Algoritmos y estructuras de datos

TAD. Tablas de hash. Conjuntos Disyuntos.

CEIS

Escuela Colombiana de Ingeniería

2024-2

Agenda

1. Diccionarios
2. Conjuntos disyuntos
3. Aspectos finales
   - Ejercicios

Agenda

Diccionario

Manzanas	$ 0.79
Queso	$ 1.99
Huevos	$ 2.49

$O(n)$

A Z

Huevos	$ 2.49
Manzanas	$ 0.79
Queso	$ 1.99

$O(\log n)$

Búsqueda binaria

Diversas aplicaciones requieren de un conjunto dinámico que soporte efectivamente las operaciones de INSERCIÓN, BÚSQUEDA y ELIMINACIÓN.

Diccionario

Un diccionario es una estructura de datos que soporta las operaciones de inserción, búsqueda y eliminación de forma efectiva.

- $INSERT(S, x)$ Adiciona el elemento apuntado por x al conjunto S
- $SEARCH(S, k)$ Dado un conjunto S y un valor clave k retorna un apuntador al elemento x de S tal que $x.key = k$, si existe. Sino retorna NIL.
- $DELETE(S, x)$ Dado un apuntador x a un elemento del conjunto S, elimina x de S.

(Queso, 1.99)

(Huevos, 2.49)

(Manzanas, 0.79)

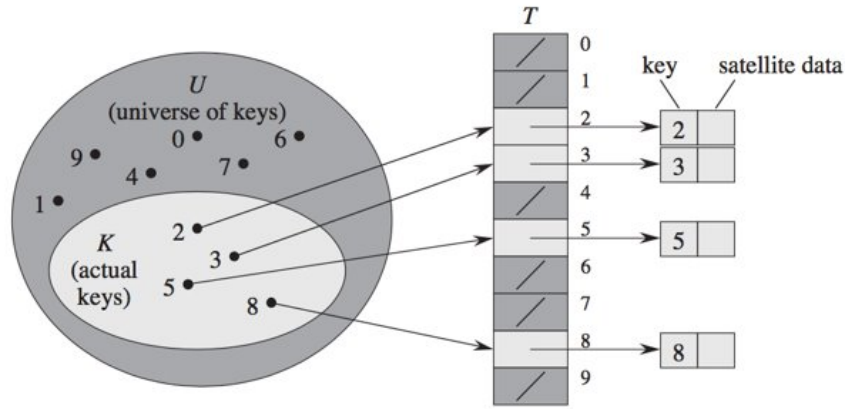
Diccionario

Direccionamiento directo

El direccionamiento directo es una técnica que funciona muy bien cuando el universo U de llaves es razonablemente pequeño.

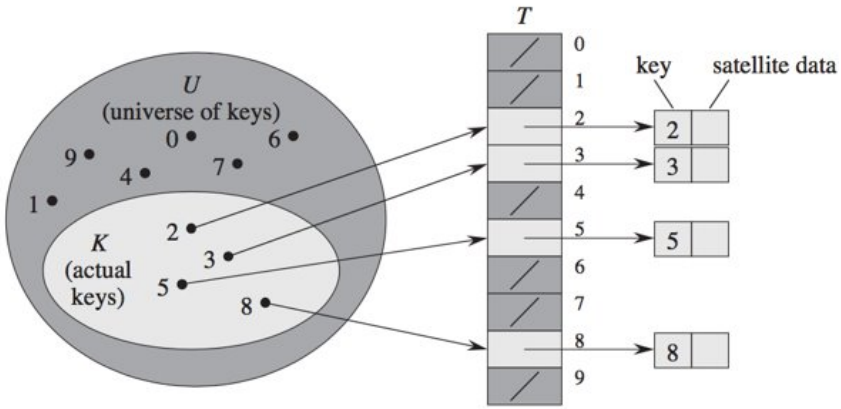
Supongan que una aplicación necesita de un conjunto dinámico en el que cada elemento tiene una llave del universo $U = 0, 1, \ldots, m - 1$, donde m es no tan grande. Se asume que no hay dos elementos con la misma llave.

Para representar el conjunto dinámico se usa un arreglo, o tabla de direccionamiento directo, definida como $T[0..m - 1]$, en donde cada posición o slot, corresponde a una llave en el universo U.



Diccionario

Direccionamiento directo



El slot k apunta a un elemento en el conjunto con llave k. Si el conjunto no contiene un elemento con llave k, entonces $T[k] = NIL$

DIRECT-ADDRESS-SEARCH(T, k)

1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)

1 $T[x.key] = x$

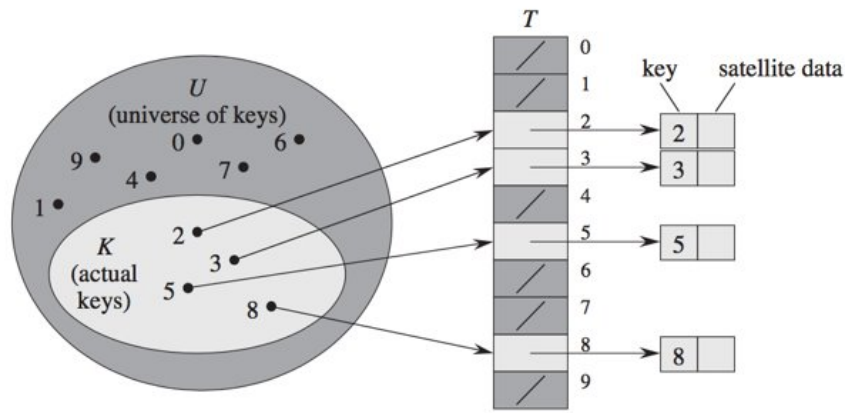
DIRECT-ADDRESS-DELETE(T, x)

1 $T[x.key] = \text{NIL}$

Diccionario

Direccionamiento directo

La falencia del direccionamiento directo es bastante obvio, si el universo U es grande, almacenar una tabla T de tamaño $|U|$ puede ser impráctico, o incluso imposible.



Diccionario

Tabla de hash

Una función hash h mapea el universo U de llaves en los slots de una tabla de hash $T[0..m-1]$

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

Donde el tamaño de m de la tabla de hash es típicamente mucho menor que $|U|$.

$h(k)$ es el valor del hash de la llave k.

$T[h(k)]$ contiene el elemento de llave k

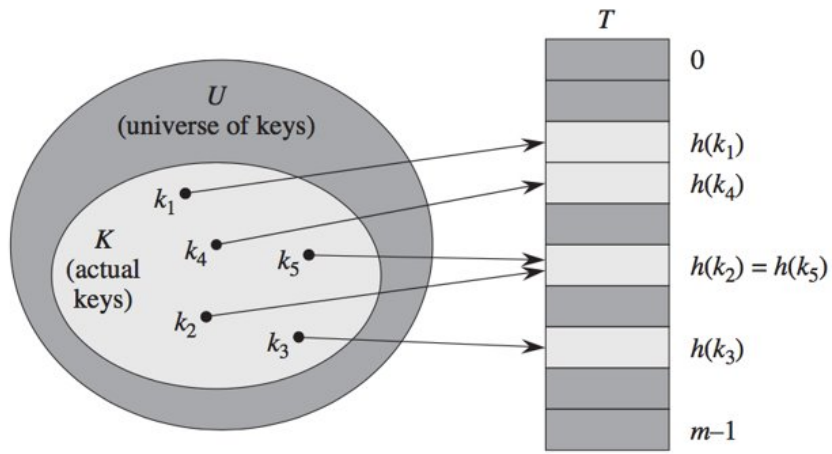


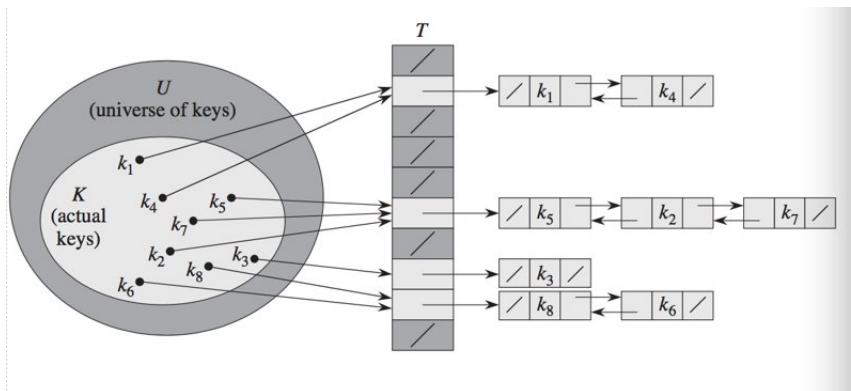
Tabla de hash

El único inconveniente es que puede haber dos llaves que tengan el mismo hash, esta situación se denomina como una colisión.

Lo ideal seria evitar las colisiones pero como $|U| > m$, al menos dos llaves tendrán el mismo valor de hash.

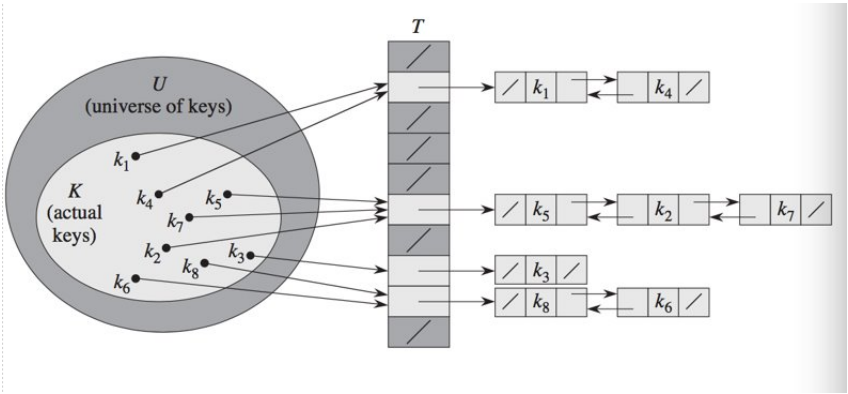
Una forma de resolver estas colisiones es con el encadenamiento.

Con este método, todos los elementos con el mismo hash se ponen en la misma lista encadenada. El slot j contiene un apuntador a la cabeza de la lista de todos los elementos almacenados con hash j.



Diccionario

Tabla hash



CHAINED-HASH-INSERT(T, x)

1 insert x at the head of list $T[h(x.key)]$

CHAINED-HASH-SEARCH(T, k)

1 search for an element with key k in list $T[h(k)]$

CHAINED-HASH-DELETE(T, x)

1 delete x from the list $T[h(x.key)]$

Diccionario

Tabla hash

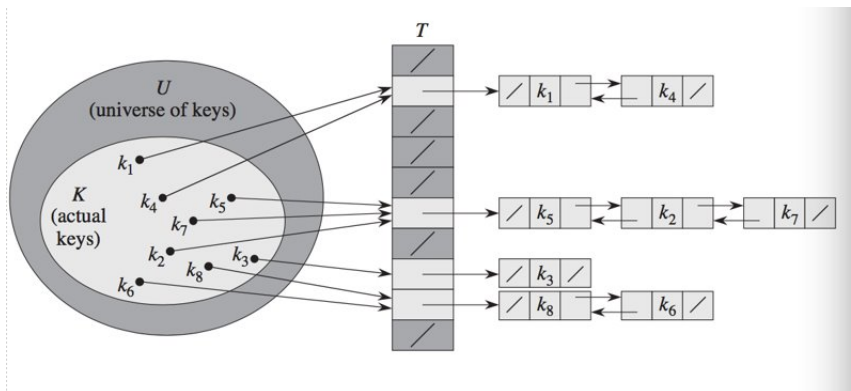
La complejidad de búsqueda es:

- En el peor de los casos: $\Theta(n)$
- En el caso promedio: $\Theta(1+\alpha)$

  $\alpha = n/m$, porque en una tabla hash T con m slots que almacena n elementos

  α se llama factor de carga

En una buena función hash "cada llave es igualmente probable que su hash apunte a cualquiera de los m slots, independientemente de a donde cualquier otra llave haya apuntado."



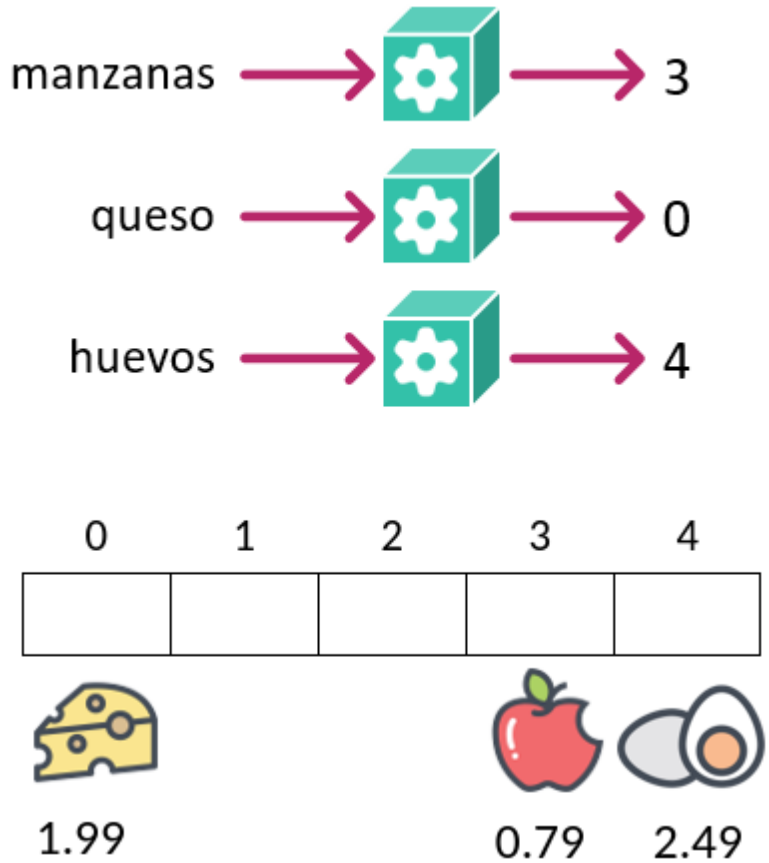
Diccionario

Funciones hash

Muchas funciones de hash asumen que el universo de llaves es el conjunto de números naturales $N = 0, 1, 2, \ldots$.
Si las llaves no son números naturales, podemos crear una forma para interpretarlos como números naturales.

- Método de división:
  $h(k) = k \ mod \ m$
- Método de multiplicación:
  $h(k) = m(k \ A \ mod \ 1)$

  Para una A en el rango $0 < A < 1$

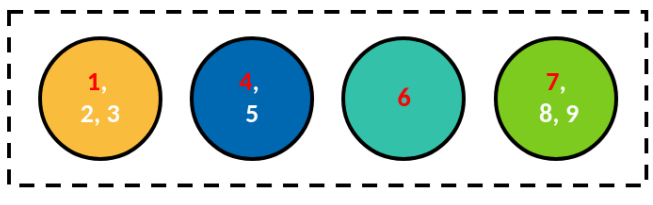


Agenda

Conjunto disyunto

Un **conjunto disyunto** es una estructura de datos que mantiene una colección $S = S_1, S_2, .., S_k$ de diferentes conjuntos dinámicos disyuntos (sin elementos en común).
Cada conjunto se identifica con un representante, el cuál es un miembro del conjunto.
Las operaciones básicas de este TAD son:

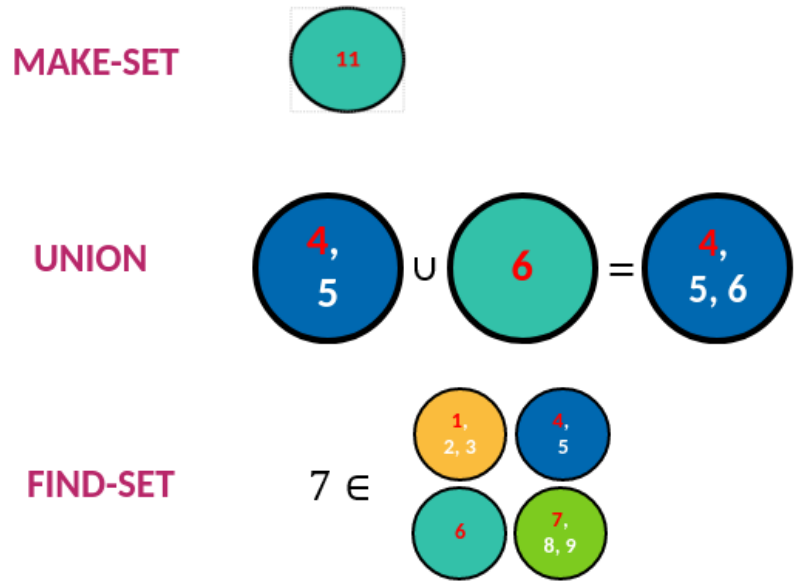
- MAKE-SET(X)
- UNION(X, Y)
- FIND-SET(X)



Conjunto disyunto

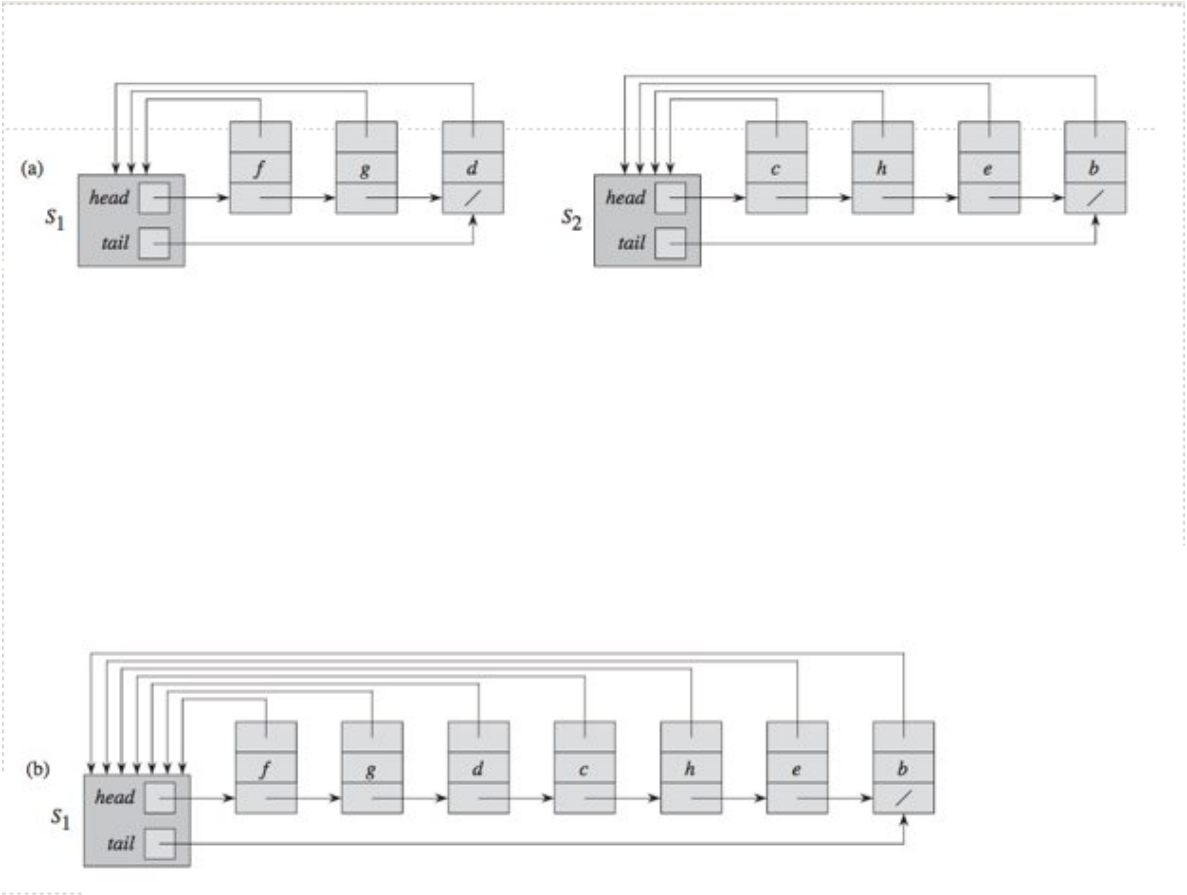
Las operaciones básicas de este TAD son:

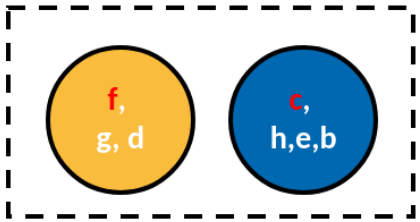
- MAKE-SET(X): Crea un nuevo conjunto cuyo único miembro y por lo tanto su representante es x. Al ser conjuntos disyuntos, x no debe estar en algún otro conjunto
- UNION(X, Y): Une los conjuntos dinámicos que contienen los objetos x y y, en un nuevo conjunto que es la unión de los dos conjuntos.
- FIND-SET(X): Retorna un puntero al representante del conjunto que contiene a x

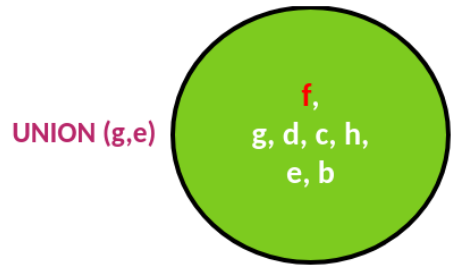


Conjunto disyunto

Representación: como lista

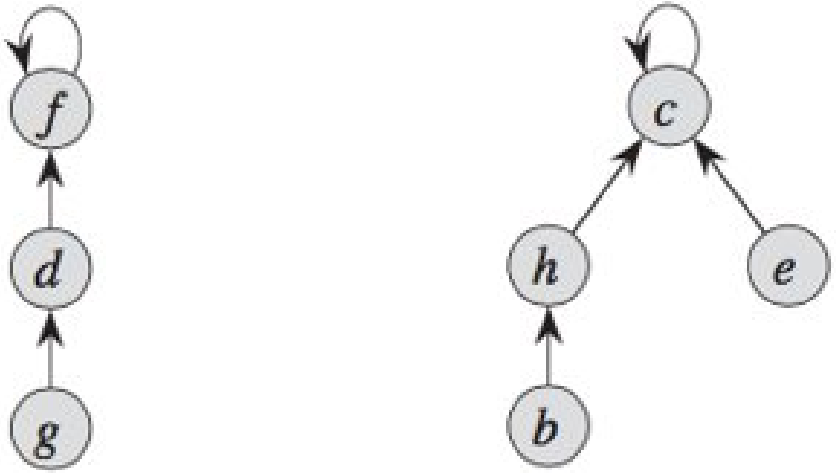


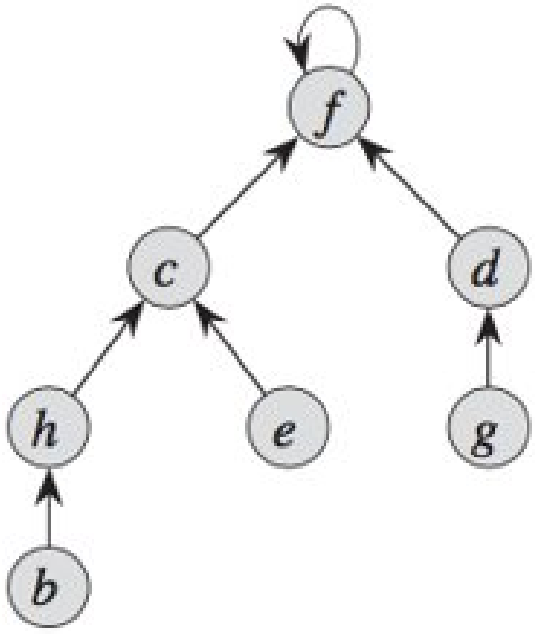


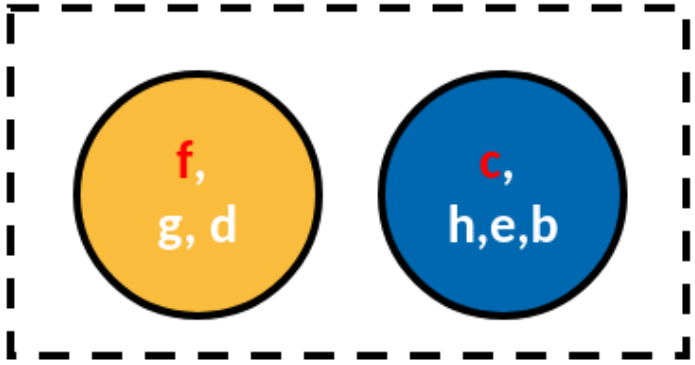


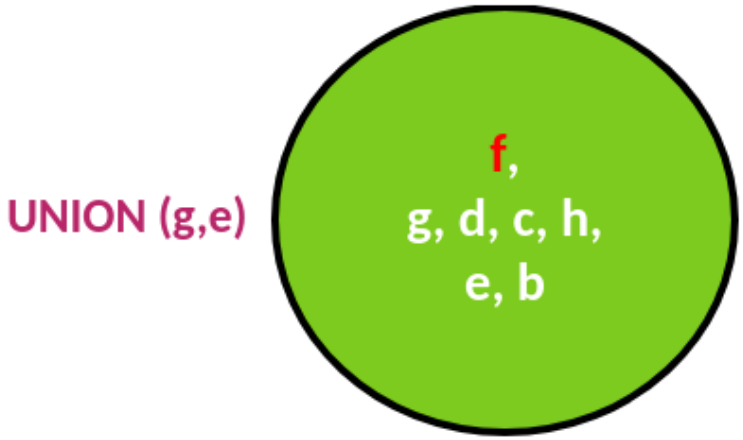
Conjunto disyunto

Representación: como bosque







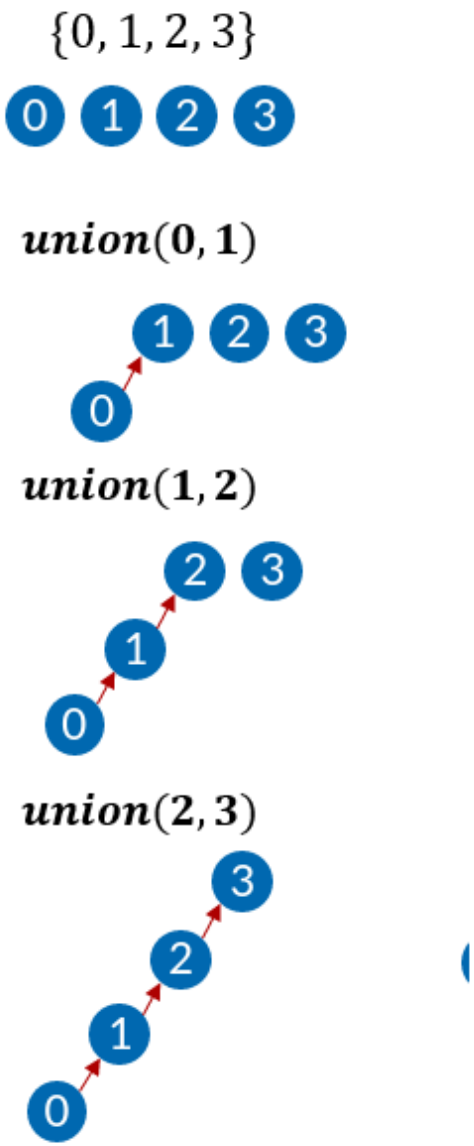


Conjunto disyunto

Representación: como bosque

Una secuencia de n-1 operaciones de unión puede crear un árbol que es una cadena lineal de n nodos. Usando dos heurísticas se puede evitar esto.

- **Unión por rank**
- **Compresión de ruta**



Conjunto_disyunto

Representación: como bosque

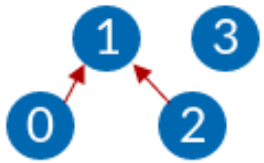
Una secuencia de n-1 operaciones de unión puede crear un árbol que es una cadena lineal de n nodos. Usando dos heurísticas se puede evitar esto.

- **Unión por rank** :La raíz del árbol con menos nodos apuntara a la raíz del árbol con mas nodos. Para cada nodo, se mantiene una propiedad denominada *rank*, el cual es el limite superior de la altura del nodo. En una unión por *rank*, la raíz con el menor *rank* apuntara a la que tenga un mayor *rank*.
- **Compresión de ruta**

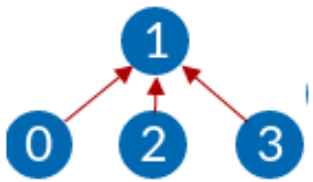
$\{0, 1, 2, 3\}$

$union(0, 1)$

$union(1, 2)$



$union(2, 3)$



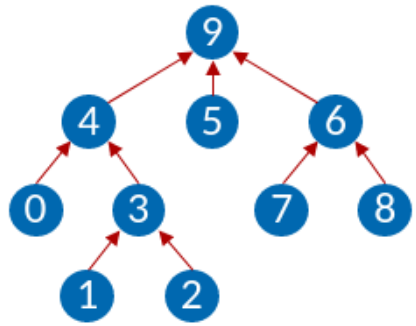
Conjunto disyunto

Representación: como bosque

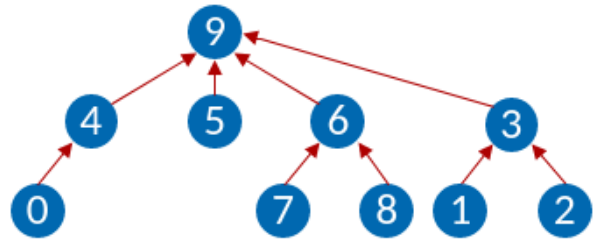
Una secuencia de n-1 operaciones de unión puede crear un árbol que es una cadena lineal de n nodos. Usando dos heurísticas se puede evitar esto.

- **Unión por rank**
- **Compresión de ruta** :Cada nodo en la ruta apunta directamente a la raíz. Esta compresión se hace durante la operación Find-Set y no modifica el *rank* del árbol.

$\{0, 1, 2, \ldots, 9\}$

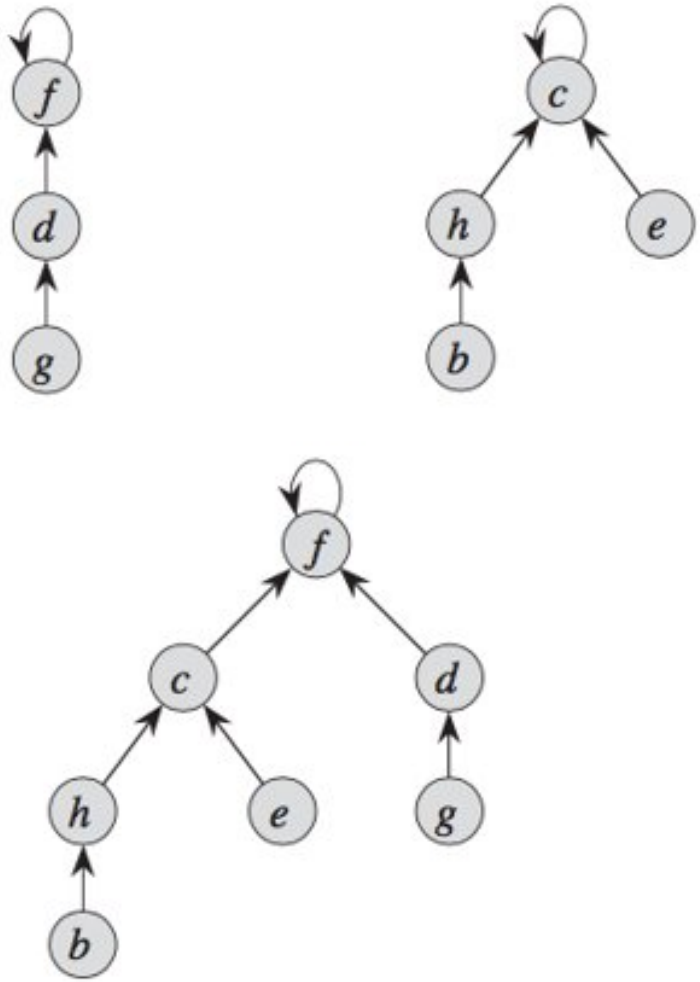


$\boldsymbol{find(3) = 9}$



Conjunto disyunto

Representación: como bosque

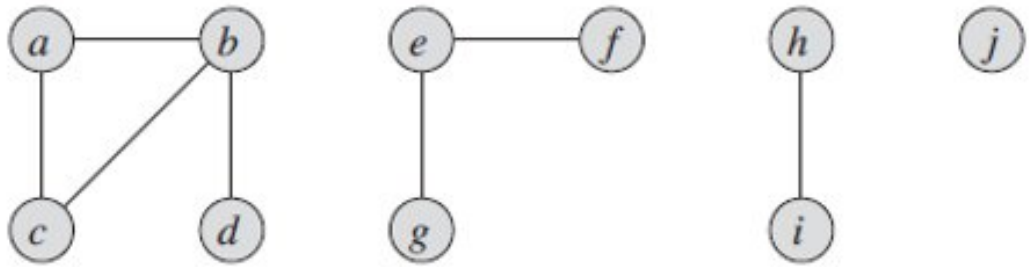



```
MAKE-SET(x)
1  x.p = x
2  x.rank = 0

UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if x.rank > y.rank
2       y.p = x
3  else x.p = y
4       if x.rank == y.rank
5            y.rank = y.rank + 1
```

```
FIND-SET(x)
1  if x ≠ x.p
2       x.p = FIND-SET(x.p)
3  return x.p
```

Conjunto disyunto

Conexiones de un grafo no dirigido



Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

Agenda

Ejercicios

1. Suponga que tiene las siguientes 4 funciones hash sobre textos:
   a. Devuelve “1” para todas las entradas
   b. Usa la longitud del texto como índice
   c. Usa el primer carácter del texto como índice, tal que todos los textos que empiezan con *a* van en la misma posición, y así con los demás
   d. Transforma cada letra a un número primo: a = 2, b = 3, c = 5, d = 7, … El resultado de la función es el residuo de dividir la suma de todos los caracteres transformados a primos entre la longitud del hash. Para un hash de longitud 10 y la palabra “bag”, el índice sería 3 + 2 + 17 % 10 = 22 % 10 = 2

   Para cada uno de los ejemplos siguientes, ¿cuáles de las funciones anteriores provee una buena distribución? Suponga que la longitud del hash es 10:
   i. Una agenda telefónica donde las llaves son los nombres y los valores son los números telefónicos. Los nombres son: Esther, Ben, Bob y Dan.
   ii. Un mapa de tamaño de la batería a poder. Los tamaños son A, AA, AAA y AAAA
   iii. Un mapa de libros a autores. Los libros son *Maus*, *Fun Home* y *Watchmen*.
2. Demuestre que pasaría con un *hash table* al insertar las llaves $5, 28, 19, 15, 20, 33, 12, 17, 10$, donde las colisiones se resuelven por encadenamiento. Suponga que el tamaño del hash es 9, y que la función hash es: $h(k) = k \ mod \ 9$