Algoritmos y Programación II [75.41]	Trabajo PrácticoGrupal Nº 2
Cátedra Lic. Gustavo Carolo	AB Indexes
1º Cuatrimestre 2014	Fecha de entrega:25 de Junio de 2014

Trabajo Práctico Grupal Nº 2: AB Indexes

(V 1.0)

Introducción

Un índice es una estructura que permite mejorar la velocidad de las operaciones de recuperación en una estructura de datos. Por medio de identificador único de cada elemento de una estructura, permite obtener un rápido acceso a elementos ordenados por el mismo. Al aumentar drásticamente la velocidad de acceso, se suelen usar, sobre aquellos campos sobre los cuales se hacen frecuentes búsquedas. El índice tiene un funcionamiento similar al índice de un libro. Almacena metadatos en forma de pares: el elemento, o la parte del elemento que se desea indexar por un lado, y su posición en la estructura, señalada por el identificador único de los elementos que les da orden físico dentro de la estructura de datos, por el otro. Para buscar un elemento a partir de una porción del mismo que cuenta con un índice, sólo hay que buscar en el índice dicha porción, para, una vez encontrado su identificador único, realizar una búsqueda por clave, dentro de la estructura utilizada.

Los índices pueden ser creados usando uno o más campos de un tipo de dato, proporcionando la base tanto para búsquedas puntuales, como para barridos o recorridos secuenciales, a partir de campos por los que no se ha ordenado la estructura.

Los índices son construidos normalmente sobre árboles B, B+, B* o sobre una mezcla de ellos, funciones de cálculo u otros métodos. Este tipo de estructura permite accesos sumamente eficientes para recuperación de datos. Sin embargo, el costo de construirlas excede ampliamente los parámetros para nuestra materia. Por este motivo, utilizaremos el concepto de índice descrito previamente, pero haciendo uso de una estructura notablemente más sencilla: El Árbol Binario.

Objetivo

En este tp el alumno deberá lograr suficiente familiaridad con la estructura de datos entendida como árbol binario, entendiendo el funcionamiento de la implementación particular utilizada, el método de ordenamiento y las formas de navegarla y recorrerla. Se espera además, que el alumno logre extraer la noción básica del funcionamiento de un índice y la idea general detrás de una estructura de tipo árbol.

Definición de TDA's

TDA_Index

Se deberá generar un nuevo tipo de dato abstracto: el índice. Haciendo uso de los conceptos obtenidos hasta ahora en la materia, se deberá definir una estructura que permita almacenar pares de valores con la idea de (clave, valor), en una estructura sencilla y de acceso rápido. La mejor estructura que conocemos para la tarea, hasta este punto, es el árbol binario.
En dicha estructura se deberán almacenar elementos capaces de guardar los pares de valores anteriormente mencionados, donde sus tipos no están definidos hasta el momento de su utilización, tal y como sucede con el caso de las estructuras de datos vistas en clase. Tal vez no esté de más agregar, que esta estructura deberá ordenarse según algún criterio dependiendo de los valores inyectados como clave. Dado que se desconoce de qué tipo se trata, también se desconoce, a priori, cómo se debieran ordenar. Por eso, para este efecto, se hará uso de punteros a función para las comparaciones de dichas claves.

Estructuras

```
typedef int (*F_Cmp)(const void*, const void*);
typedef int (*F_Clone)(void* destination, const void* source);
typedef int (*F_Destroy)(void*);
typedef int (*F_Operate)(void* value, void* shared_data);
typedef struct index_t {
    TAB data; /* esta estructura deberá tener un AB */

    /* definición a cargo del grupo */

} T_Index;
```

Se sugiere incluir en la estructura todo lo necesario para resolver las operaciones listadas en las primitivas. Se pueden declarar sub-estructuras adicionales si hubiera necesidad.

Primitivas

```
int idx_create(T_Index* i, size_t key_size, size_t value_size, F_Cmp keycmp, F_Clone
key_clone, F_Clone, value_clone, F_Destroy key_destroy, F_Destroy value_destroy);
int idx_destroy(T_Index* i);
int idx_put(T_Index* i, const void* key, const void* value /*argumento de entrada*/);
int idx_get(T_Index* i, const void* key, TListaSimple* values /*argumento de salida*/);
int idx_go_through(T_Index* i, F_Operate operate, void* shared_data, int break_value);
```

Se imponen algunas explicaciones de aspectos no obvios de las primitivas listadas arriba.
idx_create: Recibe el índice, el tamaño de dato que van a ocupar las claves, y el tamaño de dato que van a ocupar los valores. Además, recibe 5 punteros a función:

- keycmp para comparar claves
- key_clone para clonar valores del tipo de la clave (para hacer copias profundas)
- value_clone para clonar valores del tipo de los valores
- key_destroy para destruir valores del tipo de las claves
- value_destroy para destruir valores del tipo de los valores

Las primitivas de inserción (*put*) y recuperación (*get*) no ameritan más explicación que aclarar que en el caso de *put*, el value se pasa cargado con los datos que deberán guardarse, mientras que en el caso de *get*, se lo pasa vacío (reservado en memoria, pero sin datos) para que dentro de la primitiva se llene de información. Hemos decidido utilizar como argumento de salida una lista simple, ya que siendo que los índices alternativos pueden tener valores repetidos, al hacer get de un valor, pueden haber varios ids asociados, por lo tanto, no sería posible esperar sólo un valor.
La primitiva *'idx_go_through'* deberá implementar un recorrido, IN-ORDEN dentro del AB, interno al índice, permitiendo un barrido completo del índice, aplicándole a cada elemento la función apuntada por 'operate', que además del value de cada elemento, recibirá el espacio de memoria 'shared_data', compartida entre los sucesivos llamados a operate (de esta manera, se pueden compartir datos entre los llamados a operate sobre cada elemento). Finalmente, 'break_value' se agrega como una condición de corte, de manera que el recorrido interno del índice, sea interrumpido en caso de que la llamada a operate genere como resultado dicho valor.

TDA_ProjectReport

Al TDA_ProjectReport del tp anterior, se le deberá incorporar la utilización de varios índices, para permitir generar reportes más refinados que los que generaba, admitiendo así filtros en las salidas. Haciendo uso de índices que contengan, para un dado valor, clave del índice, los ids de las tareas que contengan ese valor.
Estos índices se deberán generar dentro de la estructura de Project Report. La estrategia definida para incorporarlos queda a cargo del grupo.

Archivos de salida esperados

Los archivos de salida que se esperan, dependerán de las instrucciones que se invoquen al programa de aplicación, pero deberán tener el mismo formato que el definido para la salida del tp1-grupal. Para verlos en detalle, vea la sección de programa de aplicación.

Ejecución

Programa de aplicación

En esta ocasión, se deberá generar un programa de ejecución que sea interactivo. Se invocará igual que el del trabajo práctico anterior, pero al iniciar, levantará todos los datos traídos de asana, pero en vez de resumirlos en el reporte, construirá índices por los campos:

- Due Date
- Asignee
- Tags

Una vez levantado el programa, de la misma manera que en el tp anterior, el programa deberá aceptar por standard input los comandos *query* y *report*, como se muestra a continuación.

```
query -k <key_field> -v <value>
query -p <key_field_1:value_a> -p <key_field_2:value_b> … -p <key_field_3:value_c>
query -k <key_field> -s <value_a> <value_b> ... <value_z> -a <key_field>
report -s <value_a> <value_b> ... <value_z> -k <key_field> -o <output_file>
report -o <output_file> -v <value_a> <value_b> ... <value_z> -k <key_field> -d <key_field>
```

Donde query es una simplificación de report que imprime la salida directamente a stdout.
Se indicará la clave a utilizar, de la que se deduce qué índice o índices serán utilizados, utilizando la opción 'k' (-k), y el valor para dicha clave, utilizando la opción 'v' (-v).
Como alternativa, para definir criterios mixtos de búsqueda, se podrá incluir la opción 'p' (-p), indicando el par de valores, clave:valor, separados por un símbolo ':' (dos puntos).
Para definir múltiples valores que buscar en un campo, se utilizará la opción 's' (-s), para definir el set de valores que se quieren recuperar de la estructura.
Cómo simplificación al programa de aplicación, no se utilizarán en una misma instrucción, las opciones -v y -s, ni la opción -p. Es decir que un comando -k/-v, no deberá soportar la adición de otro par de opciones -k/-s, ni se deberá soportar otra opción de filtro cuando se encuentre presente la opción -p. Sin embargo, se deberá soportar la opción de ordenar el resultado de la operación, incluyendo, en cualquier caso, la opción 'a' (-a), para indicar el campo por el que se deberá ordenar en forma ascendente, o, su contraparte, la opción 'd' (-d), para indicar el campo por el que se deberá ordenar en forma descendente. Las opciones de ordenamiento, son mutuamente excluyentes, por lo que la presencia de ambos en la misma instrucción, deberá considerarse erróneo, tal como la presencia de más de una opción -k, o cualquier combinación de opciones -v, -s y -p.
Para todos los casos, se deberá definir un comportamiento sistémico para cuando el usuario ingrese opciones inválidas. Es decir, que no es aceptable que el programa falle en caso de recibir argumentos inválidos. El mensaje devuelto en tales casos deberá incluir el texto "opciones invalidas".

Referencias

Para el manejo de las opciones de línea de comandos, investigue la librería getopt.h
http://man7.org/linux/man-pages/man3/getopt.3.html
http://manpages.ubuntu.com/manpages/karmic/es/man3/getopt.3.html
http://bulma.net/body.phtml?nIdNoticia=1290