



Online Connectivity



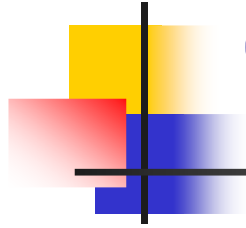
Paolo Camurati
Dip. Automatica e Informatica
Politecnico di Torino



Online Connectivity

- Solve the following problem
 - Given a set of N objects
 - Union command
 - Connects two objects
 - Find query
 - Finds connected couples
- We do not want to know the path which connect two objects but only whether such a path exists or not
- N objects (whatever they are) can be mapped on N integers (from 0 to $N-1$)

huge



Online Connectivity

- Input: sequence of integer pairs (p, q)
 - Interpretation: p is connected to q
- Output
 - Null if p and q are already connected (directly or indirectly)
 - Else (p, q)



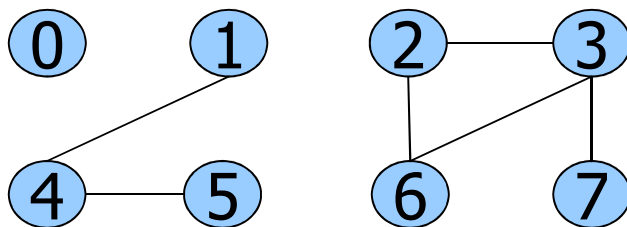
Online Connectivity

- Connectivity is an equivalence relation
 - Reflexive: *p is connected to p*
 - Symmetrical: *if p is connected to q , q is connected to p*
 - Transitive: *if p is connected to q and q is connected to r , then p is connected to r*

Online Connectivity

- Connected component

- Maximal subset of mutually reachable nodes



$\{0\}$ $\{1, 4, 5\}$ $\{2, 3, 6, 7\}$

3 connected components

- There are no elements connecte to an element outside its connected component
 - Find is checking connected component
 - Union is replacing connected component with their union



Applications

- Computer networks
 - Integers p and q represent computers
 - (p, q) connections between computers
- Electrical networks
 - Integers p and q represent contact points
 - (p, q) wires
- Programming environments
 - Integers p and q represent variables
 - (p, q) declarations of equivalent variables
- Social network
 - ...

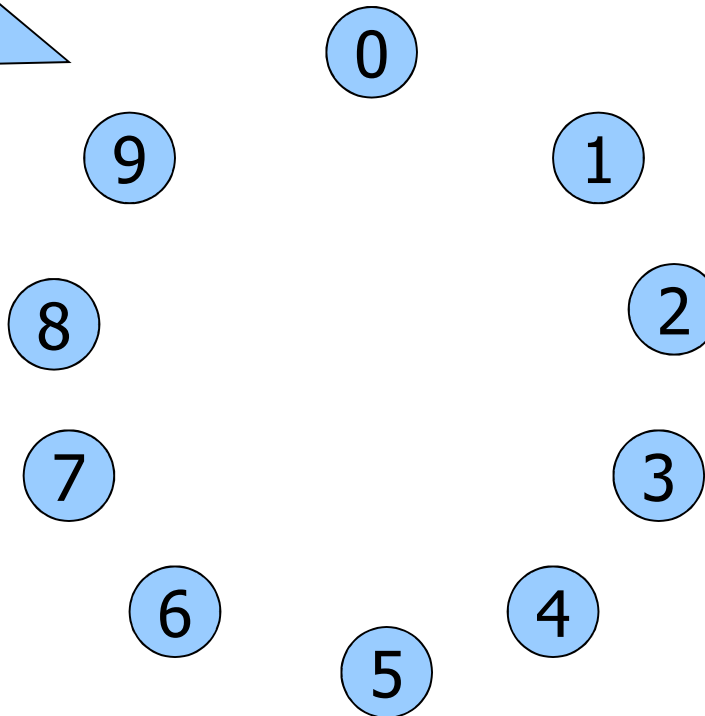


Example

■ Pairs

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

Graph:
structure representing
nodes (vertices)
and their connections
(edges)



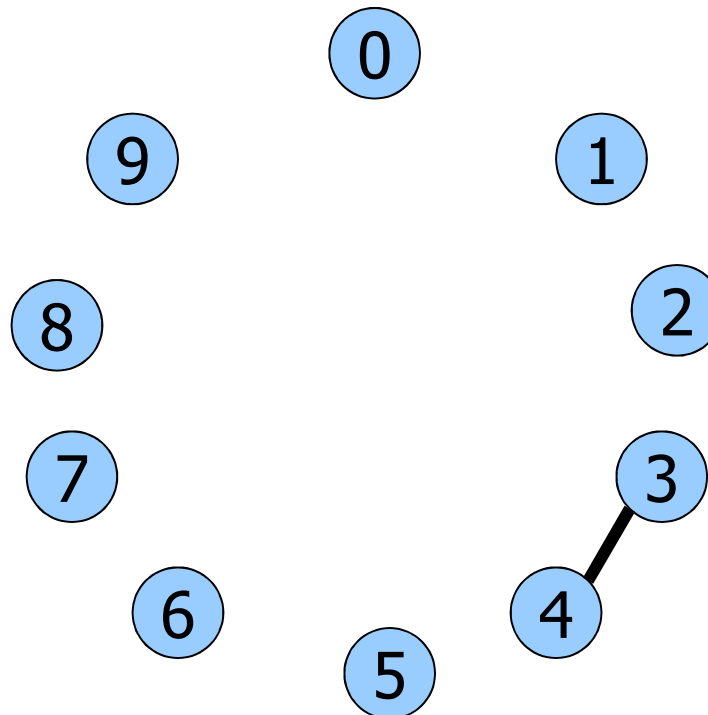


Example

■ Pairs

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

Input
3-4



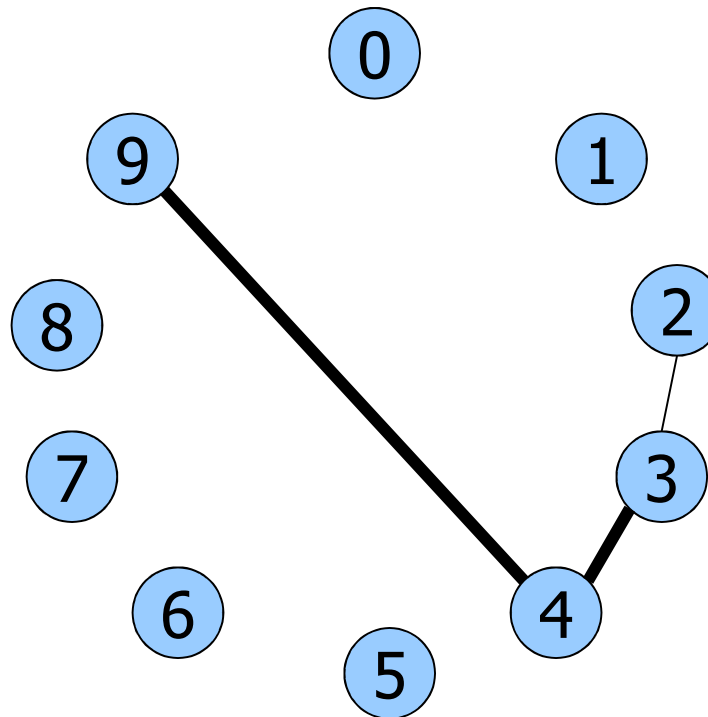
Output
3-4

Example

■ Pairs

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

Input
4-9



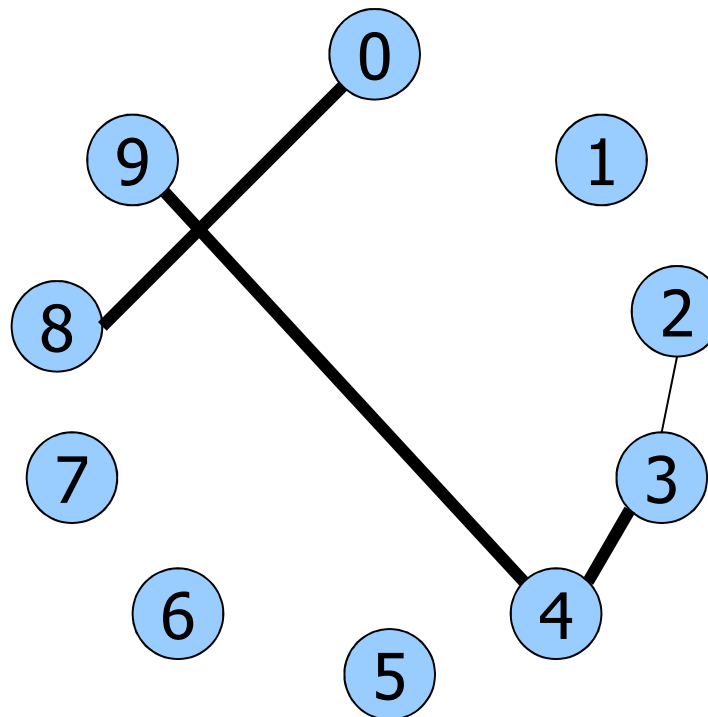
Output
4-9

Example

■ Pairs

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

Input
8-0



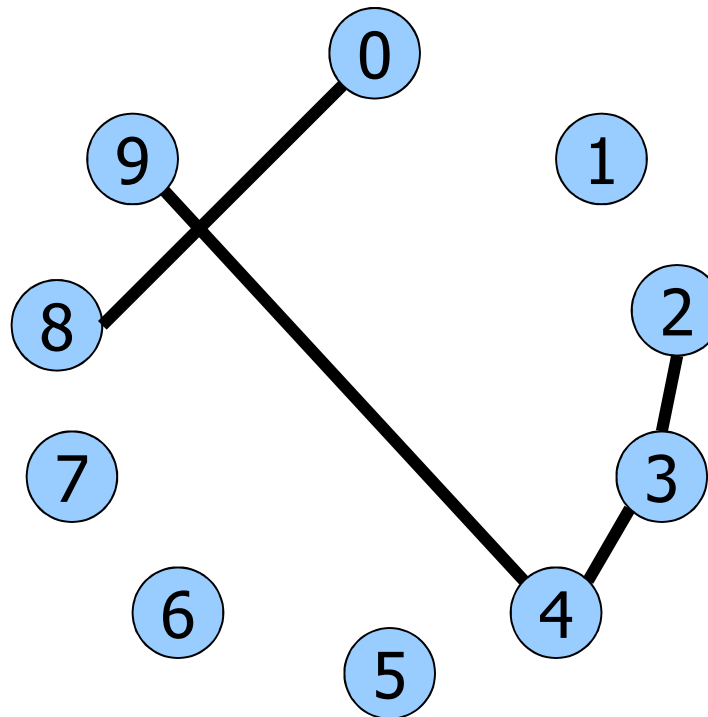
Output
8-0

Example

■ Pairs

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

Input
2-3



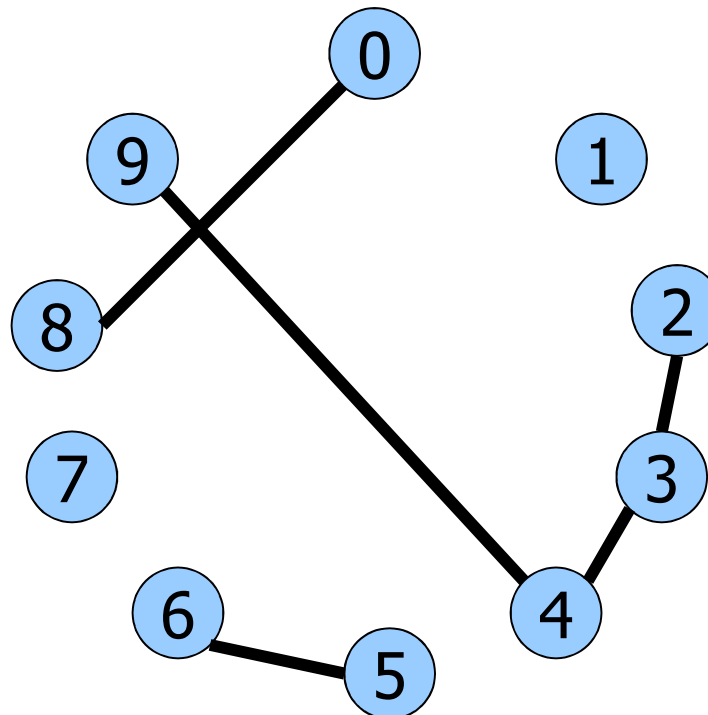
Output
2-3

Example

■ Pairs

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

Input
5-6



Output
5-6

Example

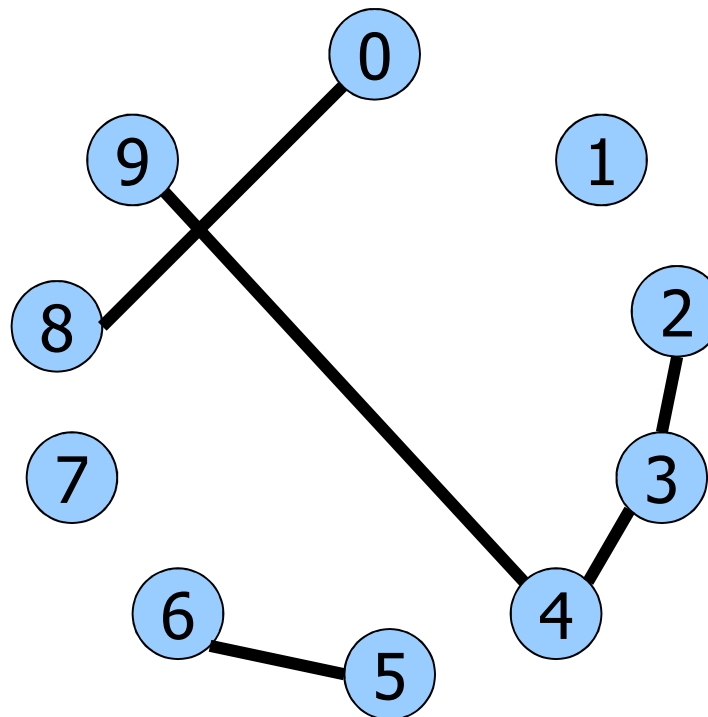
■ Pairs

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

Input
2-9

Output

Path
2-3-4-9
already exists

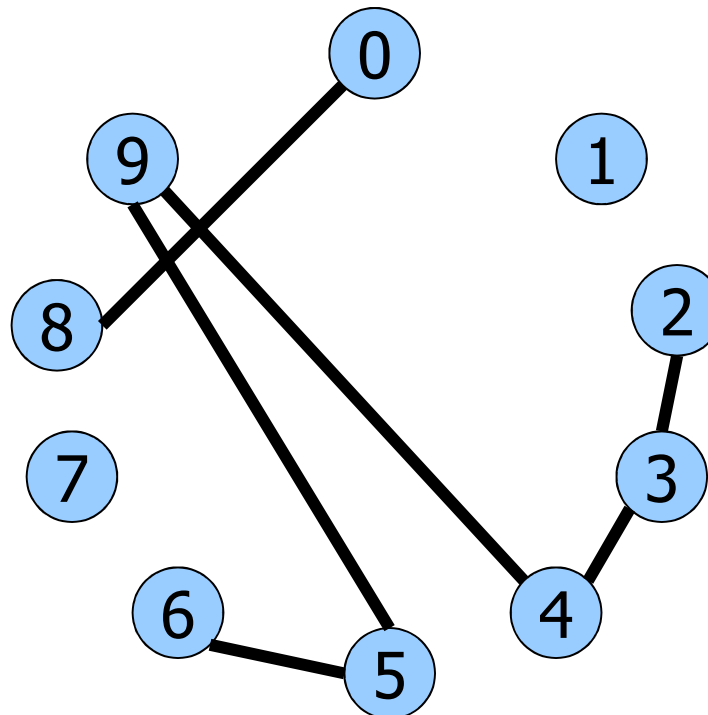


Example

■ Pairs

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

Input
5-9



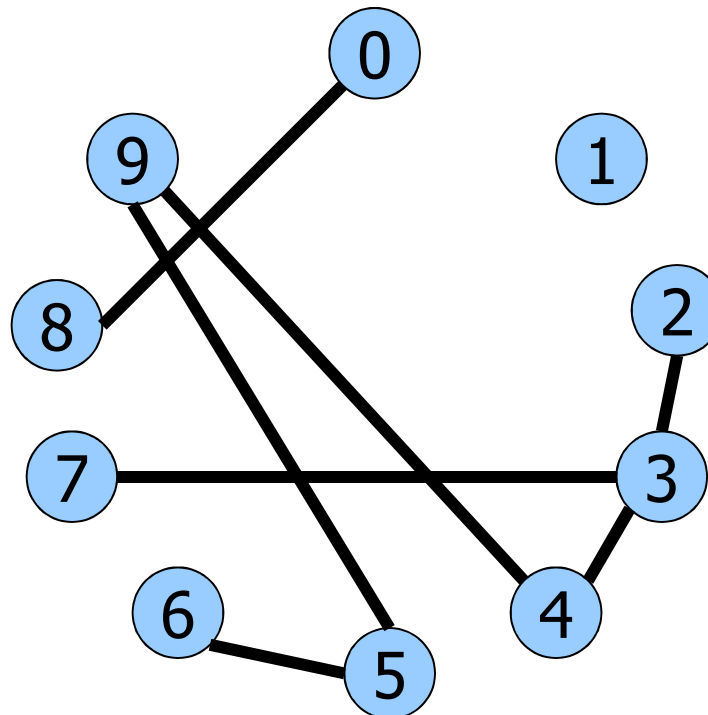
Output
5-9

Example

■ Pairs

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

Input
7-3



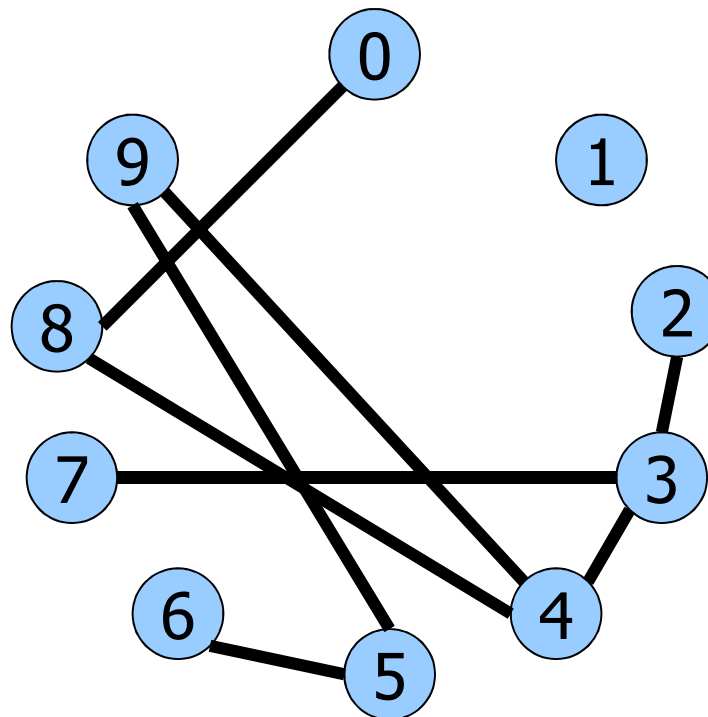
Output
7-3

Example

■ Pairs

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

Input
4-8

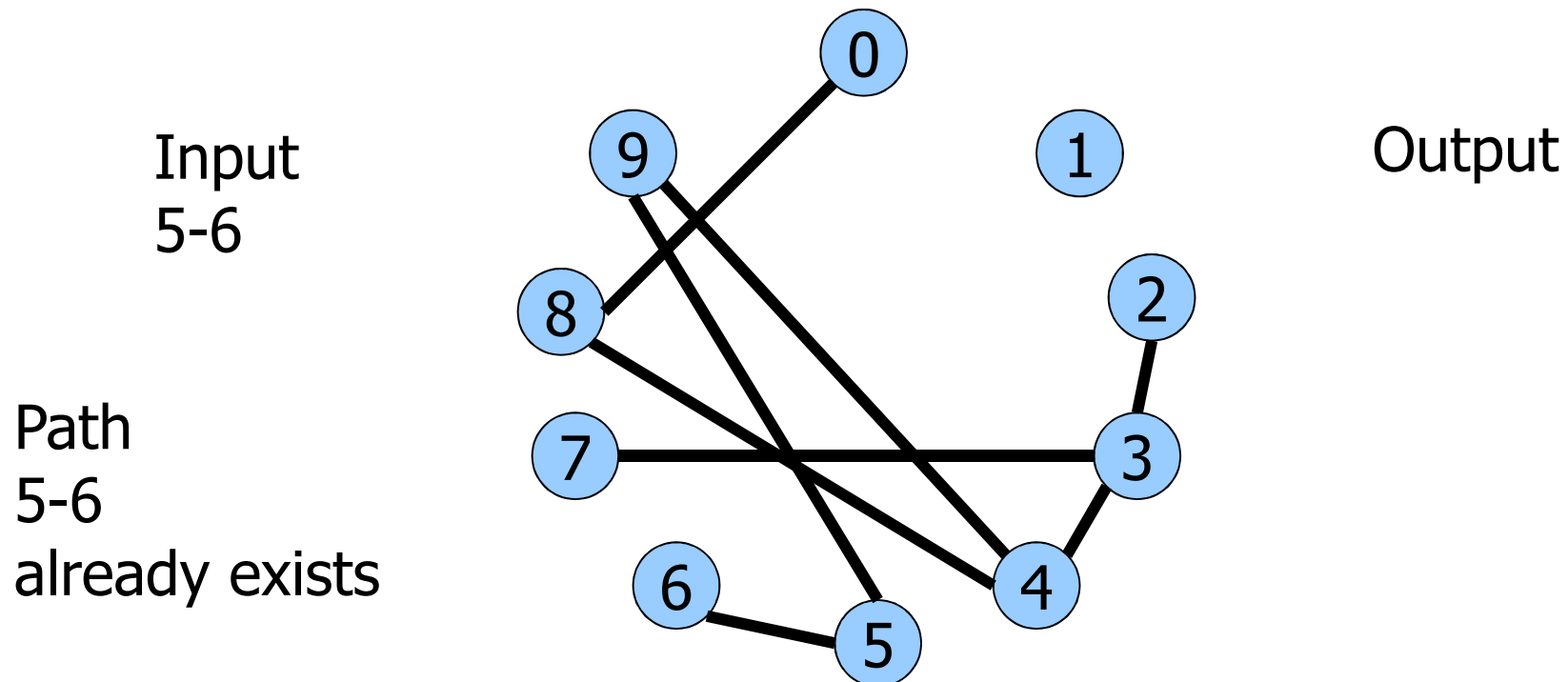


Output
4-8

Example

■ Pairs

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



Example

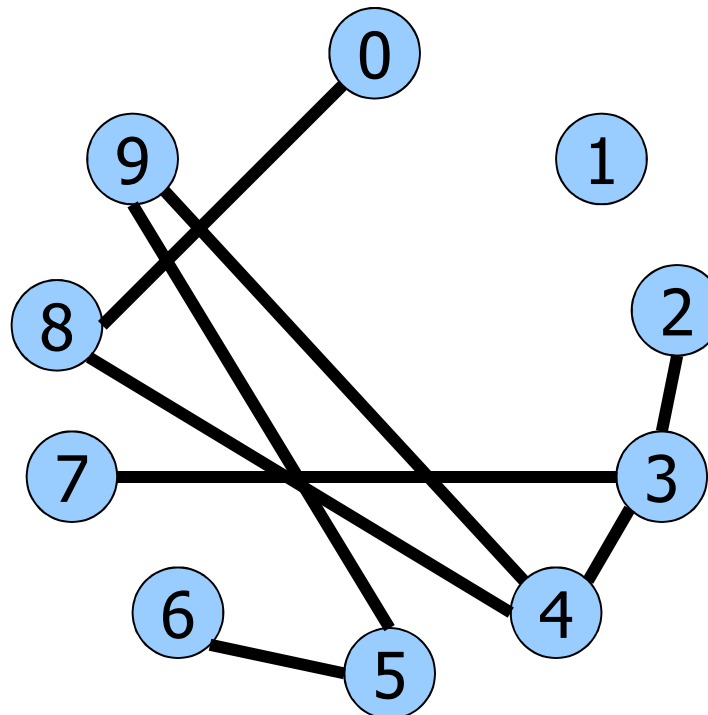
■ Pairs

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

Input
0-2

Output

Path
0-8-4-3-2
already exists

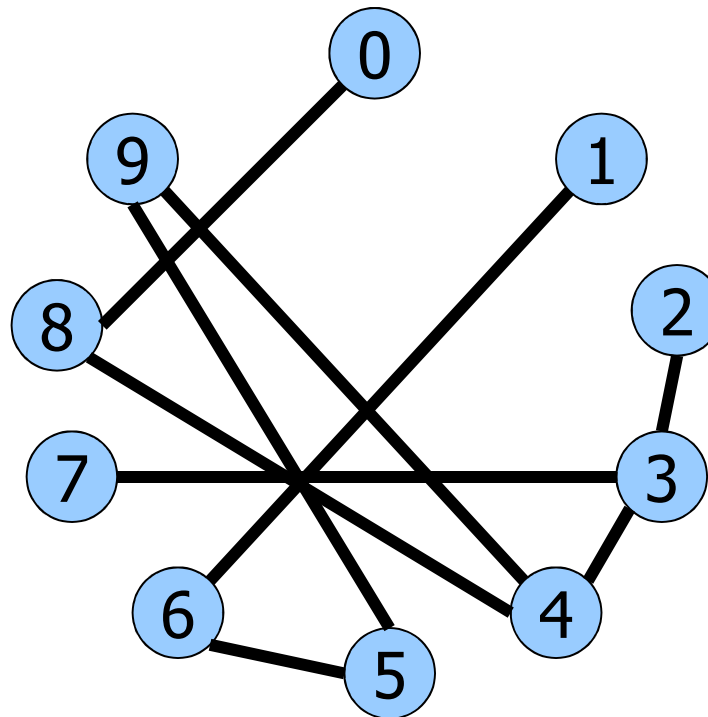


Example

■ Pairs

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

Input
6-1



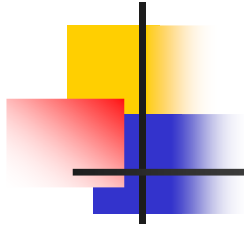
Output
6-1



Approach

Hypothesis

- We do not have the graph
- We work pair by pair
 - We keep and update information necessary to find out connectivity
 - Sets S_i of connected pairs, initially as many sets as nodes, each node being connected just with itself
- Abstract operations
 - find: find the set an object belongs to
 - union: merge two sets



- Algorithm: repeat for all pairs (p, q)
 - Read the pair (p, q)
 - Execute find on p : find an S_p such that $p \in S_p$
 - Execute find on q : find an S_q such that $q \in S_q$
 - If S_p and S_q coincide
 - Consider the next pair
 - Otherwise execute union on S_p and S_q

Quick find

- Represent sets S_i of connected pairs with array `id`
 - Initially $\text{id}[i] = i$ (no connection)

id


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

- If p and q are connected, $\text{id}[p] = \text{id}[q]$

6 and 8 are connected

id

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

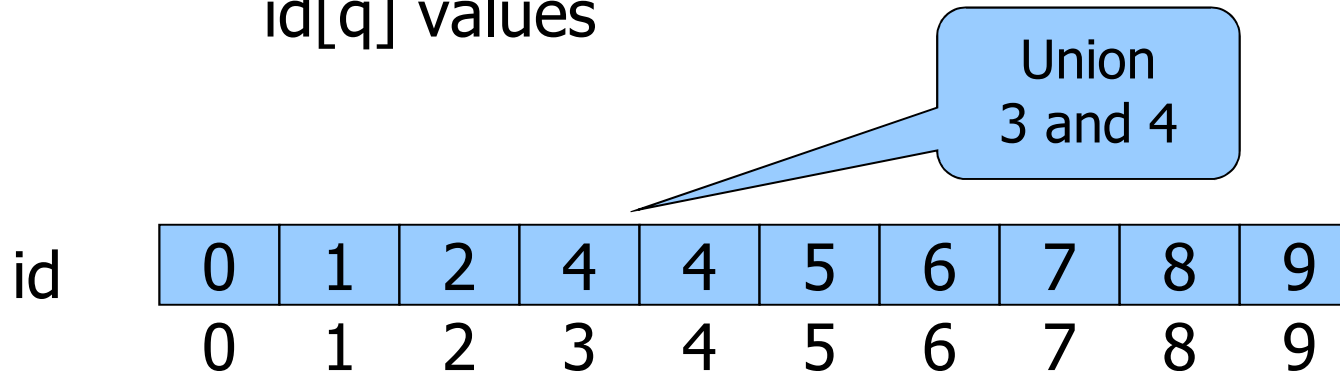




Quick find

Repeat for all pairs (p, q)

- Read pair (p, q)
- Find
 - Check if ($\text{id}[p] = \text{id}[q]$)
 - Do nothing and move to the next pair
- Else Union
 - Scan the array, replacing $\text{id}[p]$ values with $\text{id}[q]$ values

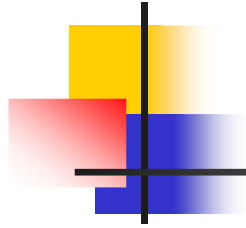




Tree representation

by hand

- Some objects represent the set they belong to
- Other objects point to the object that represents the set they belong to



Example

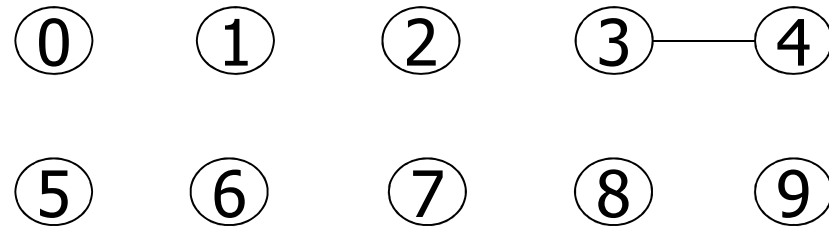
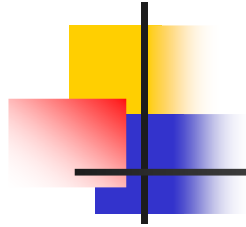
① 0 ① 1 ① 2 ① 3 ① 4
① 5 ① 6 ① 7 ① 8 ① 9

Initially

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

$$S_0 = \{0\}, S_1 = \{1\}, S_2 = \{2\}, S_3 = \{3\}, S_4 = \{4\}$$
$$S_5 = \{5\}, S_6 = \{6\}, S_7 = \{7\}, S_8 = \{8\}, S_9 = \{9\}$$

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



$p \ q = 3 \ 4$

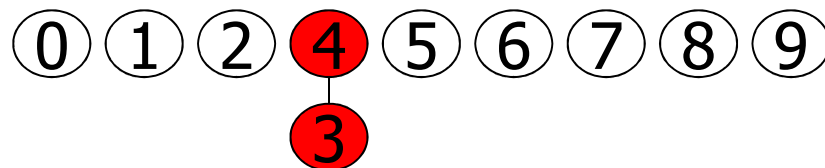
id	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

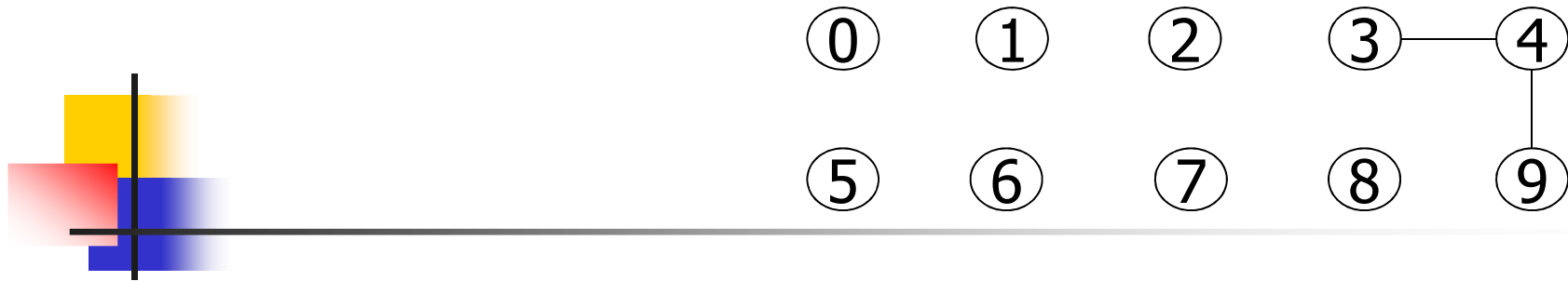
$\text{id}[p]=3 \neq \text{id}[q]=4$

replace all $\text{id}[p]$ values with $\text{id}[q]$ values

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

$S_0 = \{0\}, S_1 = \{1\}, S_2 = \{2\}, S_{3-4} = \{3,4\},$
 $S_5 = \{5\}, S_6 = \{6\}, S_7 = \{7\}, S_8 = \{8\}, S_9 = \{9\}$





$p \ q = 4 \ 9$

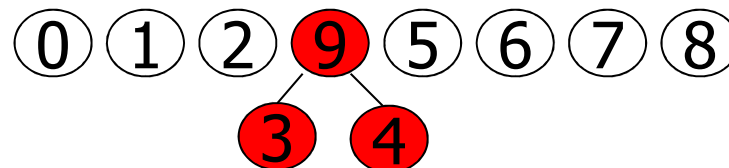
id	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

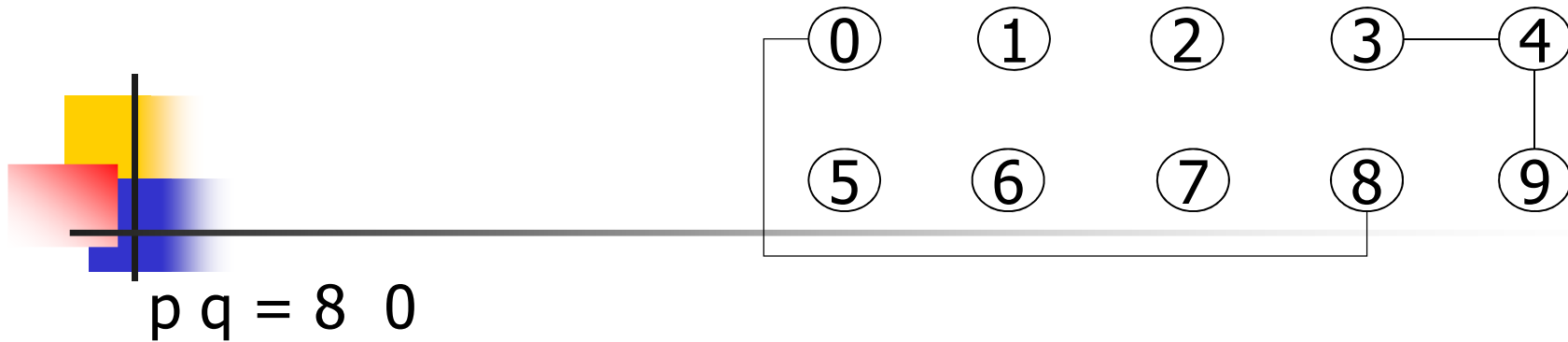
$id[p]=4 \neq id[q]=9$

replace all $id[p]$ values with $id[q]$ values

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

$S_0 = \{0\}, S_1 = \{1\}, S_2 = \{2\}, S_{3-4-9} = \{3,4,9\},$
 $S_5 = \{5\}, S_6 = \{6\}, S_7 = \{7\}, S_8 = \{8\}$





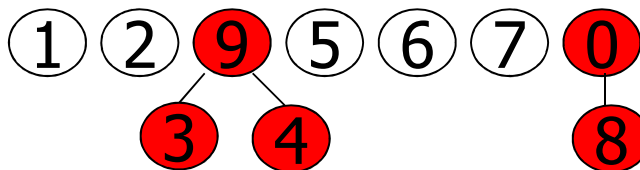
id	0	1	2	9	9	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

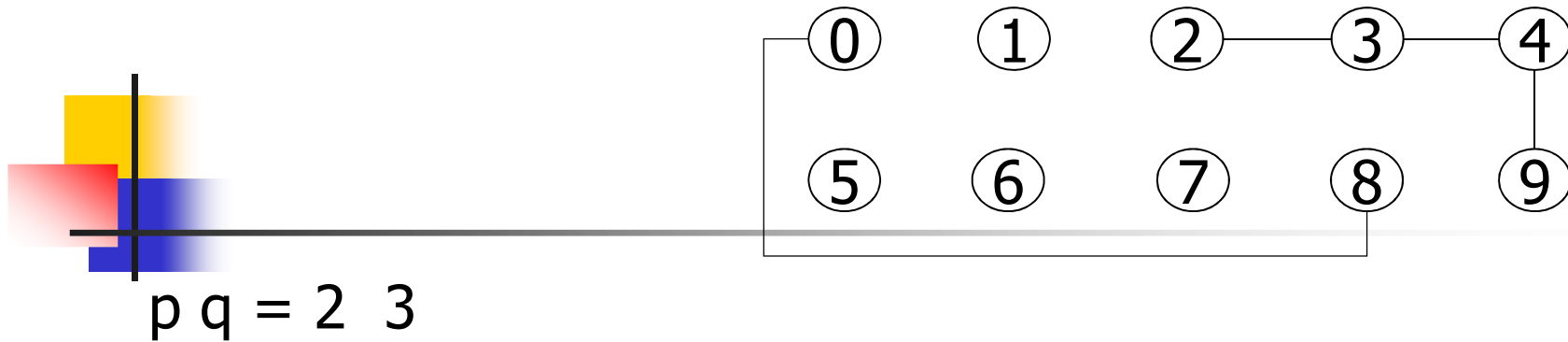
$\text{id}[p]=8 \neq \text{id}[q]=0$

replace all $\text{id}[p]$ values with $\text{id}[q]$ values

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

$S_{0-8} = \{0,8\}$, $S_1 = \{1\}$, $S_2 = \{2\}$, $S_{3-4-9} = \{3,4,9\}$,
 $S_5 = \{5\}$, $S_6 = \{6\}$, $S_7 = \{7\}$





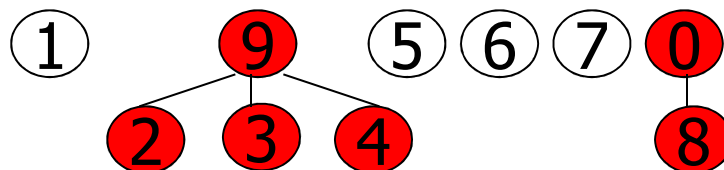
id	0	1	2	9	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

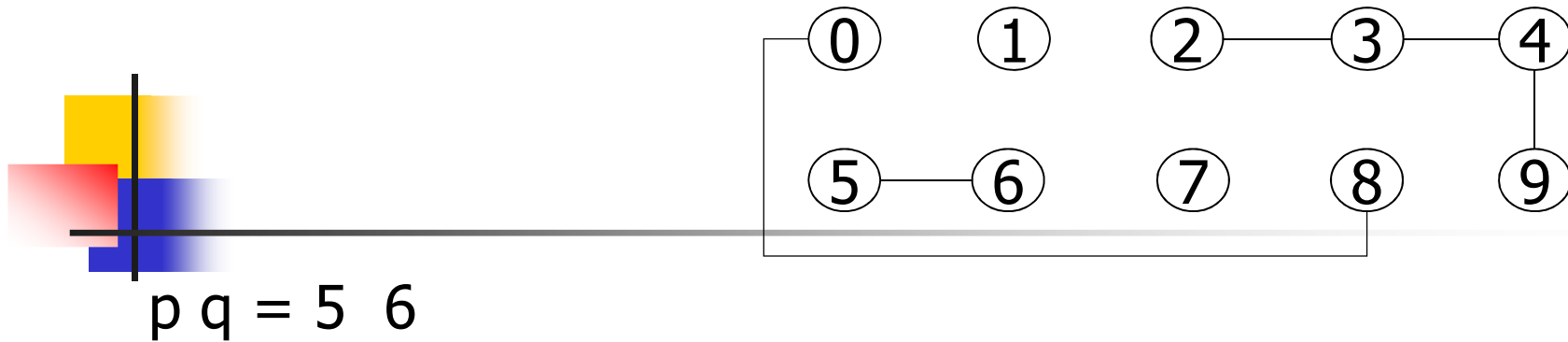
$id[p]=2 \neq id[q]=9$

replace all $id[p]$ values with $id[q]$ values

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

$S_{0-8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2-3-4-9} = \{2,3,4,9\}$,
 $S_5 = \{5\}$, $S_6 = \{6\}$, $S_7 = \{7\}$





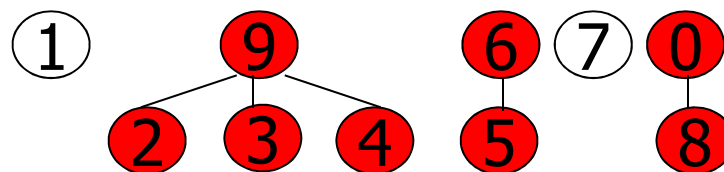
id	0	1	9	9	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

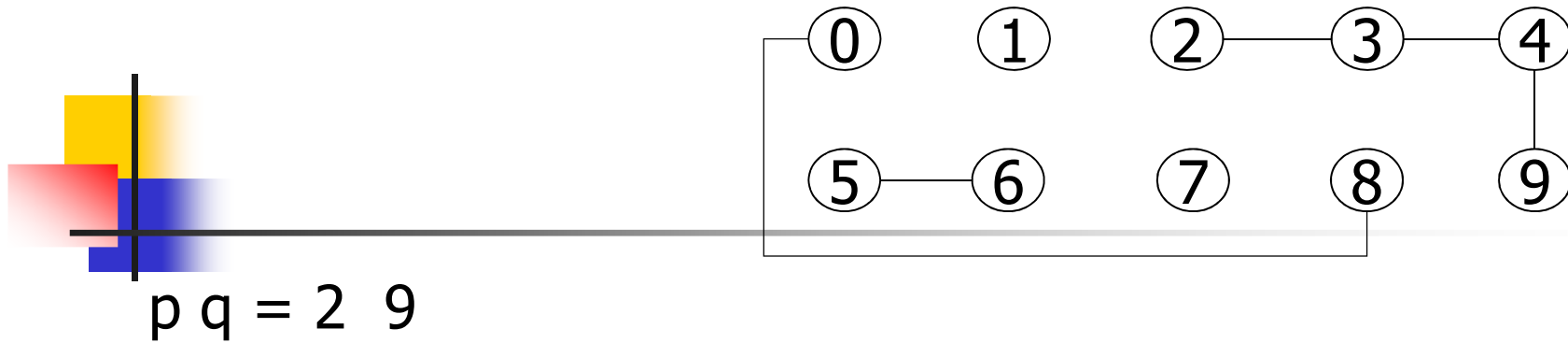
id[p]=5 \neq id[q]=6

replace all id[p] values with id[q] values

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

$S_{0-8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2-3-4-9} = \{2,3,4,9\}$,
 $S_{5-6} = \{5,6\}$, $S_7 = \{7\}$



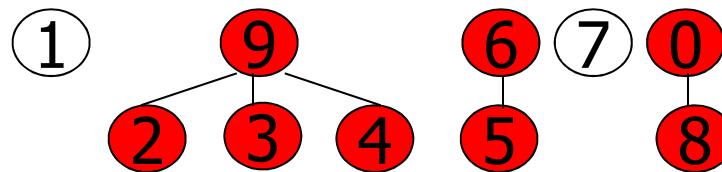


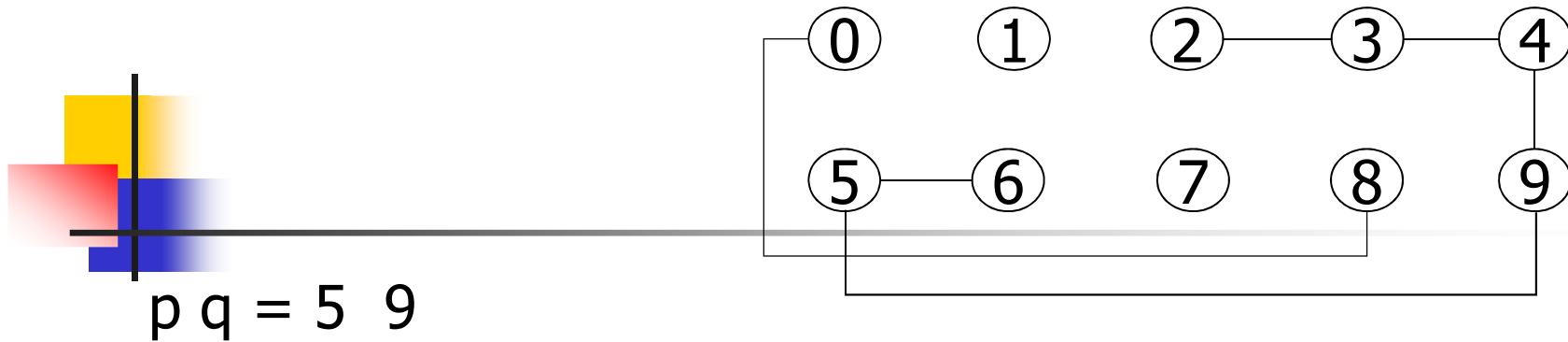
id	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

$id[p]=9 = id[q]=9$
no change

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

$S_{0-8} = \{0,8\}, S_1 = \{1\}, S_{2-3-4-9} = \{2,3,4,9\},$
 $S_{5-6} = \{5,6\}, S_7 = \{7\}$





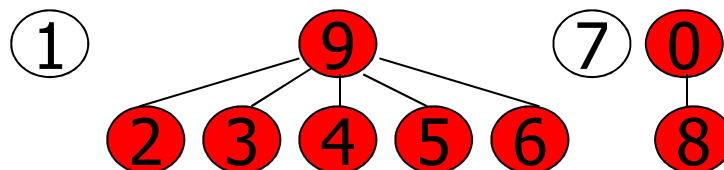
id	0	1	9	9	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

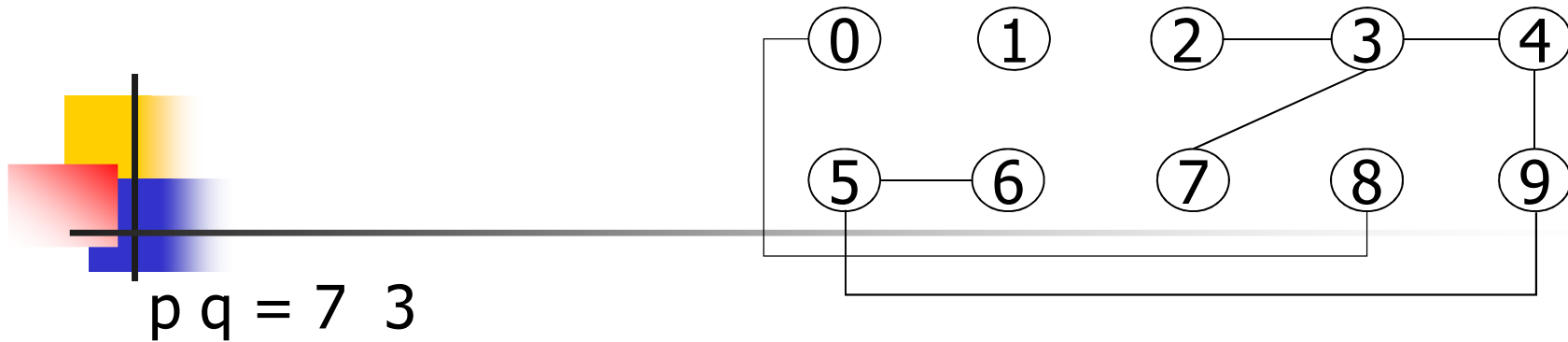
$\text{id}[p]=6 \neq \text{id}[q]=9$

replace all $\text{id}[p]$ values with $\text{id}[q]$ values

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

$S_{0-8} = \{0,8\}$, $S_1 = \{1\}$, $S_{2-3-4-5-6-9} = \{2,3,4,5,6,9\}$,
 $S_7 = \{7\}$





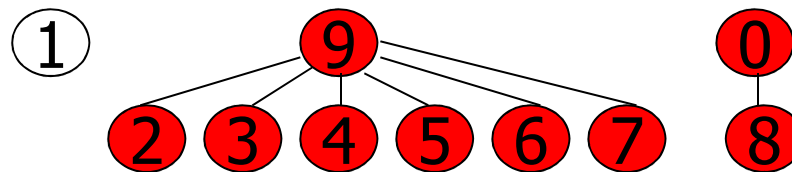
id	0	1	9	9	9	9	9	7	0	9
	0	1	2	3	4	5	6	7	8	9

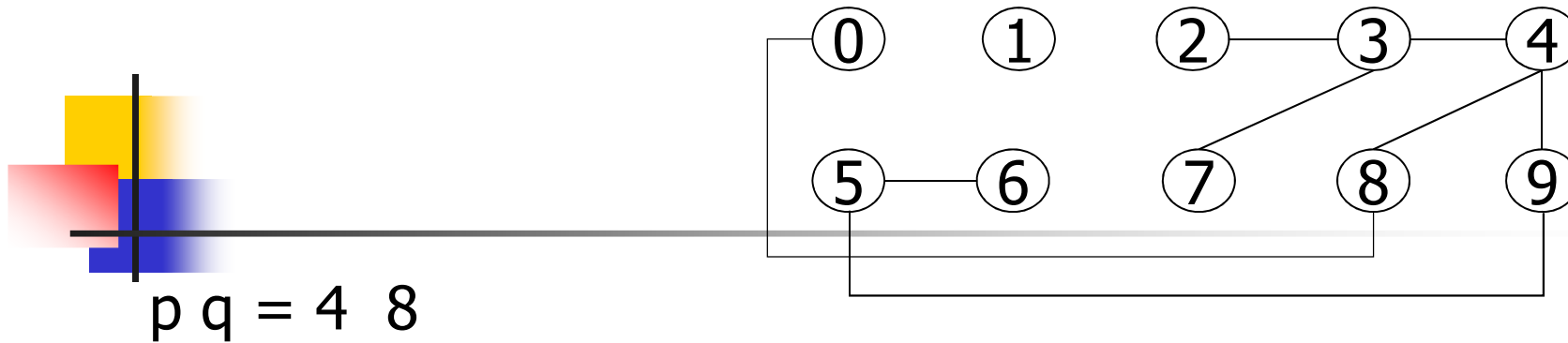
$id[p]=7 \neq id[q]=9$

replace all $id[p]$ values with $id[q]$ values

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

$$S_{0-8} = \{0,8\}, S_1 = \{1\}, S_{2-3-4-5-6-7-9} = \{2,3,4,5,6,7,9\}$$





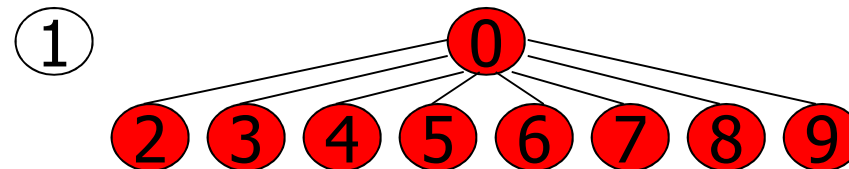
id	0	1	9	9	9	9	9	9	0	9
	0	1	2	3	4	5	6	7	8	9

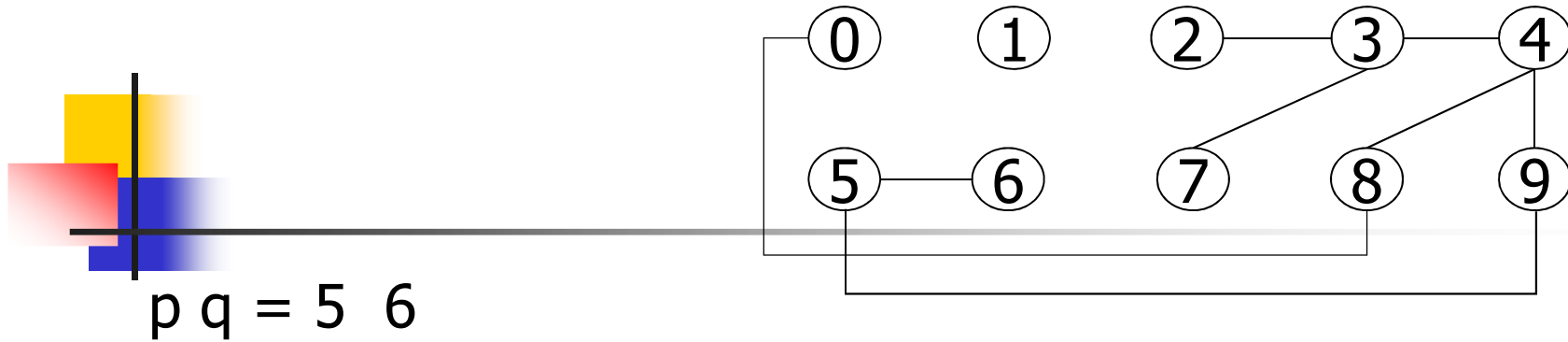
$\text{id}[p]=9 \neq \text{id}[q]=0$

replace all $\text{id}[p]$ values with $\text{id}[q]$ values

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

$S_1 = \{1\}, S_{0-2-3-4-5-6-7-8-9} = \{0, 2, 3, 4, 5, 6, 7, 8, 9\}$





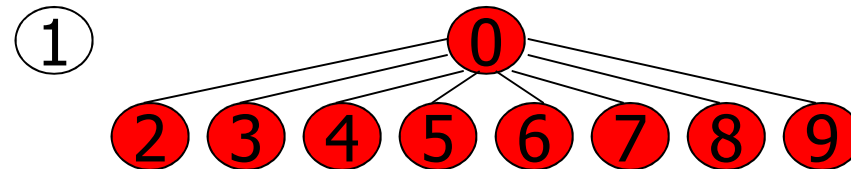
$p \ q = 5 \ 6$

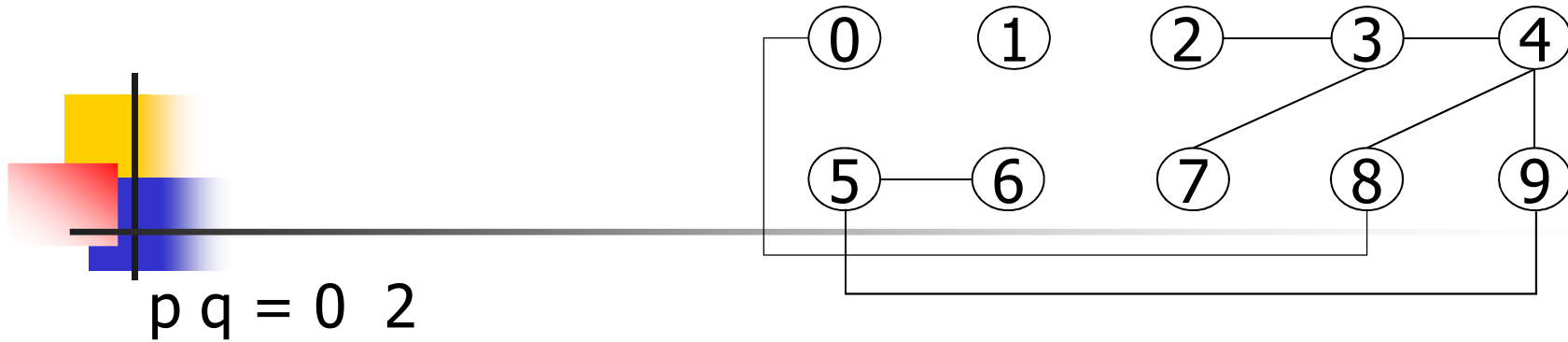
id	0	1	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

$id[p]=0 = id[q]=0$
no change

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

$$S_1 = \{1\}, S_{0-2-3-4-5-6-7-8-9} = \{0,2,3,4,5,6,7,8,9\}$$



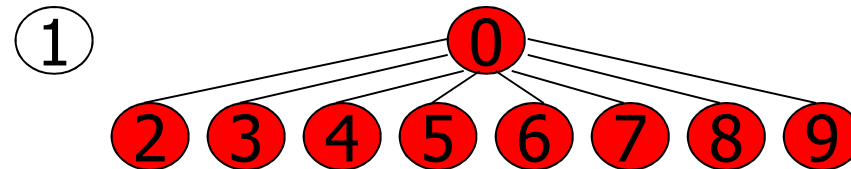


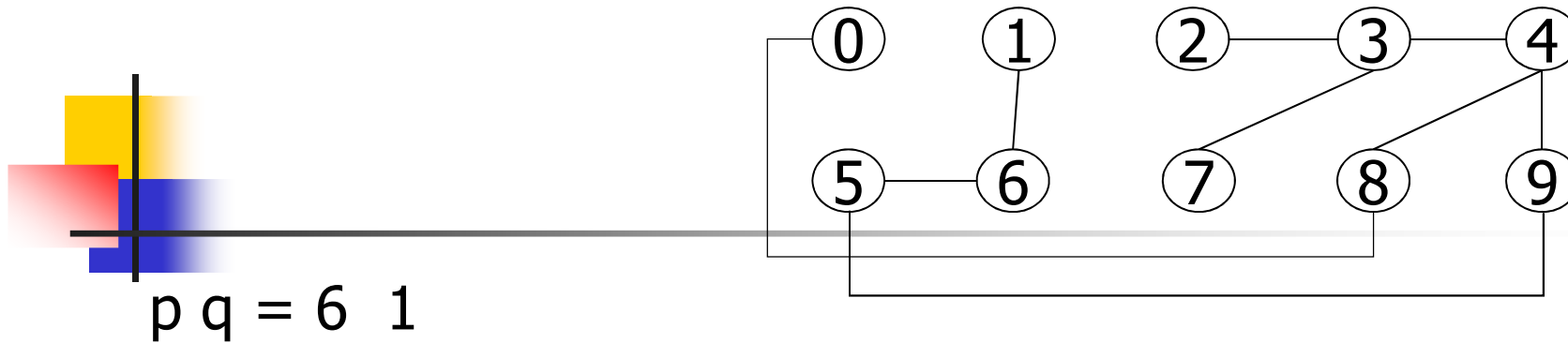
id	0	1	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

id[p]=0 = id[q]=0
no change

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

$$S_1 = \{1\}, S_{0-2-3-4-5-6-7-8-9} = \{0,2,3,4,5,6,7,8,9\}$$





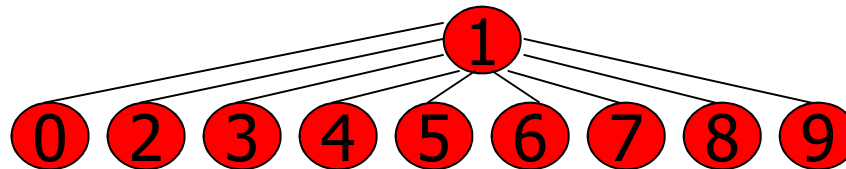
id	0	1	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

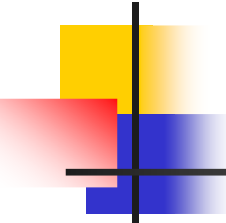
$\text{id}[p]=0 = \text{id}[q]=1$

replace all $\text{id}[p]$ values with $\text{id}[q]$ values

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

$S_{0-1-2-3-4-5-6-7-8-9} = \{0,1,2,3,4,5,6,7,8,9\}$





```
#include <stdio.h>
```

```
#define N 10000
```

```
main() {
```

```
    int i, t, p, q, id[N];
```

i entry pos

```
    for(i=0; i<N; i++)
```

```
        id[i] = i;
```

```
    printf("Input pair p q: ");
```

read pairs

```
    while (scanf("%d %d", &p, &q) ==2) {
```

```
        if (id[p] == id[q])
```

```
            printf("%d %d already connected\n", p,q);
```

```
        else {
```

```
            for (t = id[p], i = 0; i < N; i++)
```

t for tmp

```
                if (id[i] == t)
```

```
                    id[i] = id[q];
```

id i may change

```
            printf("pair %d %d not yet connected\n", p, q);
```

```
        }
```

```
    printf("Input pair p q: ");
```

```
}
```

```
}
```



Performance

■ Find

- Simple reference to cell in array `id[index]`, unit cost

■ Union

- Scan array to replace p values with q values, cost linear in array size

■ overall number of operations related to

pairs · array size

n

n

n^2

Quadratic

...

Too slow



Quick union

- Represent sets S_i of connected pairs with an array `id`
 - Initially all the objects point to themselves
 $\text{id}[i] = i$ (no connection)
 - Each object **points** either to an object to which it is connected or to itself (no loops)
Writing $(\text{id}[i])^*$ for $\text{id}[\text{id}[\text{id}[\dots \text{id}[i]]]]$
if objects i are j connected
 $(\text{id}[i])^* = (\text{id}[j])^*$

Quick union

- Each object points either to an object to which it is connected or to itself (no loops)

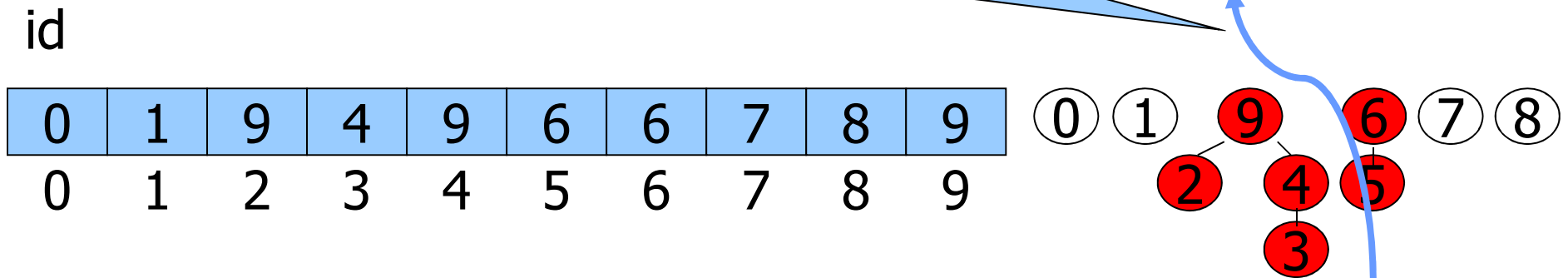
Writing $(\text{id}[i])^*$ for $\text{id}[\text{id}[\text{id}[\dots \text{id}[i]]]]$

if objects i are j connected

$$(\text{id}[i])^* = (\text{id}[j])^*$$

- Example

Keep going until it doesn't change

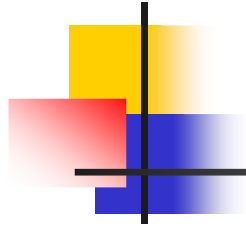




Quick union

Algorithm

- Repeat for all the pairs (p, q)
 - Read pair (p, q)
 - If $(\text{id}[p])^* = (\text{id}[q])^*$
 - Do nothing (the pair is already connected) and move on to the next pair
 - Else $\text{id}[(\text{id}[p])^*] = (\text{id}[q])^*$ (connect the pair)



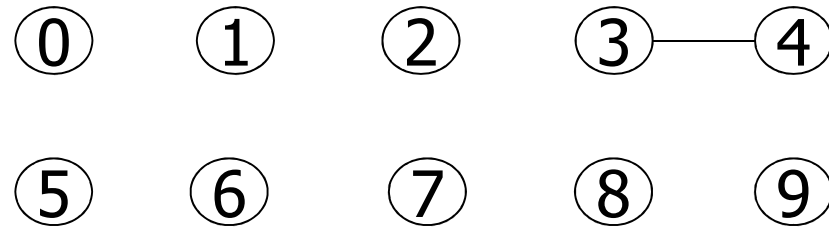
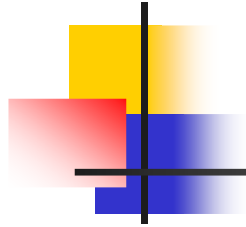
Example

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

Initially

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

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



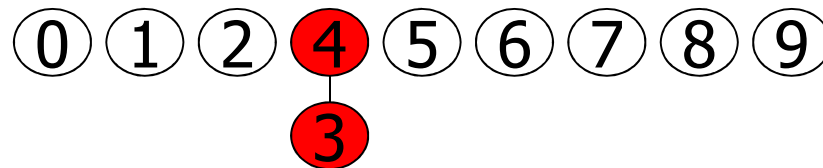
$p \ q = 3 \ 4$

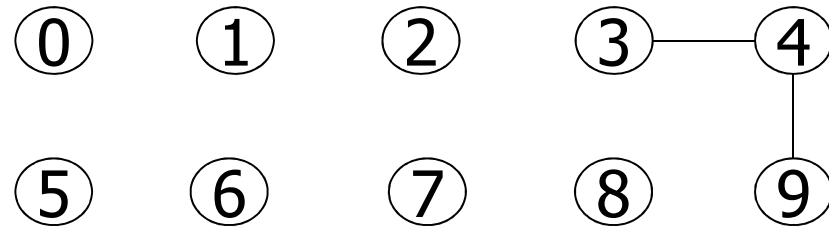
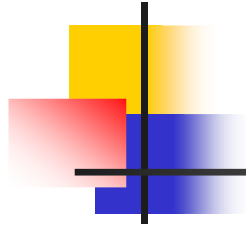
id	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

$id[p]=3 \neq id[q]=4$

p points to q : $id[p]=4$

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





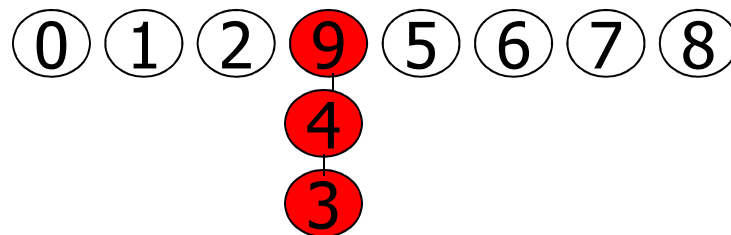
$p \ q = 4 \ 9$

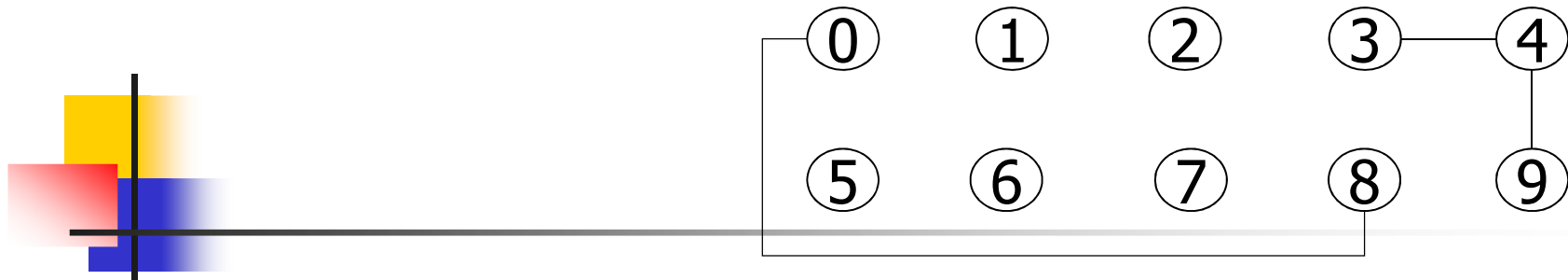
id	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

$id[p]=4 \neq id[q]=9$

p points to q : $id[p]=9$

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





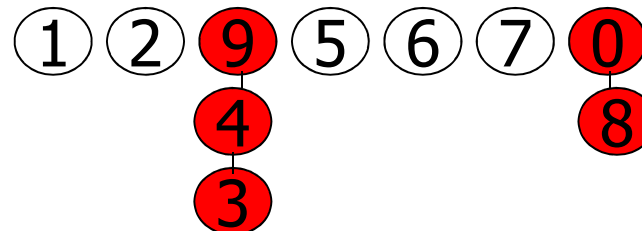
$p \ q = 8 \ 0$

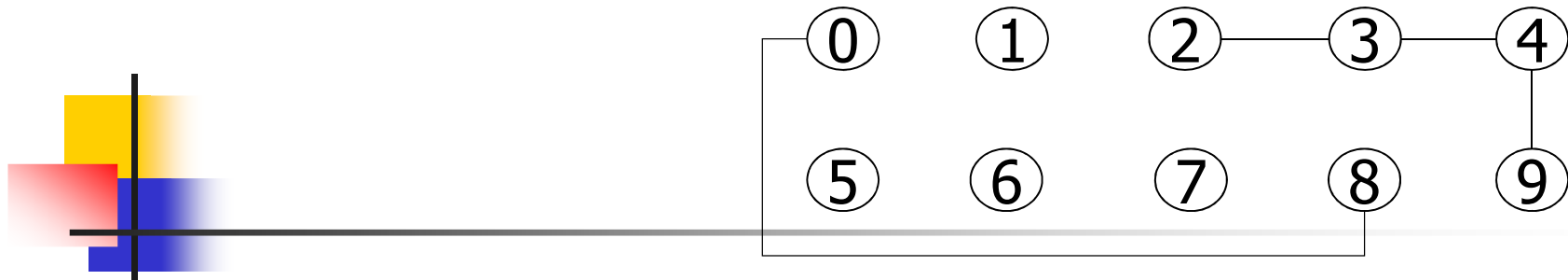
id	0	1	2	4	9	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

$id[p]=8 \neq id[q]=0$

p points to q : $id[p]=0$

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





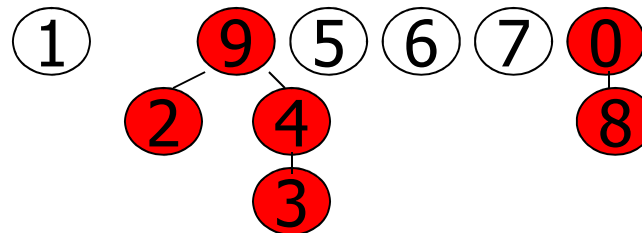
$p \ q = 2 \ 3$

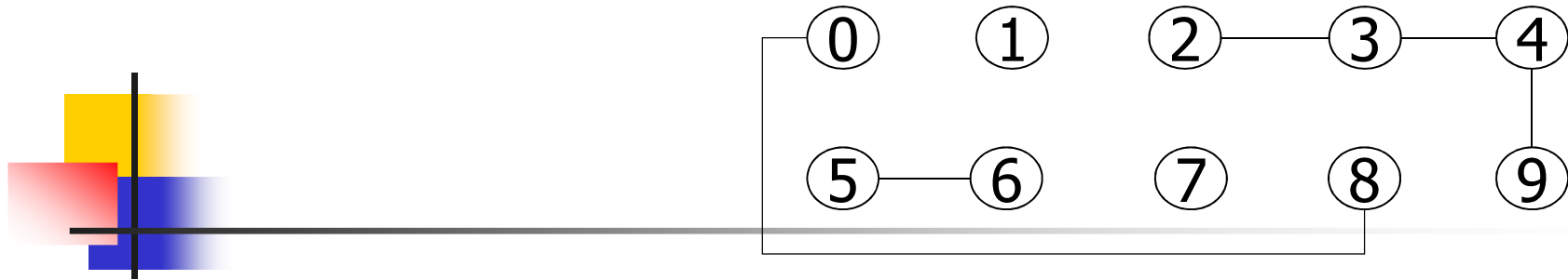
id	0	1	2	4	9	5	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

$id[p]=2 \neq id[id[id[q]]]=9$

p points to q : $id[p]=9$

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



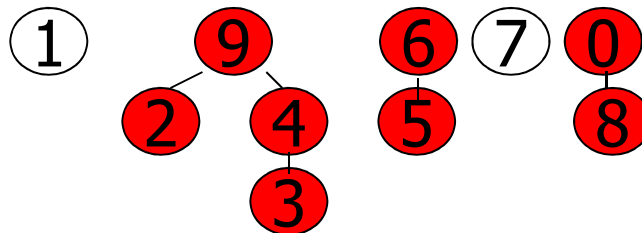


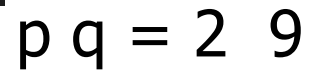
p q = 5 6

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

id[p]=5 \neq id[q]=6
p points to q: id[p]=6

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

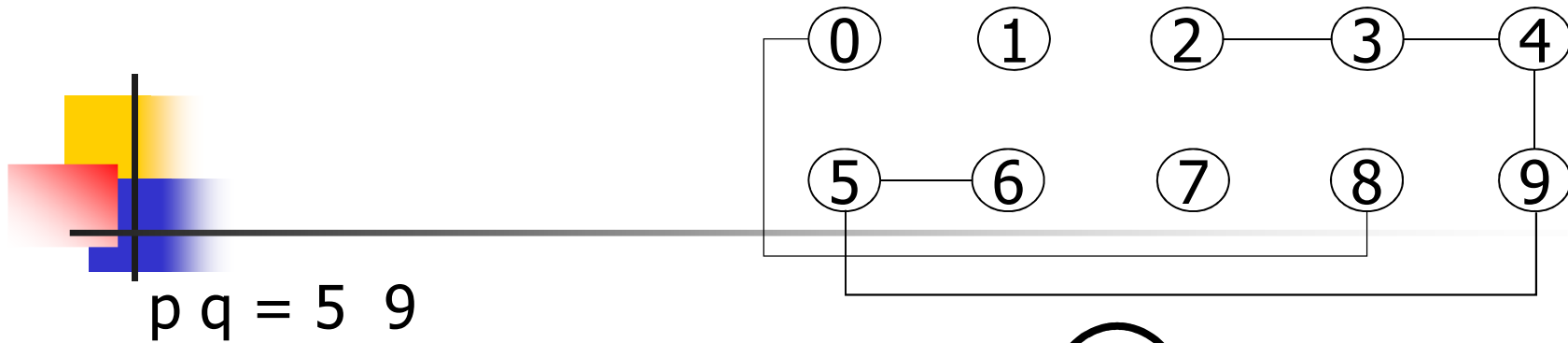




id[id[p]]=9 = id[q]=9
no change

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

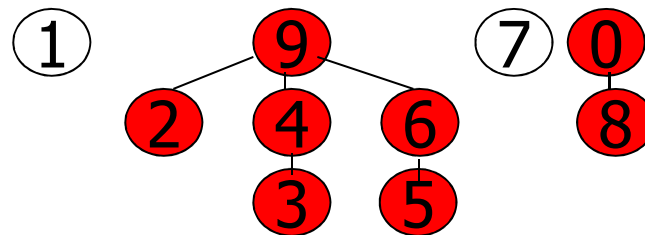


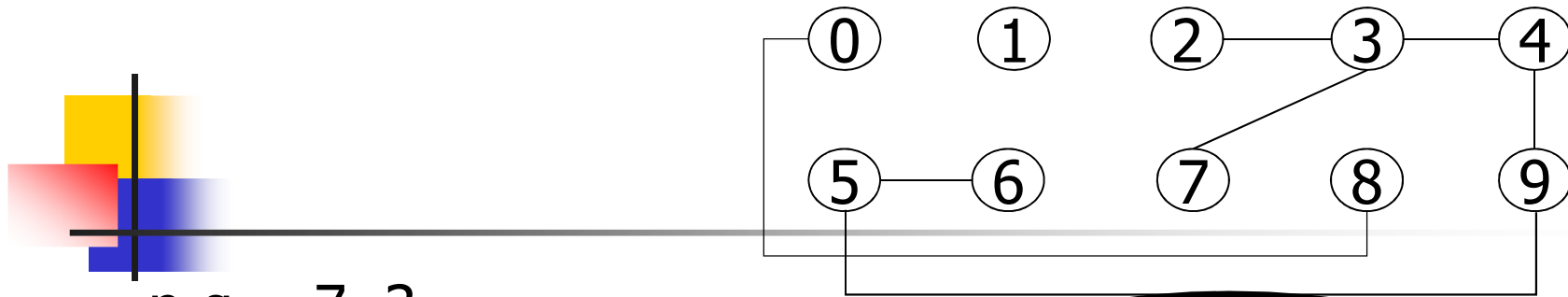


id	0	1	9	4	9	6	6	7	0	9
	0	1	2	3	4	5	6	7	8	9

$\text{id}[\text{id}[p]] = 6 \neq \text{id}[q] = 9$
 p points to q : $\text{id}[\text{id}[p]] = 9$

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





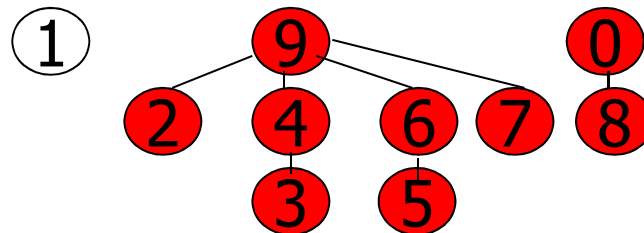
id

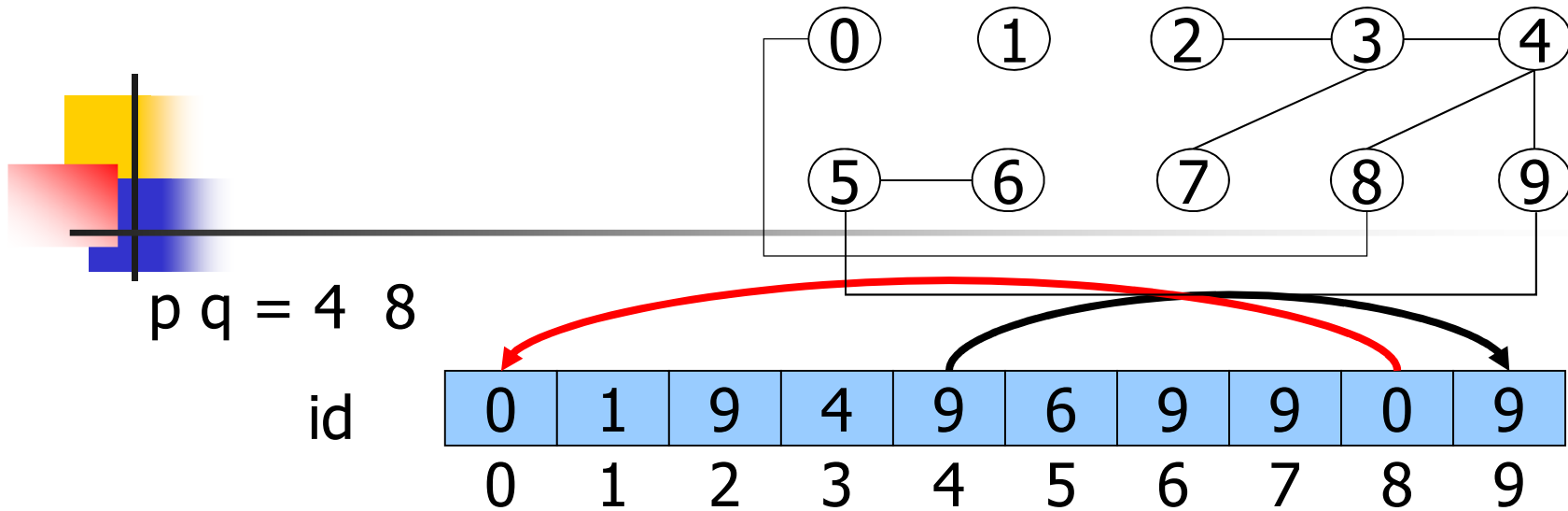
0	1	9	4	9	6	9	7	0	9
0	1	2	3	4	5	6	7	8	9

$id[p]=7 \neq id[id[id[q]]]=9$
 p points to q: $id[p]=9$

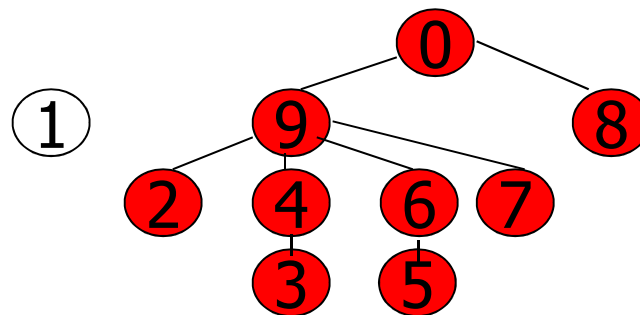
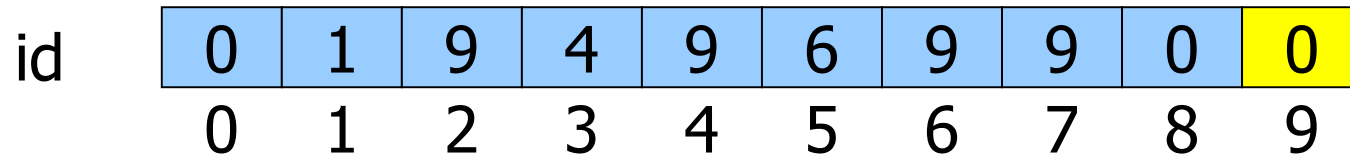
id

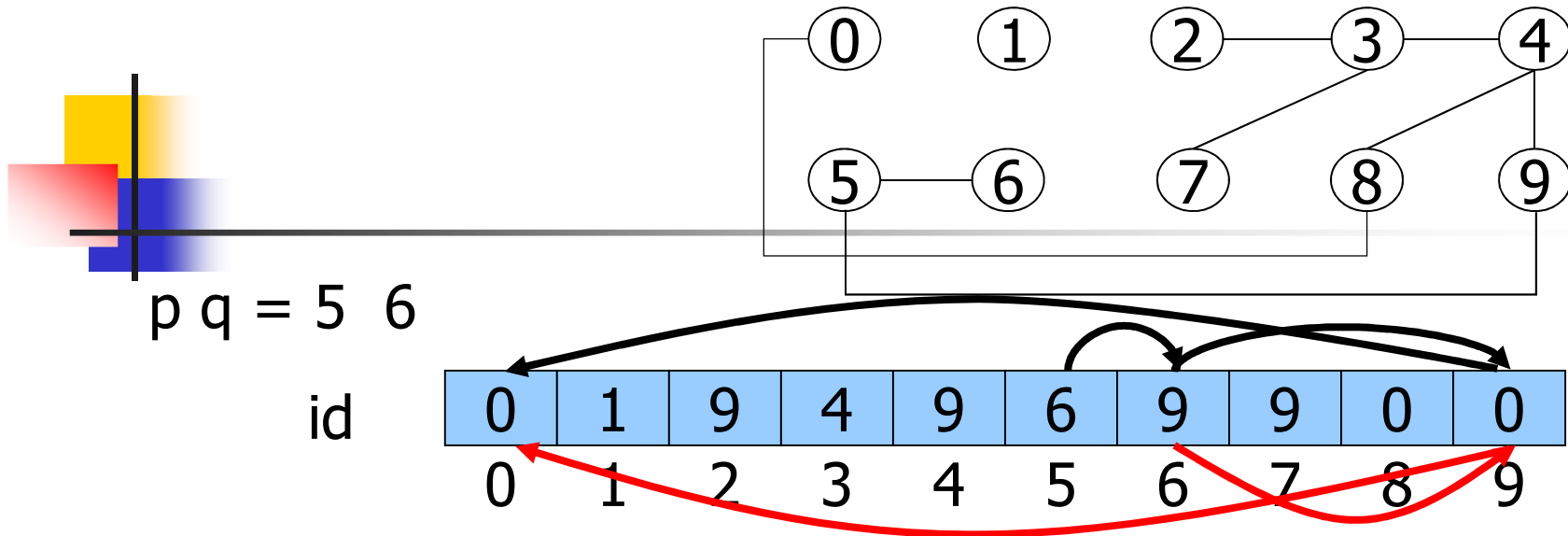
0	1	9	4	9	6	9	9	0	9
0	1	2	3	4	5	6	7	8	9



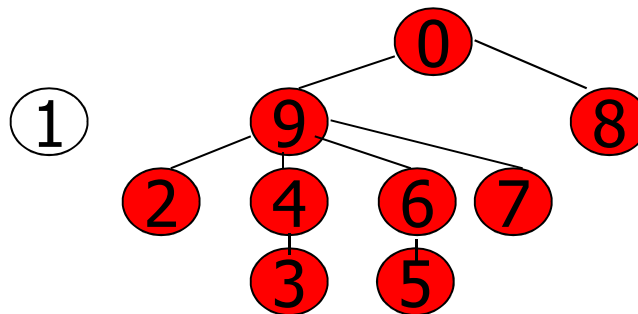
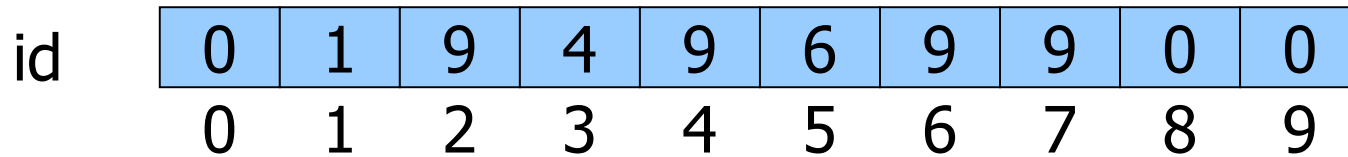


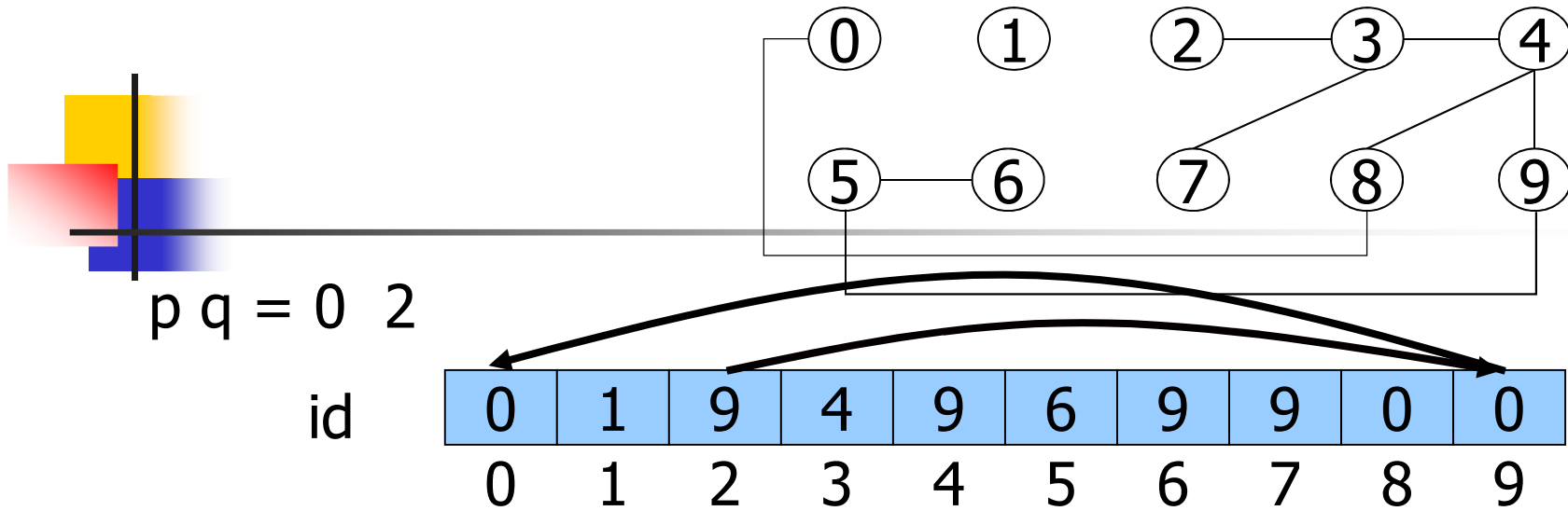
$\text{id}[\text{id}[p]] = 9 \neq \text{id}[\text{id}[q]] = 0$
 p points to q: $\text{id}[\text{id}[p]] = 0$



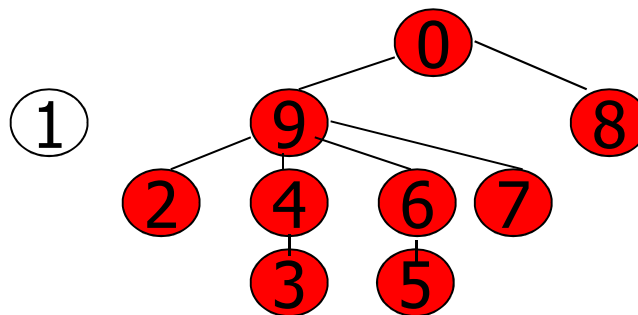
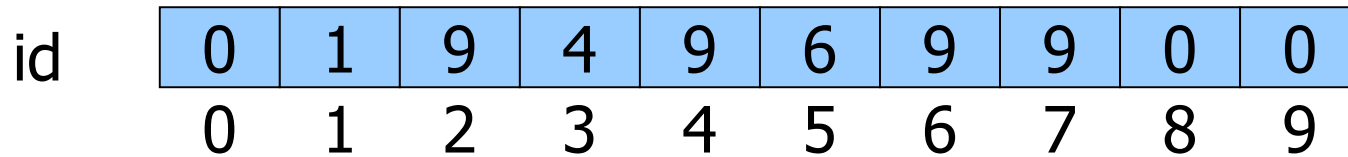


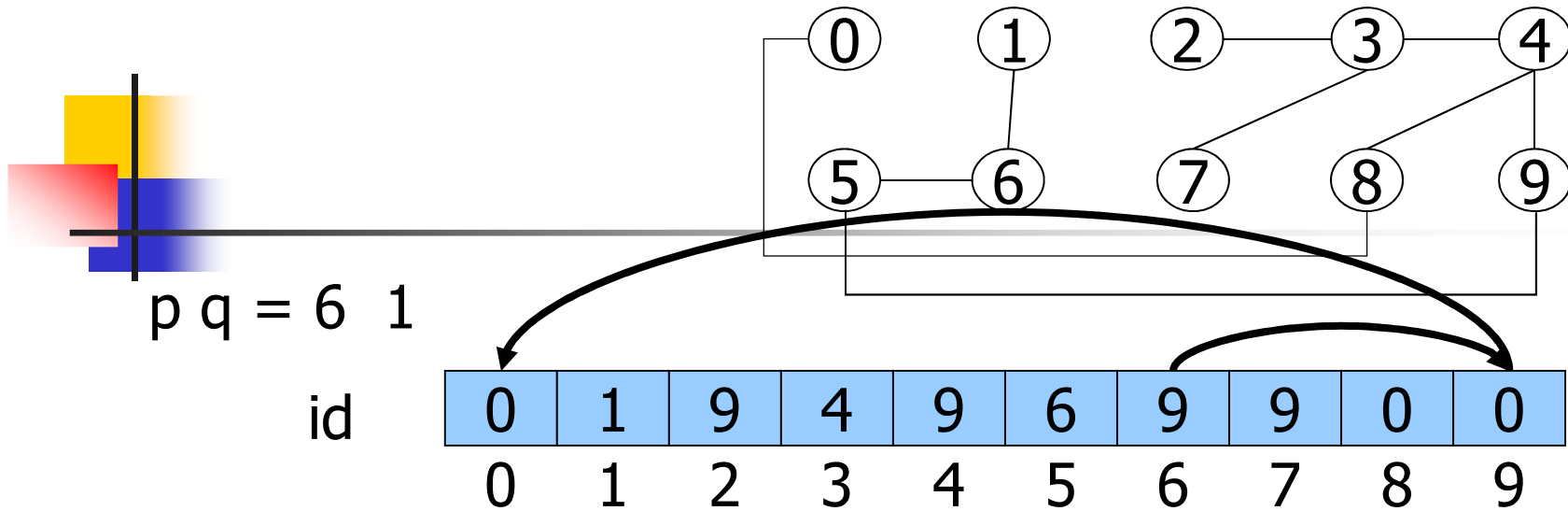
$id[id[id[p]]] = 0 = id[id[q]] = 0$
no change



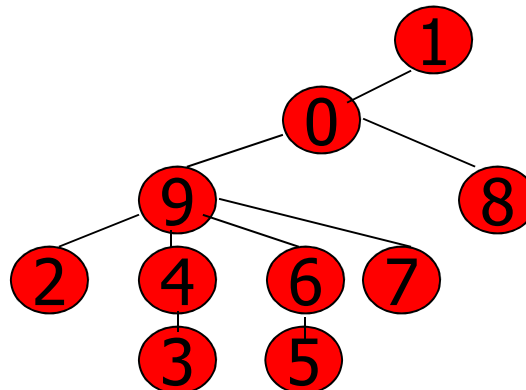
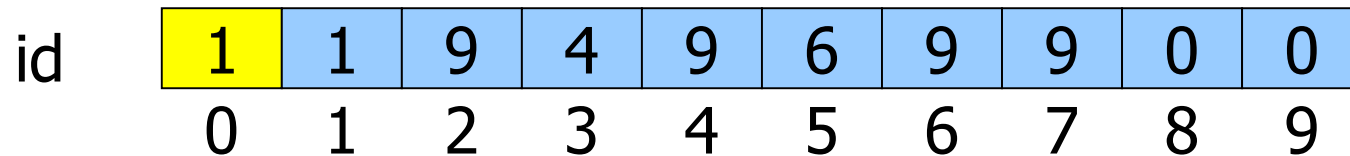


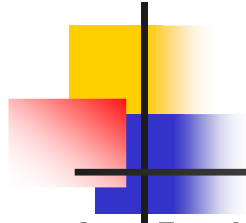
$\text{id}[p]=0 = \text{id}[\text{id}[\text{id}[q]]]=0$
no change



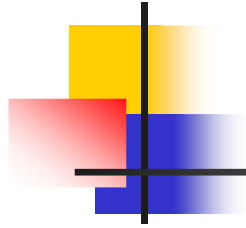


$\text{id}[\text{id}[\text{id}[p]]] = 0 \neq \text{id}[q] = 1$
 p points to q: $\text{id}[\text{id}[\text{id}[p]]] = 1$





```
#include <stdio.h>
#define N 10000
main() {
    int i, j, p, q, id[N];
    for(i=0; i<N; i++)
        id[i] = i;
    printf("Input pair p q: ");
    while (scanf("%d %d", &p, &q) == 2) {
        for (i = p; i != id[i]; i = id[i]);
        for (j = q; j != id[j]; j = id[j]);
        if (i == j) {
            printf("pair %d %d already connected\n", p,q);
        } else {
            id[i] = j;
            printf("pair %d %d not yet connected\n", p, q);
        }
        printf("Input pair p q: ");
    }
}
```

Performance

■ Find

- Scan a “chain” of objects, upper bound linear cost in the number of objects, in general well below upper bound

■ Union

- Simple, as it is enough that an object points to another object, unit cost

■ Overall number of operations related to

pairs · chain length

Still
too slow

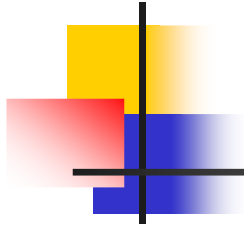


Quick union optimizations

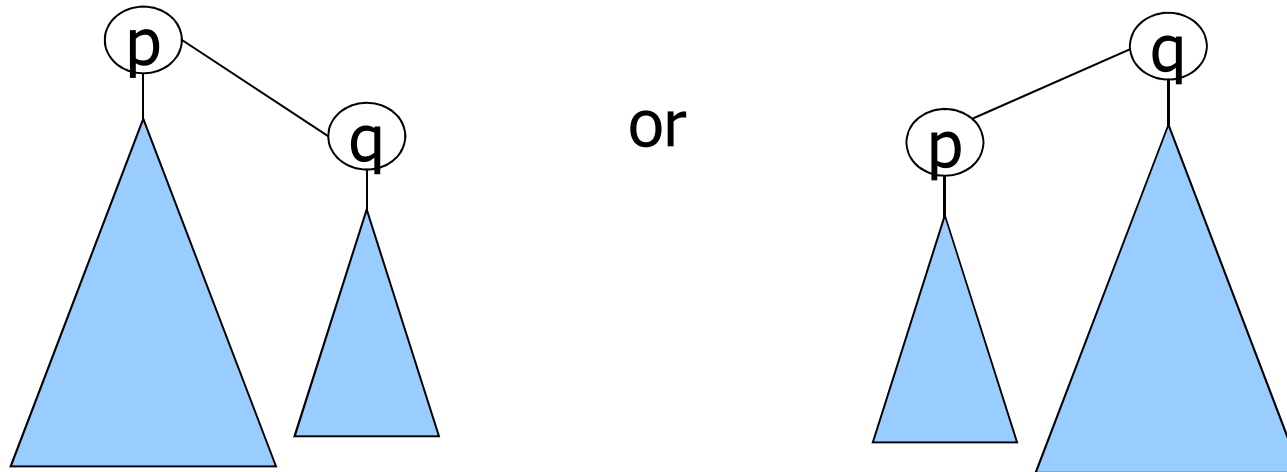
Weighted quick union

- To shorten the chain length, keep track of the number of elements in each tree (array `sz`) and connect the smaller tree to the larger one

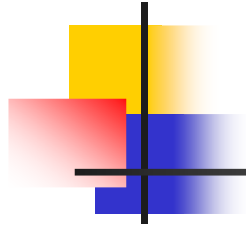
Union by height or "rank",
i.e., always link the
root of smaller tree
to root of larger tree



- According to which one is the larger, there might be 2 solutions



- It is irrelevant if p appears at the right or at the left of q



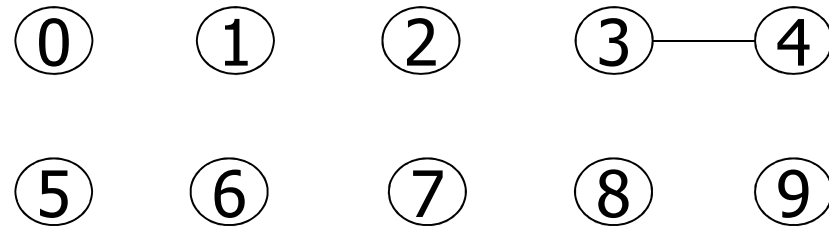
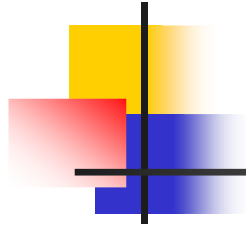
Example

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

Initially

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

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



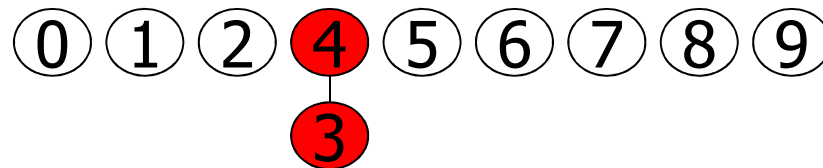
$p \ q = 3 \ 4$

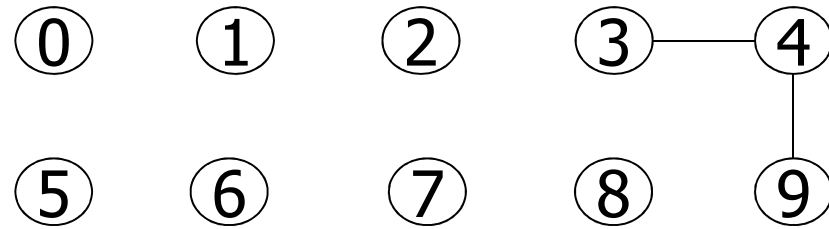
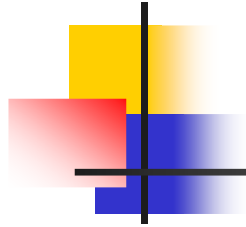
id	0	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

$\text{id}[p]=3 \neq \text{id}[q]=4$

p points to q : $\text{id}[p]=4$

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





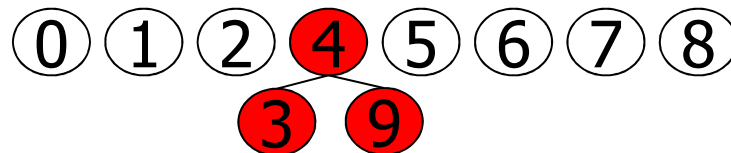
p q = 4 9

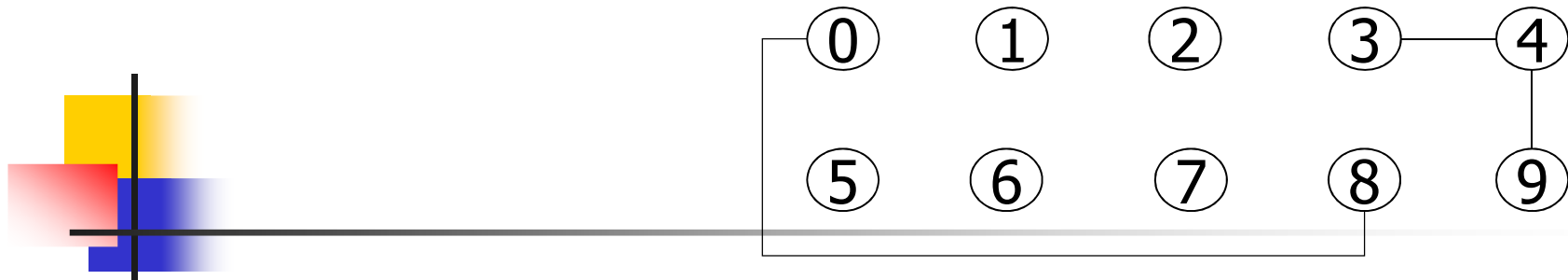
id	0	1	2	4	4	5	6	7	8	9
	0	1	2	3	4	5	6	7	8	9

$\text{id}[p]=4 \neq \text{id}[q]=9$

the smaller tree q points to the larger one p: $\text{id}[q]=4$

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





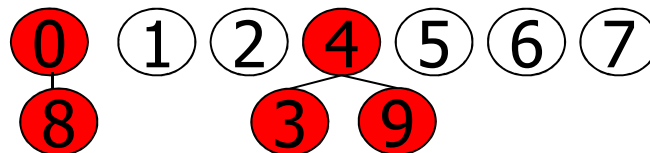
$p \ q = 8 \ 0$

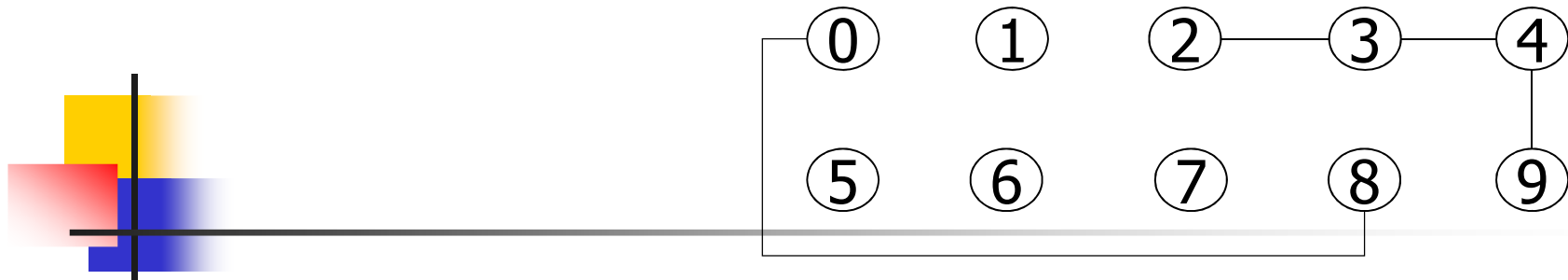
id	0	1	2	4	4	5	6	7	8	4
	0	1	2	3	4	5	6	7	8	9

$id[p]=8 \neq id[q]=0$

p points to q : $id[p]=0$

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





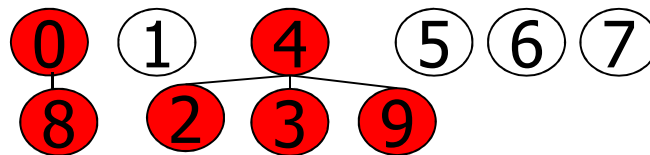
$p \ q = 2 \ 3$

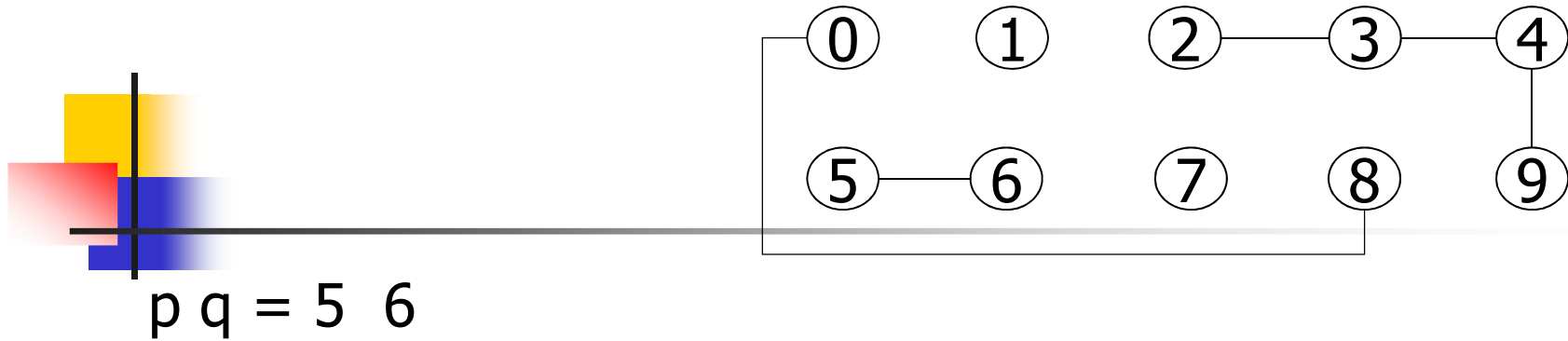
id	0	1	2	4	4	5	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

$id[p]=2 \neq id[id[q]]=4$

the smaller tree p points to the larger one q : $id[p]=4$

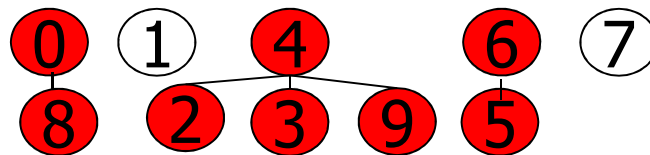
id	0	1	4	4	4	5	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

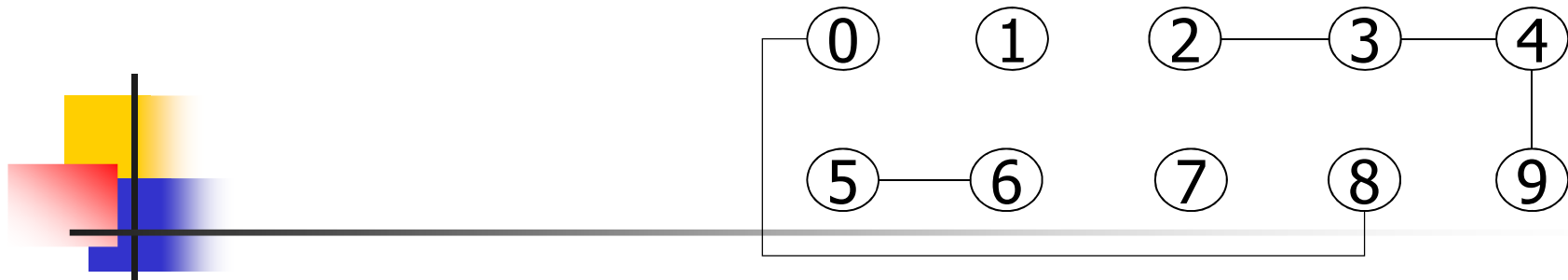




$id[p]=5 \neq id[q]=6$
 p points to q : $id[p]=6$

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



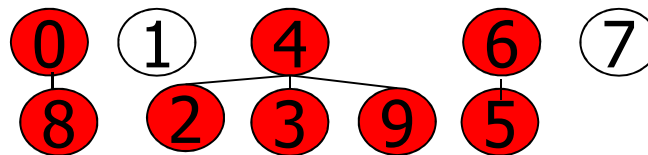


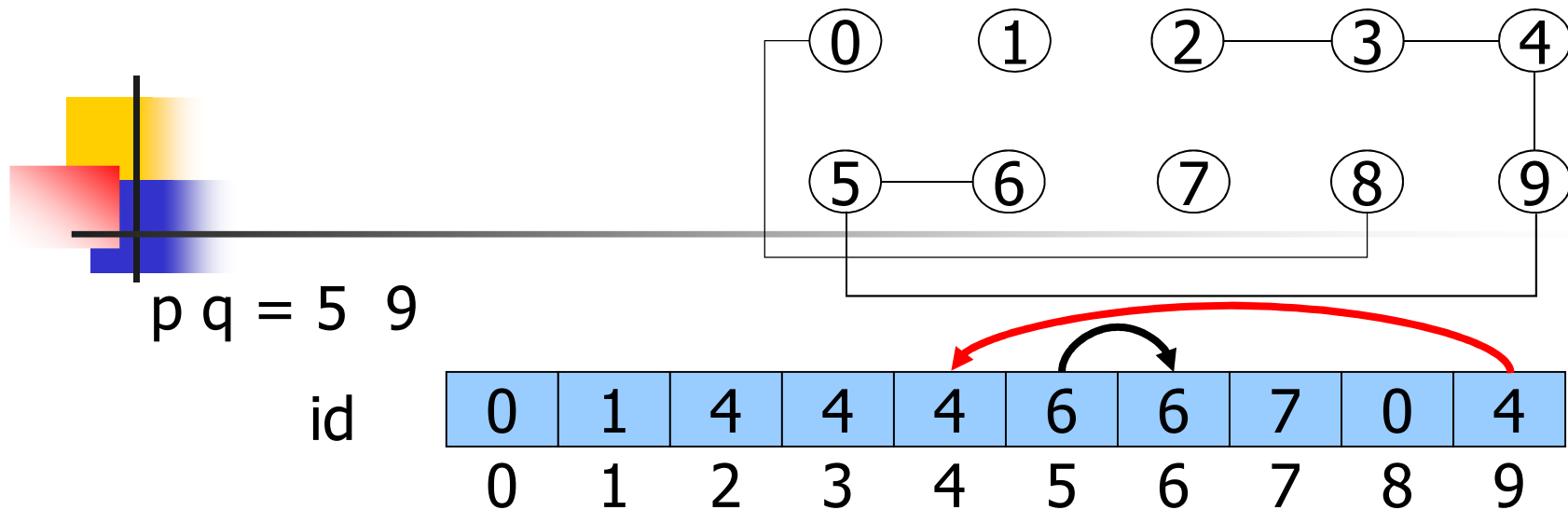
p q = 2 9

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

id[id[p]]=4 = id[q]=4
no change

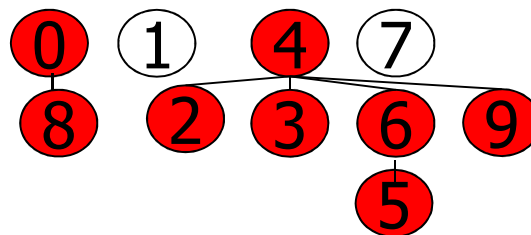
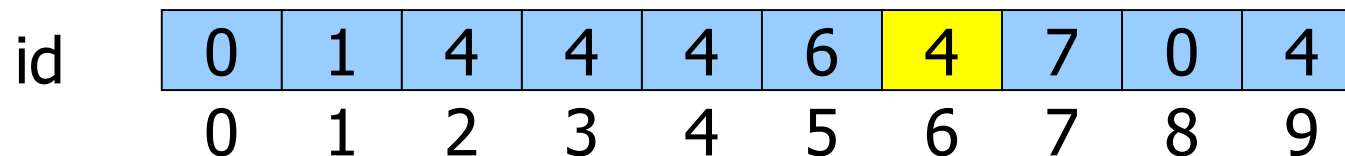
id	0	1	4	4	4	6	6	7	0	4
	0	1	2	3	4	5	6	7	8	9

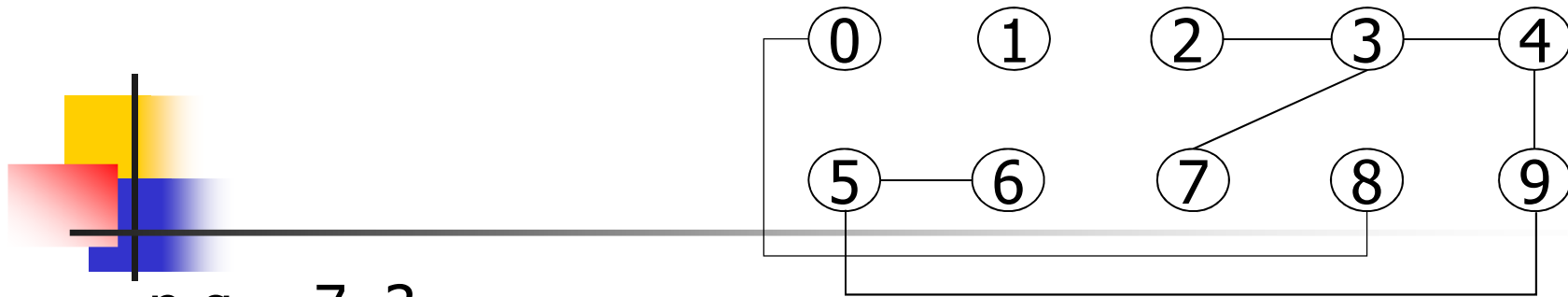




$\text{id}[\text{id}[p]] = 6 \neq \text{id}[\text{id}[q]] = 4$

the smaller tree p points to the larger one q : $\text{id}[\text{id}[p]] = 4$





p q = 7 3

id

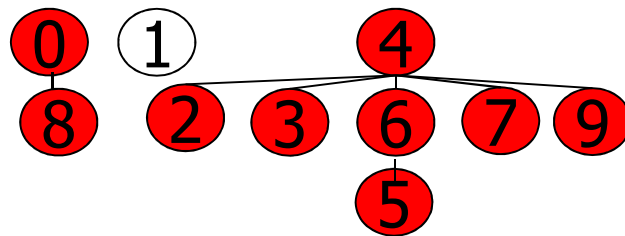
0	1	4	4	4	6	4	7	0	4
0	1	2	3	4	5	6	7	8	9

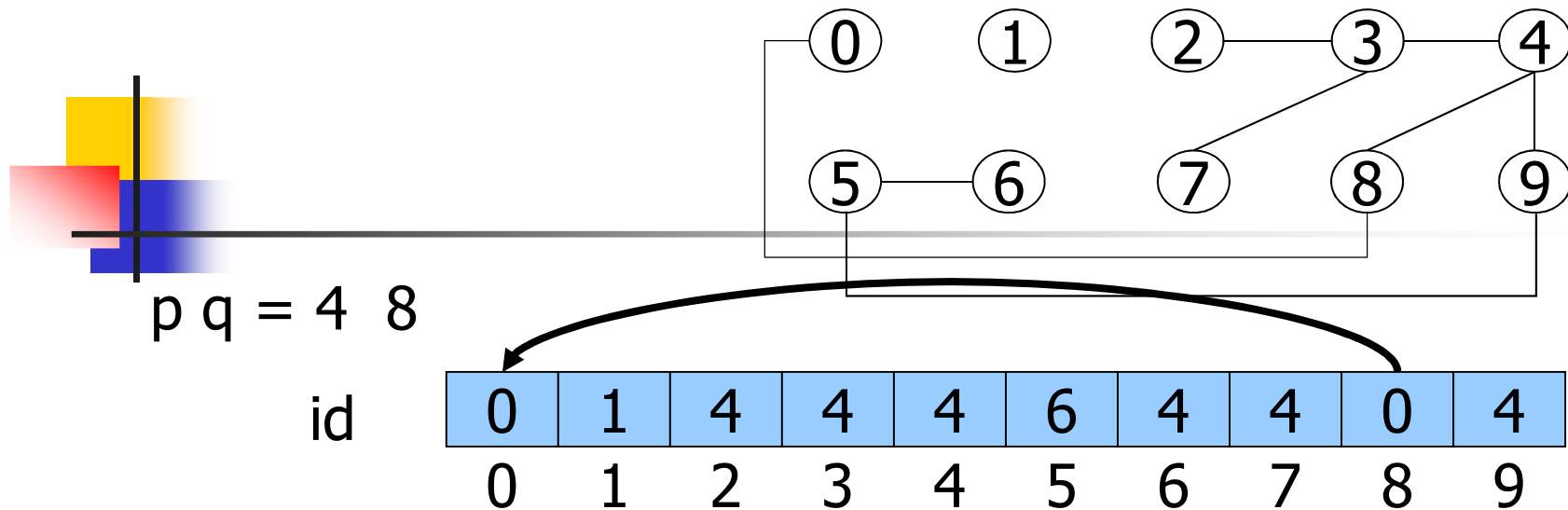
$\text{id}[p]=7 \neq \text{id}[\text{id}[q]]=4$

the smaller tree p points to the larger one q: $\text{id}[p]=4$

id

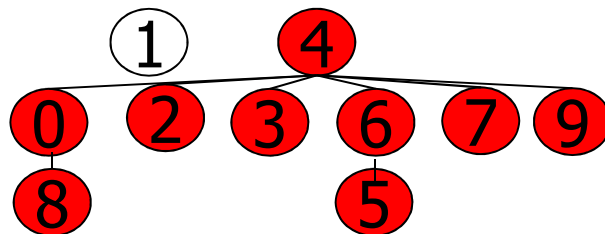
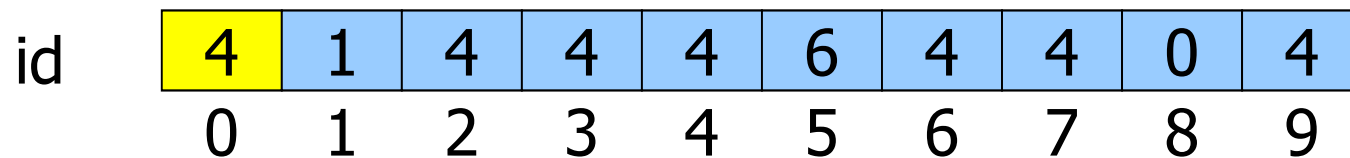
0	1	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9

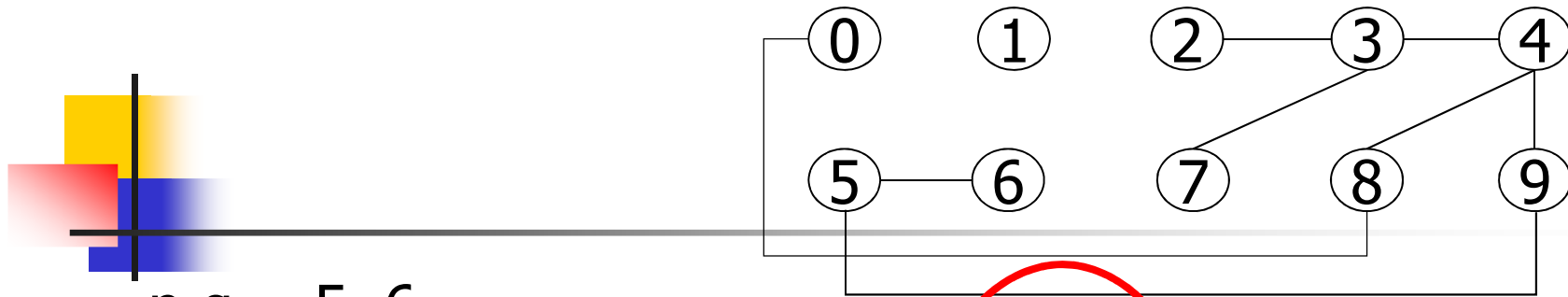




$\text{id}[p]=4 \neq \text{id}[\text{id}[q]]=0$

the smaller tree q points to the larger one p : $\text{id}[\text{id}[q]]=4$





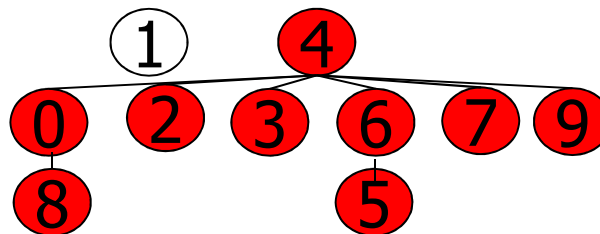
id

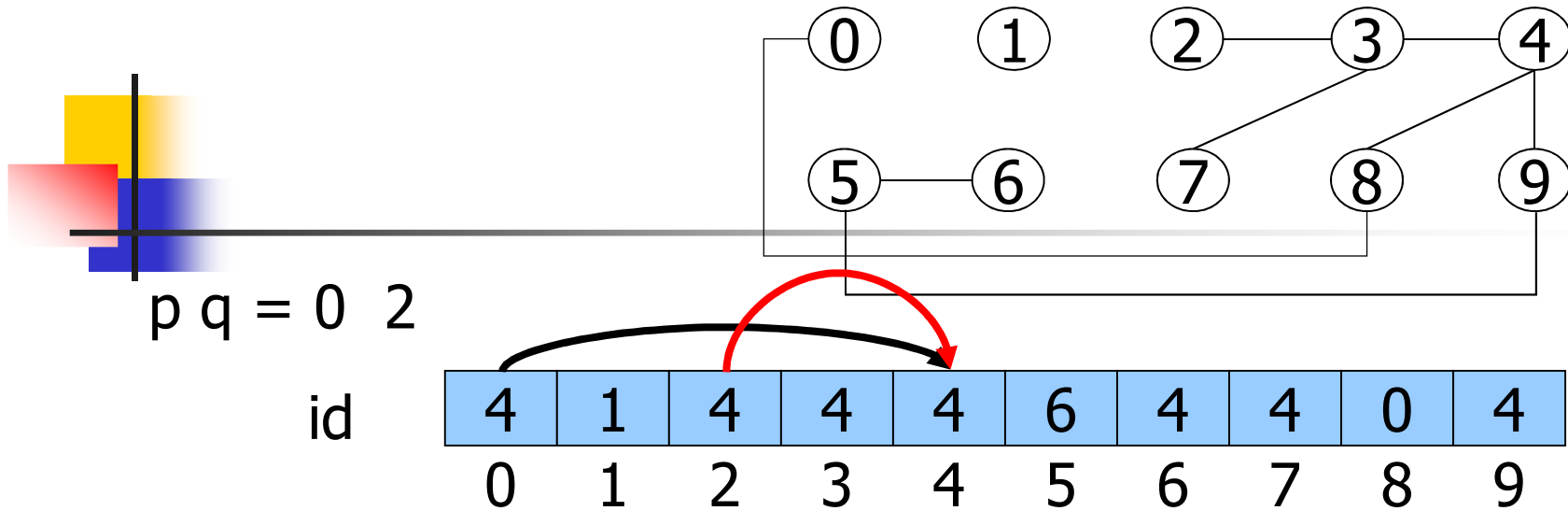
4	1	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9

$\text{id}[\text{id}[\text{id}[p]]] = 4 = \text{id}[\text{id}[q]] = 4$
no change

id

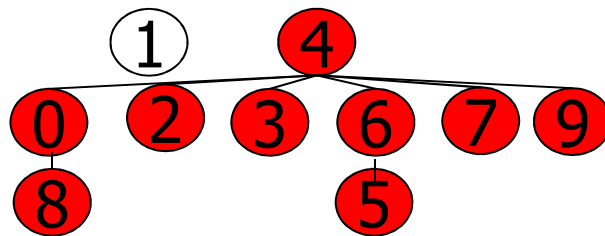
4	1	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9

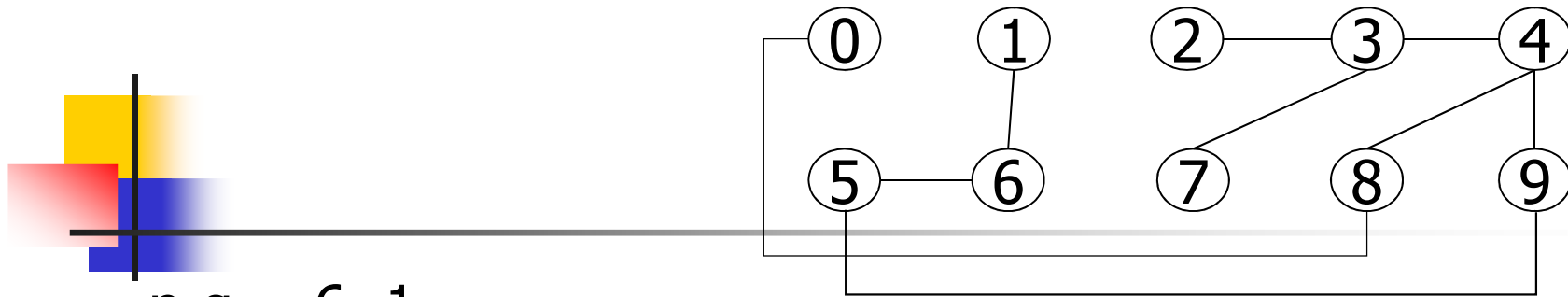




$\text{id}[\text{id}[p]] = 4 = \text{id}[\text{id}[q]] = 4$
no change

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





id

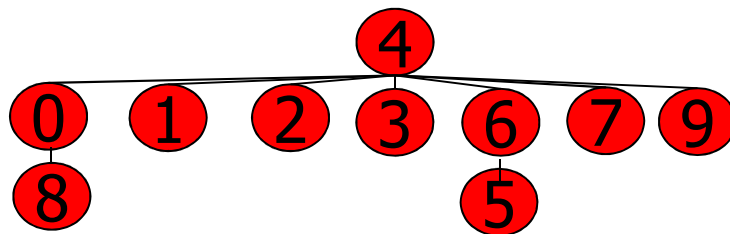
4	1	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9

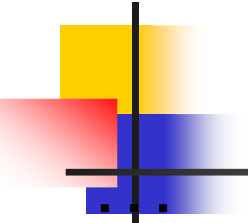
$\text{id}[\text{id}[p]] = 4 \neq \text{id}[q] = 1$

the smaller tree q points to the larger one p : $\text{id}[q] = 4$

id

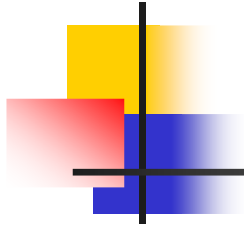
4	4	4	4	4	6	4	4	0	4
0	1	2	3	4	5	6	7	8	9





```
int i, j, p, q, id[N], sz[N];
for(i=0; i<N; i++) {
    id[i] = i; sz[i] =1;
}
printf("Input pair p q: ");
while (scanf("%d %d", &p, &q) ==2) {
    for (i = p; i!= id[i]; i = id[i]);           find root
    for (j = q; j!= id[j]; j = id[j]);
    if (i == j)
        printf("pair %d %d already connected\n", p,q);
    else {
        printf("pair %d %d not yet connected\n", p, q);
        if (sz[i] < sz[j]) {
            id[i] = j; sz[j] += sz[i];
        }
        else {
            id[j] = i; sz[i] += sz[j];
        }
    }
}
...
```

小树连大树



■ Find

- Scanning a “chain” of objects, cost at most logarithmic in the number of objects

■ Union

- Simple, because it is enough that an object points to another object, unit cost

■ Globally the number of operations is bounded by

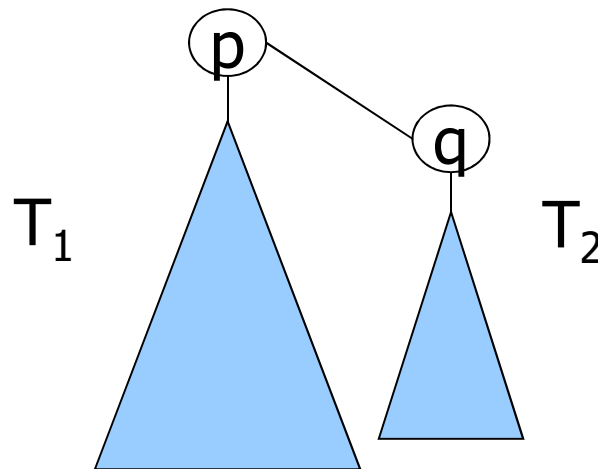
numb. of pairs * “chain” length

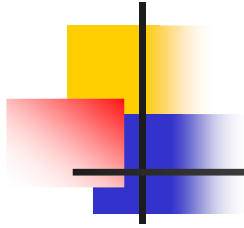
but chain length grows logarithmically !



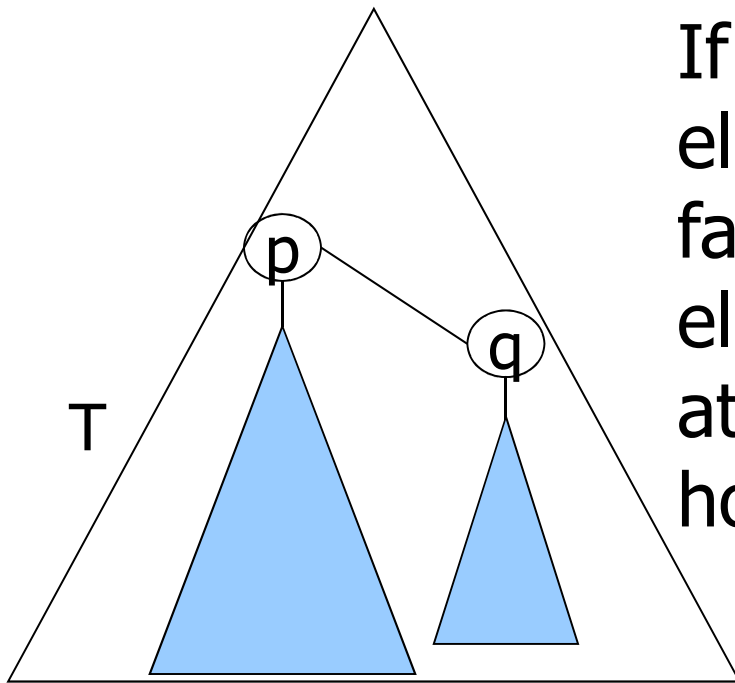
Why logarithmically?

What matters is the maximum distance between a node and the root. The distance increases by 1 when we connect a smaller tree (whose size is T_2) to a larger tree (whose size is T_1).





But if $T_1 \geq T_2$ each time we connect a smaller tree to a larger one we generate a tree whose size T is at least twice as big as T_2 .



If at each step the number of elements increases by at least a factor 2 and if there are N elements, after i steps there will be at least 2^i elements. $2^i \leq N$ must hold, thus $i \leq \log_2 N$