Proyecto 3
TAD Hash

Algoritmos y Estructuras de Datos II
Laboratorio

25 de abril de 2008

Implementar un TAD diccionario con un *hash*.

Igual que en el proyecto anterior, el diccionario debe poder ser guardado y leido desde disco. Para ello se brindan ya programadas los TAD's cinta de lectura y escritura de pares (ver en la página de la materia).

El diseño de estas modificaciones será el siguiente:

- El diccionario debe implementarse ahora con un TAD hash conservando las funcionalidades del proyecto anterior.
- Hay que implementar el nuevo TAD *hash* como se vió en el teórico. Esto es, hay que conservar los TAD *lista de asociaciones* del proyecto anterior y definir una estructura como sigue:

  ```
  struct sHash {
    ListaAsoc arr[TAM_HASH];
    int cantelem;
  };
  ```

  donde `TAM_HASH` es una constante que denota el tamaño de la tabla hash (tamaño de arrreglo) y `cantelem` es la cantidad de tuplas ingresadas. Esta estructura debe ser declarada en un archivo `.c` .
- Se debe escribir la parte pública del TAD en un archivo `hash.h`. Un esquema del `.h` puede ser el siguiente:

  ```
  #include "listaAsoc.h"

  typedef struct sHash * Hash;

  /* constructor */
  Hash
  hash_empty (void);

  /* agrega una clave y un dato */
  ```

```
void
hash_add (Hash h, Key k, Data d);

/* devuelve si existe la clave en el hash */
Bool
hash_exists (Hash h, Key k);

/* busca un dato segun la clave
   debe llamarse solo si hash_exists (l, k)
   si no lo encuentra devuelve NULL */
Data
hash_search (Hash h, Key k);

/* borra la clave y el valor asociado */
void
hash_del (Hash h, Key k);

/* devuelve la cantidad de elementos */
int
hash_length (Hash h);


/* Lee un hash desde un archivo
Hash
hash_fromFile(char *nomfile);

/* Graba un hash a un archivo
void
hash_toFile(char *nomfile, Hash h);


/* Destructor.
   Destruye la tabla y su contenido */
Hash
hash_destroy(Hash h);
```

- Todas estas funciones se implementan muy facilmente utilizando las funciones de lista de asociaciones del proyecto anterior y la función de hash (como se vió en clase).

  Solo puede haber problemas con la implementación de la función `hash_toFile`. El problema es que si se usa la función `la_toFile` para grabar cada lista de la tabla hash, solo se grabará la última (fijarse por que sucede esto). Esto puede solucionarse implementando la función

  ```
  void
  la_toCinta(cw c, ListaAsoc la);
  ```

  la cual toma una lista de asociaciones `la` y la escribe en una cinta de escritura en disco ya creada y arrancada. Con esta función implementar aquella del TAD hash.

- Dejando todos los demás TAD's igual que en el proyecto anterior (inclusive la interfase con el usuario) testear esta nueva implementación.

Ejercicio ⋆ 1 *Una vez terminado el proyecto, programar otras funciones de hash. Para ello investigar distintos algorítmos o librerías (como* gperf*). Consultar con los profesores del taller y los ayudantes.*

Ejercicio ⋆ 2 *Con la variable* `NDEBUG` *habilitar código que muestre la performance de la función de hash. Esto se puede hacer con cerraduras* `ifndef NDEBUG` *sobre código que muestre por pantalla la media y la varianza de longitudes de la listas en la tabla de hash. Si no sabe que es la media y la varianza, pregunte a los profesores y/o ayudantes.*

Comparar con diferentes funciones de hash estos resultados.

Además, de forma general, para hacer cada TAD se deberá tener en cuenta:

- Todos los TAD's deberán estar implementados con la técnica de punteros a registros vista en el teórico del laboratorio.
- Cada TAD se deberá escribir en un par de archivos separados, un `.h` ("headers") y un `.c`.
- Para implementar correctamente los TAD's, cada `.h` deberá exportar únicamente las funcionalidades del TAD que define, ocultando todos los aspectos que tienen que ver con su implementación.
- No definir las estructuras que implementan los TAD's en los archivos "headers" ya que allí solo va la parte pública del TAD y no los detalles de implementación.
- Los programas deben estar libres de "memory leaks".
- Recordar que todos los `.h` deben tener su cerradura `ifndef`.
- Utilizar `Makefile`.
- Utilizar todos los parámetros de *gcc* para detectar posibles errores a tiempo de compilación.
- Para obtener nota B el programa debe funcionar sin errores y debe cumplir con todos los puntos anteriores.
- Para obtener nota B^+ el programa debe funcionar sin errores, debe cumplir con todos los puntos anteriores y se deben hacer todos los puntos estrella.