Actividad 2.2: Verificación de las funcionalidades de una estructura de datos lineal (listas doblemente ligadas)

18 de octubre del 2023

TC1031: Programación de estructuras de datos y algoritmos fundamentales

Documentación

Dulce Nahomi Bucio Rivas A01425284

Profesor

Dr. Alberto González Sánchez

Add

```
// Método para agregar un nodo al final de la lista
template <class T>
void DoubleLinkList<T>::add(T *value)
{
    Node<T> *node = new Node<T>(value); // Creación del nodo
    if (first == NULL)
    {
        first = node; // Verifica que la lista esté vacía, en caso de estarlo, agrega el nuevo nodo al inicio
    }
    else
    {
        // Si la lista no está vacía, agrega el nuevo nodo al final de la lista
        Node<T> *ultimate = getLast(); // Para esto, primero obtiene el último nodo de la lista
        ultimate->setSig(node); // Agrega el nuevo nodo al final de la lista asignandolo como siguiente al último nodo
        node->setAnt(ultimate); // Asigna el último nodo como anterior al nuevo nodo para mantener la lista doblemente
    }
}
```

Display

```
// Método para mostrar la lista
template <class T>
void DoubleLinkList<T>::display() {
    Node<T> *p = getFirst(); // Obtiene el primer nodo de la lista
    string outputStr = "";
    while (p != NULL){ // Recorre toda la lista
        stringstream ss;
        ss << *(p->getInfo()); // Convierte el valor del nodo a string
        outputStr += ss.str() + " "; // Concatena el valor del nodo a la cadena de salida
        p = p->getSig();
    }
    cout << outputStr << endl;
}
```

InverseDisplay

```
// Método para mostrar la lista en orden inverso
template <class T>
void DoubleLinkList<T>::inverseDisplay() {
    // Usa la misma lógica que el método display, pero en este caso, recorre la lista de atrás hacia adelante con
    // el uso del método getAnt
    Node<T> *p = getLast();
    string outputStr = "";
    while (p != NULL){
        stringstream ss;
        ss << *(p->getInfo());
        outputStr += ss.str() + " ";
        p = p->getAnt();
    }
    cout << outputStr << endl;
}
```

Update

```cpp
// Método para actualizar el valor de un nodo en la lista con el uso de plantillas por un índice
template <class T>
void DoubleLinkList<T>::update(int index, T *value) {
    Node<T> *node = getFirst(); // Obtiene el primer nodo de la lista
    int contador = 0; // Crea un contador que se inicializa en 0
    while (node != NULL){
        if (contador == index){
            node->setInfo(value); // Si el contador es igual al índice, asigna el nuevo valor al nodo
            return;
        }
        contador++;
        node = node->getSig();
    }
}
```

Remove

```cpp
// Método para remover el valor de un nodo en la lista con el uso de plantillas por un índice
template <class T>
bool DoubleLinkList<T>::removeAt(int index){
    // Usa la misma lógica para eliminar un nodo por valor, pero en este caso, se recorre la lista hasta llegar al índic
    Node<T> *node = getFirst();
    Node<T> *ant = NULL;
    int contador = 0;
    while (node != NULL){
        if (contador == index){
            if (ant == NULL){
                first = node->getSig();
                first->setAnt(NULL);
            } else {
                ant->setSig(node->getSig());
                if (node->getSig()!= NULL){
                    node->getSig()->setAnt(ant);
                }
            }
            delete node;
            return true;
        }
        contador++;
        ant = node;
        node = node->getSig();
    }
    return false;
}
```

Funcionalidad

La función **Add** es ejecutada al ingresar los valores aleatoriamente a la lista doblemente ligada mientras que la función **Display** e **InverseDisplay** al desplegar dichos valores:

```
int main(){

    DoubleLinkList<int> dblist; // Creación de la lista doblemente ligada
    srand( Seed: static_cast<unsigned int>(time( Time: NULL)));
    int randoms[20]; // Creación de un arreglo de 20 números aleatorios
    for (int i = 0; i < 20; i++) {
        randoms[i] = (1 + rand() % 101);
        dblist.add( value: &randoms[i]); // Agrega los números aleatorios a la lista

    }

    // Despliega los valores de la lista y su inversa
    cout << "Lista original: \n";
    dblist.display();
    cout << "Lista inversa: \n";
    dblist.inverseDisplay();
```

Resultado:

```
Lista original:
100 88 84 41 90 75 58 41 34 40 78 5 9 83 25 24 27 50 92 100
Lista inversa:
100 92 50 27 24 25 83 9 5 78 40 34 41 58 75 90 41 84 88 100
```

Tras pedir un valor para un índice y otro para reemplazarlo, la función **Update** realiza la respectiva operación

```
Ingrese un indice:
0
Ingrese un valor para reemplazar el valor en el indice:
333
333 88 84 41 90 75 58 41 34 40 78 5 9 83 25 24 27 50 92 100
100 92 50 27 24 25 83 9 5 78 40 34 41 58 75 90 41 84 88 333
Ingrese un indice para eliminar el valor
```

Finalmente, la opción remove se realiza tras recibir un índice:

```
333 88 84 41 90 75 58 41 34 40 78 5 9 83 25 24 27 50 92 100
100 92 50 27 24 25 83 9 5 78 40 34 41 58 75 90 41 84 88 333
Ingrese un indice para eliminar el valor
1
333 84 41 90 75 58 41 34 40 78 5 9 83 25 24 27 50 92 100
100 92 50 27 24 25 83 9 5 78 40 34 41 58 75 90 41 84 333
```