Introducción a la algoritmia

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Maestría en Ciencias de la Computación
Análisis y Diseño de Algoritmos
MCOM 20300

Contenido

¿Qué es un algoritmo?

- Conjunto de reglas que, al seguirlas, realizan algún cálculo, ya sea a mano o por computadora.

¿Qué es un algoritmo?

- Los métodos que aprendimos en la primaria para sumar, restar, multiplicar y dividir son ejemplos de algoritmos.

¿Qué es un algoritmo?

- El algoritmo más famoso en la historia es el algoritmo de Euclides (siglo IV a. C.) para calcular el máximo común divisor de dos enteros.

Ejecución de un algoritmo

- No involucra decisiones subjetivas.

Ejecución de un algoritmo

- No requiere el uso de la intuición.

Ejecución de un algoritmo

- No requiere el uso de la creatividad.

¿Una receta de cocina es un algoritmo?

Lo es siempre y cuando describa precisamente como preparar un platillo, lo cual incluye:

¿Una receta de cocina es un algoritmo?

Lo es siempre y cuando describa precisamente como preparar un platillo, lo cual incluye:

- La cantidad exacta de cada ingrediente.

¿Una receta de cocina es un algoritmo?

Lo es siempre y cuando describa precisamente como preparar un platillo, lo cual incluye:

- La cantidad exacta de cada ingrediente.
- Una manera clara de cómo mezclar los ingredientes.

¿Una receta de cocina es un algoritmo?

Lo es siempre y cuando describa precisamente como preparar un platillo, lo cual incluye:

- La cantidad exacta de cada ingrediente.
- Una manera clara de cómo mezclar los ingredientes.
- La temperatura a la que se debe cocinar.

¿Una receta de cocina es un algoritmo?

Lo es siempre y cuando describa precisamente como preparar un platillo, lo cual incluye:

- La cantidad exacta de cada ingrediente.
- Una manera clara de cómo mezclar los ingredientes.
- La temperatura a la que se debe cocinar.
- El tiempo de cocción.

¿Una receta de cocina es un algoritmo?

Lo es siempre y cuando describa precisamente como preparar un platillo, lo cual incluye:

- La cantidad exacta de cada ingrediente.
- Una manera clara de cómo mezclar los ingredientes.
- La temperatura a la que se debe cocinar.
- El tiempo de cocción.

Si la receta contiene nociones vagas como “agrege sal al gusto” o “cocine hasta que este tierno”, entonces ya no se considera un algoritmo.

Algoritmos probabilísticos

Son algoritmos que hacen alguna elección aleatoria en una situación dada.

Algoritmos probabilísticos

Son algoritmos que hacen alguna elección aleatoria en una situación dada.

- Aleatorio no significa arbitrario.

Algoritmos probabilísticos

Son algoritmos que hacen alguna elección aleatoria en una situación dada.

- Los valores se eligen de tal manera que la probabilidad de elegir cada valor es conocida y controlada.

Algoritmos probabilísticos

Son algoritmos que hacen alguna elección aleatoria en una situación dada.

- "Elija un número entre 1 y 6" sin mayores detalles no es aceptable en un algoritmo.

Algoritmos probabilísticos

Son algoritmos que hacen alguna elección aleatoria en una situación dada.

- "Elija un número entre 1 y 6 de tal manera que cada valor tenga la misma probabilidad de ser elegido" es la manera correcta de decirlo.

Algoritmos aproximados

- Un conjunto de reglas que nos digan que el resultado de 23 por 51 es 1170 no es aceptable en la práctica (el resultado correcto es 1173).

Algoritmos aproximados

- Pero si queremos calcular $\sqrt{2}$ nos tenemos que conformar con un algoritmo que se aproxime, ya que es un número irracional cuya representación es infinita y no repetitiva.

Algoritmos aproximados

- En el caso de los algoritmos aproximados lo que tenemos que controlar es la precisión con la que queremos la solución, por ejemplo: con una precisión de 10^{-4} o de 10^{-10}.

Algoritmos heurísticos

- Existen problemas para los que no se conoce un algoritmo “práctico” que los resuelva.

Algoritmos heurísticos

- Para un problema de ese tipo un algoritmo que lo resuelva exactamente tardaría siglos en terminar.

Algoritmos heurísticos

- En estos casos necesitamos un conjunto de reglas que nos den una buena aproximación a la respuesta correcta y que se puedan ejecutar en un tiempo razonable.

Algoritmos heurísticos

- Estos procedimientos se basan en el optimismo y tienen mínimo sustento teórico.

Algoritmos heurísticos

- No se deben confundir con los algoritmos aproximados, recuerde que con los algortimos aproximados se puede especificar el error que estamos dispuestos a aceptar. Con los algoritmos heurísticos no podemos controlar el error, sólo podemos estimar que tan grande es.

Ahora podemos decir que la Algoritmia es el estudio de los algoritmos.

Cuando hay que resolver un problema es posible que se tengan varios algoritmos para hacerlo. Así que se tiene que hacer una elección de acuerdo a las condiciones materiales del lugar donde se tiene que resolver, se prodría seleccionar:

- El que tome menos tiempo de ejecución.

Algoritmia I

Ahora podemos decir que la Algoritmia es el estudio de los algoritmos.

Cuando hay que resolver un problema es posible que se tengan varios algoritmos para hacerlo. Así que se tiene que hacer una elección de acuerdo a las condiciones materiales del lugar donde se tiene que resolver, se prodría seleccionar:

- El que tome menos tiempo de ejecución.
- El que use menos almacenamiento.

Algoritmia I

Ahora podemos decir que la Algoritmia es el estudio de los algoritmos.

Cuando hay que resolver un problema es posible que se tengan varios algoritmos para hacerlo. Así que se tiene que hacer una elección de acuerdo a las condiciones materiales del lugar donde se tiene que resolver, se prodría seleccionar:

- El que tome menos tiempo de ejecución.
- El que use menos almacenamiento.
- El que sea más fácil de programar.

La selección depende de algunos factores, como pudieran ser:

La selección depende de algunos factores, como pudieran ser:

- Los datos involucrados.

La selección depende de algunos factores, como pudieran ser:

- Los datos involucrados.
- La manera en que se presenta el problema.

Algoritmia II

La selección depende de algunos factores, como pudieran ser:

- Los datos involucrados.
- La manera en que se presenta el problema.
- La rapidez y capacidad de almacenamiento del equipo de cómputo disponible.

Pero si ninguno de los algoritmos disponibles es completamente adecuado para las condiciones materiales que se tienen, entonces tendríamos que diseñar un nuevo algoritmo.

Algoritmia III

Así, la algoritmia es una disciplina de las ciencias de la computación que sirve para:

Algoritmia III

Así, la algoritmia es una disciplina de las ciencias de la computación que sirve para:

- Evaluar el efecto de las condiciones materiales sobre los algoritmos disponibles para que podamos elegir el que mejor se adapte a dichas condiciones.

Algoritmia III

Así, la algoritmia es una disciplina de las ciencias de la computación que sirve para:

- Evaluar el efecto de las condiciones materiales sobre los algoritmos disponibles para que podamos elegir el que mejor se adapte a dichas condiciones.
- Diseñar nuevos algoritmos que resuelvan un problema en particular.

Ejercicios

Ejercicio 1

1. En la bibliografía o en internet busque los siguientes algoritmos de ordenamiento.
    1. Bubble Sort,
    2. Heap Sort,
    3. Radix Sort.
2. Diga bajo qué condiciones materiales escogería cada uno de los algoritmos del ejercicio anterior.

Problemas y ejemplares I

- Existen varios algoritmos para resolver el problema de multiplicar dos enteros positivos, un ejemplar de este problema es multiplicar 981 por 1234.

Problemas y ejemplares I

- Dichos algoritmos no sólo proveen una forma de multiplicar estos dos números en particular.

Problemas y ejemplares I

- Ellos dan una solución general al **problema** de multiplicar dos enteros positivos.

Problemas y ejemplares II

- Decimos que (981, 1234) es un **ejemplar** de este problema.

Problemas y ejemplares II

- Multiplicar 789 por 9742 es otro ejemplar del mismo problema y lo podemos expresar como (789, 9742).

Problemas y ejemplares II

- Multiplicar -12 por 83.7 no es un ejemplar del problema, por dos razones:
  1. -12 no es positivo y
  2. 83.7 no es un entero.

Problemas y ejemplares II

- Por supuesto, (-12, 83.7) es un ejemplar de otro problema de multiplicación más general.

Problemas y ejemplares III

- La mayoría de los problemas interesantes tienen una colección infinita de ejemplares.

Problemas y ejemplares III

- No obstante, hay excepciones. Hablando estrictamente, el problema de jugar un juego perfecto de ajedrez tiene sólo un ejemplar, ya que existe solamente una única posición inicial. Más aún existe sólo un número finito de subejemplares (las posiciones intermedias legales). Sin embargo esto no significa que el problema carezca de interés algorítmico.

Dominio de definición I

- Un algoritmo debe trabajar correctamente sobre cada ejemplar del problema que resuelve.

Dominio de definición I

- Para mostrar que un algoritmo es incorrecto, sólo necesitamos encontrar un ejemplar del problema para el cual es incapaz de encontrar una respuesta correcta.

Dominio de definición I

- Tal como un teorema propuesto se puede desaprobar con un simple contraejemplo, un algoritmo se puede rechazar sobre la base de un solo resultado incorrecto.

Dominio de definición I

- Por otro lado, así como puede ser difícil probar un teorema es usualmente difícil probar la corrección de un algoritmo.

Dominio de definición II

- Para hacer esto posible, cuando se especifica un problema es importante definir su **dominio de definición**, esto es, el conjunto de ejemplares a ser considerados.

Dominio de definición II

- Así, los algoritmos para multiplicar enteros positivos no funcionarán con operandos negativos o fraccionales, al menos no sin alguna modificación.

Dominio de definición II

- Por supuesto, esto no significa que los algoritmos sean inválidos: ya que los ejemplares que involucran números negativos o fracciones no están en el dominio de definición que elegimos al enunciar el problema.

Dominio de definición III

- Cualquier dispositivo real de cálculo tiene un límite en el tamaño de los ejemplares que puede manejar, ya sea porque los números involucrados son muy grandes o porque desbordamos el espacio de almacenamiento.

Dominio de definición III

- No obstante, este límite no se le puede atribuir al algoritmo que elegimos usar.

Dominio de definición III

- Máquinas distintas tienen límites distintos y aún programas distintos que implementan el mismo algoritmo sobre la misma máquina pudieran imponer distintas restricciones.

Dominio de definición III

- En este curso nos conformaremos con probar que nuestros algoritmos son correctos en abstracto, ignorando las limitaciones prácticas presentes en cualquier programa concreto que los implemente.

¿Cuál algoritmo elegir?

- Cuando tenemos que resolver un problema, podríamos tener disponibles varios algoritmos y obviamente nos gustaría elegir el mejor, esto plantea la pregunta de cómo decidir cuál de esos algoritmos es preferible.

¿Cuál algoritmo elegir?

- Si sólo tenemos uno o dos ejemplares por resolver, de un problema muy simple, podríamos no preocuparnos demasiado por cuál algoritmo usar: en este caso simplemente podríamos elegir el que sea más fácil de programar, o aquel para el que ya existe un programa, sin preocuparnos sobre sus propiedades teóricas.

¿Cuál algoritmo elegir?

- Pero si tenemos gran cantidad de ejemplares por resolver o si el problema es muy difícil de resolver, tendriamos que elegir más cuidadosamente.

Enfoques para elegir un algoritmo

Existen dos enfoques para elegir un algoritmo:

Enfoques para elegir un algoritmo

Existen dos enfoques para elegir un algoritmo:

- El **enfoque empírico** (o **a posteriori**) consiste en programar los algoritmos en competencia y probarlos con ejemplares distintos y la ayuda de una computadora.

Enfoques para elegir un algoritmo

Existen dos enfoques para elegir un algoritmo:

- El **enfoque empírico** (o **a posteriori**) consiste en programar los algoritmos en competencia y probarlos con ejemplares distintos y la ayuda de una computadora.
- El **enfoque teórico** (o **a priori**), el cual favoreceremos en este curso, consiste en determinar matemáticamente la cantidad de recursos que necesita cada algoritmo **como una funcion del tamaño de los ejemplares considerados**.

Recursos a considerar

- Los recursos de mayor interés son el tiempo de cálculo y el espacio para almacenamiento, usualmente el primero es el más crítico.

Recursos a considerar

- En este curso usualmente compararemos a los algoritmos sobre la base de sus tiempos de ejecución.

Recursos a considerar

- Cuando hablemos de la **eficiencia** de un algoritmo simplemente estaremos diciendo que tan rápido se ejecuta.

Recursos a considerar

- Sólo ocasionalmente estaremos interesados en los requerimientos de almacenamiento de un algoritmo.

Tamaño de un ejemplar

- El **tamaño** de un ejemplar formalmente corresponde al número de bits necesarios para representar al ejemplar en una computadora, usando algún esquema de codificación definido precisamente y razonablemente compacto.

Tamaño de un ejemplar

- No obstante, para ser más claro nuestro análisis, usualmente seremos menos formales y cuando usemos la palabra tamaño entenderemos **cualquier entero** que de alguna manera mide el número de componentes en un ejemplar.

Ejemplos tamaño de un ejemplar

- Cuando hablemos sobre ordenamiento, usualmente mediremos el tamaño de un ejemplar por el número de artículos que se han de ordenar, ignorando el hecho de que cada uno de estos artículos necesitaría más de un bit para representarse sobre una computadora.

Ejemplos tamaño de un ejemplar

- Cuando hablemos sobre grafos, usualmente mediremos el tamaño de un ejemplar por el número de nodos o vértices (o ambos) involucrados.

Ejemplos tamaño de un ejemplar

- Cuando hablemos de problemas que involucren enteros nos apartaremos de esta regla general y algunas veces daremos la eficiencia de nuestros algoritmos en términos del **valor** del ejemplar considerado y no de su tamaño (el cual sería el número de bits necesarios para representar dicho valor en binario).

Medición de recursos I

- Si queremos medir la cantidad de almacenamiento que usa un algoritmo en función del tamaño de los ejemplares, tenemos disponible una unidad natural para nosotros y se llama bit.

Medición de recursos I

- Sin importar la máquina que se use la noción de bit está bien definida.

Medición de recursos II

- Si queremos medir la eficiencia de un algoritmo en términos del tiempo que toma para producir una respuesta, entonces no hay una elección tan obvia.

Medición de recursos II

- Claramente no podríamos expresar esta eficiencia por ejemplo en segundos, ya que no tenemos una computadora estándar a la cual pudieran hacer referencia todas las mediciones.

Principio de invarianza I

- Una respuesta a este problema esta dada por el **principio de invarianza**, el cual enuncia que dos implementaciones diferentes del mismo algoritmo no diferirán en eficiencia por más de alguna constante multiplicativa.

Principio de invarianza I

- Si esta constante resulta ser 5, por ejemplo, entonces sabremos que si la primera implementación toma un segundo para resolver ejemplares de cierto tamaño, entonces la segunda implementación (pudiera estar sobre una máquina distinta o escrita en un lenguaje de programación distinto) no tomará más de 5 segundos para resolver los mismos ejemplares.

Principio de invarianza I

- Más precisamente, si dos implementaciones del mismo algoritmo toman respectivamente $t_1(n)$ y $t_2(n)$ segundos para resolver un ejemplar de tamaño n, entonces siempre existen constantes positivas c y d tales que $t_1(n) \leq ct_2(n)$ y $t_2(n) \leq dt_1(n)$ siempre que n sea suficientemente grande.

Principio de invarianza II

- En otras palabras, el tiempo de ejecución de cualquiera de las dos implementaciones está acotado por un múltiplo constante del tiempo de ejecución de la otra; por lo tanto es irrelevante a cuál implementación le llamaremos primera y a cuál segunda.

Principio de invarianza II

- La condición de que n sea suficientemente grande no es realmente necesaria, como veremos posteriormente al analizar la regla del umbral.

Principio de invarianza III

- Este principio no es algo que podamos probar: simplemente enuncia un hecho que puede ser confirmado mediante observación. Más aún tiene una amplia aplicación.

Principio de invarianza III

- El principio se mantiene cierto sin importar cual computadora se usa para implementar un algoritmo (siempre y cuando la computadora tenga un diseño convencional), sin importar el lenguaje de programación y el compilador empleado, es más, sin importar la habilidad del programador (siempre y cuando él o ella no modifique realmente el algoritmo).

Principio de invarianza III

- Así un cambio de máquina podría permitirnos resolver un problema 10 o 100 veces más rápido lo cual nos incrementa la rapidez en un factor constante. Por otro lado, un cambio de algoritmo y sólo un cambio de algoritmo podría darnos una mejora que sería más y más marcada conforme aumenta el tamaño de los ejemplares.

Unidad de tiempo I

- Regresando a la pregunta de qué unidad será usada para expresar la eficiencia teórica de un algoritmo, el pricipio de invarianza nos permite decidir que no es necesaria tal unidad. En su lugar, sólo expresamos el tiempo que toma un algoritmo salvo una constante multiplicativa.

Unidad de tiempo I

- Decimos que un algoritmo para algún problema toma un tiempo **en el orden de** $t(n)$, para una función dada t, si existe una constante positiva c y una implementación del algoritmo capaz de resolver todo ejemplar de tamaño n en no más que $ct(n)$ segundos.

Unidad de tiempo I

- Para problemas numéricos, como señalamos anteriormente, n algunas veces puede ser el valor más que el tamaño del ejemplar.

Unidad de tiempo II

- El uso de segundos en esta definición es obviamente arbitrario: sólo necesitamos cambiar la constante para acotar el tiempo por $at(n)$ años o $bt(n)$ microsegundos.

Unidad de tiempo II

- Por el principio de invarianza, si cualquiera de las implementaciones del algoritmo tiene la propiedad requerida, entonces también la tendrán todos los demás, no obstante la constante multiplicativa cambiaría de una implementación a la otra.

Unidad de tiempo II

- En la siguiente sección daremos un tratamiento más riguroso de este concepto importante conocido como la **notación asintótica**. A partir de la definición formal será claro porque decimos **en el orden de** en lugar del más usual **del orden de**.

Nomenclatura de algunos órdenes

- Ciertos ordenes ocurren tan frecuentemente que vale la pena darles un nombre. Por ejemplo, suponga que el tiempo que toma un algoritmo para resolver un ejemplar de tamaño n nunca es mayor a cn segundos donde c es una constante apropiada. Entonces decimos que el algoritmo toma un tiempo en el orden de n, o más simplemente que toma tiempo **lineal**. En este caso también hablamos de un **algoritmo lineal**.

Nomenclatura de algunos órdenes

- Si un algoritmo nunca toma más de cn^2 segundos para resolver un ejemplar de tamaño n, entonces decimos que toma tiempo en el orden de n^2, o tiempo **cuadrático**, y le llamaremos un **algoritmo cuadrático**.

Nomenclatura de algunos órdenes

- Similarmente un algoritmo es **cúbico, polinomial o exponencial** si toma un tiempo en el orden de n^3, n^k o c^n, respectivamente, donde k y c son constantes adecuadas.

Advertencia I

- No caiga en la trampa de olvidar completamente a las **constantes ocultas**, como son llamadas las constantes multiplicativas usadas en estas definiciones. Comúnmente ignoraremos los valores exactos de estas constantes y **asumiremos que todas ellas tienen el mismo orden de magnitud**.

Advertencia I

- Esto nos permitirá decir, por ejemplo, que un algoritmo lineal es más rápido que uno cuadrático sin preocuparnos si nuestra afirmación es verdadera en todos los casos. Sin embargo, algunas veces es necesario ser más cuidadoso.

- Considere por ejemplo dos algoritmos cuyas implementaciones sobre una máquina dada toman respectivamente n^2 días y n^3 segundos para resolver un ejemplar de tamaño n.

- Solamente para **¡ejemplares que requieren más de 20 millones de años en ser resueltos!** es que el algoritmo cuadrático llega a ser más rápido que el algoritmo cúbico.

- Desde un punto de vista teórico, el primero es **asintóticamente** mejor que el segundo; es decir, su desempeño es mejor sobre todos los ejemplares suficientemente grandes.

- Desde un punto de vista práctico ciertamente preferiremos al algoritmo cúbico. No obstante el algoritmo cuadrático puede ser asintóticamente mejor, su constante oculta es tan grande que lo descarta para considerarlo sobre ejemplares de tamaño normal.

Ejercicios

Ejercicio 2

1. Justifique matemáticamente por qué el algoritmo cuadrático del ejemplo anterior es mejor que el algoritmo cúbico sólo para ejemplares que requieren más de 20 millones de años en ser resueltos.