Tablas Hash

- Una **tabla de dispersión o tabla hash** es un espacio de memoria donde se almacenan registros con información asociada a una serie de ***valores***.
  - La idea básica consiste en utilizar una función del *valor de referencia* (*función hash o función de dispersión*) que proporcione directamente la posición donde debe estar almacenada la información correspondiente a cada valor.
  - A estos registros ***se accede*** directamente a través de una dirección de memoria determinada por la función *hash* del ***valor de referencia***.
- Las tablas hash constituyen una herramienta muy importante para la manipulación y almacenamiento de datos en memoria secundaria.

Tablas Hash (ejemplo 1)

Ejemplo:

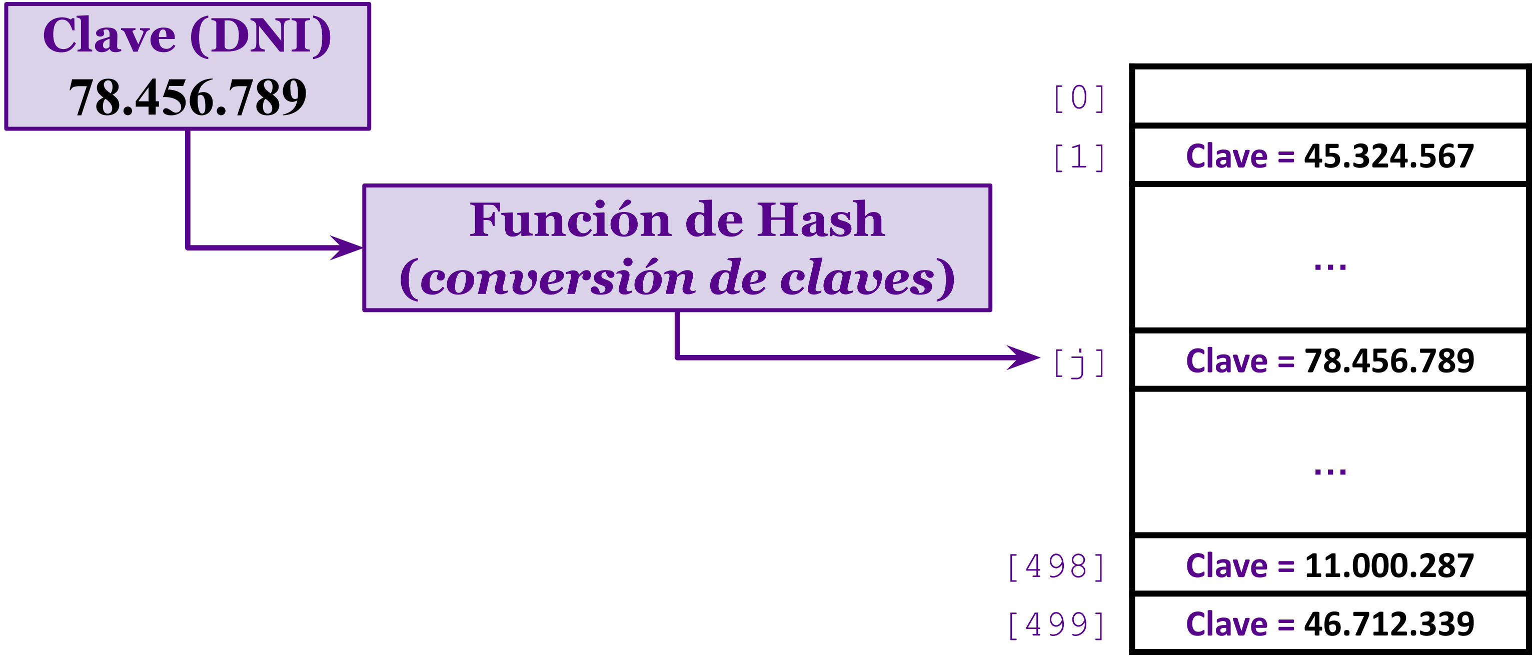
- **Empresa** con **100** empleados.
- Cada empleado tiene un *número de identificación* (**clave**) que va del **1** al **100**.
- Puede existir una ***correspondencia directa*** entre la clave (número de identificación de empleado) y la posición que ocupa ese empleado dentro del vector de **100** elementos.
- ¿Y si no tuviéramos número de identificación para cada empleado y tuviéramos que **usar su DNI**?
  ¿O si el número de empleados fuera elevado?

Tablas Hash (ejemplo 2)

Ejemplo:

- Supongamos que se quieren almacenar fichas con datos de **los alumnos** de este curso usando el ***número del DNI*** como valor de referencia.
- Dado que estos números tienen hasta **8 cifras**, hay del orden de **100.000.000** de valores posibles pero es absurdo reservar memoria para todos si sabemos que el número de alumnos no llegará a los **300**.
- Bastaría con reservar memoria para **300**, o a lo sumo para **500** fichas.
- El problema de cómo establecer la **ubicación de las fichas** de datos correspondiente a cada posible valor para que su posterior gestión sea **eficiente** se resuelve con la implementación de una tabla de dispersión.

Tablas Hash (esquema)



Función de dispersión (definición)

- Una ***función de dispersión, transformación o hashing*** es una función que, aplicada al valor de referencia, proporcionará la dirección donde se encontrará la información asociada a dicho valor.

- Dado:
  - un conjunto ***K*** de valores posibles (claves), y
  - un conjunto ***D*** de direcciones de memoria de una tabla

  una función de transformación, de dispersión o función ***hash*** es una aplicación ***h***:

$$\boxed{h: K \longrightarrow D}$$

Colisiones

- En las aplicaciones prácticas, el número de claves posibles es ***mucho mayor*** que el de los datos que realmente se van a almacenar.
- Por tanto, las funciones de transformación que se puedan utilizar serán siempre supravectivas:
    - Existirán varios valores que se transformarán en una misma posición de memoria y a esos valores se le denominan **sinónimos**.
- Si en una aplicación aparecen sinónimos se dice que se produce una **colisión** y, dado que no son totalmente inevitables, se tiene que disponer de una herramienta para **tratar** estas circunstancias de forma ***inteligente***.

Buenas funciones de dispersión

- Existen muchos métodos de transformación de claves.
- Una buena función de transformación para una tabla de dispersión debe ser <u>sencilla y rápida de computar</u>.
- Otro aspecto importante en la selección de una función de dispersión es ***evitar en lo posible las colisiones***.
    - El menor número de colisiones se produce si se utiliza toda la tabla con la misma *probabilidad*; se usan **funciones de dispersión uniformes**.
    - En estos casos, el número de sinónimos a cada valor es similar y se consigue repartir en pequeños trozos similares todo el conjunto de claves posibles.

De ahí el uso del término inglés *hashing* (***desmenuzar***)

Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna

Función de truncamiento

- Se ignora **parte** de la clave y se utiliza la parte restante directamente como índice.
- <u>**Ejemplo**</u>:
  - Supongamos que las claves son enteros de **8** dígitos (como el número de DNI) y que la tabla de transformación o tabla ***hash*** tiene **1000** posiciones.
  - Podríamos utilizar el quinto, segundo y primer dígito (desde la derecha) para hacer la conversión.
  - Por ejemplo, el elemento con clave **72.588.495** iría en la posición **895**.
- Este método es rápido pero no distribuye las claves de forma uniforme.

Función módulo de truncamiento

- <u>Ejemplo</u>:
  - Supongamos que para los **300** alumnos del curso decidimos utilizar los **tres últimos dígitos** del DNI para obtener la dirección de memoria.
  - El conjunto de **valores posibles** son los números de hasta **8** cifras y la función de dispersión proporciona la **dirección** como un número de **0** a **999**.
  - La aplicación con los números del DNI podría dar los siguientes <u>resultados</u>:

Valor	42325431	35623456	23452463	33040132	13512532	29454362	23865343	4797976
Dirección	431	456	463	132	532	362	343	976

Función módulo

- La función de dispersión del ejemplo anterior se establece por la siguiente ***fórmula***:

$$\boxed{\boldsymbol{h(k) = k} \ \mathbf{mod\ 1000}}$$

- Si todos los números del DNI desde el **0** hasta el **99.999.999** son ***igualmente probables***, la probabilidad de **cada dirección de memoria** es de **0.001**

- La **operación módulo** (que proporciona el resto de la división entre enteros) es muy ***frecuente*** en el diseño de funciones de dispersión.

Función módulo

Tamaño de la tabla:

- Para usar funciones de dispersión de este tipo hay que analizar la elección del tamaño ***t*** , ya que se puede tener un gran ***desequilibrio***.
- Esto ocurre, por ejemplo, si las claves son **frecuentemente** números acabados en **0** y se toma ***t*** **= 10**.
- Algo parecido ocurre si las claves son cadenas de caracteres de palabras de un idioma concreto que tienen ***inicios*** o ***terminaciones*** muy frecuentes (por ejemplo, los apellidos en castellano).
- Este fenómeno se presenta con mayor probabilidad cuando el tamaño de la tabla es un **número entero con muchos divisores**.
- Por tanto, una condición previa que evita ésta y otras dificultades es tomar como tamaño *t* un **número primo**, o con pocos divisores.

Función módulo

- El resultado de la operación n **mod** m (que se implementa **en C** usando el operador **%**) es el resto entero de la división del número entero n entre m, que también tiene que ser entero: $\boxed{n \ \mathbf{mod}\ m = n \ \%\ m}$
- En general, en las tablas de dispersión, se utiliza como valor de referencia de los elementos de la tabla una información numérica de un rango de variación mucho mayor que el tamaño de la tabla de dispersión a utilizar.
  - Una vez fijado el **tamaño *t* de la tabla**, la función de dispersión se aplica calculando el **resto** de dividir la clave k por t para obtener la dirección de memoria correspondiente.
  - La función de dispersión $h(k) = k \ \mathbf{mod}\ t$ da un valor entre **0** y t**–1** para la dirección de memoria.
  - Si el espacio de direcciones de la tabla es de la forma **1**, **2**, ..., t hay que usar la función de dispersión: $h(k) = 1 + (k - 1) \ \mathbf{mod}\ t$.

Función módulo

- Si el tamaño de la tabla puede alcanzar las 1000 direcciones se elige 997 como valor del tamaño ***t*** (número primo más cercano a 999).

- Tomando los números de DNI del ejemplo anterior:

$$\boldsymbol{h}(42325431) = 42325431 \;\%\; 997 = 787$$
$$\boldsymbol{h}(35623456) = 35623456 \;\%\; 997 = 646$$
$$\boldsymbol{h}(23452463) = 23452463 \;\%\; 997 = 32$$
$$\boldsymbol{h}(33040132) = 33040132 \;\%\; 997 = 549$$
$$\boldsymbol{h}(13512532) = 13512532 \;\%\; 997 = 191$$
$$\boldsymbol{h}(29454362) = 29454362 \;\%\; 997 = 988$$
$$\boldsymbol{h}(23865343) = 23865343 \;\%\; 997 = 154$$
$$\boldsymbol{h}(41797876) = 41797876 \;\%\; 997 = 645$$

Función suma

- Es muy frecuente que las claves estén formadas por cadenas de caracteres, como nombres o identificadores.
    - Una *aplicación* muy importante de las tablas de dispersión es el **almacenamiento** que realizan los ***compiladores*** de los identificadores y la información relativa.
    - Este almacenamiento debe ser eficiente para cualquier criterio de selección de ***identificadores*** de un estilo de programación.
- Una opción inmediata para estos casos es usar los valores ASCII de los caracteres para obtener una cantidad numérica, por ejemplo sumándolos, a la que se le aplica una operación aritmética.

Función suma (código)

```
int hash(int valor) {
  d = 0 ;
  x = valor;
  while (x > 0){
      y = x % 10;
      d = d + y;
      x = x / 10;
  }
  return (d % t);
}
```

Función suma (ejemplo)

- Aplicando la función hash de tipo **suma** a los números del **DNI** con una tabla de tamaño **12** se podría obtener, por ejemplo, lo siguiente:

Valor	42325431	35623456	23452463	33040132	13512532	29454362	23865343	4797976
Suma	24	34	29	16	22	37	34	49
Dirección	0	10	5	4	10	1	10	1

- Se observan en esta tabla diversos ***sinónimos*** correspondientes a las direcciones **10** y **1**; al considerar estas claves se producirán colisiones.
- Parece que el ejemplo está forzado para que aparezcan muchas colisiones (en este caso, **3** colisiones con **8** registros)
- Sin embargo, las colisiones suelen ser ***más frecuentes*** de lo que cualquiera suele esperar.

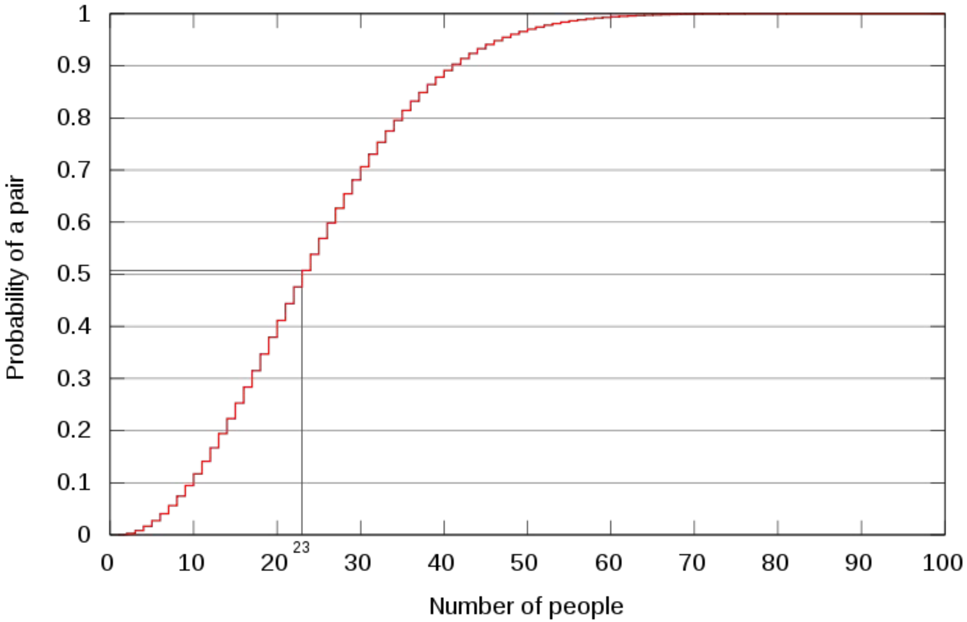
Paradoja de von Mises

- Si disponemos de una ***función de dispersión*** para una tabla de tamaño **365** en la que almacenar **23** elementos, podemos relacionar la aparición de colisiones con el hecho de que entre **23** personas elegidas al azar encontremos dos con la misma fecha de ***cumpleaños***.
  - ¿Cual es la probabilidad de que encontremos al menos dos alumnos en la clase con la misma fecha de cumpleaños?
  - La fórmula es:

$$\text{Prob(mismo cumpleaños)} = 1 - \frac{\binom{365}{alumnos}}{365^{alumnos}} = \frac{365 \cdot 364 \cdot 363 \cdot \ldots \cdot (365 - alumnos - 1)}{365 \cdot 365 \cdot 365 \cdot \ldots (alumnos) \ldots \cdot 365}$$

Paradoja de von Mises

- Con **23** alumnos la probabilidad es más del **50%** (**0.507297**), aunque 23 fechas a lo largo del año suponga poco más del **6%** del calendario ocupado
- Con **50** alumnos la probabilidad es más del **97%** (**0.970373**)
- Con **75** alumnos es del **99,97%** y casi **100%** con 100 personas (**0.9999997**)



Función de plegado

- Otro tipo de transformación corriente para claves consistentes en secuencias de caracteres es el de las transformaciones denominadas de plegado.
- La idea consiste en dividir la clave en partes de tamaño similar y sumarlas.
  - Esta suma se puede hacer **directamente** o uniendo los **extremos consecutivos** con un efecto acordeón.
- Resultan los métodos de:
  - Plegado **por desplazamiento**
  - Plegado **por las fronteras**
- Las operaciones pueden hacerse en base decimal o binaria.

Función de plegado

- Con los DNI del ejemplo anterior para obtener una dirección del rango **0..999** se realizan los **plegados** con partes de tres cifras; *si los plegados superan el rango de la tabla se toma el módulo del tamaño de la tabla.*
- Plegado **por desplazamiento**:

$$h(42|325|431) = 42 + 325 + 431 = 798$$
$$h(35|623|456) = 35 + 623 + 456 = 1114 \rightarrow 114$$
$$h(23|452|463) = 23 + 452 + 463 = 938$$
$$h(33|040|132) = 33 + 040 + 132 = 205$$

- Plegado **por las fronteras**:

$$h(42|325|431) = 42 + 523 + 431 = 996$$
$$h(35|623|456) = 35 + 326 + 456 = 817$$
$$h(23|452|463) = 23 + 254 + 463 = 740$$
$$h(33|040|132) = 33 + 040 + 132 = 205$$

Función del centro del cuadrado

- Otra función de dispersión común para claves de tipo numérico consiste en tomar unos dígitos fijos del interior del cuadrado del número entero.

Ejemplo:

- Tomando los DNIs del ejemplo anterior para obtener una dirección del rango **0..999** se toman las cifras tercera, cuarta y quinta del cuadrado de la clave.

$$42325431 \rightarrow 42325431^2 = 1791442109335761 \rightarrow 914$$
$$35623456 \rightarrow 35623456^2 = 1269030617383936 \rightarrow 690$$
$$23452463 \rightarrow 23452463^2 = 550018020766369 \rightarrow 1$$
$$33040132 \rightarrow 33040132^2 = 1091650322577424 \rightarrow 916$$

Números pseudo-aleatorios

- Un ***tipo de funciones de dispersión*** con buenas propiedades son las funciones de dispersión o funciones hash pseudo-aleatorias que utilizan las técnicas de los **generadores de números pseudo-aleatorios**.
- Estos generadores se basan en una fórmula que aprovecha el ***rebosamiento*** en los cómputos en el ordenador.
- Las **fórmulas usuales** que aplican los generadores de números pseudo-aleatorios de los lenguajes de programación estándares son fórmulas recursivas del tipo:

$$\textit{semilla} = ((\textit{multiplicador} * \textit{semilla}) + \textit{sumando}\ ) \bmod \textit{máximo}$$

Generación de números pseudo-aleatorios

- En **C++** se dispone de la función **rand** con un mecanismo basado en la fórmula anterior pero que mejora su rendimiento y que proporciona un **número entero** entre **0** y **RAND_MAX**.
- El valor de **RAND_MAX** viene definido en la librería <cstdlib> y suele ser **32767** o mayor.
- Para obtener un número real pseudo-aleatorio entre 0 y 1 con ***distribución uniforme*** se usa la fórmula:

$$\boxed{r = \text{rand}() \,/\, maximo}$$

- Para obtener un número **entero** entre 0 y n–1 con función de probabilidad uniforme se usa:

$$\boxed{x = \text{rand}() \,\%\, n}$$

Inicializar la semilla

- La semilla se puede ***iniciar*** en un valor concreto en la ejecución de un programa mediante una sentencia:

srand(semilla)

- Para asignar a la semilla un valor usando el estado del reloj del sistema se usa

srand(time(NULL))

- Esta sentencia que hace que los valores que proporciona **rand()** sean ***prácticamente irrepetibles***.
- Si siempre se proporciona el mismo valor semilla, se obtendrá siempre la misma secuencia de números.

Función pseudo-aleatoria

- Para construir una función hash o de dispersión pseudo-aleatoria se hace *intervenir la clave en la asignación de un valor a la semilla* y luego se toma el módulo del tamaño de la tabla para llevarlo a su rango.
- Si las claves de los elementos a insertar en la tabla son identificadores se deben obtener valores numéricos a partir de sus caracteres.
  - Por ejemplo, se pueden transformar en enteros sumando los valores ASCII de algunas de sus letras (o de todas).
- Entonces esta información numérica se utiliza para establecer el valor de la semilla.

Código de la función pseudo-aleatoria

- Una implementación de la función hash o función de dispersión pseudo aleatoria es:

```
int hash(int valor) {
    srand(valor);
    return rand() % t;
}
```

- Existen diversas ***recomendaciones*** sobre valores adecuados para la semilla de los generadores de números pseudo-aleatorios y, aunque hay ciertas discrepancias, todas coinciden en que **no** es conveniente usar ***semillas pares***.
  - Por eso, las funciones de dispersión pseudo-aleatorias determinan una semilla impar a partir de la clave.

Implementación de las tablas

- El ***tipo de datos* abstracto** tabla de dispersión o **tabla hash** constituye uno de los tipos de datos de mayor aplicación.
- Este tipo de datos, denominado **tabla**, está compuesto de un número de pares consistentes en:
  - Una **clave,** y
  - Un **enlace a un registro** con cierta información asociada a la clave.
- La estructura de datos de la tabla hash está compuesta por un número fijo de **celdas** que pueden estar ***vacías*** o contener ***uno*** o ***varios*** pares.

Sinónimos y colisiones

- A las celdas se accede mediante una dirección de memoria que resulta de aplicar una función hash, de transformación o de dispersión ***h*()** a la clave:

$$\boxed{h: k \rightarrow h(k)}$$

- Dos claves distintas ***x*** e ***y*** son ***sinónimas*** para la transformación si: $\boxed{h(x) = h(y)}$
- Se produce una ***colisión*** cuando aparecen dos claves distintas a las que le corresponde la misma dirección.
  - Siempre existe la posibilidad de que aparezcan sinónimos y, por tanto, de que se produzcan colisiones.
  - Por tanto, hay que disponer de herramientas para tratar las colisiones de forma inteligente.

Operaciones en las tablas

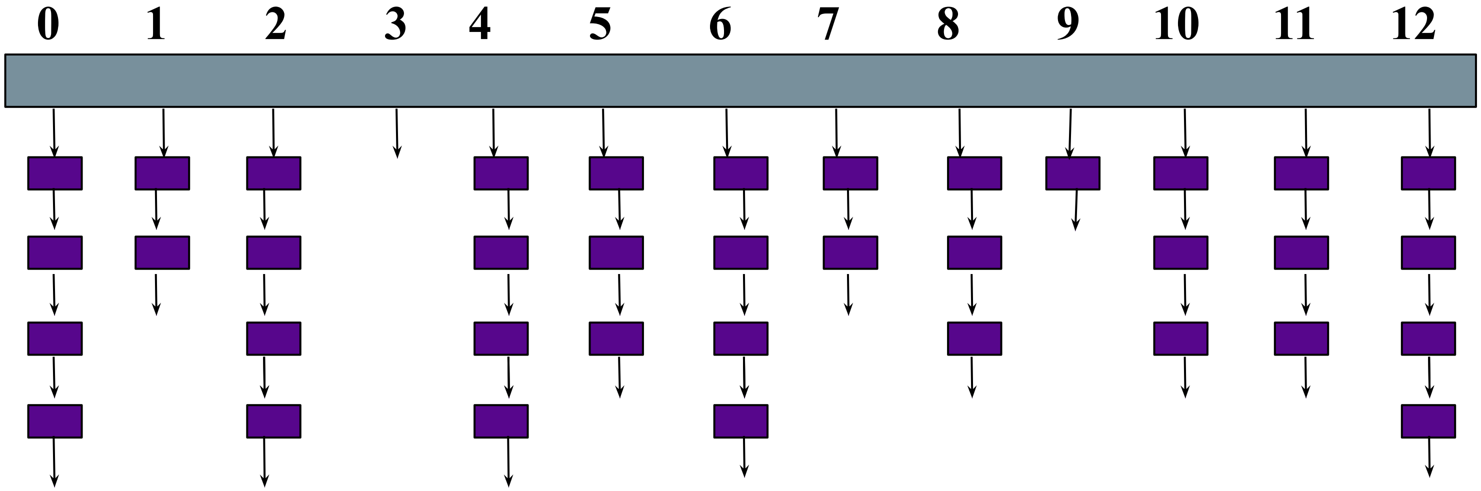
- Sobre las tablas de dispersión se consideran las **operaciones básicas** siguientes:
  - La **búsqueda** es la operación más relevante y el objetivo fundamental de esta herramienta es hacerla muy eficiente.
  - La **inserción** debe diseñarse de forma que favorezca lo más posible este objetivo.
  - La operación de **eliminación** o borrado.
- Las tablas de dispersión son ineficientes para **otras operaciones** que requieran información sobre alguna ordenación de los datos:
  - Por ejemplo, para obtener un listado de los registros ordenados por su clave.

Dispersión abierta o cerrada

- Las **celdas de la tabla** pueden tener capacidad para almacenar:
  - un registro con su clave,
  - un número fijo de ellos, o
  - una cantidad arbitraria.
- Si en cada celda se tiene espacio para una **cantidad fija** (uno o más) de registros se habla de bloque.
  - Dispersión (*hashing*) cerrada o direccionamiento abierto.
- Para disponer de espacio para una cantidad **arbitraria** de registros se usan listas enlazadas mediante memoria dinámica:
  - Dispersión (*hashing*) abierta, direccionamiento cerrado o encadenamiento separado.

Dispersión abierta

- La **dispersión abierta** consiste en usar ***una lista*** con los elementos que se dispersan en ***cada dirección***.
  - Cada celda de la tabla está compuesta por una clave y un puntero.
  - El puntero de una celda apunta a NULL si la celda está vacía.



Dispersión cerrada

- En la **<u>dispersión cerrada</u>** en cada dirección hay una celda que tiene espacio para una cantidad fija de registros (***bloque***).
- Se produce un ***desbordamiento*** en estas tablas al tratar de insertar un nuevo registro con una clave a cuya dirección de memoria le corresponde una celda que <u>ya está completa</u>.
  - *<u>El desbordamiento no se produce si se usa dispersión abierta</u>*
- Si las celdas sólo tienen capacidad para un registro, que es la implementación más sencilla, la colisión y el desbordamiento se producen <u>a la vez</u>.
  - En este caso se habla de tablas **simples**, o con celdas simples.

Tratamiento del desbordamiento

- El **tratamiento del desbordamiento** implica utilizar una estrategia para determinar un lugar alternativo donde ubicar la información correspondiente y luego poder acceder a ella ***eficientemente***.
- En la dispersión cerrada, si al tratar de insertar un elemento, la celda que le corresponde está llena (*hay desbordamiento*) se puede tratar de almacenar el elemento en otra celda con algún registro libre.
    - Por ello, se habla de ***direccionamiento abierto***.

La exploración

- El proceso para determinar qué otra celda se examina para tratar de insertar en ella el elemento se llama **exploración**.
- La estrategia de exploración debe estar completamente determinada a partir de la clave para que el posterior proceso de búsqueda pueda encontrar eficientemente el mismo registro dada su clave.
- La estrategia de exploración elemental es la **exploración lineal**:
    - Consiste en probar en las celdas que vienen a continuación hasta encontrar una celda con un registro libre.

Ejemplo de exploración lineal

- <u>**Ejemplo**</u>:
  Para los **200** alumnos del curso utilizamos los dos últimos dígitos del DNI.
  - El conjunto de claves posibles son los números de **8** cifras y la función de dispersión es la que toma los dos últimos dígitos:

$$h(x) = x \bmod 100$$

- Si la tabla tiene **100** celdas de **3** registros, el estado de la tabla puede ser:

00	01	...	32	33	34	...	99
41325200	29045301	...	41376732	30128533	37231234	...	32176499
43124600	28879800	...	38937332	-	45320434	...	-
42437300	37566201	...	42732232	-	-	...	-

Ejemplo de exploración lineal

00	01	...	32	33	34	...	99
41325200	29045301	...	41376732	30128533	37231234	...	32176499
43124600	28879800	...	38937332	-	45320434	...	-
42437300	37566201	...	42732232	-	-	...	-

- Las claves **37231234** y **45320434** son sinónimos y produjeron una colisión.
  - Sin embargo, han podido almacenarse en la dirección que les corresponde.
  - La clave **34286434** daría lugar a otra colisión pero no a un desbordamiento.
- Las claves **41376432**, **38937332** y **42732332** son sinónimos y sus registros ocupan completamente la celda correspondiente

Ejemplo de exploración lineal

00	01	...	32	33	34	...	99
41325200	29045301	...	41376732	30128533	37231234	...	32176499
43124600	28879800	...	38937332	-	45320434	...	-
42437300	37566201	...	42732232	-	-	...	-

- Si se tienen que almacenar los datos correspondientes a la clave **42347532** se produce una colisión ya que la celda está ocupada, pero como además está completa se produce un desbordamiento.
- Utilizaremos exploración lineal, para ubicar la clave en la siguiente celda que tenga hueco para ello.
  - El registro de clave **42347532** se almacena en la dirección **33**.

Ejemplo de exploración lineal

00	01	...	32	33	34	...	99
41325200	29045301	...	41376732	30128533	37231234	...	32176499
43124600	28879800	...	38937332	-	45320434	...	-
42437300	37566201	...	42732232	-	-	...	-

Seguramente esto fue lo que ocurrió en las celdas **00** y **01**:

- Cuando se insertó el registro de clave **28879800** ya estaba llena la celda de dirección **00**.
- Posteriormente se insertó el registro de clave **37566201**.

- Si ahora se tiene que insertar un elemento con clave **24874700** llevaría a buscarle ubicación en una celda más alejada.

Celdas vacías

- Por otro lado, es necesario **conocer cuándo el espacio para un registro en una celda está vacío** y no confundir el valor que pueda estar presente previamente en la posición correspondiente a la clave con una clave de un elemento ya insertado.
- Un valor especial de la clave (usualmente **0** o **–1**) indicará que el registro está vacío.
  - Si, por ejemplo, este valor es el cero entonces el cero no puede ser un valor posible de la clave.
- La tabla vacía debe ***inicializarse*** con este valor en todos sus registros y no dejar en ellos los valores que pudieran haber al iniciar el programa.

La búsqueda

- La **búsqueda del registro** correspondiente a un valor de la clave se efectúa de la siguiente manera:
  - Se empieza por la dirección que le corresponde según la función de dispersión que se esté aplicando.
  - Si el valor **no se encuentra** en ningún registro de dicha celda y queda aún espacio para almacenar el registro, la búsqueda finaliza en fracaso y se puede concluir que el elemento no está presente en la tabla.
  - Si el valor **no se encuentra** en ningún registro de dicha celda y la celda está completamente ocupada, se debe continuar la búsqueda por las direcciones en la que se habría almacenado siguiendo la estrategia de direccionamiento o exploración adoptada.

La inserción

- Por ejemplo, si la tabla se gestiona con exploración lineal se recorren las celdas que vienen a continuación
  hasta encontrar el elemento buscado o
  una celda con espacio para el registro.
  - La tabla se considera que es circular por lo que si se llega al final de la misma se vuelve a empezar por el principio.
- Sólo si la **exploración acaba en una celda con espacio libre** se podría concluir que el elemento buscado no está presente en la tabla.
- Para **insertar** un nuevo elemento en la tabla, este proceso de búsqueda acaba en la celda con espacio disponible, en la que se puede insertar el registro.

La eliminación

- Si se elimina algún elemento de la tabla se puede falsear el proceso de búsqueda de aquellos elementos para los que se exploró la celda de ese elemento al buscarle una posición en la que ubicarlo.

  Por ejemplo, si:

  - al insertar en una tabla simple el elemento **A** se produce una colisión con el elemento **B** que ya está en la tabla y
  - ***posteriormente*** se elimina el elemento **B**,
  - cuando se vaya a buscar el elemento **A** y nos encontremos que la celda que le corresponde hay espacio libre para el registro,
  - entonces interpretaríamos **erróneamente** que **A** no está en la tabla.

Tablas contaminadas

- Como remedio se pueden marcar las celdas de las que se ha eliminado algún elemento para que la exploración no se detenga en ellas.
- Para ello se utiliza un valor de clave especial pero distinto al que corresponde a una celda vacía.
  - Por ejemplo:
    - El **0** (cero) para la celda vacía, y
    - el **–1** para la que estuvo ocupada y vuelve a estar libre.
- La presencia de celdas que estuvieron ocupadas provoca un esfuerzo adicional en los procesos de búsqueda que se vuelven más ineficientes y se dice que la **tabla** está **contaminada**.

Reorganización de tablas

- Si el <u>grado de contaminación de una tabla es alto</u> llega a ser conveniente <u>reorganizar de nuevo la tabla</u> aplicando la estrategia de inserción a los elementos que forman parte de ella a la tabla vacía.
- Incluso se puede usar información relativa a la frecuencia con la que se accede a determinados elementos para <u>que sean los menos frecuentes los que se vean desplazados al producirse las colisiones</u>.
- Existen estrategias de reorganización que persiguen <u>dinámicamente</u> este objetivo.

Técnicas de exploración

- El tratamiento adecuado del desbordamiento requiere usar una ***estrategia inteligente*** para determinar la celda alternativa donde ubicar el registro.
- Estas estrategias se denominan **técnicas de exploración** y una de las más sencillas es la exploración lineal.
- Con la **exploración lineal**:
  - Cuando se produce un desbordamiento se utiliza la siguiente celda con algún hueco.
  - En la inserción, si la celda está llena se pasa a la siguiente hasta encontrar una con hueco.
  - En la búsqueda, si no se encuentra el registro en la celda, se pasa a la siguiente hasta encontrarlo o alcanzar una celda que no esté llena.
  - Al llegar al final de la tabla, se trata como una tabla circular y se reinicia la búsqueda por el principio.

Función de exploración

- Si en una tabla $\boldsymbol{T}$ la función de dispersión es $\boldsymbol{h}$, para una clave $\boldsymbol{k}$, se examinan las celdas: $\boldsymbol{T[h(k)]}$, $\boldsymbol{T[h(k)+1]}$, $\boldsymbol{T[h(k)+2]}$, ..., $\boldsymbol{T[h(k)+i]}$, ...
- En general, la **<u>dirección de exploración</u>** se expresa por:

$$\boxed{\boldsymbol{d = h(k) + g_i(k)}}$$

donde $\boldsymbol{h(k)}$ es la **función de dispersión** y $\boldsymbol{g_i(k)}$ es la **función de exploración** o función de tratamiento de colisiones.

- Al considerar la tabla siempre circular la dirección de exploración es:

$$\boxed{\boldsymbol{d = (h(k) + g_i(k)) \bmod t}}$$

donde $\boldsymbol{t}$ es el tamaño de la tabla.

- En la exploración lineal, las direcciones de búsqueda vienen dadas por:

$$\boxed{\boldsymbol{d = (h(k) + i) \bmod t}, \text{ para } \boldsymbol{i = 0, 1, 2, ..., t\!-\!1}.}$$

Departamento de Ingeniería Informática y de Sistemas
Universidad de La Laguna

Clustering

- Pero con el método de exploración lineal se van llenando celdas muy cercanas.
- Este fenómeno se denomina **clustering** y provoca ineficiencia en las operaciones de búsqueda posteriores.
- Esta dificultad se evita en parte con la estrategia de **exploración cuadrática** que utiliza la función:

$$\boldsymbol{g_i(k) = i^2}, \text{ para } i = 0, 1, 2, \ldots$$

- Si el tamaño de la tabla toma un valor de la forma $\boldsymbol{t = 4j + 3}$ entonces la versión de la exploración cuadrática que recorre las direcciones $\boldsymbol{h(k)}$, $\boldsymbol{(h(k) + i^2) \bmod t}$, $\boldsymbol{(h(k) - i^2) \bmod t}$, incluye todas sus direcciones si $\boldsymbol{1 <= i <= (t-1)/2}$.
- Con esta estrategia se puede llegar a producir un fenómeno similar al clustering, denominado **clustering secundario**.

Técnicas de exploración

- Otra forma de resolver los desbordamientos es utilizando permutaciones del rango de direcciones de la tabla asociadas a cada valor de la clave; es decir de la secuencia **[0 . . *t* – 1]**.
- La **técnica de exploración por permutación** consiste en obtener a partir de la clave ***k*** una permutación ***p*(*k*)** de las direcciones de la tabla en vez de una sola dirección.
- La estrategia de permutación pseudo-aleatoria consiste en generar una permutación ***p*(*k*)** a partir de la clave utilizando números pseudo-aleatorios.
- Una forma sencilla de obtener permutaciones es mediante los recorridos sistemáticos de paso aleatorio.
  - Es la técnica conocida como **dispersión doble**.

Dispersión doble

- En la **técnica de dispersión doble**, es una estrategia de exploración en la que, ante una colisión,
  - se genera una amplitud de paso por medio de ***otra*** función de dispersión $f(k)$ y
  - se realiza una exploración del resto de la tabla con dicha **amplitud de paso**:

$$g_i(k) = i * f(k), \text{ para } i = 0, 1, 2, \ldots,$$

- Con la estrategia de exploración doble con las dos funciones de dispersión $h(k)$ y $f(k)$ se recorren las direcciones:

$$d = h(k) + i * f(k) \bmod t, \text{ para } i = 0, 1, 2, \ldots,$$

Tamaño primo de las tablas

- Sin embargo, de esta forma no siempre se obtiene una permutación de todas las direcciones de la tabla:
  - Si el paso ***f*(*k*)** resulta un divisor del tamaño *t* de la tabla, se repiten direcciones antes de recorrerlas todas.
  - Por tanto, si el número *t* es primo estos recorridos no repiten ninguna dirección antes de haberlas recorrido todas.

- Para la exploración doble es **necesario** elegir como tamaño de la tabla **un número primo** y así garantizar que no se repite ninguna celda ninguna celda antes de explorarlas todas dando lugar a un recorrido sobre una permutación de las direcciones de la tabla

La redispersión

- La **técnica de redispersión** o *rehashing* usa una familia de funciones de exploración g_i, $i = 0, 1, 2, \ldots$, para examinar las direcciones, siendo cada g_i , una ***función de dispersión distinta***, para cada iteración i.
- La estrategia de **redispersión pseudo-aleatoria** consiste en usar una sucesión de funciones de dispersión pseudo-aleatorias o, equivalentemente, una sucesión de ***direcciones pseudo-aleatorias*** que dependan sólo de la clave.
- Una estrategia de exploración por ***permutación pseudo-aleatoria*** se obtendría seleccionando aleatoriamente, en cada iteración $i = 0, 1, 2, \ldots, t-1$, una dirección de memoria *aún no explorada*.

Eficiencia

- La eficiencia del tratamiento de la tabla depende de la carga que soporte la tabla en cada momento.
- Se llama **densidad de carga** al cociente entre el número de direcciones que se están utilizando y el número de direcciones posibles:

$$\textbf{Densidad} = \frac{\textbf{número de direcciones que se están utilizando}}{\textbf{número de direcciones posibles}}$$

- Se llama **factor de carga** al cociente entre el número de registros almacenados y el número de registros que es posible almacenar en la tabla:

$$\textbf{Factor} = \frac{\textbf{número de registros almacenados}}{\textbf{número de registros que es posible almacenar}}$$

Ejemplo

- **Ejemplo:** Supongamos que para los **200** alumnos del curso decidimos utilizar los dos últimos dígitos del DNI.
  - El conjunto de claves posibles son los números de **8** cifras y la función de dispersión se define de forma que los dos últimos dígitos asignan la dirección de memoria.
- La tabla consta de **100** celdas de tres registros.

00	01	...	32	33	34	...	99
41325200	**29045301**	**...**	**41376732**	**30128533**	**37231234**	**...**	**32176499**
43124600	**28879801**	**...**	**38937332**	**-**	**45320434**	**...**	**-**
42437300	**…**	**...**	**42732232**	**-**	**-**	**...**	**-**

Ejemplo

- Si hemos insertado **120** registros usando [redacted] celdas.

  La <u>densidad de carga</u> es: **70 / 100 = 0.7**

  El <u>factor de carga</u> es: **120 / 300 = 0.4**

00	01	...	32	33	34	...	99
41325200	**29045301**	**...**	**41376732**	**30128533**	**37231234**	**...**	**32176499**
43124600	**28879801**	**...**	**38937332**	**-**	**45320434**	**...**	**-**
42437300	**…**	**...**	**42732232**	**-**	**-**	**...**	**-**

- Si al acabar con los **200** alumnos tuviéramos **80** celdas ocupadas, estos valores serían de **0.8** y **2/3**.

Probabilidad de celda vacía

- Suponemos que la asignación de direcciones a las celdas es aleatoria y uniforme. Al insertar una nueva clave ***k*** en una tabla de tamaño ***n***, (con distribuciones de los intentos independientes).
  - La probabilidad de encontrar la celda vacía en la primera comparación es:

$$\boxed{1-\frac{k}{n}=\frac{(n-k)}{n}}$$

  - La probabilidad de que sean necesarias exactamente dos comparaciones es:

$$\boxed{\frac{k}{n}\left(1-\frac{k-1}{n-1}\right)=\frac{k}{n}\frac{(n-k)}{n-1}}$$

  - La probabilidad de encontrar la celda vacía a la tercera comparación es:

$$\boxed{\frac{k}{n}\frac{k-1}{n-1}\left(1-\frac{k-2}{n-2}\right)=\frac{k}{n}\frac{k-1}{n-1}\frac{(n-k)}{n-2}}$$

Número medio de comparaciones

- Si *c* el número de comparaciones hasta encontrar celda vacía:

$$\Pr(c=i)=\frac{k}{n}\,\frac{(k-1)}{(n-1)}\,\frac{(k-2)}{(n-2)}\cdots\frac{(k-i+2)}{(n-i+1)}\,\frac{(n-k)}{(n-i)}.$$

- Entonces:

$$E[c]=\sum_{i=1}^{n} i\Pr(c=i)=\sum_{i=1}^{n}\Pr(c\geq i)=\frac{(n+1)}{(n-k+1)}=\frac{1}{\left(1-\dfrac{k}{n+1}\right)}\approx\frac{1}{(1-\alpha)}.$$

- Por tanto, si la clave ya se encuentra almacenada en la tabla, el número medio de comparaciones hasta encontrarle hueco es: $\boxed{1/(1-\alpha)}$
- Con aproximaciones similares se llega a que el número esperado de comparaciones hasta encontrar una clave ya almacenada en la tabla es:

$$\boxed{E[c] = -1/\alpha \log(1-\alpha)}$$

Distribución aleatoria y uniforme

En la hipótesis de distribución aleatoria y uniforme de las claves en la tabla, con independencia en las distribuciones de los sucesivos intentos.

	Dispersión abierta	Exploración Lineal	Exploración Aleatoria
Búsqueda	$1 + \alpha / 2$	$1/2 (1 + 1 / (1 - \alpha))$	$- 1/\alpha \log (1 - \alpha)$
Inserción	α	$1/2 (1 + 1 / (1 - \alpha)^2)$	$1 / (1 - \alpha)$

Número medio de comparaciones

Con tablas de tamaño 1000 los valores aproximados son:

BÚSQUEDA	α = 0.10	α = 0.25	α = 0.50	α = 0.75	α = 0.90	α = 0.99
Abierta	1.05	1.12	1.25	1.37	1.45	1.49
Lineal	1.06	1.17	1.50	2.50	5.50	50.5
Redispersión	1.05	1.15	1.39	1.85	2.56	4.65

INSERCIÓN	α = 0.10	α = 0.25	α = 0.50	α = 0.75	α = 0.90	α = 0.99
Abierta	0.10	0.25	0.50	0.75	0.90	0.99
Lineal	1.12	1.39	2.50	8.50	50.5	5000.0
Redispersión	1.11	1.33	2.00	4.00	10.0	100.0

Universidad de La Laguna