Estructuras de Datos y algoritmos 1

Práctico 4

Temas: TAD Lista, TAD Pila y TAD Cola

Objetivos

- Familiarizarse con el manejo de estructuras dinámicas de tipos genéricos
- Comprender el manejo de TADs
- Definir, implementar y usar TADs Lineales (Listas, Pilas, Colas)

TAD Lista

1) Resuelva los ejercicios pendientes del teórico de estructuras dinámicas listas (teórico 5) y los ejercicios pendientes de los módulos 7, 8 y 9 (TADs Listas, Pilas, Colas).

Considere la siguiente especificación del TAD Lista:

```
/*********************************************************************************/
#ifndef LISTA_H
#define LISTA_H

#include "Constantes.h"

template <class T>
class Lista {
      friend ostream &operator<< <>(ostream&, Lista<T> &);
public:

      Lista() {}
      Lista(const Lista<T> &l){}
      virtual ~Lista() {}

      // Inserta un elemento e del tipo T al final de la lista
      virtual void Agregar(const T &e) = 0;

      // Inserta un elemento e del tipo T en la posicion p
      // Precondición: 1 <= p <= Largo()+1
      virtual void AgregarEn(const T &e, Posicion p) = 0;

      // Elimina el nodo que referencia al elemento en la posicion p y llama
      // a la funcion "seteada" para ese elemento.
      // Precondición: 1 <= p <= Largo() y la lista no es vacia
      virtual void EliminarDe(Posicion p) = 0;

      // Elimina todas las ocurrecias del elemento e de la lista
```

```
        // Elimina los nodos de la lista que referencian al elemento e y llama
        // a la funcion "seteada" para cada elemento e.
        // Precondición: La lista no es vacia
        virtual void EliminarTodos(const T &e) = 0;

        // Borra toda la lista
        // Para cada elemento llama a la funcion "seteada" para cada elemento e.
        // Precondición: La lista no es vacia
        virtual void BorrarLista() = 0;

        // Retora la cantidad de elementos de la lista
        virtual unsigned int Largo() const = 0;

        // Retorna el elemento que se encuentra en la posicion p
        // Precondición: 1 <= p <= Largo()
        virtual  T& ElementoEn(Posicion p) const = 0;

        // Almacena la función f recibida como parametro
        // Esta función tiene el conocimiento de como eliminar elementos del tipo T
        // Se invocara cada vez que sea necesario eliminar un elemento del tipo T
        virtual void SetearFuncionEliminar(void(*f)(T &)) = 0;

        // Retorna true si las dos listas sn iguales, false en otro caso
        virtual bool operator==(const Lista<T> &l) const = 0;};

#endif
/*****************************************************************************/
```

2) Implementar la especificación del TAD Lista anterior con listas enlazadas.

3) Considere una extensión que permita representar las mismas listas pero de tamaño acotado. Para esto el constructor de la clase recibirá como parámetro el tamaño máximo de elementos que almacenará la lista.

4) Implemente una función

```
template <class T>
bool Iguales(Lista<T> *l1,Lista<T> * l2);
```

que retorna `true` si las listas `l1` y `l2` son iguales, `false` en otro caso.

5) Implemente una función

```
template <class T>
bool IgualConjunto(Lista<T> *l1,Lista<T> * l2);
```

que retorna `true` si las listas `l1` y `l2` tienen los mismos elementos (pueden estar en distinto orden), `false` en otro caso.

TAD Pila

6) Especifique el TAD Pila de elementos genéricos.
7) Construya una implementación acotada eficiente.
8) Construya una implementación dinámica eficiente.

TAD Cola

9) Especifique el TAD Cola de elementos genéricos.
10) Construya una implementación acotada eficiente.
11) Construya una implementación dinámica eficiente.

Problemas de Aplicación

12) El problema de Josefo es el siguiente "juego" de suicidios en masa: n personas, numeradas de 1 a n, están sentadas en círculo. Empezando por la persona 1, se pasa el revólver. Después de m pasadas. la persona que tiene el arma se suicida, se quita el cuerpo, se cierra el círculo y el juego continúa con la persona que estaba sentada después del muerto. El último sobreviviente queda luego de n-1 disparos. Así si m = 0 y n = 5, los jugadores son muertos en órden y el jugador 5 sobrevive. Si m = 1 y n = 5 el órden de las muertes es el siguiente: 2, 4, 1, 5.

13) Una **DobleCola** es una estructura de datos consistente en una lista de elementos sobre la cual son posibles las siguientes operaciones:

    a) encolarPpio(x, d) – Inserta un elemento x en el extremo frontal de la doble cola d
    b) decolarPpio(d) – Elimina y devuelve el elemento que esta al frente de d
    c) encolarFin (x,d) - Inserta un elemento x en el extremo posterior de la doble cola d
    d) decolarFin(d) - Elimina y devuelve el elemento que esta al fondo de d

    Escriba las rutinas necesarias para implementar una doble cola de tal forma que tomen un tiempo O(1) por operación.

14) Se desea modelar el comportamiento de un campeonato en el que los equipos, identificados por (país, nombre) compiten según las siguientes reglas:

    Se mantiene la lista de equipos inscriptos (llamada Preliminar) que deben competir, la fechas de competencia se asignan según el orden en que se inscribieron y compiten de a parejas consecutivas en Preliminar, es decir, competirán el primer equipo con el segundo en inscribirse, el tercero, con el cuarto, y así sucesivamente. Si el numero es impar el equipo restante deberá esperar un nuevo ingreso para competir.

    Cuando dos equipos compiten gana uno y pierde otro. El equipo ganador pasa a la ronda de ganadores y el perdedor vuelve a Preliminar como si recién se inscribiera (es decir, queda al final).

    La ronda de ganadores posee el siguiente funcionamiento. Cuando se ingresa un nuevo equipo se coloca primero en la ronda y las reglas de con las que deberán competir son las siguientes: se toma el primero y el segundo, compiten, el perdedor es eliminado y el ganador vuelve a ser primero en la lista de ganadores hasta que finalmente quede solo un equipo que será el CAMPEON!.

Se pide:

a) Especificación de los TADs encontrados

b) Diseñar las estructuras de datos necesarias para implementar los TADs anteriores y satisfacer las siguientes consultas de la forma más eficiente posible.

c) Implementar las funciones, utilizando solamente las operaciones definidas en el TAD anteriormente:

ListaEquipos contrincantes (Campeonato c)
Dado un equipo deberá retornar la lista de equipos contrincantes.

void altaEquipo (Preliminar P, Equipo E)
Deberá modelar el comportamiento del ingreso de un equipo al campeonato.

void listar equipos (Campeonato c)
Deberá listar los equipos ordenados alfabéticamente.

void preliminares (Preliminar P, RondaGanadores R)
Deberá modelar el comportamiento de las preliminares del campeonato.

Equipo ganadores (RondaGanadores R)
Deberá modelar el comportamiento del campeonato en la ronda de ganadores devolviendo el equipo ganador.