Tecnológico de Monterrey

Act 2.1 - Linear data structure ADT execution

Liliana Solórzano Pérez A01641392

25 de Septiembre del 2022

Programming of Data Structures and Fundamental Algorithms

(Gpo 613)

Profesores

Jorge Enrique González Zapata

Luis Ricardo Peña Llamas

Code

```
//Created by Liliana Solórzano Pérez A01641392
#include <iostream>
#include <stdio.h>
#include <cstdlib>



using namespace std;

//----------------Function to create a new node----------------
struct Node{
   int data;
   Node *next;
};

//----------------Function to check if the list is empty----------------
bool isEmpty(Node *head){
   if (head == NULL)
       return true;
   else
       return false;

//----------------------Complexity O(1)--------------------------------
}



char menu(){
   char choice;
   cout << endl;
   cout << " ++++ Menu ++++ " << endl;
   cout << "1. Insert an element" << endl;
   cout << "2. Delete an element by position" << endl;
   cout << "3. Sort the list" << endl;
   cout << "4. Update an element by position" << endl;
```

```
    cout << "5. Search an element by position" << endl;
    cout << "6. Show List" << endl;
    cout << "7. Exit " << endl;
    cout << "Please choose an option from the menu: ";
    cin >> choice;
    return choice;

}


void insertAsFirtsElement(Node *&head, Node *&last, int number){
    Node *temp = new Node();
    temp->data = number;
    temp->next = NULL;
    head = temp;
    last = temp;

//-----------------------Complexity O(1) ----------------------

}

void insert(Node *&head, Node *&last, int number){
    if(isEmpty(head)){
        insertAsFirtsElement(head, last, number);
    }
    else{
        Node *temp = new Node();
        temp->data = number; //We store the data in the new node
        temp->next = NULL; //We connect the new node with the last node
        last->next = temp; //We connect the last node with the new node
        last = temp;
    }

//-----------------------Complexity O(1) ----------------------
```

```
}

void deleteElement(Node *&head, Node *&last){
   if (isEmpty(head)){
       cout << "The list is already empty" << endl;
   }
   else if (head == last){
       delete head;
       head = NULL;
       last = NULL;
   }
   else{
       Node *temp = head; //We create a temporal node to store the head
       head = head->next; //We move the head to the next node
       delete temp;
   }

//-----------------------Complexity O(1) ----------------------

}

void DeleteElementByPos(Node** head_ref, int position){
   //Check if the list is empty

       if (*head_ref == NULL) {
       return;
   }
   struct Node* temp = *head_ref;
   if (position == 1) {
       *head_ref = temp->next;
       free(temp);
       return;
   }
   for (int i = 2; temp != NULL && i < position - 1; i++) {
       temp = temp->next;
```

```
    }
    if (temp == NULL || temp->next == NULL) {
        return;
    }
    cout << "The element has been deleted" << endl;
    struct Node *next = temp->next->next;
    free(temp->next);
    temp->next = next;



}

// Sort the linked list with bubble sort
void sortLinkedList(struct Node** head_ref) {

    struct Node *current = *head_ref, *index = NULL;
    int temp;

    if (head_ref == NULL) {
        return;
    } else {
        while (current != NULL) {
        // index points to the node next to current
        index = current->next;

        while (index != NULL) {
            if (current->data > index->data) {
            temp = current->data;
            current->data = index->data;
            index->data = temp;
            }
            index = index->next;
        }
        current = current->next;
```

```
        }
        cout << "The list has been sorted" << endl;
    }
//----------------------Complexity O(n^2) ----------------------
}

void SearchElementByPos(Node *head){
    if (isEmpty(head)){
        cout << "The list is empty" << endl;
    }
    int n = head->data;
    int pos, i;
    cout << "Enter the position of the element you want to search: ";
    cin >> pos;
    if (pos > n){
        cout << "The position is not valid" << endl;
    }
    else{
        Node *temp = head;
        for (i = 0; i < pos; i++){
            temp = temp->next;
        }
        cout << "The element is: " << temp->data << endl;
    }

//------------------- Complexity of the search O(1)---------------
}

void updateElementByPos(Node *head){
    if (isEmpty(head)){
        cout << "The list is empty" << endl;
    }
    int n = head->data;
    int pos, i;
    cout << "Enter the position of the element you want to update: ";
```

```
    cin >> pos;
    if (pos > n){
        cout << "The position is not valid" << endl;
    }
    else{
        Node *temp = head;
        for (i = 0; i < pos; i++){
            temp = temp->next;
        }
        cout << "The element is: " << temp->data << endl;
        cout << "Enter the new value: ";
        cin >> temp->data;
        cout << "The element has been updated" << endl;
    }
//----------------------Complexity of the update O(1)----------------
}



void ShowList(Node *current){
    if (isEmpty(current)){
        cout << "The list is empty" << endl;
    }
    else{
        cout << "The list contains: \n";
        while (current != NULL){

            cout << current->data << " -> "; //We print the data of the
current node
            current = current->next; //We move the current node to the next
node

        }

    }
```

```
}


int main(){
   Node *head = NULL;
   Node *last = NULL;
   char choice;
   int number;

   cout << "Welcome to the linked list program" << endl;


   do{
       cout << endl;
       choice = menu();
       cout << endl;
       switch(choice){
           case '1':
               cout << "Please enter a number to add to the list: ";
               cin >> number;
               insert(head, last, number);
               cout << "The number has been added to the list" << endl;
               cout << endl;
               break;

           case '2':
               int pos;
               if (isEmpty(head)){
                   cout << "The list is already empty" << endl;
               }
               else

               cout << "Enter the position of the element you want to
delete: ";
```

```
            cin >> pos;
            DeleteElementByPos(&head, pos);

            cout << endl;
            break;

        case '3':
            sortLinkedList(&head);
            cout << endl;
            break;

        case '4':
            updateElementByPos(head);
            cout << endl;
            break;

        case '5':
            SearchElementByPos(head);
            cout << endl;
            break;

        case '6':
            ShowList(head);
            cout << endl;
            break;

        default:
            cout << "System is shutting down" << endl;
            break;


    }
}while(choice != '7');

return 0;}
```

Las funciones implementadas y su complejidad

Value: Dato que contendrá el nodo, string, number, boolean, etc.

Head: Referencia al primer nodo de la lista.

Tail: Referencia al último nodo de la lista.

Next: Referencia de un nodo al siguiente nodo.

Prev: Referencia de un nodo al nodo anterior de la lista.

Checar si la lista se encuentra vacía:

Se crea una función booleana donde se compara si head == NULL para comprobar si esta esta vacía, en caso de serlo, regresa verdadero, de lo contrario regresa falso.

Complejidad: O(1)

Insertar un nodo en una lista vacía:

Se inserta el nodo “A” en la lista vacía. Ahora el nodo “A” será el head y tail al mismo tiempo y su next y prev serán nulos ya que no hay más nodos.

Complejidad: O(1)

Eliminar un nodo de una lista doblemente ligada:

Para esto se debe de crear una variabl temporal que será la que va a almacenar el head de nuestra linked list. Una vez esto realizado, podemos eliminar el nodo que queremos.

Complejidad: O(1)

Citas en Formato APA