UNIVERSIDAD ICESI

Departamento de TIC
Algoritmos y Estructuras de Datos
Unidad 1 Laboratorio

Objetivos

Unidad 1: Análisis de Algoritmos

OE1.1. Calcular la complejidad temporal de algoritmos iterativos.
OE1.2. Calcular la complejidad espacial de algoritmos iterativos.
OE1.3. Caracterizar la entrada de un algoritmo iterativo con el fin de calcular la complejidad para el mejor y peor caso.
OE1.4. Analizar algoritmos independiente de una implementación concreta (no dependiente del lenguaje de programación).
OE1.5. Utilizar notación asintótica para describir la complejidad de algoritmos.
OE1.6. Evaluar varios algoritmos que resuelven el mismo problema en términos de sus complejidades computacionales.
OE1.7. Comprender la importancia del Modelo RAM en el proceso de análisis de algoritmos.

Enunciado

ORACLE

Oracle Corporation es una compañía especializada en el desarrollo de soluciones de nube y locales. Es la empresa encargada de desarrollar y dar soporte al lenguaje de programación utilizado en los cursos: Java. Oracle tiene su sede en la localidad californiana de Redwood City, Estados Unidos. Según la clasificación correspondiente al año 2006, ocupa el primer lugar en la categoría de las bases de datos y el séptimo lugar a nivel mundial de las compañías de tecnologías de la información.

En septiembre de 2018 la empresa lanzó al mercado su nuevo desarrollo, el Java 11. La primera versión de Java con un JDK denominado LTS o Long Term Support. Esto significa que Oracle garantiza que te dará soporte y actualizaciones para la versión durante 3 años, en lugar de tan solo 6 meses. Eso sí, a cambio de un "módico" precio (¿sabes cómo se lee "Oracle" al revés? Pues eso...)

Actualmente los ingenieros de Oracle se encuentran laborando fuertemente en su nuevo release de Java. El cual será el Java 12 y que está planeado salir al público en Marzo de 2019. Ellos se encuentran trabajando en muchas “features” en cuanto a los servicios que presta Java.

En esta etapa de desarrollo del proceso y después de estudiar junto con el equipo de mercadeo decidieron lanzar una nueva funcionalidad dentro de Java. Esta no es más que la de encontrar las raíces de un polinomio. Esta decisión fue tomada gracias a la recolección de datos y encuestas de los IDE’s que muestra que los desarrolladores necesitan este tipo de funcionalidades ya que pueden ser de gran ayuda para estudiantes de ingeniería.

Por eso Oracle ha decidido contratar a tu equipo para que sea el encargado de hacer este desarrollo. Ustedes deben, entender, seleccionar e implementar como mínimo 2 algoritmos para encontrar raíces de polinomios. Oracle espera un informe en donde se argumente claramente cuál fue el proceso de selección que tuvieron para elegir esos algoritmos de búsqueda de raíces.

El programa esperado por la empresa debe implementar:
(1) Una interfaz gráfica de usuario que le permita ingresar el polinomio que desea resolver.
(2) Una interfaz gŕafica que le permita generar aleatoriamente polinomios hasta de grado 10, los cuales puedan servir de entrada a los algoritmos que ustedes proponen.
(3) Una interfaz gŕafica que le permita ver al usuario las raíces del polinomio que fue entregado como entrada y qué método fue utilizado para encontrarlas.

Usted debe utilizar el método de la ingeniería para resolver este problema y dejar evidencia en su informe de los resultados de cada fase. Por ejemplo, en la fase 1 deben identificar claramente el problema, justificarlo y especificar los requerimientos funcionales. Recuerde revisar el Resumen del Método de la Ingeniería y el ejemplo del Método de la Ingeniería aplicado a un problema.

Entregables.
1. Análisis de complejidad temporal de cada uno de sus algoritmos
2. Análisis de complejidad espacial de cada uno de sus algoritmos
3. Especificación de Requerimientos y Diseño.
4. Implementación del programa con todo los requerimientos en su lenguaje de programación favorito.
5. Entrega del documento del método de ingeniería.

El laboratorio debe ser desarrollado en grupos de máximo 3 estudiantes. Tenga en cuenta la rúbrica de evaluación para esta actividad: Rúbrica del Laboratorio 1.

Nota: Usted debe entregar un archivo comprimido en formato zip de un directorio con únicamente 2 archivos: 1 archivo de informe en formato pdf con toda la documentación (de cada una de las fases del método y el análisis) y otro archivo comprimido de un directorio con los archivos de codificación en sus respectivos paquetes.

El nombre del archivo comprimido debe tener el formato: PRIMERAPELLIDOEST1_PRIMERAPELLIDOEST2_PRIMERAPELLIDOEST3.zip (tenga en cuenta que el separador entre cada apellido es un guion al piso).

El sistema debe permitir:

r1. Ingresar por medio de una interfaz grafica el polinomio
r2
.
.
.
.
rnf. Se debe resolver el problema eficientemente

MÉTODO DE LA INGENIERÍA

F1. Identificación del problema

la empresa Oracle desea lanzar una nueva versión en la cual se incluye algoritmos de resolución de polinomios de hasta grado 10, se debe encontrar dos algoritmos que permitan cumplir este objetivo.

Identificación de necesidades:

- R1. Se requiere una nueva funcionalidad dentro de java que permita la encontrar las raíces de un polinomio dado
- R2. Se debe poder ingresar como entrada polinomios de hasta grado 10.
- R3. El programa requiere una interfaz que permita ingresar el polinomio y verificar las raíces del polinomio
- R4. Se necesita seleccionar el algoritmo a usar para resolver el polinomio

F2. Recopilación de la información necesaria

Con el objetivo de tener total claridad en los conceptos involucrados de manera más estrecha con el problema encontrado, hacemos una búsqueda de los términos matemáticos para mayor entendimiento del problema.

Definiciones:

- **POLINOMIO:** En matemáticas, un polinomio y constantes (números fijos llamados coeficientes), o bien una sola variable. Las variables pueden tener exponentes de valores definidos naturales incluido el cero y cuyo valor máximo se conocerá como grado del polinomio. En términos más simples, un polinomio se toma como una suma de monomios, pero un monomio también se toma como un polinomio

- **FUNCIÓN:** En análisis matemático, el concepto general de **función**, **aplicación** o **mapeo** se refiere a una regla que asigna a cada elemento de un primer conjunto un único elemento de un segundo conjunto. Las funciones son relaciones entre los elementos de dos conjuntos. Por ejemplo, cada número entero posee un único cuadrado, que resulta ser un número natural (incluyendo el cero)

- **RAÍZ DE UN POLINOMIO:** Las **raíces de un polinomio** (también llamadas ceros de un polinomio) son los valores para los cuales, el **valor numérico del polinomio es igual a cero**.

Los ceros complejos del polinomio permiten determinar de forma muy rápida las características de radiación de la agrupación.

El polinomio de la agrupación uniforme se puede escribir de forma simplificada teniendo en cuenta que el polinomio es una serie geométrica de razón z.

$$p(z) = 1 + z + z^2 + z^3 + \ldots.$$

$$p(z) = \sum_{n=0}^{N-1} z^n = \frac{z^{N-1} z - 1}{z - 1} = \frac{z^N - 1}{z - 1}$$

Los ceros del polinomio son los ceros del numerador, que son las raíces complejas N-ésimas de la unidad. Se exceptúa z=1 , que también se encuentra en el denominador.

Todas las raíces se encuentran en el plano complejo z sobre un círculo centrado en el origen de radio 1, que denominaremos círculo unidad.

Las agrupaciones triangulares se pueden escribir como el cuadrado de distribuciones uniformes, por lo que los ceros serán de orden 2, y corresponderá a las raíces de la distribución uniforme de la que deriven.

$$p(z) = \left(\sum_{n=0}^{\frac{N-1}{2}} z^n \right)^2 = \left(\frac{z^{\frac{N+1}{2}} - 1}{z - 1} \right)^2$$

Las distribuciones binómicas tienen un cero de multiplicidad N situado en z=-1.

$$p(z) = (1 + z)^{N-1}$$

F3. Búsqueda de soluciones creativas

- Escoger 2 algoritmos que hallen las raíces de polinomios hasta de grado 10 de la manera más eficaz, de manera que cada vez se ingrese un polinomio de un grado mayor el algoritmo funcione de manera más rápida y con el que solucionaremos las siguientes preguntas:

  ¿debemos hallar los ceros racionales?

seleccionar un algoritmo que encuentre todos los ceros del polinomio de una variedad de opciones:

Método de Newton-Raphson:

El método de Newton-Raphson es un método abierto, en el sentido de que no está garantizada su convergencia global. La única manera de alcanzar la convergencia es seleccionar un valor inicial lo suficientemente cercano a la raíz buscada. Así, se ha de comenzar la iteración con un valor razonablemente cercano al cero (denominado punto de arranque o valor supuesto). La relativa cercanía del punto inicial a la raíz depende mucho de la naturaleza de la propia función; si ésta presenta múltiples puntos de inflexión o pendientes grandes en el entorno de la raíz, entonces las probabilidades de que el algoritmo sea divergente aumentan, lo cual exige seleccionar un valor supuesto cercano a la raíz. Una vez que se ha hecho esto, el método linealiza la función por la recta tangente en ese valor supuesto. La abscisa en el origen de dicha recta será, según el método, una mejor aproximación de la raíz que el valor anterior. Se realizarán sucesivas iteraciones hasta que el método haya convergido lo suficiente.

Método de Bairstow: Fuente: https://conzmr.wordpress.com/2017/03/06/metodo-de-bairstow/

El método de Lin Bairstow es un método iterativo que dado un polinomio se encuentran dos factores fn-2(x) y un cuadrático. Se busca un polinomio cuadrático f(x) = x^2 – rx – s que sea factor del polinomio en cuestión. Se basa en el método de Müller y de Newton-Raphson. Es un método eficaz que regresa o que busca regresar todas las raíces reales e imaginarias.

Método Weierstrass Fuente: http://homepages.math.uic.edu/~jan/mcs471f03/Project_Two/proj2/node2.html

El método Weierstrass aproxima simultáneamente todas las raíces de un polinomio ***p*** de grado ***d***, asumiendo el coeficiente con el término de mayor grado X^d de ***p*** igual a uno.Empezando en una aproximación inicial para todas las raíces en $x^{(0)} = \left(x_1{}^{(0)}, x_2{}^{(0)}, \ldots, x_d{}^{(0)} \right)$, el método usa la siguiente fórmula:

$$x_i{}^{(k+1)} = x_i{}^k - \frac{p\left(x_i{}^k\right)}{\prod_{j=1,\, j\neq i}^{d} \left(x_i{}^k - x_j{}^k\right)}, \quad i = 1, 2, \ldots, d, \ \wedge k = 0, 1, \ldots$$

F4. Transición de la formulación de ideas a los diseños preliminares

Método de Newton-Raphson:

- Mayor velocidad de convergencia respecto de otros métodos.
- Es un método abierto en el que no está garantizada su convergencia global.
- Proporciona una buena precisión en los resultados.
- Eficiente para la solución de sistemas de ecuaciones no lineales

Método de Bairstow

- Permite calcular todas las raíces de un polinomio (real o imaginaria).
- Permite calcular las raíces de manera más exacta.
- Se puede implementar en lenguaje computacional

Método de Weierstrass

- En el peor de los casos hace hasta 30 iteraciones.
- Se debe asumir el coeficiente con el término de mayor grado es igual a uno

F5. Evaluación y selección de la mejor solución

CRITERIOS

- ➔ **Criterio A. Facilidad en la implementación algorítmica**
  - ◆ (2) Compatible con operaciones aritméticas básicas de un equipo de computo moderno
  - ◆ (1) Incompatible con operaciones aritméticas básicas de un equipo de computo moderno
- ➔ **Criterio B. Precisión de la solución**
  - ◆ (2) Exacta
  - ◆ (1) Aproximada
- ➔ **Criterio C. Eficiencia**
  - ◆ (4) Constante
  - ◆ (3) Mayor a constante
  - ◆ (2) Logarítmica
  - ◆ (1) Lineal

	Criterio A	**Criterio B**	**Criterio C**	**Total**
Newton-Raphson	2	2	3	7
Bairstow	2	2	3	7
Weierstrass	2	2	1	5

Selección

Entonces descartamos el **método de Weierstrass** ya que según nuestros criterios de evaluación no alcanza el puntaje que otros métodos sí. Por lo tanto, nuestros métodos son el de **Newton-Raphson** y el de **Bairstow**

F6. Preparación de informes y especificaciones

Especificación del problema:

Problema: raíces de un polinomio hasta de grado 10

Entrada: coeficientes de una función $f(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n+1}$, siendo a los coeficientes.

Salida: todas las x que son soluciones (real o compleja) de la función de entrada $f(x)$.

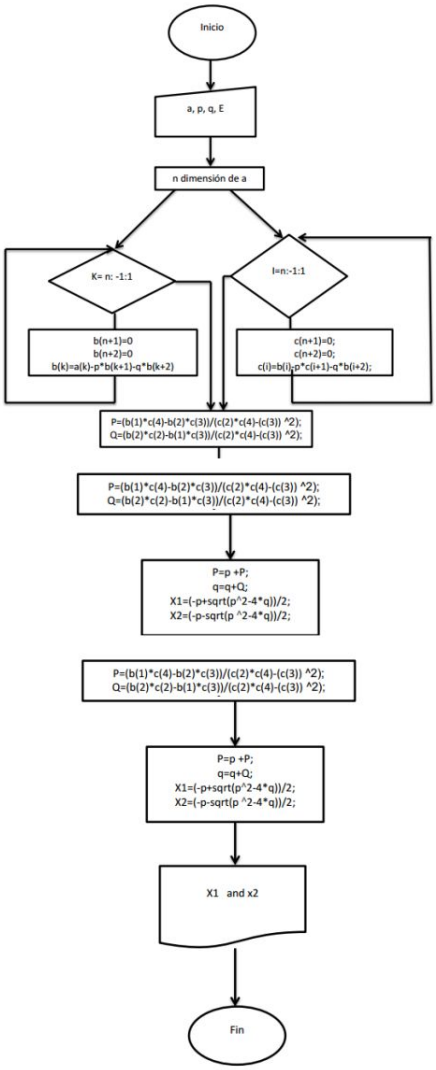
Consideraciones:

Se debe tener en cuenta que para resolver un polinomio pueden haber soluciones reales o complejas y la cantidad de soluciones o raíces es igual o menor a la del grado del polinomio.

Pseudocódigo del método Bairstow

```
a=str2num(get(handles.edit1,'string'));
p=str2double(get(handles.edit2,'string'));
q=str2double(get(handles.edit3,'string'));
E=str2double(get(handles.edit4,'string'));
n=length(a);
for k=n:-1:1
    b(n+1)=0;
    b(n+2)=0;
    b(k)=a(k)-p*b(k+1)-q*b(k+2);
end
for i=n:-1:1
    c(n+1)=0;
    c(n+2)=0;
    c(i)=b(i)-p*c(i+1)-q*b(i+2);
end
P=(b(1)*c(4)-b(2)*c(3))/(c(2)*c(4)-(c(3))^2);
Q=(b(2)*c(2)-b(1)*c(3))/(c(2)*c(4)-(c(3))^2);
while P>E & Q>E
    p=p+P;
    q=q+Q;
    for k=n:-1:1
        b(k)=a(k)-p*b(k+1)-q*b(k+2);
        b(n+1)=0;
        b(n+2)=0;
    end
    for i=n:-1:1
        c(i)=b(i)-p*c(i+1)-q*b(i+2);
        c(n+1)=0;
        c(n+2)=0;
    end
    P=(b(1)*c(4)-b(2)*c(3))/(c(2)*c(4)-(c(3))^2);
    Q=(b(2)*c(2)-b(1)*c(3))/(c(2)*c(4)-(c(3))^2);
end
p=p+P;
q=q+Q;
x1=(-p+sqrt(p^2-4*q))/2;
x2=(-p-sqrt(p^2-4*q))/2;
set(handles.edit5,'string',x1);
set(handles.edit6,'string',x2);
```

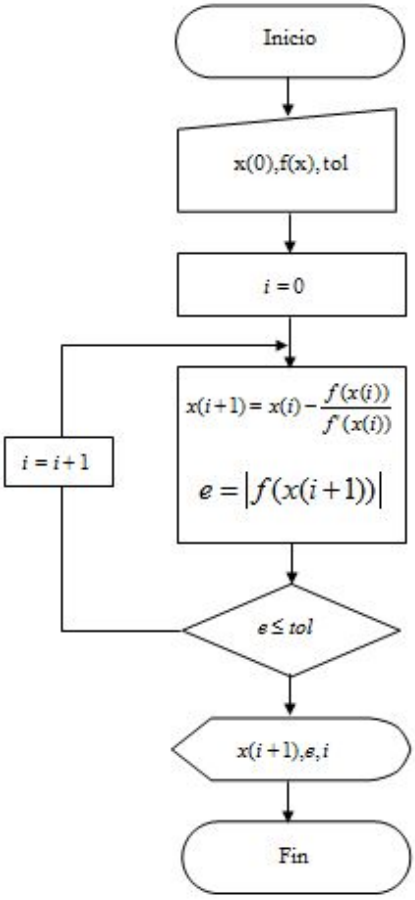
Diagrama de flujo del método Bairstow



Pseudocódigo del método Newton-Raphson

```
NewtonRaphson[x0_, max_] :=
 Module[{},
  k = 0;
  p0 = N[x0];
  Print["p0 = ", PaddedForm[p0, {16, 16}], ",   f[p0] = ", NumberForm[f[p0], 16]];
  p1 = p0;
  While[k < max,
   p0 = p1;
   p1 = p0 - f[p0]/f'[p0];
   k = k + 1;
   Print["p", k, " = ", PaddedForm[p1, {16, 16}], ",   f[", "p", k, "] = ", NumberForm[f[p1], 16]];];
  Print["  p   = ", NumberForm[p1, 16]];
  Print[" Ap   = ±", Abs[p1 - p0]];
  Print["f[p] = ", NumberForm[f[p1], 16]];]
```

Diagrama de flujo del método Newton-Raphson



F7. Implementación del diseño

newton method----------

a como un arreglo de enteros, los cuales son los coeficientes de la función
x como un entero que es el numero de raices a encontrar

```
public static String methodNewtonRaphson(int[] a,double x) {
                String value="";                                                        1
                double h = func(a,x) / derivFunc(a,x);                                  2n
                while (Math.abs(h) >= EPSILON){                                         n

                        x = x - h;                                                      n-1
                }
                value+="The value of the" + " root is : " + Math.round(x * 100.0) / 100.0+"\n";   1
                return value;                                                           1
        }
```

O(n) no contiene ciclos anidados lo que permite mantener una complejidad lineal