POLITÉCNICA

Indexación con lista enlazada

Estructura de datos

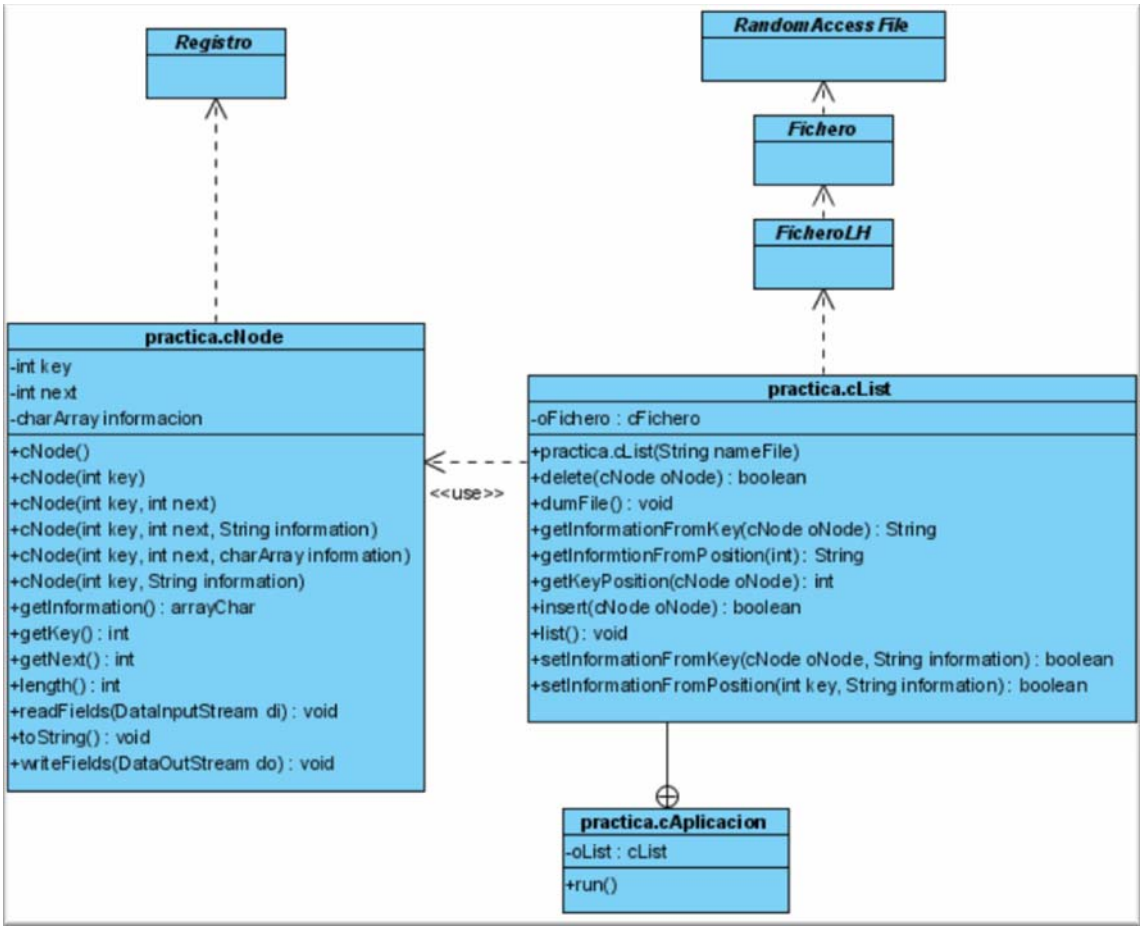
Roberto Maestre Martínez

Diseño y arquitectura del sistema

- La nomenclatura utilizada es la siguiente .

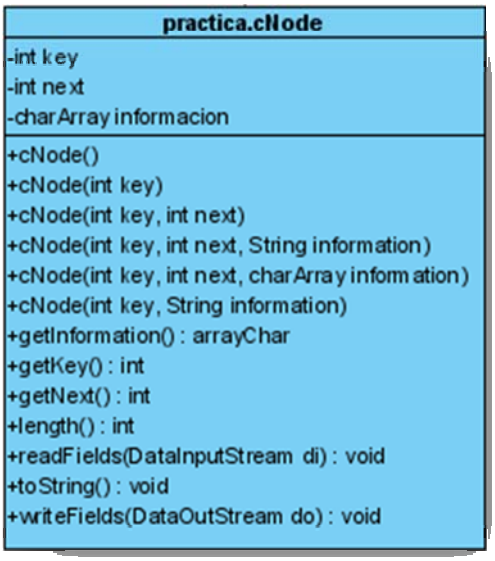
cVar – Indica nombre de clase Var
oVar – Indica un objeto de la clase Var

- El diagrama de clases utilizado es el mostrado a continuación, en el se especifican atributos, métodos, visibilidad, herencia y usos de las clases.



Implementación del sistema

- *Clases diseñadas*



Herencia -> Registro

La clase cNode nos proporcionara la estructura necesaria crear los nodos de la lista. Dichos nodos irán enlazados por el campo next. Dispone de varios constructores para iniciar los objetos con diferentes valores.

Los atributos tienen visibilidad privada ("Ley estricta de Demeter") y disponen de sus correscpondientes métodos "getters" "setters". Sobrescribe dos métodos de la clase padre Registro que son readFields(DataInputStream di) y writeFileds(DataOutputStream do) ,

Procedimientos, funciones y algoritmos a destacar

1. **Length()**

```
public int length() {
    int l = super.length();
    //// solo tocar aqui en medio
    l += Registro.INT_LENGTH;   // key
    l += Registro.INT_LENGTH;   // next
    l += Registro.CHAR_LENGTH*this.informacion.length; // informacion
    ////////////////
    return l;
}
```

Necesario para indicar la longitud del nodo (es decir, del registro en disco)

2. **readFields(DataInputStream di)**

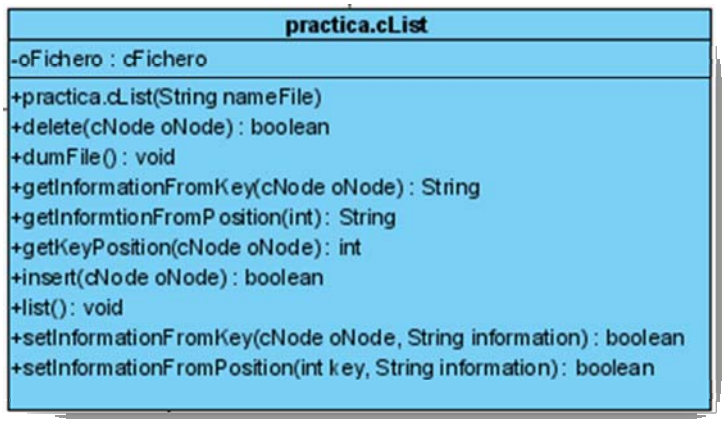
```
public void readFields(DataInputStream dis) throws IOException {
    this.key = dis.readInt();
    this.next = dis.readInt();
     for (int i=0; i<this.informacion.length;i++)
         this.informacion[i] = dis.readChar();
}
```

Necesario para recoger correctamente los valores desde el archivo en disco

3. **writeFileds(DataOutputStream do)**

```
public void readFields(DataInputStream dis) throws IOException {
    this.key = dis.readInt();
    this.next = dis.readInt();
     for (int i=0; i<this.informacion.length;i++)
         this.informacion[i] = dis.readChar();
}
```

Necesario para escribir correctamente los valores desde el archivo en disco



Rediseño: Hemos pensado que tanto en la inserción como en el borrado, debemos pasarle un objeto de la clase cNodo, el constructor de esta clase siempre necesita la clave, pero además podríamos indicarle ya directamente la información del nodo.

Herencia -> Fichero LH

Hereda de Fichero LH para implementar el manejo de la lista de huecos para añadir registros al ficheros.

Uso -> cNode

Utiliza esporádicamente la clase cNode, tanto para escribir registros como para leerlos de disco.

Procedimientos, funciones y algoritmos a destacar

1. **Constructor cList**

```
public cList(String name)
throws IOException {
    super(name, new cNode(), new cNode().length(), "rw");
    // TODO Auto-generated constructor stub
}
```

Llama al constructor de la clase padre para crear un fichero *name*, especificando la longitud new *cNode().length()* de los registros que allí se almacenaran y el tipo de acceso al fichero *"rw"*

2. **Función insertar insert(cNode oNewNode)**

```
public boolean insert(cNode oNewNode) throws IOException, ClassNotFoundException{
```

A continuación mostraremos la algorítmica utilizada en pseudocódigo

```
MIENTRAS no fin
        Leemos nodo()
          Si  (primer nodo leído AND  nuevo nodo menor que nodo leído) //Mejor de los
casos
                Posición de lista[ 1] = nuevo nodo
                Enlazar el nuevo nodo con hueco
                Antigua posición[ 1] = insertar en hueco
          SI  (lista > 2 elementos)
                Campo siguiente nodo leído = hueco
                Inserto el nuevo nodo en hueco
                Campo siguiente del nuevo nodo = campo siguiente nodo leído
          SI  (clave es repetida)
                Indico error
          SI  (lista = VACIA) //Mejor de los casos
                Inserto el nodo en la posición 1 de la lista
          SI (insertar en última posición) //Peor de los casos
                Leo el ultimo nodo
                Campo siguiente nodo leído = hueco
```

```
                    Inserto el  nuevo nodo
                    Nuevo nodo insertado = -1
REPETIR
```

La algorítmica seguida en esta función es la siguiente: preservar la posición 1 de la lista siempre con el nodo de clave menor, la posición 0 de la lista no se toca ya que maneja los huecos del fichero. Para encontrar la posición donde insertar el nuevo nodo empezaremos por el nodo de la posición física 1 de la lista, y hasta que no encontremos un nodo con clave mayor o lleguemos al final de la lista, continuamos leyendo por el campo *"next"* de los nodos.

Pos	Contenido
0	Hueco libre
1	Siempre el nodo con menor clave
.	.
n	Todos los nodos de la lista enlazados por el campo *next*

1. **Función borrar un nodo delete(cNode oNewNode)**

```
public boolean delete(cNode oDeleteNode) throws IOException, ClassNotFoundException{
```

A continuación mostraremos la algorítmica utilizada en pseudocódigo

```
MIENTRAS no fin
        Leemos nodo()
          SI (lista vacia) //Mejor de los casos
             Cortamos el algoritmo
             Devolvemos error por clave no encontrada
          SI (primer nodo leído y clave es menor) //Mejor de los casos
             Cortamos el algoritmo
             Devolvemos error por clave no encontrada
          SI (la clave a borrar es menor que el ultimo) //Peor de los casos
             Cortamos el algoritmo
             Devolvemos error por clave no encontrada
          SI (la clave a borrar es la del nodo uno)
             SI (no tiene siguiente)
                 Borro la posición[1] de la lista
           SI (tiene siguiente)
                Guardo la posición siguiente del primer nodo
                Borro el primer nodo
                Campo siguiente del primer nodo = siguiente que estaba en auxiliar
            SI (no cumple ningún de las anteriores AND la clave del nodo leído es la
buscada)
                Leo el nodo actual
                Leo el nodo anterior
                Borro el nodo actual
```

El nodo anterior apunta ahora a campo siguiente del nodo borrado

REPETIR

2. Funciones de modificación de información

```
public boolean setInformationFromKey(cNode oNodeSearch,String information) throws IOException,
```

La única función pública para modificar información en los nodos es setInformationFromKey , el cual recibe por parámetros, el nodo a buscar en disco y la información a insertar. Las demás funciones (con visibilidad privada) son utilizadas internamente por la clase o están implementadas a modo de depuración de la clase.

Nota: Las funciones de borrado / inserción devuelven un valor booleano que indica si la clave a borrar o a insertar ha sido borrada o insertada con éxito.

```
return fin&&!noBorrado;
return fin&&!noInsertado;
```

Para que la operación se realice con éxito se debe cumplir que no llegue a fin y que se haya insertado / borrado el nodo.

Tiene -> objeto cList

Tiene un objeto cList como atributo, sobre el cual ejecutara las operaciones que el usuario le indique.

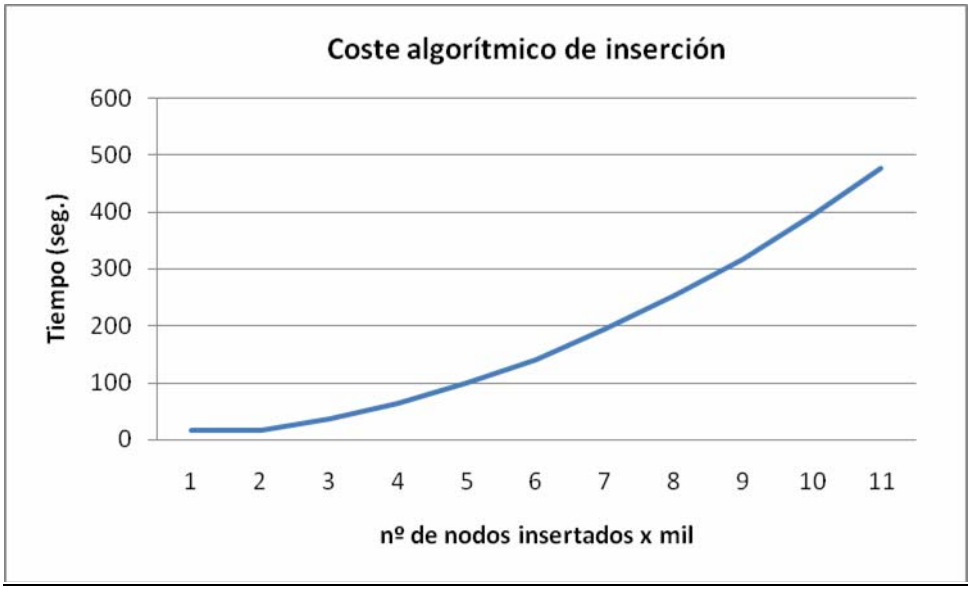
Esta clase utiliza la clase `BufferedReader` para leer de teclado las operaciones del usuario, estas operaciones, las lee mediante un bucle do while y mientras el usuario no indique la opción *"S"* de salir.

Resumen y análisis de la estructura de lista ordenada enlazada

Estudiando la estructura observamos la ineficiencia de esta para indexar información a partir de un cierto número de claves, mientras con otras estructuras como arboles binarios de búsqueda (en menor medida) o arboles B disponen de una algorítmica en recorridos e inserciones mucho más eficientes, en esta estructura, en el peor de los casos deberemos recorrer todos los nodos de la lista antes de encontrar la posición buscada y esto conlleva tantos accesos a disco para leer los nodos como posiciones recorramos. Para incrementar la eficiencia en la operación de insertado pensamos en realizar operaciones dicotómicas, pero al ser una estructura en disco y por tanto, que los nodos se guarden en posiciones aleaorias, nos encontramos con la imposibilidad de manejar la lista.

Si tenemos una lista con los nodos [1->2->…….->n] ordenados para acceder a la posición n deberemos realizar n lecturas a discos, lo que en el peor de los casos nos lleva a un orden O≈(N) .

En las pruebas obtenemos los siguientes rendimientos:



<u>Algoritmo utilizado para la elaboración del gráfico (inserciones de claves aleatorias):</u>

```
long Time1 = new Date().getTime();
oList.list();
for (int i=0; i<11000; i++) {
    oList.insert (new cNode((int)(Math.random()*10000000)));
}
long Time2 = new Date().getTime();
double totalTime = (double)((Time2 - Time1)*0.001);
System.out.println("Tiempo ejecucion "+totalTime);
```

Pruebas al sistema

1. PRUEBAS CRÍTICAS

a. La posición 1 debe guardar el nodo con menor valor

```
oList.insert(new cNode(5));
oList.insert(new cNode(4));
oList.insert(new cNode(3));
oList.insert(new cNode(2));
oList.insert(new cNode(1));
oList.insert(new cNode(0));
oList.list();
System.out.println();
oList.dumpFile();
```

```
Posicion Disco (0) VACIO
Posicion Disco (1) (0,6)
Posicion Disco (2) (5,-1)
Posicion Disco (3) (4,2)
Posicion Disco (4) (3,3)
Posicion Disco (5) (2,4)
Posicion Disco (6) (1,5)
```

Observamos como el algoritmo ha ido cambiando la posición 1 poniendo el nodo de menor clave, además ha ido cambiando los nodos que estaban en la posición 1 y son sustituidos por otro de menos clave en orden descendente desde la mayor posición libre, esto es, por la gestión de los huecos.

b. Borramos la posición 1 que guarda el nodo con menor clave

```
oList.insert(new cNode(4));
oList.insert(new cNode(3));
oList.insert(new cNode(2));
oList.insert(new cNode(1));
oList.insert(new cNode(0));
oList.delete(new cNode(0));
oList.delete(new cNode(1));
oList.list();
System.out.println();
oList.dumpFile();
```

```
Posicion Disco (0) VACIO
Posicion Disco (1) (2,3)
Posicion Disco (2) (4,-1)
Posicion Disco (3) (3,2)
Posicion Disco (4) VACIO
```

Al borrar el nodo con clave 0, el algoritmo mueve el nodo con menor clave de la lista a la posición 1 para insertarlo, así, mantenemos la posición 1 con la clave del menor valor.

c. <u>La posición 1 de la lista debe ocupar el nodo con menor valor, insertamos, borramos toda la lista y volvemos a insertar</u>

```
oList.insert(new cNode(4));
oList.insert(new cNode(3));
oList.insert(new cNode(2));
oList.insert(new cNode(1));
oList.insert(new cNode(0));
oList.delete(new cNode(0));
oList.delete(new cNode(1));
oList.delete(new cNode(2));
oList.delete(new cNode(3));
oList.delete(new cNode(4));
oList.list();
oList.insert(new cNode(5));
oList.insert(new cNode(4));
oList.insert(new cNode(99));
oList.list();
System.out.println();
oList.dumpFile();
```

```
Posicion Disco (0) VACIO
Posicion Disco (1) (4,2) |
Posicion Disco (2) (5,3) |
Posicion Disco (3) (99,-1)
Posicion Disco (4) VACIO
Posicion Disco (5) VACIO
```

La posición 1 siempre guarda el nodo con menor clave

d. Al insertar claves repetidas o borrar claves que no existen, las respectivas funciones deben devolver false en caso contrario true

```
System.out.println("Insertar Nodo con clave 1: "+oList.insert(new cNode(1)));
System.out.println("Insertar Nodo con clave 2: "+oList.insert(new cNode(2)));
System.out.println("Insertar Nodo con clave 3: "+oList.insert(new cNode(3)));

System.out.println("Insertar Nodo con clave 1: "+oList.insert(new cNode(3)));
System.out.println("Insertar Nodo con clave 3: "+oList.insert(new cNode(3)));

System.out.println("Borrar Nodo con clave 1: "+oList.delete(new cNode(1)));
System.out.println("Borrar Nodo con clave 2: "+oList.delete(new cNode(2)));
System.out.println("Borrar Nodo con clave 1: "+oList.delete(new cNode(1)));
System.out.println("Borrar Nodo con clave 2: "+oList.delete(new cNode(2)));
```

```
Insertar Nodo con clave 1: true
Insertar Nodo con clave 2: true
Insertar Nodo con clave 3: true
Insertar Nodo con clave 1: false
Insertar Nodo con clave 3: false
Borrar Nodo con clave 1: true
Borrar Nodo con clave 2: true
Borrar Nodo con clave 1: false
Borrar Nodo con clave 2: false
[3->]
```

Observamos que la salida por consola de la prueba es correcta

e. Insertar y a continuación borrar la misma clave en la lista

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
for (int c=0;c<10;c++){
    iAux=(int)(Math.random()*100);
    System.out.println("Insertar  "+iAux+": "+oList.insert(new cNode(iAux)));
    System.out.println("Borrar    "+iAux+": "+oList.delete(new cNode(iAux)));
}
oList.list();
System.out.println();
```

```
Insertar  87: true
Borrar    87: true
Insertar  75: true
Borrar    75: true
Insertar  18: true
Borrar    18: true
Insertar  73: true
Borrar    73: true
Insertar  68: true
Borrar    68: true
Insertar  87: true
Borrar    87: true
Insertar  40: true
Borrar    40: true
Insertar  32: true
Borrar    32: true
Insertar  37: true
Borrar    37: true
Insertar  44: true
Borrar    44: true
[La lista esta vacia
]
```

f. Insertar y borrar clave en la lista

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
for (int c=0;c<10;c++){
    iAux=(int)(Math.random()*100);
    System.out.println("Insertar  "+iAux+": "+oList.insert(new cNode(iAux)));
    iAux=(int)(Math.random()*100);
    System.out.println("Borrar    "+iAux+": "+oList.delete(new cNode(iAux)));
}
oList.list();
System.out.println();
```

```
Insertar  99: true
Borrar    62: false
Insertar  36: true
Borrar    46: false
Insertar  61: true
Borrar    88: false
Insertar  61: false
Borrar    89: false
Insertar  15: true
Borrar    15: true
Insertar  2: true
Borrar    97: false
Insertar  88: true
Borrar    5: false
Insertar  25: true
Borrar    25: true
Insertar  96: true
Borrar    77: false
Insertar  58: true
Borrar    48: false
[2->36->58->61->88->96->99->]
```

g. <u>Persistencia de los datos en disco, ejecutamos estos códigos en el siguiente orden</u>

i.

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
for (int c=0;c<10;c++){
    iAux=(int)(Math.random()*100);
    System.out.println("Insertar  "+iAux+": "+oList.insert(new cNode(iAux)));
    iAux=(int)(Math.random()*100);
    System.out.println("Borrar    "+iAux+": "+oList.delete(new cNode(iAux)));
}
oList.list();
System.out.println();
```

```
[7->28->32->33->60->61->63->65->76->]
```

Si a continuación ejecutamos el siguiente código

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
/*for (int c=0;c<10;c++){
    iAux=(int)(Math.random()*100);
    System.out.println("Insertar  "+iAux+": "+oList.insert(new cNode(iAux)));
    iAux=(int)(Math.random()*100);
    System.out.println("Borrar    "+iAux+": "+oList.delete(new cNode(iAux)));
}*/
oList.list();
System.out.println();
```

```
[7->28->32->33->60->61->63->65->76->]
```

Si a continuación ejecutamos

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
/*for (int c=0;c<10;c++){
    iAux=(int)(Math.random()*100);
    System.out.println("Insertar  "+iAux+": "+oList.insert(new cNode(iAux)));
    iAux=(int)(Math.random()*100);
    System.out.println("Borrar    "+iAux+": "+oList.delete(new cNode(iAux)));
}*/
oList.delete(new cNode(76));
oList.delete(new cNode(7));
oList.insert(new cNode(40));
oList.list();
System.out.println();
```

```
[28->32->33->40->60->61->63->65->]
```

Las tres ejecuciones de los distintos códigos son independientes, y comprobamos la correcta persistencia de los nodos de la lista en disco

2. PRUEBAS ALEATORIAS CON UN GRAN NÚMERO DE NODOS

a.

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
for (int c=0;c<1000;c++){
    iAux=(int)(Math.random()*50);
    oList.insert(new cNode(iAux));
    iAux=(int)(Math.random()*50);
    oList.delete(new cNode(iAux));
}
oList.list();
System.out.println();
```

```
[0->2->5->6->7->8->11->12->13->14->20->27->29->30->32->36->37-
>38->39->41->44->45->46->49->]
```

b.

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
for (int c=0;c<1000;c++){
    iAux=(int)(Math.random()*100);
    oList.insert(new cNode(iAux));
    iAux=(int)(Math.random()*100);
    oList.delete(new cNode(iAux));
}
oList.list();
System.out.println();
```

```
[5->6->8->10->11->12->15->20->21->26->29->30->31->33->35->36->41->42->43->46-
>47->49->50->53->55->58->59->65->67->68->69->72->73->75->77->78->79->80->82-
>84->85->87->88->91->94->97->]
```

c.

```
cList oList = new cList("DATA");
int iAux=0;
//PRUEBA 1
for (int c=0;c<500;c++){
    iAux=(int)(Math.random()*10000);
    oList.insert(new cNode(iAux));
    iAux=(int)(Math.random()*10000);
    oList.delete(new cNode(iAux));
}
oList.list();
System.out.println();
```

```
[40->57->79->170->178->224->230->284->286->303->321->328->352->383->388->394-
>429->440->466->488->508->536->579->587->663->674->691->711->722->723->748-
>796->813->824->834->846->872->873->893->928->1014->1017->1040->1043->1045-
>1046->1131->1148->1177->1180->1209->1214->1216->1220->1227->1232->1235->1294-
>1339->1410->1443->1455->1476->1528->1559->1568->1601->1603->1693->1715->1721-
>1771->1778->1800->1803->1821->1834->1854->1856->1901->1904->1925->1969->2023-
>2024->2042->2112->2126->2149->2167->2192->2194->2207->2210->2224->2226->2243-
>2249->2257->2283->2362->2389->2394->2410->2453->2474->2481->2529->2535->2552-
>2567->2568->2574->2585->2587->2593->2600->2634->2683->2689->2729->2736->2738-
>2769->2771->2801->2825->2829->2891->2903->2913->2920->2953->2987->3052->3074-
>3095->3126->3131->3167->3173->3187->3225->3239->3242->3273->3294->3299->3318-
>3333->3347->3405->3451->3475->3496->3534->3593->3596->3597->3601->3640->3645-
>3658->3664->3680->3684->3689->3691->3696->3729->3732->3741->3747->3760->3778-
>3787->3790->3800->3824->3853->3906->3933->3999->4000->4018->4022->4047->4079-
>4082->4089->4157->4182->4190->4214->4221->4251->4268->4270->4279->4284->4309-
>4318->4361->4391->4405->4424->4463->4475->4482->4483->4527->4533->4589->4621-
>4679->4709->4722->4723->4744->4764->4780->4823->4827->4841->4855->4897->4928-
>4933->4961->4975->4981->5028->5039->5093->5187->5189->5216->5244->5250->5262-
>5269->5270->5278->5290->5322->5338->5387->5390->5477->5478->5494->5567->5584-
>5594->5678->5680->5688->5725->5727->5788->5802->5841->5877->5882->5909->5923-
>5943->5948->5974->6006->6023->6027->6042->6070->6080->6083->6136->6207->6220-
>6221->6225->6235->6237->6330->6344->6371->6390->6395->6425->6463->6488->6512-
>6528->6547->6556->6578->6612->6625->6628->6659->6661->6663->6677->6703->6709-
>6712->6747->6756->6771->6776->6778->6791->6810->6815->6855->6882->6894->6899-
```

```
>6921->6933->6944->6958->6982->6983->7009->7037->7063->7065->7095->7106->7142-
>7180->7215->7236->7253->7313->7322->7339->7351->7366->7424->7429->7487->7502-
>7505->7516->7519->7530->7538->7544->7549->7565->7585->7632->7654->7673->7704-
>7718->7737->7757->7775->7791->7792->7814->7821->7839->7842->7867->7928->7931-
>7989->8010->8031->8059->8078->8083->8177->8185->8202->8218->8231->8262->8272-
>8310->8338->8340->8372->8383->8394->8407->8410->8418->8489->8492->8521->8540-
>8585->8596->8609->8664->8675->8694->8719->8720->8736->8776->8778->8795->8840-
>8846->8874->8920->8951->8954->8957->8964->8969->8980->9014->9019->9049->9094-
>9095->9107->9176->9180->9182->9243->9248->9265->9271->9280->9283->9287->9292-
>9352->9354->9393->9394->9417->9433->9441->9442->9460->9484->9513->9516->9540-
>9582->9585->9612->9618->9628->9748->9758->9766->9767->9795->9808->9825->9846-
>9859->9868->9870->9880->9887->9891->9945->9990->9991->]
```