Introducción al análisis de algoritmos

MAT-151

Dr. Alonso Ramirez Manzanares
CIMAT A.C.

e-mail: alram@cimat.mx
web: http://www.cimat.mx/~alram/comp_algo/

Ejemplo 1: búsqueda en el directorio (1)

- Supongamos que queremos encontrar el número de teléfono de un amigo en el directorio. Muy probablemente utilicemos variantes del algoritmo de **búsqueda binaria *(BS)*** a menos que sean muy pacientes y utilicen **búsqueda exhaustiva *(ES)***.

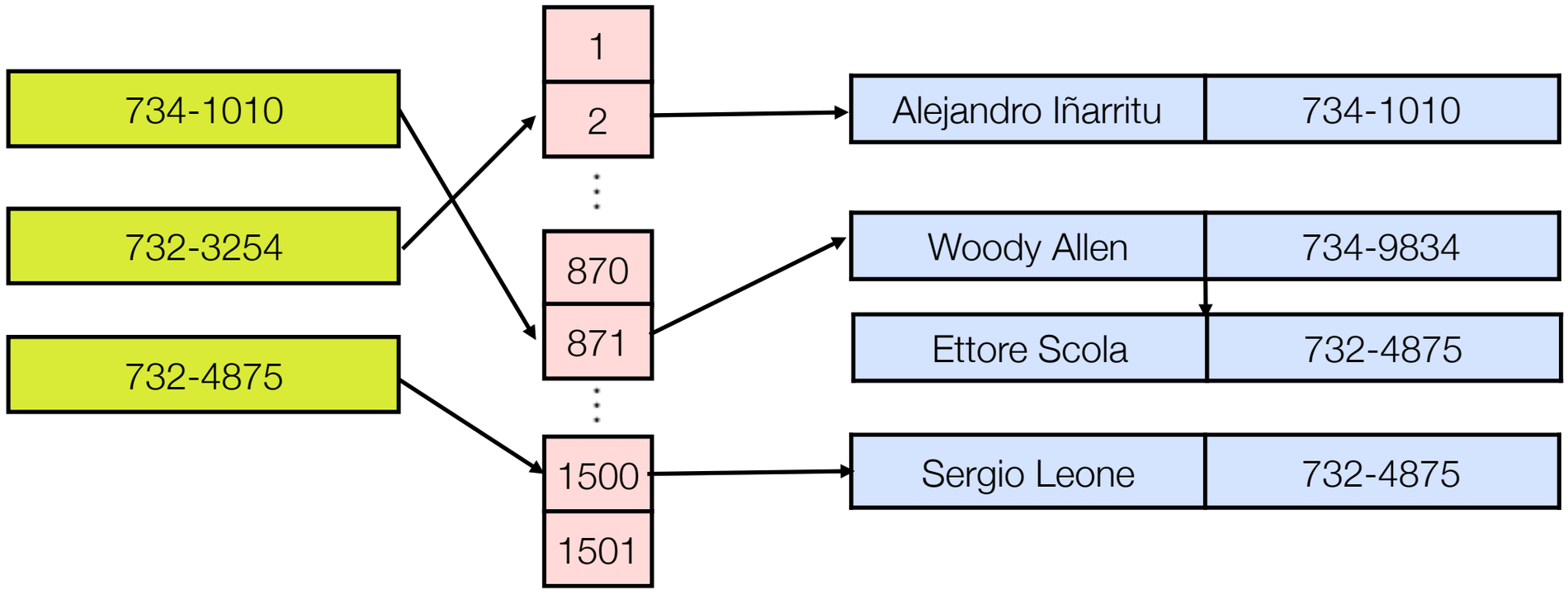
- La búsqueda binaria es *exponencialmente* (incomparablemente) más rápida que ES. Si el directorio tiene digamos, 1 millón de entradas y cada entrada toma un segundo, BS tomaría 20 segundos mientras ES tomaría alrededor de 5 días.

- La clave de la búsqueda binaria es tener una ***lista ordenada!***

- ...¿y si olvidé el nombre de la persona y quiero buscar por su número de teléfono?

Ejemplo 1: búsqueda en el directorio (2)

- Utilizar una *tabla de hash*.



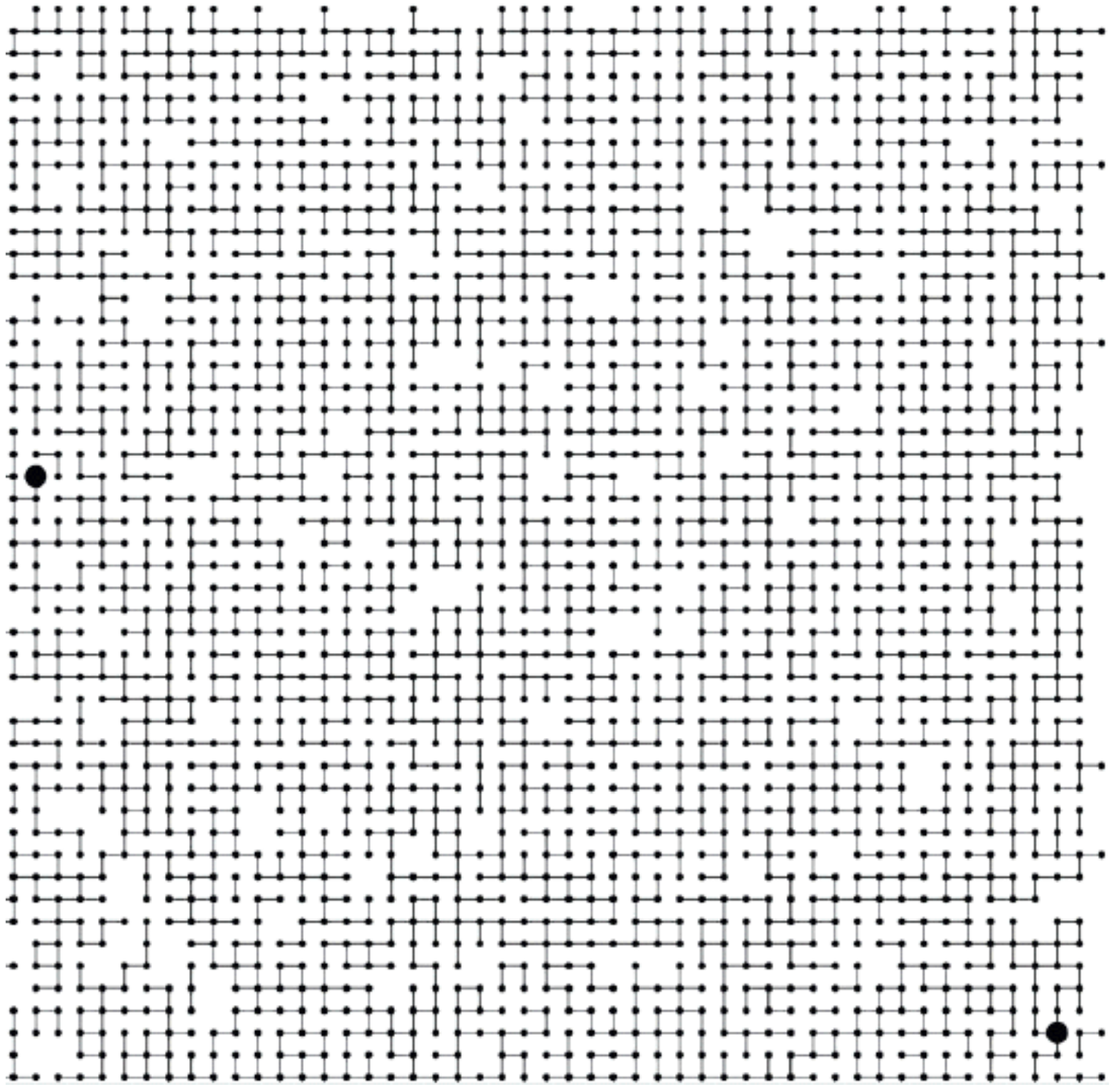
Ejemplo 2 : Problema de conectividad (1)

- Nos dan una secuencia de pares de enteros, donde cada entero representa un objeto de algún tipo y tenemos que interpretar el par *p*-*q* como "*p* está conectado con *q*".

- Relación transitiva: Si "*p* está conectado con *q*" y "*q* está con *r*", entonces "*p* está conectado con *r*".

- La meta es escribir un programa que filtre los pares externos al conjunto (guardar solo las nuevas conexiones mínimas): Para un par entrada *p*-*q*, la salida deberá ser el par solo si los pares revisados hasta entonces no implican que *p* está conectado a *q*. Si los pares visitados implican que *p* esta conectado a *q*, se debe ignorar el par *p*-*q* y continuaremos al siguiente par.

Esto es análogo saber si los 2 puntos están conectados:



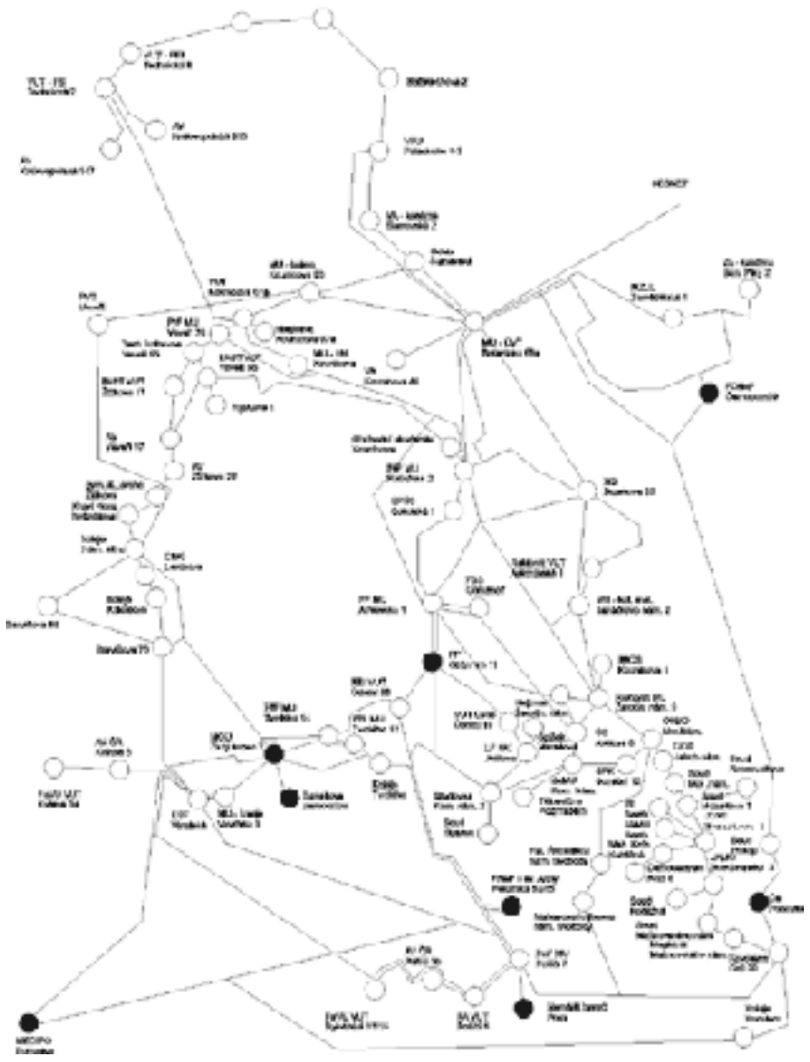
Ejemplo 2 : Problema de conectividad (2)

Dados N nodos de índices 0 a N-1

par entrada	salida	conexión
3-4	3-4	
4-9	4-9	
8-0	8-0	
2-3	2-3	
5-6	5-6	
2-9		2-3-4-9
5-9	5-9	
7-3	7-3	
4-8	4-8	
5-6		5-6
0-2		0-8-4-3-2
6-1	6-1	

Ojo, la salida nunca tendrá más de N-1 pares

Ejemplo 2:Problema de conectividad (aplicaciones)



- redes de computadoras: se pueden usar las conexiones existentes o hay que hacer una nueva conexión?

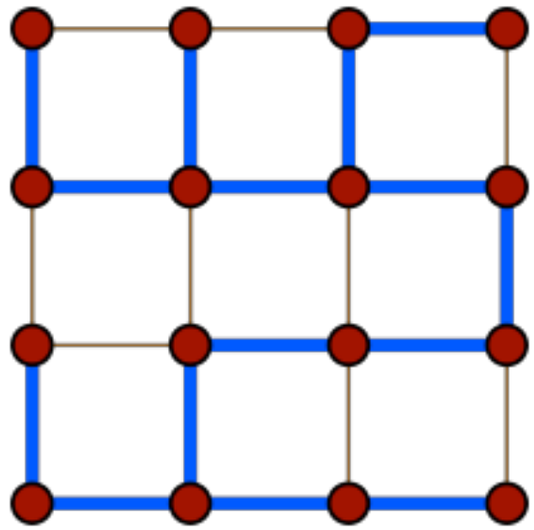
- redes eléctricas, búsqueda de caminos

- si el número de entradas es pequeño cualquier algoritmo es razonable, si el número de entradas son millones de enteros...

- ¿Cómo podemos arreglarnos para determinar si dos pares están conectados?

Ejemplo 2: Problema de conectividad (4)

- mientras más requiramos de un algoritmo, más tiempo y espacio se requieren para completar la tarea.
  - ¿están conectados p y q?
  - ¿son suficientes las conexiones existentes para que p y q estén conectados?
  - encuentra el camino para llegar de p a q
- el conjunto de pares en la entrada se llaman **grafo (informalmente es un conjunto de nodos unidos por aristas que determinar relaciones binarias entre elementos de conjuntos),** el conjunto de pares de salida se conoce como **árbol de recubrimiento (*spanning tree)*** de ese grafo.



1. Especificar el problema claramente.
2. Encontrar las operaciones necesarias para implementarlo.

Ejemplo 2: Problema de conectividad (5)

- Operaciones fundamentales para un algoritmo de conectividad (útiles para resolver tareas similares sobre la misma estructura del grafo)
  - al tener un par nuevo: determinar si representa una nueva conexión.
  - Si es así, incorporar la información que la conexión ha sido encontrada y actualizar la estructura del grafo.
  - Operaciones:
    - ***Encontrar*** el conjunto que contiene un elemento particular.
    - Reemplazar los conjuntos que contienen dos elementos dados por su ***unión***.
- el problema de conectividad puede resolverse realizando estas dos operaciones. Cada conjunto se conoce como ***componentes conectados.***

Ejemplo 2: Problema de conectividad (6)

- ¡Implementar un algoritmo simple que solucione el problema! - servirá como base para evaluar las características del desempeño. Nos interesa la ***eficiencia*** pero sobre todo nos interesa una solución ***correcta*** al problema.
- **Idea 1:** guardar todos los pares de entrada, escribir una función para recorrerlos y ver si están conectados.
  - si el número de pares entrada es muy grande, la memoria necesaria podría superar los recursos de la máquina.
  - no conocemos ahora un método inmediato para determinar si dos objetos están conectados, incluso si se pudieran almacenar todos.
  - utilizaremos **arreglos** de enteros, donde cada objeto a conectar será un elemento del arreglo: si necesitamos 1000 enteros, el arreglo será id[1000]; y nos referiremos al entero i, escribiendo id[i] para $0 \leq i < N$

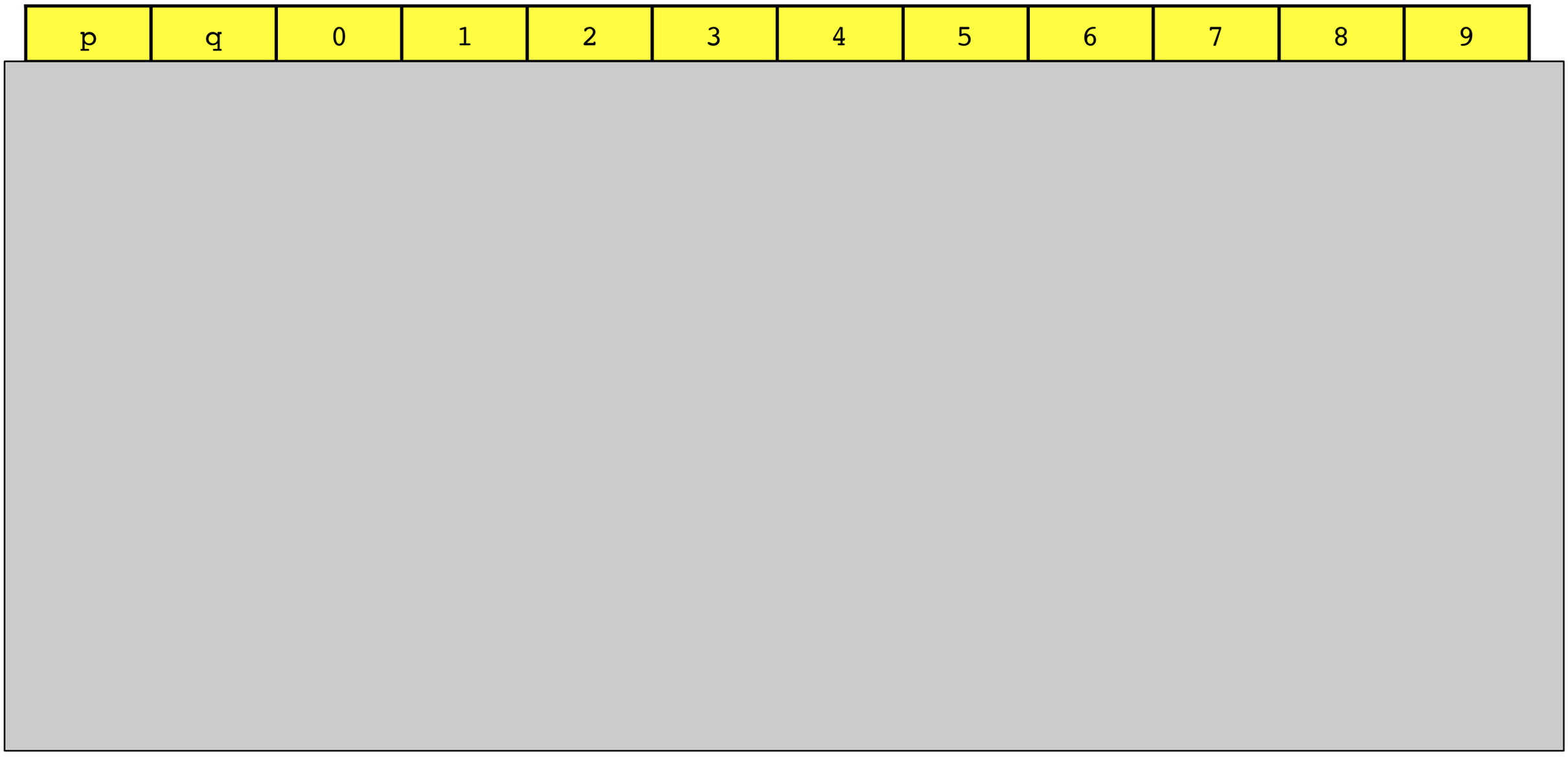
Problema de conectividad: algoritmo *quick-find*

- *p* y *q* están conectados si y solo si los elementos *p* y *q* del arreglo son iguales.
- cuando procesamos cada par *p,q*, cambiamos todos los elementos que valgan id[p] por el valor de id[q].

```
#include <iostream.h>
using namespace std;


static const int N = 10000;
```

```
int main()
{
        int i, p, q, id[N];
        for (i=0; i<N; i++)  // initialization
                id[i] = i;
        while ( cin >> p >> q ){
                int t = id[p];
                if (t == id[q]) continue; // quick find
                for ( i=0; i<N; i++) // union
                        if (id[i] == t ) id[i]=id[q];
                cout << " " << p << " " << q << endl;
      }
}
```

Problema de conectividad: algoritmo *quick-find*



Problema de conectividad: algoritmo *quick-find*

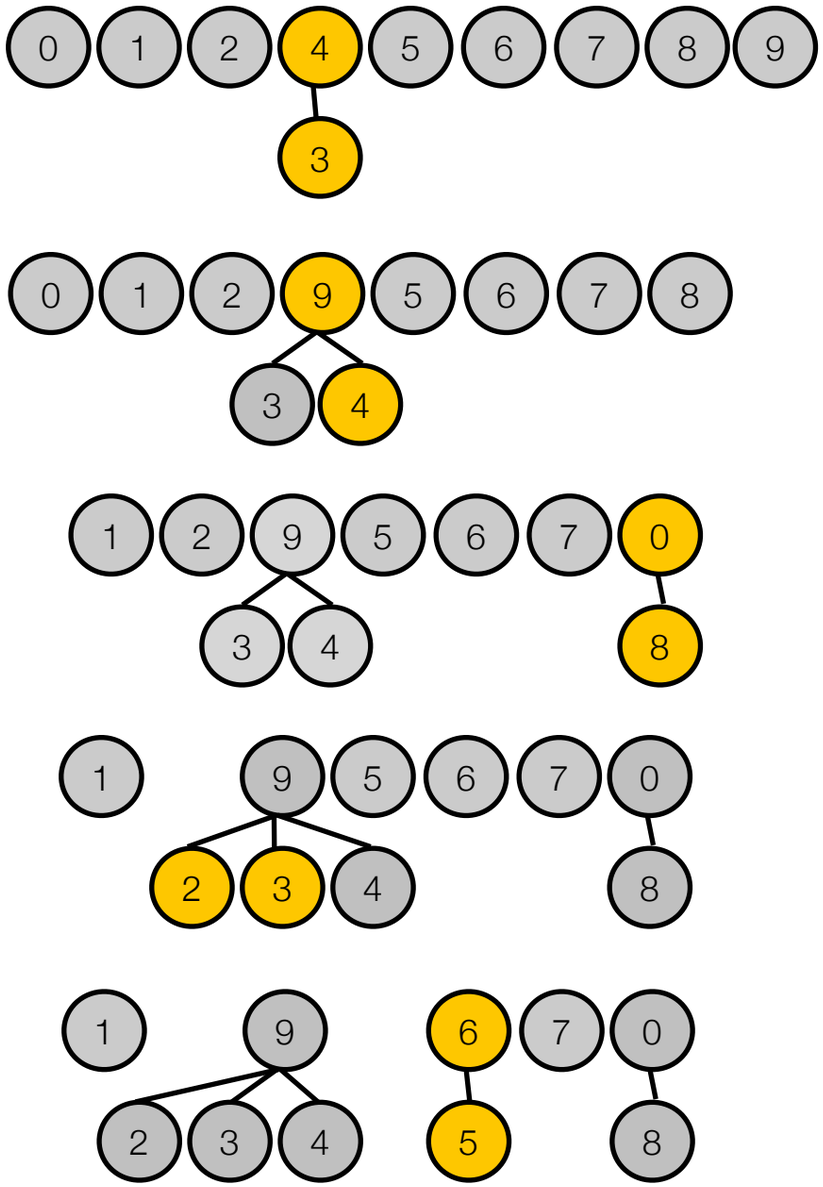
- Para implementar la operación *find* solo tenemos que probar si id[p] es igual a id[q].

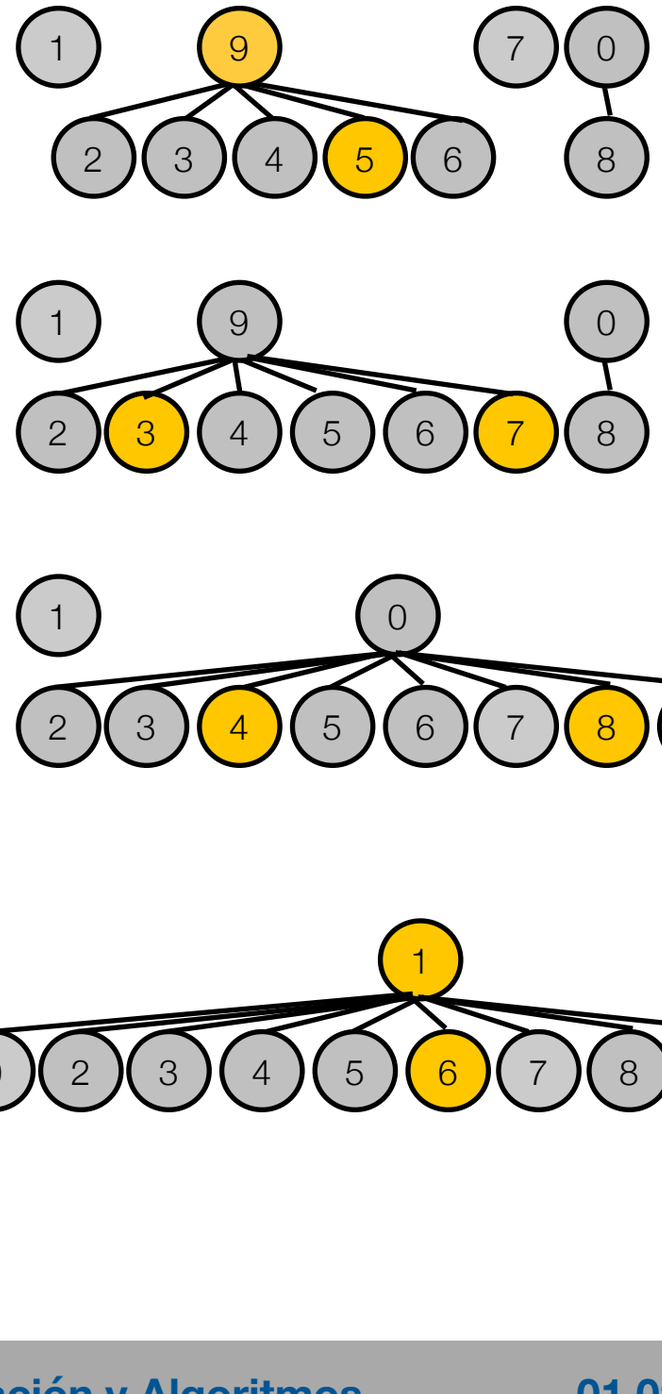
- El algoritmo *quick-find* ejecuta *MN* instrucciones para resolver un problema de conectividad, donde *N* es el numero de objetos que involucran *M* operaciones de unión.
    - Para M operaciones de unión, hay que iterar el ciclo *for* de N veces. Cada iteración requiere al menos una instrucción (verificar si la iteración se terminó)

- Si el número de objetos es muy grande el problema no se puede realizar de esta forma en una computadora moderna.

Problema de conectividad: *quick-find* (árbol)

p	q
3	4
4	9
8	0
2	3
5	6
2	9
5	9
7	3
4	8
5	6
0	2
6	1



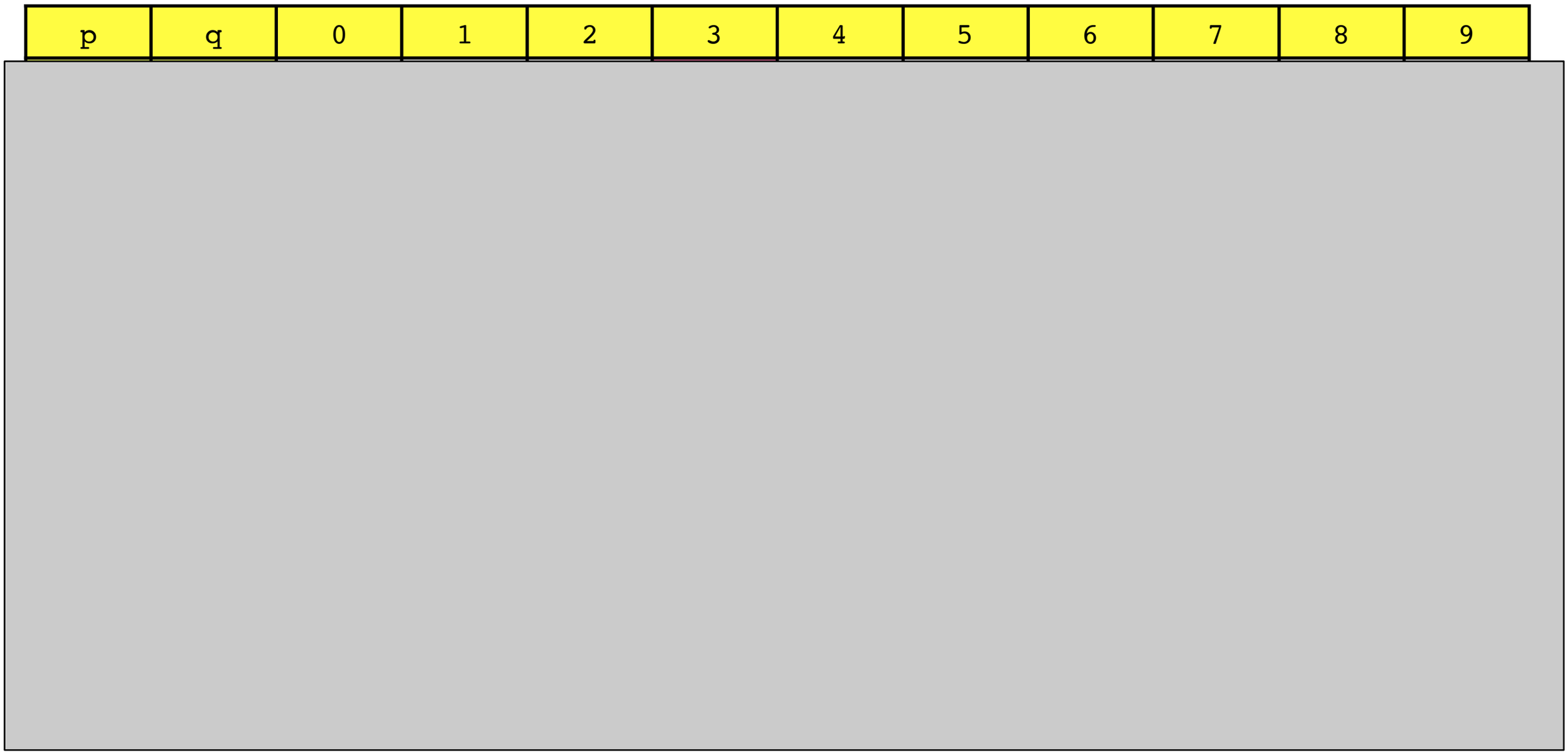


Problema de conectividad: algoritmo *quick-union*

- algoritmo basado también en un arreglo indexado por el nombre del objeto.
- cada objeto apunta a otro objeto del mismo componente conectado, en una estructura que no tiene ciclos.
- para determinar si dos objetos están en el mismo componente, hay que seguir los apuntadores hasta llegar al que apunte a si mismo.

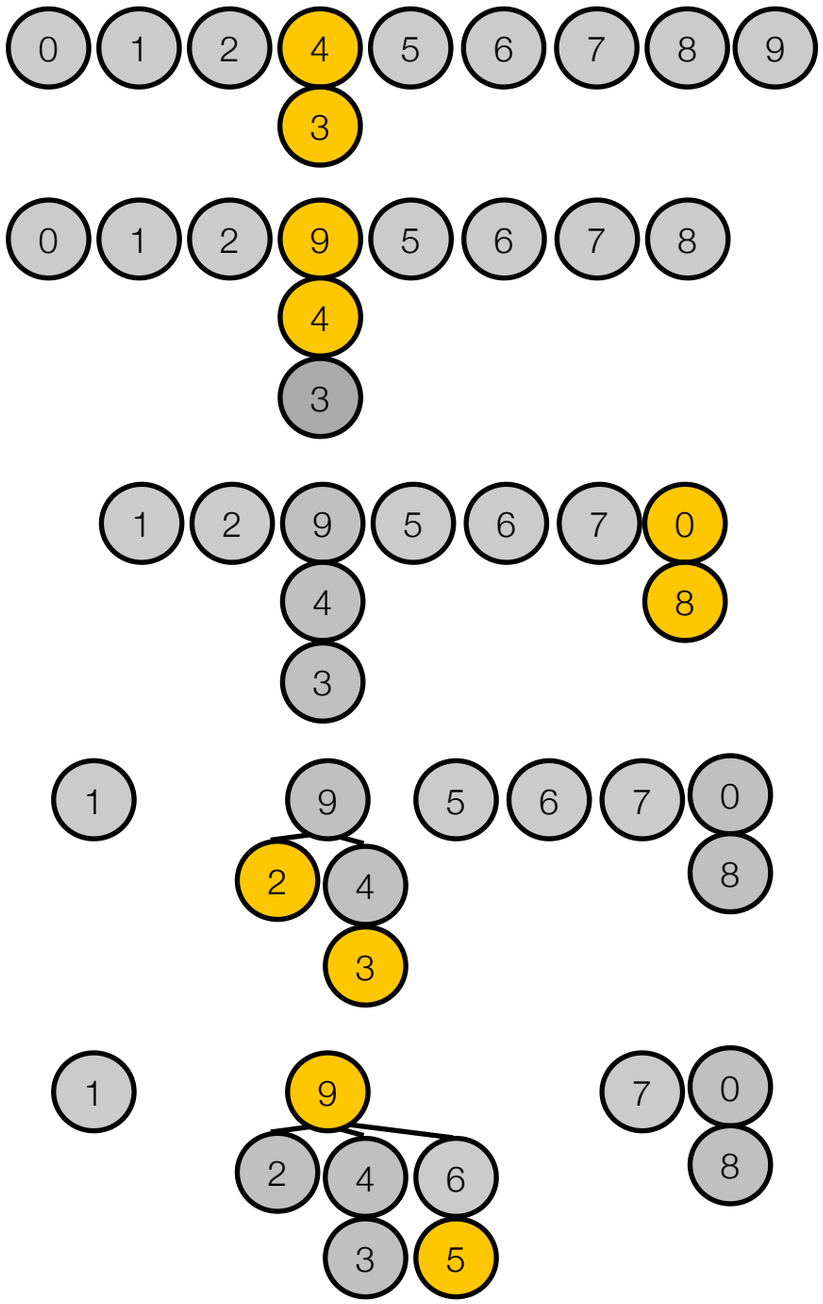
```
#include <iostream>
using namespace std;

static const int N = 10000;

int main()
{
        int i, j, p, q, id[N];
        for (i=0; i<N; i++)
                id[i] = i;
        while ( cin >> p >> q ){
                for (i=p; i!=id[i]; i=id[i]);  // find
                for (j=q; j!=id[j]; j=id[j]);
                if(i==j) continue;
                id[i]=j;  // quick union
                cout << "  << p << " " << q << endl;
      }
}
```

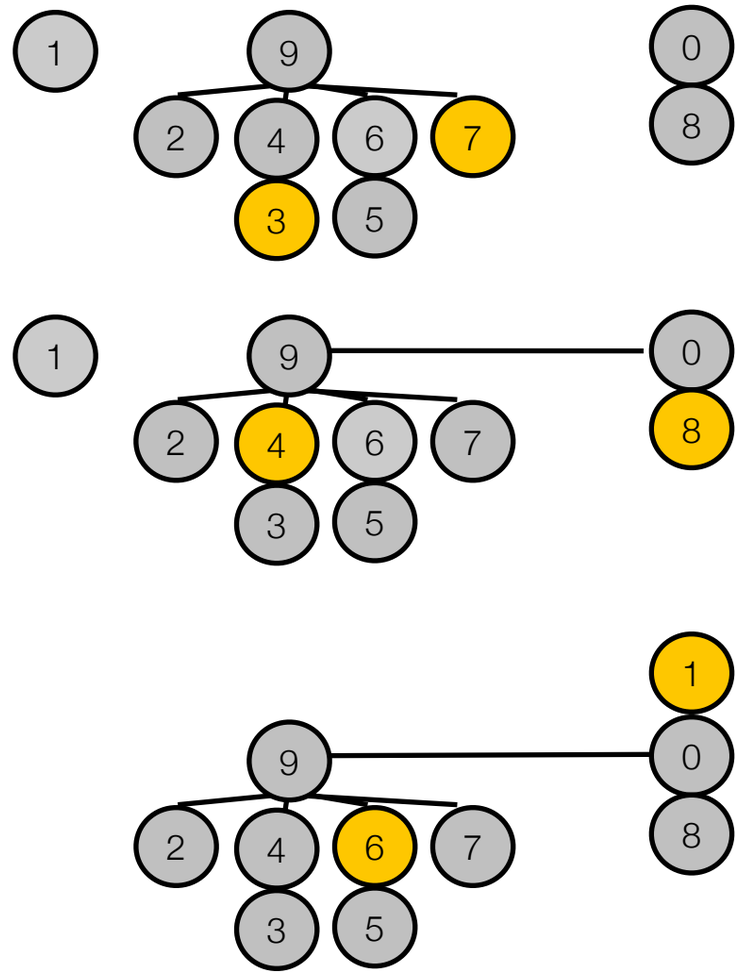
Problema de conectividad: algoritmo *quick-union*



Problema de conectividad: *quick-union*

p	q
3	4
4	9
8	0
2	3
5	6
2	9
5	9
7	3
4	8
5	6
0	2
6	1





Problema de conectividad: algoritmo *quick-union*

- quick-union NO tiene que recorrer el arreglo completo para cada par de entrada.

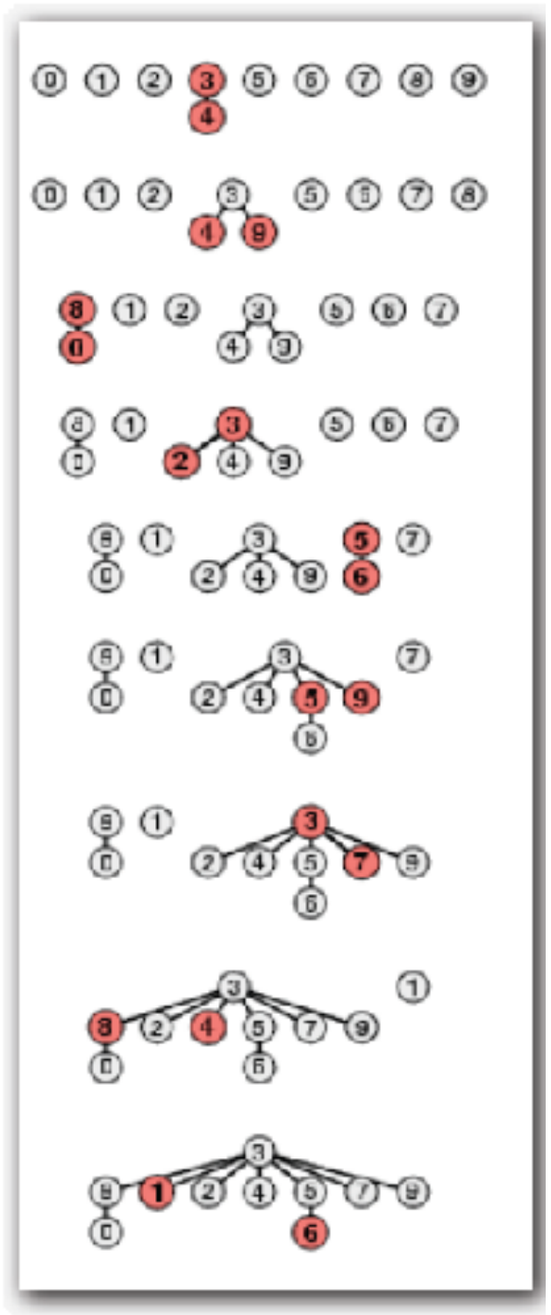
- haciendo estudios empíricos y análisis matemáticos (próximas clase) se puede determinar que el algoritmo *quick-union* es más eficiente que el algoritmo *quick-find.*

- Con M pares de entrada y N objetos, para M>N+1, el algoritmo *quick-union* puede tomar más de MN/2 instrucciones para resolver un problema de conectividad, **NOTA: no son los mismos M y N que definimos para quick-find.**

- aún así no se puede garantizar que esto ocurra para cualquier par de entradas, ya que la operación find puede ser lenta.

Problema de conectividad: *quick-union pesado*

	0	1	2	3	4	5	6	7	8	9
3-4	0	1	2	3	3	5	6	7	8	9
4-9	0	1	2	3	3	5	6	7	8	3
8-0	8	1	2	3	3	5	6	7	8	3
2-3	8	1	3	3	3	5	6	7	8	3
5-6	8	1	3	3	3	5	5	7	8	3
5-9	8	1	3	3	3	3	5	7	8	3
7-3	8	1	3	3	3	3	5	3	8	3
4-8	8	1	3	3	3	3	5	3	3	3
6-1	8	3	3	3	3	3	5	3	3	3



Utiliza un vector adicional donde se cuenta cual árbol es mayor, y siempre conecta el menor al mayor.

La versión pesada sigue a lo mas *lg N* apuntadores para determinar si 2 de *N* objetos están conectados

Codigo de quick-union pesado

```
#include <iostream>
using namespace std;
static const int N = 10000;
int main()
{
	int i, j, p, q, id[N], sz[N];
	for (i=0; i<N; i++) {
		id[i] = i; sz[i] = 1; // 2 inicializaciones
	}
	while ( cin >> p >> q ){
		for (i=p; i!=id[i]; i=id[i]);  // find
		for (j=q; j!=id[j]; j=id[j]);
		if(i==j) continue;

		if(sz[i] < sz[j]){
			id[i]=j;  sz[j]+= sz[i];// quick union + peso
		}
		else{
			id[j]=i;  sz[i]+= sz[j];// quick union + peso
		}
		cout << "  << p << " " << q << endl;
	}
}
```

Por lo tanto, ¿la complejidad de quick union pesado es?

- *M*lg(N),* para *M* pares que llegan con N nodos a investigar.

- De esta forma, agregando un poco de código, tenemos un algoritmo que es mucho mas eficiente.

Resumiendo

- Establecer el problema lo cual incluye identificar las operaciones abstractas.
- Desarrollar con cuidado una implementación concisa para un algoritmo simple.
- Desarrollar mejoras a la implementación mediante de refinamiento y validación de las ideas por medio de análisis empíricos o matemáticos (o ambos).
- Buscar representaciones abstractas de alto nivel tanto de algoritmos en operación como de estructuras de datos que nos permita un diseño optimo de las versiones.
- Analizar siempre los casos peores, pero también caracterizar los reales.

Como medir el tiempo en C++?

- Hay que incluir la libreria `<ctime>` en el archivo C++,

- Y se usa

```
clock_t start, finish;
start = clock();

// aqui va el codigo del cual se quiere medir el tiempo

finish = clock();

t  = ( (finish - start)/CLOCKS_PER_SEC );
```

Explique esta función

```
using namespace std;


long int giveMeRandomNPot10(int pot10){

        long int n=0,fact=1; //n es resultado y factor potencia de 10
        for(int i=0;i<pot10;i++){
            n += (rand() % 10) * fact;
            fact *=10;
        }
        return n;

}
```

genera números aleatorios entre 0 y 10***pot10***-*1*