Ayudantía 4

Carlos Lagos - carlos.lagosc@usm.cl

TDA lista basada en arreglos

```
class tLista {
    private:
        unsigned int maxSize; // Tamaño maximo de la lista
        unsigned int listSize; // Tamaño actual de la lista
        unsigned int curr; // Posición actual de la lista
        tElemLista* listaArray; // Arreglo con los elementos de la lista
    public:
        // Constructor
        // Inicializa maxSize,listSize,curr y pide memoria dinamica para asignarlo a listaArray
        tLista();
        // Destructor
        // Libera la memoria dinamica
        ~tLista();
        // Vacia la lista O(1)
        void clear();
}
```

Lista basada en arreglos

```
class tLista {
    private:
        // ...
    public:
        // ...
        // Inserta elemento en la posición actual O(n)
        int insert(tElemLista item);
        // Agregar un elemento al final de la lista O(1)
        int append(tElemLista item);
        // Borra el elemento actual y retorna su valor O(n)
        tElemLista erase();
        // Mueve la posición actual al comienzo de la lista O(1)
        void moveToStart();
        // Mueve la posición actual al final de la lista O(1)
        void moveToEnd();
        // Mueve la posición actual al elemento anterior de la lista O(1)
        void prev();
        // Mueve la posición actual al elemento siguiente de la lista O(1)
        void next();
}
```

Lista basada en arreglos

```
class tLista {
    private:
        // ...
    public:
        // ...
        // Retorna el número de elementos en la lista O(1)
        int length();
        // Retorna la posición del elemento actual O(1)
        int currPos();
        // Mueve la posición actual a una especificada O(1)
        void moveToPos(int pos);
        // Obtiene el valor del elemento actual de la lista O(1)
        tElemLista getValue();
}
```

Subsecuencias

Una subsecuencia de una lista se obtiene eliminando cero o más elementos de la lista original, manteniendo el orden relativo de los elementos restantes. Por ejemplo, en la lista $[1, 2, 3, 4]$, $[1, 3, 4]$ es una subsecuencia obtenida eliminando el elemento 2. Otros ejemplos de subsecuencias incluyen $[1]$, $[2]$, $[3]$, $[4]$, $[1, 2]$, $[1, 3]$, $[1, 4]$, $[2, 3]$, $[2, 4]$, $[3, 4]$, $[1, 2, 3]$, $[1, 2, 4]$, $[1, 3, 4]$, $[2, 3, 4]$, y $[1, 2, 3, 4]$. Se pide crear una función llamada `es_subsecuencia` que reciba dos TDA de listas y verifique si una lista es una subsecuencia de la otra.

Solución

```
bool es_subsecuencia(tLista L,tLista l){
    L.moveToStart();
    l.moveToStart();
    while(L.currPos() < L.length() && l.currPos() < l.length()){
        if(L.getValue() == l.getValue()){
            L.next();
            l.next();
        }else{
            L.next();
        }
    }
    if(l.currPos() == l.length()) return true;
    return false;
}
```

Implementación

```
struct tNodo{
    tElemLista info;
    tNodo* sig;
};
class tLista {
    private:
        // Puntero que apunta a una nodo que no guarda valor.
        // Es decir, es un puntero que se usa de forma auxiliar.
        tNodo* head;
        // Puntero que apunta al ultimo nodo de la estructura.
        tNodo* tail;
        // Puntero que apunta al nodo actual.
        tNodo* curr;
        unsigned int listSize;
        unsigned int pos; // posicion actual en la lista
    public:
        // ...
}
```

Lista enlazada

```
class tLista {
    private:
        //...
    public:
        tLista(); // O(1)
        ~tLista(); // O(n)
        void clear();
        int insert(tElemLista item); // O(1)
        int append(tElemLista item); // O(1)
        tElemLista erase(); // O(1)
        void tLista::moveToStart() // O(1)
        void tLista::moveToEnd(); // O(1)
        void tLista::prev(); // ???
        void tLista::next(); // O(1)
        int length(); // O(1)
        int currPos(); // O(1)
        void moveToPos(int pos); // O(n)
        tElemLista getValue(); // O(1)
}
```

Lista enlazada

```
int tLista::insert(tElemLista item) {
    tNodo* aux = curr->sig;
    curr->sig = new tNodo;
    curr->sig->info = item;
    curr->sig->sig = aux;
    if (curr == tail) tail = curr->sig;
    listSize++;
    return pos;
}
```

```
void tLista::prev() {
    tNodo* temp;
    if (curr == head) return;
    temp = head;
    while (temp->sig != curr)
        temp = temp->sig;
    curr = temp;
    pos--;
}
```

Lista enlazada

```
void tLista::moveToPos(int posicion) {
    if (posicion < 0 || posicion > listSize) return;
    curr = head;
    pos = 0;
    for (int i = 0; i < posicion; i++) {
        curr = curr->sig;
        pos++;
    }
}
```

Función "agua_y_aceite"

Implementa una función llamada `agua_y_aceite` que recibe una lista de números enteros. La función debe recorrer la lista de izquierda a derecha, copiando cada elemento en otra lista. Sin embargo, los números pares deben ser insertados al final de la nueva lista, mientras que los impares deben ser colocados al principio. La función debe retornar la lista resultante.

Solución

```
tLista agua_y_aceite(tLista L){
    tLista resultado;
    for(L.moveToStart(); L.currPos() < L.length; L.next()){
        if(L.getValue() % 2 == 0){
            resultado.moveToEnd();
            resultado.insert(L.getValue());
        }else{
            resultado.moveToStart();
            resultado.insert(L.getValue());
        }
    }
    return resultado;
}
```

Clase

```
class tPila {
    private:
        unsigned int maxSize;
        unsigned int top;
        tElemPila* stackArray;
    public:
        tPila();
        ~tPila();
        void clear(); // ???
        int push(tElemPila item); // O(1)
        void pop(); // O(1)
        tElemPila topValue(); // O(1)
        int size(); // O(1)
};
```

Función para Imprimir el Máximo de una Pila

Recibirás por consola un entero n que indica la cantidad de elementos que se meterán en la pila. Luego, recibirás n líneas, cada una conteniendo un entero que corresponde al elemento que se meterá en la pila. Cada vez que se introduce un elemento en la pila, debes imprimir el máximo de la pila.

Solución

```
struct tElemPila{int valor,maximo;};
int main(){
    tPila pila;
    int n;
    cin >> n;
    for(int i = 0; i < n; i++){
        int numero;
        cin >> numero;
        tElemPila elemento;
        elemento.valor = numero;
        elemento.maximo = numero;
        if(pila.size() > 0){
            if(elemento.maximo < pila.topValue().maximo)
                elemento.maximo = pila.topValue().maximo;
        }
        pila.push(elemento);
        cout << elemento.maximo << endl;
    }
    return 0;
}
```

Clase

```
class tCola {
    private:
        unsigned int maxSize;
        unsigned int front;
        unsigned int back;
        tElemPila* queueArray;
    public:
        void clear();
        int enqueue(tElemCola item);
        void dequeue();
        tElemCola frontValue();
        int size();
};
```

Problema de Gestión de Clientes en un Banco

En un banco, los clientes llegan y solo hay una persona atendiendo, por lo que se forma una fila. Primero, se reciben por consola los tiempos estimados que cada persona se demorará en ser atendida, manteniendo el orden de llegada. Estos datos se ingresan en una cola.

El sistema debe simular el proceso de atención de los clientes y luego imprimir, para cada persona, el tiempo en que llega y el tiempo en que termina de ser atendida. Se asume que la atención comienza en el tiempo 0.

Solución

```
int main(){
    int n;
    cin >> n;
    tCola cola;
    for(int i = 0; i < n; i++){
        int tiempo_estimado;
        cin >> tiempo_estimado;
        cola.enqueue(tiempo_estimado);
    }
    int tiempo = 0;
    int persona = 1;
    while(cola.size() > 0){
        int valor = cola.frontValue();
        cola.dequeue;
        cout << persona++ << ":" << endl;
        cout << "\t" << tiempo << endl;
        tiempo += valor;
        cout << "\t" << tiempo << endl;
    }
    return 0;
}
```