Hoja de Trabajo HT-M2-2 – Solución

Sección 4 - EDA 2025-20

Preguntas de tablas de símbolos

1. Considere una tabla de hash que utiliza encadenamiento separado para manejar las colisiones. ¿En qué situación la búsqueda de una llave en la tabla podría tener una complejidad temporal de O(N)?

A. Cuando la mayoría de las llaves se distribuyen uniformemente entre los buckets, minimizando las colisiones.
B. Cuando se requiere iterar a través de varios buckets vacíos antes de encontrar el bucket que contiene la llave buscada.
C. Cuando un solo bucket contiene todas las llaves y la llave buscada no existe.
D. Cuando es necesario hacer rehashing de la tabla debido a un aumento en el factor de carga.

2. ¿Cuál de las siguientes afirmaciones es FALSA respecto a una tabla de hash que utiliza el encadenamiento separado como método de resolución de colisiones, considerando una distribución uniforme de las llaves?

A. Elegir el tamaño de la tabla como un número primo ayuda a reducir las colisiones y a la distribución uniforme de las llaves en los buckets.
B. La eficiencia de la búsqueda de elementos depende de la distribución uniforme de las llaves en los diferentes buckets.
C. Es posible que algunos buckets permanezcan vacíos o no tan llenos mientras otros contienen más elementos.
D. La complejidad temporal de la eliminación interna de una dupla (llave-valor) es independiente de la cantidad de elementos en los buckets.

3. Suponga que se requiere insertar en una tabla de Hash las siguientes llaves, en ese orden: `"S", "H", "I", "O", "B", "L", "J", "E".` Se usa como función de hash: `hash(s) = ASCII(c) % N`.
- `ASCII(c)` corresponde al valor ASCII del carácter `c` (ver la tabla abajo).
- `N` es el tamaño de la tabla, que es igual a 5.

Letra	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
ASCII	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
ASCII % 5		1			4			2	3	4		1			4				3							

¿Cuál es el resultado de la tabla si se utiliza el método de Separate Chaining, sin contemplar factor de carga ni rehashing? Nota: El guión —, significa vacío.

A.
0: B → L —
1: —
2: H
3: S → I → H
4: O → J → E

B.
0: —
1: B → L
2: H
3: S → I
4: O → J → E

C.
0: —
1: B → L
2: H
3: I → S
4: O → J → E

D. Ninguna de las anteriores.

4. Considere una tabla de hash con manejo de colisiones mediante encadenamiento separado, de tamaño 50 que almacena 400 elementos, ¿cuál es el factor de carga de esa tabla de hash?

Factor de carga $\alpha = N/M$
N: el número de pares <llave, valor>
M: tamaño de la tabla (preferiblemente un número primo)

A. 80
B. 0.125
C. 8
D. 50

5. ¿Cuántas comparaciones de llaves se necesitan hacer, en el peor caso, para buscar una llave en una tabla de hash que utiliza el encadenamiento separado, si se insertan 4 llaves distintas con el mismo hash (índice en el arreglo subyacente) H1, luego se insertan 6 llaves nuevas y distintas todas con el mismo hash H2, y finalmente se insertan 2 llaves más, también distintas entre sí y con un hash H3 que es diferente a H1 y a H2?

A. 4
B. 6
C. 2
D. 12

6. Considere una tabla de hash vacía con capacidad M = 7, que maneja colisiones mediante Separate Chaining. Se define un factor de carga máximo de N/M = 0.5, con N siendo el número de llaves. La política de rehashing aumenta el tamaño de la tabla al siguiente número primo mayor que el doble del tamaño actual (17). La función de inserción asegura que el factor de carga no se sobrepase. Se planea agregar el conjunto de llaves `{"B", "D", "F", "J", "N", "P"}` en el orden proporcionado. Los valores de hash(...) correspondientes a cada llave son: `hash("B") = 66, hash("D") = 68, hash("F") = 70, hash("J") = 74, hash("N") = 78, hash("P") = 80`. La función hashValue(key) es: `abs(hash(key)) % M` y se aplica para determinar la posición inicial de cada llave. ¿Cuál será el estado final de la tabla hash después de insertar todas las llaves?

A. M = 17; 0: D, 2: F, 6: J, 10: N, 12: P, 15: B, Resto vacío
B. M = 17; 0: D, J, P, 2: F, 6: B, N, Resto vacío
C. M = 17; 0: P, 1: B, 2: D, 3: F, 5: J, 8: N, Resto vacío
D. M = 17; 0: D, 2: F, 6: J, 10: N, 12: P, 16: B, Resto vacío

Explicación:
Opción A:
Primer intento:
- M = 7
- factor de carga máximo de N/M = 0.5
- Rehashing: Siguiente número primo mayor que el doble del tamaño actual (17)

Inserción {"B", "D", "F", "J", "N", "P"} = 66, 68, 70, 74, 78, 80
La función hashValue(key) es: abs(hash(key)) % M

hashValue("B") = 66 % 7 = 3, Factor de carga = N/M = 1/7 = 0.14
hashValue("D") = 68 % 7 = 5, Factor de carga = N/M = 2/7 = 0.28
hashValue("F") = 70 % 7 = 0, Factor de carga = N/M = 3/7 = 0.42
hashValue("J") = 74 % 7 = 4, Factor de carga = N/M = 4/7 = 0.57, Hay rehashing

Intento final:
- M = 17
- Factor de carga máximo de N/M = 0.5
- Rehashing: Siguiente número primo mayor que el doble del tamaño actual (17)
Inserción {"B", "D", "F", "J", "N", "P"} = 66, 68, 70, 74, 78, 80
La función hashValue(key) es: abs(hash(key)) % M

hashValue("B") = 66 % 17 = 15, Factor de carga = N/M = 1/17 = 0.05
hashValue("D") = 68 % 17 = 0, Factor de carga = N/M = 2/17 = 0.11
hashValue("F") = 70 % 17 = 2, Factor de carga = N/M = 3/17 = 0.17
hashValue("J") = 74 % 17 = 6, Factor de carga = N/M = 4/17 = 0.23
hashValue("N") = 78 % 17 = 10, Factor de carga = N/M = 5/17 = 0.29
hashValue("P") = 80 % 17 = 12, Factor de carga = N/M = 6/17 = 0.35 (nunca se llegó al límite)

7. Considere una tabla de hash vacía que usa el método de Separate Chaining con una capacidad (M) de 7. Suponga que cada bucket en la tabla está implementado con una lista sencillamente encadenada donde se agrega al principio (`add_first()`) las siguientes parejas llave-valor y en este orden: `elements = [(3, 'x'), (14, 'y'), (24, 'z'), (8, 'w'), (2, 'v'), (15, 'u'), (21, 't'), (6, 's')]`. Si asumimos la función de hash h(key) es: `key % M`, después de insertar `(put(k, v))` todos los elementos de la lista, ¿cuántas inserciones y colisiones tiene la tabla hash al finalizar la carga?

A. 8 inserciones, 2 colisiones
B. 8 inserciones, 3 colisiones
C. 7 inserciones, 2 colisiones
D. 7 inserciones, 3 colisiones

Explicación:
Opción B:
Elemento (3, 'x'): 3 % 7 = 3, bucket 3. No hay colisión
Elemento (14, 'y'): 14 % 7 = 0, bucket 0. No hay colisión
Elemento (24, 'z'): 24 % 7 = 3, bucket 3. **Hay una colisión, 1**
Elemento (8, 'w'): 8 % 7 = 1, bucket 1. No hay colisión
Elemento (2, 'v'): 2 % 7 = 2, bucket 2. No hay colisión
Elemento (15, 'u'): 15 % 7 = 1, bucket 1. **Hay una colisión, 2**
Elemento (21, 't'): 21 % 7 = 0, bucket 0. **Hay una colisión, 3**
Elemento (6, 's'): 6 % 7 = 6, bucket 6. No hay colisión

8. Considere una tabla de hash vacía que utiliza el método de Separate Chaining con capacidad M=4. Supongamos que cada bucket está implementado con una lista encadenada donde se agregan al final (`add_last()`) las parejas llave-valor que se listan a continuación y en este orden: `elements = [(4, 'm'), (8, 'n'), (12, 'o'), (16, 'p'), (20, 'q'), (24, 'r'), (28, 's'), (32, 't')]`. Si utilizamos una función de hash basada en la fórmula `hashValue(key) = key%(M//2)`, después de insertar todos los elementos de la lista en la tabla y asumiendo que la búsqueda en cada bucket inicia en la cabeza, ¿cuál es la complejidad temporal de la búsqueda del elemento (32, 't')?

A. O(1)
B. O (n log(n))
C. O(n)
D. O(n2)

Explicación:
La función de hash asigna las llaves a los buckets usando `key % (M // 2)`, lo que significa que `M//2 = 2`. Esto provoca que todas las llaves se asignen al mismo bucket (debido a colisiones), formando una única lista encadenada en ese bucket. La búsqueda en esa lista encadenada es O(n), ya que se debe recorrer todos los elementos.

9. Considere una tabla de hash vacía que utiliza el método de Separate Chaining. Supongamos que cada bucket está implementado con una lista encadenada que solo tiene una referencia a la cabeza, y en donde se agregan al inicio las parejas llave-valor que se listan a continuación y este orden, `elements = [(4, 'm'), (8, 'n'), (12, 'o'), (16, 'p'), (20, 'q'), (24, 'r'), (28, 's'), (32, 't')]` Suponga que no hubo colisiones y que la búsqueda en cada bucket inicia en la cabeza, ¿Cuál es la complejidad temporal de eliminar el elemento (32, 't')?

A. O(1)
B. O (n log(n))
C. O(n)
D. O(n3)

10. Considere una tabla de hash vacía que utiliza el método de Separate Chaining, en donde, debido a colisiones, un solo bucket contiene todas las parejas llave-valor. Supongamos que el bucket está implementado como una lista encadenada simple que tiene solo una referencia a la cabeza, y en donde se agregan al final las parejas llave-valor que se listan a continuación y en este orden: `elements = [(4, 'm'), (8, 'n'), (12, 'o'), (16, 'p'), (20, 'q'), (24, 'r'), (28, 's'), (32, 't')]` ¿Cuál es la complejidad temporal de insertar el elemento (32, 'z')?
Nota: ignorar cualquier rehashing o factor de carga)

A. O(1)
B. O (n log(n))
C. O(n)
D. O(n2)

11. Considere una tabla de hash vacía que utiliza el método de Separate Chaining con capacidad M=8. Supongamos que cada bucket está implementado con una lista encadenada donde se agregan al final las parejas llave-valor que se listan a continuación y en ese orden, `elements = [(4, 'm'), (8, 'n'), (12, 'o'), (16, 'p'), (20, 'q'), (24, 'r'), (28, 's'), (32, 't')]`. Si utilizamos una función de hash basada en la fórmula hashValue(key) = `(key//3)%(M//2)`, después de insertar todos los elementos de la lista en la tabla, ¿cuál es el estado final de la tabla, ignorando cualquier operación de rehash y factor de carga?

A.
Bucket 0: (24, 'o'), (12, 'r')
Bucket 1: (16, 'm'), (4, 'p'), (28, 's')
Bucket 2: (8, 'n'), (20, 'q'), (32, 't')
Bucket 3: Vacío

B.
Bucket 0: (24, 'o'), (12, 'r')
Bucket 1: (28, 'm'), (16, 'p'), (4, 's')
Bucket 2: (32, 'n'), (20, 'q'), (8, 't')
Bucket 3: Vacío

C.
Bucket 0: (12, 'o')
Bucket 1: (4, 'm'), (16, 'p'), (28, 's')
Bucket 2: (8, 'n'), (20, 'q')
Bucket 3: (24, 'r')
Bucket 4: (32, 't')

D.
Bucket 0: (12, 'o'), (24, 'r')
Bucket 1: (4, 'm'), (16, 'p'), (28, 's')
Bucket 2: (8, 'n'), (20, 'q'), (32, 't')
Bucket 3: Vacío

Explicación:
Opción D:
- M=8
elements = [(4, 'm'), (8, 'n'), (12, 'o'), (16, 'p'), (20, 'q'), (24, 'r'), (28, 's'), (32, 't')]
hashValue(key) = (key//3)%(M//2)

llave 4: (4//3)%(8//2) = 1%4 = 1. (4, 'm') en bucket con índice 1
llave 8: (8//3)%(8//2) = 2%4 = 2. (8, 'n') en bucket con índice 2
llave 12: (12//3)%(8//2) = 4%4 = 0. (12, 'o') en bucket con índice 0
llave 16: (16//3)%(8//2) = 5%4 = 1. (16, 'p') en bucket con índice 1
llave 20: (20//3)%(8//2) = 6%4 = 2. (20, 'q') en bucket con índice 2
Para la llave 24: (24//3)%(8//2) = 8%4 = 0. (24, 'r') en bucket con índice 0
Para la llave 28: (28//3)%(8//2) = 9%4 = 1. (28, 's') en bucket con índice 1
Para la llave 32: (32//3)%(8//2) = 10%4 = 2. (32, 't') en bucket con índice 2