Universidad Internacional de Valencia (VIU)

15GIIN – Estructuras de Datos y Algoritmos

Quinto Portafolio – ACT5

Alumno: Gagliardo Miguel Angel

Ejercicio 1

b)

Lineal:

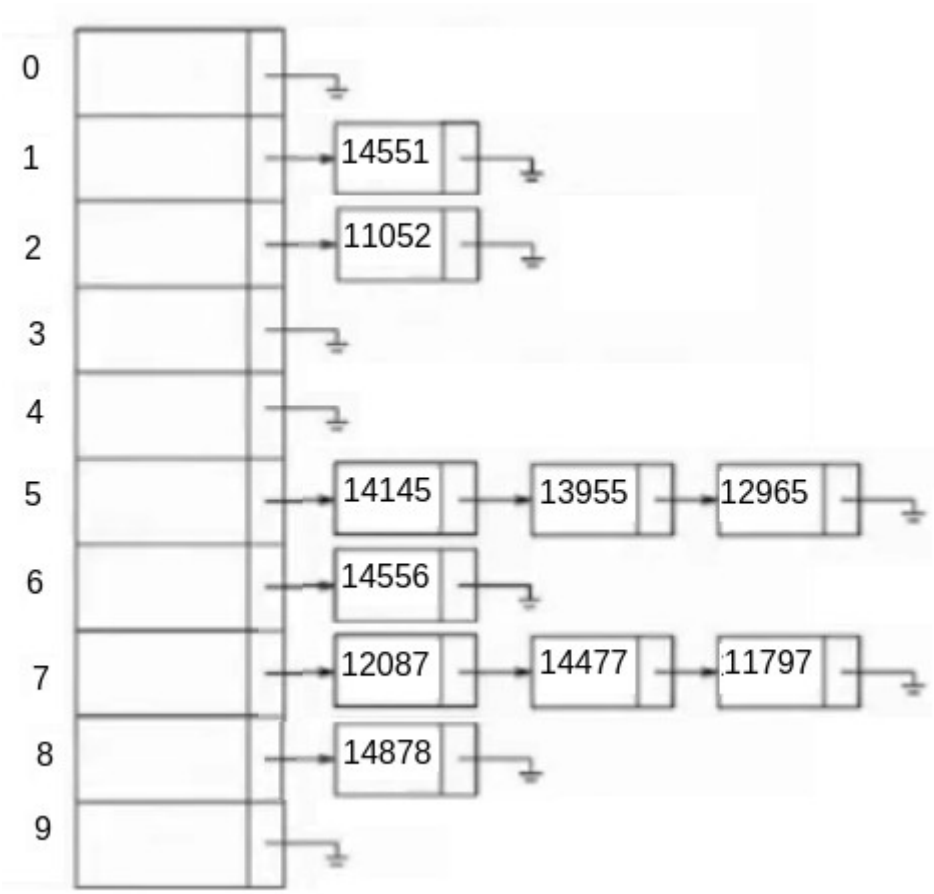
	Empty Table	After 14145	After 12087	After 14477	After 13955	After 11797	After 14878	After 14556	After 14551	After 12965	After 11052	Final
0							14878					**14878**
1								14556				**14556**
2									14551			**14551**
3										12965		**12965**
4											11052	**11052**
5		14145										**14145**
6					13955							**13955**
7			12087									**12087**
8				14477								**14477**
9						11797						**11797**

Cuadratica:

	Empty Table	After 14145	After 12087	After 14477	After 13955	After 11797	After 14878	After 14556	After 14551	After 12965	After 11052
0								14556			
1						11797					
2									14551		
3											11052
4										12965	
5		14145									
6					13955						
7			12087								
8				14477							
9							14878				

c)

Formato grafico:



Formato Tabla:

	Empty Table	After 14145	After 12087	After 14477	After 13955	After 11797	After 14878	After 14556	After 14551	After 12965	After 11052	Final	
0													**0**
1									14551			**14551**	**1**
2											11052	**11052**	**2**
3													**3**
4													**4**
5		14145			13955					12965		**12965**	**5**
6								14556				**14556**	**6**
7			12087	14477		11797						**11797**	**7**
8							14878					**14878**	**8**
9													**9**

d) Para el caso de hash cerrado y para este ejemplo, la cantidad de elementos en la tabla es 10 y el tamaño de la tabla tambien es 10, por tanto el promedio seria 10/10 = 1.

Para el caso de hash abierto sin embargo y para este mismo ejemplo, tenemos que manejar listas como se ve puede ver en la tabla, lo cual quiere decir que no solo tengo que recorrer toda la tabla una vez, si no tambien ingresar en cada nodo (lista) y recorrer la lista para verificar si el elemento se encuentra alli. Para nuestro ejemplo seran 14/10 = 1.4
Normalmente para hashing abierto se recomienda aumentar el tamaño de la tabla (usualmente doblar el tamaño), lo cual quiere decir que necesitamos mas espacio en memoria (mas costoso) para poder tener un mejor factor de carga, sin embargo para hash cerrado el tamaño de la tabla se va ocupando como un todo a medida que se van ingresando los elementos 1 en 1, lo cual

quiere decir un mayor tiempo de computo (para buscar los elementos sea utilizando un factor lineal o cuadratico) pero un mejor uso de la memoria. Pienso que el caso del hash cerrado es mejor.

Nota: Para ambos casos se da por descontado que O(1) es la complejidad de obtener el hash de un elemento.

2)

[10, 9, 8, 6, **5**, 9, 8, 6, 5, 1]; i = 10, j = 1, pivot = (0+9)/2 = 4

--

Intercambio i por j dado que 1 < 10:

[1, 9, 8, 6, **5**, 9, 8, 6, 5, 10]; **i = 1, j = 10, pivot = 5**

--

[1, **9**, 8, 6, **5**, 9, 8, 6, **5**, 10];
i=9 dado que no es menor que el pivot
j=5 dado que no es mayor estricto que el pivot
pivot=5

--

Intercambio el numero de la posicion i por el de la posicion j, y me muevo i+1 y j-1:

[1, 5, **8**, 6, **5**, 9, 8, **6**, 9, 10]; Ahora i=8; j=6; pivot = 5

[1, 5, **8**, 6, **5**, 9, 8, 6, 9, 10];
i=8 dado que no es menor que el pivot
Ahora j=5 dado que antes j=6 (es mayor que el pivot) entonces nos movemos a la posicion j-1 (8), que es mayor que el pivot, decremento una vez mas j (ahora 9) que, otra vez, es mayor que el pivot, me muevo una vez mas j-1 (ahora 5) que es igual al pivot.

Intercambio una vez mas i por j, entonces:

[1, 5, 5, 6, 8, 9, 8, 6, 9, 10];

Ahora una vez decrementado j-1 y aumentado i+1 nos queda:

i=6
j=5

Dado que se han cruzado, quiere decir que los elementos que estan a la izquierda de i (1, 5, 5) son menores o iguales al pivote, y desde i en adelante(6, 8, 9, 8, 6, 9, 10) son mayores o iguales al pivote.
Ahora con el subarray izquierdo (1,5,5) pasamos a ordenar recursivamente:

[1,5,5]; left=1; center=5; right=5

1) Ordenamos left, right y center: [1,5,5]. i=1, pivote=5, j=5

2) 1 es menor que el pivote, por lo cual ahora i=5 (el mismo que el pivote). Por otro lado j=5 **no** es mayor estricto que el pivote, por lo tanto se queda alli.

[1,5,5]: i=5, pivote=5 y j=5

3) Luego se cruzan los indices:

[1,5,5]: j=1, pivote=5 e i=5.

Recursivamente se ordenaran 1 y 5, en ambos casos se aplica INSERTION SORT con 1 solo elemento y el array queda ordenado.

Finalmente el subarray izquierdo nos queda identico ya estaba ordenado en un principio: [1,5,5].