Análisis de algoritmos
Escuela Politécnica Superior, UAM, 2025–2026

Conjunto de Prácticas no. 3

Fecha de entrega de la práctica

- Grupos del Miércoles: 9 de Diciembre.
- Grupos del Jueves: 10 de Diciembre.

En esta práctica se trata el desarrollo y el análisis de algoritmos de búsqueda sobre diccionarios. Se determinará experimentalmente el tiempo medio de búqueda de los elementos en un diccionario que emplea como tipo de datos una tabla.

1 TAD Diccionario

Creación de un TAD diccionario definido en **search.h** por las siguientes estructuras:

```
#define NOT_FOUND -2
#define SORTED 1
#define NOT_SORTED 0

typedef struct dictionary {
 int size; /* size of the table */
 int n_data; /* number of data in the table */
 char order; /* sorted table (SORTED) or unsorted (NOT_SORTED) */
 int *table; /* tabla de datos
} DICT, *PDICT
```

La estructura **DICT** se utilizará para implementar un diccionario empleando una tabla (ordenada o desordenada) como tipo abstracto de datos.
Implementad las siguientes rutinas en el fichero **search.c**:

```
PDICT init_dictionary (int size,char order);
void free_dictionary(PDICT pdict);
int insert_dictionary(PDICT dict,int key);
int massive_insertion_dictionary (PDICT pdict,int *keys,int n_keys);
int search_dictionary(PDICT pdict,int key,int *ppos,pfunc_search method);

int bin\_search(int *table,int F,int L,int key,int *ppos);
int lin\_search(int *table,int F,int L,int key,int *ppos);
int lin\_auto\_search(int *table,int F,int L, int key,int *ppos);
```

donde **pfunc_search** es un puntero a la función de búsqueda, definido como:

```
  typedef int (* pfunc_search)(int*, int,int,int,int*);
```

que corresponde con las declaraciones de funciones de búsqueda **bin_search**, **lin_search** y **lin_auto_search** del fichero search.h.

- La función **init_dictionary** creará un diccionario vacío (sin datos) del tipo indicado por sus argumentos, **size** indicará el tamaño inicial que deseamos darle al diccionario y **order** indicará usando las constantes **SORTED** o **NOT_SORTED** si se emplea una tabla ordenada o desordenada como estructura de datos. La rutina reservará memoria para una tabla con al menos **size** posiciones.

- La función **insert_dictionary** introduce el elemento clave en la posición correcta del diccionario definido por **pdict** en función del valor del campo **order** del diccionario. Si dicho campo toma el valor **NOT_SORTED** la rutina simplemente insertará el elemento al final de la tabla.

  Por contra, si el campo **order** toma el valor **SORTED**, la rutina insertará el elemento en la posición correcta que mantenga la tabla ordenada mediante una sola iteración del método de inserción, es decir, colocará el elemento al final de la tabla y luego situará el valor insertado en el sitio correcto utilizando el siguiente pseudocódigo

  ```
  A=T[U]; j=U-1;
  while (j >= P && T[j]>A);
     T[j+1]=T[j]; j--;
  T[j+1]=A;
  ```

  Aviso: no es necesario realizar una llamada a ningún algoritmo de ordenación, simplemente añadir el código anterior a vuestra rutina **insert_dictionary** para gestionar el caso de mantener la tabla ordenada.

- La función **massive_insertion_dictionary** introduce las **n_keys** claves del que hay en la tabla **keys** en el diccionario mediante la realización de **n_keys** llamadas sucesivas (una por cada clave) a la función **insert_dictionary** .

- La función **search_dictionary** buscará una clave en el diccionario definido por **pdict** usando la rutina indicada por **method**, la función devolverá la posición de la clave en la tabla por medio de la posición apuntada por **ppos**.

- La función **free_dictionary** liberará toda la memoria reservada por las rutinas del TAD diccionario.

- Las rutinas **bin_search**, **lin_search** y **lin_auto_search** implementarán los algoritmos de búsqueda binaria, búsqueda lineal y búsqueda lineal autoorganizada (cuando se encuentra una clave, se intercambia con la posición anterior, excepto si la clave encontrada ya está en la primera posición de la tabla) respectivamente, las funciones devolverán la posición de la clave en la tabla o la constante **NOT_FOUND** en el caso de que la clave no se encuentre en la tabla por medio de la posición apuntada por **ppos**.

Todas las funciones devolverán el número de Operaciones Básicas que han empleado o **ERR**, en función de que su desarrollo fuera el deseado o no, exceptuando **init_dictionary** que devolverá **NULL** si ha ocurrido un error o bien el diccionario construido si todo se ha realizado correctamente.
El programa C de prueba para estas rutinas es **exercise1.c** que está incluido en **code_p3.zip**.
Modificar el programa **exercise1** para que permita comprobar el funcionamiento tanto de la búsqueda binaria como de búsqueda lineal sobre una tabla ordenada.

2 Comparación del rendimiento de búsqueda

En este apartado se deberán implementar una serie de funciones para realizar medidas de rendimiento de las funciones de búsqueda desarrolladas en el apartado anterior. Para ello debe añadir en el fichero **times.c** y la correspondiente declaración en el fichero **times.h** la función:

```
short average_search_time(pfunc_busqueda method, pfunc_key_generator generator,
    char order,
    int N,
    int n_times,
    PTIME_AA ptime);
```

donde:

- **order** indica si se usarán tablas ordenadas en el TAD diccionario.
- **N** indica el tamaño del diccionario, es decir, el número de elementos que contiene.
- **n_times** representa el número de veces que se busca cada una de las N claves que hay en el diccionario.
- **ptime** es un puntero a una estructura de tipo **TIME_AA** que a la salida de la función contendrá:
  - El tamaño del diccionario (i.e el argumento **N**) en el campo **N**.

- El número de claves buscadas (i.e **N*n_times**) en el campo **n_elems**.
- El tiempo medio de ejecución (en segundos) en el campo **time**.
- El número promedio de veces que se ejecutó la OB en el campo **avg_ob**.
- El número mínimo de veces que se ejecutó la OB en el campo **min_ob**.
- El número máximo de veces que se ejecutó la OB en el campo **max_ob**.

.

Además la rutina recibe un puntero a la función de búsqueda (parámetro **method**) y otro puntero a la función que se debe usar para generar las claves que deben ser buscadas (parámetro **generator**) y que ya se encuentra implementada en los ficheros **search.c** y **search.h** suministrados.

La rutina **average_search_time** devuelve **ERR** en caso de error y **OK** en el caso de que las búsquedas se realicen correctamente.

A modo de ayuda, la función **average_search_time** debe realizar los siguientes pasos:

1. Crear un diccionario de tamaño **N**.
2. Crear una permutación de tamaño **N** mediante la rutina **generate_perm**.
3. Insertar en el diccionario los elementos de la permutación anterior mediante el uso de la función int **massive_insertion_dictionary** .
4. Reservar memoria para la tabla que va a contener las **n_times*N** claves en el rango 1 a N a buscar.
5. Llenar la tabla anterior con las **n_times*N** claves a buscar mediante el uso del generador de claves. (**Aviso: los generadores de claves generan números de 1 a N, por tanto es importante que vuestras permutaciones también sean de los números de 1 a N**).
6. Medir el tiempo (reloj y OBs) que tarda en buscar las **n_times*N** claves almacenadas en la tabla anterior.
7. Rellenar correctamente los campos de la estructura **ptime**.
8. Liberar memoria y salir.

Además se debe implementar la función:

```
short generate_search_times(pfunc_search method, pfunc_key_generator generator,
 int order, char* file, int num_min,  int num_max, int incr, int n_times);
```

que sirve para automatizar la toma de tiempos. Esta función llama a la función anterior con un tamaño de diccionario 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. Además debe guardar los resultados en el fichero **file** mediante la función ya implementada **short save_time_table(char* file, PTIME_AA time, int N)**. El resto de parámetros son equivalentes a la rutina anterior.
Con estas rutinas de medición de tiempos se deben realizar los siguientes pruebas:

1. Comparar tiempo medio y operaciones básicas medias de las búsquedas lineal y binaria. El método de búsqueda lineal se aplicará sobre diccionarios desordenados y la búsqueda binaria en diccionarios ordenados. Se deben buscar todas las claves contenidas en el diccionario una sola vez (es decir n_times=1) y la función de generación de claves debe ser **uniform_key_generator** incluida en search.h y search.c. **Se debe hacer la comparación al menos para tamaños de diccionarios desde 1000 a 10000 elementos**.

2. Sin embargo, no siempre las claves a buscar siguen una distribución uniforme sino que habrá unas claves que se buscarán con más frecuencia que otras. Bajo estas condiciones se puede mejorar el tiempo medio de extracción si hacemos que las claves más frecuentes se puedan obtener en tiempo cercano a O(1). Un modo de realizar esto es mediante búsqueda lineal auto–organizada que utiliza las propias claves a buscar para ir colocando las claves más comunes al inicio de la lista. Para que la búsqueda auto–organizada tenga efecto se deberán buscar muchas claves para conseguir que la colocación de las claves en la lista alcance su orden óptimo.

   Las listas autoorganizadas son particularmente útiles cuando la consulta a los datos siguen reglas como la de 80/20: el 80% de las consultas preguntan por el 20% de las claves, o cuando la frecuencia de los datos siguen distribuciones potenciales como la de Zipf, como ocurre con las palabras de un texto: hay pocas palabras con mucha frecuencia y muchas palabras con poca frecuencia, esto también parece suceder con los enlaces de una página web.

   Para realizar las pruebas, se deberá trabajar con la función generadora de claves **potential_key_generator**, que genera claves con una distribución no uniforme. Con este generador de claves se deberá comparar el tiempo medio

y número de operaciones básicas medias para la búsqueda binaria en diccionarios ordenados con la búsqueda auto–organizada (**lin_auto_search**) en diccionarios desordenados, cambiando el valor del parámetro **n_times** para 1, 100 y 10000.

El programa C de prueba para estas rutinas es exercise2.c incluido en **code_p3.zip**.

Cuestiones teóricas

1. ¿Cuál es la operación básica de **bin_search**, **lin_search** y **lin_auto_search**?
2. Dar tiempos de ejecución en función del tamaño de entrada n para el caso peor $W_{SS}(n)$ y el caso mejor $B_{SS}(n)$ de **bin_search** y **lin_search**. Utilizar la notación asinstótica (O, Θ, o, Ω,etc) siempre que se pueda.
3. Cuando se utiliza **lin_auto_search** y la distribución no uniforme dada ¿Cómo varía la posición de los elementos de la lista de claves según se van realizando más búsquedas?
4. ¿Cuál es el orden de ejecución medio de **lin_auto_search** en función del tamaño de elementos en el diccionario n para el caso de claves con distribución no uniforme dado? Considerar que ya se ha realizado un elevado número de búsquedas y que la lista está en situación más o menos estable.
5. Justifica lo más formalmente que puedas la corrección (o dicho de otra manera, el por qué busca bien) del algoritmo **bin_search**.

Material a entregar en cada uno de los apartados

Documentación: La documentación constará de los siguientes apartados:

1. **Introducción:** Consiste en una descripción de tipo técnico del trabajo que se va a realizar, qué objetivos se pretenden alcanzar, qué datos de entrada requiere vuestro programa y que 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:** En el último apartado, incluir las respuestas en papel a las cuestiones teóricas anteriores.

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).
Adicionalmente, las prácticas deberán ser guardadas en algun medio de almacenamiento (lapiz usb, CD o DVD, disco duro, disco virtual remoto, etc) por el alumno para el día del examen de prácticas en Diciembre.

Ojo: Se recalca la importancia de llevar un lapiz usb **ademas de otros medios de almacenamiento como discos usb, cd, disco virtual remoto, email a direccion 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.

Asimismo, una copia de los fuentes y la documentación deberán enviarse por el sistema electrónico de entrega de prácticas, como se indica a continuación.

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 a través de Moodle por medio de la página web **moodle.uam.es**.