Algoritmos de búsqueda y hashing

1

Algoritmos de búsqueda

2

Algoritmos de búsqueda: Importancia

- Información creciente en el mundo
- Necesidad de valores específicos
- Rapidez en la necesidad de la información

3

Algoritmos que veremos

- Linear search
- Binary search
- Exponential search
- Fibonacci search

4

{10,50,30,70,80,60,20,90,40}

Arreglo de prueba desordenado

{10,20,30,40,50,60,70,80,90}

Arreglo de prueba ordenado

Número a buscar: 20

5

Linear search

- Simple
- Como su nombre lo indica: búsqueda lineal
- Pseudocódigo
  1. Asigna el primer elemento del arreglo como elemento actual
  2. Comprueba si el elemento actual es igual al elemento buscado
  3. Si elemento actual NO ES IGUAL a elemento buscado, entonces asigna el siguiente elemento del arreglo como elemento actual y vuelvo al paso 2
  4. Si elemento actual ES IGUAL a elemento buscado, entonces el algoritmo termina

6

Linear search

{10,50,30,70,80,60,20,90,40}

7

Pregunta LS.1

- Mejor caso

a) N
b) 1
c) N/2
d) Ninguna de las anteriores

8

Pregunta LS.2

- Peor caso

a) N
b) 1
c) N/2
d) Ninguna de las anteriores

9

Pregunta LS.3

- Complejidad

a) O(N)
b) O(1)
c) Ω(N)
d) Ω(1)

10

Binary search

- Divide el espacio de solución por la mitad en cada sub-búsqueda
- El arreglo DEBE estar ordenado, de lo contrario no funciona
- Pseudocódigo (asume arreglo ordenado de menor a mayor de izquierda a derecha)
  1. Asigna el elemento de la mitad del arreglo como elemento actual
  2. Comprueba si el elemento actual es igual, menor o mayor al elemento buscado
  3. Si elemento actual ES MENOR a elemento buscado, entonces asigna el elemento de la mitad del sub arreglo izquierdo como elemento actual y vuelvo al paso 2
  4. Si elemento actual ES MAYOR a elemento buscado, entonces asigna el elemento de la mitad del sub arreglo derecho como elemento actual y vuelvo al paso 2
  5. Si elemento actual ES IGUAL a elemento buscado, entonces el algoritmo termina

11

Binary search

{10,20,30,40,50,60,70,80,90}

12

Pregunta BS.1

- Mejor caso

a) N
b) 1
c) N/2
d) $\log_2 N$

13

Pregunta BS.2

- Peor caso

a) N
b) 1
c) N/2
d) $\log_2 N$

14

Pregunta BS.3

- Complejidad

a) $O(\log_2 N)$
b) O(1)
c) Ω(N)
d) $\Omega(\log_2 N)$

15

Exponential search

- Divide el espacio de búsqueda en sub-espacios exponencialmente más grandes en cada iteración. En cada sub-espacio realiza un binary search.
- El arreglo DEBE estar ordenado, de lo contrario no funciona
- Pseudocódigo (asume arreglo ordenado de menor a mayor de izquierda a derecha)
  1. Asigna el primer elemento del arreglo como elemento actual y asigna una variable aux a 1
  2. Comprueba si el elemento actual es mayor al elemento buscado
  3. Si elemento actual ES MAYOR a elemento buscado, entonces duplica aux y asigna el elemento que se encuentra a aux espacios a la derecha como elemento actual. Vuelve al paso 2
  4. Si elemento actual ES MENOR O IGUAL a elemento buscado, entonces realiza binary search entre los índices aux/2 y aux

16

Exponential search

{10,20,30,40,50,60,70,80,90}

17

Pregunta ES.1

- Mejor caso

a) N
b) 1
c) N/2
d) $\log_2 N$

18

Pregunta ES.2

- Peor caso

a) N
b) $2 \cdot \log_2 N$
c) N/2
d) $\log_2 N$

19

Pregunta ES.3

- Complejidad

a) $O(\log_2 N)$
b) O(1)
c) $\Omega(N)$
d) $\Omega(\log_2 N)$

20

Fibonacci search

- Divide el espacio de búsqueda en sub-espacios más chicos en cada iteración, siguiendo una sucesión de Fibonacci (inversa).
- El arreglo DEBE estar ordenado, de lo contrario no funciona
- Pseudocódigo (asume arreglo ordenado de menor a mayor de izquierda a derecha)
  1. Busca primer índice Fibonacci mayor al largo del arreglo. Genera F(i), F(i-1) y F(i-2) y offset = -1
  2. Compara elemento en posición min(offset + F(i-2),largo arreglo) con elemento buscado
  3. Si elemento buscado ES MAYOR entonces se cambian los tres valores de Fibonacci: F(i) = F(i-1), F(i-1) = F(i-2), F(i-2) = F(i) – F(i-1) y se asigna al offset el valor de min(offset + F(i-2),largo arreglo)
  4. Si elemento buscado ES MENOR entonces se cambian los tres valores de Fibonacci: F(i) = F(i-2), F(i-1) = F(i-1) - F(i-2), F(i-2) = F(i) – F(i-1)

21

Fibonacci search

1 2 3 4 5 6 7 8 9 13

{10,20,30,40,50,60,70,80,90}

22

Pregunta FS.1

- Mejor caso

a) N
b) 1
c) N/2
d) $\log_2 N$

23

Pregunta FS.2

- Peor caso

a) N
b) $\log_2 N$
c) N/2
d) $\log_2 N^2$

24

Pregunta FS.3

- Complejidad

a) $O(\log N)$
b) $O(1)$
c) $\Omega(N)$
d) $\Omega(\log N)$

25

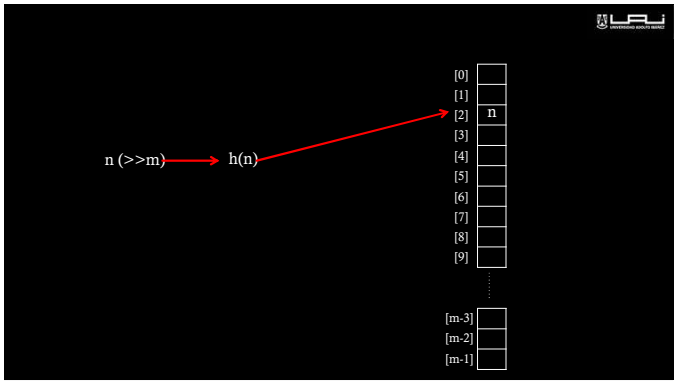
Hashing

26

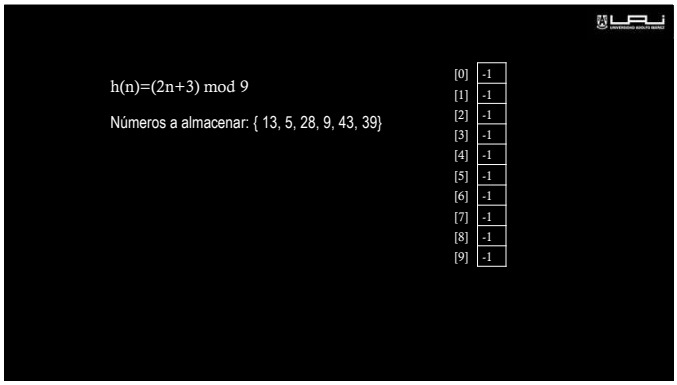
Hashing: Importancia

- Diversidad de datos pueden ser guardados eficientemente
- Contraseñas
- Certificados digitales

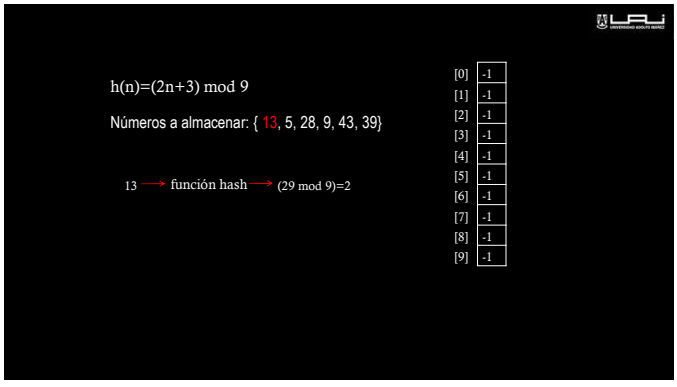
27


n (>>m)
h(n)
[0]
[1]
[2]
n
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[m-3]
[m-2]
[m-1]

28


h(n)=(2n+3) mod 9
Números a almacenar: { 13, 5, 28, 9, 43, 39}
[0] -1
[1] -1
[2] -1
[3] -1
[4] -1
[5] -1
[6] -1
[7] -1
[8] -1
[9] -1

29


h(n)=(2n+3) mod 9
Números a almacenar: { 13, 5, 28, 9, 43, 39}
13 función hash (29 mod 9)=2
[0] -1
[1] -1
[2] -1
[3] -1
[4] -1
[5] -1
[6] -1
[7] -1
[8] -1
[9] -1

30

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

13 ⟶ función hash ⟶ (29 mod 9)=2

[0]	-1
[1]	-1
[2]	13
[3]	-1
[4]	-1
[5]	-1
[6]	-1
[7]	-1
[8]	-1
[9]	-1

31

h(n)=(2n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

5 ⟶ función hash ⟶ (13 mod 9)=4

[0]	-1
[1]	-1
[2]	13
[3]	-1
[4]	-1
[5]	-1
[6]	-1
[7]	-1
[8]	-1
[9]	-1

32

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

5 ⟶ función hash ⟶ (13 mod 9)=4

[0]	-1
[1]	-1
[2]	13
[3]	-1
[4]	5
[5]	-1
[6]	-1
[7]	-1
[8]	-1
[9]	-1

33

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

28 ⟶ función hash ⟶ (59 mod 9)=6

[0]	-1
[1]	-1
[2]	
[3]	-1
[4]	
[5]	-1
[6]	-1
[7]	-1
[8]	-1
[9]	-1

34

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

28 ⟶ función hash ⟶ (59 mod 9)=6

[0]	-1
[1]	-1
[2]	13
[3]	-1
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	-1
[9]	-1

35

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

9 ⟶ función hash ⟶ (21 mod 9)=3

[0]	-1
[1]	-1
[2]	13
[3]	-1
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	-1
[9]	-1

36

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

9 ⟶ función hash ⟶ (21 mod 9)=3

[0]	-1
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	-1
[9]	-1

37

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

43 ⟶ función hash ⟶ (89 mod 9)=8

[0]	-1
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	-1
[9]	-1

38

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

43 ⟶ función hash ⟶ (89 mod 9)=8

[0]	-1
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	43
[9]	-1

39

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

39 ⟶ función hash ⟶ (81 mod 9)=0

[0]	-1
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	43
[9]	-1

40

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

39 ⟶ función hash ⟶ (81 mod 9)=0

[0]	39
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	43
[9]	-1

41

h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

¿ESTÁ EL NÚMERO 8?

[0]	39
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	43
[9]	-1

42

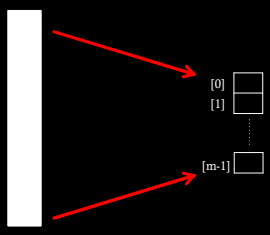
h(n)=(2*n+3) mod 9

Números a almacenar: { 13, 5, 28, 9, 43, 39}

¿ESTÁ EL NÚMERO 13?

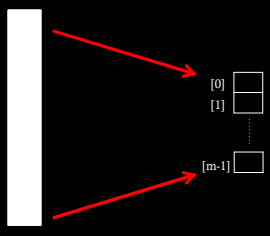
[0]	39
[1]	-1
[2]	13
[3]	9
[4]	5
[5]	-1
[6]	28
[7]	-1
[8]	43
[9]	-1

43



Cualquier función que permita mapear un dato en un rango arbitrario a otro dato en un rango limitado sirve como función hash

44



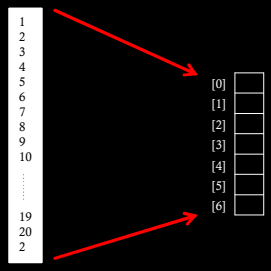
Cada casilla de la tabla hash se conoce también como un *bucket*

45

Una buena función hash distribuye los datos "uniformemente" en las casillas

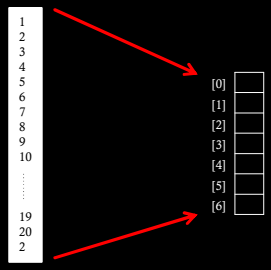
46

Supón que el rango de datos va del 1 al 21 y que la tabla hash tiene 7 casillas



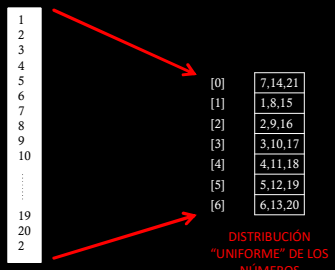
47

Una buena función hash: h(n)= n%7



48

Una buena función hash:
h(n)= n%7

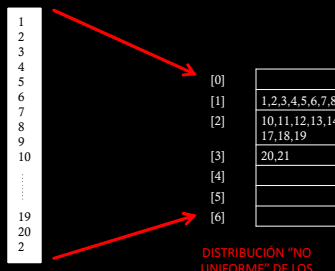


[0]	7,14,21
[1]	1,8,15
[2]	2,9,16
[3]	3,10,17
[4]	4,11,18
[5]	5,12,19
[6]	6,13,20

DISTRIBUCIÓN "UNIFORME" DE LOS NÚMEROS

49

Una mala función hash:
h(n)= (n/10+1)%7



[0]	
[1]	1,2,3,4,5,6,7,8,9
[2]	10,11,12,13,14,15,16, 17,18,19
[3]	20,21
[4]	
[5]	
[6]	

DISTRIBUCIÓN "NO UNIFORME" DE LOS NÚMEROS

50

- Una función hash mapea un dato de un rango arbitrario a otro rango definido y limitado
- Una buena función hash distribuye "uniformemente" los datos en los *buckets* de la tabla

Estructura de Datos y Algoritmos - TICS311 51

51

Colisiones

52

Función hash: h(n)= n%7
Números a almacenar: {7, 43, 18, 21}

[0]	
[1]	
[2]	
[3]	
[4]	
[5]	
[6]	

53

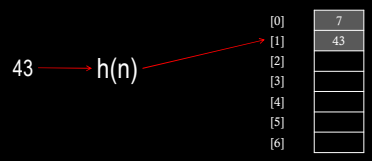
Función hash: h(n)= n%7
Números a almacenar: {7, 43, 18, 21}

7 → h(n)

[0]	7
[1]	
[2]	
[3]	
[4]	
[5]	
[6]	

54

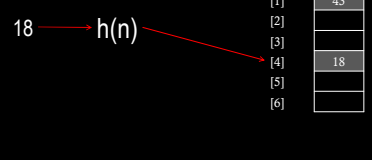
Función hash: h(n)= n%7
Números a almacenar: {7, 43, 18, 21}



[0]	7
[1]	43
[2]	
[3]	
[4]	
[5]	
[6]	

55

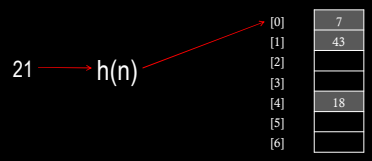
Función hash: h(n)= n%7
Números a almacenar: {7, 43, 18, 21}



[0]	7
[1]	43
[2]	
[3]	
[4]	18
[5]	
[6]	

56

Función hash: h(n)= n%7
Números a almacenar: {7, 43, 18, 21}



[0]	7	!!
[1]	43	
[2]		
[3]		
[4]	18	
[5]		
[6]		

57

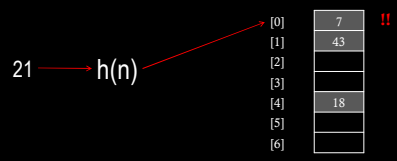
Aumentar el número de buckets en un factor X:

- Reactivo: cuando se produce una colisión, se aumenta el tamaño de la tabla
- Pro-activo: cuando el **factor de carga** excede un valor predeterminado, se aumenta el tamaño de la tabla

$$\text{Factor\_de\_carga} = \frac{Casillas\ usadas}{Casillas\ totales}$$

58

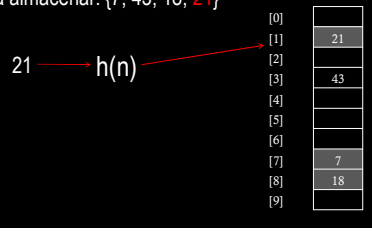
Función hash: h(n)= n%7
Números a almacenar: {7, 43, 18, 21}



59

Aumentar la tabla en 1.5

Función hash: h(n)= n%10
Números a almacenar: {7, 43, 18, 21}

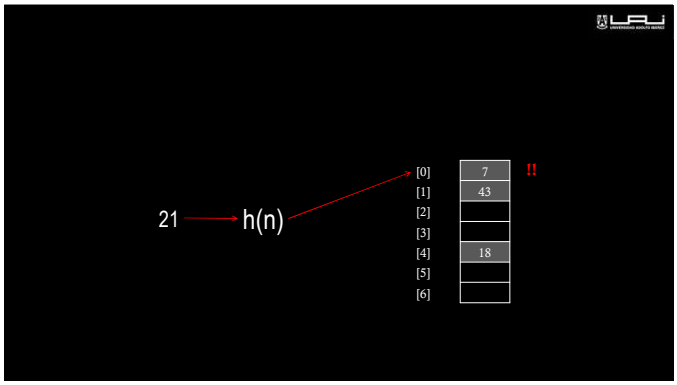


60

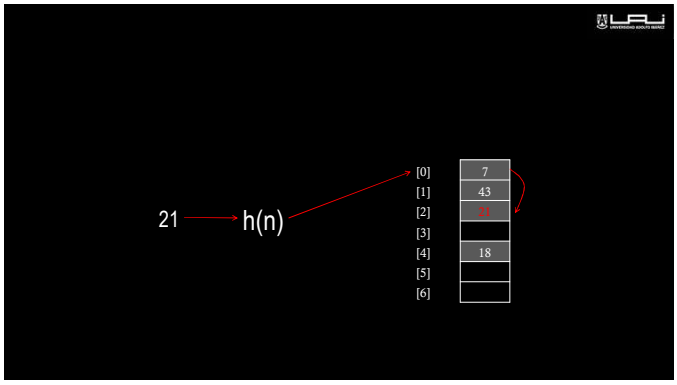
Sondeo lineal

- usar la siguiente posición desocupada

61



62

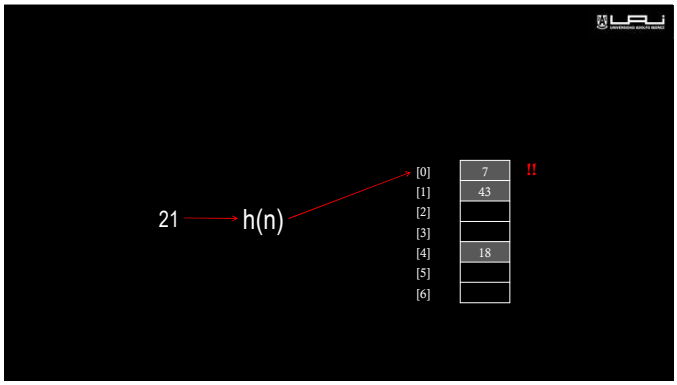


63

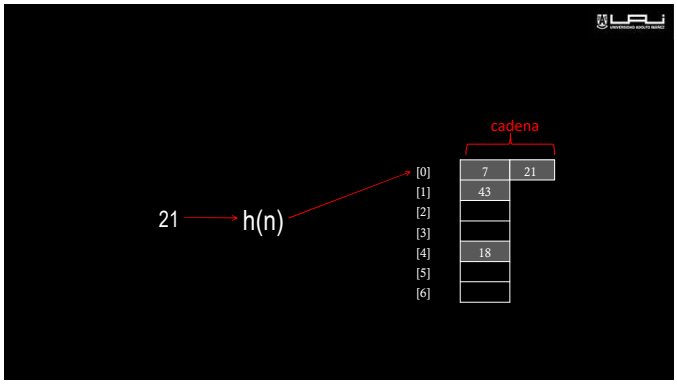
Encadenamiento separado

- agregar posiciones a la derecha de la tabla

64



65



66

Complejidad

- Tabla ideal (sin colisiones):
  - Insertar un elemento es θ(1)
  - Buscar un elemento es θ(1)

	Insertar	Buscar
Sondeo lineal	Θ(m)	Θ(m)
Encadenamiento separado	Θ(m)	Θ(m)

67

Las colisiones son INEVITABLES

68

Actividad

- Escribe un programa en C para:
  - Crear un arreglo de 9 posiciones, inicialmente lleno de -1.
  - Usando la función hash h(n)=(2*n+3) mod 9, ingresa los siguientes números al arreglo: {13, 5, 28, 9, 43, 39, 23, 27, 17}
  - Ahora registra los resultados de buscar los siguientes números en el arreglo: {18, 9, 17, 5}. ¿Detectan algún problema?
  - Describan el problema que ocurrió en el ejercicio anterior e inventen (OJO: "inventen", no "busquen en Internet") al menos dos maneras de resolverlo.
  - Implementa las soluciones propuestas y vuelve a buscar los mismos números.
  - ¿Se resolvió el problema? ¿Podría volver a haber problemas incluso con la solución implementada?

69