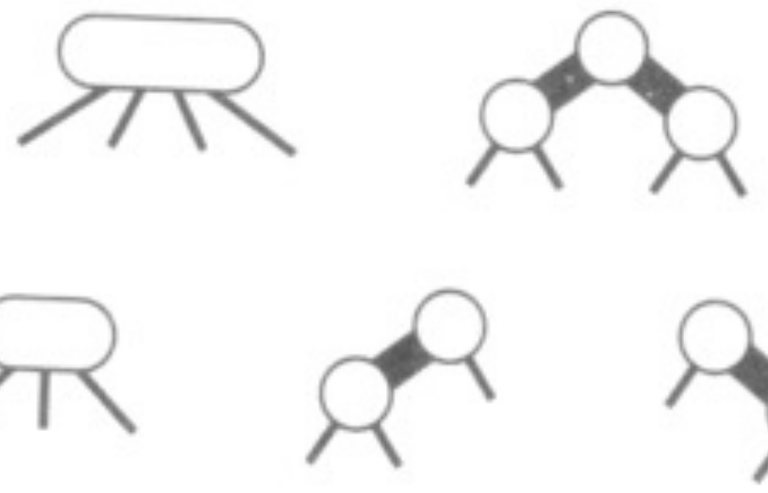
Arboles balanceados 2 (caso particular: rojo negros) y Tablas Hash.

mat-151

Árboles rojo-negros

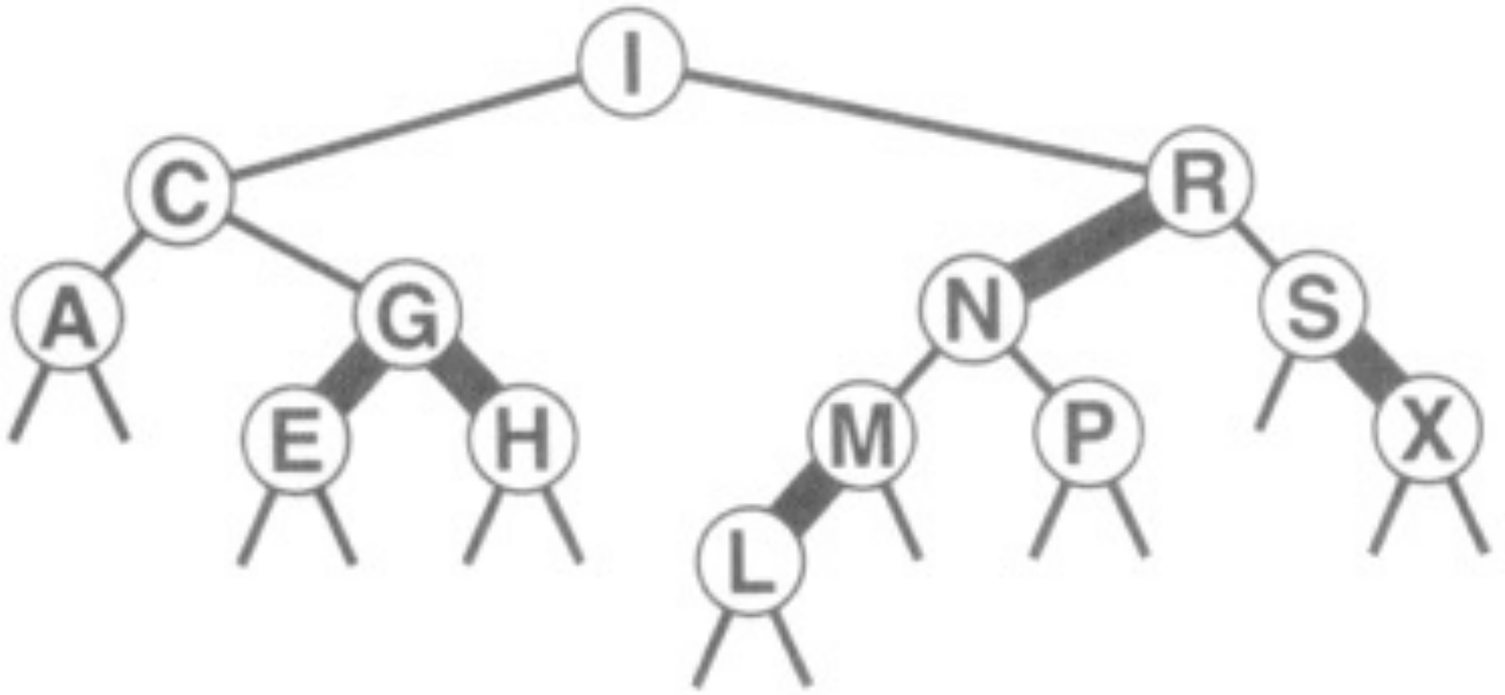
- Estos están basados en los árboles 2-3-4.

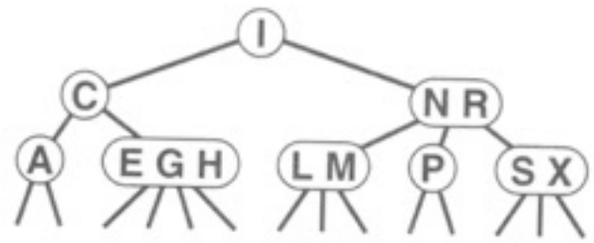


El color del nodo es el color del link que llega a él.

De tal forma que las conexiones internas son Rojas (anchas) y las conexiones externas son negras (delgadas).

Árboles rojo-negros





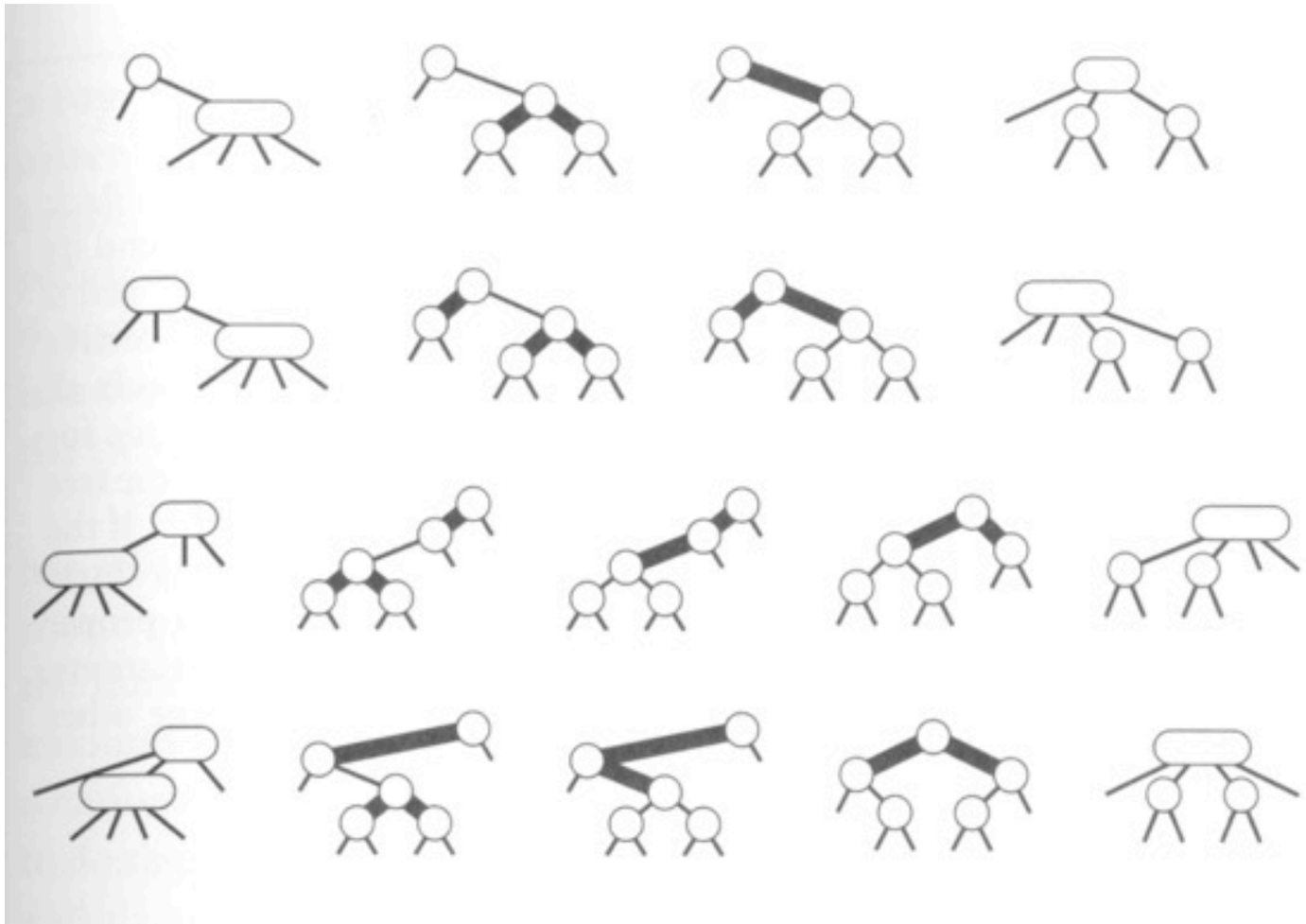
Arbol rojo-negro, (i) mismo que el anterior.

(ii)La búsqueda es la misma que en un BST.

Cualquier camino de raiz-hoja tiene 3 conexones negras.

Árboles rojo-negros

- Divsión de los 4-node en un árbol rojo negro.



(Si un nodo tiene 2 hijos rojos es un 4-node)

Las operaciones comprenden el cambiar de color los nodos(ligas) y hacer 1 o 2 rotaciones.

Árboles rojo-negros

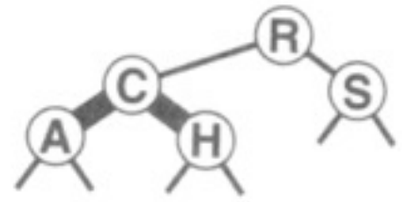
- El codigo de inserción en un arbol rojo-negro. El nodo que se va a insertar **se crea rojo por default** en el constructor `node(x)`.

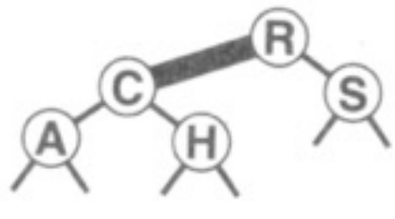
Tarea, explicar este código, línea a línea

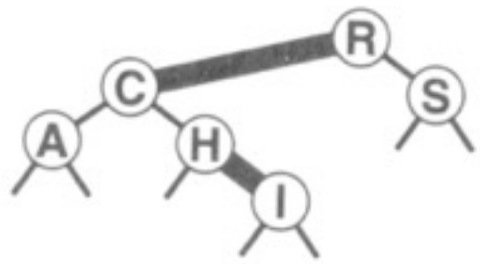
```
private:
  int red(link x)
    { if (x == 0) return 0; return x->red; }
  void RBinsert(link& h, Item x, int sw)
    {
      if (h == 0) { h = new node(x); return; }
      if (red(h->l) && red(h->r))
      { h->red = 1; h->l->red = 0; h->r->red = 0; }
      if (x.key() < h->item.key())
        {
          RBinsert(h->l, x, 0);
          if (red(h) && red(h->l) && sw) rotR(h);
          if (red(h->l) && red(h->l->l))
            { rotR(h); h->red = 0; h->r->red = 1; }
        }
      else
        {
          RBinsert(h->r, x, 1);
          if (red(h) && red(h->r) && !sw) rotL(h);
          if (red(h->r) && red(h->r->r))
            { rotL(h); h->red = 0; h->l->red = 1; }
        }
    }
public:
  void insert(Item x)
    { RBinsert(head, x, 0); head->red = 0; }
```

Árboles rojo-negros

- Ejemplo de la inserción de I

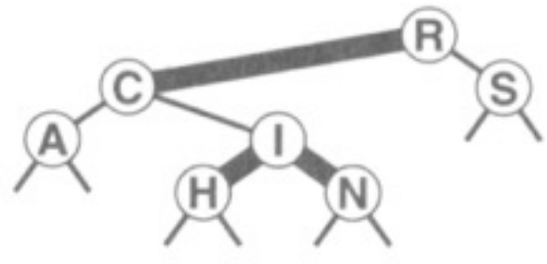




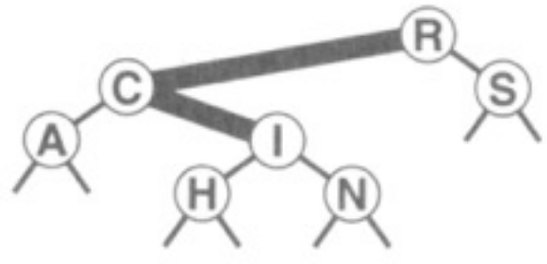


Se divide el 4-node que tiene C con un cambio de color y agregando el nuevo nodo al final, convirtiendo donde está H de un 2-node a un 3-node.

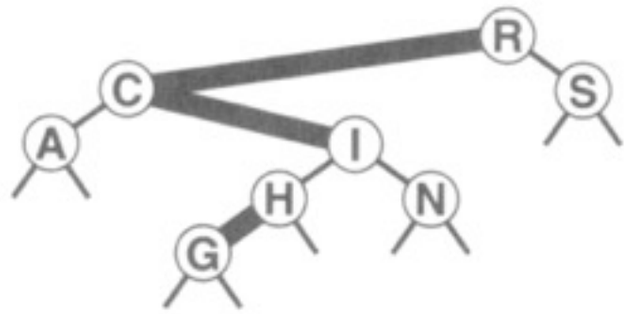
Árboles rojo-negros



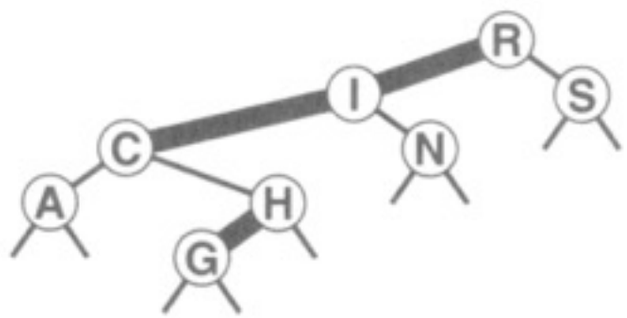
Resultado de insertar G.



-Se divide el 4-node que tiene I con un cambio de color,

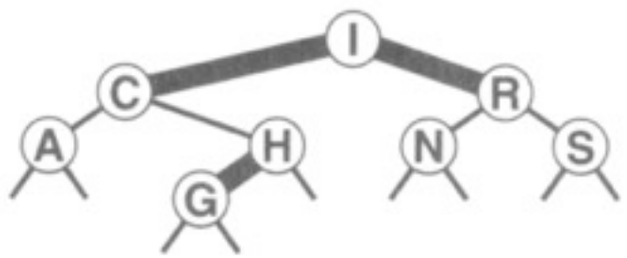


-se agrega el nuevo nodo hasta el final, formando un 3-node

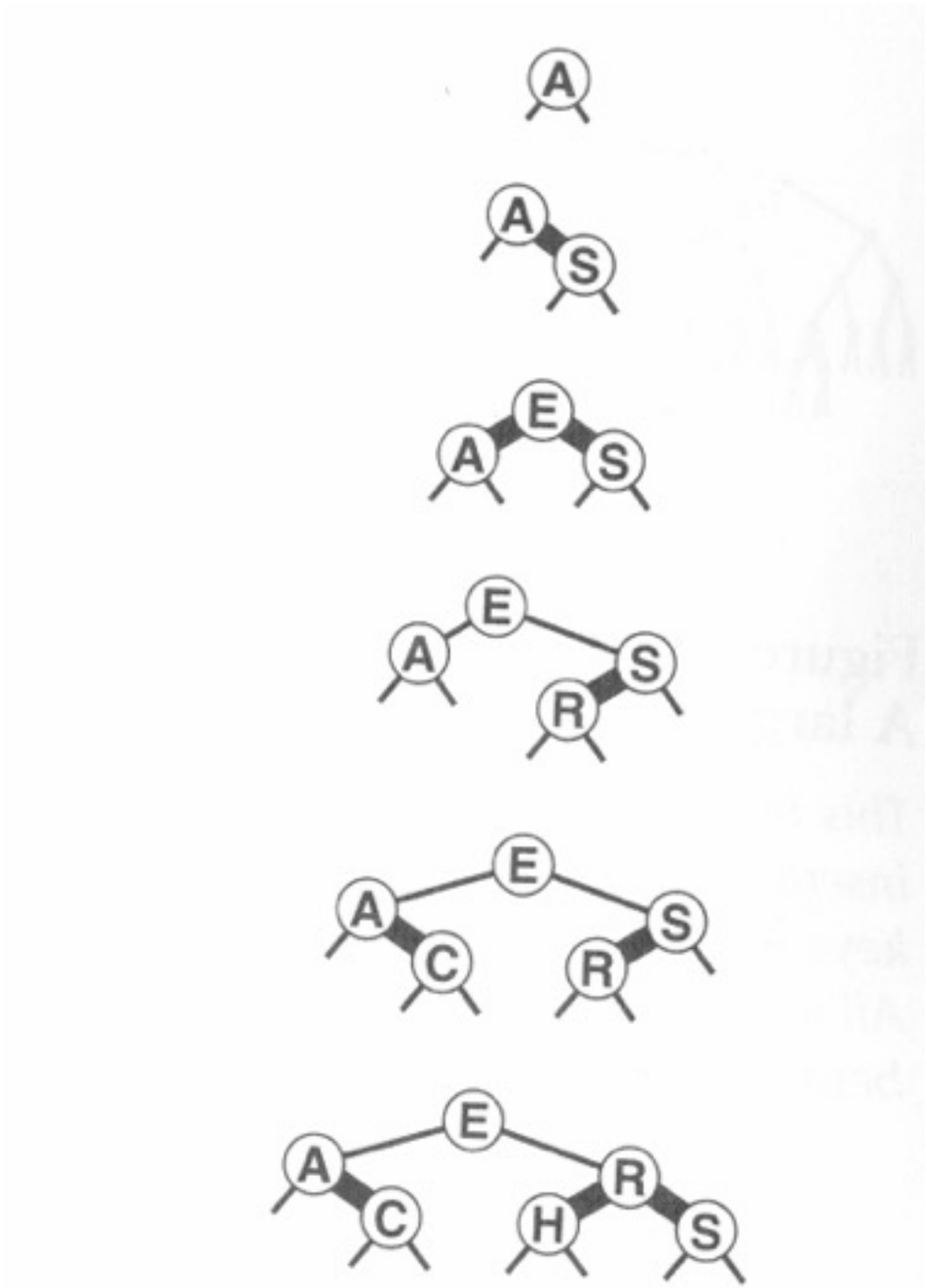


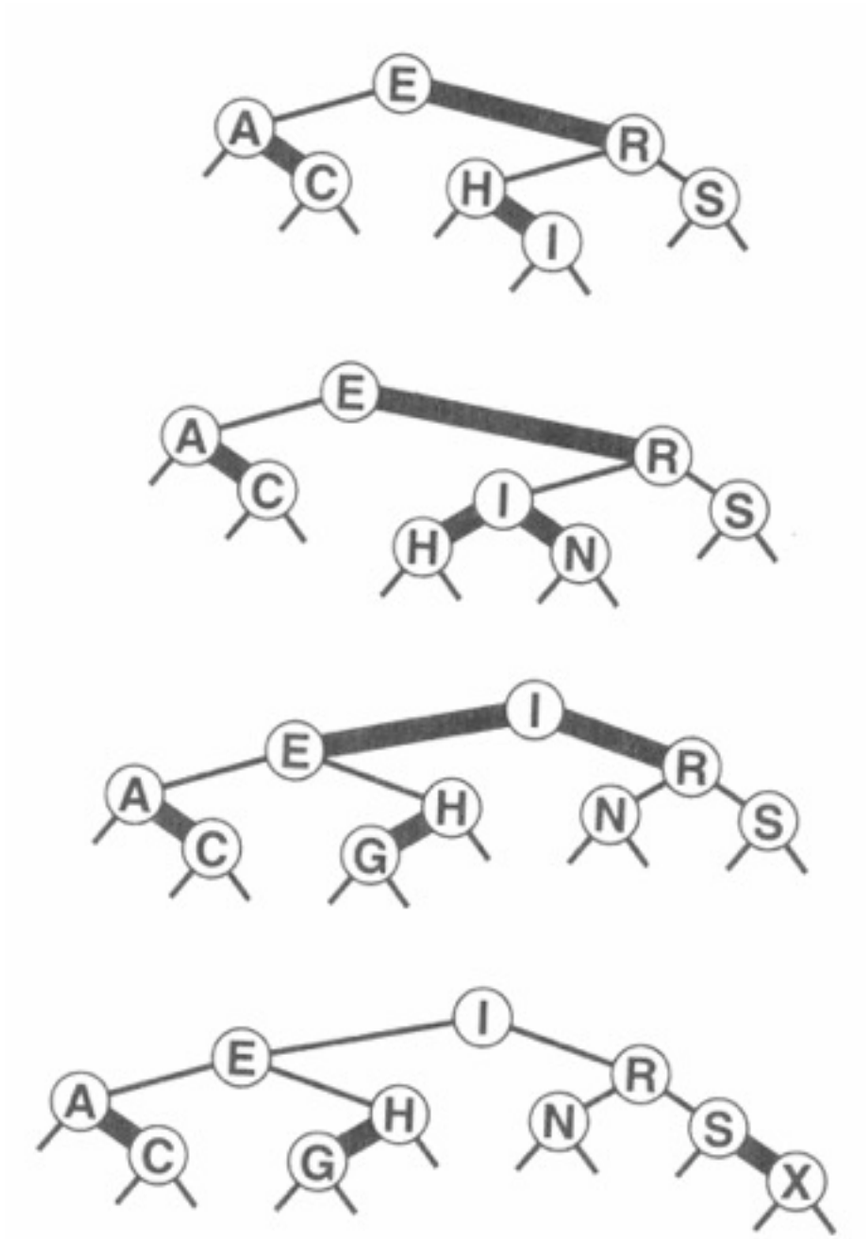
- luego se hace una rotación izquierda en C y

-una rotación derecha en R y cambios de color.

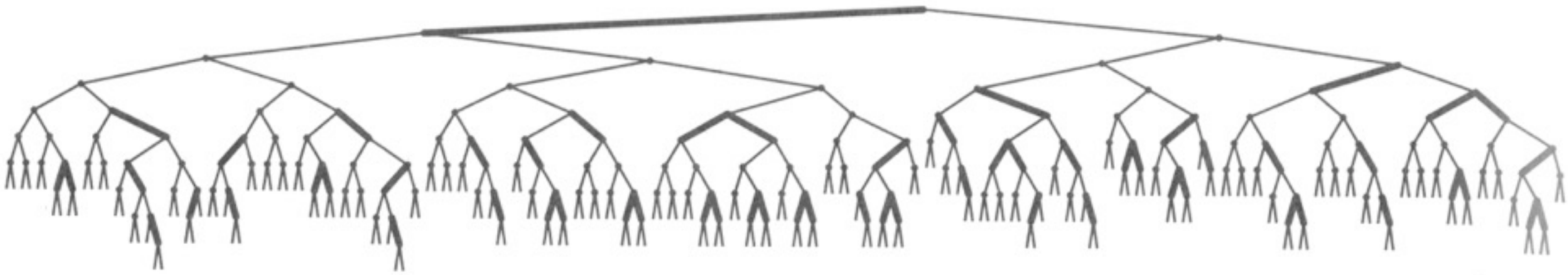


Árboles rojo-negros (construcción) A S E R C H I N G X

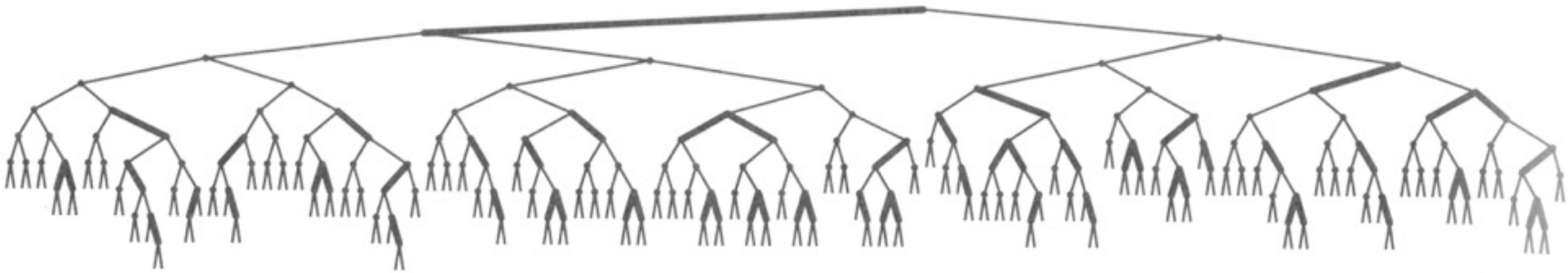




Árboles rojo-negros (uno grande)

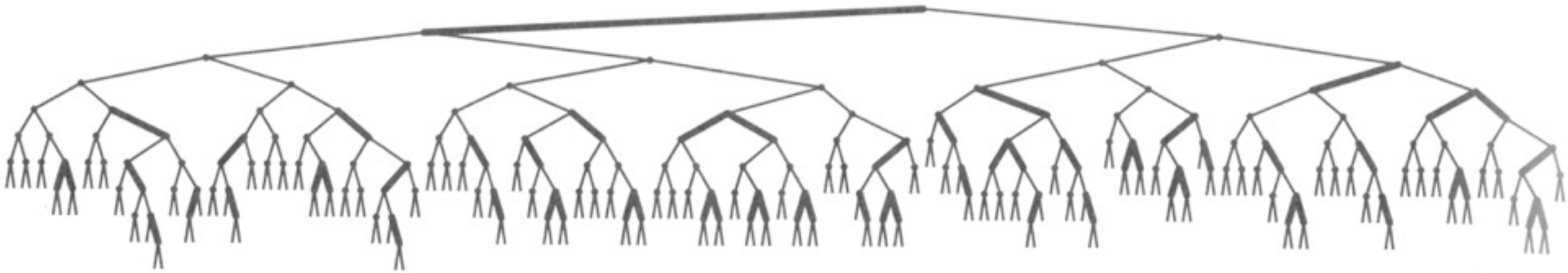


Árboles rojo-negros (uno grande)



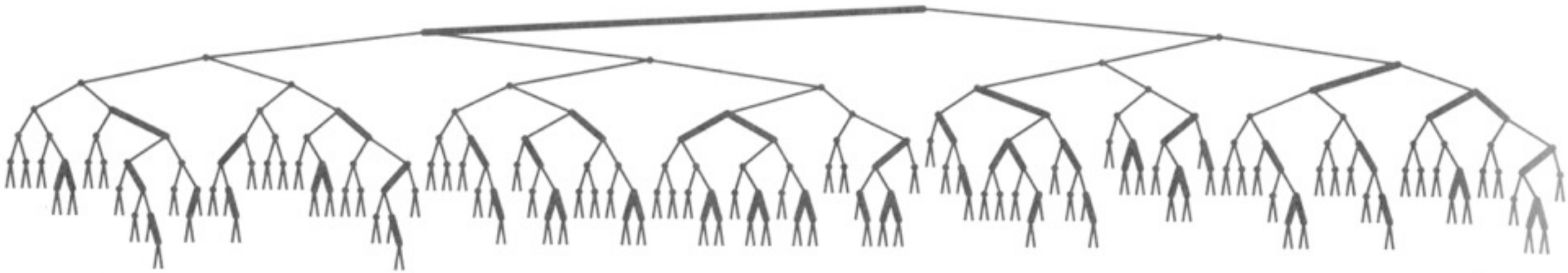
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:

Árboles rojo-negros (uno grande)



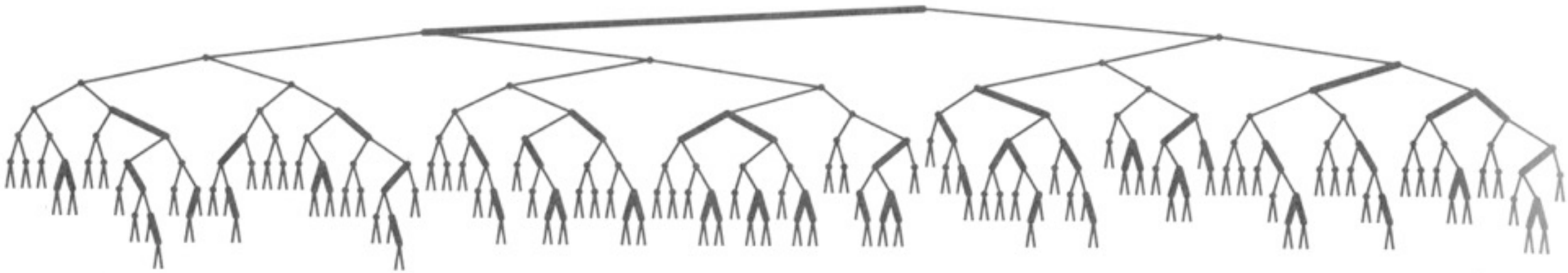
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.

Árboles rojo-negros (uno grande)



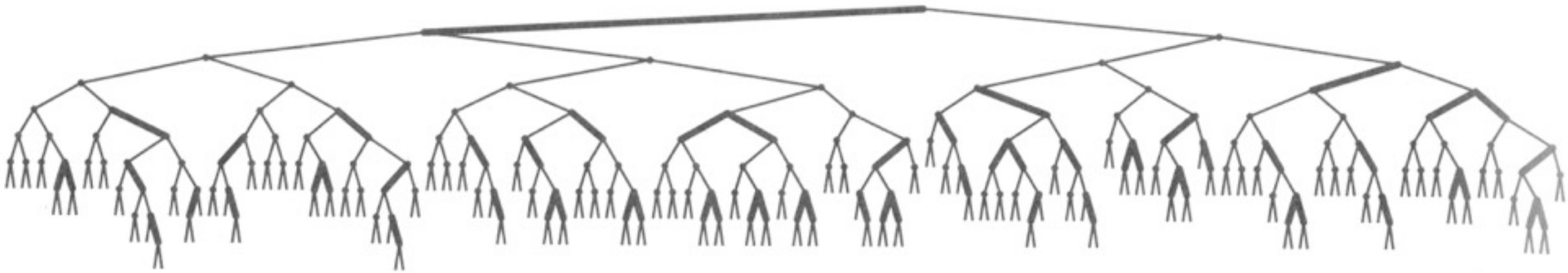
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.
  - La raíz es negra.

Árboles rojo-negros (uno grande)



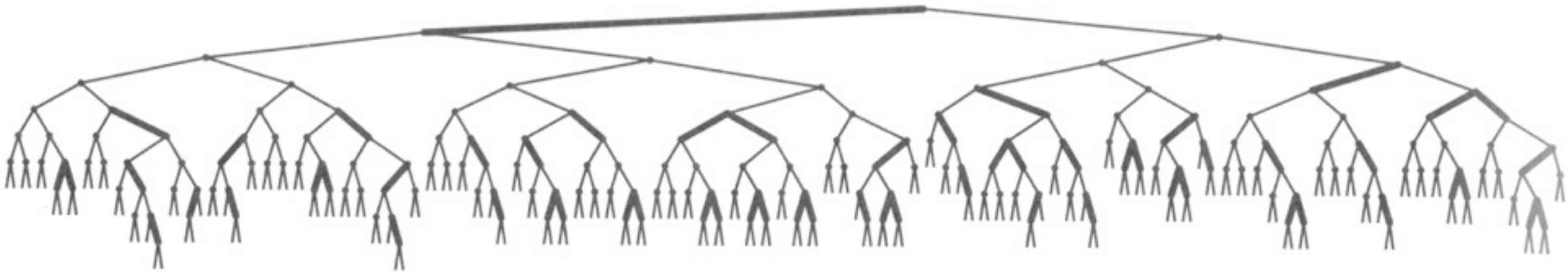
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.
  - La raíz es negra.
  - Cada hoja (apunta a NULL) es negra.

Árboles rojo-negros (uno grande)



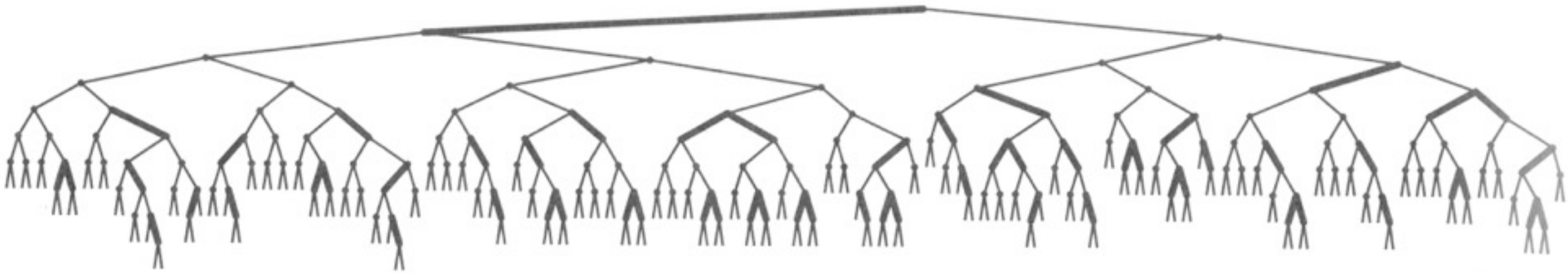
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.
  - La raíz es negra.
  - Cada hoja (apunta a NULL) es negra.
  - Si un nodo es rojo, ambos hijos son negros.

Árboles rojo-negros (uno grande)



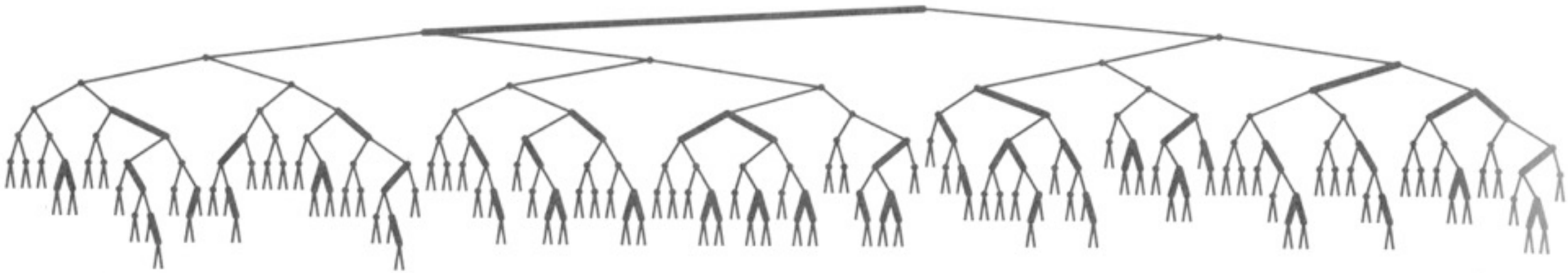
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.
  - La raíz es negra.
  - Cada hoja (apunta a NULL) es negra.
  - Si un nodo es rojo, ambos hijos son negros.
  - Para cada nodo, todos los caminos a partir del nodo que vayan hacia las hojas contienen el mismo número de nodos negros.

Árboles rojo-negros (uno grande)



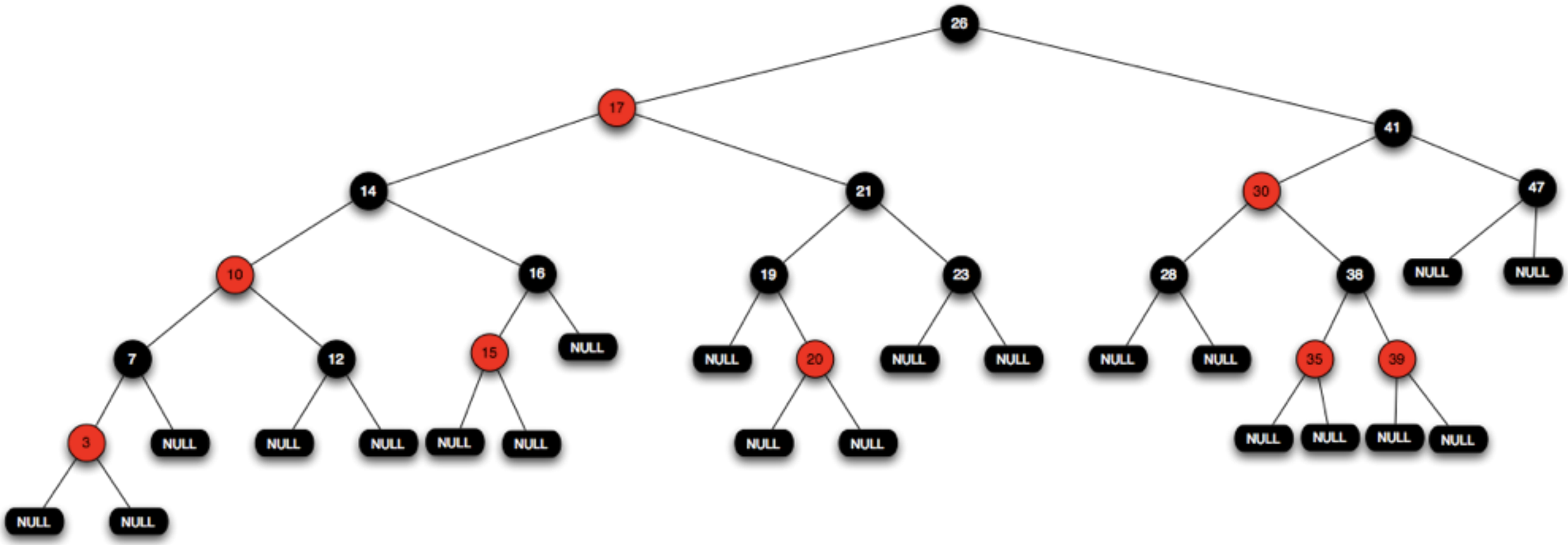
- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.
  - La raíz es negra.
  - Cada hoja (apunta a NULL) es negra.
  - Si un nodo es rojo, ambos hijos son negros.
  - Para cada nodo, todos los caminos a partir del nodo que vayan hacia las hojas contienen el mismo número de nodos negros.
- Tratamos NULL como un nodo de color negro con sus valores p,left,right,y llave en valores arbitrarios.

Árboles rojo-negros (uno grande)



- Un árbol binario es rojo-negro si satisface las siguientes propiedades:
  - Cada nodo es rojo o negro.
  - La raíz es negra.
  - Cada hoja (apunta a NULL) es negra.
  - Si un nodo es rojo, ambos hijos son negros.
  - Para cada nodo, todos los caminos a partir del nodo que vayan hacia las hojas contienen el mismo número de nodos negros.
- Tratamos NULL como un nodo de color negro con sus valores p,left,right,y llave en valores arbitrarios.
- El padre de la raiz es un nodo NULL.

Ejemplo: Árboles rojo-negros



Tablas de *Hash* (de dispersión dinámica)

- mat-151

Diccionarios,mapas o tablas de símbolos

- Estructuras de datos donde los registros tienen típicamente una llave y datos.

Diccionarios,mapas o tablas de símbolos

- Estructuras de datos donde los registros tienen típicamente una llave y datos.

Diccionarios,mapas o tablas de símbolos

- Estructuras de datos donde los registros tienen típicamente una llave y datos.

- Soportan principalmente operaciones de inserción, eliminación y búsqueda.

Diccionarios,mapas o tablas de símbolos

- Estructuras de datos donde los registros tienen típicamente una llave y datos.

- Soportan principalmente operaciones de inserción, eliminación y búsqueda.
- Estas operaciones hacen que el conjunto de registros sea dinámico, es decir, que varia con el tiempo (conjuntos más utilizados en computación).

Diccionarios,mapas o tablas de símbolos

- Estructuras de datos donde los registros tienen típicamente una llave y datos.

- Soportan principalmente operaciones de inserción, eliminación y búsqueda.
- Estas operaciones hacen que el conjunto de registros sea dinámico, es decir, que varia con el tiempo (conjuntos más utilizados en computación).
- La implementación más sencilla es una tabla de acceso directo.

Implementación de diccionarios/mapas

Implementación de diccionarios/mapas

implementación	desempeño
arreglos indexados por llave	eficientes en tiempo y espacio solo si el mapa no está disperso
dos arreglos paralelos uni-dimensionales:	
- ordenados	eficientes en tiempo: ordenar la tabla durante la operación de inserción resulta en operación *search* muy rápida.
- desordenados	eficientes en tiempo para insertar pero búsquedas y eliminaciones lentas.
lista ligada	búsqueda de duplicados. Memoria adicional para links. Se requiere hacer una búsqueda para las operaciones *remove* y *get.*
árbol binario de búsqueda	más eficientes que los demás si el árbol está bien balanceado.
tablas de hash	--

Tablas de acceso directo

Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.

Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.

Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.

Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.
- Podemos especificar un arreglo **T**[0...M-1] que representa al conjunto dinámico **S** tal que:

Tablas de acceso directo

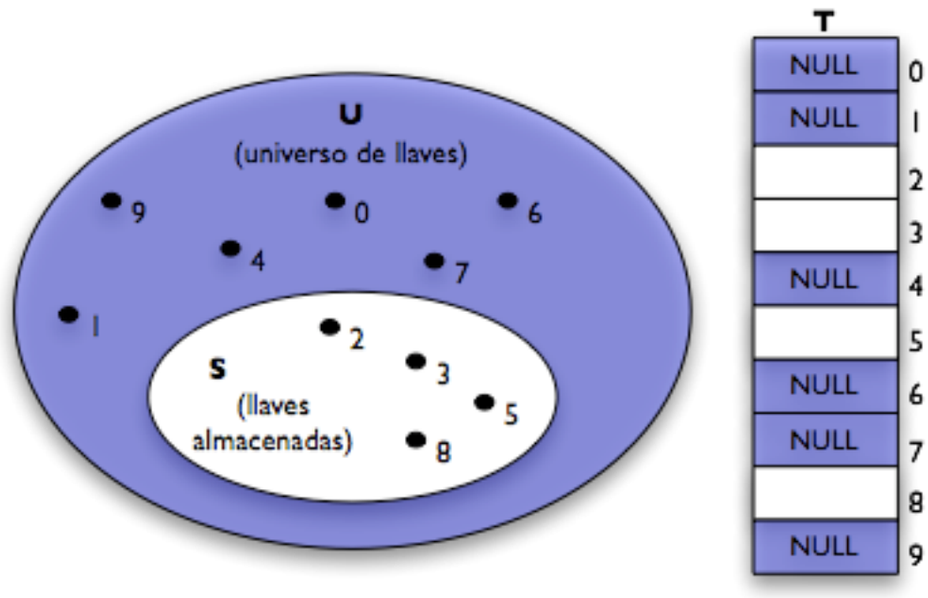
- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.
- Podemos especificar un arreglo **T**[0...M-1] que representa al conjunto dinámico **S** tal que:

$$\begin{cases} T[k] = x & \text{if } x \in S \quad (x.key = k) \\ T[k] = \text{NULL} & \text{en caso contrario.} \end{cases}$$

Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.
- Podemos especificar un arreglo **T**[0...M-1] que representa al conjunto dinámico **S** tal que:

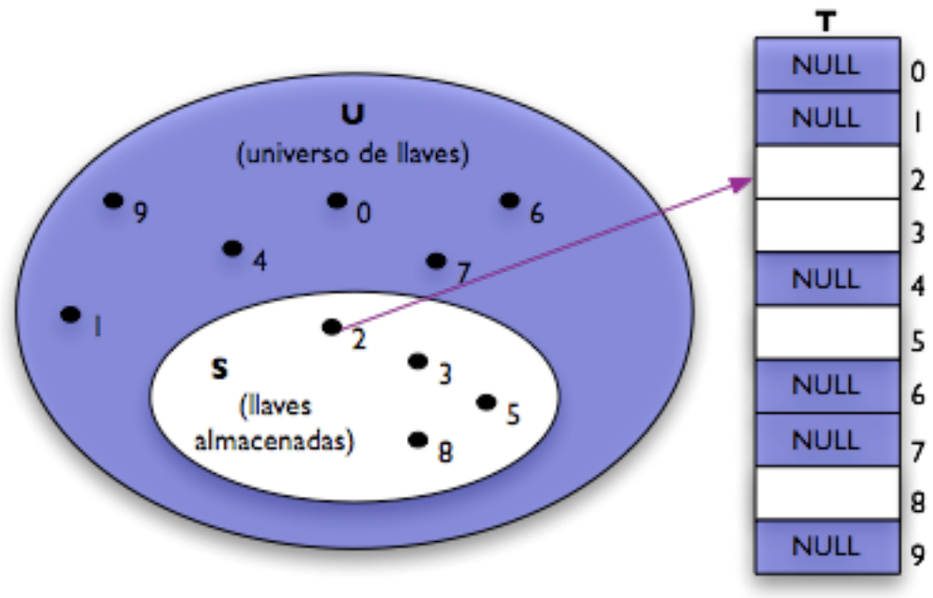
$$\begin{cases} T[k] = x & \text{if } x \in S \quad (x.key = k) \\ T[k] = \text{NULL} & \text{en caso contrario.} \end{cases}$$



Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.
- Podemos especificar un arreglo **T**[0...M-1] que representa al conjunto dinámico **S** tal que:

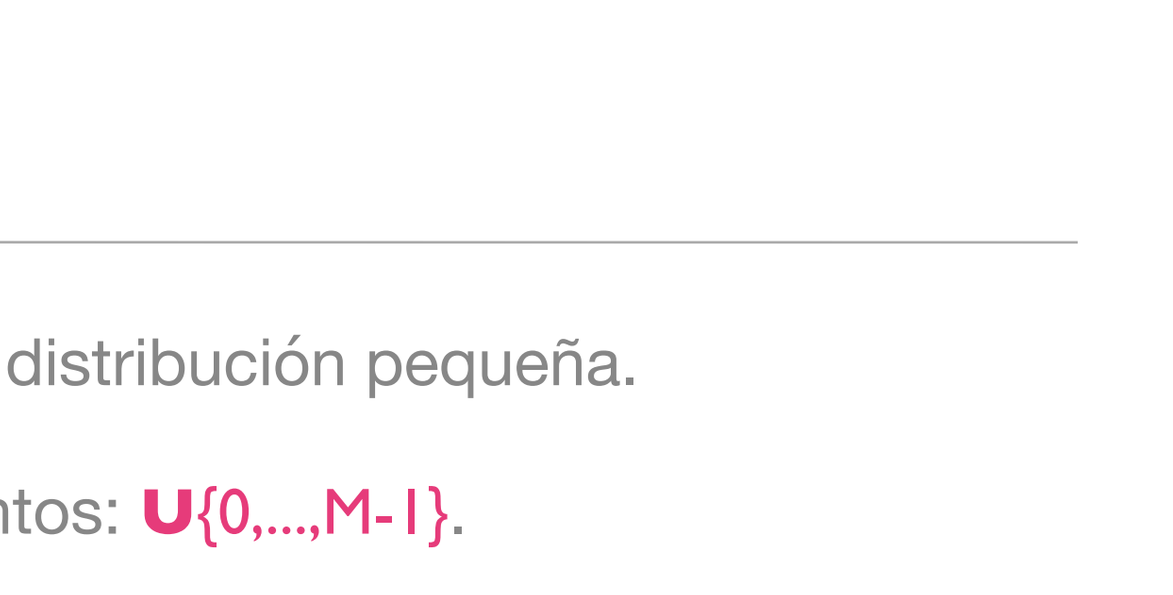
$$\begin{cases} T[k] = x & \text{if } x \in S \quad (x.key = k) \\ T[k] = \text{NULL} & \text{en caso contrario.} \end{cases}$$



Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.
- Podemos especificar un arreglo **T**[0...M-1] que representa al conjunto dinámico **S** tal que:

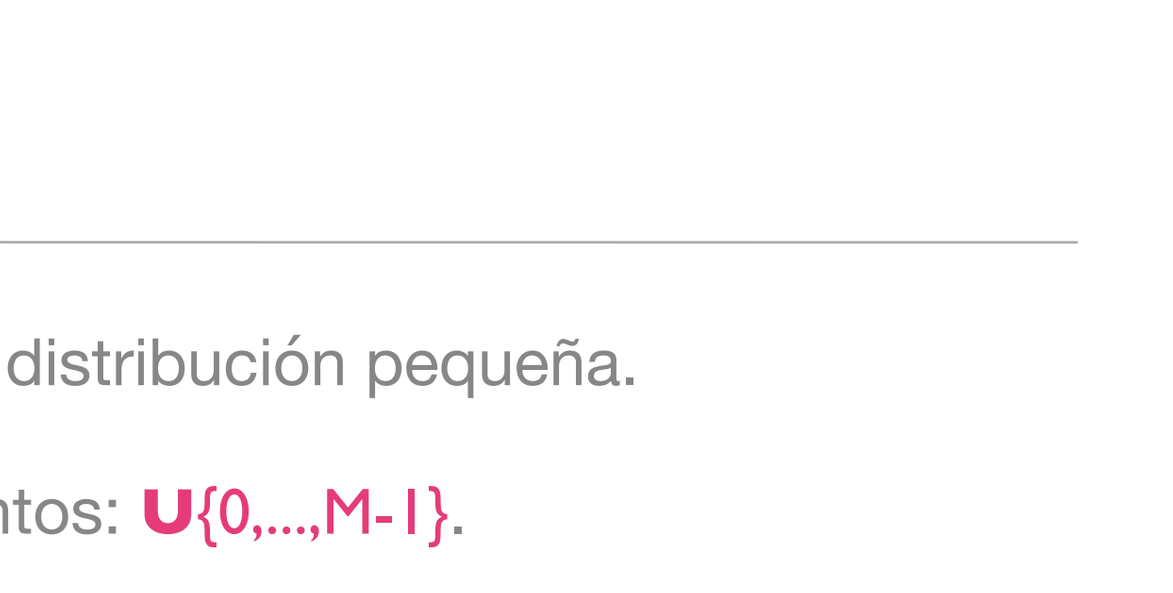
$$\begin{cases} T[k] = x & \text{if } x \in S \quad (x.key = k) \\ T[k] = \text{NULL} & \text{en caso contrario.} \end{cases}$$



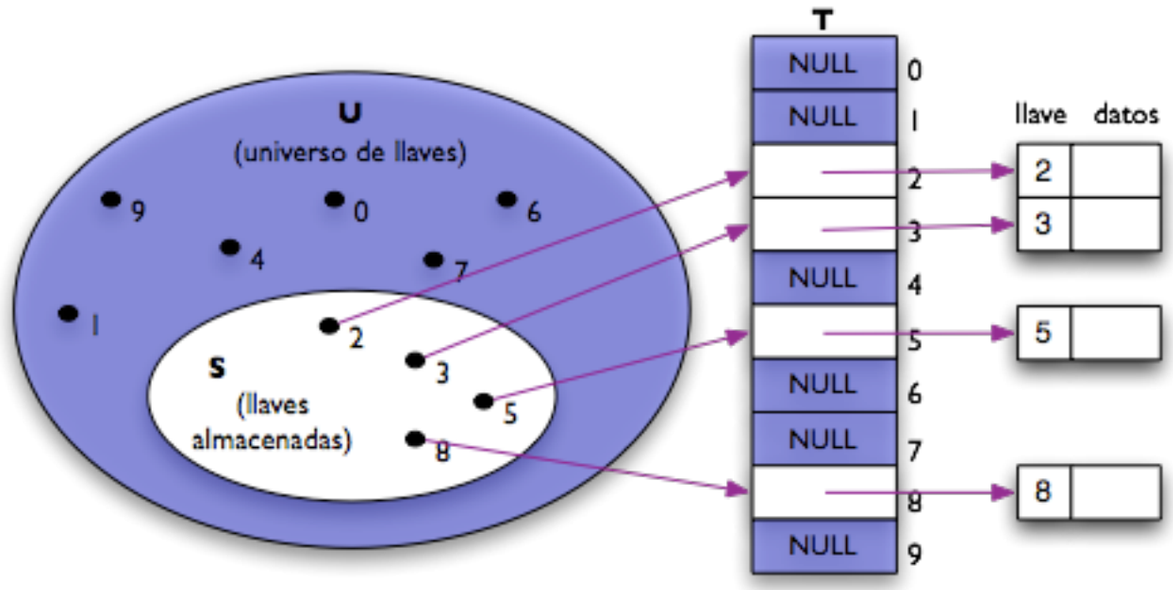
Tablas de acceso directo

- Útiles cuando las llaves se eligen de una distribución pequeña.
  - tenemos un conjunto **U** con M elementos: **U**{0,...,M-1}.
  - todas las llaves son distintas.
- Podemos especificar un arreglo **T**[0...M-1] que representa al conjunto dinámico **S** tal que:

$$\begin{cases} T[k] = x & \text{if } x \in S \quad (x.key = k) \\ T[k] = \text{NULL} & \text{en caso contrario.} \end{cases}$$

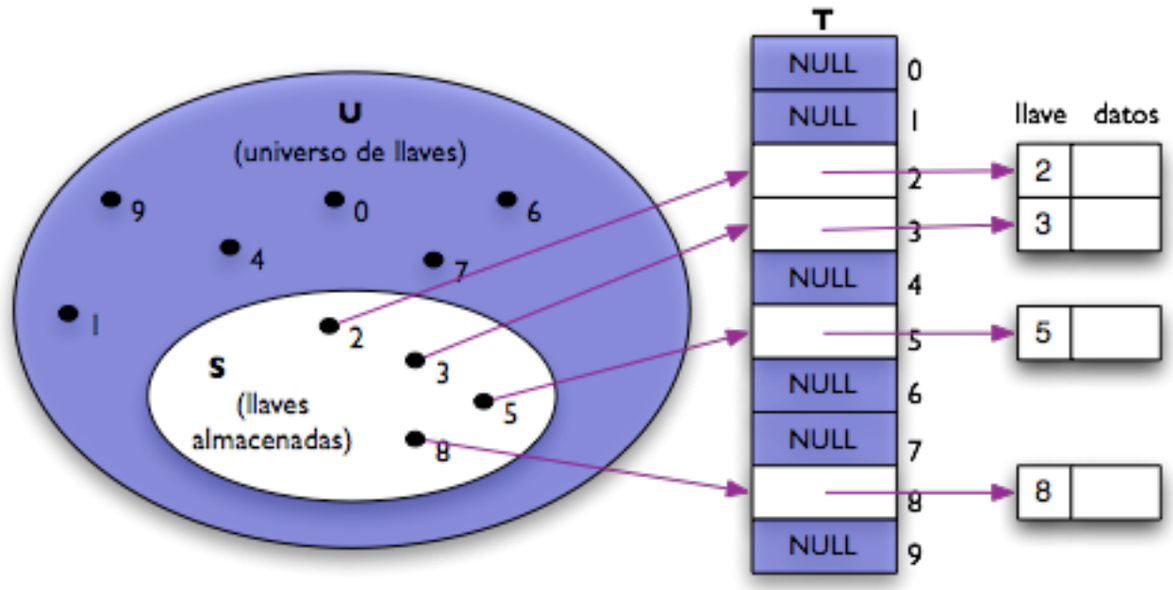


Tablas de acceso directo: **operaciones**



Tablas de acceso directo: **operaciones**

- Las operaciones de un diccionario son triviales de implementar.

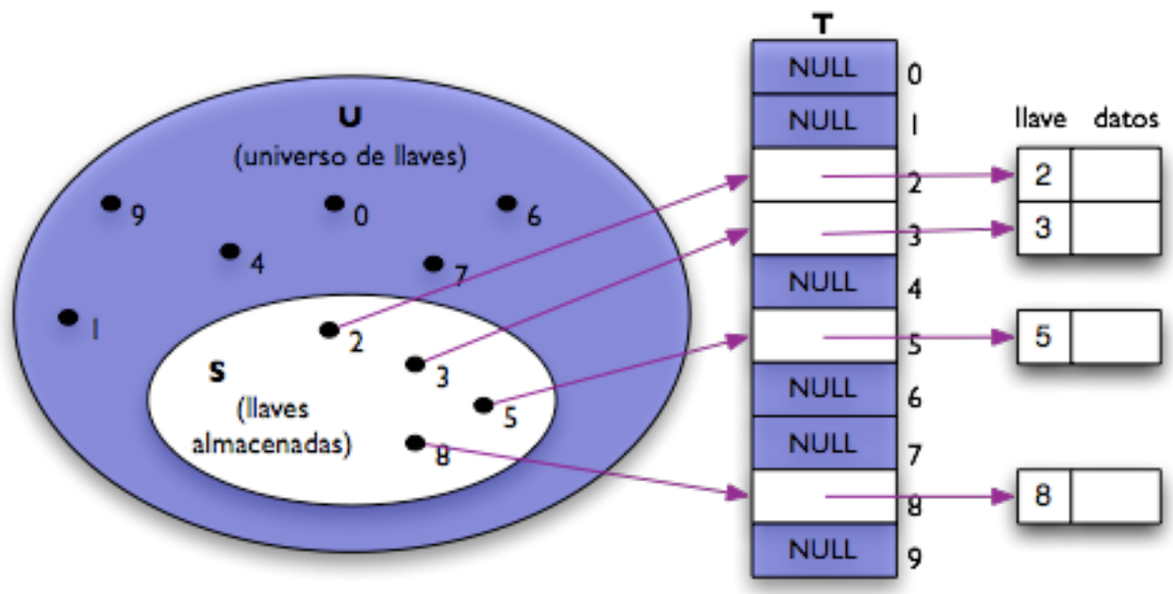


Tablas de acceso directo: **operaciones**

- Las operaciones de un diccionario son triviales de implementar.

DIRECT-ADDRESS-SEARCH(T, k)

1. return T[k]



Tablas de acceso directo: **operaciones**

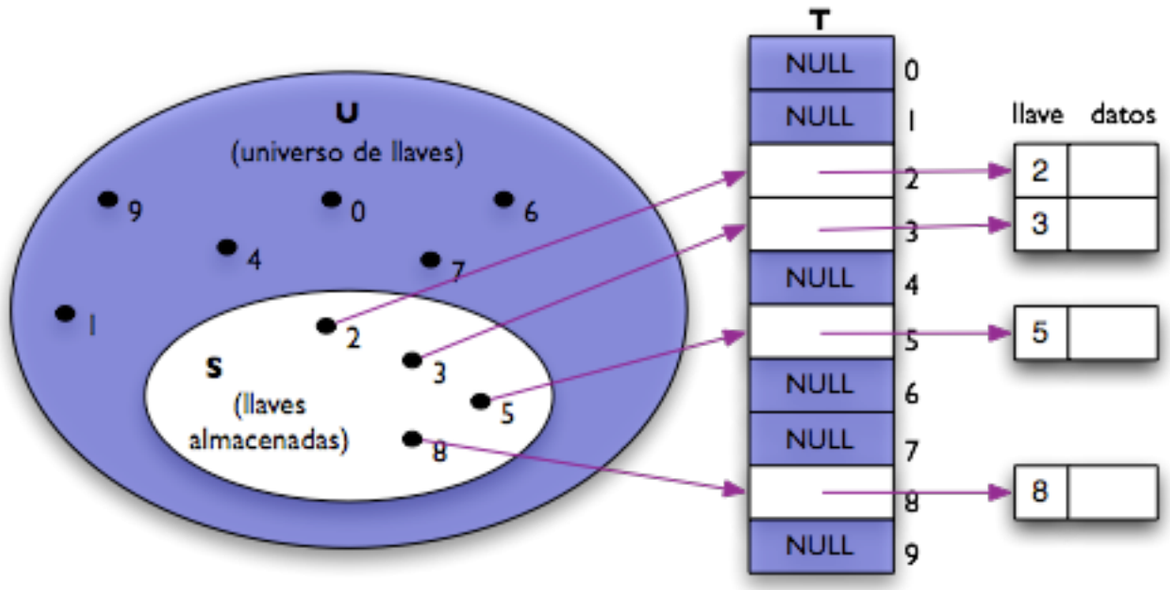
- Las operaciones de un diccionario son triviales de implementar.

DIRECT-ADDRESS-SEARCH(T, k)

1. return T[k]

DIRECT-ADDRESS-INSERT(T, x)

- T[x.key] ← x



Tablas de acceso directo: **operaciones**

- Las operaciones de un diccionario son triviales de implementar.

DIRECT-ADDRESS-SEARCH(T, k)

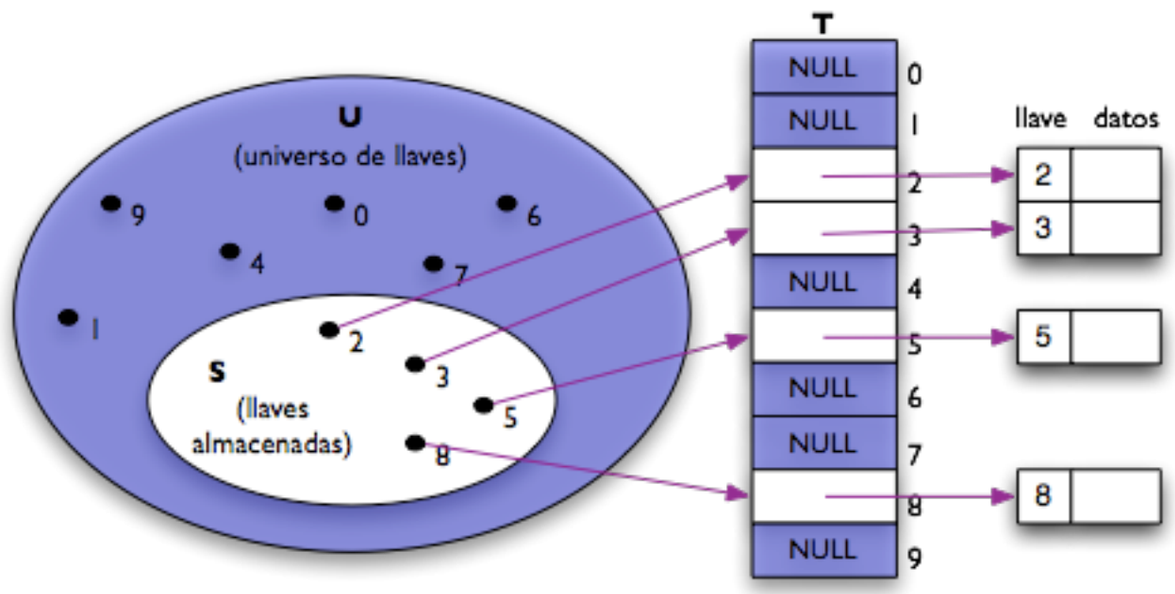
1. return T[k]

DIRECT-ADDRESS-INSERT(T, x)

- T[x.key] ← x

DIRECT-ADDRESS-DELETE(T, x)

- T[x.key] ← NULL



Tablas de acceso directo: **operaciones**

- Las operaciones de un diccionario son triviales de implementar.

DIRECT-ADDRESS-SEARCH(T, k)

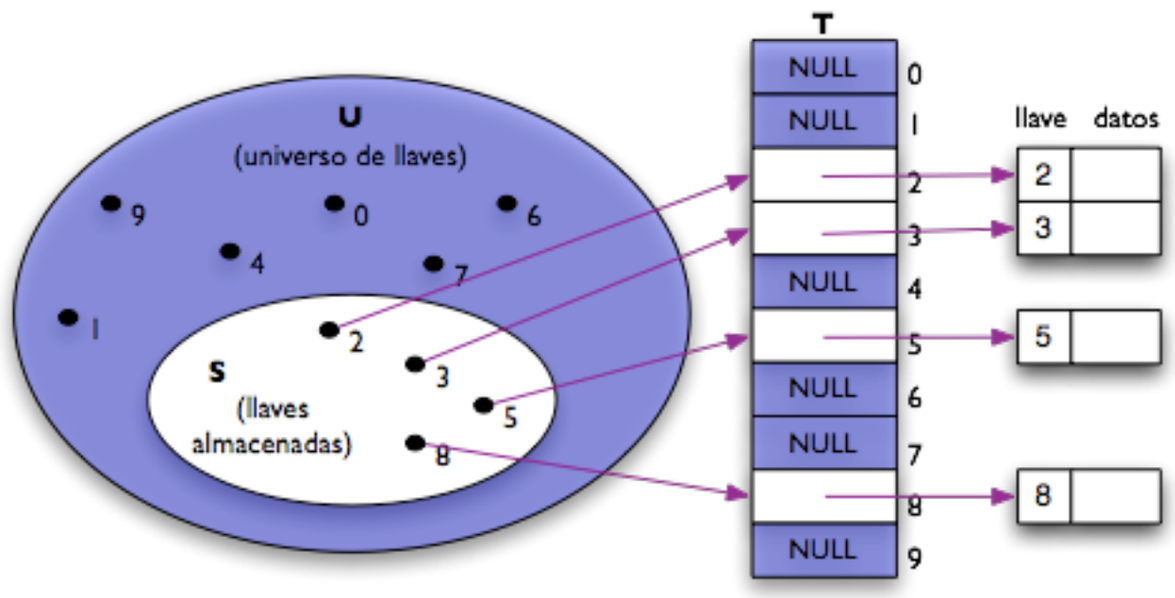
1. return T[k]

DIRECT-ADDRESS-INSERT(T, x)

- T[x.key] ← x

DIRECT-ADDRESS-DELETE(T, x)

- T[x.key] ← NULL



- Estas operaciones toman O(1) como tiempo de ejecución en el peor caso.

Tablas de acceso directo: **operaciones**

- Las operaciones de un diccionario son triviales de implementar.

DIRECT-ADDRESS-SEARCH(T, k)

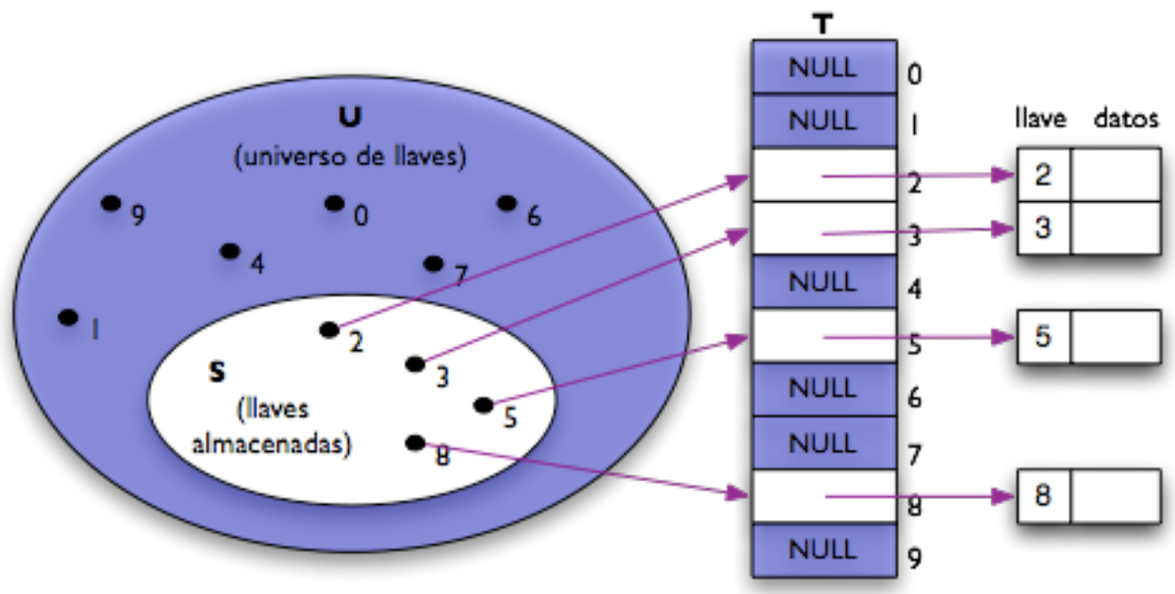
1. return T[k]

DIRECT-ADDRESS-INSERT(T, x)

- T[x.key] ← x

DIRECT-ADDRESS-DELETE(T, x)

- T[x.key] ← NULL



- Estas operaciones toman O(1) como tiempo de ejecución en el peor caso.
- ¿Limitación de las tablas de acceso directo?

Tablas de acceso directo: **límites**

Tablas de acceso directo: **límites**

- m-l puede ser muy grande, ej. valores de 64 bits y algunos miles de valores dentro de S.

Tablas de acceso directo: **límites**

- m-l puede ser muy grande, ej. valores de 64 bits y algunos miles de valores dentro de S.
- T [0... 2^{64}-1] donde se usan solo unos pocos registros.

Tablas de acceso directo: **límites**

- m-l puede ser muy grande, ej. valores de 64 bits y algunos miles de valores dentro de S.
- T [0... 2^{64}-1] donde se usan solo unos pocos registros.
- otro ejemplo: nombres de personas como llave

Tablas de acceso directo: **límites**

- m-l puede ser muy grande, ej. valores de 64 bits y algunos miles de valores dentro de S.
- T [0... 2^{64}-1] donde se usan solo unos pocos registros.
- otro ejemplo: nombres de personas como llave
- Idea: tener una estructura de datos con las mismas propiedades pero más pequeña.

Tablas de *hash*

Tablas de *hash*

- Estructura de datos que soporta las operaciones de un diccionario: INSERT, SEARCH, DELETE.

Tablas de *hash*

- Estructura de datos que soporta las operaciones de un diccionario: INSERT, SEARCH, DELETE.
- En el peor caso es equivalente a buscar un elemento en una lista ligada: $\Theta(n)$

Tablas de *hash*

- Estructura de datos que soporta las operaciones de un diccionario: INSERT, SEARCH, DELETE.
- En el peor caso es equivalente a buscar un elemento en una lista ligada: $\Theta(n)$
- En práctica, su desempeño es de $O(1)$

Tablas de *hash*

- Estructura de datos que soporta las operaciones de un diccionario: INSERT, SEARCH, DELETE.
- En el peor caso es equivalente a buscar un elemento en una lista ligada: $\Theta(n)$
- En práctica, su desempeño es de $O(1)$
- Es la generalización de un arreglo ordinario (permite direccionamiento directo)

Tablas de *hash*

- Estructura de datos que soporta las operaciones de un diccionario: INSERT, SEARCH, DELETE.
- En el peor caso es equivalente a buscar un elemento en una lista ligada: $\Theta(n)$
- En práctica, su desempeño es de $O(1)$
- Es la generalización de un arreglo ordinario (permite direccionamiento directo)
- El direccionamiento directo es posible cuando tenemos suficiente memoria para almacenar una posición para cada llave posible.

Tablas de indexado directo vs. tablas de hash

Tablas de indexado directo vs. tablas de hash

- Tiempo de ejecución O(1).

Tablas de indexado directo vs. tablas de hash

- Tiempo de ejecución O(1).
- Si el universo de llaves **U** es grande, almacenar una tabla **T** es impráctico o imposible.

Tablas de indexado directo vs. tablas de hash

- Tiempo de ejecución O(1).
- Si el universo de llaves **U** es grande, almacenar una tabla **T** es impráctico o imposible.
- Si el conjunto de llaves **K** es mucho más pequeño que **U**, se desperdicia mucho espacio.

Tablas de indexado directo vs. tablas de hash

- Tiempo de ejecución $O(1)$.
- Si el universo de llaves **U** es grande, almacenar una tabla **T** es impráctico o imposible.
- Si el conjunto de llaves **K** es mucho más pequeño que **U**, se desperdicia mucho espacio.
- Con una tabla de hash, el espacio necesario en este caso es $\Theta(K)$ y se mantienen los beneficios para una búsqueda de $O(1)$.

Tablas de indexado directo vs. tablas de hash

- Tiempo de ejecución $O(1)$.
- Si el universo de llaves **U** es grande, almacenar una tabla **T** es impráctico o imposible.
- Si el conjunto de llaves **K** es mucho más pequeño que **U**, se desperdicia mucho espacio.
- Con una tabla de hash, el espacio necesario en este caso es $\Theta(K)$ y se mantienen los beneficios para una búsqueda de $O(1)$.
- Con indexado directo el elemento con llave k se almacena en el campo k.

Tablas de indexado directo vs. tablas de hash

- Tiempo de ejecución $O(1)$.
- Si el universo de llaves **U** es grande, almacenar una tabla **T** es impráctico o imposible.
- Si el conjunto de llaves **K** es mucho más pequeño que **U**, se desperdicia mucho espacio.
- Con una tabla de hash, el espacio necesario en este caso es $\Theta(K)$ y se mantienen los beneficios para una búsqueda de $O(1)$.
- Con indexado directo el elemento con llave k se almacena en el campo k.
- Con *hashing*, el elemento con llave k se almacena en el campo H(k).

Tablas de *hash*

$\mathcal{H}$

Tablas de *hash*

- Utilizar una función, llamada función de hash $\mathcal{H}$, para hacer un mapeo de las llaves del conjunto dinámico a campos de la tabla T.

Tablas de *hash*

- Utilizar una función, llamada función de hash $\mathcal{H}$, para hacer un mapeo de las llaves del conjunto dinámico a campos de la tabla T.
  - tenemos el conjunto **U** de llaves posibles,

Tablas de *hash*

- Utilizar una función, llamada función de hash $\mathcal{H}$, para hacer un mapeo de las llaves del conjunto dinámico a campos de la tabla T.
  - tenemos el conjunto **U** de llaves posibles,
  - una tabla **T** con m campos o índices **T**[0...m-1],

Tablas de *hash*

- Utilizar una función, llamada función de hash $\mathcal{H}$, para hacer un mapeo de las llaves del conjunto dinámico a campos de la tabla T.
  - tenemos el conjunto **U** de llaves posibles,
  - una tabla **T** con m campos o índices **T**[0...m-1],
  - un conjunto dinámico **S** (pequeño respecto a U)

Tablas de *hash*

- Utilizar una función, llamada función de hash $\mathcal{H}$, para hacer un mapeo de las llaves del conjunto dinámico a campos de la tabla T.
  - tenemos el conjunto **U** de llaves posibles,
  - una tabla **T** con m campos o índices **T**[0...m-1],
  - un conjunto dinámico **S** (pequeño respecto a U)
- $\mathcal{H}$ va a distribuir los elementos de **S** en la tabla **T**.

Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

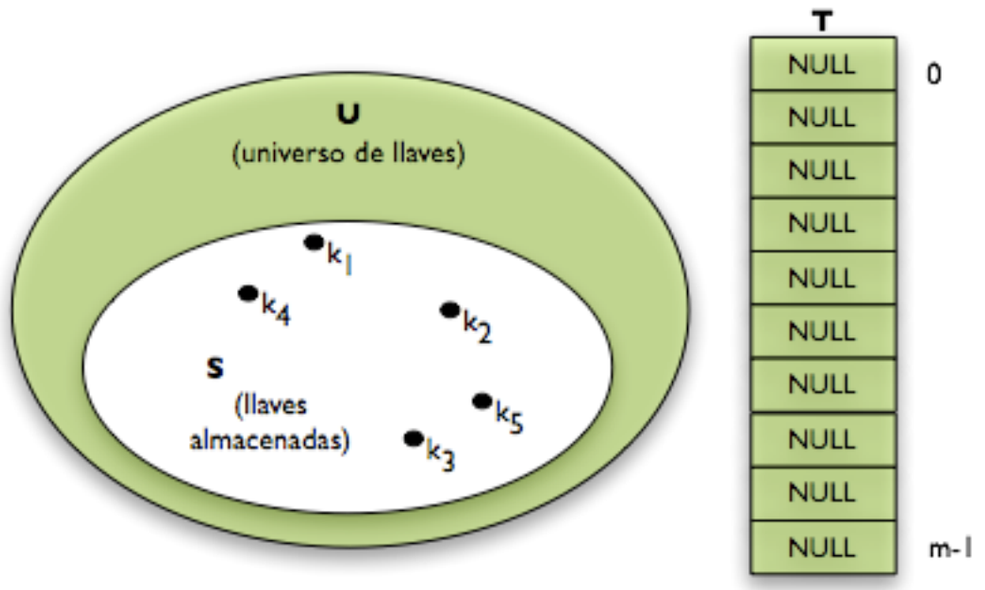
- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

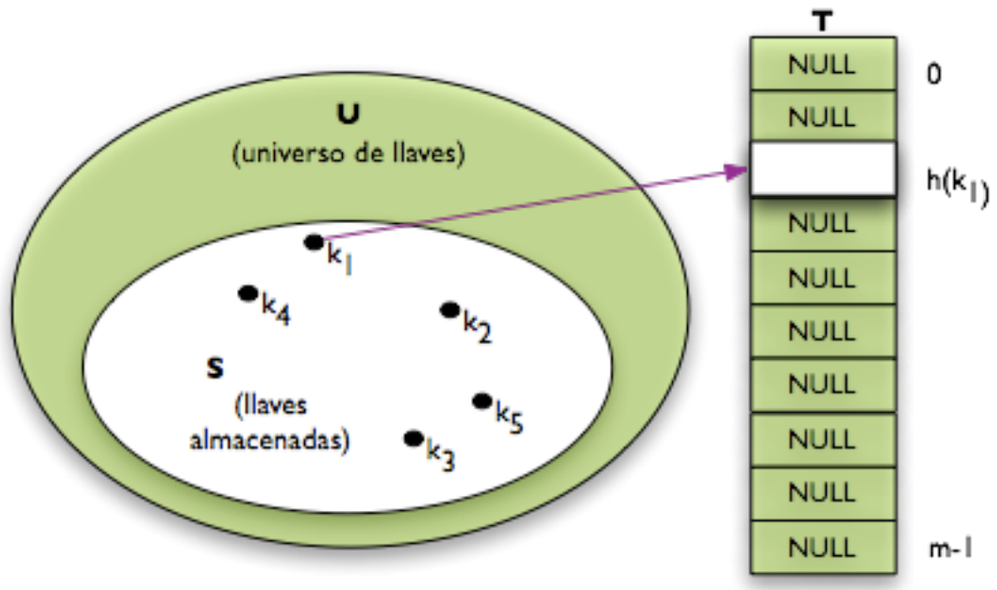


Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

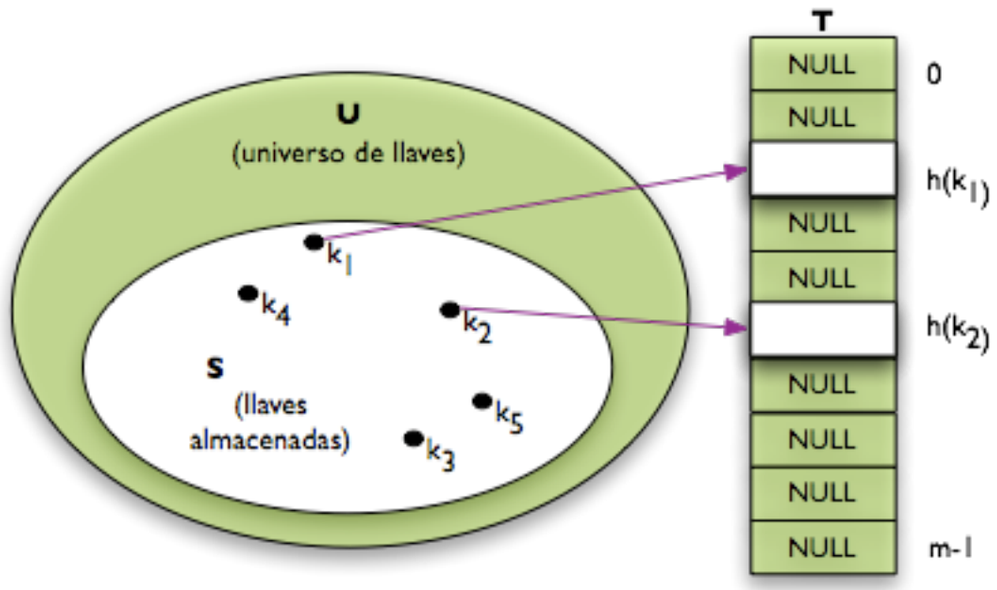


Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

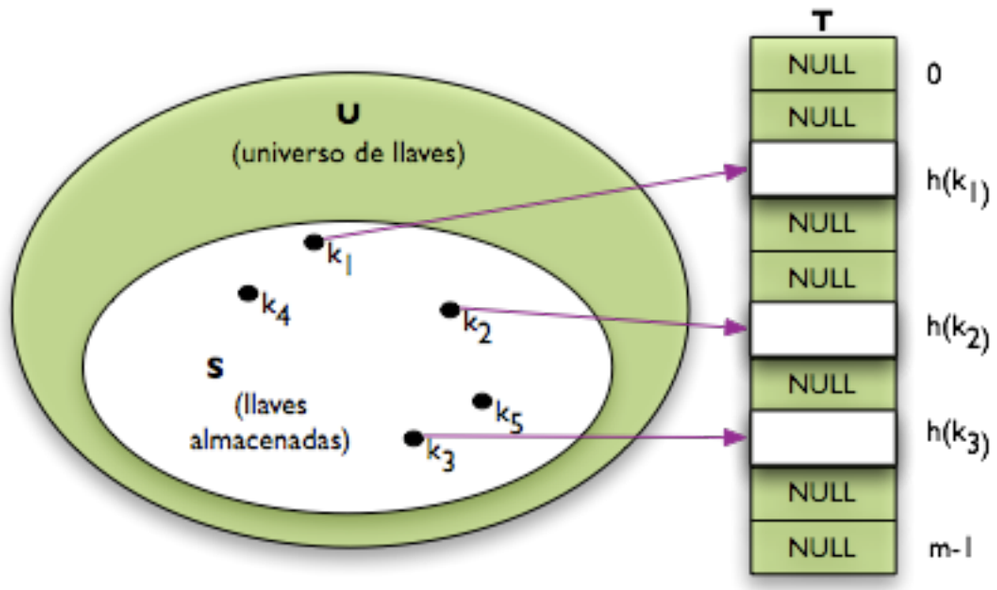


Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

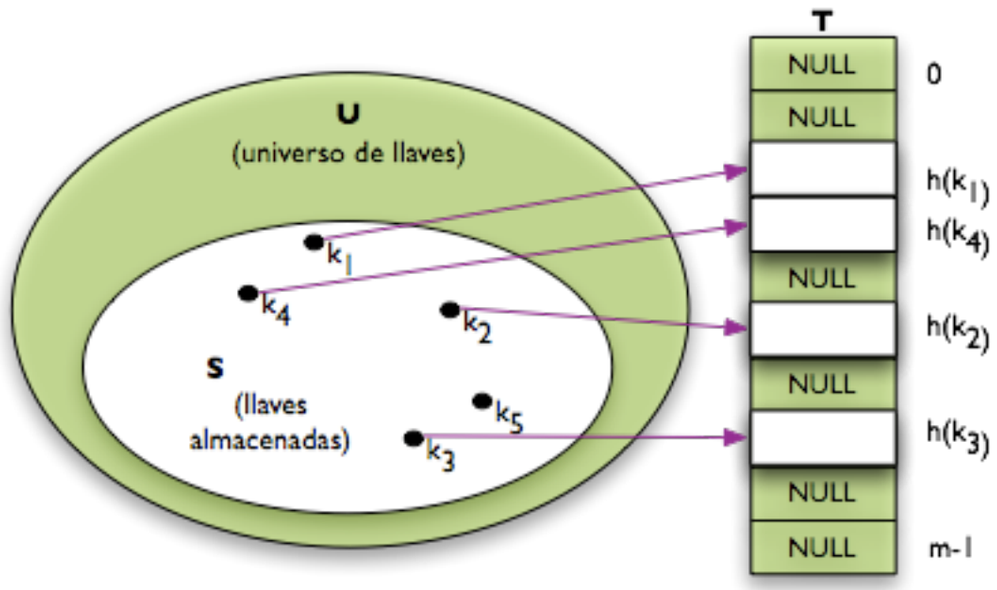


Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.



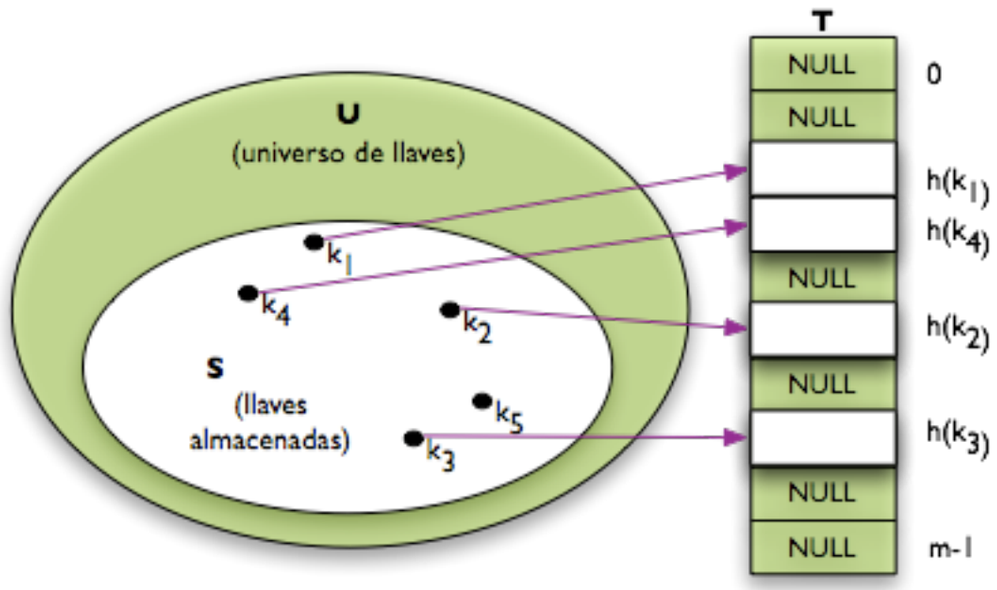
Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

- ¿Problema?
  - dos llaves pueden apuntar al mismo índice debido al uso de la función de hash: colisión.



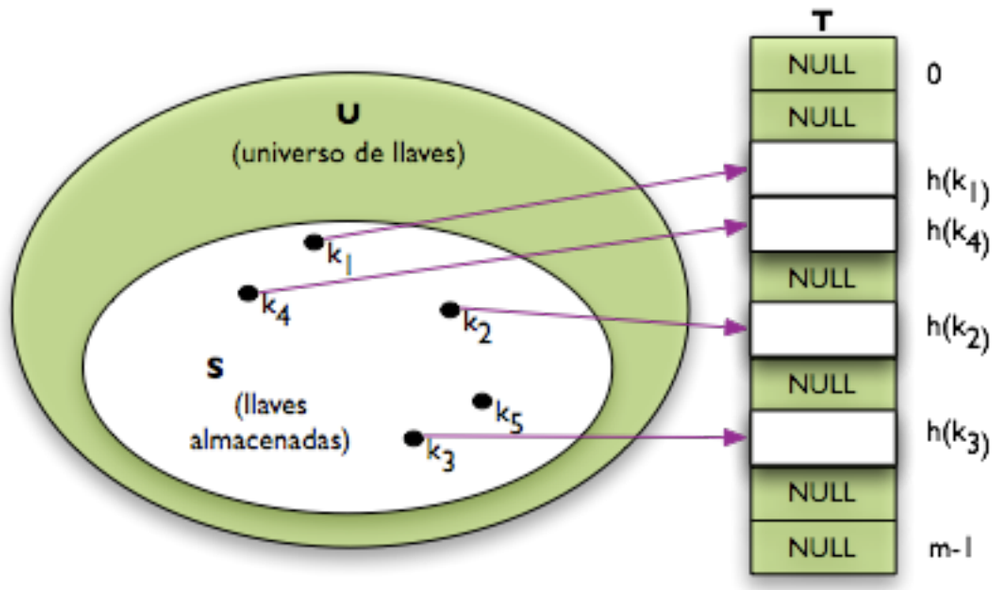
Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

- ¿Problema?
  - dos llaves pueden apuntar al mismo índice debido al uso de la función de hash: colisión.
- Existen técnicas efectivas para resolver los conflictos creados por colisiones.



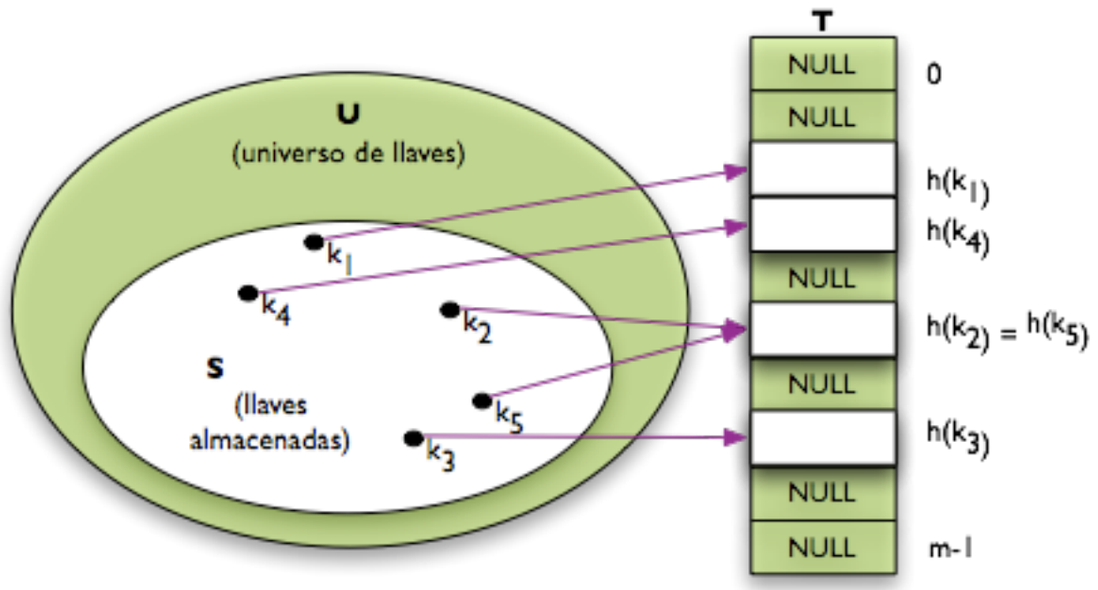
Tablas de hash: **función de hash H**

- La función H mapea el universo **U** de llaves a los campos o índices de una tabla de hash **T**[0..m-1]:

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

- Decimos que un elemento con llave k se mapea al índice h(k) y que h(k) es el valor de hash de la llave k.

- ¿Problema?
  - dos llaves pueden apuntar al mismo índice debido al uso de la función de hash: colisión.
- Existen técnicas efectivas para resolver los conflictos creados por colisiones.



Tablas de hash: **resolviendo colisiones**

$\mathcal{H}$

$\mathcal{H}$

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?

$\mathcal{H}$

$\mathcal{H}$

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.

$\mathcal{H}$

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.
- Idea: función que parezca producir valores aleatorios?

$\mathcal{H}$

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.
- Idea: función que parezca producir valores aleatorios?
  - $\mathcal{H}$ debe ser determinista, es decir, debe dar el mismo resultado con los mismos parámetros de entrada.

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.
- Idea: función que parezca producir valores aleatorios?
  - $\mathcal{H}$ debe ser determinista, es decir, debe dar el mismo resultado con los mismos parámetros de entrada.
  - La función $\mathcal{H}$ de valores aparentemente aleatorios reducirá el número de colisiones pero no las eliminará (#**U >>** m).

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.
- Idea: función que parezca producir valores aleatorios?
  - $\mathcal{H}$ debe ser determinista, es decir, debe dar el mismo resultado con los mismos parámetros de entrada.
  - La función $\mathcal{H}$ de valores aparentemente aleatorios reducirá el número de colisiones pero no las eliminará (#**U >>** m).
- Dos técnicas para evitar colisiones:

Tablas de hash: **resolviendo colisiones**

- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.
- Idea: función que parezca producir valores aleatorios?
  - $\mathcal{H}$ debe ser determinista, es decir, debe dar el mismo resultado con los mismos parámetros de entrada.
  - La función $\mathcal{H}$ de valores aparentemente aleatorios reducirá el número de colisiones pero no las eliminará (#**U >>** m).
- Dos técnicas para evitar colisiones:
  - encadenamiento (chaining),

Tablas de hash: **resolviendo colisiones**

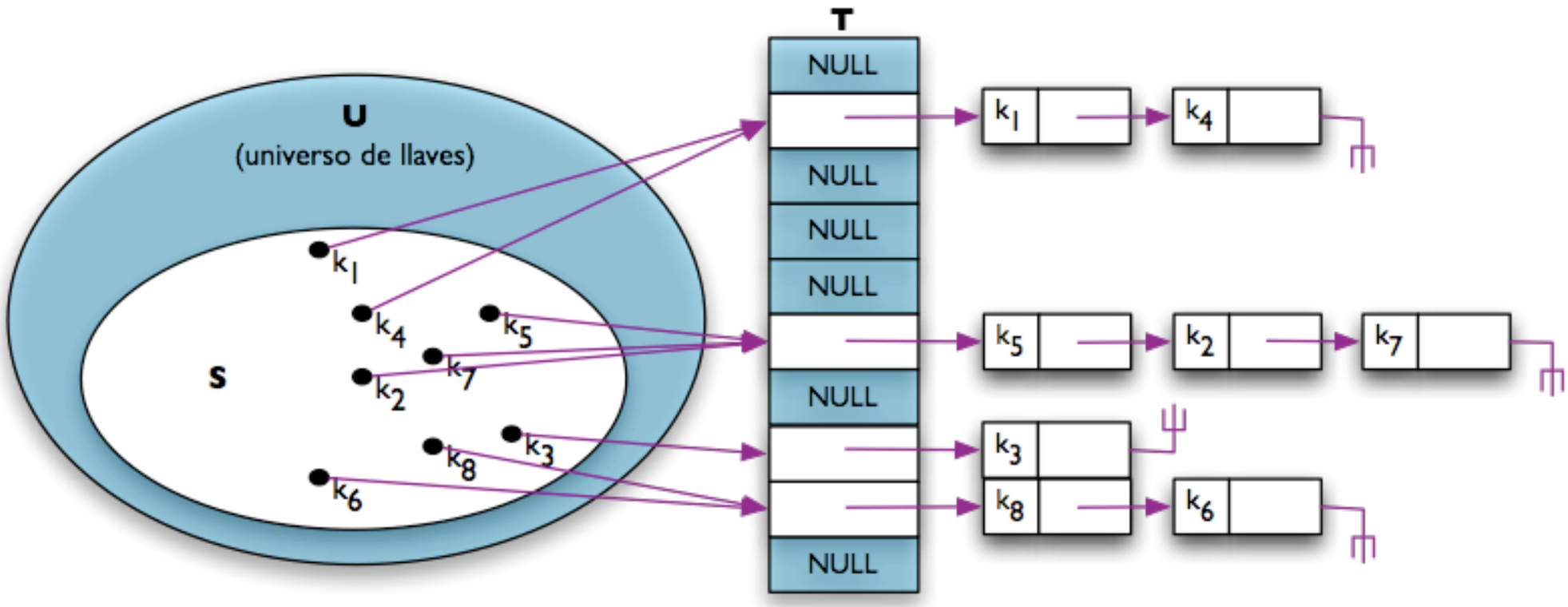
- ¿Cómo resolver colisiones (sin perder datos) ?
- La manera más simple sería tener una $\mathcal{H}$ tal que no genere índices repetidos.
- Idea: función que parezca producir valores aleatorios?
  - $\mathcal{H}$ debe ser determinista, es decir, debe dar el mismo resultado con los mismos parámetros de entrada.
  - La función $\mathcal{H}$ de valores aparentemente aleatorios reducirá el número de colisiones pero no las eliminará (#**U >>** m).
- Dos técnicas para evitar colisiones:
  - encadenamiento (chaining),
  - direccionamiento abierto.

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Idea: poner una liga a los elementos en el mismo campo haciendo una lista ligada.

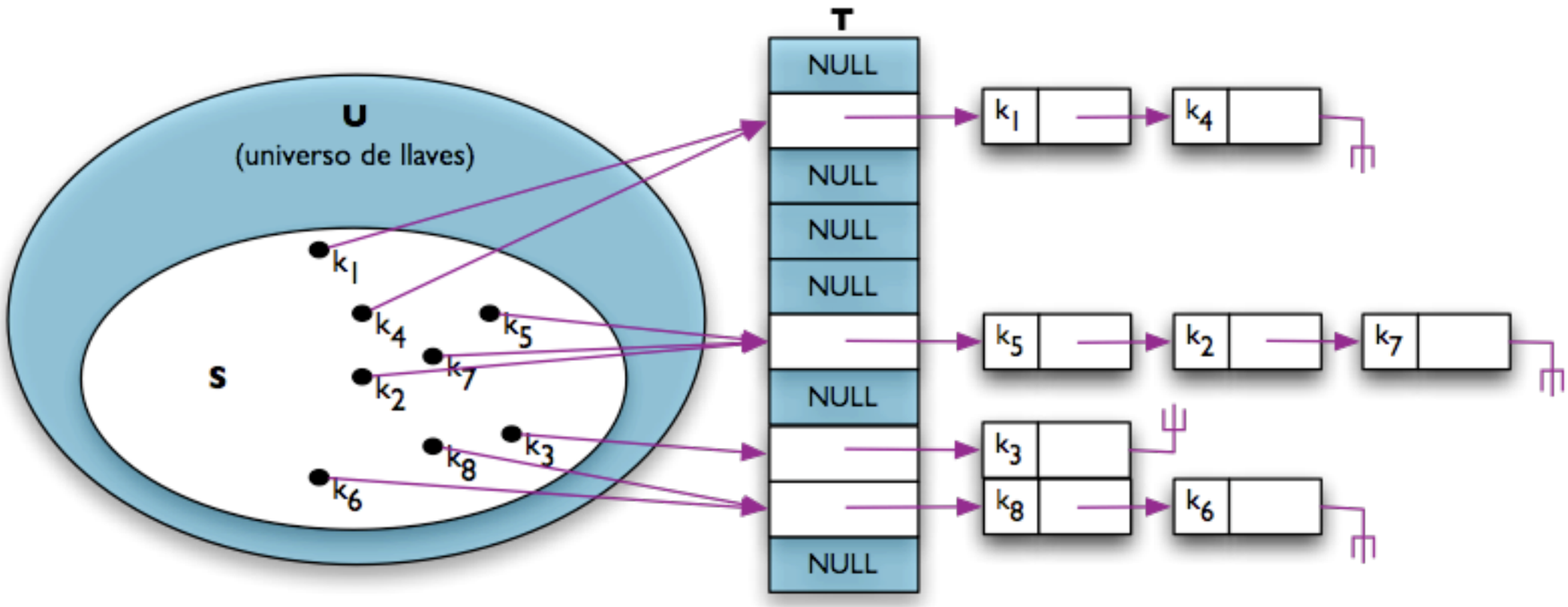
Tablas de hash: **resolviendo colisiones por encadenamiento**

- Idea: poner una liga a los elementos en el mismo campo haciendo una lista ligada.



Tablas de hash: **resolviendo colisiones por encadenamiento**

- Idea: poner una liga a los elementos en el mismo campo haciendo una lista ligada.



- El campo j contiene un apuntador al inicio de la lista de todos los elementos que se mapean (hash) a j.

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

CHAINED-HASH-INSERT(T, x)

- insertar x al inicio de la lista T[h(x.key)]

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

CHAINED-HASH-INSERT(T, x)

- insertar x al inicio de la lista T[h(x.key)]

- O(1) en el peor caso (supone que el elemento no ha sido introducido previamente)

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

CHAINED-HASH-INSERT(T, x)

- insertar x al inicio de la lista T[h(x.key)]

- O(1) en el peor caso (supone que el elemento no ha sido introducido previamente)

CHAINED-HASH-SEARCH(T, k)

- buscar un elemento con llave k en la lista T[h(k)]

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

CHAINED-HASH-INSERT(T, x)

- insertar x al inicio de la lista T[h(x.key)]

- O(1) en el peor caso (supone que el elemento no ha sido introducido previamente)

CHAINED-HASH-SEARCH(T, k)

- buscar un elemento con llave k en la lista T[h(k)]

- O(tl) proporcional al tamaño de la lista en el peor caso.

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

CHAINED-HASH-INSERT(T, x)

- insertar x al inicio de la lista T[h(x.key)]

- O(1) en el peor caso (supone que el elemento no ha sido introducido previamente)

CHAINED-HASH-SEARCH(T, k)

- buscar un elemento con llave k en la lista T[h(k)]

- O(tl) proporcional al tamaño de la lista en el peor caso.

CHAINED-HASH-DELETE(T, x)

- borrar x de la lista T[h(x.key)]

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Las operaciones de un diccionario se implementan fácilmente cuando las colisiones se resuelven por encadenamiento.

CHAINED-HASH-INSERT(T, x)

- insertar x al inicio de la lista T[h(x.key)]

- O(1) en el peor caso (supone que el elemento no ha sido introducido previamente)

CHAINED-HASH-SEARCH(T, k)

- buscar un elemento con llave k en la lista T[h(k)]

- O(tl) proporcional al tamaño de la lista en el peor caso.

CHAINED-HASH-DELETE(T, x)

- borrar x de la lista T[h(x.key)]

- O(tl) proporcional al tamaño de la lista en el peor caso.

Tablas de hash: resolviendo colisiones por encadenamiento

Tablas de hash: **resolviendo colisiones por encadenamiento**

- ¿Cómo se comporta el hashing con encadenamiento?¿Cuánto tiempo toma buscar un elemento con una llave dada?

Tablas de hash: **resolviendo colisiones por encadenamiento**

- ¿Cómo se comporta el hashing con encadenamiento?¿Cuánto tiempo toma buscar un elemento con una llave dada?
- Factor de carga (load factor) α de **T**: dada una tabla de hash **T** con m campos que almacena n elementos,

Tablas de hash: **resolviendo colisiones por encadenamiento**

- ¿Cómo se comporta el hashing con encadenamiento?¿Cuánto tiempo toma buscar un elemento con una llave dada?
- Factor de carga (load factor) α de **T**: dada una tabla de hash **T** con m campos que almacena n elementos,

$$\alpha = n/m$$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- ¿Cómo se comporta el hashing con encadenamiento?¿Cuánto tiempo toma buscar un elemento con una llave dada?
- Factor de carga (load factor) α de **T**: dada una tabla de hash **T** con m campos que almacena n elementos,

$$\alpha = n/m$$

- Esto es el número promedio de elementos almacenados en una cadena (en un campo).

Tablas de hash: **resolviendo colisiones por encadenamiento**

- ¿Cómo se comporta el hashing con encadenamiento?¿Cuánto tiempo toma buscar un elemento con una llave dada?
- Factor de carga (load factor) α de **T**: dada una tabla de hash **T** con m campos que almacena n elementos,

$$\alpha = n/m$$

- Esto es el número promedio de elementos almacenados en una cadena (en un campo).
- El análisis se realizará en términos de α que puede ser mayor, menor o igual a 1.

Tablas de hash: **resolviendo colisiones por encadenamiento**

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: $\Theta(n)$ más el tiempo de cálcular la función de hash.

$\mathcal{H}$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: $\Theta(n)$ más el tiempo de cálcular la función de hash.
- Caso promedio:

$$\mathcal{H}$$

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: $\Theta(n)$ más el tiempo de cálcular la función de hash.
- Caso promedio:
  - depende que tan bien distribuya la función $\mathcal{H}$ el conjunto de llaves n en los m campos.

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: $\Theta(n)$ más el tiempo de cálcular la función de hash.
- Caso promedio:
  - depende que tan bien distribuya la función $\mathcal{H}$ el conjunto de llaves n en los m campos.
  - Suposiciones: función de hash ideal

Tablas de hash: **resolviendo colisiones por encadenamiento**

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: $\Theta(n)$ más el tiempo de cálcular la función de hash.
- Caso promedio:
  - depende que tan bien distribuya la función $\mathcal{H}$ el conjunto de llaves n en los m campos.
  - Suposiciones: función de hash ideal
    - distribución uniforme (misma probabilidad de mapearse a cada campo sin importar el campo al que se mapearon los valores anteriores): hashing simple uniforme.

Tablas de hash: resolviendo colisiones por encadenamiento

cuál es la probabilidad de que la llave se mapee al campo 15?

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: $\Theta(n)$ más el tiempo de cálcular la función de hash.
- Caso promedio:
  - depende que tan bien distribuya la función $\mathcal{H}$ el conjunto de llaves n en los m campos.
  - Suposiciones: función de hash ideal
    - distribución uniforme (misma probabilidad de mapearse a cada campo sin importar el campo al que se mapearon los valores anteriores): hashing simple uniforme.

Tablas de hash: resolviendo colisiones por encadenamiento

- Peor caso:
  - todas las n llaves se mapean al mismo campo, creando una lista de tamaño n.
  - lista ligada más elaborada.
  - tiempo de ejecución: Θ(n) más el tiempo de cálcular la función de hash.

cuál es la probabilidad de que la llave se mapee al campo 15?

1/m

- Caso promedio:
  - depende que tan bien distribuya la función $\mathcal{H}$ el conjunto de llaves n en los m campos.
  - Suposiciones: función de hash ideal
    - distribución uniforme (misma probabilidad de mapearse a cada campo sin importar el campo al que se mapearon los valores anteriores): hashing simple uniforme.

Tablas de hash: resolviendo colisiones por encadenamiento

Para $j = 0, 1, \ldots, m-1$, llamaremos n_j al largo de la lista almacenada en el campo T[j] de manera que $n = n_0 + n_1 + \ldots + n_{m-1}$.

Tablas de hash: resolviendo colisiones por encadenamiento

Para $j = 0, 1, \ldots, m-1$, llamaremos n_j al largo de la lista almacenada en el campo T[j] de manera que $n = n_0 + n_1 + \ldots + n_{m-1}$.

y el valor promedio de n_j es $E[n_j] = \alpha = n/m$.

Tablas de hash: resolviendo colisiones por encadenamiento

Para $j = 0, 1, \ldots, m-1$, llamaremos n_j al largo de la lista almacenada en el campo T[j] de manera que $n = n_0 + n_1 + \ldots + n_{m-1}$.

y el valor promedio de n_j es $E[n_j] = \alpha = n/m$.

- Dividimos en dos casos, el de búsqueda exitosa y búsqueda no-exitosa.

Tablas de hash: resolviendo colisiones por encadenamiento

Para $j = 0, 1, \ldots, m-1$, llamaremos n_j al largo de la lista almacenada en el campo T[j] de manera que $n = n_0 + n_1 + \ldots + n_{m-1}$.

y el valor promedio de n_j es $E[n_j] = \alpha = n/m$.

- Dividimos en dos casos, el de búsqueda exitosa y búsqueda no-exitosa.
- ¿Cuál es el tiempo de ejecución promedio de una búsqueda no-exitosa?