Programación 3 – TUDAI (2024)
Trabajo Práctico Nº3: Estructuras de dispersión (Hashing)

EJERCICIO 1

Dado el conjunto de elementos: X= {68, 42, 47, 5, 76, 95, 23, 88, 90, 85, 31, 71, 60, 10, 46, 61, 50, 92, 74, 6, 97, 66, 1, 56, 27, 7, 14, 92} Realizar la inserción de los mismos en una estructura de Hashing con la que se especifica en cada punto, con las siguientes técnicas de tratamiento de desbordes, muestre gráficamente cómo se va armando la estructura y cómo queda luego de insertar hasta el último elemento:

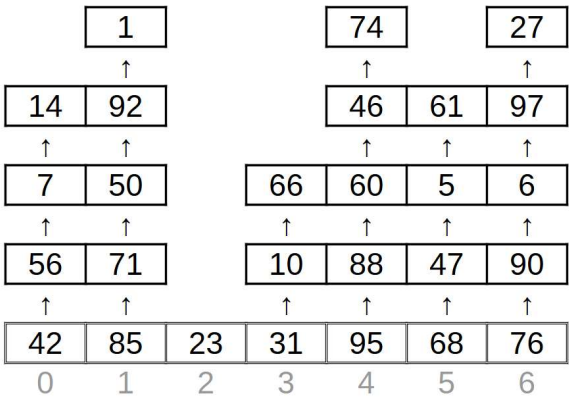
a) **Hashing separado (con M=7 y rp=1, rs=1).**
b) **Hashing separado con crecimiento (con el comportamiento de HashTable de JAVA) (con M=7, ρd=0,9).**

a. Hashing separado

M = 7
rp = 1
rs = 1

Función de hashing: h(x) = x % M → **h(x) = x % 7**

h(68) = 5	h(88) = 4	h(46) = 4	h(66) = 3
h(42) = 0	h(90) = 6	h(61) = 5	h(1) = 1
h(47) = 5	h(85) = 1	h(50) = 1	h(56) = 0
h(5) = 5	h(31) = 3	h(92) = 1	h(27) = 6
h(76) = 6	h(71) = 1	h(74) = 4	h(7) = 0
h(95) = 4	h(60) = 4	h(6) = 6	h(14) = 0
h(23) = 2	h(10) = 3	h(97) = 6	h(92) = 1 (*)



(*) Las claves repetidas no se insertan en la tabla.

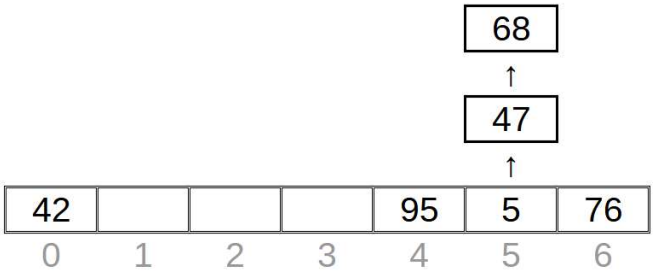
b. Hashing separado con crecimiento

M = 7
ρd = 0.9
rp = 1

L = floor(M * rp * ρd)
= floor(7 * 1 * 0.9)
= floor(6.3)

L = 6

h(68) = 5 h(42) = 0 h(47) = 5 h(5) = 5 h(76) = 6 h(95) = 4 (*)



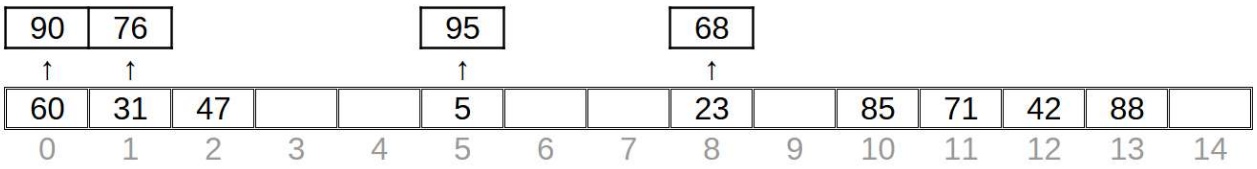
(*) Se alcanza el límite de crecimiento L, por lo que se llama al método rehash().

M = M * 2 + 1
M = 7 * 2 + 1
M = 15

h(x) = x % 15

L = floor(M * rp * ρd)
= floor(15 * 1 * 0.9)
= floor(13.5)
L = 13

h(42) = 12 h(76) = 1 h(31) = 1
h(95) = 5 h(23) = 8 h(71) = 11
h(68) = 8 h(88) = 13 h(60) = 0 (*)
h(47) = 2 h(90) = 0
h(5) = 5 h(85) = 10



(*) Se alcanza el límite de crecimiento L, por lo que se llama al método rehash().

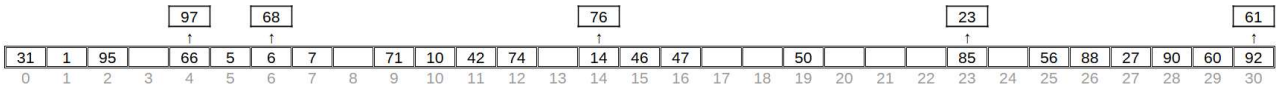
M = M * 2 + 1
M = 15 * 2 + 1
M = 31

h(x) = x % 31

L = floor(M * rp * ρd)
= floor(31 * 1 * 0.9)
= floor(27,9)

L = 27

h(60) = 29	h(23) = 23	h(46) = 15	h(66) = 4
h(90) = 28	h(68) = 6	h(61) = 30	h(1) = 1
h(31) = 0	h(85) = 23	h(50) = 19	h(56) = 25
h(76) = 14	h(71) = 9	h(92) = 30	h(27) = 27
h(47) = 16	h(42) = 11	h(74) = 12	h(7) = 7
h(5) = 5	h(88) = 26	h(6) = 6	h(14) = 14 (*)
h(95) = 2	h(10) = 10	h(97) = 4	



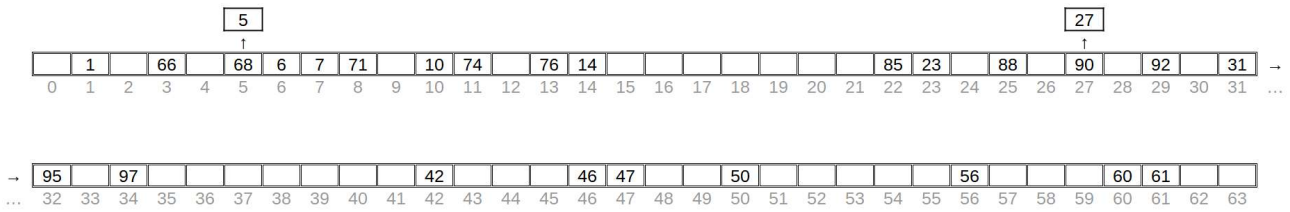
(*) Se alcanza el límite de crecimiento L, por lo que se llama al método rehash().

M = M * 2 + 1
M = 31 * 2 + 1
M = 63

h(x) = x % 63

L = floor(M * rp * ρd)
= floor(63 * 1 * 0.9)
= floor(56.7)
L = 56

h(31) = 31	h(68) = 23	h(46) = 46	h(27) = 27
h(1) = 1	h(7) = 7	h(47) = 47	h(90) = 27
h(95) = 32	h(71) = 8	h(50) = 50	h(60) = 60
h(66) = 3	h(10) = 10	h(85) = 22	h(92) = 29
h(97) = 34	h(74) = 11	h(23) = 23	h(61) = 61
h(5) = 5	h(14) = 14	h(56) = 56	h(92) = 29 (*)
h(6) = 6	h(76) = 13	h(88) = 25	



(*) Las claves repetidas no se insertan en la tabla.

EJERCICIO 2
Conteste si es posible o no y justifique:

a) **Si se quisiera listar en orden todas las claves almacenadas en una estructura de hashing, ¿se podría? ¿Cómo? ¿Es la estructura más adecuada?**
Listar en orden todas las claves almacenadas en una tabla de hashing es posible. Para lograrlo se debe recorrer toda la estructura, guardando cada uno de sus elementos en una estructura auxiliar (como un arreglo de igual tamaño que la cantidad de elementos que contenga la tabla) para luego ordenarlos mediante algún algoritmo como Merge Sort o Quick Sort. Para esto se debe:
    1. Recorrer cada una de las ranuras del primer balde de la estructura primaria, copiando su clave en la estructura auxiliar.
    2. Cuando se termina de recorrer el balde, continuar al siguiente en la lista vinculada (estructura secundaria).
    3. Cuando se termina de recorrer la lista, avanzar al siguiente balde de la estructura principal y repetir el proceso.
    4. Cuando se recorre toda la tabla y se copian todos los elementos, ordenar la estructura auxiliar.

    Este procedimiento es muy ineficiente, ya que requiere recorrer cada balde (con sus ranuras) de la estructura primaria y secundaria, resultando en una complejidad temporal de $O(n^2)$, siendo n la cantidad de ranuras de la estructura (rp y rs).

b) **¿Qué tipos de servicios resuelve un hashing? ¿Es posible responder, por ejemplo, "la lista de todos los alumnos que obtuvieron una nota mayor que x en un curso dado"?**
Las estructuras de hashing resuelven los siguientes servicios:
    - Búsqueda eficiente: Dada una clave de búsqueda (como un número, un texto u otro tipo de dato), una estructura de hashing puede recuperar rápidamente el valor asociado sin necesidad de buscar secuencialmente sobre cada elemento. Su complejidad temporal busca ser constante (O(1)), o cercano.
    - Almacenamiento eficiente: Una estructura de hashing puede almacenar grandes cantidades de datos de manera eficiente, distribuyendo los datos en una tabla hash utilizando funciones de dispersión (funciones de hashing) que permite un acceso rápido.
    - Evitar duplicados: Al poder acceder rápidamente a los elementos guardados en la tabla, es posible comprobar de manera muy eficiente si el valor que se desea insertar ya se encuentra en la estructura. En dicho caso, se anulará la operación (no pueden repetirse claves de búsqueda, ya que esto alteraría su eficiencia).
    - Seguridad: En criptografía, las funciones de hashing se utilizan para proteger contraseñas y otros datos confidenciales almacenando sus valores hash en lugar de los datos reales.

    Por otro lado, estas estructuras no son las más adecuadas si lo que se desea es listar sus elementos respetando un orden o rango determinado (como en el ejemplo citado en la consigna).

EJERCICIO 3

Se está desarrollando una aplicación que almacena los datos y el saldo de las tarjetas de compra de comida del comedor de una universidad. Cada cliente es identificado por su número de DNI, y se poseen además sus datos personales y de la carrera que estudia: DNI, nombre, apellido, fecha de nacimiento, domicilio, CP de su ciudad de origen, saldo de la cuenta y nombre de la carrera que estudia.

Se quiere:

a) **Dado un DNI de cliente, responder el saldo de su cuenta.**
b) **Imprimir un listado de nombres y apellidos de todos los clientes que tienen en su saldo de cuenta menos de un valor X dado.**
c) **Dado un código postal, listar todos los clientes que provengan de esa ciudad.**

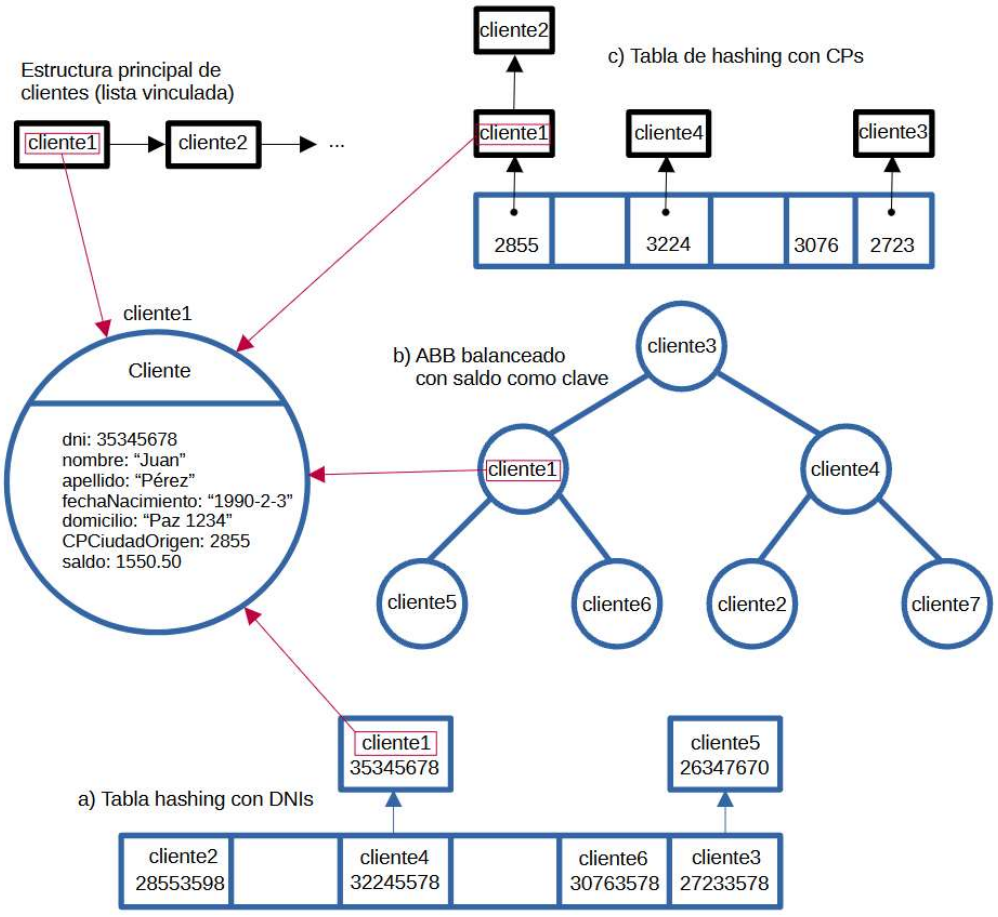
Proponga y describa qué estructuras de datos utilizaría para responder eficientemente a los servicios pedidos. Muestre gráficamente cómo se relacionan.

a) La estructura principal que almacenará todos los objetos de tipo `Cliente` podría ser una lista vinculada. Cada cliente tendrá como atributos: `dni`, `nombre`, `apellido`, `fechaNacimiento`, `domicilio`, `CPCiudadOrigen`, `saldo` y `carrera`.

b) a) Puede utilizarse una tabla de hashing que utilice como clave de búsqueda los DNI de los clientes, y asociado a ella un objeto `Cliente`. Esto permitirá una búsqueda rápida de tiempo constante o cercano.

c) b) Se puede emplear un árbol binario de búsqueda balanceado (como AVL o árbol Rojo-Negro) que contenga en sus nodos referencias a objetos `Cliente`, ordenados por saldos de cuenta. De esta forma, es posible listar clientes con un saldo específico, dentro de un rango dado, o incluso ordenarlos.

c) Puede emplearse otra tabla de hashing que utilice como clave de búsqueda los CP de las ciudades de origen y se asocie a cada balde de la estructura primaria una lista vinculada con los clientes de igual ciudad de origen.



EJERCICIO 4

Se desea desarrollar una aplicación para mejorar la atención de una biblioteca en cuanto a la búsqueda de libros dentro del catálogo disponible. Cada libro estará compuesto por un identificador único y datos propios de los libros (título, autor, géneros, año de publicación, cantidad de ejemplares, etc.).
Se sabe, además, que los libros nuevos se agregan al catálogo en horarios fuera de la atención al público.
Se desean proveer los siguientes servicios:

a) **Obtener la cantidad de ejemplares de un libro dado su identificador único.**
b) **Obtener todos los libros de un género dado.**
c) **Obtener todos los libros publicados entre dos años de publicación dados.**

Responda y justifique:
1) ¿Qué estructura de datos utilizaría para almacenar todos los libros en memoria dentro de la aplicación?
2) ¿Cómo resolvería cada uno de los servicios solicitados? ¿Utilizaría alguna estructura adicional de acceso para mejorar el costo de respuesta de cada servicio?

1) Para almacenar todos los libros en memoria podría utilizarse una tabla de hashing ya que permite una rápida inserción, búsqueda y eliminación de elementos. Como clave de búsqueda podría usarse el identificador único de cada libro, y como valor una referencia a un objeto de tipo `Libro` con todos sus atributos (`titulo`, `autor`, `generos`, `anioPublicacion`, `cantidadEjemplares`, etc.).

2) Para resolver cada uno de los servicios solicitados podrían utilizarse estructuras auxiliares como las siguientes:

d) Para obtener la cantidad de ejemplares de un libro, podría utilizarse la estructura principal (tabla de hashing) utilizando como clave de búsqueda el identificador único. Si el libro es encontrado, se devuelve el valor de su atributo `cantidadEjemplares`.
e) Para obtener todos los libros de un género dado puede utilizarse otra tabla de hashing, pero utilizando como clave de búsqueda el nombre del género deseado (por ejemplo “Aventura”). A cada una de esas claves podría asociarse una lista vinculada con las referencias a los libros de dicho género.
f) Para obtener todos los libros publicados entre dos años dados, podría utilizarse un árbol binario de búsqueda balanceado (como AVL o árbol Rojo-Negro) que utilice como clave de bifurcación el año de publicación de cada libro. Esta estructura puede resultar ventajosa ya que permite acceder a sus elementos en rangos determinados y, si se lo desea, de forma ordenada.

Programación 3 – TUDAI (2024)
Trabajo Práctico Nº3: Estructuras de dispersión (Hashing)

EJERCICIO 1

Dado el conjunto de elementos: X= {68, 42, 47, 5, 76, 95, 23, 88, 90, 85, 31, 71, 60, 10, 46, 61, 50, 92, 74, 6, 97, 66, 1, 56, 27, 7, 14, 92} Realizar la inserción de los mismos en una estructura de Hashing con la que se especifica en cada punto, con las siguientes técnicas de tratamiento de desbordes, muestre gráficamente cómo se va armando la estructura y cómo queda luego de insertar hasta el último elemento:

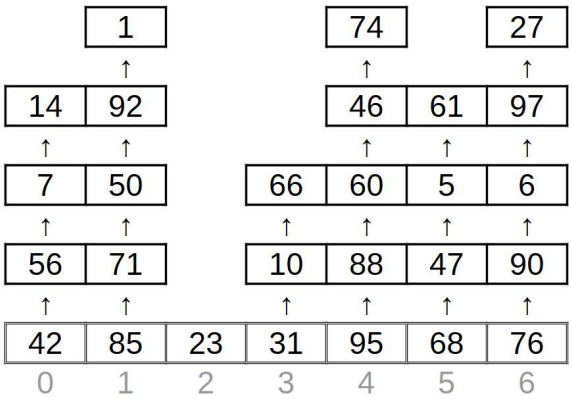
a) **Hashing separado (con M=7 y rp=1, rs=1).**
b) **Hashing separado con crecimiento (con el comportamiento de HashTable de JAVA) (con M=7, ρd=0,9).**

a. Hashing separado

M = 7
rp = 1
rs = 1

Función de hashing: h(x) = x % M → **h(x) = x % 7**

h(68) = 5	h(88) = 4	h(46) = 4	h(66) = 3
h(42) = 0	h(90) = 6	h(61) = 5	h(1) = 1
h(47) = 5	h(85) = 1	h(50) = 1	h(56) = 0
h(5) = 5	h(31) = 3	h(92) = 1	h(27) = 6
h(76) = 6	h(71) = 1	h(74) = 4	h(7) = 0
h(95) = 4	h(60) = 4	h(6) = 6	h(14) = 0
h(23) = 2	h(10) = 3	h(97) = 6	h(92) = 1 (*)



(*) Las claves repetidas no se insertan en la tabla.

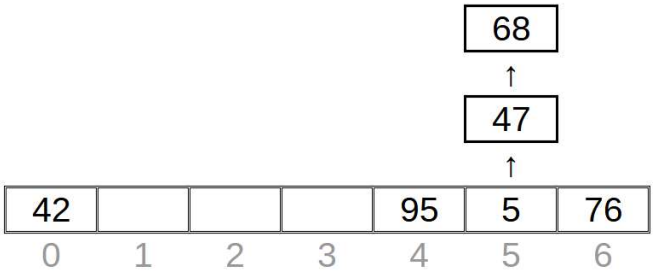
b. Hashing separado con crecimiento

M = 7
ρd = 0.9
rp = 1

L = floor(M * rp * ρd)
= floor(7 * 1 * 0.9)
= floor(6.3)

L = 6

h(68) = 5 h(42) = 0 h(47) = 5 h(5) = 5 h(76) = 6 h(95) = 4 (*)



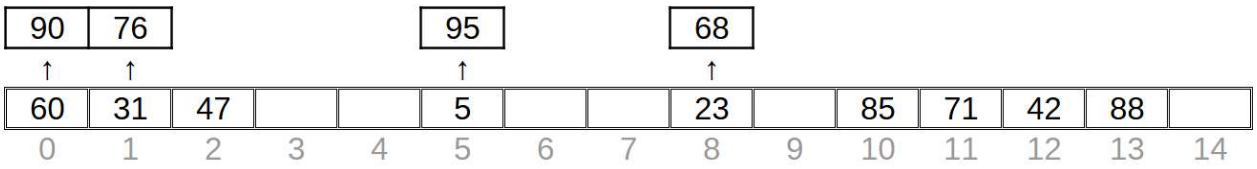
(*) Se alcanza el límite de crecimiento L, por lo que se llama al método rehash().

M = M * 2 + 1
M = 7 * 2 + 1
M = 15

h(x) = x % 15

L = floor(M * rp * ρd)
= floor(15 * 1 * 0.9)
= floor(13.5)
L = 13

h(42) = 12 h(76) = 1 h(31) = 1
h(95) = 5 h(23) = 8 h(71) = 11
h(68) = 8 h(88) = 13 h(60) = 0 (*)
h(47) = 2 h(90) = 0
h(5) = 5 h(85) = 10



(*) Se alcanza el límite de crecimiento L, por lo que se llama al método rehash().

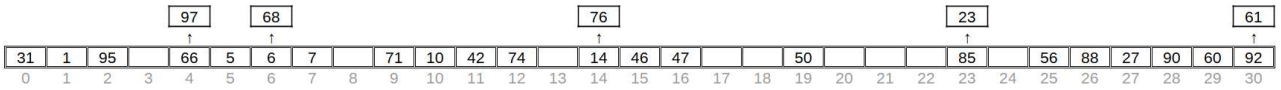
M = M * 2 + 1
M = 15 * 2 + 1
M = 31

h(x) = x % 31

L = floor(M * rp * ρd)
= floor(31 * 1 * 0.9)
= floor(27,9)

L = 27

h(60) = 29	h(23) = 23	h(46) = 15	h(66) = 4
h(90) = 28	h(68) = 6	h(61) = 30	h(1) = 1
h(31) = 0	h(85) = 23	h(50) = 19	h(56) = 25
h(76) = 14	h(71) = 9	h(92) = 30	h(27) = 27
h(47) = 16	h(42) = 11	h(74) = 12	h(7) = 7
h(5) = 5	h(88) = 26	h(6) = 6	h(14) = 14 (*)
h(95) = 2	h(10) = 10	h(97) = 4	



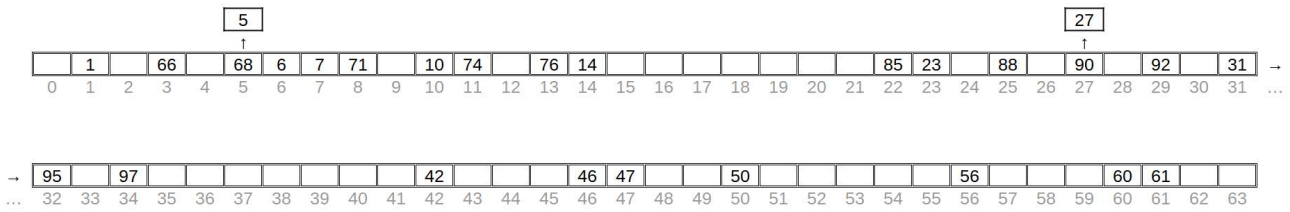
(*) Se alcanza el límite de crecimiento L, por lo que se llama al método rehash().

M = M * 2 + 1
M = 31 * 2 + 1
M = 63

h(x) = x % 63

L = floor(M * rp * ρd)
= floor(63 * 1 * 0.9)
= floor(56.7)
L = 56

h(31) = 31	h(68) = 23	h(46) = 46	h(27) = 27
h(1) = 1	h(7) = 7	h(47) = 47	h(90) = 27
h(95) = 32	h(71) = 8	h(50) = 50	h(60) = 60
h(66) = 3	h(10) = 10	h(85) = 22	h(92) = 29
h(97) = 34	h(74) = 11	h(23) = 23	h(61) = 61
h(5) = 5	h(14) = 14	h(56) = 56	h(92) = 29 (*)
h(6) = 6	h(76) = 13	h(88) = 25	



(*) Las claves repetidas no se insertan en la tabla.

EJERCICIO 2
Conteste si es posible o no y justifique:

a) **Si se quisiera listar en orden todas las claves almacenadas en una estructura de hashing, ¿se podría? ¿Cómo? ¿Es la estructura más adecuada?**
Listar en orden todas las claves almacenadas en una tabla de hashing es posible. Para lograrlo se debe recorrer toda la estructura, guardando cada uno de sus elementos en una estructura auxiliar (como un arreglo de igual tamaño que la cantidad de elementos que contenga la tabla) para luego ordenarlos mediante algún algoritmo como Merge Sort o Quick Sort. Para esto se debe:
    1. Recorrer cada una de las ranuras del primer balde de la estructura primaria, copiando su clave en la estructura auxiliar.
    2. Cuando se termina de recorrer el balde, continuar al siguiente en la lista vinculada (estructura secundaria).
    3. Cuando se termina de recorrer la lista, avanzar al siguiente balde de la estructura principal y repetir el proceso.
    4. Cuando se recorre toda la tabla y se copian todos los elementos, ordenar la estructura auxiliar.

    Este procedimiento es muy ineficiente, ya que requiere recorrer cada balde (con sus ranuras) de la estructura primaria y secundaria, resultando en una complejidad temporal de $O(n^2)$, siendo n la cantidad de ranuras de la estructura (rp y rs).

b) **¿Qué tipos de servicios resuelve un hashing? ¿Es posible responder, por ejemplo, "la lista de todos los alumnos que obtuvieron una nota mayor que x en un curso dado"?**
Las estructuras de hashing resuelven los siguientes servicios:
    - Búsqueda eficiente: Dada una clave de búsqueda (como un número, un texto u otro tipo de dato), una estructura de hashing puede recuperar rápidamente el valor asociado sin necesidad de buscar secuencialmente sobre cada elemento. Su complejidad temporal busca ser constante (O(1)), o cercano.
    - Almacenamiento eficiente: Una estructura de hashing puede almacenar grandes cantidades de datos de manera eficiente, distribuyendo los datos en una tabla hash utilizando funciones de dispersión (funciones de hashing) que permite un acceso rápido.
    - Evitar duplicados: Al poder acceder rápidamente a los elementos guardados en la tabla, es posible comprobar de manera muy eficiente si el valor que se desea insertar ya se encuentra en la estructura. En dicho caso, se anulará la operación (no pueden repetirse claves de búsqueda, ya que esto alteraría su eficiencia).
    - Seguridad: En criptografía, las funciones de hashing se utilizan para proteger contraseñas y otros datos confidenciales almacenando sus valores hash en lugar de los datos reales.

    Por otro lado, estas estructuras no son las más adecuadas si lo que se desea es listar sus elementos respetando un orden o rango determinado (como en el ejemplo citado en la consigna).

EJERCICIO 3

Se está desarrollando una aplicación que almacena los datos y el saldo de las tarjetas de compra de comida del comedor de una universidad. Cada cliente es identificado por su número de DNI, y se poseen además sus datos personales y de la carrera que estudia: DNI, nombre, apellido, fecha de nacimiento, domicilio, CP de su ciudad de origen, saldo de la cuenta y nombre de la carrera que estudia.

Se quiere:

a) **Dado un DNI de cliente, responder el saldo de su cuenta.**
b) **Imprimir un listado de nombres y apellidos de todos los clientes que tienen en su saldo de cuenta menos de un valor X dado.**
c) **Dado un código postal, listar todos los clientes que provengan de esa ciudad.**

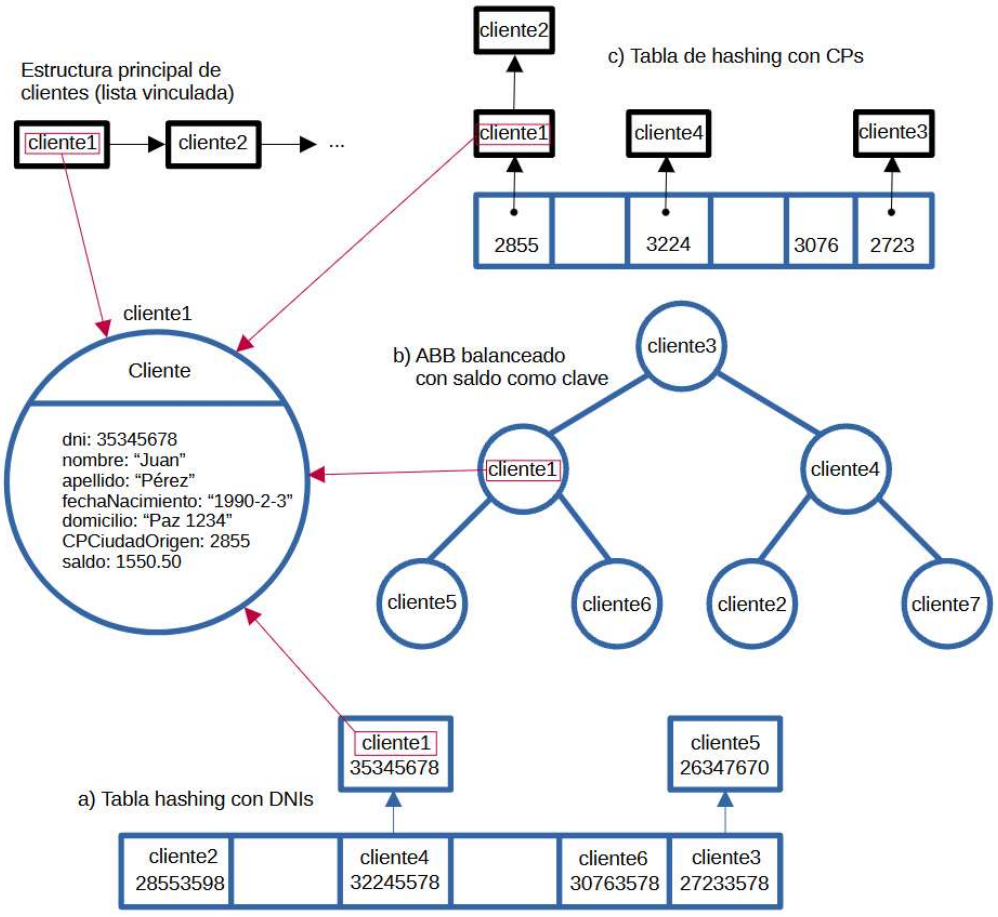
Proponga y describa qué estructuras de datos utilizaría para responder eficientemente a los servicios pedidos. Muestre gráficamente cómo se relacionan.

a) La estructura principal que almacenará todos los objetos de tipo `Cliente` podría ser una lista vinculada. Cada cliente tendrá como atributos: `dni`, `nombre`, `apellido`, `fechaNacimiento`, `domicilio`, `CPCiudadOrigen`, `saldo` y `carrera`.

b) a) Puede utilizarse una tabla de hashing que utilice como clave de búsqueda los DNI de los clientes, y asociado a ella un objeto `Cliente`. Esto permitirá una búsqueda rápida de tiempo constante o cercano.

c) b) Se puede emplear un árbol binario de búsqueda balanceado (como AVL o árbol Rojo-Negro) que contenga en sus nodos referencias a objetos `Cliente`, ordenados por saldos de cuenta. De esta forma, es posible listar clientes con un saldo específico, dentro de un rango dado, o incluso ordenarlos.

c) Puede emplearse otra tabla de hashing que utilice como clave de búsqueda los CP de las ciudades de origen y se asocie a cada balde de la estructura primaria una lista vinculada con los clientes de igual ciudad de origen.



EJERCICIO 4

Se desea desarrollar una aplicación para mejorar la atención de una biblioteca en cuanto a la búsqueda de libros dentro del catálogo disponible. Cada libro estará compuesto por un identificador único y datos propios de los libros (título, autor, géneros, año de publicación, cantidad de ejemplares, etc.).
Se sabe, además, que los libros nuevos se agregan al catálogo en horarios fuera de la atención al público.
Se desean proveer los siguientes servicios:

a) **Obtener la cantidad de ejemplares de un libro dado su identificador único.**
b) **Obtener todos los libros de un género dado.**
c) **Obtener todos los libros publicados entre dos años de publicación dados.**

Responda y justifique:
1) ¿Qué estructura de datos utilizaría para almacenar todos los libros en memoria dentro de la aplicación?
2) ¿Cómo resolvería cada uno de los servicios solicitados? ¿Utilizaría alguna estructura adicional de acceso para mejorar el costo de respuesta de cada servicio?

1) Para almacenar todos los libros en memoria podría utilizarse una tabla de hashing ya que permite una rápida inserción, búsqueda y eliminación de elementos. Como clave de búsqueda podría usarse el identificador único de cada libro, y como valor una referencia a un objeto de tipo `Libro` con todos sus atributos (`titulo`, `autor`, `generos`, `anioPublicacion`, `cantidadEjemplares`, etc.).

2) Para resolver cada uno de los servicios solicitados podrían utilizarse estructuras auxiliares como las siguientes:

d) Para obtener la cantidad de ejemplares de un libro, podría utilizarse la estructura principal (tabla de hashing) utilizando como clave de búsqueda el identificador único. Si el libro es encontrado, se devuelve el valor de su atributo `cantidadEjemplares`.
e) Para obtener todos los libros de un género dado puede utilizarse otra tabla de hashing, pero utilizando como clave de búsqueda el nombre del género deseado (por ejemplo “Aventura”). A cada una de esas claves podría asociarse una lista vinculada con las referencias a los libros de dicho género.
f) Para obtener todos los libros publicados entre dos años dados, podría utilizarse un árbol binario de búsqueda balanceado (como AVL o árbol Rojo-Negro) que utilice como clave de bifurcación el año de publicación de cada libro. Esta estructura puede resultar ventajosa ya que permite acceder a sus elementos en rangos determinados y, si se lo desea, de forma ordenada.