Tecnológico de Monterrey.

Campus Querétaro.

TC2038. Análisis y diseño de Algoritmos A

M.C. Ramona Fuentes Valdéz

rfuentes@tec.mx

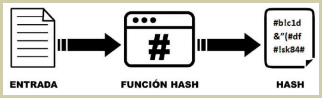
63

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Introducción



- Los algoritmos de *hashing* son muy útiles en muchas áreas de las ciencias computacionales, sobre todo porque, usando una buena función de *hash*, es posible reducir la búsqueda a un tiempo casi constante O(1).

Ejemplo:

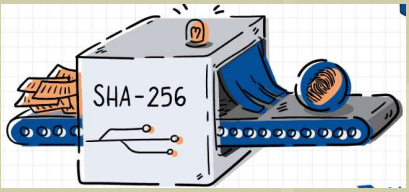
- Dadas las cadenas S_1, de longitud n_1, y S_2, de longitud n_2, indicar si son iguales o no.
  - Si realizamos la comparación a **fuerza bruta**, el algoritmo resultante tiene un orden $O(\min(n_1, n_2))$.
  - Sin embargo, usando una buena **función de hash** podríamos lograr un orden O(1).

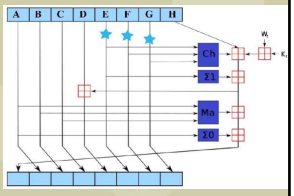
Aplicaciones

- Las funciones de *hash* criptográficas se utilizan ampliamente en las criptomonedas para pasar información de transacciones de forma anónima.

Ejemplo:

- Bitcoin, la criptomoneda original y más grande, utiliza la función *hash* SHA-256 en su algoritmo.



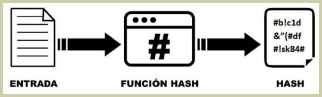


64

Técnicas de diseño de algoritmos

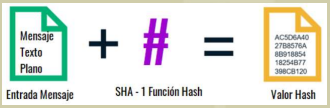
Manejo de strings

Hash string

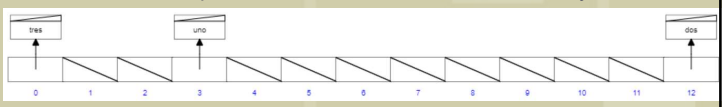


¿Qué es el *hash string*?

- Es un algoritmo matemático que transforma una determinada información en una cadena de texto conformada por números y letras. La cual tiene una longitud fija y sigue un orden único e irrepetible.



- Hashing es una técnica utilizada para realizar inserciones, eliminaciones y búsquedas.



Open hashing, https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

- Esta estructura de datos, sin embargo, no es eficiente en operaciones que requieren información de orden entre los elementos, como findMin, findMax e imprimir la tabla completa en orden.

65

Técnicas de diseño de algoritmos

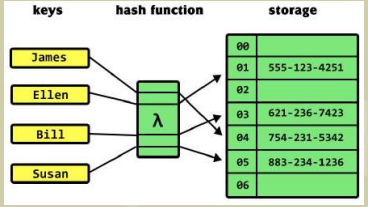
Manejo de strings

Hash string

Ideas básicas

- La idea principal de una función *hash* para cadenas es convertir una cadena en un número entero.
  - La función debe cumplir la condición de que, si dos cadenas son iguales, les debe asignar el mismo número entero.
  - Si la función asigna el mismo número a dos cadenas diferentes se dice que hubo una colisión.
  - La función ideal sería aquella que a cada cadena le asignará un entero diferente.

- En realidad, es muy complicado hacer una función con tal condición, pero es suficiente con generar una función donde la *probabilidad de asignarle el mismo entero a dos cadenas diferentes sea muy baja*, y ese tipo de funciones sí se han podido generar.
  - Un ejemplo es la llamada Polynomial Rolling Hash Function.



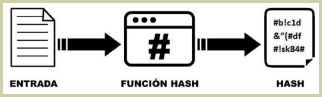
66

Técnicas de diseño de algoritmos

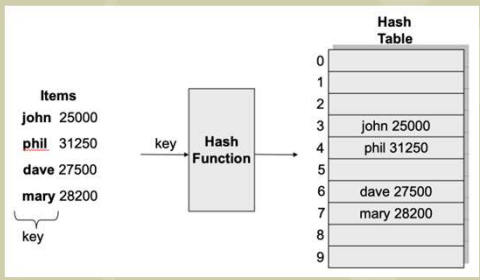
Manejo de strings

Hash string

Ideas básicas



- La tabla *hash* ideal es una matriz de algún tamaño fijo que contenga los elementos.
- Un elemento almacenado debe tener un miembro de datos, llamado llave (*key*), que se utilizará para calcular el valor del índice (value index) del elemento.
  - La key podría ser un número entero, una cadena, etc.
    - Ejemplo: un nombre o ID que forma parte de una gran estructura de empleados
- El tamaño de la matriz es TableSize.
- Los elementos que se almacenan en la tabla *hash* están indexados por valores de *0* a *TableSize – 1*.
- Cada key se asigna a algún número en el rango de *0* a *TableSize – 1*.
- El mapeo se llama función hash.

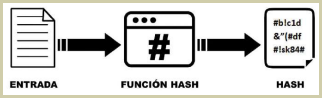


Técnicas de diseño de algoritmos

Manejo de strings

Hash string

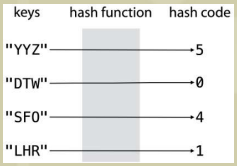
Función Hash



- La función *hash* → debe ser simple de calcular.
- Debe distribuir las llaves uniformemente entre las celdas.
  - La idea del *hash* es distribuir las entradas (pares clave/valor) en una matriz.
  - La función *hash* asigna las llaves a índices de matriz (a menudo llamados depósitos o *buckets*).

    $h : U \rightarrow \{ 0, 1, 2, \ldots, m - 1 \}$

    donde:
    U es el universo de llaves, y
    m es el tamaño del arreglo.



0	"Detroit"
1	"London Heathrow"
2	
3	
4	"San Francisco"
5	"Toronto Pearson"
6	
7	

  - ❖ Si se supiera de antemano qué claves aparecerán, se pudieran escribir funciones *hash* perfectas, *pero* no es así.

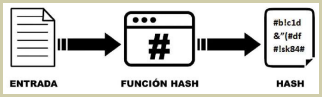
- Por lo general, el proceso de *hash* consta de dos pasos:
  1. Calcular un código *hash* entero a partir de la clave (que puede ser un objeto de cualquier tipo).
  2. Comprimir (reducir) el código *hash* a un número entre 0 y *m - 1*.

Técnicas de diseño de algoritmos

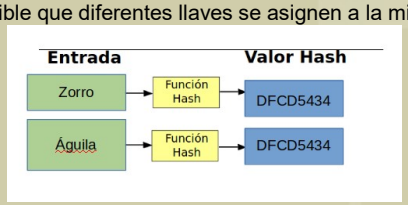
Manejo de strings

Hash string

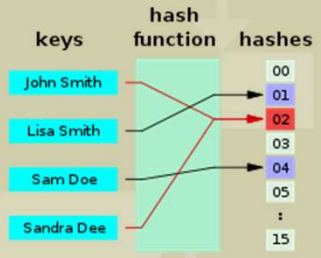
Principales problemas



- Las llaves pueden no ser numéricas.
- El número de llaves posibles es mucho mayor que el espacio disponible en la tabla.
- Es posible que diferentes llaves se asignen a la misma ubicación.



- La función *hash* no es uno a uno => colisión.
  - Si hay demasiadas colisiones, el rendimiento de la tabla *hash* se verá afectado drásticamente.



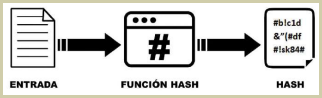
69

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

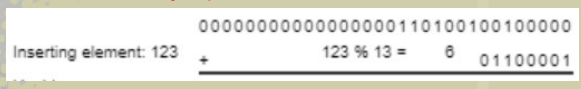
Función Hash

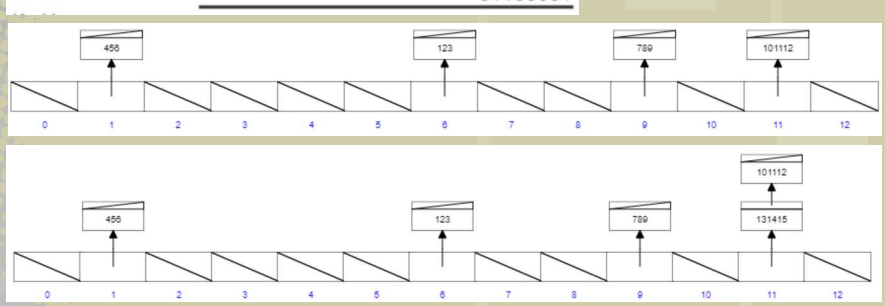


- Hash Integer

- Si las llaves de entrada son números enteros, la estrategia general es: Key *mod* TableSize.
  - A menos que la llave tenga algunas propiedades no deseables.
    - Ejemplo: todas las claves terminan en 0 y se utilizó mod 10.

Ejemplo:





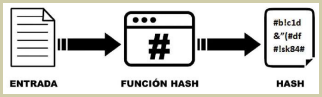
70

Técnicas de diseño de algoritmos

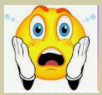
Manejo de strings

Hash string

Función Hash

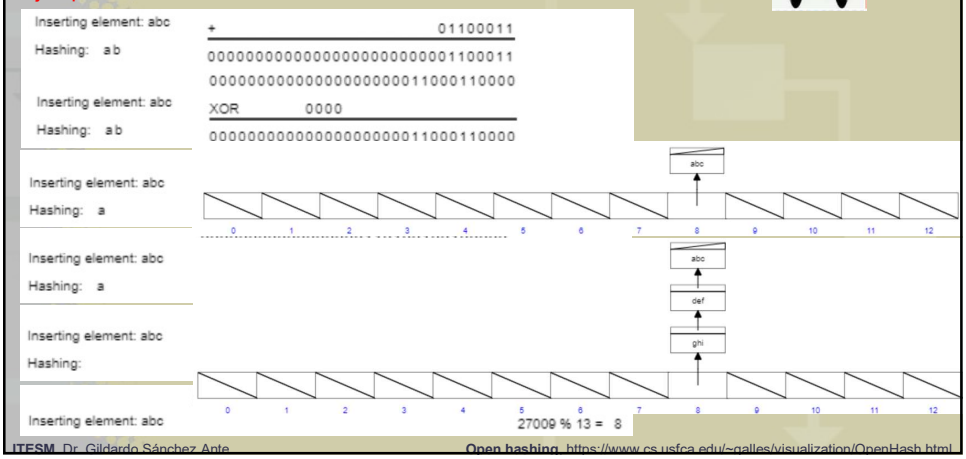


- Hash Strings



- Si las llaves son cadenas, la función *hash* necesita más atención.
  - Primero hay que convertir en un valor numérico.

Ejemplo:



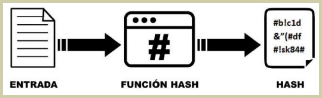
Open hashing https://www.cs.usfca.edu/~galles/visualization/OpenHash.html

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Función Hash



Caso 1. Sume los valores ASCII de todos los caracteres de la llave (key).

- Sencillo de implementar y rápido.
- Sin embargo, si el tamaño de la tabla es grande, la función no distribuye bien las llaves.
  - Ejemplo: Tamaño de la tabla = 10000, longitud de la llave <= 8, la función *hash* solo puede asumir valores entre 0 y 1016.

Caso 2. Examine sólo los primeros 3 caracteres de la llave.

- En teoría se pueden generar 26 * 26 * 26 = 17576 palabras diferentes. Sin embargo, por ejemplo, el idioma inglés no es aleatorio, sólo son posibles 2851 combinaciones diferentes.
- Por lo tanto, esta función, aunque es fácilmente computable, tampoco es apropiada si la tabla *hash* es razonablemente grande.

Caso 3. Polinomial

- Un enfoque más eficaz es calcular un polinomio cuyos coeficientes sean los valores enteros de los caracteres de la cadena.
  - Ejemplo: para una cadena *s* con longitud *n*+1, se podría calcular un polinomio en *x* y tomar el resultado *mod* el tamaño de la tabla.

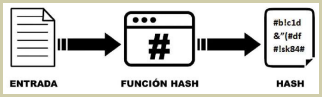
$$H(s) = \sum_{i=0}^{n} \text{s.charAt}(i) \cdot x^i$$

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Función Hash



Caso 3. Polinomial → Polynomial Rolling Hash Function

- Sea *S*, una cadena de longitud *n*, a la que se le quiere aplicar la función *hash*, y sean *S[0]*, *S[1]*, ..., *S[n − 1]* las representaciones en enteros de cada uno de los caracteres que lo forman, la función *Polynomial Rolling Hash Function* para la cadena *S* se define como:

$$prhs(S) = (S[0] * p^0 + S[1] * p^1 + \ldots + S[n-1] * p^{n-1}) \bmod m$$

$$= \left(\sum_{i=0}^{n-1} S[i] * p^i\right) \bmod m$$

Donde, *p* y *m* son números enteros positivos que se deben seleccionar.
- *Una forma común es que ambos sean números enteros primos.*

- En el caso de **p**, se recomienda que sea un número primo cercano al número de caracteres en el alfabeto. Ejemplo: para los caracteres en español, que son 27, el número primo más cercano es el 31, por lo que *p = 31* es una buena opción. Si se incluyen también las mayúsculas, el conjunto sube a 54 y entonces *p = 53* es una buena opción.
- En lo que respecta a **m**, debe ser un número muy grande, debido a que la probabilidad de que haya colisión es, aproximadamente, *1/m*. Una buena selección ha sido un número primo muy grande, por ejemplo, m = 10^9 + 9, lo que nos da una probabilidad de colisión de alrededor de 10^{-9}.

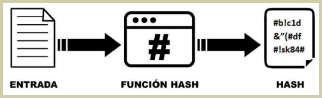
73

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Función Hash



Caso 3. Polinomial → Polynomial Rolling Hash Function

- Sea *S*, una cadena de longitud *n*, a la que se le quiere aplicar la función *hash*, y sean *S[0]*, *S[1]*, ..., *S[n − 1]* las representaciones en enteros de cada uno de los caracteres que lo forman, la función *Polynomial Rolling Hash Function* para la cadena *S* se define como:

$$prhs(S) = (S[0] * p^0 + S[1] * p^1 + \ldots + S[n-1] * p^{n-1}) \bmod m$$

$$= \left(\sum_{i=0}^{n-1} S[i] * p^i\right) \bmod m$$

Donde, *p* y *m* son números enteros positivos que se deben seleccionar.
- *Una forma común es que ambos sean números enteros primos.*

- Si bien, una probabilidad de colisión de 10^{-9} es muy baja, si se hace una sola comparación. En algunos problemas es necesario comparar una cadena con un conjunto de otras cadenas. Si ese conjunto es de 10^6 elementos, la probabilidad de que suceda una colisión aumenta a 10^{-3}, y si se comparan las 10^6 cadenas entre sí, la probabilidad se acerca peligrosamente a 1.
- Una forma muy común y simple de obtener mejores probabilidades de colisión, incluso con muchas comparaciones, es hacer dos funciones *hash*, con diferentes *p* y *m*. Se aplican las dos funciones *hash* y sólo se declaran iguales las cadenas si el resultado de ambas funciones son iguales. En este caso, si *m* para la segunda función es también cercana a 10^9, aplicar las dos funciones es equivalente a tener una *m* aproximadamente igual a 10^{18}.

74

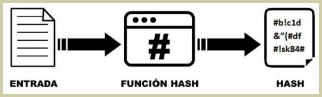
Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Función Hash

Caso 3. Polinomial



Algoritmo

```
long long prhf ( const std :: string &s) {
  int p = 31;
  int m = 1e9 + 9;
  long long result = 0;
  long long power = 1;

  for (int i = 0; i < s.size (); i++)  {
    result = (result + ((s[i] - 'a' + 1) * power)) % m;
    power = (power * p) % m;
  }
 return result ;
}
```


75

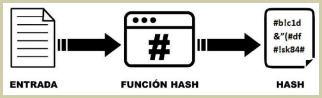
Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Función Hash

Caso 3. Polinomial



Ejemplo

➢ Sea *S* = a b c d e f g h i j k, P = c d e f

h_p = h(**cdef**) = h(3, 4, 5, 6) = 18

Primera propuesta

h_s =
- h(abcd) = h(1, 2, 3, 4) = 10
- h(bcde) = h(abcd) - 1 + 5 = 10 - 1 + 5 = 14
- **h(cdef) = h(bcde) - 2 + 6 = 14 - 2 + 6 = 18**

Segunda propuesta

h(abcd) = h(1, 2, 3, 4)
= 1 x 10^3 + 2 x 10^2 + 3 x 10^1 + 4 x 10^0
= 1000 + 200 + 30 + 4
= 1234

h(bcde) = (h(abcd) - 1 x 10^3) x 10 + 5
= (1234 - 1000) x 10 + 5
= 2345

h(cdef) = (h(bcde) - 2 x 10^3) x 10 + 6
= (2345 - 2000) x 10 + 6
= 3456

Multiplicador: 10^{n-1}

Donde *n* es la posición del carácter desde la derecha.

Sugerencia:
Toma el valor que sea mayor al valor más grande del alfabeto.

Tabla	
a	1
b	2
c	3
d	4
e	5
f	6
g	7
h	8
i	9
j	10
k	11
…	

76

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Función Hash

Caso 3. Polinomial

Ejemplo

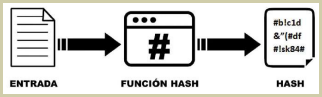


Tabla	
a	1
b	2
c	3
d	4
e	5
f	6
g	7
h	8
i	9
j	10
k	11
…	

- Sea S = a b c d e f g h i j k, P = c d e f

Tercera propuesta

Operación: *mod* X

Donde *X* es un número primo grande.

h(abcd) = h(1, 2, 3, 4)
= $(1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0) \bmod 113$
= 1234 mod 113
= 104

h(bcde) = (((h(abcd) - 1 x 10^3 mod 113) x 10) mod 113 + 5) mod 113
= (((104 - 96) x 10) mod 113 + 5) mod 113
= ((80 mod 113) + 5) mod 113
= 85

h(cdef) = (((h(bcde) - 1 x 10^3 mod 113) x 10) mod 113 + 6) mod 113
= (((85 - 79) x 10) mod 113 + 6) mod 113
= ((60 mod 113) + 6) mod 113
= 66

$10^{99} \bmod 113 = ?$ ⇒ (A x B) mod C = ((A mod C) x (B mod C)) mod C

$10^2 \bmod 113 = ((10 \bmod 113) \times (10 \bmod 113)) \bmod 113$

Números pequeños

77

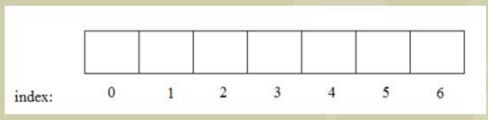
Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Ejercicio:

- Usando la función *hash* H(K) = K *mod* M, inserte estas llaves enteras: 701, 145, 217, 19 en la siguiente tabla:

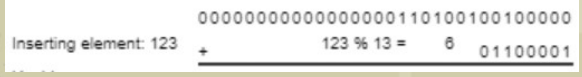


Proceso:

Hash Integer

- Si las llaves de entrada son números enteros, la estrategia general es: Key *mod* TableSize.

Ejemplo:

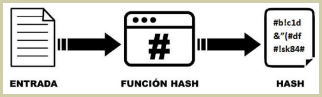


78

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

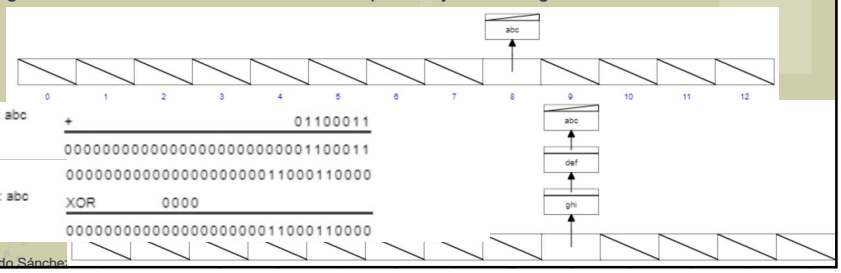


En manejo de cadenas de texto…

- Encuentra la longitud de la subcadena palindrómica más larga de una cadena.
  - Por ejemplo, aunque la cadena **abaccab** tiene múltiples subcadenas palindrómicas, como **aba** y **acca**, la subcadena palindrómica más larga es **baccab**.

Función Hash

- Asigna una llave a un número
  - el resultado debe limitarse a algún rango
  - pasar la misma llave siempre debería dar el mismo resultado
- Las llaves deben distribuirse en un rango
  - ¡Muy malo si todo tiene un valor *hash* de 1!
  - debería "parecer aleatorio"
- ¿Cómo se escribiría una función *hash* para objetos String?

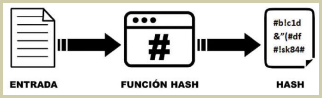


79

Técnicas de diseño de algoritmos

Manejo de strings

Hash string



En manejo de cadenas de texto…

- Encuentra la longitud de la subcadena palindrómica más larga de una cadena.
  - Por ejemplo, aunque la cadena **abaccab** tiene múltiples subcadenas palindrómicas, como **aba** y **acca**, la subcadena palindrómica más larga es **baccab**.

Algoritmo

- Calcular el valor del *hash* del patrón buscado.
- Calcular el valor del *hash* del texto de 0 a *m-1*.
- Compara el *hash* de ambos *strings*.
  - Si los valores *hash* son iguales, entonces se compara si ambos *strings* son realmente iguales.
  - Si no son iguales, se recorre la cadena y se calcula el *hash* del siguiente conjunto de valores del texto hasta que se termina.

Pattern m = 3

0	1	2
B	o	b

Text n = 6

0	1	2	3	4	5
H	i		B	o	b
H	i				
	i		B		
			B	o	
			B	o	b

80

Técnicas de diseño de algoritmos

Manejo de strings

Hash string

Ejercicio:

- Usando los datos que se muestran en la tabla, y considerando un valor de *n* = *4*, realiza las siguientes operaciones:
  - Para cada columna, calcula a[i] = (la suma de los ASCII de cada *char* en la columna)
  - Con el dato de cada columna realiza la operación: sumaDatoX % 256.
  - El resultado obtenido se debe representar en hexadecimal.
  - Se concatenan cada 2 columnas con el resultado del punto anterior para mostrar la salida esperada.

Proceso:

	Dato 1	Dato 2	Dato 3	Dato 4
	e	s	t	o
	b	a	Salto línea	d
	espacio	d	e	Salto línea
	n	o	espacio	S
	n	e	r	espacio
	i	v	o	[
A[i] → Suma ASCII				
Módulo 256				
Valor hexadecimal				
Concatenar				

81