LISTA

- **DEFINICIÓN**

Una lista es una secuencia de elementos de un tipo determinado. Los elementos están ordenados de forma lineal según las posiciones que ocupan. Todos los elementos, salvo el primero, tienen un único predecesor. Y todos los elementos, salvo el último, tienen un único sucesor.

Posición fin: Posición siguiente a la del último elemento. No está ocupada nunca por ningún elemento.

- **ESPECIFICACIÓN**
  - Lista CrearLista()

    Post: Crea y devuelve la lista vacia.
  - Void Insertar(const tElemento& x, posición p)

    Pre: L=(a1,a2…an) 1<= p <= n+1

    Post: L=(a1…,a_{p-1},x,a_p,…,an)
  - Void Eliminar(posición p)

    Pre: L=(a1,a2…an) 1<= p <= n

    Post: L=(a1…,a_{p-1},x,a_p,…,an)
  - tElemento& Elemento(posición p)

    Pre: L=(a1,a2…an) 1<= p <= n

    Post: Devuelve a_p, el element que ocupa la posición o de la lista L.
  - posición Buscar(const tElemento& x) const

    Post: Devuelve la posición de la primera ocurrencia de x en la lista. Si x no está en la lista, devuelve la posición fin().
  - posición Siguiente(posición p) const

    Pre: L=(a1,a2…an) 1<= p <= n

    Post: Devuelve la posición que sigue a p.

- posición Anterior(posición p) const
  Pre: L=(a1,a2…an) 1<= p <= n
  Post: Devuelve la posición anterior a p.
- posición Primera() const
  Post: Devuelve la primera posición de la lista. Si la lista está vacía, devuelve la posición fin().
- posición fin() const
  Post: Devuelve la última posición de la lista, la siguiente a la del último elemento. Esta posición siempre está vacía, no existe ningún elemento que la ocupe.

- **IMPLEMENTACIÓN VECTORIAL PSEUDOESTÁTICA**

Los elementos se pueden almacenar en celdas contiguas de una matriz o vector de un tamaño dado. Además, será necesario guardar la longitud que tenga la lista en cada momento, o bien el índice de la celda que contenga el último elemento. La i-ésima posición está representada por

i-1 debido a que en la posición fin no se almacena nada.

Permite acceso directo a elementos(Buscar()) a través del índice. Para insertar o eliminar es necesario desplazar todos los elementos(costoso en función del número).

Exige especificar el tamaño máximo de la lista (INCONVENIENTE)

Función fin() → orden constante

Función Eliminar() → orden lineal

Función Insertar() → orden lineal

Función Buscar() → orden lineal

- **IMPLEMENTACIÓN CELDAS ENLAZADAS**

Consiste en una secuencia enlazada de nodos que almacenan elementos consecutivos. Cada nodo almacena un elemento y un puntero al nodo que contiene el siguiente elemento. El nodo que contiene el último elemento posee un puntero nulo.

Cuando se realice alguna operación que modifique el primer elemento, será necesario cambiar ese puntero inicial para que apunte al nuevo primer elemento.

PUNTERO PASADO POR REFERENCIA A OPERACIONES DEL TAD QUE LO MODIFIQUEN.

Usamos otro puntero que apunte a este.

LISTA= PUNTERO A PUNTERO AL PRIMER ELEMENTO

LISTA VACIA= PUNTERO NULL

No se reserva espacio de memoria de tamaño fijo y se evita el desplazamiento de elemento al insertar/eliminar. Ocupa espacio para almacenar solo los punteros.

POSICIÓN ELEMENTO= PUNTERO AL NODO QUE LO CONTIENE

Las inserciones y eliminaciones en la primera posición de la lista hay que tratarlas de forma especial, ya que el primer nodo no tiene ninguno que le preceda.

Por tanto las funciones insertar y eliminar cambian:

- Void Insertar(const tElemento& x, posición& p)
- Void Eliminar (posición& p)

- **IMPLEMENTACIÓN LISTA ENLAZADA CON NODO CABECERA**

Surge para resolver el problema de la dependencia de la representación.

La posición del i-ésimo elemento es un puntero al i-ésimo nodo. Con esto, también conseguimos no tener que recorrer la lista para situar un puntero en el nodo anterior, al insertar y eliminar elementos, pues ahora la posición de un elemento es un puntero.

El hecho de que el primer nodo de la lista no tenga predecesor obliga a un tratamiento especial de las inserciones y supresiones de la primera posición, por ello, colocamos un **nodo cabecera** al principio de la lista en el que no almacenamos ningún elemento, sólo representa la posición del primer elemento.

Con este nodo, conseguimos que todo elemento tenga un predecesor, por lo que las inserciones y eliminaciones se harían siempre igual.

Cuando sólo hay un puntero al nodo cabecera, el parámetro puede pasarse por VALOR, por tanto las especificaciones no cambian.

Función fin() → orden lineal

Función Eliminar() → orden constante

Función Insertar() → orden constante

Función Buscar() → orden lineal

Función Anterior() → orden lineal

Función Primera() →orden constante

- **IMPLEMENTACIÓN LISTA DOBLEMENTE ENLAZADA**

Cada nodo tiene dos enlaces, uno al nodo anterior y otro al siguiente. Son útiles cuando es necesario recorrer eficientemente una lista, tanto hacia delante como hacia atrás, o también, cuando se quiere que la operación Anterior() sea rápida.

Función fin() → orden constante

Función Eliminar() → orden constante

Función Insertar() → orden constante

Función Buscar() → orden lineal

Función Anterior() → orden constante

Función Primera() →orden constante

ELIMINACIÓN LISTA DOBLEMENTE ENLAZADA

Inconvenientes:

- La cabecera de la función no se corresponde con la especificación del TAD.

- Casos especiales: Eliminar el primer nodo, eliminar el último nodo

MODIFICACIONES PARA EVITAR CASOS ESPECIALES

- Introducir un nodo cabecera
- Posición elemento=Puntero a nodo anterior

Aún así, el último elemento se elimina de forma diferente.

- **IMPLEMENTACIÓN LISTA CIRCULAR**

Para evitar tratar la eliminación del último elemento como caso especial, se usa la lista circular.

El campo siguiente del último nodo apunta a la cabecera y el campo anterior, al último nodo.

Función fin() → orden lineal

Función Eliminar() → orden constante

Función Insertar() → orden constante

Función Buscar() → orden lineal

Función Anterior() → orden lineal

Función Primera() →orden constante

- **PREGUNTAS**

  - Si los elementos de una lista están ordenados, podemos asegurar un coste lineal en la búsqueda, y el tiempo en caso promedio será mejor que cuando no están ordenados.

    FALSO

  - No es posible realizar búsquedas en una lista en orden logarítmico, aunque los elementos de la lista estén ordenados.

    VERDADERO

  - A la hora de implementar las colas con prioridad, es posible resolverlo con colas y con listas, pero obviamente con colas es más eficiente.

    FALSO

  - La operación fin() del TAD Lista es de orden constante en la representación vectorial, igual que en la enlazada, obviamente si tenemos un puntero apuntando allí.

    VERDADERO

  - La operación fin() del TAD Lista es de orden lineal en la representación vectorial, frente a coste constante en la representación enlazada, obviamente si tenemos un puntero apuntando allí.

    FALSO

  - En el TAD Lista, el coste de la operación anterior() es lineal si utilizamos la representación simplemente enlazada con nodo cabecera.

    VERDADERO

  - En el TAD Lista circular, el coste de la operación fin() está en O(1).

    FALSO, no existe operación fin

  - Es posible realizar búsquedas en una lista en orden logarítmico, pero exigen que los elementos de la lista estén ordenados.

    FALSO

  - En el TAD Lista no existe ninguna forma de implementar la operación fin() en coste de O(1) en ninguna representación enlazada.

    FALSO