Ejercicio 1.

1. La función hash a utilizar será la sumatoria del número de abonado. A la que luego le haremos el módulo con el tamaño de la tabla para conocer su posición.
2. Al ser un sondeo cuadrático, el factor de carga debe ser 0,5.
    a. 8 elementos * 2 = tamaño 16
    b. el siguiente primo de 16 = **tamaño 17**

3.

Abonado	Código hash	Posición
María	21	4
Pedro	25	8
Juan	23	6
Laura	26	9
Lucía	19	2
Ana	29	12
José	28	11
Florencia	22	5

4.

Posición	Abonado	Comparaciones
0		
1		
2	Lucía	1
3		
4	María	1
5	Florencia	1
6	Juan	1
7		
8	Pedro	1
9	Laura	1

10		
11	José	1
12	Ana	1
13		
14		
15		
16		

TOTAL de comparaciones: 8
Promedio de comparaciones = 8 comparaciones / 8 inserciones = 1
Promedio de comparaciones = 1

4.

Lenguaje Natural:
Dada una clave, vamos a la posición de la tabla donde debería estar. Si la posición es vacía, el elemento no está. Si en la posición hay un elemento, nos fijamos si es igual al que estamos buscando. En caso que sea igual, el elemento está en la tabla. Si no es igual, calculamos la siguiente posición de la tabla donde podría estar el elemento. Repetimos esto hasta encontrar un elemento igual o una posición vacía.

Precondiciones:
La tabla asegura un f.c. máximo de 0,5 en todo momento.

Postcondiciones:
El tamaño de la tabla no se ve modificado.

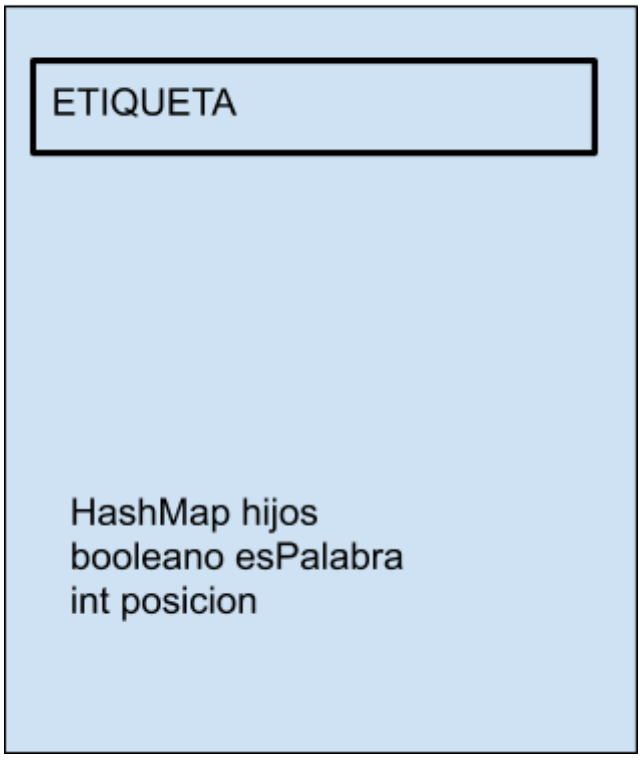
Pseudocódigo:

```
buscar(unaClave:int) : devuelve int
COMIENZO
      comparaciones <- 0
      claveHash0 <- funcionHashing(unaClave)
      claveHash <- claveHash0
      i <- 1
      SI (tabla[claveHash] ES nulo) ENTONCES
         DEVOLVER -1
      SINO ENTONCES
         MIENTRAS (tabla[claveHash] <> NULO) HACER
            comparaciones <- comparaciones +1
            SI (tabla[claveHash] == (unaClave)) ENTONCES
               devolver comparaciones;
            FIN SI
            claveHash <- claveHash0 + (i*i)
            i++
            MIENTRAS claveHash >= tabla.length HACER
               claveHash <- claveHash - tabla.length
            FIN MIENTRAS
            SI (comparaciones >= tabla.length) HACER
               DEVOLVER comparaciones * (-1)
            FIN SI
         FIN MIENTRAS
         DEVOLVER (comparaciones + 1) * (-1)
      FIN SI
   FIN


funcionHashing(unaClave:int) : devuelve int
COMIENZO
         DEVOLVER (unaClave/tabla.length) mod 17
FIN
```

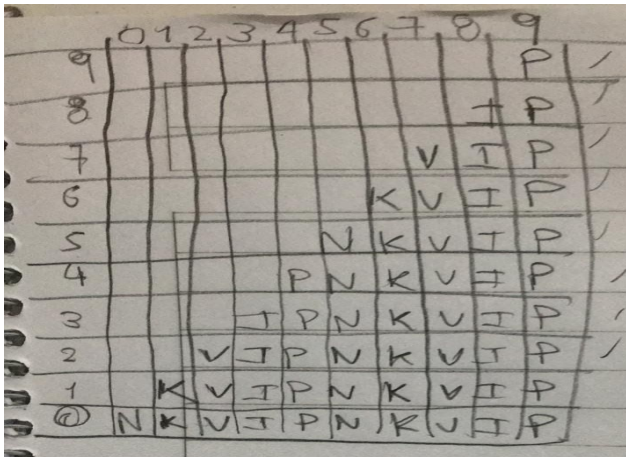
Ejercicio 2.

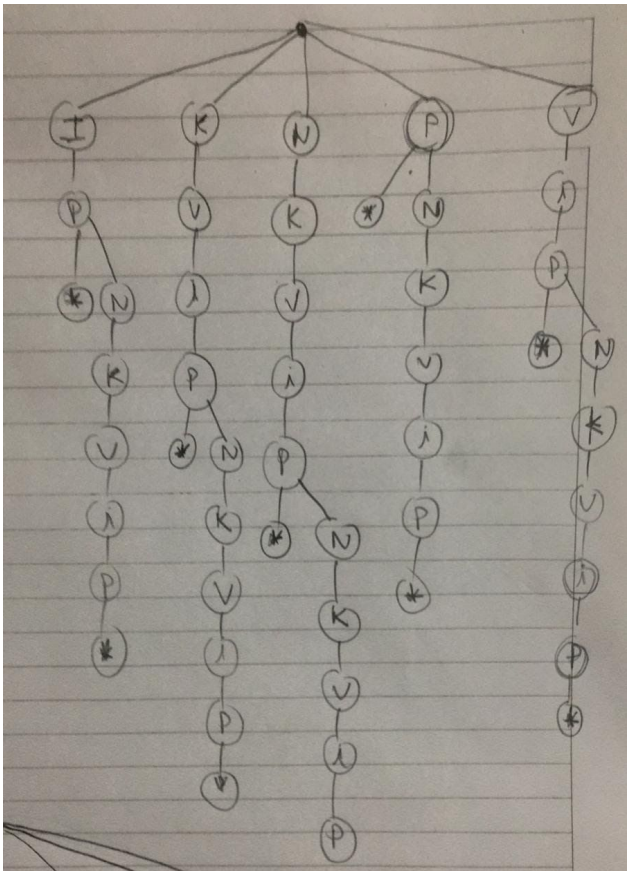
1.

NODO



2.
Imágenes





3.

Lenguaje Natural:

Dado el patrón a buscar, pararnos en el nodo del último caracter del patrón. A partir de ese nodo, bajar hacia todas sus hojas descendientes. En cada hoja encontraremos la posición en que se encuentra cada sufijo y por ende el patrón buscado.

Precondiciones:

El árbol de sufijos está creado correctamente.
El patrón está formado por caracteres del alfabeto.

Postcondiciones:

El árbol no fue modificado.

Método TArbol:

buscar(String palabra): devuelve una lista con las posiciones del patrón
COMIENZO
    SI (raiz <> NULO) ENTONCES
        devolver raiz.buscar(palabra);
    SI NO

```
        devolver nuevaLista
    FIN SI
FIN
```

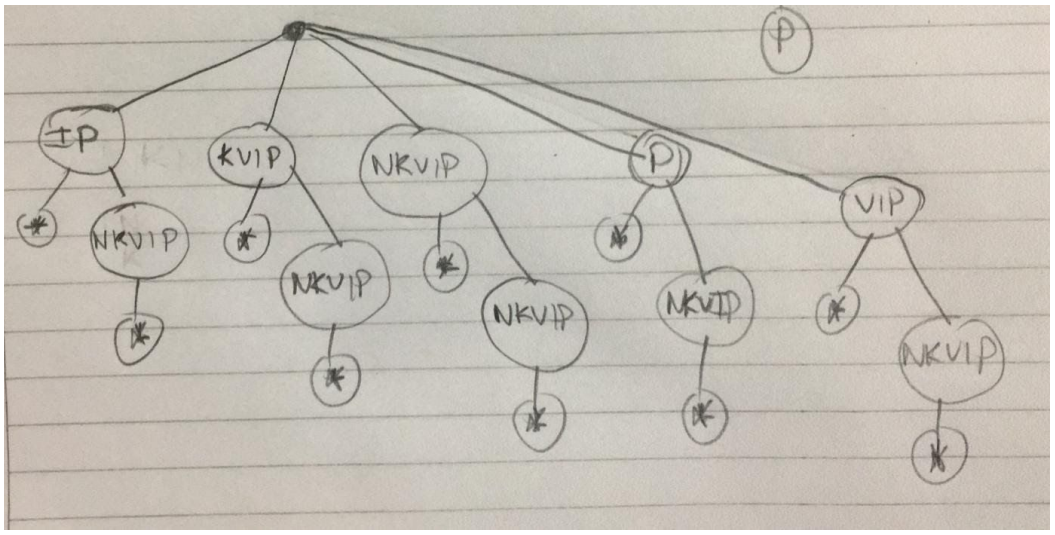
Método TNodo:

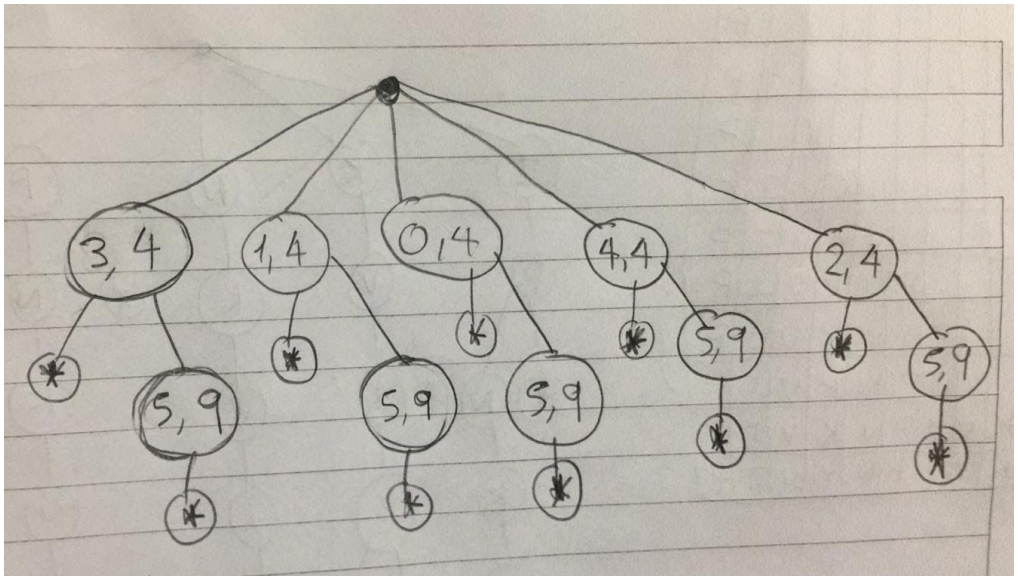
```
buscar(String patron): devuelve una lista con las posiciones del patrón
COMIENZO
    TNodoTrieSufijos nodo <- this                              O(1)
    PARA CADA caracter en PATRON HACER                         *c
      SI (!nodo.hijos.containsKey(caracter)) ENTONCES          O(1)
        devolver nuevaLista
      FIN SI
      nodo <- nodo.hijos(caracter)                             O(1)
    FIN PARA

    PARA CADA posicion EN nodo.posiciones HACER
      posicionesComienzo.agregar(posicion)
    FIN PARA
    devolver posicionesComienzo; O(1)
  FIN
```

4. Ejecución para “VI”
    a. Nos paramos en el nodo que contiene a “I”.
    b. A partir de “I” bajamos hasta encontrar la “P” (esPalabra es verdadero) del sufijo “VIP”
    c. La posición de “VIP” en la secuencia es a partir del 7.
    d. Continuamos bajando hasta encontrar la “P” (esPalabra es verdadero) del sufijo “VIPNKVIP”.
    e. La posición de “VIPNKVIP” en la secuencia es a partir de 2.
    f. Al finalizar tenemos dos posiciones de “VI” en la secuencia.

5. Aplicando la regla de la multiplicación, nos queda de O(1)*c = O(c).

6. Trie comprimido


P
IP
KVIP
NKVIP
P
VIP
NKVIP
NKVIP
NKVIP
NKVIP
NKVIP


3, 4
1, 4
0, 4
4, 4
2, 4
5, 9
5, 9
5, 9
5, 9
5, 9