CS 251, Fall 2017

Data Structures and Algorithms

PURDUE UNIVERSITY

Professor Mike Atallah (LE 2)
Professor Xavier Tricoche (LE 1)

Overview

- **Course overview**
- **Introductory example: *Union find***

Course information

Lectures:

- Lecture 2: MWF 9:30am-10:20am, MATH 175 (Atallah)
- Lecture 1: MWF 10:30am-11:20am, MATH 175 (Tricoche)

Office hours:

- Prof. Atallah: MWF after LE2, LWSN 2116D, matallah@purdue.edu
- Prof. Tricoche: MWF 1pm - 2pm, LWSN 3154P, xmt@purdue.edu

Teaching Assistants:

- Lukasz Burzawa
- Habiba Farrukh
- Priyank Jain
- Negin Karisani
- Hafiz Khalil
- Seunghoon Lee
- Christopher May
- Meher Pindiprolu
- Dakshil Shah

TA Mailing list:

cs251-fall17-ta@cs.purdue.edu

Always email the list instead individual TAs

Course information

Web page:

- https://www.cs.purdue.edu/homes/cs251
- General course info
- Syllabus
- Slides
- Project description and homework
- Schedule

Piazza:

- http://piazza.com/purdue/fall2017/cs251
- Project related Q&A's
- Announcements

Course goals

Program = Algorithm + Data Structures

- Algorithm: method for solving a problem.
- Data structure: method to store information.
- CS251: programming and problem solving, with applications.

- In this course you will learn how the representation of data in the computer has an impact on the performance of a program
- We will cover several different types of data structures and algorithms that utilize these data structures
- You will also improve your programming skills

Course content

- Class logistics and initial example (today)
- Simple proof techniques (next time)
- Stacks and queues
- Program analysis
- Sorting algorithms
- Heaps and priority queues
- Trees
- Search trees
- Hash tables
- Graphs
- Text processing
- Compression

Course information

Textbook:

- Algorithms, Sedgewick and Wayne, 4th edition

Other course information:

- Read chapter/section before class
- Print slides and bring to class
- Take notes in class

Prerequisites

The class assumes that you have either (i) good Java background, or (ii) basic Java + OO programming background

- Data types
- Control statements
- Arrays, simple classes
- Inheritance and polymorphism
- Exceptions
- Interfaces and abstract classes

The algorithms will be presented in pseudocode or Java

Programming

Resources:

- Java API: https://docs.oracle.com/javase/7/docs/api/
- Basics: http://www.cs.princeton.edu/introcs/home/
- Style: https://google.github.io/styleguide/javaguide.html (Google guidelines)
- Style: https://www.securecoding.cert.org/confluence/display/java/Java+Coding+Guidelines

Course resources

Schedule:

- See course web page:
  - No lectures on Sept 4 (Labor day), Oct 2 (October Break), Nov 22,24 (Thanksgiving)
  - Midterm exam: evening: Oct 18, 8pm-10pm LILY G126 & 1105
  - Final exam: TBD
- Links to slides, assignments, and additional readings are on the schedule
  - Password protected, username: **cs251-fall17**, password: **algofs17**
  - Material is copyrighted: Do not distribute

Coursework and grading

Assignments: 50%

- 2 written homeworks: **10%** (2 x 5%)
- 5 programming projects: **40%** (5 x 8%)
- Due at 11:59pm via electronic submission

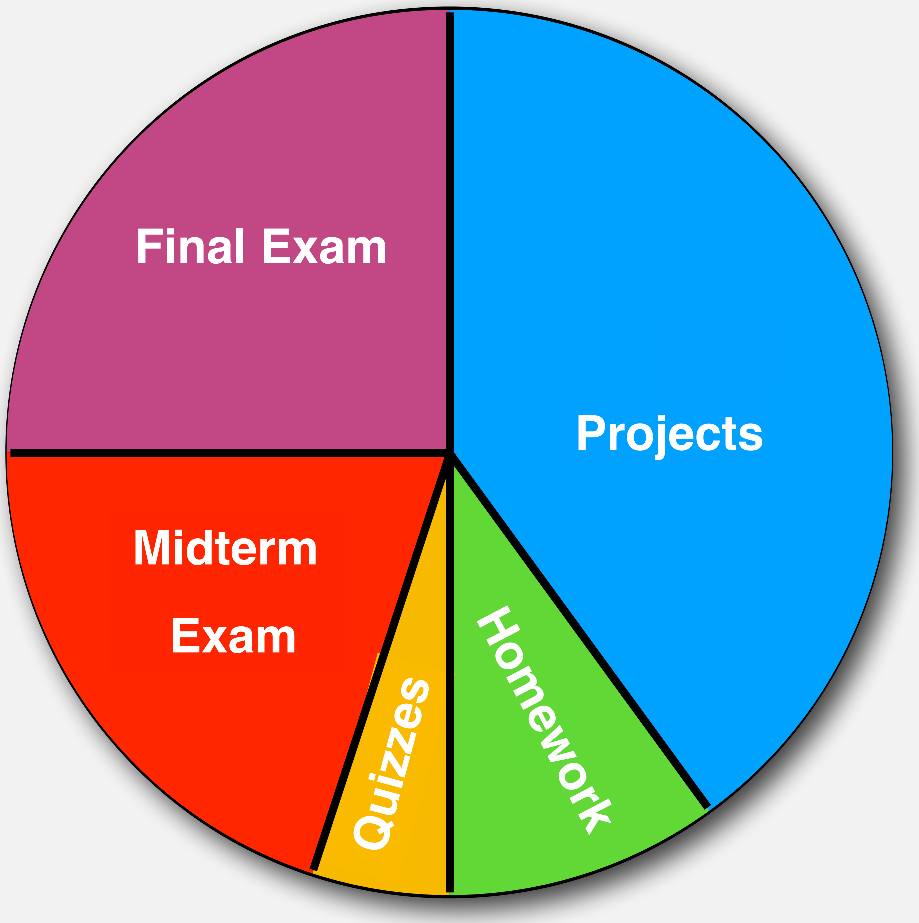
Quizzes: 5%

- iClicker

Exams: 45%

- Closed-book, closed notes.
- Midterm (evening exam, Oct 18): **20%**
- Final (scheduled by Registrar, TBD): **25%**
  - comprehensive but emphasis on topics covered after midterm

Grades will be reported on Blackboard



Course logistics

Email

- We will use an umbrella mailing list for class-wide (LE 1 and 2) announcements: fall-2017-cs-25100umbxmt001@lists.purdue.edu
- The mail alias cs251-fall17-ta@cs.purdue.edu is for contacting the TAs
- Use appropriate language when sending messages

Course logistics

Course content:

- The course moves very fast
- Attend all lectures
  - Lectures will assume that you have read the material from the text. We will build on that.
- Attendance for PSOs is highly recommended
- Quizzes will periodically check basic understanding of material and attendance

Lecture etiquette:

- Students are expected to focus their attention on the lecture (e.g., no distraction through electronic devices)
- No talking among students
- Before class allow instructor to prepare before asking questions

Course policies

Missing exams:

- If you cannot make an exam, contact the instructor BEFORE the exam, otherwise you will receive 0 on the exam
- Exceptions: documented medical and family emergencies only

Late policy:

- Each person will be allowed **4 days** of extensions which can be applied to any combination of assignments during the semester without penalty
  - Use of a partial day will be counted as a full day
  - Use of extension must be stated explicitly in the submission header or by email to the TAs, otherwise late penalties will apply
  - Extensions cannot be applied after the final day of classes
  - Extensions cannot be rearranged after they are granted. Use them wisely!
- After that a late **penalty of 20% per day** will be assigned
- Assignments will not be accepted if they are more than five days late

Course policies

Campus emergencies:

- Course requirements, deadlines, and grading are subject to change
- Course website and email list will be used to notify you
  - Emergencies include: pandemics, weather extremes, hazardous spills, safety issues, etc
- In case of contagious illness:
  - Do not attend lectures or PSOs
  - Contact instructor via email to make arrangements

Ethics

We encourage you to interact amongst yourselves:

- You may discuss and obtain help with basic concepts covered in lectures or the textbook, homework specification (but not solution), and program implementation (but not design)

However, this is NOT a team programming course:

- Work turned in should reflect your own efforts and knowledge.
- Sharing or copying solutions is unacceptable. It can result in failure for the course AND exclusion from Purdue (for repeated offenders).
- We use copy detection software, so do not copy code and make changes (either from the Web or from other students).
- You are expected to take reasonable precautions to prevent others from using your work.

Read and SIGN the Academic Integrity Policy on the web page

- Only those who have signed it will be allowed to take the midterm exam

Example: Union Find

- dynamic connectivity
- applications
- quick find

Subtext of today's example (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm (and data structure) to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

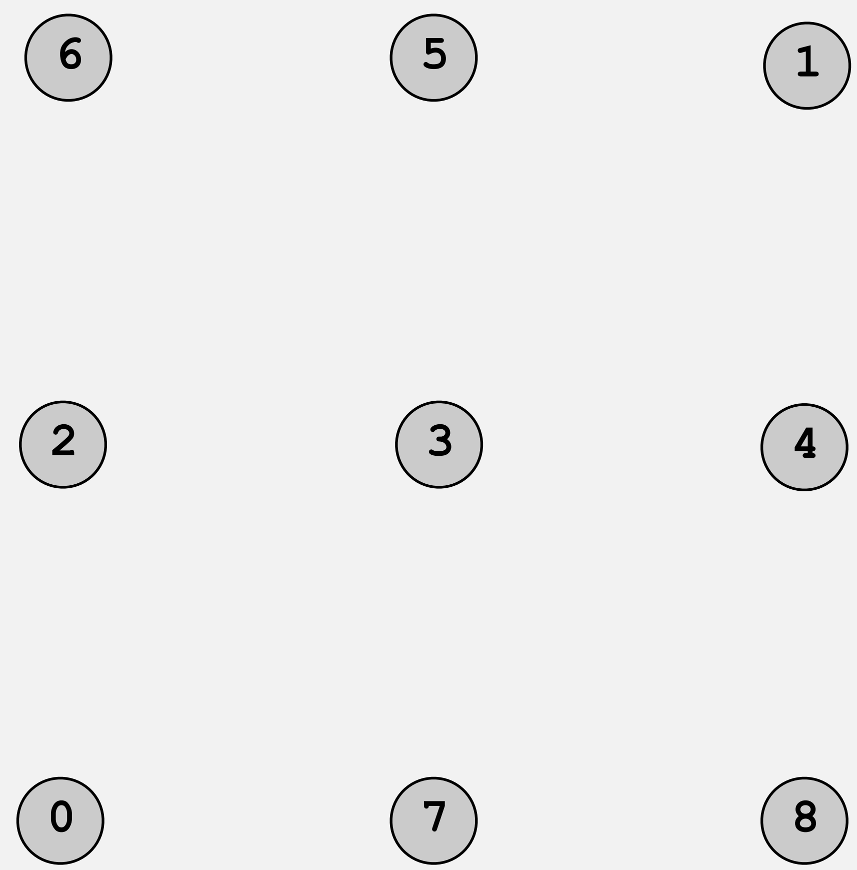
- dynamic connectivity
- applications
- quick find

Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path



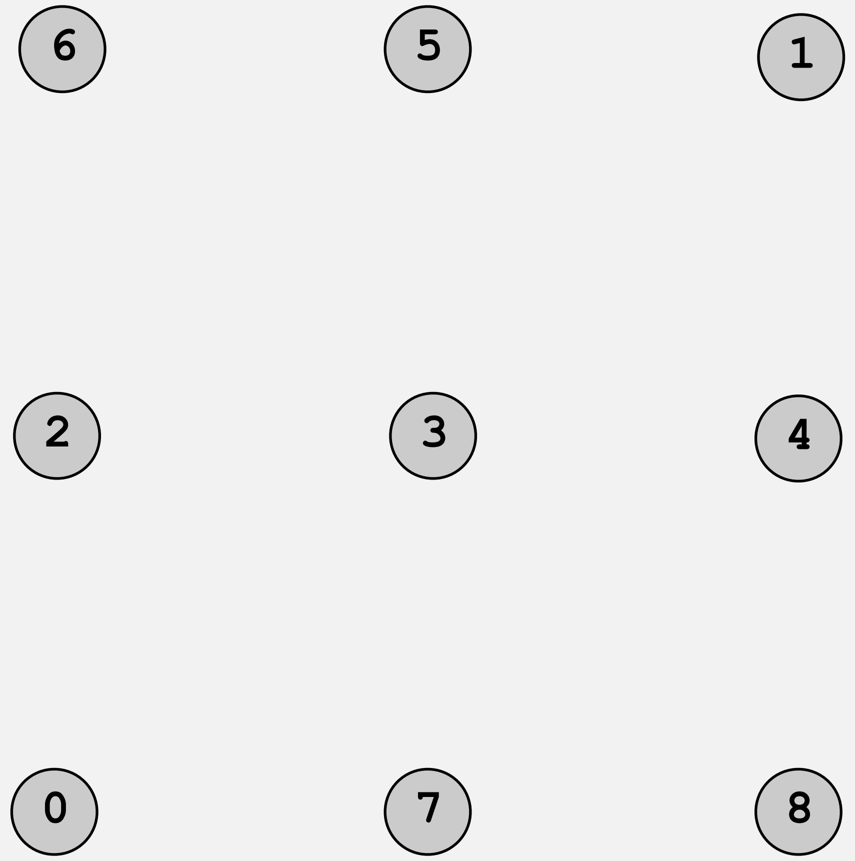
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
```



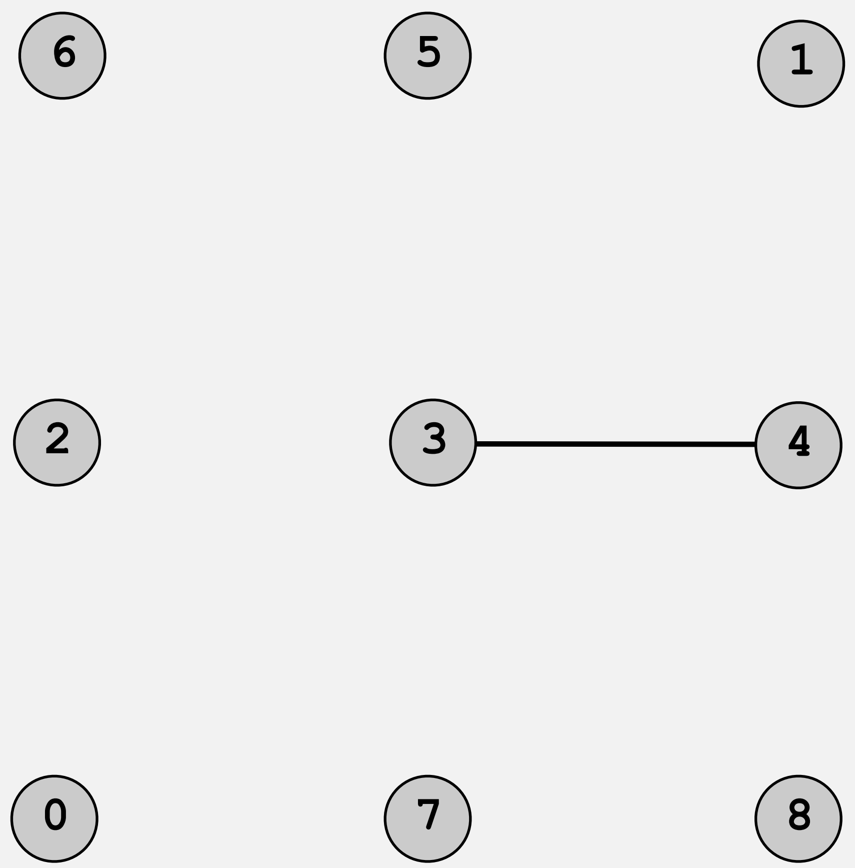
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
```



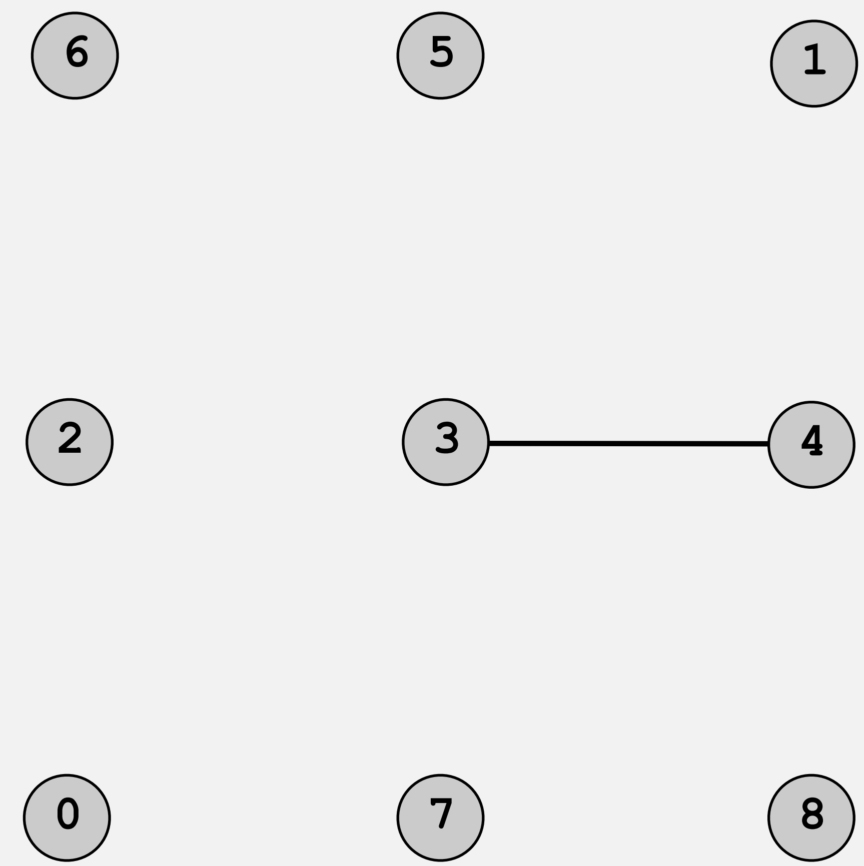
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
```



Dynamic connectivity

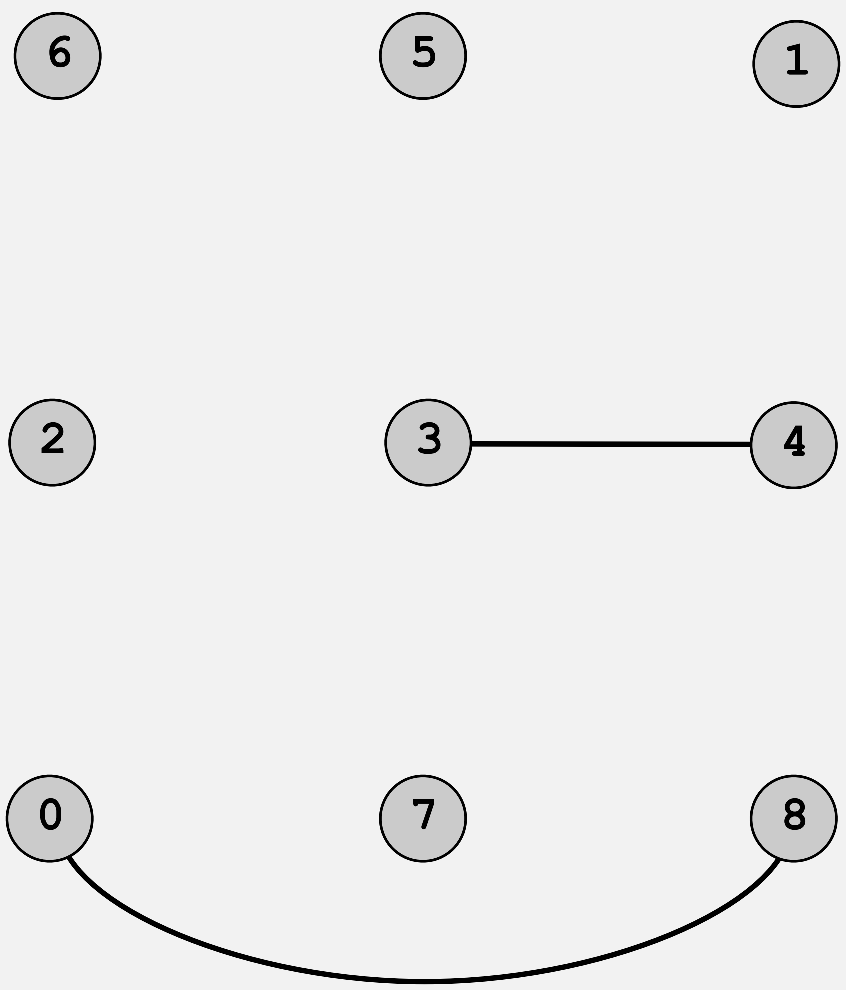
Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)

union(8, 0)
```



Dynamic connectivity

Given a set of objects

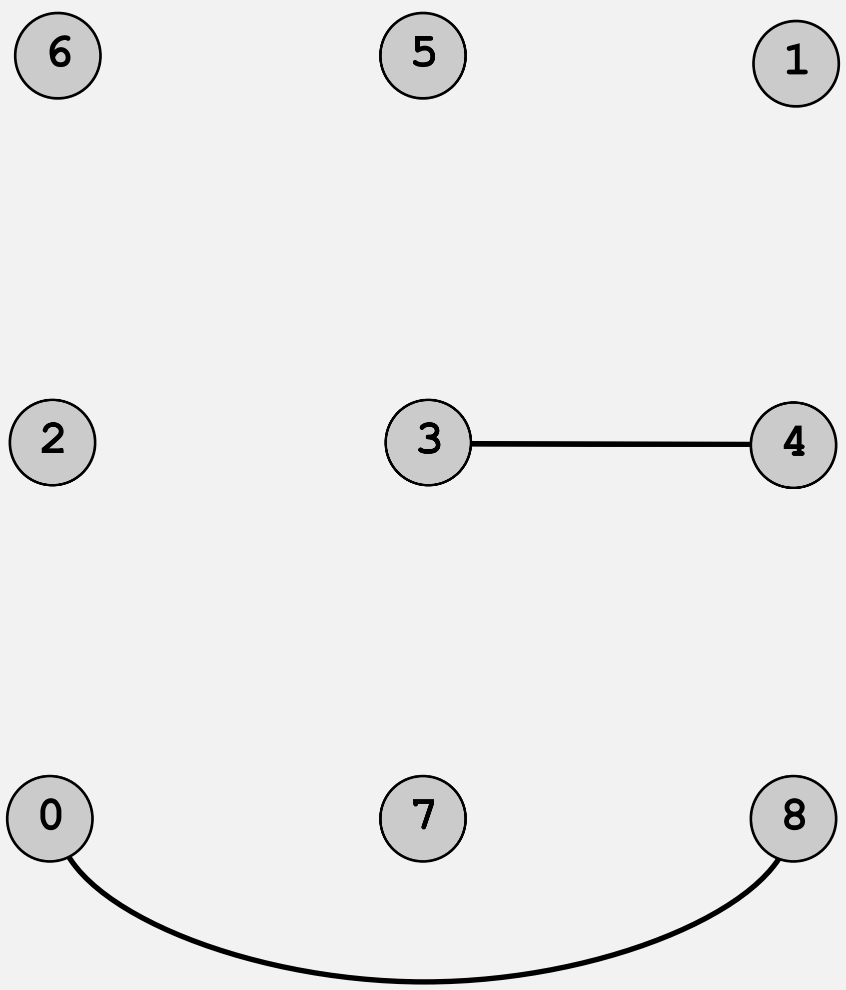
- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)

union(8, 0)

union(2, 3)
```



Dynamic connectivity

Given a set of objects

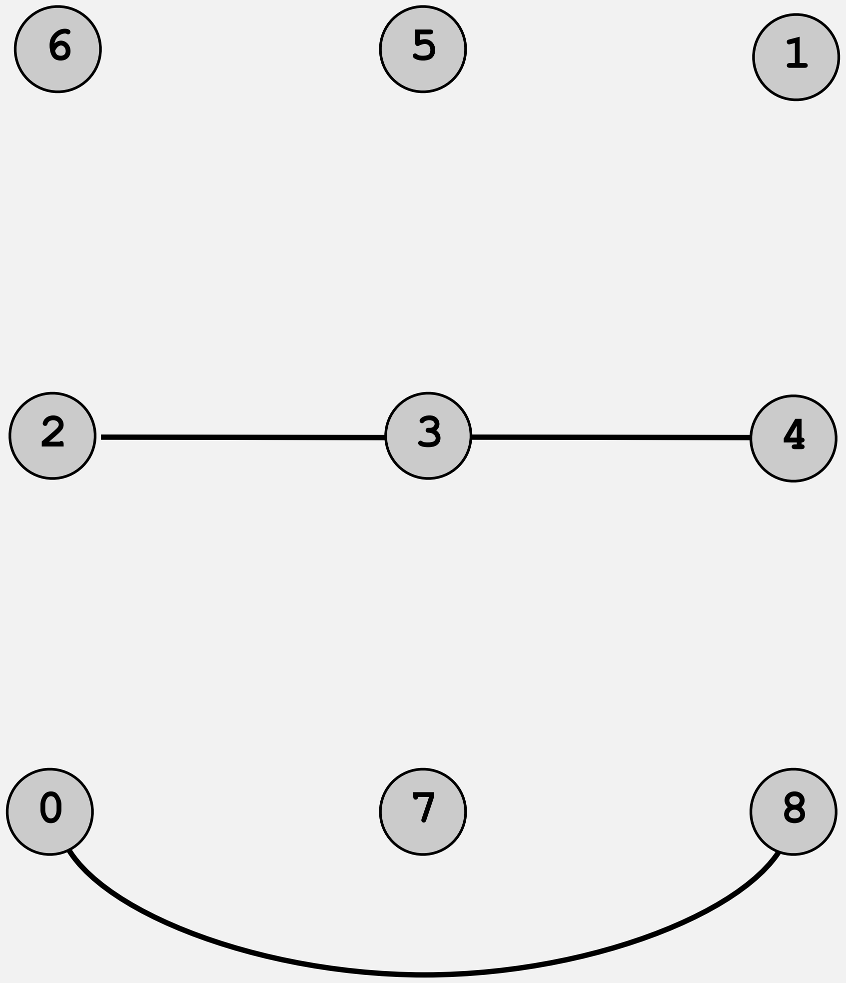
- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)

union(8, 0)

union(2, 3)
```



Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

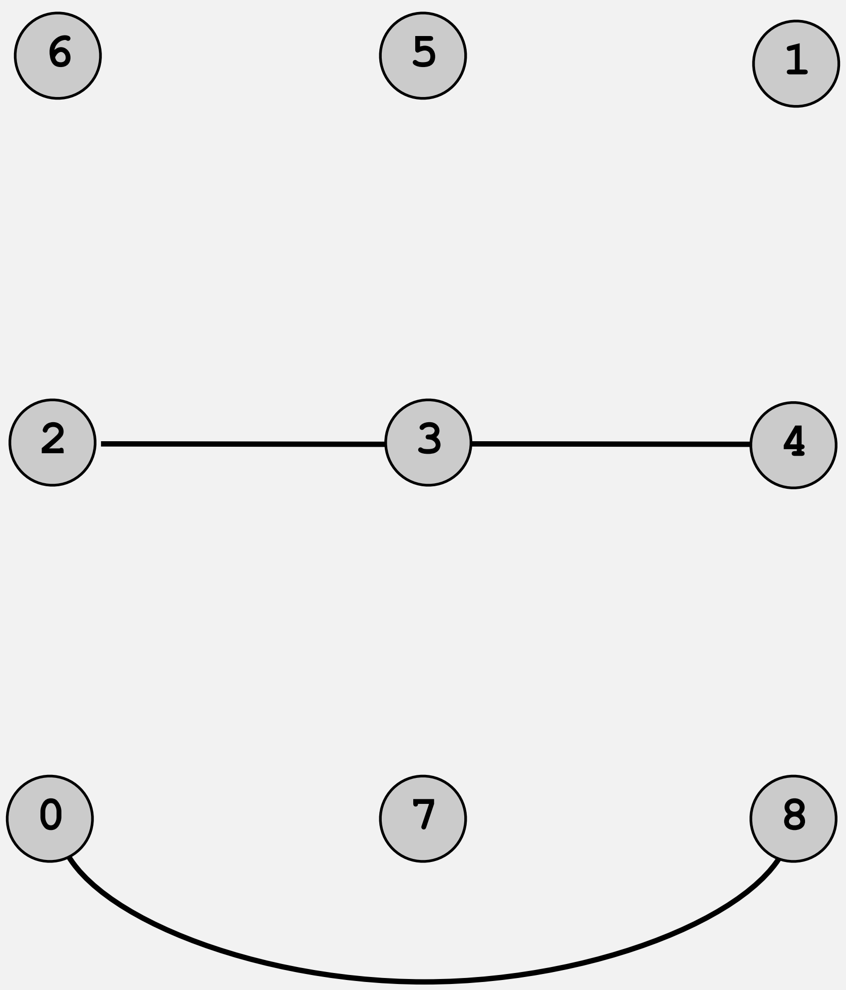
more difficult problem: find the path

```
union(3, 4)

union(8, 0)

union(2, 3)

union(5, 6)
```



Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

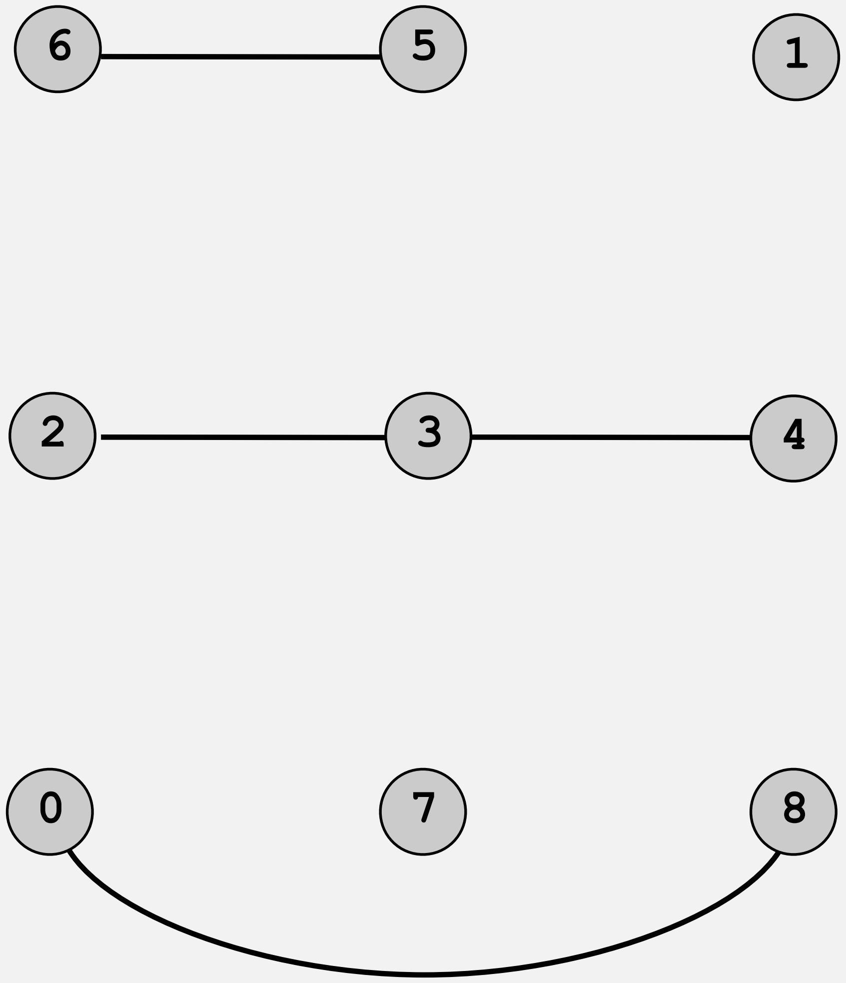
more difficult problem: find the path

```
union(3, 4)

union(8, 0)

union(2, 3)

union(5, 6)
```



Dynamic connectivity

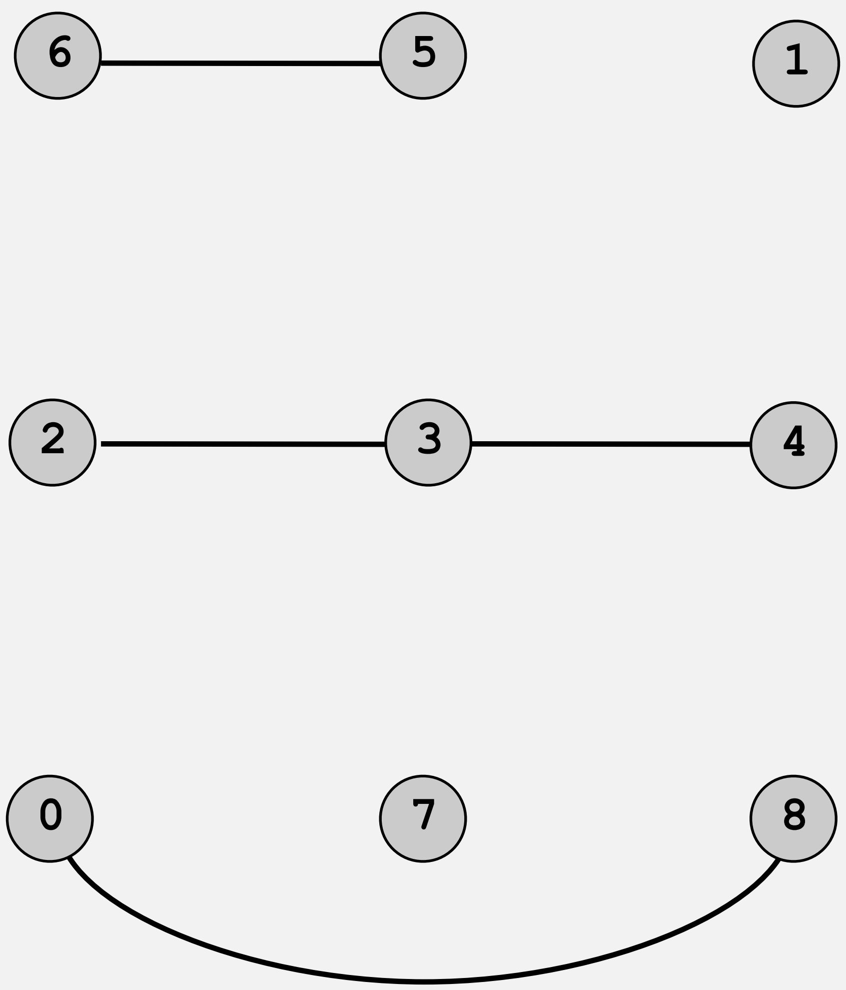
Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)
```

no



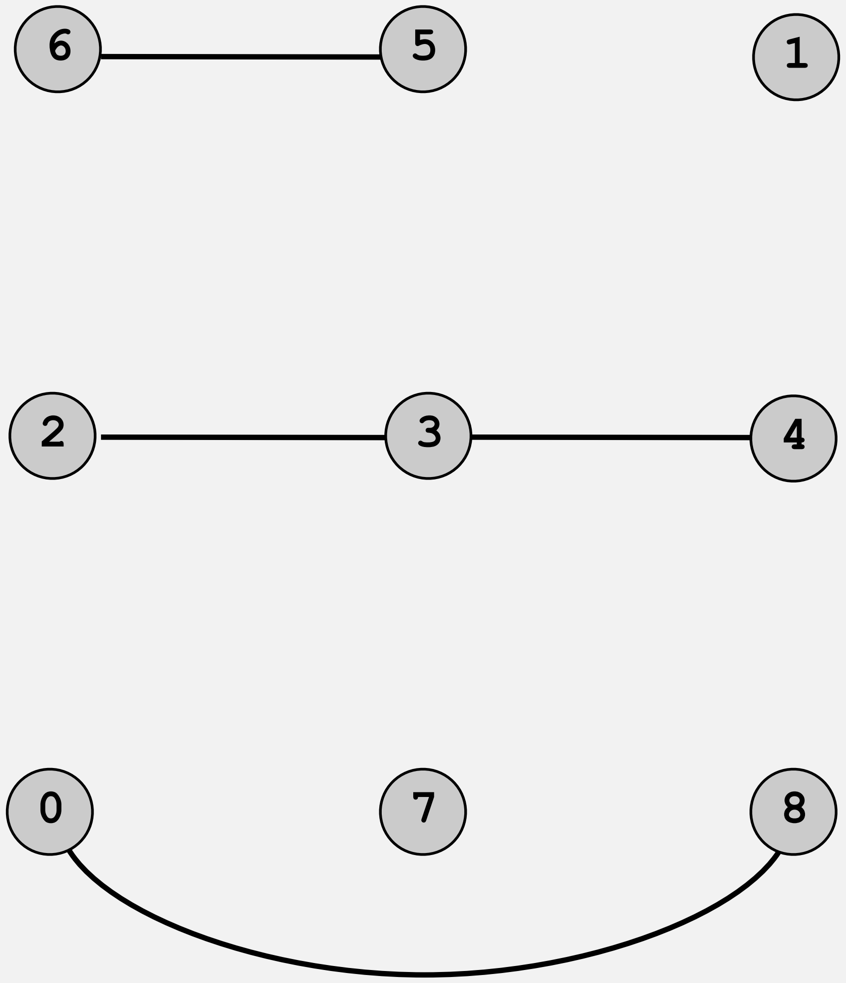
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)      no
 find(2, 4)      yes
```



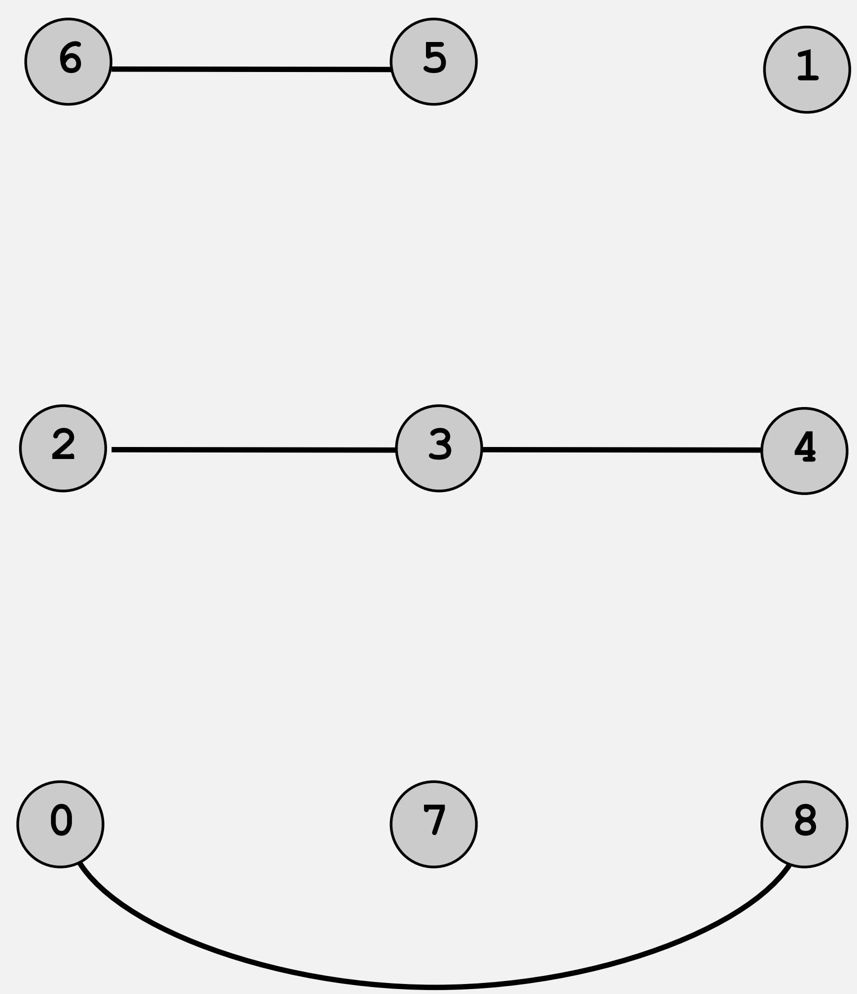
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)        no
 find(2, 4)        yes
union(5, 1)
```



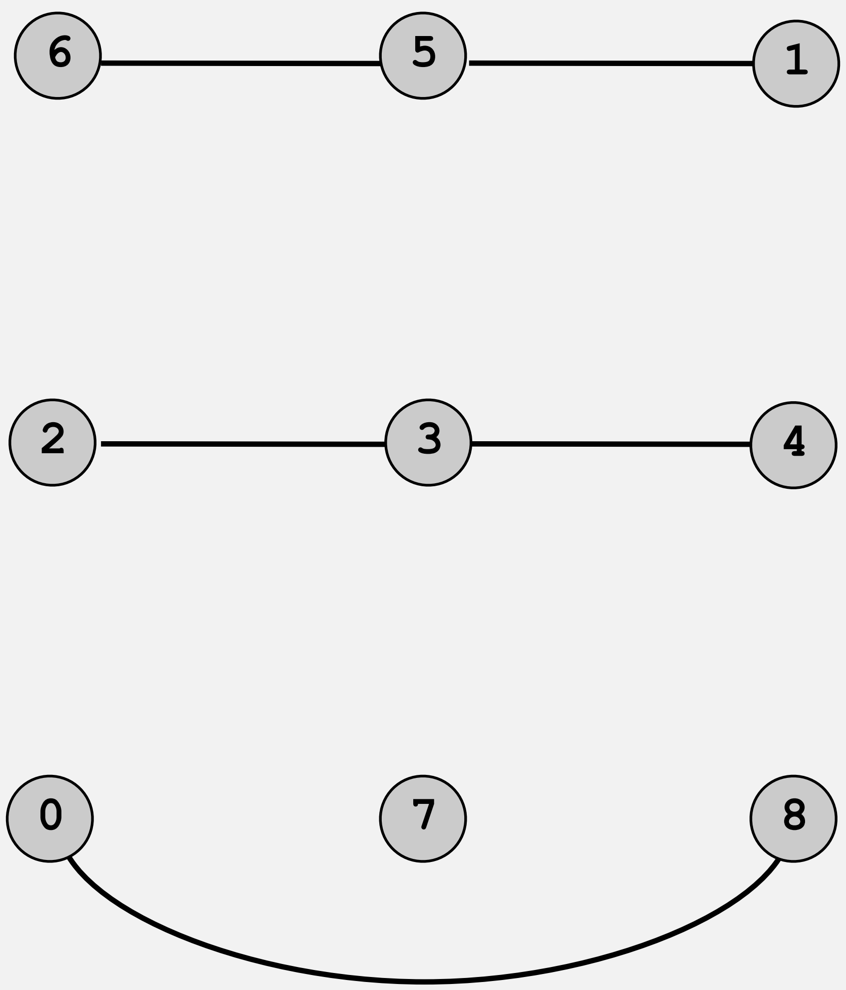
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)      no
 find(2, 4)      yes
union(5, 1)
```



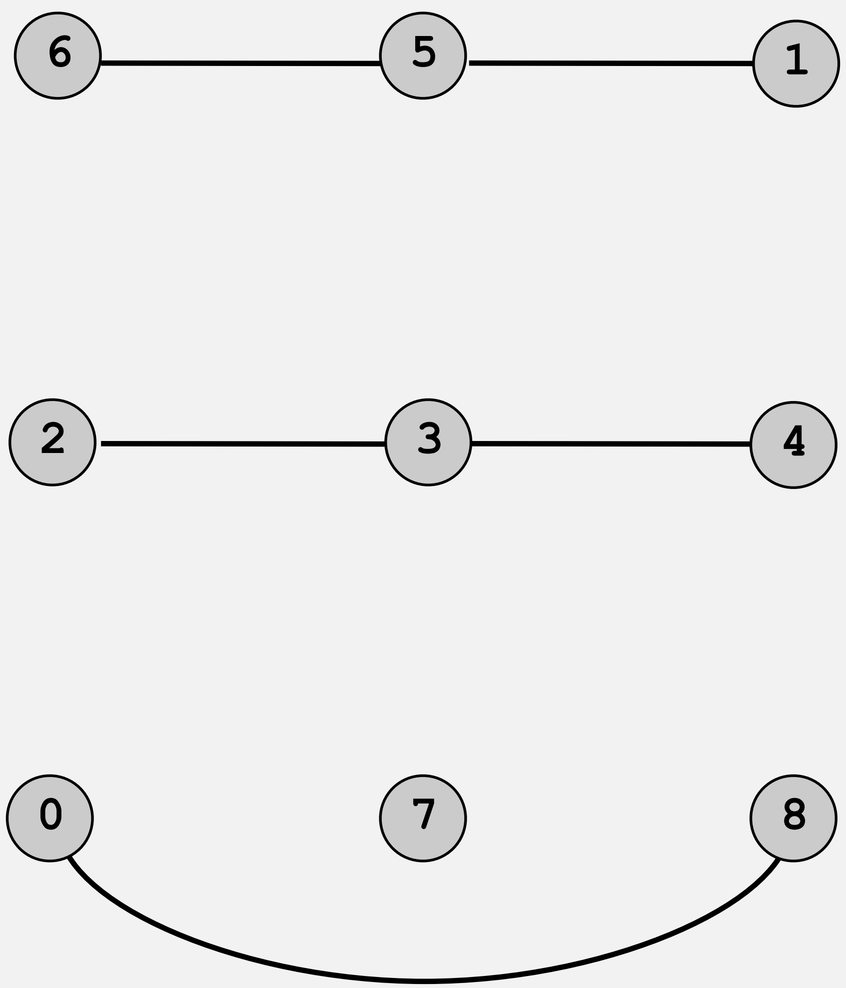
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)        no
 find(2, 4)        yes
union(5, 1)
union(7, 3)
```



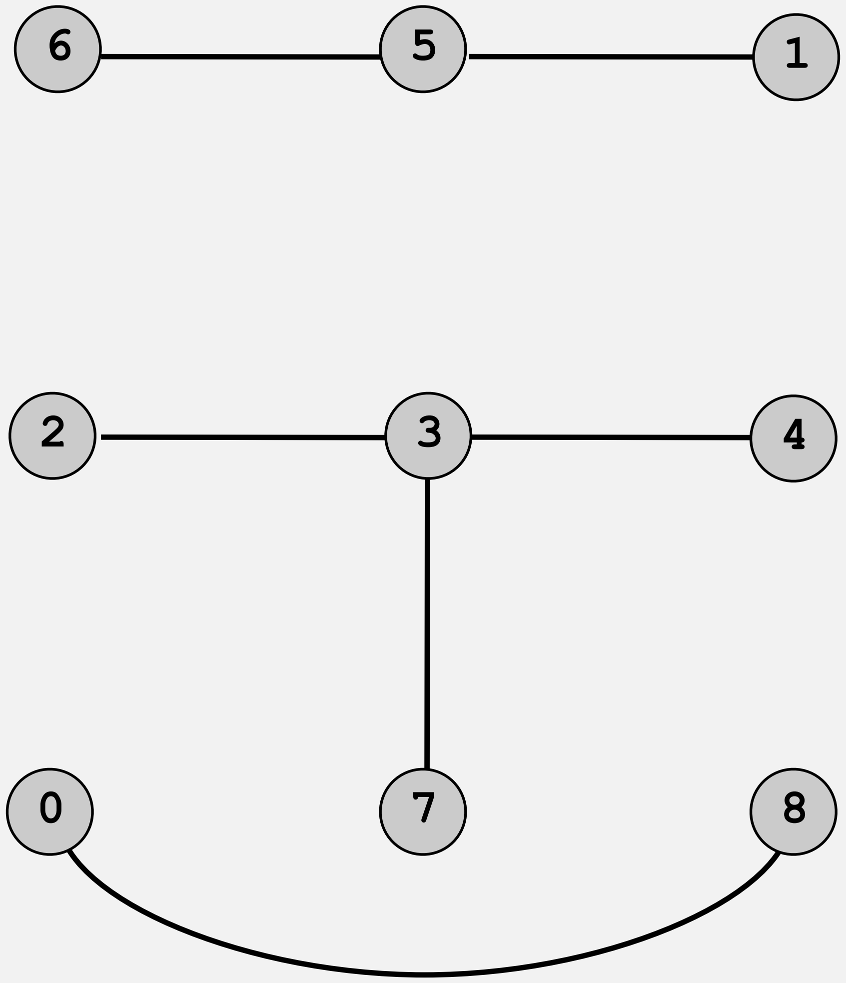
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)      no
 find(2, 4)      yes
union(5, 1)
union(7, 3)
```



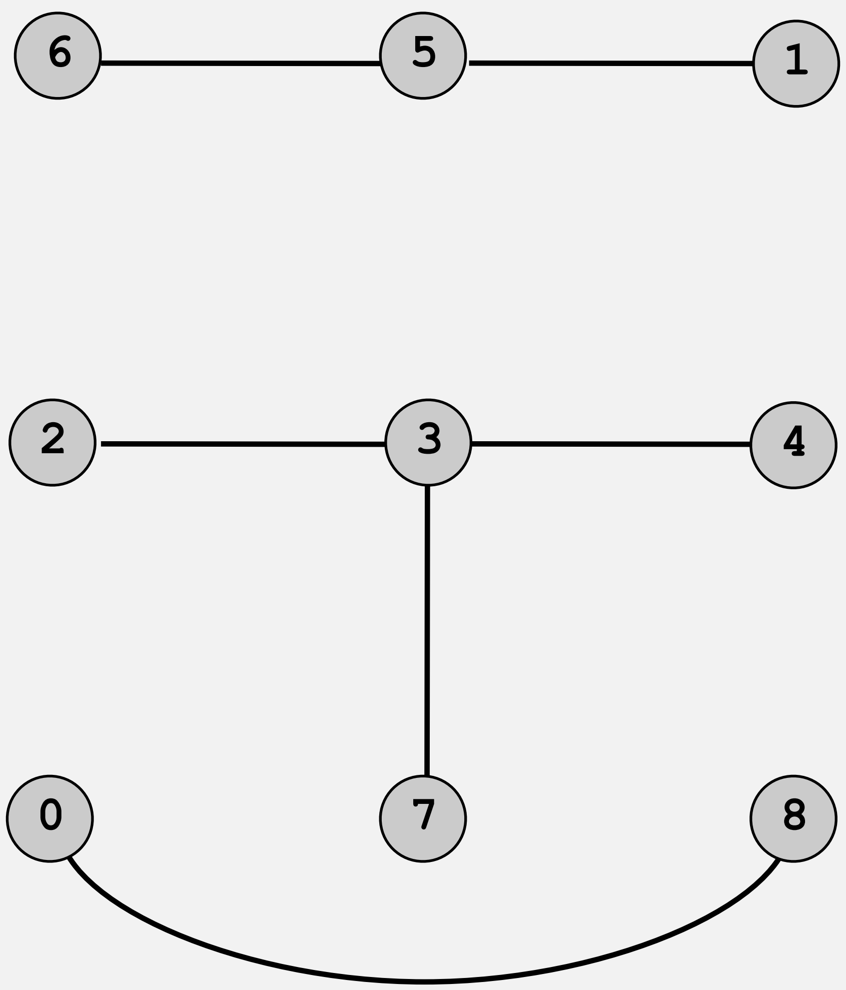
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)      no
 find(2, 4)      yes
union(5, 1)
union(7, 3)
union(1, 6)
```



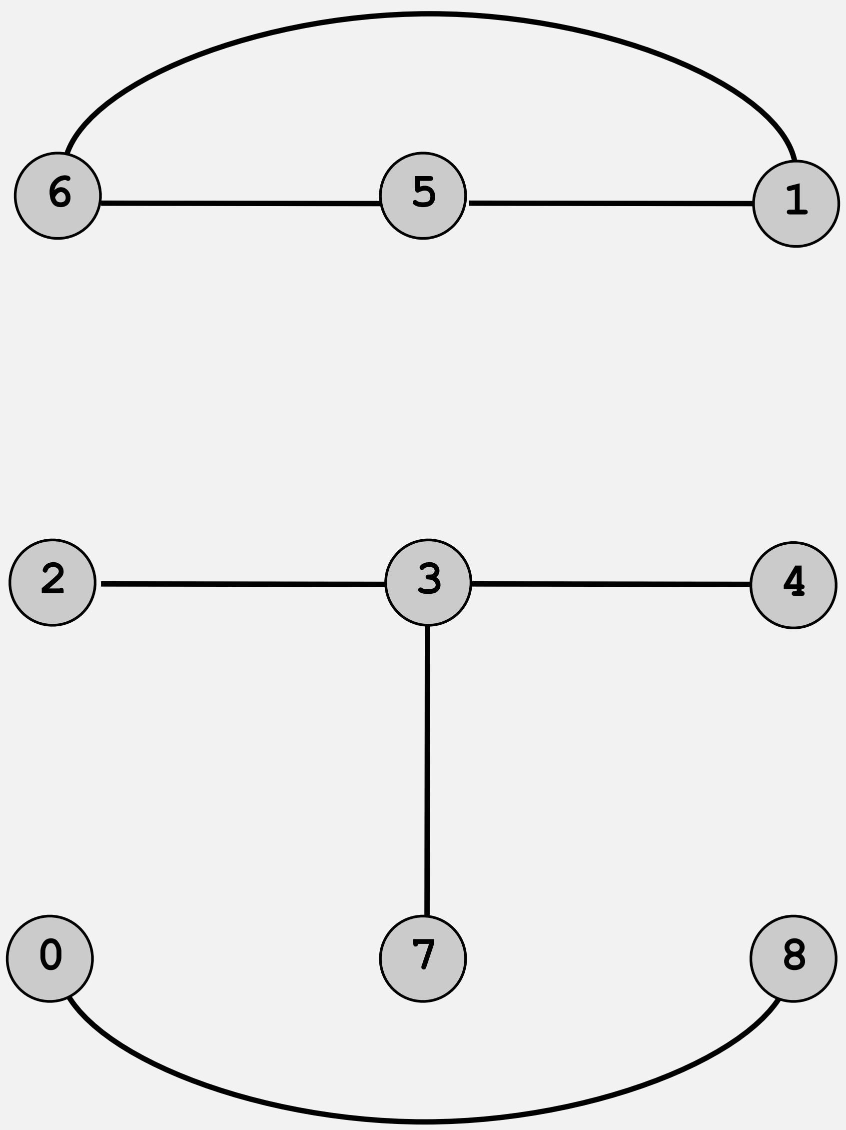
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)        no
 find(2, 4)        yes
union(5, 1)
union(7, 3)
union(1, 6)
```



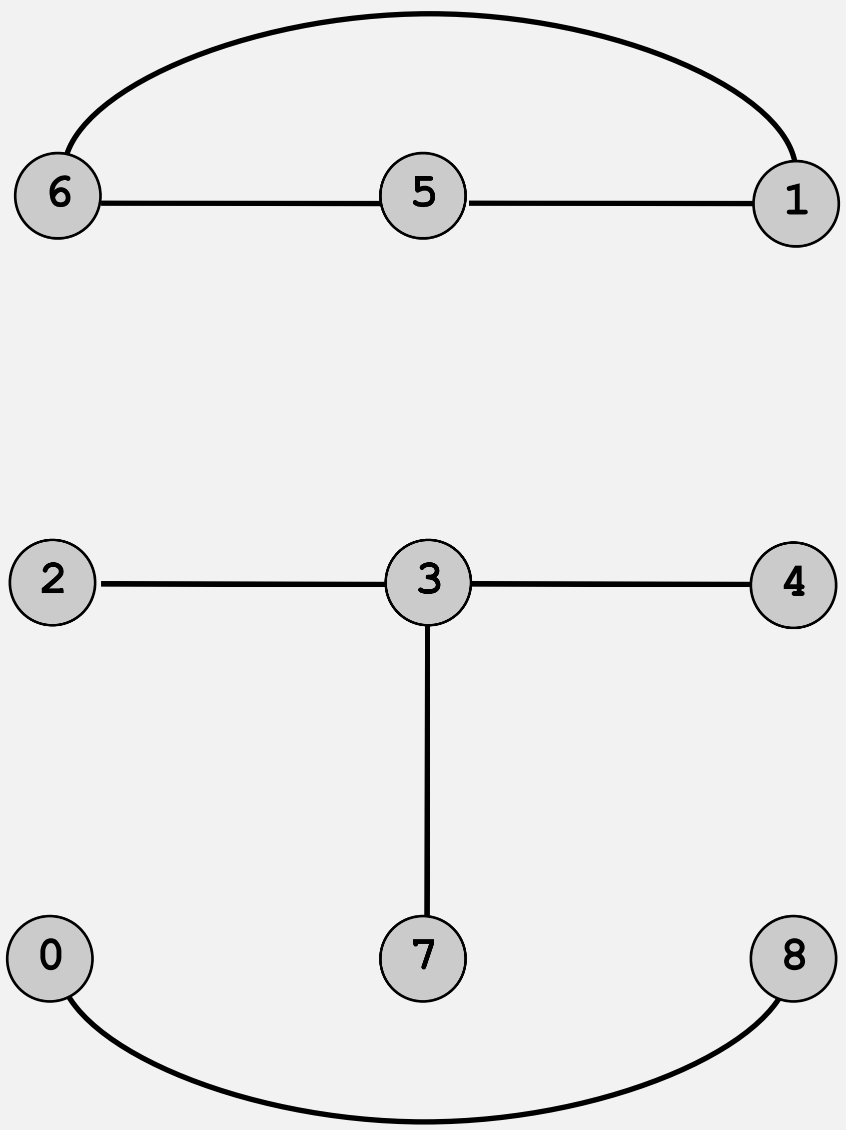
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)        no
 find(2, 4)        yes
union(5, 1)
union(7, 3)
union(1, 6)
union(4, 8)
```



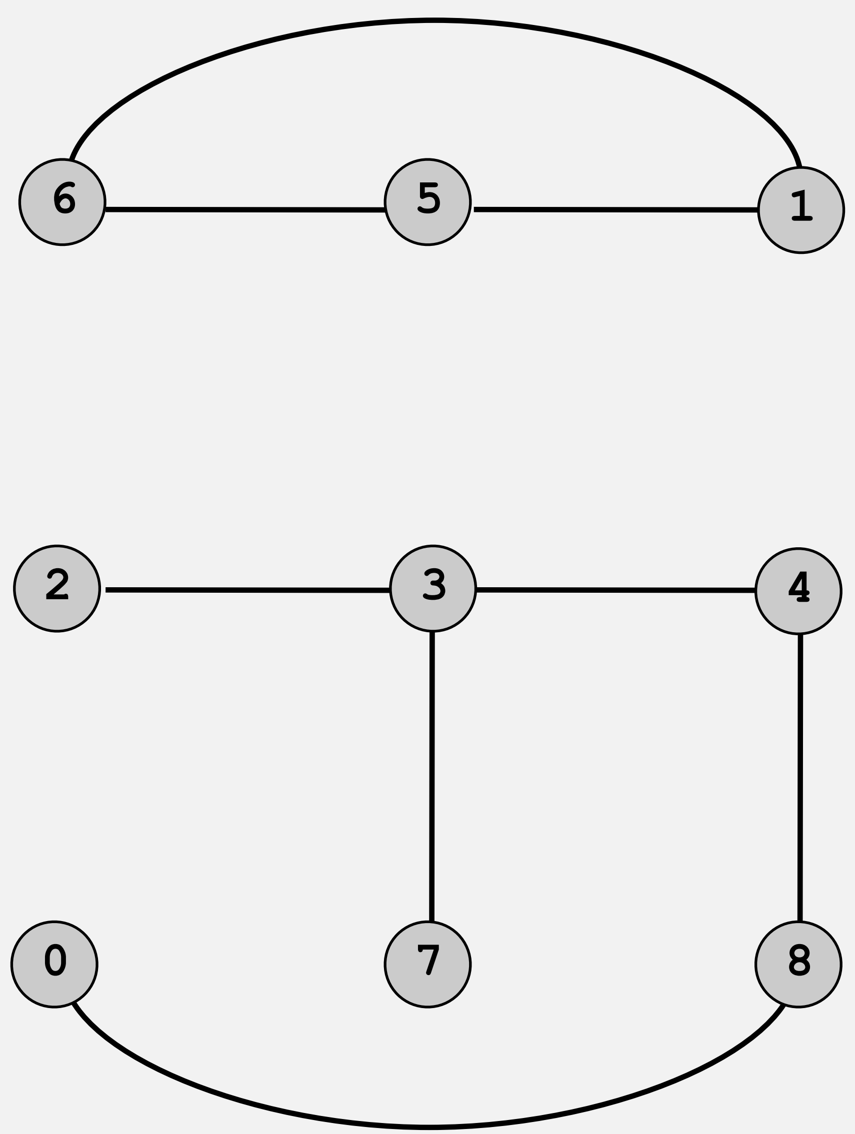
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)        no
 find(2, 4)        yes
union(5, 1)
union(7, 3)
union(1, 6)
union(4, 8)
```



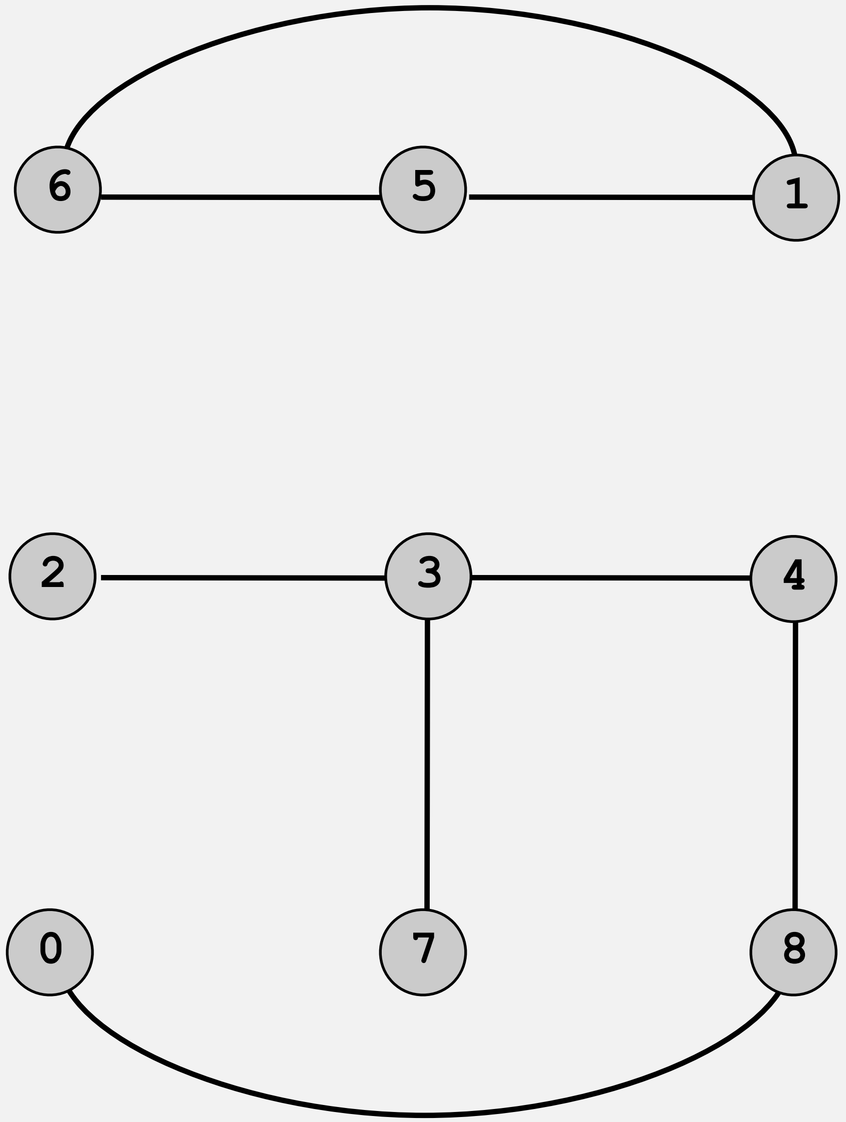
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

more difficult problem: find the path

`union(3, 4)`	
`union(8, 0)`	
`union(2, 3)`	
`union(5, 6)`	
`find(0, 2)`	no
`find(2, 4)`	yes
`union(5, 1)`	
`union(7, 3)`	
`union(1, 6)`	
`union(4, 8)`	
`find(0, 2)`	yes



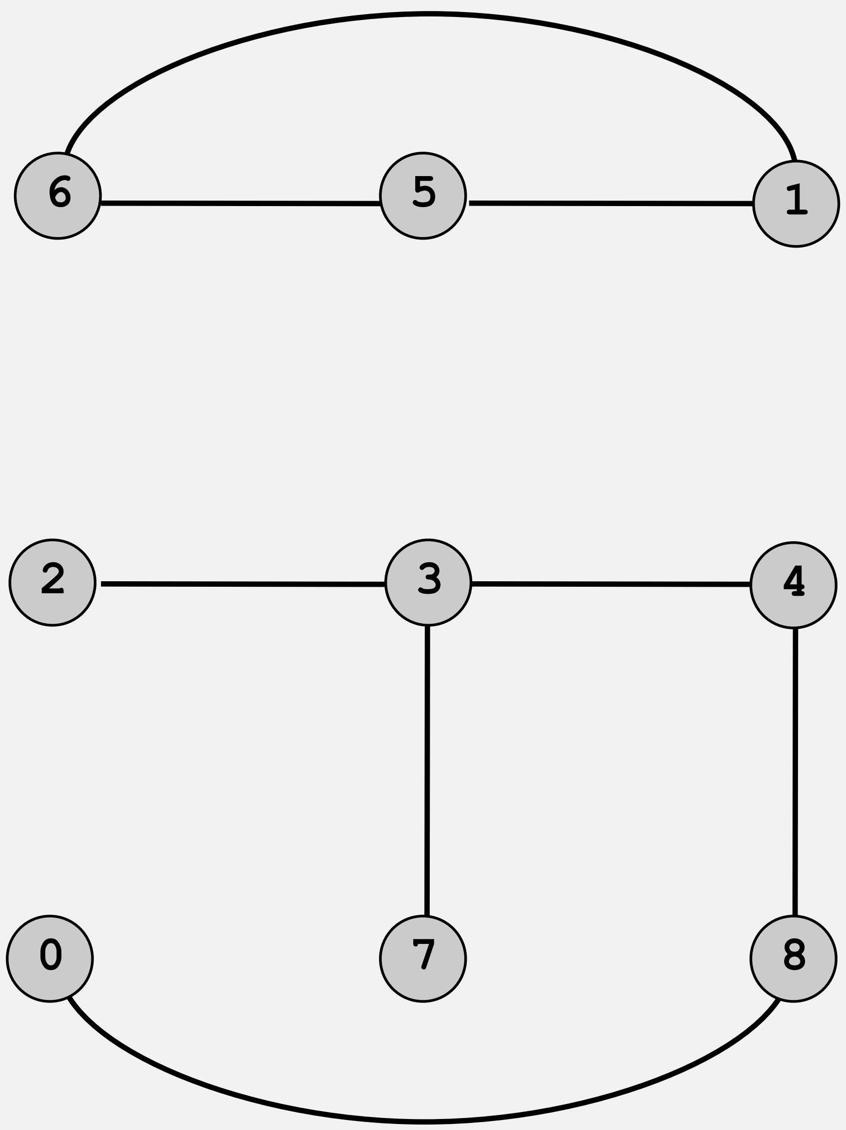
Dynamic connectivity

Given a set of objects

- Union: connect two objects.
- Find: is there a path connecting the two objects?

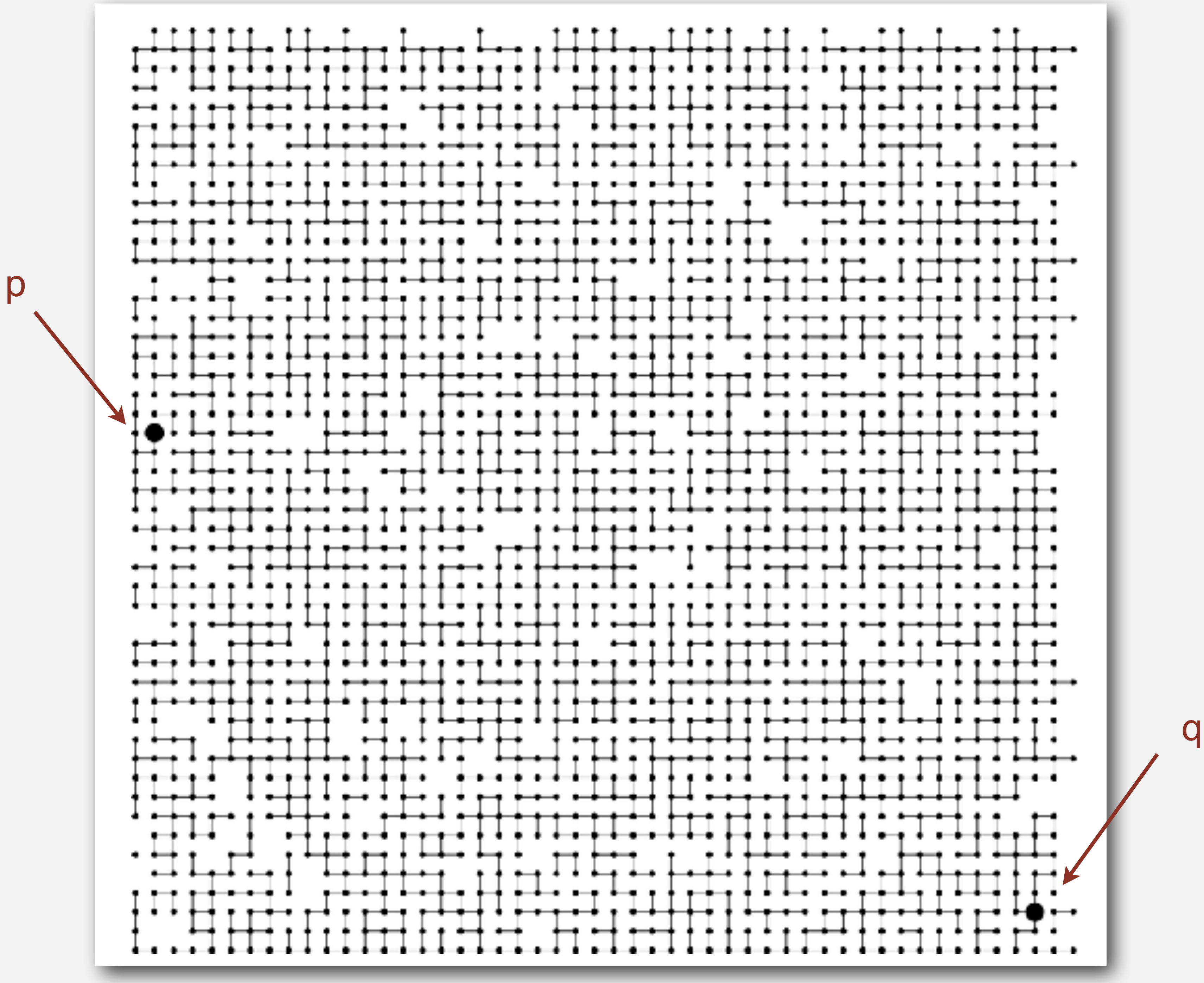
more difficult problem: find the path

```
union(3, 4)
union(8, 0)
union(2, 3)
union(5, 6)
 find(0, 2)       no
 find(2, 4)       yes
union(5, 1)
union(7, 3)
union(1, 6)
union(4, 8)
 find(0, 2)       yes
 find(2, 4)       yes
```



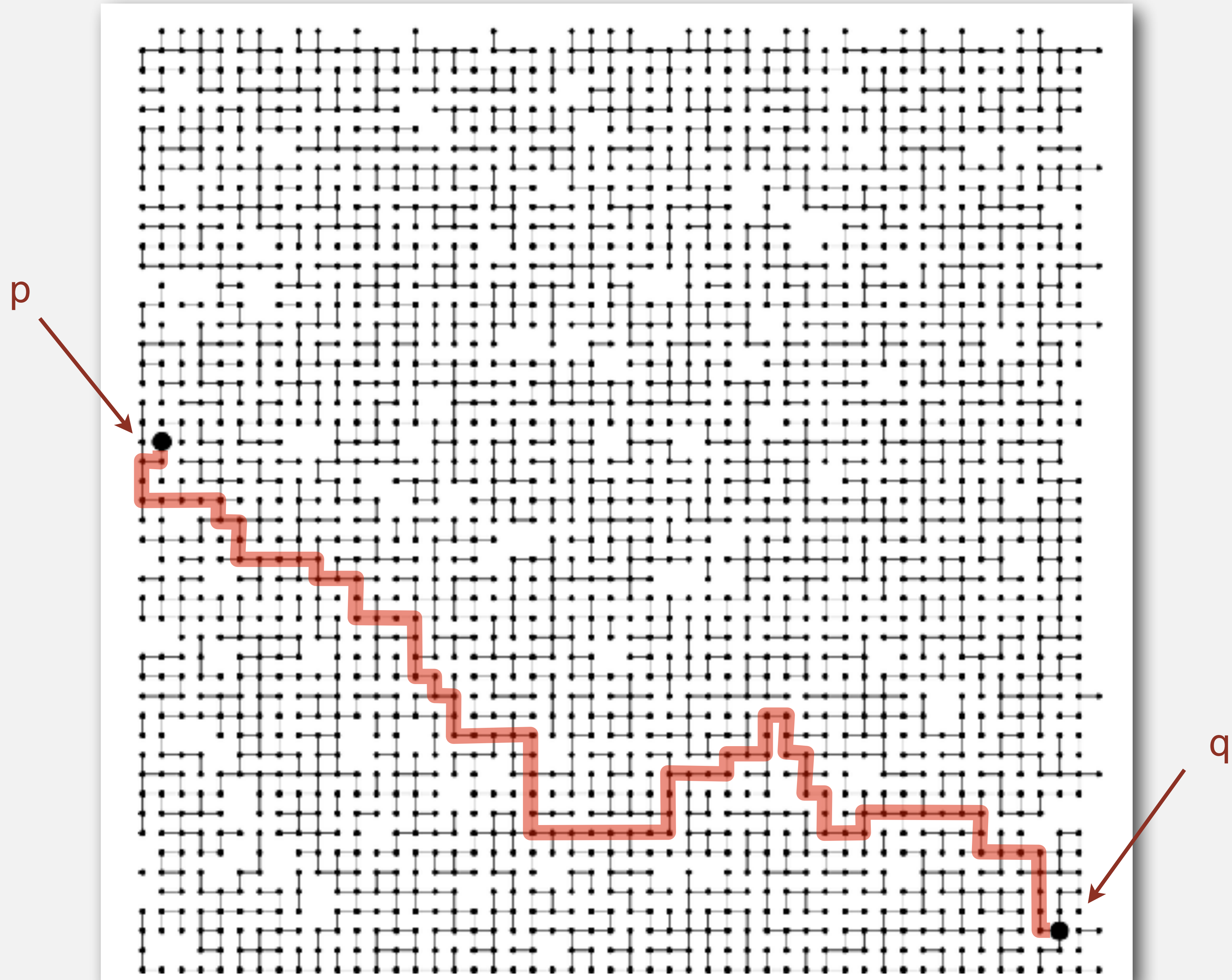
Connectivity example

Q. Is there a path from p to q?



Connectivity example

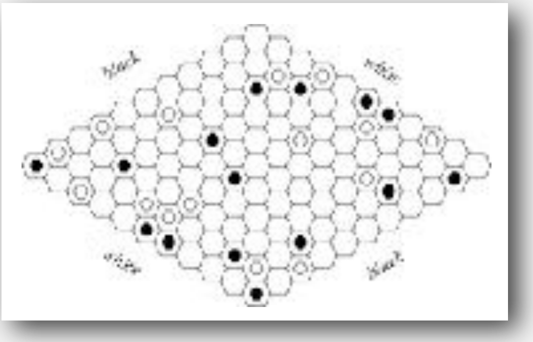
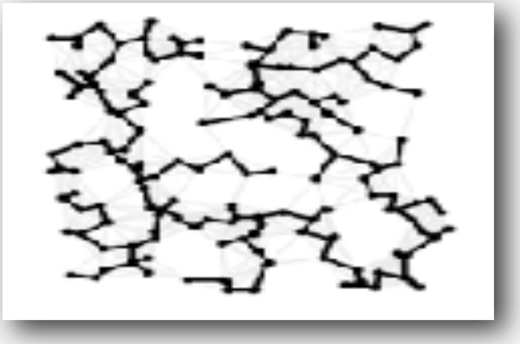
Q. Is there a path from p to q?



A. Yes.

Union-find applications

- Percolation (cf. 1st project)
- Games (Go, Hex).
- ✓ Network connectivity.
- Least common ancestor.
- Equivalence of finite state automata.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Morphological attribute openings and closings.
- Matlab's `bwlabel()` function in image processing.


Modeling the objects

Dynamic connectivity applications involve manipulating objects of all types.

- Pixels in a digital photo.
- Computers in a network.
- Variable names in Fortran.
- Friends in a social network.
- Transistors in a computer chip.
- Elements in a mathematical set.
- Metallic sites in a composite system.

When programming, convenient to name objects 0 to N-1.

- Use integers as array index.
- Suppress details not relevant to union-find.

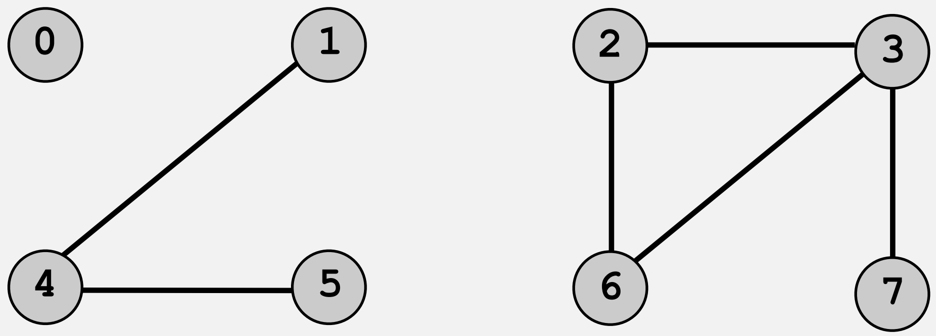
can use symbol table to translate from object names to integers: stay tuned (Chapter 3)

Modeling the connections

We assume "is connected to" is an equivalence relation:

- Reflexive: p is connected to p.
- Symmetric: if p is connected to q, then q is connected to p.
- Transitive: if p is connected to q and q is connected to r, then p is connected to r.

Connected components. Maximal set of objects that are mutually connected.



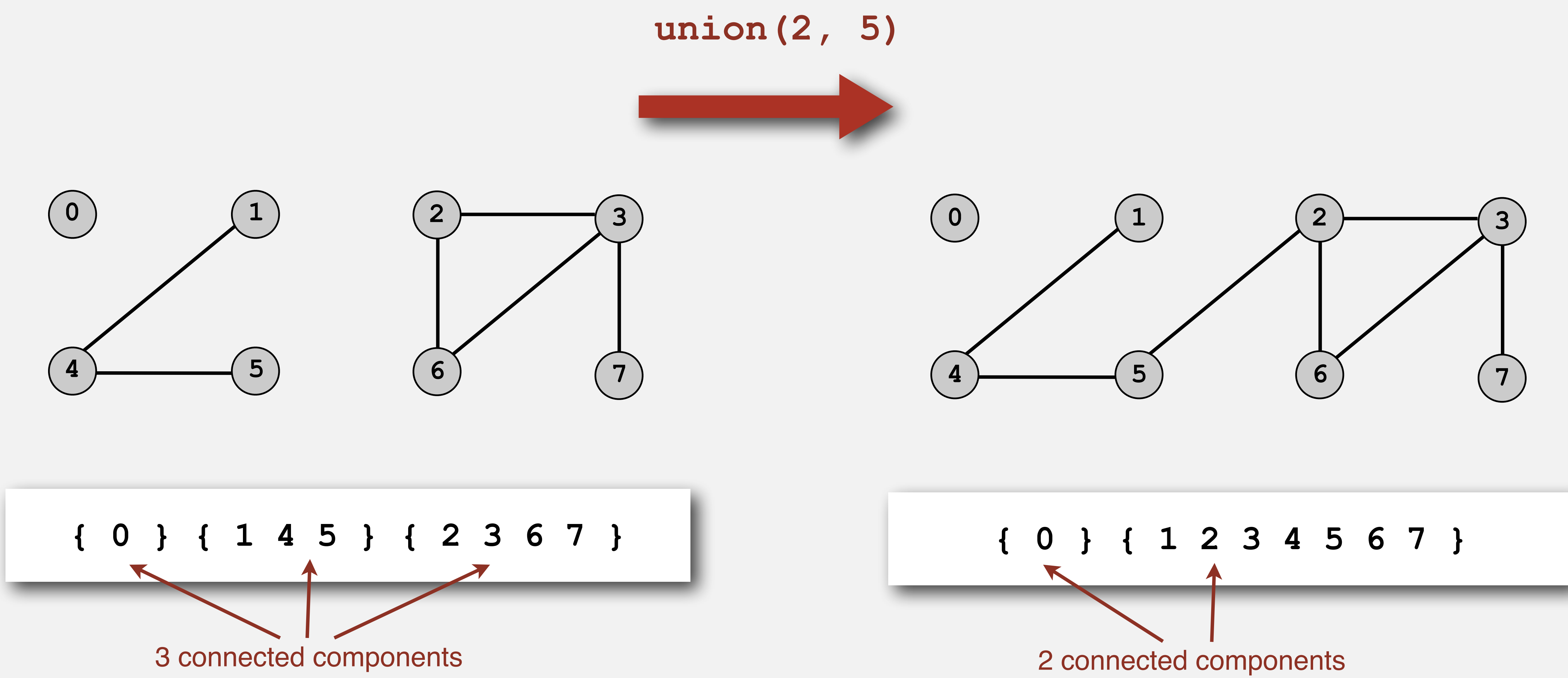
`{ 0 } { 1 4 5 } { 2 3 6 7 }`

3 connected components

Implementing the operations

Find query. Check if two objects are in the same component.

Union command. Replace components containing two objects with their union.



Union-find data type (API)

Goal. Design efficient data structure for union-find.

- Number of objects N can be huge.
- Number of operations M can be huge.
- Find queries and union commands may be intermixed.

```
public class UF
```

	`UF(int N)`	*create union-find data structure with N objects and no connections*
`boolean`	`find(int p, int q)`	*are p and q in the same component?*
`void`	`union(int p, int q)`	*add connection between p and q*
`int`	`count()`	*number of components*

Dynamic-connectivity client

- Read in number of objects N from standard input.
- Repeat:
  - read in pair of integers from standard input
  - write out pair if they are not already connected

```
public static void main(String[] args)
{
   int N = StdIn.readInt();
   UF uf = new UF(N);
   while (!StdIn.isEmpty())
   {
      int p = StdIn.readInt();
      int q = StdIn.readInt();
      if (uf.find(p, q)) continue;
      uf.union(p, q);
      StdOut.println(p + " " + q);
   }
}
```

```
% more tiny.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
```

- dynamic connectivity
- applications
- **quick find**

Quick-find [eager approach]

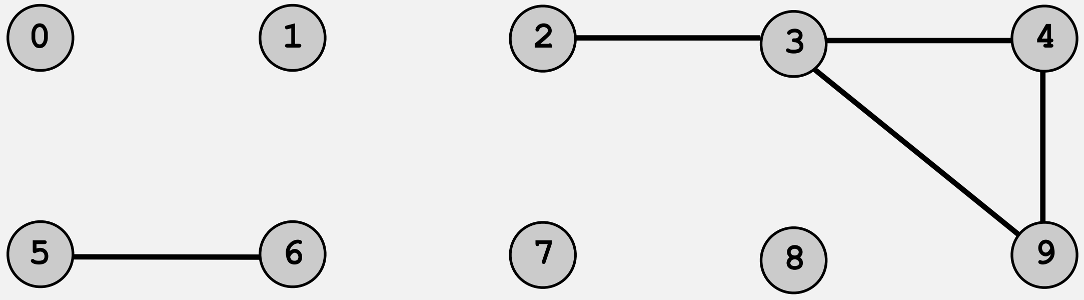
Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` in same component iff they have the same id.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	6	6	7	8	9

5 and 6 are connected

2, 3, 4, and 9 are connected



Quick-find [eager approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` in same component iff they have the same id.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	6	6	7	8	9

5 and 6 are connected

2, 3, 4, and 9 are connected

Find. Check if `p` and `q` have the same id.

`id[3] = 9`; `id[6] = 6`

3 and 6 in different components

Quick-find [eager approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` in same component iff they have the same id.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	9	9	6	6	7	8	9

5 and 6 are connected

2, 3, 4, and 9 are connected

Find. Check if `p` and `q` have the same id.

`id[3] = 9`; `id[6] = 6`

3 and 6 in different components

Union. To merge sets containing `p` and `q`, change all entries with `id[p]` to `id[q]`.

i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	6	6	6	6	6	7	8	6

union of 3 and 6

2, 3, 4, 5, 6, and 9 are connected

problem: many values can change

Quick-find example

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

id[p] *and* id[q] *differ, so* union() *changes entries equal to* id[p] *to* id[q] *(in red)*

id[p] *and* id[q] *match, so no change*

Quick-find: Java implementation

```
public class QuickFindUF
{
   private int[] id;

   public QuickFindUF(int N)
   {
      id = new int[N];
      for (int i = 0; i < N; i++)
         id[i] = i;
   }

   public boolean find(int p, int q)
   {  return id[p] == id[q];  }

   public void union(int p, int q)
   {
      int pid = id[p];
      int qid = id[q];
      for (int i = 0; i < id.length; i++)
         if (id[i] == pid) id[i] = qid;
   }
}
```

set id of each object to itself

(N array accesses)

check whether **p** and **q**

are in the same component

(2 array accesses)

change all entries with **id[p]** to **id[q]**

(linear number of array accesses)

Quick-find is too slow

Cost model. Number of array accesses (for read or write).

algorithm	init	union	find
quick-find	N	N	1

Quick-find defect.

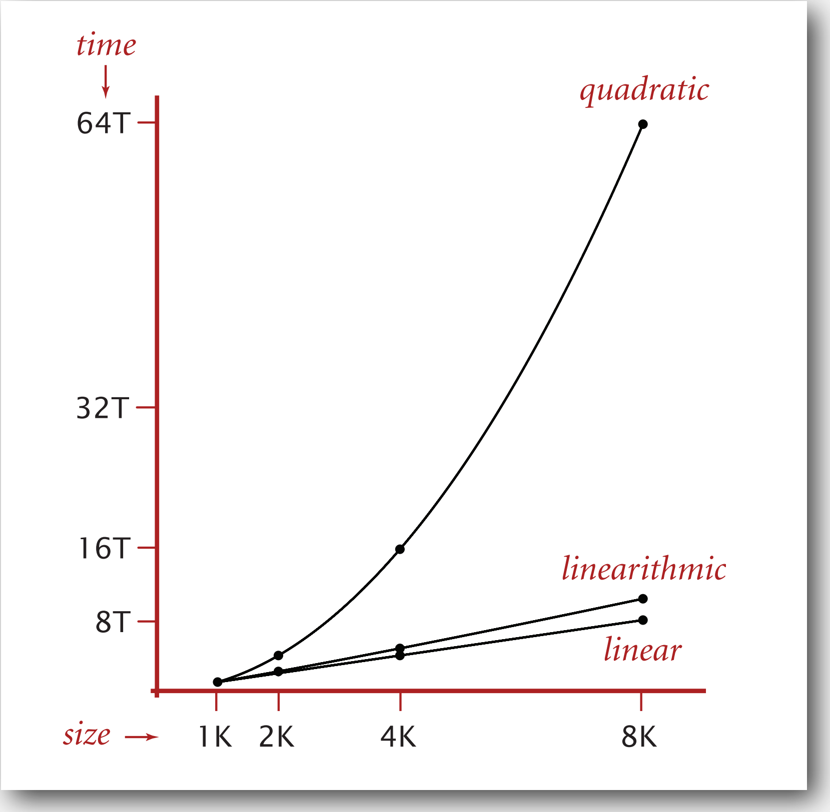
- Union too expensive.
- Ex. Takes N^2 array accesses to process sequence of N union commands on N objects. This is a quadratic algorithm.

Quadratic algorithms do not scale

Rough standard (for now).

- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second.

a truism (roughly) since 1950!



Ex. Huge problem for quick-find.

- 10^9 union commands on 10^9 objects.
- Quick-find takes more than 10^{18} operations.
- 30+ years of computer time!

Paradoxically, quadratic algorithms get worse with newer equipment.

- New computer may be 10x as fast.
- But, has 10x as much memory so problem may be 10x bigger.
- With quadratic algorithm, takes 10x as long!

Subtext of today's example (and this course)

Steps to developing a usable algorithm.

- Model the problem.
- Find an algorithm (and data structure) to solve it.
- Fast enough? Fits in memory?
- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

Subtext of today's example (and this course)

Steps to developing a usable algorithm.

✓ Model the problem.

✓ Find an algorithm (and data structure) to solve it.

✓ Fast enough? NO.

- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

Subtext of today's example (and this course)

Steps to developing a usable algorithm.

✓ Model the problem.

✓ Find an algorithm (and data structure) to solve it.

✓ Fast enough? NO.

- If not, figure out why.
- Find a way to address the problem.
- Iterate until satisfied.

Will continue this example after we cover analysis methods and additional data structures