Córdoba, 23 de Febrero del 2023

Examen Final

Algoritmos y Estructuras de Datos II - Taller

TAD Diccionario

Se debe programar el Tipo Abstracto de Datos "*Diccionario*", implementándolo **como una lista enlazada de nodos**. Cada nodo almacena una clave de tipo `key_t` y un valor o dato de tipo `value_t`, ambos tipos sinónimos del tipo `string_t`. Notar que los elementos del diccionario, a diferencia de la implementación con árboles, estarán desordenados. La representación elegida además tiene una estructura principal con un puntero hacia la cadena de nodos. Deben modificar las estructuras e implementar las funciones del TAD de manera tal que determinar cuántos elementos tiene el diccionario se haga en tiempo constante (O(1)). El TAD debe ser **opaco,** es decir que la estructura de representación del TAD no debe ser visible en el `.h`. El TAD debe tener los siguientes métodos

Función	Descripción
`dict_t dict_empty(void)`	Crea un diccionario vacío
`dict_t dict_add(dict_t dict, key_t word, value_t value)`	Agrega una nueva palabra `word` junto con su definición `def`. En caso que `word` ya esté en el diccionario, se actualiza su definición con `def`.
`value_t dict_search(dict_t dict, key_t word)`	Devuelve la definición de la palabra `word` contenida en el diccionario `dict`. Si la palabra no se encuentra devuelve `NULL`
`bool dict_exists(dict_t dict, key_t word)`	Indica si la palabra `word` está en el diccionario `dict`
`unsigned int dict_length(dict_t dict)`	Devuelve la cantidad de palabras que tiene actualmente el diccionario `dict`, **debe ser de orden constante**
`dict_t dict_remove(dict_t dict, key_t word)`	Elimina la palabra `word` del diccionario. Si la palabra no se encuentra devuelve el diccionario sin cambios.
`dict_t dict_remove_all(dict_t dict)`	Elimina todas las palabras del diccionario `dict`
`void dict_dump(dict_t dict, FILE *file)`	Escribe el contenido del diccionario `dict` en el archivo `file`
`dict_t dict_destroy(dict_t dict)`	Destruye la instancia `dict` liberando toda la memoria utilizada.

En `dict.h` están descriptas las pre y post-condiciones de los métodos. **Deben chequear las pre y post-condiciones de usando** `assert()`. Particularmente deben completar la invariante de representación y verificarla.

Para probar el TAD proveemos un archivo **`main.c`** con una interfaz para manipular el diccionario. Para compilar además cuentan con un **`Makefile`** por lo que se compila haciendo

```
$ make
```

y luego

```
$ ./dictionary
```

Consideraciones

- Si **`dict.c`** no compila, no se aprueba el examen.
- Si **`stack_size()`** no es de orden constante baja muchísimos puntos
- No implementar la invariante baja puntos
- No chequear pre y post condiciones baja puntos
- Los *memory leaks* bajan puntos
- No se pueden modificar los archivos .h provistos por la cátedra
- No se puede modificar el **`Makefile`** debiendo ser posible compilar el código usando el original
- Entregar código muy improlijo puede restar puntos
- Para sacar **P** en el examen **se debe** hacer una invariante no trivial.
- Se recomienda usar las herramientas **`valgrind`** y **`gdb`**.