RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

Índice de contenido

1.DATOS, ALGORITMOS Y PROGRAMAS....2
1.1.ALGORITMOS....2
1.2.PROGRAMAS....4
1.3.DATOS....4
2.RESOLUCIÓN DE PROBLEMAS....5
1.Análisis del Problema....5
2.Diseño de Algoritmos....5
3.Verificación de Algoritmos....8
3.Herramientas para la representación de Algoritmos....9
3.1.Diagramas de flujo....10
3.2.Pseudocódigo....12

1. RESOLUCIÓN DE PROBLEMAS Y ALGORITMOS

La principal razón por la que las personas aprenden a programar es para utilizar el ordenador como una herramienta para la resolución de problemas. Ayudado por un ordenador, la obtención de la solución a un problema se puede dividir en dos fases:

1. Fase de resolución del problema
2. Fase de implementación en el ordenador.

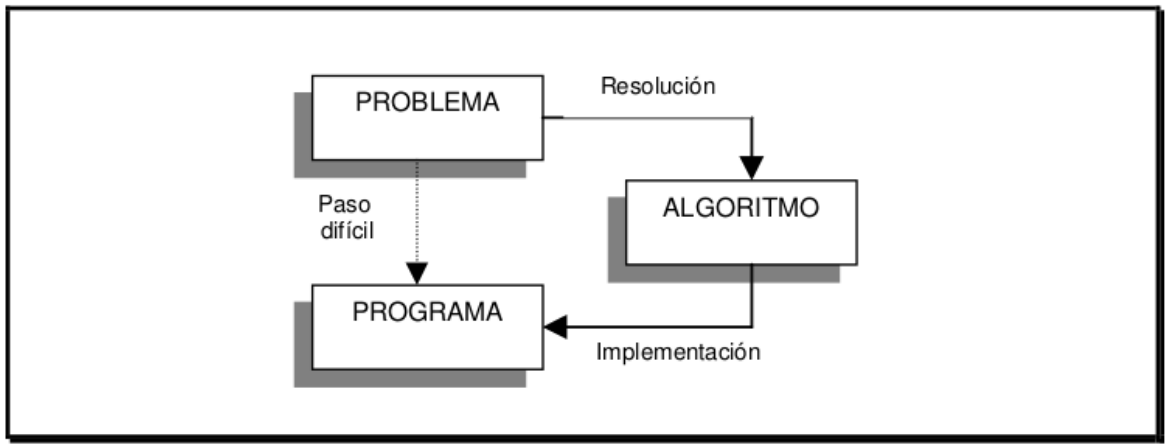


Figura 1. Fases de resolución e implementación

El resultado de la primera fase es el diseño de un algoritmo, que no es más que una secuencia ordenada de pasos que conduce a la solución de un problema concreto, sin ambigüedad alguna, en un tiempo finito. Sólo cuando dicho algoritmo haya sido probado y validado, se deberá entrar en detalles de implementación en un determinado lenguaje de programación; al algoritmo así expresado se denomina programa .

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como del ordenador que los ejecuta. El lenguaje de programación es tan sólo un medio para comunicarle al ordenador la secuencia de acciones a realizar y el ordenador sólo actúa como mecanismo para obtener la solución.

1.1. ALGORITMOS

Un algoritmo es un método para resolver un problema. Más exactamente es el conjunto de pasos o instrucciones que se deben seguir para resolver un tipo específico de problema.

La palabra algoritmo se deriva de la traducción al latín de la palabra *Alkhô-warîzmi*, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX .

Debe cumplir las siguientes condiciones:

- Precisión: el algoritmo debe indicar el orden de realización de cada acción, de forma

clara y sin ambigüedades. Además, el algoritmo debe ser concreto en el sentido de contener sólo el número de pasos precisos para llegar a la solución (no deben darse pasos de más).

- Repetitividad: el algoritmo debe poder repetirse tantas veces como se quiera. Si se sigue el algoritmo dos veces con los mismos datos de entrada, se obtendrán los mismos datos de salida independientemente del momento de ejecución.
- Finitud : el algoritmo debe terminar en algún momento. Si el algoritmo nunca acaba no obtendremos ninguna solución y, como se ha señalado, el objetivo principal de un algoritmo es obtener la solución de un problema.

Ejemplo de algoritmo para subir una escalera:

```
Levantar un pie hasta la altura del primer escalón
Adelantarlo hasta apoyarlo en dicho escalón
Levantar el otro pie hasta la altura del escalón siguiente.
Repetir los dos pasos anteriores mientras queden escaleras
```

Además, a la hora de estudiar la calidad de un algoritmo, es deseable que los algoritmos presenten también otra serie de características como son:

- Validez: el algoritmo construido hace exactamente lo que se pretende hacer.
- Eficiencia: el algoritmo debe dar una solución en un tiempo razonable. Por ejemplo, para sumar 20 a un número dado podemos dar un algoritmo que sume uno veinte veces, pero esto no es muy eficiente. Sería mejor dar un algoritmo que lo haga de un modo más directo.
- Optimización: se trata de dar respuesta a la cuestión de si el algoritmo diseñado para resolver el problema es el mejor. En este sentido y como norma general, será conveniente tener en cuenta que suele ser mejor un algoritmo sencillo que no uno complejo, siempre que el primero no sea extremadamente ineficiente.

En el algoritmo se plasman las tres partes fundamentales de una solución informática:

- Entrada: información dada al algoritmo.
- Proceso: cálculos necesarios para encontrar la solución del problema.
- Salida: resultados finales de los cálculos.

El algoritmo describe una transformación de los datos de entrada para obtener los datos de salida a través de un procesamiento de la información.

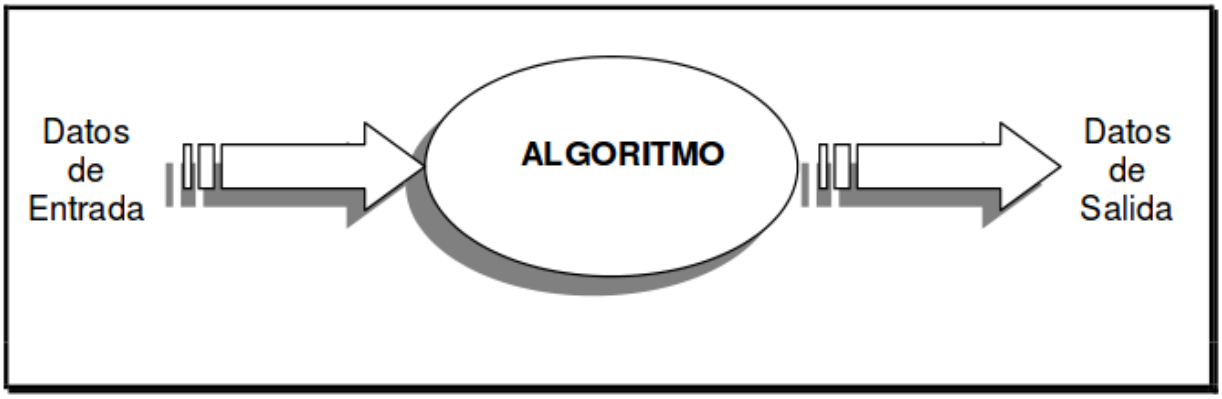


Figura 2. Entrada, algoritmo y salida

1.2. PROGRAMAS

Llegados a este punto, podemos preguntarnos, ¿cuál es la diferencia entre programa y algoritmo?. Para que una computadora resuelva un problema según un algoritmo éste tiene que ser convertido a un programa traduciéndolo a algún lenguaje de programación.

Los algoritmos no son directamente interpretables por la computadora, pero son independientes de cualquier lenguaje de programación.

1.3. DATOS

Para poder operar, un programa necesita datos.

Un dato es un símbolo físico que se utiliza para representar la información. Por datos se entiende un valor numérico, una serie de caracteres, un valor lógico... que pueda ser objeto de tratamiento informático.

Los programas actúan sobre los datos de entrada y los transforman para convertirlos en información de salida.

Los datos pueden ser simples o aparecer en forma de colección formando lo que se denomina una estructura sobre las que se definen operaciones. Este último sería el caso de arrays, registros, matrices, listas...

2. FASES EN LA RESOLUCIÓN DE PROBLEMAS

Los pasos a seguir en la fase de resolución del problema son tres:

1. Análisis del problema.

2. Diseño del algoritmo.
3. Verificación del algoritmo.

1. Análisis del Problema

Es el primer paso para encontrar una solución computacional a un problema dado y requiere el máximo de creatividad por parte del programador.

El primer objetivo que nos debemos plantear es obtener una correcta comprensión de la naturaleza del problema. El análisis del problema exige una primera lectura del problema a fin de obtener una idea general de lo que se solicita. Una segunda lectura deberá servir para responder a las preguntas:

1) ¿Qué información debe proporcionar la resolución del problema?

2) ¿Qué datos se necesitan para resolver el problema?

La respuesta a la primera pregunta indicará los resultados deseados o salida del programa .

La respuesta a la segunda pregunta indicará qué datos se deben proporcionar o las entradas del problema .

- *Ejemplo: Las básculas calculan a partir del peso de un producto y de su precio por kilogramo, el correspondiente precio final y a partir de la cantidad de dinero entregada, la cantidad de dinero que debe ser devuelta. Analizar el problema.*

  La información de entrada para el problema es el peso y precio por kilogramo del producto junto con la cantidad de dinero entregado para pagar, a partir de lo cual se calculan las salidas del programa: el precio final del producto y la cantidad devuelta .

2. Diseño de Algoritmos

Un ordenador no tiene la capacidad de pensar y resolver el problema por sí mismo; una vez que el problema ha quedado bien definido debemos plantearnos buscar una secuencia de pasos que lo resuelvan e indiquen al ordenador las instrucciones a ejecutar, es decir, hemos de encontrar un buen algoritmo .

Aunque en la solución de problemas sencillos parezca evidente la codificación en un lenguaje de programación concreto, es aconsejable el uso de algoritmos, a partir de los cuales se pasa al programa simplemente conociendo las reglas de sintaxis del lenguaje de programación a utilizar

Ejemplo: Los pasos a seguir para calcular los datos de salida del ejemplo anterior a partir de los datos de entrada son:

precio ← peso x precio por kilogramo

vuelta ← dinero entregado - precio

Sin embargo, la soluciones a problemas más complejos pueden requerir muchos más pasos. Las estrategias seguidas usualmente a la hora de encontrar algoritmos para problemas complejos son:

- Partición o divide y vencerás: consiste en dividir un problema grande en unidades más

pequeñas que puedan ser resueltas individualmente.

Ejemplo: Podemos dividir el problema de limpiar una casa en labores más simples correspondientes a limpiar cada habitación.

- Resolución por analogía: dado un problema, se trata de recordar algún problema similar que ya esté resuelto. Los dos problemas análogos pueden incluso pertenecer a áreas de conocimiento totalmente distintas.

  Ejemplo: El cálculo de la media de las temperaturas de las provincias andaluzas y las media de las notas de los alumnos de una clase se realiza del mismo modo.

Evidentemente la conjunción de ambas técnicas hace más efectiva la labor de programar: dividir un problema grande en trozos más pequeños ya resueltos.

Ejemplo: Consideremos el problema de calcular la longitud y la superficie de círculo dado su radio. Este problema se puede dividir en cuatro subproblemas:

1) Lectura, desde el teclado, de los datos necesarios.
2) Cálculo de la longitud.
3) Cálculo de la superficie.
4) Mostrar los resultados por pantalla.

El problema ha quedado reducido a cuatro subproblemas más simples. La solución a cada uno de estos subproblemas es un refinamiento del problema original.

El proceso para obtener la solución final consiste en descomponer en subproblemas más simples y a continuación dividir éstos nuevamente en otros subproblemas de menor complejidad, y así sucesivamente; la descomposición en subproblemas deberá continuar hasta que éstos se puedan resolver de forma directa. Este método de resolución de problemas no triviales da lugar a lo que se conoce como diseño descendente o diseño por refinamientos sucesivos:

Se comienza el proceso de solución con un enunciado muy general o abstracto de la solución del problema, de donde se identifican las tareas más importantes a ser realizadas, y el orden en el que se ejecutarán. A continuación se procede repetidamente refinando por niveles, de manera que con cada descomposición sucesiva se obtiene una descripción más detallada incluyendo nuevas acciones a realizar. El proceso finaliza cuando el algoritmo esté lo suficientemente detallado y completo para ser traducido a un lenguaje de programación.

Esta técnica es parte de las recomendaciones de una metodología general de desarrollo de programas denominada programación estructurada .

Ejemplo: Vamos a diseñar un algoritmo para cambiar una bombilla fundida. Una primera solución vendría dada en dos pasos:

1) Quitar la bombilla fundida.
2) Colocar la nueva bombilla.

Esto parece resolver el problema, pero supóngase que se está tratando de entrenar un nuevo robot doméstico para que efectúe esta tarea. En tal caso, los pasos no son lo bastante simples

y hay detalles que no se han especificado. Cada uno de estos pasos iniciales se podría refinar aún más:

1. Quitar la bombilla antigua:

> 1.1 Situar la escalera debajo de la bombilla fundida.
>
> 1.2 Subir por la escalera hasta alcanzar la bombilla.
>
> 1.3 Girar la bombilla en sentido antihorario hasta soltarla.

2. Colocar la nueva bombilla:

> 2.1 Elegir una nueva bombilla de la misma potencia que la fundida.
>
> 2.2 Enroscar la bombilla nueva en sentido horario hasta que quede apretada.
>
> 2.3 Bajar de la escalera.
>
> 2.4 Comprobar que la bombilla funciona.

El algoritmo para reemplazar la bombilla quemada consta de siete pasos:

> 1) Situar la escalera debajo de la bombilla fundida.
>
> 2) Subir por la escalera hasta alcanzar la bombilla.
>
> 3) Girar la bombilla en sentido antihorario hasta soltarla.
>
> 4) Elegir una nueva bombilla de la misma potencia que la fundida.
>
> 5) Enroscar la bombilla nueva en sentido horario hasta que quede apretada.
>
> 6) Bajar de la escalera.
>
> 7) Comprobar que la bombilla funciona.

Pero aún no se han diseñado los pasos con la suficiente precisión. Si suponemos que disponemos de una caja de bombillas, el proceso de elegir la nueva bombilla con la misma potencia será el siguiente:

2.1 Repetir hasta que la bombilla sea válida

> 2.1.1. Elegir una bombilla
>
> 2.1.2. Si la potencia de la nueva bombilla es igual a la de la vieja bombilla
>
> > 2.1.2.1. Entonces la bombilla es válida
> >
> > 2.1.2.2. Si no la bombilla no es válida

Aparecen dos conceptos importantes a la hora de describir algoritmos: el concepto de decisión , y el concepto de repetición , que veremos más adelante.

3. Verificación de Algoritmos

Una vez que se ha descrito el algoritmo utilizando una herramienta adecuada, es necesario comprobar que realiza las tareas para las que fue diseñado y produce los resultados correctos y esperados a partir de la información de entrada.

Este proceso se conoce como prueba del algoritmo y consiste básicamente en recorrer todos los caminos posibles del algoritmo comprobando en cada caso que se obtienen los resultados esperados. Para lo cual realizaremos una ejecución manual del algoritmo con datos significativos que abarquen todo el posible rango de valores y comprobaremos que la salida coincide con la esperada en cada caso.

La aparición de errores puede conducir a tener que rediseñar determinadas partes del algoritmo que no funcionaban bien y a aplicar de nuevo el proceso de localización de errores, definiendo nuevos casos de prueba y recorriendo de nuevo el algoritmo con dichos datos.

Ejemplo: Diseñar un algoritmo que calcule el valor absoluto de un número.

Hay un único dato de entrada al algoritmo que es el número para el cual queremos calcular el valor absoluto. El dato de salida es el valor absoluto calculado para el número original.

Una primera aproximación al algoritmo podría ser:

```
1) Leer número del teclado
2) Valor absoluto ← – número
3) Escribir en pantalla valor absoluto
```

Si probamos el algoritmo con dato de entrada –3 obtenemos que el valor absoluto de éste es 3, lo cual es correcto. Pero el algoritmo diseñado no es correcto, ya que no funciona para datos de entrada positivos. En efecto, según el algoritmo anterior el valor absoluto de 3 es –3, lo cual no es cierto.

Dicha observación da lugar al siguiente algoritmo modificado, que funciona tanto para números positivos como negativos:

```
1) Leer número del teclado
2) Si número < 0 entonces
        Valor absoluto ← – número
        en otro caso
        Valor absoluto ← número
3) Escribir en pantalla valor absoluto
```

3. Herramientas para la representación de Algoritmos

Durante el proceso de diseño del algoritmo es preciso disponer de alguna herramienta para describirlo, se necesita disponer de un lenguaje algorítmico con el que reflejar las sucesivas

acciones que resuelven el problema y que, además, soporte lo mejor posible el proceso sucesivo de refinamiento en subproblemas. Una primera aproximación consistiría en utilizar para describir el algoritmo el lenguaje natural (en nuestro caso el español), pero debido a los innumerables problemas que plantea como la imprecisión o la ambigüedad, se ha optado por utilizar otras herramientas algorítmicas que describan con mayor exactitud la secuencia de acciones y el orden en el que han de ejecutarse.

La característica común de cualquier lenguaje algorítmico es que debe ser independiente del lenguaje de programación a utilizar en la codificación. Un factor importante es que permita una traducción clara del algoritmo al programa. La decisión final del lenguaje depende de otras consideraciones y cualquier algoritmo debe de poder implementarse en cualquier lenguaje de programación.

La estructura de un algoritmo se puede representar con un diagrama estructurado en forma de bloques, donde se muestren los sucesivos refinamientos a partir del problema inicial.

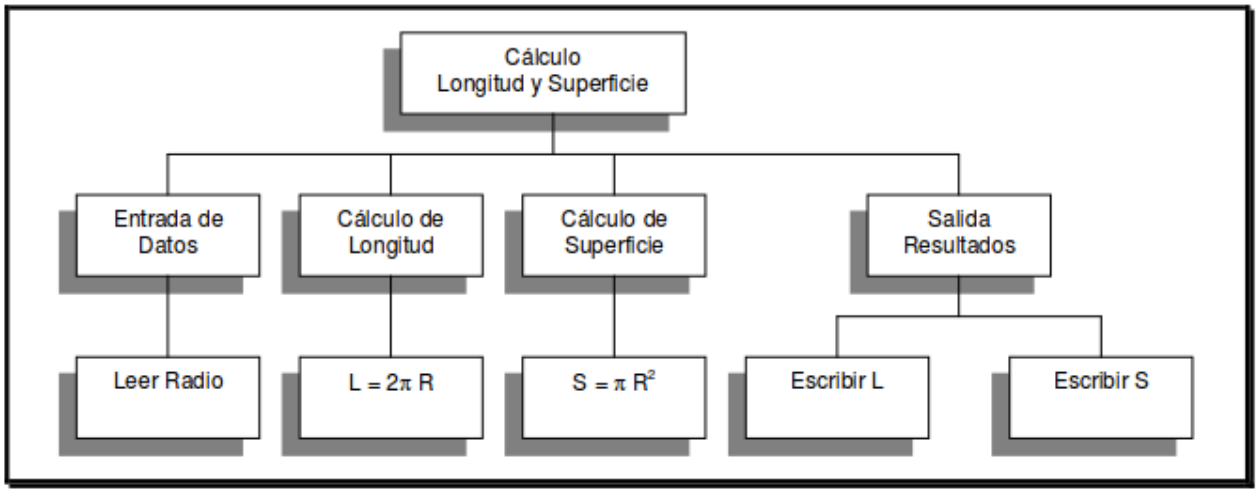


Figura 3. Diagrama de bloques para el cálculo de la superficie y longitud del círculo

Las herramientas de programación utilizadas como lenguajes algorítmicos que estudiaremos a continuación son:

1) Diagramas de flujo.

2) Pseudocódigo.

Los diagramas de flujo han sido la herramienta de programación clásica por excelencia y la más usada. Tienen la ventaja de que muestran el flujo lógico del algoritmo de una manera clara, pero tienen varias limitaciones que han hecho usarlos cada vez menos. Las limitaciones son que su complejidad aumenta con la complejidad del problema, son difíciles de actualizar y oscurecen la estructura del algoritmo.

Como alternativa a los diagramas de flujo y cada vez más en uso está el pseudocódigo , que permite una aproximación del algoritmo al lenguaje natural y por tanto una redacción rápida del mismo, con el inconveniente de la pérdida consecuente de precisión.

Con cualquiera de estas herramientas, tiene que quedar bien reflejado el flujo de control del algoritmo, que es el orden temporal en el cual se ejecutan los pasos individuales del algoritmo. El flujo normal de un algoritmo es el flujo lineal o secuencial de los pasos (un paso a continuación de otro). Para apartarse del flujo normal lineal están las estructuras de control, que son construcciones algorítmicas que afectan directamente al flujo de control de un algoritmo. Una permite repetir automáticamente un grupo de pasos (repetición). Otra permite seleccionar una acción de entre un par de alternativas especificas, teniendo en cuenta determinas condiciones (selección).

El teorema de Böhm - Jacopini [C.Böhm, G.Jacopini, Comm. ACM vol.9, nº5, 366-371, 1966] establece que:

> "Todo programa propio se puede escribir utilizando únicamente las estructuras de control secuencial, condicional e iterativa".

Un programa se define como propio si cumple las siguientes condiciones:

- Si tiene un solo punto de entrada y un solo punto de salida
- Existen caminos desde la entrada hasta la salida que pasan por todas las partes del programa.
- Todas las instrucciones son ejecutables y no existen bucles sin fin.

También llamado "Teorema Fundamental de la Programación Estructurada".

3.1. Diagramas de flujo

Con frecuencia es más sencillo expresar ideas gráficamente que mediante texto, un intento de proporcionar una visión gráfica de la descripción de algoritmos son los diagramas de flujo.

Los diagramas de flujo son una herramienta gráfica para descripción de algoritmos. Un diagrama de flujo consta de una serie de símbolos estándar, que representan las distintas acciones del algoritmo, conectados mediante líneas que indican el orden en el cual deben realizarse las operaciones.

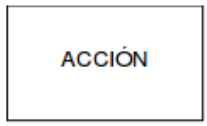
Un diagrama de flujo muestra la lógica del algoritmo, acentuando los pasos individuales y sus interconexiones.

Un diagrama de flujo debe reflejar :

- El comienzo del programa
- Las operaciones que el programa realiza
- El orden en que se realizan
- El final del programa

Los símbolos utilizados han sido normalizados por las organizaciones ANSI (American National Standard Institute) y por ISO (International Standard Organization) y son los siguientes:

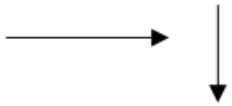
Indican el inicio y el fin del algoritmo. Del primero sólo puede salir una línea y al segundo sólo puede llegar una línea.



Indica una acción a realizar. Tiene una única línea de entrada y otra de salida.

Indica una acción de entrada (lectura de datos desde el teclado) o salida (escritura de un dato por pantalla). Tiene una única línea de entrada y otra de salida.

Se usan para tomar una decisión, dependiendo de cierta condición. Tiene una línea de entrada y dos de salidas (una por la cual salir, si la condición es cierta, y otra, si es falsa).



Se usan para conectar los símbolos anteriores y especifican el flujo del algoritmo.

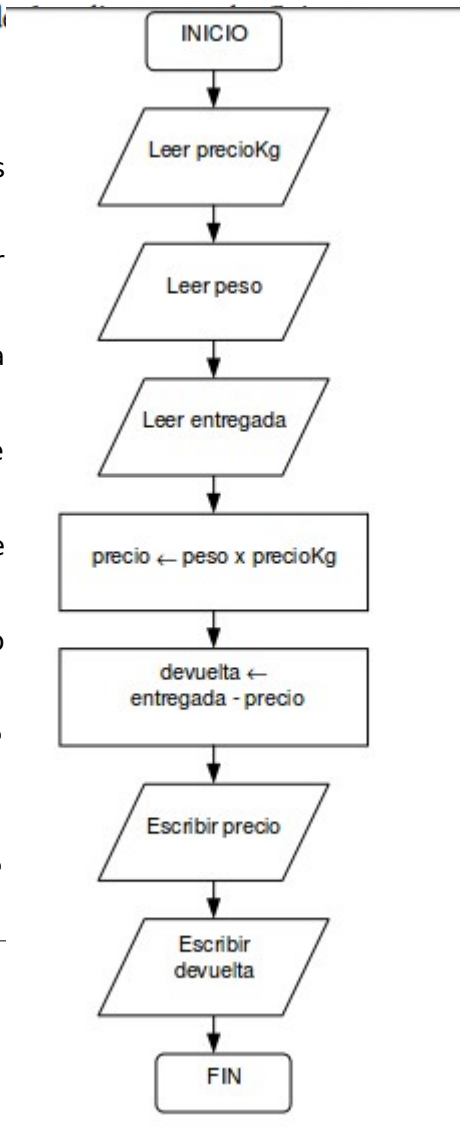
Se usan para conectar distintos puntos d
evitando cruces de flechas.

En la representación es conveniente seguir las siguientes reglas:

- El comienzo del programa figurará en la parte superior del diagrama.
- Los símbolos de comienzo y fin deberán aparecer una única vez.
- El flujo de las operaciones será de arriba a abajo y de izquierda a derecha.
- Se debe guardar cierta simetría en la representación de bifurcaciones y bucles.
- Se evitarán los cruces de líneas de flujo, utilizando conectores.

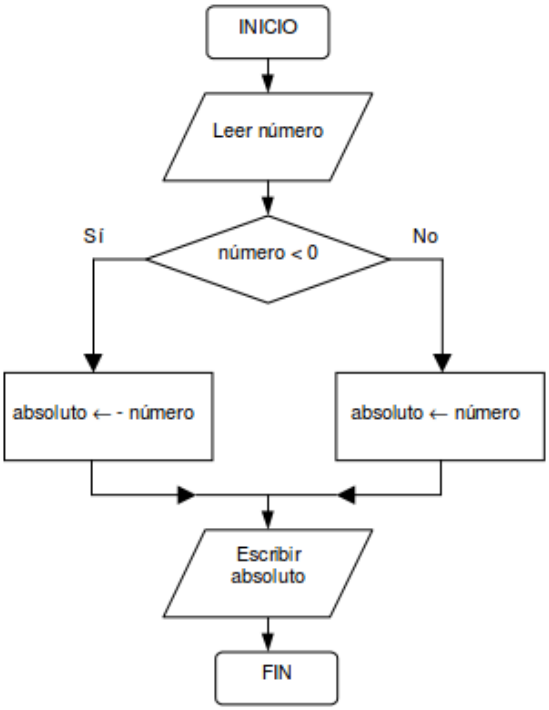
Ejemplo: Dibujar un diagrama de flujo para el problema de calcular el precio y la cantidad a devolver por la compra de un producto.

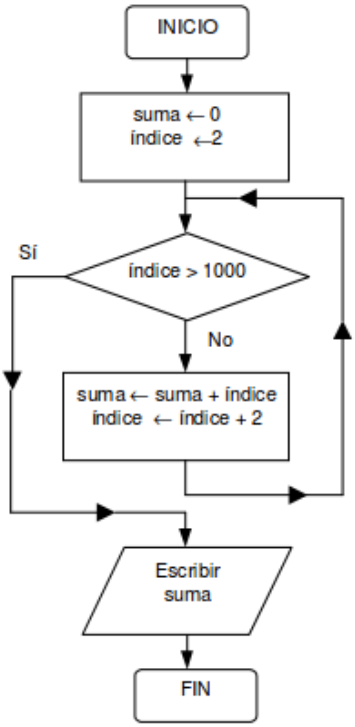
Ejemplo: Dibujar un diagrama de flujo para el problema de



calcular el valor absoluto de un número. Escribir precio

Ejemplo: Dibujar un diagrama de flujo para calcular la siguiente suma: Escribir 2 + 4 + 6 + ... 998 + 1000





En los últimos años la actitud de los profesionales de la computación hacia el una vez popular diagrama de flujo se ha enfriado notablemente, sobre todo en base a que los diagramas de flujo muestran la lógica de control de un algoritmo pero oscurecen su estructura.

3.2. Pseudocódigo

Los programas deben ser escritos en un lenguaje que pueda entender el ordenador, pero no olvidemos que nuestra forma normal de expresar algo es en lenguaje natural. De la aproximación entre ambos surge una herramienta para la descripción de algoritmos: el pseudocódigo . Es por tanto un lenguaje algorítmico que permite representar las construcciones básicas de los lenguajes de programación, pero a su vez, manteniéndose próximo al lenguaje natural.

A pesar de su flexibilidad el pseudocódigo tiene que atenerse a una serie de normas para que los algoritmos construido resulten legibles, claros y fácilmente codificables, con este fin se le imponen algunas restricciones tales como:

- Los identificadores usados han de tener un significado de acuerdo a su contenido.
- El conjunto de sentencias debe ser completo, en el sentido de permitir especificar cualquier tarea a realizar con suficiente detalle.

- Contener un conjunto de palabras reservadas.

Las principales ventajas del pseudocódigo son:

1. Podemos centrarnos sobre la lógica del problema olvidándonos de la sintaxis de un lenguaje concreto.
2. Es fácil modificar el algoritmo descrito.
3. Es fácil traducir directamente a cualquier lenguaje de programación el algoritmo obtenido.

El principal inconveniente que presenta el uso del pseudocódigo como lenguaje de descripción de algoritmos es la imprecisión.

El pseudocódigo que nosotros utilizaremos consta exactamente de las siguientes construcciones:

- INICIO : Indica el comienzo del algoritmo
- FIN : Indica la finalización del algoritmo.
- LEER : Se usa para leer un dato del teclado.
- ESCRIBIR : Se usa para escribir un dato por pantalla.
- SI <c> ENTONES <aSí> EN OTRO CASO <aNo> FINSI : donde <c> es una condicción que puede ser cierta o falsa y <aSí> y <aNo> son dos acciones. Indica realizar la acción <aSí> si la condición <c> es cierta o realizar la condición <aNo> si ésta es falsa.
- MIENTRAS <c> HACER <a> FINMIENTRAS : donde <c> es una condición que puede ser cierta o falsa y <a> es una acción. Indica repetir la acción <a> mientras la condición <c> sea cierta. Se deja de repetir en el momento en que <c> se hace falsa. Las comprobaciones de la condición se hacen justo antes de realizar la acción.
- REPETIR <a> HASTA QUE <c> : donde <c> es una condición que puede ser cierta o falsa y <a> es una acción. Indica repetir la acción <a> hasta que la condición sea cierta. Se deja de repetir en el momento en que <c> se hace cierta. Las comprobaciones de la condición se hacen justo después de realizar la acción.

Ejemplo: Veamos como se expresa en pseudocódigo la solución al problema del peso:

```
INICIO
    LEER precioKg
    LEER peso
    LEER entragada
    precio ← peso x precioKg
    devuelta ← entregada – precio
    ESCRIBIR precio
```

```
        ESCRIBIR devuelta
FIN
```

❑ Ejemplo: Cálculo del valor absoluto de un número:

```
INICIO
    LEER número
    SI número < 0 ENTONCES
        absoluto ← - número
    EN OTRO CASO
        absoluto ← número
    FINSI
    ESCRIBIR absoluto
FIN
```

Ejemplo: calculo de la suma 2 + 4 + 6 + ... 998 + 1000:

```
INICIO
    suma ← 0
    índice ← 2
    MIENTRAS indice <= 1000 HACER
        suma ← suma + i
        índice ← índice + 2
    FINMIENTRAS
    ESCRIBIR suma
FIN
```

que también puede ser expresado del siguiente modo:

```
INICIO
    suma ← 0
    índice ← 2
    REPETIR
        suma ← suma + índice
        índice ← índice + 2
```

```
    HASTA QUE i > 1000
    ESCRIBIR suma
FIN
```

Relación de Problemas (Tema 2)

1. Escribir diagramas de flujo para los siguientes problemas:

    a) Leer tres números del teclado, calcular el mayor de ellos y escribir éste por pantalla.

    b) Leer tres números del teclado y determinar si la suma de cualquier pareja de ellos es igual al otro. Se deberá escribir por pantalla “Sí” si la propiedad se cumple o “No” en otro caso.

2. Leer números del teclado hasta que se obtenga un cero y calcular la media de los valores leídos hasta ese momento. Se deberá escribir por pantalla la media. El cero no se tendrá en cuenta en el cálculo de la media.

3. Leer los tres coeficientes de la ecuación de segundo grado ax 2 +bx+c = 0 y escribir las soluciones correspondientes por pantalla.

4. Calcular el producto 1 x 2 x 3 ...x 1000. El resultado deberá escribirse por pantalla.

5. Escribir algoritmos para los problemas anteriores utilizando pseudocódigo.

6. ¿Qué resultado escribirá en pantalla el siguiente algoritmo?

```
INICIO
    n ← 2
    doble ← n x 3
    ESCRIBIR n
    ESCRIBIR doble
FIN
```