Análisis de algoritmos
Escuela Politécnica Superior, UAM, 2020–2021

Conjunto de Prácticas no. 1

Fecha de entrega de la práctica

- Grupos del Lunes: 18 de Octubre.
- Grupos del Miércoles: 20 de Octubre.
- Grupos del Viernes: 22 de Octubre.

La entrega de los códigos fuentes y la documentación en formato electrónico correspondiente a las prácticas de la asignatura AA se realizará en la fecha indicada por medio de la página web **https://moodle.uam.es/**.
La documentación impresa se entregará al profesor al comienzo de la siguiente clase de prácticas.

Introducción

Esta primera práctica servirá como asentamiento para futuras prácticas. Se generarán varios ficheros .C **permutaciones.c ordenacion.c tiempos.c** donde se irán guardando a modo de librería los códigos de las distintas funciones C que se van a implementar; igualmente se incluirán en ficheros .H **permutaciones.h ordenacion.h tiempos.h** los prototipos de las funciones implementadas en los ficheros .C y los # **define** genéricos, tales como:

```
#define ERR  -1
#define OK   (!(ERR))
```

Será obligatorio ajustarse al esquema de prácticas que se da, es decir, los prototipos de las funciones no deben modificarse (excepto si lo indica el profesor de prácticas).
Consejos en la implementación de las rutinas:

1. Comprobad que los argumentos de entrada de la rutina son correctos. Independencia de las funciones.
2. Antes de salir de una rutina o del programa liberad toda la memoria y cerrad todos los ficheros que ya no se utilicen.
3. Comprobad que los ficheros se pueden abrir, en caso negativo actuad como en el punto 2.
4. Comprobad en cada caso que al reservar memoria, ésta se ha realizado eficientemente y en el caso de que no se hubiera podido reservar, antes de salir de la rutina o el programa proceded como en el punto 2.
5. Comprobad los programas en ejecuciones extremas para verificar el buen comportamiento del programa (p.e. tablas muy grandes, entradas ilegales, ...).
6. Utilizad entornos de ejecución en modo protegidos (Windows XP, Vista, Windows 7, Windows 8, Windows 10, Unix, Linux, OSX/iOS ... Nunca DOS puro).
7. **Las prácticas serán corregidas en un sistema Unix o bien Linux, con lo cual los programas deberán desarrollarse en ANSI C para que sean portables.**

En la plataforma Moodle está disponible el fichero **practica1.zip** de plantillas C que incluyen los prototipos de las funciones requeridas a lo largo de esta práctica así como programas de prueba para cada apartado.
La práctica comenzará desarrollando en el fichero **permutaciones.c** un conjunto de rutinas C que generen permutaciones aleatorias de un cierto número de elementos. Dichas rutinas se usarán más adelante para obtener las entradas de los algoritmos de ordenación. A continuación se implementará en el fichero **ordenacion.c** algún algoritmo conocido de ordenación y finalmente se implementarán en el fichero **tiempos.c** rutinas que medirán los tiempos de ejecución del algoritmo de ordenación implementado empleando las permutaciones generadas.
Cada práctica se realizará en un período de cuatro a cinco semanas.

Primer Bloque

Generación de permutaciones

1. La rutina C **rand** de la librería **stdlib** genera números aleatorios equiprobables entre 0 y el valor de **RAND_MAX**. Usar esta función para construir una rutina **int aleat_num (int inf, int sup)** que genere un número aleatorio equiprobable entre los enteros **inf, sup**, ambos inclusive. Implementad dicha rutina dentro del fichero **permutaciones.c**.

   Utilizad el fichero **ejercicio1.c** para comprobar el correcto funcionamiento de la rutina que acabáis de implementar.

   Comprobad mediante la elaboración de un histograma si la generación aleatoria es equiprobable para cada dígito.

   Obs: Se valorará el nivel de aleatoriedad de la función implementada, por lo que se recomienda pensar detenidamente sobre su diseño en lugar de implementar la primera idea que se tenga. Como ayuda se recomienda consultar el capítulo 7 (Random Numbers) del libro "Numerical recipes in C: the art of scientific computing" del cual hay varias copias disponibles en la biblioteca.

2. El siguiente pseudocódigo proporciona un método de generación de permutaciones aleatorias

   ```
   para i de 1 a N:
      perm[i] = i;

   para i de 1 a N:
      intercambiar perm[i] con perm[aleat_num(i, N)];
   ```

   donde **aleat_num** es la rutina del ejercicio anterior.

   Implementad en el fichero **permutaciones.c** una rutina C **int *genera_perm(int N)** para dicho algoritmo.

   Utilizad el fichero **ejercicio2.c** para comprobar el correcto funcionamiento de la rutina que acabáis de implementar.

Segundo bloque

3. Inplemenentad en el fichero **permutaciones.c** la rutina **int **genera_permutaciones(int n_perms, int N)** que genera mediante la función **genera_perm** del ejercicio anterior **n_perms** permutaciones equiprobables de **N** elementos cada una.

   Utilizad el fichero **ejercicio3.c** para comprobar el correcto funcionamiento de la rutina que acabáis de implementar.

Algoritmos de ordenación

En esta sección se determinará experimentalmente el tiempo medio de ejecución y el número medio, mejor y peor de veces que se ejecuta la Operación Básica (OB) del algoritmo de ordenación por inserción *InsertSort* sobre tablas de diferentes tamaños, y se comparará con el análisis teórico.

4. Implementad en el fichero **ordenacion.c** una función **int InsertSort(int *tabla, int ip, int iu)** para el método de ordenación por inserción *InsertSort*, esta función devuelve ERR en caso de error o el número de veces que se ha ejecutado la OB en el caso de que la tabla se ordene correctamente, **tabla** es la tabla a ordenar, **ip** es el primer elemento de la tabla e **iu** es el último elemento de la tabla.

   Utilizad el fichero **ejercicio4.c** para comprobar el correcto funcionamiento de la rutina que acabáis de implementar.

Tercer Bloque

Tiempos de ejecución

5. Definir en **tiempos.h** la siguiente estructura que servirá para almacenar los tiempos de ejecución de un algoritmo sobre un conjunto de permutaciones:

```
typedef struct tiempo {
   int N;             // Tamano de la entrada
   int n_elems;       // numero total de elementos que se promedian
   double tiempo;     // tiempo promedio de reloj
   double medio_ob;   // numero promedio de veces que se ejecuta la OB
   int min_ob;        // minimo de ejecuciones de la OB
   int max_ob;        // maximo de ejecuciones de la OB
} TIEMPO, *PTIEMPO;
```

Implementad en el fichero **tiempos.c** la función:

short tiempo_medio_ordenacion(pfunc_ordena metodo,
int n_perms, int N, PTIEMPO ptiempo),

donde **pfunc_ordena** es un puntero a la función de ordenación, definido como:

typedef int (* pfunc_ordena)(int*, int, int);

este typedef deberá incluirse en `ordenacion.h`, **n_perms** representa el número de permutaciones a generar y ordenar por el método que se use (en este caso método de inserción), **N** es el tamaño de cada permutación y **ptiempo** es un puntero a una estructura de tipo **TIEMPO** que a la salida de la función contendrá el número de permutaciones promediadas en el campo **n_elems**, el tamaño de las permutaciones en el campo **N**, el tiempo medio de ejecución (en segundos) en el campo **tiempo**, el número promedio de veces que se ejecutó la OB en el campo **medio_ob**, el número mínimo de veces que se ejecutó la OB **min_ob** y el número máximo de veces que se ejecutó la OB en el campo **max_ob** para el algoritmo de ordenación **ordena_metodo** sobre las permutaciones generadas por la función **genera_permutaciones**.

La rutina **tiempo_medio_ordenacion** devuelve devuelve ERR en caso de error y OK en el caso de que las tablas se ordenen correctamente.

Implementad además la función:

short genera_tiempos_ordenacion(pfunc_ordena metodo, char * fichero,
int num_min,int num_max,int incr, int n_perms)

que escribe en el fichero **fichero** los tiempos medios, y los números promedio, mínimo y máximo de veces que se ejecuta la OB en la ejecución del algoritmo de ordenación **metodo** con **n_perms** permutaciones de tamaños en el rango desde **num_min** hasta **num_max**, ambos incluidos, usando incrementos de tamaño **incr**. La rutina devolverá el valor ERR en caso de error y OK en caso contrario.

genera_tiempos_ordenacion llamará a una función

short guarda_tabla_tiempos(char *fichero, PTIEMPO tiempo, int n_tiempos)

con la que se imprime en un fichero una tabla con cinco columnas correspondientes al tamaño **N**, al tiempo de ejecución **tiempo** y al número promedio bf medio_ob, máximo bf max_ob y mínimo **min_ob** de veces que se ejecuta la OB; el array **tiempo** guarda los tiempos de ejecución y **n_tiempos** es el número de elementos del array **tiempo**.

Utilizad el fichero **ejercicio5.c** para medir los tiempos de ejecución del algoritmo **InsertSort**

Comentario: No sería extraño que al ejecutar vuestro programa obtengáis que el tiempo de ejecución es cero, eso es debido a que la velocidad del procesador es tan alta que no llega ni siquiera a consumir un TIC del reloj medido mediante la función **clock** durante la llamada a **tiempo_medio_ordenacion**. Luego, para ver el tiempo de ejecución será necesario introducir algún mecanismo de retardo o utilizar otra función más precisa para la medición de tiempos.

A modo de sugerencia, la mejor opción para evitar la situación anterior es elegir un número mayor de tablas a promediar o elegir unos tamaños de tabla mayores.

Otra opción es utilizar funciones más precisas que la función de librería estándar **clock** para la medición de tiempos. Por ejemplo en UNIX y LINUX existe la función del sistema **clock_gettime** que da una precisión de nanosegundos (10^{-9}). Esta función no esta definida en ANSI C y no existe en Windows (aunque sí existen llamadas equivalentes) ni OSX.

6. En ocasiones es interesante ordenar una tabla en valores de mayor a menor. Implementar una rutina **int InsertSortInv(int* tabla, int ip, int iu)** con los mismos argumentos y retorno que **InsertSort** pero que ordene la tabla en orden inverso (de mayor a menor), obtener los tiempos de ejecución y comparar los resultados obtenidos con los de la rutina **InsertSort**.

Cuestiones sobre la práctica

1. Justifica tu implementación de **aleat_num** ¿en qué ideas se basa? ¿de qué libro/artículo, si alguno, has tomado la idea? Propón un método alternativo de generación de números aleatorios y justifica sus ventajas/desventajas respecto a tu elección.
2. Justifica lo más formalmente que puedas la corrección (o dicho de otra manera, el porqué ordena bien) del algoritmo **InsertSort**.
3. ¿Por qué el bucle exterior de **InsertSort** no actúa sobre el primer elemento de la tabla?
4. ¿Cuál es la operación básica de **InsertSort**?
5. Dar tiempos de ejecución en función del tamaño de entrada n para el caso peor $W_{BS}(n)$ y el caso mejor $B_{BS}(n)$ de **InsertSort**. Utilizad la notación asintótica (O, Θ, o, Ω,etc) siempre que se pueda.
6. Compara los tiempos obtenidos para **InsertSort** e **InsertSortInv**, justifica las similitudes o diferencias entre ambos (es decir, indicad si las gráficas son iguales o distintas y por qué).

Material a entregar en cada uno de los apartados

Documentación: La documentación constará de los siguientes apartados:

1. **Introducción:** Consiste en una descripción técnica del trabajo que se va a realizar, qué objetivos se pretenden alcanzar, qué datos de entrada requiere vuestro programa y qué datos se obtienen de salida, así como cualquier tipo de comentario sobre la práctica.
2. **Código impreso:** El código de la rutina según el apartado. Como código también va incluida la cabecera de la rutina.
3. **Resultados:** Descripción de los resultados obtenidos, gráficas comparativas de los resultados obtenidos con los teóricos y comentarios sobre los mismos.
4. **Cuestiones:** Respuestas en papel a las cuestiones teóricas.

Esta documentación se le entregará al profesor de prácticas, en la clase correspondiente al día siguiente de entrega de la práctica. En la portada deberá incluirse los nombres de los alumnos. Todos los ficheros necesarios para compilar la práctica y la documentación se guardarán en un único fichero comprimido, en formato zip, o tgz (tgz representa un fichero tar comprimido con gzip). El nombre de dicho fichero será **nombre1_apellido1_nombre2_apellido2.zip** o **nombre1_apellido1_nombre2_apellido2.tgz**. Donde **nombre_apellido** es el nombre y apellido de cada uno de los miembros de la pareja.
Adicionalmente, las prácticas deberán ser guardadas en algún medio de almacenamiento (lápiz usb, CD o DVD, disco duro, disco virtual remoto, etc) por el alumno para el día del examen de prácticas en Enero.

Ojo: Se recalca la importancia de llevar un lápiz usb **además de otros medios de almacenamiento como discos usb, cd, disco virtual remoto, email a dirección propia, etc**, ya que no se garantiza que puedan montarse y accederse todos y cada uno de ellos durante el examen, lo cual supondría la calificación de suspenso en prácticas.

Con el objetivo de normalizar la compilación y ejecución de los diversos programas se sugiere utilizar la herramienta make junto al fichero Makefile incluido en el fichero **practica1.zip**. En este fichero Makefile se implementarán las opciones ejercicio1, ejercicio2, ..., ejercicio5 que compilarán los programas de los diferentes apartados. Igualmente se implementarán las opciones ejercicio1_test, ejercicio2_test,..., ejercicio5_test que ejecutarán los programas. También se pueden emplear entornos de desarrollos tipo Netbeans, Anjuta, Visual Studio, etc.

Instrucciones para la entrega de los códigos de prácticas

La entrega de los códigos fuentes correspondientes a las prácticas de la asignatura AA se realizará por medio de la página web **https://moodle.uam.es/**.