Software para la enseñanza-aprendizaje de algoritmos estructurados

J. Jesús Arellano Pimentel[1], Omar S. Nieva García[1], Rocío Solar González[1], Getsemaní Arista López[1]

[1] Universidad del Istmo Campus Tehuantepec, Bo. Sta. Cruz Tagolaba, Sto. Domingo Tehuantepec, Oaxaca, México.
jjap@sandunga, omarng@bianni, solgr@sandunga{.unistmo.edu.mx}, arloget@gmail.com

Resumen

Actualmente existen diversas herramientas de software que sirven como recurso didáctico en la enseñanza-aprendizaje de algoritmos estructurados a nivel superior. Sin embargo, la gran mayoría solo hace énfasis en el diseño y prueba de los algoritmos. En el presente trabajo se propone un nuevo software, con base en la heurística de resolución de problemas de Polya, que da soporte a las fases de análisis y planteamiento del problema, además del diseño y traza completa de la prueba. Se realiza un análisis cualitativo entre el software propuesto y otras tres herramientas ampliamente utilizadas a nivel superior. La estructura y funcionalidad del software propuesto contribuye a que el estudiante adquiera, practique y ejercite la capacidad de resolver problemas de forma metódica a través de soluciones algorítmicas estructuradas.

Palabras clave: Algoritmos estructurados, Software de enseñanza-aprendizaje, Heurística de Polya.

Abstract

Currently there are several software tools that serve as a teaching resource in the teaching and learning of structured algorithms at a higher level. However, the vast majority only emphasizes the design and testing of algorithms. In this paper we propose new software, based on problem-solving heuristics of Polya, which supports the phases of analysis and problem statement, besides the complete design and test trace. Qualitative analysis is performed between the proposed software and three other widely used tools at a higher level. The structure and function of the proposed software helps students acquire, practice and exercise the ability to solve problems methodically through structured algorithmic solutions

Keywords: Structured Algorithms, Software for teaching & learning, Polya heuristics.

1. Introducción

Hoy en día, los estudiantes de nivel superior que cursan carreras de ingeniería y especialmente aquellos inscritos en carreras afines a las ciencias computacionales, necesitan aprender a programar soluciones algorítmicas a problemas computables, utilizando diferentes paradigmas con diversos lenguajes y herramientas de programación.

En ocasiones, la forma habitual de enseñar a programar es forzar al estudiante a resolver un gran número de ejercicios y problemas con el método de codificar, probar y corregir hasta lograr que el programa produzca los resultados correctos. En este esquema de enseñanza-aprendizaje se deja de lado la importancia de entender a cabalidad un problema para concebir un algoritmo de solución, esto a su vez ocasiona que más que formar programadores se esten formando codificadores. Cabe recordar que codificar y programar no son lo mismo [1], por un lado, codificar es la acción de escribir en un lenguaje de programación una solución ya encontrada, por otro lado, programar parte desde la comprensión del problema para planear y diseñar una solución algorítmica que posteriormente será codificada.

En general, es un hecho aceptado por la comunidad académica que aprender a programar algoritmos resulta muy difícil para la mayoría de los estudiantes [2]. Afortunadamente, la tendencia de diez años a la fecha es utilizar herramientas de software como apoyo didáctico para facilitar la enseñanza-aprendizaje de algoritmos. Entre estas herramientas se encuentran las que explotan el uso de micromundos tales como Alice [3] y JKarelRobot [4] usadas principalmente en educación básica y media superior. A nivel universitario es más frecuente el uso de herramientas basadas en representaciones de pseudocódigo o diagramas de flujo, entre estas destacan: PSeInt [5], RAPTOR [6][7][8] y DFD [9][10]; además, a nivel superior también se emplean entornos de desarrollo integrado como ZinjaI [11] que

permiten visualizar el diagrama de flujo a partir del código fuente que utilizan.

El inconveniente que observamos con las herramientas antes mencionadas, y la gran mayoría de las que también están disponibles de forma gratuita en línea, es que asumen que el estudiante ya sabe cómo analizar un problema y esbozar una solución en términos de los datos que debe recibir y los que debe producir un algoritmo que pretende dar solución a un determinado problema. En otras palabras, no están estrechamente vinculadas con alguna heurística de resolución de problemas como la propuesta por Polya [12].

En la sección 2 del presente trabajo se analizan tres de las herramientas de mayor uso a nivel universitario. En la sección 3 se describen los antecedentes, estructura y funcionamiento del nuevo software propuesto. La sección 4 presenta los resultados de una comparación cualitativa entre las tres herramientas analizadas y el nuevo software desde tres perspectivas. Finalmente la sección 5 se presenta las conclusiones y las líneas de investigación futuras.

2. Selección y análisis del software

Afín de tener un cierto grado de equidad entre las herramientas de software a comparar en este trabajo, es preciso indicar los dos principales criterios de selección:

1. Haberse construido con el propósito de ser un recurso didáctico en la enseñanza-aprendizaje de algoritmos.
2. Emplear diagramas de flujo para la representación de las soluciones algorítmicas.

El primer criterio deja fuera de nuestro análisis a los ambientes de desarrollo integrado como ZinjaI. El segundo criterio permite excluir aquellas herramientas con base en micromundos tales como Alice o JKarelRobot.

En cada una de las herramientas a analizar se resolverá un problema sencillo que está presente en prácticamente todos los cursos iniciales de programación o algoritmos: "Elevar el número 2 a una potencia entera positiva dada por el usuario".

2.2. DFD

Esta herramienta de software surge en 1988 de un proyecto colombiano llamado *Editor e Intérprete de Algoritmos Representados en Diagramas de Flujo* [9] con el objetivo de lograr eliminar ciertas dificultades presentes en el estudio de algoritmos básicos. La última versión disponible en línea (octubre de 2008) lleva el nombre de FreeDFD y tiene su página Web en Google Code [10]. En esta página DFD se redefine como " un editor, intérprete y depurador de algoritmos representados en diagramas de flujo". Con esta herramienta el usuario puede trabajar con expresiones complejas que involucren constantes, variables, funciones y operadores. Los tipos de datos que maneja son: reales, cadenas de caracteres y lógicos; también se permite el uso de arreglos. Durante la ejecución de un diagrama se detectan errores de sintaxis y de conformación de subprogramas.

Su implementación se realizó en el lenguaje C++ y se ejecuta bajo plataformas Windows de 32 bits. La última versión soporta los idiomas: español, inglés y portugués. No requiere de la instalación adicional de librerías y viene con un directorio de ejemplos y ayuda al estilo Win32. Como trabajo futuro pretenden portarlo completamente a entornos Linux.

A decir de sus autores la interfaz gráfica con la que cuenta (ver Figura 1) facilita el trabajo con diagramas simulando el trabajo que se realizaría en papel.

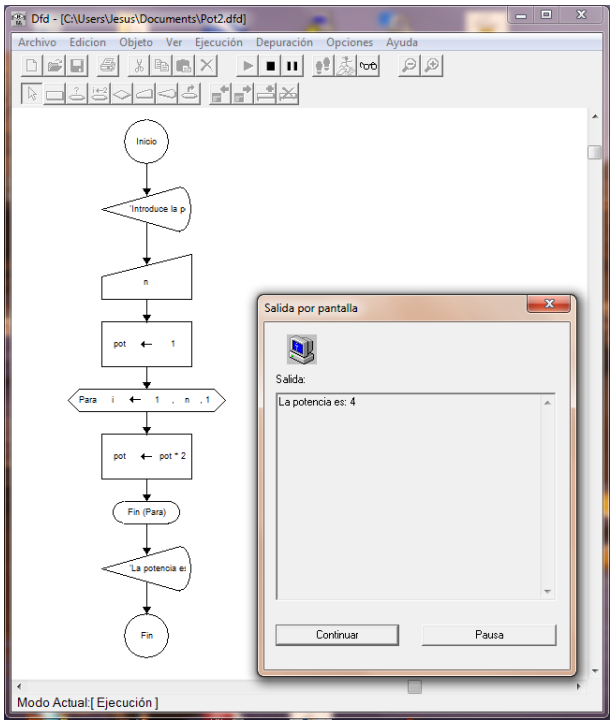


Figura 1: Interfaz de la herramienta DFD.

Desde nuestro punto de vista la herramienta DFD presenta los siguientes inconvenientes: 1) carece de soporte para el análisis del problema y planteamiento de la solución, inclusive no cuenta con algún elemento para editar el enunciado del problema, 2) la prueba del diagrama basa su ejecución en dos cuadros de diálogo, uno presenta las salidas a pantalla y el otro solicita las entradas al usuario sin mostrar la traza completa del algoritmo que incluya la evaluación de las expresiones lógicas inmersas en las condiciones de

selección o cíclicas, 3) no utiliza la notación estándar de la mayoría de los libros de algoritmos para representar las estructuras cíclicas *para* y *mientras*, 4) no se realizan validaciones semánticas para evitar ciclos infinitos, 5) el tamaño de los elementos del diagrama no se ajusta a la longitud del texto que contienen y tampoco acepta la edición de múltiples líneas de texto, 6) no cuenta con la estructura cíclica *hacer hasta*, 7) no traduce el diagrama a algún lenguaje de programación estructurado, y 8) solo se ejecuta de forma completa en plataformas Windows, para las plataformas Linux debe instalarse el Wine.

2.3. RAPTOR

RAPTOR (acrónimo del inglés *Rapid Algorithmic Prototyping Tool for Ordered Reasoning*) surge en 2004 como un proyecto de la Academia de la Fuerza Aérea de Estados Unidos. En [6] se define como: "un ambiente de programación con base en diagramas de flujo, diseñado específicamente para ayudar a los estudiantes a visualizar sus algoritmos y evitar el bagaje sintáctico". En su página Web [8] la última actualización tiene fecha del 13 de septiembre de 2012.

El propósito del software es reducir las dificultades inherentes a los entornos no visuales y a la sintaxis rígida de los lenguajes de programación. Un usuario puede agregar, seleccionar, copiar, cortar y pegar símbolos al diagrama de tipo: asignación, llamado a procedimiento, entrada, salida, selección y ciclo. La construcción del diagrama fuerza al usuario a generar un diagrama de flujo estrictamente estructurado. La sintaxis de estos símbolos es flexible en el sentido de que se pueden utilizar varios estilos relacionados con lenguajes de programación como Ada, C, o Pascal; esto incluye tanto sentencias como operadores. Además, permite el uso de más de 40 funciones de librería (trigonometría, números aleatorios y dibujo).

RAPTOR fue implementado combinando los lenguajes Ada, C# y C++, de tal forma que se ejecuta bajo la plataforma .NET. Por tal motivo su ejecución en Linux requiere de la instalación de librerías adicionales.

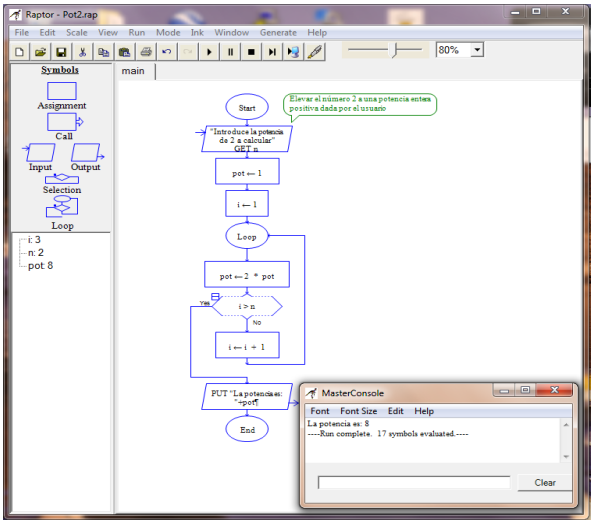


Figura 2: Interfaz de la herramienta RAPTOR.

La interfaz gráfica del software (ver Figura 2) permite la adición de comentarios a los símbolos del diagrama de flujo, por tanto es posible editar el enunciado inicial del problema como un comentario del símbolo inicio (*Start*). Los comentarios pueden estar en cualquier idioma, sin embargo la herramienta esta diseñada completamente para el idioma inglés.

Desde nuestro punto de vista la herramienta RAPTOR tiene los siguientes inconvenientes: 1) aún cuando permite editar comentarios relacionados con los símbolos del diagrama, carece de soporte para el análisis del problema y planteamiento de la solución, 2) a pesar de que la prueba del algoritmo señala cuál es el símbolo en ejecución y se visualiza el cambio de valor de las variables, no se presenta la traza completa de la ejecución en una pantalla que incluya la evaluación de las expresiones lógicas inmersas en las condiciones o ciclos, 3) no utiliza la notación estándar de la mayoría de los libros de algoritmos para representar la estructura cíclica *hacer hasta*, 4) no se realizan validaciones semánticas para evitar ciclos infinitos, 5) el símbolo de asignación solo acepta una asignación lo cual incrementa el tamaño de un diagrama, 6) no cuenta con las estructuras cíclicas *para* y *mientras*, 7) solo se ejecuta de forma completa en plataformas Windows, para Linux debe instalarse la plataforma Mono, y 8) el único idioma disponible es el inglés.

2.4. PSeInt

PSeInt (abreviatura de Pseudo Intérprete) surge en 2003 como un proyecto para la materia de Programación I en la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral. Se trata de un intérprete de pseudocódigo basado en los contenidos de la cátedra de Fundamentos de Programación [5]. Está herramienta se diseñó para que los estudiantes sin experiencia en programación aprendieran la lógica y conceptos

básicos de un algoritmo computacional mediante la utilización de un simple y limitado pseudo-lenguaje intuitivo y en español.

Este software pretende facilitarle al principiante la tarea de escribir algoritmos en pseudocódigo presentando un conjunto de ayudas y asistencias para brindarle la capacidad de encontrar errores y comprender la lógica del pseudo-lenguaje generando el diagrama de flujo a partir de este.

La interfaz gráfica de PSeInt (ver Figura 3) permite la edición del pseudocódigo con resaltado tipográfico, autocompletado, ayudas emergentes, plantillas de comandos e indentado inteligente para facilitar su empleo y aprendizaje. Además el lenguaje del pseudocódigo es configurable.

En su página Web, PSeInt dispone de versiones ejecutables para las plataformas Windows, Linux y Mac OS. Además en la página también es posible descargar diversos manuales, documentación y ejemplos, así como acceder a foros de discusión.

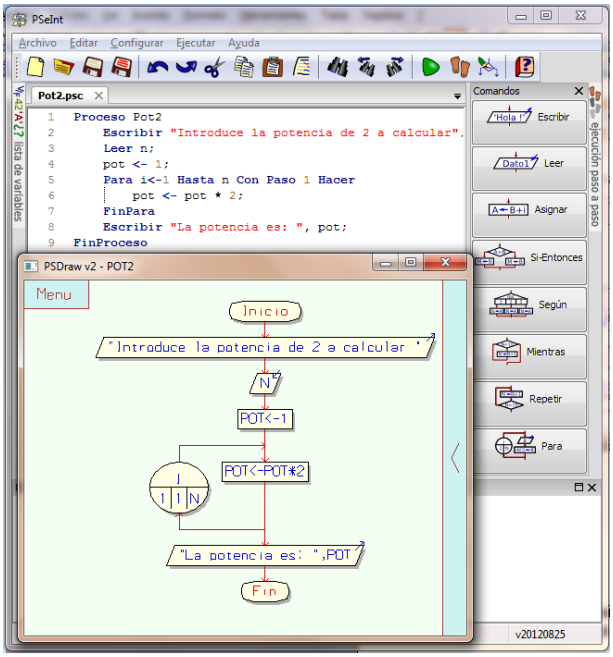


Figura 3: Interfaz de la herramienta PSeInt.

Desde nuestro punto de vista la herramienta PSeInt tiene los siguientes inconvenientes: 1) aún cuando permite editar comentarios inmersos en el pseudocódigo, se carece de soporte para el análisis del problema y planteamiento de la solución, 2) la ejecución simula a la mayoría de los ambientes de desarrollo y no visualiza el cambio de valor de las variables, mucho menos presenta la traza completa de la ejecución, 3) el diagrama de flujo generado no concuerda del todo con la notación estándar de la mayoría de los libros de algoritmos en lo que respecta a entrada, salida y el ciclo *para*, 4) no se realizan validaciones semánticas para evitar ciclos infinitos, 5) la edición del diagrama de flujo solo permite cambiar el contenido de los elementos del diagrama, 6) la tipografía de las variables del pseudocódigo no corresponde con las del diagrama de flujo, y 7) al tratarse de una herramienta que emplea principalmente el pseudocódigo, aún cuando se tiene autocompletado, se incrementa la posibilidad de errores de sintaxis y construcción de las estructuras secuenciales y de control para la especificación de algoritmos, lo cual es un problema común en este tipo de herramientas.

3. Descripción y análisis de la nueva propuesta de software

3.1 Antecedentes

La idea de desarrollar una nueva herramienta de software se planteo desde 2009 en [13] como parte de un método de enseñanza-aprendizaje que pone énfasis en el desarrollo de capacidades de abstracción y resolución de problemas. El software fue concebido como el principal apoyo didáctico de la metodología propuesta. Posteriormente, en [14] se presentó una primera versión prototipo del software que originalmente trabajaba sobre plataformas Linux con cierta portabilidad para sistemas Windows. Finalmente en [15] se muestra la experiencia en el desarrollo y uso de este prototipo, además se plantean mejoras para implementarlas en una nueva versión del software, entre las cuales se destacan:

- Traducción del diseño formal del algoritmo dado en diagrama de flujo a pseudocódigo y código fuente en C.
- Visualizar mediante el uso de distintos colores las diversas estructuras de control en un diagrama de flujo para ayudar a una mejor comprensión de estas.
- Empleo de funciones básicas de librería en las operaciones de cálculo. Funciones tales como: raíz cuadrada, seno, coseno, etc.
- Mayor portabilidad entre distintos sistemas operativos implementando la herramienta en el lenguaje Java en lugar de C para Linux.
- Asignación de valores iniciales a los elementos de entrada, salida y auxiliares en la etapa Planear del prototipo.
- Adición de la evaluación de las condiciones en la traza de la corrida de escritorio del diagrama de flujo durante la etapa Probar

Además se corrigieron diversos fallos detectados mediante pruebas exhaustivas a la versión prototipo.

3.2. Estructura y funcionamiento

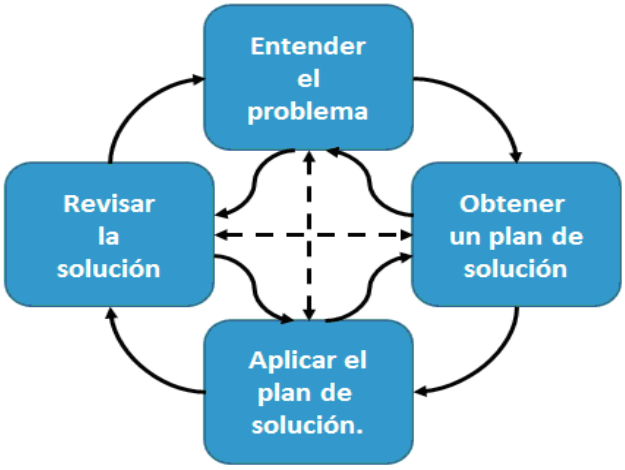
La estructura y funcionamiento del software propuesto se diseño con base en la heurística de resolución de problemas de G. Polya. En la Figura 4 muestra la concordancia que existe entre la heurística de Polya (inciso a) y el software (inciso b). La etapa *Entender el problema* concuerda con la etapa *Analizar*. En la heurística esta primera etapa tiene el propósito de entender el problema a través de una serie de cuestionamientos sobre el enunciado inicial, en lo que respecta al software en la etapa Analizar se edita y analiza el enunciado inicial del problema con base en una serie de 8 preguntas que deben ser resueltas a fin de obtener una descripción más detallada y comprensible del problema.

La etapa *Obtener un plan de solución* concuerda con la etapa *Planear*. En la heurística, el objetivo es idear una estrategia que permita llegar de los datos disponibles hasta la resolución de la incógnita, en el software se trata de utilizar las respuestas de la etapa anterior afín de planear una solución en términos de los elementos de entrada y/o salida así como los elementos auxiliares, asignándoles a todos ellos un rol, un identificador, un tipo de dato (entero, real o carácter) y un valor inicial.

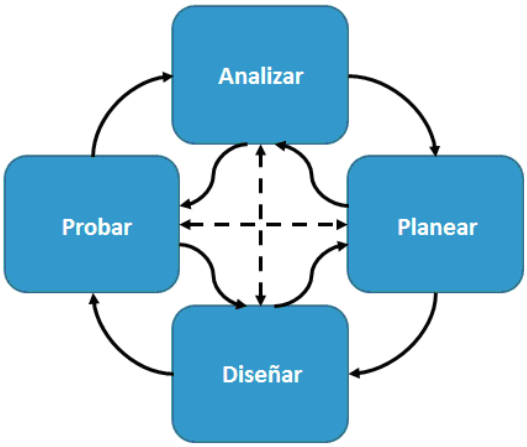
La etapa *Aplicar el plan de solución* concuerda con la etapa *Diseñar un algoritmo*. En la heurística se debe llevar a cabo la estrategia sugerida verificando cada uno de sus pasos, en el software se genera un diagrama de flujo inicial con los elementos previamente registrados, dicho diagrama lo deberá editar el usuario afín de construir una solución algorítmica al problema. Cabe mencionar que la herramienta fuerza al usuario a construir un diagrama de flujo estrictamente estructurado.

Por último, la etapa *Revisar la solución* concuerda con la etapa *Probar*. La heurística indica que se debe analizar la solución para estar seguros que el resultado producido es el que se esperaba, en el software esta etapa consiste en realizar una traza o corrida de escritorio completa del algoritmo diseñado en la etapa anterior, mostrando la evolución del cambio de valor en las variables, la salida a pantalla, la entrada interactiva de datos y la evaluación de las expresiones lógicas de las condiciones y ciclos.

A grandes rasgos la funcionalidad de la herramienta puede visualizase en el diagrama de casos de uso de la Figura 5, aquí es posible observar, además de las etapas de la herramienta como casos de uso, los distintos tipos de elementos (símbolos) que se pueden insertar en un diagrama de flujo (leer, calcular, escribir, condicional, ciclo *para*, ciclo *mientras* y ciclo *hacer mientras*), así como las capacidades de edición durante la etapa de diseño del software.



a) Etapas de la heurística de Polya.



b) Etapas del software propuesto

Figura 4: Concordancia entre las etapas de la heurística de Polya y las etapas del software propuesto.

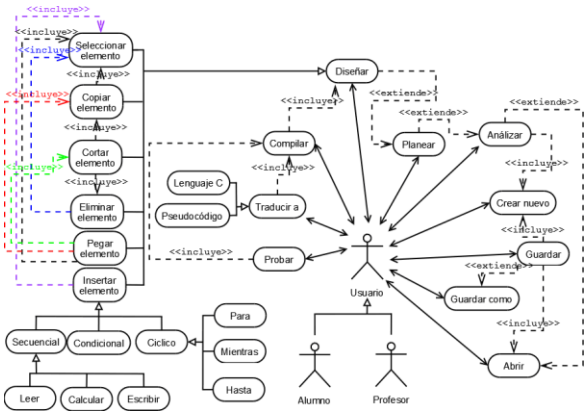


Figura 5: Diagrama de casos de uso del software propuesto.

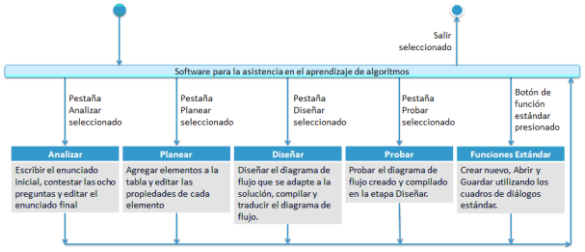


Figura 6: Diagrama de estados del software propuesto.

La interacción que el usuario puede tener con el software se muestra en el diagrama de estados de la Figura 6. Tomando en cuenta que se trata de etapas más que pasos, el software permite al usuario acceder a cualquiera de las pestañas relacionadas con cada una de las etapas de la heurística, sin embargo el software emite mensajes de advertencia al usuario cuando una etapa previa a la actual no se ha concluido satisfactoriamente o es incorrecta. No obstante el orden correcto para el uso del software inicia en la etapa Analizar, sigue con la etapa Planear, continúa con la etapa Diseñar y concluye con la etapa Probar. Es importante resaltar que si el diagrama de flujo contiene algún error sintáctico o semántico no se podrá realizar ni la traducción ni la prueba del algoritmo hasta que sean corregidos todos los errores.

3.3. Caso ejemplo

A fin de mostrar con detalle el uso del software, a continuación se presentan cada una de las etapas y pantallas de la interfaz de usuario involucradas en la solución del problema resuelto con las otras herramientas en la Sección 2: "Elevar el número 2 a una potencia entera positiva dada por el usuario".

En la etapa Analizar (ver Figura 7) las 8 preguntas con sus respectivas respuestas son:

1. ¿Cuáles son los datos iniciales del problema? R= El número 2.
2. ¿Qué es necesario preguntar para completar los datos iniciales? R= La potencia *n* que se desea calcular.
3. ¿De dónde se tomarán los datos iniciales? R= El problema no lo dice.
4. ¿Cuáles son los supuestos? R= 1) Los valores serán tomados del teclado, 2) El número dado por el usuario es entero y positivo, 3) Se presentarán los resultados por pantalla con el enunciado: "La potencia es:" seguido del resultado de la operación.
5. ¿Cuál es la incógnita o incógnitas? R= La potencia *n* del número 2. Es decir el resultado de elevar 2 a la *n*.
6. ¿Qué es lo que se quiere resolver o calcular? R= Multiplicar el número 2 *n* veces, donde *n* es un número entero positivo dado por el usuario. Considerar que: 2 elevado a la 0 es 1, 2 elevado a la 1 es 2, 2 elevado a la 2 es 4, etc.
7. ¿Qué información ha de presentarse como resultado? R= Según los supuestos, mostrar un mensaje que diga: "La potencia es: ", seguido por el resultado de la operación.
8. ¿A través de qué forma se presentarán los resultados? R= Según los supuestos, por pantalla.

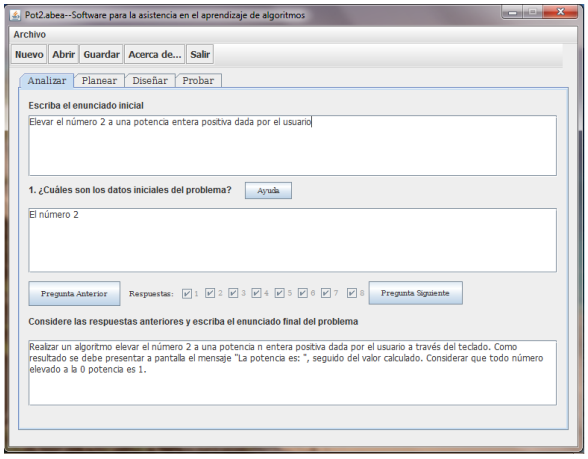


Figura 7: Interfaz de la etapa Analizar.

Las respuestas a las preguntas anteriores permiten formular un problema bien definido y más comprensible de la siguiente manera: Realizar un algoritmo para elevar el número 2 a una potencia entera positiva *n* dada por el usuario a través del teclado. Como resultado se debe presentar el mensaje "La potenccia es:", seguido del valor calculado. Considerar que todo número elevado a la 0 potencia es igual a 1.

La etapa de Planear (ver Figura 8) permite esbozar el plan de solución identificando primeramente los elementos de entrada y/o de salida, además de los elementos auxiliares, a estos elementos se les debe asignar un tipo de dato, un identificador y un valor inicial. Para nuestro caso los elementos se derivan de las respuestas a las preguntas 2, 5 y 6 para la entrada, la salida y la variable auxiliar, respectivamente.

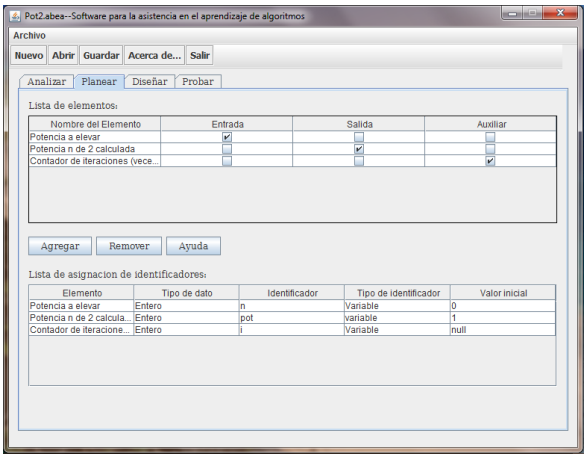


Figura 8: Interfaz de la etapa Planear.

La respuesta a la pregunta 6 también vislumbra el cómo en función del dato de entrada *n* se debe obtener el resultado de salida *pot*.

En la etapa Diseñar (ver Figura 9) se usa la información vertida en la etapa anterior para generar de forma automática un diagrama de flujo inicial que contiene las variables con sus correspondientes valores iniciales. El software provee de una barra de herramientas para adicionar y editar los símbolos del diagrama de flujo necesarios para dar solución al problema.

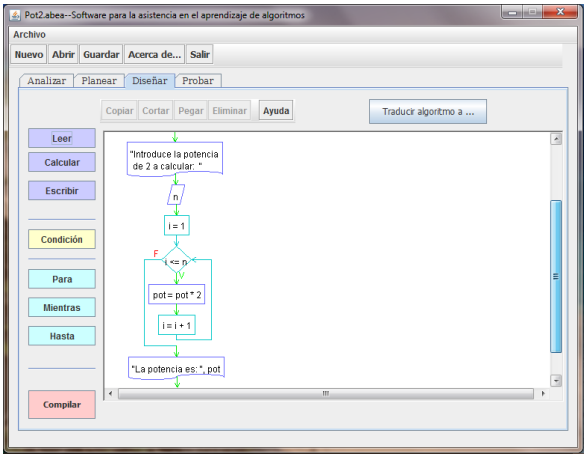


Figura 9: Interfaz de la etapa Diseñar.

A partir del enunciado final del problema se sabe que el elemento de entrada *n* deberá solicitarse al usuario, por tanto se adicionan al diagrama un símbolo escribir para mostrar el mensaje de solicitud y un símbolo leer para la entrada del dato *n*. La respuesta a la pregunta 6 sugiere usar una estructura cíclica, además se conoce el número exacto de veces que ha de multiplicarse el número 2, entonces la estructura a adicionar es un ciclo *para*, cuyos elementos son de un color distinto al de las otras estructuras para mejorar la comprensión de la lógica del algoritmo. Finalmente, del enunciado final también se desprende que el resultado *pot* deberá mostrarse al usuario con un mensaje determinado, por tanto el último símbolo a adicionar en el diagrama será uno de tipo escribir.

Una vez que la solución algortimica ha sido diseñada mediante el diagrama de flujo, se procede a su compilación para detectar errores de sintaxis o semánticos. La semántica verifica que las variables se utilicen según su rol y tipo de dato, además se valida que al menos una de las variables de las expresiones lógicas que conforman las condiciones de paro de los ciclos sea modificada, ya sea por una lectura o asignación dentro del propio ciclo, con esto se minimiza el riesgo de ciclos infinitos. Si uno de los símbolo del diagrama tiene algún tipo de error, entonces se notifica al usuario del error correspondiente y el símbolo del diagrama con su contenido se traza de color rojo para identificarlo con facilidad y posteriormente corregirlo.

Cuando el diagrama esta libre de errores, entonces es posible ejecutar la etapa Probar (ver Figura 10). Esta etapa permite ejecutar símbolo a símbolo el diagrama de flujo señalando en todo momento cuál es el símbolo en ejecución. Para obtener la traza completa del algoritmo todos los símbolos del diagrama de flujo son numerados de forma subsecuente. La traza contiene el número de símbolo que se ejecutó y los cambios de valor que se originaron en las variables así como la informacón mostrada en la pantalla de salida, también se presenta la evaluación de las expresiones lógicas inmersas en las condiciones o ciclos para identificar claramente el flujo de control que siguió la corrida de escritorio. Lo anterior ayuda a comprender la lógica de la solución algorítmica.

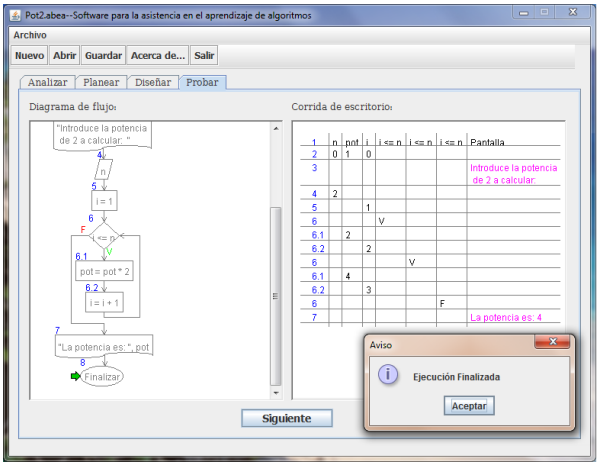


Figura 10: Interfaz de la etapa Probar.

4. Resultados

Los resultados se enfocaron desde tres perspectivas diferentes: 1) Los recursos didácticos que provee cada herramienta para facilitar la enseñanza-aprendizaje de algoritmos estructurados, 2) El soporte que las

herramientas de software dan al paradigma estructurado, y 3) Los recursos tecnológicos involucrados con cada herramienta para su puesta a punto y explotación.

4.2. Recursos didácticos

Los recursos didácticos de interés analizados en cada una de las herramientas son:

1. Soporte de una metodología para la resolución de problemas.
2. Edición interactiva del algoritmo representado en diagrama de flujo.
3. Visualización completa de todo el contenido dentro de cada símbolo del diagrama de flujo
4. Validaciones semánticas para evitar ciclos infinitos y usar variables según su rol.
5. Visualización completa de la traza del algoritmo durante la etapa de prueba incluyendo evaluación de expresiones lógicas.
6. Sistema de ayuda con documentación completa y múltiples ejemplos.
7. Concordancia entre la notación empleada en la mayoría de los libros de algoritmos y la notación empleada en el software para el diseño del diagrama de flujo.

La Tabla 1 sintetiza los recursos didácticos que provee cada una de las herramientas de software. Se ha asignado un puntaje arbitrario entre 0 y 10 puntos, donde 0 representa que definitivamente no se cuenta con el recurso didáctico, y 10 representa que si se cumple a cabalidad.

Tabla 1: Recursos didácticos disponibles en el software.

No. de Recurso	DFD	RAPTOR	PSeInt	Software propuesto
1	0	0	0	10
2	10	10	5	10
3	5	10	10	10
4	0	0	0	10
5	5	7	5	10
6	10	5	10	5
7	5	6	6	9
Totales	35	38	36	64

La comparación empírica de la Tabla 1 muestra que el puntaje alcanzado por el software propuesto es mayor respecto a cualquiera de las otras herramientas, ello se debe a que dispone principalmente de una heurística de resolución de problemas en su diseño estructural, además de otros recursos que las otras herramientas de software no atienden completamente o carecen de ellos.

4.3. Soporte para el paradigma estructurado

El soporte que el software provee para el paradigma estructurado determina el alcance de la herramienta y su posible uso dentro de cursos introductorios de algoritmos y programación estructurada. Las características relacionadas con el paradigma estructurado observadas en el software analizado son las siguientes:

1. Tipos primitivos de datos: enteros, reales, caracteres y lógicos.
2. Arreglos de datos, unidimensionales y bidimensionales.
3. Estructuras de selección simple, compuesta y múltiple.
4. Estructuras cíclicas *para*, *mientras* y *hacer hasta*.
5. Funciones de librería para matemáticas y manejo de cadenas.
6. Procedimientos definidos por el usuario.
7. Traducción a un lenguaje de programación de alto nivel, preferentemente estructurado.

La Tabla 2 sintetiza las 7 características anteriores presentes en las herramientas de software. El puntaje asignado a cada herramienta sigue el mismo criterio que en la Tabla 1.

Tabla 2: Soporte del software para el paradigma estructurado.

No. de Característica	DFD	RAPTOR	PSeInt	Software propuesto
1	10	10	10	7
2	10	10	10	0
3	6	6	10	7
4	6	3	10	10
5	10	10	10	6
6	8	8	10	0
7	0	8	10	10
Totales	50	55	70	40

El puntaje mostrado en la Tabla 2 del software propuesto respecto a cualquiera de las otras herramientas de software es menor. Principalmente se debe a dos aspectos fundamentales: 1) aún carece de soporte para el manejo de arreglos, y 2) también

carece de soporte para construir procedimientos definidos por el usuario.

4.4. Recursos tecnológicos

En cuanto a los recursos tecnológicos involucrados con las herramientas de software consideramos principalmente cuatro de ellos:

1. Ejecución multiplataforma (Linux, Windows y MacOS).
2. Facilidad de instalación en distintas plataformas.
3. Página Web dedicada específicamente al software para proveer de nuevas versiones, documentación y ejemplos.
4. Blog de ayuda para el usuario.

La Tabla 3 sintetiza los 4 recursos anteriores aplicando el mismo criterio que en las dos tablas anteriores.

Tabla 3: Soporte del software para el paradigma estructurado.

No. de Recurso	DFD	RAPTOR	PSeInt	Software propuesto
1	5	4	10	10
2	6	6	10	10
3	8	9	10	5
4	0	0	10	6
Totales	19	19	40	31

En la Tabla 3 sobresale PSeInt, seguido del software propuesto, esto es a consecuencia de que ambas herramientas son multiplataforma. Lo que pone en segundo lugar al software propuesto es que aún no cuenta con una página Web dedicada de forma exclusiva, sin embargo es posible encontrar descargas, documentación y un foro de discusión a través de las páginas de dos de sus autores [16][17].

4.5. Resultados preliminares del uso del software

La influencia positiva que tienen el tipo de herramientas analizadas en este trabajo cuando se pretende enseñar y/o aprender algoritmos y habilidades de programación quedo de manifiesto en [6] y [7]. En cuanto al software que aquí se propone también se han recabado resultados preeliminares satisfactorios de la primera versión prototipo en cursos introductorios de algoritmos, en [15] se reportó que el 93% de los estudiantes que usaron el software mostraron más interés en realizar sus tareas y ejercicios de clase, además el 100% de los alumnos considera que el uso de la herramienta de software los ayudó de manera significativa en el análisis y diseño de las soluciones algorítmicas a los problemas planteados, de hecho el índice de aprobación fue 19% más alto con los estudiantes que usaron el software respecto a los estudiantes que no lo utilizaron.

5. Conclusiones

El software para la enseñanza-aprendizaje de algoritmos estructurados es un recurso didáctico muy valioso que debe ser considerado por profesores y alumnos relacionados con cursos introductorios de algoritmos y programación a nivel superior.

El software propuesto contribuye a que el estudiante adquiera, practique y ejercite la capacidad de resolver problemas de forma metódica aventajando a herramientas similares desde la perspectiva de los recursos didácticos de los cuales dispone, destaca el hecho de que su estructura y diseño se basa en la heurística de resolución de problemas de Polya y la etapa de prueba provee la traza completa del algoritmo incluyendo la evaluación de las expresiones lógicas involucradas en las condiciones y ciclos. Sin embargo, desde la perspectiva del soporte al paradigma estructurado aún se le deben adicionar dos funcionalidades más, el soporte para arreglos de datos y el soporte para procedimientos definidos por el usuario. Estas dos funcionalidades son parte del trabajo futuro de la herramienta, así como la creación de una página Web dedicada exclusivamente al software.

Como línea de investigación futura se planea la publicación de un libro de algoritmos con enfoque en la abstracción y resolución de problemas cuyo principal apoyo didáctico será el software propuesto.

Referencias

[1] G. Levine, Introducción a la computación y programación estructurada. McGraw-Hill, 1989.

[2] J. Bennedsen, M. E. Caspersen, Failure rates in introductory programming, SIGCSE Bulletin, (2007), pp. 32-36.

[3] S. Cooper, W. Dann, R. Pausch, Teaching objects-first in introductory computer science. Proceedings of the 34h SIGCSE technical symposium on computer science education (2003). pp. 191-195.

[4] D. Buck, D. J. Stucki, Jkarelrobot: a casestudy in supporting levels of cognitive development in the computer science curriculum, SIGCSE Bulletin, (2001), pp. 16-20.

[5] P. Novara, PSeInt (2012) .Disponible en: http://pseint.sourceforge.net/, consultado en septiembre de 2012.

[6] M. C. Carlisle, T. A. Wilson, J. W. Humphries, S. M. Hadfield, RAPTOR: Introducing Programming to Non-Majors with Flowcharts. Journal of Computing in Small Colleges, Vol. 19, (2004), pp. 52-60, Consortium for Computing Sciences in Colleges.

[7] M. C. Carlisle, T. A. Wilson, J. W. Humphries, S. M. Hadfield, RAPTOR: a visual programming environment for teaching algorithmic problem solving, in Proceedings of the 36th SIGCSE technical symposium on Computer science education (2005), pp. 176-180, St. Louis, Missouri, USA.

[8] T. A. Wilson, M. C. Carlisle, J. W. Humphries, J. Moore, RAPTOR home page. Disponible en: http://raptor.martincarlisle.com/, consultado en septiembre de 2012.

[9] F. Cárdenas, N. Castillo, E. Daza, Editor e intérprete de algoritmos representados en diagramas de flujo. Informpática educativa, Vol. 11, No. 1, (1998), pp. 101-106, UNIANDES-LIDIE. Disponible en: http://www.colombiaaprende.edu.co/html/estudiantesuperior/1608/article-109522.html

[10] FreeDFD: editor de algoritmos representados en diagramas de flujo, (2008). Disponible en: http://code.google.com/p/freedfd/, consultado en septiembre de 2012.

[11] P. Novara, ZinjaI (2012). Disponible en: http://zinjai.sourceforge.net/, consultado en septiembre de 2012.

[12] G. Polya, How to Solve It. Princeton Science Library Edition, 2004.

[13] O. S. Nieva, J. J. Arellano, Método de enseñanza de algoritmos centrado en 2 dimensiones, 4º Simposio Internacional en Sistemas Telemáticos y Organizaciones Inteligentes SITOI (2009), pp. 881-897, Xalapa, Veracruz, México.

[14] J. J. Arellano, O. S. Nieva, ABEA, Herramienta de apoyo en la enseñanza de algoritmos y habilidades de programación, VI Semana Nacional de Ingeniería Electrónica SENIE (2010), pp. 193-202, Huajuapan de León, Oaxaca, México.

[15] J. J. Arellano, O. S. Nieva, G. Arista, R. Solar, ABEA, Experiencia en el Desarrollo y Uso de un Software para Enseñar algoritmos, 8vo Congreso Internacional de Cómputo en Optimización y Software CICos (2011), pp. 265-276, Cuernavaca, Morelos, México.

[16] J. J. Arellano, http://www.unistmo.edu.mx/~jjap/, consultado en septiembre de 2012.

[17] O. S. Nieva, Blog de capacitación, http://blogcapacitacion.wordpress.com/category/algoritmos/, consultado en septiembre de 2012.

Dirección de Contacto del Autor/es:

J. Jesús Arellano Pimentel
Ingeniería en Computación
Universidad del Istmo
Tehuantepec, Oaxaca
México
jjap@sandunga.unistmo.edu.mx
http://www.unistmo.edu.mx/~jjap

Omar S. Nieva García
Ingeniería en Computación
Universidad del Istmo
Tehuantepec, Oaxaca
México
omarmg@bianni.unistmo.edu.mx

Rocío Solar González
División de Estudios de Posgrado
Universidad del Istmo
Tehuantepec, Oaxaca
México
solgr@sandunga.unistmo.edu.mx

Getsemaní Arista López
Ingeniería en Computación
Universidad del Istmo
Tehuantepec, Oaxaca
México
arloget@gmail.com

M. en C. J. Jesús Arellano Pimentel es Profesor Investigador de Tiempo Completo en la Universidad del Istmo, Campus Tehuatepec. Es miembro del Cuerpo Académico de Ing. en Computación de la UNISTMO.

M. en C. Omar S. Nieva García es Profesor Investigador de Tiempo Completo en la Universidad del Istmo, Campus Tehuatepec. Es miembro del Cuerpo Académico de Ing. en Computación de la UNISTMO.

Dra. Rocío Solar González es Profesor Investigador de Tiempo Completo en la Universidad del Istmo, Campus Tehuatepec. Actualmente esta adscrita a la División de Estudios de Posgrado de la UNISTMO.

Ing. Getsemaní Arista López es Ingeniero en Computación por la Universidad del Istmo. Acutalmente se desempeña como Encargada del Centro de Autoacceso de la UNISTMO.