Algorithms and Data Structures
Princeton University
Spring 2008

Kevin Wayne

COS 226 course overview

What is COS 226?

- Intermediate-level survey course.
- Programming and problem solving with applications.
- Algorithm: method for solving a problem.
- Data structure: method to store information.

topic	data structures and algorithms
data types	stack, queue, union-find, priority queue
sorting	quicksort, mergesort, heapsort, radix sorts
searching	hash table, BST, red-black tree
graphs	BFS, DFS, Prim, Kruskal, Dijkstra
strings	KMP, Regular expressions, TST, Huffman, LZW
geometry	Graham scan, k-d tree, Voronoi diagram

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. CD player, DVD, MP3, JPG, DivX, HDTV, ...

Transportation. Airline crew scheduling, map routing, ...

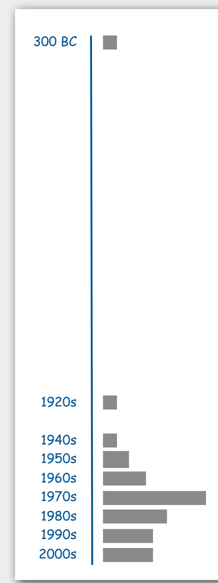
Physics. N-body simulation, particle collision simulation, ...

...

Why study algorithms?

Old roots, new opportunities.

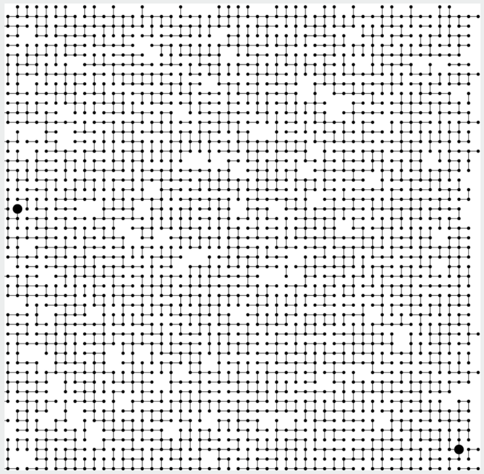
- Study of algorithms dates at least to Euclid
- Some important algorithms were discovered by undergraduates!



Why study algorithms?

To solve problems that could not otherwise be addressed.

Ex. Network connectivity. [stay tuned]



Why study algorithms?

For intellectual stimulation.

> " *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.* " — *D. E. Knuth*

Why study algorithms?

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

Computational models are replacing mathematical models in scientific enquiry

$$E = mc^2$$
$$F = ma$$
$$F = \frac{G m_1 m_2}{r^2}$$
$$\left[-\frac{\hbar^2}{2m} \nabla^2 + V(r) \right] \Psi(r) = E\ \Psi(r)$$

20th century science
(formula based)

```
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.

Nintendo
Adobe
Google


D E Shaw & Co
Morgan Stanley
Microsoft
YAHOO!
PIXAR

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.
- They may unlock the secrets of life and of the universe.
- For fun and profit.

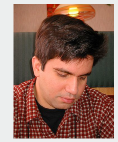
Why study anything else?

The usual suspects

Lectures. Introduce new material, answer questions.

first precept meets Thursday

Precepts. Answer questions, solve problems, discuss programming assignment.

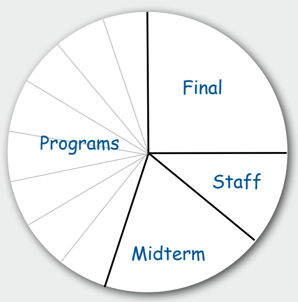



#	When	Where	Who	Office hours
L01	M W 11-12:20	Frist 302	Kevin Wayne	M 2-3, T 11-12
P01	Th 12:30	Friend 110	Moses Charikar	T 2:30-4:30
P01A	Th 12:30	Friend 111	Szymon Rusinkiewicz	F 1-3
P02	Th 1:30	Friend 007	Szymon Rusinkiewicz	F 1-3
P02A	Th 1:30	Friend 110	Moses Charikar	T 2:30-4:30
P03	Th 3:30	Friend 109	Nadia Heninger	M 3-5

Coursework and grading

8 programming assignments. 45%
- Available via course website.
- Due 11:55pm, starting Tuesday 2/12.
- Electronic submission.

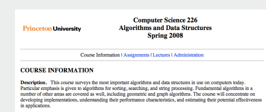
Exams.
- Closed-book with cheatsheet.
- Midterm. 20%
- Final. 25%



Staff discretion. 10%
- Participation in lecture and precepts.
- Everyone needs to meet me at office hours!

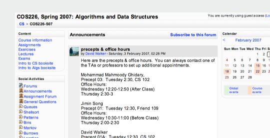
Resources (web)

Course content.
- Course info.
- Lecture slides.
- Programming assignments.



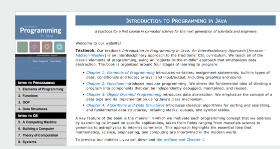
`http://www.princeton.edu/~cos226`

Course administration.
- Check grades.
- Submit assignments.



`https://moodle.cs.princeton.edu/course/view.php?id=31`

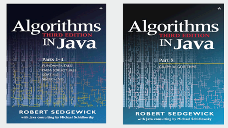
Booksites.
- Brief summary of content.
- Download code from lecture.



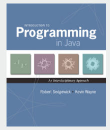
`http://www.cs.princeton.edu/IntroProgramming`
`http://www.cs.princeton.edu/algs4`

Resources (books)

Algorithms in Java, 3rd edition
- Parts 1-4. [sorting, searching]
- Part 5. [graph algorithms]



Introduction to Programming in Java
- Basic programming model.
- Elementary AofA and data structures.



Algorithms in Pascal(!)/C/C++, 2nd edition
- Strings.
- Geometric algorithms.

Algorithms in Java 4th edition [in preparation]

Questions

Not registered? You can't submit assignments until register in SCORE.

Change precept? Make the precept change in SCORE yourself;
see Donna O'Leary (CS 410) for serious conflicts.

Union-Find Algorithms

- dynamic connectivity
- quick find
- quick union
- improvements
- applications

Subtext of today's lecture (and this course)

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.

The scientific method.

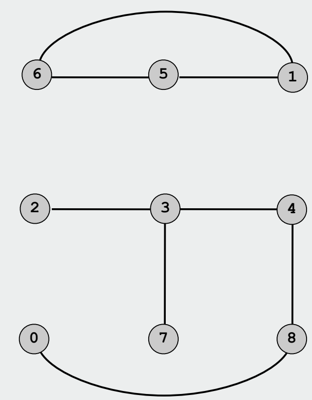
Mathematical analysis.

- dynamic connectivity
- quick find
- quick union
- improvements
- applications

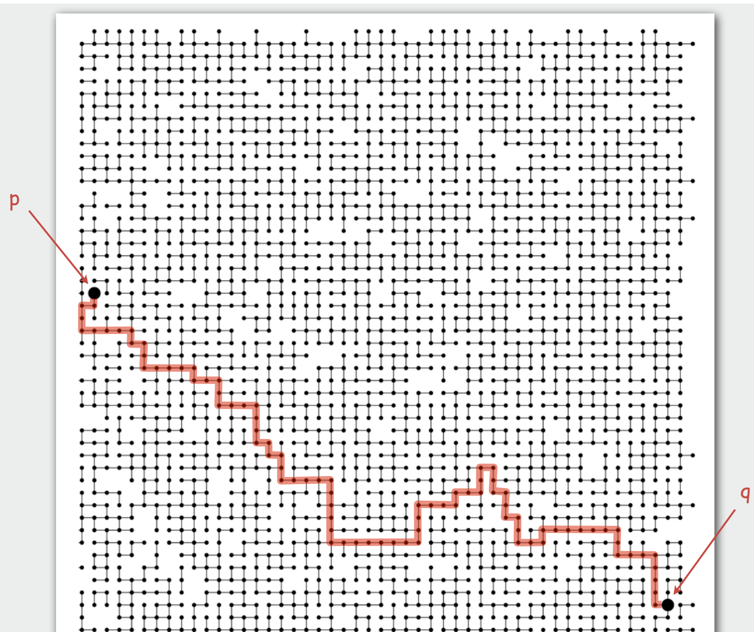
Dynamic connectivity

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

```
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
```



Network connectivity: larger example



Modeling the objects

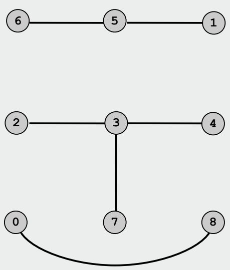
Dynamic connectivity applications involve manipulating objects of all types.
- Variable name aliases.
- Pixels in a digital photo.
- Computers in a network.
- Web pages on the Internet.
- Transistors in a computer chip.
- Metallic sites in a composite system.

When programming, convenient to name objects 0 to N-1.
- Use integers as array index.
- Suppress details not relevant to union-find.
- Could use symbol table to translate from object names to integers.

Modeling the connections

Transitivity. If `p` is connected to `q` and `q` is connected to `r`, then `p` is connected to `r`.

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



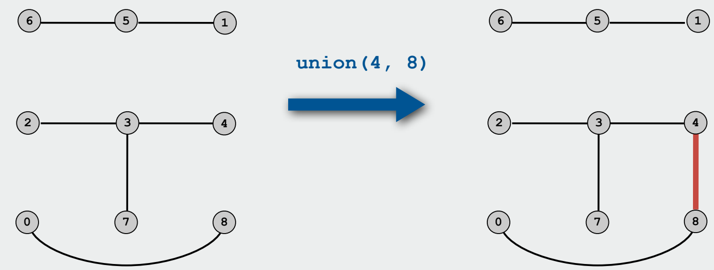
`{ 1 5 6 } { 2 3 4 7 } { 0 8 }`

connected components

Implementing the operations

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

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



`{ 1 5 6 } { 2 3 4 7 } { 0 8 }` → `{ 1 5 6 } { 0 2 3 4 7 8 }`

Union-find data type

Goal. Design efficient data structure for union-find.

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

public class UnionFind		
	`UnionFind(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 subset?*
`void`	`union(int p, int q)`	*replace subsets containing p and q with their union*

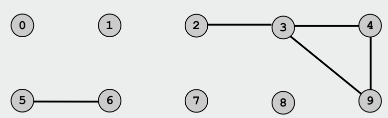
Quick-find [eager approach]

Data structure.

- Integer array `id[]` of size `N`.
- Interpretation: `p` and `q` are connected if 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` are connected if 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 not connected

Union. To merge subsets 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

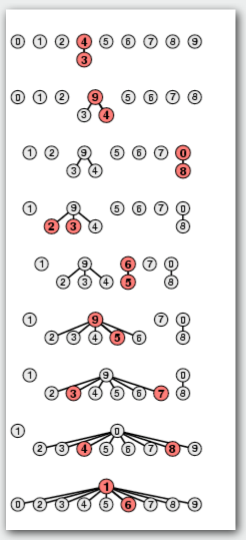
union of 3 and 6
2, 3, 4, 5, 6, and 9 are connected

problem: many values can change

Quick-find example

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

problem: many values can change



Quick-find: Java implementation

```java
public class QuickFind
{
   private int[] id;

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

set id of each object to itself
(N operations)

check if p and q have same id
(1 operation)

change all entries with id[p] to id[q]
(N operations)

Quick-find is too slow

Quick-find defect.

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

algorithm	union	find
quick-find	N	1

Ex. May take N^2 operations to process N union commands on N objects.

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!

- dynamic connectivity
- quick find
- **quick union**
- improvements
- applications

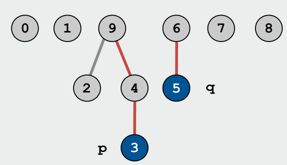
Quick-union [lazy approach]

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

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



3's root is 9; 5's root is 6

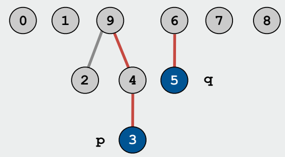
Quick-union [lazy approach]

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

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



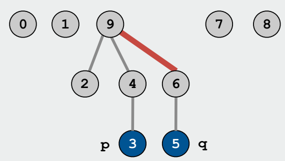
3's root is 9; 5's root is 6
3 and 5 are not connected

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

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

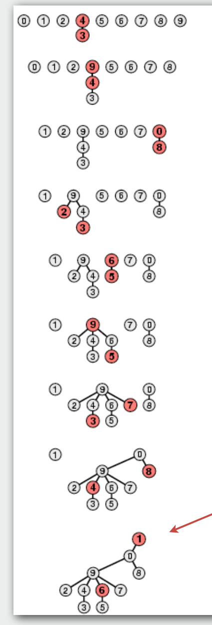
i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	9	7	8	9

only one value changes



Quick-union example

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



problem:
trees can get tall

Quick-union: Java implementation

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

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

   private int root(int i)
   {
      while (i != id[i]) i = id[i];
      return i;
   }

   public boolean find(int p, int q)
   {
      return root(p) == root(q);
   }

   public void unite(int p, int q)
   {
      int i = root(p), j = root(q);
      id[i] = j;
   }
}
```

set id of each object to itself
(N operations)

chase parent parents until reach root
(depth of i operations)

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

change root of p to point to root of q
(depth of p and q operations)

Quick-union is also too slow

Quick-find defect.

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

Quick-union defect.

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

algorithm	union	find
quick-find	N	1
quick-union	N *	N

worst case

* includes cost of finding root

- dynamic connectivity
- quick find
- quick union
- **improvements**
- applications

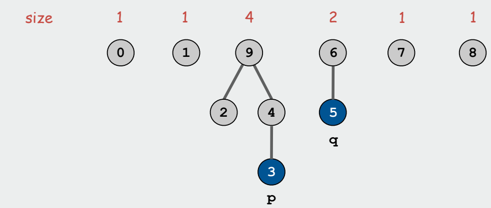
Improvement 1: weighting

Weighted quick-union.

- Modify quick-union to avoid tall trees.
- Keep track of size of each subset.
- Balance by linking small tree below large one.

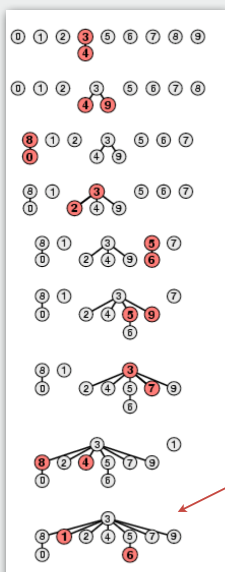
Ex. Union of 3 and 5.

- Quick union: link 9 to 6.
- Weighted quick union: link 6 to 9.



Weighted quick-union example

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



no problem:
trees stay flat

Weighted quick-union: Java implementation

Data structure. Same as quick-union, but maintain extra array `sz[i]` to count number of objects in the tree rooted at `i`.

Find. Identical to quick-union.

```
return root(p) == root(q);
```

Union. Modify quick-union to:

- Merge smaller tree into larger tree.
- Update the `sz[]` array.

```
int i = root(p), j = root(q);
if  (sz[i] < sz[j]) { id[i] = j; sz[j] += sz[i]; }
else                { id[j] = i; sz[i] += sz[j]; }
```

Weighted quick-union analysis

Analysis.

- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.
- Fact: depth is at most lg N. [needs proof]

algorithm	union	find
quick-find	N	1
quick-union	N *	N
weighted QU	lg N *	lg N

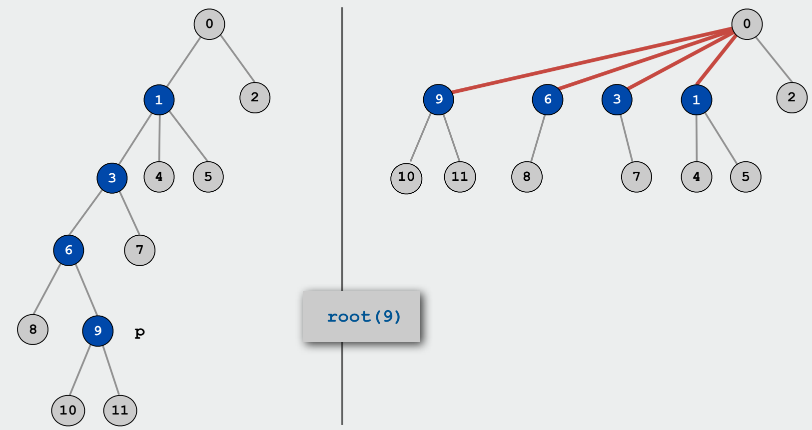
* includes cost of find

Q. Stop at guaranteed acceptable performance?

A. No, easy to improve further.

Improvement 2: path compression

Quick union with path compression. Just after computing the root of `p`, set the `id` of each examined node to `root(p)`.



Path compression: Java implementation

Standard implementation: add second loop to `root()` to set the id of each examined node to the root.

Simpler one-pass variant: make every other node in path point to its grandparent.

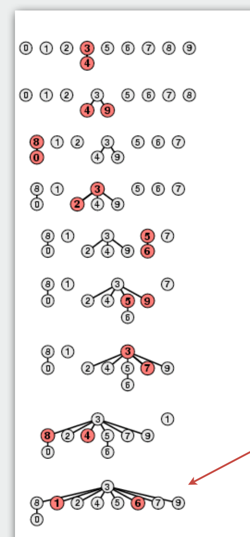
```
public int root(int i)
{
   while (i != id[i])
   {
      id[i] = id[id[i]];
      i = id[i];
   }
   return i;
}
```

only one extra line of code !

In practice. No reason not to! Keeps tree almost completely flat.

Weighted quick-union with path compression example

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	3	3	3	3



no problem: trees stay VERY flat

WQUPC performance

Theorem. [Tarjan 1975] Starting from an empty data structure, any sequence of M union and find operations on N objects takes O(N + M lg* N) time.

- Proof is very difficult.
- But the algorithm is still simple!

actually O(N + M α(M, N))
see COS 423

Linear algorithm?

- Cost within constant factor of reading in the data.
- In theory, WQUPC is not quite linear.
- In practice, WQUPC is linear.

because lg* N is a constant in this universe

N	lg* N
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

lg* function
number of times needed to take the lg of a number until reaching 1

Amazing fact. No linear-time linking strategy exists.

Summary

Bottom line. WQUPC makes it possible to solve problems that could not otherwise be addressed.

algorithm	worst-case time
quick-find	M N
quick-union	M N
weighted QU	N + M log N
QU + path compression	N + M log N
weighted QU + path compression	(M + N) lg* N

M union-find operations on a set of N objects

Ex. [10^9 unions and finds with 10^9 objects]

- WQUPC reduces time from 30 years to 6 seconds.
- Supercomputer won't help much; good algorithm enables solution.

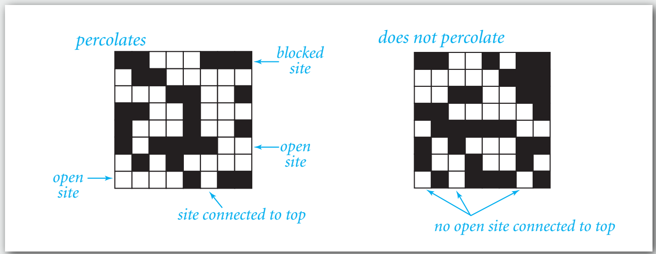
Union-find applications

- Percolation.
- Games (Go, Hex).
- Image processing.
- ✓ Network connectivity.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Morphological attribute openings and closings.

Percolation

A model for many physical systems:

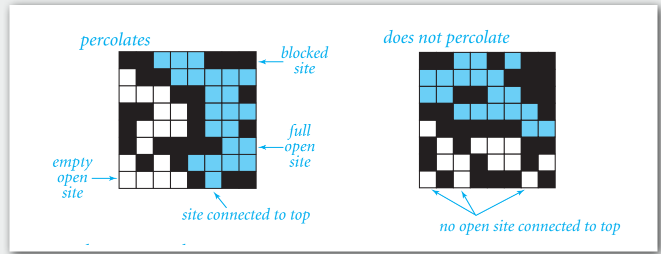
- N-by-N grid of sites.
- Each site is open with probability p (or blocked with probability 1-p).
- System percolates if top and bottom are connected by open sites.



Percolation

A model for many physical systems:

- N-by-N grid of sites.
- Each site is open with probability p (or blocked with probability 1-p).
- System percolates if top and bottom are connected by open sites.



Percolation

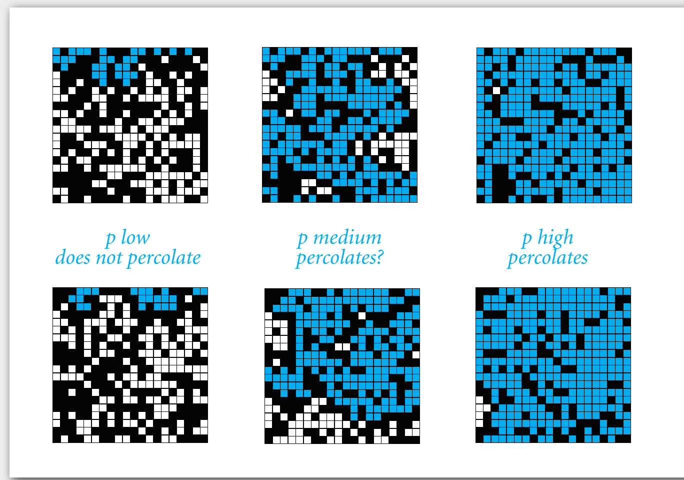
A model for many physical systems:

- N-by-N grid of sites.
- Each site is open with probability p (or blocked with probability 1-p).
- System percolates if top and bottom are connected by open sites.

model	system	vacant site	occupied site	percolates
electricity	material	conductor	insulated	conducts
fluid flow	material	empty	blocked	porous
social interaction	population	person	empty	communicates

Likelihood of percolation

Depends on site vacancy probability p.

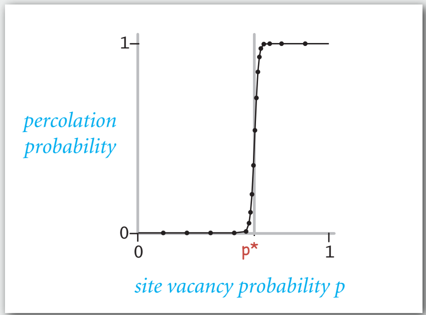


Percolation phase transition

Theory guarantees a sharp threshold p*.

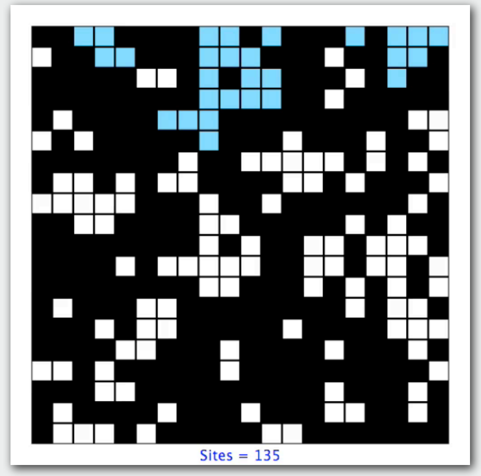
- p > p*: almost certainly percolates.
- p < p*: almost certainly does not percolate.

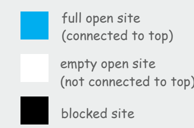
Q. What is the value of p* ?



Monte Carlo simulation

- Initialize whole grid to be blocked.
- Make random sites open until top connected to bottom.
- vacancy percentage estimates p*.



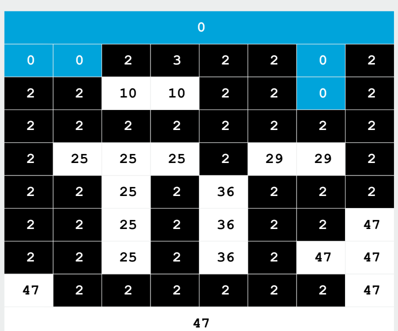


UF solution to find percolation threshold

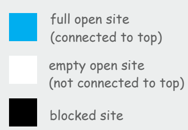
How to check whether system percolates?

- Create object for each site (and virtual top and bottom objects).
- Sites are in same set if connected by vacant sites.
- Percolates if top and bottom row are in same set.

virtual top row →



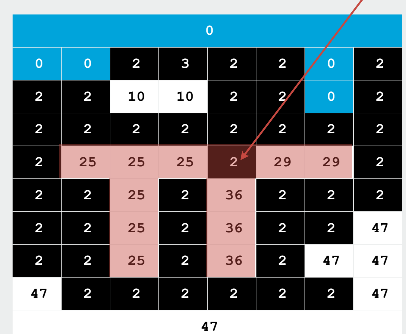
virtual bottom row →

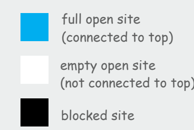


UF solution to find percolation threshold

Q. How to make declare a new site vacant?

declare this site vacant

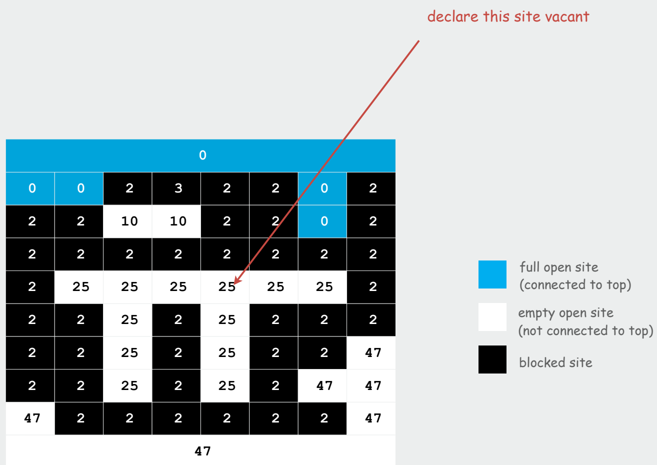




UF solution to find percolation threshold

Q. How to make declare a new site vacant?

A. Take union of site and all adjacent vacant sites.

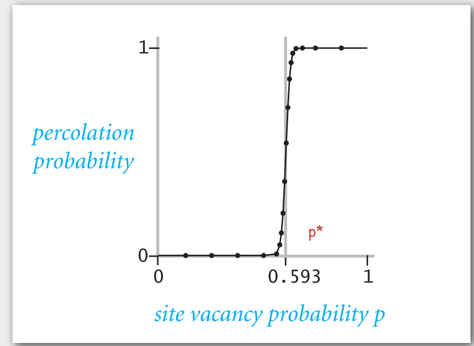


Percolation threshold

Q. What is percolation threshold p* ?

A. About 0.592746 for large square lattices.

percolation constant known
only via simulation



Subtext of today's lecture (and this course)

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.

The scientific method.

Mathematical analysis.