1

Planteamiento general

Aproximación a los Algoritmos

Hoy en día, y de acuerdo con la nomenclatura para los campos de la Ciencia y Tecnología (según proyecto de la UNESCO), dentro del área de las Matemáticas, las Ciencias de la Computación contienen, al menos, 26 materias características, lo que les confiere un carácter marcadamente heterogéneo. Aunque como consecuencia de esta heterogeneidad no existe una definición de Ciencias de la Computación que se acepte universalmente, si podemos hacer un breve recorrido por las más destacadas, a fin de sacar una idea lo más aproximada posible del contenido de la materia. En este sentido consideraremos que el objeto de las Ciencias de la Computación es la resolución de problemas por medio de un ordenador, es decir, las Ciencias de la Computación tratan, por una parte, del análisis de los problemas y, por otra, de como escribir programas que permitan que estos sean resueltos mediante un ordenador. De acuerdo con esta definición, los ordenadores se presentan como una herramienta, siendo el tema fundamental de las Ciencias de la Computación el de la resolución de problemas. Hay que indicar, no obstante, que en el ámbito de influencia anglosajón el término "Computer Science" es algo más amplio y equivale al marco que aquí se le da al de Informática, incluyendo a los propios ordenadores como objeto de estudio. Esto puede comprobarse viendo como en "The Carnegie Mellon Curriculum for Undergraduate Computer Science" se establece que "Las Ciencias de la Computación consisten en el estudio de los ordenadores y en el de los fenómenos relativos a la computación, los algoritmos, los programas y la programación".

Pero desde el punto de vista algo más restringido que aquí nos interesa, la ACM establece: La disciplina Ciencias de la Computación es "el estudio sistemático de los procesos algorítmicos que describen y transforman información: teoría, análisis, diseño, eficiencia, implementación, y aplicación." Así, no debemos entender que el objetivo de las Ciencias de la Computación sea la construcción de programas sino "el estudio sistemático de los algoritmos y estructuras de datos, y específicamente de sus propiedades formales". Pero esto no significa, desde luego, que no se deba prestar atención a la construcción de programas, ya que en definitiva el programa es el último paso a realizar para la resolución de un problema mediante ordenador.

El verdadero origen de la palabra algoritmo (o algorismo durante mucho tiempo) no se conoció hasta el siglo XIX, aunque desde los tiempos de Leibnitz era de uso corriente. Se pensaba en una degeneración del termino logaritmo, y se daba por seguro que estaba relacionada con el de arithmos (número). Fueron los orientalistas quienes explicaron el acertijo. En efecto, uno de los más grandes

matemáticos árabes del siglo IX de nuestra era, Abu ‘Abd Allah Muhhamad ibn Musa al-Khwarizmi (literalmente: Padre de Abdullah, Mohamed, hijo de Moisés, nativo de Khwarizm hoy Khiwa) con su obra “Kitab al-jabr wa’l-muqabala” (Reglas de Ecuaciones y Restauración), ayudó a difundir las matemáticas árabes por el mundo occidental a tal grado que del título de su obra se ha desprendido el término álgebra (al-jabr). Con el paso del tiempo y por defectos de pronunciación, su nombre se difundió simplemente como Al-Juarismi y de éste los términos guarismo y algorismo (usados para referirse a cualquier método de cómputo usando la notación arábiga de numeración). El término algorismo también fue corrompido en su pronunciación hasta derivar uno más difundido en latín como algoritmus, empleado desde el siglo XVII por los matemáticos para referirse a procedimientos de cálculo, a raíz de la traducción al latín de otro de los libros de nuestro protagonista, que se tituló “Algoritmi de Numero Indorum”. Finalmente, la palabra que conocemos no apareció en un diccionario sino hasta la edición de 1957 del “Webster´s New World Dictionary”.

Aclarado pues ese origen, de modo algo formal, podemos definir el concepto algoritmo que nos interesa como una secuencia ordenada de pasos, exentos de ambigüedad, tal que al llevarse a cabo con fidelidad, dará como resultado que se realice la tarea para la que se ha diseñado, (se obtenga la solución del problema planteado) en un tiempo finito.

Desde esta perspectiva, para resolver un problema con un ordenador es necesario, en principio, diseñar un algoritmo que describa la forma en que debe efectuarse el proceso de hallar la solución y posteriormente expresar cada uno de esos pasos de la forma adecuada para que la maquina lo pueda llevar a cabo, esto es, expresar el algoritmo como un programa en un lenguaje de programación. Por ultimo, lograr que el ordenador ejecute el programa correctamente. El algoritmo es, por tanto, un concepto central de las Ciencias de la Computación.

Conceptual y formalmente los algoritmos son independientes, tanto del lenguaje de programación en que se expresen posteriormente, como del propio ordenador que los ejecuta. Ello, entraña la posibilidad de diseñar algoritmos y estudiarlos independientemente de la tecnología del momento: los resultados han de continuar siendo validos a pesar de la aparición de nuevos ordenadores y nuevos lenguajes de programación.

No obstante, el desarrollo de nuevos lenguajes de programación y nuevos ordenadores puede ser y es una preocupación legitima de las Ciencias de la Computación, como un medio para conseguir mayor eficiencia en la resolución de los problemas. El objetivo de desarrollar algoritmos para resolver problemas, ha sido siempre considerado por los matemáticos como especialmente importante. A través de los siglos, los intentos por desarrollar algoritmos capaces de resolver cualquier problema no han tenido éxito. Básicamente debido a Hilbert, durante muchos años se creyó que si cualquier problema podía iniciarse de manera precisa, entonces con suficiente esfuerzo sería posible encontrar una solución con el tiempo (o tal vez podría proporcionarse en el transcurso del tiempo una prueba de que no existe solución). En otras palabras, se creía que no había problema que fuera tan intrínsecamente difícil que en principio nunca pudiera resolverse.

En la década de los 30 se produjeron una serie de investigaciones, que mostraron la inexistencia de un algoritmo del tipo que buscaba Hilbert.

Posteriormente, a partir de la formalización del concepto de algoritmo, se planteo la cuestión de clasificar los problemas según que se pueda siempre encontrar la solución por medio de un algoritmo (problemas computables) o que no se pueda asegurar que existan algoritmos que siempre produzcan una solución (problemas no computables).

Al principio de la década de los 50, la necesidad de resolver problemas prácticos, en su mayoría problemas computables de tamaño cada vez mayor, junto con el comienzo del desarrollo de los ordenadores, y la posibilidad de utilizar estos para la resolución de los problemas, cambio el interés principal, del estudio de la computabilidad o no computabilidad de los problemas, al estudio de la complejidad de los problemas computables, es decir al estudio de como encontrar algoritmos que no solo hallen una solución, sino que la hallen en el mínimo número de operaciones (en tiempo mínimo) y utilizando el mínimo espacio posible, en otras palabras, buscar algoritmos mas eficaces.

Posteriormente se clasificaron los problemas computables, en dos tipos: aquellos para los que existía un algoritmo de complejidad polinómica (en el tamaño de la entrada del problema) y aquellos problemas para los que parece que los mejores algoritmos tienen una complejidad exponencial. Al principio de los años 70, dichos tipos de problemas cristalizaron en las clases P y NP.

Actualmente, el sentido moderno de algoritmo es algo bastante similar a receta, proceso, método, técnica, procedimiento, rutina, con la excepción de que la misma palabra algoritmo connota algo un poco diferente de todo eso. Además de ser un conjunto finito de reglas que constituyen una secuencia de operaciones encaminadas a resolver un tipo especifico de problema, un algoritmo tiene cinco características primordiales.

a) Finitud, de modo que un algoritmo debe terminar siempre tras un número finito de etapas.
b) Especificidad, en el sentido que cada etapa debe estar precisamente definida; las acciones que hay que llevar a cabo deben estar rigurosamente especificadas para cada caso.
c) El input, es decir los valores que se le dan inicialmente antes de que el algoritmo comience. Estos inputs se toman de conjuntos de objetos preespecificados.
d) El output, es decir, resultados que son cantidades específicamente relacionadas con los inputs.
e) Efectividad. Se espera generalmente que un algoritmo sea efectivo, lo que significa que todas las operaciones que hay que realizar en el algoritmo deben ser lo suficientemente básicas como para que, en principio, se hagan exactamente y en un periodo finito de tiempo por un hombre que solo use lápiz y papel.

Hay que destacar que la característica de finitud no es realmente fuerte en la práctica. Efectivamente, un algoritmo útil debería requerir no ya un número finito de etapas, sino un número asumible de ellas. Por ejemplo, hay un algoritmo que determina cuando en el ajedrez, desde cierta posición, las blancas ganaran obligatoriamente. Se trata de un algoritmo que resuelve un problema de gran interés, pero del que no se consigue conocer la respuesta por el enorme tiempo de ejecución que requiere.

En la practica no solo queremos algoritmos, sino que queremos buenos

algoritmos en algún sentido propio de cada usuario. Un criterio de bondad es la longitud del tiempo consumido para desarrollar el algoritmo; esto puede expresarse por el número de veces que se ejecuta cada etapa. Otros criterios son la adaptabilidad del algoritmo a los ordenadores, su simplicidad y elegancia o el costo económico que su confección y puesta a punto puede acarrear.

A menudo tendremos varios algoritmos para un mismo problema, y deberemos decidir cual es el mejor o cual es el que tenemos que escoger, según algún criterio que habrá que fijar, para resolverlo. Esto nos lleva al importante e interesante campo del análisis de Algoritmos objeto de este curso. Esquemáticamente, el problema que se quiere resolver consiste en, dado un algoritmo, determinar ciertas características que sirven para evaluar su rendimiento.

El estudio de los algoritmos incluye el de diferentes e importantes áreas de investigación y docencia. Globalmente hablando, ese estudio de los algoritmos se suele llamar Algorítmica, y desde un punto de vista atomizado y lógico, en esa disciplina se distinguen las cinco siguientes áreas mas destacadas:

a) La construcción. El acto de crear un algoritmo es un arte que nunca debiera automatizarse. Pero, indudablemente, no se puede dominar si no se conocen a la perfección las técnicas que existen para el diseño de los algoritmos. El área de la construcción de algoritmos engloba el estudio de los métodos que facilitando esa tarea se han demostrado en la practica mas útiles.
b) La expresión. El motivo mas importante de la programación Estructurada es que los algoritmos tengan una expresión lo mas clara y concisa posible. Siendo este un tema central del estudio de los algoritmos, requerirá que se le dedique tanto esfuerzo como sea necesario, a fin de conseguir lo que se suele denominar un buen estilo.
c) La validación. Cuando se ha construido el algoritmo, se hace necesario demostrar que calcula correctamente sobre inputs legales, lo que se conoce como validar el algoritmo. El propósito de la validación es el de asegurar que el algoritmo trabajara correctamente independientemente de la casuística que suponga el lenguaje de programación empleado, o la tecnología hardware que se use. Cuando se ha demostrado la validez del método, es cuando puede escribirse el programa, comenzando una segunda fase del trabajo.

   Esa fase es la de la verificación del programa. Una demostración de la corrección exige que la solución se establezca en dos formas. Una consiste en expresar el programa como un conjunto de asertos, dados en términos del cálculo de predicados, sobre las entradas, las salidas y las variables. La segunda es la llamada especificación, y también se suele expresar en términos del cálculo de predicados. La demostración consiste en demostrar que esas dos formas son equivalentes de modo que para una misma entrada, dan una misma salida.
d) El análisis. Cuando se ejecuta un algoritmo se hace uso de la CPU para realizar operaciones, y se usa la memoria para disponer del programa y de sus datos. El análisis de algoritmos se refiere al proceso de determinar cuándo tiempo de cálculo y cuanto almacenamiento requerirá un algoritmo. Uno de los resultados mas valiosos de este tipo de estudios, es que nos permite realizar juicios cuantitativos sobre el valor de un algoritmo sobre otros. Otra consecuencia importante de dicho estudio es que puede permitirnos predecir

si nuestro software reunirá algún requisito que pueda existir. Las cuestiones típicas dentro de este área se refieren al estudio del funcionamiento de un algoritmo en el caso mas favorables, en el mas desfavorable y en el promedio.

e) El test de los programas. El test de un programa es en realidad la última fase que se realiza. Básicamente supone la corrección de los errores que se detectan, y la comprobación del tiempo y espacio que son necesarios para su ejecución.

Elección de un algoritmo

Supongamos que ante un cierto problema tenemos varios algoritmos para emplear. El problema es que algoritmo elegir. Tomemos, por ejemplo, el caso de la multiplicación de dos enteros positivos, operación para la que no disponemos de maquina alguna, pero si de lápiz y papel. El algoritmo clásico de la multiplicación es bien conocido, por lo que no entraremos en detalles. Sin embargo, hay un algoritmo un poco diferente para hacer esa misma tarea, es el llamado algoritmo de multiplicación a la rusa e, incluso para el caso de enteros muy grandes, hay algoritmos mas eficientes que estos.

El algoritmo de multiplicación a la rusa consiste en lo siguiente,

1) Escribir multiplicador y multiplicando en dos columnas
2) Hasta que el número bajo el multiplicador sea 1, repetir:
   a) Dividir el número bajo el multiplicador por 2, ignorando los decimales
   b) Doblar el número bajo el multiplicando sumándolo a si mismo
   c) Rayar cada fila en la que el número bajo el multiplicador sea par, o añadir los números que queden en la columna bajo el multiplicando.

Supuesto que tenemos que multiplicar 73 (multiplicador) por 28 (multiplicando), tendríamos el siguiente desarrollo del algoritmo.

Multiplicador	Multiplicando	Resultado	Resultado acumulado
73	28	28	28
36	56	--	28
18	112	--	28
9	224	224	252
4	448	--	252
2	896	--	252
1	1792	1792	**2044**

Así, si tenemos varios algoritmos resolviendo un mismo problema, queremos saber con cual de ellos nos quedamos, lo que explica de nuevo que nuestro objetivo no son los programas, sino los algoritmos.

Problemas y casos

El algoritmo de la multiplicación a la rusa, no es exactamente un modo de multiplicar dos números concretos, sino que da la solución para el problema de

multiplicar dos enteros positivos, por lo que diremos que (73,28) es un caso de este problema.
Los problemas más interesantes incluyen una colección infinita de casos. Sin embargo, ocasionalmente, consideraremos problemas finitos, como puede ser el ajedrez. Un algoritmo debe trabajar correctamente en cualquier caso del problema en cuestión. Para demostrar que un algoritmo es incorrecto, solo necesitamos encontrar un caso del problema que no produzca la respuesta correcta. Por otro lado, usualmente es más difícil probar la corrección de un algoritmo. Cuando especificamos un problema, es importante definir el dominio de definición, es decir, el conjunto de casos que se considera. La diferencia que existe entre algoritmos y programas se pone de manifiesto porque cualquier sistema de computo real tiene un limite sobre el tamaño de los casos que puede manejar. Sin embargo, este límite no puede atribuirse al algoritmo que escojamos para usar.
El tamaño de un caso x se corresponde formalmente al número de bits necesarios para representar el caso en un ordenador usando algún código precisamente definido y razonablemente compacto. Para hacer nuestro análisis mas claro, sin embargo, a menudo usaremos la palabra tamaño entendiendo cualquier entero que, de algún modo, mida el número de componentes en un caso. Por ejemplo, cuando hablemos de ordenación, un caso considerando n ítems, generalmente se considerara como de tamaño n, aunque cada ítem pueda necesitar mas de un ítem para ser representado. Cuando a veces hablamos de problemas numéricos, damos la eficiencia de nuestros algoritmos en términos del valor del caso que se esta considerando, mas que sobre su tamaño (que es el número de bits necesarios para representar este valor en binario).

Análisis del caso peor y promedio.

El tiempo consumido por un algoritmo puede variar mucho entre dos casos diferentes del mismo tamaño. Para ilustrar esto, consideremos los dos siguientes algoritmos elementales de ordenación

```
PROCEDIMIENTO INSERCION (T[1..n])
  for i := 2 to n do
     x:= T[i]; j := i-1
     while j > 0 and x < T[j] do T[j+1] := T[j]
                              j := j-1
     T[j+1] := x
PROCEDIMIENTO SELECCION (T[1..n])
  for i:= 1 to n-1 do
  minj := i; minx := T[i]
  for j := i+1 to n do
     if T[j] < minx then minj := j
                         minx := T[j]
  T[minj] := T[i];
  T[i] := minx
```

Sean U y V dos arrays de n elementos, tales que U esta ya ordenado en orden ascendente, mientras que V lo esta en orden descendente. Un sencillo ejemplo sirve para comprobar que los dos algoritmos anteriores consumen más tiempo sobre V que sobre U. En efecto V representa el peor caso para estos dos algoritmos: Ningún array de n elementos requiere más trabajo. No obstante, el tiempo requerido por el algoritmo de ordenación por selección no es muy sensible al orden original del array a ordenar: el test

if T[j] < minx

se ejecuta exactamente el mismo número de veces en cualquier caso. La variación en el tiempo de ejecución solo se debe al número de veces que se ejecutan las asignaciones en la parte then de este test. Para verificar esto, se programó este algoritmo y se encontró que el tiempo requerido para ordenar un número dado de elementos usando el método de selección no variaba en mas de un 15% independientemente del orden inicial de los elementos a ordenar.

La situación es bastante diferente si comparamos los tiempos consumidos por el algoritmo de ordenación por inserción sobre U y V. Por un lado Insercion(U) es muy rápido, porque la condición que controla el ciclo while es siempre falsa, y por tanto el algoritmo se ejecuta en tiempo lineal. Por otro lado, Insercion(V) consume un tiempo cuadrático, ya que el ciclo while se ejecuta i-1 veces para cada valor de i. La variación en el tiempo es, por tanto, bastante considerable, y además, aumenta con el número de elementos a ordenar. Una implementación del algoritmo probo que Insercion(U) consumía menos de 1/5 de segundo si U es un array de 5.000 elementos ya ordenados en orden ascendente, mientras que Insercion(V) consumía tres minutos y medio cuando V es un array de 5.000 elementos ordenados descendentemente.

Si pueden ocurrir tan grandes variaciones, ¿como podremos hablar del tiempo consumido por un algoritmo solo en función del tamaño del caso que consideremos?. Consideraremos el peor caso del algoritmo, es decir, para cada tamaño solo consideraremos aquellos casos de ese tamaño en los que el algoritmo requiera el máximo tiempo. Así, por ejemplo, diremos que la ordenación por inserción tiene tiempo cuadrático en el peor de los casos.

El análisis del peor caso es apropiado para un algoritmo cuyo tiempo de respuesta es critico. Por ejemplo, si la cuestión fuera controlar una planta de energía nuclear, sería crucial conocer un límite superior para el tiempo de respuesta del sistema, independientemente del caso que se considerara. Por otro lado, en situaciones en las que un mismo algoritmo hay que usarlo muchas veces o en muchos casos distintos, puede ser importante conocer el tiempo promedio de ejecución en casos de tamaño n. Hemos visto que el tiempo consumido por el algoritmo de ordenación por inserción varia entre el orden de n y el orden de n^2 . Si podemos calcular el tiempo promedio consumido por el algoritmo en las n! formas diferentes en que se pueden inicialmente suponer ordenados los elementos del caso (supuesto que son todos distintos), tendríamos una idea verosímil del tiempo empleado para ordenar un array inicialmente ordenado aleatoriamente (puede verse que es, también, en el orden de n). El algoritmo de ordenación por inserción toma, por tanto, tiempo cuadrático tanto en el promedio como en el peor caso, aunque en algunos casos puede ser muchísimo mas rápido. Mas adelante veremos otro

algoritmo de ordenación que también consume tiempo cuadrático en el peor caso, pero que requiere un tiempo en el orden de $n \log n$ en promedio. Sin embargo, aunque ese algoritmo tiene un mal peor caso, es de los más rápidos algoritmos en promedio.

Usualmente es más difícil analizar la conducta del caso promedio que la del peor caso. Además, en algún sentido puede ser menos informativo que el análisis de la conducta en el peor caso. Por eso, en lo que sigue, solo nos referiremos al análisis del peor de los casos, salvo que se indique lo contrario.