Universidad Autónoma Gabriel René Moreno Carrera de Ingeniería de Sistemas

Materia: Estructura de Datos II

Fecha: 26-09-2024

Nro.	Grupo	Actividad	Nombre Estudiante(s)	Porcentaje Terminado
1	18	LAB-8.	Daniela Perez Gomez	100%
2	15	LAB-8.	Gustavo Ibarra Cuellar	100%

Santa Cruz – Bolivia

LAB-8 LISTAS DOBLEMENTE ENLAZADAS

Comentario: nos centramos en optimizar en operaciones en inserción , teniendo en cuenta el redimensionamiento automático de la lista doblemente enlazada. Esto nos permitió gestionar eficientemente las variaciones en el tamaño de la lista.

```
public Lista2() {
    this.cantElem = 0;
    this.prim = this.ult = null;
  }

  @Override
  public String toString() {
    StringBuilder s = new StringBuilder("[");
    Nodo2 p = this.prim;
    while (p != null) {
      s.append(p.elem);
      if (p.prox != null) {
        s.append(",");
      }
      p = p.prox;
    }
    s.append("]");
    return s.toString();
  }

  public boolean vacio() {
    return this.cantElem == 0;
  }
```

1.-insertarUlt(x)

```
  public void insertarUlt(int x) {
    if (vacio()) {
      this.prim = this.ult = new Nodo2(null, x, null);
    } else {
      this.ult.prox = new Nodo2(this.ult, x, null);
      this.ult = this.ult.prox;
    }
    this.cantElem++;
  }
```

2.-insertarPrimx)

```
public void insertarPrim(int x) {
   if (vacio()) {
      this.prim = this.ult = new Nodo2(null, x, null);
   } else {
      this.prim = new Nodo2(null, x, this.prim);
      this.prim.prox.ant = this.prim;
   }
   this.cantElem++;
}
```

3.-insertarIesimo(x,i)

```
public void insertarIesimo(int x, int i) {
   if (i < 0 || i > cantElem) {
      throw new IndexOutOfBoundsException("Índice fuera de rango");
   }

   Nodo2 p = this.prim, op = null;
   int k = 0;

   while (p != null && k != i) {
      op = p;
      p = p.prox;
      k++;
   }
   insertarNodo(x, op, p);
}

private void insertarNodo(int x, Nodo2 op, Nodo2 p) {
   Nodo2 nuevo = new Nodo2(op, x, p);
   if (op == null) {
      this.prim = nuevo;
   } else {
      op.prox = nuevo;
   }
   if (p != null) {
      p.ant = nuevo;
   } else {
      this.ult = nuevo;
   }
   this.cantElem++;
}

public void insertarLugar(int x) {
   Nodo2 p = prim, op = null;
```

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

  public void ordenarAsc() {
    if (vacio()) return;
    for (Nodo2 p = this.prim; p != null; p = p.prox) {
      Nodo2 menor = p;
      for (Nodo2 p2 = p; p2 != null; p2 = p2.prox) {
        if (p2.elem < menor.elem) {
          menor = p2;
        }
      }
      int aux = p.elem;
      p.elem = menor.elem;
      menor.elem = aux;
    }
  }

  public void ordenarDesc() {
    if (vacio()) return;
    for (Nodo2 p = this.prim; p != null; p = p.prox) {
      Nodo2 mayor = p;
      for (Nodo2 p2 = p; p2 != null; p2 = p2.prox) {
        if (p2.elem > mayor.elem) {
          mayor = p2;
        }
      }
      int aux = p.elem;
      p.elem = mayor.elem;
      mayor.elem = aux;
    }
  }
```

4.-insertarLugarDesc(x)

```
  public void insertarLugarDesc(int x) {
    insertarUlt(x);
    ordenarDesc();
  }
```

5.-insertarLugarAsc(x)

```
public void insertarLugarAsc(int x) {
   insertarUlt(x);
   ordenarAsc();
}
```

6.-insertarIesimo(L2,i)

```
public void insertarIesimo(Lista2 L2, int i) {
   if (L2.vacio()) return;
   Nodo2 p = L2.prim;
   while (p != null) {
      this.insertarIesimo(p.elem, i);
      p = p.prox;
   }
}
```

7.-insertarPrim(L2)

```
public void insertarPrim(Lista2 L2) {
   if (L2.vacio()) return;
   Nodo2 p = L2.prim;
   while (p != null) {
      this.insertarPrim(p.elem);
      p = p.prox;
   }
}
```

8.-insertarUlt(L2)

```
public void insertarUltI(Lista2 L2) {
   if (L2.vacio()) return;
   Nodo2 p = L2.prim;
   while (p != null) {
      this.insertarUlt(p.elem);
      p = p.prox;
   }
}
```

9.-iguales()

```
public boolean iguales() {
   if (this.prim == null) {
      return true;
   }
   Nodo2 p = this.prim;
   while (p != null) {
      if (p.elem != this.prim.elem) {
```

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

10.-L1.diferentes()

```
public boolean diferentes() {
    Nodo2 p1 = this.prim;
    while (p1 != null) {
        Nodo2 p2 = p1.prox;
        while (p2 != null) {
            if (p1.elem == p2.elem) {
                return false;
            }
            p2 = p2.prox;
        }
        p1 = p1.prox;
    }
    return true;
}

public boolean ascendente() {
    if (vacio()) return false;
    Nodo2 p = this.prim;
    while (p.prox != null) {
        if (p.elem > p.prox.elem) {
            return false;
        }
        p = p.prox;
    }
    return true;
}

public boolean descendente() {
    if (vacio()) return false;
    Nodo2 p = this.prim;
    while (p.prox != null) {
        if (p.elem < p.prox.elem) {
            return false;
        }
        p = p.prox;
    }
```

```
    return true;
  }
```

11.-L1.ordenado()

```
  public boolean ordenado() {
    return ascendente() || descendente();
  }
```

12.-L1.indexOf(x)

```
  public int indexOf(int x) {
    Nodo2 p = this.prim;
    int pos = 0;
    while (p != null) {
      if (p.elem == x)
        return pos;
      pos++;
      p = p.prox;
    }
    return -1;
  }
  public Nodo2 direccion(int n) {
    Nodo2 p = this.prim;
    for (int i = 0; i < n && p != null; i++) {
      p = p.prox;
    }
    return p;
  }

  public void invertir() {
    if (vacio()) return;
    Nodo2 p = this.prim;
    for (int i = 0; i < cantElem / 2; i++) {
      int aux = p.elem;
      Nodo2 p2 = direccion(cantElem - i - 1);
      p.elem = p2.elem;
      p2.elem = aux;
      p = p.prox;
    }
  }
```

13.-L1.lastIndexOf(x)

```
  public Nodo2 lastIndexOf(int x) {
    Nodo2 last = null;
    Nodo2 p = this.prim;
    while (p != null) {
```

```
        if (p.elem == x) {
            last = p;
        }
        p = p.prox;
    }
    return last;
}
```

13.-L1.palindrome()

```
public boolean esPalindromo() {
    if (vacio()) return true;

    Lista2 listaAux = new Lista2();
    Nodo2 p = this.prim;

    while (p != null) {
        listaAux.insertarUlt(p.elem);
        p = p.prox;
    }

    listaAux.invertir();
    Nodo2 original = this.prim;
    Nodo2 invertida = listaAux.prim;

    while (original != null && invertida != null) {
        if (original.elem != invertida.elem) {
            return false;
        }
        original = original.prox;
        invertida = invertida.prox;
    }
    return true;
}
```