CLASIFICACION, ORDENACION Y BUSQUEDA INTELIGENTE

Autor:
Pablo Andrés Benavides Bastidas

Asesor:
Ing. Rodrigo Naranjo

CONTENIDO

V ALTERNATIVAS

1. PROPUESTA DE ALTERNATIVAS
2. EVALUACION DE PROPUESTAS
3. VISON FUTURISTA

VI METODOLOGIA

1. ¿QUE ES UNA METODOLOGIA?
2. ¿DONDE SE APLICA UNA METODOLOGIA?
3. ¿ES NECESARIA UNA METODOLOGIA?
4. ESTUDIO DE LAS METODOLOGIAS EXISTENTES APLICABLES

VII LA PROPUESTA

1. PROPUESTA DE UNA NUEVA METODOLOGIA DE CLASIFICACION, ORDENAMIENTO Y BUSQUEDA INTELIGENTE APLICABLE A VARIOS FORMATOS DE INFORMACION
2. ANALISIS DE APLICABILIDAD DE LA NUEVA METODOLOGIA
3. CAMPOS DE DESARROLLO

VIII LA APLICACIÓN

1. DETERMINACION DE LA APLICACIÓN SOBRE EL ESTUDIO
2. BASES Y OBJETIVOS DE LA APLICACIÓN
3. LIMITANTES
4. DESARROLLO DE LA APLICACIÓN
5. PRUEBAS DE LA APLICACIÓN
6. EVALUACION Y RESULTADOS DE LA APLICACIÓN

IX VERIFICACION DE HIPOTESIS

1. EVALUACION DE RESULTADOS GENERALES
2. VERIFICACION DE HIPOTESIS
3. CORRECCION DE ERRORES
4. INFORME

X CONCLUSIONES Y RECOMENDACIONES

1. CONCLUSIONES
2. RECOMENDACIONES
3. PALABRAS FINALES

XI ANEXOS

1. CRONOGRAMA DE ACTIVIDADES
2. PLANIFICACION DE ESTUDIO
3. EL SISTEMA DE EVALUACION
4. LA APLICACIÓN
5. EVALUACIONES Y PRUEBAS
6. RESULTADOS
7. BIBLIOGRAFIA

Prefacio

La revolución

Existe una revolución inexorable que hacen posible la tecnología y las ideas descritas en este proyecto. La forma en que los humanos aprenden y tiene acceso a la información, así como la forma dinámica en que ésta se presenta han desencadenado una revolución electrónica mucho más compleja y poderosa que la liberación de la letra impresa que ocurrió hace 500 años en Europa Central. La última revolución, encabezada por Gutemberg, Grolier, Aldo Manuzio y quienes han construido y utilizado las prensas de impresión, desató las transformaciones de la condición humana más poderosa y permanentes, que han excedido por mucho lo que la gente de ese tiempo imaginó.

Tal es la cantidad de información que maneja el individuo en la actualidad que existe una relación directa entra la cantidad de información y la velocidad de acceso a ella. Todos desean encontrar ese dato para ya, *"el tiempo es oro"* y no solo basta con poseer un equipo de computación de última tecnología con la mejor velocidad de procesamiento y la mayor cantidad de almacenamiento primario y secundario; todo esto sería un recurso desperdiciado si en contraste utilizamos un sistema de manejo de información cuyos algoritmos de manipulación (ordenación, clasificación, búsqueda) de información sean muy lentos o desperdicien tiempo precioso haciendo comparaciones inútiles e innecesarias.

Una invitación

Si gracias a sus talentos innatos como desarrollador de software, puede distinguir la frágil silueta del futuro entre las nieblas de multimedia, los nuevos formatos de datos, las clases, objetos incrustados, OLE, DDE, y quien sabe cuantos tipos de datos que aparecen día tras día; de seguro comprenderá su gran responsabilidad en la generación de sistemas de manipulación de datos que aprovechen las tecnologías actuales y puedan manipular esos volúmenes tan grandes de información.

Por ello la invitación está hecha para que no se basen simplemente en los datos, estudios y conclusiones formadas en este proyecto sino que sean investigadores críticos y analíticos que puedan determinar sus propias metodologías dependiendo de sus propias necesidades.

Que este proyecto sea la base o incentivo de otros trabajos más específicos y proyectos de investigación. El campo a tratar es muy amplio y pueden quedar ciertas áreas sin ser topadas con la debida profundidad.

Pablo Andrés Benavides Bastidas
Ibarra, Ecuador
Febrero, 1998-02-27

Agradecimientos

Quisiera agradecer a mi familia y a muchos colegas que me dieron su tiempo y espacio para desarrollar este proyecto de investigación.

Un agradecimiento especial a todos aquellos quienes me ayudaron a revisar el contenido del proyecto, facilitaron material de apoyo, ayudaron a probar los algoritmos y determinar las mejores alternativas; muchos estuvieron ahí cuando los necesité. Me gustaría agradecerles a todos su tiempo y las facilidades que brindaron para este proyecto:

Ing. Rodrigo Naranjo	ASESOR DEL PROYECTO DE TESIS
Ing. Miguel Orquera	DECANO FICA (1998)
Ing Jorge Caraguay	DIRECTOR EISIC – FICA (1998)

Algoritmos

Ordenación, Clasificación y Búsqueda Inteligente de Información

parte

I

Introducción

La mejor teoría es inspirada por la práctica,
la mejor práctica es inspirada por la teoría

Donald Knuth

Introducción

En la actualidad la mayoría de las actividades que debemos realizar, están basadas en el manejo de información (llegando en algunos casos a manipular cantidades inimaginables de esta). Y a pesar de ello el usuario común ajeno a la realidad a la que se enfrentan los programadores y desarrolladores de herramientas para manejo de información, exige el manejo de los nuevos formatos de información tan de moda en estos momentos (voz, texto, imágenes, etc.) de una manera rápida y efectiva, e incluso piden que el nuevo sistema sea capaz de utilizar criterios de búsqueda definidos por el usuario durante la ejecución del mismo. Todo reporte o resultado de búsqueda debe ser realizada con el criterio "lo quiero para ayer . . . ".

Un buen ejemplo de esta situación la podemos observar en los buscadores de información "Yahoo" ó "Alta Vista" en INTERNET. Si nos imaginásemos el trabajo interno -a nivel de programación- que realizan estos buscadores, la cantidad de información que manipulan, los criterios que utilizan y sobretodo la velocidad de respuesta de sus búsquedas, nos daríamos cuenta quizás de cual es la verdadera "programación".

Nuestro campo de desarrollo -de seguro- va a estar orientado al manejo de información, si tomamos en cuenta las exigencias de los usuarios actuales -quienes piden cosas que para nosotros a primera vista podrían ser imposibles de elaborar-, y si a todo esto añadimos la "completa" bibliografía sobre algoritmos de búsqueda y clasificación que tenemos a nuestro alcance, podremos con claridad determinar nuestra situación futura como programadores y desarrolladores de sistemas.

Es así que el presente proyecto quiere satisfacer y cubrir algunas de estas necesidades para los programadores, ofreciendo una metodología de aplicación de algoritmos de clasificación, organización y búsqueda inteligente de información en varios formatos. Se tratará de una herramienta de estudio e investigación orientada al programador, sustentada por un estudio detallado de los algoritmos de clasificación, ordenamiento y búsqueda inteligente aplicables a grandes volúmenes de información. Y se presentará una aplicación demostrativa de los estudios realizados como ejemplo de su funcionalidad.

Información

Recolección de Información sobre Algoritmos

parte

III

Recolección de Información

Los sistemas complejos tienden a opomerse a su propia función.

Principio de Le Chatelier
Leyes de Murphy II

3. Recolección de Información correspondiente a los Algoritmos de Clasificación, Ordenamiento y Búsqueda Inteligente

Clasificación

"... No existe nada más difícil de emprender, más peligroso de dirigir, ni de más incierto éxito que la iniciativa de la introducción de un nuevo orden de las cosas..."

NICCOLO MACHIAVELLI

Un tema que aparece frecuentemente en programación es sin duda el reordenamiento de elementos en orden ascendente o descendente. Imaginar lo duro que sería el uso de un diccionario si sus palabras no estuvieran dispuestas en orden alfabético; de forma similar, el orden en que los registros están almacenados en la memoria de un ordenador tiene frecuentemente una profunda influencia en la velocidad y simplicidad de los algoritmos que los tratan.

Aunque los diccionarios definen ***Clasificación*** como el proceso de separación u ordenación de cosas de acuerdo con su clase o tipo, es tradicional para los programadores de ordenadores utilizar la palabra en el sentido mucho más especial de clasificar cosas en un orden ascendente o descendente. El proceso quizá debería llamarse *ordenación* y no clasificación pero cualquier intento de llamarlo *ordenación* conducía pronto a confusiones debido a los diferentes significados ligados a dicha palabra. Considérese, por ejemplo, la siguiente sentencia: "Puesto que dos de nuestras unidades de cinta estaban bajo órdenes de trabajo, había ordenado hacer una orden de pedido, en orden a ordenar rápidamente los datos según varios órdenes de magnitud". La terminología matemática abunda en todavía más sentidos de orden (el orden de un grupo, el orden de una permutación, el orden de un punto de bifurcación, relaciones de orden, etc.). así concluimos que la palabra *ordenar* puede conducir al caos.

Algunas de las principales aplicaciones de la clasificación son:

a) Resolución de problemas de *reunión*, donde todos los elementos de igual identificación se tratan juntos, supóngase que tenemos 10.000 elementos en orden aleatorio, algunos de los cuales tienen valores idénticos y que deseamos reordenar este fichero de forma que aquellos registros de idéntico valor aparezcan en posiciones consecutivas. Esencialmente este es el problema de *clasificación* en el sentido clásico de la palabra, de forma que los valores estén en orden ascendente, $v_1 \leq v_2 \leq \ldots \leq v_{10000}$ (La eficiencia que es posible en este procedimiento, explica porque ha cambiado el sentido original de la *clasificación*)

b) Si dos o más ficheros se han clasificado en el mismo orden, es posible obtener en un solo paso secuencial de los ficheros, todos los emparejamientos. Este es el principio que ***Perry Mason*** utiliza para ayudarle a solucionar el caso del crimen de *El caso de la enlutada irascible* (1951, cuando fue preguntado de la manera de consultar una lista de números de licencias de manera rápida, contestó que simplemente ordenarían la lista y buscarían los duplicados). Normalmente es mucho más económico acceder a una lista de información en secuencia desde el principio al fin, que acceder a l azar saltando de un punto a otro, excepto que la lista sea lo suficientemente pequeña para almacenarla en una memoria de acceso directo de alta velocidad. Efectuando clasificaciones, es posible utilizar un acceso secuencial en grandes ficheros, como sustitución factible del direccionamiento directo.

c) La clasificación también es una ayuda para problemas de búsqueda, como veremos más adelante, y por consiguiente, ayuda a obtener las salidas del ordenador más convenientes para la utilización humana. De hecho, una lista que se ha clasificado en orden alfabético parece frecuentemente bastante fiable, aunque la información numérica asociada se haya calculado incorrectamente.

Aunque la clasificación tradicionalmente se ha utilizado generalmente para el proceso de datos en aplicaciones de gestión, actualmente es una herramienta básica que un programador debe tener en cuenta para usarla en una amplia variedad de situaciones.

Uno de los primeros sistemas de software de gran capacidad que demostraba la versatilidad de la clasificación, fue el *Larc Scientific Compiler* desarrollado por la Computer Sciences Corporation en 1960. Este compilador optimizado para un lenguaje FORTRAN extendido, hacía uso abundante de clasificaciones de forma que los distintos algoritmos de compilación aparecían junto con las pertinentes partes del programa fuente en la secuencia conveniente. El primer paso era un examen léxico que dividía el programa fuente en elementos, representando cada uno un identificador (por ejemplo, el nombre de una variable), una constante, un operador, etc. Se asignaban varios números de secuencia a cada elemento; clasificados por el nombre y un número de secuencia apropiado, se procesaban conjuntamente todos los usos de un identificador determinado. A las *definiciones de referencia de entrada* que especificaban, por ejemplo, que el identificador era un nombre de función, parámetro o una variable dimensionada, se les asignaba el menor número de secuencia, de forma que aparecieran primero entre los elementos que tuvieran un identificador determinado; ello facilita la verificación de uso conflictivo de un identificador, y asignación de memoria en declaraciones EQUIVALENTE, etc. La información de cada identificador reunida de este modo se ligaba a cada elemento; de esta forma la *tabla de símbolos* de identificadores no era necesario mantenerla en la memoria de alta velocidad. Los elementos actualizados se clasificaban según otro número de secuencia, con lo que se obtenía esencialmente el programa fuente de nuevo en su orden original, con la excepción de que la secuencia estaba diseñada especialmente para colocar las expresiones aritméticas en una forma

prefija polaca más idónea. También se usaban clasificaciones en fases posteriores para facilitar la optimización de bucles, intercalar mensajes de error en el listado, etc. Resumiendo, el compilado estaba diseñado de forma que virtualmente todo el proceso se podía efectuar secuencialmente desde ficheros que estaban almacenados en una memoria auxiliar sobre tambor.

Otra, más evidente aplicación de clasificación se encuentra en las rutinas de edición de ficheros, donde cada línea se identifica por un número clave. Mientras un usuario está introduciendo adiciones y cambios, no es necesario mantener todo el fichero en memoria. Las líneas modificadas pueden clasificarse a continuación (usualmente están desordenadas), y se intercalan con el fichero original. Ello conduce a una forma razonablemente eficiente de utilización de la memoria en situaciones de multiprogramación.

Los fabricantes de ordenadores estiman que más de un 25% del tiempo de utilización de sus ordenadores se gastan en clasificaciones. Existen muchas instalaciones en las que las clasificaciones ocupan más de la mitad del tiempo del ordenador. Según estas estadísticas podemos sacar la conclusión que o bien:

i) existen muchas aplicaciones importantes de la clasificación, o
ii) muchas personas clasifican cuando no debieran, o
iii) se utilizan comúnmente algoritmos de clasificación ineficientes

La verdad real probablemente incluye algo de las tres alternativas. En cualquier caso podemos observar que la clasificación es digna de serios estudios como una materia práctica.

Pero incluso si la clasificación fuera inútil, la recompensa que obtendríamos sería suficiente razón para estudiarla de todas formas. Los algoritmos ingeniosos, que se han descubierto, muestran que la clasificación es, por derecho propio un tema interesante de investigar. Existen muchos problemas fascinantes no resueltos en esta área (así como bastantes resueltos).

Desde una perspectiva más amplia podremos encontrar también que los algoritmos de clasificación constituyen un interesante *caso de estudio* de cómo abordar en general problemas de programación de ordenadores. Se ilustrarán muchos principios importantes de manipulación de estructuras de datos. Examinaremos la evolución de varias técnicas de clasificación, en un intento de indicar como el programador puede ir descubriendo las ideas básicas por sí mismo (si ya ha afrontado el problema anteriormente). Por extrapolación de este caso de estudio, podemos aprender bastante acerca del propósito que ayuda a diseñar buenos algoritmos para otros problemas.

Las técnicas de clasificación suministran también una excelente ilustración de las ideas generales implicadas en le *análisis de algoritmos*, o sea, las ideas usadas para determinar las características de rendimiento de algoritmos para poder elegir inteligentemente entre métodos en competencia. Los programadores con inclinaciones

matemáticas encontrarán en esta sección algunas técnicas para estimar la rapidez en algoritmos de cálculo y para resolver relaciones de recurrencia complicadas. Por otra parte, el material se ha ordenado de forma que los programadores puedan fácilmente saltarse estos cálculos.

Definición del problema de aplicación

Antes de continuar debemos definir nuestro problema con más claridad, e introducir alguna terminología. Primeramente tomaremos N elementos para clasificar:

$$R_1, R_2, \ldots, R_N$$

A cada elemento los denominaremos *registro*, y al conjunto de N registros *fichero.* Cada registro R_j tiene una clave K_j, que gobierna el proceso de clasificación. Puede existir en el registro además de la clave información adicional: *la información satélite* extra no tiene ningún efecto en el proceso de clasificación, excepto que permanece con el mismo registro.

Se especifica entre las claves una relación de ordenación "<" de forma que, para tres valores a, b, c, se cumplen las siguientes condiciones:

a) Exactamente una de las posibilidades a < b, a = b, b < a es cierta. (Ley de la tricotomía)
b) Si a < b y b < c, entonces a < c. (Ley de la transitividad)

Estas dos opciones caracterizan el concepto matemático de *ordenación lineal*, llamado también *ordenación total*. Cualquier relación "<" que satisfaga a) y b) puede clasificarse por la mayor parte de los métodos mencionados en este capítulo, aunque algunas técnicas de clasificación requieren claves numéricas o alfabéticas con la ordenación usual.

El objetivo de la clasificación es determinar una permutación p(1)p(2) > p(N) de los registros, que coloque las claves en orden no decreciente:

$$K_{p(1)} \leq K_{p(2)} \leq \ldots \leq K_{p(N)} \qquad (1)$$

Una clasificación se llama *estable*, si además exigimos que los registros con igual clave retengan su orden relativo original, o sea, si

$$p(i) < p(j) \quad \text{cuando} \quad K_{p(i)} = K_{p(j)} \quad \text{e} \quad i < j \qquad (2)$$

En algunos casos desearemos que los registros estén físicamente ordenados en memoria de forma que sus claves estén en orden, mientras que en otros casos será

suficiente tener una tabla auxiliar que especifique de alguna forma la permutación, de modo que pueda accederse a los registros en el orden de sus claves.

Algunos métodos de clasificación suponen la existencia de uno o ambos valores "∞" y "-∞", que se definen como mayor que o menor que todas las claves, respectivamente:

$$-\infty < K_j < \infty, \quad 1 \le j \le N \qquad (3)$$

Estos valores se sustituyen por claves artificiales y pueden usarse también como marcas indicadoras. El caso de igualdad está excluido en (3); si la igualdad pudiera ocurrir, los algoritmos deberían modificarse de forma que fueran operativos, pero normalmente a costa de algo de elegancia y eficiencia.

Pueden distinguirse dos tipos de clasificaciones, *clasificación interna*, en donde los registros se mantienen en memorias de acceso al azar de alta velocidad; y *clasificación externa* cuando el número de registros excede a la capacidad de memoria en cualquier momento. La clasificación interna permite una mayor flexibilidad en la estructura y acceso a los datos, mientras que la clasificación externa nos muestra como tratar con cierto rigor las restricciones de acceso.

El tiempo para clasificar N registros, utilizando un algoritmo de clasificación de uso general aceptable, es aproximadamente proporcional a N log N; efectuaremos cerca de log N *pasadas* sobre los datos. Este es el tiempo mínimo posible, como veremos en más adelante. Así si doblamos el número de registros, su clasificación exigirá más del doble de tiempo, permaneciendo igual el resto. (realmente, cuando N tiende a infinito, una mejor estimación del tiempo necesario para clasificar es $N(\log N)^2$, si las claves son distintas, ya que el tamaño de las claves crece por lo menos como log N; pero para usos prácticos, en realidad N nunca tiende a infinito)

CLASIFICACION INTERNA

Antes de empezar la discusión sobre cuál es el mejor *clasificador*, realizando un pequeño experimento. ¿Cómo se podría resolver el siguiente problema de programación?

Las posiciones de memoria R+1, R+2, R+3, R+4 y R+5 contienen 5 números. Escribir un programa que reordene estos números, si es necesario, de forma que estén en orden creciente.

Nota. Si está familiarizado con algún método de clasificación, es recomendable que haga lo posible por olvidarlo momentáneamente; imagine que está atacando este problema por primera es, sin ningún conocimiento de cómo proceder.

Es muy posible que su solución al problema anterior sea uno de los siguientes tipos de clasificación que analizaremos:

1. Por Inserción
2. Por Intercambio
3. Por Selección
4. Por Enumeración
5. Un Procedimiento de Clasificación Especial
6. Una Actitud Perezosa
7. Una nueva Supertécnica de Clasificación

Si el problema se hubiera propuesto para 1000 elementos y no únicamente para 5 podría haber descubierto alguna de las más sutiles técnicas que se mencionarán posteriormente. De todas formas, cuando se ataca un problema nuevo, a menudo es prudente hallar algunos procedimientos que sea evidente que funcionen, y luego intentar perfeccionarlo.

Se han inventado muchos algoritmos de clasificación diferentes y trataremos el mayor número posible de ellos en esta investigación. Este número de métodos tan alarmante constituye solo una fracción de los algoritmos que se han ideado hasta ahora, la mayoría de ellos que ahora están obsoletos serán omitidos en nuestro estudio, sin embargo es posible que algunos de ellos sean mencionados brevemente en alguna instancia.

¿Por qué existen tantos métodos *X*?, donde *X* se entiende sobre el conjunto de todos los problemas; y la respuesta es que cada método tiene sus propias ventajas e inconvenientes, de modo que de ja en inferioridad los otros según las configuraciones de los datos y el hardware. Desafortunadamente, no se conoce la *mejor* forma de clasificar, existen muchos *mejores métodos*, dependiendo de lo que se tenga que clasificar, en qué máquina y para qué propósito.

En palabras de Rudyard Kipling, “hay novecientas noventa formas de establecer las leyes de una tribu, y cada una de ellas es correcta”.

Aprender las características de cada método de clasificación es una buena idea, porque para aplicaciones particulares puede hacerse una elección inteligente. Afortunadamente, no es una tarea formidable aprender estos algoritmos, ya que tienen interrelaciones interesantes.

En el presente apartado nos centraremos en las *clasificación interna*, cuando el número de registros a clasificar es lo suficientemente pequeño como para que se proceso pueda efectuarse en la memoria rápida de un ordenador.

En algunos caso desearíamos que los registros estén físicamente ordenados en memoria, de forma que sus claves estén en orden, mientras que en otros casos puede ser

suficiente obtener una tabla auxiliar que especifique la permutación. Si los registros y/o las claves ocupan bastantes palabras de memoria, a menudo es mejor construir una nueva tabla de direcciones encadenadas en vez de estar moviendo registros voluminosos. Este método recibe el nombre de *clasificación por tabla de direcciones*. Si la clave es pequeña, pero la información satélite del registro es larga, se toma junto con la dirección para mayor velocidad, recibiendo el nombre de *clasificación por claves*. Otros esquemas de clasificación utilizan un campo de encadenamiento que se incluye en cada registro; estos encadenamientos se tratan de tal forma que, en el resultado final, los registros están formados encadenados formando una lista lineal directa, en la que cada encadenamiento apunta al registro siguiente, recibe el nombre de *clasificación por listas.*

Después de clasificar por los métodos de tabla de direcciones o de listas, se pueden reordenar los registros en orden creciente tal como se deseaba. Existen varias formas para efectuarlo, necesitando únicamente memoria adicional para contener un registro, o alternativamente podemos simplemente mover los registros a una nueva área capaz de contenerlos a todos. El último método es usualmente dos veces más rápido que el primero, pero necesita aproximadamente dos veces más de espacio en memoria. En muchas aplicaciones no es necesario mover los registros, ya que los campos de encadenamiento a menudo son adecuados para las subsiguientes operaciones de direccionamiento.

Todos los métodos que examinaremos los ilustraremos de cuatro formas:

a) una descripción del algoritmo
b) un organigrama
c) un programa en pseudocódigo / pascal / c
d) un ejemplo del método de clasificación aplicado a un conjunto dado de números

Ordenación

"*. . . la eficiencia de una ordenación puede determinar en gran medida la eficiencia del programa entero, una buena rutina de ordenación es muy deseable. . . "*

Algoritmos de Ordenación y Consideraciones de Eficiencia

En muchas de nuestras actividades cotidianas nos hemos encontrado con el hecho prioritario de mantener "listas de elementos" ordenados: registros de estudiantes ordenados por el número de identificación, enteros ordenados de menor a mayor, palabras ordenadas alfabéticamente, etc. Por supuesto, el objetivo de mantener listas ordenadas es facilitar la búsqueda. Dada una apropiada estructura de datos, un elemento particular de la lista puede encontrarse más apropiadamente si la lista está ordenada.

El poner una lista desordenada de elementos de forma ordenada (ordenación) es una operación muy útil y común. Se han escrito libros enteros sobre varios algoritmos de ordenación, así como sobre algoritmos de búsqueda de un elemento particular en una lista ordenada. El objetivo es encontrar el mejor algoritmo de ordenación. Como la eficiencia de una ordenación puede determinar en gran medida la eficiencia del programa entero, una buena rutina de ordenación es muy deseable. Esta es un área en la que algunas veces se anima a los programadores a sacrificar claridad en favor de rapidez de ejecución.

¿QUE ES BUENO?

¿Qué entendemos por un buen algoritmo de ordenación? ¿Cómo podemos comparar dos algoritmos que realizan la misma tarea?

Para hacer tal comparación, debemos primero definir un conjunto de medidas objetivas que se puedan aplicar a cada algoritmo. El análisis de los algoritmos es un área importante de la informática teórica, se quiere determinar cuál de dos algoritmos requiere menos trabajo para realizar una tarea en particular.

¿Cómo medimos el trabajo que realizan dos algoritmos? La primera solución que nos viene a la mente es simplemente codificar los algoritmos y comparar entonces los tiempos de ejecución, ejecutando los dos programas, ejecutando los dos programas. Aquel con el tiempo de ejecución más corto será claramente el mejor algoritmo. ¿O no lo es? Usando esta técnica, realmente sólo podemos determinar que el Programa A es más eficiente que el Programa B *sobre una computadora en particular*. Los tiempos de ejecución son específicos de una computadora particular. Por supuesto, podríamos probar los algoritmos sobre todas las computadoras posibles, pero queremos una medida más general.

Una segunda posibilidad es contar el número de instrucciones o sentencias ejecutadas. Sin embargo, esta medida varía con el lenguaje de programación usado, así con el estilo individual del programador. Para normalizar algo esta medida, podríamos contar el número de pasos a través de bucles críticos del algoritmo. Si cada iteración implica un constante aumento de trabajo, esta medida nos dará criterio significativo de eficiencia.

Estos pensamientos nos conducen a la idea de aislar una operación fundamental del algoritmo y contar el número de veces que se realiza la operación. Supongamos, por ejemplo, que estuviésemos buscando un cierto valor en un array. Podemos contar cuántas comparaciones entre valor buscado y los elementos del array se necesitan para localizar el valor. Si estuviésemos sumando los elementos de un array se necesitan para localizar el valor. Si estuviésemos sumando los elementos de una array de enteros, podríamos contar las operaciones de adición necesarias. Observe que esta cantidad es función del número de elementos del array. Para un array de N elementos, habrá N-1 operaciones de suma. Por lo que podemos comparar los algoritmos para el caso general,

no sólo para un array con un tamaño específico. Si quisiéramos comparar algoritmos de producto de dos matrices reales, podríamos encontrar una medida que combine las operaciones multiplicación y suma de números reales, requeridas para el producto de matrices.

Este último ejemplo nos lleva a una constante consideración interesante: Algunas veces una operación dominará el algoritmo de tal forma, que las otras operaciones se desvanecerán como "ruido de fondo". Si queremos comprar elefantes y peces de colores, por ejemplo, y estamos considerando dos tiendas de animales, realmente sólo necesitamos comprar los precios de los elefantes; el coste de los peces de colores es trivial en la comparación. De forma similar, la multiplicación de reales es mucho más cara que la suma en términos de tiempo de computadora, por lo que la operación suma es un factor trivial en la eficiencia de todo algoritmo de producto de matrices; podemos contar sólo las operaciones de multiplicación, podemos encontrar a menudo una operación que domina el algoritmo, relegando efectivamente a las otras a nivel de ruido.

En nuestro estudio de los algoritmos de ordenación, usaremos el número de comparaciones como medida de eficiencia de cada algoritmo. Cada ordenación tiene la misma entrada (un array desordenado) y producirá la misma salida (un array ordenado con los mismo elementos). Diremos que la Ordenación A es más "eficiente" que la Ordenación B si puede realizar la misma tarea con menor cantidad de trabajo. Es decir, una ordenación más eficiente, según nuestra medida, requerirá menos comparaciones.

Como en el caso de la suma de los elementos de un array, realmente no tendremos que contar el número de operaciones críticas sobre un array de un tamaño particular. El número de comparaciones será alguna *función* del número de elementos (N) del array. Por tanto, podemos expresar el número de comparaciones en términos de N (por ejemplo, $N+5$ o N^2) antes que como un valor entero (por ejemplo, 52).

A continuación se evaluarán los siguientes algoritmos:

- Ordenación por selección directa
- Ordenación por burbuja
- Ordenaciones O (N log N)
- Ordenación por mezcla
- Ordenación rápida
- Ordenación por montículo
- Ordenación con Claves y Punteros

ORDENACION POR SELECCIÓN DIRECTA

Si usted está manejando una lista de nombres y le piden que la ponga en orden alfabético, puede usar este método general:

1. Encontrar el nombre que va primero en el alfabeto y escribirlo sobre una segunda hoja de papel
2. Tachar el nombre de la lista original
3. Continuar este ciclo hasta que se hayan tachado todos los nombres de la lista original y se hayan escrito en la segunda lista, la cual estará ahora ordenada

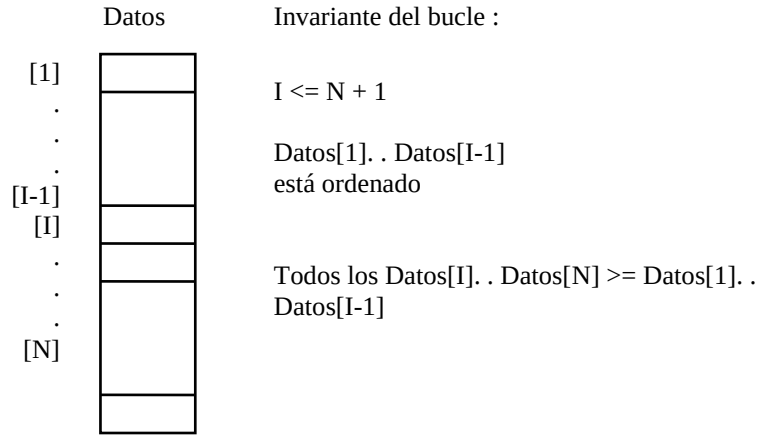
Este algoritmo es fácil de trasladar a un programa de computadora, pero tiene un inconveniente: Requiere espacio en memoria para almacenar dos listas completas. Aunque no hemos hablado sobre consideraciones de espacio de memoria, esta duplicación es claramente un despilfarro de recursos, no importa que en la actualidad dispongamos de grandes espacios de almacenamiento y recursos de memoria virtual realmente grandes. Sin embargo, un pequeño ajuste a este método manual nos evitará la necesidad de duplicar el espacio ocupado. Cuando usted tacha un nombre de la lista original, queda un espacio libre. En vez de escribir el valor mínimo en una segunda lista, puede cambiarlo por el valor que ocupa actualmente su lugar correcto en la lista.

Veamos el ejemplo de la Figura

1. El valor más pequeño de Datos está en la posición [4]. Por tanto, intercambiamos los contenidos de Datos[1] y Datos[4]. Ahora el valor más pequeño está en su lugar correcto del array
2. El valor más pequeño de la lista ordenada del array (Datos[2]. . Datos[5]) está en la posición [3], por lo cual intercambiamos Datos[2] y Datos[3]. Ahora las dos primeras posiciones del array están ordenadas
3. El valor más pequeño de la parte desordenada del array está en Datos[3], su posición correcta, luego no hacemos nada. *Realmente, el algoritmo intercambia el valor consigo mismo, en vez de comprobar un caso especial*
4. El valor más pequeño de la parte desordenada del array (Datos[4]. . Datos[5]) está en la posición [5] ; intercambiamos Datos[4] y datos[5]
5. Datos[5] debe contener el valor más grande del array. Los valores de Datos están ordenados.

	Datos	Datos	Datos	Datos
[1]	126	**1**	**1**	**1**
[2]	43	43	**26**	**26**
[3]	26	26	43	**43**
[4]	1	126	126	**113**
[5]	113	113	113	**126**
	(a) Original	(b) Primera iteración	(c),(d) Segunda iteración	(e) Cuarta iteración

El algoritmo para la ordenación por selección se ilustran en la Figura siguiente:



(a) Figura anterior

(b) Encontrar el valor más pequeño de Datos[I]. . Datos[N]

(c) Cambiarlo con Datos[I]

(d) Incrementar I en 1 y volver al paso (a)

Descompongamos el paso (b) del algoritmo. Nuestra tarea es encontrar el índice del valor mínimo en la sección desordenada del array. Necesitaremos un bucle que comience con I y compare los valores del subarray Datos[I]. . Datos[N], devolviendo el índice del valor más pequeño, Míndice. La variable de control del bucle, J, que representa el movimiento del cursor a través de la parte desordenada del array, comenzará en I+1. En cada iteración preguntaremos: ¿El valor de Datos[J] es menor que el valor de Datos[Míndice]? Si es así, actualizamos Míndice. Incrementamos J para continuar con la comprobación hasta alcanzar el final del array (cuando J > N). Míndice es ahora el índice del valor más pequeño en la parte desordenada del array. El algoritmo se ilustra a continuación :

Cada bucle se expresa en términos de la relación de las variables del bucle, es decir la invariante del bucle. El invariante para el algoritmo es: “Los elementos desde Datos[1] hasta e incluyendo Datos[N] son mayores o iguales que cualquiera de los valores de Datos[1]. . Datos[i-1]”

Al medir la cantidad de trabajo necesitado por este algoritmo en términos del número de comparaciones realizadas. Existen dos bucles, uno anidado dentro del otro, y la comparación está en el bucle interior. La primera vez que se pasa por el bucle interior, se realizan N-1 comparaciones., la siguiente vez N-2 comparaciones y así sucesivamente, hasta que se realiza una comparación en la última iteración. Esto da un total de:

$$(N-1) + (N-2) + (N-3) + . \ . \ . \ + 1 = (N\ (N-1)/2\ )$$

Para realizar nuestro objetivo de ordenación de un array de N elementos, la ordenación por selección directa requiere N(N-1)/2 comparaciones. Para un array de 10 elementos, por ejemplo, se necesitan 45 comparaciones. Observe que un array de doble tamaño necesitará más del cuádruple del número de comparaciones. Observe también que una determinada colocación de los valores en el array no afecta a la cantidad de trabajo que hay que hacer. Incluso si el array está ordenado antes de llamar al procedimiento realizará N(N-1)/2 comparaciones.

Búsqueda

"*. . . un programa de computación hace lo que se le ordena hacer, no lo que se quiere que haga. . . "*
Tercera Ley de Greer, Leyes de Murphy II

Para entender la importancia que tienen los motores de búsqueda y los índices imaginemos una enciclopedia cuyo contenido no estuviera ordenado alfabéticamente; encontrar un término en concreto sería cuestión de suerte. Tomaremos como base un claro y actual ejemplo de gran almacenamiento de información en diferentes formatos y de donde los usuarios desean accesar a esta de la manera más rápida y precisa, esta es INTERNET.

Dada la importancia y utilización actual de Internet, debemos de tener muy en cuenta este nuevo medio de intercomunicación y almacenamiento de tanta y tanta información sobre diversos temas y bajo distintos tópicos e idiomas. Es así que Internet los conceptos, ideas, servidores y servicios están dispersos por todo el mundo, en un anárquico desorden. Desde el navegador experto hasta el que acaba de conocer Internet y no asimila todavía toda la importancia que tiene esta red de ordenadores en la Historia de la Humanidad, todos naufragarían habitualmente en el mar de la información digital si no fuera por los índices y los motores de búsqueda.

Como todo en Internet, los pioneros de estos engendros fueron universitarios que se dedicaron a recopilar direcciones de la web, organizarlas y clasificarlas por temas. La forma de obtener los recursos era por medio de correo electrónico, es decir, era el *webmaster* o el responsable del mantenimiento quien mediante un mensaje daba de alta y decidía en qué categoría debía ir su web. Una vez que se tiene un formato de base de datos todo el árbol parece natural que se introduzca una herramienta de búsqueda por palabras y conceptos. Sin embargo, esta forma de alimentar los árboles tiene varios inconvenientes: el primero de ellos es que se necesita de un componente activo por parte de los creadores de información, que tienen que dar de alta sus web; el segundo de ellos es el mantenimiento, puesto que todos queremos estar, y damos alegremente de alta nuestras páginas, olvidándonos maliciosamente darlas de baja cuando ya no existen.

La otra forma es buscar en la Red. Existen programas que partiendo de una dirección IP[1] que se obtiene de un servidor de nombres de dominio, exploran todos los enlaces de la página web de partida y de forma recurrente, de tal manera un hilo lógico, no salta de la máquina de partida. De cada fichero HTML[2], que encuentra, introduce su dirección, título y el texto de las primeras líneas en una base de datos a la que se interrogará posteriormente. El punto en contra que tiene esta manera de mantener la base de datos es la duplicación de contenidos; por ejemplo, si una página tiene una referencia (enlace[3]) con el texto "PROGRASYS(c)" y apunta a otra página con el mismo título, aparecerán dos referencias al mismo contenido.

Existe otro tipo de buscadores que tan sólo actúan sobre un dominio. Este es el caso de motores de búsqueda que podemos encontrar en la páginas web de las grandes corporaciones, como Microsoft, IBM, Netscape o Sun.

El método de funcionamiento de un motor de búsqueda en Internet es el siguiente:

Toda búsqueda en uno de estos motores se realiza en tres pasos:

1. Consiste en recoger la información por parte del usuario;
2. Procesar el *query*[4] (la pregunta a la base de datos), y
3. Generar la página o páginas web con los resultados en forma de enlaces

El segundo es el más complicado debido a que buscar cadenas y subcadenas en texto en una base de datos de tamaño nada despreciable en un tiempo aceptable no es nada trivial. Es cierto que es un problema ampliamente estudiado y que existen algoritmos para todos los gustos (los cuales serán tratados más adelante), pero no deja de ser el paso más complicado.

La recogida de datos se realiza mediante formularios electrónicos codificados en HTML. Aunque las últimas tendencias apuntan a la utilización de lenguajes de mayor nivel, como Java[5] o controles ActiveX[6], la forma más sencilla y natural es utilizar las ventajas del lenguaje universal de la Web. El siguiente paso es transferir el dato buscado al programa que se encargue de encontrar todas las coincidencias en la base de datos. Del transporte de los datos se encarga un programa CGI (Common Gate Interfaces) que los pasa en forma de cadena al programa que se encarga de realizar la búsqueda. Esta aplicación está escrita en código nativo para la máquina donde corre el servidor web y tiene además la obligación de generar el código de la página donde se represente la respuesta. Por último, queda que el navegador cargue la página generada por el motor de búsqueda; de este trabajo se encarga el mismo CGI que recoge las claves.

Existe otra forma de crear un motor de búsqueda utilizando sólo tecnología ActiveX, Java o ambas simultáneamente. En este caso se debe incluir el *applet*[7] o el control en la página HTML, y será éste quien se encargue de buscar y generar la página de respuesta. El problema que tiene esta forma de actuar en Internet es que toda la base de datos debe viajar por la red y, dada la velocidad de transferencia usual, no es muy utilizado. Sin embargo para Intranets[8], donde las velocidades de transferencia son mucho mayores y la cantidad de información es menor, resulta una solución factible.

Tristemente, en cualquier ámbito de la vida es muy difícil encontrar a nadie que dé algo por nada. En los principios de esta red, la idea era la colaboración desinteresada entre los usuarios, el mítico *"hoy por ti, mañana por mí"*. Con el desembarco generalizado de los habitantes del mundo en la Red y sobre todo de las entidades comerciales y financieras más importantes del planeta, los intereses económicos también han llegado a Internet. Para los propietarios de este tipo de páginas, el medio

más factible de recopilar dinero es a través de publicidad, mediante pequeños gráficos colocados en la parte superior. Y desde el punto de vista del publicista, las páginas buscadoras son las más rentables, porque son, sin duda, las más visitadas.

La cantidad de motores de búsqueda que podemos encontrar en Internet aumenta cada día. Sin embargo la forma de buscar con ellas es tan sencillo que no hace falta nada más que visitar una de ellas y probar su funcionamiento. Por si acaso queda alguna duda, el texto buscado se introduce en la caja de texto; próximo a ésta encontraremos un botón serigrafiado o un gráfico con la palabras "Buscar", "Search", "Go" o "Go To Get It"; una vez introducida la cadena que se quiere buscar debemos pulsar este botón para que se inicie la búsqueda. Cerca podemos encontrar listas desplegables donde seleccionar opciones tales como el número de resultados máximo por página, dominios donde buscar, tipos de recursos (news, web, gopher…) o el tipo de lógica a emplear con los argumentos tipo.

Las respuestas vuelven en forma de página web; por cada resultado hay una pequeña descripción, el título, un enlace a la página y, a veces, un resultado que indica en forma de tanto por ciento la puntuación de la coincidencia.

Prácticamente todas las herramientas de búsqueda admiten la utilización de la lógica matemática, unos mediante listas desplegables en la que se debe introducir entre las distintas palabras separadas por espacios que pongamos en la caja de texto y otros mediante conjunciones en inglés mezcladas en el texto. Estos últimos, técnicamente más avanzados, permiten también el uso de paréntesis. Por norma, siempre se resuelven primero las operaciones que están entre paréntesis. Se suelen utilizar los 4 operadores básicos, tres de ellos binario (necesitan dos argumentos) y uno de ellos monario (necesita un argumento), que son:

AND: Equivale a la conjunción "y" e indica que han de aparecer en el mismo documento los dos términos que están a cada lado del operador. Por ejemplo, la cadena PC **AND** COMPUTERS indicará que sólo busque cadenas en las que aparezcan "PC" y "COMPUTERS", juntas o separadas.

OR: El equivalente en nuestra lengua es la conjunción "o"; indica que queremos buscar todas las cadenas en donde aparezca uno, otro o ambos argumentos. Por ejemplo PC **OR** COMPUTERS devolverá las cadenas que contengan la palabra "PC", las que contengan "COMPUTERS" y las que tengan "PC" y "COMPUTERS".

XOR: De este operador no hay equivalente claro en nuestro idioma; la mejor aproximación es un "o" exclusivo, es decir funciona como OR, salvo que se excluye el caso en el que aparezcan en la misma cadena los dos argumentos. Por ejemplo, PC **XOR** COMPUTERS devolverá las cadenas que contengan la palabra "PC" y las que contengan "COMPUTERS", pero no las que contengan "PC" y "COMPUTERS".

NOT: Es el único operador que tan sólo necesita un argumento y su función es negar lo que viene después. Por ejemplo, PC **AND NOT** COMPUTERS devolverá las cadenas en las que aparezca la palabra "PC" y no aparezca "COMPUTERS".

La combinación de estos elementos y la utilización de los paréntesis permite infinidad de combinaciones, teles como ORDENADOR **OR** (486 **OR** PENTIUM **OR** PENTIUM PRO).

METODOS DE BUSQUEDA EN PROGRAMAS DE INTELIGENCIA ARTIFICIAL

Debido a que la búsqueda es el núcleo de muchos procesos inteligentes, es adecuado estructurar los programas de I.A. de forma que se facilite describir y desarrollar el proceso de búsqueda. Los sistemas de producción proporcionan tales estructuras. Más abajo se introduce una definición de sistema de producción. Conviene no confundirse con otros usos de la palabra producción, tales como lo que se produce en una fabrica. Un sistema de producción consiste en:

- Un conjunto de reglas compuestas por una parte izquierda (un patrón) que determina la aplicabilidad de la regla, y una parte derecha, que describe la operación que se lleva a cabo si se aplica la regla[9]
- Una o más bases de datos/conocimiento que contengan cualquier tipo de información apropiada para la tarea en particular. Partes de la base de datos pueden ser permanentes, mientras que otras pueden hacer referencia sólo a la solución del problema actual. La información almacenada en estas bases de datos debe estructurarse de forma adecuada
- Una estrategia de control que especifique el orden en el que las reglas se comparan con la base de datos, y la forma de resolver los conflictos que surjan cuando varias reglas puedan ser aplicadas a la vez
- Un Aplicador de reglas

Hasta aquí la definición a sido muy general. También abarca una familia de intérpretes generales de sistemas de producción que incluye :

Lenguajes básicos de sistemas de producción como OPS5 y ACT

Sistemas más complejos, con frecuencia híbridos, denominados armazones de sistemas expertos, los cuales poseen entornos completos para la construcción de sistemas expertos basados en conocimiento.

Arquitecturas generales de resolución de problemas tales como SOAR, sistema basado en un conjunto específico de hipótesis motivadas cognoscitivamente por la naturaleza del problema a resolver.

Todos estos sistemas mantienen la arquitectura de un sistema, primero hay que reducirlo a una forma en la que pueda darse una definición precisa. Esto puede lograrse mediante la definición de un espacio de estados del problema (incluyendo los estados iniciales y finales), y de un conjunto de operadores para trasladarse a través del espacio. El problema se reduce entonces a buscar una ruta para trasladarse a través del espacio que una un estado inicial con un estado objetivo.

Estrategias de control

El primer requisito que debe cumplir una buena estrategia de control es que cause algún cambio. Considérese el problema de las jarras de agua del apartado anterior. Suponga que se implementa una sencilla estrategia de control que cada vez empiece por la primera regla de la lista y elija la primera que encuentre que es aplicable. Si se hace esto, nunca se encontrará la solución al problema. Se estará llenando la jarra de cuatro litro de agua indefinidamente. Las estrategias de control que no causan cambio de estado nunca alcanzan la solución.

El segundo requisito que debe cumplí una buena estrategia de control es que sea sistemática. He aquí otra simple estrategia de control para el problema de las jarras de agua: en cada ciclo, elegir aleatoriamente una de entre todas las reglas aplicables. Esta estrategia es mejor que la primera, produce cambios y puede encontrar la solución eventualmente. Sin embargo, puede volver al mismo estado varias veces durante el proceso y suele utilizar mucho más pasos de los necesarios. Debido a que la estrategia de control no es sistemática, es posible utilizar secuencias de operadores no apropiadas varias veces hasta encontrar finalmente la solución. El requisito de que una estrategia de control sea sistemática se corresponde con una necesidad de cambio local (en el curso de un paso sencillo) Una estrategia de control sistemática para el problema de las jarras de agua podría ser la siguiente: se construye un árbol cuya raíz sea el estado inicial; todas las ramificaciones de la raíz se generan al aplicar cada una de las reglas resultantes de la aplicación de todas las reglas adecuadas. En la figura se muestra el estado actual del árbol[10]. Se continúa con este proceso hasta que alguna regla produce un estado objetivo. Este proceso, denominado búsqueda primero en anchura (breadth-first search), se describe con precisión de la siguiente forma.

Algoritmo : Búsqueda primero en anchura

1. Crear una variable llamada LISTA-NODOS y asignarle el estado inicial.

2. Hasta que se encuentre un estado objetivo o LISTA-NODOS este vacía, hacer
   a) Eliminar el primer elemento de LISTA-NODOS y llamarlo E. SI LISTA-NODOS esta vacía, terminar
   b) Para que cada regla se empareje con el estado descrito en E hacer:
      I. Aplicar la regla para generar un nuevo estado
      II. Si el nuevo estado es un estado objetivo, terminar y devolver este estado
      III. En caso contrario, añadir el nuevo estado al final de LISTA-NODOS.

También pueden servir otras estrategias de control sistemáticas. Por ejemplo, se podría continuar por una sola rama del árbol hasta encontrar una solución o hasta que se tome la decisión de terminar la búsqueda por esa dirección. Terminar la búsqueda por una ruta tiene sentido cuando se llega a un callejón sin salida, se produce un estado ya alcanzado o la ruta se alarga mas de lo especificado en algún limite de "inutilidad". Si esto ocurre se produce una vuelta hacia atrás (backtracking)

Algoritmo: Búsqueda en profundidadd

1. Si el estado inicial es un estado objetivo, terminar y devolver un éxito
2. En caso contrario, hacer lo siguiente hasta que se marque un éxito o un fracaso:
   a) Generar un sucesor, E, del estado inicial. Si no existen mas sucesores, marcar un fracaso
   b) Llamar a la búsqueda primero en profundidad con E como estado inicial
   c) Si se devuelve un éxito, marcar un éxito. En caso contrario continuar con el ciclo

Al comparar estos dos sencillos métodos surgen las siguientes observaciones:

Ventajas de la búsqueda primero en profundidad

- Necesita menos memoria ya que solo se almacenan los nodos del camino que se sigue en ese instante. esto contrasta con la búsqueda primero en anchura en la que debe almacenarse todo el árbol que haya sido generado hasta ese momento
- Si se tiene suerte (o si se tiene cuidado en ordenar los estados alternativos sucesores), la búsqueda primero en profundidad puede encontrar una solución sin tener que examinar gran parte del espacio de estados. En el caso de la búsqueda primero en anchura debe examinarse todas las partes del árbol de nivel n antes de comenzar con los nodos de nivel n+1. Esto es particularmente relevante en el caso de que existan varias soluciones aceptables. La búsqueda primero en profundidad acaba al encontrar una de ellas

Ventajas de la búsqueda primero en anchura

- La búsqueda primero en anchura no queda explorando callejones sin salida. Esto se contrapone con la búsqueda primero en profundidad en la que se puede seguir una ruta infructuosa durante mucho tiempo, y quizás para siempre, antes de acabar en un estado sin sucesores. Esto es particularmente un problema en la búsqueda primero en profundidad si hay ciclos (p.ej. un estado tiene como sucesor un estado que es también uno de sus antecesores), a no ser que se tenga un cuidado especial en verificar tales situaciones
- Si existe una solución, la búsqueda primero en anchura garantiza que se logre encontrarla. Además si existen múltiples soluciones, se encuentra la solución mínima(es decir, la que requiere el mínimo numero de pasos).Esto esta garantizado pro el hecho de que no se explora una ruta larga hasta que se hayan examinado todas las rutas mas cortas que ella. En cambio en la búsqueda primero en profundidad es posible encontrar una solución larga en alguna parte del árbol, cuando puede existir otra mucho mas corta en alguna parte inexplorada del mismo

Lo deseable seria que se pudieran combinar estos dos métodos.

Búsqueda Heurística

Con el fin de resolver problemas complicados con eficiencia, con frecuencia es necesario comprometer los requisitos de movilidad y sistematicidad, y construir una estructura de control que no garantice encontrar la mejor respuesta pero que casi siempre encuentre una buena solución. De esta forma, surge la idea de heurística.

Una heurística es una técnica que aumenta la eficiencia de un proceso de búsqueda, posiblemente sacrificando demandas de completitud. Las heurísticas son como los guías de turismo: resultan adecuados en el sentido de que generalmente suelen indicar las rutas interesantes; son malos en el sentido de que pueden olvidar puntos de interés para ciertas personas. Otras pueden causar que una buena ruta sea pasada por alto.

Al utilizar heurísticas se pueden esperar buenas, aunque posiblemente no optimas, soluciones a problemas difíciles, tales como el del viajante de comercio, en un tiempo menor al exponencial. Existen además heurística generales que son utilizadas para un amplio dominio de acción. Pero es posible construir heurísticas de dominio especial para problemas específicos.

Heurística del vecino mas próximo (nearest neighbor heuristic)

Es un buen ejemplo de una heurística de propósito general para varios problemas combinatorios. Consiste en seleccionar en cada paso la alternativa localmente superior. Al aplicarla al problema del viajante de comercio, surge el siguiente proceso:

1. Seleccionar arbitrariamente una ciudad de comienzo
2. Para seleccionar la siguiente ciudad, fijarse en las ciudades que todavía no se han visitado y seleccionar aquella que sea mas cercana. Ir a esa ciudad
3. Repetir el paso 2 hasta que todas las ciudades hayan sido visitadas

Este procedimiento se ejecuta en un tiempo proporcional a N^2 , lo cual es una mejora significativa frente a N!, y es posible demostrar la existencia de un limite superior en el error cometido.

Es posible determinar el rango de error cometido, sin embargo en muchos problemas de I.A. no es posible dar esos limites tan tranquilizadores, esto es así por dos motivos :

- Para los problemas del mundo real, normalmente es difícil medir con precisión el valor de una solución particular
- Para los problemas del mundo real, normalmente es adecuado introducir heurísticas basadas en un conocimiento relativamente desestructurado

Existen dos formas fundamentales de incorporacion de conocimiento heuristico especifico del dominio a un proceso de búsqueda basado en reglas:

- En las mismas reglas. Pro ejemplo, las reglas de un sistema para jugar ajedrez podrían no solo describir el conjunto de movimientos legales, sino incluso un conjunto de movimientos “sensatos” determinados por el programador de las reglas
- Como una función heurística que evalúa los estados individuales del problema y determinar su grado de “deseabilidad”

Una función heurística es una correspondencia entre las descripciones de estados del problema hacia alguna medida de deseabilidad, normalmente representada por números. Los aspectos del problema que se consideran , como se evalúan estos aspectos, y los pesos que se dan a los aspectos individuales, se eligen de forma que el valor que la función heurística da a un nodo en el proceso de búsqueda, sea una estimación tan buena como sea posible para ver si ese nodo pertenece a la ruta que conduce a la solución.

El propósito de una función heurística es el de guiar el proceso de búsqueda en la dirección mas provechosa sugiriendo que camino se debe seguir primero cuando hay

mas de uno disponible. Cuando mas exactamente estime la función heurística los méritos de cada nodo del árbol (o grafo) mas directo sería el proceso de solución. En el caso extremo, la función heurística seria tan buena que esencialmente no se necesitara proceso de búsqueda pues el sistema encontraría directamente las solución. Sin embargo debido a bastantes problemas, el coste computacional para obtener los valores superaría el esfuerzo ahorrado en el proceso de búsqueda. Después de todo seria posible hacer una función heurística perfecta realizando una búsqueda completa desde el nodo en cuestión, determinando si conduce a una solución adecuada. En general, existe un compromiso entre el coste de evaluación de una función heurística y el ahorro de tiempo de búsqueda que proporciona la función

TECNICAS DE BÚSQUEDA HEURÍSTICA

Estas técnicas se denominan con frecuencia métodos débiles , estas técnicas proporcionan un marco donde situar el conocimiento del dominio específico, ya sea manualmente o como resultado de un aprendizaje automático . Es por ello que siguen formando el núcleo de la mayoría de los sistemas de I.A..

Algunos métodos de búsqueda Heurística son:

- Generación y prueba (generate-and-text)
- Escalada (hill climbing)
- Búsqueda el primero mejor (best-first search)
- Reducción del problema (problem reduction)
- Verificación de restricciones (constraint satisfaction)
- Análisis de medios y fines (means-ends analysis)

GENERACION Y PRUEBA

La estrategia de generación y prueba es la más simple de todas.

Algoritmo: Generación y prueba

Generar una posible solución. O generar un objetivo particular en el espacio del problema, o generar un camino a partir de un estado inicial.
Verificar si realmente el objetivo elegido es una solución comparándolo con el objetivo final o comparando el camino elegido con el conjunto de estados objetivos aceptables.

Si se ha encontrado la solución, terminar. Si no, volver al paso 1.

Si la solución existe se la encontrará en algún momento. Pero se puede consumir mucho tiempo. Es un algoritmo de búsqueda en profundidad ya que las soluciones

deben generarse antes que se comprueben, además puede funcionar de formas que genere las soluciones de manera aleatoria, sin la garantía de que alguna vez se encuentre la solución..

La forma más sencilla de implementar una generación y prueba sistemática es mediante un árbol de búsqueda primero en profundidad y con vuelta-atrás. También se puede recorrer un grafo en lugar de un árbol, para problemas sencillos es una técnica razonable, para problemas complicados este método no es muy eficiente pero combinado con técnicas de restricción en el espacio de búsqueda puede resultar muy eficaz.

ESCALADA

Este método es una variante del de generación y prueba; existe realimentación a partir del procedimiento de prueba para ayudar al generador por cuál dirección debe moverse en el espacio de búsqueda. La escalada se usa frecuentemente cuando se dispone de una buena función heurística para evaluar los estados . Pero cuando no se dispone de otro tipo de conocimiento provechoso.

Escalada simple es la forma más sencilla de implementar el método de escalada.

Algoritmo: Escalada simple.

Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como actual.

Repetir hasta que se encuentre la solución o hasta que no queden nuevos operadores del estado actual:

Seleccionar un operador que no haya sido aplicado con anterioridad al estado actual y aplicarlo con anterioridad al estado actual y aplicarlo para generar un nuevo estado.
Evaluar el nuevo estado.

Si es un estado objetivo devolverlo y terminar

Si no es un estado objetivo, pero es mejor que el estado actual, convertirlo en estado actual.

Si no es mejor que el estado actual continuar con el bucle.

A diferencia del de generación y prueba , este utiliza una función de evaluación como la forma de introducir conocimiento específico de la tarea realizada en el proceso de control. La utilización de este conocimiento hace que sean métodos de búsqueda heurística .

Para que este algoritmo funcione es necesario definir el término mejor. Dependiendo del caso sería un valor más alto o un valor más bajo de una función heurística. La interpretación depende del programador.

Escalada por la máxima pendiente

Una variación del método de la escalada simple consiste en considerar todos los posibles movimientos a partir del estado actual y elegir el mejor de ellos como nuevo estado. También se denomina búsqueda del gradiente (gradient search).

Algoritmo: Escalada por la máxima pendiente

Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario. Continuar con el estado inicial como estado actual.

Repetir hasta que se encuentre una solución o hasta que una iteración completa no produzca un cambio en el estado actual:

Sea SUCC un estado tal que algún posible sucesor del estado actual sea mejor que este SUCC.

Para cada operador aplicado al estado actual hacer lo siguiente:

Aplicar el operador y generar un nuevo estado.

Evaluar el nuevo estado. Si es un estado objetivo, devolverlo y terminar. Si no, compararlo con SUCC, si es mejor, asignar a SUCC este nuevo estado. Si no es mejor, dejar SUCC como esta.

Si SUCC es mejor que el estado actual, hacer que el estado actual sea SUCC.

Tanto la escalada básica como la de la máxima pendiente pueden no encontrar una solución. Cualquiera del los dos algoritmos puede acabar sin encontrar un estado objetivo, y en cambio encontrar un estado del que no sea posible generar nuevos estados mejores que él, esto ocurre si el programa se encuentra con un máximo local, una meseta o una cresta.

Un máximo local es un estado que es mejor que todos sus estado vecinos, pero no es mejor que otros estados de otros lugares. Cuando se encuentran en la cercanía de la solución son frustrantes y se denominan estribaciones (foot-hills).

Una meseta (plateau) es un área del espacio de búsqueda en la que el conjunto de estados vecinos posee el mismo valor.

Una cresta (ridge) es un tipo especial de máximo local. Es un área del espacio de búsqueda más alta que las áreas circundantes y que además posee en ella misma una inclinación.

Existen algunas formas de evitar estos problemas:

Volver atrás hacia algún nodo anterior e intentar seguir un camino diferente. Es razonable si el nodo posee otra dirección que de la impresión de ser tan prometedora, o casi tan prometedora, como la que se elogio. Se debe mantener una lista de caminos que casi se han seguido y volver a uno de ellos. Se utiliza para superar máximos locales.

Realizar un gran salto en alguna dirección para intentar buscar una nueva parte del espacio de búsqueda. Este método está especialmente indicado para superar mesetas.

Si la única regla aplicable describe pequeños pasos. Aplicarla varias veces en la misma dirección.

Aplicar dos o más reglas antes de realizar la evaluación. Esto se corresponde con movimientos en varias direcciones a la vez. Este método es especialmente bueno para superar las crestas.

Incluso con estas medidas de apoyo la escalada no siempre es muy eficaz.

Especialmente es muy inadecuada cuando el valor de la función heurística cambia bruscamente al alejarse de una solución. Esto ocurre frecuentemente cuando aparece algún tipo de efecto umbral.

El método de la escalada comparte con otros métodos locales la ventaja de provocar una explosión combinatoria menor que los métodos globales. Sin embargo tiene falta de garantías de que va ha resultar eficaz.

No siempre es posible construir una función heurística perfecta, ya que sigue siendo muy ineficiente con espacios problemas grandes y escabrosos. Sin embargo, suele ser adecuado si se combina con otros métodos que consigan que se mueva correctamente por la vecindad en general.

Enfriamiento simulado

El enfriamiento simulado es una variante de la escalada, que al comienzo del proceso , pueden realizarse algunos movimientos descendentes. La idea consiste en realizar una exploración lo suficientemente amplia al principio, ya que la solución final es relativamente insensible con el estado inicial. Eliminando la probabilidad de caer en un máximo local, una meseta o una cresta. Se usa el término función objetivo en lugar de función heurística.

El enfriamiento simulado (Kirkpstrick et al., 1983) se basa en el proceso físico de la aleación, en la que ciertas sustancias físicas se funden, para llegar a un proceso gradual de enfriamiento. De está forma, el proceso consiste el el descenso por un valle en el que la función objetivo es el nivel de energía. Las sustancias físicas normalmente evolucionan hacia configuraciones de más baja energía, de forma que el descenso ocurre de forma natural. Pero existen probabilidades de que se produzcan transiciones hacia estados con energía más alta y está dada por la función:

$$p=e^{(dE/kT)}$$

Donde dE es el cambio positivo en el nivel de energía, T es la temperatura y k es la constante de Boltzmann. De está forma , estos saltos son más probables en los comienzos del proceso, cuando la temperatura permanece alta, y son más improbables al final, cuando la temperatura baja. Es decir que los grandes saltos ascendentes solo se permiten al principio, pero conforme el proceso va progresando, sólo se permitirán movimientos ascendentes relativamente pequeños hasta que finalmente el proceso converja a una configuración de mínimo local.

La velocidad con que se enfría el sistema a lo largo del proceso se denomina programa de enfriamiento, se alcanza un mínimo local pero no global, si se utiliza un programa más lento, es más probable que corresponda a un mínimo global. El plan de enfriamiento óptimo para cada problema de enfriamiento específico se suele hallar de forma empírica.

Sin embargo, es necesario elegir un plan para los valores de T.

El algoritmo de enfriamiento simulado difiere sólo un poco del procedimiento de cada escalada simple. Aparecen tres diferencias:

1. El programa de enfriamiento tiene que mantenerse.
2. Los movimientos hacia estados peores pueden aceptarse.
3. Es una buena idea hacer un mantenimiento, además del estado actual, del mejor estado que se haya encontrado. Si el estado final es peor que el estado anterior el estado anterior estará disponible aún.

Algoritmo: Enfriamiento simulado

Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar.

En caso contrario, continuar con el estado inicial como estado actual.

Inicializar EL-MEJOR-HASTA-AHORA con el estado actual.

Inicializar T de acuerdo con el programa de enfriamiento.

Repetir hasta que se encuentr4e solución o hasta que no queden operadores que aplicar al estado actual.

Seleccionar un operador que aún no se haya aplicado al estado actual para producir un estado nuevo.

Evaluar el nuevo estado. Calcular:

dE=(valor del estado actual)-(valor del nuevo estado)

Si el nuevo estado es un estado objetivo, devolverlo y terminar

Si no es un estado objetivo, pero es mejor que el estado actual, convertirlo en estado actual. Hacer también que EL-MEJOR-HASTA-AHORA sea el nuevo estado.

Si no es mejor que el estado actual, convertirlo en estado actual con la probabilidad de p'. Este paso se implementa invocando un generador de número aleatorios que genere un número de rango [0,1] si este número es menor que p', se acepta el movimiento, si no, no se hace nada

Revisar T cuando sea necesario, de acuerdo con el programa de enfriamiento.

Devolver como respuesta EL-MEJOR-HASTA-AHORA.

El programa de enfriamiento se compone de tres partes:

El valor inicial de la temperatura.

El criterio que se sigue para decidir cuándo se va a reducir la temperatura del sistema.

La magnitud del decremento que sufre la temperatura en cada cambio.

Existe también un cuarto punto que es cuando terminar, se utiliza en los problemas en los que el número de movimientos que pueden realizarse en un cierto estado, es muy elevado.

BÚSQUEDA EL PRIMERO MEJOR
Los Grafos O

Se trata de de representar una forma de combinar las ventajas que presentan tanto la búsqueda primero en anchura así como la primero en profundidad.

La ventaja de la búsqueda primero en profundidad es de que permite encontrar una solución sin tener que expandirse completamente por todas las ramas, mientras que la búsqueda en anchura presenta la ventaja de que no se queda atrapada en callejones sin salida.

Una de las formas de combinar las ventajas de las búsquedas consiste en seguir un único camino a la vez y cambiar la ruta cuando haya mejores oportunidades.

En la implementación de un procedimiento de búsqueda obre un grafo necesitamos dos listas de nodos :

Nodos abiertos.- Son nodos que se han generado y a los que se les ha aplicado la función heurística, pero que todavía no han sido examinados. Esta lista de nodos abiertos es en realidad una cola en donde se da mayor prioridad a los nodos que presenten mejores ventajas.

Nodos Cerrados.- Son los nodos a los que se les ha examinado es necesario tenerlos en memoria ya que a veces se necesita hacer búsqueda en un grafo y no en un árbol.

A parte de conocer los nodos abiertos y cerrados se necesita un función heurística que haga una estimación de los méritos de cada nodo.

Algoritmo : Búsqueda el primero mejor

Comenzar con abiertos conteniendo solo el estado inicial

Mientras no se queden nodos abiertos en hacer :

Tomar el mejor de abiertos.

Generar sus sucesores

Para cada sucesor hacer

Si no se ha generado con anterioridad, evaluarlo, añadirlo a Abiertos y almacenar en su padre.

Si ya se ha generado antes, cambiar al padre si el nuevo camino es mejor que el anterior.

En este caso, se actualiza el costo empleado para alcanzar el nodo y a los sucesores que pudiera tener.

ALGORITMO A*

1.- Empieza con abiertos conteniendo solo el nodo inicial. Poner el valor g de ese nodo a 0 (cero), su valor h' al que corresponda, y su valor f´ a h´, es decir, a h´. Inicializar cerrados como una lista vacía.

2.- Repetir el siguiente procedimiento hasta encontrar un nodo objetivo : si no existen nodos en abiertos, ignorar del fallo. Caso contrario tomar aquel nodo de abiertos que tenga el menor valor de f´. Llamarle el MEJOR-NODO. Quitarlo de Abiertos y ponerlo en Cerrados. Mirar si MEJOR-NODO es un nodo objetivo, si lo es salir e informar de lo sucedido. Bien si MEJOR-NODO lo único que desea es el Nodo, o todo el camino empleado en unir y saber la ruta entre el estado inicial y MEJOR-NODO si se está interesado en la ruta. Caso contrario generar los sucesores de de MEJOR-NODO pero sin colocar MEJOR-NODO apuntando a ellos.

   a.- Para cada sucesor hacer lo siguiente :

   b.- Poner a SUCESOR apuntando a MEJOR-NODO. Estos enlaces hacia atrás hacen posible que se recupere el camino una vez que se ha encontrado la solución.

   c.- Calcular g(SUCESOR)=g(MEJOR-NODO) + el coste de ir desde MEJOR-NODO hasta SUCESOR.

   d.- Mirar si SUCESOR está contenido en ABIERTOS (es decir ya ha sido generado pero no procesado}. Si es así se denomina a este nodo VIEJO puesto que el nodo ya existe en el grafo se puede desecharlo y añadir VIEJO a la lista de nodos SUCESORES de MEJOR-NODO ahora debe decidirse si el enlace paterno de VIEJO debe apuntar a MEJOR-NODO.

   e.- Si SUCESOR no se encontraba en ABIERTOS, mirar i se encuentra en CERRADOS. Si es así, llamar VIEJO al nodo de CERRADOS y añadirlo a la lista de sucesores de MEJOR-NODO.

   f.- Si SUCESOR no estaba ya en ABIERTOS o en CERRADOS, ponerlos en ABIERTOS y añadirlo a la lista de sucesores de MEJOR-NODO. Calcular f´(SUCESOR) = g´(SUCESOR)+ h´(SUCESOR).

AGENDAS

Un algoritmo de búsqueda mediante agenda se basa para organizare en las justificaciones que este utilice para realizarla y un valor que represente un peso total de las evidencias que nodo dirá que la tarea sea útil.

Algoritmo :

1.- Mientras se alcance un objetivo o la agenda esté vacía hacer :

   a.- Elegir la tarea m{as prometedora de la Agenda

   b.- Ejecutar la tarea asignándole un número de recursos definidos por su importancia entre estos tenemos el tiempo y el espacio. La ejecución de la tarea generará nodos sucesores, para cada uno de ellos hacer lo siguiente :

I.- Mirar si ya se encuentra en la agenda. Si es así, mirar si la razón por la que debe realizarse ya está en la lista de justificaciones. Si lo está ignorar la evidencia actual. Si la justificación no lo está añadir a la lista, y si la tarea todavía no está, insertarla.

II.- Calcular el valor de la nueva tarea, combinando la evidencia de todas sus justificaciones . No es necesario que todas las justificaciones tengan un igual peso. A menudo es útil asociar a cada justificación una medida de su valor, e este paso se combinan esas medidas para predecir un valor total de la tarea.

Una de las formas de encontrar la tarea más prometedora por agendas es mantener una agenda ordenada por valores : así cuando se crea una nueva tarea, se inserta en su lugar adecuado. Cuando se cambian las justificaciones de una tarea se recalcula su valor y se traslada a su nueva localización en la lista. Pero se gasta tiempo ya que se tiene a la agenda siempre en orden, lo cual es innecesario ya que lo único que necesita saber es cual es el primer elemento correcto.

REDUCCION DE PROBLEMAS

Como hemos vista se han utilizado búsquedas con Grafos O . en los cuales se desea encontrar un camino simple hasta un objetivo. Estas estructuras reflejan el hecho de que se conoce como llegar desde un nodo hasta un estado objetivo si se puede descubrir como llegar desde ese nodo hasta un objetivo a lo largo de las ramas que cuelgan de el.

GRAFOS Y-O

El grafo Y-O es una estructura que es útil para la representación de la solución en problemas que pueden resolverse descomponiéndoles en conjuntos más pequeños, cada uno de los cuales debe a su vez resolverse. Esta descomposición genera dos arcos denominados arcos Y, cada uno de estos puede apuntar a cualquier número de nodos sucesores de manera que todos ellos deben resolverse para el arco que apunte a una solución.

Objetivo: Adquirir una T.V.
Objetivo : robar una T.V.
Objetivo : Ganar dinero
Objetivo : Comprar una T.V.

Para encontrar soluciones en un grafo Y-O se necesita un algoritmo similar a una búsqueda de el primero mejor, pero con la capacidad de manipulación apropiada de los arcos Y. Este algoritmo deberá encontrar un camino desde el nodo inicial del grafo hacia un conjunto de nodos que representen los estados solución. Nótese que puede resultar necesario considerar más de un estado solución ya que cada brazo de un arco Y puede conducir a su propia solución.

REFERENCIAS

[1] Dirección IP
[2] HTML
[3] Enlace
[4] Query
[5] Lenguaje Java
[6] Controles Activex
[7] Applet
[8] Intranet
[9] Esta opción sobre el uso de las partes izquierda y derecha es natural para las reglas hacia adelante. Muchos sistemas de reglas hacia atrás intercambian estas partes
[10] Las reglas 3,4,11 y 12 se ignoran a la hora de construir el árbol de búsqueda