Dispersión y Dispersión Extendida

Unidad 9.

Almacenamiento y Estructuras de Archivos

Definiciones

- Una función *hash* es como una caja negra que produce una dirección cada vez que se le introduce una llave

- Formalmente es una función $h(K)$ que transforma una llave K en una dirección

Definiciones

- La dirección resultante es utilizada como la base para el almacenamiento y recuperación de registros

- *Hashing* es como el indexado ya que involucra asociar una llave con una dirección relativa

Hashing vs Indexado

- Con el *hashing*, las direcciones generadas parecen ser aleatorias – no existe una conexión inmediata y obvia entre la llave y la localidad en donde se almacena la información

- *Hashing* en algunas ocasiones es conocido como aleatorización

Hashing vs Indexado

- *Con hashing*, dos llaves diferentes pueden ser transformadas para utilizar la misma dirección de tal manera que dos registros pueden ser enviados al mismo espacio en el archivo

- Cuando esto ocurre, se dice que se ha generado una colisión y se debe encontrar una manera de tratar con ella

Función *Hash*

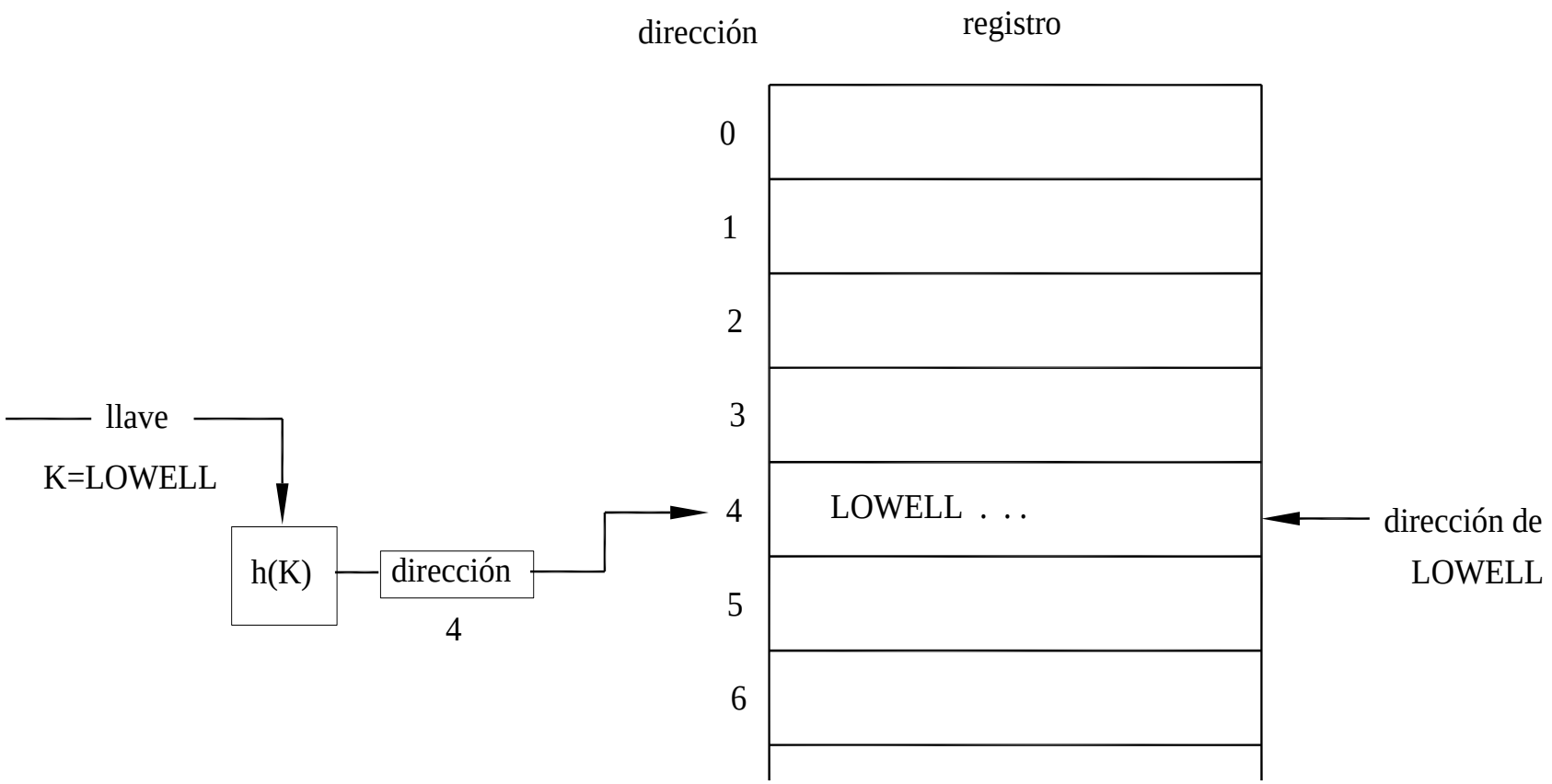
- Tomar dos valores de la representación ASCII de los dos primeros caracteres del nombre

- Multiplicarlos y usar los tres dígitos de la derecha para obtener el resultado de la dirección

Ejemplo, cont...

Nombre	Código ASCII de los dos primeros caracteres	Producto	Dirección
BALL	66 65	66 x 65 = 4290	290
LOWELL	76 79	76 x 79 = 6004	004
TREE	84 82	84 x 82 = 6888	888

Ejemplo, cont...



Colisiones

- Ahora suponga que existe una llave con el valor OLIVER, dado que OLIVER comienza con las mismas letras que LOWELL, se produce la misma dirección (004)

Definiciones

- Existe una colisión entre el registro de OLIVER y el registro de LOWELL

- Cuando esto ocurre, estas llaves se conocen como *sinónimos*

Soluciones

- Las colisiones causan problemas, no se pueden colocar dos registros en el mismo espacio
- Se tienen dos maneras de resolver el problema:
  - Escoger algoritmos *hash* partiendo de la base de cómo serían las colisiones
  - Modificar el comportamiento al momento de almacenar los registros

Algoritmo *hashing* perfecto

- La solución ideal para las colisiones es encontrar un algoritmo de transformación que evite las colisiones, tal algoritmo es llamado "algoritmo de *hashing* perfecto", pero es muy complicado encontrar un algoritmo de este tipo

Soluciones prácticas

- Una solución más practica es reducir el número de colisiones a un número aceptable

- Por ejemplo: si solo una de diez búsquedas de un registro resulta en una colisión, el promedio de búsquedas en el disco es considerado como bajo

Reduciendo las colisiones

- Esparcir los registros:
  - Las colisiones ocurren cuando dos o mas registros compiten por la misma dirección
  - Un algoritmo que distribuya los registros de manera aleatoria entre las direcciones disponibles no generará grandes cantidades de registros alrededor de ciertas direcciones

Reduciendo colisiones

- Usar memoria extra:
  - Es fácil encontrar un algoritmo *hash* que evite las colisiones si se tienen pocos registros para distribuir entre muchas direcciones de memoria que cuando se tienen el mismo número de registros y el mismo número de direcciones

Reduciendo colisiones

- Colocar más de un registro en una dirección:
  - Hasta ahora se ha asumido que cada localidad de un registro físico puede contener solamente un registro
  - No existe una razón para que no se pueda crear un archivo de tal manera que cada dirección del archivo pueda contener varios registros

Algoritmo *hash* simple

Objetivo

- El objetivo al elegir un algoritmo de *hashing* debe ser distribuir los registros lo más uniformemente posible

Algoritmo

- Representar la llave de manera numérica
- Unir y sumar
- Dividir entre un número primo y usar el residuo como dirección

Paso 1

- Representar la llave en forma numérica.
  - Si la llave es un número, este paso ya se encuentra completo, si es una cadena de caracteres, se debe tomar el código ASCII de cada carácter y utilizar para formar un número

Paso 1

	76	79	87	69	76	76	32	32	32	32	32
LOWELL=											
	L	O	W	E	L	L					

Paso 2

- Unir dos piezas del número y sumarlas, en el algoritmo que se está manejando se tienen piezas de dos números cada una:

76 79 | 87 69 | 76 76 | 32 32 | 32 32 | 32 32 |

Paso 2

- Estos números pueden ser vistos como enteros

- Por esta razón se pueden realizar operaciones numéricas con ellos

Paso 2 (Problema de desbordamiento)

7679 + 8769 + 7676 + 3232 + 3232 = 30 588

Si se suma el último 3232, se causa un sobreflujo (33 820)

Paso 2

- Cuidando el desbordamiento:
  - En la mayoría de los casos el tamaño de los números que se pueden sumar está limitado, en algunas computadoras los valores enteros que exceden el 32,767 producen un sobre flujo o se expresan en manera negativa

Paso 2

- Utilizando el módulo:
  - Una manera de limitar el resultado de la sumatoria es utilizar la operación módulo
  - Se aplica la operación módulo 19,937

Paso 2

- ¿19,937?
- Se utilizó el 19,937 como límite superior en lugar de otro número por que las operaciones asociadas con el operador módulo son más que una manera de mantener el número pequeño, son parte del trabajo de transformación de la función *hash*
- La división entre un número primo usualmente produce una distribución más aleatoria que la generada con un número que no lo es

Paso 2

$7,679 + 8,769$	$\rightarrow 16,448$	$\rightarrow 16,448 \bmod 19,937$	$\rightarrow 16,448$
$16,448 + 7676$	$\rightarrow 24,124$	$\rightarrow 24,124 \bmod 19,937$	$\rightarrow 4,187$
$4,187 + 3232$	$\rightarrow 7,419$	$\rightarrow 7,419 \bmod 19,937$	$\rightarrow 7,419$
$7,419 + 3232$	$\rightarrow 10,651$	$\rightarrow 10,651 \bmod 19,937$	$\rightarrow 10,651$
$10,651 + 3232$	$\rightarrow 13,883$	$\rightarrow 13,883 \bmod 19,937$	$\rightarrow 13,883$

Paso 3

- Dividir entre el tamaño del espacio de dirección
  - El propósito de este paso es reducir el tamaño del número producido en el paso 2 de tal manera que caiga en el rango de direcciones de registros en el archivo

Paso 3

- Se puede representar esta operación simbólicamente de la siguiente manera:

- Si *s* representa la suma producida en el paso 2 (13,883), *n* representa el divisor (el número de direcciones en el archivo) y *a* representa la dirección que se desea producir, se aplica la fórmula

  - $a = s \bmod n$

Elegir *n*

- La elección de *n* tiene un efecto importante en que tan bien se dispersan los registros
- Un número primo es usualmente utilizado como el divisor debido a que los primos tienden a distribuir los residuos de manera más eficiente que los no primos
- Elegir un número primo lo más cercano posible al tamaño del espacio de direcciones

Paso 3

- Este número determina el tamaño del espacio de direcciones, para un archivo con 74 registros, una buena elección podría ser 101, lo que dejaría el archivo con un 74.3 % de su capacidad (74/101 = 0.743)

Paso 3

- Si el 101 es el tamaño del espacio de direcciones, la dirección del registro que se ha manejado en el ejemplo sería:

$$a = 13{,}883 \bmod 101$$
$$= 46$$

- Por lo tanto, el registro cuya llave es LOWELL será asignado al registro número 46 en el archivo

Ejercicio

- Calcular la dirección en la que se almacenará la cadena COMPUTACION una cadena de 12 caracteres para la llave
- Utilizar modulo 19,937
- Utilizar el número 101 para distribuir entre las direcciones

Código ASCII

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null
SOH	Start of heading
STX	Start of text
ETX	End of text
EOT	End of transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell
BS	Backspace
HT	Horizontal tab
LF	Line feed
VT	Vertical tab

FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
DLE	Data link escape
DC1	Device control 1
DC2	Device control 2
DC3	Device control 3
DC4	Device control 4
NAK	Negative acknowledge
SYN	Synchronous idle
ETB	End of transmission block

CAN	Cancel
EM	End of medium
SUB	Substitute
ESC	Escape
FS	File separator
GS	Group separator
RS	Record separator
US	Unit separator
SP	Space
DEL	Delete

Pseudocódigo

```
int funcionHash(char [12] llave, int maxElementos)
COMIENZA
        PARA (i←0 hasta 12 i+=2)
        COMIENZA
                aux ← 100*llave[i] + llave[i+1];
                suma+=aux;
                suma = suma % 19937
        TERMINA
hashing = suma % maxElementos;
REGRESA hashing;
TERMINA
```

Distribución de registros entre las direcciones

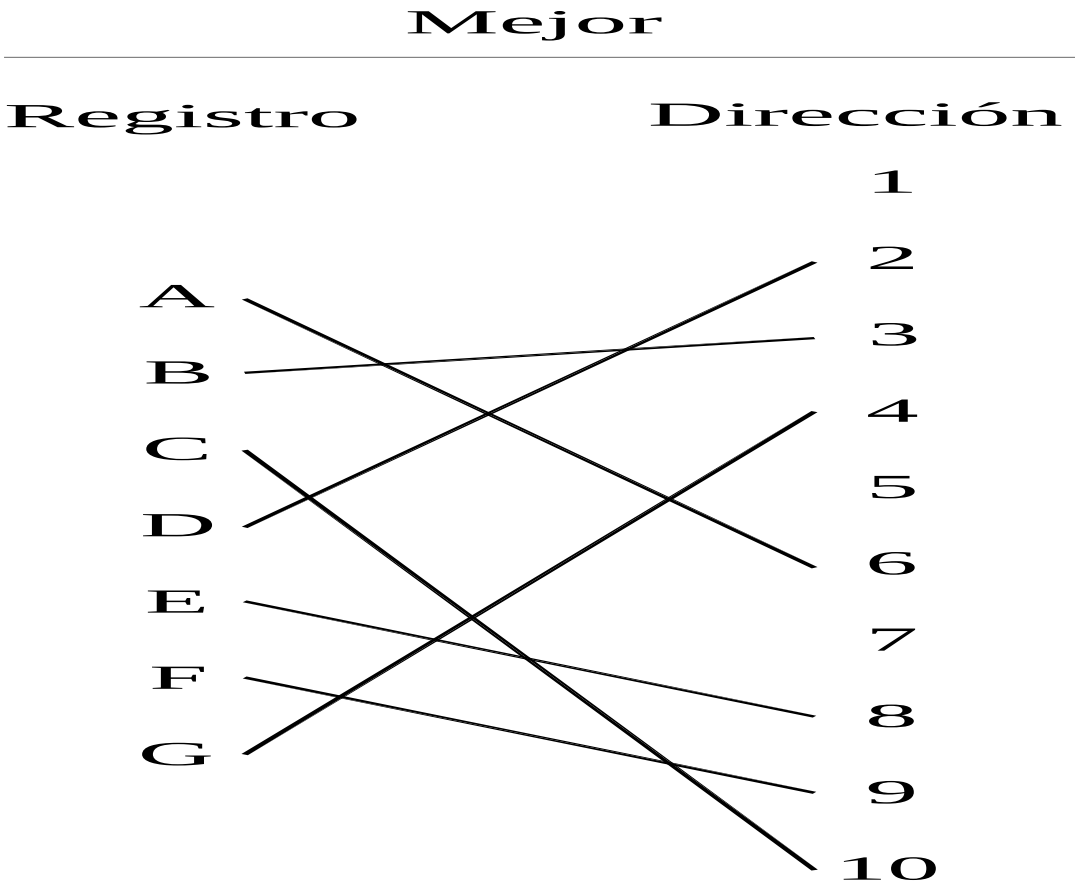
Distribución de registros

- Existen diferentes tipos de distribuciones para los registros en el archivo:
  - Sin sinónimos o uniforme (mejor de los casos)
  - Todos son sinónimos (peor de los casos)
  - Pocos sinónimos

Distribución uniforme

- Es el mejor de los casos
- No existen colisiones
- Cada registro se envía a una dirección diferente

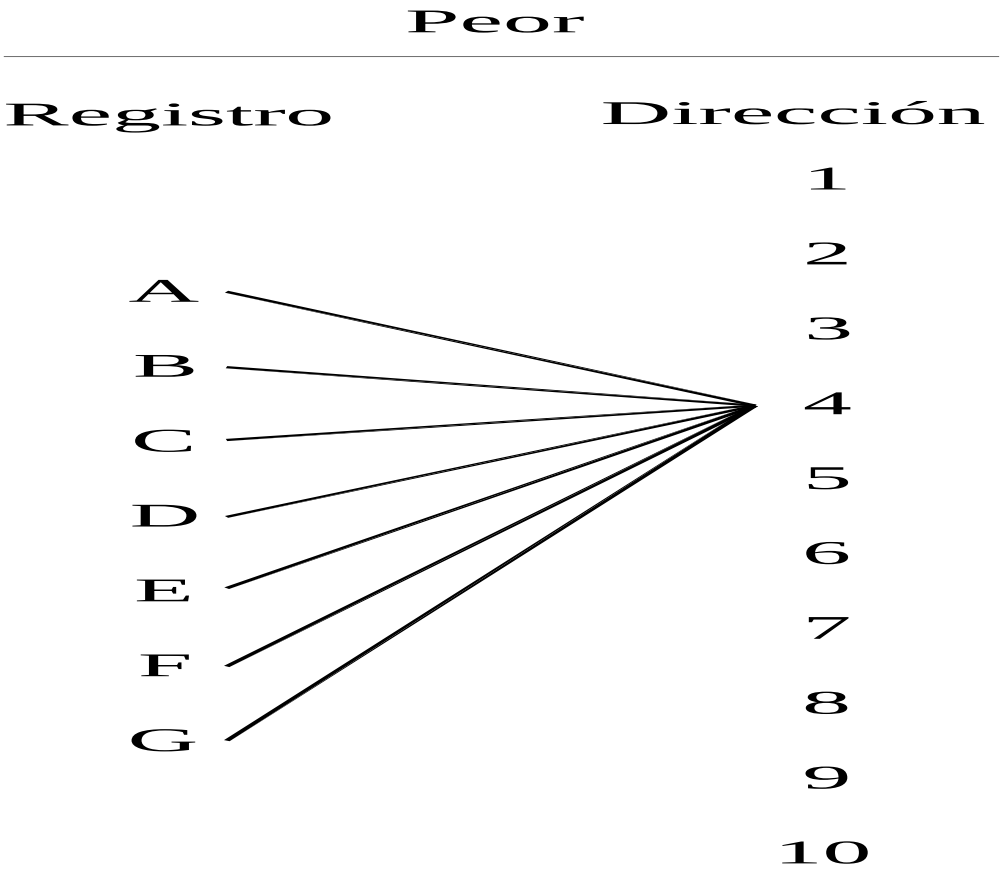
Ejemplo



Peor caso posible

- Es cuando todos los registros son enviados a la misma dirección
- Esto resulta en un máximo de colisiones

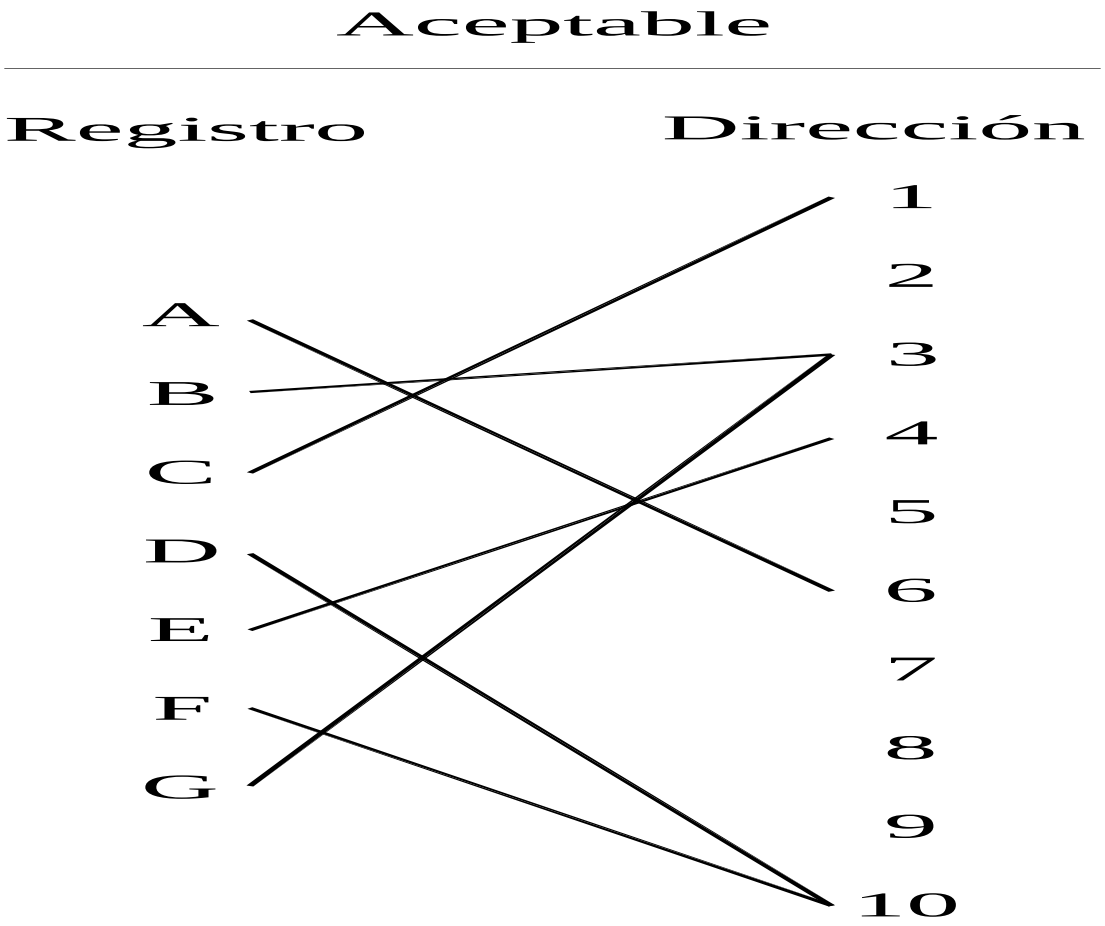
Ejemplo



Caso promedio

- Caso donde se presentan pocas colisiones
- Las llaves se distribuyen de manera aleatoria
- Cada dirección tiene la misma posibilidad de ser elegida

Ejemplo



Otros métodos de *Hashing*

- Otros métodos que pueden ser mejores que una distribución aleatoria:
  - Examinar las llaves en búsqueda de un patrón
  - Unir partes de la llave
  - Dividir la llave entre un número

Examinar las llaves en búsqueda de un patrón

- Algunas veces las llaves caen en patrones que naturalmente se dispersan ellos mismos
- Esto es más común en las llaves numéricas que en las alfabéticas
- Esto puede llevar a no tener sinónimos, si alguna parte de la llave muestra tal patrón, se debe extraer esa parte y utilizarla como llave

Unir partes de llaves

- Involucra la extracción de dígitos de una parte de la llave y la adición de las parte extraídas

- Este método destruye los patrones originales de la llave pero preserva la separación entre ciertos subconjuntos de llaves que se separan entre ellas

Dividir la llave entre un número

- La división entre el tamaño del espacio de direcciones y el uso del residuo se utiliza debido a que el propósito de la función *hash* es producir una dirección dentro de un cierto rango

- La división preserva la secuencia de las llaves consecutivas

- Sin embargo, si existen muchas llaves consecutivas, la división puede resultar en muchas colisiones

Otras opciones

- Existen otros métodos que pueden ser examinados cuando por alguna razón los métodos anteriores no funcionan adecuadamente

Elevar al cuadrado la llave y tomar la mitad

- Tratar la llave como un numero largo, elevar al cuadrado y extraer el número de dígitos necesarios para el resultado
- Ejemplo:

  Llave = 453, $(453)^2 = 205\ 209$

  Extrayendo los dos dígitos de la mitad se obtiene 52, que sería la dirección

Elevar al cuadrado la llave y tomar la mitad

- Este método usualmente produce buenos resultados
- Una desventaja es que se necesita mucha precisión aritmética

Transformar la raíz

- Convertir la llave a otra base que en la que se está trabajando
- Tomar como resultado el módulo con la dirección más grande
- Ejemplo:
  - Direcciones entre 0 y 99
  - La llave es 453, convertida a base 11 equivale a 382
  - 382 mod 99=85

Transformar la raíz

- Este tipo de transformación es más confiable que el método anterior, aunque elevar y extraer los dígitos de la mitad ha demostrado tener buenos resultados con ciertos conjuntos de llaves

Prediciendo la distribución de los registros

Introducción

- Se desea predecir el número de colisiones que podrían ocurrir en un archivo que puede mantener solo un registro por dirección

Predicciones

- Se desea calcular la probabilidad de que:
  - Ninguna llave haya sido enviada a una dirección
  - Exactamente una llave haya sido enviada a una dirección.
  - Exactamente dos llaves hayan sido enviadas a una dirección (dos sinónimos)
  - Exactamente tres, cuatro o más llaves hayan sido enviadas a una dirección
  - Todas las llaves en el archivo tengan la misma dirección

Predicciones

- Suponga que existen *N* direcciones en un archivo, cuando una sola llave es distribuida, existen dos posibilidades respecto a la dirección dada:
  - A - La dirección no es elegida
  - B - La dirección es elegida

Dirección elegida

- La probabilidad de que una cierta dirección sea elegida es:

$$p(B) = b = \frac{1}{N}$$

Dirección no elegida

- La probabilidad de que una cierta dirección NO sea elegida es:

$$p(A) = a = \frac{N-1}{N} = 1 - \frac{1}{N}$$

Llaves distribuidas

- Si se tienen dos llaves, la probabilidad de que ambas llaves sean enviadas a la misma dirección es:

$$p(BB) = bxb = \frac{1}{N} x \frac{1}{N}$$

Llaves distribuidas

- La probabilidad de que la segunda llave sea distribuida hacia otra dirección diferente a la que tiene la primera es:

$$p(AB) = bxa = \frac{1}{N} x \left(1 - \frac{1}{N}\right)$$

Distribuciones

- En general, se puede conocer la probabilidad de que cierta secuencia ocurra
- Ejemplo:

N=10 direcciones

BABBA

p(BABBA) = b x a x b x b x a

$=a^2b^3 = (0.9)^2(0.1)^3$

Ejemplo

- Suponer que se tienen 4 llaves para distribuir entre 10 direcciones

- Calcular la probabilidad de que exactamente dos ocupen la misma dirección

Solución

- 1. Obtener todas las combinaciones posibles:

AAAA
AAAB
AABA
AABB
ABAA
ABAB
ABBA
ABAA
BAAA
BAAB
BABA
BABB
BBAA
BBAB
BBBA
BBBB

Solución

- Extraer aquellas que cumplen lo especificado

AABB
ABAB
ABBA
BAAB
BABA
BBAA

Solución

- La probabilidad de cada caso es b^2a^2

$$\left(1-\frac{1}{N}\right)^2\left(\frac{1}{N}\right)^2 = (0.9)^2(0.1)^2 = 0.0081$$

Ejercicio

- Se distribuyen 3 llaves en 15 direcciones, calcular la probabilidad de que una dirección contenga exactamente 2 llaves.

- Escribir los casos en los que esto sucedería

Solución

- Como se tienen seis posibles combinaciones independientes una de la otra, la probabilidad es la suma de cada una:

$$= 6xa^2b^2 = 6(0.0081) = 0.0486$$

Función de *Poisson*

- Se puede utilizar la función de *Poisson* para estimar la probabilidad de que una dirección dada pueda tener un cierto número de registros

Función de *Poisson*

- La función de *Poisson*, que se denota por *p(x)* está dada por:

$$p(x) = \frac{(r/N)^x e^{-(r/N)}}{x!}$$

- N = el número de direcciones disponibles
- r = el número de registros para ser almacenados
- x = el número de registros asignados a la dirección dada

Función de *Poisson*

- Entonces $p(x)$ es la probabilidad de que una dirección dada pueda contener x registros asignada a ella

Ejemplo

- Se distribuirán 10 llaves en 25 direcciones

- Calcular la probabilidad de que una dirección contenga 6 llaves

Solución

$$p(6) = \frac{(10/25)^6 e^{-(10/25)}}{6!}$$

$$p(6) = \frac{(4.096x10^{-3})e^{-0.4}}{720}$$

$$p(6) = 3.8133x10^{-6}$$

Ejercicio

- Utilizando los datos del ejemplo anterior: calcular la probabilidad de que una dirección contenga:
  - Cero llaves
  - Dos llaves

Otras funciones de *Poisson*

- La función de *Poisson* se puede utilizar para predecir el número de direcciones que tendrán un cierto número de registros asignados

- Si se tienen *N* direcciones, entonces el número de direcciones esperado con *x* registros asignados a ellas es:

$$Np(x)$$

Ejemplo:

- Del ejemplo anterior, el número de direcciones que contendrían exactamente 6 llaves es:

$$25 * p(6)$$
$$25 * 3.8133 \times 10^{-6}$$
$$=9.533 \times 10^{-5}$$

- Es decir que ninguna dirección contendría seis llaves

Ejercicio

- Calcular cuantas direcciones tendrían:
  - Cero llaves asignadas
  - Exactamente dos llaves asignadas

Direcciones con sinónimos

- Para calcular la cantidad de direcciones que contendrán uno o más sinónimos se tiene:

$$N[p(2) + p(3) + p(4) + \ldots]$$

- Normalmente se utilizan solo las *p()* que aportan valores significativos

Porcentaje de registros con sobre flujo

- Si solo un registro puede asignarse a una dirección la cantidad de registros que causarán sobre flujo es:
  - Para *p(2)*, un registro no genera sobre flujo y uno si
  - Para *p(3)*, un registro no genera sobre flujo y dos si
  - Para *p(n)*, un registro no genera sobre flujo y *n-1* si

Porcentaje de registros con sobre flujo

- N x [p(2) + 2 x p(3) + 3 x p(4) + *n-1* x *p(n)*]
- Donde *n* representa el número de llaves

Densidad de empaquetamiento

Introducción

- La elección de un algoritmo adecuado es fundamental en las funciones *hashing*

- Otra manera de bajar el número de colisiones es utilizar memoria extra

Densidad de empaquetamiento

- Se refiere a la cantidad de registros (r) que serán almacenados en los espacios disponibles (N):

$$\frac{Número\ de\ registros}{Número\ de\ espacios} = \frac{r}{N}$$

Densidad de empaquetamiento

- La densidad de empaquetamiento proporciona una medida de la cantidad de espacio que es utilizado en un archivo

- Es el único valor necesario para evaluar el desempeño cuando se trabaja con *hashing*

Densidad de empaquetamiento

- Entre más registros se coloquen en un tamaño de archivo dado, mayor será la probabilidad de que se tengan colisiones

Prediciendo colisiones

- Se necesita una descripción cuantitativa de los efectos de modificar la densidad de empaquetamiento

- Se necesita predecir el número de colisiones que pudieran ocurrir para una densidad dada

Ejemplo

- Suponga que se tienen 1000 direcciones para contener 500 registros en un archivo distribuido de manera aleatoria, y que cada dirección puede contener un registro.

Calcular:

- 1. La densidad de empaquetamiento
- 2. Cuántas direcciones podrían no tener ningún registro asignado a ellas.
- 3. Cuántas podrían tener solo un registro asignado a ellas
- 4. Cuántas tendrían un registro y uno o más sinónimos asignados a ellas.
- 5. Asumiendo que solo un registro pueda ser asignado a cada dirección, cuantos registros que causen sobre flujo pueden esperarse.
- 6. Qué porcentaje de direccionestendrán sobre flujo.

1. Densidad de empaquetamiento

$$\frac{r}{N} = \frac{500}{1000} = 0.5 \quad \rightarrow 50\%$$

2. Direcciones sin llave/registro asignada a ellas

$$p(0) = \frac{(500/1000)^0 e^{-(500/1000)}}{0!} = \frac{e^{-0.5}}{0!} = 0.60653$$

$$Np(0) = 1000x0.60653 = 607 \ direcciones$$

3. Direcciones con un registro asignado a ellas

$$p(1) = \frac{(500/1000)^1 e^{-(500/1000)}}{1!} = \frac{(0.5)e^{-0.5}}{1!} = 0.3032$$

$$Np(1) = 1000x0.3032 = 303\ direcciones$$

Direcciones con uno o más registros asignados a ellas

$$p(2) = \frac{(500/1000)^2 e^{-(500/1000)}}{2!} = \frac{(0.5)^2 e^{-0.5}}{2!} = 0.0758$$

$$p(3) = \frac{(500/1000)^3 e^{-(500/1000)}}{3!} = \frac{(0.5)^3 e^{-0.5}}{3!} = 0.0.126$$

$$p(4) = \frac{(500/1000)^4 e^{-(500/1000)}}{4!} = \frac{(0.5)^4 e^{-0.5}}{4!} = 0.001595$$

$$1000\left[p(2) + p(3) + p(4)\right] = 90 \textit{ direcciones con sobreflujo}$$

Direcciones que causarán sobre flujo

$$N\left[1xp(2)+2xp(3)+3xp(4)\right]$$
$$=1000\left[0.0758+2x0.01263+3x0.001595\right]$$
$$=1000\left[0.0758+0.02526+0.004785\right]$$
$$=1000\left[0.06791\right]$$
$$=106\ direcciones$$

Porcentaje de registros con sobre flujo

$$\frac{106}{500} = 21.2\%$$

Solucionando el problema de colisiones

Introducción

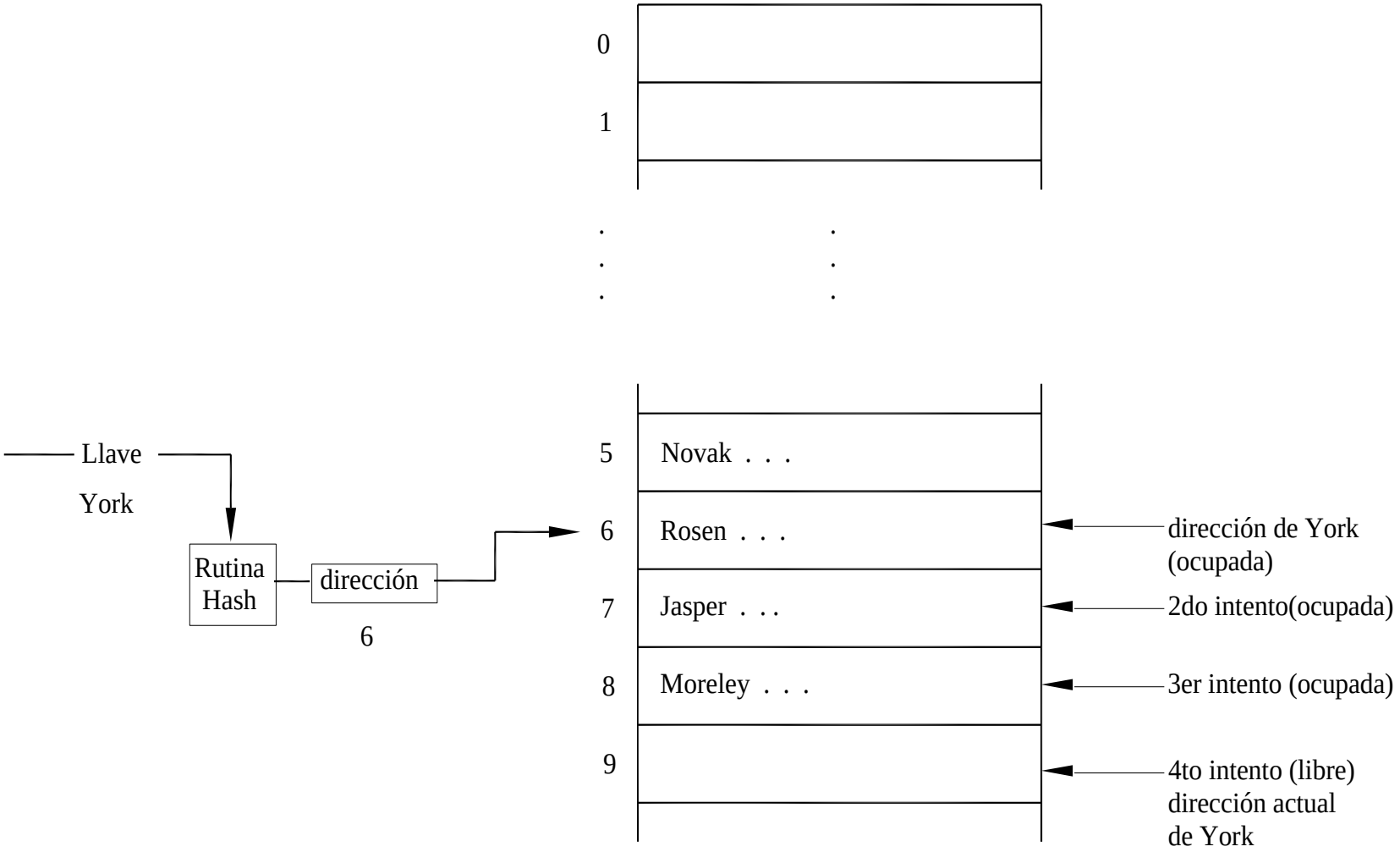
- Aunque el algoritmo *hash* sea eficiente, es muy probable que ocurran colisiones

- Existen diferentes técnicas para solucionar este problema:

  - Sobre flujo progresivo
  - Baldes o cubetas

Sobre flujo progresivo

Sobre flujo progresivo



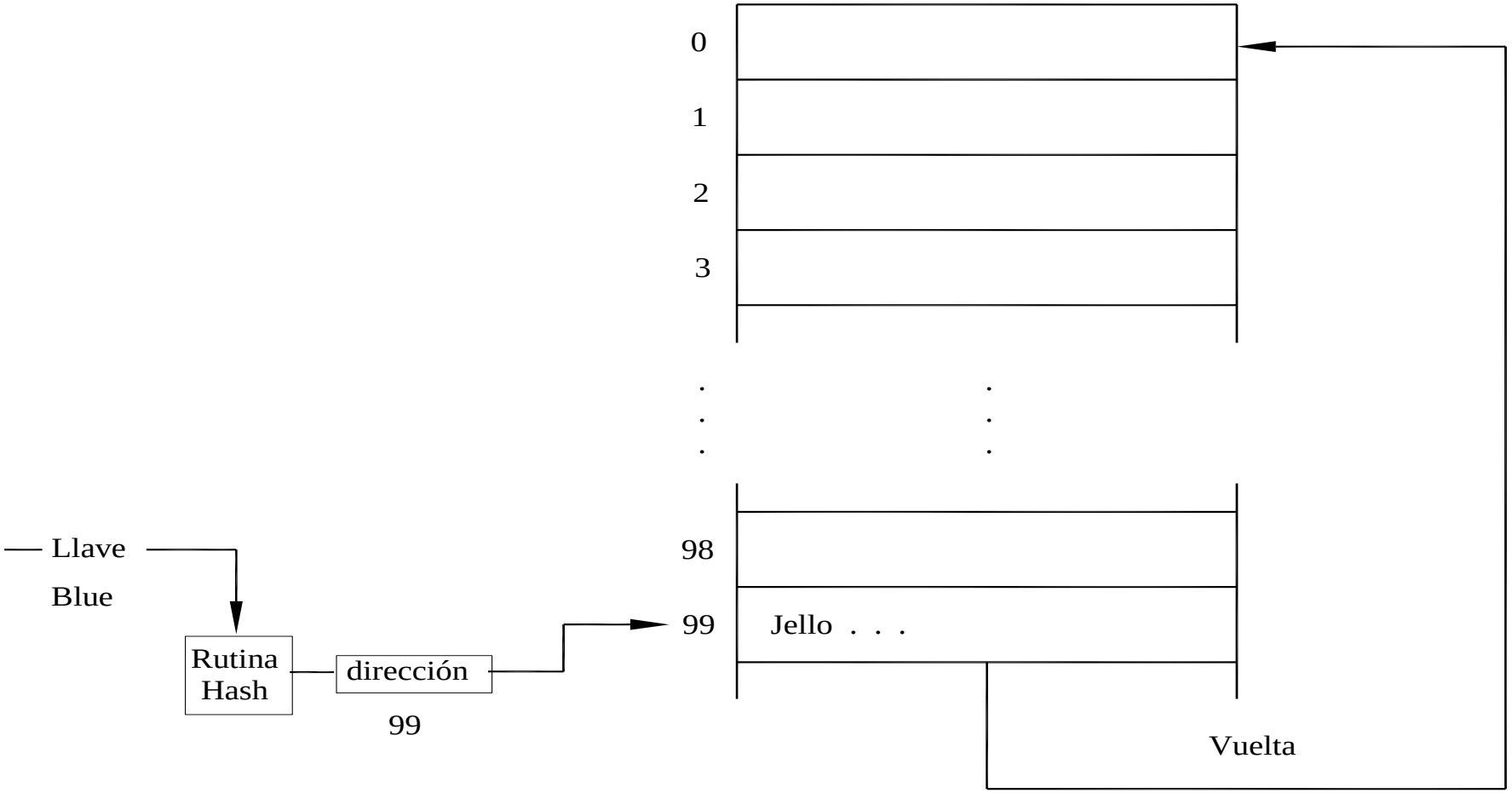
Solución

- Se desea almacenar un registro cuya llave es York
- Se distribuye a la misma dirección que la llave Rosen y se produce un sobre flujo
- El método de sobre flujo progresivo, busca en las siguientes direcciones sucesivas en secuencia hasta que encuentra una vacía que en este caso es la dirección 9

Fin de archivo

- Cuando se desea buscar un elemento, o una posición para insertarlo y se llega al final del archivo, se comienza a recorrer desde el inicio

Búsqueda



Búsqueda

- Una búsqueda de un registro comienza en la dirección apuntada por su llave

- Cuando se busca un registro, pero este no fue colocado en el archivo, pueden ocurrir dos casos:

Posibles casos

- Si se encuentra una dirección vacía, la rutina de búsqueda supone que el registro no se encuentra en el archivo

- Si el archivo está lleno, y se regresa a donde se había comenzado la búsqueda, está claro que el registro no se encuentra en el archivo, cuando esto ocurre la búsqueda se puede convertir en un procedimiento muy lento

Ventajas

- La principal ventaja de la técnica de sobreflujo progresivo es su simplicidad

Desventajas

- La razón para evitar el sobre flujo progresivo, son las búsquedas extras que se tienen que realizar en algún momento

- Si se tiene un gran número de colisiones, habrá una gran cantidad de registros que no ocupen el lugar que les corresponde

Longitud de búsqueda

- Se refiere al número de accesos requeridos para recuperar un registro de la memoria secundaria

- En el contexto de *hashing* la longitud de búsqueda para un registro se incrementa cada vez que se presenta una colisión

Promedio de longitud de búsqueda

- El promedio de la longitud de búsqueda se define de la siguiente manera:

$$\frac{longitud\ \ de\ busqueda\ total}{número\ de\ registros}$$

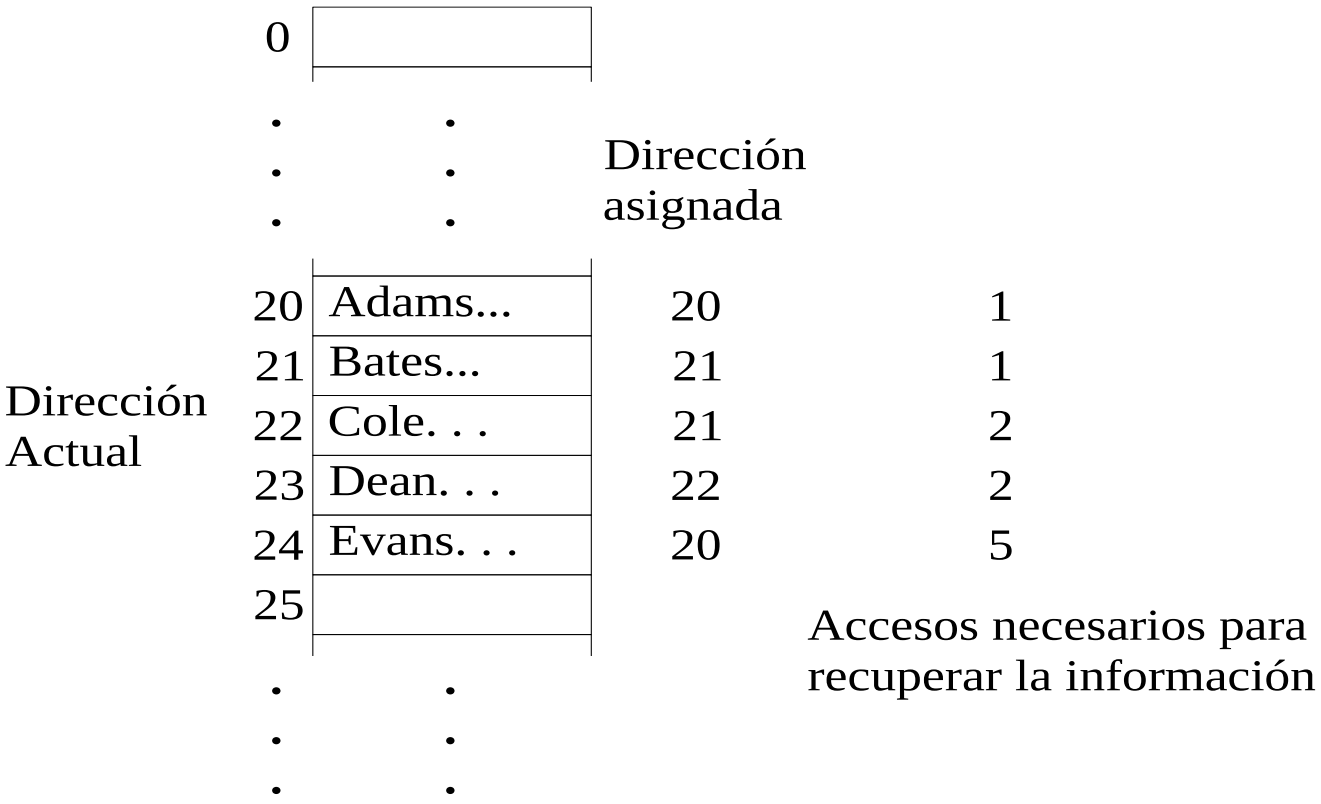
Ejemplo

Llave	Dirección
Adams	20
Bates	21
Cole	21
Dean	22
Evans	20

Ejemplo, cont...

- Se insertan en un archivo inicialmente vacío
- Si se utiliza el sobre flujo progresivo para resolver colisiones, solo dos registros se encontrarán en la dirección que les fue asignada
- El resto requerirá accesos extra para su recuperación

Ejemplo:


0
Dirección asignada
Dirección Actual
20 Adams... 20 1
21 Bates... 21 1
22 Cole. . . 21 2
23 Dean. . . 22 2
24 Evans. . . 20 5
25
Accesos necesarios para recuperar la información

Ejemplo...

- El promedio de la longitud de búsqueda en el archivo es:

- Sin colisiones, el promedio de la longitud de búsqueda es 1

$$\frac{1+1+2+2+5}{5} = 2.2$$

Desventajas

- Si se tiene una gran cantidad de colisiones, la búsqueda promedio se vuelve muy costosa

Uso de baldes o cubetas

Almacenamiento en baldes

- Algunas veces es ventajoso pensar en los registros siendo agrupados juntos en bloques más que individualmente

- La palabra balde o cubeta, se utiliza para describir un bloque de registros que es recuperado en un acceso al disco, especialmente cuando esos registros comparten una dirección

Ejemplo

- Se tienen las siguientes llaves y sus direcciones *hash*:

Llave	Dirección
Green	30
Hall	30
Jenks	32
King	33
Land	33
Marx	33
Nutt	33

Ejemplo, cont...

- Aquí cada dirección es un balde capaz de contener a los registros que son sinónimos
- Cada una es capaz de contener tres sinónimos
- Cuando un registro es almacenado o recuperado, su dirección del balde es determinada por la función *hashing*

Uso de baldes

Contenido del balde

Green. . .	Hall. . .	
Jenks. . .		
King. . .	Land. . .	Marks. . .

Ejemplo, cont...

- El balde completo es cargado en memoria primaria y una búsqueda sucesiva en el bloque recupera la llave deseada

- Con esta técnica, los sobre flujos ocurren con mucha menor frecuencia que cuando se almacena solo un registro por dirección

Desempeño de los baldes

- La fórmula utilizada para calcular la densidad de empaquetamiento es cambiada dado que cada dirección de balde puede contener más de un registro

Desempeño de los baldes

- Se debe considerar el número de direcciones (baldes) y el número de registros que se puede colocar en cada dirección (tamaño del balde)

Desempeño

- Si N es el número de direcciones y b es el número de registros que caben en un balde, entonces bN es el número disponible de localidades para los registros, si r es el número de registros en el archivo, entonces:

$$densidad\ de\ empaquetamiento = \frac{r}{bN}$$

Ejemplo

- Suponer que se tiene un archivo en donde 750 registros serán almacenados, considerar que hay dos maneras de organizar el archivo:

  - Almacenar los 750 registros en 1000 localidades, donde cada localidad pueda contener un solo registro, la densidad sería:

$$\frac{750}{1000} = 75\%$$

Desempeño

- Almacenar los 750 registros en 500 localidades, donde cada localidad puede contener 2 registros

$$\frac{750}{2 \; x \; 500} = 75\%$$

Desempeño

	Archivo sin baldes	Archivo con baldes
Número de registros	r=750	r=750
Número de direcciones	N=1000	N=500
Tamaño del balde	b=1	b=2
Densidad de empaquetamiento	0.75	0.75
Rango de registros a direcciones	r/N=0.75	r/N=1.5

Desempeño

- Para determinar el número de registros con sobre flujo que se esperan en cada archivo, se utiliza la función de *Poisson:*

$$p(x) = \frac{(r/N)^x e^{-(r/N)}}{x!}$$

Desempeño

	Archivo sin baldes	Archivo con baldes
p(x)	*(r/N=0.75)*	*(r/N=1.5)*
p(0)	0.472	0.223
p(1)	0.354	0.335
p(2)	0.133	0.251
p(3)	0.033	0.126
p(4)	0.006	0.047
p(5)	0.001	0.014
p(6)	---	0.004
p(7)	---	0.001

Desempeño

- Existe una mayor probabilidad de que una misma llave se introduzca en una dirección utilizando baldes

- Sin embargo, la mitad de esos sinónimos no resultan en un sobre flujo

Desempeño

- Calculando el número de registros para los que se espera un sobre flujo se tiene

$$N\ x\ [1\ x\ p(2) + 2\ x\ p(3) + 3\ x\ p(4) + 4\ x\ p(5) + \ldots]$$

$$si\ r/N = 0.75\ y\ N = 1000:$$

$$1000\ x\ [1\ x\ 0.1328 + 2\ x\ 0.0332 + 3\ x\ 0.0062 + 4\ x\ 0.0009 + 5\ x\ 0.0001] = 222$$

- El 222 representa un 29.6% de registros con sobre flujo.

Desempeño con baldes

- Para el caso donde el tamaño del balde es 2 y se utilizan 500 direcciones, se tiene:

$$N\ x\ [1\ x\ p(3) + 2\ x\ p(4) + 3\ x\ p(5) + 4\ x\ p(6) + ...]$$

$$si\ r/N = 1.5\ y\ N = 500:$$

$$500\ x\ [1\ x\ 0.1255 + 2\ x\ 0.0471 + 3\ x\ 0.0141 + 4\ x\ 0.0035 + 5\ x\ 0.0008] = 140$$

- El 140 representa que el 18.7% de las direcciones tendrán sobre flujo

Desempeño

- El uso de baldes mejora el desempeño de *hashing* de manera sustancial
- Sin embargo no es posible incrementar el tamaño del balde de manera arbitraria
- Una regla para determinar el tamaño máximo es que no se utilicen baldes de tamaño mayor que una pista

Implementación

Conceptos

- Dado que una función *hash* depende de que exista un número de direcciones disponibles, el tamaño lógico de un archivo *hash* debe ser calculado antes de que pueda llenarse con registros

Conceptos

- Dado que el *RRN* de un registro en el archivo está relacionado únicamente con su llave, cualquier procedimiento de añadir, eliminar o modificar un registro debe hacerse sin modificar el lazo que existe entre el archivo y su dirección, si este lazo se rompe, el registro ya no estará disponible

Estructura del balde

- La única diferencia entre un archivo con baldes y uno en donde solo se puede tener un registro por dirección es que en un archivo con baldes, cada dirección tiene espacio suficiente para contener más de un registro, y todos los que se encuentren en el mismo balde comparten la dirección

Estructura del balde

- Cada balde contiene un contador que mantiene el dato de cuantos registros se han almacenado en él

- Las colisiones se presentan cuando la adición de un nuevo registro causa que el contador exceda el número de registros que el balde puede contener

Inicializando el archivo *Hash*

- Dado que el tamaño de un archivo *hash* debe permanecer fijo, se debe reservar el espacio físico para el archivo antes de que se coloquen los registros en él

- Esto es realizado creando un archivo con espacios vacíos para los registros, y después llenándolos conforme se agreguen datos

Llenado de un archivo

- Un programa que llena un archivo *hash* es similar a los que se han manejado con los archivos índices con dos diferencias:
  - El programa utiliza la función *hash* para producir una dirección para cada llave
  - El programa busca un espacio libre comenzando con la dirección indicada, si se encuentra ocupada, o el balde está lleno continúa buscando.
  - Finalmente el nuevo registro es insertado en el archivo

Eliminación

- Eliminar un registro es más complicado que insertarlo debido a dos razones:
  - El espacio liberado por la eliminación no debe estorbar a búsquedas posteriores
  - Debe ser posible reutilizar el espacio para futuras adiciones

Problemas con la eliminación

- Cuando el sobre flujo progresivo es utilizado, una búsqueda para un registro termina cuando se encuentra una dirección vacía, debido a esto, no se deben dejar direcciones vacías

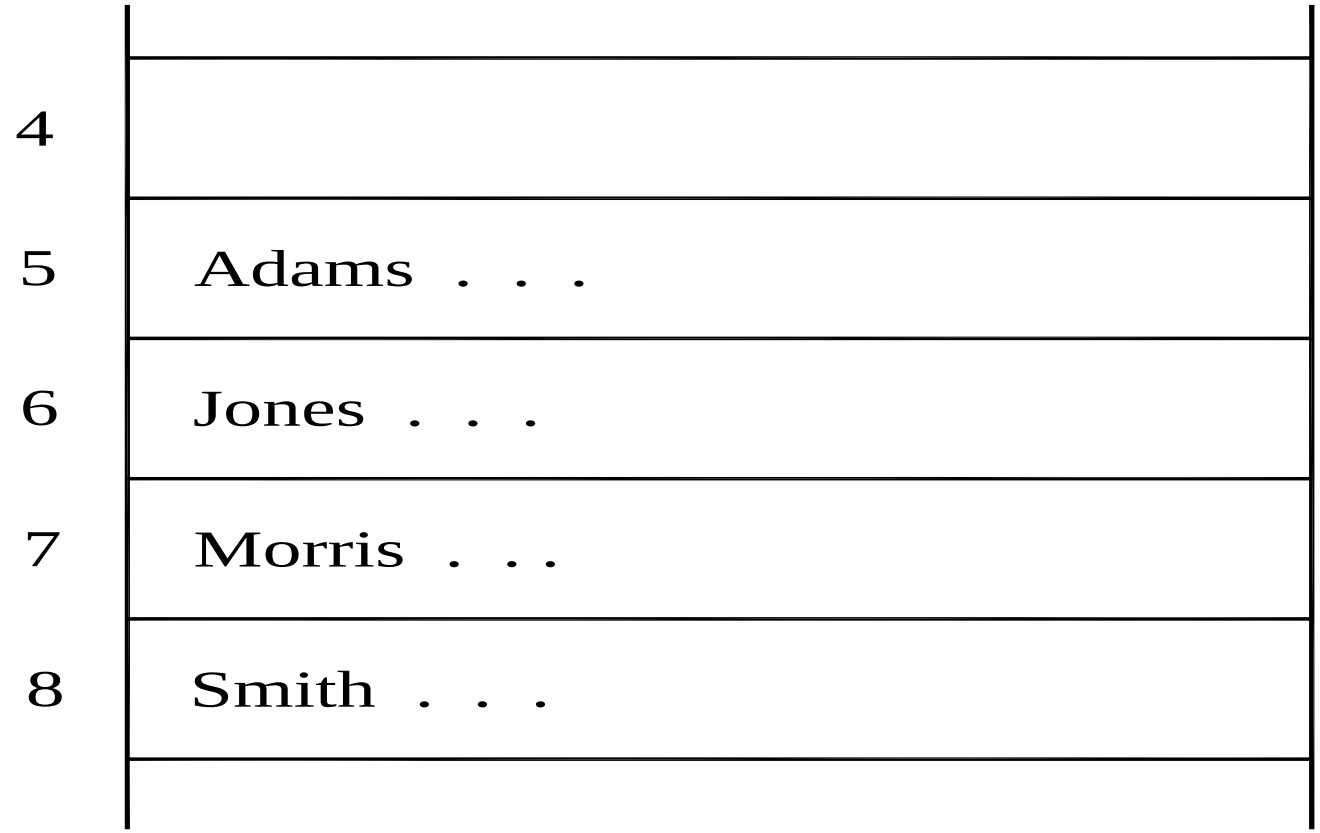
Ejemplo

- Por ejemplo, Adams, Jones, Morris y Smith son almacenados en un archivo *hash* cuyas direcciones pueden contener solo un registro, Adams y Smith son enviados a la dirección 5 y Jones y Morris a la 6

Ejemplo, cont...

Registro	Dirección asignada	Dirección actual
Adams	5	5
Jones	6	6
Morris	6	7
Smith	5	8

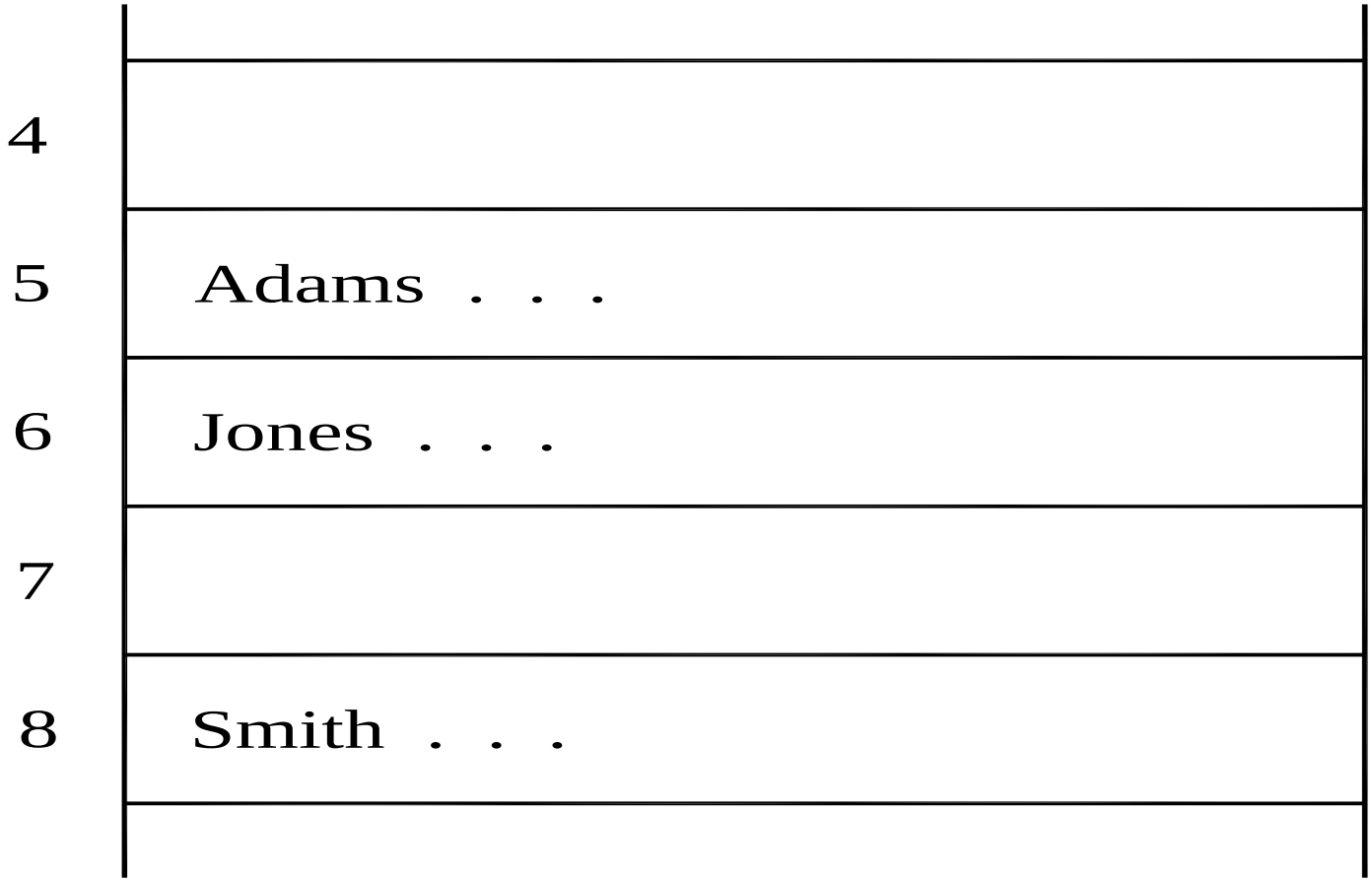
Ejemplo, cont...



Ejemplo, cont...

- Si se elimina Morris, queda un espacio vacío, una búsqueda por Smith comenzaría en la dirección 5, y posteriormente buscaría en las direcciones 6 y 7, como la dirección 7 está vacía, la búsqueda termina y el programa supone que Smith no se encuentra en el archivo

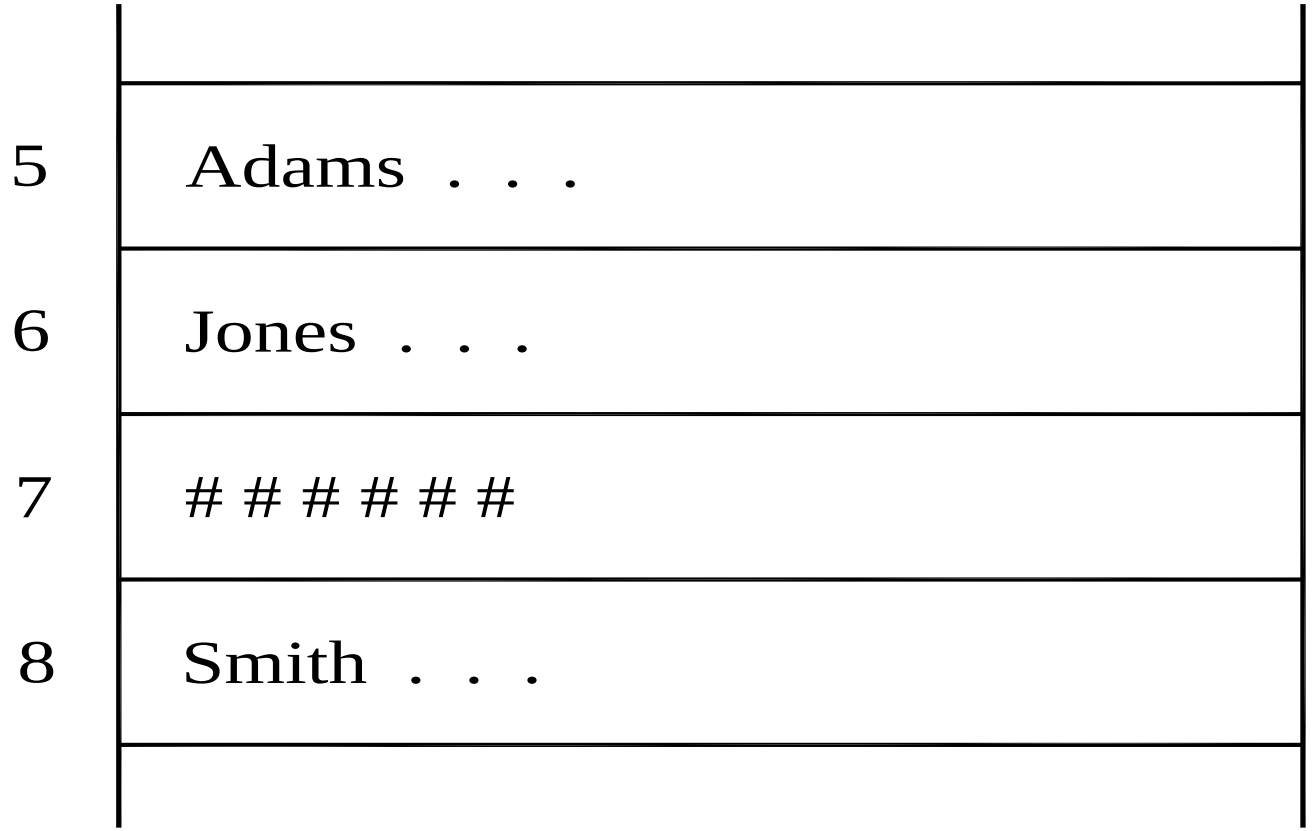
Ejemplo, cont...



Lápidas

- Una solución consiste en reemplazar el archivo eliminado (o solo su llave) con una marca que indique que no se encuentra disponible, este tipo de marcas son conocidas como “lápidas” y resuelven los problemas descritos previamente:
  - El espacio liberado no rompe la secuencia de una búsqueda
  - El espacio liberado está disponible para futuras inserciones

Uso de lápidas



Manejo de lápidas

- Con la introducción del uso de lápidas, la inserción de registros se complica un poco más, debido a que los programas buscan por un espacio vacío, o en este caso con la llave que indique una lápida, si el programa encuentra esta llave, posiblemente no note que ya se encontraban los datos en el registro

Manejo de lápidas

- Para prevenir esto, el programa debe recorrer el cluster entero de llaves y lápidas contiguas para asegurar que no existan llaves duplicadas

- Posteriormente debe regresar e insertar el registro en la dirección adecuada

Otros métodos de manejo de colisiones

Introducción

- A pesar de su simplicidad el uso de *hashing* aleatorio con sobre flujo progresivo y un tamaño de balde adecuado da un rendimiento adecuado

- Si el rendimiento no es suficientemente adecuado, existen otras técnicas

Doble *Hashing*

Conceptos

- Uno de los problemas con el sobre flujo extendido es que muchos registros pueden ser distribuidos en baldes que son continuos
- Cuando esto ocurre se forman clusters de registros
- Esto lleva a búsquedas muy costosas

Funcionamiento

- Una manera de evitar esto es almacenar el registro utilizando el valor *hash* del valor *hash* obtenido previamente

Conceptos

- Cuando una colisión ocurre, una segunda función *hash* es aplicada a la llave para producir un número *c* que es un número primo relativo al número de direcciones

- Si esa nueva dirección (dirección de sobre flujo) ya está ocupada, *c* se suma a ella para producir otra dirección, este procedimiento continúa hasta que se encuentra una dirección disponible

Sobre flujo progresivo encadenado

Sobre flujo progresivo encadenado

- Los sinónimos son ligados juntos utilizando apuntadores

- Cada dirección contiene un número indicando la localidad del siguiente registro que se encontraría en la misma dirección

Ejemplo

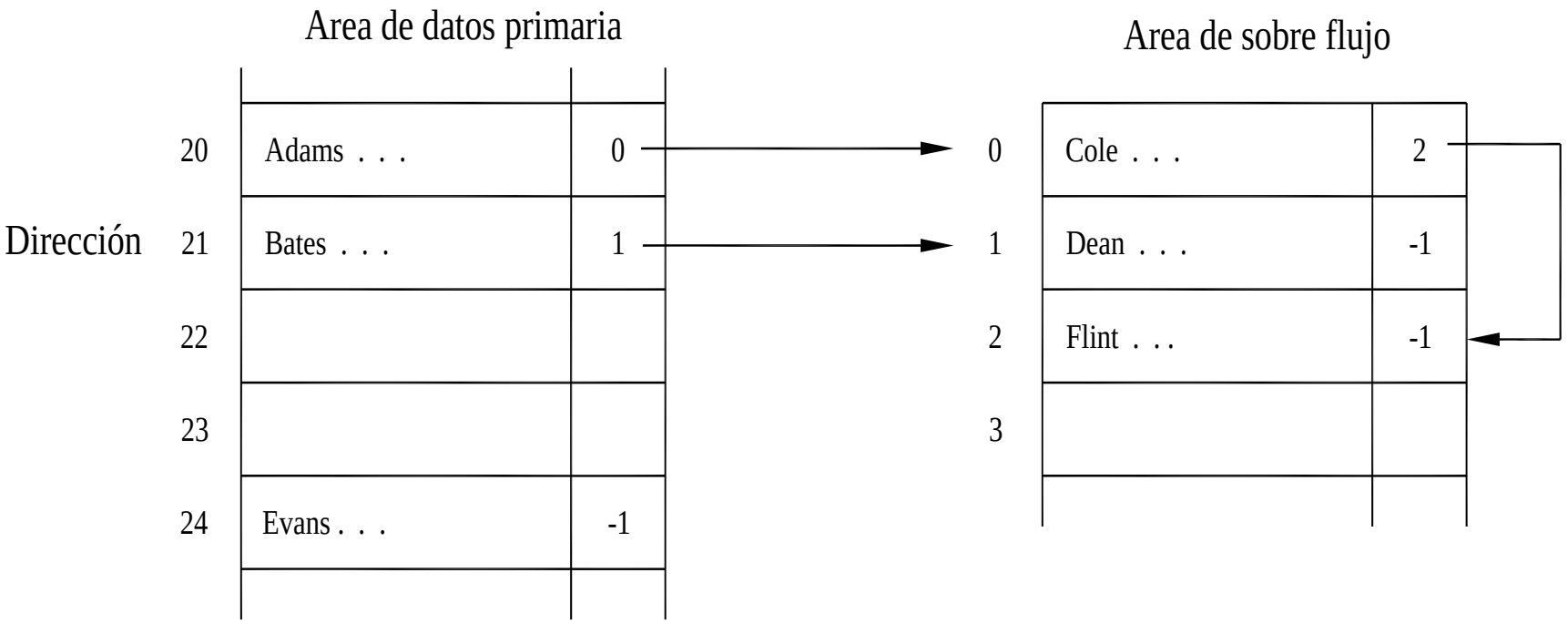
- Adams, Cole y Flint son sinónimos
- Bates y Dean son sinónimos

Dirección asignada	Dirección actual	Llave	Dirección del sinónimo	Longitud de búsqueda
20	20	Adams	22	1
21	21	Bates	23	1
20	22	Cole	25	2
21	23	Dean	-1	2
24	24	Evans	-1	1
20	25	Flint	-1	3

Área especial para sobre flujo

- Una manera de evitar que los registros que causan sobre flujo ocupen direcciones que no les corresponden es moverlos a un área separada
- El conjunto de direcciones es llamada el área de datos primaria
- El conjunto de direcciones para los registros el sobre flujo es llamada área de sobre flujo

Ejemplo


Area de datos primaria
Area de sobre flujo
Dirección
20
Adams . . .
0
21
Bates . . .
1
22
23
24
Evans . . .
-1
0
Cole . . .
2
1
Dean . . .
-1
2
Flint . . .
-1
3

Ventaja y desventaja

- El uso de un área separada para sobre flujo simplifica el proceso de eliminación y adición

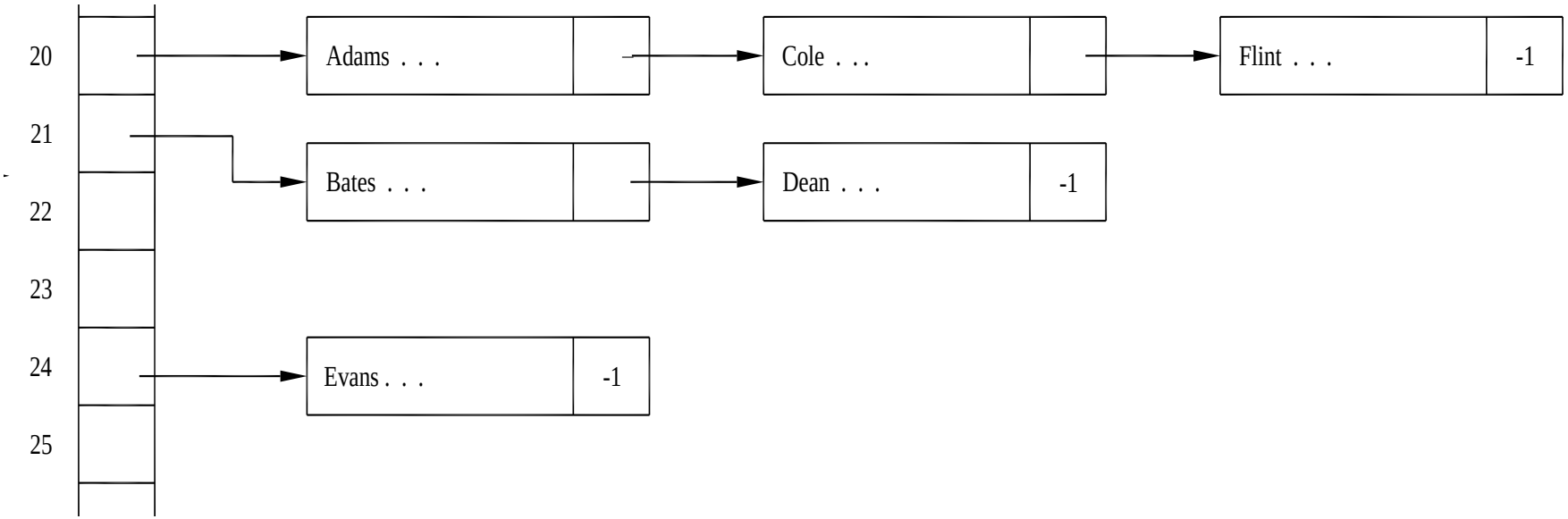
- Si el área de sobre flujo se encuentra en un cilindro diferente cada búsqueda de un registro de sobre flujo puede involucrar un movimiento de cabezas de lectura muy costoso

Tablas esparcidas

- Archivo *hash* que no contiene registros, sino solamente apuntadores a los registros

- El archivo es un índice que es examinado por *hashing* en lugar de otro método

Ejemplo



Ventajas

- Este tipo de organización provee muchas de las ventajas del manejo de índices

- Además la búsqueda en el archivo de índices solo requiere un acceso