Estructuras de datos — Solemne 2

Profesores: Luis Bastías, Rodrigo Paredes, Iván Zuñiga
Ayudantes: Patricia Albornoz, Francisco Claude, Hans Ulloa

Sin apuntes, 1:30 horas

P1. Transpose Lists

Utilizando listas dinámicas simples, implemente el TDA Transpose List. Las operaciones del TDA son las siguientes:

a. (2 puntos) **Insert**(x): inserta el elemento x al comienzo de la lista.

b. (2 puntos) **Delete**(x): borra al elemento x de la lista (si es que está).

c. (2 puntos) **Find**(x): busca el elemento x en la lista. Si la búsqueda es exitosa intercambia el elemento con el anterior. Nota: si x es el primer elemento no cambia de posición.

Respuesta

Utilizando listas con un nodo cabeza ficticia *head*. Pseudocodigo a la C++.

---

void **Insert** (Elem x)
1. $head \rightarrow sig$ = new Nodo(x, $head \rightarrow sig$)

Nodo ***Prev** (Elem x)
1. Nodo $*aux = head$
2. **While** $aux \rightarrow sig$!= NULL **Do**
3. **If** $aux \rightarrow sig \rightarrow info == x$ **Then Return** aux // lo encontré
4. **Else** $aux = aux \rightarrow sig$
5. **Return** NULL // x no está en la lista

---

void **Delete** (Elem x)

1. Nodo *$auxP$ = **Prev**(x) // $auxP$ apunta al anterior de x
2. **If** $auxP$ == NULL **Then Return** // x no está en la lista, nada que hacer
3. Nodo *$aux = auxP \rightarrow sig$ // aux apunta al nodo que contiene x
4. $auxP \rightarrow sig = aux \rightarrow sig$ // descuelgo nodo x
5. delete(aux) // libero la memoria del nodo x

Nodo ***Prev2** (Elem x)

1. **If** $head \rightarrow sig$!= NULL AND $head \rightarrow sig \rightarrow info == x$ **Then**
2. **Return** NULL // es el primero de la lista
3. Nodo *$aux = head$
4. **While** $aux \rightarrow sig \rightarrow sig$!= NULL **Do**
5. **If** $aux \rightarrow sig \rightarrow sig \rightarrow info == x$ **Then Return** aux // lo encontré
6. **Else** $aux = aux \rightarrow sig$
7. **Return** NULL // x no está en la lista

---

boolean **Find** (Elem x)

1. Nodo *$auxP$ = **Prev**(x) // $auxP$ apunta al anterior de x
2. **If** $auxP$ == NULL **Then Return** FALSE // x no está en la lista, nada que hacer
3. **If** $auxP == head$ **Then Return** TRUE // x está al comienzo de la lista, fin
4. Nodo *$aux = auxP \rightarrow sig$ // aux apunta al nodo que contiene x
5. Nodo *$auxP2$ = **Prev2**(x) // $auxP2$ apunta al anterior del anterior de x
6. $auxP2 \rightarrow sig = aux$
7. $auxP \rightarrow sig = aux \rightarrow sig$
8. $aux \rightarrow sig = auxP$
9. **Return** TRUE

Soluciones alternativas de **Delete** y **Find** sin **prev** ni **prev2**.

boolean ***Delete** (Elem x)

1. **If** $head \rightarrow sig$ == NULL **Then** Return FALSE // lista vacia, nada que hacer
2. Nodo *$aux = head$
3. **While** $aux \rightarrow sig$!= NULL **Do**
4. **If** $aux \rightarrow sig \rightarrow info == x$ **Then** // lo encontré
5. Nodo *$aux2 = aux \rightarrow sig$ // $aux2$ apunta al nodo que contiene x

```
6.          aux → sig = aux2 → sig // descuelgo al nodo x
7.          delete(aux2) // libero la memoria del nodo x
8.          Return TRUE
9.      Else aux = aux → sig
10. Return FALSE  // x no está en la lista
```

```
boolean Find (Elem x)
1.  If head → sig == NULL Then Return FALSE // lista vacia, nada que hacer
2.  If head → sig → info == x Then
3.      Return TRUE // x está al comienzo de la lista, fin
4.  Nodo *aux = head
5.  While aux → sig → sig != NULL Do
6.      If aux → sig → sig → info == x Then // lo encontré
7.          Nodo *aux2 = aux → sig, *aux3 = aux2 → sig
            // aux2 apunta al anterior de x, aux3 apunta a x
8.          aux → sig = aux3, aux2 → sig = aux3 → sig, aux3 → sig = aux2
9.          Return TRUE
10.     Else aux = aux → sig
11. Return FALSE // x no está en la lista
```

P2. Cape nane

Las sectas secretas tienen raros ritos, tanto para iniciar a sus integrantes como para elegir a sus representantes. La célebre secta de los "Cape nane" tiene un rito específico para la designación de sus dirigentes, y consiste en lo siguiente:

Se reunen los integrantes en torno a un círculo y partiendo desde el presidente actual comienzan a orar repitiendo la siguiente frase sagrada: *Ca-pe na-ne-nú e-ne te-ne-tú sa-lis-te tú*. Cada uno de los integrantes pronuncia una sílaba de la oración y el que pronuncia la última sílaba (el último "tu") sale del círculo. Luego de varias repeticiones de la frase sagrada el círculo se va achicanndo hasta que al final queda sólo un integrante que es nombrado como Presidente por el siguiente periodo.

Al comienzo de la ceremonia se lanza una moneda al aire, si es que sale cara, comienzan a repetir en el sentido de las agujas del reloj; si sale sello, en contra de las agujas.

Considere que el presidente actual esta en la posición 0. Suponga también que existe el TDA Moneda, que tiene la función lanzar que responde cara o sello (al azar).

a. (2 puntos) Se le solicita que defina una estructura de datos apropiada para modelar el problema de la seleción del nuevo presidente.

b. (4 puntos) Se le solicita que escriba un algoritmo que imprima las posiciones en que se eliminan candidatos en orden de eliminación. Considere la moneda como un argumento de la función.

Respuesta

Parte a. Como estructura de datos, basta con una lista circular doble enlazada, en donde cada nodo de la lista almacena a uno de los integrantes de la secta. El header de la lista apunta a la celda que contiene al presidente actual.

Parte b. Vamos a asumir que los integantes ya están en la lista, y que tenemos disponibles las operaciones típicas de listas doble enlazadas. También vamos a asumir que el resultado de la moneda es fijo (CARA o SELLO durante todo el desarrollo del algoritmo), y si dice CARA seguimos el puntero *sig*, si dice SELLO seguimos el puntero *prev*.

Notemos que la frase sagrada contiene 14 sílabas. Pseudocódigo a la C++.

---

void **Extract** (Nodo *n)

```
    // este es un método de la clase ListaDobleEnlaceCircular
1.  If head == NULL Then Return // lista vacia, nada que hacer
2.  If head → sig == head Then // queda un elemento en la lista
3.      If n == head Then // y es el elemento que vamos a borrar
4.          head = NULL, delete(n) // la lista quedó vacia
5.      Return // termina este caso
6.  // tenemos más de 1 elemento en la lista, descolgamos n de la lista
7.  n → sig → prev = n → prev, n → prev → sig = n → sig
    // OJO si vamos a borrar lo apuntado por head, movemos a head
8.  If n == head Then head = head → sig
9.  delete(n) // liberamos el nodo
```

---

void **EleccionPresidente** (ListaDobleEnlaceCircular *secta*, Moneda m)

```
1.  Nodo * aux = secta.head // se parte de la posición del presidente actual
2.  Nodo * aux2; // usamos aux2 para marcar el siguiente en la cuenta
```

```
3.  While secta.head != NULL Do
4.      // son 14 sílabas, es decir, 13 saltos
5.      If m == CARA Then
6.          For i = 1...13 Do aux2 = aux → sig // oración
7.          aux2 = aux → sig // guardo aux2 el siguiente en la cuenta
8.      If m == SELLO Then
9.          For i = 1...13 Do aux2 = aux → prev // oración
10.         aux2 = aux → prev // guardo aux2 el siguiente en la cuenta
11.     printf("%s ", aux → nombre) // imprimo al candidato eliminado
12.     secta.Extract(aux) // y lo extraigo
13.     aux = aux2 // actualizo aux para seguir con la cuenta
```

Notemos que el último candidato eliminado es el próximo presidente de la secta.

P3. Treesort

Considere que tiene una colección de n numeros, con elementos repetidos. Considere que en total son c clases de distintos números. Por ejemplo, en la colección {1 5 2 4 6 1 3 2 3 6 4 2 1 6 3}, se tienen 15 números y hay 6 clases distintas.

a. (4.5 puntos) Proponga un algoritmo que permita ordenar la colección de números utilizando un árbol de búsqueda binaria en tiempo $O(n \log c)$ en promedio.

b. (1.5 puntos) Justifique el costo.

Respuesta

Parte a. Vamos a usar un ABB en que los nodos tienen los campos {Item $info$, int rep, Nodo izq, Nodo der}. Los nodos están ordenados según $info$, que para estos efectos, sin perder generalidad, vamos a suponer que son enteros.

Lo importante acá va a ser modificar el método de inserción en un ABB. Esto es, cuando insertamos un elemento, si es nuevo lo marcamos con una repetición, pero si es antiguo, le incrementamos el contador.

Entonces, primero viene el método de inserción, y luego el que ordena los elementos en con el árbol.

```
ABB *Insert (ABB *raiz, Item x)
    // version recursiva
1.  If raiz == NULL Then raiz = new Nodo(x,1,NULL,NULL)
2.  Else If raiz → info == x Then raiz → rep = raiz → rep + 1
3.  Else If raiz → info > x Then raiz → izq = Insert(raiz → izq, x)
4.  Else raiz → der = Insert(raiz → der, x)
5.  Return raiz
```

```
ABB *Insert (ABB *raiz, Item x)
    // version no recursiva
1.  If raiz = NULL Then Return raiz = new Nodo(x,1,NULL,NULL)
2.  aux = raiz
3.  While aux != NULL Do
4.      If aux → info == x Then aux → rep = aux → rep + 1, Return
5.      Else If aux → info > x Then aux = aux → izq
6.      Else aux = aux → der
7.  aux = new Nodo(x,1,NULL,NULL)
```

```
void Ordena (Lista l)
1.  ABB *a
2.  For i = 0 . . . l.lenght()-1 Do
3.      a = Insert(a, l[i])
```

Parte b. El costo del algoritmo es $O(n \log c)$ en promedio por lo siguiente. Como el ABB tiene c nodos, uno por cada clase, cada inserción tiene costo $O(\log c)$ en promedio. Se realizan n inserciones en este ABB luego el costo total es $O(n \log c)$ en promedio.