PRÁCTICA 3. Análisis de algoritmos. Ejemplos.

Evaluación de polinomios.

Problema de la búsqueda estática.

Problema de la subsecuencia máxima.

¿Qué es el análisis de algoritmos?

Un algoritmo es un conjunto de instrucciones claramente especificadas que el ordenador debe seguir para resolver un problema. Una vez que se da un algoritmo para resolver un problema y se ha probado que es correcto, el siguiente paso es determinar la cantidad de recursos, tales como tiempo y espacio que el algoritmo necesitará para su aplicación. Este paso se llama análisis de algoritmos.

El tiempo necesario para ejecutar un algoritmo depende casi siempre de la cantidad de datos que el mismo debe procesar. El tiempo de ejecución es, por tanto función del tamaño de la entrada. El valor exacto de esta función depende de muchos factores, tales como la velocidad del procesador, la calidad del compilador, y en algunos casos de la calidad del programa. Las curvas en función del tiempo típicas en el análisis de algoritmos son: lineal, nlogaritmica, cuadrática y cúbica.

Se utiliza la notación O para representar el índice de crecimiento. Esta notación nos permite establecer un orden relativo entre las funciones, comparando los términos dominantes. Por ejemplo, el tiempo de ejecución de un algoritmo cuadrático se describe como $O(N^2)$, del orden de N al cuadrado

Problema de la Subsecuencia Máxima

> Dada la secuencia de enteros (posiblemente negativos) $A_1,A_2,...A_N$, encontrar (e identificar la subsecuencia correspondiente) el valor máximo de $\Sigma_{k=i}^{j} A_k$. Cuando todos los enteros son negativos entenderemos que la subsecuencia de suma máxima es la vacía, siendo su suma cero.

Hay muchos algoritmos radicalmente diferentes (en términos de eficiencia) que pueden ser utilizados para resolver el problema de la subsecuencia máxima.

1. El algoritmo obvio de fuerza bruta, de orden $O(N^3)$.
2. El algoritmo mejorado de $O(N^2)$, que elimina el bucle interno, ya que $\Sigma_{k=i}^{j} A_k = A_j + \Sigma_{k=i}^{j-1} A_k$
3. El algoritmo lineal, de orden $O(N)$, que utiliza una hábil observación para saltarse un gran número de subsecuencias.

Problema de la Búsqueda Estática

Dado un entero X y un vector A, devolver la posición de X en A o una indicación de si no esta presente. Si X aparece más de una vez, devolver cualquier aparición. El vector A nunca será modificado. Cuando el vector no esta ordenado se utiliza el algoritmo de la búsqueda secuencial, esta es de orden lineal O(N). Si el vector esta ordenado, se utiliza el algoritmo de la búsqueda binaria, esta es de orden logarítmica O(logN).

Recursos Adicionales:

Junto con está practica se incluye una clase que el alumno debe incorporar al paquete com.mp.utilidades.

- NumerosAleatorios.java

Se recomienda también estudiar la API de:

- La clase System:
  1. El método currentTimeMillis( ) para obtener el tiempo en milisegundos.
  2. El método nanoTime() para obtener el tiempo en nanosegundos (Nuevo en JDK1.5)
- La clase Random (en java.util) para la obtención de números aleatorios, (Usada en NumerosAleatorios.java).
- La clase DecimalFormat (en java.text) para formatear la salida de números.
- El método printf( ) para formatear la salida, similar al printf del lenguaje C. (Nuevo en JDK1.5).
- La clase SringBuffer (en java.lang) para el ejercicio cuarto. Sus métodos append( ) y toString ().

Ejercicios a realizar:

1) Diseñe e implemente un programa para evaluar un polinomio P(X) de grado N, por ejemplo para N=31:

$$A_{31}X^{31}+A_{30}X^{30}+\ldots+A_0X^0.$$

Donde $A_{31...}$ A_0 son los coeficientes del polinomio.

En este caso se tomará el grado del polinomio N como el tamaño de la entrada.

Para el diseño de este ejercicio utilice dos clases:

Polinomio con las propiedades que representan el polinomio: grado de tipo int, X de tipo double y coeficientes de tipo array doubles.

SolucionesPolinomio con los métodos estáticos que resuelven los distintos algoritmos

Utilice dos algoritmos para resolver la potencia:

(a) $O(n^2)$, calculando la potencia directamente

(método double polinomioCuadratico (Polinomio polinomio))

(b) O(n), esta última almacenando las potencias de X ya calculadas

(método double polinomioLineal(Polinomio polinomio))

Utilice el algoritmo de Horner, ver:

http://es.wikipedia.org/wiki/Algoritmo_de_Horner

Obtenga los datos empíricos de tiempo de ejecución en pantalla mediante un programa principal, para distintos tamaños de la entrada.

Nota: nombre paquete **com.mp.practica3.ejercicio1**

Nota: Para las pruebas de unidad, el paquete se situará en: **com.mp.practica3.ejercicio1.test**

2) Implemente tres de los algoritmos que resuelven el problema de la obtención de la subsecuencia máxima, para obtener los datos empíricos de la siguiente tabla (tiempos de ejecución observados en milisegundos). Pruebe los tres algoritmos obteniendo los tiempos de ejecución observados de los diferentes algoritmos para el cálculo de la subsecuencia máxima, construyendo la siguiente tabla:

N	*O*(N^3)	*O*(N^2)	*O*(N)
10	…	…	…
100	…	…	…
1.000	…	…	…
....			

Para el diseño de este ejercicio utilice dos clases:

Subsecuencia con las propiedades que representan la subsecuencia: Número de elementos de la subsecuencia, rango de los valores que pueden tener los enteros, un array de enteros con los valores de la subsecuencia. Utilice números aleatorios para generar los valores.

SolucionesSubsecuencia con los métodos estáticos que resuelven los distintos algoritmos

Nota: nombre paquete **com.mp.practica3.ejercicio2**

Nota: Para las pruebas de unidad, el paquete se situará en : **com.mp.practica3.ejercicio2.test**

3) Implemente el algoritmo de la búsqueda lineal y el algoritmo iterativo de la búsqueda binaria sobre un vector ordenado.

Obtenga los datos empíricos sólo para la búsqueda binaria mediante un programa principal siguiendo la tabla. Los tiempos de ejecución observados preséntelos en nanosegundos.

N	**Tiempo (ns)**	**T/N**	**T/ (N^2)**	**T/(logN)**
10000	…	…	…	…
20000	…	…	…	…
40000	…	…	…	…
....				

Para el diseño de este ejercicio utilice dos clases:

Busqueda con las propiedades que representen la búsqueda.

SolucionesBusqueda con los métodos estáticos que resuelven los distintos algoritmos

Nota: nombre paquete **com.mp.practica3.ejercicio3**

Nota: Para las pruebas de unidad, el paquete se situará en: **com.mp.practica3.ejercicio3.test**

4) Realice un programa para probar la eficiencia de la concatenación de cadenas utilizando las clases String y StringBuffer. Junto con la práctica se ha añadido un archivo CosteString.java que debe terminar de implementar. El programa debe leer 10, 100, 1000 … veces una línea de caracteres y almacenarla en memoria utilizando en un caso un String y en otro un StringBuffer. Este ejercicio es una aproximación a la lectura de las líneas de un archivo de texto.

   Evalue el tiempo utilizando un String para concatenar las líneas.

   Evalue el tiempo utilizando un StringBuffer para concatenar las líneas.

      Construya como salida una tabla como:

N	**String**	**StringBuffer**
10	…	…
100	…	…
1.000	…	…
....		

Nota: nombre paquete **com.mp.practica3.ejercicio4**