un argumento mudo, en este caso LISTA. Pero al ejecutar tercero debe utilizarse, pues sino, un error sería detectado.

3.3.3 FUNCIONES NUMERICAS

Las funciones numéricas en LISP consisten en las operaciones matemáticas básicas, teniendo en el muLISP la ventaja (o desventaja) de que se cuenta con infinita precisión entera.

Algunas de las principales funciones numéricas del lenguaje LISP son:

MINUS [n] regresa el negativo de n, donde n es un entero

```
$ (MINUS 10)
-10

$ (MINUS -8)
8
```

PLUS [n1, n2,..., nm] regresa la suma de los m términos

```
$ (PLUS 3 4)
7

$ (PLUS -7 2 -8)
-13
```

DIFFERENCE [n1, n2,..., nm] regresa la diferencia de los m terminos

```
$ (DIFFERENCE 3 4 5)
-6

$ (DIFFERENCE -3 1 0)
-4
```

TIMES [n1, n2,..., nm] regresa el producto de los m terminos

```
$ (TIMES 2 4 5)
40

$ (TIMES 3 5 4)
60
```

REMAINDER [n, m] es la función n modulo m

```
$ (REMAINDER 7 3)
1

$ (REMAINDER 20 3)
2
```

DIVIDE [n, m] es la función de división entera, regresa el resultado de la división y el residuo

```
$ (DIVIDE 7 3)
(2 . 1)

$ (DIVIDE 10 4)
(2 . 2)
```

GCD [n1, n2,..., nm] es la función máximo común divisor

```
$ (GCD 12 -16 20)
4

$ (GCD 0 5)
5
```

3.3.4 FUNCIONES DE PROCESAMIENTO DE LISTAS

En el manejo de expresiones simbólicas, al utilizar listas para la construcción de procedimientos y funciones no bastan las funciones primitivas. La filosofía de LISP es evaluar algún argumento de una función y retornar un valor. Esto último es efectivo, cuando los argumentos sean listas o símbolos.

Sin embargo, si se quiere evaluar a un símbolo, debe dotarse a éste de algún atributo o valor; para que la operación sobre él pueda ejecutarse. Al proceso de establecer un valor para un símbolo se le denomina **asignación** (la figura 3.1 ilustra éste procedimiento).

SIMBOLO ←ATRIBUTO ←VALOR

figura 3.1

Las funciones de asignación son generalmente empleadas para dotar de un valor a una variable dentro de un programa. Las siguientes funciones son funciones de asignación:

SETQ [nombre, objeto] asigna a nombre objeto y regresa objeto

```
$ (SETQ ESTADOS '(JALISCO COLIMA SINALOA))
(JALISCO COLIMA SINALOA)
```

si se teclea:

```
$ ESTADOS
(JALISCO COLIMA SINALOA)

$ (CAR ESTADOS)
JALISCO
```

Como puede observarse al manejar ESTADOS ya no es necesario el apóstrofe, pues ESTADOS ya ha dejado de ser un objeto sin atributo. Si se tecleara con apótrofe se tendría:

```
$ (CAR 'ESTADOS)
ESTADOS
```

tomándose ESTADOS como un elemento de una lista y no como una lista, he ahí la diferencia cuándo se utiliza apóstrofe y cuándo no.

POP [nombre] extrae el tope de la pila y actualiza el apuntador

```
$ (SETQ PILA '(A B C D))

(A B C D)

$ (POP PILA)
A

$ (POP PILA)
B

$ PILA
(C D)
```

PUSH [objeto, nombre] inserta <objeto> en el tope de la pila y actualiza el apuntador

```
$ (PUSH 'Z PILA)
(Z C D)

$ (PUSH 'Y PILA)
(Y Z C D)

$ PILA
(Y Z C D)
```

Otro tipo de funciones importantes en el procesamiento de listas lo forman las funciones llamadas **constructores de listas**, APPEND ya vista como función primitiva pertenece a esta última categoria. Las siguientes son funciones constructoras de listas:

CONS [objeto1, objeto2] regresa una nueva lista dependiendo de objeto, si es lista o no

```
$ (CONS 'A '( B C D))
(A B C D)

$ (CONS 'MEXICO '(PAIS AMIGO))
(MEXICO PAIS AMIGO)
```

LIST [objeto1, objeto2,..., objeton)] regresa una lista que consiste de los elementos objeto1 hasta objeto n

```
$ (LIST 'A 'B 'C 'D)
(A B C D)

$ (LIST 'A '(B C) 'D)
(A (B C) D)
```

Ahora bien, pese a que APPEND, CONS y LIST tienen un mismo propósito, el de construir listas, su funcionamiento es completamente diferente. Para ilustrar obsérvense los siguientes ejemplos:

```
$ (APPEND '(A B) '(C D))
(A B C D)

$ (CONS '(A B) '(C D))
((A B) C D)

$ (LIST '(A B) '(C D))
((A B) (C D))
```

De los ejemplos anteriores se desprenden las siguientes conclusiones:

APPEND actúa sobre los elementos que contienen los argumentos y asi construye la nueva lista.

CONS toma el primer argumento como tal y actúa sobre los elementos del segundo argumento, formando así la nueva lista.

LIST actúa tomando como elementos a los argumentos y asi construye la nueva lista.

REVERSE [lista, objeto] regresa los elementos de la lista en sentido inverso y añade <objeto>, aunque <objeto> es optativo.

```
$ (REVERSE '(BELGICA PARAGUAY MEXICO))
(MEXICO PARAGUAY BELGICA)

$ (REVERSE '(TOPE BASE) '(NODO))
(BASE TOPE . NODO)
```

Normalmente REVERSE es llamado con un solo argumento, aunque a menudo es muy útil hacer el llamado con dos argumentos.

LAST [lista] regresa el último nodo de la lista

```
$ (LAST '(A B C D E))
(E)

$ (CAR (LAST '$ (A B C D E))
E
```

Notese que en el segundo ejemplo sí se obtiene el último elemento de la lista. Es diferente un nodo a un elemento en particular.

3.4 RECURSION

La recursión es a menudo un concepto difícil de entender para la mayoría de la gente. Sin embargo, constituye una valiosa herramienta para la programación, pues ahorra lineas de código en algunos procesos. La recursión es precisamente una de las potencialidades del lenguaje LISP. ¿ Pero qué es la recursión ?.

definición de recursión

Es la definición de un objeto, presentando un proceso para producir dicho objeto. Es la definición de una función en términos de sí misma.

A una función se le llama recursiva, si se llama a si misma directamente o mediante un intermediario.

Un ejemplo típico de una definición recursiva es el factorial de un número, de enorme importancia en Probabilidad y Estadística. La definición de factorial es la siguiente:

$$N! = \begin{cases} 1 \text{ si } N = 0 \\ N*(N-1)! \text{ si } N > 0 \end{cases}$$

Si se observa el segundo renglon de la definición, N*(N-1)! es un llamado recursivo.

Ahora, ¿ Cómo podría llevarse esta definición a LISP y construir una función recursiva que calcule el factorial de un número ?. La respuesta es simple, se utilizará el formato para definición de funciones visto con anterioridad y se traducirá la definición al lenguaje LISP. La construcción de la función es la siguiente:

```
$ (DEFUN FACTORIAL (LAMBDA (N)
      ((ZEROP N) 1)
      (TIMES N (FACTORIAL (DIFFERENCE N 1))) ))
FACTORIAL
```

Notese que en el segundo ejemplo sí se obtiene el último elemento de la lista. Es diferente un nodo a un elemento en particular.

3.4 RECURSION

La recursión es a menudo un concepto difícil de entender para la mayoría de la gente. Sin embargo, constituye una valiosa herramienta para la programación, pues ahorra lineas de código en algunos procesos. La recursión es precisamente una de las potencialidades del lenguaje LISP. ¿ Pero qué es la recursión ?.

definición de recursión

Es la definición de un objeto, presentando un proceso para producir dicho objeto. Es la definición de una función en términos de sí misma.

A una función se le llama recursiva, si se llama a si misma directamente o mediante un intermediario.

Un ejemplo típico de una definición recursiva es el factorial de un número, de enorme importancia en Probabilidad y Estadística. La definición de factorial es la siguiente:

$$N! = \begin{cases} 1 \text{ si } N = 0 \\ N*(N-1)! \text{ si } N > 0 \end{cases}$$

Si se observa el segundo renglon de la definición, N*(N-1)! es un llamado recursivo.

Ahora, ¿ Cómo podría llevarse esta definición a LISP y construir una función recursiva que calcule el factorial de un número ?. La respuesta es simple, se utilizará el formato para definición de funciones visto con anterioridad y se traducirá la definición al lenguaje LISP. La construcción de la función es la siguiente:

```
$ (DEFUN FACTORIAL (LAMBDA (N)
      ((ZEROP N) 1)
      (TIMES N (FACTORIAL (DIFFERENCE N 1))) ))
FACTORIAL
```

En el segundo renglón ((ZEROP N) 1), es la condición que establece que si N es igual a cero, la función toma el valor de uno, de otra forma, se multiplica N por el factorial de N-1 y es aquí donde la función se llama a si misma. Pero ¿ cómo lo hace ? . Obsérvese con atención el siguiente ejemplo:

```
$ FACTORIAL(4)
24
```

En el calculo del factorial de 4, la función siguió el siguiente proceso:

```
FACTORIAL(4)
N = 0 ?  NO, entonces   (TIMES 4 (FACTORIAL(3))

FACTORIAL(3)
N = 0 ?  NO, entonces   (TIMES 3 (FACTORIAL(2))

FACTORIAL(2)
N = 0 ?  NO, entonces   (TIMES 2 (FACTORIAL(1))

FACTORIAL(1)
N = 0 ?  NO, entonces   (TIMES 1 (FACTORIAL(0))

FACTORIAL(0)
N = 0 ?  SI, entonces   FACTORIAL(0) = 1
```

Una vez calculado FACTORIAL(0) que es el nivel más profundo, pueden calcularse los factoriales pendientes hasta llegar al nivel superior, en este caso cuatro. El cálculo continúa de la siguiente forma:

```
FACTORIAL(0)=1
FACTORIAL(1)=(TIMES 1 FACTORIAL(0)) =  1
FACTORIAL(2)=(TIMES 2 FACTORIAL(1)) =  2
FACTORIAL(3)=(TIMES 3 FACTORIAL(2)) =  6
FACTORIAL(4)=(TIMES 4 FACTORIAL(3)) = 24
```

Internamente la computadora mediante un stack, está guardando los llamados pendientes, cuando N es igual a cero comienzan las sustituciones hacía arriba, hasta obtener el valor deseado (ver figura 3.2).

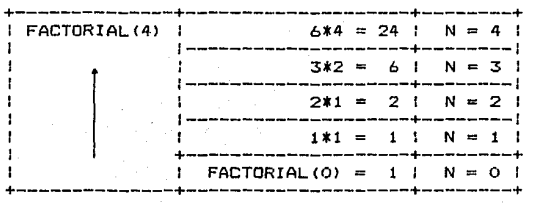


figura 3.2

Una forma de visualizar el concepto es por medio de un dibujo que ilustre la recursión, para el pasado ejemplo puede verse la figura 3.3.

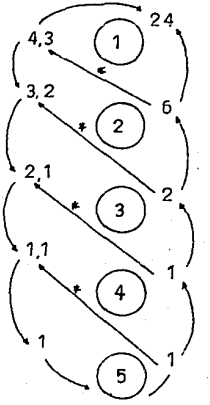


figura 3.3

La mayoría de las funciones en LISP, han sido definidas en forma recursiva, es por ello que este concepto debe quedar perfectamente entendido para poder aprovecharlo al máximo.

En los siguientes ejemplos se presentan definiciones de funciones recursivas utilizando listas, con lo cual se reafirma el concepto.

A.- Construir la función LISTAEQ, la cual compara si dos listas son iguales.

Para poder realizar esta tarea es necesario contar con un algoritmo, antes de programar. Un buen algoritmo es el siguiente:

1.- Llamar LISTAEQ(lista1 lista2)

2.- Si las dos listas son ya nulas (NULL) regresar T y terminar

3.- Si lista2 es nula (NULL) regresar NIL y terminar

4.- Si el CAR de lista1 no es igual (EQ) al CAR de lista2 regresar NIL y terminar, de lo contrario ir a 5

5.- Llamar LISTAEQ(CDR lista1 lista2)

El algoritmo anterior expresado en un programa en LISP queda de la siguiente forma:

```
$ (DEFUN LISTAEQ (LAMBDA (LISTA1 LISTA2)
        ((NULL LISTA1) (NULL LISTA2))
        ((NULL LISTA2) NIL)
        ((NOT (EQ (CAR LISTA1) (CAR LISTA2))) NIL)
        (LISTAEQ (CDR LISTA1) (CDR LISTA2)) )))
LISTAEQ
```

Al llamar a la función:

```
$ (LISTAEQ '(A B C) '(A B C))
T

$ (LISTAEQ '() '())
T

$ (LISTAEQ '(PERRO GATO CANARIO) '(PERRO GATO LEON))
NIL
```

B.- Construir MIEMBRO, la función que determine si un nombre es miembro de una lista.

El algoritmo correspondiente a este problema es el siguiente:

1.- Llamar MIEMBRO(nombre lista)

2.- Si la lista es ya nula (NULL) regresar NIL y terminar

3.- Si nombre es igual al CAR de lista regresar T y terminar, si-no llamar MIEMBRO(nombre CDR lista)

La traducción a LISP queda de la siguiente forma:

```
$ (DEFUN MIEMBRO (LAMBDA (NOMBRE LISTA)
                  ((NULL LISTA) NIL)
                  ((EQ NOMBRE (CAR LISTA) T)
                  (MIEMBRO NOMBRE (CDR LISTA)) )
MIEMBRO
```

Al llamar a la función:

```
$ (MIEMBRO 'pera '(manzana pera uva))
T

$ (MIEMBRO 'X '(A B C))
NIL

$ (MIEMBRO 'MEXICO '(BRASIL COLOMBIA VENEZUELA MEXICO))
T
```

En resumen, en el caso de la programación para resolver la evaluación de posiciones en un juego, el concepto de recursión es uno de los mejor utilizados. Asi como se construyeron las funciones anteriores pueden construirse otras para los más diversos propósitos dentro de la manipulación simbólica. En el caso particular del ambiente de la programación heurística para la solución de problemas de búsqueda, se requiere ampliamente el concepto de recursión y su manejo en LISP.

BIBLIOGRAFIA DEL CAPITULO 3

LIBROS

[Char 1980] ARTIFICIAL INTELLIGENCE PROGRAMMING
CHARNIAK, EUGENE
RIESBECK, CHRISTOPHER K.
McDERMOTT, DREW V.
1980 LAWRENCE ERLBAUM ASSOCIATES, PUBLISHER
HILLSDALE, NEW JERSEY
323 pp.

[Wins 1979] ARTIFICIAL INTELLIGENCE WINSTON, PATRICK
1979 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
444 pp.

[Wins 1981] LISP SECOND EDITION WINSTON, PATRICK
HORN, BERTHOLD KLAUS PAUL
1981 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
435 pp.

ARTICULOS

[McCa 1960] RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION BY MACHINE McCARTHY, J.
1960 COMUNICATIONS ACM 11 pp. 184-195

[Viso 1984] BREVE HISTORIA DE ALGUNOS LENGUAJES DE PROGRAMACION
VISO, ELISA
1984 CERO UNO CERO No.4 VOL. 4
FUNDACION ARTURO ROSENBLEUTH MEXICO
5 pp. 31-35

MANUALES

[Muli 1983] MICROSOFT muLISP ARTIFICIAL INTELLIGENCE DEVELOPMENT SYSTEM RICH, ALBERT D.
1983 THE SOFT WAREHOUSE
HONOLULU
140 pp.

CAPITULO 4

LENGUAJE LISP COMO HERRAMIENTA A LA PROGRAMACION HEURISTICA

4.1 REPRESENTACION DE REGISTROS EN LISP

4.1.1 NOMBRES

En el capítulo 3 se trataron los lineamientos básicos del lenguaje LISP así como ejemplos sencillos de programación. En este capítulo se presentarán técnicas más avanzadas de la programación en LISP.

Hay tres tipos fundamentales de objeto datos primitivos en LISP: NOMBRES NUMEROS y NODOS. LISP provee numerosas funciones para reconocer, comparar, combinar y operar a los distintos objeto datos. Estos pueden construir complejas estructuras de datos que llegan a representar virtualmente, cualquier modelo de la vida real.

Como se recordará LISP sólo tiene dos tipos de objetos: ATOMOS y LISTAS. En LISP, los átomos son subdivididos en NOMBRES y NUMEROS. Y las listas son sólo un subconjunto de una estructura más general conocida como árbol binario; un árbol como se recordará está constituido de NODOS.

El primer tipo de los objeto datos que se discutirá es el NOMBRE. Asociado con cada nombre en LISP hay cuatro atributos:

1.- Cadena de impresión-nombre: Una única cadena de caracteres ASCII usada por el sistema para identificar el nombre en la entrada y desplegar el nombre en la salida. Una cadena de impresión-nombre no puede ser cambiada.

2.- Valor actual: Un valor de nombre puede ser cualquiera de los objeto datos incluso el nombre mismo. El valor por default de un nombre es el nombre mismo.

3.- Lista de propiedad: La lista de propiedad contiene los valores de la propiedad del nombre indexada en llaves de propiedad. La lista de propiedad por default de un nombre es la lista nula (NULL list).

4.- Definición de función: Una función de definición de un nombre es aplicada a los argumentos dados cuando el nombre es llamado como una función. La definición de función por default llama al error "Undefined Function".

La función reconocedora NAME regresa T si su argumento es un nombre de otra manera regresa NIL. Por ejemplo:

```
$ (NAME 'XYZ)
T
```

```
$ (NAME 41)
NIL

$ (NAME '(DOG CAT COW))
NIL
```

Como se discutió anteriormente EQ es usado para determinar si dos nombres son idénticos. Por ejemplo:

```
$ (EQ 'APPLE (CAR '(APPLE ORANGE LEMON)))
T
```

Notese que mayúsculas y minúsculas forman distintos nombres. Por ejemplo:

```
$ (EQ 'Apple 'APPLE)
NIL
```

Aunque los espacios en blanco, los paréntesis y otros caracteres tienen un significado especial para LISP, una cadena entrecomillada debe ser usada para crear un nombre cuyos caracteres son su cadena impresión-nombre. Simplemente se encierra la cadena con comillas para crear un nombre. Por ejemplo:

```
$ "Este es un nombre!"
Este es un nombre!
```

Normalmente las comillas alrededor de nombre contienen caracteres especiales que no serán desplegados cuando nombre sea un dato de salida. (Nota: Las comillas son automáticamente desplegadas alrededor de nombre si la variable de control PRIN1 es NIL).

La cadena vacia, tecleada como "", es también un nombre. Por ejemplo:

```
$ (NAME "")
T
```

Las comillas pueden ser incluidas como una cadena impresión-nombre tecleando una comilla por cada comilla de la cadena. Por ejemplo:

```
$ "Una persona dijo:""La constancia es la clave del éxito"""
Una persona dijo:"La constancia es la clave del éxito"
```

Cuando se introduzcan nombres usando comillas, es esencial balancear las comillas. Después de introducir una comilla, LISP continuará tratándolo como un nombre simple hasta que un apóstrofe sea leído.

A un nombre se le puede asignar un valor. El valor puede ser cualquiera de los objeto datos incluso él mismo. El valor por default de un nombre es el nombre mismo.

En el capítulo 3 al tratarse las funciones de asignación, se explicó claramente cómo se realiza la asignación de un valor o atributo a un objeto a través de la función SETQ.

4.1.2 NUMEROS

Un número en LISP es una cadena de dígitos. Los números negativos son precedidos por el signo menos. Como el valor de un número es el número mismo, no es necesario el apóstrofe:

```
$ 41
41

$ -75
-75
```

La función NUMBERP regresa T si su argumento es un número de otra forma regresa NIL.

```
$ (NUMBERP 100)
T

$ (NUMBERP 'CINCO)
NIL

$ (SETQ PRIMOS '(2 3 5 7 11))
(2 3 5 7 11)

$ (NUMBERP (CAR PRIMOS))
T
```

Si un número es introducido entre comillas ¿Qué es, número o nombre ? Los siguientes ejemplos dan respuesta a esta pregunta:

```
$ (NUMBERP "137")
NIL

$ (NAME "137")
T
```

Los nombres y números son llamados átomos. Cómo se recordará la función primitiva ATOM reconoce si un objeto es un átomo o no. Los siguientes ejemplos son aplicados a nombres y números:

```
$ (ATOM 'manzana)
T

$ (ATOM 123)
T
```

4.1.3 NODOS

El nodo es el tercer tipo de objeto datos y es la estructura de datos primaria en LISP. Cada nodo consiste de una celda CAR y una celda CDR (ver figura 4.1). La celda de un nodo, solo puede apuntar a un tipo de objeto datos como: nombre, número o nodo.

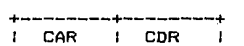


figura 4.1

Los nodos son a veces llamados "pares punteados", por la notación punto usada para representar los nodos en LISP. Si X y Y son objeto datos, la expresión:

(X . Y)

denota un nodo cuya celda CAR apunta a X y cuya celda CDR apunta a Y. Aunque un poco incomoda, la notación punto es capaz de mostrar la estructura de objeto datos en LISP.

En muchas aplicaciones es más conveniente pensar una estructura de datos como una lista lineal más que como un árbol binario. La notación lista es la adecuada para ello. La lista:

(X1 X2 ... Xn)

representa una cadena de nodos unidos a través de sus celdas CDR y cuya celda CAR apunta a los elementos de la lista. Para ser una lista válida la celda CDR del último nodo de la lista debe apuntar al átomo NIL. La estructura de lista mencionada anteriormente, se expresa en términos de notación punto como:

(X1 . (X2 . (... .(Xn . NIL) ...)))

La función READ acepta ambas notaciones. En notación lista, la lista vacia (NULL) es denotada por NIL. Sin embargo, cuando "()" es leído por READ, NIL es regresado.

Las funciones de impresión PRIN1 y PRINT usan notación lista a la máxima extensión posible. Cuando imprimen estructuras de datos, por ejemplo:

(AZUL . (CLARO . MARINO))

Da como salida por PRINT o PRIN1:

(AZUL CLARO . MARINO)

Hay que hacer notar que un nodo es una estructura no atómica, esto último hay que tenerlo muy en cuenta cuando se programa en LISP.

4.2 MANIPULACION DE ARBOLES EN LISP

El concepto de nodo tratado en la sección 4.1.3 es de vital importancia para la creación y manipulación de árboles binarios. Para entender estos importantes conceptos es necesario observar cómo son tratados y almacenados los datos en la computadora.

Los datos pueden ser almacenados en la memoria de la máquina, considerando que la localidad se conoce como su **dirección**. Una dirección es análoga a una dirección de una persona en una carta puesta en el buzón. El dato almacenado es análogo a la misma correspondencia que se encuentra en el buzón. Como sucede con los buzones, el contenido de la memoria en forma análoga también puede cambiar.

Supóngase que se desea representar el nodo que consiste del nombre FERNANDO y su edad 23. Se puede almacenar el nombre FERNANDO comenzando en la localidad siete, su edad 23 en la localidad dos, y comenzando en la localidad cuatro se puede almacenar un nodo, el cual consiste de un par de direcciones la siete y la dos (ver figura 4.2):

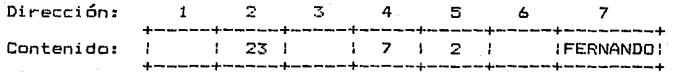


figura 4.2

LISP maneja el lugar específico de los datos con la memoria automáticamente, ya sean nombres o números inclusive.

La figura 4.3 representa el mismo nodo, pero en forma más simple:

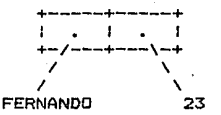


figura 4.3

En la representación anterior se utilizó una caja para ilustrar un nodo. Sin embargo es más conveniente visualizarlo en notación punto como muestra la figura 4.4.

figura 4.4

La anterior representación gráfica de un nodo tomando en cuenta la notación punto (vista anteriormente en la sección 4.1.3) es la adecuada para representar árboles binarios. ¿ Por qué ? simplemente porque al constar de una raíz hipotética (el punto) y dos ramas se puede interpretar precisamente como un árbol binario.

En resumen, se tiene que la representación en términos de árbol binario da una conceptualización más natural de los nodos y su interrelación. Sin embargo, por otro lado la representación en términos de una lista lineal constituye la mejor forma de interpretación para el lenguaje LISP.

Precisamente una de las representaciones lineales para LISP, es la notación punto (vista en la sección 4.1.3). Por ejemplo el nodo visto con anterioridad (figura 4.4) en notación punto es representado de la siguiente manera:

(FERNANDO . 23)

Como se recordará el elemento de la izquierda es llamado el CAR del nodo. Mientras que el elemento de la derecha es llamado el CDR del nodo. Los elementos de un nodo pueden ser cualquiera de los objeto datos incluyendo otro nodo. Por ejemplo, el apellido de FERNANDO puede ser incluido en el árbol binario de la manera siguiente (ver figura 4.5):

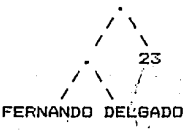


figura 4.5

La representación punto correspondiente es la siguiente:

((FERNANDO . DELGADO) . 23)

Si se añade el hecho de que FERNANDO es un estudiante, el árbol binario que representa esto es el que muestra la figura 4.6

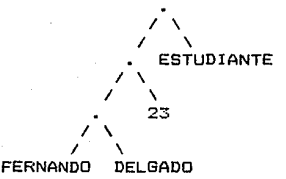


figura 4.6

El árbol representado en notación punto es el siguiente:

(((FERNANDO . DELGADO) . 23) . ESTUDIANTE)

Sin embargo, no es la única forma de representar los datos en notación punto. Una forma alternativa es:

((FERNANDO . DELGADO) . (23 . ESTUDIANTE))

y el correspondiente árbol binario es el que muestra la figura 4.7

figura 4.7

Como puede observarse, los nodos interrelacionados pueden ser usados para representar virtualmente cualquier estructura de datos en forma de árbol. La estructura de árbol es la forma más natural de representar una colección de nodos, como una lista de objeto datos, más que como un árbol binario con cierta profundidad. Por ejemplo, los elementos de un conjunto son generalmente desplegados como una lista.

LISP representa una lista como una colección ligada de nodos cuya celda CAR apunta a los miembros de la lista y cuya celda CDR apunta al próximo nodo. La lista es terminada por una celda CDR que apunta a NIL (ver figura 4.8).

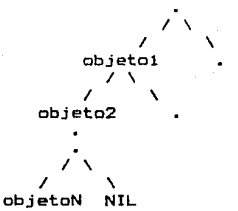


figura 4.8

Cuando el árbol binario (ver figura 4.8) se rota 45 grados en dirección de las manecillas del reloj, es fácil visualizar porque puede ser usado para representar una lista de objeto datos (ver figura 4.9).

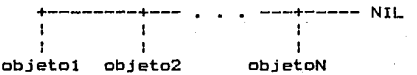


figura 4.9

La estructura lineal de listas sugiere una representación de impresión externa, que es mucho más fácil de leer que la notación punto correspondiente. El muLISP automáticamente despliega una estructura de lista (como la de la figura 4.9) en su notación lista correspondiente:

```
(objeto1 objeto2 ... objetoN)
```

en lugar de usar la notación punto:

```
(objeto1 . (objeto2 . ... (objetoN . NIL) ...))
```

muLISP despliega funciones usando notación lista donde es posible y notación punto cuando es necesario. Por ejemplo, una estructura de la forma como la que muestra la figura 4.10.

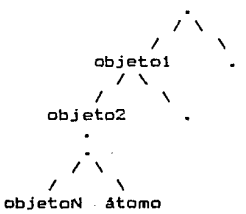


figura 4.10

donde <átomo> no es el nombre NIL, es desplegada en una notación mezclada como la siguiente:

(objeto1 objeto2 - - - objetoN . átomo)

Las funciones de lectura de datos de entrada del muLISP aceptan notación lista, notación punto y notación mezclada. Sin embargo, alguno de los elementos de una lista pueden ser otras listas o alguna expresión más general. Los siguientes ejemplos muestran como muLISP despliega algunas expresiones:

```
$ '(PERRO . (GATO . (VACA . BECERRO)))
(PERRO GATO VACA . BECERRO)

$ '((EDAD . 23) . (HOBBIES . (TENIS . COMPUTACION)))
((EDAD . 23) HOBBIES TENIS . COMPUTACION)
```

4.3 COMO TRABAJAN LOS ALGORITMOS DE BUSQUEDA

En el capítulo 2 se explicó a detalle el algoritmo de poda alfa-beta sin embargo, queda la duda de cómo es que realizá la búsqueda de opciones. Ya que el algoritmo alfa-beta es el que se va a emplear para la programación de los juegos del capítulo 5, es bastante adecuado profundizar en él y observar cómo es que trabaja.

Primeramente se debe contar con una función de evaluación estática. Esta función se encarga de evaluar una posición en el tablero del juego y regresar un valor que indica qué tan buena es esa posición para ganar el juego. Dependiendo de ese valor se selecciona o no la jugada. El valor se da en positivo (+) si se trata de un nivel en que le toca mover al jugador (nivel maximizante) y en negativo (-) si se trata de un nivel en que le toca mover al adversario (nivel minimizante). Por supuesto la función difiere de acuerdo al juego que se esté jugando.

Supóngase que a partir de la posición del nodo A (nivel maximizante) se necesita saber qué jugada es la más conveniente, la que lleva a B o C. Obsérvese con detenimiento la figura 4.11 y su explicación.

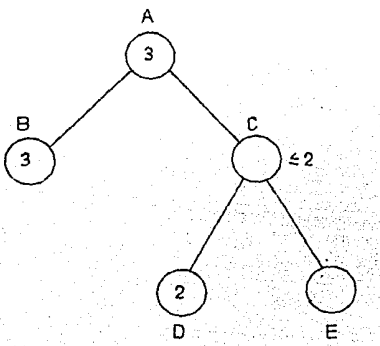


figura 4.11

De A se profundiza hasta D (ver figura 4.11), donde la función de evaluación da un valor de 2, el valor propagado a C será uno menor o igual a 2 por ser un nivel minimizante. En este punto ya no es necesario continuar expandiendo al nodo C, porque el valor propagado de A ya no será alterado, pues B tiene ya un valor de 3 y aunque el valor de C pueda cambiar por uno menor por tratarse de un nivel minimizante, siempre se seleccionará B pues C no podrá tener nunca un valor mayor a B.

Y el valor propagado a A será 3 por tratarse de nivel maximizante. En esta situación ocurre lo que se denomina corte alfa, produciendose un ahorro sustancial en el análisis de las jugadas al no tener la necesidad de analizar a los descendientes de C.

Supóngase que ahora estando en A se desea saber qué jugada es la mejor para el adversario (nodo B). Obsérvese con detenimiento la figura 4.12 y su explicación.

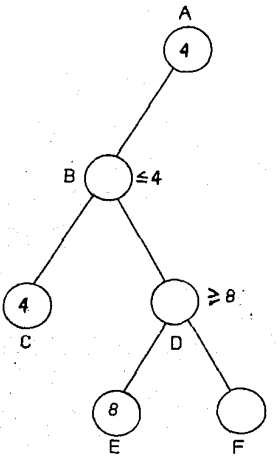


figura 4.12

A partir de A se profundiza hasta el nodo E donde la función de evaluación da un valor de 8, por lo tanto el valor propagado para el nodo D será uno mayor o igual que 8 por tratarse de un nivel maximizante. En esta situación dado que el nodo C tiene un valor propagado de 4, ya no es necesario continuar expandiendo al nodo D, ya que tenga el valor que tenga no podrá afectar al valor propagado de B, pues B seleccionará el menor por tratarse de un nivel minimizante, aunque D tenga descendencia con valores menores su valor propagado será 8 o más y si los tiene menores C siempre será menor que D. En esta situación es cuando ocurre el corte beta, y el adversario seleccionará la jugada que lo lleva a C. Ahorrándose el análisis de los descendientes de D.

Observando cómo es que trabaja el algoritmo de poda alfa-beta, se puede ver claramente que en la forma de profundizar hasta un nodo, trabaja como el algoritmo de búsqueda en profundidad, es decir, necesita de una cota de profundidad para luego hacer el retroceso "backtracking" y seleccionar la jugada adecuada, ya sea que el juego se esté viendo desde el punto de vista del jugador (alfa) o del adversario (beta).

4.4 ANALISIS DE LA COMPLEJIDAD

Como se mencionó en el capítulo 2, el algoritmo de poda alfa-beta tiene un enemigo y éste es la explosión en forma combinatoria. A pesar de ser un algoritmo eficiente, con grandes árboles de juegos se vuelve lento e ineficiente. Esto se menciona con el fin de que se comprenda su verdadera capacidad y no se espere más de lo que puede en realidad hacer.

Cuando por suerte o por cualquier otro motivo, el árbol de juegos que se examina está perfectamente ordenado, el número de evaluaciones estáticas necesarias para encontrar el mejor movimiento está dado por la siguiente fórmula:

$$\text{número de evaluaciones} = \begin{cases} \sum_{d/2}^{} 2b^{d/2} - 1 & \text{para d inclusive} \\ + & \\ \sum_{(d-1)/2}^{(d-1)/2} b^{(d-1)/2} + b^{(d-1)/2} - 1 & \text{hasta d-1} \end{cases}$$

donde b = factor de bifurcación

y d = grado de profundidad de la búsqueda

Sin embargo, fuera de este afortunado caso no es posible determinar cuál será el número de evaluaciones estáticas en promedio.

Pero a pesar de las anteriores desventajas, si bien es cierto que el algoritmo alfa-beta no detiene la explosión combinatoria, sí reduce su tasa de crecimiento como se puede observar en la gráfica de la figura 4.13. Donde el eje de las x's corresponde al grado de profundidad de la búsqueda y el eje de las y's al número de evaluaciones estáticas. Se puede apreciar claramente que se retarda bastante el crecimiento en forma combinatoria, pues mientras, sin el alfa-beta después del

grado de profundidad 6 y un número de evaluaciones estáticas de aproximadamente 256 es cuando comienza el crecimiento combinatorio, con el alfa-beta se retrasa hasta el grado de profundidad 14 con el mismo número de evaluaciones estáticas.

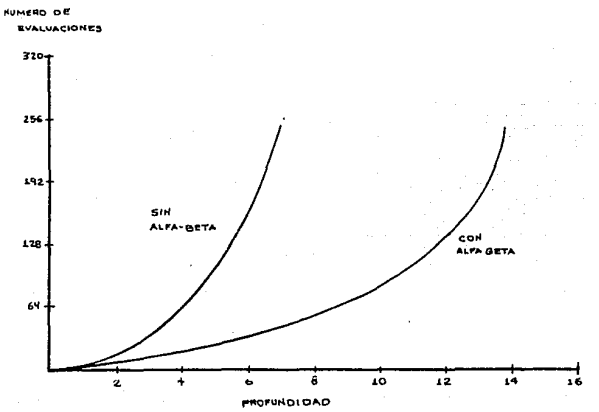


figura 4.13

Si se desea profundizar más en el tema puede verse el trabajo realizado por N.M. Darwish [Darw 1983], donde hace un análisis muy completo del algoritmo y bastante extenso para poder ser tratado en este trabajo. Tambien puede verse un buen análisis hecho por Knuth [Knut 1975].

4.5 PROGRAMACION EN LISP DE LOS ALGORTIMOS DE BUSQUEDA

Para poder programar los juegos del capítulo 5, es necesario definir algunas funciones de uso común. Puesto que se trata de juegos diferentes, sólo en lo que concierne al algoritmo alfa-beta se tendrá un mismo procedimiento. Pero éste será el que determine el mayor o menor valor, de acuerdo a valores calculados por una función de evaluación. Esto se debe a que la función de evaluación es distinta en cada caso, o quizá no es necesaria.

Las dos funciones fundamentales son: MAXIMO y MINIMO. MAXIMO es la función que obtiene el valor máximo de una lista de valores. MINIMO se encarga de obtener el mínimo de una lista de valores.

Dos definiciones bastante eficientes son las siguientes:

```
$ (DEFUN MAX-AUX (LAMBDA (NUM LISTA)
         (LOOP
             ((ATOM LISTA) NUM)
             ((( GREATERP NUM (CAR LISTA)))
               (SETQ NUM (CAR LISTA)) )
               (POP LISTA) ) ))
MAX-AUX


$ (DEFUN MAXIMO (LAMBDA LISTA
                  (MAX-AUX (CAR LISTA) (CDR LISTA)) ))
MAXIMO
```

Al hacer el llamado de la función se tiene:

```
$ (MAXIMO '(-10 2 5 77 8 66))
77
```

La definición de MINIMO es análoga a la de MAXIMO, solo que usando la función LESSP, en lugar de la de GREATERP, como se puede observar a continuación.

```
$ (DEFUN MIN-AUX (LAMBDA (NUM LISTA)
         (LOOP
             ((ATOM LISTA) NUM)
             ((LESSP NUM (CAR LISTA)))
              (SETQ NUM (CAR LISTA) ))
              (POP LISTA) )))
MIN-AUX


$ (DEFUN MINIMO (LAMBDA LISTA
         (MIN-AUX (CAR LISTA) (CDR LISTA)) ))
MINIMO
```

Al hacer el llamado de la función se tiene:

```
$ (MINIMO '(2 3 -9 -11 77))
-11
```

En lo que respecta a la búsqueda en profundidad del algoritmo, se utilizará la estructura de árbol discutida en este capítulo. Para realizar las tareas específicas se considera la definición de funciones propias, así como las funciones PUSH, POP, REVERSE y otras vistas con anterioridad. Para la representación de una posición en el tablero, lo más adecuado es usar listas y los comparadores lógicos.

Si se desea conocer estas técnicas de programación más a fondo, puede verse el apendice A, que contiene los programas de aplicación a juegos.

BIBLIGRAFIA DEL CAPITULO 4

LIBROS

[Char 1980] ARTIFICIAL INTELLIGENCE PROGRAMMING
CHARNIAK, EUGENE
RIESBECK, CHRISTOPHER K.
McDERMOTT, DREW V.
1980 LAWRENCE ERLBAUM ASSOCIATES, PUBLISHER
HILLSDALE, NEW JERSEY
323 pp.

[Pear 1984] HEURISTICS:INTELLIGENT SEARCH STRATEGIES FOR COMPUTER PROBLEM SOLVING PEAR, JUDEA
1984 ADDISON WESLEY PUBLISHING CO. INC. TEL AVIV
382 pp.

[Rich 1983] ARTIFICIAL INTELLIGENCE RICH, ELAINE
INTERNATIONAL STUDENT EDITION
1983 McGRAW HILL BOOK CO. SINGAPURE
436 pp.

[Slag 1971] ARTIFICIAL INTELLIGENCE: THE HEURISTIC PROGRAMMING APPROACH SLAGLE, JAMES R.
1971 McGRAW HILL BOOK CO. NEW YORK
196 pp.

[Wins 1979] ARTIFICIAL INTELLIGENCE WINSTON, PATRICK
1979 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
444 pp.

[Wins 1981] LISP SECOND EDITION WINSTON, PATRICK
HORN, BERTHOLD KLAUS PAUL
1981 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
435 pp.

ARTICULOS

[Baud 1978] ON THE BRANCHING FACTOR OF THE ALFA-BETA PRUNNING ALGORITHM BAUDET, G.
1978 ARTIFICIAL INTELLIGENCE 27 pp. 173-199
NORTH-HOLLAND

[Darw 1983] A QUANTITATIVE ANALYSIS OF THE ALPHA-BETA PRUNNING ALGORITHM DARWISH, NEVIN M.
1983 ARTIFICIAL INTELLIGENCE 21 pp. 405-433
NORTH-HOLLAND

[Knut 1975] AN ANALYSIS OF ALFA-BETA PRUNNING ALGORITHM
KNUTH, D.
1975 ARTIFICIAL INTELLIGENCE 37 pp. 293-326
NORTH-HOLLAND

MANUALES

[Muli 1983] MICROSOFT muLISP ARTIFICIAL INTELLIGENCE DEVELOPMENT SYSTEM RICH, ALBERT D.
1983 THE SOFT WAREHOUSE
HONOLULU
140 pp.

CAPITULO 5

APLICACIONES A JUEGOS

5.1 CLASIFICACION DE JUEGOS

El sorprendente desarrollo de la computación en los últimos años, y la popularidad de las computadoras personales, han contribuido al nacimiento de miles de programas de aplicación dedicados a entretener al usuario. A pesar de que el carácter lúdico de este tipo de programas, puede peyorizarlos, no cabe la menor duda de que existen verdaderas obras de arte dentro del software de juegos. Es más, de hecho existe un estudio estadístico que muestra que la mayoría de las computadoras personales utilizadas fuera del ámbito empresarial, se dedican mayoritariamente a dos actividades principales: aprendizaje de programación y juegos; por lo tanto no hay que tomar tan a la ligera un juego, ya que no todos los juegos tienen el único propósito de entretener o divertir. En el mundo de los juegos no todo es superficialidad.

Los diferentes programas de juegos pueden clasificarse en cinco grandes grupos:

1.- Juegos didácticos

2.- Juegos de simulación de modelos reales

3.- Juegos de entretenimiento

4.- Juegos musicales

5.- Juegos "inteligentes"

JUEGOS DIDACTICOS

Una de las mejores formas de divulgar una disciplina para su aprendizaje, es presentarla en forma amena y divertida, con objeto de que el alumno no pierda interés en el estudio. El objetivo de estos juegos es que el usuario llegue a dominar la materia tratada. Existen varios programas de juegos en este grupo, en diversas áreas como: Historia, Matemáticas, Computación, etc..

JUEGOS DE SIMULACION DE MODELOS REALES

El objetivo de estos juegos es reproducir un modelo de la vida real; dentro de sus aplicaciones destacan los entrenamientos, los cuales evitan riesgos innecesarios de ser efectuadas las prácticas en el modelo real. Otra aplicación la constituye el observar un fenómeno y variar características

para poder pronósticar su comportamiento futuro. Por supuesto también sirven para pasar un buen rato.

JUEGOS DE ENTRETENIMIENTO

Como su nombre lo indica el único propósito de estos juegos, es divertir al usuario. Dentro de este grupo destacan los videojuegos, como los juegos de laberintos, guerras espaciales, etc.. A pesar de que fueron hechos tan solo para divertir, estudios recientes indican que avivan los reflejos del sistema locomotor humano.

JUEGOS MUSICALES

Este tipo de juegos permite realizar composiciones musicales al usuario y después oir su obra. También pueden instruir al usuario sobre los diferentes aspectos de la música.

JUEGOS "INTELIGENTES"

Este tipo de juegos simulan la inteligencia humana y compiten con el usuario haciendo que utilice su razonamiento y lógica para poder vencer. En esta clase de juegos se aplican técnicas heurísticas para que el programa sea capaz de tener un comportamiento "inteligente". Este tipo de juegos son de gran utilidad para comprender algunos procesos del pensamiento humano y además hacen que el usuario desarrolle su razonamiento lógico y su agilidad mental. Dentro de estos juegos cabe destacar al ajedrez, las damas, el gato, etc..

Este capítulo se aboca a la tarea de observar el comportamiento de los programas de algunos juegos "inteligentes", en su interacción con el usuario. Los juegos que se han elegido son tres: el gato, el cuatro en raya y la mente maestra.

5.2 GATO

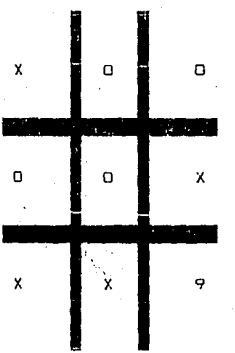
Este es de los juegos más populares y más sencillos de aprender. Las reglas del juego son las siguientes:

1.- Se juega en un tablero cuadrado con 9 casillas.

2.- La salida será indistinta, ya que no proporciona ventaja alguna.

3.- El jugador que primero tire, seleccionará una de las nueve casillas, que será marcada con un "O" o con una "X", aquel jugador que coloque tres "O" o tres "X" en forma horizontal, vertical o diagonal, habrá ganado el juego, aunque existe la posibilidad de empate. No esta permitido que un jugador tire dos veces seguidas.

La figura 5.1 muestra un tablero del juego, durante una confrontación del programa contra un usuario.



Numero de casilla? : 9

figura 5.1

Después de la observación del comportamiento con diversos usuarios, se ha llegado a la conclusión que el programa trabaja en forma eficiente y no puede ser derrotado por ningún usuario. Esto se debe a la sencillez del juego que no presenta un número muy grande de opciones, y a que un análisis del juego dío por resultado que no se debe tirar en ciertas casillas, por lo que no se tiene que analizar todas las opciones. Además la probabilidad de empate es muy alta, casi del 90 %.

En realidad el único defecto del programa es su lentitud para decidir, comparada con programas similares escritos en otros lenguajes, pero esto último se debe a esa característica del lenguaje LISP.

5.3 CUATRO EN RAYA

Este otro juego fué uno de los más novedosos en los últimos años de la decada de los setentas en nuestro país. Sigue una filosofía similar a la del gato. Las reglas del juego son las siguientes:

1.- Se juega en un tablero de 42 casillas, siete columnas por seis renglones.

2.- La salida es indistinta, ya que no proporciona ninguna ventaja.

3.- El jugador que primero tire seleccionará una de las siete columnas tirando con una "@" o con un "*", la jugada seleccionada quedará hasta abajo de la columna, si no hay una ficha anterior, o encima de la ficha anteriormente tirada. No esta permitido que un mismo jugador tire dos veces seguidas.

4.- El jugador que primero logre colocar cuatro de sus fichas en forma diagonal, vertical u horizontal, habrá ganado el juego, aunque existe la posibilidad de empate.

La figura 5.2 muestra el tablero del juego con la tirada inicial.

Este juego que parece fácil de programar, en realidad es todo lo contrario. Un análisis del juego más profundo, da como resultado la extremada complejidad para encontrar una adecuada función que guíe óptimamente la elección de la mejor jugada. A pesar de que aparentemente solo tiene que seleccionar una de las siete columnas, el problema radica en el análisis de todas las posibles combinaciones que puede recibir como respuesta a su tirada. Por ejemplo con una cota de profundidad de 1, analiza solo 7 jugadas, esto ocurre en la primera tirada. Pero con una cota de profundidad de 3 tendría que analizar 9,107 jugadas, creciendo este factor conforme avanza el juego. El problema principal como se explicaba anteriormente en el capítulo 4 es la explosión combinatoria. En resumen, el algoritmo alfa-beta es inadecuado para este juego, al igual que el compilador muLISP, pues es necesario contar con mayor memoria, lo que apenas daría un nivel aceptable de juego. Es claro que aquí se necesita un tipo de técnica heurística más

eficiente y compleja, la cual desgraciadamente escapa a los límites de este trabajo.

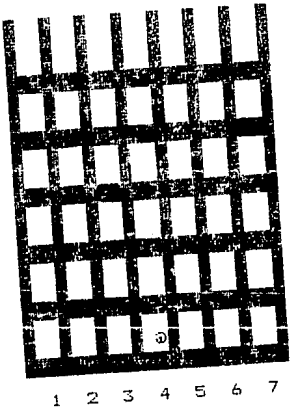


figura 5.2

5.4 MENTE MAESTRA

Este es un juego donde la lógica juega un papel muy importante. Las reglas del juego son las siguientes:

1.- El objetivo del juego consiste en adivinar un código de cuatro colores de un total de seis diferentes colores: azul, rojo, verde, blanco, amarillo y negro (AZU ROJ VER BLA AMA NEG).

2.- Un jugador pone el código, pudiendo repetir colores si así lo desea y el otro intentará adivinar la clave en el menor número de tiradas.

3.- El jugador que pone el código debera proporcionar cierta información al que trata de adivinarlo, dándole el número de colores puestos en posición correcta y el número de colores que pertenecen al código pero que están en posición errónea.

4.- Una vez que un jugador descifre el código, pasará a poner un código para que el otro jugador trate de adivinarlo. Ganará el jugador que adivine el código en el menor número de tiradas.

La figura 5.3 muestra la acción del programa durante la interacción con un usuario.

```
                --- MENTE MAESTRA ---

              Un juego donde la logica es clave

Da un numero entre 1 y 100:  3

Deseas A>divinar o P>oner la clave?  (P/A) P

                Movimiento: 1   ROJ  AZU  ROJ  AZU  1 0

                Movimiento: 2   ROJ  VER  BLA  AMA  1 0

                Movimiento: 3   ROJ  NEG  NEG  NEG  1 0

                Movimiento: 4   BLA  AZU  BLA  NEG  4 0

bravo, lo logre!  y solo me tomo:  4 movimientos.

Quieres jugar de nuevo?  (S/N)
```

figura 5.3

Este juego presenta una complejidad aparente, pues el total de permutaciones para un código es de 1,296. Sin embargo, contandose con cierta información la complejidad disminuye. El comportamiento del juego es bastante bueno. Esto se observó al cabo de varios (un total de diez) juegos con diferentes usuarios. En estos juegos sólo fué derrotado una vez y empato dos veces, aunque es claro que no es "invecible", pues de alguna manera el azar puede ayudar al usuario, si se puede decir que tiene un buen nivel de juego.

Para terminar con el presente trabajo, sólo resta exponer algunas observaciones y puntos de vista referentes a los tópicos tratados durante el desarrollo del mismo. Esto será hecho en las conclusiones de este trabajo.

BIBLIOGRAFIA DEL CAPITULO 5

LIBROS

[Rich 1983] ARTIFICIAL INTELLIGENCE RICH, ELAINE
INTERNATIONAL STUDENT EDITION
1983 McGRAW HILL BOOK CO. SINGAPURE
436 pp.

[Slag 1971] ARTIFICIAL INTELLIGENCE: THE HEURISTIC PROGRAMMING APPROACH SLAGLE, JAMES R.
1971 McGRAW HILL BOOK CO. NEW YORK
196 pp.

[Soft 1986] SOFTWARE CURSO PRACTICO DE PROGRAMACION
TOMO DOS
1986 EDICIONES NUEVA LENTE MEXICO
1040 pp.

[Stev 1981] ESTADISTICA PARA ADMINISTRACION Y ECONOMIA CONCEPTOS Y APLICACIONES STEVENSON, WILLIAM J.
1981 EDITORIAL HARLA MEXICO
585 pp.

[Pear 1984] HEURISTICS:INTELLIGENT SCARCH STRATEGIES FOR COMPUTER PROBLEM SOLVING PEAR, JUDEA
1984 ADDISON WESLEY PUBLISHING CO. INC. TEL AVIV
382 pp.

[Wins 1979] ARTIFICIAL INTELLIGENCE WINSTON, PATRICK
1979 ADDISON WESLEY PUBLISHING CO. INC.
MASSACHUSETS
444 pp.

ARTICULOS

[Gonz 1985] ¿ JUGAMOS GATO ? GONZALEZ, ERIKA
INFORMACION CIENTIFICA Y TECNOLOGICA VOL. 7
NUM. 109 INTELIGENCIA ARTIFICIAL
1985 CONACYT MEXICO
3 pp. 42-44

CONCLUSIONES

Los temas tratados en los capítulos anteriores, exponen un panorama general del estado actual de la Inteligencia Artificial, las principales técnicas heurísticas, así como un estudio del lenguaje de programación LISP y algunas aplicaciones a juegos. Las conclusiones emanadas de este trabajo a manera de epílogo son:

a) Aparentemente los juegos no representan grandes dificultades, sin embargo, como se observó en el juego cuatro en raya, cuando las opciones de juego se multiplican, sobre todo en el medio juego, se crea un problema muy difícil de resolver por no contar con un ambiente de programación más completo. La solución a este problema puede ser la utilización de técnicas de programación más complejas, como el uso de bases de conocimiento que sean capaces de "aprender", para evitar el tedioso y lento análisis de jugadas, o bien contar con una computadora VAX y los diálectos de LISP: INTERLISP o MACLISP, que es una configuración excelente para desarrollo de trabajos en IA.

El lenguaje LISP es muy adecuado para procesamiento simbólico, sin embargo, la lentitud de procesamiento así como las facilidades para programar, son dos aspectos que deberán mejorarse en los compiladores para microcomputadoras. En particular el muLISP debe contar con un editor propio más eficiente y no obligar al usuario a utilizar un editor externo para la realización de sus programas. Debe tambien mejorar su tiempo de respuesta en la evaluación de programas, y en la carga y lectura de los mismos.

En lo que se refiere a las técnicas heurísticas a pesar de sus limitaciones actuales, son de gran utilidad para mejorar algoritmos de búsqueda en estructuras de árbol. En un futuro cercano éstas técnicas serán mejoradas, con lo que se logrará un avance importante en el área.

b) Las técnicas heurísticas y su programación, no solo se enfocan a los juegos. Dentro del campo de análisis de confiabilidad y toma de decisiones, la búsqueda guiada heurísticamente a través de árboles de decisión es un tema relevante. Otra área de aplicación importante son los sistemas expertos, donde las técnicas vistas anteriormente constituyen la base para la construcción de los mismos.

Se puede decir en cierta forma, que la inteligencia es programable, aunque limitada a ciertos procesos como juegos, sistemas expertos, toma de decisiones, etc.. Actualmente no se ha logrado que una máquina encuentre soluciones brillantes a problemas para los que no ha sido programada, es decir carece de talento deductivo.

c) La Inteligencia Artificial esta en proceso de desarrollo y aunque se han obtenido logros importantes, todavia se esta lejos de alcanzar la construcción de una máquina con un nivel de inteligencia similar a la del ser humano.

La próxima decada estará llena de sorpresas por el proyecto japonés de la quinta generación de computadoras. Se busca la creación de sistemas que procesen conocimiento e información teniendo la función de resolver problemas de muy alto nivel. Se intenta dotar a éstos sistemas con interfases de funciones "inteligentes" capaces de entender el lenguaje natural (hablado y escrito), imágenes, etc., y con la capacidad de realizar inferencias y programación automática.

El impacto tecnológico en la sociedad de los avances actuales, así como de los próximos, revolucionará en forma notable el uso y manejo de la información, por lo que se deberá estar preparado para ello.

RECOMENDACIONES

Se escriben estas líneas con la finalidad de que futuros tesistas, aprovechen la experiencia del autor durante el desarrollo de esta tesis. Desde un punto de vista estrictamente personal, se sugiere seguir los siguientes pasos:

1.- Primeramente, si no se tiene tema de tesis, escoja una área de particular interés y que sea de total agrado. Lea material abundante acerca de esta área, con el fin de encontrar un tema especifico para desarrollar.

2.- Una vez seleccionado el tema, asegúrese de que en verdad lo puede desarrollar, no seleccione temas que estén fuera de su capacidad. No por ser un tema modesto deja de tener válidez y complejidad. Busque un asesor que conozca bien el área del conocimiento donde se ubica el tema seleccionado, y pídale que le sugiera bibliografía. Junte toda la bibliografía posible acerca del tema: Libros, artículos de revistas, reportes técnicos, periódicos, etc..

3.- Lea profundamente acerca del tema y observe con detenimiento cómo son presentados los temas en los libros. Esto último con el fin de dar una idea de cómo se podría desglosar el tema seleccionado en capítulos y subtemas. Analice el estilo con que son presentados los temas, y use esta experiencia para formar su propio estilo de redacción. Lea literatura de todo tipo para ampliar su vocabulario. Procure tocante a este último aspecto consultar diccionarios de la lengua y de sinónimos.

4.- Una vez bien estructurado el tema, comience a escribir sus ideas. Si puede inscribirse a un curso de redacción, mejor que mejor.

5.- Observe en los libros cómo se hacen las referencias bibliograficas, la presentación de figuras, tablas, índices, etc..

6.- Si tiene la facilidad de usar un procesador de palabras, aproveche sus ventajas: corrección rápida y fácil del texto, obtención del listado cuando es necesario, y se pueden tener varias copias para evitar accidentes o lamentables pérdidas.

7.- Aproveche al máximo los seminarios de tesis. En ellos se aclaran dudas, se exponen temas para adquirir soltura en la expresión oral y se dan muy acertados consejos y tips.

8.- Acuerde con su asesor fechas de entrega de avances, así como fechas en la que el asesor le entregue el material para hacerle las debidas correcciones. De esta forma tanto usted como él establecen un compromiso de trabajo.

9.- El cuerpo básico de una tesis debe incluir: carátula, dedicatorias y agradecimientos, índice, introducción, capitulado, conclusiones, anexos o apéndices (en caso de ser necesarios) y bibliografía consultada.

10.- Dedique todo el tiempo que le sea posible al desarrollo de su tesis, si puede escriba diario y no se desespere si su avance es lento. Cuando menos lo espere su trabajo comenzará a tener cierto cuerpo. Recuerde que un triunfo lleva a otros (en este caso un capítulo lleva a otros) y que hacer una tesis es un trabajo serio y responsable que de ninguna manera es fácil, pero tampoco es demasiado complicado.

APENDICE A

PROGRAMAS

```
% programa: GATO.LIB  TESIS: FUNDAMENTOS DE PROGRAMACION HEURISTICA
                              Y APLICACIONES  1987  %


(LOOP (PRIN1 (QUOTE *)) (EVAL (READ)) ((NULL RDS)) )
(PUTD (QUOTE DEFUN) (QUOTE (NLAMBDA (NAM$ EXP$)
  (PUTD NAM$ EXP$)
  NAM$ )))

(DEFUN SETQQ (NLAMBDA (NAM$ EXP$)
  (SET NAM$ EXP$)
  NAM$ ))

(DEFUN GATO (LAMBDA ()
  (SETQ RDS)
  (CLRSCRN)
  (TERPRI 7)
  (SPACES (QUOTIENT (DIFFERENCE (LINELENGTH) 20) 2))
  (PRTSENT (QUOTE (*** " " G A T O " " ***)) 2)
  (SPACES (QUOTIENT (DIFFERENCE (LINELENGTH) 31) 2))
  (PRTSENT (QUOTE (Intenta vencer a la computadora)) 3)
  (TERPRI 2)
  (SETQ CONT 0) (SETQ F 0)
  (SETQ LISTA (QUOTE (1 2 3 4 5 6 7 8 9)))
  (LOOP
    ( ((QUERY (QUOTE (Quieres jugar primero "?"))
              (QUOTE S) (QUOTE N) )
       (TERPRI)
       (TABLERO)
       (TIRAHUMANO) )
      (TERPRI) (TABLERO)
      (TIRAMAQUINA) )
    (TERPRI)
    ((NULL (QUERY (QUOTE (Quieres jugar de nuevo "?"))
           (QUOTE S) (QUOTE N))))
    (TERPRI 2) ) ))


(DEFUN QUERY (LAMBDA (TEXT YES NO READCH CHAR)
  (PRTSENT TEXT)
  (PRIN1 (QUOTE "("))
  (PRIN1 YES) (PRIN1 (QUOTE "/"))
  (PRIN1 NO) (PRIN1 (QUOTE ") "))
  (LOOP
    (SETQ CHAR (READCH))
    ((EQ CHAR YES)
      (PRINT CHAR) )
    ((EQ CHAR NO)
      (PRINT CHAR)
      NIL )
```

```
    (PRIN1 (ASCII 7)) ) ))


(DEFUN TABLERO (LAMBDA ()
(CLRSCRN)
  (SETQ DOT (ASCII 219))
  (DRAW (LINE -3 9 -3 -9) (LINE 3 9 3 -9) (LINE -9 3 9 3)
        (LINE -9 -3 9 -3)  )
        (CURSOR 6 32) (PRIN1 1)
        (CURSOR 6 40) (PRIN1 2)
        (CURSOR 6 48) (PRIN1 3)
        (CURSOR 12 32)(PRIN1 4)
        (CURSOR 12 40)(PRIN1 5)
        (CURSOR 12 48)(PRIN1 6)
        (CURSOR 18 32)(PRIN1 7)
        (CURSOR 18 40)(PRIN1 8)
        (CURSOR 18 48)(PRIN1 9)    ))


(DEFUN TIRAHUMANO (LAMBDA (NUMERO)
(ADD1 CONT)
(CURSOR 23 10)
(LOOP
(PRTSENT (QUOTE (Numero de casilla  "?" :)))
((PLUSP (SETQ NUMERO (RATOM) ))) (TERPRI) )
   ((EQ CONT 5 ) T)
   ((EQ NUMERO 1) ((CURSOR 6 32)    (PRIN1 O)
                   (CURSOR 12 40)  (PRIN1 X)   (TIRAHUMANO) ) )
   ((EQ NUMERO 2) ((CURSOR 6 40)    (PRIN1 O)
                   (CURSOR 18 40)  (PRIN1 X)  (TIRAHUMANO) ) )
   ((EQ NUMERO 3) ((CURSOR 6 48)    (PRIN1 O)
                   (CURSOR 18 32)  (PRIN1 X)  (TIRAHUMANO) ) )
   ((EQ NUMERO 4) ((CURSOR 12 32)   (PRIN1 O)
                   (CURSOR 12 48)  (PRIN1 X)  (TIRAHUMANO) ) )
   ((EQ NUMERO 5) ((CURSOR 12 40)   (PRIN1 O)
                   (CURSOR 6 32)   (PRIN1 X)  (TIRAHUMANO) ) )
   ((EQ NUMERO 6) ((CURSOR 12 48)   (PRIN1 O)
                   (CURSOR 12 32)  (PRIN1 X)  (TIRAHUMANO) ) )
   ((EQ NUMERO 7) ((CURSOR 18 32)   (PRIN1 O)
                   (CURSOR 6 48)   (PRIN1 X)   (TIRAHUMANO) ) )
   ((EQ NUMERO 8) ((CURSOR 18 40)   (PRIN1 O)
                   (CURSOR 6 40)   (PRIN1 X)  (TIRAHUMANO) ) )
   ((EQ NUMERO 9) ((CURSOR 18 48)   (PRIN1 O)
                   (CURSOR 12 48)  (PRIN1 X)  (TIRAHUMANO) ) )
                                                              ))


(DEFUN CHECATIRADA (LAMBDA (NUMERO LISTA)
(MIEMBRO) ))

(DEFUN MIEMBRO (LAMBDA (NUMERO LISTA)
(NIL  (REMBER1 NUMERO LISTA)
      (PRTSENT (QUOTE (jugada ilegal !) ))
      (SYSTEM) ) ))
```

```
(DEFUN TIRAHUMANO1 (LAMBDA (NUMERO)
(ADD1 CONT)
(CURSOR 23 10)
(LOOP
(PRTSENT (QUOTE (Numero de casilla  "?" :)))
((PLUSP (SETQ NUMERO (RATOM) ))) (TERPRI) )
   ((EQ NUMERO 1) ((CURSOR 6 32)    (PRIN1 O)  ))
   ((EQ NUMERO 2) ((CURSOR 6 40)    (PRIN1 O)  ))
   ((EQ NUMERO 3) ((CURSOR 6 48)    (PRIN1 O)  ))
   ((EQ NUMERO 4) ((CURSOR 12 32)   (PRIN1 O)  ))
   ((EQ NUMERO 5) ((CURSOR 12 40)   (PRIN1 O)  ))
   ((EQ NUMERO 6) ((CURSOR 12 48)   (PRIN1 O)  ))
   ((EQ NUMERO 7) ((CURSOR 18 32)   (PRIN1 O)  ))
   ((EQ NUMERO 8) ((CURSOR 18 40)   (PRIN1 O)  ))
   ((EQ NUMERO 9) ((CURSOR 18 48)   (PRIN1 O)  ))
   ((EQ CONT 5) NIL)    ))


(DEFUN TIRAMAQUINA (LAMBDA (NUMERO)
(SETQ F 1)
(CURSOR 12 40) (PRIN1 X)
(TIRAHUMANO1)
  (EQ NUMERO 1) ( (CURSOR 6 40) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 12 32) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 18 48) (PRIN1 X)   )

  (EQ NUMERO 2) ( (CURSOR 12 48) (PRIN1 X)
                (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                (TIRAHUMANO1) (NIL (MEMBER 7) (CURSOR 18 48)
                                               (PRIN1 X) )
                              (CURSOR 18 32) (PRIN1 X)
                  (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !)))
                  )

  (EQ NUMERO 3) ( (CURSOR 6 40) (PRIN1 X)
                (TIRAHUMANO1) (CURSOR 12 32) (PRIN1 X)
                (TIRAHUMANO1) (CURSOR 18 48) (PRIN1 X)
                (TIRAHUMANO1) (CURSOR 18 32) (PRIN1 X)   )

  (EQ NUMERO 4) ( (CURSOR 6 40) (PRIN1 X)
                (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                (TIRAHUMANO1) ( NIL (MEMBER 1 LISTA) (CURSOR 18 32)
                                    (PRIN1 X)  )
                              (CURSOR 6 32) (PRIN1 X)
                  (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !)))
                  )

   (EQ NUMERO 6) ( (CURSOR 6 40) (PRIN1 X)
                (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                (TIRAHUMANO1) ( NIL (MEMBER 1 LISTA) (CURSOR 18 32)
                                    (PRIN1 X)  )
                              (CURSOR 6 32) (PRIN1 X)
                  (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !)))
```

```
(DEFUN TIRAHUMANO1 (LAMBDA (NUMERO)
(ADD1 CONT)
(CURSOR 23 10)
(LOOP
(PRTSENT (QUOTE (Numero de casilla  "?" :)))
((PLUSP (SETQ NUMERO (RATOM) ))) (TERPRI) )
   ((EQ NUMERO 1) ((CURSOR 6 32)    (PRIN1 O)   ))
   ((EQ NUMERO 2) ((CURSOR 6 40)    (PRIN1 O)   ))
   ((EQ NUMERO 3) ((CURSOR 6 48)    (PRIN1 O)   ))
   ((EQ NUMERO 4) ((CURSOR 12 32)   (PRIN1 O)   ))
   ((EQ NUMERO 5) ((CURSOR 12 40)   (PRIN1 O)   ))
   ((EQ NUMERO 6) ((CURSOR 12 48)   (PRIN1 O)   ))
   ((EQ NUMERO 7) ((CURSOR 18 32)   (PRIN1 O)   ))
   ((EQ NUMERO 8) ((CURSOR 18 40)   (PRIN1 O)   ))
   ((EQ NUMERO 9) ((CURSOR 18 48)   (PRIN1 O)   ))
   ((EQ CONT 5) NIL)    ))


(DEFUN TIRAMAQUINA (LAMBDA (NUMERO)
(SETQ F 1)
(CURSOR 12 40) (PRIN1 X)
(TIRAHUMANO1)
  (EQ NUMERO 1) ( (CURSOR 6 40) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 12 32) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 18 48) (PRIN1 X)  )

  (EQ NUMERO 2) ( (CURSOR 12 48) (PRIN1 X)
                 (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                 (TIRAHUMANO1) (NIL (MEMBER 7) (CURSOR 18 48)
                                                (PRIN1 X) )
                               (CURSOR 18 32) (PRIN1 X)
                   (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !)))
                    )

  (EQ NUMERO 3) ( (CURSOR 6 40) (PRIN1 X)
                 (TIRAHUMANO1) (CURSOR 12 32) (PRIN1 X)
                 (TIRAHUMANO1) (CURSOR 18 48) (PRIN1 X)
                 (TIRAHUMANO1) (CURSOR 18 32) (PRIN1 X)  )

  (EQ NUMERO 4) ( (CURSOR 6 40) (PRIN1 X)
                 (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                 (TIRAHUMANO1) ( NIL (MEMBER 1 LISTA) (CURSOR 18 32)
                                     (PRIN1 X)  )
                               (CURSOR 6 32) (PRIN1 X)
                   (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !)))
                    )

  (EQ NUMERO 6) ( (CURSOR 6 40) (PRIN1 X)
                 (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                 (TIRAHUMANO1) ( NIL (MEMBER 1 LISTA) (CURSOR 18 32)
                                     (PRIN1 X)  )
                               (CURSOR 6 32) (PRIN1 X)
                   (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !)))
```

```
)

  (EQ NUMERO 7) ( (CURSOR 6 40) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 18 48) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 12 32) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)  )

  (EQ NUMERO 8) ( (CURSOR 12 48) (PRIN1 X)
                  (TIRAHUMANO1) (CURSOR 6 48) (PRIN1 X)
                  (TIRAHUMANO1) ( NIL (MEMBER 7 LISTA) (CURSOR 18 48)
                                      (PRIN1 X)  )
                                (CURSOR 18 32) (PRIN1 X)
                    (CURSOR 24 1) (PRTSENT (QUOTE (te he vencido !))2)
                    )      ))


(DEFUN ADD1 (LAMBDA (X)
            (PLUS X 1) ))



(DEFUN REMBER1 LAMBDA (NUMERO LISTA)
  ((NULL LISTA) NIL)
  ((EQ NUMERO (CAR LISTA)) (CDR LISTA))
  (CONS (CAR LISTA) (REMBER1 NUMERO (CDR LISTA))) ))


(DEFUN PRETERPRI (LAMBDA (NUM)
  ((GREATERP (PLUS (SPACES) NUM) (LINELENGTH))
    (TERPRI) ) ))

(DEFUN PRTSENT (LAMBDA (LST NUM)
  (LOOP
    ((NULL LST))
    ((NULL (CDR LST))
      (PRIN1 (CAR LST)) )
    ( ((OR
         (MEMBER (CADR LST) TRMLIS)
         (MEMBER (CADR LST) PCTLIS) )
       (PRETERPRI (ADD1 (LENGTH (CAR LST))))
       (PRIN1 (POP LST)) )
      ((MEMBER (CADR LST) SEPLIS)
        ((NULL (CDDR LST))
          (PRETERPRI (ADD1 (LENGTH (CAR LST))))
          (PRIN1 (POP LST)) )
        (PRETERPRI (PLUS (ADD1 (LENGTH (CAR LST))) (LENGTH (CADDR LST)))

        (PRIN1 (POP LST))
        (PRIN1 (POP LST)) ) )
    (PRIN1 (CAR LST))
    ((NULL (CDR LST)))
    ( ((MEMBER (POP LST) TRMLIS)
        ((EQ (ADD1 (SPACES)) (LINELENGTH))
          (TERPRI) )
        (SPACES 2) )
      ((EQ (SPACES) (LINELENGTH))
```

```
        (TERPRI) )
      (SPACES 1) ) )
  ((PLUSP NUM)
    (TERPRI NUM) )
  ((ED (ADD1 (SPACES)) (LINELENGTH))
    (TERPRI) )
  (SPACES 2) ))

(SETQQ TRMLIS ("." "!" "?"))

(SETQQ PCTLIS ("," ";" ")" "(" ":"))

(SETQQ SEPLIS ("?" "-"))

(GATO (RDS))



% programa: MENTE.LIB  TESIS: FUNDAMENTOS DE PROGRAMACION HEURISTICA
                              Y APLICACIONES  1987  %

(LOOP (PRIN1 (QUOTE *)) (EVAL (READ)) ((NULL RDS)) )

(PUTD (QUOTE DEFUN) (QUOTE (NLAMBDA (NAM$ EXP$)
  (PUTD NAM$ EXP$)
  NAM$ )))

(DEFUN SETQQ (NLAMBDA (NAM$ EXP$)
  (SET NAM$ EXP$)
  NAM$ ))


(CLRSCRN)
(DEFUN MENTE (LAMBDA (KEYLIST HELPMK HELPBR SEED)
  (SETQ RDS)
  (TERPRI 5)
  (CLRSCRN)
  (SPACES (QUOTIENT (DIFFERENCE (LINELENGTH) 20) 2))
  (PRTSENT (QUOTE (--- " " MENTE MAESTRA " " ---)) 2)
  (SPACES (QUOTIENT (DIFFERENCE (LINELENGTH) 31) 2))
  (PRTSENT (QUOTE (Un juego donde la logica es clave)) 3)
  (SETQ KEYLIST (QUOTE (
    (AZU VER BLA AMA ROJ NEG)
    (ROJ AMA VER NEG AZU BLA)
    (NEG AZU AMA VER BLA ROJ)
    (AMA BLA ROJ NEG AZU VER)
      )))
  (LOOP
    (PRTSENT (QUOTE (Da un numero entre 1 y 100 :)))
    ((PLUSP (SETQ SEED (RATOM))))
```

```
    (TERPRI) )
  (TERPRI 1)
  (LOOP
    ( ((QUERY (QUOTE (Deseas  A>divinar o  P>oner la clave "?"))
            (QUOTE P) (QUOTE A))
        (TERPRI)
        (HELPMK)
        (CODEBREAKER) )
      (TERPRI)
      (HELPBR)
      (CODEMAKER) )
    (TERPRI)
    ((NULL (QUERY (QUOTE ( Quieres jugar de nuevo "?"))
            (QUOTE S) (QUOTE N))))
    (TERPRI 2) (CLRSCRN) )))

(DEFUN QUERY (LAMBDA (TEXT YES NO READCH CHAR)
  (PRTSENT TEXT)
  (PRIN1 (QUOTE "("))
  (PRIN1 YES) (PRIN1 (QUOTE "/"))
  (PRIN1 NO) (PRIN1 (QUOTE ") "))
  (LOOP
    (SETQ CHAR (READCH))
    ((EQ CHAR YES)
      (PRINT CHAR) )
    ((EQ CHAR NO)
      (PRINT CHAR)
      NIL )
    (PRIN1 (ASCII 7)) ) ))



 (DEFUN HELPMK (LAMBDA ()
  ((EVAL HELPMK))
  (SETQ HELPMK T))))

(DEFUN HELPBR (LAMBDA (CTR)
  ((EVAL HELPBR))
  (SETQ HELPBR T)
  (SETQ CTR 900)
  (LOOP
    ((ZEROP CTR))
    ( ((ZEROP (REMAINDER CTR 150)) (PRIN1 (QUOTE ". "))) )
    (SETQ CTR (SUB1 CTR)) )
  (TERPRI 2)
  (TERPRI) ))


(DEFUN CODEBREAKER (LAMBDA (MOVE GRAPH CTR KEYLST)
  (SETQ KEYLIST (MAPLIST KEYLIST (QUOTE PERMUTE)))
  (SETQ MOVE (FRSTMOV (CAR KEYLIST)))
  (SETQ GRAPH (NUGRAPH MOVE))
  (SETQ CTR 1)
  (LOOP
    (PRTMOVE MOVE)
```

```
    ((EQ (RATOM) 4)
      (RATOM) (TERPRI)
      (PRTSENT (QUOTE (bravo "." lo logre ! y solo me tomo:)))
      (PRIN1 CTR)
      (PRINT (QUOTE " movimientos.")) )
    (SETQ GRAPH (MKGRAPH MOVE RATOM (RATOM) MOVE NIL GRAPH KEYLIST))
    (SETQ CTR (ADD1 CTR))
    (SETQ ERROR NIL)
    (SETQ KEYLST KEYLIST)
    (SETQ MOVE (MKMOVE GRAPH (CAR KEYLST) (SETQ KEYLST (CDR KEYLST))))
    (TERPRI)
    ((EVAL ERROR)
      (PRTSENT (QUOTE ( Has cometido un error o quieres
        hacer trampa "." )) 1) ) ) ))

(DEFUN MAPLIST (LAMBDA (LST FUN)
  ((NULL LST) NIL)
  (CONS (FUN (CAR LST)) (MAPLIST (CDR LST) FUN)) ))

(DEFUN PERMUTE (LAMBDA (LST1 LST2 LST3)
  ((NULL LST1)
    (NCONC LST2 LST3) )
  ((NULL (CDR LST1))
    (NCONC (CONS (CAR LST1) LST3) LST2) )
  ((NULL (CDDR LST1))
    (NCONC (PERMUTE (CONS (CAR LST1) LST2))
           (PERMUTE (CONS (CADR LST1) LST3))) )
  (PERMUTE (CDDDR LST1) (CONS (CADR LST1) LST3)
         (CONS (CADDR LST1) (CONS (CAR LST1) LST2))) ))

(DEFUN FRSTMOV (LAMBDA (KEY)
  (LIST (CAR KEY) (CADDR KEY) (CAR KEY) (CADDR KEY)) ))

(DEFUN NUGRAPH (LAMBDA (MOVE)
  ((NULL MOVE) T)
  (NUROW (NUGRAPH (CDR MOVE)) (CAR KEYLIST)) ))

(DEFUN NUROW (LAMBDA (GRAPH KEY)
  ((NULL KEY) NIL)
  (CONS GRAPH (NUROW GRAPH (CDR KEY))) ))

(DEFUN PRTMOVE (LAMBDA (MOVE)
  (SPACES (QUOTIENT (DIFFERENCE (LINELENGTH) 32) 2))
  (PRIN1 (QUOTE "Movimiento: ")) (PRIN1 CTR) (SPACES 3)
  (LOOP
    ((NULL MOVE))
    (PRIN1 (CAR MOVE))
    (SPACES 2)
    (SETQ MOVE (CDR MOVE)) ) ))

(DEFUN MKMOVE (LAMBDA (GRAPH KEY)
  (LOOP
    ((NULL GRAPH)
      (SETQ ERROR T) NIL)
    ((CAR GRAPH)
```

```
      ((ATOM (CAR GRAPH))
        (LIST (CAR KEY)) )
       (CONS (CAR KEY) (MKMOVE (CAR GRAPH) (CAR KEYLST)
         (SETQ KEYLST (CDR KEYLST)))) )
     (SETQ GRAPH (CDR GRAPH))
     (SETQ KEY (CDR KEY)) ) ))

(DEFUN MKGRAPH (LAMBDA (MOVE BLACKS WHITES FREE UNUSED GRAPH KEYLST)
  ((NULL MOVE)
    ((ZEROP BLACKS)
      (EQ WHITES (INCOMMON FREE UNUSED))) )
  (MKNODE GRAPH (CAR KEYLST)) ))

(DEFUN MKNODE (LAMBDA (GRAPH KEY)
  ((NULL KEY) NIL)
  ((NULL (CAR GRAPH))
    (CONSNIL (MKNODE (CDR GRAPH) (CDR KEY))) )
  ((EQ (CAR MOVE) (CAR KEY))
    ((PLUSP BLACKS)
      (CONSNULL (MKGRAPH (CDR MOVE) (SUB1 BLACKS) WHITES
        (REMBER1 (CAR KEY) FREE) UNUSED (CAR GRAPH) (CDR KEYLST))
          (MKNODE (CDR GRAPH) (CDR KEY))) )
    (CONSNIL (MKNODE (CDR GRAPH) (CDR KEY))) )
  (CONSNULL (MKGRAPH (CDR MOVE) BLACKS WHITES FREE
    (CONS (CAR KEY) UNUSED) (CAR GRAPH) (CDR KEYLST))
      (MKNODE (CDR GRAPH) (CDR KEY))) ))


(DEFUN CODEMAKER (LAMBDA (CODE MOVE CTR)
  (SETQ KEYLIST (MAPLIST KEYLIST (QUOTE PERMUTE)))
  (SETQ CODE (MKCODE KEYLIST))
  (SETQ CTR 1)
  (LOOP
    (SPACES (QUOTIENT (DIFFERENCE (LINELENGTH) 32) 2))
    (PRIN1 (QUOTE "Movimiento: ")) (PRIN1 CTR) (SPACES 3)
    ((CODEMATCH CODE (READMOVE CODE) 0)
      (TERPRI)
      (PRIN1 (QUOTE "Te tomo:  "))
      (PRIN1 CTR)
      ( ((EQ CTR 1)
          (PRINT (QUOTE " movimiento.")) )
        (PRINT (QUOTE " movimientos.")) )
      ((LESSP CTR 6)
        (PRTSENT (QUOTE (Eres bueno "," juguemos otra vez !)) 1) )
      ((LESSP CTR 8)
        (PRTSENT (QUOTE (Eso fue regular "," intenta mejorar
          tu record )) 1) )
      (PRTSENT (QUOTE (Mejor toma un descanso y vuelve a intentarlo)) 1)

    (TERPRI)
    (SETQ CTR (ADD1 CTR)) ) ))

(DEFUN MKCODE (LAMBDA (KEYLST)
  ((NULL KEYLST) NIL)
  (CONS (NTHCAR (CAR KEYLST) (PLUS (RANDOM) 1))
    (MKCODE (CDR KEYLST))) ))
```

```
(DEFUN CODEMATCH (LAMBDA (CODE1 MOVE1 BLACKS CODE2 MOVE2)
  ((NULL CODE1)
    (SPACES (PLUS (QUOTIENT (DIFFERENCE (LINELENGTH) 32) 2) 28))
    ((EQ (PRIN1 BLACKS) 4))
    (SPACES 2)
    (SAMETYPE CODE2 MOVE2 0)
    NIL )
  ((EQ (CAR CODE1) (CAR MOVE1))
    (CODEMATCH (CDR CODE1) (CDR MOVE1) (ADD1 BLACKS) CODE2 MOVE2) )
  (CODEMATCH (CDR CODE1) (CDR MOVE1) BLACKS (CONS (CAR CODE1) CODE2)
    (CONS (CAR MOVE1) MOVE2)) ))

(DEFUN SAMETYPE (LAMBDA (CODE MOVE WHITES)
  ((NULL CODE)
    (PRINT WHITES) )
  ((MEMBER (CAR CODE) MOVE)
    (SAMETYPE (CDR CODE) (REMBER1 (CAR CODE) MOVE) (ADD1 WHITES)) )
  (SAMETYPE (CDR CODE) MOVE WHITES) ))

(DEFUN READMOVE (LAMBDA (CODE)
  ((NULL CODE) NIL)
  (CONS (RATOM) (READMOVE (CDR CODE))) ))

(DEFUN ADD1 (LAMBDA (X)
  (PLUS X 1) ))

(DEFUN SUB1 (LAMBDA (X)
  (DIFFERENCE X 1) ))

(DEFUN REMBER1 (LAMBDA (X L)
  ((NULL L) NIL)
  ((EQ X (CAR L)) (CDR L))
  (CONS (CAR L) (REMBER1 X (CDR L))) ))

(DEFUN REPLACE (LAMBDA (X Y L)
  ((NULL L) NIL)
  ((EQ X (CAR L))
    (CONS Y (REPLACE X Y (CDR L))) )
  (CONS (CAR L) (REPLACE X Y (CDR L))) ))

(DEFUN NTHCAR (LAMBDA (L N)
  ((NOT (PLUSP N)) NIL)
  (LOOP
    ((EQ N 1) (CAR L))
    (SETQ N (DIFFERENCE N 1))
    (SETQ L (CDR L)) ) ))

(DEFUN RANDOM (LAMBDA ()
  (SETQ SEED (REMAINDER (PLUS 2113233 (TIMES SEED 271821)) 9999991))
  (REMAINDER SEED 4) ))

(DEFUN CONSNIL (LAMBDA (X)
  ((NULL X) NIL)
  (CONS NIL X) ))
```

```
(DEFUN CONSNULL (LAMBDA (X Y)
  ((NULL X)
    (CONSNIL Y) )
  (CONS X Y) ))

(DEFUN INCOMMON (LAMBDA (LST1 LST2)
  (SETQ TOT 0)
  (LOOP
    ((OR (NULL LST1) (NULL LST2)) TOT)
    ( ((MEMBER (CAR LST1) LST2)
        (SETQ LST2 (REMBER1 (CAR LST1) LST2))
        (SETQ TOT (PLUS TOT 1)) ) )
    (SETQ LST1 (CDR LST1)) ) ))

(DEFUN PRTSENT (LAMBDA (LST NUM)
  (LOOP
    ((NULL LST))
    ((NULL (CDR LST))
      (PRIN1 (CAR LST)) )
    ( ((OR
          (MEMBER (CADR LST) TRMLIS)
          (MEMBER (CADR LST) PCTLIS) )
        (PRETERPRI (ADD1 (LENGTH (CAR LST))))
        (PRIN1 (POP LST)) )
      ((MEMBER (CADR LST) SEPLIS)
        ((NULL (CDDR LST))
          (PRETERPRI (ADD1 (LENGTH (CAR LST))))
          (PRIN1 (POP LST)) )
        (PRETERPRI (PLUS (ADD1 (LENGTH (CAR LST))) (LENGTH (CADDR LST)))
        (PRIN1 (POP LST))
        (PRIN1 (POP LST)) ) )
    (PRIN1 (CAR LST))
    ((NULL (CDR LST)))
    ( ((MEMBER (POP LST) TRMLIS)
        ((EQ (ADD1 (SPACES)) (LINELENGTH))
          (TERPRI) )
        (SPACES 2) )
      ((EQ (SPACES) (LINELENGTH))
        (TERPRI) )
      (SPACES 1) ) )
  ((PLUSP NUM)
    (TERPRI NUM) )
  ((EQ (ADD1 (SPACES)) (LINELENGTH))
    (TERPRI) )
  (SPACES 2) ))

(DEFUN PRETERPRI (LAMBDA (NUM)
  ((GREATERP (PLUS (SPACES) NUM) (LINELENGTH))
    (TERPRI) ) ))


(SETQQ TRMLIS ("." "!" "?"))

(SETQQ PCTLIS ("," ";" ")" "(" ":"))
```

```
(SETQQ SEPLIS ("." "-"))
(MENTE (RDS))
```

APENDICE B

FUENTES PARA MAYOR INFORMACION

1.- REVISTAS

* SIGART Newsletter - ACM (Asociation for Computing Machinery)

* Artificial Intelligence

* Cognitive Science - Cognitive Science Society

* AI Magazine - American Association for AI (AAAI)

* Pattern Analysis and Machine Intelligence - IEEE

* International Journal on Robotics Research

* IEEE Transactions on Systems, Man and Cybernetics

2.- CONFERENCIAS

* International Joint Conference on AI (IJCAI) - biannual

* AAAI Annual Conference

3.- LIBROS DE RECIENTE PUBLICACION

* Barr, A. and Feigenbaum, E.A., The Handbook of Artificial Intelligence, Vols. I, II, Los Altos, CA, W. Kaufmann, 1981, 1982.

* Clocksin, W.F. and Mellish, C.S., Programming in PROLOG, New York, Spinger - Verlag, 1981.

* Cohen, P.R. and Feingenbaum, E.A., (Eds.), The Handbook of Artificial Inteligence, Vol. III, Los Altos, CA, W. Kaufmann, 1982.

* Davis, R., and Lenat, D.B., Knowledge - Based Systems in Artificial Intelligence, New York, McGraw-Hill, 1982.

* Feingenbaum, E.A. and McCorduck, P., The Fifth Generation, Reading, Mass, Addison-Wesley, 1983.

* Hayes-Root, F. (Ed.), Building Expert Systems, Reading, Mass, Addison-Wesley, 1983.

* Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds.), Machine Learning-An Artificial Intelligence Approach, Palo Alto, Tioga, 1983.

* Nilsson, M.J., Principles of Artificial Intelligence, Palo Alto, CA, Tioga, 1980.

* Rich, E., Artificial Intelligence, New York, McGraw-Hill, 1983.

* Simon, H.A., The Sciences of The Artificial, 2nd Ed., Cambridge, Mass, MIT press, 1981.

* Sowa, J.F., Conceptual Structures, Information Processing in Mind and Machine, Reading, Mass, Addison-Wesley, 1983.

* Szolovits, P. (Ed.), Artificial Intelligence in Medicine, Boulder, CO, Westview press, 1982.

* Wilensky, R., Planning and Understanding, Reading, MA, Addison Wesley, 1982.

* Winston, P.H. and Horn, B.K.P., LISP, Reading, Mass, Addison-Wesley, 1982.

G L O S A R I O

A

Algoritmo: Conjunto de reglas para resolver un problema en un número finito de pasos.

Ambiente: Estado actual de un programa en la computadora.

Ambiente de Programación: Conjunto total de programación que incluye la Interface, los lenguajes, el editor y otras herramientas de programación.

Ambiente Interactivo: Sistema Computacional en el cual el usuario interactúa (diáloga) con el sistema (en tiempo real) durante el proceso de desarrollo o corrida de un programa.

Análisis: Estudio cuantitativo y cualitativo de un hecho.

Antecedente: Lado izquierdo de una regla de producción.

Aplicación Funcional: Tarea genérica desarrollada durante una aplicación.

Arbol: Gráfica dirigida que representa el estado de un problema.

Arbol AND/OR: Arbol que representa un estado en un juego, es llamado AND/OR por la expansión conjuntiva y la expansión disyuntiva en las diferentes opciones del juego.

Archivo: Conjunto de registros. Bloque de datos grabados en un dispositivo de almacenamiento, bajo un nombre dado.

Argumento: Expresión cuyo valor es usado por una función para calcular un resultado. Cuando una función es llamada, el argumento es evaluado y el valor es pasado a la función.

Aritmética Racional: Estudio de las aplicaciones de todos los números que puedan ser expresados como el radio de dos enteros. Incluye la aritmética entera como un subconjunto.

Arquitectura Computacional: Forma en que los elementos de una computadora están conectados para realizar una función específica.

Atomo: Elemento individual indivisible. Proposición lógica que no puede ser dividida en otras proposiciones.

Autonomo: Sistema con capacidad de acción independiente.

ASCII: (American Standard Code for Information Interchange). Código Estándard Americano para Intercambio de Información. Es el código estándard para la transmisión, procesamiento y

grabación de la información. Las letras del alfabeto, los dígitos decimales, los signos de puntuación y otros caracteres especiales son representados por un único código de siete bits.

B

Bactracking: Retroceso a un nodo visto con anterioridad, para expanderlo en otra dirección durante una búsqueda.

Base de Conocimiento: Base de datos en Inteligencia Artificial que no necesariamente consiste de archivos de datos uniformes. Se forma generalmente de hechos, inferencias y procedimientos, correspondientes al tipo de información necesaria para una posible solución a un problema.

Base de Datos: Colección ordenada de datos de algun dominio.

Binario: Condición en la cual hay sólo dos alternativas. Sistema Binario (base dos) el cual consta de dos digitos 1 y 0.

Bit: Dígito binario. Unidad mínima de procesamiento de datos. Sin embargo, un conjunto de bits puede representar cualquier estructura de datos.

Bootsrap Loader: Programa generalmente alojado en la memoria ROM que es el encargado de cargar el sistema operativo.

Búsqueda Ciega: Enfoque en el cual la búsqueda se realiza sin tener conocimiento alguno.

Búsqueda en Profundidad: Algoritmo de búsqueda, en el cual se profundiza en una dirección de acuerdo a una cota de profundidad.

Búsqueda Horizontal: Algoritmo de búsqueda, en el cual la búsqueda se realiza a todo lo ancho del árbol.

Byte: Unidad de procesamiento de datos, generalmente compuesta de 8 bits.

C

Caracteres de Control: Conjunto de caracteres en código ASCII usados para direcciones de control de comunicación con los dispositivos.

Cláusula: Construcción sintáctica formada de una sentencia y un predicado, forma parte de una declaración lógica o de una oración en gramática.

Cláusula de Horn: Conjunto de declaraciones unidas por operadores lógicos AND. Usada en PROLOG.

Código de Máquina: Conjunto de instrucciones que son directamente ejecutadas por un procesador o microprocesador.

Cognición: Proceso intelectual mediante el cual el conocimiento es obtenido por percepción o ideas.

Compilar: Procedimiento mediante el cual se traduce un programa fuente escrito en un lenguaje de alto nivel a código de máquina.

Concurrencia: Propiedad de los sistemas multiusuario, que permite trabajar a varios usuarios casi simultaneamente.

Conectivos: Declaración en lógica compuesta de elementos unidos por operadores (por ejemplo AND, OR) cuyo valor depende del de sus elementos.

Conjunción: Problema compuesto de varios subproblemas. Fórmula lógica construida por conectivos.

Consecuente: Lado derecho de una regla de producción.

Contexto: Conjunto de circunstancias o hechos que definen una situación particular.

Cursor: Símbolo que aparece en una pantalla generalmente parpadeando, y cuya posición indica el lugar donde será desplegado el próximo carácter.

D

Debugging: Correción de errores en un plan. Depuración de errores en un programa.

Deducción: Proceso de razonamiento en el cual la conclusión se obtiene de premisas dadas.

Deducción Natural: Razonamiento informal.

Dependencia Conceptual: Enfoque en procesamiento de lenguaje natural, en el cual las oraciones son traducidas a conceptos básicos expresados como un pequeño conjunto de semánticas primitivas.

Disco duro: Dispositivo de almacenamiento compuesto de un disco rígido cubierto con una pelicula magnética.

Disk Drive: Unidad impulsora de diskettes.

Diskette: Disco flexible.

E

Editor: Herramienta de software que es utiliza para editar programas y textos en una computadora.

Embed: Empotrar. Escribir un lenguaje de computadora de alto nivel, utilizando otro lenguaje de alto nivel.

Emular: Simular un sistema.

Equivalente: Que tiene el mismo valor (en lógica).

Espacio de búsqueda: Representación de todos los posibles estados de un problema, mediante una gráfica implícita.

Estado del problema: Condición de un problema en instante en particular.

Estructura de control: Estrategia de razonamiento. Estrategia para manipular el dominio del conocimiento y llegar a una solución de un problema especifico.

Estructura de datos: Forma en la que los datos son almacenados por la computadora. Esquema convencional para representar datos y registros.

Explosión combinatoria: El rápido crecimiento de las posibles opciones dentro de una búsqueda. Si cada nodo tiene un promedio de n ramificaciones, el espacio de búsqueda tiende a expanderse en orden de n elevado a la d. Donde d es la profundidad de la búsqueda.

Evento: Hecho, circunstancia o acción.

F

Floppy disks: Discos flexibles. Diskette.

FRANZLISP: Diálecto del lenguaje LISP.

Fuentes de Conocimiento: Componente de un sistema experto el cual se encarga de almacenar conocimiento específico de una área en particular.

Función: Relación que asigna a cada elemento del dominio uno y solo uno del contradominio. Procedimiento que calcula un único valor a partir de un conjunto de argumentos.

Función de Evaluación: Función (generalmente heurística) que calcula un valor en una posición, para seleccionar la mejor trayectoria en una búsqueda.

G

GPS: General Problem Solver. Resolvedor General de Problemas.

Gráfica: Conjunto de nodos conectados por arcos.

Gráfica de Estados: Gráfica implícita que representa el conjunto de diferentes opciones en un problema o búsqueda.

Gráfica Dirigida: Gráfica cuyos arcos apuntan en alguna dirección.

H

Hardware: Componentes físicos de los que está hecha una computadora, tales como circuitos, chips, etc.

Head Alignment: Orden dada a un disk drive para leer o escribir información de o en un floppy-disk.

Heurística: Reglas empiricas para guiar una búsqueda. Parte de la Inteligencia Artificial que se encarga del estudio de técnicas para guiar una búsqueda.

Hexádecimal: Notació para representar números en base 16.

HOL: High Order Language. Lenguaje de Alto Nivel.

I

IA: Inteligencia Artificial.

Idéntico(a): Dos proposiciones en lógica que tienen el mismo valor.

Implicación: Conectivo lógico que indica que si la primera declaración es verdadera, la declaración que se deriva de ella es también verdadera.

Inferencia: Proceso en el cual se obtiene una conclusión a partir de un conjunto inicial de proposiciones, cuya validez es conocida o asumido.

Inferir: Derivar u obtener un conocimiento por razonamiento.

Interacción: Comunicación o diálogo entre un usuario y un sistema o computadora.

Interface: Sistema por medio del cual los usuarios pueden interactuar con una computadora.

INTERLISP: Diálecto del lenguaje LISP.

Intérprete: Programa que decodifica individualmente cada instrucción de un lenguaje, traduciendolas a código de máquina.

Intuición: Facultad de percibir claramente una idea o verdad sin ayuda de la razón.

I/O: Input-Output. Entrada-Salida.

J

Jerarquía: Sistema en el cual la organización de sus elementos está ordenada por niveles, donde el más alto es el de mayor importancia o rango.

K

Kilobyte: Unidad de almacenamiento de datos que consta de 1024 bytes.

L

Lenguaje Ensamblador: Lenguaje de bajo nivel para un procesador o microprocesador.

Lenguaje de Alto Nivel: Lenguaje de computación que consiste de instrucciones más simples que el lenguaje de máquina y que generan un código ejecutable a partir de su compilación.

LISP: List Processing Language. Lenguaje Procesador de Listas.

Lista: Secuencia de cero o más elementos, enmarcados por un paréntesis. Cada elemento de la lista constituye un átomo.

Lógica Booleana: Sistema lógico binario nombrado así en honor de George Boole. Funciona con los operadores lógicos como AND, OR, NOT y ciertas reglas.

M

MACLISP: Uno de los diálectos más populares del lenguaje LISP.

Manejador de Base de Conocimiento: Sistema manejador de una base de datos pero compuesta de conocimiento.

Manejador de Base de Datos: Sistema manejador de base de datos, que incluye el almacenamiento, operaciones y recuperación de la información de algun campo del conocimiento.

Mantenimiento: Conjunto de operaciones que se realizan a un sistema para mantener en óptimas condiciones su funcionamiento.

Máquina de Von-Newman: Arquitectura estándard que usa proceso secuencial.

Megahertz (MHz): Unidad de frecuencia equivalente a un millón de ciclos por segundo.

Memoria: Dispositivo computacional de almacenamiento de datos.

Metaregla: Regla de alto nivel usada para razonar acerca de reglas de bajo nivel.

Microcódigo: Programa de computadora escrito en lenguaje de máquina.

Monitor: Dispositivo donde se puede visualizar la salida de la información y que facilita la interacción. Pantalla.

N

Negación: Cambiar una proposición a su opuesta.

Nodo: Un punto en una gráfica que unido a otros puntos representa algún problema.

Nodo Raíz: Nodo inicial en una representación en estructura de árbol.

Nodo Terminal: El nodo final en una ramificación dentro de una gráfica.

Notación Prefix: Representación de lista (usada en LISP) donde la función conectivo o predicado es dado antes de los argumentos.

O

On-Line: Información que puede ser leída inmediatamente por la computadora. Comunicación en línea directa con la computadora.

Operación Lógica: Ejecución de una instrucción singular por parte de la computadora.

Operadores: Procedimientos o acciones generalizadas que pueden ser usadas para cambiar situaciones.

P

Parser: Fase de la compilación que determina las relaciones sintácticas entre los componentes de una instrucción dada y el lenguaje de programación.

Percepción: Proceso activo en el cual las hipótesis son formadas con la naturaleza del ambiente o por lo que captan los sentidos.

Pista: Cualquiera de los circulos concéntricos en que es dividido un disco.

Plan: Secuencia de acciones para transformar un estado inicial en una situación satisfactoria.

Portabilidad: Característica con que son dotados ciertos programas, para lograr su transferencia de un sistema a otros.

Predicado: Parte de una proposición que hace una aserción (por ejemplo: estados en realción de un atributo) acerca de elementos individuales.

Predicado Lógico: Modificación de la lógica proposicional que permite el uso de variables y funciones de variables.

Predicado Lógico de Primer Orden: Forma popular entre la comunidad de IA, usada para representar conocimiento y ejecutar inferencias lógicas.

Premisa: Primera proposición dentro de secuencias de razonamiento.

PROLOG: Programming Logic. Lenguaje de programación con enfoque a la IA.

Promt: Símbolo o carácter distintivo de un compilador, que señala el principio de línea.

Proposición: Una sentencia (en lógica) que puede ser verdadera o falsa.

Q

Quinta Generación de Computadoras: Proyecto cuyo fin es fabricar máquinas "inteligentes". Actualmente en desarrollo en Japón.

R

RAM (Random Access Memory): Mecanismo de acceso rápido a escritura y lectura de información en los microprocesadores.

Razonamiento: Facultad intelectual que permite actuar acertadamente distinguiendo lo falso de lo verdadero.

Recolección de Basura: Técnica para reciclar las celdas que ya no estan en uso en la memoria de la máquina.

Red de Computadoras: Sistema de comunicación interconectado entre computadoras.

Registro: Un bloque de datos de un archivo en disco que es almacenado como una unidad. Una celda de almacenamiento que es usada por el microprocesador para hacer operaciones lógicas y aritmeticas. Conjunto de bytes.

Reglas de Producción: Un módulo estructural de conocimiento que representa un sólo tipo de conocimiento y que genera otros a partir de una estructura del tipo IF-THEN.

Reset: Procedimiento para regresar a la computadora a su estado inicial inmediatamente después de ser encendida.

ROM (Read Only Memory): Dispositivo de almacenamiento del cual sólo puede ser leída la información, pero no cambiada. A menudo se graba en ella el cargador del sistema operativo.

S

Sèctor: Sección de un disco que consta de un conjunto de pistas. Generalmente tiene la forma de una rebanada de pastel.

Sentido Común: La habilidad para actuar apropiadamente en situaciones cotidianas, basada en la experiencia adquirida a lo largo de la vida. Bajo nivel de razonamiento basado en un cúmulo de experiencia.

Seudo-Código: Forma de escribir un procedimiento con palabras orientadas a un lenguaje de programación.

Silogismo: Argumento deducido en lógica y basado en dos premisas.

Slot: Ranura. Un elemento en una representación de Minsky que será llenado con información acerca de una particular situación.

Software: Programas, sistemas y paquetes de computación. La parte de la programación.

Stack: Estructura de datos usada para almacenar temporalmente información y que es de la forma LIFO (Last In First Out) el último elemento en entrar es el primero en salir.

T

Técnicas de Búsqueda Heurísticas: Métodos de búsqueda que usan conocimiento para guiar la búsqueda.

Teorema: Proposición que debe ser probada, basándose en una teoría dada.

Tiempo Compartido: Ambiente de computación en el cual varios usuarios pueden usar el sistema en forma concurrente.

Trayectoria: Camino particular tomado en una dirección en una gráfica de estados.

Trayectoria de Solución: Trayectoria que lleva al nodo meta en un algoritmo de búsqueda.

U

Unificación: Procedimiento para obtener instancias.

Union: Conjunción.

V

Valor False: Uno de los dos posibles valores en lógica.

Valor True: El otro de los dos posibles valores en lógica.

Variable: Entidad matemática que cambia de valor cuando se le asignan valores dentro de un procedimiento. Nombre usado por un programa de computadora para almacenar un dato.

Variable Global: Variable inherente a todo el programa.

Variable Local: Variable particular de un procedimiento dentro de un programa.