Análisis de Algoritmos

Práctica 3

Grupo 1261

Pareja 09

Ari Handler Gamboa

Adrián Lorenzo Mateo

1. TAD Diccionario

1.1 Introducción

En este bloque se procederá a desarrollar un diccionario empleando una tabla (ordenada o desordenada) como tipo abstracto de datos. Por otro lado La estructura TIEMPO servirá para almacenar los tiempos de ejecución de los algoritmo de búsqueda sobre el conjunto de claves. Adicionalmente también se diseñarán las funciones de búsqueda lineal, búsqueda binaria (sobre tablas ordenadas) y búsqueda lineal auto–organizada (en la que si un elemento es encontrado en la tabla, se intercambia con el inmediatamente anterior, excepto si se trata del primer elemento de la tabla) .

1.2 Código

```
int swap(int pos_1, int pos_2, int *vector)
{
    int temp;

    if((vector == NULL) || ((pos_1 < 0) || (pos_2 < 0)))
        return ERR;

    temp = *(vector+pos_1);
    *(vector+pos_1) = *(vector+pos_2);
    *(vector+pos_2) = temp;

    return 0;

}
```

Función auxiliar que intercambia los elementos en las posiciones **pos_1** y **pos_2** de la tabla **vector**, devolviendo **ERR** en caso de que haya habido algún error con los argumentos de entrada u 0 en caso contrario.

```
PDICC ini_diccionario (int tamanio, char orden)
{
    PDICC nuevoDiccionario;

    /* Control de errores de los argumentos de entrada */
    if (tamanio <= 0 ||
     ((orden != ORDENADO)&&(orden != NO_ORDENADO)))
        return NULL;

    /* Creacion del TAD Diccionario */
    if ((nuevoDiccionario = (PDICC)malloc(sizeof(DICC))) ==
                                NULL)
        return NULL;

    /* Reserva de memoria para la tabla de enteros del
diccionario */
    if ((nuevoDiccionario->tabla =
           (int *)calloc(tamanio,sizeof(int))) == NULL){
        free(nuevoDiccionario);
        return NULL;
    }

    /* Completado del resto de campos del diccionario */
    nuevoDiccionario->tamanio = tamanio;
    nuevoDiccionario->n_datos = 0;
    nuevoDiccionario->orden = orden;

    return nuevoDiccionario;
}
```

Función que reserva toda la memoria necesaria para un nuevo diccionario de tamaño de tabla asociado **tamanio** e indicando si esta estará ordenada a o no según el parámetro **orden**. Devolverá **NULL** en el caso de que se produzca algún error en los argumentos de entrada o durante la reserva de memoria, en caso contrario devolverá el nuevo diccionario.

```
void libera_diccionario(PDICC pdicc)
{
    free(pdicc->tabla);
    free(pdicc);
}
```

Rutina que libera toda la memoria reservada correspondiente al diccionario **pdicc**.

```
int inserta_diccionario(PDICC pdicc, int clave)
{
    int i, j;
    int clave_aux, cuenta_obs=0;

    /* Control de errores de los argumentos de entrada */
    if (pdicc == NULL)
        return ERR;

    /* Comprobacion de sobrepaso del tamano de tabla */
    if (pdicc->n_datos == pdicc->tamanio)
        return ERR;

    /* Si la tabla esta ordenada, se utiliza InserSort para
insertar el elemento */
    if (pdicc->orden == ORDENADO){

    /* Para ordenar, inserto al final y comparo con las
anteriores */
    pdicc->tabla[pdicc->n_datos] = clave;
    i = pdicc->n_datos;
    /* Se realizan swaps sucesivos hasta dejar el elemento en
     su posicion ordenada */
    while ((i>0) && (pdicc->tabla[i] < pdicc->tabla[i-1])){
        cuenta_obs++;
        swap(i, i-1, pdicc->tabla);
        i--;
     }

    }
    /* Si no esta ordenada, se inserta al final */
    if (pdicc->orden == NO_ORDENADO){

        pdicc->tabla[pdicc->n_datos] = clave;
    }
    /* El numero de datos ahora es 1 mas grande */
    pdicc->n_datos++;

    return cuenta_obs;
}
```

Función que inserta un elemento **clave** en la tabla asociada al diccionario **pdicc**, utilizando el algoritmo **InsertSort** en caso de que dicha tabla se encuentre ordenada o insertándola al final en caso de que no. Esta función devolverá **ERR** en caso de que se produzca algún error en los argumentos de entrada o bien que la tabla se encuentre al límite de su capacidad. En caso contrario devolverá el número de operaciones básicas (comparación de claves) ejecutadas durante la inserción de la clave.

```
int insercion_masiva_diccionario (PDICC pdicc,int *claves, int
n_claves)
```

```
{
    int i;
    int cuenta_obs = 0;
    int ret;

    /* Control de errores de los argumentos de entrada */
    if (pdicc == NULL || claves == NULL || n_claves < 0)
        return ERR;

    /* Comprobacion de sobrepaso del tamano de tabla */
    if (pdicc->tamanio < (pdicc->n_datos + n_claves))
        return ERR;

    /* Bucle que llama a la insercion de un unico elemento por
     cada clave en el array de claves */
    for(i=0; i<n_claves ;i++){
        if ((ret = inserta_diccionario(pdicc, claves[i])) ==
                                                    ERR)
            return ERR;
        cuenta_obs += ret;
    }

    return cuenta_obs;
}
```

Rutina que inserta en el diccionario **pdicc** los **n_claves** primeros elementos de la tabla de claves **claves**, para ello llamará iterativamente a la función **inserta_diccionario** por cada elemento. La función devolverá **ERR** en el caso de que se produzca algún error con los argumentos de entrada, se sobrepase el tamaño límite de la tabla asociada al diccionario o falle la inserción de una clave. En caso contrario devolverá la suma de operaciones básicas ejecutadas durante la inserción.

```
int busca_diccionario(PDICC pdicc, int clave, int *ppos,
                      pfunc_busqueda metodo)
{
    int cuenta_obs=0;

    /* Control de errores de los argumentos de entrada */
    if (pdicc == NULL || metodo == NULL)
        return ERR;

    /* Se llama al metodo de busqueda. La salida de dicho
     metodo se devuelve directamente puesto que el CDE
     correspondiente se hara en un nivel superior. */
    cuenta_obs = metodo(pdicc->tabla, 0, pdicc->n_datos-1,
clave, ppos);

    return cuenta_obs;
}
```

Rutina que busca en el diccionario **pdicc** la clave **clave** mediante el método de búsqueda **metodo**, devolviendo la posición de dicho elemento en la tabla asociada (en caso de que se encuentre) en **ppos**. Esta rutina devuelve **ERR** en caso de que se produzca algún error con los

argumentos de entrada o el número de operaciones básicas ejecutadas por el algoritmo de ordenación.

```
void imprime_diccionario(PDICC pdicc)
{
    int i;

    if (pdicc == NULL)
        return;

    printf("\nTamanio del diccionario = %d\n", pdicc->tamanio);
    printf("# de elementos del diccionario = %d\n", pdicc-
>n_datos);
    printf("Ordenado = %s\n", pdicc->orden == ORDENADO ? "Si" :
"No");
    printf("Tabla = ");
    for(i=0; i<pdicc->n_datos ;i++){
        printf("%d\t", pdicc->tabla[i]);
        if((i%10) == 9)
            printf("\n");
    }

    printf("\n");

    return;
}
```

Función que imprime por pantalla toda la información perteneciente al diccionario **pdicc**, esto es, su tamaño máximo, el número de elementos de la tabla, si se encuentra ordenada y todos sus elementos en una matriz de 9 claves de ancho.

```
int bbin(int *tabla,int P,int U,int clave,int *ppos)
{
    int M; /*indice medio de la tabla*/
    int ob_count = 0; /*contador de operaciones basicas*/

    /* Control de argumentos de entrada */
    if (tabla == NULL || P<0 || U<P)
        return ERR;

    /* Algoritmo de busqueda binaria */
    while (P <= U){
        M = (P + U)/2;
        ob_count++;
        /*CDC*/
        if (tabla[M] == clave){
            *ppos = M;
            return ob_count;
        }
        if (clave < tabla[M])
            U = M - 1;
        else
            P = M + 1;
```

```
    }

    return NO_ENCONTRADO;
}
```

Función que implementa la búsqueda binaria del elemento **clave** sobre la tabla **tabla** desde el índice **P** hasta el índice **U**, devolviendo la posición de dicho elemento (en caso de encontrarse) en el argumento **ppos**. Esta función devolverá **ERR** en caso de que se produzca algún error con los argumentos de entrada, el número de operaciones básicas ejecutadas en caso de que se encuentre la clave o **NO_ENCONTRADO** en caso contrario.

```
int blin(int *tabla,int P,int U,int clave,int *ppos)
{
    int i;
    int cuenta_obs = 0;

    if (tabla == NULL || P>U)
        return ERR;

    for(i=0; i<=U ;i++){
        cuenta_obs++;
        if (tabla[i] == clave){
            *ppos = i;
            return cuenta_obs;
        }
    }

    return NO_ENCONTRADO;
}
```

Rutina que implementa la búsqueda lineal del elemento **clave** sobre la tabla **tabla** desde el índice **P** hasta el índice **U**, devolviendo la posición de dicho elemento (en caso de encontrarse) en el argumento **ppos**. Esta rutina devolverá **ERR** en caso de que se produzca algún error con los argumentos de entrada, el número de operaciones básicas ejecutadas en caso de que se encuentre la clave o **NO_ENCONTRADO** en caso contrario.

```
int blin_auto(int *tabla,int P,int U,int clave,int *ppos)
{
    int i;
    int cuenta_obs = 0;
    int aux;

    /* Control de argumentos de entrada */
    if (tabla == NULL || P>U)
        return ERR;
```

```
     /* Algoritmo de busqueda lineal auto-organizada */
    for(i=0; i<=U ;i++){
        cuenta_obs++; /* CDC * /
        /* Si no es el primer elemento realiza un swap */
        if ((tabla[i] == clave)&&(i!=0)){
            swap(i, i-1, tabla);
            *ppos = i;
            return cuenta_obs;
        }
        /*si es el primer elemento no intercambia*/
        else if((tabla[i] == clave)&&(i==0))
            return cuenta_obs;
    }

    return NO_ENCONTRADO;
}
```

Rutina que implementa la búsqueda lineal auto-organizada del elemento **clave** sobre la tabla **tabla** desde el índice **P** hasta el índice **U**, devolviendo la posición de dicho elemento (en caso de encontrarse) en el argumento **ppos**. Esta rutina devolverá **ERR** en caso de que se produzca algún error con los argumentos de entrada, el número de operaciones básicas ejecutadas en caso de que se encuentre la clave o **NO_ENCONTRADO** en caso contrario.

1.3 Resultados

1.3.1 Resultados de las pruebas realizadas con ejercicio1

- **tamanio <= 0:** La rutina ini_diccionario encuentra el error devolviendo NULL.

```
Practica numero 3, apartado 1
Realizada por: Ari Handler, Adrián Lorenzo
Grupo: 261
Error: No se puede Iniciar el diccionario
```

Al utilizar la herramienta **Valgrind** y utilizando las banderas de ejecución correspondientes a la opción **verbose** y de identificación de fugas de memoria obtenemos el resultado siguiente:

```
==6282==
==6282== HEAP SUMMARY:
==6282==     in use at exit: 0 bytes in 0 blocks
==6282==   total heap usage: 0 allocs, 0 frees, 0
```

```
bytes allocated
==6282==
==6282== All heap blocks were freed -- no leaks are
possible
==6282==
==6282== For counts of detected and suppressed
errors, rerun with: -v
==6282== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 14 from 7)
```

- **tamanio = 1 clave = 0 (primer elemento) :** El programa se ejecuta correctamente.

```
Practica numero 3, apartado 1
Realizada por: Ari Handler, Adrián Lorenzo
Grupo: 261


Tamanio del diccionario = 1
# de elementos del diccionario = 1
Ordenado = No
Tabla = 0


>>Busqueda lineal


Clave 0 encontrada en la posicion 0 en 1 op.
basicas


>>Busqueda lineal auto-organizada


Clave 0 encontrada en la posicion 0 en 1 op.
basicas


Tamanio del diccionario = 1
# de elementos del diccionario = 1
Ordenado = Si
Tabla = 0


>>Busqueda binaria


Clave 0 encontrada en la posicion 0 en 1 op.
basicas
```

Al utilizar la herramienta **Valgrind** y utilizando las banderas de ejecución correspondientes a la opción **verbose** y de identificación de fugas de memoria obtenemos el resultado siguiente:

```
==6298==
==6298== HEAP SUMMARY:
==6298==     in use at exit: 0 bytes in 0 blocks
==6298==   total heap usage: 6 allocs, 6 frees, 48
bytes allocated
==6298==
==6298== All heap blocks were freed -- no leaks are
possible
==6298==
==6298== For counts of detected and suppressed
errors, rerun with: -v
==6298== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 14 from 7)
```

- **tamanio > 1 clave >= tamanio (no se encuentra):** El programa se ejecuta correctamente.

```
Practica numero 3, apartado 1
Realizada por: Ari Handler, Adrián Lorenzo
Grupo: 261

Tamanio del diccionario = 4
# de elementos del diccionario = 4
Ordenado = No
Tabla = 3   1     0      2

>>Busqueda lineal

La clave 5 no se encontro en la tabla

>>Busqueda lineal auto-organizada

La clave 5 no se encontro en la tabla

Tamanio del diccionario = 4
# de elementos del diccionario = 4
Ordenado = Si
Tabla = 0   1     2      3
```

```
>>Busqueda binaria


La clave 5 no se encontro en la tabla
```

Al utilizar la herramienta **Valgrind** y utilizando las banderas de ejecución correspondientes a la opción **verbose** y de identificación de fugas de memoria obtenemos el resultado siguiente:

```
==6309==
==6309== HEAP SUMMARY:
==6309==     in use at exit: 0 bytes in 0 blocks
==6309==   total heap usage: 6 allocs, 6 frees, 96
bytes allocated
==6309==
==6309== All heap blocks were freed -- no leaks are
possible
==6309==
==6309== For counts of detected and suppressed
errors, rerun with: -v
==6309== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 14 from 7)
```

- **tamanio > 1 clave < tamanio (se encuentra):** El programa se ejecuta correctamente.

```
Practica numero 3, apartado 1
Realizada por: Ari Handler, Adrián Lorenzo
Grupo: 261


Tamanio del diccionario = 4
# de elementos del diccionario = 4
Ordenado = No
Tabla = 0   2     3      1


>>Busqueda lineal


Clave 3 encontrada en la posicion 2 en 3 op.
basicas


>>Busqueda lineal auto-organizada
```

```
Clave 3 encontrada en la posicion 2 en 3 op.
basicas


Tamanio del diccionario = 4

# de elementos del diccionario = 4

Ordenado = Si

Tabla = 0   1     2      3


>>Busqueda binaria


Clave 3 encontrada en la posicion 3 en 3 op.
basicas
```

Al utilizar la herramienta **Valgrind** y utilizando las banderas de ejecución correspondientes a la opción **verbose** y de identificación de fugas de memoria obtenemos el resultado siguiente:

```
==6314==

==6314== HEAP SUMMARY:

==6314==     in use at exit: 0 bytes in 0 blocks

==6314==   total heap usage: 6 allocs, 6 frees, 96
bytes allocated

==6314==

==6314== All heap blocks were freed -- no leaks are
possible

==6314==

==6314== For counts of detected and suppressed
errors, rerun with: -v

==6314== ERROR SUMMARY: 0 errors from 0 contexts
(suppressed: 14 from 7)
```