



GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: REPRESENTACIÓN DE ALGORITMOS

1. Introducción

Los concursos ICPC y IOI su esencia es la solución de un conjunto de problemas lógicos y matemáticos con el uso de habilidades en análisis, diseño e implementación de soluciones algorítmicas las cuales son ejecutadas por una computadora a través de un programa que se va implementar con el uso de algún lenguaje de programación.

Un gran porcentaje de las soluciones incorrectas a los problemas radica en un mal análisis y diseño del algoritmo o sencillamente obviar este paso y pasar directamente a la codificación el resto son producto de detalles propios del lenguaje de programación seleccionado.

En la siguiente guía vamos a abordar las diferentes formas de representar un algoritmo que nos va a permitir elaborar nuestros propios algoritmos y con ello chequear su eficacia o falla que presente, de igual forma puede ayudarnos en la codificación del mismo.

2. Conocimientos previos

2.1. Algoritmo

Es un conjunto de pasos secuenciales y ordenados que permiten lograr un objetivo. Que sean pasos secuenciales significa que deben ser ejecutados uno después de otro y que sean pasos ordenados quiere decir que deben llevar un orden quasi-obligatorio (u obligatorio en la mayoría de los casos).

3. Desarrollo

Podemos expresar un algoritmo de muchas maneras, incluyendo lenguaje natural, diagramas de flujo, pseudocódigo y diagrama NASSI – SCHNEIDERMAN.








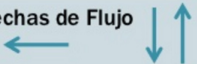

3.1. Lenguaje natural

El lenguaje natural es popular, pues se nos da naturalmente y puede comunicar los pasos de un algoritmo a una audiencia general. Cuando desarrollamos algoritmos, a menudo trabajamos con personas que saben programación y con algunos que no; pero todos conocen el lenguaje natural.

Sin embargo, el lenguaje natural tiene inconvenientes. Tiende a ser ambiguo y a estar definido vagamente, pues carece de estructura precisa. Esto dificulta que otros sigan un algoritmo y se sientan seguros de que es correcto. Los diagramas de flujo y el pseudocódigo son formatos más estructurados que pueden expresar un algoritmo de manera más precisa, y son populares con científicos de computación y programadores.

3.2. Diagramas de flujo

Una manera más formal de expresar un algoritmo es con un diagrama de flujo, un diagrama con cajas conectadas por flechas.

símbolo	Función	Símbolo	Función
Terminal 	Indicar el inicio y fin del diagrama	Teclado 	Introducir datos manualmente por el teclado
Entrada/salida 	Entrada o salida simple de información	Decisión 	Indica operaciones lógicas o de comparación y tienen dos salidas dependiendo del resultado.
Proceso 	Realizar cualquier operación o calculo con la información	Conectores 	Une dos partes del diagrama a la misma o diferente página
Salida a Impresora 	Salida de informacion a la impresora	Flechas de Flujo 	Indica la direccion del flujo de la información
Salida a Pantalla 	Mostrar información de salida a la pantalla		

Objetivos del diagrama de flujo

- Ofrecer una descripción visual de las actividades implicadas en un proceso mostrando la relación secuencial ente ellas.
- Facilitar la rápida comprensión de cada actividad y su relación con las demás, el flujo de la información, las ramas en el proceso, el número de pasos del proceso, etc.
- Facilitar la selección de indicadores de proceso.
- Estimula el pensamiento analítico en el momento de estudiar un proceso, haciendo más factible generar alternativas útiles.

Expresar un algoritmo como un diagrama de flujo nos permite visualizar el algoritmo a nivel alto, además de que nos obliga a pensar muy cuidadosamente en la secuenciación y selección. ¿Cuál flecha va a cuál nodo? ¿Faltan flechas? Estos son los tipos de preguntas valiosas que pueden surgir al traducir un algoritmo a un diagrama de flujo.

3.3. Pseudocódigo

Finalmente la mayoría de los algoritmos se transforman en código para ejecutar en una computadora. Antes de eso, los programadores a menudo prefieren expresar un algoritmo en pseudocódigo: un código que utiliza construcciones de un lenguaje de programación, pero que en realidad, no se ejecuta.

Cada programador escribe pseudocódigo de manera diferente, pues no hay un estándar oficial, así que puedes toparte con pseudo-código que se vea muy diferente.

Expresar un algoritmo en pseudocódigo ayuda a un programador a pensar en términos familiares, sin preocuparse por la sintaxis y detalles específicos. También le provee a los científicos de computación una forma independiente del lenguaje para expresar un algoritmo, de manera que

los programadores de cualquier lenguaje puedan tomarla, leer el pseudo-código y traducirlo a su lenguaje favorito.

Escribir en pseudocódigo es similar a escribir en un lenguaje de programación. Cada paso del algoritmo está escrito en una línea propia en secuencia. Por lo general, instrucciones se escriben en mayúsculas, variables en minúsculas y mensajes en mayúsculas y minúsculas.

Al escribir pseudocódigo, todos suelen tener su propio estilo de presentar las cosas, ya que lo leen los humanos y no una computadora. Sus reglas son menos rigurosas que las de un lenguaje de programación. Sin embargo, hay algunas reglas simples que ayudan a que el pseudocódigo sea más universalmente entendido.

Reglas para escribir pseudocódigo

- Siempre escribe en mayúscula la palabra inicial (a menudo una de las 6 construcciones principales).
- Tener sólo una declaración por línea.
- Aplicar sangría para mostrar jerarquía, mejorar la legibilidad y mostrar construcciones anidadas.
- Finaliza siempre las secciones de varias líneas con cualquiera de las palabras clave END (ENDIF, ENDWHILE, etc.).
- Mantén tus declaraciones independientes del lenguaje de programación.
- Mantenlo simple, conciso y legible.

Una de las ventajas que presenta esta forma de representación de algoritmos es que facilita mucho su escritura posterior en un lenguaje de programación y le permite al diseñador aprender de una manera más sencilla debido a su familiaridad con cualquier lenguaje de programación.

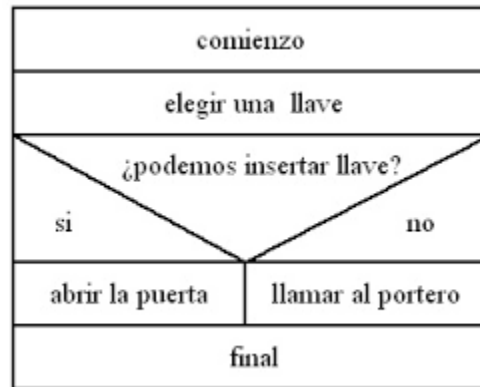
Otra ventaja con que se cuenta, es que es independiente de la plataforma o lenguaje de programación que se desee utilizar, ya que su escritura no implica palabras reservadas de ningún tipo de lenguaje.

3.4. Diagrama NASSI – SCHNEIDERMAN

Conocido como diagrama N-S, se considera como otra forma de representar algoritmos. Se basa en escribir las instrucciones en bloques o cuadros de texto. Este diagrama también se conoce con el nombre de diagrama de Chapin y es una combinación de la escritura en pseudocódigo y del diagrama de flujo.

En el diagrama N-S, las flechas que indican el flujo del algoritmo, se reemplazan por las cajas de texto, en las cuales se escriben las instrucciones respectivas.

Como se puede apreciar, cada bloque contiene una instrucción, en el caso de definición de variables, se recomienda utilizar un bloque por cada tipo de datos distinto que se genere, es decir si existen varios datos enteros, su definición abarcaría un solo bloque, si por el contrario se requieren definir datos dobles, ellos ocuparían otro bloque y así sucesivamente.

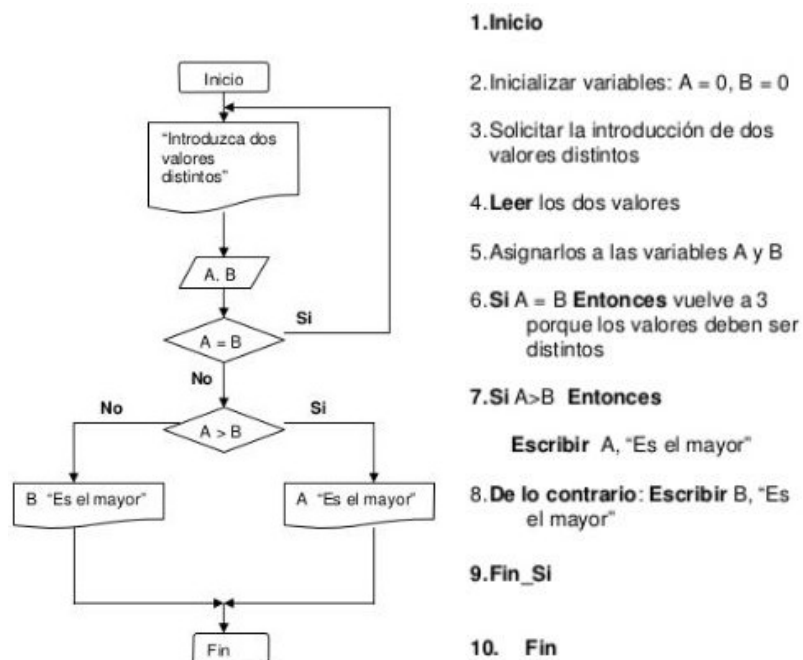


Las principales desventajas de este diagrama radica en si el algoritmo es demasiado largo, este tipo de diagrama es difícil de realizar y consume bastante tiempo de elaboración dos elementos muy importante a tener en cuenta ya que en las soluciones a problemas algunos de los algoritmos pueden ser largos y no podemos perder mucho tiempo en la elaboración ya que en una competencia el factor tiempo es importante.

4. Implementación

Vamos a ver como podemos utilizar algunas de las representaciones hasta ahora abordadas. Tomemos pues el siguiente problema de ejemplo:

Hacer un algoritmo que permita leer 2 números diferentes y nos diga cual es el mayor de los 2 números.



Aquí hemos representado el algoritmo solución del mismo problema de la misma manera la izquierda por diagrama de flujo mientras a la derecha podemos observar el pseudocódigo del mismo problema.

5. Complejidad

La complejidad de las representaciones va depender en gran medida de dos factores, el primero va ser la complejidad en si del algoritmo que se desea representar y en segundo de la habilidad del concursante en el trabajo con determinada representación.

Por lo general se comienza por el diagrama de flujo pero a medida que se va cogiendo experiencia por parte de los concursantes van desarrollando sus algoritmos con pseudocódigo.

6. Aplicaciones

Es una pérdida de tiempo; como dicen, ¿por qué escribir código dos veces?.

Eso podría ser correcto en el caso de problemas simples y directos. Sin embargo, a medida que aumentan la complejidad y el tamaño del problema, comienzan a darse cuenta de que la generación de la representación del algoritmo facilita mucho la escritura del código real. Le ayuda a darse cuenta de posibles problemas o fallas de diseño en el algoritmo antes de la etapa de desarrollo.

Por lo tanto, ahorra más tiempo y esfuerzo en corregir errores y evitar errores. Además, el las diferentes representaciones de algoritmos permitió a los programadores comunicarse de manera más eficiente con otras personas de diferentes orígenes, ya que ofrece la idea del algoritmo sin la complejidad de las restricciones de sintaxis.

Una representación del algoritmo claro, conciso y directo puede marcar una gran diferencia en el camino que va desde la idea hasta la implementación, un camino tranquilo para el programador. Es una de las herramientas generales subestimadas por la comunidad de programación pero, desafiante, debe utilizarse más.

7. Ejercicios propuestos

A continuación le proponemos una lista de ejercicios los cuales análisis y diseñe un algoritmo solución utilizando una de las siguientes variantes: diagrama de flujo o pseudocódigo:

- [DMOJ - A Plus B](#)
- [DMOJ - Hello World](#)
- [DMOJ - Par o Impar](#)
- [DMOJ - El Ogro Ork](#)
- [DMOJ - Las Notas de Ork](#)

- DMOJ - Sum