III. Sumandos y Mochila. Lista de Listas

Implementar con la Lógica de utilizar Listas de Listas de Soluciones los ejercicios propuestos en los enunciados de **I.** y **II.** de los Problemas de Sumandos y los Problemas de la Mochila.

Ejercicio 1:

Encontrar los sumandos posibles en una Lista.

```
public static void sumandos(LinkedList<Integer> L,
                            LinkedList<LinkedList<Integer>> L2,
                            int i,
                            int n){
        int sum = suma(L);
        if(sum>n)return;
        if(sum==n){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            L.add(k);
            sumandos(L,L2,k,n);
            L.removeLast();
        }
    }
```

Ejercicio 2:

Encontrar todos los sumandos posibles diferentes en una Lista.

```
public static void sumandosDif(LinkedList<Integer> L,
                               LinkedList<LinkedList<Integer>> L2,
                               int i,
                               int n){
        int sum = suma(L);
        if(sum>n)return;
        if(sum==n && distintos(L)){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            L.add(k);
            sumandosDif(L,L2,k,n);
            L.removeLast();
        }
    }
```

Ejercicio 3:

Encontrar todos los sumandos posibles iguales en una Lista.

```
public static void sumandosEq(LinkedList<Integer> L,
                              LinkedList<LinkedList<Integer>> L2,
                              int i,
                              int n){
        int sum = suma(L);
        if(sum>n)return;
        if(sum==n && iguales(L)){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            L.add(k);
            sumandosEq(L,L2,k,n);
            L.removeLast();
        }
    }
```

Ejercicio 4:

Encontrar todos los sumandos primos posibles en una Lista.

```
public static void sumandosPrimos(LinkedList<Integer> L,
                                  LinkedList<LinkedList<Integer>> L2,
                                  int i,
                                  int n){
        criba();
        _sumandoPrimos(L,L2, i, n);
    }

private static void _sumandoPrimos(LinkedList<Integer> L,
                                   LinkedList<LinkedList<Integer>> L2,
                                   int i,
                                   int n){
        int sum = suma(L);
        if(sum>n)return;
        if(sum==n && primos(L)){
             L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            L.add(k);
            sumandosPrimos(L,L2,k,n);
            L.removeLast();
        }
    }
```

Ejercicio 5:

Encontrar todos los sumandos entre a y b inclusive en una Lista.

```
public static void sumandosRango(LinkedList<Integer> L,
                                 LinkedList<LinkedList<Integer>> L2,
                                 int a,
                                 int b,
                                 int i,
                                 int n){
        int sum = suma(L);
        if(sum>n)return;
        if(sum==n && rango(L,a,b)){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            L.add(k);
            sumandosRango(L,L2,a,b,k,n);
            L.removeLast();
        }
    }
```

Ejercicio 6:

Encontrar los factores posibles en una Lista.

```
public static void productos(LinkedList<Integer> L,
                             LinkedList<LinkedList<Integer>> L2,
                             int i,
                             int n){
        int prod = producto(L);
        if(prod>n)return;
        if(prod==n){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            if(n%k==0){
                L.add(k);
                productos(L,L2,k,n);
                L.removeLast();
            }
        }
    }
```

Ejercicio 7:

Encontrar todos los factores posibles diferentes en una Lista.

```
public static void productosDif(LinkedList<Integer> L,
                                    LinkedList<LinkedList<Integer>> L2,
                                    int i,
                                    int n){
        int prod = producto(L);
        if(prod>n)return;
        if(prod==n && distintos(L)){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            if(n%k==0){
                L.add(k);
                productosDif(L,L2,k,n);
                L.removeLast();
            }
        }
    }
```

Ejercicio 8:

Encontrar todos los factores posibles iguales en una Lista.

```
public static void productosEq(LinkedList<Integer> L,
                                   LinkedList<LinkedList<Integer>> L2,
                                   int i,
                                   int n){
        int prod = producto(L);
        if(prod>n)return;

        if(prod==n && iguales(L)){
            L2.add((LinkedList)L.clone());
        }

        for(int k=i;k<=n;++k){
                L.add(k);
                productosEq(L,L2,k,n);
                L.removeLast();

        }
    }
```

Ejercicio 9:

Encontrar todos los factores primos posibles en una Lista.

```
public static void productosPrimos(LinkedList<Integer> L,
                                   LinkedList<LinkedList<Integer>> L2,
                                   int i, int n){
        criba();
        _productosPrimos(L,L2, i, n);
    }

    private static void _productosPrimos(LinkedList<Integer> L,
                                         LinkedList<LinkedList<Integer>> L2,
                                         int i,int n){
        int prod = producto(L);
        if(prod>n)return;
        if(prod==n && primos(L)){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            if(n%k==0){
                L.add(k);
                _productosPrimos(L,L2,k,n);
                L.removeLast();
            }
        }
    }
```

Ejercicio 10:

Encontrar todos los factores entre a y b inclusive en una Lista.

```
public static void productosRango(LinkedList<Integer> L,
                                  LinkedList<LinkedList<Integer>> L2,
                                  int i,
                                  int a,
                                  int b,
                                  int n){
        int prod = producto(L);
        if(prod>n)return;
        if(prod==n && rango(L,a,b)){
            L2.add((LinkedList)L.clone());
        }
        for(int k=i;k<=n;++k){
            if(n%k==0){
                L.add(k);
                productosRango(L,L2,k,a,b,n);
                L.removeLast();
            }
        }
    }
```

Ejercicio 11:

Encontrar todas las combinaciones de pesos de objetos que se pueden transportar en la mochila.

```
public static void mochila(LinkedList<Integer> L1,
                           LinkedList<Integer> L2,
                           LinkedList<LinkedList<Integer>> L3,int max,int
                               i){
        int sum = suma(L2);
        if(sum>max) return;
        L3.add((LinkedList)L2.clone());
        int k = i;
        while(k<L1.size()){
            L2.add(L1.get(k));
            mochila(L1,L2,L3,max,k+1);
            L2.removeLast();
            k++;
        }
    }
```

Ejercicio 12:

Encontrar todas las combinaciones de pesos diferentes que se pueden transportar en la mochila.

```
public static void mochilaDif(LinkedList<Integer> L1,
                               LinkedList<Integer> L2,
                               LinkedList<LinkedList<Integer>> L3,int max,int
                                  i){
        int sum = suma(L2);
        if(sum>max) return;
        if(distintos(L2)){
            L3.add((LinkedList)L2.clone());
        }
        int k = i;
        while(k<L1.size()){
            L2.add(L1.get(k));
            mochilaDif(L1,L2,L3,max,k+1);
            L2.removeLast();
            k++;
        }
    }
```

Ejercicio 13:

Encontrar todas las combinaciones de pesos entre a y b inclusive que se pueden transportar en la mochila.

```
public static void mochilaRango(LinkedList<Integer> L1,
                                 LinkedList<Integer> L2,
                                 LinkedList<LinkedList<Integer>> L3,
                                 int max,int i,int a,int  b){
        int sum = suma(L2);
        if(sum>max) return;
        if(rango(L2,a,b)){
            L3.add((LinkedList)L2.clone());
        }
        int k = i;
        while(k<L1.size()){
            L2.add(L1.get(k));
            mochilaRango(L1,L2,L3,max,k+1,a,b);
            L2.removeLast();
            k++;
        }
    }
```

Ejercicio 14:

Encontrar todas las combinaciones de pesos de objetos que se pueden transportar en la mochila.

```
public void mochilaMayor(LinkedList<Integer> L1,
                         LinkedList<Integer> L2,
                         LinkedList<LinkedList<Integer>> L3,
                         int max,
                         int i
                         ){
        LinkedList<Integer> sol = _mochilaMayor(L1, L2, max, i);

        _mochilaMayorSoluciones(L1, L2, sol.size(), L3, max, i);

        System.out.println(L3);
    }



public static void _mochilaMayorSoluciones(LinkedList<Integer> L1,
                                            LinkedList<Integer> L2,
                                            int s,
                                            LinkedList<LinkedList<Integer>> L4,
                                            int max,int i){
        int sum = suma(L2);

        if(sum>max) return;

        if(L2.size()==s){
            L4.add((LinkedList)L2.clone());
        }

        int k = i;
        while(k<L1.size()){
            L2.add(L1.get(k));

                    _mochilaMayorSoluciones(L1,L2,s,L4,max,k+1);

            L2.removeLast();
            k++;
        }


    }
```

Ejercicio 15:

Encontrar todas las combinaciones de pesos de objetos que se pueden transportar en la mochila.

```
public void mochilaClosest(LinkedList<Integer> L1,
                           LinkedList<Integer> L2,
                           LinkedList<LinkedList<Integer>> L3,
                           int max,
                           int i
                           ){
        LinkedList<Integer> sol = _mochilaClosest(L1, L2, max, i);

        _mochilaClosestSoluciones(L1, L2, sol.size(), L3, max, i);

        System.out.println(L3);
    }

public static LinkedList<Integer>  _mochilaClosest(LinkedList<Integer>
   L1,LinkedList<Integer> L2,int max,int i){

        int sum = suma(L2);
        if(sum>max) return new LinkedList<>();

        int k = i;
        LinkedList<Integer> T= (LinkedList)L2.clone();
        LinkedList<Integer> TMP;
        while(k<L1.size()){
            L2.add(L1.get(k));
            TMP = _mochilaClosest(L1,L2,max,k+1);
            if(diferencia(TMP,max)<diferencia(T,max)){
                T = TMP;
            }
            L2.removeLast();
            k++;
        }

        return T;
    }

public static void _mochilaClosestSoluciones(LinkedList<Integer> L1,
                                             LinkedList<Integer> L2,
                                             int s,
                                             LinkedList<LinkedList<Integer>> L4,
                                             int max,int i){
        int sum = suma(L2);
```

```
        if(sum>max) return;

        if(diferencia(L2.size())==s){
            L4.add((LinkedList)L2.clone());
        }

        int k = i;
        while(k<L1.size()){
            L2.add(L1.get(k));

                        _mochilaClosestSoluciones(L1,L2,s,L4,max,k+1);

            L2.removeLast();
            k++;
        }


    }
```

Anexos

Funciones auxiliares para realizar los ejercicios

```
public static int suma(LinkedList<Integer> L){
        int total = 0;
        for(int i=0;i<L.size();++i){
            total=total + L.get(i);
        }
        return total;
    }

    public static int producto(LinkedList<Integer> L){
        int total = 1;
        for(int i=0;i<L.size();++i){
            total *= L.get(i);
        }
        return total;
    }

    public static boolean distintos(LinkedList<Integer> L){
         Set<Integer> s = new HashSet<Integer>();
         for(int i=0;i<L.size();++i){
            s.add(L.get(i));
         }
        return L.size()==s.size();
    }
```

```
    public static boolean iguales(LinkedList<Integer> L){
         Set<Integer> s = new HashSet<Integer>();
         for(int i=0;i<L.size();++i){
            s.add(L.get(i));
         }
        return s.size()==1;
    }

    public static int n = 500000;
    public static boolean prime[] = new boolean[n+1];

    public static void criba()
    {

        for(int i=0;i<n;i++)
            prime[i] = true;
        prime[0]=prime[1]=false;
        for(int p = 2; p*p <=n; p++)
        {
            if(prime[p] == true)
            {
                for(int i = p*p; i <= n; i += p)
                    prime[i] = false;
            }
        }
    }


     public static boolean primos(LinkedList<Integer> L){
         for(int i=0;i<L.size();++i){
            if(!prime[L.get(i)])
                return false;
         }
        return true;
    }


    public static boolean rango(LinkedList<Integer> L,int a,int b){
         for(int i=0;i<L.size();++i){
            if(L.get(i)<a || L.get(i)>b)
                return false;
         }
        return true;
    }
```

```
    public static int diferencia(LinkedList<Integer> L,int max){
        return max-suma(L);
    }
```