ALGORITHMS: THEORY, DESIGN AND IMPLEMENTATION

- ‣ *overview*
- ‣ *why study algorithms?*
- ‣ *resources*

Why study algorithms?

Their impact is broad and far-reaching.

Internet. Web search, packet routing, distributed file sharing, ...

Biology. Human genome project, protein folding, ...

Computers. Circuit layout, file system, compilers, ...

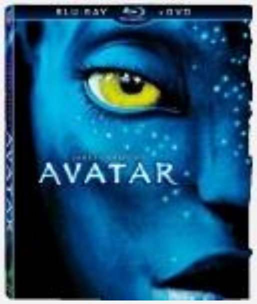
Computer graphics. Movies, video games, virtual reality, ...

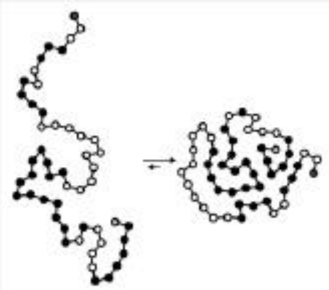
Security. Cell phones, e-commerce, voting machines, ...

Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

Social networks. Recommendations, news feeds, advertisements, ...

Physics. N-body simulation, particle collision simulation, ...

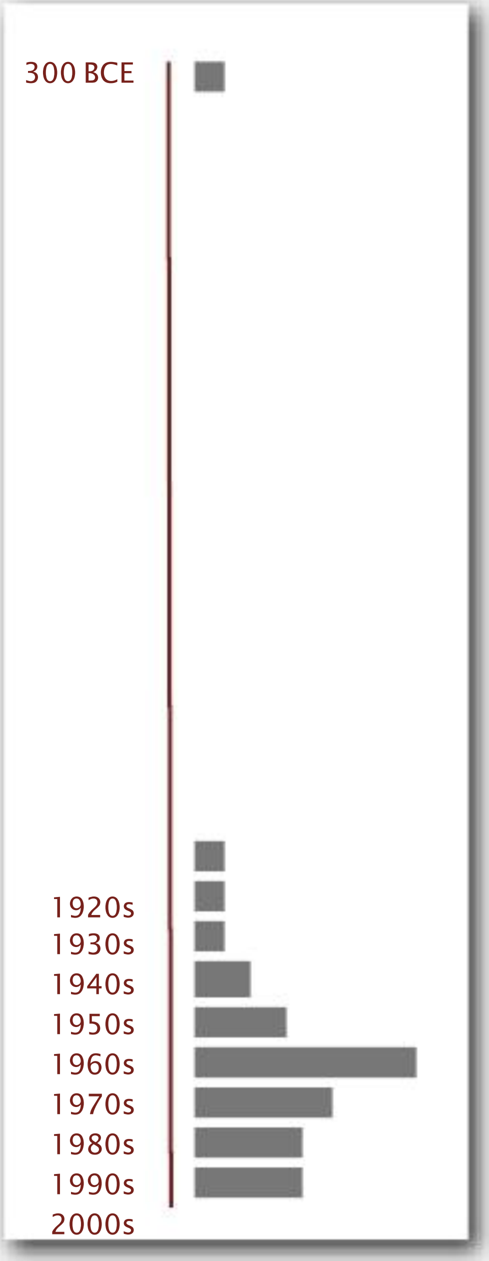


Why study algorithms?

Old roots, new opportunities.

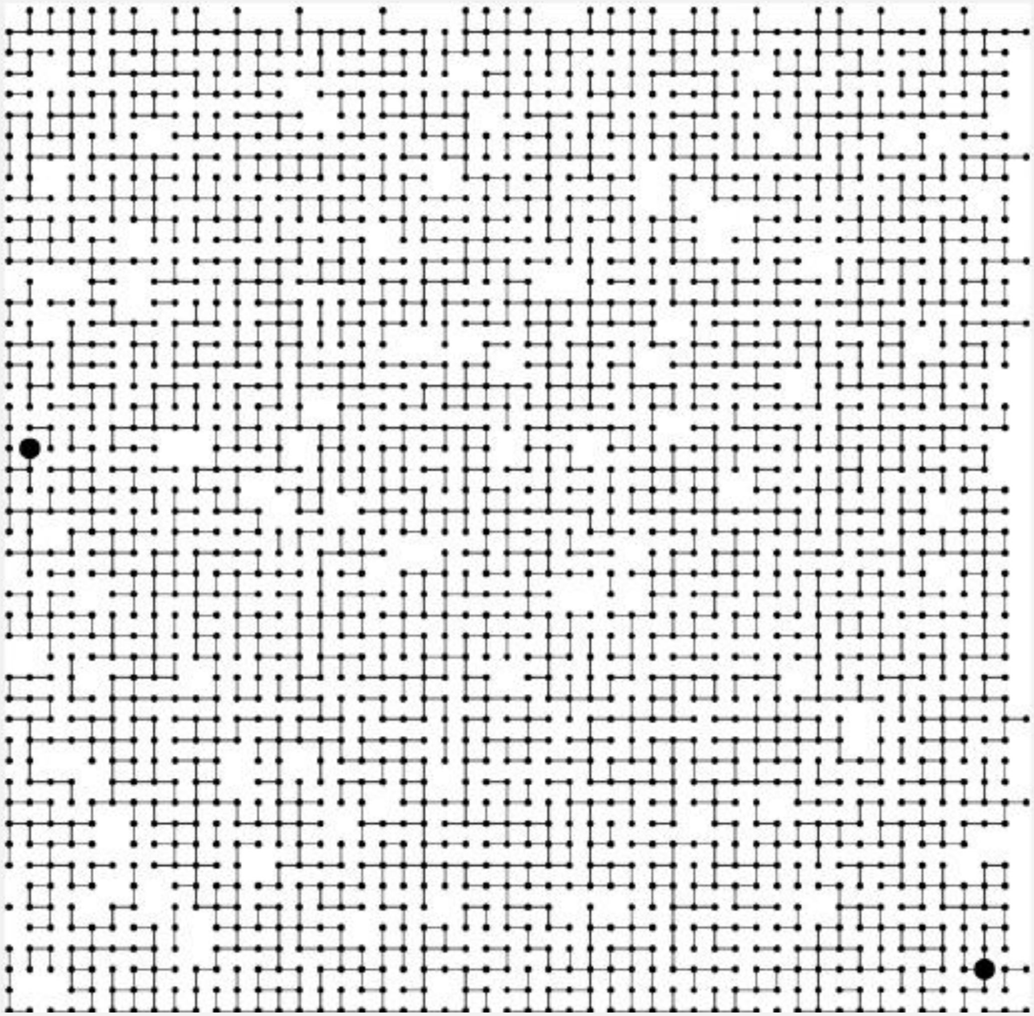
- Study of algorithms dates at least to Euclid.
- Formalized by Church and Turing in 1930s.
- Some important algorithms may be discovered by yourself!



Why study algorithms?

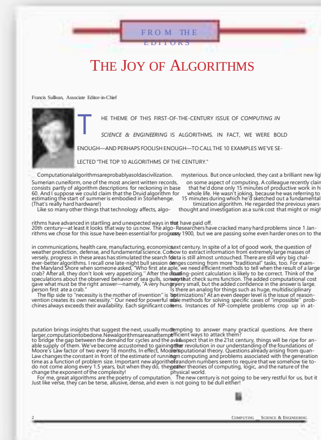
To solve problems that could not otherwise be addressed.

Ex. Network connectivity. [stay tuned]



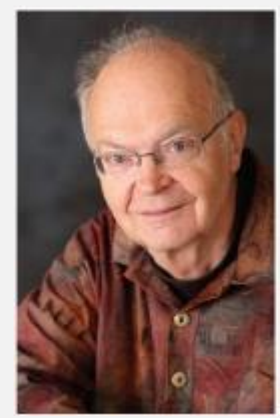
Why study algorithms?

For intellectual stimulation.


FROM THE EDITORS

THE JOY OF ALGORITHMS

> " *For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing.* " — *Francis Sullivan*



> " *An algorithm must be seen to be believed.* " — *Donald Knuth*

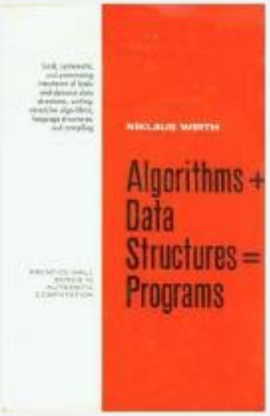
Why study algorithms?

To become a proficient programmer.

" I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. "

— Linus Torvalds (creator of Linux)

" Algorithms + Data Structures = Programs. " *— Niklaus Wirth*



Why study algorithms?

They may unlock the secrets of life and of the universe.

Computational models are replacing math models in scientific inquiry.

$$E = mc^2$$

$$F = ma$$

$$F = \frac{Gm_1m_2}{r^2}$$

```
for (double t = 0.0; true; t = t + dt)
   for (int i = 0; i < N; i++)
   {
      bodies[i].resetForce();
      for (int j = 0; j < N; j++)
         if (i != j)
            bodies[i].addForce(bodies[j]);
   }
```

21st century science
(algorithm based)

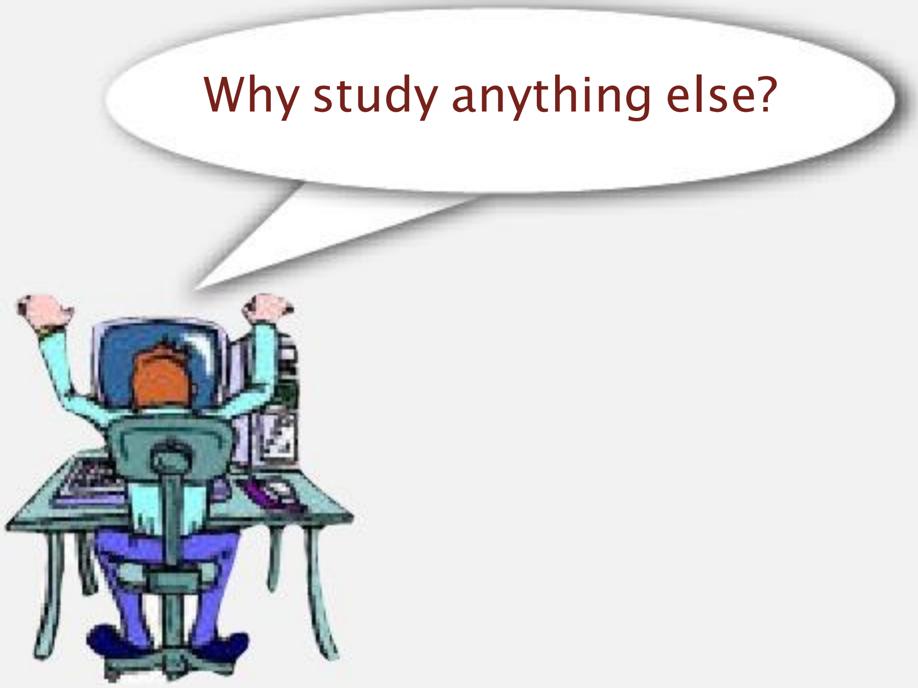
" Algorithms: a common language for nature, human, and computer. " — *Avi Wigderson*

Why study algorithms?

For fun and profit.

Why study algorithms?

- Their impact is broad and far-reaching.
- Old roots, new opportunities.
- To solve problems that could not otherwise be addressed.
- For intellectual stimulation.
- To become a proficient programmer.
- They may unlock the secrets of life and of the universe.
- For fun and profit.



INDICATIVE ROAD MAP FOR THE STRUCTURE OF CONTENTS

	Brute-force	Divide-and-conquer	Decrease-and-conquer	Transform-and-conquer	Greedy	Dynamic programming
Sorting algorithms	√ (LW3)	√ (LW4)	√ (LW5)	√ (LW6)		
Search algorithms	√	√ (LW8)		√ (LW8)		
Graph algorithms	√ (LW9)	√ (LW9)	√	√	√ (LW9)	
Path finding algorithms	√				√ (LW10)	√
Network flow algorithms	√				√ (LW11)	√
Computational geometry algorithms	√				√ (LW12)	√

SOME LOGISTICS ABOUT THE MODULE

- Case studies, i.e., problems and algorithmic solutions, will be discussed in groups during Q&A sessions, as part of the Problem Based Learning approach to our teaching. **Therefore, please come prepared to these sessions**.
- Surveys may be contacted in order to reflect on your perception of difficulty to algorithmically resolve the problem.

SOME LOGISTICS ABOUT THE MODULE

- Programming exercises for the tutorials will address implementation of algorithms in Java and within a framework, which is already prepared for you.
- Two assessment components, a **course work** and an **exam**. Please check details on Blackboard's module site.

KARATSUBA MULTIPLICATION OF INTEGER NUMBERS

- Having studied the plain text notes or having watched the recommended videos for LW1, please apply the Karatsuba Algorithm for the multiplication of the numbers:
  - 2019
  - 1963

CASE STUDY FOR TUTORIALS

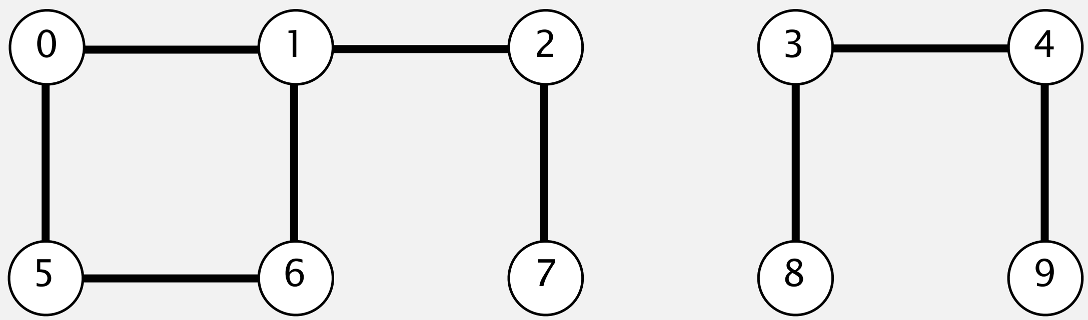
UNION-FIND ALGORITHMS AND THE CONNECTIVITY PROBLEM

Dynamic connectivity

Given a set of N objects.

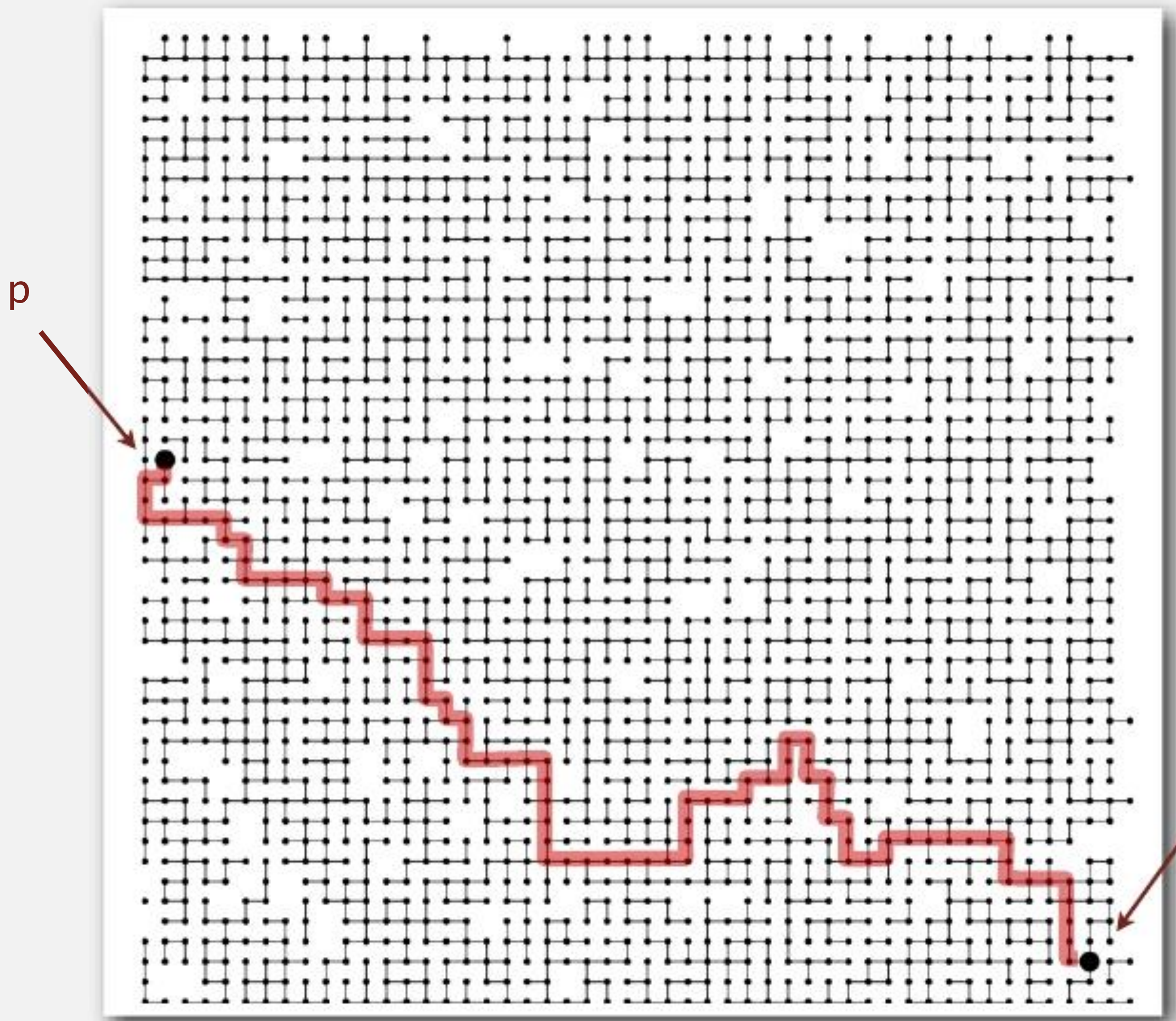
- Union command: connect two objects.
- Find/connected query: is there a path connecting the two objects?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
connected(0, 7)   ✘
connected(8, 9)   ✓
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
connected(0, 7)   ✓
```



Connectivity example

Q. Is there a path connecting p and q ?



A. Yes.

Modeling the objects

Applications involve manipulating objects of all types.

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

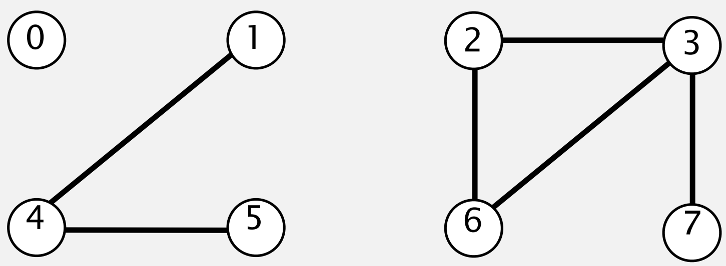
can use symbol table to translate from site names to integers: stay tuned (Chapter 3)

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.



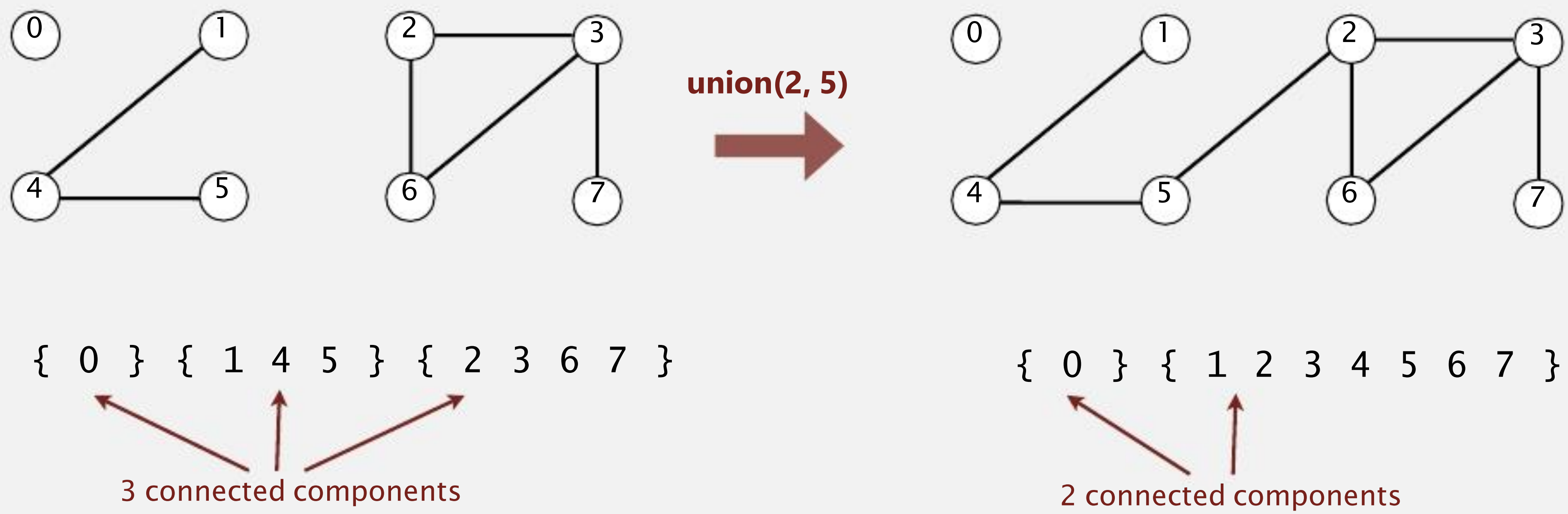
{ 0 } { 1 4 5 } { 2 3 6 7 }

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)`	*initialize union-find data structure with N objects (0 to $N-1$)*
`void`	`union(int p, int q)`	*add connection between p and q*
`boolean`	`connected(int p, int q)`	*are p and q in the same component?*
`int`	`find(int p)`	*component identifier for p (0 to $N-1$)*
`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
  - if they are not yet connected, connect them and print out pair

```
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.connected(p, q))
      {
         uf.union(p, q);
         StdOut.println(p + " " + q);
      }
   }
}
```

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

Quick-find [eager approach]

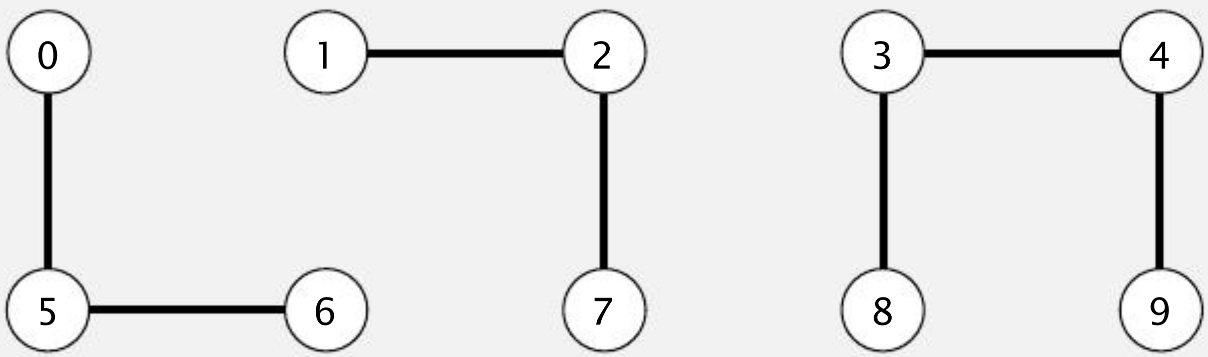
Data structure.

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

if and only if

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

0, 5 and 6 are connected
1, 2, and 7 are connected
3, 4, 8, and 9 are connected



Quick-find [eager approach]

Data structure.

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

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

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

`id[6] = 0; id[1] = 1`
6 and 1 are not connected

Union. To merge components containing `p` and `q`, change all entries whose id equals `id[p]` to `id[q]`.

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

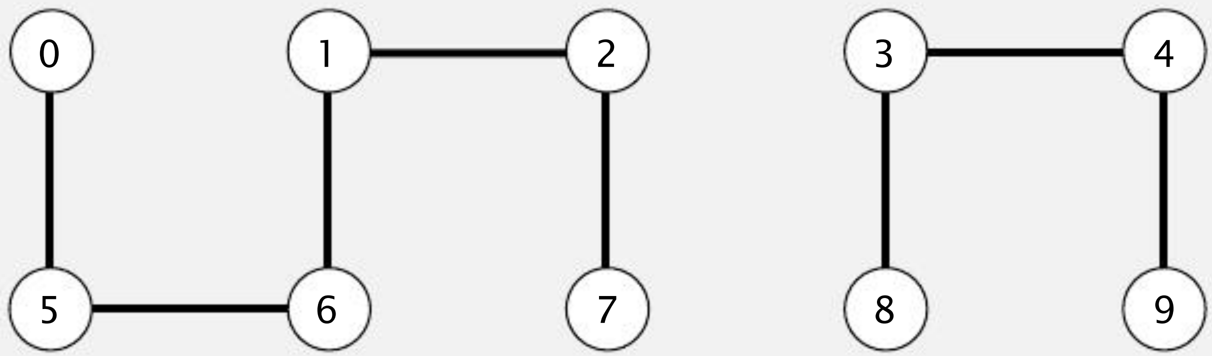
after union of 6 and 1

problem: many values can change

Quick-find demo

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

Quick-find demo



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

Quick-find: Java implementation – Your first exercise ☺

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

   public QuickFindUF(int N)
   {


         }


   public boolean connected(int p, int q)
   {                                  }

   public void union(int p, int q)
   {





   }
}
```

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]
(at most 2N + 2 array accesses)

Quick-find is too slow

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

algorithm	initialize	union	find
quick-find	N	N	1

order of growth of number of array accesses

Quick-find defect. Union too expensive? More on these aspects of performance next week…..

quadratic

Ex. Takes N^2 array accesses to process sequence of N union commands on N objects.

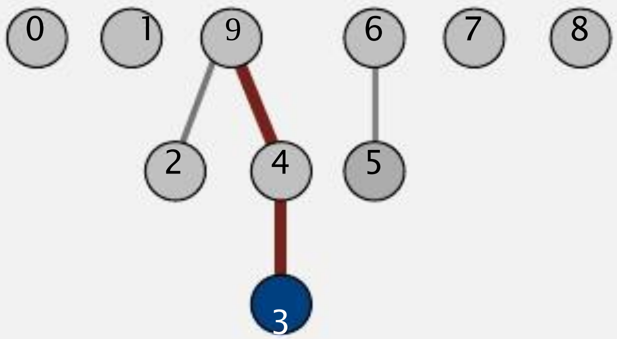
Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

keep going until it doesn't change
(algorithm ensures no cycles)

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



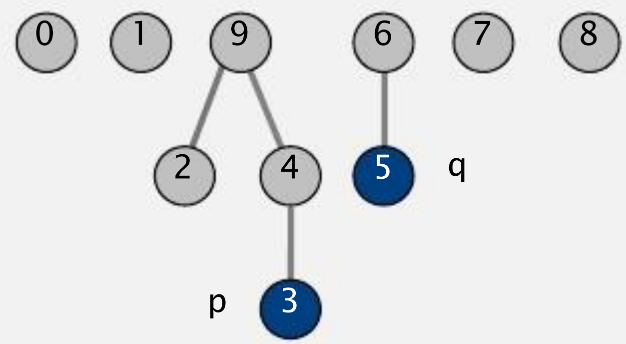
root of 3 is 9

Quick-union [lazy approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `id[i]` is parent of `i`.
- Root of `i` is `id[id[id[...id[i]...]]]`.

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



root of 3 is 9
root of 5 is 6

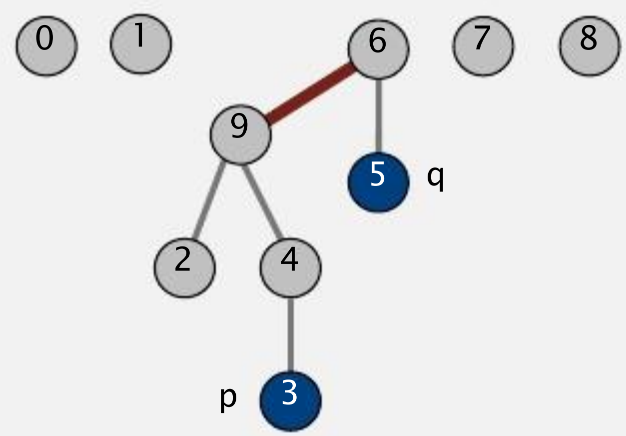
3 and 5 are not connected

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

Union. To merge components containing `p` and `q`, set the id of `p`'s root to the id of `q`'s root.

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

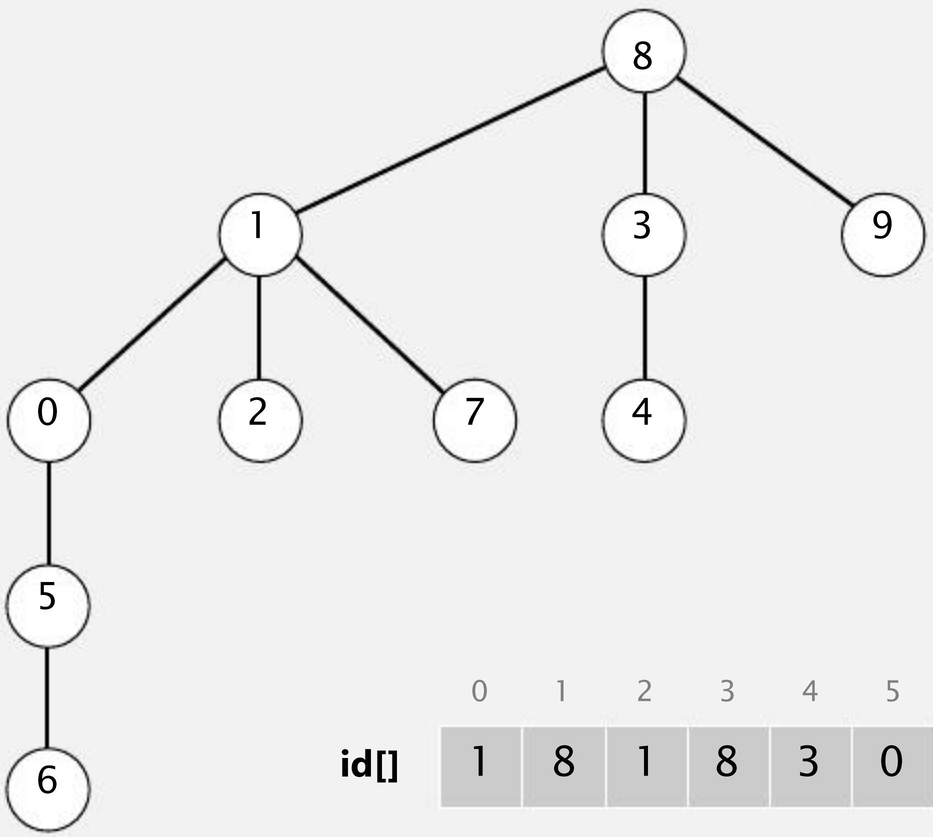
only one value changes



Quick-union demo

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

Quick-union demo



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

Quick-union: Java implementation – Your second exercise ☺

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

   public QuickUnionUF(int N)
   {

   }

   private int root(int i)
   {

   }

   public boolean connected(int p, int q)
   {

   }

   public void union(int p, int q)
   {

   }
}
```

set id of each object to itself
(N array accesses)

chase parent pointers until reach root
(depth of i array accesses)

check if p and q have same root
(depth of p and q array accesses)

change root of p to point to root of q
(depth of p and q array accesses)

Quick-union is also too slow

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

algorithm	initialize	union	find
quick-find	N	N	1
quick-union	N	N †	N

worst case (quick-union)

† includes cost of finding roots

Quick-find defect.

- Union too expensive (N array accesses).
- Trees are flat, but too expensive to keep them flat.

Quick-union defect.

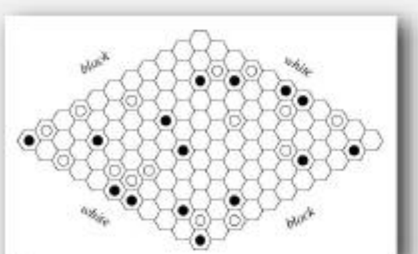
- Trees can get tall.
- Find too expensive (could be N array accesses).

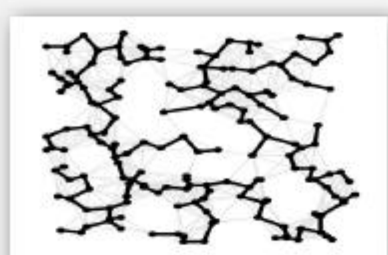
Union-find applications

- Percolation.
- Games (Go, Hex).

✓ Dynamic connectivity.

- Least common ancestor.
- Equivalence of finite state automata.
- Hoshen-Kopelman algorithm in physics.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Morphological attribute openings and closings.
- Matlab's `bwlabel()` function in image processing.





In a nutshell

Steps to developing a usable algorithm.

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