Algoritmos y Estructuras de Datos

Guía Práctica 6
Diseño con estructuras simples
Primer Cuatrimestre 2025

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1. Implementamos el TAD Secuencia sobre una **lista simplemente enlazada** usando

```
NodoLista<T> es struct<
    valor: T,
    siguiente: NodoLista<T>,
 >

 Módulo ListaEnlazada<T> implementa Secuencia⟨T⟩ <
      var primero: NodoLista<T> // "puntero" al primer elemento
      var último: NodoLista<T> // "puntero" al primer elemento
      var longitud: int  // cantidad total de elementos

 proc NuevaListaVacía ( ): ListaEnlazada<T>
    res := new ListaEnlazada⟨T⟩
    res.primero := null
    res.último := null
    res.longitud := 0
    return res


 proc AgregarAdelante (inout l: ListaEnlazada<T>, in e: T):
    nodo := new NodoLista⟨T⟩
    nodo.value := e
    nodo.siguiente := null

    if L.longitud == 0 then
       l.primero := nodo
       l.último := nodo
    else
       nodo.siguiente := l.primero
       l.primero := nodo
    end if
    l.longitud := l.longitud + 1


 proc Pertenece (in l: ListaEnlazada<T>, in e: T): bool
    res := false
    actual := l.primero
    while actual ≠ null do
       if actual.valor == e then
          res := true
       end if
       actual := actual.siguiente
    end while
    return res

 >
```

- Escriba los algoritmos para los siguientes procs y calcule su complejidad
  - `proc agregarAtras`(inout l : ListaEnlazada$\langle T\rangle$, in $t : T$)
  - `proc obtener`(in l : ListaEnlazada$\langle T\rangle$, in $i : \mathbb{Z}$) $: T$
  - `proc eliminar`(inout l : ListaEnlazada$\langle T\rangle$, in $i : \mathbb{Z}$)
  - `proc concatenar`(inout $l1$: ListaEnlazada$\langle T\rangle$, in $l2$: ListaEnlazada$\langle T\rangle$)
- Escriba el invariante de representación para este módulo en castellano

- Dado el siguiente invariante de representación, indique si es correcto. En caso de no serlo, corrijalo:

  `pred InvRep` (l: ListaEnlazada$\langle T \rangle$) {

  $accesible(l.primero, l.ultimo) \land largoOK(l.primero, l.longitud)$

  }

  `pred largoOK` (n: NodoLista$\langle T \rangle$, $largo$: $\mathbb{Z}$) {

  $(n = null \land largo = 0) \lor (largoOK(n.siguiente, largo - 1))$

  }

  `pred accesible` (n_0: NodoLista$\langle T \rangle$, n_1: NodoLista$\langle T \rangle$) {

  $n_1 = n_0 \lor (n_0.siguiente \neq null \land_L accesible(n_0.siguiente, n_1))$

  }

Ejercicio 2. Implemente el TAD `ConjuntoAcotado<T>` (definido en el apunte de TADs) usando la siguiente estructura.

```
Módulo ConjuntoArr<T> implementa ConjuntoAcotado<T> <
      var datos: Array<T>
      var tamaño: int
>
```

- Escriba el invariante de representación y la función de abstracción.
- Escriba los algoritmos para las operaciones `conjVacío` y `pertence`
- Escriba el algoritmo para la operación `agregar`
- Escriba los algoritmos para las operaciones `unir` e `intersecar`.
- Escriba el algoritmo para la operación `sacar`.
- Calcule la complejidad de cada una de estas operaciones
- Qué cambios haría en su implementación si se quiere que la operación `agregar` sea lo más rápida posible? Y si se quiere acelerar la operación `buscar`? Indique los cambios en la estructura, el invariante de representación, la función de abstracción y los algoritmos.

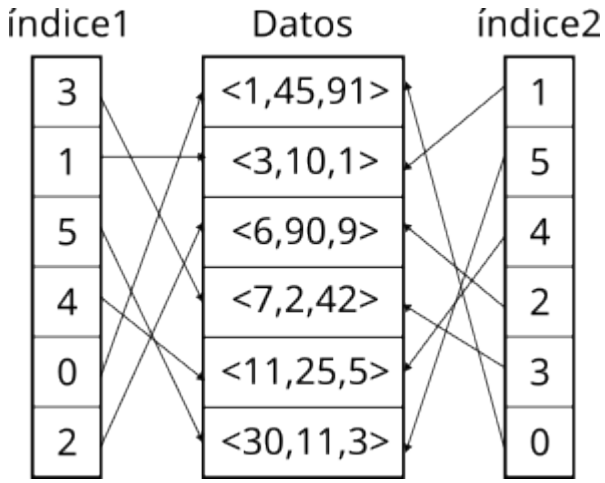
Ejercicio 3. Implementar el TAD Conjunto$\langle T \rangle$ (definido en el apunte de TADs) usando la siguiente estructura

```
Módulo ConjuntoLista<T> implementa Conjunto⟨T⟩ <
      var datos: ListaEnlazada<T>
      var tamaño: int
>
```

- Escriba el invariante de representación y la función de abstracción.
- Escriba los algoritmos para las operaciones `conjVacío` y `pertence`
- Escriba el algoritmo para la operación `agregar`, `agregarRápido` y `sacar`
- Escriba los algoritmos para las operaciones `unir` e `intersecar`.
- Calcule la complejidad de cada una de estas operaciones

Ejercicio 4. Un *índice* es una estructura secundaria que permite acceder más rápidamente a los datos a partir de un determinado criterio. Básicamente un índice guarda *posiciones* o *punteros* a los elementos en un orden en particular, diferente al orden original.

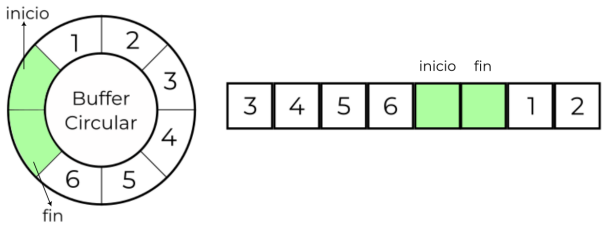
Imagine una secuencia de tuplas con varias componentes, ordenada por su primer componente. Algunas veces vamos a querer buscar (rápido) por las demás componentes. Podríamos guardar los datos en sí en un arreglo y tener arreglos con las posiciones ordenadas por las demás componentes. A estos arreglos se los denomina índices.



En la figura, si recorremos los datos en el orden en el que están guardados, obtenemos:
`[<1, 45, 91>, <3, 10, 1>, <6, 90, 9>, <7, 2, 42>, <11, 25, 5>, <30, 11, 3>]`
Si lo recorremos usando el índice 1 (que apunta a los elementos en función de la segunda componente) obtenemos:
`[<7, 2, 42>, <3, 10, 1>, <30, 11, 3>, <11, 25, 5>, <1, 45, 91>, <6, 90, 9>]`

- Escriba la estructura propuesta
- Escriba el invariante de representación y la función de abstracción, en castellano y en lógica para el TAD $\mathsf{Conjunto}\langle\mathsf{Tupla}\langle\mathbb{Z},\mathbb{Z},\mathbb{Z}\rangle\rangle$
- Escriba el algoritmo de `BuscarPor` que busca por alguna componente
- Escriba los algoritmos de `agregar` y `sacar`

Ejercicio 5. Una forma eficiente de implementar el TAD Cola en su versión acotada (con una cantidad máxima de elementos predefinida), es mediante un *buffer circular*. Esta estructura está formada por un array del tamaño máximo de la cola (n) y dos índices (*inicio* y fin), para indicar adonde empieza y adonde termina la cola. El chiste de esta estructura es que, al llegar al final del arreglo, si los elementos del principio ya fueron consumidos, se puede reusar dichas posiciones.



- Elija una estructura de representación
- Escriba el invariante de representación y la función de abstracción
- Escriba los algoritmos de las operaciones `encolar` y `desencolar`
- ¿Por qué tiene sentido utilizar un buffer circular para una cola y no para una pila?

Ejercicio 6. Implementar los siguientes TADs (cuyas especificaciones están en el apunte) sobre arreglo y sobre lista enlazada. Calcule las complejidades de las operaciones

- `Pila<T>`
- `Diccionario<K,V>`