Curso de Estructuras de Datos y Algoritmos – 2008 (Nocturno)
Práctico 9

1.Considere la siguiente especificación funcional mínima del Tipo Abstracto de Datos Diccionario de Naturales:

- **Dicc Vacio () ;**

/* Retorna el diccionario vacío */

- **Dicc Insertar (T a; Dicc D);**

/* Retorna un nuevo diccionario que consiste de todos los elementos de D más el elemento a, sies que a no era ya un elemento de D */

- **boolean EsVacio (Dicc D);**

/* Retorna TRUE si D es vacío */

- **boolean Pertenece (T a, Dicc D);**

/* Retorna TRUE si a es un elemento de D */

- **Dicc Borrar (T a, Dicc D);**

/* Borra el elemento a del diccionario D. Si a no pertenece a D, el diccionario retornado es D */

a. Escribir en C/C++ módulos de implementación para el TAD Diccionario de Naturales,
mediante listas encadenadas: ordenadas y no ordenadas.
b. Escribir en C/C++ módulos de implementación para el TAD Diccionario de Naturales, mediante la representación clásica de árboles binarios de búsqueda.
c. Escribir en C/C++ módulos de implementación para el TAD Diccionario de Naturales donde los elementos del tipo base pertenecen al subrango 1..N, para algún natural N, mediante un arreglo de bits (o vector booleano).
d. Escribir en C/C++ un módulo de implementación para el TAD Diccionario *acotado* de Naturales, usando una estructura estática. Conviene mantener los elementos ordenados en la estructura ?. Analice las operaciones, en particular: Pertenece.
E .Comparar las implementaciones según cantidad de comparaciones entre elementos de tipo T (en este caso Naturales) que se realizan para las operaciones **Vacio**, **Insertar**, **Pertenece.** No se deberá considerar otro tipo de comparaciones (por ejemplo variables de control de una iteración).

2.Considere Diccionarios de elementos de tipo T. En este ejercicio asumiremos al tipo T como Strings de 10 caracteres. Importamos el tipo y las operaciones del módulo Diccionario de la siguiente forma:

```
#include “Diccionario.h”
Borrar, Pertenece ;
```

Se pide:
a) Escribir en C/C++ una función que reciba una lista de elementos de tipo T y devuelva TRUE si la mayoría de ellos pertenece a DA.

```
boolean SonMayoria?(ListaT l, Dicc DA);
```

b) Escribir en C/C++ una función que reciba una lista de elementos de tipo T y devuelva TRUE si todos ellos pertenecen a DB.

```
boolean Todos?(ListaT l, Dicc DB);
```

c) Escribir en C/C++ un procedimiento que reciba una lista `l` de elementos de tipo T y pase **todos** los elementos de DA a DB que están en `l` .

```
void Pasar(ListaT l, ListaT  &DA, Dicc DB);
```

d) ¿Qué modificaciones haría si quisiera recordar la cantidad de veces que un elemento estuvo en cada diccionario? ¿Seguirían siendo Diccionarios?

3. Considere la siguiente especificación funcional mínima del Tipo Abstracto de Datos Set de Naturales:

- `Set Vacio (); /* Retorna el conjunto vacío */`
- `Set Insertar (T a,Set A);`
  `/* Retorna un nuevo conjunto que consiste de todos los elementos de A más el elemento a, si es que a no era ya un elemento de A */`
- `boolean EsVacio (Set A); /* Retorna TRUE si A es vacío */`
- `boolean Pertenece (T a, Set A);`
  `/* Retorna TRUE si a es un elemento de A */`
- `Set Union (Set A,Set B );`
  `/* Retorna el conjunto de los elementos que pertenecen a A, o B o a ambos */`
- `Set InterSec (Set A, Set B) /* Retorna el conjunto de los elementos que pertenecen a A y a B */`
- `Set Dif (Set A, Set B); /* Retorna el conjunto de los elementos que pertenecen a A y no pertenecen a B */`

a. Escribir en C/C++ módulos de implementación para el TAD Set de Naturales, mediante listas encadenadas: ordenadas y no ordenadas.
b. Escribir en C/C++ un módulo de implementación para el TAD Set de Naturales, mediante la representación clásica de árboles binarios de búsqueda.
c. Escribir en C/C++ un módulo de implementación para el TAD Set de Naturales, mediante la representación de arreglos de bits.
d. Comparar las implementaciones según cantidad de comparaciones (de la misma forma que en el Ejercicio 1 parte e.) que se realizan para las operaciones **Vacio**, **Insertar**, **Pertenece**, **InterSec**.

4. Usando solamente el conjunto mínimo de funciones especificadas en el módulo de definición Sets, implementar las siguientes operaciones sobre conjuntos:
a. `Borrar(a, A)` --> Retorna el conjunto resultante de quitar el elemento a del conjunto A. Si a no pertenece al conjunto A esta operación retorna este último conjunto.
b. `Subconjunto(A, B)` --> Retorna `TRUE` si el conjunto A esta incluido en el conjunto B.
c. `Igual(A, B)` --> Retorna `TRUE` si los conjuntos A y B contienen los mismos elementos.
d. `Sdiff(A, B)` --> Diferencia simétrica, retorna el conjunto de los elementos que pertenecen a A o a B pero no a los dos conjuntos.

5.Se pretende escribir un programa **C/C++** que imprima los números primos menores o iguales a 5000 utilizando el método de la *criba de Eratóstene*s. Este método consiste en un ciclo (iteración) que utiliza un conjunto *C* (la criba) el cual, desde el inicio del ciclo y tras cada paso de éste, cumple la siguiente propiedad: *C* consiste en los naturales *n* tales que $p <= n <= 5000$ para un cierto *p* que es el siguiente primo menor o igual a 5000 a ser listado. En cada paso del ciclo se lista *p* y luego se eliminan de la criba *p* y todos sus múltiplos. Observar que esto garantiza que la propiedad especificada arriba es mantenida por cada paso del ciclo. El ciclo termina cuando la criba queda vacía.

a. Diseñar el programa utilizando un **TAD** cuyos objetos sean los subconjuntos de los naturales menores o iguales que 5000. Utilizar una operación que cree el "subconjunto universal", es decir el conjunto de los naturales menores o iguales que 5000.
b. Implementar el **TAD** introducido en la parte anterior utilizando vectores booleanos.
c. Indicar qué modificaciones deberían efectuarse si se requiriera listar los primos menores o iguales a k, donde *k* es leído de la entrada estándar.
d. Extender el **TAD** especificado en **a**), así como su implementación mediante vectores booleanos, con las operaciones de *unión, intersección y diferencia.*

6.Considere una realidad donde se necesita disponer del tipo fecha, no considerando años bisiestos, pero sí los diferentes largos de los meses. La definición del tipo fecha es la siguiente:

```
typedef  struct Fecha{unsigned short  dd;unsigned short mm; unsigned short aa};
```

a. Se desea disponer de un tipo conjunto de fechas a los efectos de almacenar un máximo de 20 fechas en cada uno. Defina la estructura de datos que considere conveniente, justificando (explicar porqué considera conveniente dicha estructura, la forma de usarla y los valores que pueden tomar los componentes de la misma). Discuta alternativas para implementar como constante la cardinalidad máxima.
b. Escribir una operación que lea un conjunto de fechas del teclado. Inicialmente se leerá un número entre 1 y 20 que se tomará como la cantidad de fechas a ingresar. Cada fecha deberá validarse y leerse nuevamente en caso de no ser válida.
c. Escribir una función que dado un conjunto de fechas, devuelva las correspondientes a un mes dado (por su número). Debería ser posible omitir el mes de la invocación en cuyo caso se tomará Enero. El resultado será devuelto en un nuevo conjunto.
d. Escribir una operación que dados un conjunto de fechas y una fecha elimine aquellas menores que la dada. Deberá devolverse el mismo conjunto modificado.

7. Como se vio en el Ejercicio 7 del Practico 6, una Bolsa o Multiconjunto (bag) es un tipo abstracto de datos que especifica una colección de elementos en la que, a diferencia del Conjunto, pueden existir ocurrencias repetidas de los mismos. Por ejemplo, la Bolsa {1, 3, 3, 4} no es la misma Bolsa que {1,3, 4}. El orden de los elementos es irrelevante, distinguiéndose así de la noción de lista.
Para el módulo de definición del TAD Bolsa de Naturales de la parte ***a*** del Ejercicio 7 del Practico 6 que se ha descrito:

a. Escribir en **C/C++** el correspondiente módulo de implementación para el TAD Bolsa de Naturales, mediante listas encadenadas:
i. Ordenadas con elementos repetidos y sin elementos repetidos (llevando cuenta del número de ocurrencias de cada elemento)
ii.No ordenadas con elementos repetidos y sin elementos repetidos (llevando cuenta del número de ocurrencias de cada elemento)
b. Escribir en **C/C++** un módulo de implementación para el TAD Bolsa de Naturales, mediante la representación clásica de árboles binarios de búsqueda:
i. Con elementos repetidos ii.Sin elementos repetidos (llevando cuenta del número de ocurrencias de cada elemento)
c. Escribir en **C/C++** un módulo de implementación para el TAD Bolsa de Naturales donde los elementos del tipo base pertenecen al subrango 1..N, para algún natural N, mediante un vector y llevando cuenta del número de ocurrencias de cada elemento.

8.Considere la siguiente especificación funcional mínima para el Tipo Abstracto de Datos Tabla (Mapping o Asociación) para el dominio `Dom` y el rango `Rango` :

- `Tabla CrearTabla (); /* Retorna la tabla vacía */`
- `Tabla Insertar (Dom d, Rango r,Tabla T);`
  `/* Define T(d) = r, independientemente de que T(d) estuviera ya definido */`
- `Rango Recuperar (Dom d, Tabla T);`
  `/* Retorna T(d) si T está definida para d*/`
- `bolean EstaDef? (Dom d, Tabla T);`
  `/* retorna TRUE si T está definida para d */`
- `Tabla Borrar (Dom d, Tabla T); /* Borra T(d) de la tabla si T está definida para d */`

a. Escribir en **C/C++** un módulo de implementación para el TAD Tabla, mediante listas encadenadas, suponiendo que los elementos de `Dom` tienen asociado un orden (se puede comparar mediante <, > e =): ordenadas y no ordenadas.
b. Escribir en **C/C++** un módulo de implementación para el TAD Tabla, mediante la representación clásica de árboles binarios de búsqueda, suponiendo que los elementos de `Dom` tienen asociado un orden.
c. Escribir en **C/C++** un módulo de implementación para el TAD Tabla mediante un vector, donde `Dom` es un subrango 1..N, para algún natural N.
d. Comparar las implementaciones según cantidad de comparaciones (similar al Ejercicio 1 parte
e.) que se realizan para las operaciones.