Programario: Unidad 2 - Tema 2

Said Polanco-Martagón

April 8, 2017

1 Instrucciones

Siga las instrucciones descritas en el **Programario 1**.

2 Problemas

1. Escribir la función `Count()` que cuente el número de veces que un *int* ocurre en una lista enlazada. Pruebe su resultado.

2. Escriba la función `GetNth()` que toma como entrada una lista enlazada y un entero; esta función retorna el valor contenido en la i-esima posición. `GetTh()` utiliza el estándar de numeración de C en el cual se inicia a contar en 0. Si la lista enlazada no contiene datos, implemente una estrategia de manejo de errores.

   ```
   void GetNthTest() {
   struct node* myList = ListaEnlazada();
   int lastNode = GetNth(myList, 2);
   }
   ```

   Nota implemente las funciones de: insertar, borrar, mostrar de la lista enlazada.

3. Implemente la función recursiva `DeleteList()` que toma una lista enlazada, desasigna toda su memoria y establece su raíz en `NULL`.

   ```
   void DeleteListTest() {
   struct node* myList = ListaEnlazada();
   // Crea la lista R->1-> 2->3
   DeleteList(&myList); /* borra los nodos y establece la lista en NULL*/
   }
   ```

4. Implemente la clase `Lista`. El comportamiento de dicha lista dependerá de la opción ingresada en el constructor

   - `PILA`: Los métodos `push()` y `pop()` se comportarán como un pila.
   - `COLA`: Los métodos `push()` y `pop()` se comportarán como un cola.

5. Implemente una función InsertNth() que puede insertar un nuevo nodo en cualquier índice dentro de una lista. Si el índice no existe, rellene los elementos intermedios al índice.

```
void InsertNthTest() {
struct node* head = NULL;
InsertNth(&head, 0, 13); //R->13
InsertNth(&head, 1, 42); //R->13->42
InsertNth(&head, 4, 5);  // R->13->42->null->null->5
InsertNth(&head, 1, 9); // R->13->9->42->null->null->5
DeleteList(&head);
}
```

6. Escriba la función `SortedInsert()` que, dada una lista se ordenará en orden creciente. El nodo se inserta en la posición ordenada correcta en la lista.

```
void SortedInsert(struct node** headRef, struct node* newNode) {
// Your code...
```

7. Escribe una función `InsertSort()` que, dada una lista, reorganiza sus nodos para que se ordenen en orden creciente.

8. Escribe la función `Append()` que toma dos listas, 'a' y 'b', agrega 'b' al final de 'a' y luego establece 'b' en NULL (ya que ahora se está arrastando al final de 'a').

9. Implemente una función que divida en dos sublistas una lista enlazada dada, una para la mitad delantera y otra para la mitad posterior. Si el número de elementos es impar, el elemento extra debe ir en la lista frontal.

10. Retome el problema anterior. Pero esta vez, antes de separarla, ésta deberá estar ordenada.

11. Escribir función `RemoveDuplicates()` que toma una lista ordenada en orden creciente y elimina cualquier nodo duplicado de la lista. La lista sólo debe ser recorrida una vez.

12. Esta es una variante de `Push()`. En lugar de crear un nuevo nodo y empujarlo en la lista dada, `MoveNode()` toma dos listas, elimina el nodo frontal de la segunda lista y lo empuja al frente de la primera.

```
void MoveNodeTest() {
struct node* a = ListaUnoDosTres(); // the list {1, 2, 3}
struct node* b = ListaUnoDosTres();

MoveNode(&a, &b);
// a == {1, 1, 2, 3}
// b == {2, 3}
}
```

13. Implemente la función `AlternatingSplit()` que toma una lista y divide sus nodos para hacer dos listas más pequeñas. Las sublistas deben estar hechas de elementos alternantes de la lista original. Así que si la lista original es a, b, a, b, a, entonces una sublista debería ser a, a, a y la otra debería ser b, b.

14. Dadas dos listas, fusionar sus nodos juntos para hacer una lista, tomando nodos alternativamente entre las dos listas. Así que `ShuffleMerge()` con 1, 2, 3 y 7, 13, 1 será 1, 7, 2, 13, 3, 1. Si se agota cualquiera de las listas, todos los nodos deben tomarse de la otra lista.

15. Implemente la función `SortedMerge()` que tome un vector de raices de listas enlazadas, cada una de las cuales se debe ordenar en orden creciente. `SortedMerge()` combina las listas en una lista que está en orden creciente.

16. Implemente la función `inversa()` que invierte una lista mediante la reorganización de todos los punteros y raíz de la lista. Se deberá visitar cada uno sólo una vez.

17. Retome el problema anterior. E implemente la solución recursiva.