Introducción

De manera intuitiva, un algoritmo es una serie de pasos que toma una entrada y da una salida. Comenzaremos platicando qué tipo de problemas nos interesa resolver y luego qué tipo de soluciones nos interesa dar.

Problemas algorítmicos

Cuando estamos hablando de resolver problemas algorítmicos, usualmente no nos van a importar problemas muy particulares, en donde sólo damos una solución «para un caso». Típicamente, lo que queremos es resolver problemas generales, en donde las posibilidades a las que nos enfrentaremos son varias, complejas e inesperadas.

Ejemplo. El siguiente no es un problema general: «Ordena los números $5,7,3,10,12,1$».

Si bien es un problema que se puede resolver con la computadora, consiste únicamente de un caso particular, con unos pocos números. $\square$
A un caso pequeño como estos se le conoce como una **instancia**. De esta forma, cuando estamos planteando un problema de los sí nos van a interesar, una cosa que tenemos que hacer es definir todas las instancias que nuestro algoritmo debe resolver.

Definición. Un **problema algorítmico** consiste de:

- Un **problema** a resolver.
- Una **entrada**, que nos dice cuáles o cómo son las instancias que esperamos resolver.
- Una **salida** que nos dice qué es lo que esperamos obtener y con qué características.

Ejemplo. Un problema algorítmico puede ser el siguiente.

Problema: Ordenar una lista finita de números enteros.

Entrada: Una lista finita de números enteros: $a_1,a_2,\ldots,a_n$.

Salida: Una lista con esos mismos números, pero ordenada, es decir, una lista $b_1,b_2,\ldots,b_n$ tal que $b_i\leq b_j$ si $i\leq j$ y tal que ambas listas tengan los mismos elementos. $\square$

La descripción de las entradas que podemos recibir y las salidas que esperamos dar usualmente están en términos matemáticos, o en términos de estructuras de datos, de las cuales hablaremos después.

Algoritmos

Un **algoritmo** es una serie de pasos que resuelve un problema algorítmico. Esta serie de pasos debe resolver todas las posibles instancias, sin importar cual sea. Mediante un cierto proceso, debemos explicar poco a poco cómo podemos tomar la entrada, procesarla y dar una salida que resuelva el problema con las condiciones establecidas.

Un algoritmo puede ser descrito de varias formas:

- Con palabras
- Con pseudocódigo
- Con código de algún lenguaje de programación (Python, C++, PERL, PHP, Javascript, etc)

Hacia arriba es más entendible para humanos y más flexible, pero más difícil de que lo ejecute una computadora. Hacia abajo es

menos entendible y requiere más formalidad. Hacia abajo tiene la ventaja de que una computadora lo puede ejecutar.

Comúnmente se dice que hacia arriba en la lista es una descripción de **nivel más alto** y hacia abajo es una descripción de **nivel más bajo**. Aquí la palabra **nivel** no quiere decir «dificultad» sino más bien qué tantos «factores» están involucrados. Hasta arriba, el lenguaje humano es muy complejo: tiene sutilezas, depende mucho del contexto, es más expresivo. Hasta abajo, el lenguaje es muy objetivo pues se basa en pocas reglas que se deben cumplir.

Ejemplo. Supongamos que queremos resolver el siguiente problema algorítmico:

Problema. Contar números pares.

Entrada. Un entero positivo \(n\) y una lista de enteros \(a_1,a_2,\ldots,a_n\).

Salida. La cantidad de enteros en la lista que sean números pares.

Para resolver este problema, podemos dar un algoritmo en palabras como sigue:

Llevaremos una cuenta de cuántos números pares van. Lo que haremos es recorrer la lista de izquierda a derecha. Intentamos dividir el número entre 2. Si el residuo que queda es 0, entonces aumentamos la cuenta de pares. Seguimos así hasta terminar. Al final nuestra cuenta dará la cantidad de pares que hay.

También podemos explicar el algoritmo en pseudocódigo:

```
funcion contar_pares(lista):
    un contador empieza en 0
    para cada número en la lista:
        si el residuo al dividir el número entre 2 es cero:
            aumentamos el contador en 1
    reportamos el contador
```

Observa que el pseudocódigo es muy casual. No es necesario mantener una sintaxis estricta y podemos todavía explicar con algunas palabras.

Finalmente, podríamos dar el código exacto en algún lenguaje de programación para resolver el problema. A continuación se muestra un código en Python para ello:

```
def contar_pares(L):
    cont=0
    for k in L:
        if k%2==0:
            cont+=1
    print(cont)
```

Ponemos esto en una celda de código a continuación y vemos que esté funcionando

```
def contar_pares(L):
    cont=0
    for k in L:
        if k%2==0:
            cont+=1
    print(cont)

contar_pares([4,3,67,1,6,23,16])
contar_pares([5,1,7,12,65,12])
contar_pares([-21,0,5,-2,4,9,3,8,10])
```

```
3
2
5
```

Parece ser que nuestro algoritmo funciona en las instancias que dimos. ¿Será que siempre funciona? \(\square\)

Propiedades de algoritmos

Propiedades de los algoritmos

Lo que más nos interesa de los algoritmos son las siguientes tres cosas:

- Que sean **correctos**, es decir, que resuelvan *todas* las instancias que nos interesan del problema correctamente.
- Que sean **eficientes**, es decir, que no tomen ni mucho tiempo ni mucho espacio para ser ejecutados en términos del tamaño de la entrada.
- Que sean **fáciles de implementar**, decir, que se puedan escribir en código de manera sencilla.

La primera es una propiedad fundamental. El algoritmo puede resolver bien algunas instancias, pero si no resuelve bien todas ellas, entonces podemos estar en problemas al momento de aplicarlo.

La segunda y tercera propiedades son más bien cosas deseables, aunque a veces hay que comprometer alguna de ellas para lograr la otra. Por ejemplo, en ocasiones un algoritmo muy bueno en tiempo, será malo en cantidad de memoria usada. O bien un algoritmo muy fácil de implementar puede que ocupe mucho espacio. Aunque a veces todo puede salir bien, tenemos que acostumbrarnos a que usualmente hay más de un algoritmo que resuelva de manera correcta un problema. Y en ocasiones tenemos que hacer un intercambio entre su eficiencia en tiempo, en espacio y en su facilidad de implementación.

Dado que puede haber más de un algoritmo para resolver un mismo problema, es imporante que tengamos algunas herramientas para poder comparar algoritmos entre sí. En las siguientes entradas hablaremos de cómo evaluar un algoritmo en términos de correctitud y eficiencia.

Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. El problema de «hacer chiles en nogada», ¿es un problema algoritmico de los que nos interesa resolver? ¿Por qué?
2. Dadas n ciudades, saber cómo viajar entre ellas para recorrer todas en la menor cantidad posible de tiempo, ¿es un problema algorítmico de los que nos interesa resolver? ¿Por qué?
3. A continuación se muestra un algoritmo en Python que encuentra el máximo de una lista no vacía L de enteros positivos. Estudia qué está sucediendo y pasa el algoritmo tanto a una descripción en palabras, como a una descripción en pseudocódigo.

```
def dar_max(L):
    M=-1
    for j in L:
        if j>M:
            M=j
    print(M)
```

4. La siguiente descripción en palabras explica cómo sumar todos los elementos de una lista no vacía de enteros. Conviértela a código en Python. «Dada una lista no vacía de enteros, podemos sumar todos ellos como sigue. Tomamos el primer elemento. Luego, le sumamos el segundo elemento. Luego, a esa suma le sumamos el tercer elemento. Luego el cuarto elemento. Y así sucesivamente, hasta que terminemos de sumar todos los elementos. El resultado final es la suma de todos los números de la lista».
5. El siguiente es un problema algorítmico de los que queremos resolver:

   Problema: Pareja de distancia mínima.

   Entrada: Puntos $p_1,p_2,p_3,\ldots,p_n$ en el plano.

   Salida: Dos valores distintos i y j en $\{1,2,\ldots,n\}$ tales que p_i y p_j sean los dos puntos más cercanos de todos los puntos dados.

   ¿Cómo mejorarías todavía más esta descripción del problema? ¿Es importante ser más preciso en algún sentido, por ejemplo con respecto a la métrica que se usará? Si no se menciona la métrica, ¿qué supondrías que te está pidiendo? ¿Se te ocurre un

algoritmo que ayude a resolver este problema? Da descripciones en palabras, pseudocódigo y código para este algoritmo.

Anterior
Diseño y análisis de algoritmos

Siguiente
Algoritmos incorrectos

Por Leonardo Ignacio Martínez Sandoval