Prácticas de Algorítmica.
3º de Grado en Ingeniería Informática.
Curso 2020-2021.

Práctica 1

Objetivos.

Con esta práctica se pretende que el alumno se familiarice con el cálculo de tiempos de ejecución de un determinado algoritmo en función del tamaño del ejemplar y hacer una estimación empírica de esos tiempos en función de dicho tamaño. Para ello, el alumno deberá implementar un programa en C++ donde se calculen los tiempos de ejecución de varios algoritmos para distintos tamaños del ejemplar y posteriormente se estime, utilizando un enfoque híbrido, la complejidad temporal de esos algoritmos, obteniendo la función de tiempo en función del tamaño del ejemplar. Esta función nos permitiría hacer estimaciones de tiempos de ejecución del algoritmo para cualquier tamaño del ejemplar.

El programa resultante tendrá un menú con tres apartados. Cada apartado se invocará en la correspondiente opción del menú mediante una función de medio nivel que no tendrá ningún parámetro. De esta forma, cada opción se podría tratar como un módulo independiente que podría ser exportado a cualquier otro programa. Los prototipos de dichas funciones serán:
void ordenacionHeapSort();
void determinanteIterativo();
void determinanteRecursivo();
El programa principal tendrá un menú que contendrá solo las llamadas a estas funciones.

Apartado 1 (3.5 puntos)

1. *Implementación el método de ordenación del montículo (heapsort). Para ello se tendrán en cuenta las siguientes consideraciones:*
    1. *El vector será de elementos de tipo entero y se rellenará aleatoriamente con valores entre 0 y 9999999. Para ello usad la función* ***void rellenarVector(vector<int> &v).***
    2. *La función de ordenación tendrá el prototipo* ***bool heapSort(vector <int> &v);*** *y devolverá true si la ordenación es correcta y false si no lo es.*
    3. *Para comprobar que está ordenado implementad una función que lo compruebe* ***(bool estaOrdenado(const vector <int> &v);).***
    4. *Para realizar las pruebas de este apartado, el usuario introducirá el valor mínimo del número de elementos del vector, el valor máximo, el incremento del valor del número de elementos y el número de veces que se repetirá la ordenación para cada valor del número de elementos. Por ejemplo, si el mínimo es 1000, el máximo es 5000, el incremento es 100 y el número de repeticiones es 5, se probará primero generando un vector de 1000 elementos, repitiendo el experimento 5 veces (un vector nuevo cada vez), y se calculará la media de tiempos de esas 5 pruebas y ese será el valor correspondiente de tiempo empleado para ese n=1000. Después con 1100 y se repite 5 veces y así hasta llegar a 5000. Esto se hará para todos los posibles valores de n. En la documentación se adjunta un ejemplo para ver como se calculan los tiempos de ejecución. Los tiempos se almacenarán en un vector de la STL de tipo double. La función que obtiene los tiempos de las distintas pruebas tendrá el siguiente prototipo:* ***void tiemposOrdenacionHeapSort(int nMin, int nMax, int repeticiones, vector <double> &tiemposReales, vector <double> &numeroElementos);****. Dentro de esta función se irán creando los vectores de*

distinto tamaño, se invocará a la función de ordenación y se irán almacenando los tiempos obtenidos y los distintos valores de n en los vectores que se pasan como parámetros. (Nota: Aunque los valores de n son enteros, como se van a usar para calcular los sumatorios y otra serie de operciones mixtas con números doubles, se almacenarán como doubles para no tener problemas)

5. *Almacenad en un fichero de texto los valores de n empleados (primera columna) y los tiempos reales correspondientes a esos valores de n (segunda columna).*
6. *Se probará ajustar una curva del tipo.* ***t(n) =a0 + a1*n*log(n).*** *Para ello se usará una función de prototipo* ***void ajusteNlogN(const vector <double> &n, const vector <double> &tiemposReales, vector <double> &a).*** *Donde a será un vector de coeficientes de la curva de ajuste.**.*** *Este ajuste se puede obtener usando el ajuste polinómico de grado 1 por medio del cambio de variable z = nlog(n) (Ver ejemplo en el anexo). El código para resolver el sistema de ecuaciones que proporciona los coeficientes del ajuste polinómico se suministra con el material de la práctica. Lo único que hay que hacer es pasar a las funciones suministradas las matrices del sistema de ecuaciones. Los elementos de dichas matrices se obtendrán calculando los sumatorios correspondientes. Para calcular los sumatorios de cualquier tipo implementad una función general con este prototipo* ***double sumatorio(vector <double> &n, vector <double> &t, int expN, int expT);*** *donde n y t son las variables y expN y expT son los exponentes respectivos de las variables en el sumatorio. Cuando solo intervenga una variable en el sumatorio, el exponente de la otra será 0.*
7. *Ahora, teniendo en cuenta la función ajustada en el paso anterior, se obtendrán los tiempos estimados mediante dicha función. Para ello usar la función de prototipo* ***void calcularTiemposEstimadosNlogN(const vector <double> &n, const vector <double> &a, vector <double> &tiemposEstimados).*** *Donde a será un vector de coeficientes de la curva de ajuste. De esta forma, al final de este paso, tendremos los tiempos reales de los algoritmos, que son los tiempos obtenidos en el apartado 4, y los tiempos estimados mediante los ajustes por mínimos cuadrados.*
8. *Obtener el coeficiente de determinación del ajuste, sabiendo que es la varianza de los tiempos estimados dividida por la varianza de los tiempos reales. Para ello usar la función de prototipo* ***double calcularCoeficienteDeterminacion(const vector <double> &tiemposReales, const vector <double> &tiemposEstimados).***
9. *Una vez obtenidos los tiempos estimados se guardarán en un fichero de texto para poder representarlos posteriormente usando el programa* ***gnuplot*** *(se suministra un ejemplo de uso)**.*** *La información se guardará por columnas en el siguiente orden: tamaño del ejemplar, tiempo real y tiempo estimado.*
10. *Para finalizar, el programa ha de mostrar la ecuación de las curva ajustadas por mínimos cuadrados, su coeficiente de determinación y dar la posibilidad al usuario si quiere hacer una estimación de tiempos para un determinado valor del tamaño del ejemplar, en cuyo caso mostrará el tiempo de esa estimación en días. Esta opción ha de poder repetirse hasta que el usuario introduzca un tamaño de ejemplar igual a 0. Esto es útil cuando el tiempo es muy elevado para un tamaño de ejemplar relativamente grande. Por ejemplo, la ordenación de un vector de 100000 millones de elementos tardaría en calcularse varios días, y mediante la curva ajustada podemos obtener de una forma muy fiable el tiempo que tardaría en calcularse. Para ello usar la función* ***double calcularTiempoEstimadoNlogN(const double &n, vector <double> &a).*** *Donde a será un vector de coeficientes del polinomio de ajuste.*

Apartado 2 (3.5 puntos)

2. *Implementar el cálculo del determinante de una matriz de tipo double por el método de triangularización, teniendo en cuenta las siguientes consideraciones:*
    1. *La matriz será de elementos de tipo double y habrá dos opciones: se podrán pedir los elementos para que los introduzca el usuario o se rellenerá aleatoriamente con valores entre 0.95 y 1.05.*
    2. *Realizar las mismas pruebas que en el caso del apartado 1, teniendo en cuenta las siguientes diferencias:*
        1. *Para un valor de n solo se realiza una prueba.*
        2. *Se ajustará un polinomio de grado 3. La función del ajuste será* ***void ajustePolinomico(const vector <double> &n, const vector <double> &tiemposReales, vector <double> &a).*** *Donde a será un vector de coeficientes del polinomio de ajuste.*
        3. *La función para calcular los tiempos estimados será* ***void calcularTiemposEstimadosPolinomico(const vector <double> &n, const vector <double> &tiemposReales, const vector <double> &a, vector <double> &tiemposEstimados).***
        4. ***La función para estimar el tiempo a partir de un valor de n será double calcularTiempoEstimadoPolinómico(const double &n, const vector <double> &a)***

Apartado 2 (3.0 puntos)

3. *Implementar el cálculo del determinante de una matriz de tipo double usando un algoritmo recursivo. El algoritmo recursivo de un determinante de orden n, obtiene el determinante a partir de n determinantes de orden (n-1), como ya habréis estudiado en matemáticas. Lo único que cambia respecto al apartado 2 es la obtención de la curva de ajuste. En este caso se usará la curva* ***t(n) = a0 + a1*n!. Para obtenerla se procederá haciendo un cambio de variable, lo mismo que en el apartado 1. En este caso el cambio será z = n! Y así tendremos un ajuste polinómico de grado 1.***

Anexo

Para estimar los parámetros de un ajuste polinómico de orden m se puede usar el siguiente sistema de acuaciones (http://es.slideshare.net/diegoegas/regresion-polinomial-2512264):

$$
\begin{aligned}
a_0 n &+ a_1 \sum x_i &+ a_2 \sum x_i^2 &+ \dots &+ a_m \sum x_i^m &= \sum y_i \\
a_0 \sum x_i &+ a_1 \sum x_i^2 &+ a_2 \sum x_i^3 &+ \dots &+ a_m \sum x_i^{m+1} &= \sum x_i y_i \\
a_0 \sum x_i^2 &+ a_1 \sum x_i^3 &+ a_2 \sum x_i^4 &+ \dots &+ a_m \sum x_i^{m+2} &= \sum x_i^2 y_i \\
\vdots & & \vdots & & \vdots & \vdots \\
a_0 \sum x_i^m &+ a_1 \sum x_i^{m+1} &+ a_2 \sum x_i^{m+2} &+ \dots &+ a_m \sum x_i^{2m} &= \sum x_i^m y_i
\end{aligned}
$$

- *En este sistema de ecuaciones las incógnitas son los valores de los a_i. Los valores de la variable independiente x_i se corresponden con el tamaño del ejemplar(valores de n) y los valores de la variable dependiente y_i se corresponden con los tiempos reales (valores de t). La n que aparece en el primer término de la primera ecuación del sistema es el tamaño de la muestra de tiempos con los que estamos trabajando.*

Cada sumatorio tendrá tantos sumandos como valores de n se hayan usado para el tamaño del ejemplar. Por ejemplo, si al ordenar se usan valores de n desde 1000 hasta 10000, de 1000 en 1000 (11 valores de n), cada sumatorio tendrá 11 elementos.

- *Se suministra el código fuente para resolver un sistema lineal de ecuaciones, cuyo prototipo es:*

void resolverSistemaEcuaciones(vector < vector < double > > A, vector < vector < double > > B, int n, vector < vector < double > > &X);

donde:
A es la matriz de coeficientes de nxn
B es la matriz de terminos independientes de nx1
n es el orden de las matrices
X es el valor de las variables que se obtienen resolviendo el sistema de orden nx1

Nota: De este código se puede extraer el código para resolver el determinante iterativo.

Declaracion y reserva de matrices usando el tipo vector de la STL

vector < vector < double > > matrizDatos;
matrizDatos = vector< vector< double > >(filas, vector< double >(columnas)); //Matriz de filas x columnas.

Ejemplo práctico 1.

En este ejemplo se va a ajustar un polinomio de grado 2 a una curva que representa tiempos frente a valores de n. Este ejemplo podría ser el de la ordenación de un vector por inserción. En este caso, la curva sería del tipo

$$t(n) = a0 + a1 * n + a2 * n^2.$$

En la imagen se muestran los distintos valores de n y t para 11 observaciones, así como los elementos que componen el sistema de ecuaciones para obtener la curva de ajuste.

	n	t(tiempo real)	n^2	n^3	n^4	n*t	$t*n^2$	t(estimado)
	100	900	10000	1000000	100000000	90000	9000000	892,1688
	110	1080	12100	1331000	146410000	118800	13068000	1086,1228
	120	1310	14400	1728000	207360000	157200	18864000	1299,1368
	130	1525	16900	2197000	285610000	198250	25772500	1531,2108
	140	1785	19600	2744000	384160000	249900	34986000	1782,3448
	150	2030	22500	3375000	506250000	304500	45675000	2052,5388
	160	2360	25600	4096000	655360000	377600	60416000	2341,7928
	170	2635	28900	4913000	835210000	447950	76151500	2650,1068
	180	2995	32400	5832000	1049760000	539100	97038000	2977,4808
	190	3320	36100	6859000	1303210000	630800	119852000	3323,9148
	200	3710	40000	8000000	1600000000	742000	148400000	3689,4088
Sumatorios	1450	19940	218500	34075000	5473330000	3114100	500823000	19936,818
Varianza		874710						868493,111

este caso, el sistema resultante sería:

*11*a0 + 1450*a1 + 218500*a2 = 19940*
*1450*a0 + 218500*a1 + 34075000*a2 = 3114100*
*218500*a0 + 34075000*a21+ 5473330000*a2 = 500823000*

***(Nota:** El coeficiente de a0 en la primera ecuación es 11 porque la muestra tiene 11 elementos.)*

De donde a0 = 0.9288, a1 = -0.6176, a2 = 0.0953.
De esta forma, la ecuación de la curva para obtener los tiempos estimados es:

$$t(n) = 0.9288 - 0.6176 * n + 0.0953 * n^{2.}$$

Esta función permitiría estimar el tiempo para cualquier valor de n.

Los tiempos estimados mediante la curva son los que están en la última columna y se han obtenido usando la ecuación de la curva.
Por último, el ***coeficiente de determinación*** *sería la varianza de los tiempos estimados dividida entre la varianza de los tiempos reales = 868493.111/874710 = 0.992, el cual es un valor bastante bueno (99.2%)*

Ejemplo práctico 2:
En este ejemplo se va a ajustar una curva del tipo nlog(n) a una curva que representa tiempos frente a valores de n. Este ejemplo podría ser el de la ordenación de un vector por el quicksort.
En este caso, la curva sería

$$t(n) = a0 + a1 * n * log(n).$$

Para obtener el ajuste, se puede realizar el cambio de variable z = nlog(n) y tendríamos una situación similar a cuando se ajusta un polinomio de grado 1(ecuación de una recta).
Por tanto, la curva de ajuste quedaría como

$$t(n) = a0 + a1 * z.$$

Veamos como quedaría la tabla de las observaciones.

	n	t(tiempo real)	z=nlog(n)	z^2	z*t	t(estimado)
	100,000	50,500	200,000	40000,000	10100,000	49,813
	110,000	58,411	224,553	50424,138	13116,296	55,288
	120,000	58,400	249,502	62251,123	14570,989	60,852
	130,000	66,463	274,813	75521,985	18264,742	66,496
	140,000	67,592	300,458	90274,965	20308,427	72,215
	150,000	80,783	326,414	106545,896	26368,591	78,003
	160,000	83,032	352,659	124368,509	29281,942	83,856
	170,000	89,335	379,176	143774,679	33873,816	89,769
	180,000	93,690	405,949	164794,632	38033,290	95,740
	190,000	102,093	432,963	187457,119	44202,353	101,764
	200,000	109,541	460,206	211789,562	50411,517	107,839
Sumatorios	1450,000	859,839	3606,693	1257202,607	298531,964	861,636
Varianza		374,830				371,157

El sistema resultante sería

*11*a0 + 3606,693*a1 = 859.84*
3606,693 a0 + 1257202,607*a1 = 298531,964*

(Nota: *El coeficiente de a0 en la primera ecuación es 11 porque la muestra tiene 11 elementos.)*

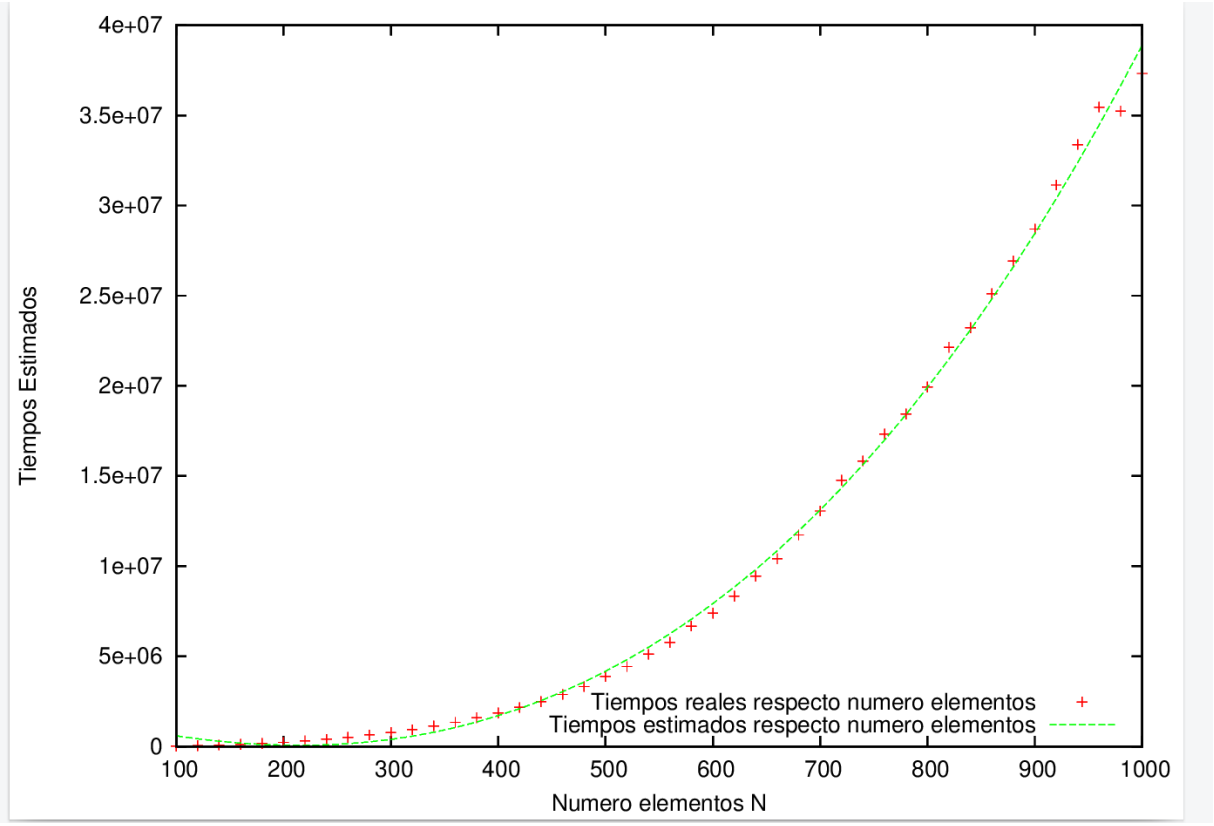
De donde a0 = 5.213 y a1 = 0.223. La ecuación de la curva para obtener los tiempos estimados sería:

$$t(n) = 5.213 + 0.223 * n * log(n).$$

Esta función permitiría estimar el tiempo para cualquier valor de n.
Los tiempos estimados mediante la curva son los que están en la última columna y se han obtenido usando la ecuación de la curva.

*Por último, **el coeficiente de determinación** sería la varianza de los tiempos estimados dividida entre la varianza de los tiempos reales = 371,157/374,830= 0.99, el cual es un valor bastante bueno (99.0%).*

Finalmente se muestra una gráfica en la que se ha ajustado un polinomio de grado 3 para el algoritmo del producto de matrices. Las cruces en rojo son los puntos correspondientes a los valores de n y de los tiempos reales, la curva en verde es la curva obtenida. Como se puede apreciar el ajuste es bastante bueno y con dicha curva se podría estimar por ejemplo que tiempo emplearía el algoritmo del producto de matrices para multiplicar matrices de orden 10000.



Fecha de comienzo: 15 de setiembre de 2020
Fecha de Entrega: 6 de Octubre de 2020.