UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

ESTRUCTURA DE DATOS II

CONTENIDO: TAREA-4. EJERCICIOS BÁSICOS SOBRE LISTAS ENCADENADAS SIMPLES.

PORCENTAJE TERMINADO: 100%

GRUPO: 14

INTEGRANTES	DT	HG	HI	EVAL
Flores Veizaga Eudenia Gandira	2	1	1	90
Garcia Taboada Brayan Albaro	2	1	3	100
Gedalge Cayhuara Cristian Gabriel	2	1	3	100
Haquin Serrano Rodrigo	2	1	1	90
Hernandez Lijeron Roly	2	1	3	100

//DT, días trabajados

//HG, horas grupo

//HI, horas individual

// Eval

Fecha de presentación : jueves, 28 de marzo de 2024

Fecha Presentada : jueves, 28 de marzo de 2024

Días de Atraso : 0

Codigo

```
public class ListaEncadenada {
 public Nodo prim;
 public int cantElem;
 public Nodo ult;
 public ListaEncadenada(){
   prim=ult=null;
   cantElem=0;
}
 //1. L1.toString() : Método que devuelve una cadena, que representa la secuencia de elementos de la lista L1.
 public String toString(){
   String s1="[";
   Nodo p=prim;
   while(p!=null){
      s1+=p.elem;
      if(p.prox!=null){
         s1+=", ";
      }
      p=p.prox;
   }
   return s1+"]";
}

 //2. L1.insertarPrim(x) : Método que inserta el elemento x, al inicio de la lista L1.

 public void insertarPrim(int x){
    if(vacia()){
       prim=ult=new Nodo(x,null);
    } else {
```

```
        prim=new Nodo(x,prim);
    }
    cantElem++;
}

//3.L1.insertarUlt(x) : Método que inserta el elemento x, al inicio de la lista L1.
public void insertarUlt(int x){
    if(vacia()){
        prim=ult=new Nodo(x,null);
    } else {
        ult=ult.prox=new Nodo(x,null);
    }
    cantElem++;
}
public void insertarIesimo(int x,int i){
    Nodo p=prim;
    Nodo ap=null;
    int k=0;
    while(p!=null && i>k){
        ap=p;
        p=p.prox;
        k++;
    }
    insertarNodo(x,ap,p);
}

//4. L1.iguales() : Método Lógico que devuelve True, si todos los elementos de la lista L1 son iguales.
public Boolean iguales(){
    Nodo p=prim;
    while(p!=null ){
```

```
        if(p.elem!=p.prox.elem){
            return false;
        }
        p=p.prox;
    }
    return true;
}
```

//5.L1.diferentes() : Método Lógico que devuelve True, si todos los elementos de la lista L1 son diferentes.

```
public boolean diferentes() {
        Nodo cabeza = prim;
        int elemento = cabeza.elem;
        while (cabeza.prox != null) {
            if (elemento != cabeza.prox.elem) {
                    return false;
            }
            cabeza = cabeza.prox;
        }
        return true;
}
```

//6. L1.mayorElem() : Método que devuelve el mayor elemento de la lista L1.

```
public int mayorElem(){
    Nodo p=prim;
    int mayor = p.elem;
    while(p!=null ){
        p=p.prox;
        if(mayor<p.elem){
            mayor=p.elem;
        }
    }
```

```
    return mayor;
}
//7.      L1.menorElem() : Método que devuelve el menor elemento de la lista L1.
public int menorElem(){
    Nodo p=prim;
    int menor = p.elem;
    while(p!=null ){
        p=p.prox;
        if(menor>p.elem){
            menor=p.elem;
        }
    }
    return menor;
}
//8.      L1.ordenado()  : Método Lógico que devuelve True, si todos los elementos de la lista L1 están ordenados en forma ascendente o descendente.
    public boolean ordenado() {
       return Ascendente() || Descendente();
    }

   public boolean Ascendente() {
       Nodo p = prim;
       while (p.prox != null) {
           if (p.elem > p.prox.elem) {
               return false;
           }
           p = p.prox;
       }
       return true;
   }
```

```
public boolean Descendente() {
    Nodo p = prim;
    while (p.prox != null) {
        if (p.elem < p.prox.elem) {
            return false;
        }
        p = p.prox;
    }
    return true;
}
```

//9. L1.pares() : Método lógico que devuelve True, si todos los elementos de la lista L1 son pares.

```
public boolean pares() {
    Nodo p = prim;
    while (p != null) {
        if (!(p.elem % 2 == 0)) {
            return false;
        }
        p = p.prox;
    }
    return true;
}
```

//10. L1.parImpar() : Método lógico que devuelve True, si la lista L1 contiene al menos un elemento par e impar.

```
public boolean parImpar() {
    Nodo p = prim;
    boolean par = false, impar = false;
    while (p != null &&(!par || !impar)) {
        if (p.elem % 2 == 0) {
            par = true;
        } else {
```

```
            impar = true;
          }
          p = p.prox;
       }
       return par && impar;
    }
```

//11. L1.reemplazar(x, y) : Método que reemplaza todas las ocurrencias del elemento x por el elemento y en la lista L1.

```
public void reemplazar(int x,int y){
    Nodo p=prim;
    while(p!=null ){
        if(p.elem==x){
            p.elem=y;
        }
     p=p.prox;
    }
}
```

//12. L1.seEncuentra(x) : Método Lógico que devuelve True, si el elemento x, se encuentra en la lista L1.

```
public Boolean seEncuentra(int x){
     Nodo p=prim;
    while(p!=null ){
        if(p.elem==x){
            return true;
        }
     p=p.prox;
    }
    return false;
}
```

//13. L1.frecuencia(x) : Método que devuelve la cantidad de veces que aparece el elemento x en la lista L1.

```
public int frecuencia(int x){
```

```
    Nodo p=prim;
    int frec=0;
    while(p!=null ){
        if(p.elem==x){
            frec++;
        }
     p=p.prox;
    }
    return frec;
}
```

//14. L1.existeFrec(k) : Método Lógico que devuelve True, si existe algún elemento que se repite exactamente k-veces en la lista L1.

```
   public boolean existeFrec(int k) {
      Nodo P = this.prim;
      while (P != null) {
         if (k == frecuencia(P.elem)) {
            return true;
         }
         P = P.prox;
      }
      return false;
   }
```

//15. L1.mismasFrec() : Método Lógico que devuelve True, si todos los elementos de la lista L1 tienen la misma frecuencia.

```
   public boolean mismaFrec() {
      Nodo P = this.prim;
      int frec = frecuencia(P.elem);
      while (P != null) {
         if (frec != frecuencia(P.elem)) {
            return false;
         }
         P = P.prox;
```

```
    }
    return true;
  }
```

//16. L1.poker() : Método Lógico que devuelve True, si los elementos de la lista L1 forman poker. (Todos los elementos son iguales excepto uno)

```
  public boolean poker() {
    Nodo P = this.prim;
    int ele1 = P.elem;
    while (P != null) {
      if (ele1 != P.elem) {
        int carta1 = frecuencia(ele1);
        int carta2 = frecuencia(P.elem);
        if (carta1 == 1 && carta2 > 1 && (carta1 + carta2) == this.cantElem) {
          return true;
        } else {
          if (carta1 > 1 && carta2 == 1 && (carta1 + carta2) == this.cantElem) {
            return true;
          }
        }
      }
      P = P.prox;
    }
    return false;
  }
```

//17. L1.existePar() : Método lógico que devuelve True, si la lista L1 contiene al menos un elemento par.

```
  public boolean existePar() {
    Nodo p = prim;
    while (p != null) {
      if (p.elem % 2 == 0) {
        return  true;
      }
```

```
            p=p.prox;
        }
        return false;
    }
```

//18. L1.existeImpar() : Método lógico que devuelve True, si la lista L1 contiene al menos un elemento impar.

```
    public boolean existeImpar() {
        Nodo p = prim;
        while (p != null) {
            if (p.elem % 2 != 0)
                return  true;
            p=p.prox;
        }
        return false;
    }
```

//19. L1.todoPares() : Método lógico que devuelve True, si todos los elementos de la lista L1 son pares.

```
    public boolean todoPares() {
        return pares();
    }
```

//20. L1.todoImpares() : Método lógico que devuelve True, si todos los elementos de la lista L1 son impares.

```
    public boolean todoImpares() {
        Nodo p = prim;
        while (p != null) {
            if (!(p.elem % 2 != 0)) {
                return false;
            }
            p = p.prox;
        }
        return true;
    }
```

//21. L1.existeParImpar() : Método lógico que devuelve True, si en la lista L1 al menos existe un elemento par y un elemento impar.

```
public boolean existeParImpar(){
   return parImpar();
}
```

//22. L1.alternos() : Método lógico que devuelve true, si la lista L1 contiene elementos en la siguiente secuencia: par, impar, par, impar, . . . or impar, par, impar, par, ...

```
public boolean alternos(){
   Nodo p=prim;
   while (p.prox!=null){
      if(!(((p.elem%2==0) && (p.prox.elem !=0)) ||
            ((p.elem%2 != 0) && (p.prox.elem == 0)))){
         return false;
      }
      p=p.prox;
   }
   return true;
}
```

//23. L1.insertarUlt(L2) : Método que inserta los elementos de la Lista L2, al final de la Lista L1.

```
public void insertarUlt(ListaEncadenada L2){
    Nodo p=L2.prim;
    while(p!=null ){
     insertarUlt(p.elem);
     p=p.prox;
    }
}
```

//24. L1.insertarLugarAsc(x) : Método que inserta el elemento x, en su lugar correspondiente en la Lista L1, ordenada de menor a mayor.

```
public void insertarLugar(int x){
    Nodo p=prim;
    Nodo ap=null;
```

```
    while(p!=null && x>p.elem){
        ap=p;
        p=p.prox;
    }
    insertarNodo(x,ap,p);
}
```

//25. L1.insertarLugarDesc(x) : Método que inserta el elemento x, en su lugar correspondiente en la Lista L1, ordenada de mayor a menor.

```
public void insertarLugarDesc(int x){
    Nodo p=prim;
    Nodo ap=null;
    while(p!=null && x<p.elem){
        ap=p;
        p=p.prox;
    }
    insertarNodo(x,ap,p);
}
```

//26. L1.intercalar(L2, L3) : Método que intercala los elementos de las Listas L2 con L3 en L1.

```
public void intercalar(ListaEncadenada L2,ListaEncadenada L3){
   Nodo p=L2.prim;
   Nodo p1=L3.prim;
   while (p!=null || p1!=null){
      if(p!=null){
      insertarUlt(p.elem);
      insertarUlt(p.elem);
      }
      if(p!=null){
      insertarUlt(p1.elem);
      p1=p1.prox;
      }
   }
```

```
}
```

```
//27.   Adicionar al menos 5 métodos cualesquiera de consulta interesantes.
public void insertarNodo(int x,Nodo ap,Nodo p){
   if(ap==null) insertarPrim(x);
   else
   if (p==null) insertarUlt(x);
   else{
      ap.prox=new Nodo(x,p);
      cantElem++;
   }
}
 //metodo que devuelve true si sus valores van en orden:mayor,menor,mayor,menor...p
viceversa
public boolean zicZac(){
  Nodo p=prim;
  while (p.prox!=null){
     if(!((p.elem>=p.prox.elem) ||
           (p.elem<=p.prox.elem))){
        return false;
     }
     p=p.prox;
  }
  return true;
}

//devuelve true si la lista esta vacia y false si no lo esta

public Boolean vacia(){
   return prim==null && ult==null;
```

```
}
//devuelve la posicion indexada del elemento x en la lista
public int posicion(int x){
   Nodo p=prim;
   int pos=0;
   while(p!=null){
      if(x==p.elem){
         break;
      }
      pos++;
      p=p.prox;
   }
   return pos;
}

//devuelve la suma de los elementos de la lista
public int suma(){
   Nodo p=prim;
   int sum=0;
   while (p!=null){
      sum+=p.elem;
      p=p.prox;
   }
   return sum;
}
//devuelve la suma de los primeros n elementos
public int suma(int n){

   Nodo p=prim;
   int sum=0;
   while (p!=null && n!=0){
```

```
        sum+=p.elem;
        p=p.prox;
        n--;
    }
    return sum;
 }

}
class Nodo {
   public int elem;
   public Nodo prox;
   public Nodo (int elem, Nodo prox) {
      this.elem=elem;
      this.prox=prox;

   }
}
```