

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

ALGORITHMS PART I

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

Course overview

What is this course?

- 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, bag, union-find, priority queue
sorting	quicksort, mergesort, heapsort
searching	BST, red-black BST, hash table
graphs	BFS, DFS, Prim, Kruskal, Dijkstra
strings	radix sorts, tries, KMP, regexps, data compression
advanced	B-tree, suffix array, maxflow

part 1

part 2

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, ...

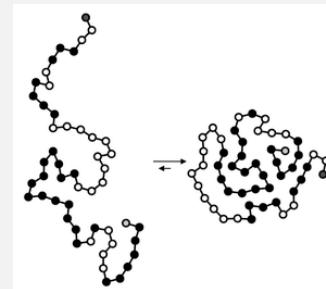
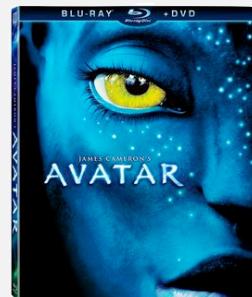
Multimedia. MP3, JPG, DivX, HDTV, face recognition, ...

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

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

⋮

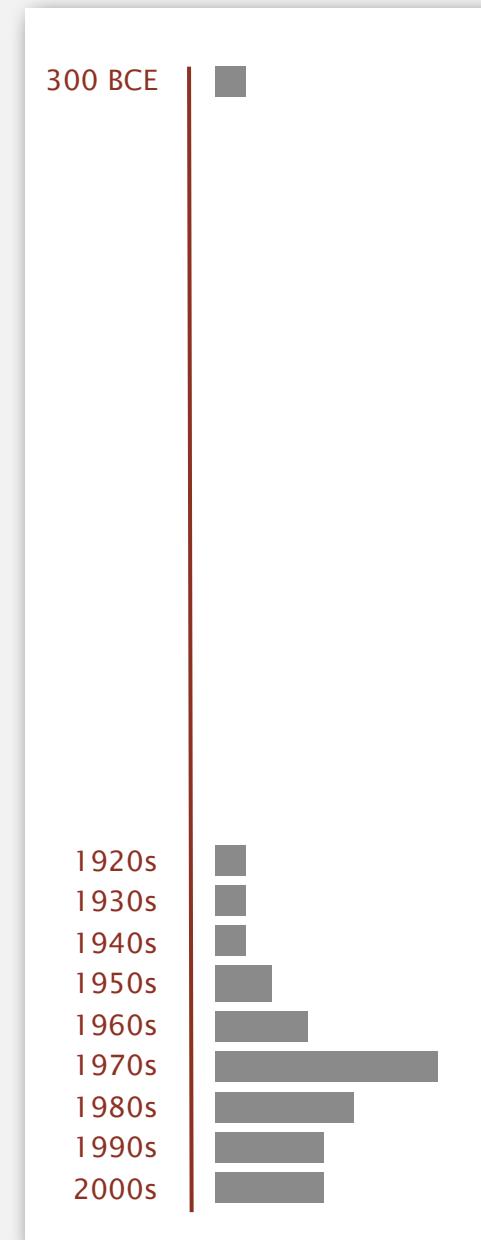
Google™
YAHOO!
bing™



Why study algorithms?

Old roots, new opportunities.

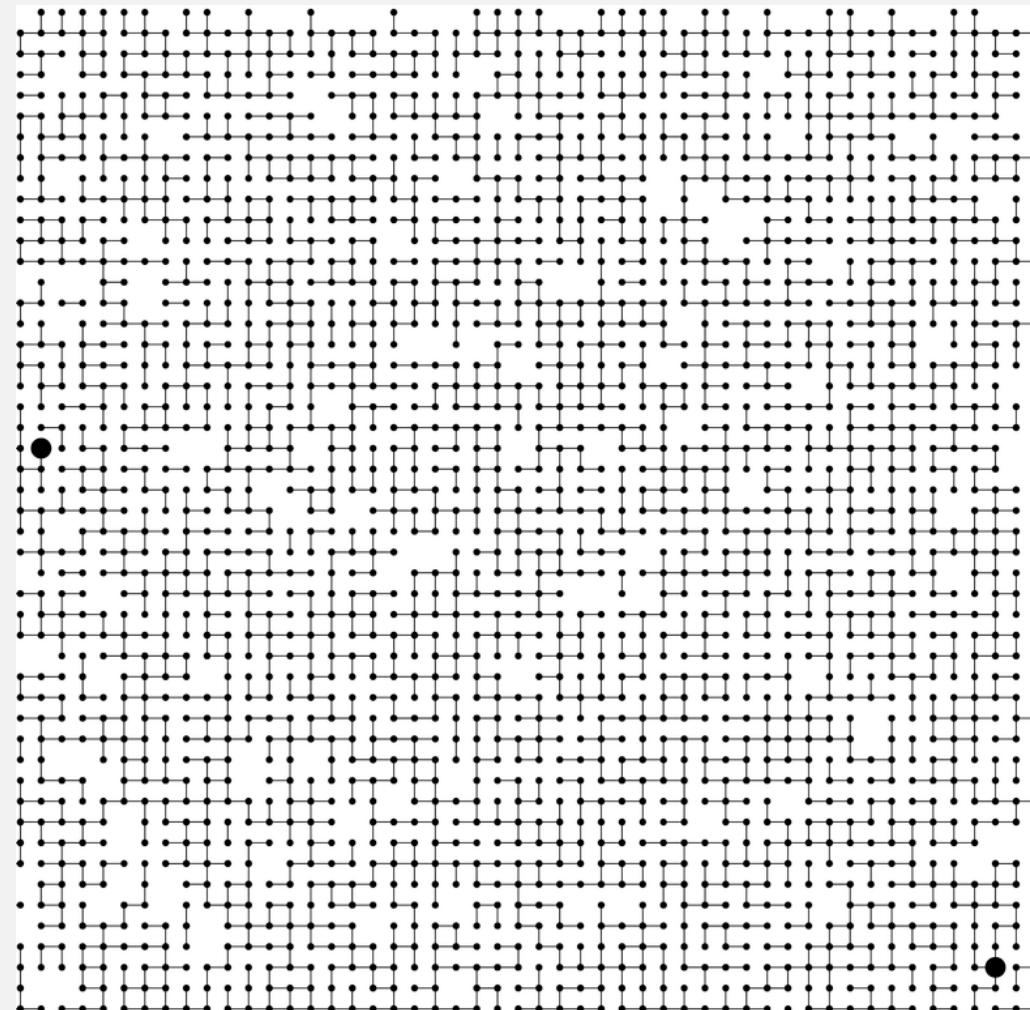
- Study of algorithms dates at least to Euclid.
- Formalized by Church and Turing in 1930s.
- Some important algorithms were discovered by undergraduates in a course like this!



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

FROM THE
EDITORS

THE JOY OF ALGORITHMS



Francis Sullivan, Associate Editor-in-Chief

The theme of this first-of-the-century issue of COMPUTING IN SCIENCE & ENGINEERING IS ALGORITHMS. IN FACT, WE'RE BOLD ENOUGH—AND PERHAPS FOOLISH ENOUGH—to call the examples we've selected "THE JOY OF ALGORITHMS OF THE CENTURY."

Computational algorithms are probably as old as civilization. Seminal algorithms, one of the most ancient written records, appeared in the Rhind Mathematical Papyrus, dating from about 1650 BC. And suppose we could claim that the Drazil algorithm for estimating the square root of a number was the first algorithm? (That's really hard to believe.)

Like it or not, though, the technology effects algorithms have advanced in startling and unexpected ways in the 20th century. In fact, it's because of the technology effects, algorithms we chose for this issue have enabled the progress in computing that has transformed our world. From weather prediction, defense, and fundamental science, to medical breakthroughs, to the Internet, to the latest video game, to chess-beating algorithms. I recall one last night had written on the chalkboard: "An algorithm must be seen to be believed." I might add that they don't look very appealing. (After all, most people don't know what an algorithm is, and most people don't know what must be the right answer—usually.) Anyways, hang on to your hats.

The Rivalry "Society is the method of invention." The computer revolution has shown that this is true. The computer revolution brings insights that suggest the next, usually much larger leap forward. The computer revolution has shown us how to bridge the gap between the demand for cycles and the availability of cycles. The computer revolution has shown us that Moore's Law holds for over many, many months. In effect, Moore's Law holds for over many, many years. The computer revolution shows us that the complexity of an algorithm grows at most linearly as a function of problem size. Important algorithms become more important, and less important algorithms become less important. The computer revolution has shown us that the complexity of an algorithm does not change the expression of this complexity!

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

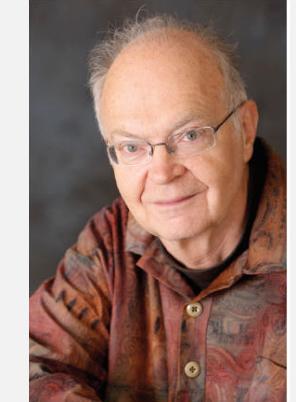
The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

The computer revolution has shown us that the theory of computation is not just a theory. It is a theory that can be applied to real-world problems.

“An algorithm must be seen to be believed.” — Donald Knuth



Why study algorithms?

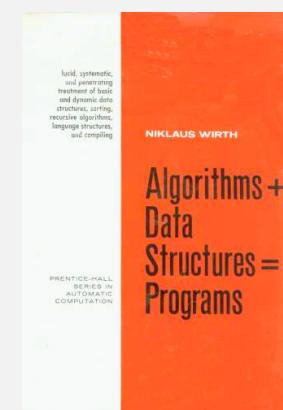
To become a proficient programmer.

“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.

$$E = mc^2$$

$$F = ma$$

$$\left[-\frac{\hbar^2}{2m} \nabla^2 + V(r) \right] \Psi(r) = E \Psi(r)$$

20th century science
(formula based)

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

```
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)

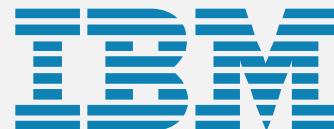
“Algorithms: a common language for nature, human, and computer.” — Avi Wigderson

Why study algorithms?

For fun and profit.



Morgan Stanley



DE Shaw & Co

ORACLE®



YAHOO!

amazon.com

Microsoft®

P X A R
ANIMATION STUDIOS

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.

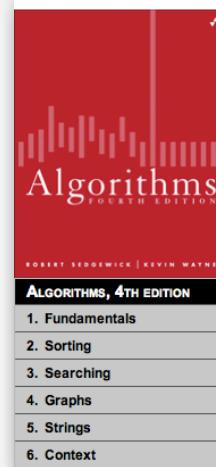
Why study anything else?



Resources

Booksite.

- Lecture slides.
- Download code.
- Summary of content.



ALGORITHMS, 4TH EDITION

essential information that
every serious programmer
needs to know about
algorithms and data structures

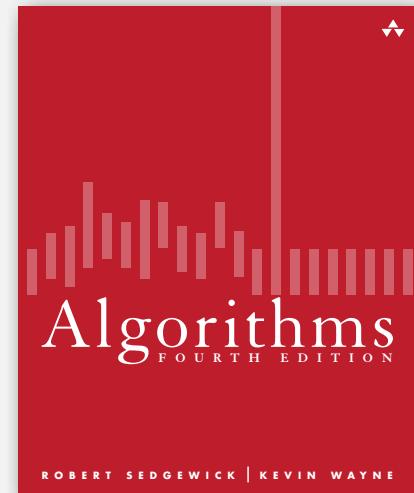
Textbook. The textbook *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne [[Amazon](#) · [Addison-Wesley](#)] surveys the most important algorithms and data structures in use today. The textbook is organized into six chapters:

- *Chapter 1: Fundamentals* introduces a scientific and engineering basis for comparing algorithms and making predictions. It also includes our programming model.
- *Chapter 2: Sorting* considers several classic sorting algorithms, including insertion sort, mergesort, and quicksort. It also includes a binary heap implementation of a priority queue.
- *Chapter 3: Searching* describes several classic symbol table implementations, including binary search trees, red-black trees, and hash tables.

<http://algs4.cs.princeton.edu>

Textbook (optional).

- *Algorithms, 4th edition* by Sedgewick and Wayne.
- More extensive coverage of topics.
- More topics.



ISBN 0-321-57351-X

Prerequisites

Prerequisites.

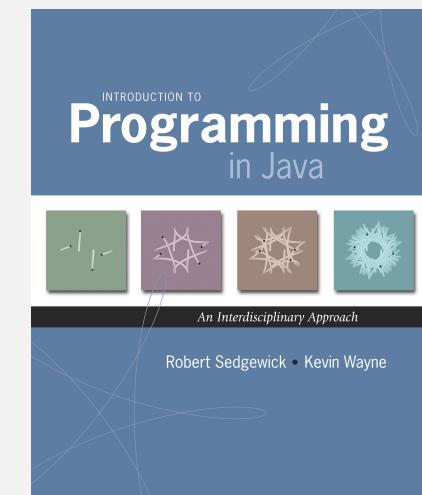
- Programming: loops, arrays, functions, objects, recursion.
- Java: we use as expository language.
- Mathematics: high-school algebra.

Review of prerequisite material.

- Quick: Sections 1.1 and 1.2 of *Algorithms, 4th edition*.
- In-depth: *An Introduction to programming in Java: an interdisciplinary approach* by Sedgewick and Wayne.

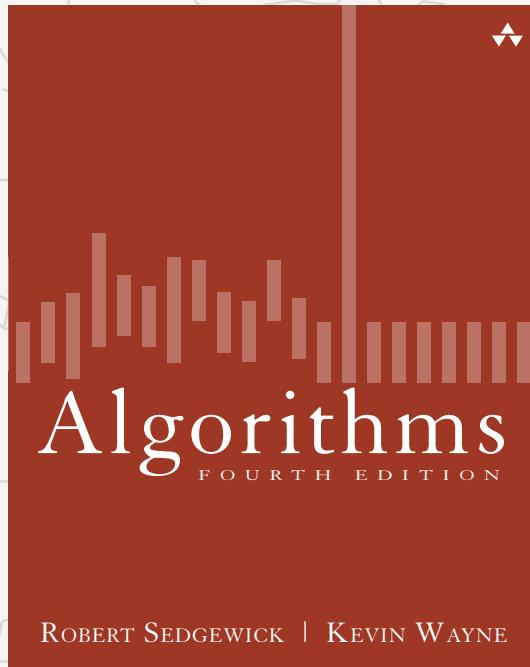
Programming environment.

- Use your own, e.g., Eclipse.
- Download ours (see instructions on web).



Quick exercise. Write a Java program.

ISBN 0-321-49805-4
<http://introcs.cs.princeton.edu>



<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

Cast of characters



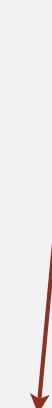
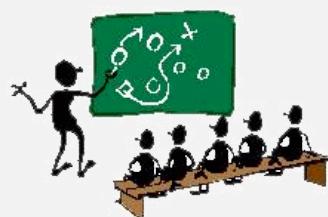
Programmer needs to develop
a working solution.



Client wants to solve
problem efficiently.



Theoretician wants
to understand.

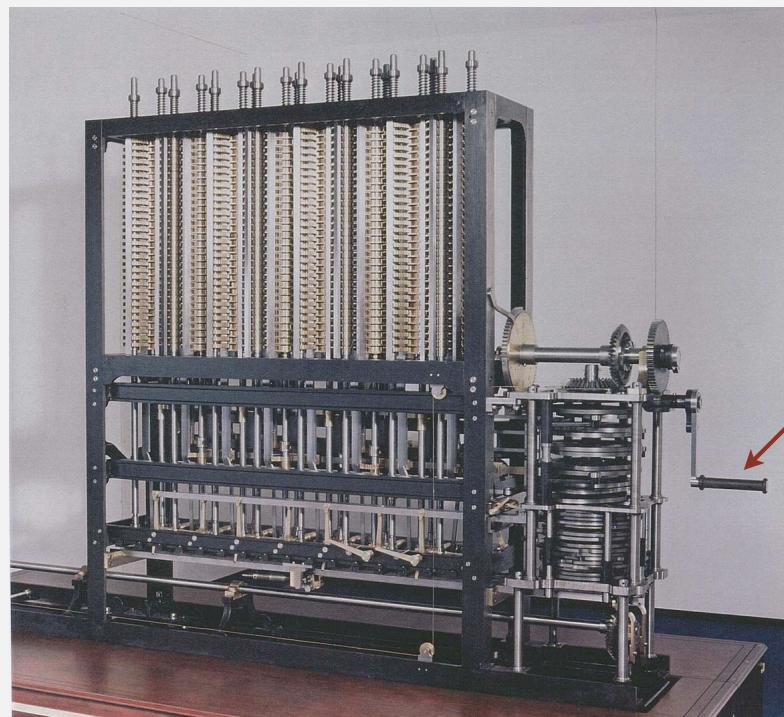
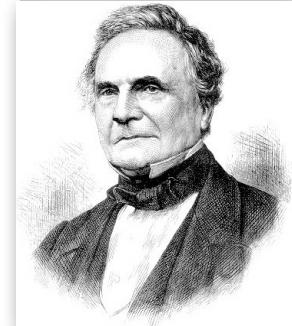


Student might play
any or all of these
roles someday.

Basic **blocking** and **tackling**
is sometimes necessary.
[this lecture]

Running time

“As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time? ” — Charles Babbage (1864)

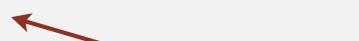


how many times do you have to turn the crank?

Analytic Engine

Reasons to analyze algorithms

Predict performance.

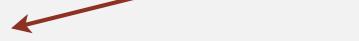


this course

Compare algorithms.



Provide guarantees.



Understand theoretical basis.



theory of algorithms

Primary practical reason: avoid performance bugs.



**client gets poor performance because programmer
did not understand performance characteristics**



Some algorithmic successes

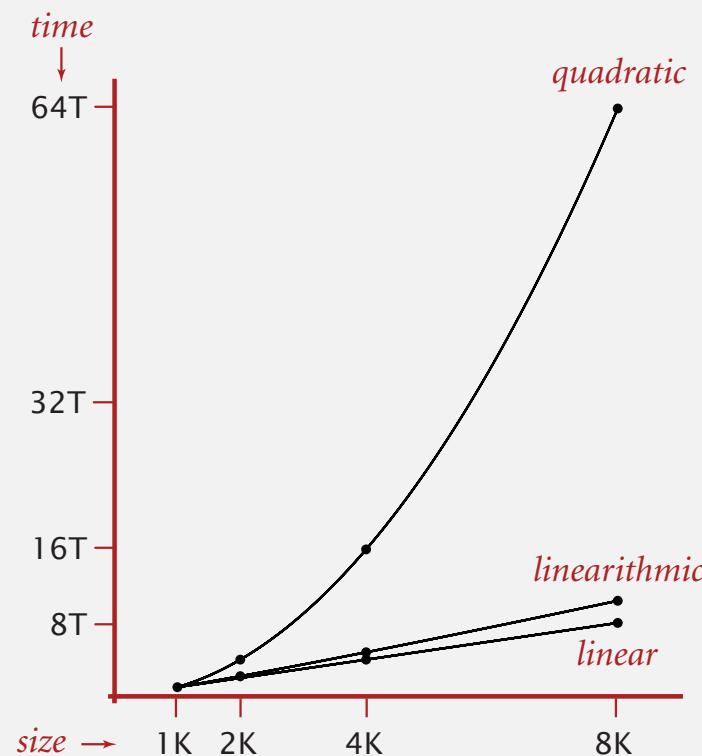
Discrete Fourier transform.

- Break down waveform of N samples into periodic components.
- Applications: DVD, JPEG, MRI, astrophysics,
- Brute force: N^2 steps.
- FFT algorithm: $N \log N$ steps, **enables new technology**.



Friedrich Gauss

1805



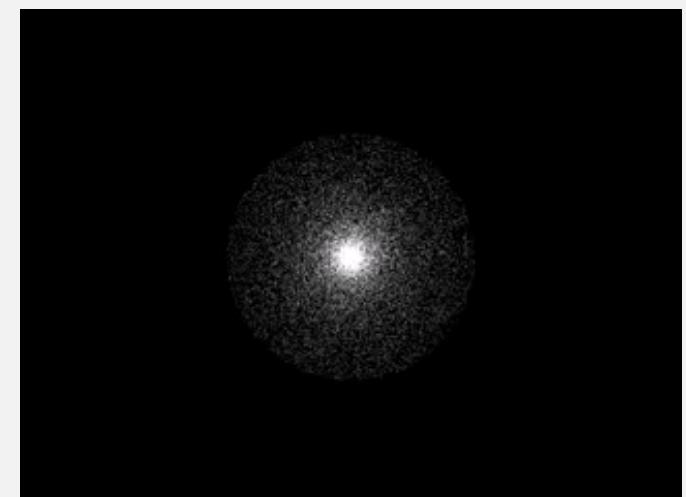
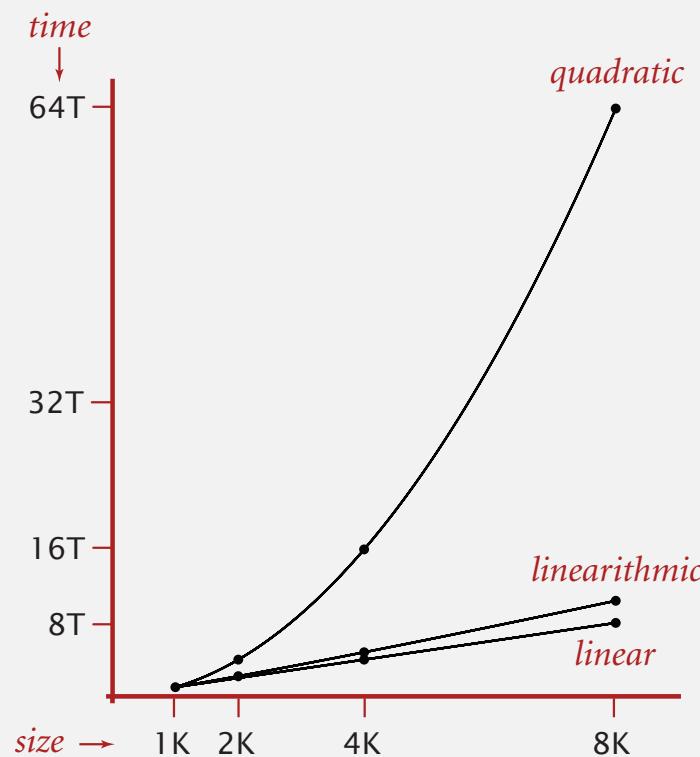
Some algorithmic successes

N-body simulation.

- Simulate gravitational interactions among N bodies.
- Brute force: N^2 steps.
- Barnes-Hut algorithm: $N \log N$ steps, enables new research.



Andrew Appel
PU '81



The challenge

Q. Will my program be able to solve a large practical input?

Why is my program so slow ?

Why does it run out of memory ?



Insight. [Knuth 1970s] Use scientific method to understand performance.

Scientific method applied to analysis of algorithms

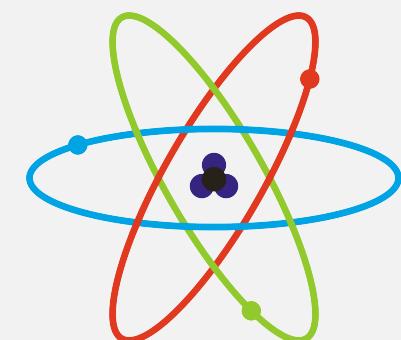
A framework for predicting performance and comparing algorithms.

Scientific method.

- **Observe** some feature of the natural world.
- **Hypothesize** a model that is consistent with the observations.
- **Predict** events using the hypothesis.
- **Verify** the predictions by making further observations.
- **Validate** by repeating until the hypothesis and observations agree.

Principles.

- Experiments must be **reproducible**.
- Hypotheses must be **falsifiable**.



Feature of the natural world. Computer itself.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ ***observations***
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

Example: 3-SUM

3-SUM. Given N distinct integers, how many triples sum to exactly zero?

```
% more 8ints.txt  
8  
30 -40 -20 -10 40 0 10 5  
  
% java ThreeSum 8ints.txt  
4
```

	a[i]	a[j]	a[k]	sum
1	30	-40	10	0
2	30	-20	-10	0
3	-40	40	0	0
4	-10	0	10	0

Context. Deeply related to problems in computational geometry.

3-SUM: brute-force algorithm

```
public class ThreeSum
{
    public static int count(int[] a)
    {
        int N = a.length;
        int count = 0;
        for (int i = 0; i < N; i++)
            for (int j = i+1; j < N; j++)
                for (int k = j+1; k < N; k++) ← check each triple
                    if (a[i] + a[j] + a[k] == 0) ← for simplicity, ignore
                        count++;                                integer overflow
        return count;
    }

    public static void main(String[] args)
    {
        int[] a = In.readInts(args[0]);
        StdOut.println(count(a));
    }
}
```

Measuring the running time

Q. How to time a program?

A. Manual.



% java ThreeSum 1Kints.txt



70

% java ThreeSum 2Kints.txt



528

% java ThreeSum 4Kints.txt



4039

Measuring the running time

Q. How to time a program?

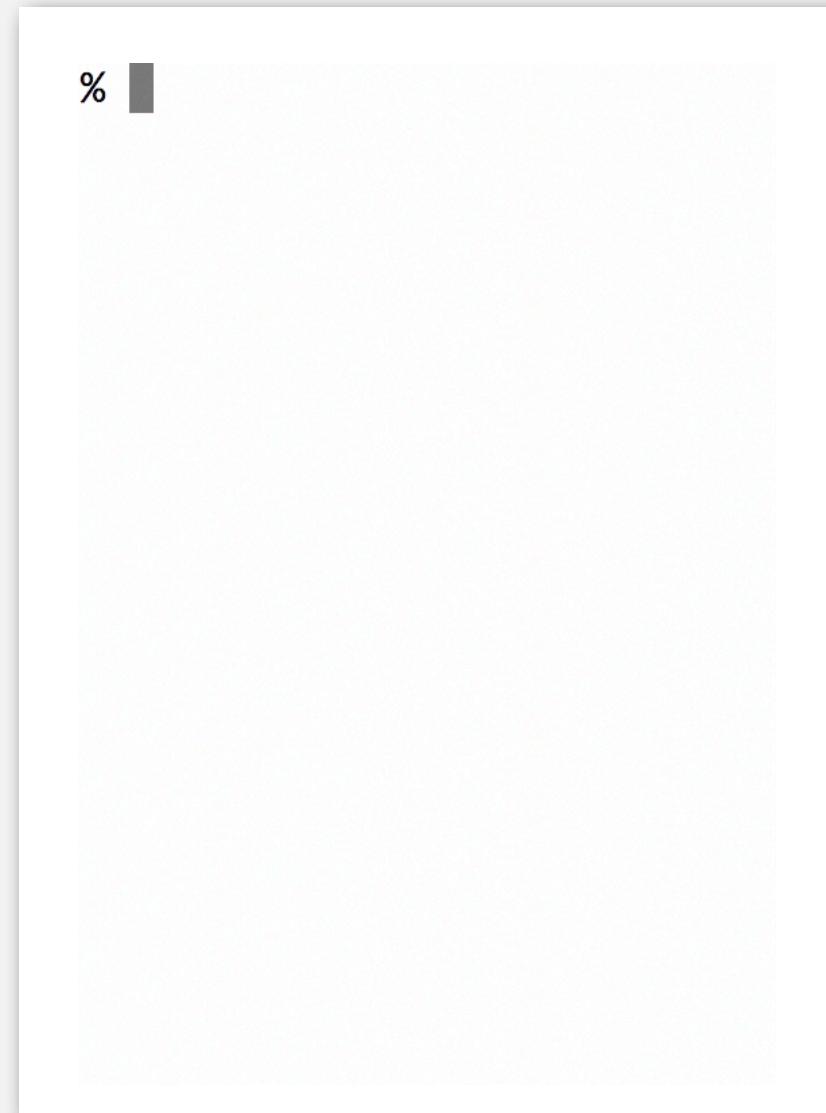
A. Automatic.

```
public class Stopwatch  (part of stdlib.jar )  
  
    Stopwatch()           create a new stopwatch  
  
    double elapsedTime() time since creation (in seconds)
```

```
public static void main(String[] args)  
{  
    int[] a = In.readInts(args[0]);  
    Stopwatch stopwatch = new Stopwatch();  
    StdOut.println(ThreeSum.count(a));  
    double time = stopwatch.elapsedTime();  
}
```

Empirical analysis

Run the program for various input sizes and measure running time.



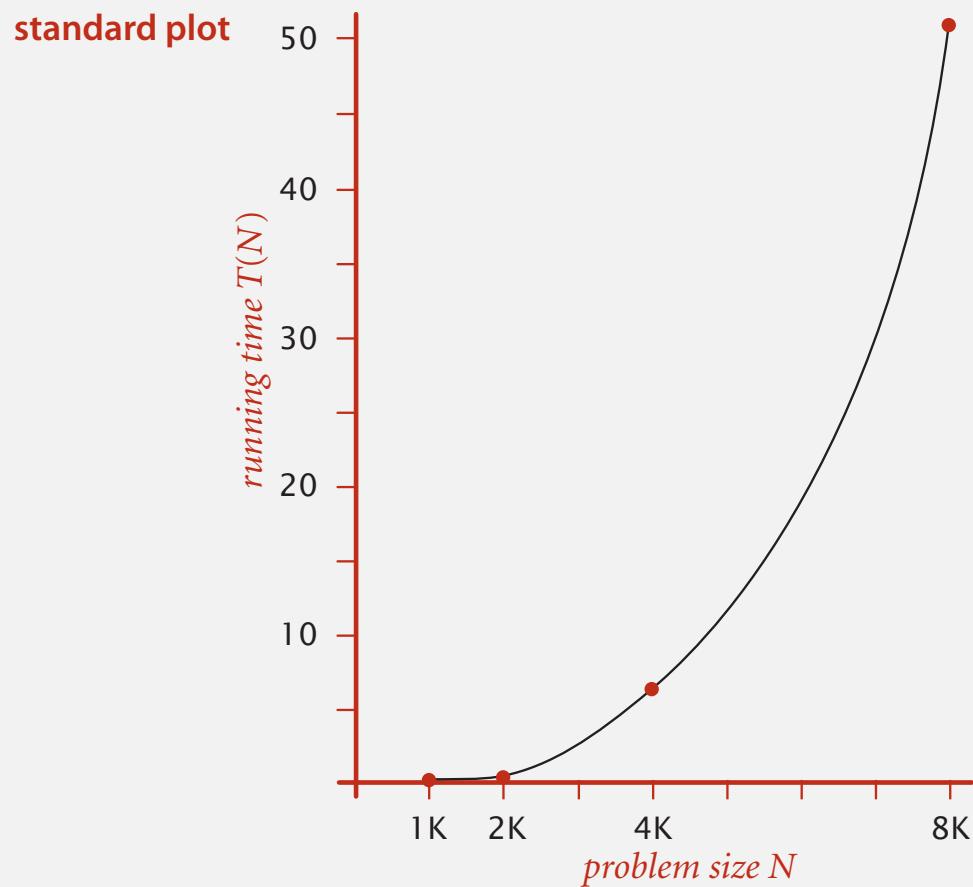
Empirical analysis

Run the program for various input sizes and measure running time.

N	time (seconds) †
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

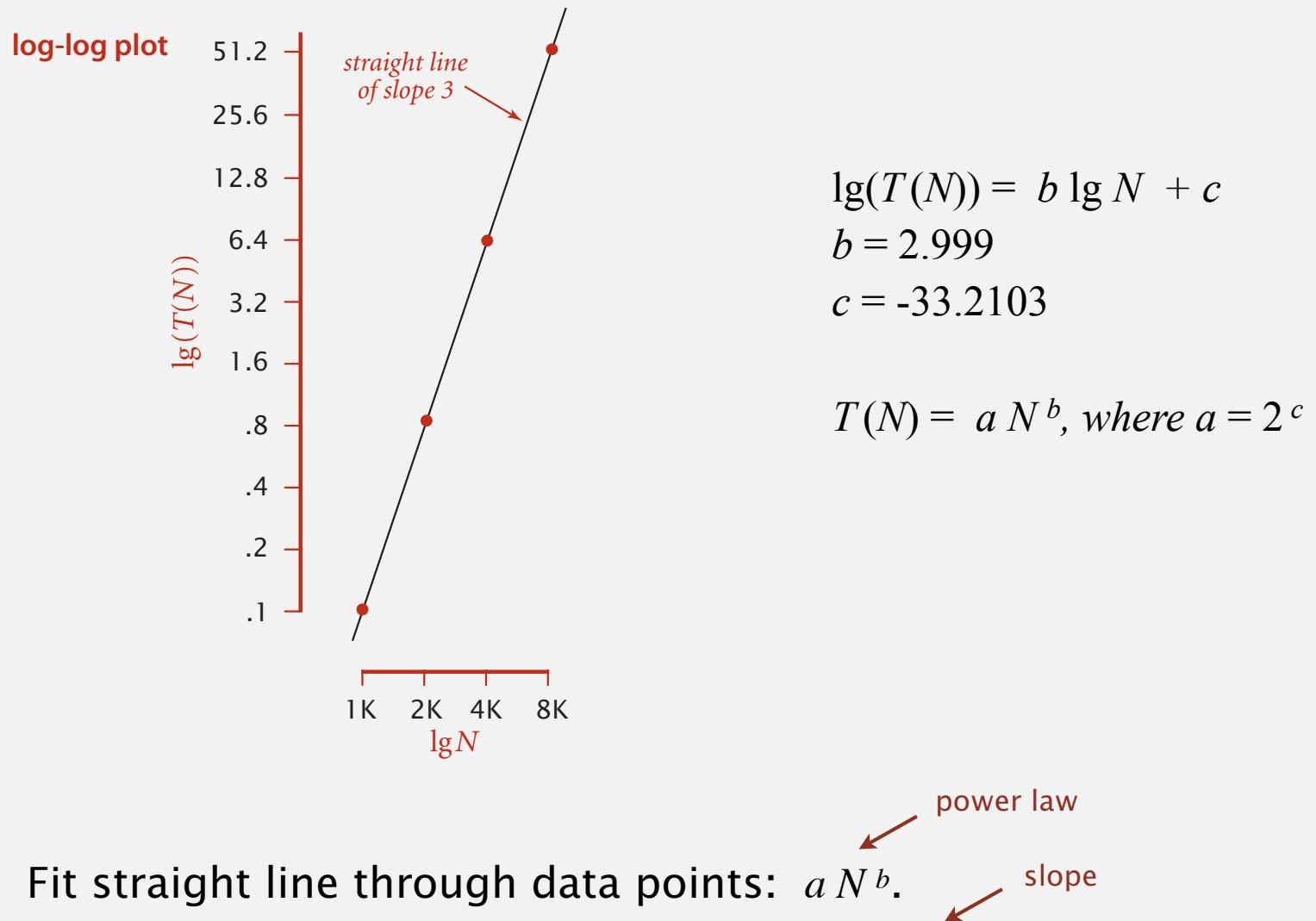
Data analysis

Standard plot. Plot running time $T(N)$ vs. input size N .



Data analysis

Log-log plot. Plot running time $T(N)$ vs. input size N using log-log scale.



Regression. Fit straight line through data points: $a N^b$.

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.

Prediction and validation

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.



"order of growth" of running time is about N^3 [stay tuned]

Predictions.

- 51.0 seconds for $N = 8,000$.
- 408.1 seconds for $N = 16,000$.

Observations.

N	time (seconds) †
8,000	51.1
8,000	51.0
8,000	51.1
16,000	410.8

validates hypothesis!

Doubling hypothesis

Doubling hypothesis. Quick way to estimate b in a power-law relationship.

Run program, **doubling** the size of the input.

N	time (seconds) [†]	ratio	lg ratio
250	0.0		—
500	0.0	4.8	2.3
1,000	0.1	6.9	2.8
2,000	0.8	7.7	2.9
4,000	6.4	8.0	3.0
8,000	51.1	8.0	3.0

↑
seems to converge to a constant $b \approx 3$

Hypothesis. Running time is about $a N^b$ with $b = \lg$ ratio.

Caveat. Cannot identify logarithmic factors with doubling hypothesis.

Doubling hypothesis

Doubling hypothesis. Quick way to estimate b in a power-law relationship.

Q. How to estimate a (assuming we know b) ?

A. Run the program (for a sufficient large value of N) and solve for a .

N	time (seconds) †
8,000	51.1
8,000	51.0
8,000	51.1

$$51.1 = a \times 8000^3$$

$$\Rightarrow a = 0.998 \times 10^{-10}$$

Hypothesis. Running time is about $0.998 \times 10^{-10} \times N^3$ seconds.



almost identical hypothesis
to one obtained via linear regression

Experimental algorithmics

System independent effects.

- Algorithm.
 - Input data.
- 
- determines exponent b
in power law

System dependent effects.

- Hardware: CPU, memory, cache, ...
 - Software: compiler, interpreter, garbage collector, ...
 - System: operating system, network, other apps, ...
- 
- determines constant a
in power law

Bad news. Difficult to get precise measurements.

Good news. Much easier and cheaper than other sciences.



e.g., can run huge number of experiments

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ ***observations***
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

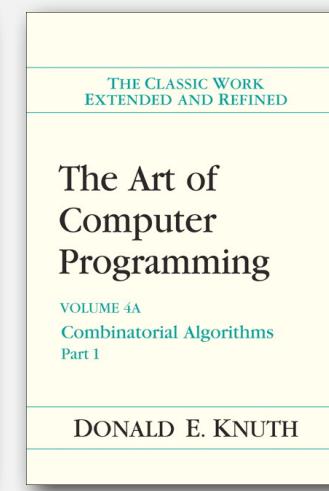
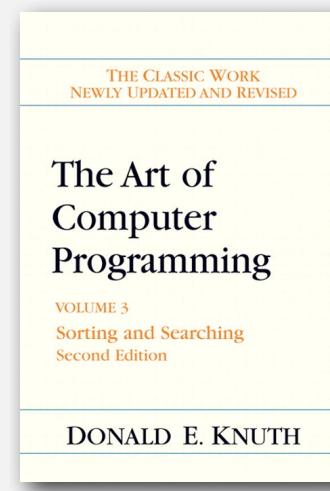
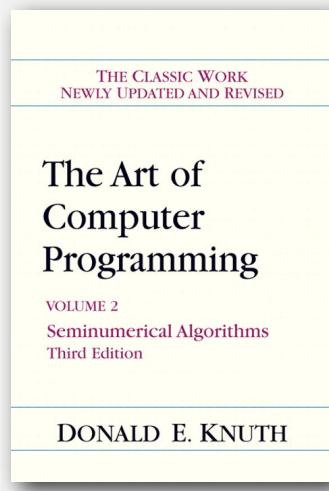
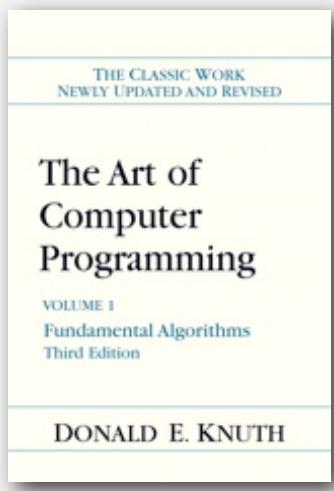
1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ ***mathematical models***
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

Mathematical models for running time

Total running time: sum of cost \times frequency for all operations.

- Need to analyze program to determine set of operations.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.



Donald Knuth
1974 Turing Award

In principle, accurate mathematical models are available.

Cost of basic operations

operation	example	nanoseconds †
integer add	$a + b$	2.1
integer multiply	$a * b$	2.4
integer divide	a / b	5.4
floating-point add	$a + b$	4.6
floating-point multiply	$a * b$	4.2
floating-point divide	a / b	13.5
sine	<code>Math.sin(theta)</code>	91.3
arctangent	<code>Math.atan2(y, x)</code>	129.0
...

† Running OS X on Macbook Pro 2.2GHz with 2GB RAM

Cost of basic operations

operation	example	nanoseconds †
variable declaration	<code>int a</code>	c_1
assignment statement	<code>a = b</code>	c_2
integer compare	<code>a < b</code>	c_3
array element access	<code>a[i]</code>	c_4
array length	<code>a.length</code>	c_5
1D array allocation	<code>new int[N]</code>	$c_6 N$
2D array allocation	<code>new int[N][N]</code>	$c_7 N^2$
string length	<code>s.length()</code>	c_8
substring extraction	<code>s.substring(N/2, N)</code>	c_9
string concatenation	<code>s + t</code>	$c_{10} N$

Novice mistake. Abusive string concatenation.

Example: 1-SUM

Q. How many instructions as a function of input size N ?

```
int count = 0;  
for (int i = 0; i < N; i++)  
    if (a[i] == 0)  
        count++;
```

operation	frequency
variable declaration	2
assignment statement	2
less than compare	$N + 1$
equal to compare	N
array access	N
increment	N to $2N$

Example: 2-SUM

Q. How many instructions as a function of input size N ?

```
int count = 0;  
for (int i = 0; i < N; i++)  
    for (int j = i+1; j < N; j++)  
        if (a[i] + a[j] == 0)  
            count++;
```

$$\begin{aligned}0 + 1 + 2 + \dots + (N - 1) &= \frac{1}{2}N(N - 1) \\&= \binom{N}{2}\end{aligned}$$

operation	frequency
variable declaration	$N + 2$
assignment statement	$N + 2$
less than compare	$\frac{1}{2}(N + 1)(N + 2)$
equal to compare	$\frac{1}{2}N(N - 1)$
array access	$N(N - 1)$
increment	$\frac{1}{2}N(N - 1)$ to $N(N - 1)$

tedious to count exactly

Simplifying the calculations

“It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then given them various weights. We might, for instance, count the number of additions, subtractions, multiplications, divisions, recording of numbers, and extractions of figures from tables. In the case of computing with matrices most of the work consists of multiplications and writing down numbers, and we shall therefore only attempt to count the number of multiplications and recordings. ” — Alan Turing

ROUNDING-OFF ERRORS IN MATRIX PROCESSES

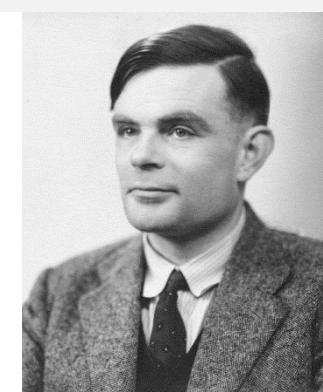
By A. M. TURING

(National Physical Laboratory, Teddington, Middlesex)

[Received 4 November 1947]

SUMMARY

A number of methods of solving sets of linear equations and inverting matrices are discussed. The theory of the rounding-off errors involved is investigated for some of the methods. In all cases examined, including the well-known ‘Gauss elimination process’, it is found that the errors are normally quite moderate: no exponential build-up need occur.



Simplification 1: cost model

Cost model. Use some basic operation as a proxy for running time.

```
int count = 0;  
for (int i = 0; i < N; i++)  
    for (int j = i+1; j < N; j++)  
        if (a[i] + a[j] == 0)  
            count++;
```

$$\begin{aligned}0 + 1 + 2 + \dots + (N - 1) &= \frac{1}{2}N(N - 1) \\&= \binom{N}{2}\end{aligned}$$

operation	frequency
variable declaration	$N + 2$
assignment statement	$N + 2$
less than compare	$\frac{1}{2}(N + 1)(N + 2)$
equal to compare	$\frac{1}{2}N(N - 1)$
array access	$N(N - 1)$
increment	$\frac{1}{2}N(N - 1)$ to $N(N - 1)$

cost model = array accesses
(we assume compiler/JVM do not optimize array accesses away!)

Simplification 2: tilde notation

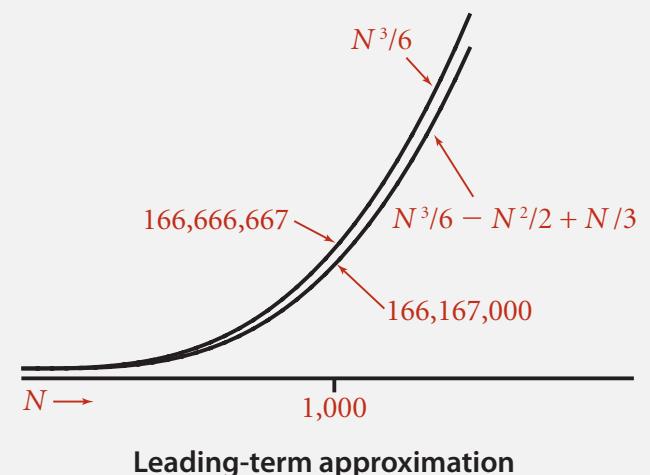
- Estimate running time (or memory) as a function of input size N .
- Ignore lower order terms.
 - when N is large, terms are negligible
 - when N is small, we don't care

Ex 1. $\frac{1}{6}N^3 + 20N + 16 \sim \frac{1}{6}N^3$

Ex 2. $\frac{1}{6}N^3 + 100N^{4/3} + 56 \sim \frac{1}{6}N^3$

Ex 3. $\frac{1}{6}N^3 - \frac{1}{2}N^2 + \frac{1}{3}N \sim \frac{1}{6}N^3$

discard lower-order terms
(e.g., $N = 1000$: 500 thousand vs. 166 million)



Technical definition. $f(N) \sim g(N)$ means $\lim_{N \rightarrow \infty} \frac{f(N)}{g(N)} = 1$

Simplification 2: tilde notation

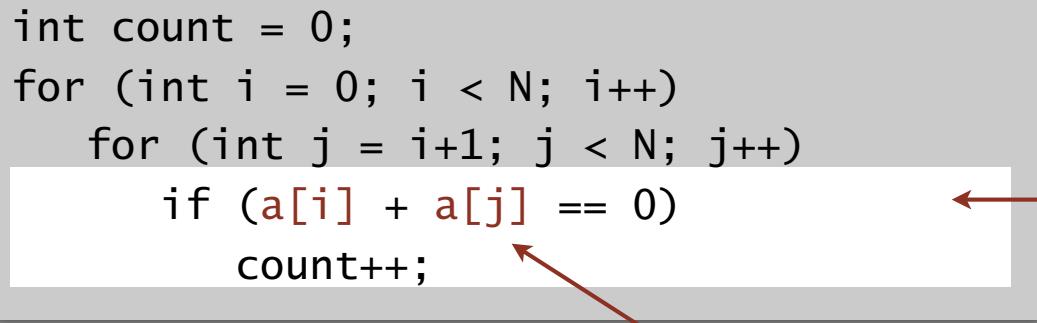
- Estimate running time (or memory) as a function of input size N .
- Ignore lower order terms.
 - when N is large, terms are negligible
 - when N is small, we don't care

operation	frequency	tilde notation
variable declaration	$N + 2$	$\sim N$
assignment statement	$N + 2$	$\sim N$
less than compare	$\frac{1}{2} (N + 1) (N + 2)$	$\sim \frac{1}{2} N^2$
equal to compare	$\frac{1}{2} N (N - 1)$	$\sim \frac{1}{2} N^2$
array access	$N (N - 1)$	$\sim N^2$
increment	$\frac{1}{2} N (N - 1)$ to $N (N - 1)$	$\sim \frac{1}{2} N^2$ to $\sim N^2$

Example: 2-SUM

Q. Approximately how many array accesses as a function of input size N ?

```
int count = 0;  
for (int i = 0; i < N; i++)  
    for (int j = i+1; j < N; j++)  
        if (a[i] + a[j] == 0)  
            count++;
```



A. $\sim N^2$ array accesses.

$$\begin{aligned}0 + 1 + 2 + \dots + (N - 1) &= \frac{1}{2} N(N - 1) \\&= \binom{N}{2}\end{aligned}$$

Bottom line. Use cost model and tilde notation to simplify counts.

Example: 3-SUM

Q. Approximately how many array accesses as a function of input size N ?

```
int count = 0;
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j++)
        for (int k = j+1; k < N; k++)
            if (a[i] + a[j] + a[k] == 0) ← "inner loop"
                count++;
```

A. $\sim \frac{1}{2} N^3$ array accesses.

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!}$$
$$\sim \frac{1}{6} N^3$$

Bottom line. Use cost model and tilde notation to simplify counts.

Estimating a discrete sum

Q. How to estimate a discrete sum?

A1. Take discrete mathematics course.

A2. Replace the sum with an integral, and use calculus!

Ex 1. $1 + 2 + \dots + N$.

$$\sum_{i=1}^N i \sim \int_{x=1}^N x dx \sim \frac{1}{2} N^2$$

Ex 2. $1^k + 2^k + \dots + N^k$.

$$\sum_{i=1}^N i^k \sim \int_{x=1}^N x^k dx \sim \frac{1}{k+1} N^{k+1}$$

Ex 3. $1 + 1/2 + 1/3 + \dots + 1/N$.

$$\sum_{i=1}^N \frac{1}{i} \sim \int_{x=1}^N \frac{1}{x} dx = \ln N$$

Ex 4. 3-sum triple loop.

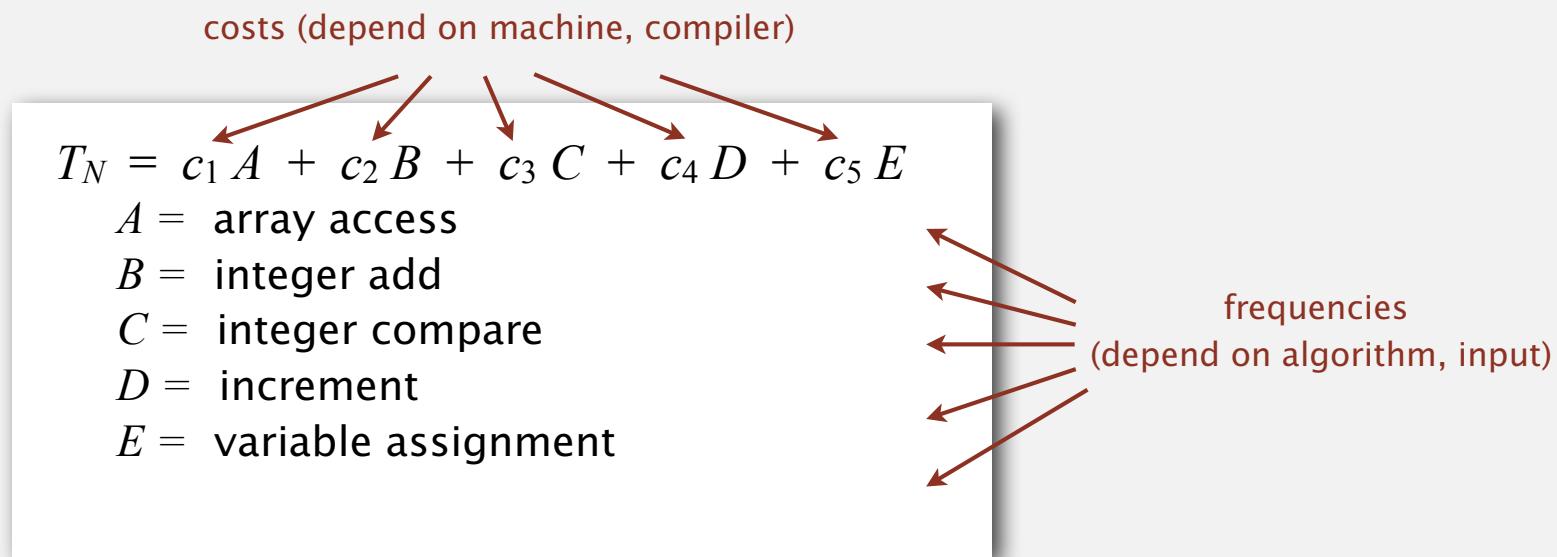
$$\sum_{i=1}^N \sum_{j=i}^N \sum_{k=j}^N 1 \sim \int_{x=1}^N \int_{y=x}^N \int_{z=y}^N dz dy dx \sim \frac{1}{6} N^3$$

Mathematical models for running time

In principle, accurate mathematical models are available.

In practice,

- Formulas can be complicated.
- Advanced mathematics might be required.
- Exact models best left for experts.



Bottom line. We use approximate models in this course: $T(N) \sim c N^3$.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ ***mathematical models***
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ ***order-of-growth classifications***
- ▶ *theory of algorithms*
- ▶ *memory*

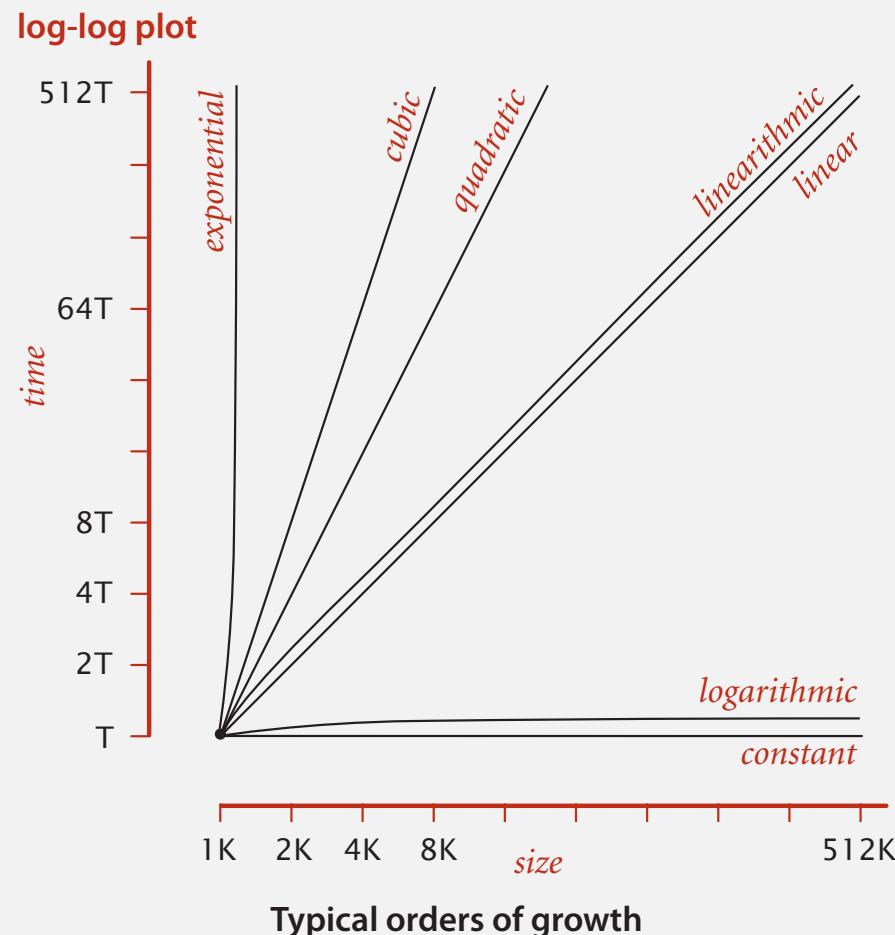
Common order-of-growth classifications

Good news. the small set of functions

$1, \log N, N, N \log N, N^2, N^3$, and 2^N

suffices to describe order-of-growth of typical algorithms.

order of growth discards
leading coefficient



Common order-of-growth classifications

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	constant	$a = b + c;$	statement	add two numbers	1
$\log N$	logarithmic	<pre>while (N > 1) { N = N / 2; ... }</pre>	divide in half	binary search	~ 1
N	linear	<pre>for (int i = 0; i < N; i++) { ... }</pre>	loop	find the maximum	2
$N \log N$	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }</pre>	double loop	check all pairs	4
N^3	cubic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }</pre>	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$

Practical implications of order-of-growth

growth rate	problem size solvable in minutes			
	1970s	1980s	1990s	2000s
1	any	any	any	any
$\log N$	any	any	any	any
N	millions	tens of millions	hundreds of millions	billions
$N \log N$	hundreds of thousands	millions	millions	hundreds of millions
N^2	hundreds	thousand	thousands	tens of thousands
N^3	hundred	hundreds	thousand	thousands
2^N	20	20s	20s	30

Bottom line. Need linear or linearithmic alg to keep pace with Moore's law.

Binary search demo

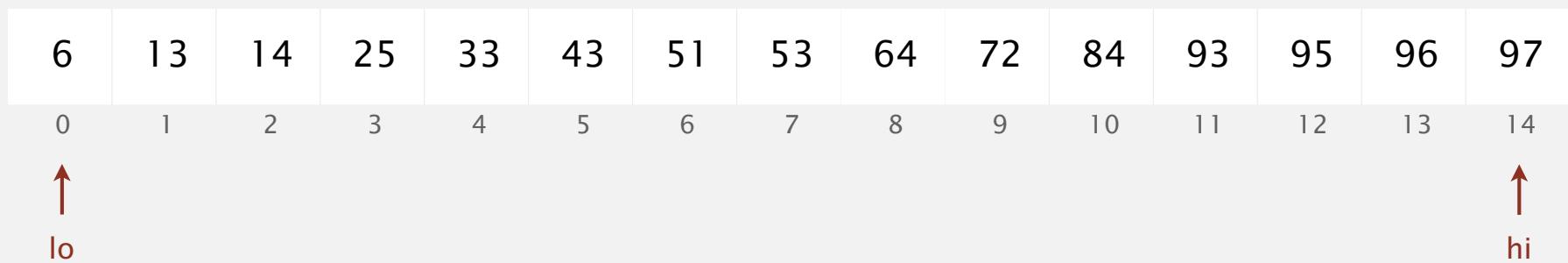
Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.



successful search for 33



Binary search: Java implementation

Trivial to implement?

- First binary search published in 1946; first bug-free one in 1962.
- Bug in Java's `Arrays.binarySearch()` discovered in 2006.

```
public static int binarySearch(int[] a, int key)
{
    int lo = 0, hi = a.length-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        if      (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

one "3-way compare"

Invariant. If key appears in the array `a[]`, then $a[lo] \leq key \leq a[hi]$.

Binary search: mathematical analysis

Proposition. Binary search uses at most $1 + \lg N$ key compares to search in a sorted array of size N .

Def. $T(N) = \# \text{ key compares to binary search a sorted subarray of size } \leq N$.

Binary search recurrence. $T(N) \leq T(N/2) + 1$ for $N > 1$, with $T(1) = 1$.

↑
left or right half

↑
possible to implement with one
2-way compare (instead of 3-way)

Pf sketch.

$$\begin{aligned} T(N) &\leq T(N/2) + 1 && \text{given} \\ &\leq T(N/4) + 1 + 1 && \text{apply recurrence to first term} \\ &\leq T(N/8) + 1 + 1 + 1 && \text{apply recurrence to first term} \\ &\dots \\ &\leq T(N/N) + 1 + 1 + \dots + 1 && \text{stop applying, } T(1) = 1 \\ &= 1 + \lg N \end{aligned}$$

An $N^2 \log N$ algorithm for 3-SUM

Sorting-based algorithm.

- Step 1: Sort the N (distinct) numbers.
- Step 2: For each pair of numbers $a[i]$ and $a[j]$, binary search for $-(a[i] + a[j])$.

input

30 -40 -20 -10 40 0 10 5

sort

-40 -20 -10 0 5 10 30 40

binary search

(-40, -20)	60
(-40, -10)	50
(-40, 0)	40
(-40, 5)	35
(-40, 10)	30
⋮	⋮
(-40, 40)	0
⋮	⋮
(-20, -10)	30
⋮	⋮
(-10, 0)	10
⋮	⋮
(10, 30)	-40
(10, 40)	-50
(30, 40)	-70

only count if
 $a[i] < a[j] < a[k]$
to avoid
double counting

Analysis. Order of growth is $N^2 \log N$.

- Step 1: N^2 with insertion sort.
- Step 2: $N^2 \log N$ with binary search.

Comparing programs

Hypothesis. The sorting-based $N^2 \log N$ algorithm for 3-SUM is significantly faster in practice than the brute-force N^3 algorithm.

N	time (seconds)
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1

ThreeSum.java

N	time (seconds)
1,000	0.14
2,000	0.18
4,000	0.34
8,000	0.96
16,000	3.67
32,000	14.88
64,000	59.16

ThreeSumDeluxe.java

Guiding principle. Typically, better order of growth \Rightarrow faster in practice.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ ***order-of-growth classifications***
- ▶ *theory of algorithms*
- ▶ *memory*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ ***theory of algorithms***
- ▶ *memory*

Types of analyses

Best case. Lower bound on cost.

- Determined by “easiest” input.
- Provides a goal for all inputs.

Worst case. Upper bound on cost.

- Determined by “most difficult” input.
- Provides a guarantee for all inputs.

Average case. Expected cost for random input.

- Need a model for “random” input.
- Provides a way to predict performance.

Ex 1. Array accesses for brute-force 3-SUM.

Best: $\sim \frac{1}{2} N^3$

Average: $\sim \frac{1}{2} N^3$

Worst: $\sim \frac{1}{2} N^3$

Ex 2. Comparisons for binary search.

Best: ~ 1

Average: $\sim \lg N$

Worst: $\sim \lg N$

Types of analyses

Best case. Lower bound on cost.

Worst case. Upper bound on cost.

Average case. “Expected” cost.

Actual data might not match input model?

- Need to understand input to effectively process it.
- Approach 1: design for the worst case.
- Approach 2: randomize, depend on probabilistic guarantee.

Theory of algorithms

Goals.

- Establish “difficulty” of a problem.
- Develop “optimal” algorithms.

Approach.

- Suppress details in analysis: analyze “to within a constant factor”.
- Eliminate variability in input model by focusing on the worst case.

Optimal algorithm.

- Performance guarantee (to within a constant factor) for any input.
- No algorithm can provide a better performance guarantee.

Commonly-used notations in the theory of algorithms

notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ 10 N^2 $5 N^2 + 22 N \log N + 3N$ ⋮	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	10 N^2 100 N $22 N \log N + 3 N$ ⋮	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$ ⋮	develop lower bounds

Theory of algorithms: example 1

Goals.

- Establish “difficulty” of a problem and develop “optimal” algorithms.
- Ex. 1-SUM = “*Is there a 0 in the array?*”

Upper bound.

A specific algorithm.

- Ex. Brute-force algorithm for 1-SUM: Look at every array entry.
- Running time of the optimal algorithm for 1-SUM is $O(N)$.

Lower bound.

Proof that no algorithm can do better.

- Ex. Have to examine all N entries (any unexamined one might be 0).
- Running time of the optimal algorithm for 1-SUM is $\Omega(N)$.

Optimal algorithm.

- Lower bound equals upper bound (to within a constant factor).
- Ex. Brute-force algorithm for 1-SUM is optimal: its running time is $\Theta(N)$.

Theory of algorithms: example 2

Goals.

- Establish “difficulty” of a problem and develop “optimal” algorithms.
- Ex. 3-SUM.

Upper bound. A specific algorithm.

- Ex. Brute-force algorithm for 3-SUM.
- Running time of the optimal algorithm for 3-SUM is $O(N^3)$.

Theory of algorithms: example 2

Goals.

- Establish “difficulty” of a problem and develop “optimal” algorithms.
- Ex. 3-SUM.

Upper bound.

A specific algorithm.

- Ex. Improved algorithm for 3-SUM.
- Running time of the optimal algorithm for 3-SUM is $O(N^2 \log N)$.

Lower bound.

Proof that no algorithm can do better.

- Ex. Have to examine all N entries to solve 3-SUM.
- Running time of the optimal algorithm for solving 3-SUM is $\Omega(N)$.

Open problems.

- Optimal algorithm for 3-SUM?
- Subquadratic algorithm for 3-SUM?
- Quadratic lower bound for 3-SUM?

Algorithm design approach

Start.

- Develop an algorithm.
- Prove a lower bound.

Gap?

- Lower the upper bound (discover a new algorithm).
- Raise the lower bound (more difficult).

Golden Age of Algorithm Design.

- 1970s-.
- Steadily decreasing upper bounds for many important problems.
- Many known optimal algorithms.

Caveats.

- Overly pessimistic to focus on worst case?
- Need better than “to within a constant factor” to predict performance.

Commonly-used notations

notation	provides	example	shorthand for	used to
Tilde	leading term	$\sim 10 N^2$	$10 N^2$ $10 N^2 + 22 N \log N$ $10 N^2 + 2 N + 37$	provide approximate model
Big Theta	asymptotic growth rate	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$	classify algorithms
Big Oh	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$	develop lower bounds

Common mistake. Interpreting big-Oh as an approximate model.

This course. Focus on approximate models: use Tilde-notation

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ ***theory of algorithms***
- ▶ *memory*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ ***memory***

Basics

Bit. 0 or 1. NIST most computer scientists
Byte. 8 bits. ↓ ↓
Megabyte (MB). 1 million or 2^{20} bytes.
Gigabyte (GB). 1 billion or 2^{30} bytes.



64-bit machine. We assume a 64-bit machine with 8 byte pointers.

- Can address more memory.
- Pointers use more space.

some JVMs "compress" ordinary object
pointers to 4 bytes to avoid this cost



Typical memory usage for primitive types and arrays

type	bytes
boolean	1
byte	1
char	2
int	4
float	4
long	8
double	8

for primitive types

type	bytes
char[]	$2N + 24$
int[]	$4N + 24$
double[]	$8N + 24$

for one-dimensional arrays

type	bytes
char[][]	$\sim 2 MN$
int[][]	$\sim 4 MN$
double[][]	$\sim 8 MN$

for two-dimensional arrays

Typical memory usage for objects in Java

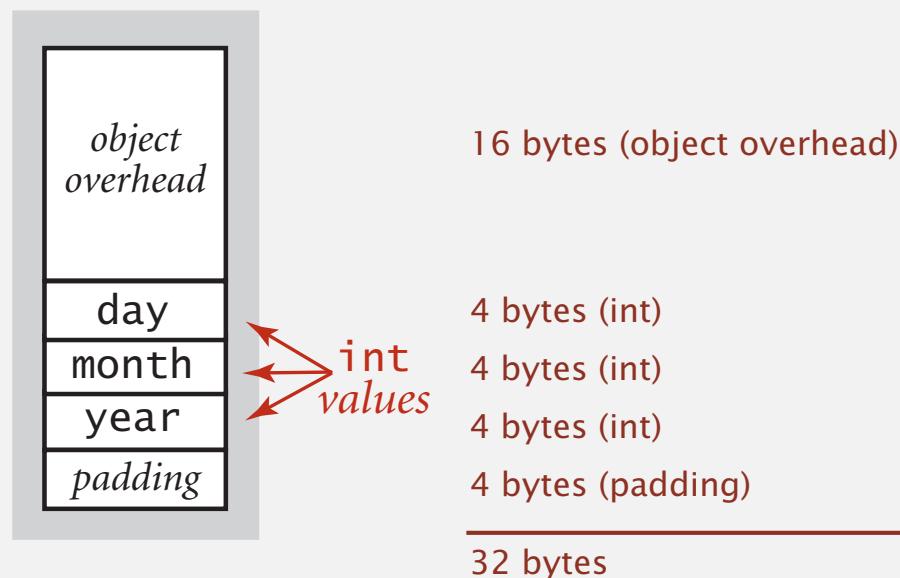
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Each object uses a multiple of 8 bytes.

Ex 1. A Date object uses 32 bytes of memory.

```
public class Date
{
    private int day;
    private int month;
    private int year;
    ...
}
```



Typical memory usage for objects in Java

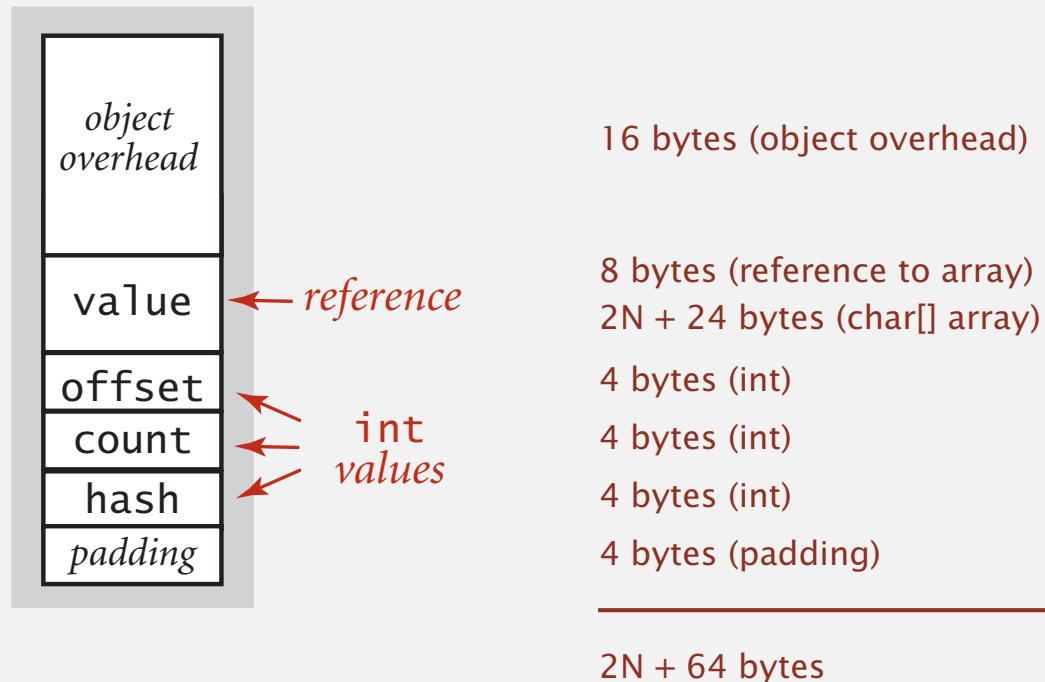
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Each object uses a multiple of 8 bytes.

Ex 2. A virgin String of length N uses $\sim 2N$ bytes of memory.

```
public class String
{
    private char[] value;
    private int offset;
    private int count;
    private int hash;
    ...
}
```



Typical memory usage summary

Total memory usage for a data type value:

- Primitive type: 4 bytes for int, 8 bytes for double, ...
- Object reference: 8 bytes.
- Array: 24 bytes + memory for each array entry.
- Object: 16 bytes + memory for each instance variable
+ 8 bytes if inner class (for pointer to enclosing class).
- Padding: round up to multiple of 8 bytes.

Shallow memory usage: Don't count referenced objects.

Deep memory usage: If array entry or instance variable is a reference,
add memory (recursively) for referenced object.

Example

- Q. How much memory does WeightedQuickUnionUF use as a function of N ?
Use tilde notation to simplify your answer.

```
public class WeightedQuickUnionUF
{
```

```
    private int[] id;
    private int[] sz;
    private int count;
```

```
    public WeightedQuickUnionUF(int N)
    {
```

```
        id = new int[N];
        sz = new int[N];
        for (int i = 0; i < N; i++) id[i] = i;
        for (int i = 0; i < N; i++) sz[i] = 1;
    }
    ...
}
```

16 bytes
(object overhead)

8 + (4N + 24) each
reference + int[] array

4 bytes (int)

4 bytes (padding)

- A. $8N + 88 \sim 8N$ bytes.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ ***memory***

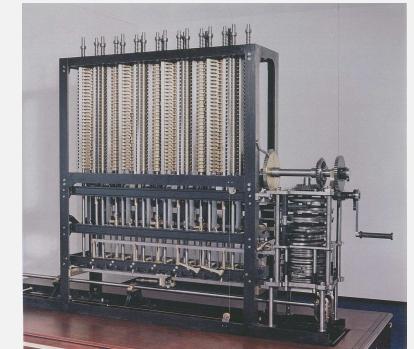
Turning the crank: summary

Empirical analysis.

- Execute program to perform experiments.
- Assume power law and formulate a hypothesis for running time.
- Model enables us to **make predictions**.

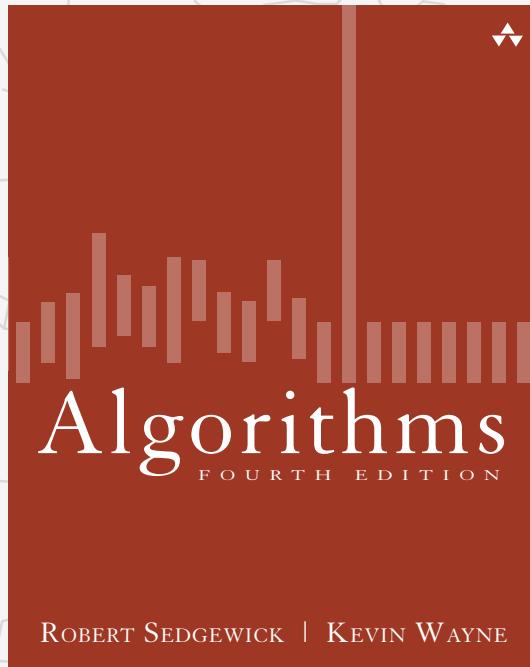
Mathematical analysis.

- Analyze algorithm to count frequency of operations.
- Use tilde notation to simplify analysis.
- Model enables us to **explain behavior**.



Scientific method.

- Mathematical model is independent of a particular system; applies to machines not yet built.
- Empirical analysis is necessary to validate mathematical models and to make predictions.



<http://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

- ▶ *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.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

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) ✓

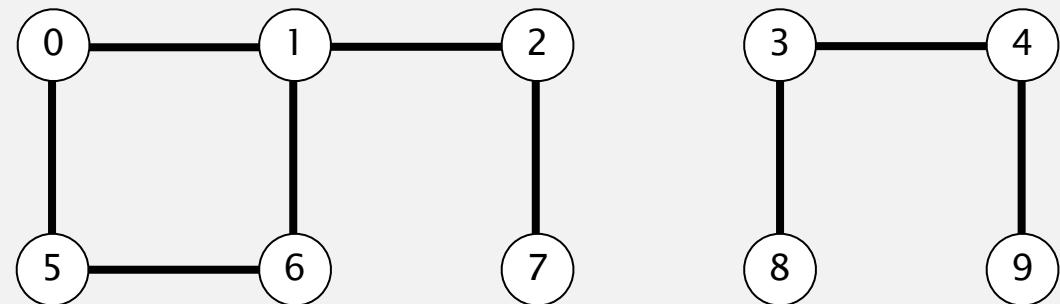
union(5, 0)

union(7, 2)

union(6, 1)

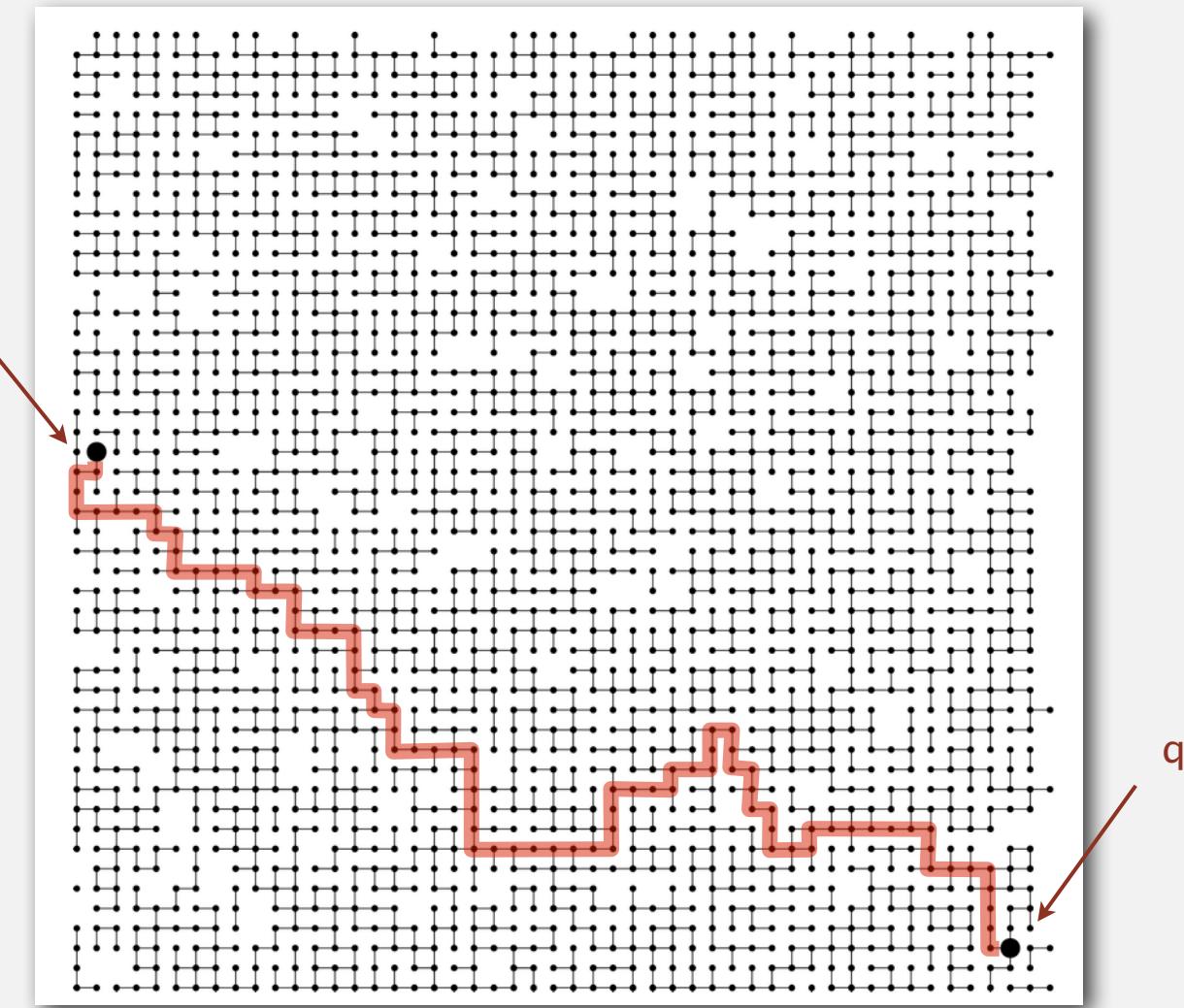
union(1, 0)

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.



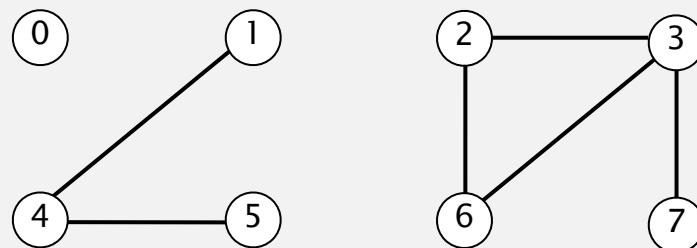
can use symbol table to translate from site
names to integers: stay tuned (Chapter 3)

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.



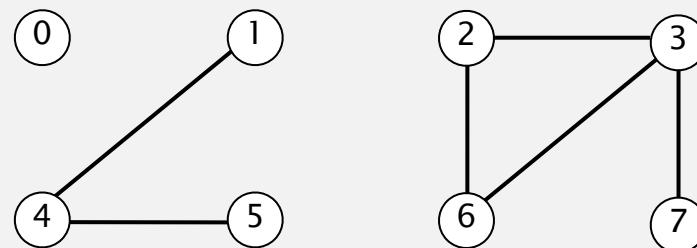
$\{ 0 \} \{ 1 4 5 \} \{ 2 3 6 7 \}$

3 connected components

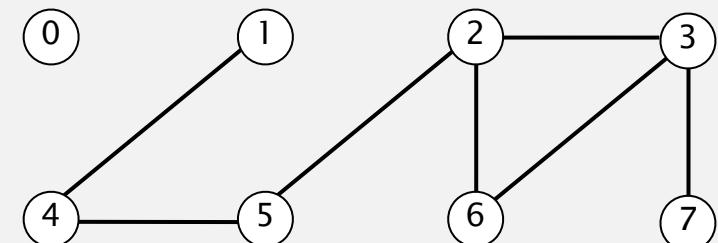
Implementing the operations

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

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



union(2, 5)



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

3 connected components

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

2 connected components

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
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Quick-find [eager approach]

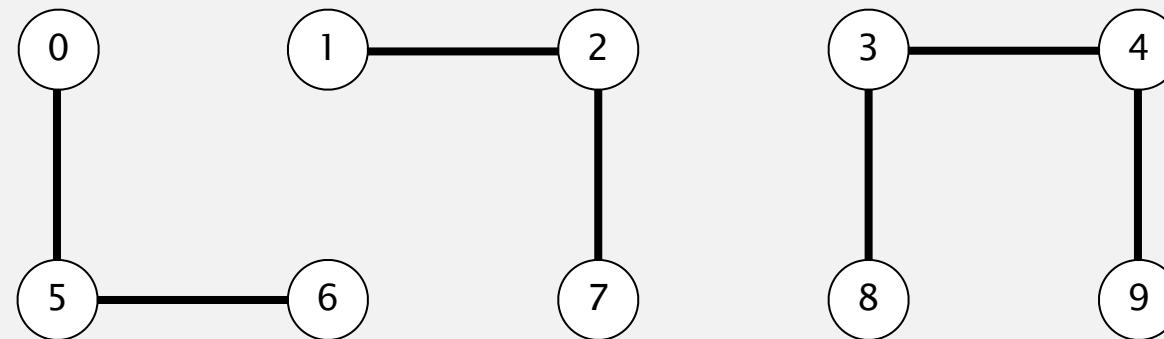
Data structure.

- 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
<code>id[]</code>	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 $\text{id}[]$ 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
$\text{id}[]$	0	1	1	8	8	0	0	1	8	8

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

$\text{id}[6] = 0; \text{id}[1] = 1$
6 and 1 are not connected

Union. To merge components containing p and q , change all entries whose id equals $\text{id}[p]$ to $\text{id}[q]$.

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	1	1	1	8	8	1	1	1	8	8
	↑				↑	↑				

problem: many values can change

after union of 6 and 1

Quick-find demo



0

1

2

3

4

5

6

7

8

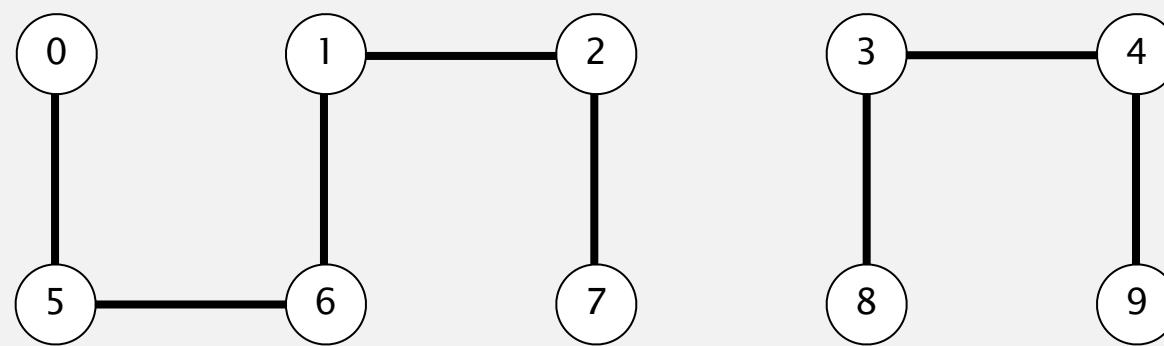
9

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

```
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 connected(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 $\text{id}[p]$ to $\text{id}[q]$
(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.

quadratic

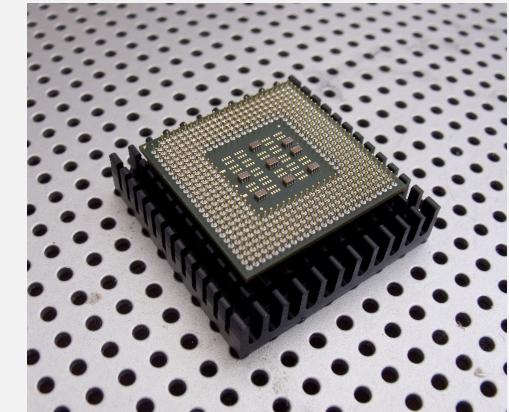
Ex. Takes N^2 array accesses to process sequence of N union commands on N objects.

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!

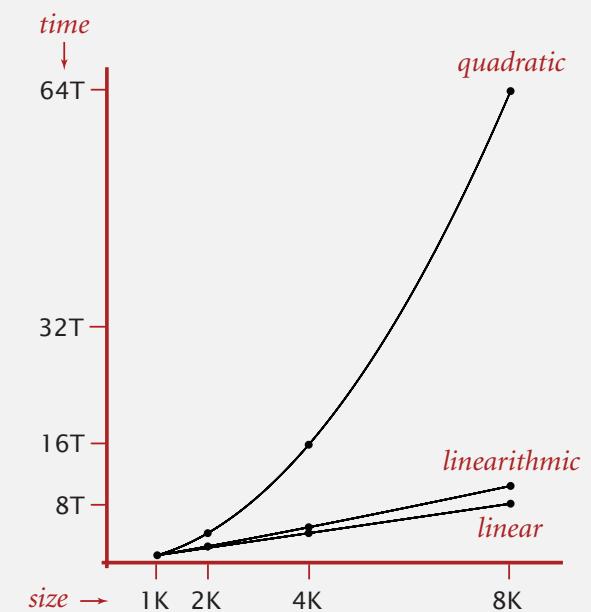


Ex. Huge problem for quick-find.

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

Quadratic algorithms don't scale with technology.

- New computer may be 10x as fast.
- But, has 10x as much memory ⇒ want to solve a problem that is 10x as big.
- With quadratic algorithm, takes 10x as long!



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

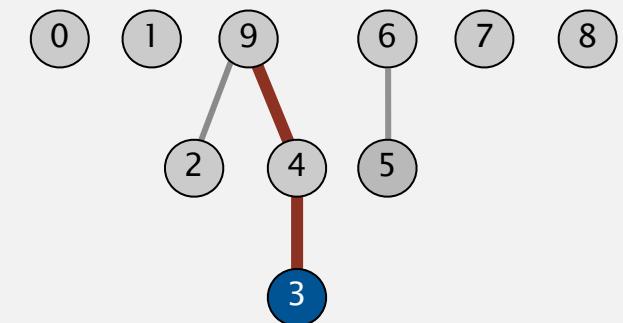
- ▶ *dynamic connectivity*
- ▶ *quick find*
- ▶ *quick union*
- ▶ *improvements*
- ▶ *applications*

Quick-union [lazy approach]

Data structure.

- Integer array $\text{id}[]$ of size N .
- Interpretation: $\text{id}[i]$ is parent of i . keep going until it doesn't change
(algorithm ensures no cycles)
- Root of i is $\text{id}[\text{id}[\text{id}[\dots\text{id}[i]\dots]]]$.

	0	1	2	3	4	5	6	7	8	9
$\text{id}[]$	0	1	9	4	9	6	6	7	8	9



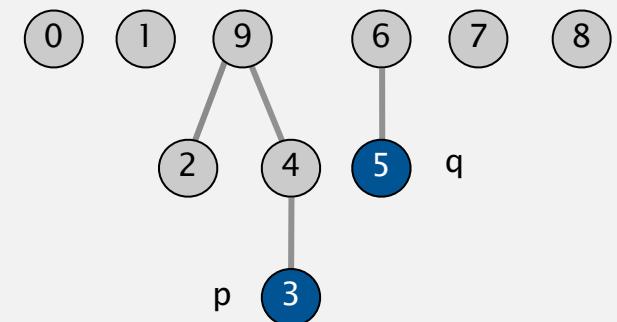
root of 3 is 9

Quick-union [lazy approach]

Data structure.

- Integer array $\text{id}[]$ of size N .
- Interpretation: $\text{id}[i]$ is parent of i .
- Root of i is $\text{id}[\text{id}[\text{id}[\dots\text{id}[i]\dots]]]$.

0	1	2	3	4	5	6	7	8	9	
$\text{id}[]$	0	1	9	4	9	6	6	7	8	9



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

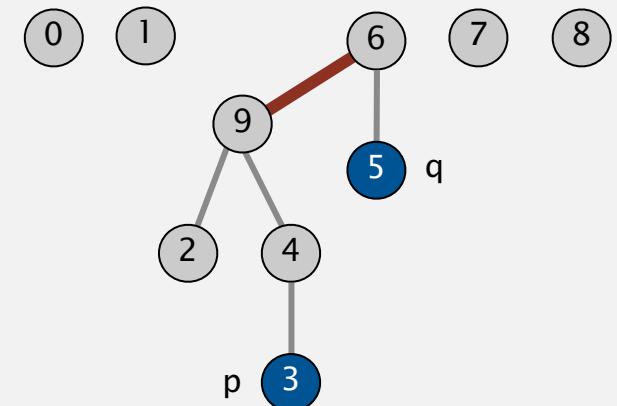
root of 3 is 9
root of 5 is 6

3 and 5 are not connected

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	
$\text{id}[]$	0	1	9	4	9	6	6	7	8	6

↑
only one value changes



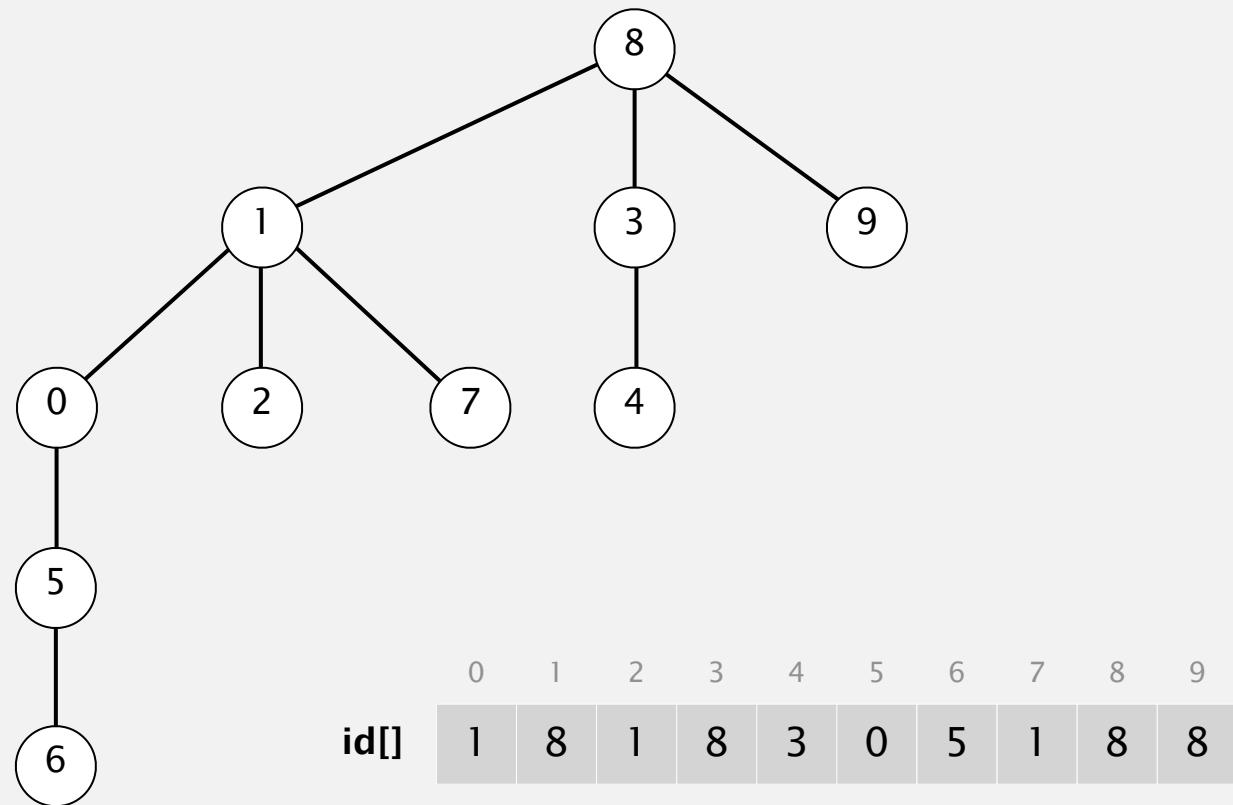
Quick-union demo



0 1 2 3 4 5 6 7 8 9

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

Quick-union demo



Quick-union: Java implementation

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

    public QuickUnionUF(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 connected(int p, int q)
    {
        return root(p) == root(q);
    }

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

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

† 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).

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

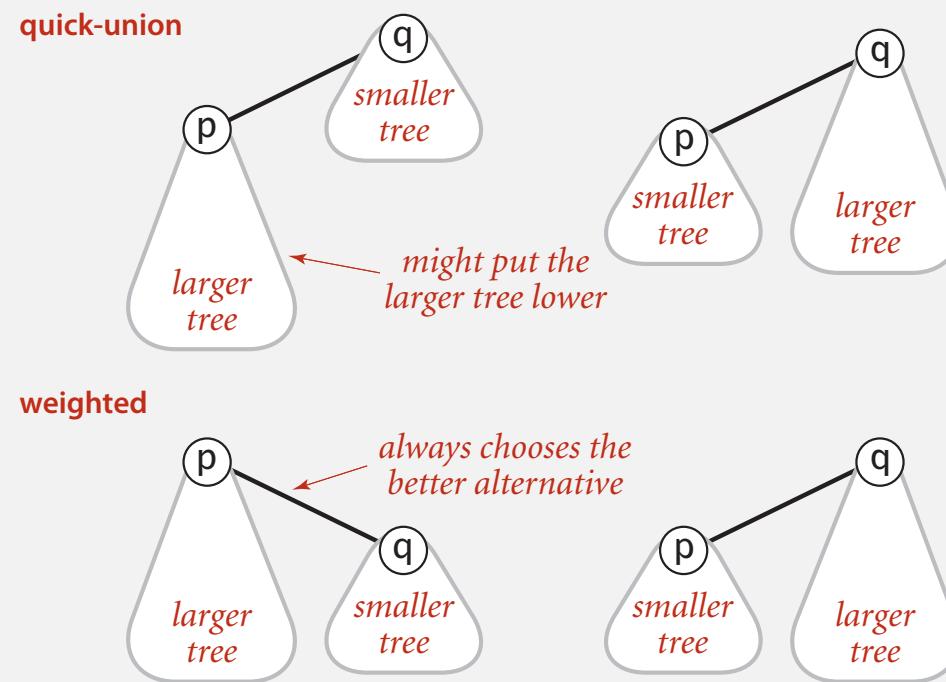
1.5 UNION-FIND

- ▶ *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 tree (number of objects).
- Balance by linking root of smaller tree to root of larger tree.



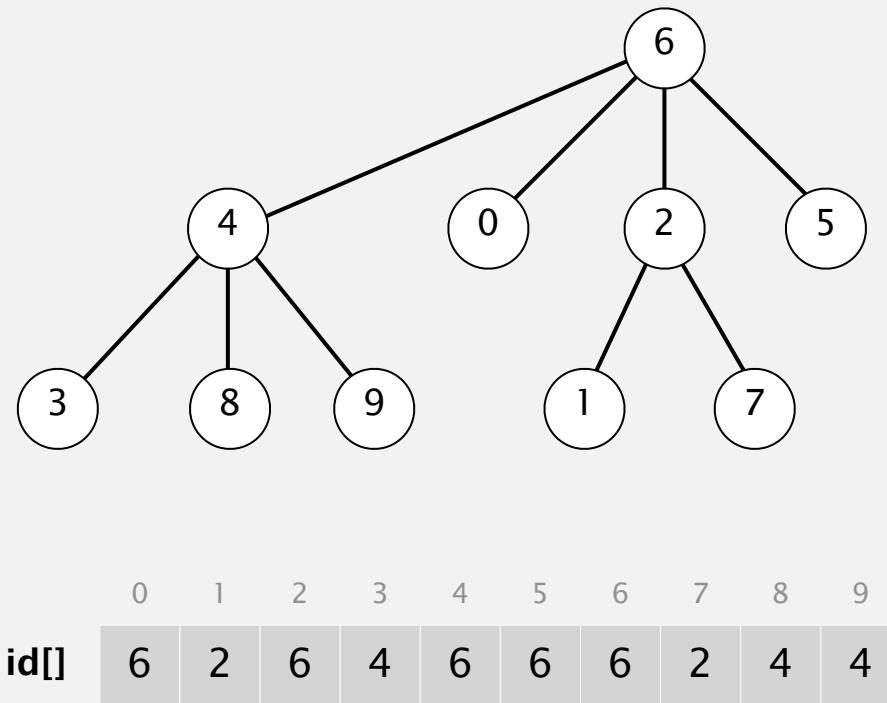
Weighted quick-union demo



0 1 2 3 4 5 6 7 8 9

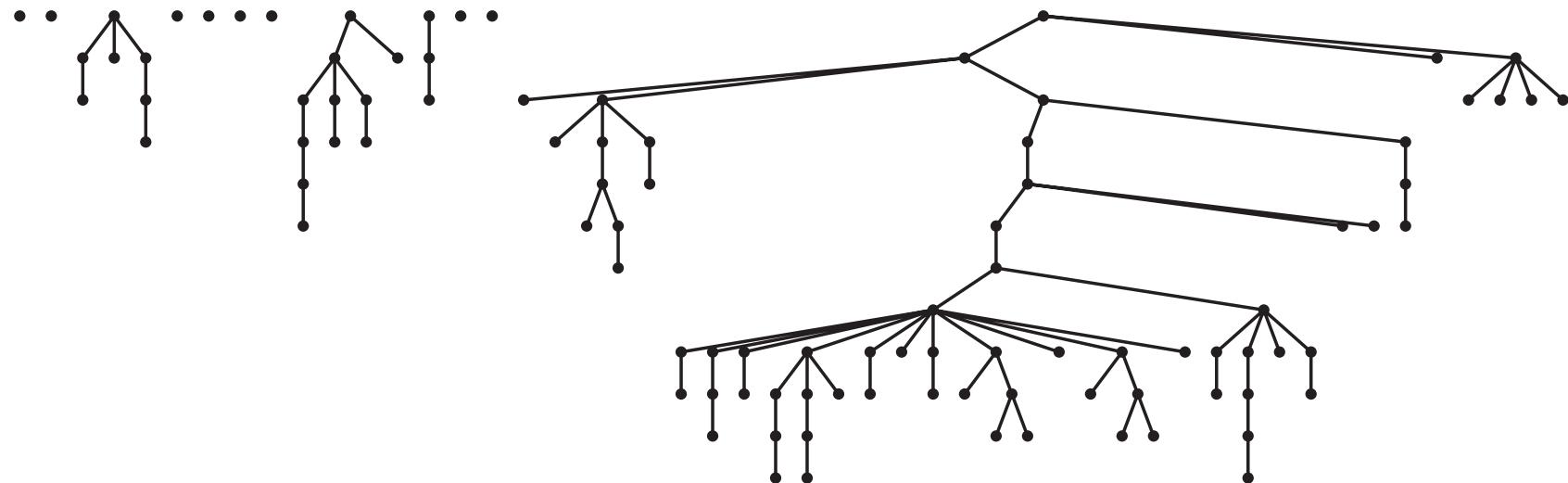
	0	1	2	3	4	5	6	7	8	9
id[]	0	1	2	3	4	5	6	7	8	9

Weighted quick-union demo



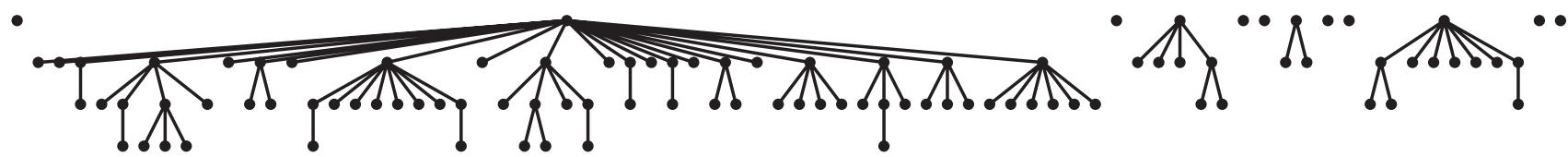
Quick-union and weighted quick-union example

quick-union



average distance to root: 5.11

weighted



average distance to root: 1.52

Quick-union and weighted quick-union (100 sites, 88 union() operations)

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:

- Link root of smaller tree to root of larger tree.
- Update the $sz[]$ array.

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

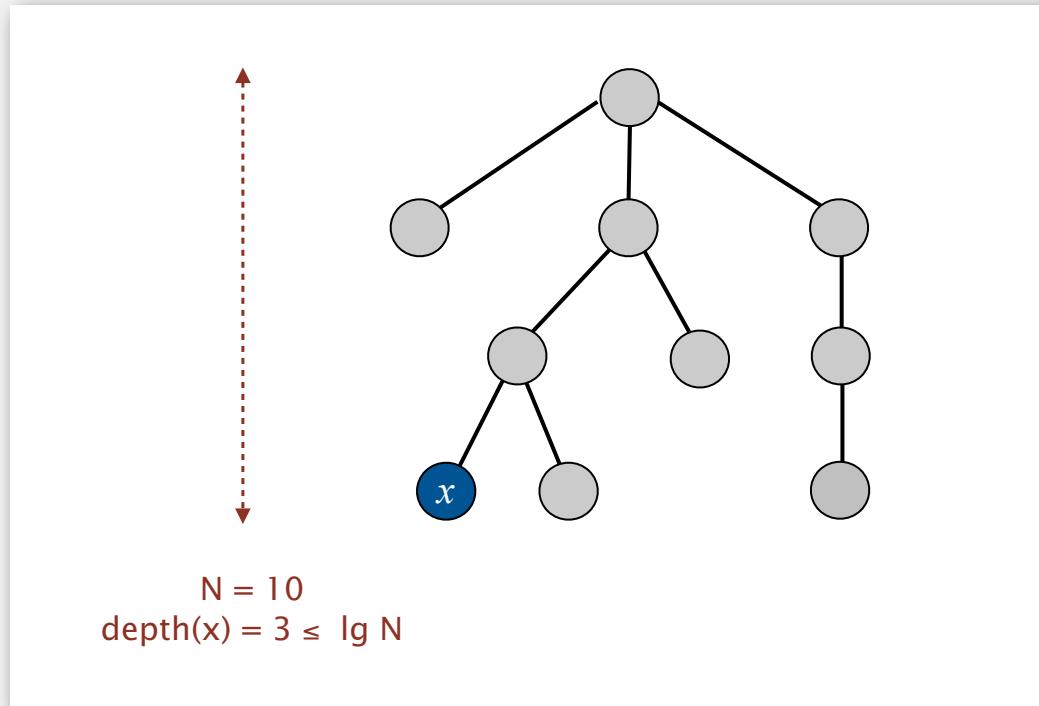
Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q .
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

\lg = base-2 logarithm



Weighted quick-union analysis

Running time.

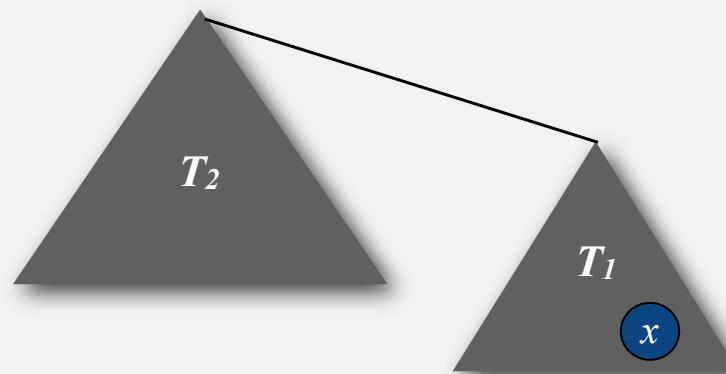
- Find: takes time proportional to depth of p and q .
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

Pf. When does depth of x increase?

Increases by 1 when tree T_1 containing x is merged into another tree T_2 .

- The size of the tree containing x at least doubles since $|T_2| \geq |T_1|$.
- Size of tree containing x can double at most $\lg N$ times. Why?



Weighted quick-union analysis

Running time.

- Find: takes time proportional to depth of p and q .
- Union: takes constant time, given roots.

Proposition. Depth of any node x is at most $\lg N$.

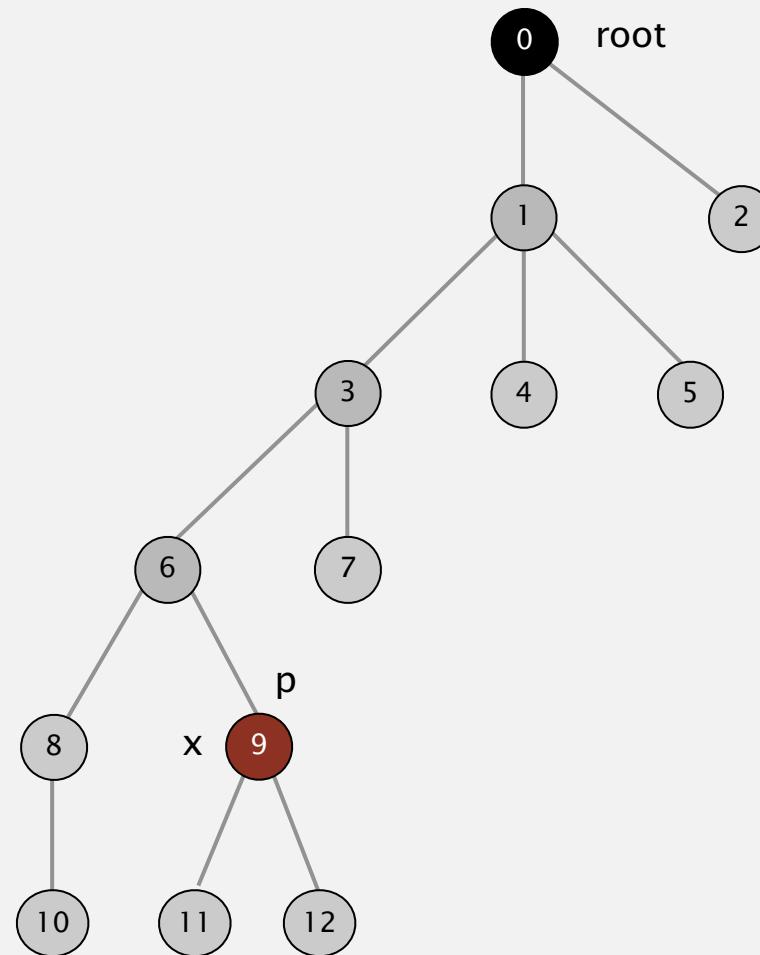
algorithm	initialize	union	connected
quick-find	N	N	1
quick-union	N	N^{\dagger}	N
weighted QU	N	$\lg N^{\dagger}$	$\lg N$

\dagger includes cost of finding roots

- Q. Stop at guaranteed acceptable performance?
A. No, easy to improve further.

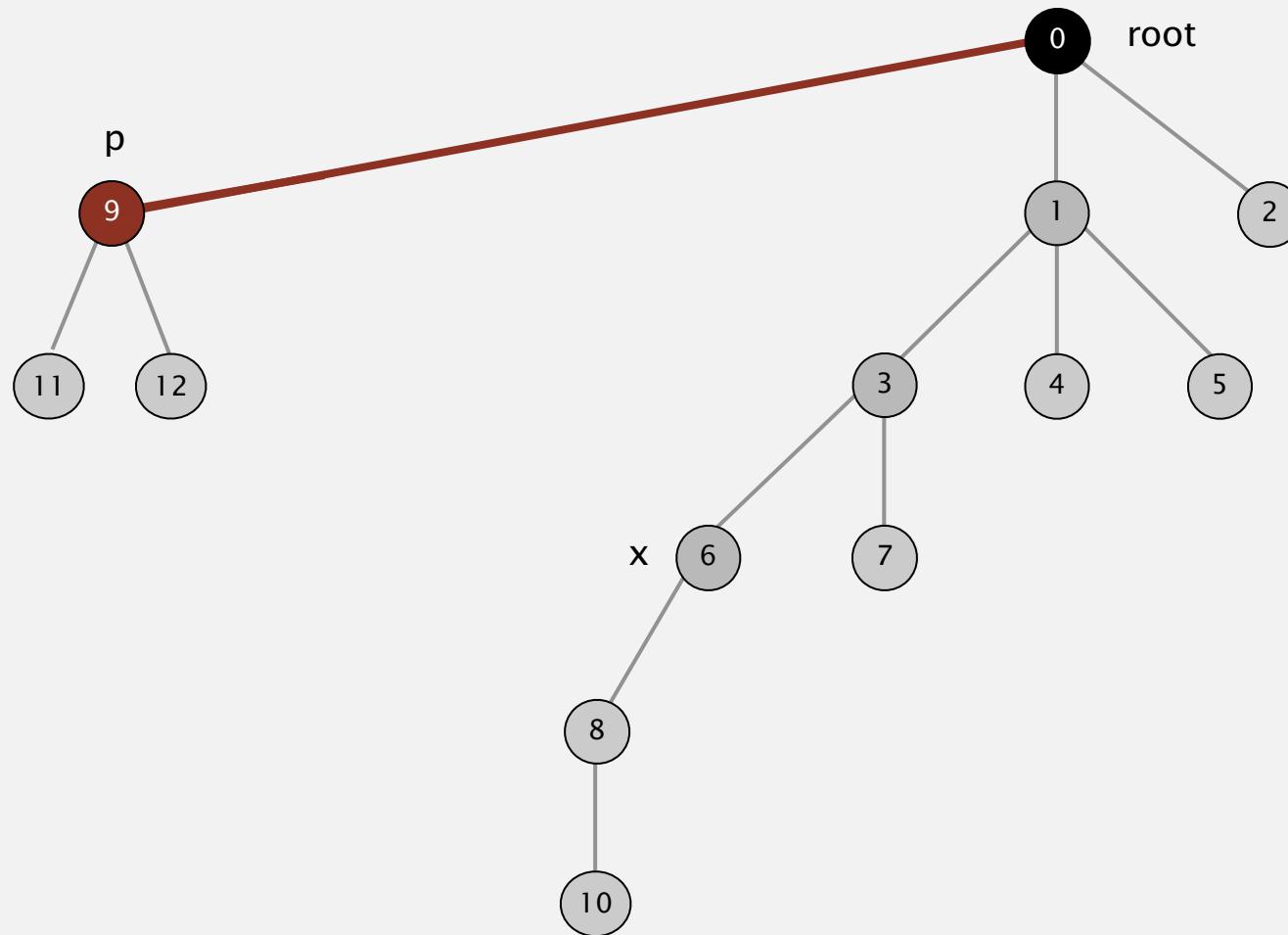
Improvement 2: path compression

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



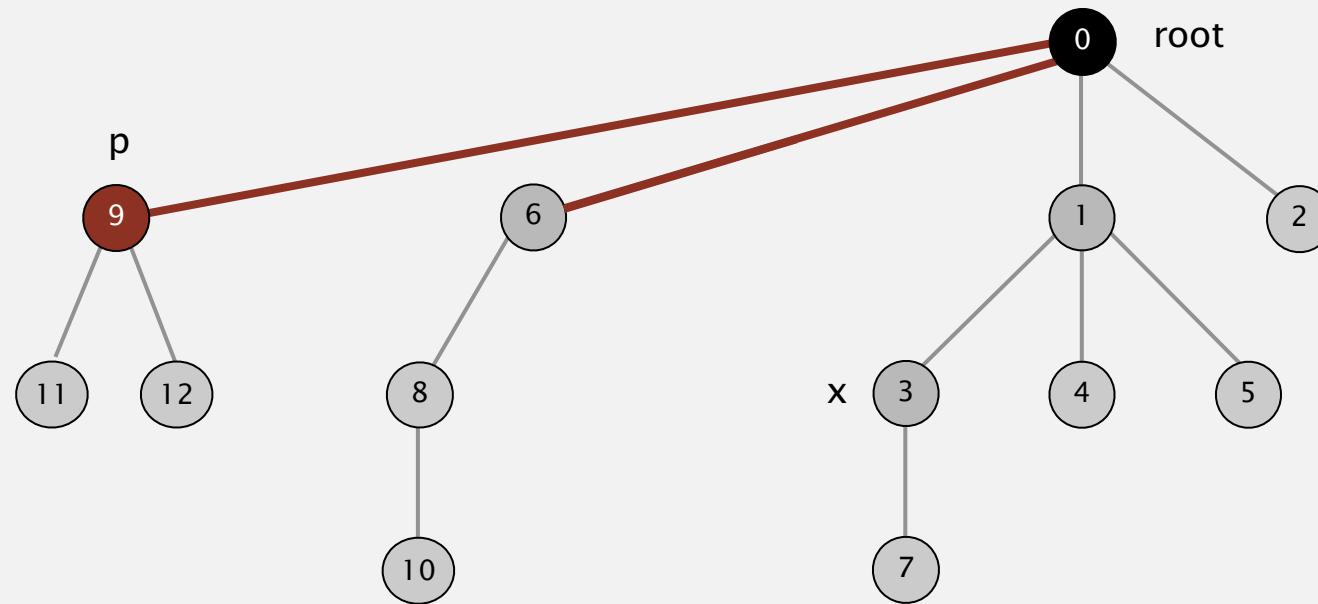
Improvement 2: path compression

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



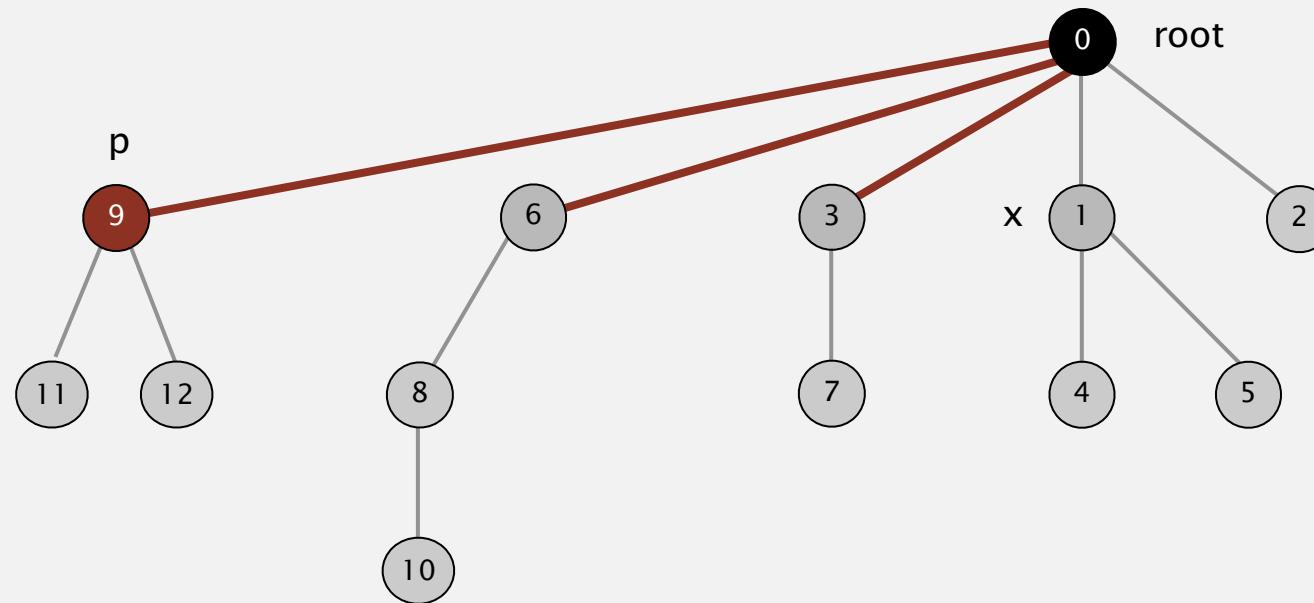
Improvement 2: path compression

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



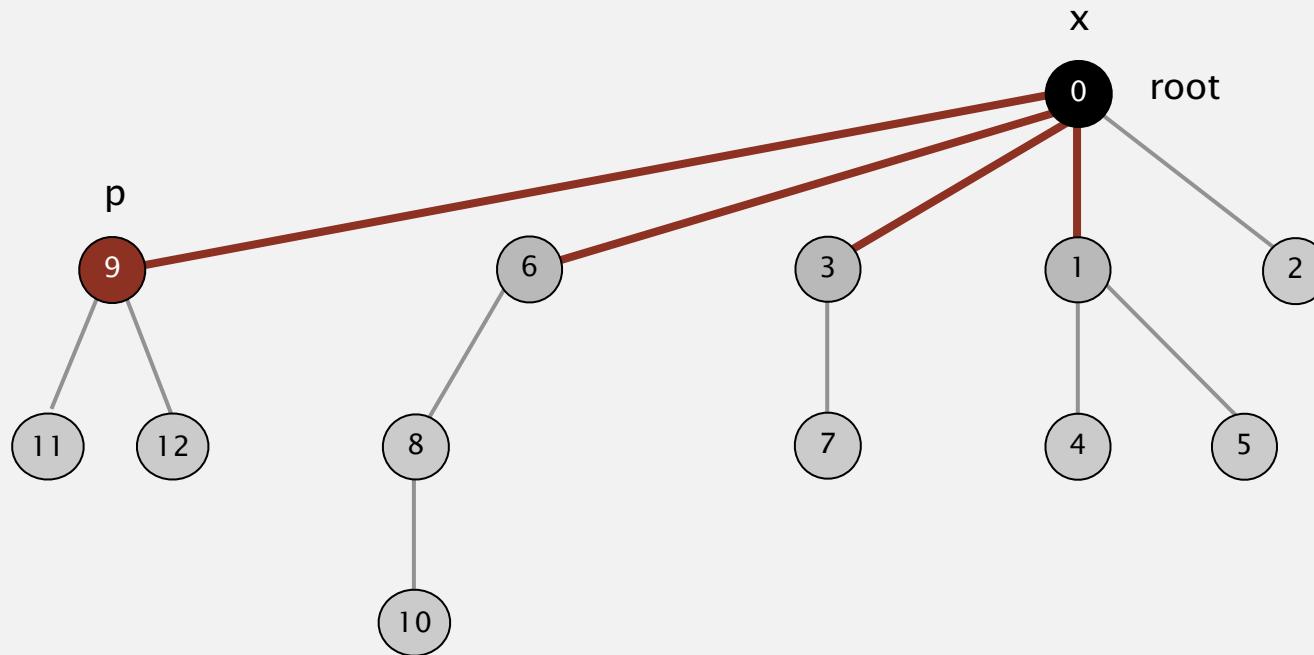
Improvement 2: path compression

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



Improvement 2: path compression

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



Path compression: Java implementation

Two-pass 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 (thereby halving path length).

```
private int root(int i)
{
    while (i != id[i])
    {
        id[i] = id[id[i]]; ← only one extra line of code !
        i = id[i];
    }
    return i;
}
```

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

Weighted quick-union with path compression: amortized analysis

Proposition. [Hopcroft-Ulman, Tarjan] Starting from an empty data structure, any sequence of M union–find ops on N objects makes $\leq c(N + M \lg^* N)$ array accesses.

- Analysis can be improved to $N + M \alpha(M, N)$.
- Simple algorithm with fascinating mathematics.

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

iterate log function

Linear-time algorithm for M union–find ops on N objects?

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

Amazing fact. [Fredman-Saks] No linear-time algorithm exists.

in "cell-probe" model of computation

Summary

Bottom line. Weighted quick union (with path compression) 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	$N + M \lg^* N$

M union-find operations on a set of N objects

Ex. [10⁹ unions and finds with 10⁹ objects]

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

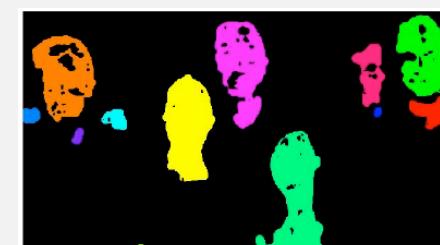
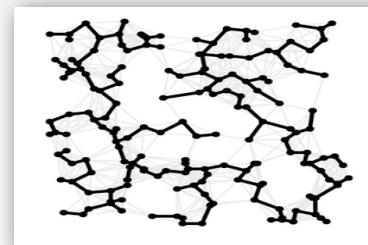
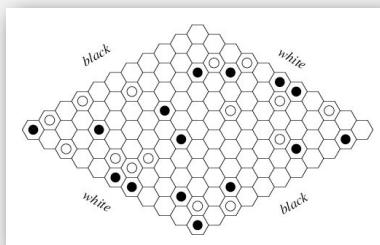
<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Union-find applications

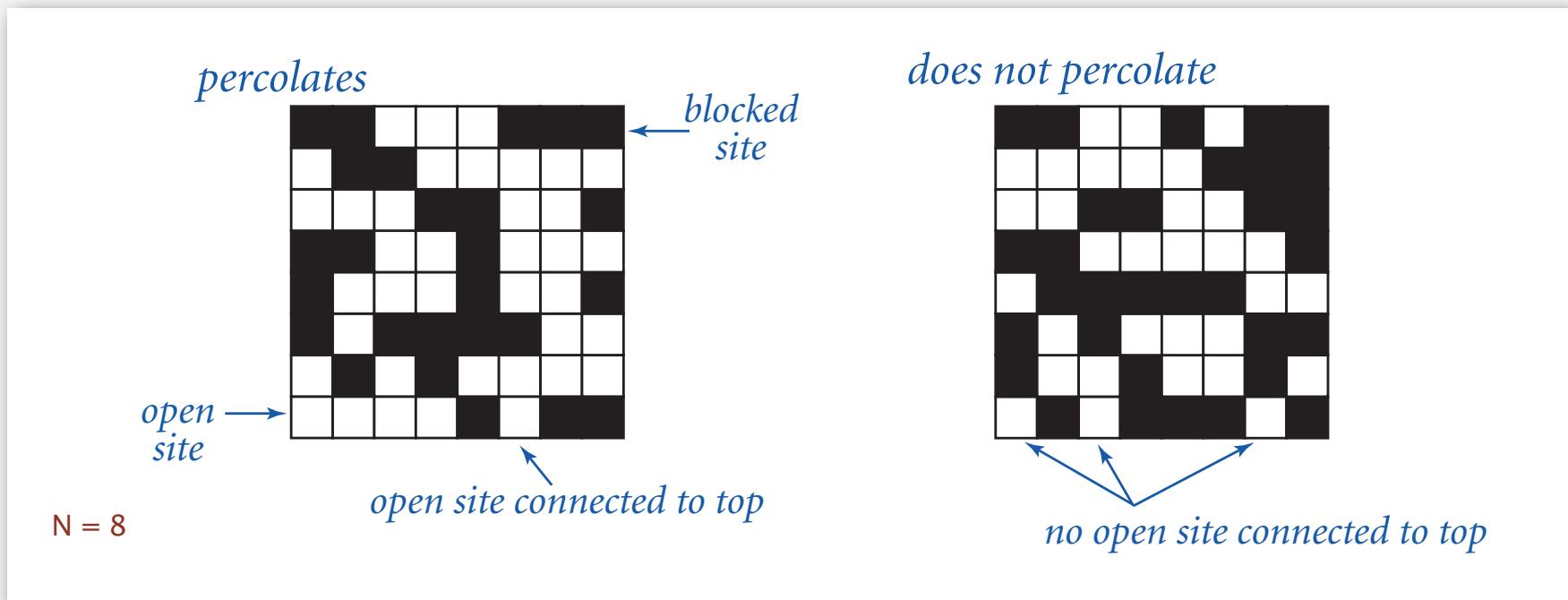
- Percolation.
- Games (Go, Hex).
- ✓ Dynamic connectivity.
 - Least common ancestor.
 - Equivalence of finite state automata.
 - Hoshen-Kopelman algorithm in physics.
 - Hinley-Milner polymorphic type inference.
 - Kruskal's minimum spanning tree algorithm.
 - Compiling equivalence statements in Fortran.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



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** iff top and bottom are connected by open sites.



Percolation

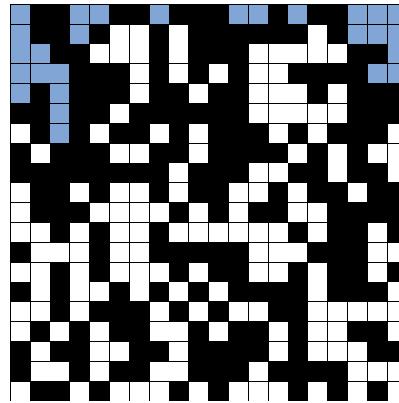
A model for many physical systems:

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

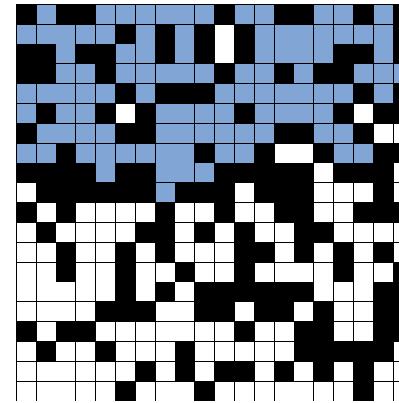
model	system	vacant site	occupied site	percolates
electricity	material	conductor	insulated	conducts
fluid flow	material	empty	blocked	porous
social interaction	population	person	empty	communicates

Likelihood of percolation

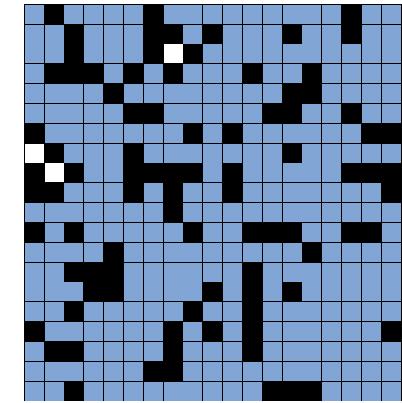
Depends on site vacancy probability p .



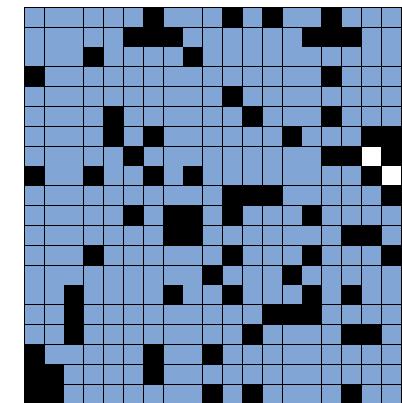
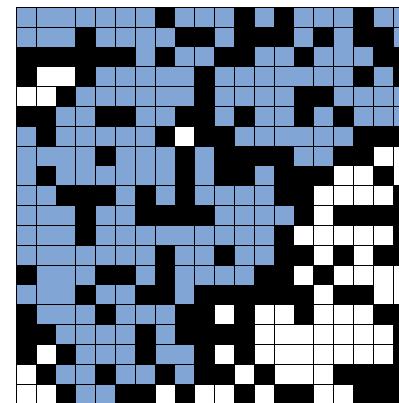
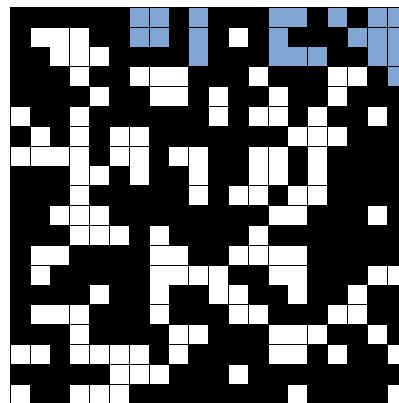
p low (0.4)
does not percolate



p medium (0.6)
percolates?



p high (0.8)
percolates

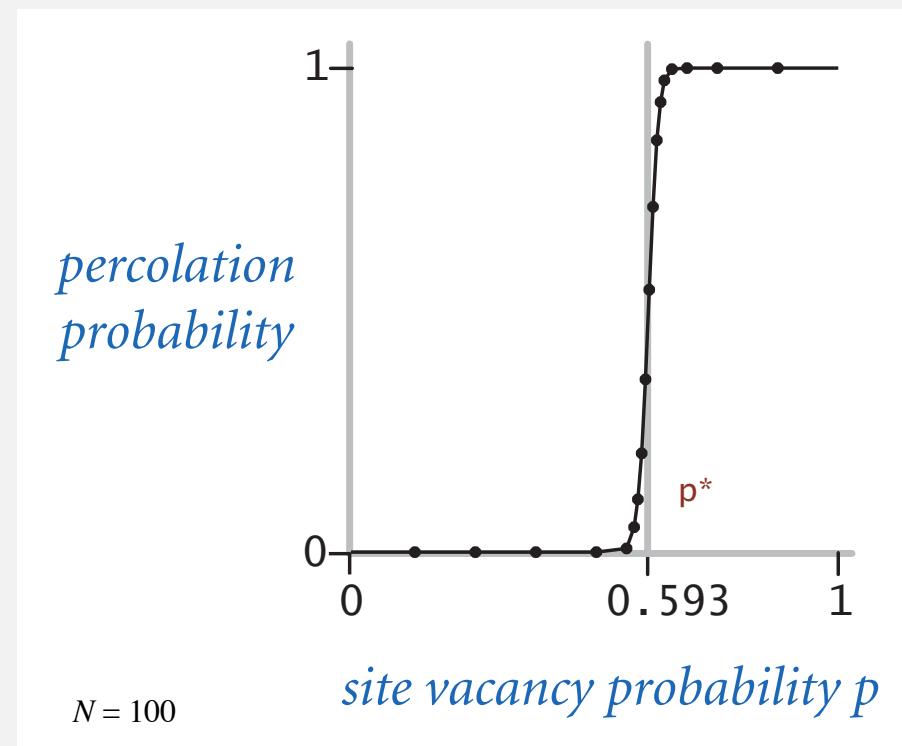


Percolation phase transition

When N is large, theory guarantees a sharp threshold p^* .

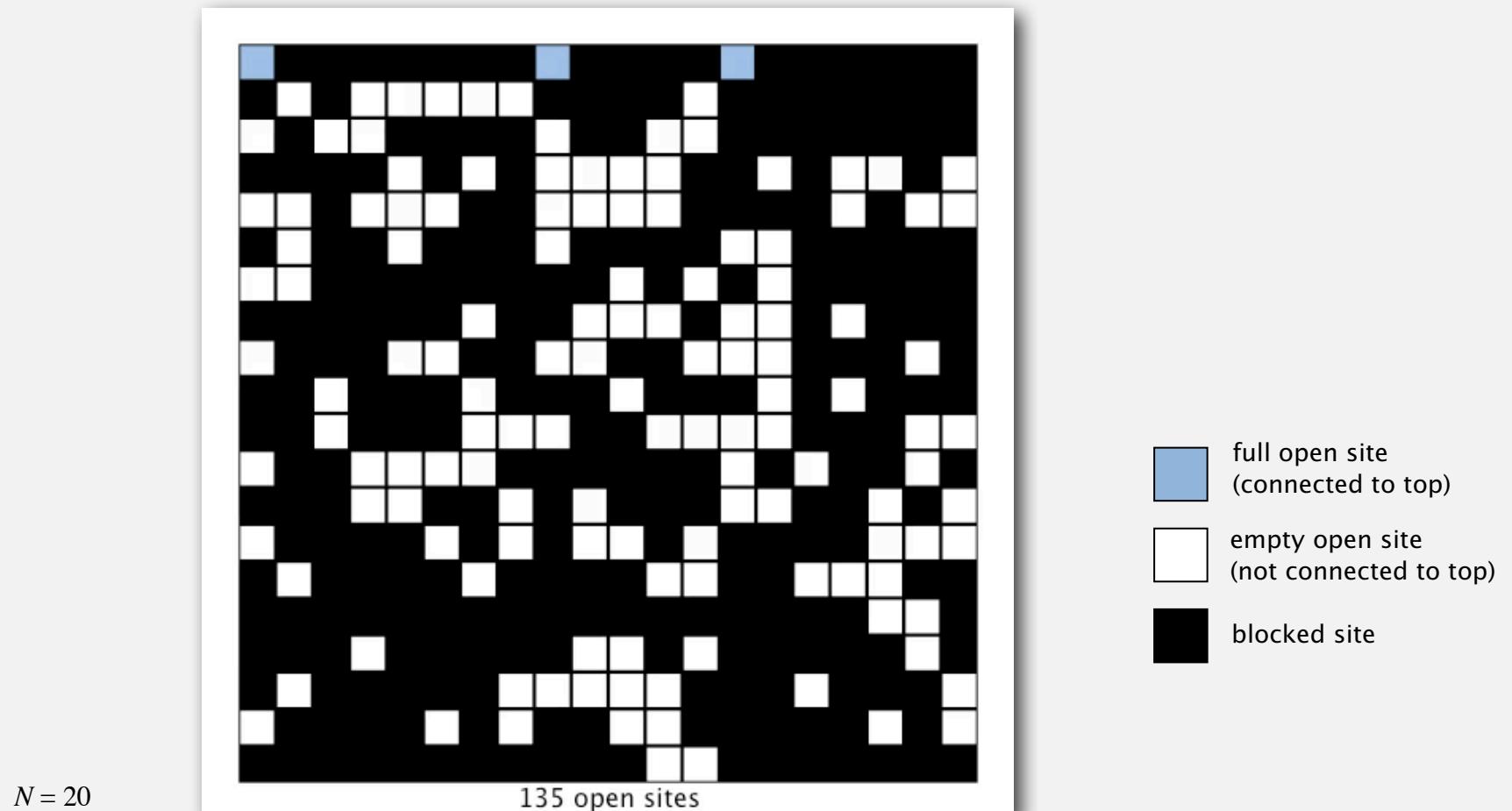
- $p > p^*$: almost certainly percolates.
- $p < p^*$: almost certainly does not percolate.

Q. What is the value of p^* ?



Monte Carlo simulation

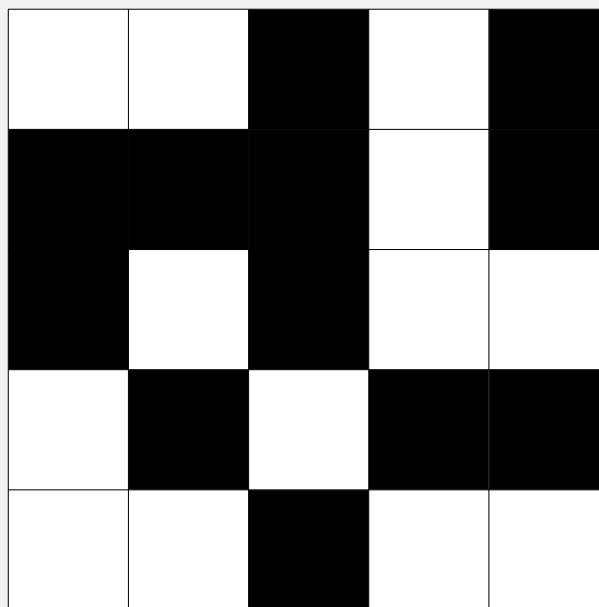
- Initialize N -by- N whole grid to be blocked.
- Declare random sites open until top connected to bottom.
- Vacancy percentage estimates p^* .



Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N -by- N system percolates?

$N = 5$



open site

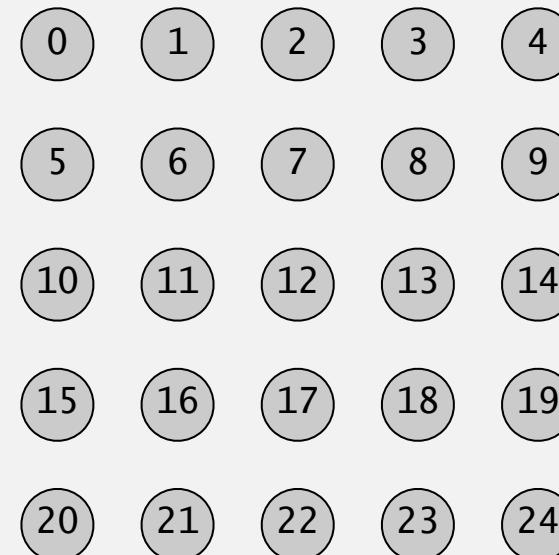
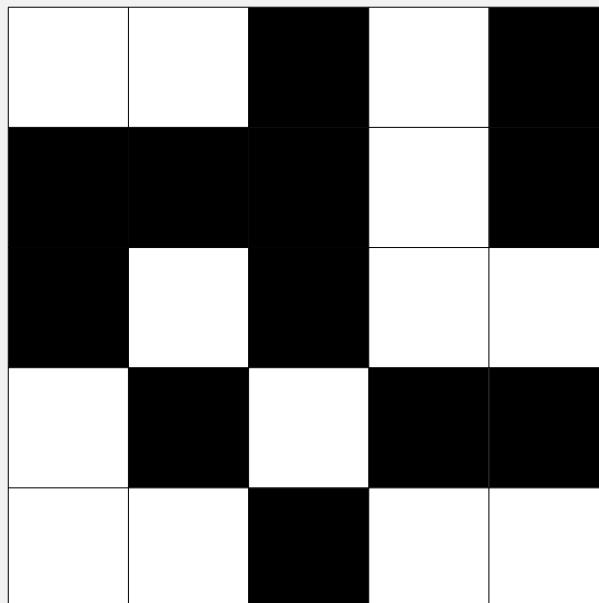
blocked site

Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N -by- N system percolates?

- Create an object for each site and name them 0 to $N^2 - 1$.

$N = 5$



open site

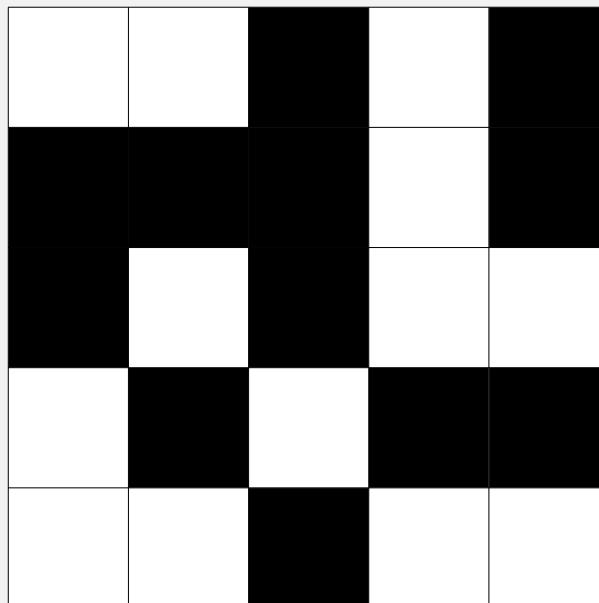
blocked site

Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N -by- N system percolates?

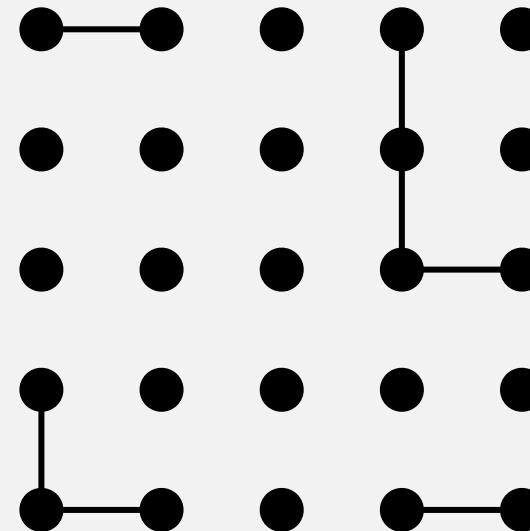
- Create an object for each site and name them 0 to $N^2 - 1$.
- Sites are in same component if connected by open sites.

$N = 5$



open site

blocked site

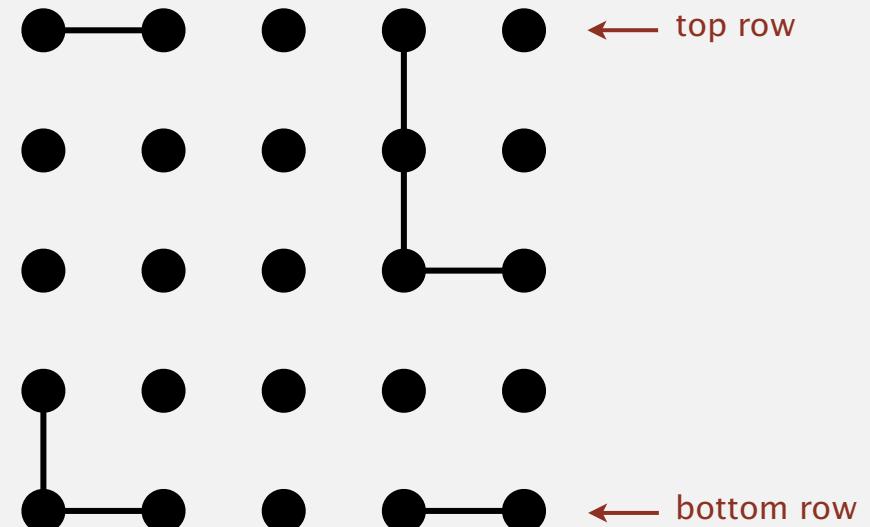
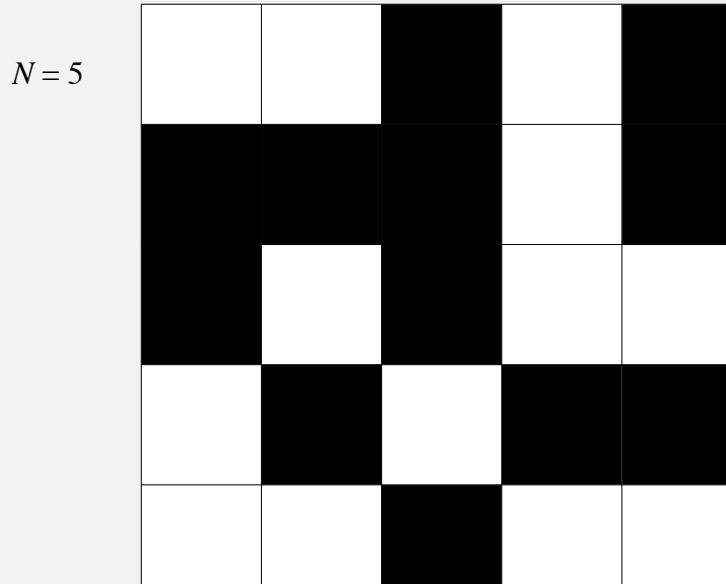


Dynamic connectivity solution to estimate percolation threshold

Q. How to check whether an N -by- N system percolates?

- Create an object for each site and name them 0 to $N^2 - 1$.
- Sites are in same component if connected by open sites.
- Percolates iff any site on bottom row is connected to site on top row.

brute-force algorithm: N^2 calls to connected()



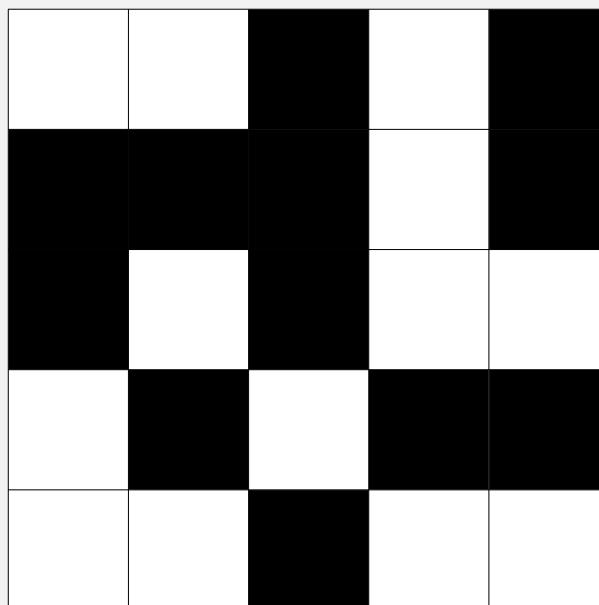
Dynamic connectivity solution to estimate percolation threshold

Clever trick. Introduce 2 virtual sites (and connections to top and bottom).

- Percolates iff virtual top site is connected to virtual bottom site.

efficient algorithm: only 1 call to connected()

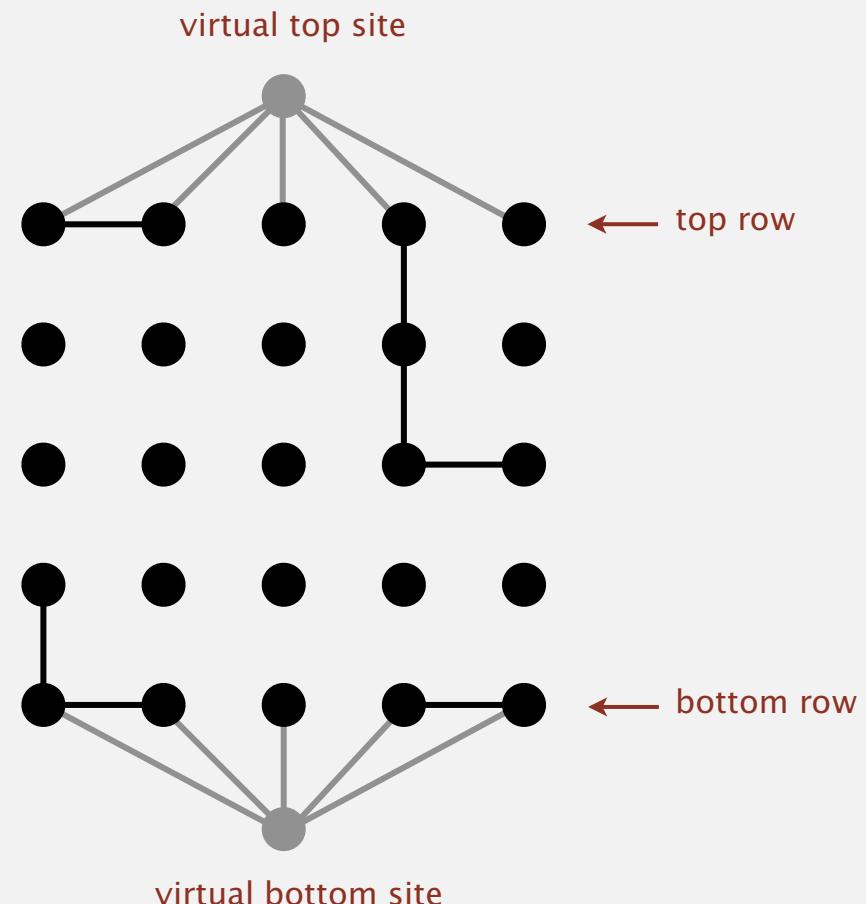
$N = 5$



open site

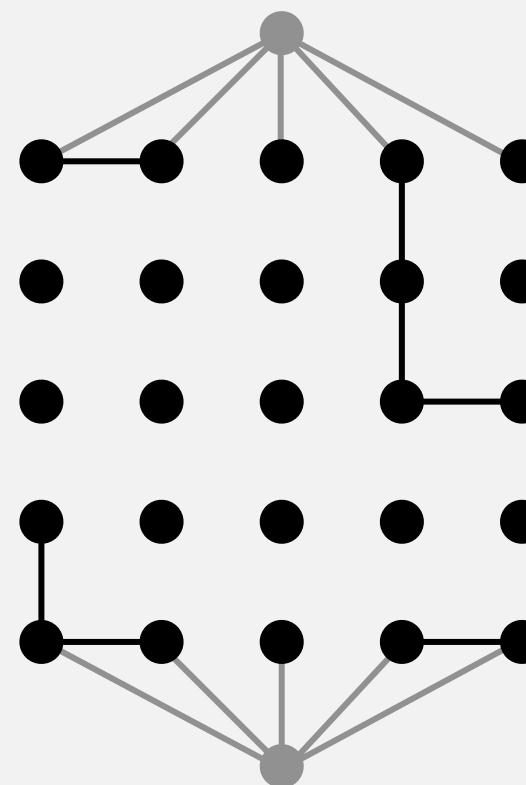
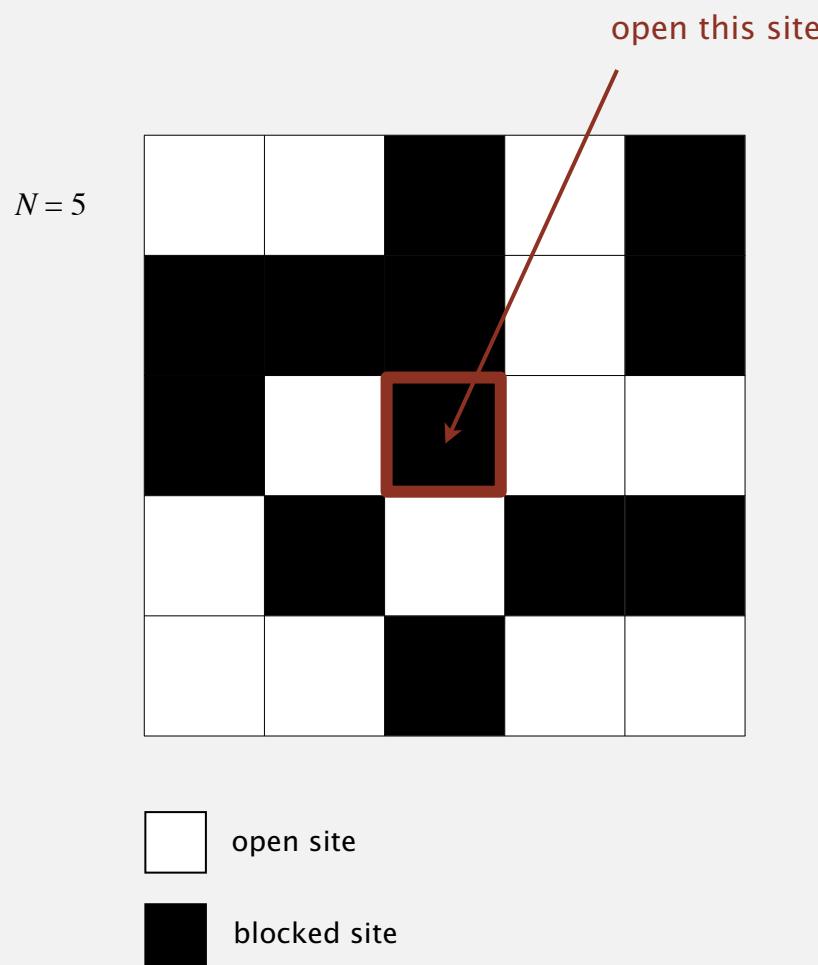


blocked site



Dynamic connectivity solution to estimate percolation threshold

Q. How to model opening a new site?



Dynamic connectivity solution to estimate percolation threshold

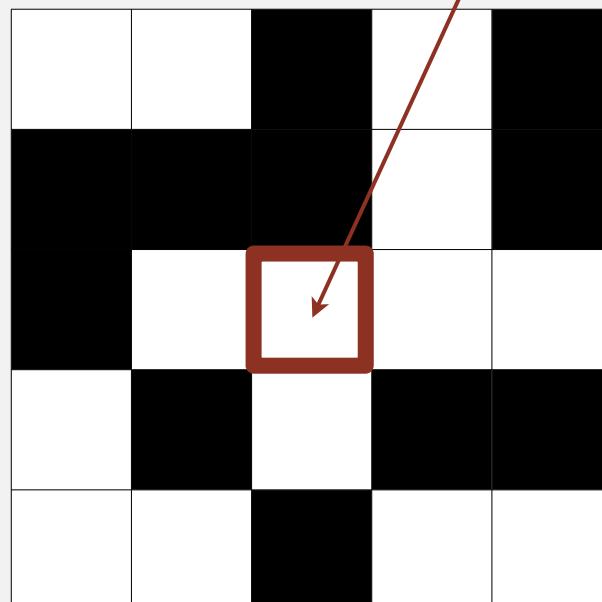
Q. How to model opening a new site?

A. Connect newly opened site to all of its adjacent open sites.

up to 4 calls to union()

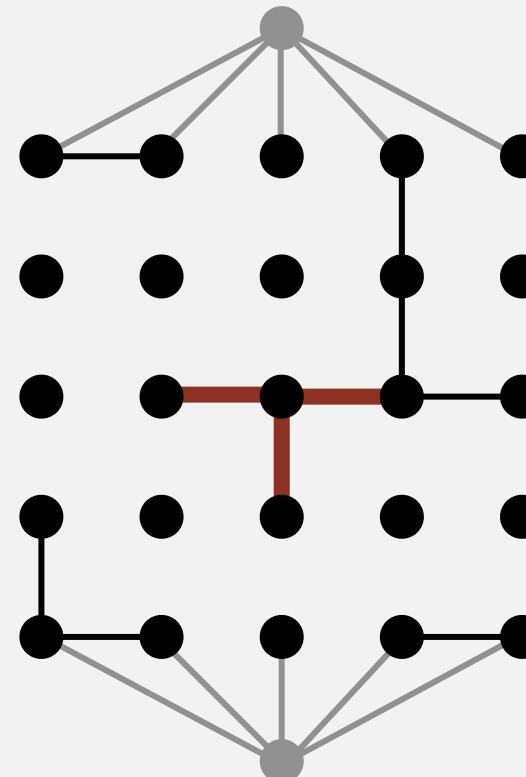
$N = 5$

open this site



open site

blocked site

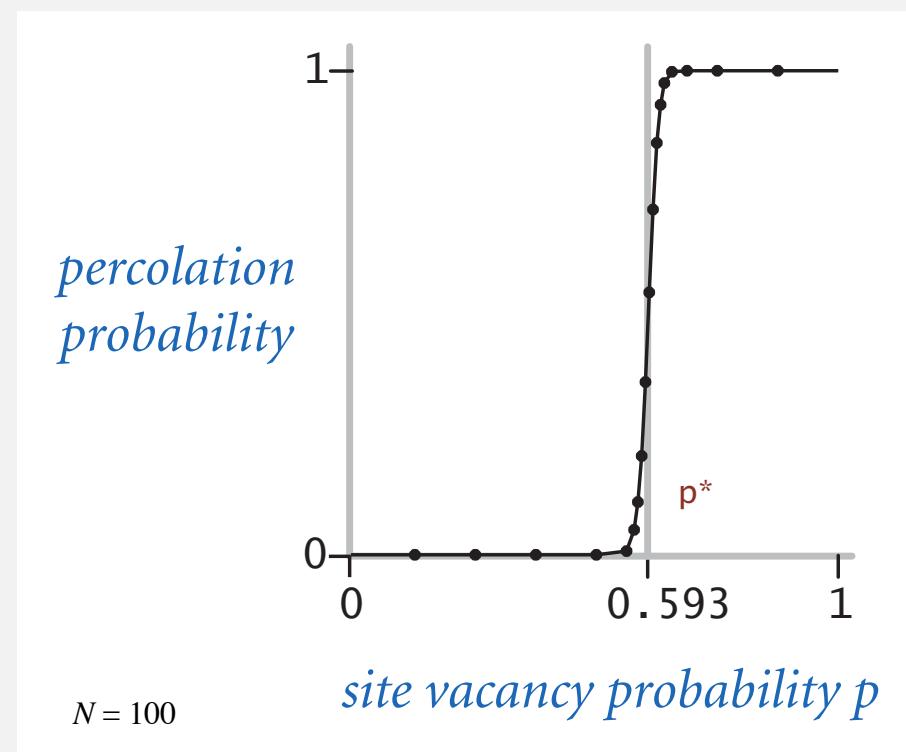


Percolation threshold

Q. What is percolation threshold p^* ?

A. About 0.592746 for large square lattices.

constant known only via simulation



Fast algorithm enables accurate answer to scientific question.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

- ▶ *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.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



ROBERT SEDGEWICK | KEVIN WAYNE

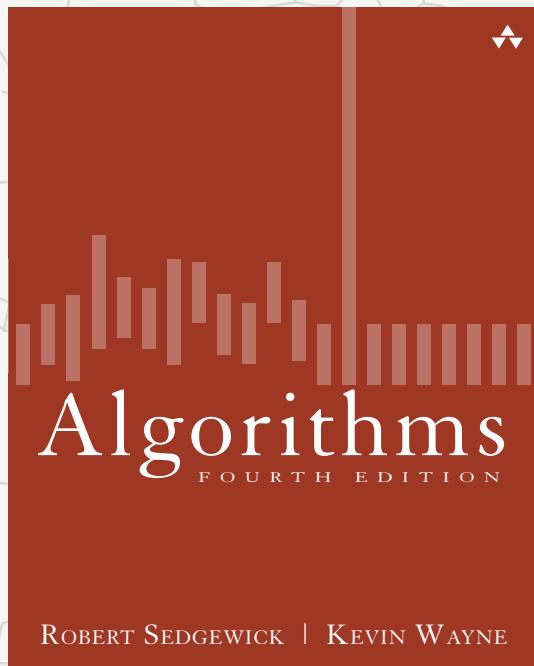
<http://algs4.cs.princeton.edu>

1.5 UNION-FIND

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Sorting problem

Ex. Student records in a university.

item →	Chen	3	A	991-878-4944	308 Blair
	Rohde	2	A	232-343-5555	343 Forbes
	Gazsi	4	B	766-093-9873	101 Brown
	Furia	1	A	766-093-9873	101 Brown
key →	Kanaga	3	B	898-122-9643	22 Brown
	Andrews	3	A	664-480-0023	097 Little
	Battle	4	C	874-088-1212	121 Whitman

Sort. Rearrange array of N items into ascending order.

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

Sample sort client 1

Goal. Sort **any** type of data.

Ex 1. Sort random real numbers in ascending order.

seems artificial, but stay tuned for an application

```
public class Experiment
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Double[] a = new Double[N];
        for (int i = 0; i < N; i++)
            a[i] = StdRandom.uniform();
        Insertion.sort(a);
        for (int i = 0; i < N; i++)
            StdOut.println(a[i]);
    }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Sample sort client 2

Goal. Sort **any** type of data.

Ex 2. Sort strings from file in alphabetical order.

```
public class StringSorter
{
    public static void main(String[] args)
    {
        String[] a = In.readStrings(args[0]);
        Insertion.sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}
```

```
% more words3.txt
bed bug dad yet zoo ... all bad yes
```

```
% java StringSorter words3.txt
all bad bed bug dad ... yes yet zoo
```

Sample sort client 3

Goal. Sort **any** type of data.

Ex 3. Sort the files in a given directory by filename.

```
import java.io.File;
public class FileSorter
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            StdOut.println(files[i].getName());
    }
}
```

```
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```

Callbacks

Goal. Sort **any** type of data.

Q. How can sort() know how to compare data of type Double, String, and java.io.File without any information about the type of an item's key?

Callback = reference to executable code.

- Client passes array of objects to sort() function.
- The sort() function calls back object's compareTo() method as needed.

Implementing callbacks.

- Java: **interfaces**.
- C: function pointers.
- C++: class-type functors.
- C#: delegates.
- Python, Perl, ML, Javascript: first-class functions.

Callbacks: roadmap

client

```
import java.io.File;
public class FileSorter
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            StdOut.println(files[i].getName());
    }
}
```

object implementation

```
public class File
    implements Comparable<File>
{
    ...
    public int compareTo(File b)
    {
        ...
        return -1;
        ...
        return +1;
        ...
        return 0;
    }
}
```

Comparable interface (built in to Java)

```
public interface Comparable<Item>
{
    public int compareTo(Item that);
}
```

key point: no dependence
on File data type

sort implementation

```
public static void sort(Comparable[] a)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
            else break;
}
```

Total order

A **total order** is a binary relation \leq that satisfies

- Antisymmetry: if $v \leq w$ and $w \leq v$, then $v = w$.
- Transitivity: if $v \leq w$ and $w \leq x$, then $v \leq x$.
- Totality: either $v \leq w$ or $w \leq v$ or both.

Ex.

- Standard order for natural and real numbers.
- Alphabetical order for strings.
- Chronological order for dates or times.
- ...

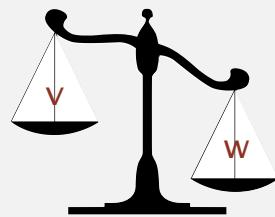


an intransitive relation

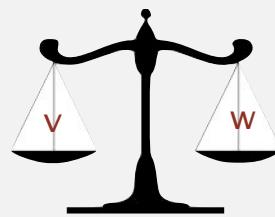
Comparable API

Implement `compareTo()` so that `v.compareTo(w)`

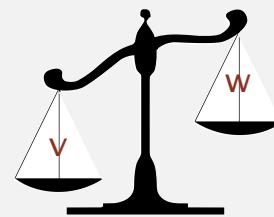
- Is a total order.
- Returns a negative integer, zero, or positive integer if v is less than, equal to, or greater than w , respectively.
- Throws an exception if incompatible types (or either is `null`).



less than (return -1)



equal to (return 0)



greater than (return +1)

Built-in comparable types. `Integer`, `Double`, `String`, `Date`, `File`, ...

User-defined comparable types. Implement the Comparable interface.

Implementing the Comparable interface

Date data type. Simplified version of java.util.Date.

```
public class Date implements Comparable<Date>
{
    private final int month, day, year;

    public Date(int m, int d, int y)
    {
        month = m;
        day   = d;
        year  = y;
    }

    public int compareTo(Date that)
    {
        if (this.year < that.year) return -1;
        if (this.year > that.year) return +1;
        if (this.month < that.month) return -1;
        if (this.month > that.month) return +1;
        if (this.day   < that.day)  return -1;
        if (this.day   > that.day)  return +1;
        return 0;
    }
}
```

only compare dates
to other dates

Two useful sorting abstractions

Helper functions. Refer to data through compares and exchanges.

Less. Is item v less than w ?

```
private static boolean less(Comparable v, Comparable w)
{  return v.compareTo(w) < 0;  }
```

Exchange. Swap item in array $a[]$ at index i with the one at index j .

```
private static void exch(Comparable[] a, int i, int j)
{
    Comparable swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

Testing

Goal. Test if an array is sorted.

```
private static boolean isSorted(Comparable[] a)
{
    for (int i = 1; i < a.length; i++)
        if (less(a[i], a[i-1])) return false;
    return true;
}
```

Q. If the sorting algorithm passes the test, did it correctly sort the array?

A.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

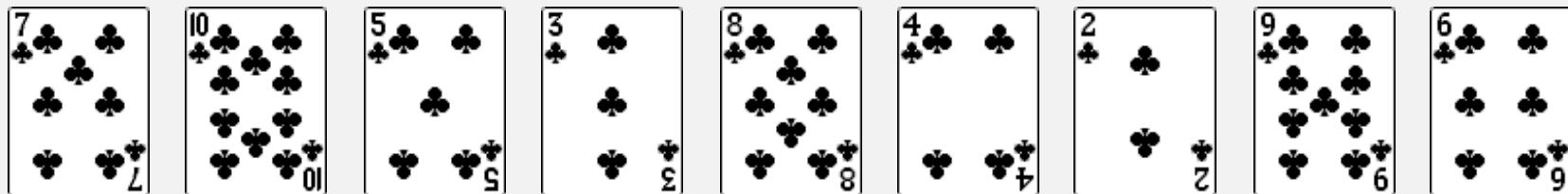
<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ ***selection sort***
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Selection sort demo

- In iteration i , find index min of smallest remaining entry.
- Swap $a[i]$ and $a[\text{min}]$.



initial

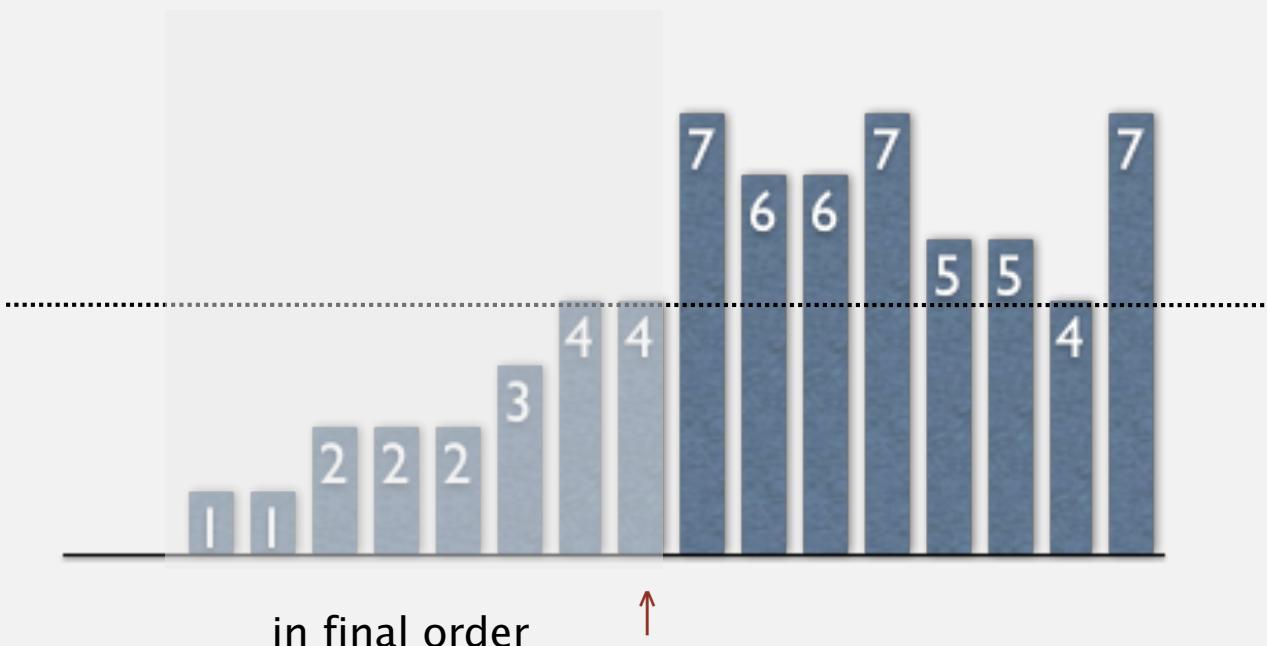


Selection sort

Algorithm. \uparrow scans from left to right.

Invariants.

- Entries to the left of \uparrow (including \uparrow) fixed and in ascending order.
- No entry to right of \uparrow is smaller than any entry to the left of \uparrow .



Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```



- Identify index of minimum entry on right.

```
int min = i;
for (int j = i+1; j < N; j++)
    if (less(a[j], a[min]))
        min = j;
```



- Exchange into position.

```
exch(a, i, min);
```



Selection sort: Java implementation

```
public class Selection
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
        {
            int min = i;
            for (int j = i+1; j < N; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }

    private static boolean less(Comparable v, Comparable w)
    { /* as before */ }

    private static void exch(Comparable[] a, int i, int j)
    { /* as before */ }
}
```

Selection sort: mathematical analysis

Proposition. Selection sort uses $(N-1) + (N-2) + \dots + 1 + 0 \sim N^2/2$ compares and N exchanges.

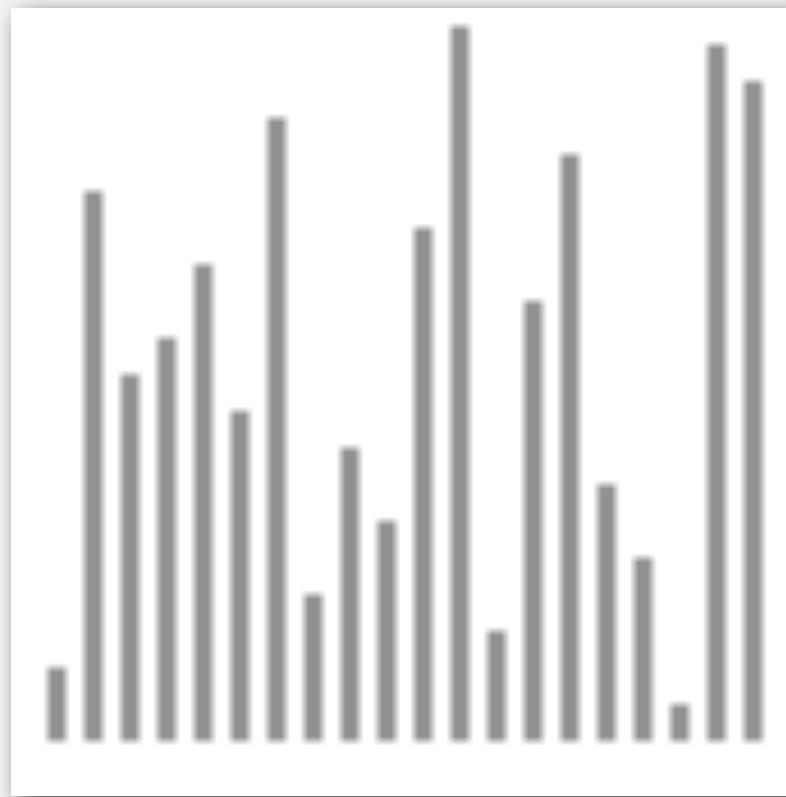
		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

Trace of selection sort (array contents just after each exchange)

Running time insensitive to input. Quadratic time, even if input is sorted.
Data movement is minimal. Linear number of exchanges.

Selection sort: animations

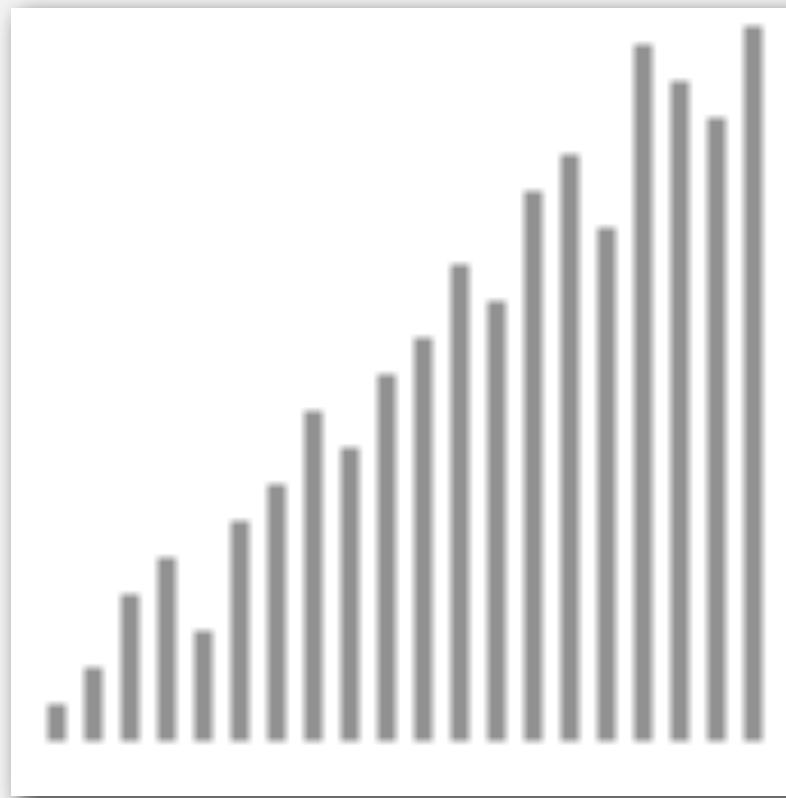
20 random items



<http://www.sorting-algorithms.com/selection-sort>

Selection sort: animations

20 partially-sorted items



<http://www.sorting-algorithms.com/selection-sort>

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ ***selection sort***
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

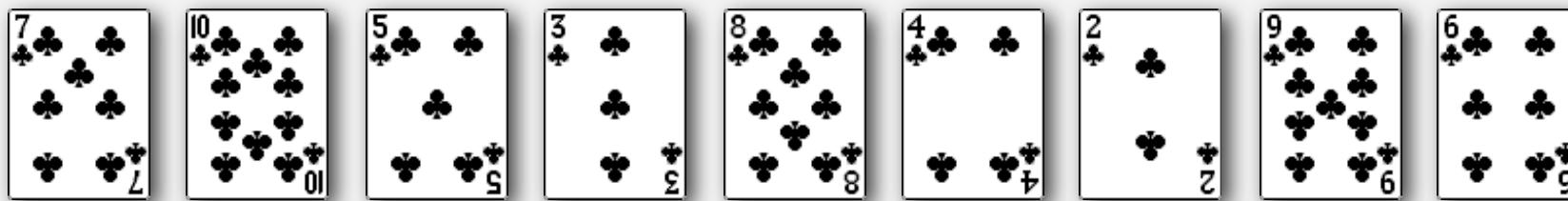
<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Insertion sort demo

- In iteration i , swap $a[i]$ with each larger entry to its left.



Insertion sort

Algorithm. ↑ scans from left to right.

Invariants.

- Entries to the left of ↑ (including ↑) are in ascending order.
- Entries to the right of ↑ have not yet been seen.



Insertion sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```



- Moving from right to left, exchange $a[i]$ with each larger entry to its left.

```
for (int j = i; j > 0; j--)  
    if (less(a[j], a[j-1]))  
        exch(a, j, j-1);  
    else break;
```



Insertion sort: Java implementation

```
public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0; j--)
                if (less(a[j], a[j-1]))
                    exch(a, j, j-1);
                else break;
    }

    private static boolean less(Comparable v, Comparable w)
    { /* as before */ }

    private static void exch(Comparable[] a, int i, int j)
    { /* as before */ }
}
```

Insertion sort: mathematical analysis

Proposition. To sort a randomly-ordered array with distinct keys, insertion sort uses $\sim \frac{1}{4} N^2$ compares and $\sim \frac{1}{4} N^2$ exchanges on average.

Pf. Expect each entry to move halfway back.

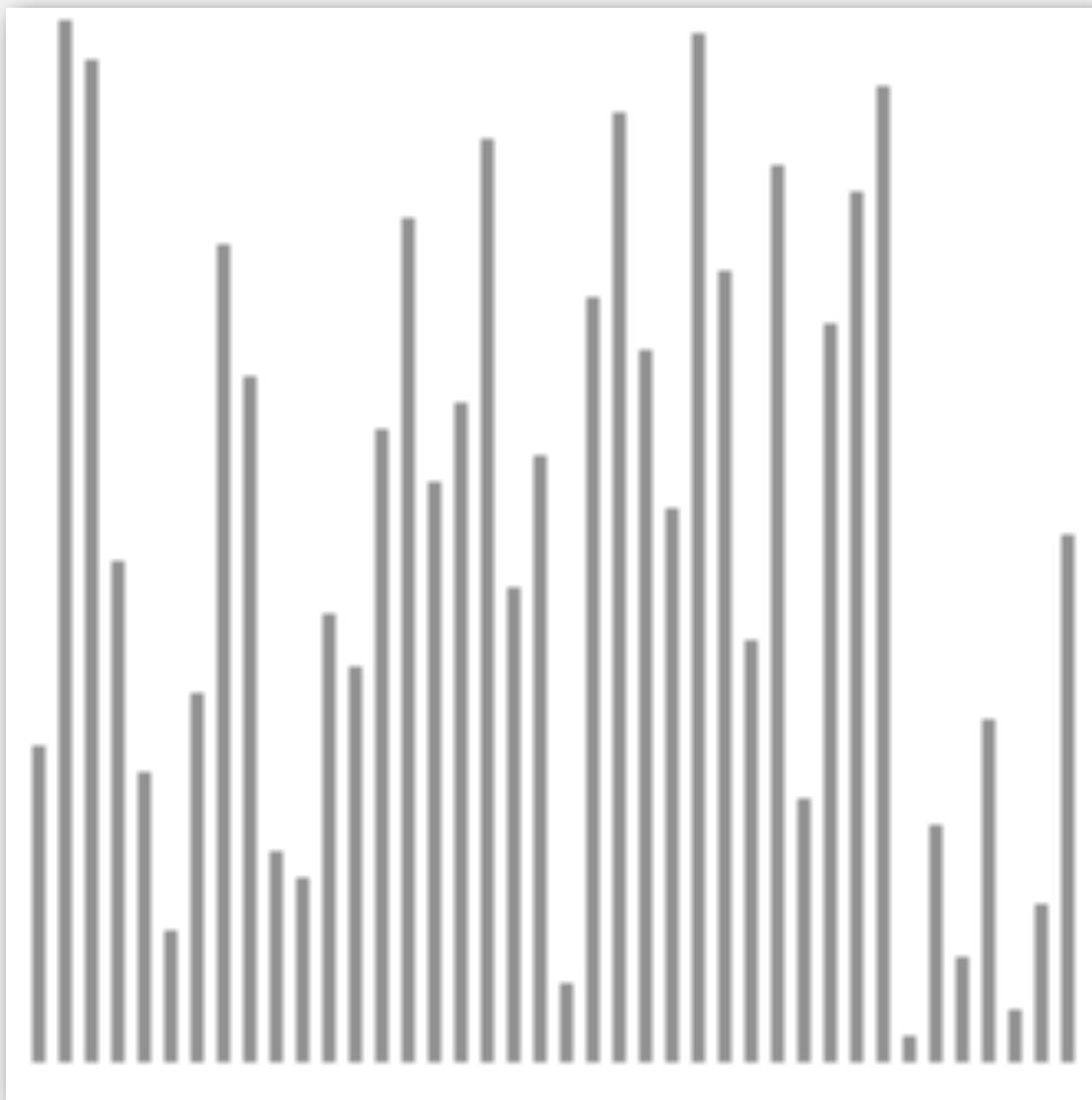
		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

Trace of insertion sort (array contents just after each insertion)

Insertion sort: trace

Insertion sort: animation

40 random items



<http://www.sorting-algorithms.com/insertion-sort>

- ▲ algorithm position
- ▬ in order
- ▬ not yet seen

Insertion sort: best and worst case

Best case. If the array is in ascending order, insertion sort makes $N-1$ compares and 0 exchanges.

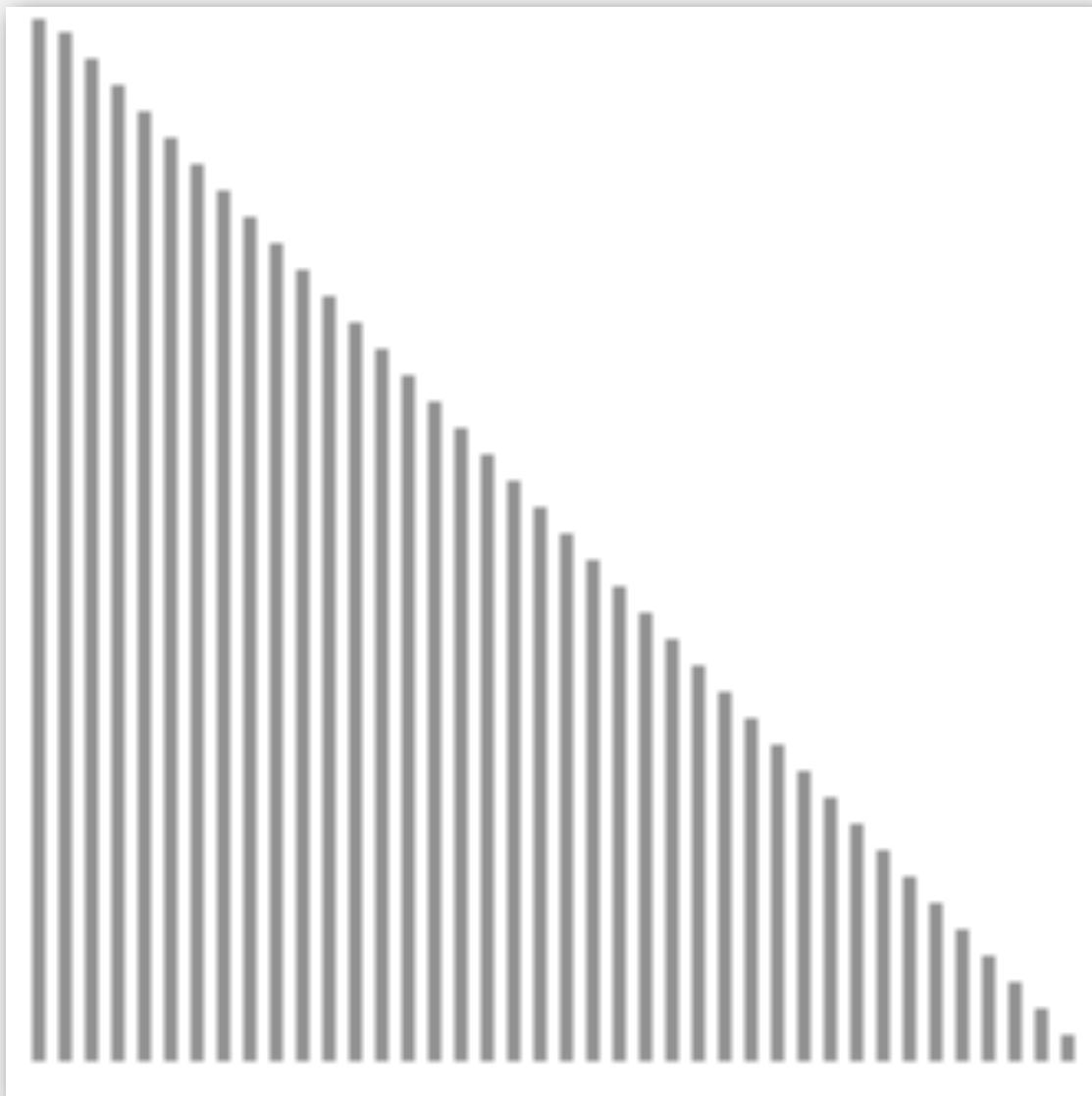
A E E L M O P R S T X

Worst case. If the array is in descending order (and no duplicates), insertion sort makes $\sim \frac{1}{2} N^2$ compares and $\sim \frac{1}{2} N^2$ exchanges.

X T S R P O M L E E A

Insertion sort: animation

40 reverse-sorted items

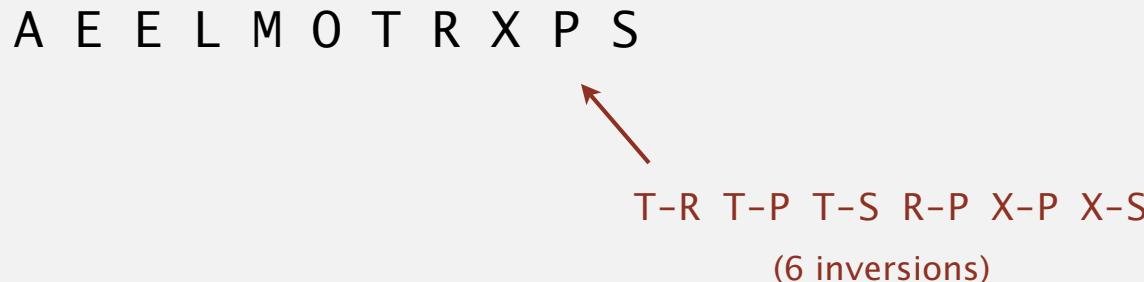


<http://www.sorting-algorithms.com/insertion-sort>

- ▲ algorithm position
- ▬ in order
- ▬ not yet seen

Insertion sort: partially-sorted arrays

Def. An **inversion** is a pair of keys that are out of order.



Def. An array is **partially sorted** if the number of inversions is $\leq c N$.

- Ex 1. A subarray of size 10 appended to a sorted subarray of size N .
- Ex 2. An array of size N with only 10 entries out of place.

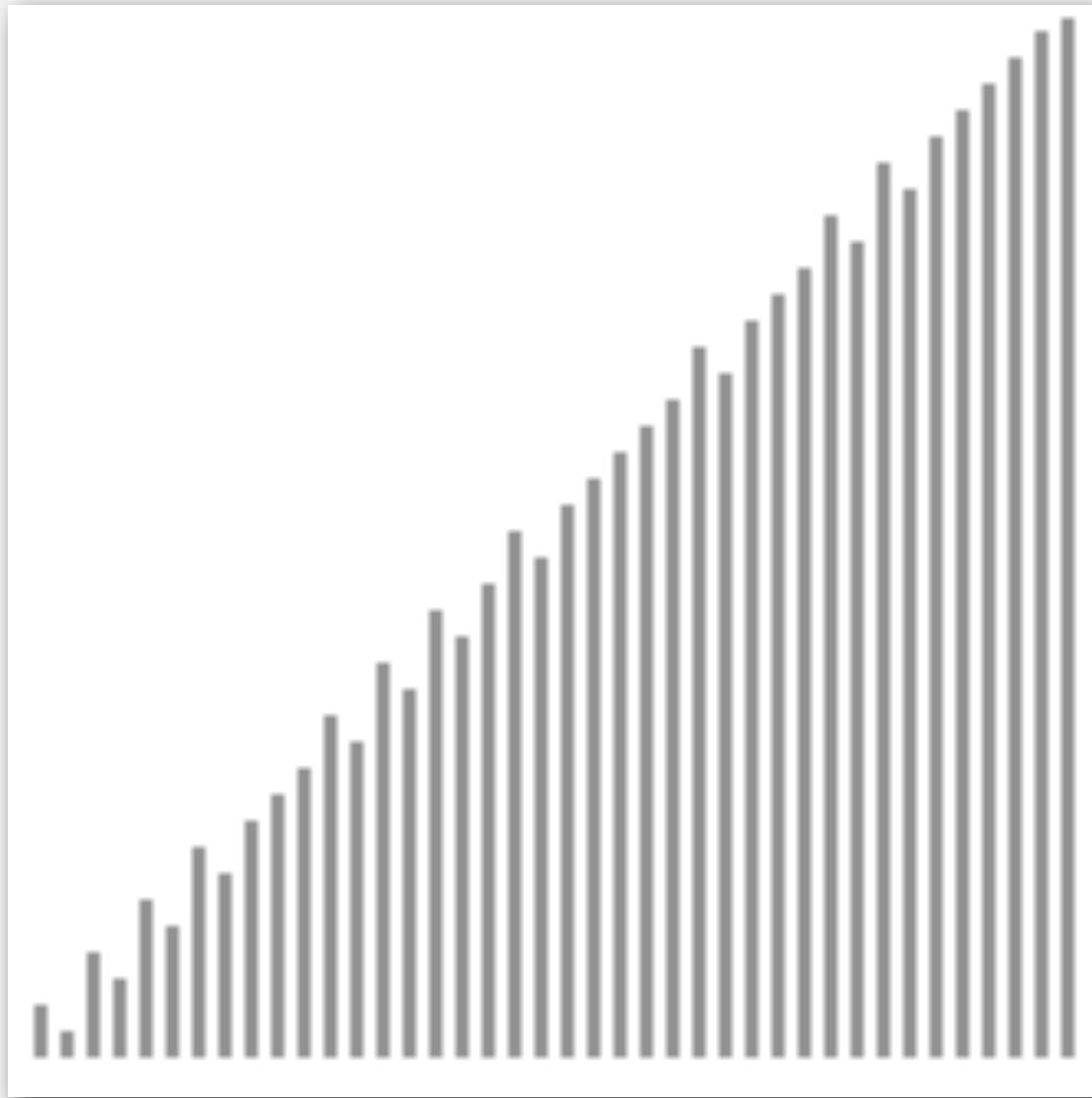
Proposition. For partially-sorted arrays, insertion sort runs in linear time.

Pf. Number of exchanges equals the number of inversions.

$$\text{number of compares} = \text{exchanges} + (N - 1)$$

Insertion sort: animation

40 partially-sorted items



- ▲ algorithm position
- in order
- not yet seen

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ ***shellsort***
- ▶ *shuffling*
- ▶ *convex hull*

Shellsort overview

Idea. Move entries more than one position at a time by *h*-sorting the array.

an *h*-sorted array is *h* interleaved sorted subsequences



Shellsort. [Shell 1959] *h*-sort array for decreasing sequence of values of *h*.

input	S	H	E	L	L	S	O	R	T	E	X	A	M	P	L	E
13-sort	P	H	E	L	L	S	O	R	T	E	X	A	M	S	L	E
4-sort	L	E	E	A	M	H	L	E	P	S	O	L	T	S	X	R
1-sort	A	E	E	E	H	L	L	L	M	O	P	R	S	S	T	X

h -sorting

How to h -sort an array? Insertion sort, with stride length h .

3-sorting an array

M	O	L	E	E	X	A	S	P	R	T
E	O	L	M	E	X	A	S	P	R	T
E	E	L	M	O	X	A	S	P	R	T
E	E	L	M	O	X	A	S	P	R	T
A	E	L	E	O	X	M	S	P	R	T
A	E	L	E	O	X	M	S	P	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T
A	E	L	E	O	P	M	S	X	R	T

Why insertion sort?

- Big increments \Rightarrow small subarray.
- Small increments \Rightarrow nearly in order. [stay tuned]

Shellsort example: increments 7, 3, 1

input

S O R T E X A M P L E

1-sort

A E L E O P M S X R T

A E L E O P M S X R T

A E L E O P M S X R T

A E L O P M S X R T

A E L O P M S X R T

A E L M O P M S X R T

A E L M O P M S X R T

A E L M O P S X R T

A E L M O P S X R T

A E L M O P R S X T

A E L M O P R S T X

7-sort

S O R T E X A M P L E

M O R T E X A S P L E

M O R T E X A S P L E

M O L T E X A S P R E

M O L E E X A S P R T

3-sort

M O L E E X A S P R T

E O L M E X A S P R T

E E L M O X A S P R T

E E L M O X A S P R T

A E L E O X M S P R T

A E L E O X M S P R T

A E L E O P M S X R T

A E L E O P M S X R T

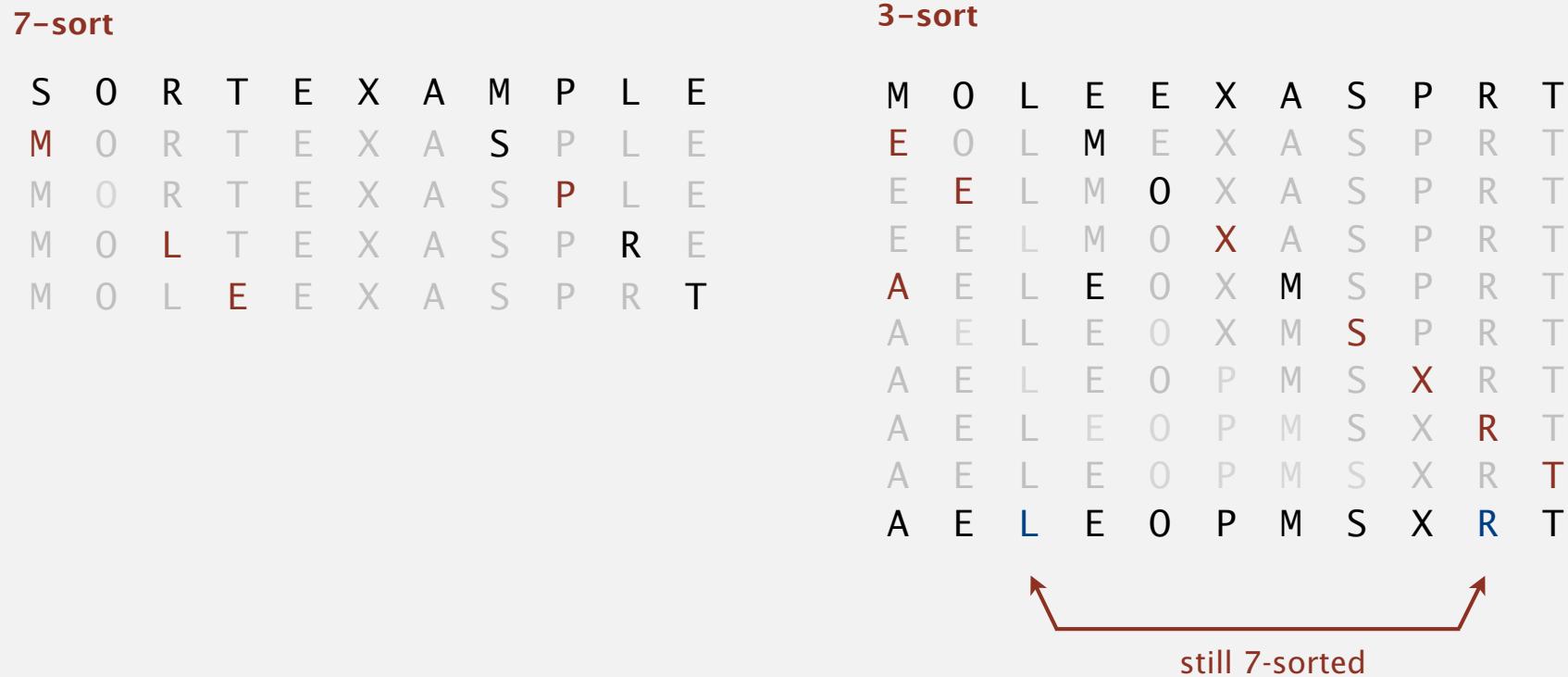
A E L E O P M S X R T

result

A E E L M O P R S T X

Shellsort: intuition

Proposition. A g -sorted array remains g -sorted after h -sorting it.



Challenge. Prove this fact—it's more subtle than you'd think!

Shellsort: which increment sequence to use?

Powers of two. 1, 2, 4, 8, 16, 32, ...

No.

Powers of two minus one. 1, 3, 7, 15, 31, 63, ...

Maybe.

→ $3x + 1$. 1, 4, 13, 40, 121, 364, ...

OK. Easy to compute.

Sedgewick. 1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, ...

Good. Tough to beat in empirical studies.

merging of $(9 \times 4^i) - (9 \times 2^i) + 1$
and $4^i - (3 \times 2^i) + 1$

Shellsort: Java implementation

```
public class Shell
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;

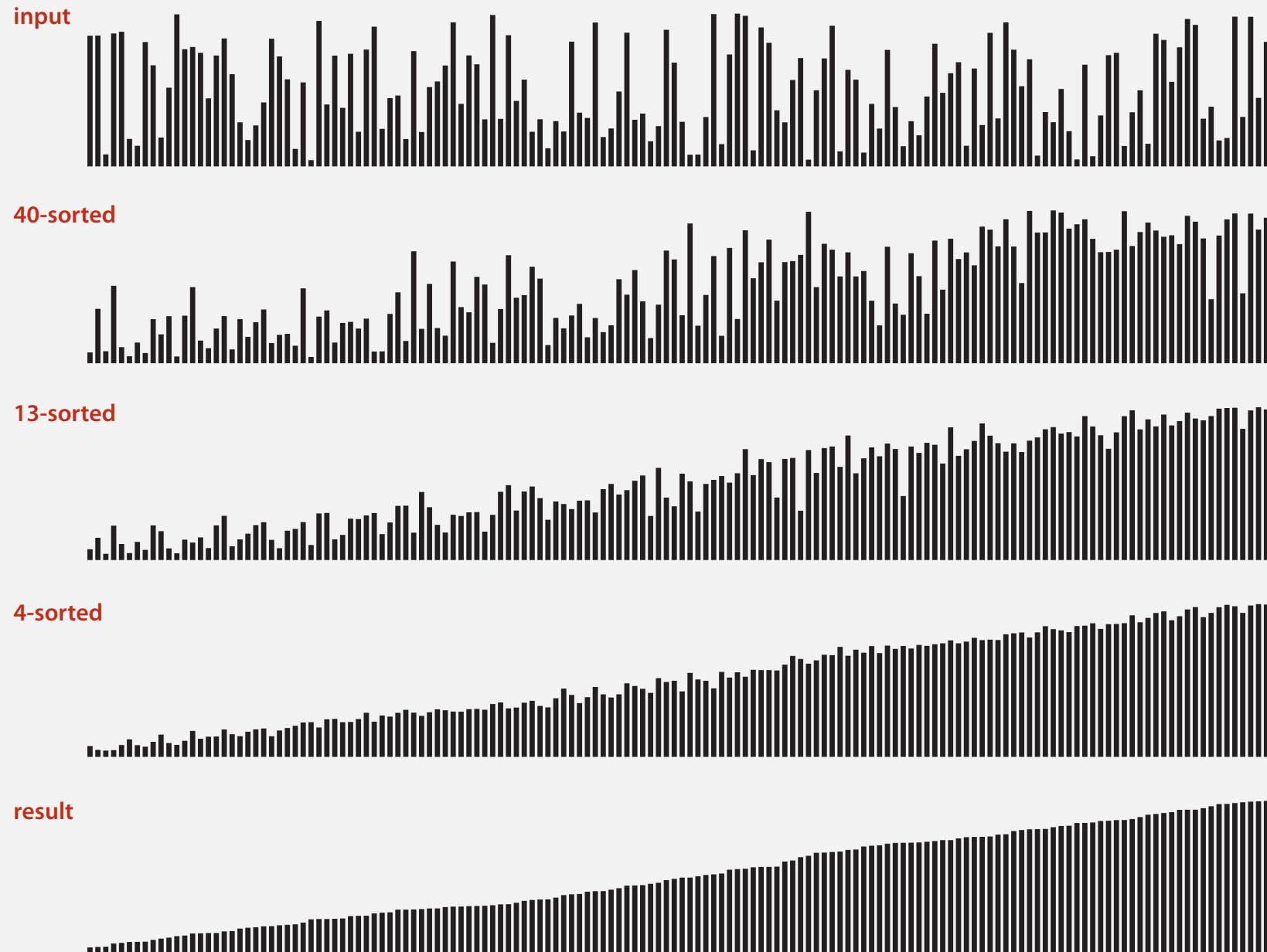
        int h = 1;
        while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, ... ← 3x+1 increment sequence

        while (h >= 1)
        { // h-sort the array.
            for (int i = h; i < N; i++) ← insertion sort
            {
                for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }

            h = h/3; ← move to next increment
        }

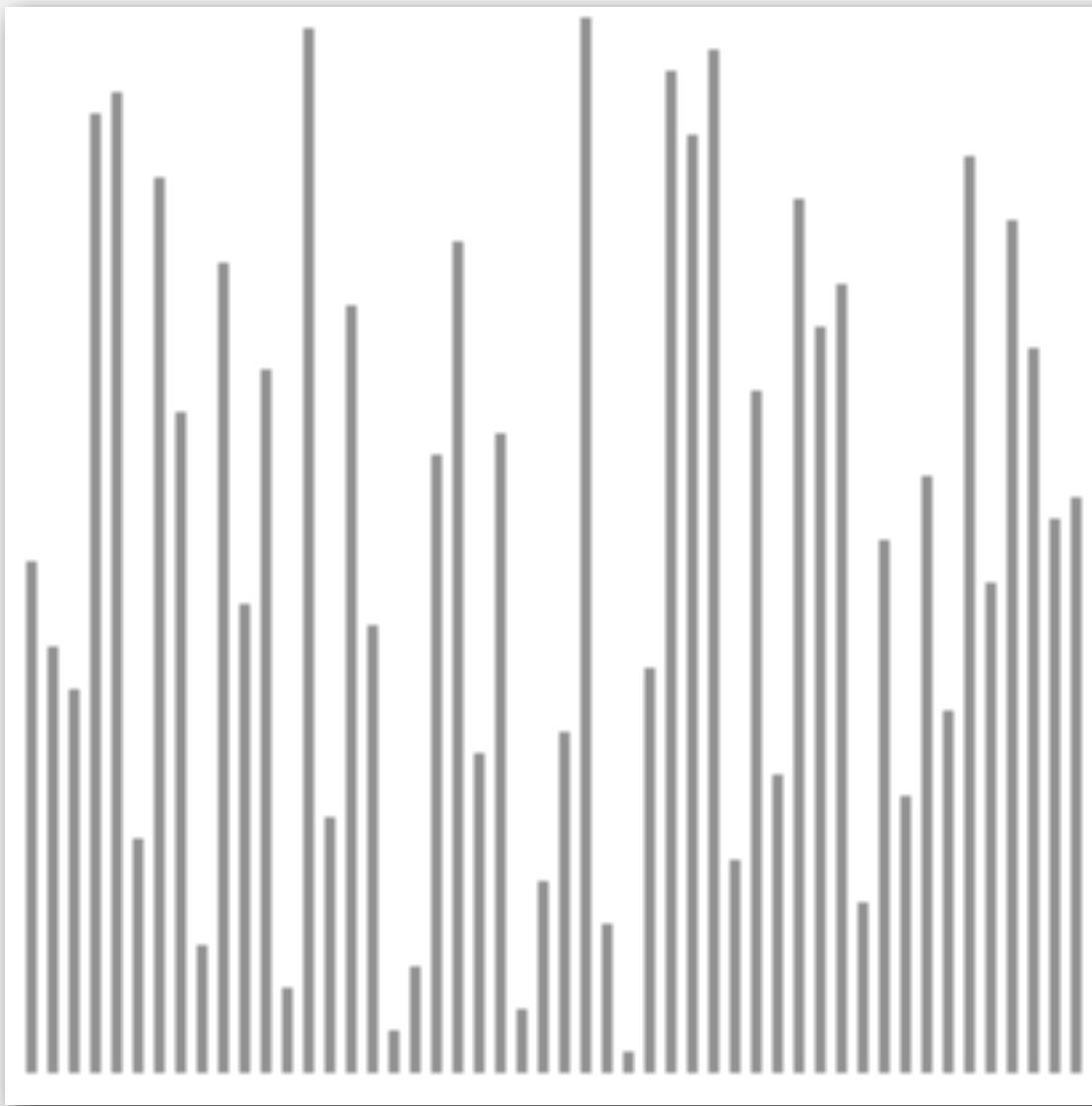
        private static boolean less(Comparable v, Comparable w)
        { /* as before */ }
        private static boolean exch(Comparable[] a, int i, int j)
        { /* as before */ }
    }
}
```

Shellsort: visual trace



Shellsort: animation

50 random items

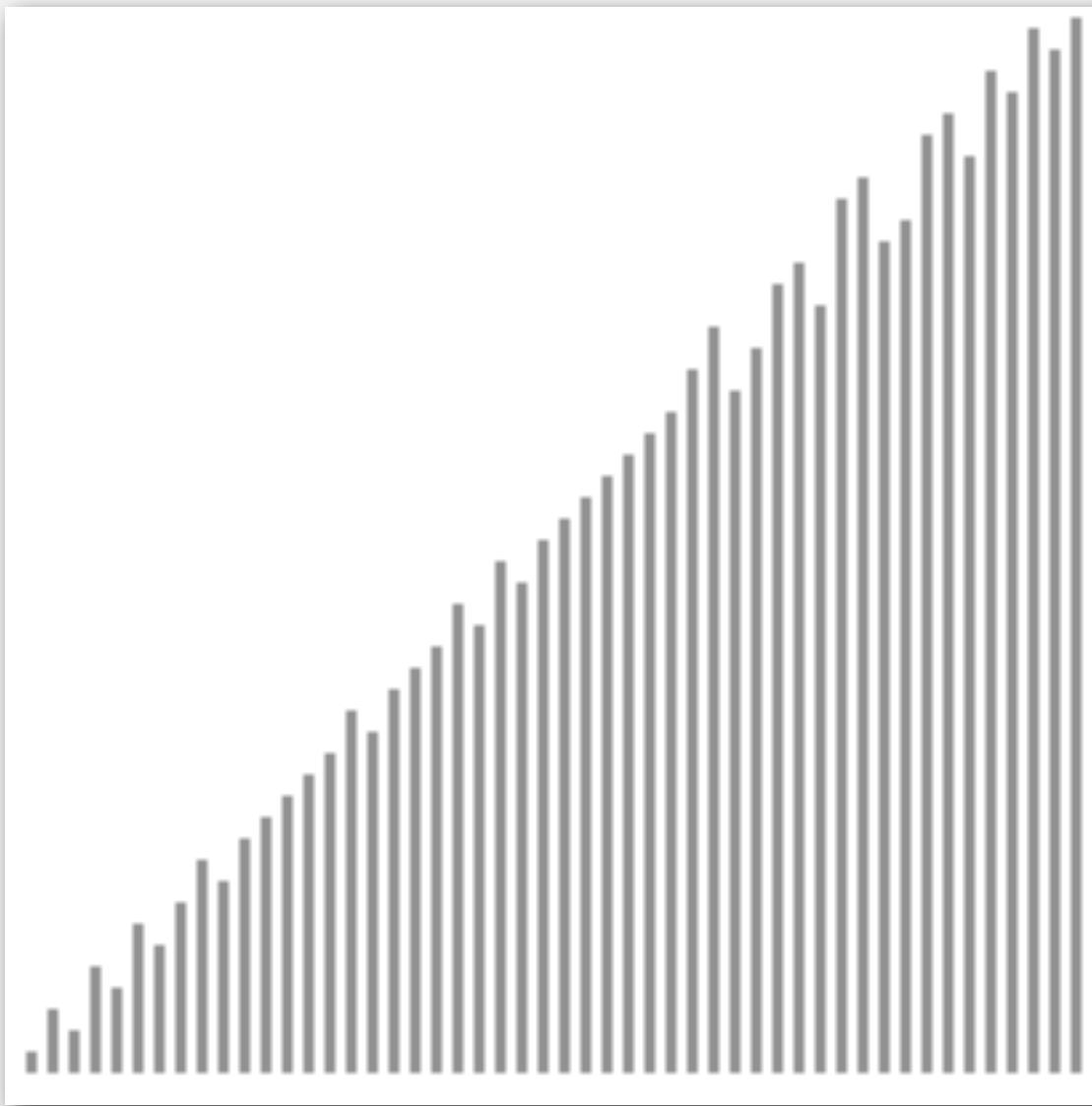


<http://www.sorting-algorithms.com/shell-sort>

- ▲ algorithm position
- █ h-sorted
- █ current subsequence
- █ other elements

Shellsort: animation

50 partially-sorted items



<http://www.sorting-algorithms.com/shell-sort>

The legend consists of four entries:

- A red triangle pointing upwards followed by the text "algorithm position".
- A thick black horizontal bar followed by the text "h-sorted".
- A dark grey horizontal bar followed by the text "current subsequence".
- A light grey horizontal bar followed by the text "other elements".

Shellsort: analysis

Proposition. The worst-case number of compares used by shellsort with the $3x+1$ increments is $O(N^{3/2})$.

Property. Number of compares used by shellsort with the $3x+1$ increments is at most by a small multiple of N times the # of increments used.

N	compares	$N^{1.289}$	$2.5 N \lg N$
5,000	93	58	106
10,000	209	143	230
20,000	467	349	495
40,000	1022	855	1059
80,000	2266	2089	2257

measured in thousands

Remark. Accurate model has not yet been discovered (!)

Why are we interested in shellsort?

Example of simple idea leading to substantial performance gains.

Useful in practice.

- Fast unless array size is huge.
- Tiny, fixed footprint for code (used in embedded systems).
- Hardware sort prototype.

Simple algorithm, nontrivial performance, interesting questions.

- Asymptotic growth rate?
- Best sequence of increments? ← open problem: find a better increment sequence
- Average-case performance?

Lesson. Some good algorithms are still waiting discovery.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ ***shellsort***
- ▶ *shuffling*
- ▶ *convex hull*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

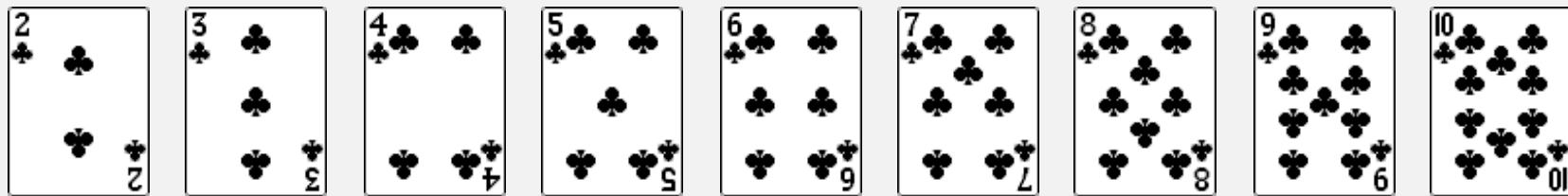
<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ ***shuffling***
- ▶ *convex hull*

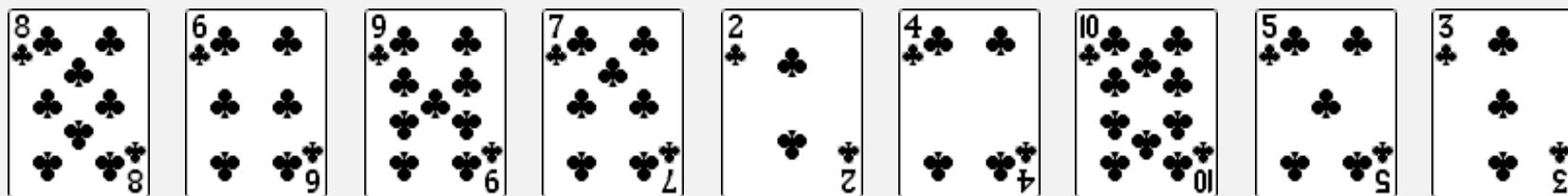
How to shuffle an array

Goal. Rearrange array so that result is a uniformly random permutation.



How to shuffle an array

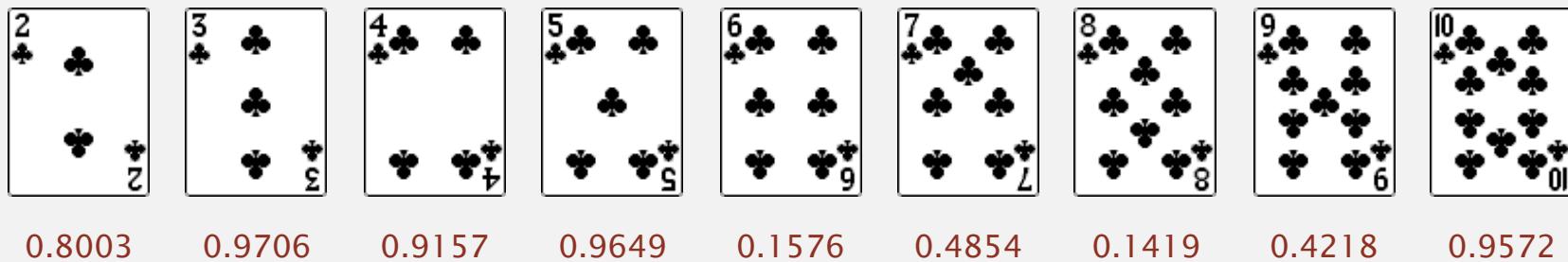
Goal. Rearrange array so that result is a uniformly random permutation.



Shuffle sort

- Generate a random real number for each array entry.
- Sort the array.

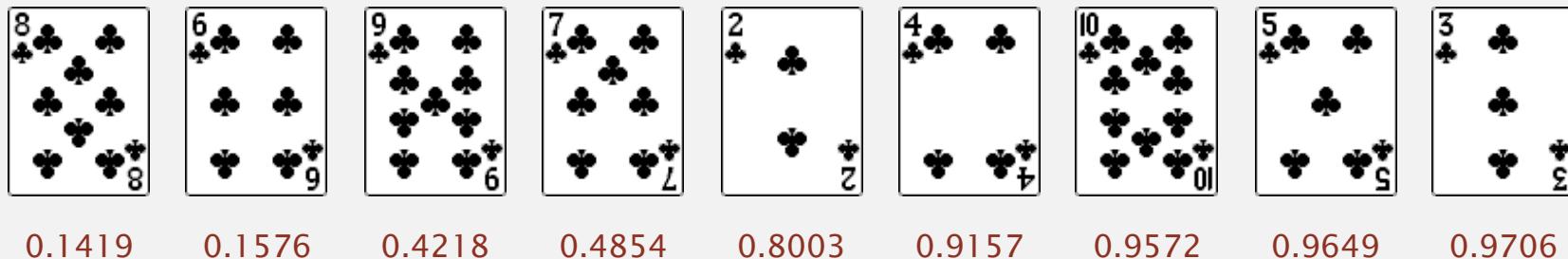
useful for shuffling
columns in a spreadsheet



Shuffle sort

- Generate a random real number for each array entry.
- Sort the array.

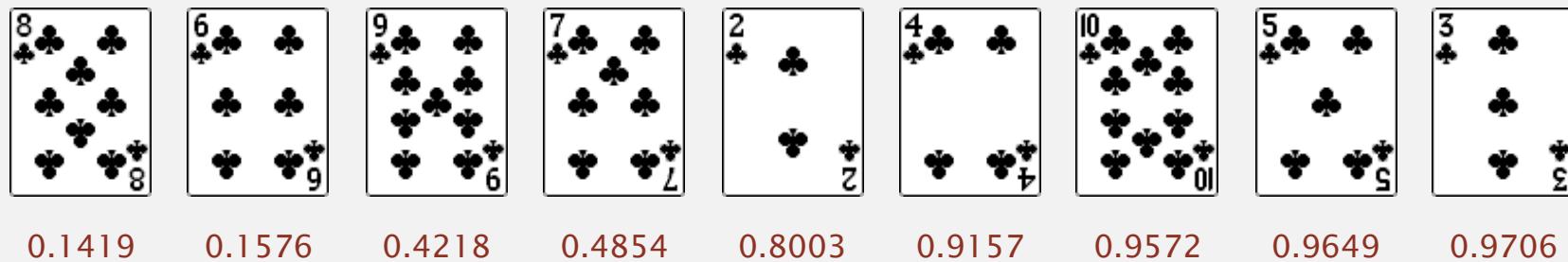
useful for shuffling
columns in a spreadsheet



Shuffle sort

- Generate a random real number for each array entry.
- Sort the array.

useful for shuffling
columns in a spreadsheet



Proposition. Shuffle sort produces a uniformly random permutation
of the input array, provided no duplicate values.

assuming real numbers
uniformly at random

War story (Microsoft)

Microsoft antitrust probe by EU. Microsoft agreed to provide a randomized ballot screen for users to select browser in Windows 7.

<http://www.browserchoice.eu>

Select your web browser(s)

**Google chrome**

A fast new browser from Google. Try it now!

**Safari**

Safari for Windows from Apple, the world's most innovative browser.

**mozilla Firefox**

Your online security is Firefox's top priority. Firefox is free, and made to help you get the most out of the

**Opera browser**

The fastest browser on Earth. Secure, powerful and easy to use, with excellent privacy protection.

**Windows Internet Explorer 8**

Designed to help you take control of your privacy and browse with confidence. Free from Microsoft.



appeared last
50% of the time

War story (Microsoft)

Microsoft antitrust probe by EU. Microsoft agreed to provide a randomized ballot screen for users to select browser in Windows 7.

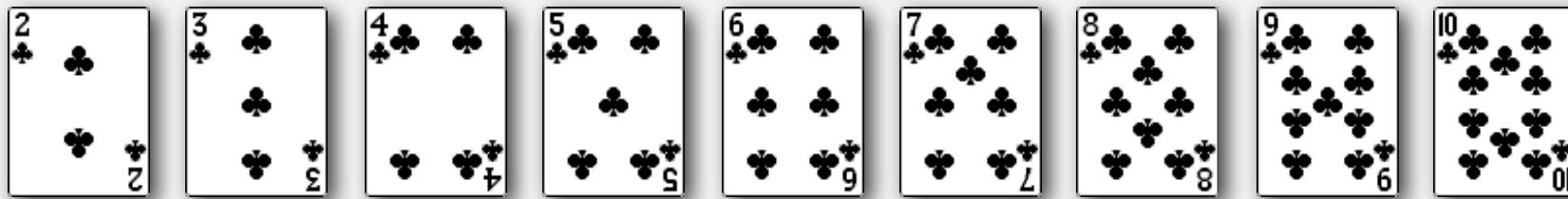
Solution? Implement shuffle sort by making comparator always return a random answer.

```
public int compareTo(Browser that)
{
    double r = Math.random();
    if (r < 0.5) return -1;
    if (r > 0.5) return +1;
    return 0;
}
```

← browser comparator
(should implement a total order)

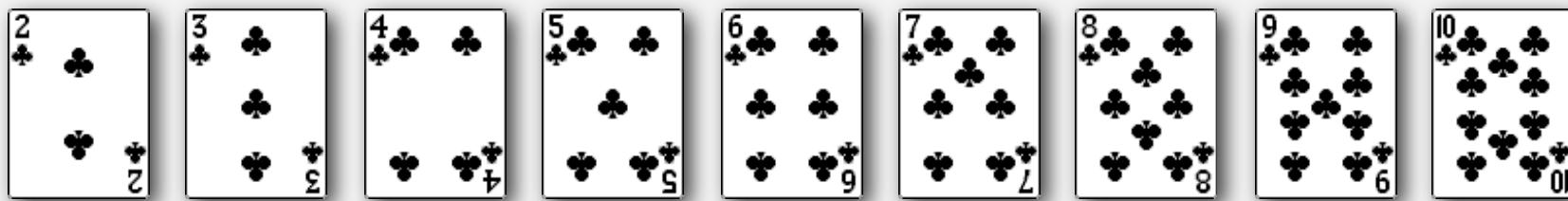
Knuth shuffle demo

- In iteration i , pick integer r between 0 and i uniformly at random.
- Swap $a[i]$ and $a[r]$.



Knuth shuffle

- In iteration i , pick integer r between 0 and i uniformly at random.
- Swap $a[i]$ and $a[r]$.



Proposition. [Fisher-Yates 1938] Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.

→ assuming integers
uniformly at random

Knuth shuffle

- In iteration i , pick integer r between 0 and i uniformly at random.
- Swap $a[i]$ and $a[r]$.

common bug: between 0 and $N - 1$
correct variant: between i and $N - 1$

```
public class StdRandom
{
    ...
    public static void shuffle(0bject[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
        {
            int r = StdRandom.uniform(i + 1);           ← between 0 and i
            exch(a, i, r);
        }
    }
}
```

War story (online poker)

Texas hold'em poker. Software must shuffle electronic cards.



How We Learned to Cheat at Online Poker: A Study in Software Security
<http://itmanagement.earthweb.com/entdev/article.php/616221>

War story (online poker)

Shuffling algorithm in FAQ at www.planetpoker.com

```
for i := 1 to 52 do begin
    r := random(51) + 1;           ← between 1 and 51
    swap := card[r];
    card[r] := card[i];
    card[i] := swap;
end;
```

- Bug 1. Random number r never 52 \Rightarrow 52nd card can't end up in 52nd place.
- Bug 2. Shuffle not uniform (should be between 1 and i).
- Bug 3. `random()` uses 32-bit seed \Rightarrow 2^{32} possible shuffles.
- Bug 4. Seed = milliseconds since midnight \Rightarrow 86.4 million shuffles.

“The generation of random numbers is too important to be left to chance.”

— Robert R. Coveyou

War story (online poker)

Best practices for shuffling (if your business depends on it).

- Use a hardware random-number generator that has passed both the FIPS 140-2 and the NIST statistical test suites.
- Continuously monitor statistic properties:
hardware random-number generators are fragile and fail silently.
- Use an unbiased shuffling algorithm.



Bottom line. Shuffling a deck of cards is hard!

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ ***shuffling***
- ▶ *convex hull*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

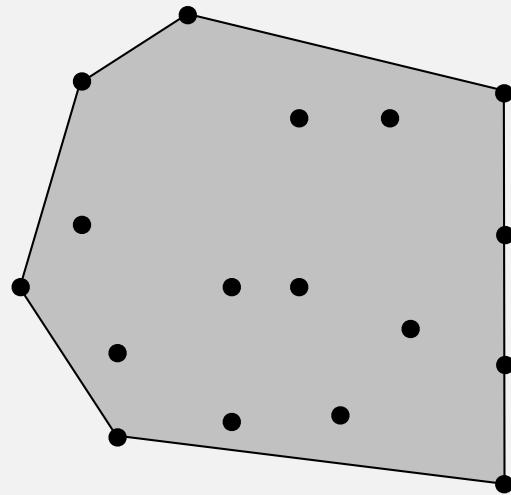
<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ ***convex hull***

Convex hull

The **convex hull** of a set of N points is the smallest perimeter fence enclosing the points.

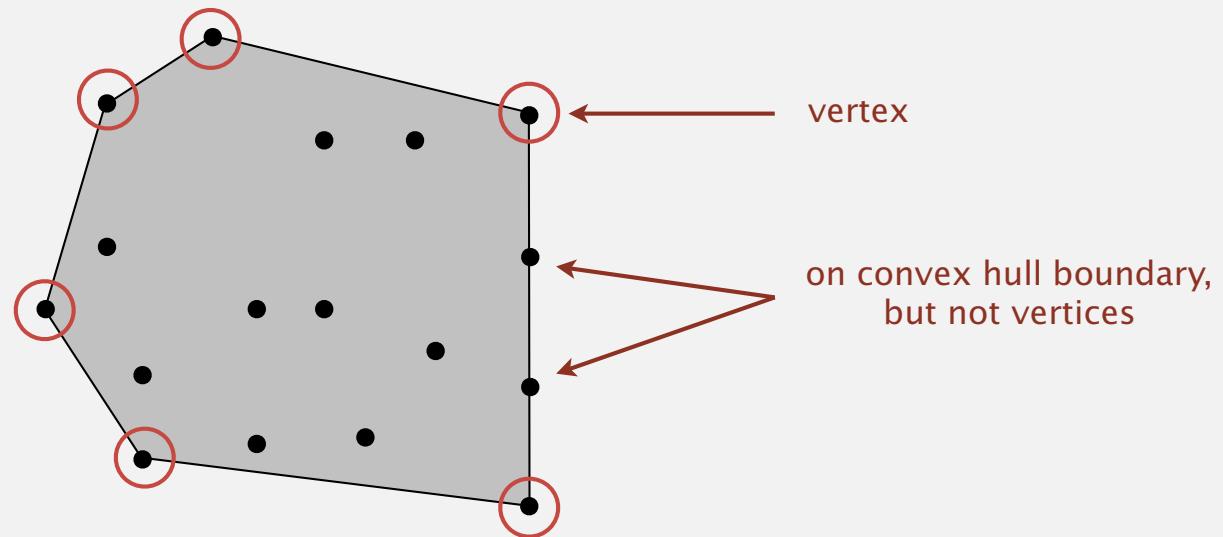


Equivalent definitions.

- Smallest convex set containing all the points.
- Smallest area convex polygon enclosing the points.
- Convex polygon enclosing the points, whose vertices are points in set.

Convex hull

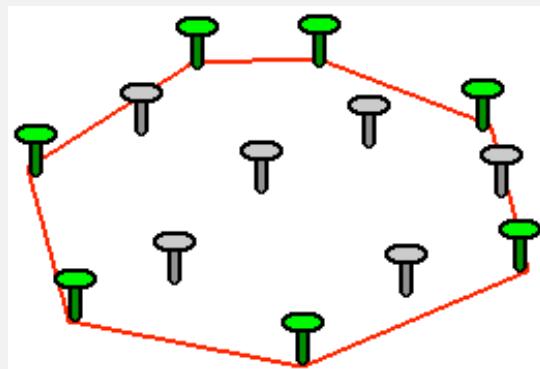
The **convex hull** of a set of N points is the smallest perimeter fence enclosing the points.



Convex hull output. Sequence of vertices in counterclockwise order.

Convex hull: mechanical algorithm

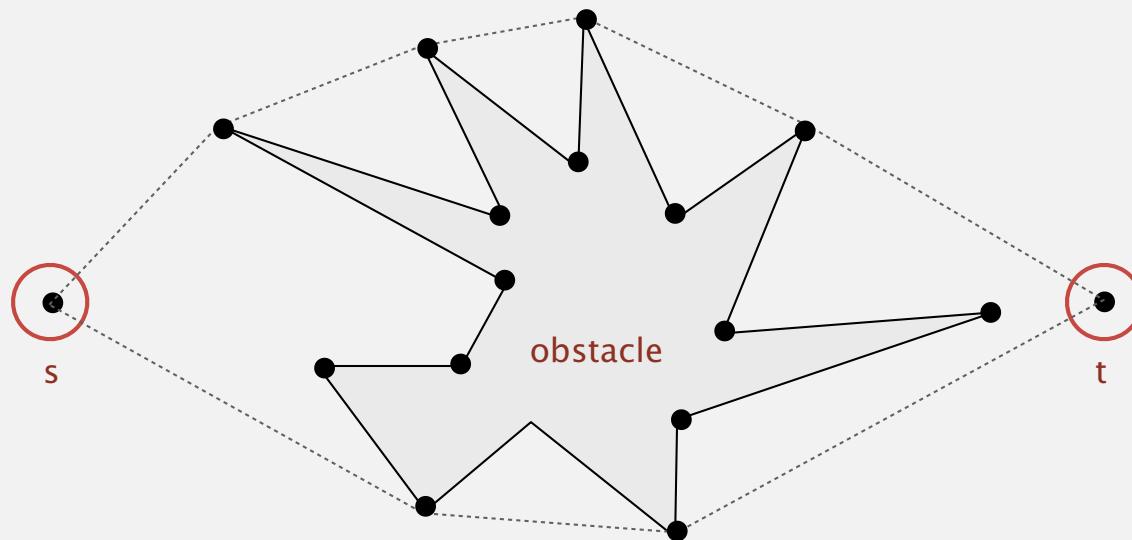
Mechanical algorithm. Hammer nails perpendicular to plane; stretch elastic rubber band around points.



http://www.dfanning.com/math_tips/convexhull_1.gif

Convex hull application: motion planning

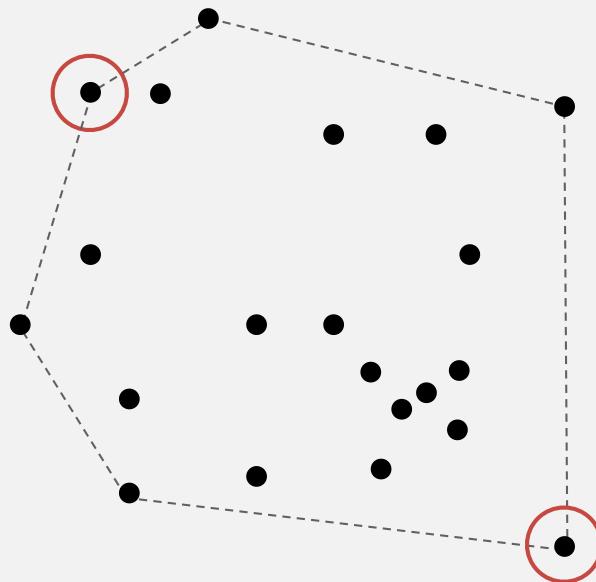
Robot motion planning. Find shortest path in the plane from s to t that avoids a polygonal obstacle.



Fact. Shortest path is either straight line from s to t or it is one of two polygonal chains of convex hull.

Convex hull application: farthest pair

Farthest pair problem. Given N points in the plane, find a pair of points with the largest Euclidean distance between them.

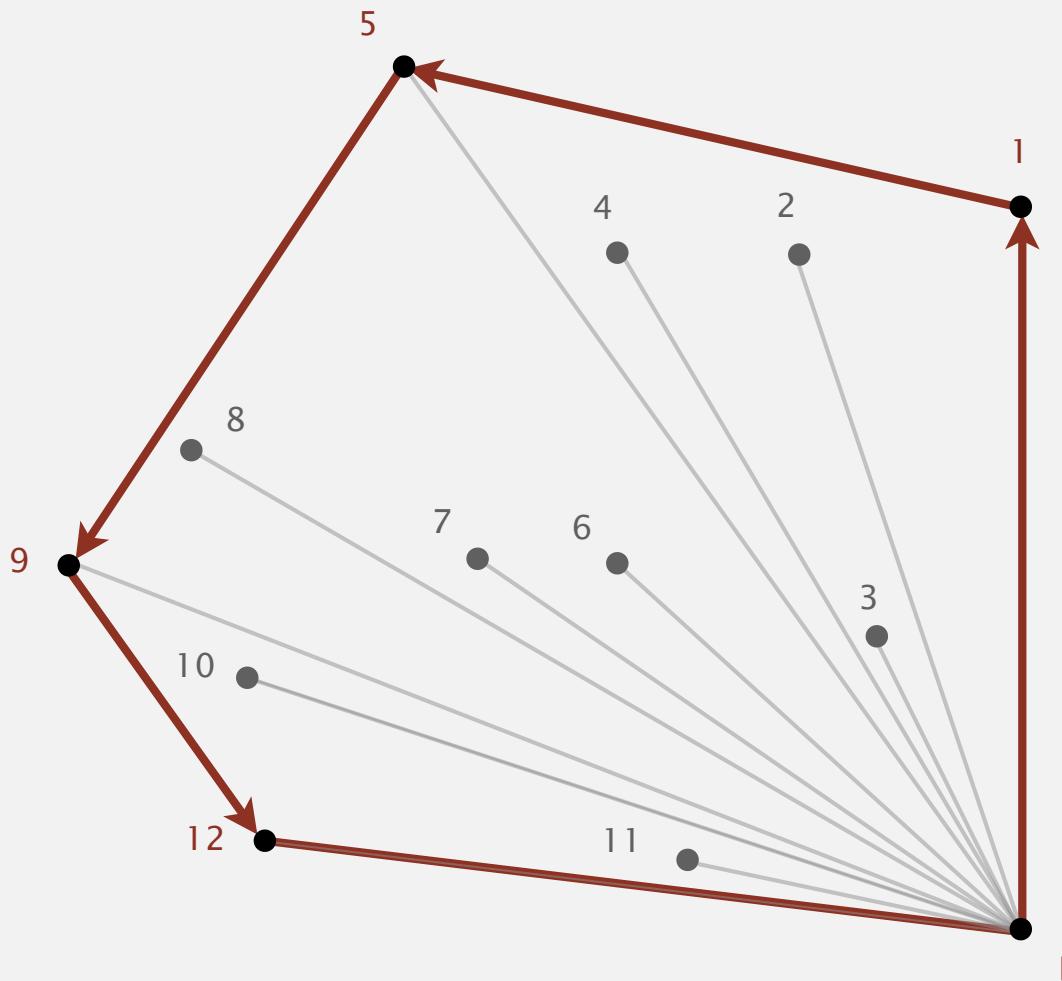


Fact. Farthest pair of points are extreme points on convex hull.

Convex hull: geometric properties

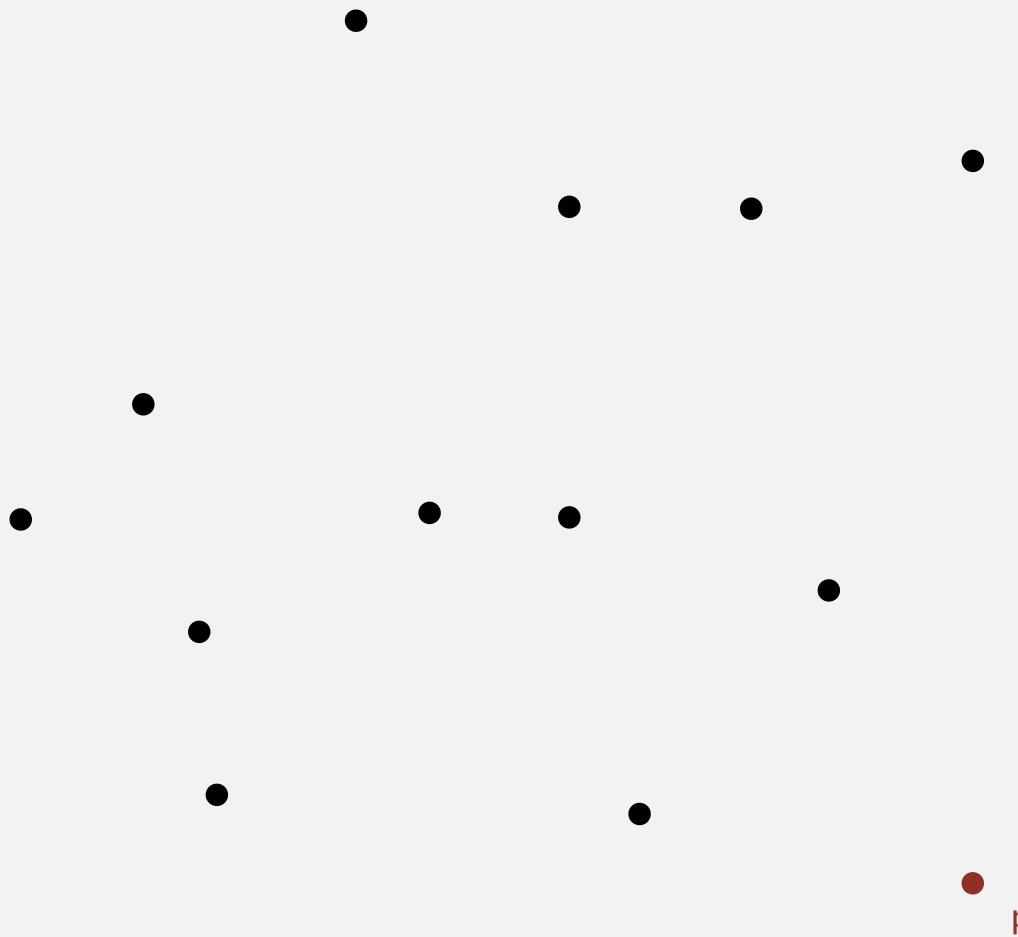
Fact. Can traverse the convex hull by making only counterclockwise turns.

Fact. The vertices of convex hull appear in increasing order of polar angle with respect to point p with lowest y -coordinate.



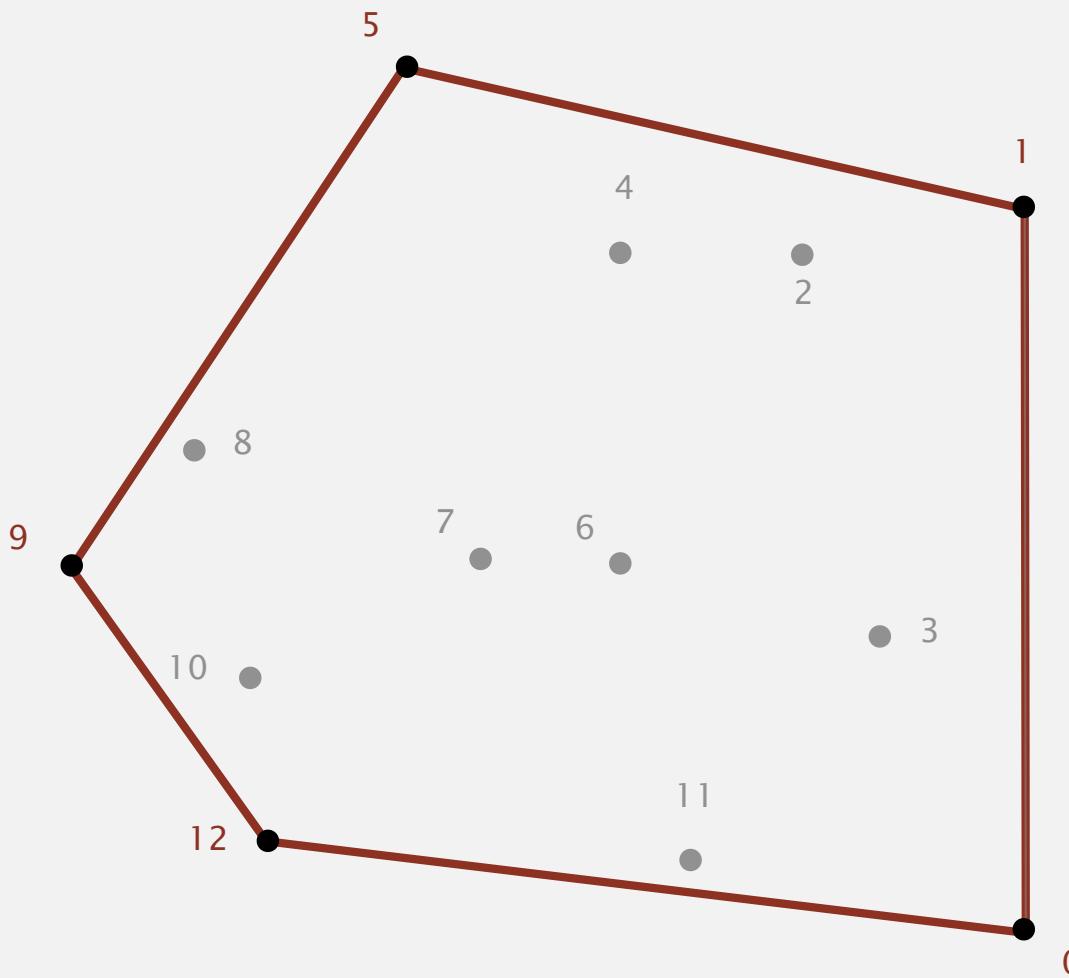
Graham scan demo

- Choose point p with smallest y -coordinate.
- Sort points by polar angle with p .
- Consider points in order; discard unless it creates a ccw turn.



Graham scan demo

- Choose point p with smallest y -coordinate.
- Sort points by polar angle with p .
- Consider points in order; discard unless it creates a ccw turn.



Graham scan: implementation challenges

- Q. How to find point p with smallest y -coordinate?

A. Define a total order, comparing by y -coordinate. [next lecture]

- Q. How to sort points by polar angle with respect to p ?

A. Define a total order **for each point p** . [next lecture]

- Q. How to determine whether $p_1 \rightarrow p_2 \rightarrow p_3$ is a counterclockwise turn?

A. Computational geometry. [next two slides]

- Q. How to sort efficiently?

A. Mergesort sorts in $N \log N$ time. [next lecture]

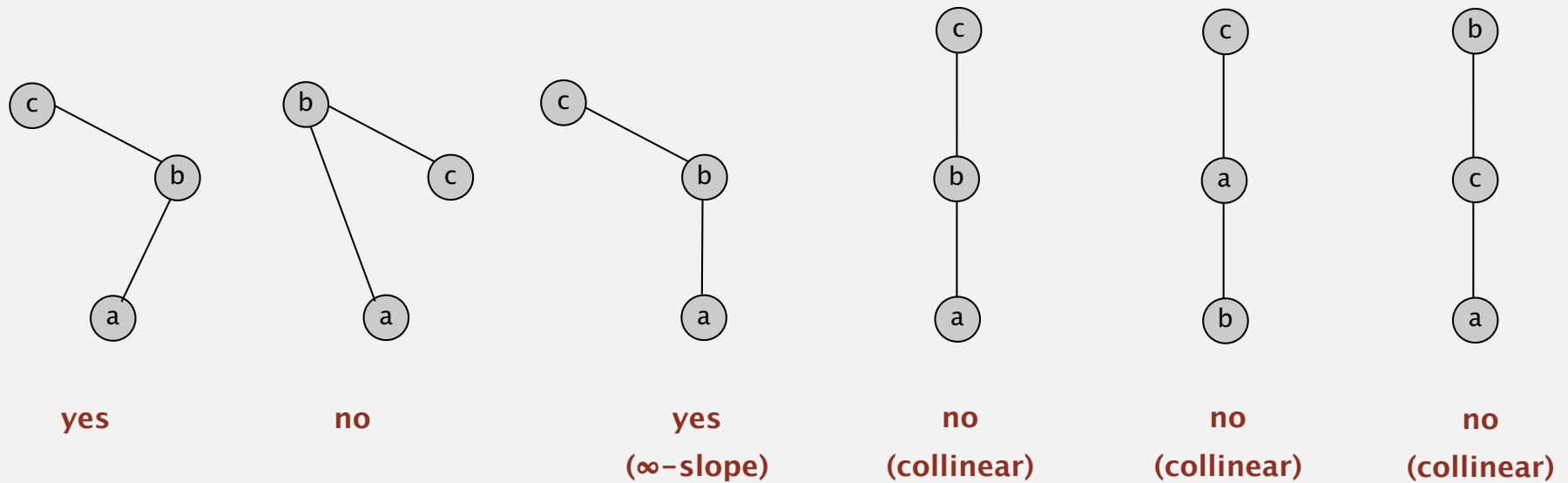
- Q. How to handle degeneracies (three or more points on a line)?

A. Requires some care, but not hard. [see booksite]

Implementing ccw

CCW. Given three points a , b , and c , is $a \rightarrow b \rightarrow c$ a counterclockwise turn?

is c to the left of the ray $a \rightarrow b$



Lesson. Geometric primitives are tricky to implement.

- Dealing with degenerate cases.
- Coping with floating-point precision.

Implementing ccw

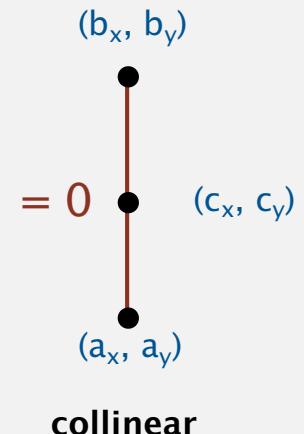
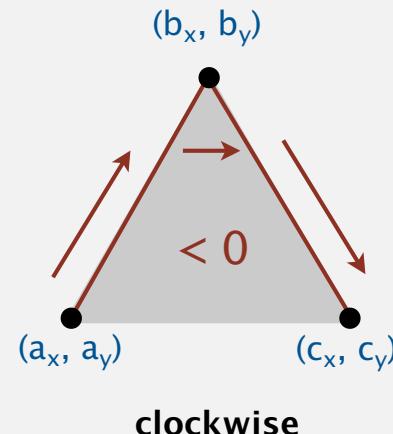
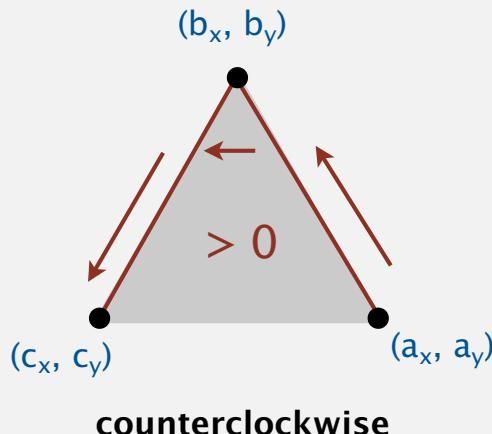
CCW. Given three points a , b , and c , is $a \rightarrow b \rightarrow c$ a counterclockwise turn?

- Determinant (or cross product) gives 2x signed area of planar triangle.

$$2 \times \text{Area}(a, b, c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

(b - a) × (c - a)

- If signed area > 0 , then $a \rightarrow b \rightarrow c$ is counterclockwise.
- If signed area < 0 , then $a \rightarrow b \rightarrow c$ is clockwise.
- If signed area $= 0$, then $a \rightarrow b \rightarrow c$ are collinear.



Immutable point data type

```
public class Point2D
{
    private final double x;
    private final double y;

    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    ...

    public static int ccw(Point a, Point b, Point c)
    {
        int area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
        if      (area2 < 0) return -1; // clockwise
        else if (area2 > 0) return +1; // counter-clockwise
        else                  return 0; // collinear
    }
}
```

danger of
floating-point
roundoff error

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

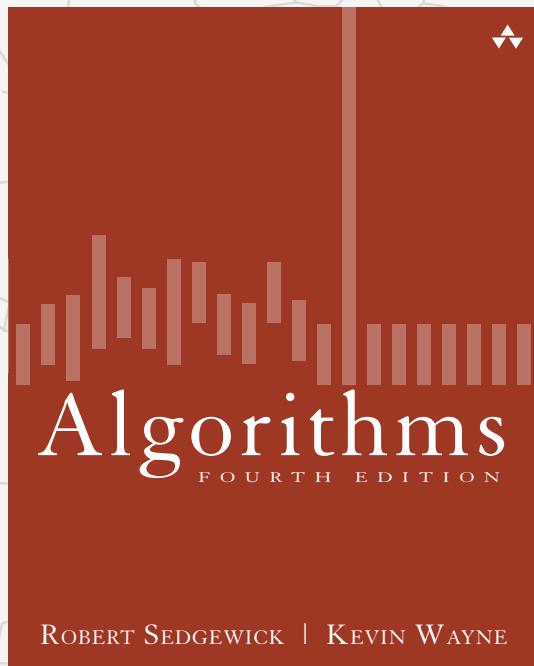
<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ ***convex hull***

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

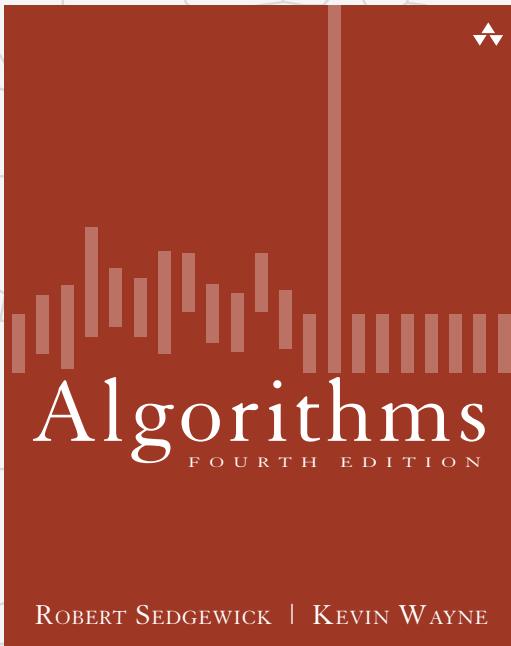


ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *shellsort*
- ▶ *shuffling*
- ▶ *convex hull*



<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Two classic sorting algorithms

Critical components in the world's computational infrastructure.

- Full scientific understanding of their properties has enabled us to develop them into practical system sorts.
- Quicksort honored as one of top 10 algorithms of 20th century in science and engineering.

Mergesort. [this lecture]

- Java sort for objects.
- Perl, C++ stable sort, Python stable sort, Firefox JavaScript, ...

Quicksort. [next lecture]

- Java sort for primitive types.
- C qsort, Unix, Visual C++, Python, Matlab, Chrome JavaScript, ...

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

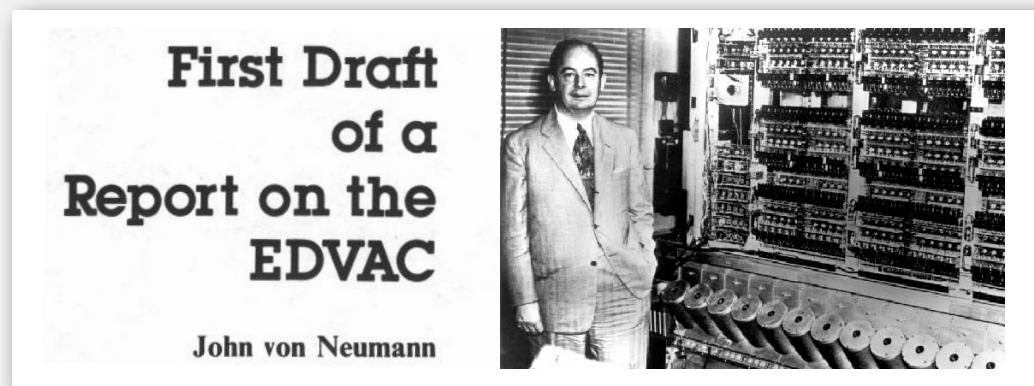
Mergesort

Basic plan.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves.

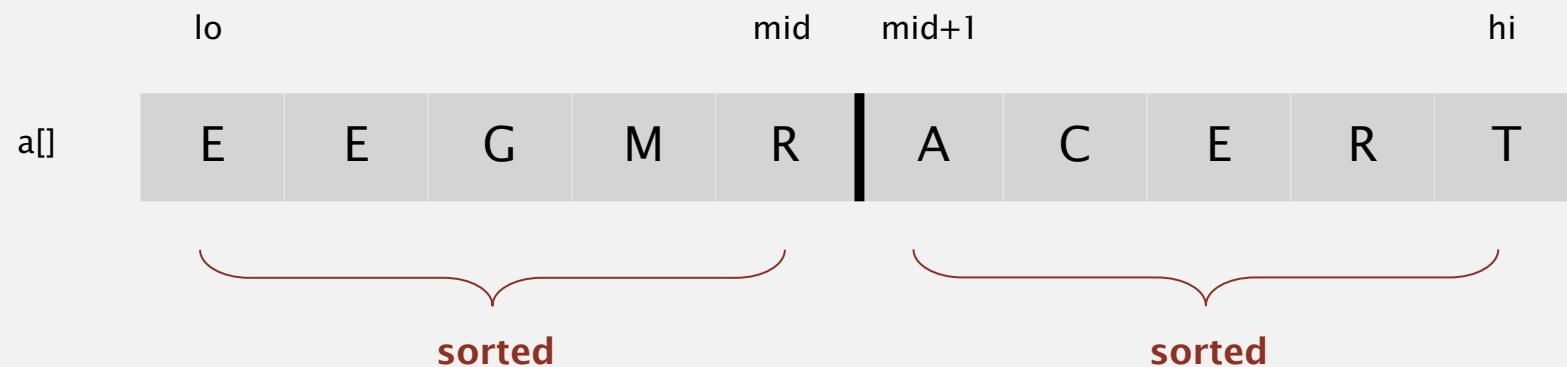
input	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E	
sort left half	E	E	G	M	M	O	R	R	S	T	E	X	A	M	P	L	E
sort right half	E	E	G	M	M	O	R	R	S	A	E	E	L	M	P	T	X
merge results	A	E	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Mergesort overview



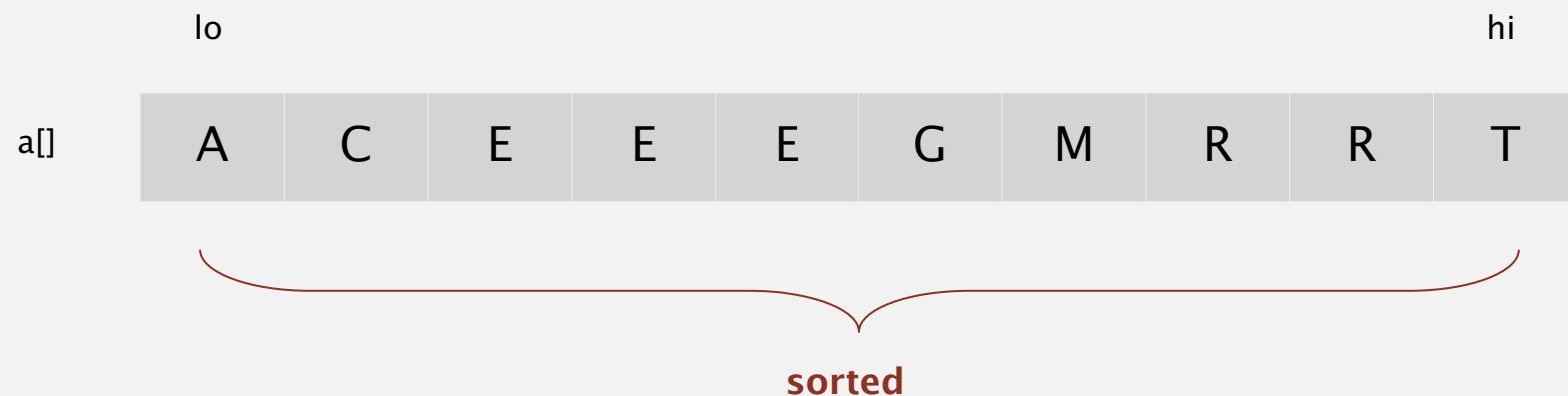
Abstract in-place merge demo

Goal. Given two sorted subarrays $a[lo]$ to $a[mid]$ and $a[mid+1]$ to $a[hi]$, replace with sorted subarray $a[lo]$ to $a[hi]$.



Abstract in-place merge demo

Goal. Given two sorted subarrays $a[lo]$ to $a[mid]$ and $a[mid+1]$ to $a[hi]$, replace with sorted subarray $a[lo]$ to $a[hi]$.



Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    assert isSorted(a, lo, mid);      // precondition: a[lo..mid] sorted
    assert isSorted(a, mid+1, hi);   // precondition: a[mid+1..hi] sorted

    for (int k = lo; k <= hi; k++)                                copy
        aux[k] = a[k];

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)                                merge
    {
        if      (i > mid)          a[k] = aux[j++];
        else if (j > hi)          a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else                      a[k] = aux[i++];
    }

    assert isSorted(a, lo, hi);      // postcondition: a[lo..hi] sorted
}
```



Assertions

Assertion. Statement to test assumptions about your program.

- Helps detect logic bugs.
- Documents code.

Java assert statement. Throws exception unless boolean condition is true.

```
assert isSorted(a, lo, hi);
```

Can enable or disable at runtime. ⇒ No cost in production code.

```
java -ea MyProgram    // enable assertions  
java -da MyProgram    // disable assertions (default)
```

Best practices. Use assertions to check internal invariants;
assume assertions will be disabled in production code. ←

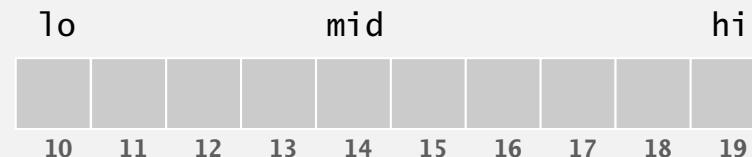
do not use for external
argument checking

Mergesort: Java implementation

```
public class Merge
{
    private static void merge(...)

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid+1, hi);
        merge(a, aux, lo, mid, hi);
    }

    public static void sort(Comparable[] a)
    {
        aux = new Comparable[a.length];
        sort(a, aux, 0, a.length - 1);
    }
}
```



Mergesort: trace

	a[]																		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
lo	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E			
hi	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E			
merge(a, 0, 0, 1)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E			
merge(a, 2, 2, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E			
merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E			
merge(a, 4, 4, 5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E			
merge(a, 6, 6, 7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E			
merge(a, 4, 5, 7)	E	G	M	R	E	S	O	R	S	T	E	X	A	M	P	L	E		
merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E			
merge(a, 8, 8, 9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E			
merge(a, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E			
merge(a, 8, 9, 11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E			
merge(a, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E			
merge(a, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L			
merge(a, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P			
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X			
merge(a, 0, 7, 15)	A	E	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X		

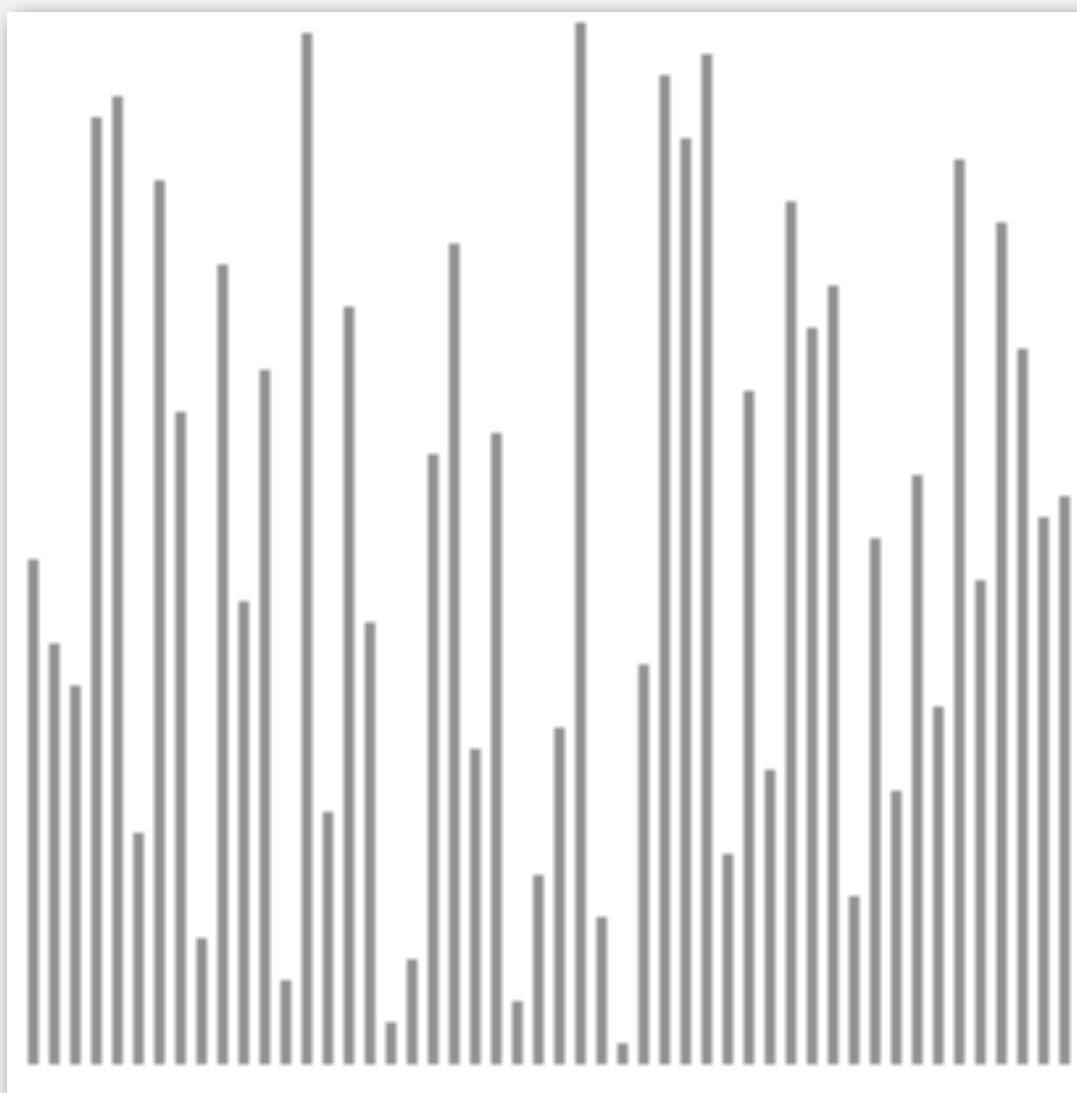
Trace of merge results for top-down mergesort



result after recursive call

Mergesort: animation

50 random items

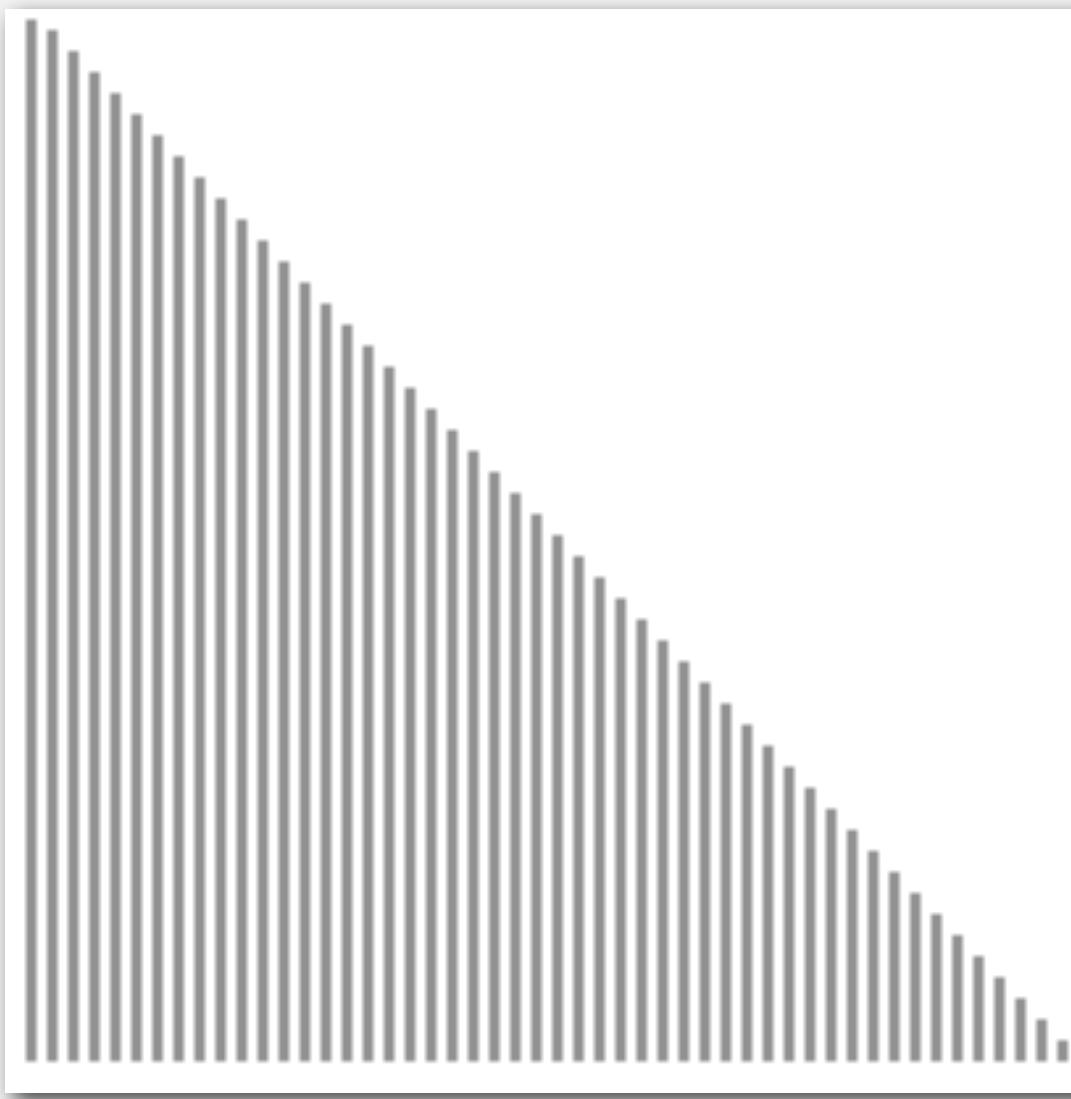


<http://www.sorting-algorithms.com/merge-sort>

- ▲ algorithm position
 - █ in order
 - █ current subarray
 - █ not in order

Mergesort: animation

50 reverse-sorted items



<http://www.sorting-algorithms.com/merge-sort>

- ▲ algorithm position
 - █ in order
 - █ current subarray
 - █ not in order

Mergesort: empirical analysis

Running time estimates:

- Laptop executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

computer	insertion sort (N^2)			mergesort ($N \log N$)		
	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

Bottom line. Good algorithms are better than supercomputers.

Mergesort: number of compares and array accesses

Proposition. Mergesort uses at most $N \lg N$ compares and $6N \lg N$ array accesses to sort any array of size N .

Pf sketch. The number of compares $C(N)$ and array accesses $A(N)$ to mergesort an array of size N satisfy the recurrences:

$$C(N) \leq C(\lceil N/2 \rceil) + C(\lfloor N/2 \rfloor) + N \quad \text{for } N > 1, \text{ with } C(1) = 0.$$



$$A(N) \leq A(\lceil N/2 \rceil) + A(\lfloor N/2 \rfloor) + 6N \quad \text{for } N > 1, \text{ with } A(1) = 0.$$

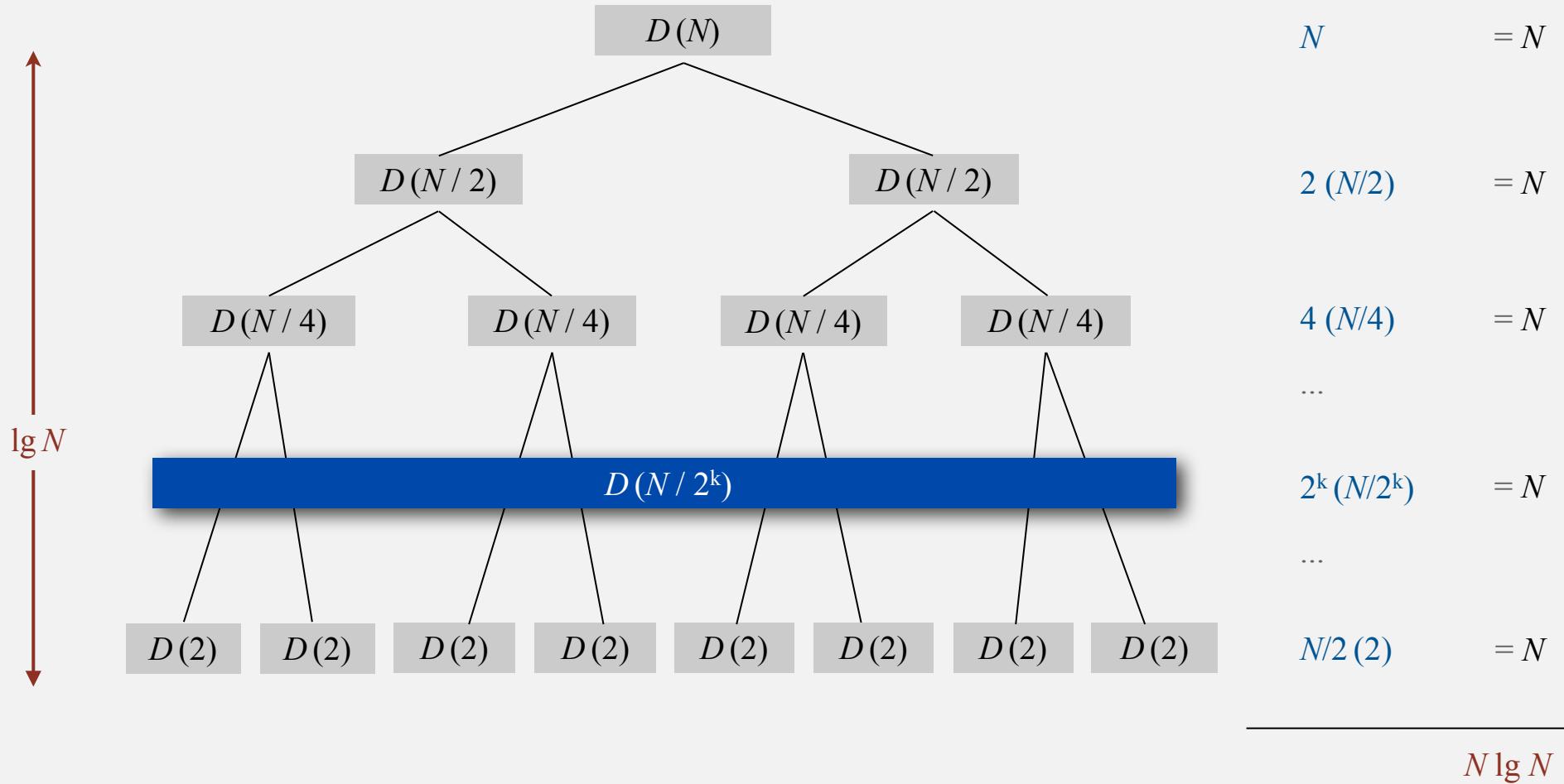
We solve the recurrence when N is a power of 2. ← result holds for all N

$$D(N) = 2D(N/2) + N, \text{ for } N > 1, \text{ with } D(1) = 0.$$

Divide-and-conquer recurrence: proof by picture

Proposition. If $D(N)$ satisfies $D(N) = 2 D(N / 2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 1. [assuming N is a power of 2]



$$N \lg N$$

Divide-and-conquer recurrence: proof by expansion

Proposition. If $D(N)$ satisfies $D(N) = 2D(N/2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 2. [assuming N is a power of 2]

$$D(N) = 2D(N/2) + N$$

given

$$D(N)/N = 2D(N/2)/N + 1$$

divide both sides by N

$$= D(N/2)/(N/2) + 1$$

algebra

$$= D(N/4)/(N/4) + 1 + 1$$

apply to first term

$$= D(N/8)/(N/8) + 1 + 1 + 1$$

apply to first term again

...

$$= D(N/N)/(N/N) + 1 + 1 + \dots + 1$$

stop applying, $D(1) = 0$

$$= \lg N$$

Divide-and-conquer recurrence: proof by induction

Proposition. If $D(N)$ satisfies $D(N) = 2 D(N / 2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 3. [assuming N is a power of 2]

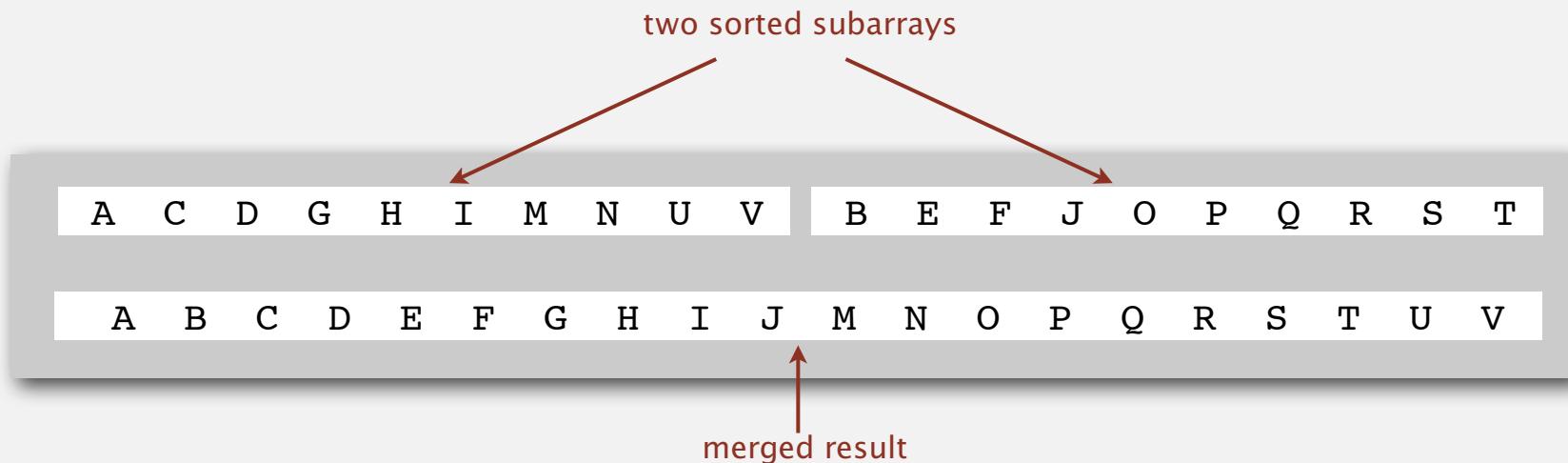
- Base case: $N = 1$.
- Inductive hypothesis: $D(N) = N \lg N$.
- Goal: show that $D(2N) = (2N) \lg (2N)$.

$$\begin{aligned} D(2N) &= 2 D(N) + 2N && \text{given} \\ &= 2 N \lg N + 2N && \text{inductive hypothesis} \\ &= 2 N (\lg (2N) - 1) + 2N && \text{algebra} \\ &= 2 N \lg (2N) && \text{QED} \end{aligned}$$

Mergesort analysis: memory

Proposition. Mergesort uses extra space proportional to N .

Pf. The array $\text{aux}[]$ needs to be of size N for the last merge.



Def. A sorting algorithm is **in-place** if it uses $\leq c \log N$ extra memory.

Ex. Insertion sort, selection sort, shellsort.

Challenge for the bored. In-place merge. [Kronrod, 1969]

Mergesort: practical improvements

Use insertion sort for small subarrays.

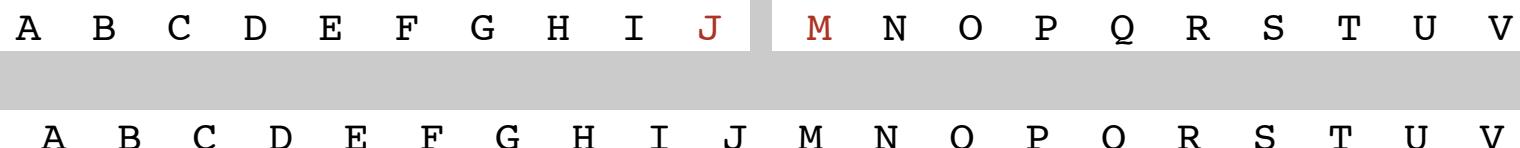
- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for ≈ 7 items.

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo + CUTOFF - 1) Insertion.sort(a, lo, hi);
    int mid = lo + (hi - lo) / 2;
    sort (a, aux, lo, mid);
    sort (a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

Mergesort: practical improvements

Stop if already sorted.

- Is biggest item in first half \leq smallest item in second half?
- Helps for partially-ordered arrays.



```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    if (!less(a[mid+1], a[mid])) return;
    merge(a, aux, lo, mid, hi);
}
```

Mergesort: practical improvements

Eliminate the copy to the auxiliary array. Save time (but not space) by switching the role of the input and auxiliary array in each recursive call.

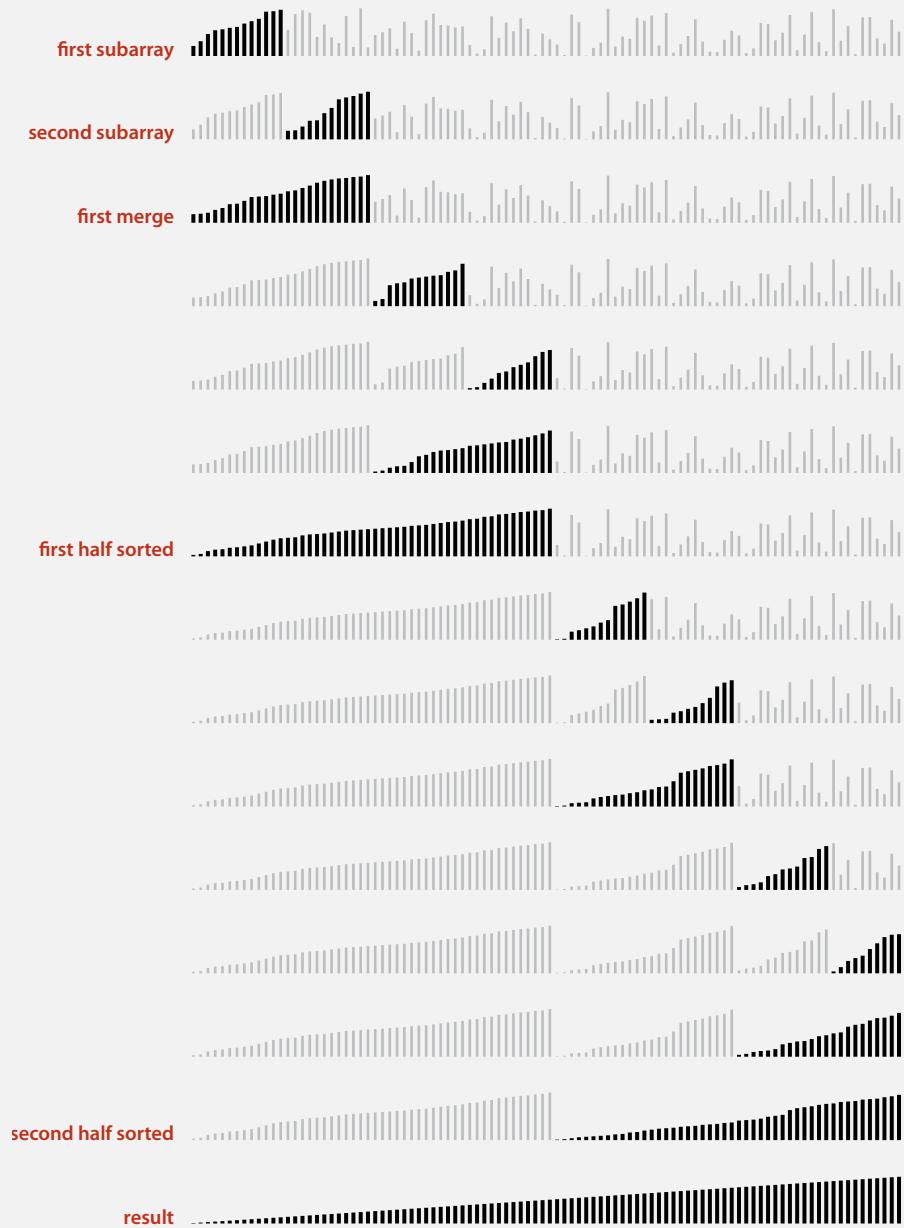
```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if (i > mid)          aux[k] = a[j++];
        else if (j > hi)       aux[k] = a[i++];
        else if (less(a[j], a[i])) aux[k] = a[j++]; ← merge from a[] to aux[]
        else                   aux[k] = a[i++];
    }
}

private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort (aux, a, lo, mid);
    sort (aux, a, mid+1, hi);
    merge(aux, a, lo, mid, hi);
}
```

↑

switch roles of aux[] and a[]

Mergesort: visualization



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Bottom-up mergesort

Basic plan.

- Pass through array, merging subarrays of size 1.
- Repeat for subarrays of size 2, 4, 8, 16,

a[i]																	
sz = 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
merge(a, 0, 0, 1)	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E	
merge(a, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E	
merge(a, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E	
merge(a, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L	
sz = 2	merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a, 4, 5, 7)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L	
merge(a, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L	
merge(a, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P	
sz = 4	merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X	
sz = 8	merge(a, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom line. No recursion needed!

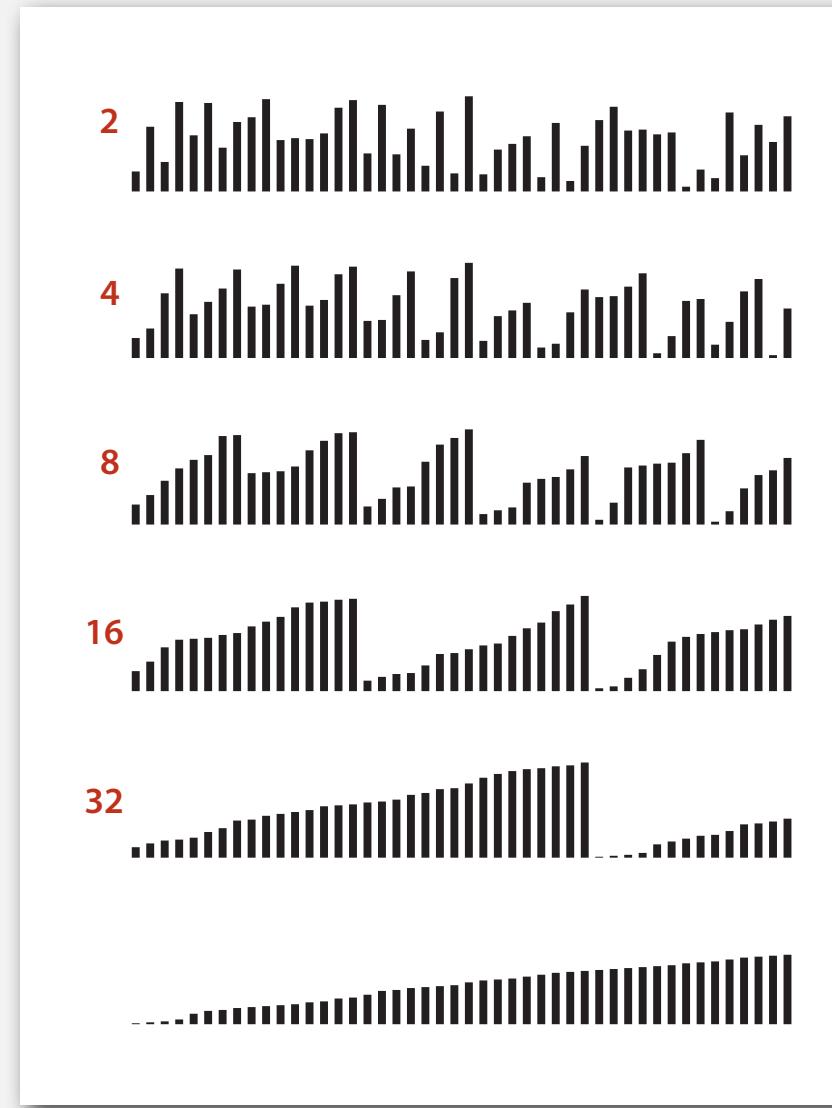
Bottom-up mergesort: Java implementation

```
public class MergeBU
{
    private static void merge(...)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
    }
}
```

Bottom line. Concise industrial-strength code, if you have the space.

Bottom-up mergesort: visual trace



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Complexity of sorting

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X .

Model of computation. Allowable operations.

Cost model. Operation count(s).

Upper bound. Cost guarantee provided by **some** algorithm for X .

Lower bound. Proven limit on cost guarantee of **all** algorithms for X .

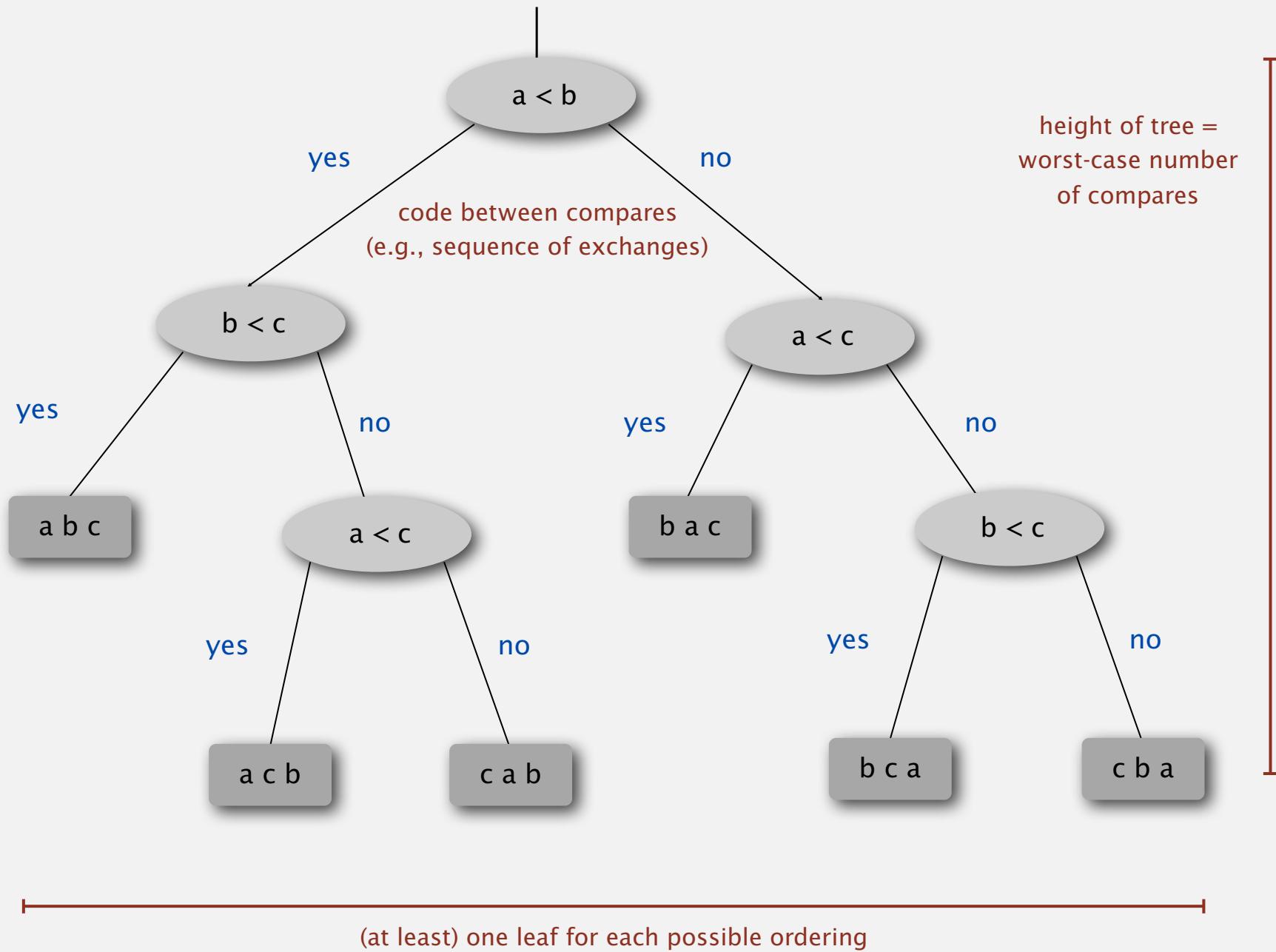
Optimal algorithm. Algorithm with best possible cost guarantee for X .

lower bound \sim upper bound

Example: sorting.

- Model of computation: decision tree. ← can access information only through compares
(e.g., Java Comparable framework)
- Cost model: # compares.
- Upper bound: $\sim N \lg N$ from mergesort.
- Lower bound: ?
- Optimal algorithm: ?

Decision tree (for 3 distinct items a, b, and c)

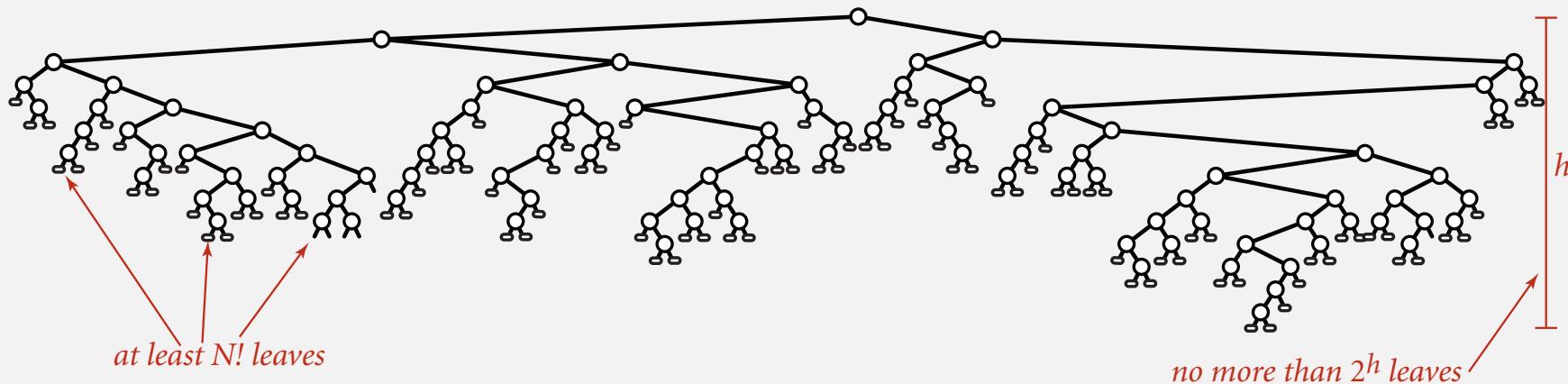


Compare-based lower bound for sorting

Proposition. Any compare-based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst-case.

Pf.

- Assume array consists of N distinct values a_1 through a_N .
- Worst case dictated by **height** h of decision tree.
- Binary tree of height h has at most 2^h leaves.
- $N!$ different orderings \Rightarrow at least $N!$ leaves.



Compare-based lower bound for sorting

Proposition. Any compare-based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst-case.

Pf.

- Assume array consists of N distinct values a_1 through a_N .
- Worst case dictated by **height** h of decision tree.
- Binary tree of height h has at most 2^h leaves.
- $N!$ different orderings \Rightarrow at least $N!$ leaves.

$$\begin{aligned} 2^h &\geq \# \text{leaves} \geq N! \\ \Rightarrow h &\geq \lg(N!) \sim N \lg N \end{aligned}$$

↑
Stirling's formula

Complexity of sorting

Model of computation. Allowable operations.

Cost model. Operation count(s).

Upper bound. Cost guarantee provided by some algorithm for X .

Lower bound. Proven limit on cost guarantee of all algorithms for X .

Optimal algorithm. Algorithm with best possible cost guarantee for X .

Example: sorting.

- Model of computation: decision tree.
- Cost model: # compares.
- Upper bound: $\sim N \lg N$ from mergesort.
- Lower bound: $\sim N \lg N$.
- Optimal algorithm = mergesort.

First goal of algorithm design: optimal algorithms.

Complexity results in context

Compares? Mergesort **is** optimal with respect to number compares.

Space? Mergesort **is not** optimal with respect to space usage.

Lessons. Use theory as a guide.

Ex. Don't design sorting algorithm that guarantees $\frac{1}{2} N \lg N$ compares.

Ex. Design sorting algorithm that is both time- and space-optimal?

Complexity results in context (continued)

Lower bound may not hold if the algorithm has information about:

- The initial order of the input.
- The distribution of key values.
- The representation of the keys.

Partially-ordered arrays. Depending on the initial order of the input,
we may not need $N \lg N$ compares.

insertion sort requires only $N-1$
compares if input array is sorted

Duplicate keys. Depending on the input distribution of duplicates,
we may not need $N \lg N$ compares.

stay tuned for 3-way quicksort

Digital properties of keys. We can use digit/character compares instead of
key compares for numbers and strings.

stay tuned for radix sorts

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Algorithms

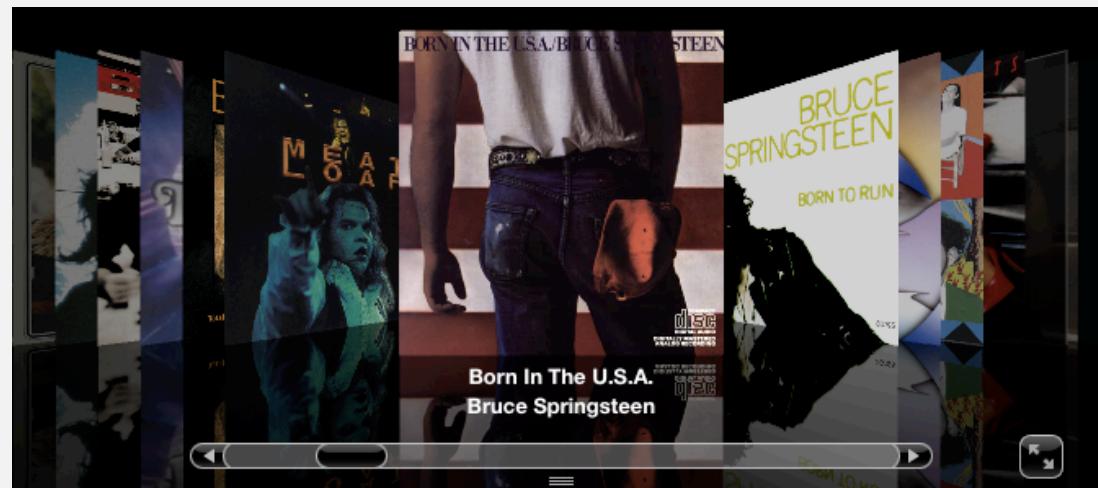
ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ **comparators**
- ▶ *stability*

Sort music library by artist name



	Name	Artist	Time	Album
12	<input checked="" type="checkbox"/> Let It Be	The Beatles	4:03	Let It Be
13	<input checked="" type="checkbox"/> Take My Breath Away	BERLIN	4:13	Top Gun – Soundtrack
14	<input checked="" type="checkbox"/> Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	<input checked="" type="checkbox"/> Dancing With Myself	Billy Idol	4:43	Don't Stop
16	<input checked="" type="checkbox"/> Rebel Yell	Billy Idol	4:49	Rebel Yell
17	<input checked="" type="checkbox"/> Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	<input checked="" type="checkbox"/> Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
19	<input checked="" type="checkbox"/> The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
20	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	<input checked="" type="checkbox"/> Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	<input checked="" type="checkbox"/> Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	<input checked="" type="checkbox"/> Hurricane	Bob Dylan	8:32	Desire
25	<input checked="" type="checkbox"/> The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	<input checked="" type="checkbox"/> Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
28	<input checked="" type="checkbox"/> Runaway	Bon Jovi	3:53	Cross Road
29	<input checked="" type="checkbox"/> Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	<input checked="" type="checkbox"/> Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	<input checked="" type="checkbox"/> Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	<input checked="" type="checkbox"/> Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	<input checked="" type="checkbox"/> Holding Out For A Hero	Bonny Tyler	5:49	Meat Loaf And Friends
34	<input checked="" type="checkbox"/> Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	<input checked="" type="checkbox"/> Thunder Road	Bruce Springsteen	4:51	Born To Run
36	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
37	<input checked="" type="checkbox"/> Jungleland	Bruce Springsteen	9:34	Born To Run
38	<input checked="" type="checkbox"/> Tug! Tug! Tug! (To Everything)	The Rude	3:57	Forrest Gump The Soundtrack (Disc 2)

Sort music library by song name

	Name	Artist	Time	Album
1	Alive	Pearl Jam	5:41	Ten
2	All Over The World	Pixies	5:27	Bossanova
3	All Through The Night	Cyndi Lauper	4:30	She's So Unusual
4	Allison Road	Gin Blossoms	3:19	New Miserable Experience
5	Ama, Ama, Ama Y Ensancha El ...	Extremoduro	2:34	Deltoya (1992)
6	And We Danced	Hooters	3:50	Nervous Night
7	As I Lay Me Down	Sophie B. Hawkins	4:09	Whaler
8	Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
9	Automatic Lover	Jay-Jay Johanson	4:19	Antenna
10	Baba O'Riley	The Who	5:01	Who's Better, Who's Best
11	Beautiful Life	Ace Of Base	3:40	The Bridge
12	Beds Of Roses	Bon Jovi	6:35	Cross Road
13	Black	Pearl Jam	5:44	Ten
14	Bleed American	Jimmy Eat World	3:04	Bleed American
15	Borderline	Madonna	4:00	The Immaculate Collection
16	Born To Run	Bruce Springsteen	4:30	Born To Run
17	Both Sides Of The Story	Phil Collins	6:43	Both Sides
18	Bouncing Around The Room	Phish	4:09	A Live One (Disc 1)
19	Boys Don't Cry	The Cure	2:35	Staring At The Sea: The Singles 1979–1985
20	Brat	Green Day	1:43	Insomniac
21	Breakdown	Deerheart	3:40	Deerheart
22	Bring Me To Life (Kevin Roen Mix)	Evanescence Vs. Pa...	9:48	
23	Californication	Red Hot Chili Pepp...	1:40	
24	Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
25	Can't Get You Out Of My Head	Kylie Minogue	3:50	Fever
26	Celebration	Kool & The Gang	3:45	Time Life Music Sounds Of The Seventies – C
27	Chaiwa Chaiwa	Sukhwinder Singh	5:11	Bombay Dreams

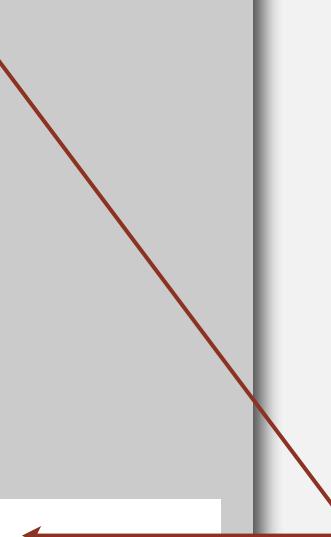
Comparable interface: review

Comparable interface: sort using a type's natural order.

```
public class Date implements Comparable<Date>
{
    private final int month, day, year;

    public Date(int m, int d, int y)
    {
        month = m;
        day   = d;
        year  = y;
    }

    ...
    public int compareTo(Date that)
    {
        if (this.year < that.year) return -1;
        if (this.year > that.year) return +1;
        if (this.month < that.month) return -1;
        if (this.month > that.month) return +1;
        if (this.day   < that.day)  return -1;
        if (this.day   > that.day)  return +1;
        return 0;
    }
}
```



natural order

Comparator interface

Comparator interface: sort using an alternate order.

```
public interface Comparator<Key>
    int compare(Key v, Key w)           compare keys v and w
```

Required property. Must be a total order.

Ex. Sort strings by:

- Natural order. Now is the time pre-1994 order for digraphs ch and ll and rr
- Case insensitive. is Now the time ↓
- Spanish. café cafetero cuarto churro nube ñoño
- British phone book. McKinley Mackintosh
- . . .

Comparator interface: system sort

To use with Java system sort:

- Create Comparator object.
- Pass as second argument to Arrays.sort().

```
String[] a;           uses natural order
...
Arrays.sort(a);      uses alternate order defined by
...
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);    Comparator<String> object
...
Arrays.sort(a, Collator.getInstance(new Locale("es")));
...
Arrays.sort(a, new BritishPhoneBookOrder());
...
```

The diagram illustrates the relationship between natural order and alternate order. It starts with a grey box containing code examples. A red arrow points from the first example, 'Arrays.sort(a);', to the text 'uses natural order'. Another red arrow points from the second example, 'Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);', to the text 'uses alternate order defined by Comparator<String> object'. A third red arrow points from the third example, 'Arrays.sort(a, new BritishPhoneBookOrder());', to the same alternate order text.

Bottom line. Decouples the definition of the data type from the definition of what it means to compare two objects of that type.

Comparator interface: using with our sorting libraries

To support comparators in our sort implementations:

- Use Object instead of Comparable.
- Pass Comparator to sort() and less() and use it in less().

insertion sort using a Comparator

```
public static void sort(Object[] a, Comparator comparator)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0 && less(comparator, a[j], a[j-1]); j--)
            exch(a, j, j-1);
}

private static boolean less(Comparator c, Object v, Object w)
{ return c.compare(v, w) < 0; }

private static void exch(Object[] a, int i, int j)
{ Object swap = a[i]; a[i] = a[j]; a[j] = swap; }
```

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the compare() method.

```
public class Student
{
    public static final Comparator<Student> BY_NAME      = new ByName();
    public static final Comparator<Student> BY_SECTION = new BySection();
    private final String name;
    private final int section;
    ...
    private static class ByName implements Comparator<Student>
    {
        public int compare(Student v, Student w)
        { return v.name.compareTo(w.name); }
    }

    private static class BySection implements Comparator<Student>
    {
        public int compare(Student v, Student w)
        { return v.section - w.section; }
    }
}
```

one Comparator for the class

this technique works here since no danger of overflow

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the compare() method.

`Arrays.sort(a, Student.BY_NAME);`

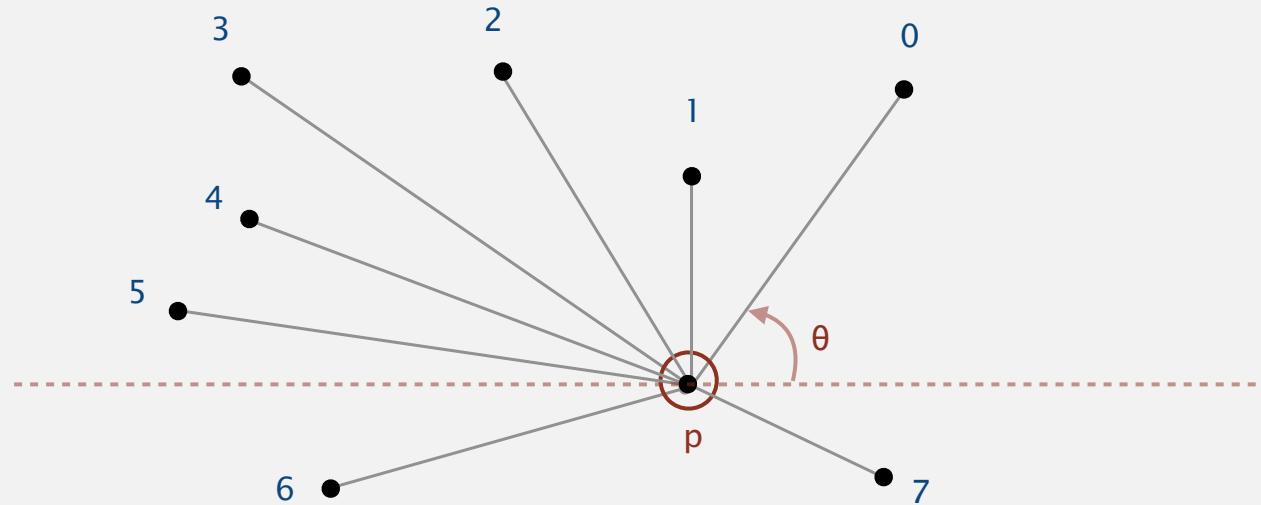
Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

`Arrays.sort(a, Student.BY_SECTION);`

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Andrews	3	A	664-480-0023	097 Little
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Kanaga	3	B	898-122-9643	22 Brown
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	766-093-9873	101 Brown

Polar order

Polar order. Given a point p , order points by polar angle they make with p .



```
Arrays.sort(points, p.POLAR_ORDER);
```

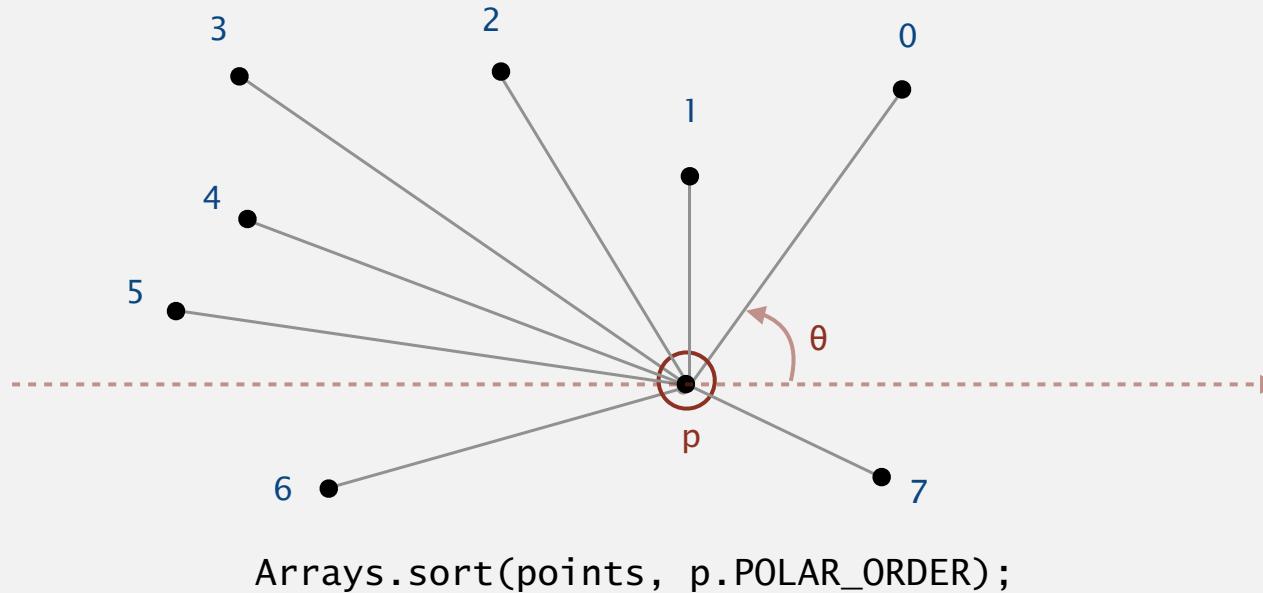
Application. Graham scan algorithm for convex hull. [see previous lecture]

High-school trig solution. Compute polar angle θ w.r.t. p using `atan2()`.

Drawback. Evaluating a trigonometric function is expensive.

Polar order

Polar order. Given a point p , order points by polar angle they make with p .



A ccw-based solution.

- If q_1 is above p and q_2 is below p , then q_1 makes smaller polar angle.
- If q_1 is below p and q_2 is above p , then q_1 makes larger polar angle.
- Otherwise, $ccw(p, q_1, q_2)$ identifies which of q_1 or q_2 makes larger angle.

Comparator interface: polar order

```
public class Point2D
{
    public final Comparator<Point2D> POLAR_ORDER = new PolarOrder();
    private final double x, y;
    ...
    private static int ccw(Point2D a, Point2D b, Point2D c)
    { /* as in previous lecture */ }

    private class PolarOrder implements Comparator<Point2D>
    {
        public int compare(Point2D q1, Point2D q2)
        {
            double dx1 = q1.x - x;
            double dy1 = q1.y - y;

            if (dy1 == 0 && dy2 == 0) { ... }
            else if (dy1 >= 0 && dy2 < 0) return -1;
            else if (dy2 >= 0 && dy1 < 0) return +1;
            else return -ccw(Point2D.this, q1, q2);
        }
    }
}
```

one Comparator for each point (not static)

p, q1, q2 horizontal

q1 above p; q2 below p

q1 below p; q2 above p

both above or below p

to access invoking point from within inner class

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ **comparators**
- ▶ *stability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ ***stability***

Stability

A typical application. First, sort by name; **then** sort by section.

`Selection.sort(a, Student.BY_NAME);`

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

`Selection.sort(a, Student.BY_SECTION);`

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Andrews	3	A	664-480-0023	097 Little
Kanaga	3	B	898-122-9643	22 Brown
Gazsi	4	B	766-093-9873	101 Brown
Battle	4	C	874-088-1212	121 Whitman

@#%&@! Students in section 3 no longer sorted by name.

A **stable** sort preserves the relative order of items with equal keys.

Stability

Q. Which sorts are stable?

A. Insertion sort and mergesort (but not selection sort or shellsort).

sorted by time	sorted by location (not stable)	sorted by location (stable)
Chicago 09:00:00	Chicago 09:25:52	Chicago 09:00:00
Phoenix 09:00:03	Chicago 09:03:13	Chicago 09:00:59
Houston 09:00:13	Chicago 09:21:05	Chicago 09:03:13
Chicago 09:00:59	Chicago 09:19:46	Chicago 09:19:32
Houston 09:01:10	Chicago 09:19:32	Chicago 09:19:46
Chicago 09:03:13	Chicago 09:00:00	Chicago 09:21:05
Seattle 09:10:11	Chicago 09:35:21	Chicago 09:25:52
Seattle 09:10:25	Chicago 09:00:59	Chicago 09:35:21
Phoenix 09:14:25	Houston 09:01:10	Houston 09:00:13
Chicago 09:19:32	Houston 09:00:13	Houston 09:01:10
Chicago 09:19:46	Phoenix 09:37:44	Phoenix 09:00:03
Chicago 09:21:05	Phoenix 09:00:03	Phoenix 09:14:25
Seattle 09:22:43	Phoenix 09:14:25	Phoenix 09:37:44
Seattle 09:22:54	Seattle 09:10:25	Seattle 09:10:11
Chicago 09:25:52	Seattle 09:36:14	Seattle 09:10:25
Chicago 09:35:21	Seattle 09:22:43	Seattle 09:22:43
Seattle 09:36:14	Seattle 09:10:11	Seattle 09:22:54
Phoenix 09:37:44	Seattle 09:22:54	Seattle 09:36:14

Note. Need to carefully check code ("less than" vs. "less than or equal to").

Stability: insertion sort

Proposition. Insertion sort is **stable**.

```
public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
                exch(a, j, j-1);
    }
}
```

i	j	0	1	2	3	4
0	0	B ₁	A ₁	A ₂	A ₃	B ₂
1	0	A ₁	B ₁	A ₂	A ₃	B ₂
2	1	A ₁	A ₂	B ₁	A ₃	B ₂
3	2	A ₁	A ₂	A ₃	B ₁	B ₂
4	4	A ₁	A ₂	A ₃	B ₁	B ₂
		A ₁	A ₂	A ₃	B ₁	B ₂

Pf. Equal items never move past each other.

Stability: selection sort

Proposition. Selection sort is **not** stable.

```
public class Selection
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
        {
            int min = i;
            for (int j = i+1; j < N; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }
}
```

i	min	0	1	2
0	2	B ₁	B ₂	A
1	1	A	B ₂	B ₁
2	2	A	B ₂	B ₁

Pf by counterexample. Long-distance exchange might move an item past some equal item.

Stability: shellsort

Proposition. Shellsort sort is **not** stable.

```
public class Shell
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        int h = 1;
        while (h < N/3) h = 3*h + 1;
        while (h >= 1)
        {
            for (int i = h; i < N; i++)
            {
                for (int j = i; j > h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }
            h = h/3;
        }
    }
}
```

h	0	1	2	3	4
	B ₁	B ₂	B ₃	B ₄	A ₁
4	A ₁	B ₂	B ₃	B ₄	B ₁
1	A ₁	B ₂	B ₃	B ₄	B ₁

Pf by counterexample. Long-distance exchanges.

Stability: mergesort

Proposition. Mergesort is **stable**.

```
public class Merge
{
    private static Comparable[] aux;
    private static void merge(Comparable[] a, int lo, int mid, int hi)
    { /* as before */ }

    private static void sort(Comparable[] a, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, lo, mid);
        sort(a, mid+1, hi);
        merge(a, lo, mid, hi);
    }

    public static void sort(Comparable[] a)
    { /* as before */ }
}
```

Pf. Suffices to verify that merge operation is stable.

Stability: mergesort

Proposition. Merge operation is stable.

```
private static void merge(...)  
{  
    for (int k = lo; k <= hi; k++)  
        aux[k] = a[k];  
  
    int i = lo, j = mid+1;  
    for (int k = lo; k <= hi; k++)  
    {  
        if (i > mid) a[k] = aux[j++];  
        else if (j > hi) a[k] = aux[i++];  
        else if (less(aux[j], aux[i])) a[k] = aux[j++];  
        else a[k] = aux[i++];  
    }  
}
```

0	1	2	3	4	5	6	7	8	9	10
A ₁	A ₂	A ₃	B	D	A ₄	A ₅	C	E	F	G

Pf. Takes from left subarray if equal keys.

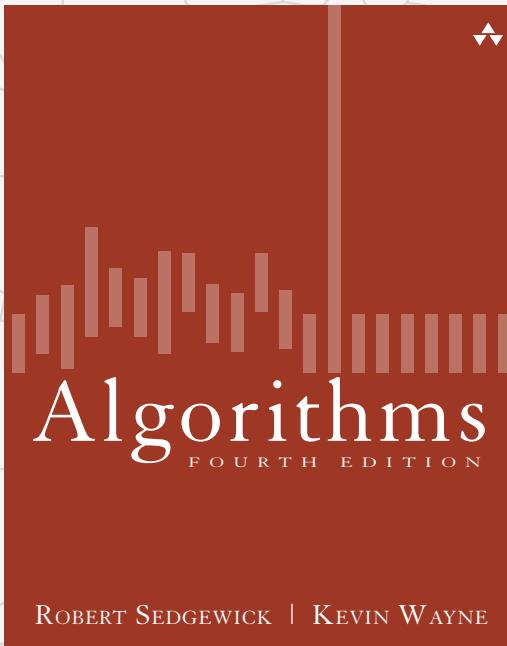
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ ***stability***



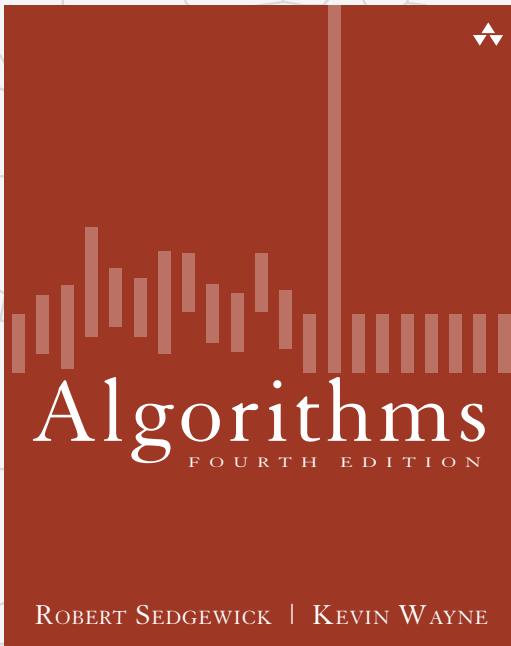
<http://algs4.cs.princeton.edu>

2.2 MERGESORT

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *sorting complexity*
- ▶ *comparators*
- ▶ *stability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

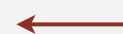
- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Two classic sorting algorithms

Critical components in the world's computational infrastructure.

- Full scientific understanding of their properties has enabled us to develop them into practical system sorts.
- Quicksort honored as one of top 10 algorithms of 20th century in science and engineering.

Mergesort.



last lecture

- Java sort for objects.
- Perl, C++ stable sort, Python stable sort, Firefox JavaScript, ...

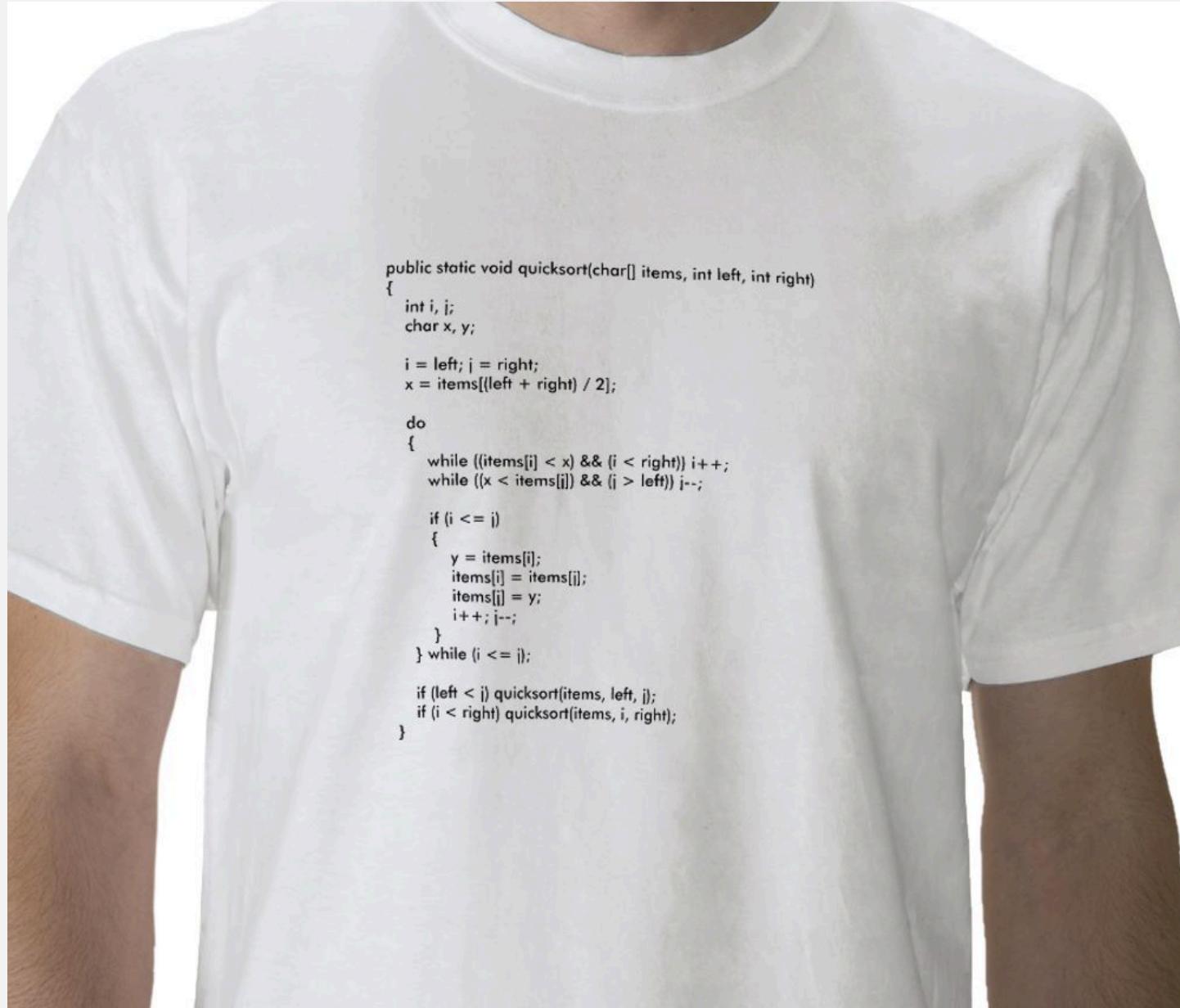
Quicksort.

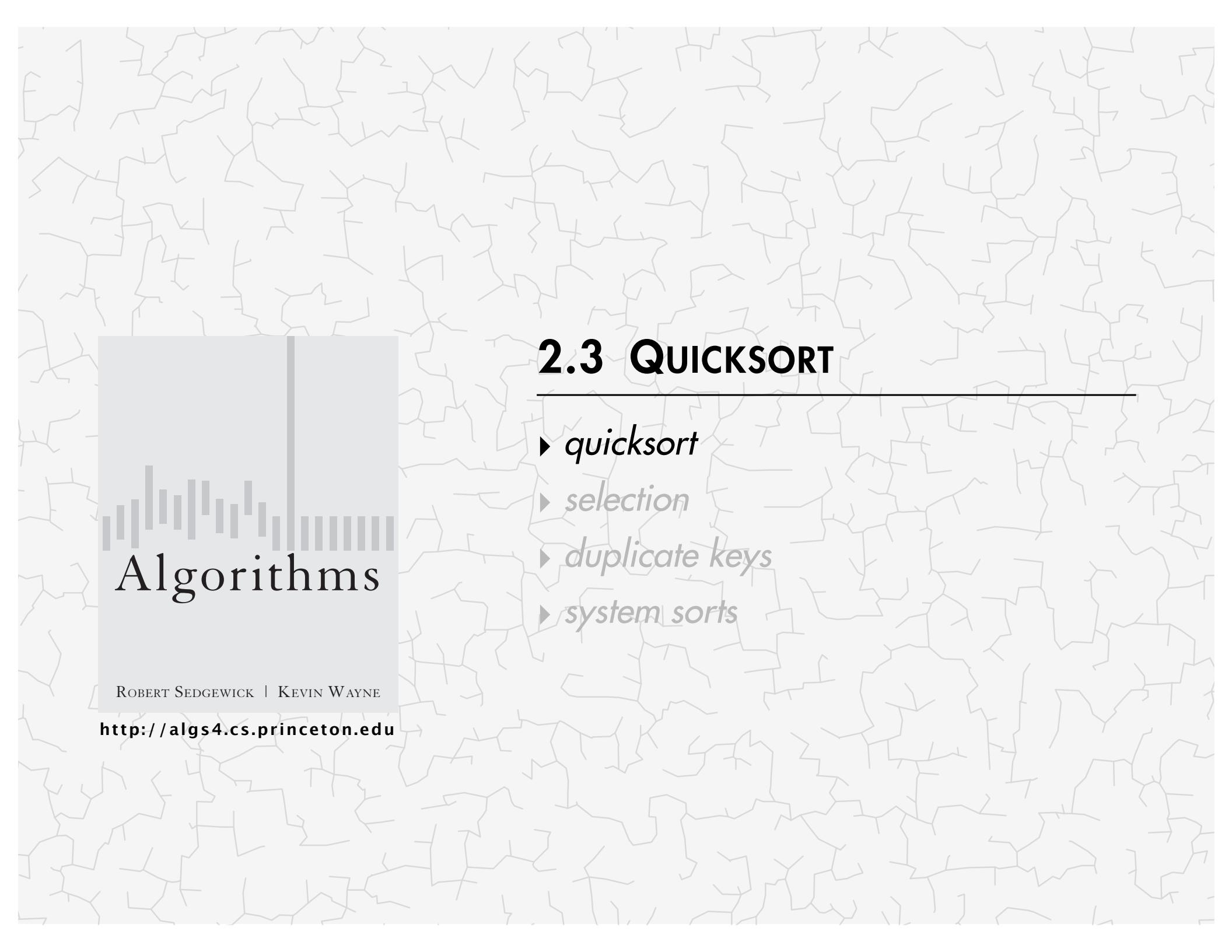


this lecture

- Java sort for primitive types.
- C qsort, Unix, Visual C++, Python, Matlab, Chrome JavaScript, ...

Quicksort t-shirt





Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

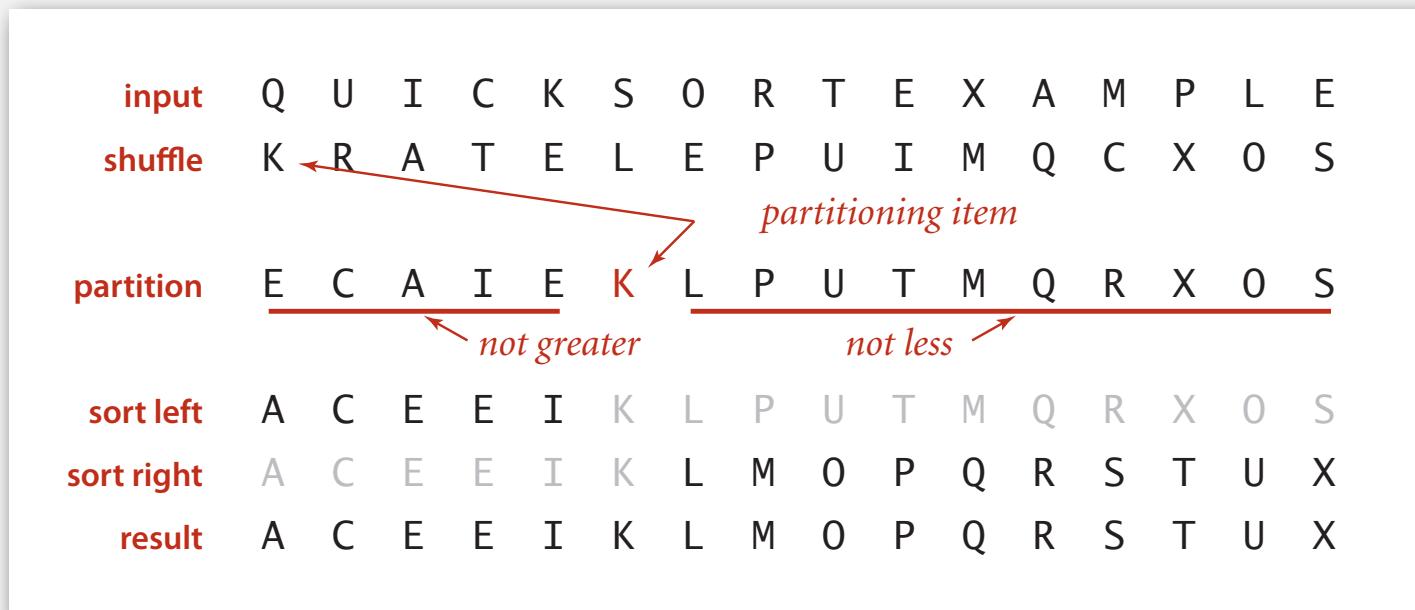
Quicksort

Basic plan.

- **Shuffle** the array.
- **Partition** so that, for some j
 - entry $a[j]$ is in place
 - no larger entry to the left of j
 - no smaller entry to the right of j
- **Sort** each piece recursively.



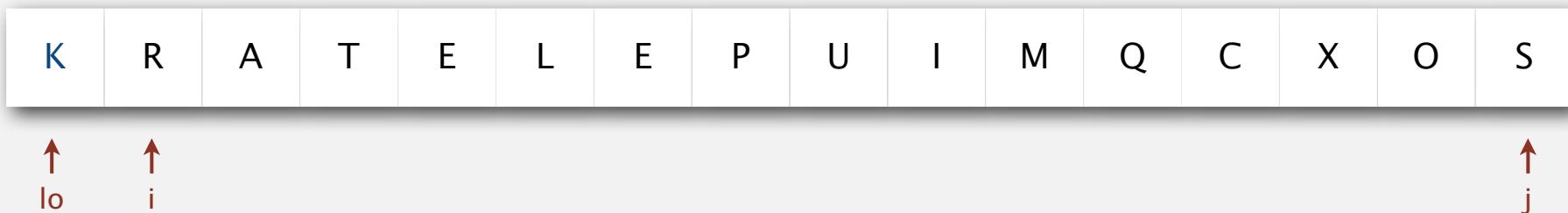
Sir Charles Antony Richard Hoare
1980 Turing Award



Quicksort partitioning demo

Repeat until i and j pointers cross.

- Scan i from left to right so long as $(a[i] < a[lo])$.
- Scan j from right to left so long as $(a[j] > a[lo])$.
- Exchange $a[i]$ with $a[j]$.



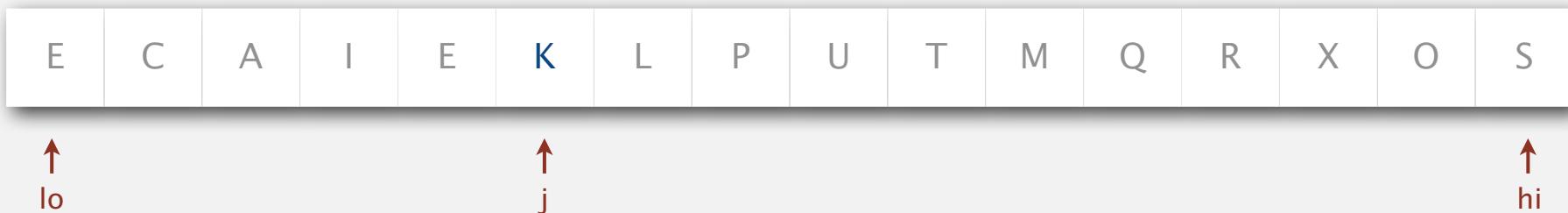
Quicksort partitioning demo

Repeat until i and j pointers cross.

- Scan i from left to right so long as $(a[i] < a[lo])$.
- Scan j from right to left so long as $(a[j] > a[lo])$.
- Exchange $a[i]$ with $a[j]$.

When pointers cross.

- Exchange $a[lo]$ with $a[j]$.



partitioned!

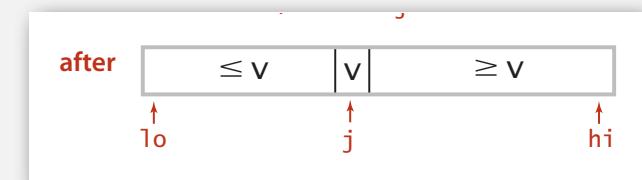
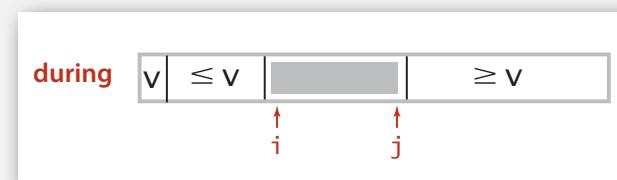
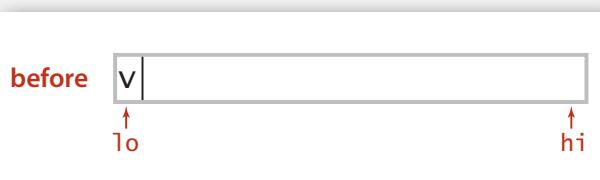
Quicksort: Java code for partitioning

```
private static int partition(Comparable[] a, int lo, int hi)
{
    int i = lo, j = hi+1;
    while (true)
    {
        while (less(a[++i], a[lo]))           find item on left to swap
            if (i == hi) break;

        while (less(a[lo], a[--j]))           find item on right to swap
            if (j == lo) break;

        if (i >= j) break;                  check if pointers cross
        exch(a, i, j);                   swap
    }

    exch(a, lo, j);                  swap with partitioning item
    return j;                        return index of item now known to be in place
}
```



Quicksort: Java implementation

```
public class Quick
{
    private static int partition(Comparable[] a, int lo, int hi)
    { /* see previous slide */ }

    public static void sort(Comparable[] a)
    {
        StdRandom.shuffle(a);
        sort(a, 0, a.length - 1);
    }

    private static void sort(Comparable[] a, int lo, int hi)
    {
        if (hi <= lo) return;
        int j = partition(a, lo, hi);
        sort(a, lo, j-1);
        sort(a, j+1, hi);
    }
}
```

← shuffle needed for performance guarantee
(stay tuned)

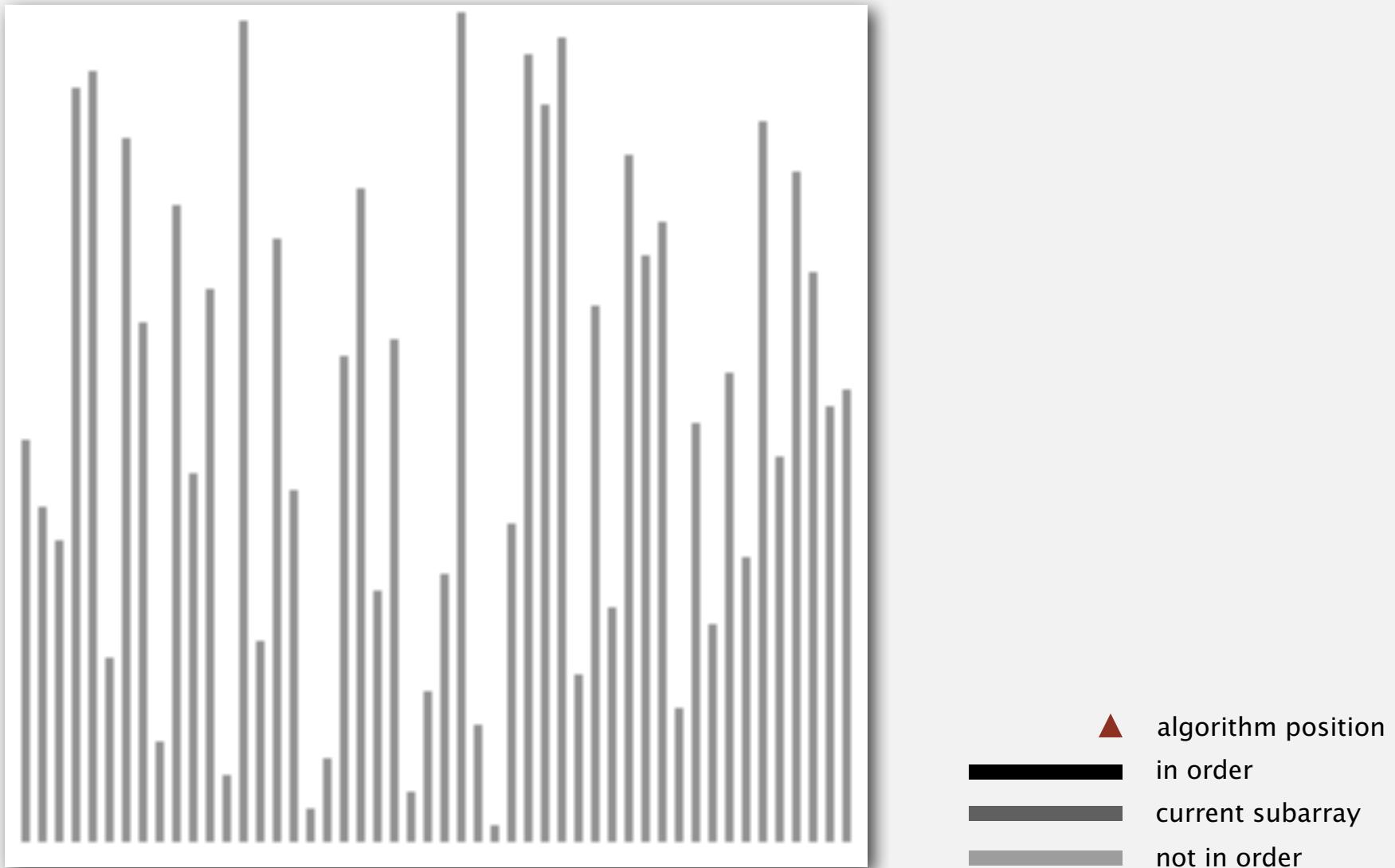
Quicksort trace

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
initial values			Q	U	I	C	K	S	O	R	T	E	X	A	M	P	L	E	
random shuffle			K	R	A	T	E	L	E	P	U	I	M	Q	C	X	0	S	
0	5	15	E	C	A	I	E	K	L	P	U	T	M	Q	R	X	0	S	
0	3	4	E	C	A	E	I	K	L	P	U	T	M	Q	R	X	0	S	
0	2	2	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	0	S	
0	0	1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	0	S	
1	1	1	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	0	S	
4	4	4	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	0	S	
6	6	15	A	C	E	E	I	K	L	P	U	T	M	Q	R	X	0	S	
7	9	15	A	C	E	E	I	K	L	M	O	P	T	Q	R	X	U	S	
7	7	8	A	C	E	E	I	K	L	M	M	O	P	T	Q	R	X	U	S
8	8	8	A	C	E	E	I	K	L	M	M	O	P	T	Q	R	X	U	S
10	13	15	A	C	E	E	I	K	L	M	M	O	P	S	Q	R	T	U	X
10	12	12	A	C	E	E	I	K	L	M	M	O	P	R	Q	S	T	U	X
10	11	11	A	C	E	E	I	K	L	M	M	O	P	Q	R	S	T	U	X
10	10	10	A	C	E	E	I	K	L	M	M	O	P	Q	R	S	T	U	X
14	14	15	A	C	E	E	I	K	L	M	M	O	P	Q	R	S	T	U	X
15	15	15	A	C	E	E	I	K	L	M	M	O	P	Q	R	S	T	U	X
result			A	C	E	E	I	K	L	M	O	P	Q	R	S	T	U	X	

Quicksort trace (array contents after each partition)

Quicksort animation

50 random items



<http://www.sorting-algorithms.com/quick-sort>

Quicksort: implementation details

Partitioning in-place. Using an extra array makes partitioning easier (and stable), but is not worth the cost.

Terminating the loop. Testing whether the pointers cross is a bit trickier than it might seem.

Staying in bounds. The $(j == lo)$ test is redundant (why?), but the $(i == hi)$ test is not.

Preserving randomness. Shuffling is needed for performance guarantee.

Equal keys. When duplicates are present, it is (counter-intuitively) better to stop on keys equal to the partitioning item's key.

Quicksort: empirical analysis

Running time estimates:

- Home PC executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

	insertion sort (N^2)			mergesort ($N \log N$)			quicksort ($N \log N$)		
computer	thousand	million	billion	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min	instant	0.6 sec	12 min
super	instant	1 second	1 week	instant	instant	instant	instant	instant	instant

Lesson 1. Good algorithms are better than supercomputers.

Lesson 2. Great algorithms are better than good ones.

Quicksort: best-case analysis

Best case. Number of compares is $\sim N \lg N$.

lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
initial values			H	A	C	B	F	E	G	D	L	I	K	J	N	M	O
random shuffle			H	A	C	B	F	E	G	D	L	I	K	J	N	M	O
0	7	14	D	A	C	B	F	E	G	H	L	I	K	J	N	M	O
0	3	6	B	A	C	D	F	E	G	H	L	I	K	J	N	M	O
0	1	2	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O
0	0	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O	
2	2	A	B	C	D	F	E	G	H	L	I	K	J	N	M	O	
4	5	6	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O
4	4	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O	
6	6	A	B	C	D	E	F	G	H	L	I	K	J	N	M	O	
8	11	14	A	B	C	D	E	F	G	H	J	I	K	L	N	M	O
8	9	10	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O
8	8	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O	
10	10	A	B	C	D	E	F	G	H	I	J	K	L	N	M	O	
12	13	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
12	12	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
14	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
A B C D E F G H I J K L M N O																	

Quicksort: worst-case analysis

Worst case. Number of compares is $\sim \frac{1}{2} N^2$.

			a[]														
lo	j	hi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0	0	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	1	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
2	2	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
3	3	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
4	4	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	5	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
6	6	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
7	7	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
8	8	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
9	9	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
10	10	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
11	11	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
12	12	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
13	13	14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
14		14	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Quicksort: average-case analysis

Proposition. The average number of compares C_N to quicksort an array of N distinct keys is $\sim 2N \ln N$ (and the number of exchanges is $\sim \frac{1}{3}N \ln N$).

Pf. C_N satisfies the recurrence $C_0 = C_1 = 0$ and for $N \geq 2$:

$$C_N = \underset{\text{partitioning}}{(N+1)} + \left(\frac{C_0 + C_{N-1}}{N} \right) + \left(\frac{C_1 + C_{N-2}}{N} \right) + \dots + \left(\frac{C_{N-1} + C_0}{N} \right)$$

- Multiply both sides by N and collect terms: partitioning probability

$$NC_N = N(N+1) + 2(C_0 + C_1 + \dots + C_{N-1})$$

- Subtract this from the same equation for $N - 1$:

$$NC_N - (N-1)C_{N-1} = 2N + 2C_{N-1}$$

- Rearrange terms and divide by $N(N+1)$:

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N} + \frac{2}{N+1}$$

Quicksort: average-case analysis

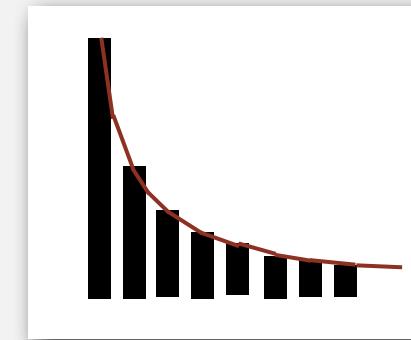
- Repeatedly apply above equation:

$$\begin{aligned}\frac{C_N}{N+1} &= \frac{C_{N-1}}{N} + \frac{2}{N+1} \\ &= \frac{C_{N-2}}{N-1} + \frac{2}{N} + \frac{2}{N+1} \quad \leftarrow \text{substitute previous equation} \\ &= \frac{C_{N-3}}{N-2} + \frac{2}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\ &= \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \dots + \frac{2}{N+1}\end{aligned}$$

← previous equation

- Approximate sum by an integral:

$$\begin{aligned}C_N &= 2(N+1) \left(\frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{N+1} \right) \\ &\sim 2(N+1) \int_3^{N+1} \frac{1}{x} dx\end{aligned}$$



- Finally, the desired result:

$$C_N \sim 2(N+1) \ln N \approx 1.39N \lg N$$

Quicksort: summary of performance characteristics

Worst case. Number of compares is quadratic.

- $N + (N - 1) + (N - 2) + \dots + 1 \sim \frac{1}{2} N^2$.
- More likely that your computer is struck by lightning bolt.

Average case. Number of compares is $\sim 1.39 N \lg N$.

- 39% more compares than mergesort.
- **But** faster than mergesort in practice because of less data movement.

Random shuffle.

- Probabilistic guarantee against worst case.
- Basis for math model that can be validated with experiments.

Caveat emptor. Many textbook implementations go **quadratic** if array

- Is sorted or reverse sorted.
- Has many duplicates (even if randomized!)

Quicksort properties

Proposition. Quicksort is an **in-place** sorting algorithm.

Pf.

- Partitioning: constant extra space.
- Depth of recursion: logarithmic extra space (with high probability).

can guarantee logarithmic depth by recurring on smaller subarray before larger subarray

Proposition. Quicksort is **not stable**.

Pf.

i	j	0	1	2	3
		B_1	C_1	C_2	A_1
1	3	B_1	C_1	C_2	A_1
1	3	B_1	A_1	C_2	C_1
0	1	A_1	B_1	C_2	C_1

Quicksort: practical improvements

Insertion sort small subarrays.

- Even quicksort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for ≈ 10 items.
- Note: could delay insertion sort until one pass at end.

```
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= lo + CUTOFF - 1)
    {
        Insertion.sort(a, lo, hi);
        return;
    }
    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort: practical improvements

Median of sample.

- Best choice of pivot item = median.
- Estimate true median by taking median of sample.
- Median-of-3 (random) items.



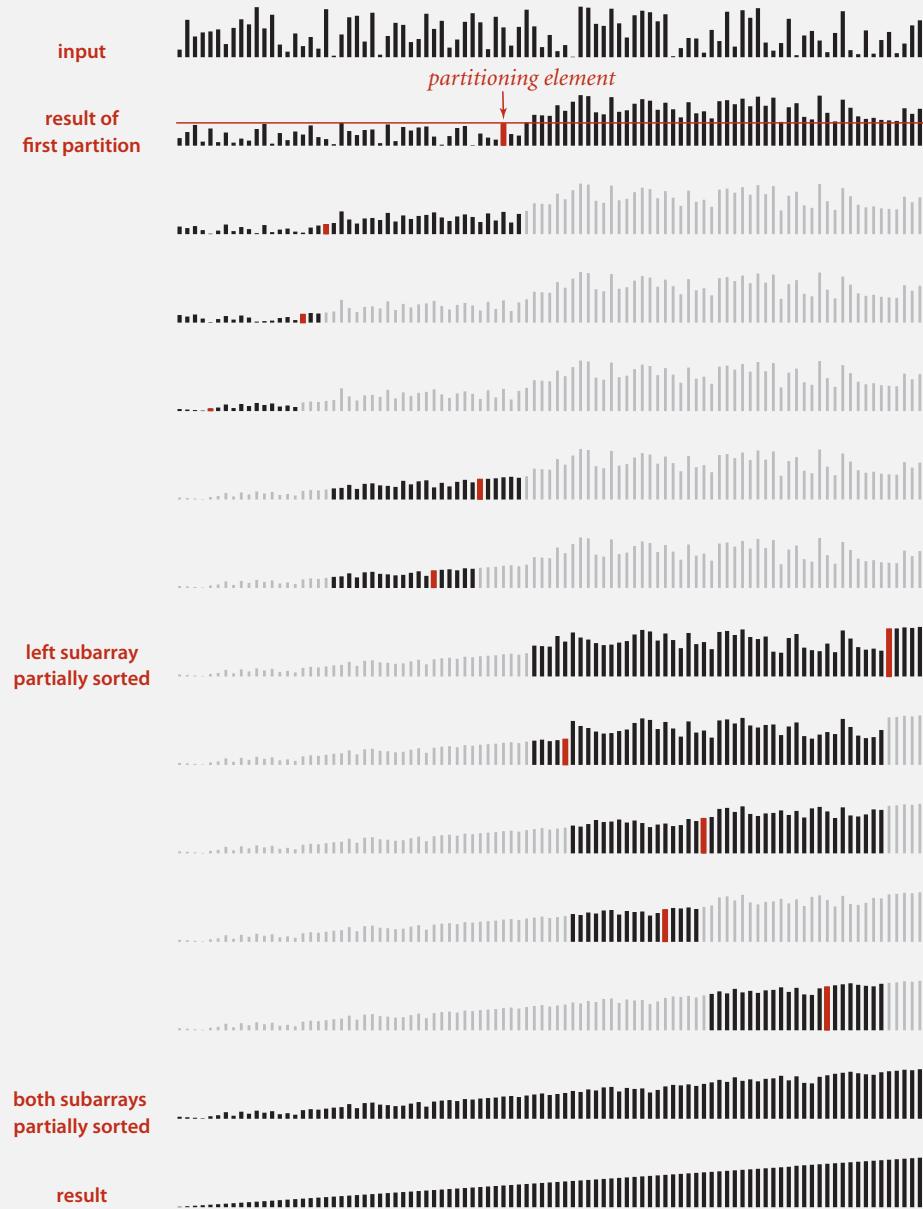
~ $12/7 N \ln N$ compares (slightly fewer)
~ $12/35 N \ln N$ exchanges (slightly more)

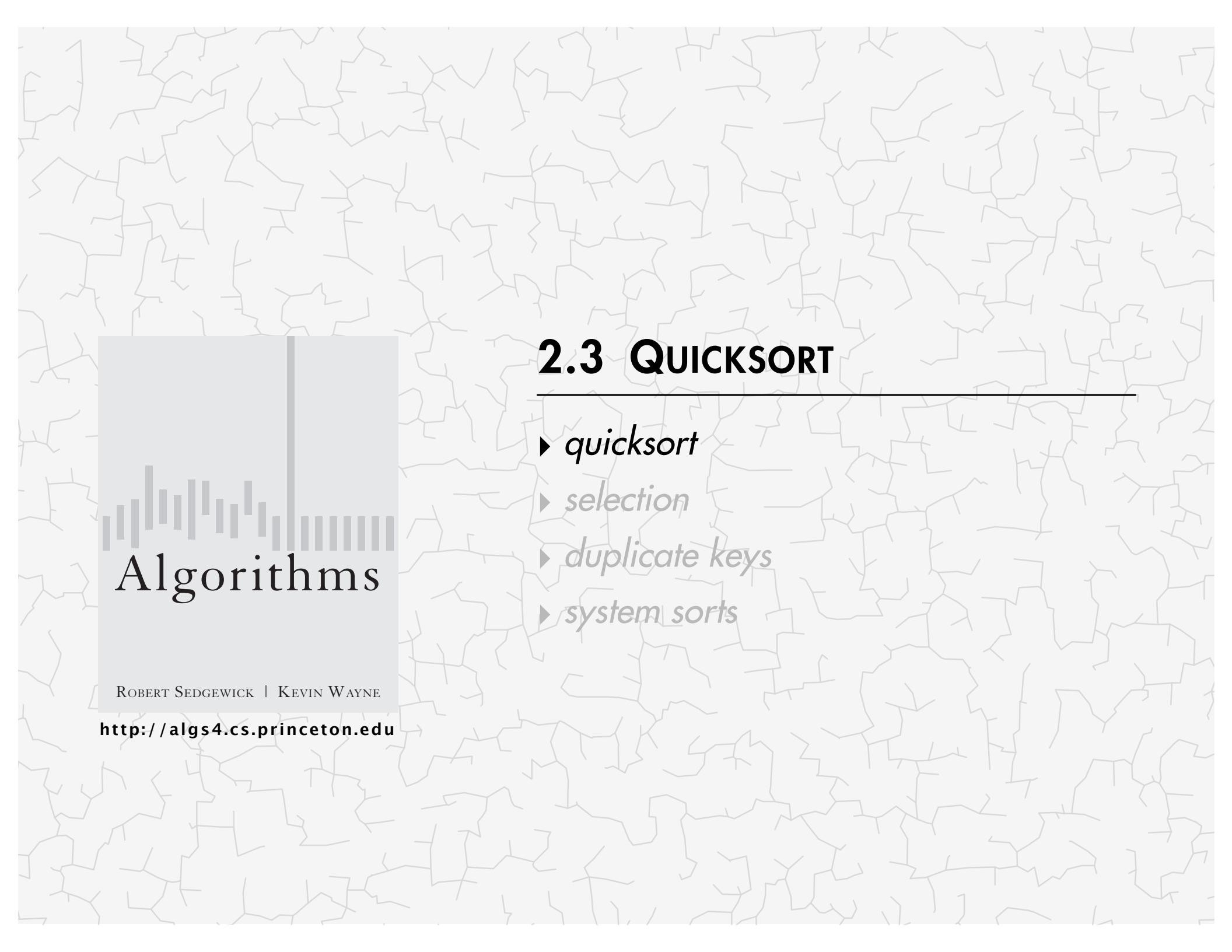
```
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= lo) return;

    int m = medianOf3(a, lo, lo + (hi - lo)/2, hi);
    swap(a, lo, m);

    int j = partition(a, lo, hi);
    sort(a, lo, j-1);
    sort(a, j+1, hi);
}
```

Quicksort with median-of-3 and cutoff to insertion sort: visualization





Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Selection

Goal. Given an array of N items, find the k^{th} largest.

Ex. Min ($k = 0$), max ($k = N - 1$), median ($k = N/2$).

Applications.

- Order statistics.
- Find the "top k ."

Use theory as a guide.

- Easy $N \log N$ upper bound. How?
- Easy N upper bound for $k = 1, 2, 3$. How?
- Easy N lower bound. Why?

Which is true?

- $N \log N$ lower bound?  is selection as hard as sorting?
- N upper bound?  is there a linear-time algorithm for each k ?

Quick-select

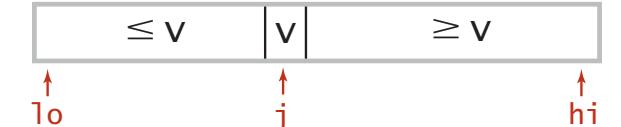
Partition array so that:

- Entry $a[j]$ is in place.
- No larger entry to the left of j .
- No smaller entry to the right of j .

Repeat in **one** subarray, depending on j ; finished when j equals k .

```
public static Comparable select(Comparable[] a, int k)
{
    StdRandom.shuffle(a);
    int lo = 0, hi = a.length - 1;
    while (hi > lo)
    {
        int j = partition(a, lo, hi);
        if      (j < k) lo = j + 1;
        else if (j > k) hi = j - 1;
        else            return a[k];
    }
    return a[k];
}
```

if $a[k]$ is here
set hi to $j-1$ if $a[k]$ is here
set lo to $j+1$



Quick-select: mathematical analysis

Proposition. Quick-select takes linear time on average.

Pf sketch.

- Intuitively, each partitioning step splits array approximately in half:
 $N + N/2 + N/4 + \dots + 1 \sim 2N$ compares.
- Formal analysis similar to quicksort analysis yields:

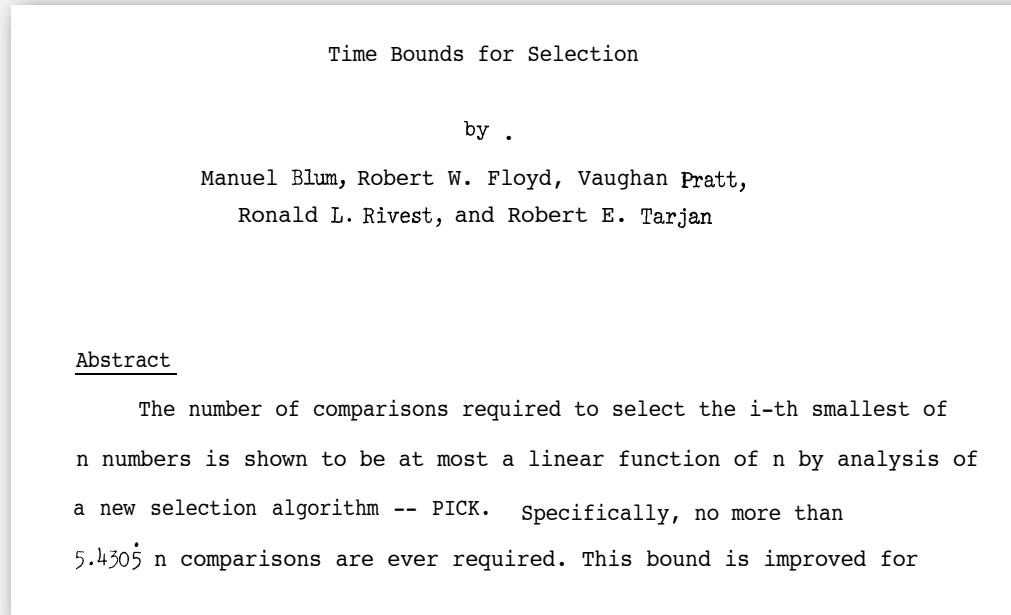
$$C_N = 2N + k \ln(N/k) + (N-k) \ln(N/(N-k))$$


 $(2 + 2 \ln 2) N$ to find the median

Remark. Quick-select uses $\sim \frac{1}{2} N^2$ compares in the worst case, but (as with quicksort) the random shuffle provides a probabilistic guarantee.

Theoretical context for selection

Proposition. [Blum, Floyd, Pratt, Rivest, Tarjan, 1973] Compare-based selection algorithm whose worst-case running time is linear.



Remark. But, constants are too high \Rightarrow not used in practice.

Use theory as a guide.

- Still worthwhile to seek practical linear-time (worst-case) algorithm.
- Until one is discovered, use quick-select if you don't need a full sort.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Duplicate keys

Often, purpose of sort is to bring items with equal keys together.

- Sort population by age.
- Remove duplicates from mailing list.
- Sort job applicants by college attended.

Typical characteristics of such applications.

- Huge array.
- Small number of key values.

```
Chicago 09:25:52
Chicago 09:03:13
Chicago 09:21:05
Chicago 09:19:46
Chicago 09:19:32
Chicago 09:00:00
Chicago 09:35:21
Chicago 09:00:59
Houston 09:01:10
Houston 09:00:13
Phoenix 09:37:44
Phoenix 09:00:03
Phoenix 09:14:25
Seattle 09:10:25
Seattle 09:36:14
Seattle 09:22:43
Seattle 09:10:11
Seattle 09:22:54
```

↑
key

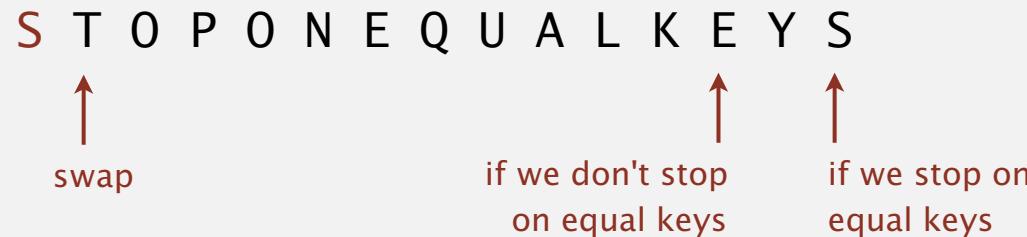
Duplicate keys

Mergesort with duplicate keys. Between $\frac{1}{2}N\lg N$ and $N\lg N$ compares.

Quicksort with duplicate keys.

- Algorithm goes **quadratic** unless partitioning stops on equal keys!
- 1990s C user found this defect in `qsort()`.

several textbook and system
implementation also have this defect



Duplicate keys: the problem

Mistake. Put all items equal to the partitioning item on one side.

Consequence. $\sim \frac{1}{2} N^2$ compares when all keys equal.

B A A B A B B **B** C C C A A A A A A A A A A **A**

Recommended. Stop scans on items equal to the partitioning item.

Consequence. $\sim N \lg N$ compares when all keys equal.

B A A B A **B** C C B C B A A A A A **A** A A A A A A

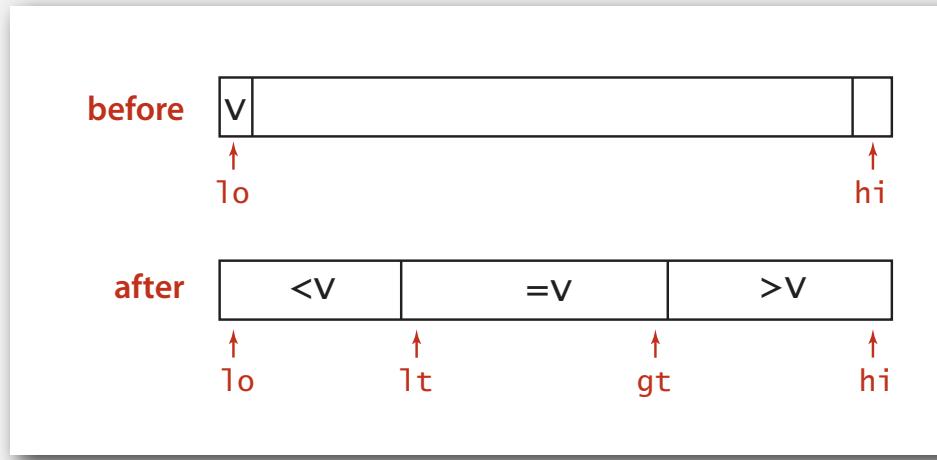
Desirable. Put all items equal to the partitioning item in place.

A A A **B** B B B C C C **A** A A A A A A A A A A A A

3-way partitioning

Goal. Partition array into 3 parts so that:

- Entries between lt and gt equal to partition item v .
- No larger entries to left of lt .
- No smaller entries to right of gt .



Dutch national flag problem. [Edsger Dijkstra]

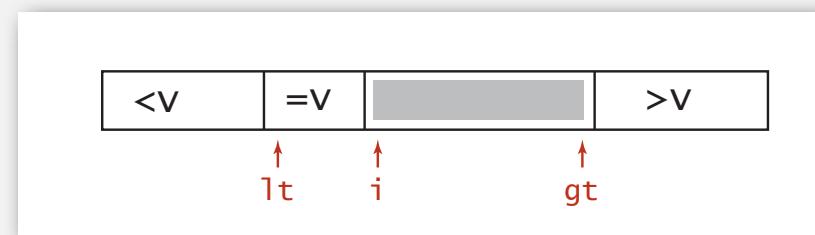
- Conventional wisdom until mid 1990s: not worth doing.
- New approach discovered when fixing mistake in C library `qsort()`.
- Now incorporated into `qsort()` and Java system `sort`.

Dijkstra 3-way partitioning demo

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$; increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$; decrement gt
 - $(a[i] == v)$: increment i

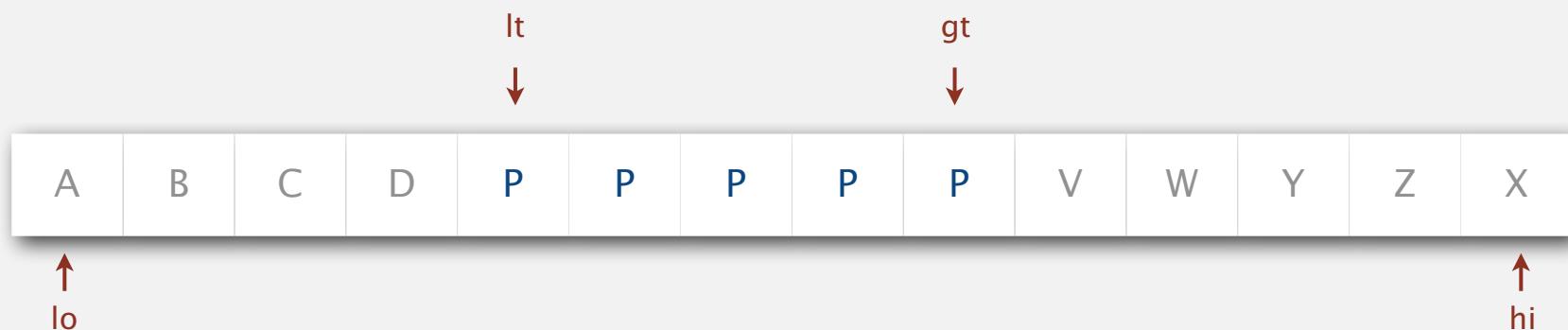


invariant

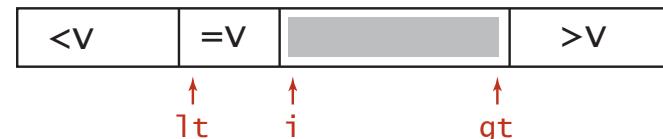


Dijkstra 3-way partitioning demo

- Let v be partitioning item $a[lo]$.
- Scan i from left to right.
 - $(a[i] < v)$: exchange $a[lt]$ with $a[i]$; increment both lt and i
 - $(a[i] > v)$: exchange $a[gt]$ with $a[i]$; decrement gt
 - $(a[i] == v)$: increment i



invariant



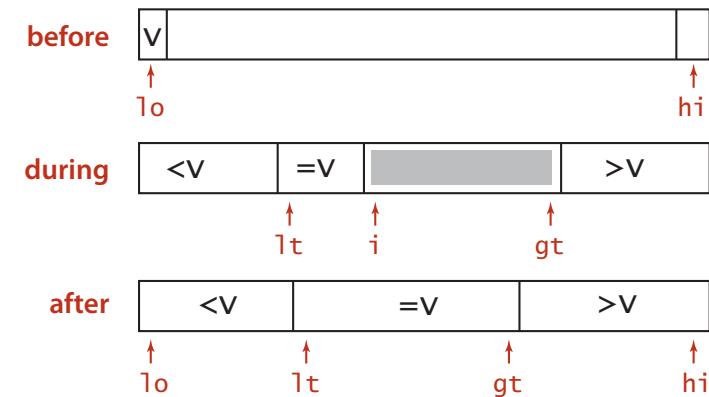
Dijkstra's 3-way partitioning: trace

lt	i	gt	a[]									
			0	1	2	3	4	5	6	7	8	9
0	0	11	R	B	W	W	R	W	B	R	R	W
0	1	11	R	B	W	W	R	W	B	R	R	W
1	2	11	B	R	W	W	R	W	B	R	R	W
1	2	10	B	R	R	W	R	W	B	R	R	W
1	3	10	B	R	R	W	R	W	B	R	R	W
1	3	9	B	R	R	B	R	W	B	R	R	W
2	4	9	B	B	R	R	R	W	B	R	R	W
2	5	9	B	B	R	R	R	W	B	R	R	W
2	5	8	B	B	R	R	R	W	B	R	R	W
2	5	7	B	B	R	R	R	R	B	R	R	W
2	6	7	B	B	R	R	R	R	B	R	R	W
3	7	7	B	B	B	R	R	R	R	R	R	W
3	8	7	B	B	B	R	R	R	R	R	R	W
3	8	7	B	B	B	R	R	R	R	R	R	W

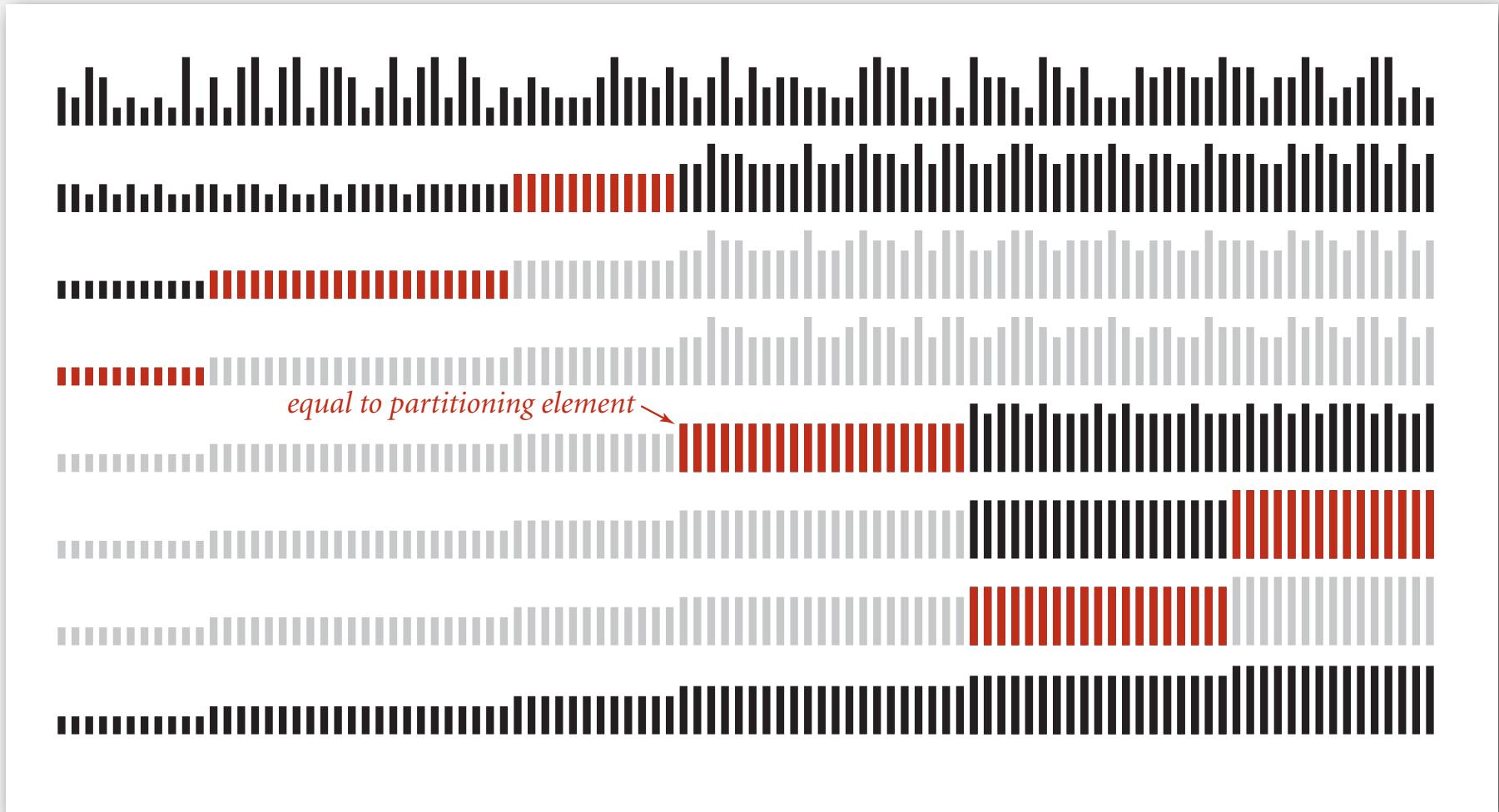
3-way partitioning trace (array contents after each loop iteration)

3-way quicksort: Java implementation

```
private static void sort(Comparable[] a, int lo, int hi)
{
    if (hi <= lo) return;
    int lt = lo, gt = hi;
    Comparable v = a[lo];
    int i = lo;
    while (i <= gt)
    {
        int cmp = a[i].compareTo(v);
        if      (cmp < 0) exch(a, lt++, i++);
        else if (cmp > 0) exch(a, i, gt--);
        else                i++;
    }
    sort(a, lo, lt - 1);
    sort(a, gt + 1, hi);
}
```



3-way quicksort: visual trace



Duplicate keys: lower bound

Sorting lower bound. If there are n distinct keys and the i^{th} one occurs x_i times, any compare-based sorting algorithm must use at least

$$\lg \left(\frac{N!}{x_1! x_2! \cdots x_n!} \right) \sim - \sum_{i=1}^n x_i \lg \frac{x_i}{N}$$

← *N lg N when all distinct;
linear when only a constant number of distinct keys*

compares in the worst case.

Proposition. [Sedgewick-Bentley, 1997]

proportional to lower bound

Quicksort with 3-way partitioning is **entropy-optimal**.

Pf. [beyond scope of course]

Bottom line. Randomized quicksort with 3-way partitioning reduces running time from linearithmic to linear in broad class of applications.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Sorting applications

Sorting algorithms are essential in a broad variety of applications:

- Sort a list of names.
- Organize an MP3 library. obvious applications
- Display Google PageRank results.
- List RSS feed in reverse chronological order.

- Find the median.
- Identify statistical outliers. problems become easy once items are in sorted order
- Binary search in a database.
- Find duplicates in a mailing list.

- Data compression.
- Computer graphics. non-obvious applications
- Computational biology.
- Load balancing on a parallel computer.

- . . .

Java system sorts

[Arrays.sort\(\)](#).

- Has different method for each primitive type.
- Has a method for data types that implement Comparable.
- Has a method that uses a Comparator.
- Uses tuned quicksort for primitive types; tuned mergesort for objects.

```
import java.util.Arrays;

public class StringSort
{
    public static void main(String[] args)
    {
        String[] a = StdIn.readStrings();
        Arrays.sort(a);
        for (int i = 0; i < N; i++)
            StdOut.println(a[i]);
    }
}
```

Q. Why use different algorithms for primitive and reference types?

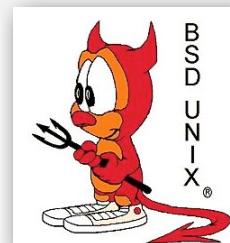
War story (C qsort function)

AT&T Bell Labs (1991). Allan Wilks and Rick Becker discovered that a `qsort()` call that should have taken a few minutes was consuming hours of CPU time.



At the time, almost all `qsort()` implementations based on those in:

- Version 7 Unix (1979): quadratic time to sort organ-pipe arrays.
- BSD Unix (1983): quadratic time to sort random arrays of 0s and 1s.



Engineering a system sort

Basic algorithm = quicksort.

- Cutoff to insertion sort for small subarrays.
- Partitioning scheme: 3-way partitioning. [like Dijkstra]
- Partitioning item.
 - small arrays: middle entry
 - medium arrays: median of 3
 - large arrays: Tukey's ninther [next slide]

Engineering a Sort Function

JON L. BENTLEY
M. DOUGLAS McILROY
AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, U.S.A.

SUMMARY

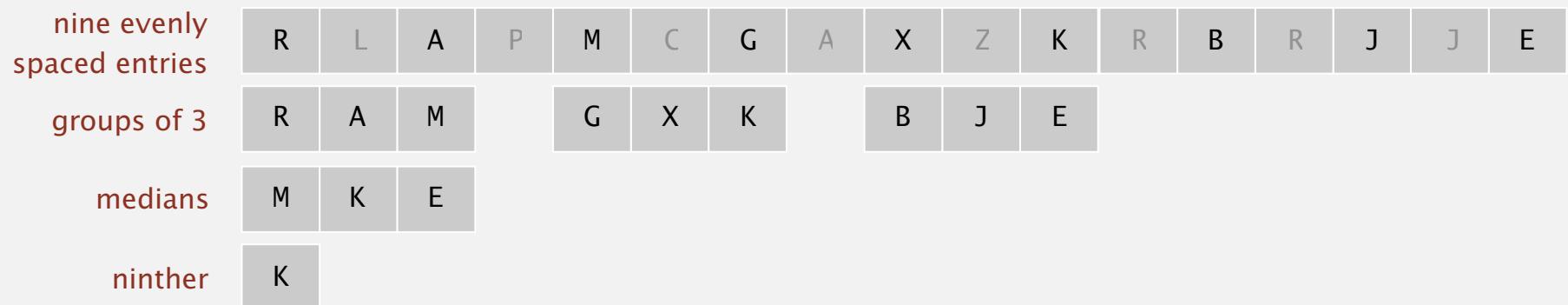
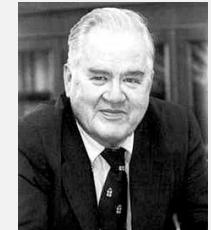
We recount the history of a new `qsort` function for a C library. Our function is clearer, faster and more robust than existing sorts. It chooses partitioning elements by a new sampling scheme; it partitions by a novel solution to Dijkstra's Dutch National Flag problem; and it swaps efficiently. Its behavior was assessed with timing and debugging testbeds, and with a program to certify performance. The design techniques apply in domains beyond sorting.

Now widely used. C, C++, Java,

Tukey's ninther

Tukey's ninther. Median of the median of 3 samples, each of 3 entries.

- Approximates the median of 9.
- Uses at most 12 compares.



Q. Why use Tukey's ninther?

A. Better partitioning than random shuffle and less costly.

Achilles heel in Bentley-McIlroy implementation (Java system sort)

Q. Based on all this research, Java's system sort is solid, **right?**

A. No: a killer input.

- Overflows function call stack in Java and crashes program.
- Would take quadratic time if it didn't crash first.

```
% more 250000.txt
```

```
0
```

```
218750
```

```
222662
```

```
11
```

```
166672
```

```
247070
```

```
83339
```

```
...
```



250,000 integers
between 0 and 250,000

```
% java IntegerSort 250000 < 250000.txt
```

```
Exception in thread "main"
```

```
java.lang.StackOverflowError
```

```
at java.util.Arrays.sort1(Arrays.java:562)
```

```
at java.util.Arrays.sort1(Arrays.java:606)
```

```
at java.util.Arrays.sort1(Arrays.java:608)
```

```
at java.util.Arrays.sort1(Arrays.java:608)
```

```
at java.util.Arrays.sort1(Arrays.java:608)
```

```
...
```

Java's sorting library crashes, even if
you give it as much stack space as Windows allows

System sort: Which algorithm to use?

Many sorting algorithms to choose from:

Internal sorts.

- Insertion sort, selection sort, bubblesort, shaker sort.
- Quicksort, mergesort, heapsort, samplesort, shellsort.
- Solitaire sort, red-black sort, splaysort, **Yaroslavskiy sort**, psort, ...

External sorts. Poly-phase mergesort, cascade-merge, oscillating sort.

String/radix sorts. Distribution, MSD, LSD, 3-way string quicksort.

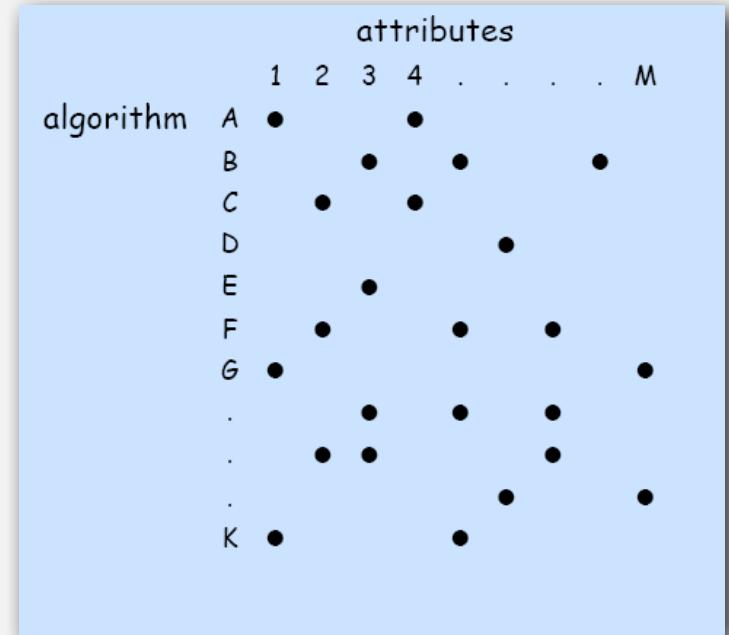
Parallel sorts.

- Bitonic sort, Batcher even-odd sort.
- Smooth sort, cube sort, column sort.
- GPUsort.

System sort: Which algorithm to use?

Applications have diverse attributes.

- Stable?
- Parallel?
- Deterministic?
- Keys all distinct?
- Multiple key types?
- Linked list or arrays?
- Large or small items?
- Is your array randomly ordered?
- Need guaranteed performance?



many more combinations of attributes than algorithms

Elementary sort may be method of choice for some combination.

Cannot cover **all** combinations of attributes.

Q. Is the system sort good enough?

A. Usually.

Sorting summary

	inplace?	stable?	worst	average	best	remarks
selection	✓		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	✓	✓	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	✓		?	?	N	tight code, subquadratic
merge		✓	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable
quick	✓		$N^2 / 2$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
3-way quick	✓		$N^2 / 2$	$2N \ln N$	N	improves quicksort in presence of duplicate keys
???	✓	✓	$N \lg N$	$N \lg N$	N	holy sorting grail

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

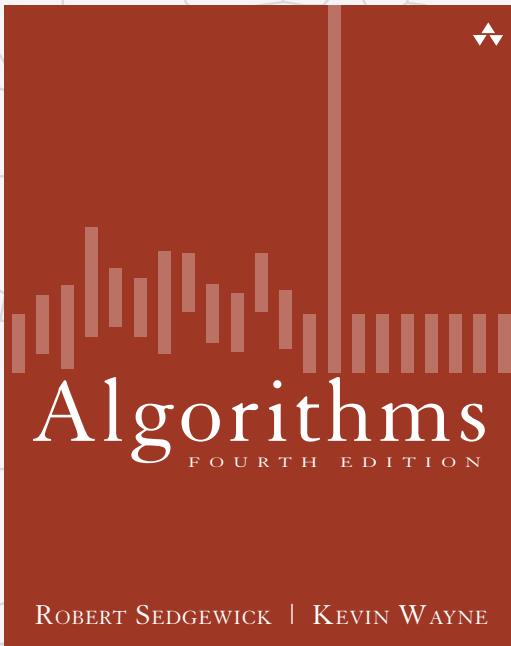
<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

2.3 QUICKSORT

- ▶ *quicksort*
- ▶ *selection*
- ▶ *duplicate keys*
- ▶ *system sorts*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Priority queue

Collections. Insert and delete items. Which item to delete?

Stack. Remove the item most recently added.

Queue. Remove the item least recently added.

Randomized queue. Remove a random item.

Priority queue. Remove the **largest** (or **smallest**) item.

<i>operation</i>	<i>argument</i>	<i>return value</i>
<i>insert</i>	P	
<i>insert</i>	Q	
<i>insert</i>	E	
<i>remove max</i>		Q
<i>insert</i>	X	
<i>insert</i>	A	
<i>insert</i>	M	
<i>remove max</i>		X
<i>insert</i>	P	
<i>insert</i>	L	
<i>insert</i>	E	
<i>remove max</i>		P

Priority queue API

Requirement. Generic items are Comparable.

```
public class MaxPQ<Key extends Comparable<Key>>
```

Key must be Comparable
(bounded type parameter)

MaxPQ()	<i>create an empty priority queue</i>
MaxPQ(Key[] a)	<i>create a priority queue with given keys</i>
void insert(Key v)	<i>insert a key into the priority queue</i>
Key delMax()	<i>return and remove the largest key</i>
boolean isEmpty()	<i>is the priority queue empty?</i>
Key max()	<i>return the largest key</i>
int size()	<i>number of entries in the priority queue</i>

Priority queue applications

- Event-driven simulation. [customers in a line, colliding particles]
- Numerical computation. [reducing roundoff error]
- Data compression. [Huffman codes]
- Graph searching. [Dijkstra's algorithm, Prim's algorithm]
- Number theory. [sum of powers]
- Artificial intelligence. [A* search]
- Statistics. [maintain largest M values in a sequence]
- Operating systems. [load balancing, interrupt handling]
- Discrete optimization. [bin packing, scheduling]
- Spam filtering. [Bayesian spam filter]

Generalizes: stack, queue, randomized queue.

Priority queue client example

Challenge. Find the largest M items in a stream of N items.

- Fraud detection: isolate \$\$ transactions.
- File maintenance: find biggest files or directories.



N huge, M large

Constraint. Not enough memory to store N items.

```
% more tinyBatch.txt
Turing      6/17/1990   644.08
vonNeumann 3/26/2002   4121.85
Dijkstra    8/22/2007   2678.40
vonNeumann  1/11/1999   4409.74
Dijkstra    11/18/1995   837.42
Hoare       5/10/1993   3229.27
vonNeumann  2/12/1994   4732.35
Hoare       8/18/1992   4381.21
Turing      1/11/2002   66.10
Thompson    2/27/2000   4747.08
Turing      2/11/1991   2156.86
Hoare       8/12/2003   1025.70
vonNeumann  10/13/1993  2520.97
Dijkstra    9/10/2000   708.95
Turing      10/12/1993  3532.36
Hoare       2/10/2005   4050.20
```

```
% java TopM 5 < tinyBatch.txt
Thompson    2/27/2000   4747.08
vonNeumann  2/12/1994   4732.35
vonNeumann  1/11/1999   4409.74
Hoare       8/18/1992   4381.21
vonNeumann  3/26/2002   4121.85
```

sort key

Priority queue client example

Challenge. Find the largest M items in a stream of N items.

- Fraud detection: isolate \$\$ transactions.
- File maintenance: find biggest files or directories.



N huge, M large

Constraint. Not enough memory to store N items.

use a min-oriented pq

```
MinPQ<Transaction> pq = new MinPQ<Transaction>();  
while (StdIn.hasNextLine())  
{  
    String line = StdIn.readLine();  
    Transaction item = new Transaction(line);  
    pq.insert(item);  
    if (pq.size() > M) ← pq contains  
        pq.delMin();          largest M items  
}  
}
```

Transaction data
type is Comparable
(ordered by \$\$)

Priority queue client example

Challenge. Find the largest M items in a stream of N items.

order of growth of finding the largest M in a stream of N items

implementation	time	space
sort	$N \log N$	N
elementary PQ	$M N$	M
binary heap	$N \log M$	M
best in theory	N	M

Priority queue: unordered and ordered array implementation

<i>operation</i>	<i>argument</i>	<i>return value</i>	<i>size</i>	<i>contents (unordered)</i>	<i>contents (ordered)</i>
<i>insert</i>	P		1	P	P
<i>insert</i>	Q		2	P Q	P Q
<i>insert</i>	E		3	P Q E	E P Q
<i>remove max</i>		Q	2	P E	E P
<i>insert</i>	X		3	P E X	E P X
<i>insert</i>	A		4	P E X A	A E P X
<i>insert</i>	M		5	P E X A M	A E M P X
<i>remove max</i>		X	4	P E M A	A E M P
<i>insert</i>	P		5	P E M A P	A E M P P
<i>insert</i>	L		6	P E M A P L	A E L M P P
<i>insert</i>	E		7	P E M A P L E	A E E L M P P
<i>remove max</i>		P	6	E M A P L E	A E E L M P

A sequence of operations on a priority queue

Priority queue: unordered array implementation

```
public class UnorderedMaxPQ<Key extends Comparable<Key>>
{
    private Key[] pq;      // pq[i] = ith element on pq
    private int N;          // number of elements on pq

    public UnorderedMaxPQ(int capacity)
    {   pq = (Key[]) new Comparable[capacity];  }

    public boolean isEmpty()
    {   return N == 0; }

    public void insert(Key x)
    {   pq[N++] = x;  }

    public Key delMax()
    {
        int max = 0;
        for (int i = 1; i < N; i++)
            if (less(max, i)) max = i;
        exch(max, N-1);
        return pq[--N];
    }
}
```

no generic
array creation

less() and exch()
similar to sorting methods

null out entry
to prevent loitering

Priority queue elementary implementations

Challenge. Implement **all** operations efficiently.

order of growth of running time for priority queue with N items

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
goal	$\log N$	$\log N$	$\log N$

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

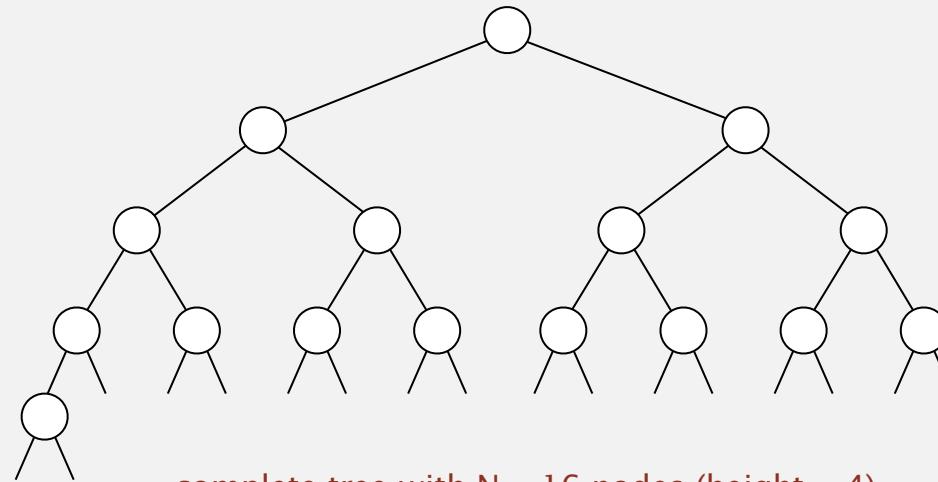
2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ ***binary heaps***
- ▶ *heapsort*
- ▶ *event-driven simulation*

Complete binary tree

Binary tree. Empty or node with links to left and right binary trees.

Complete tree. Perfectly balanced, except for bottom level.



Property. Height of complete tree with N nodes is $\lfloor \lg N \rfloor$.

Pf. Height only increases when N is a power of 2.

A complete binary tree in nature



Hyphaene Compressa - Doum Palm

© Shlomit Pinter

Binary heap representations

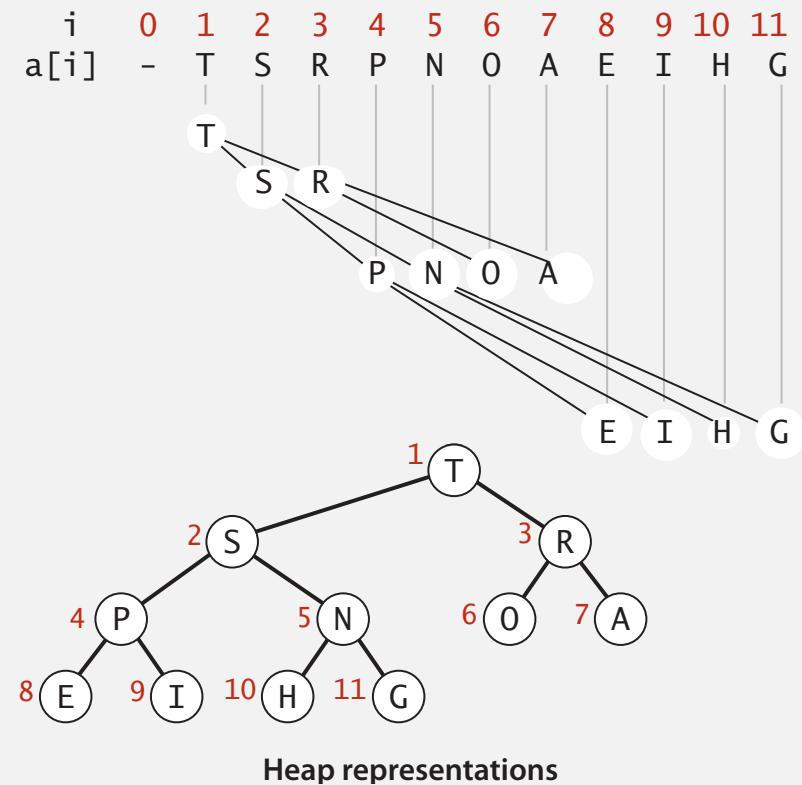
Binary heap. Array representation of a heap-ordered complete binary tree.

Heap-ordered binary tree.

- Keys in nodes.
- Parent's key no smaller than children's keys.

Array representation.

- Indices start at 1.
- Take nodes in **level** order.
- No explicit links needed!

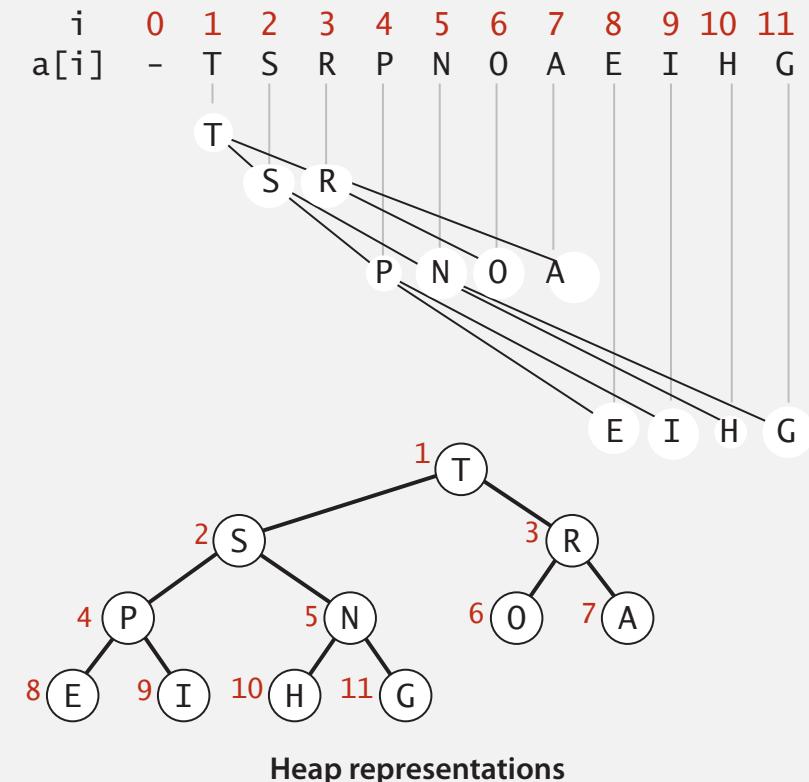


Binary heap properties

Proposition. Largest key is $a[1]$, which is root of binary tree.

Proposition. Can use array indices to move through tree.

- Parent of node at k is at $k/2$.
- Children of node at k are at $2k$ and $2k+1$.



Promotion in a heap

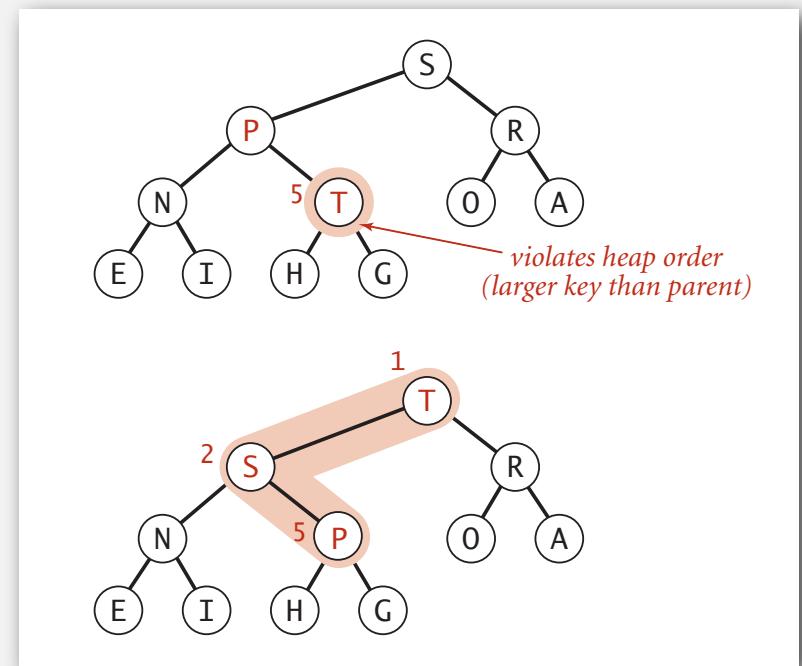
Scenario. Child's key becomes **larger** key than its parent's key.

To eliminate the violation:

- Exchange key in child with key in parent.
- Repeat until heap order restored.

```
private void swim(int k)
{
    while (k > 1 && less(k/2, k))
    {
        exch(k, k/2);
        k = k/2;
    }
}
```

parent of node at k is at k/2



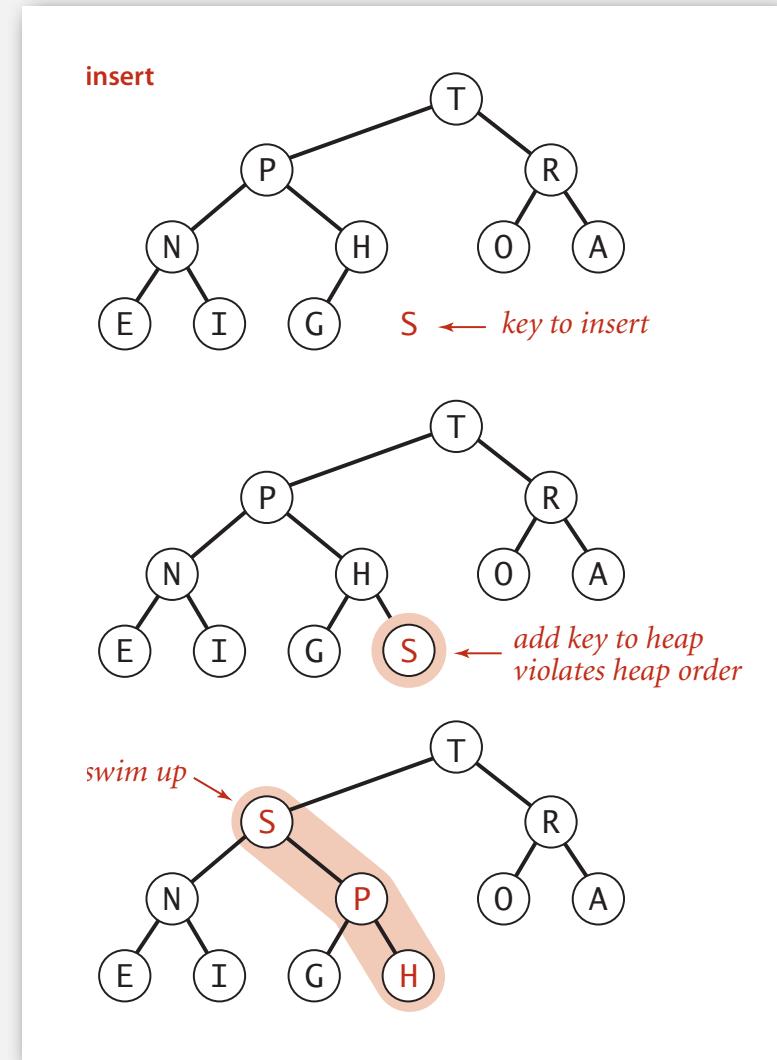
Peter principle. Node promoted to level of incompetence.

Insertion in a heap

Insert. Add node at end, then swim it up.

Cost. At most $1 + \lg N$ compares.

```
public void insert(Key x)
{
    pq[++N] = x;
    swim(N);
}
```



Demotion in a heap

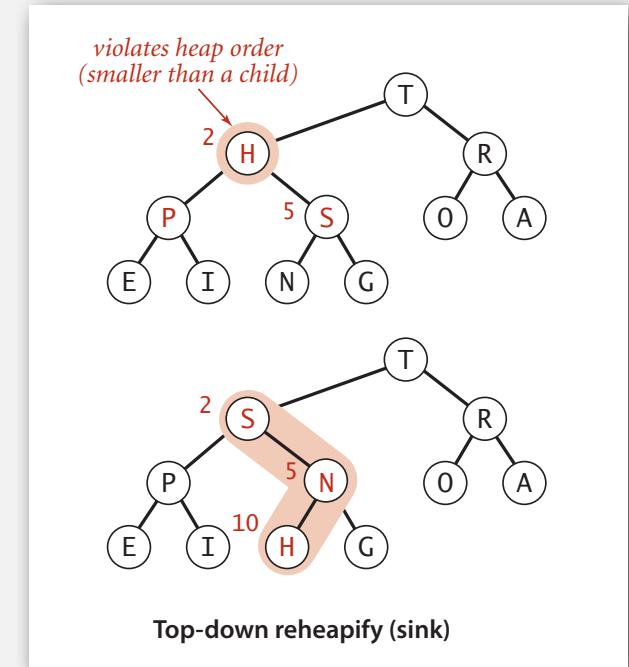
Scenario. Parent's key becomes **smaller** than one (or both) of its children's.

To eliminate the violation:

- Exchange key in parent with key in larger child.
- Repeat until heap order restored.

why not smaller child?

```
private void sink(int k)
{
    while (2*k <= N)          children of node at k
    {                           are 2k and 2k+1
        int j = 2*k;
        if (j < N && less(j, j+1)) j++;
        if (!less(k, j)) break;
        exch(k, j);
        k = j;
    }
}
```



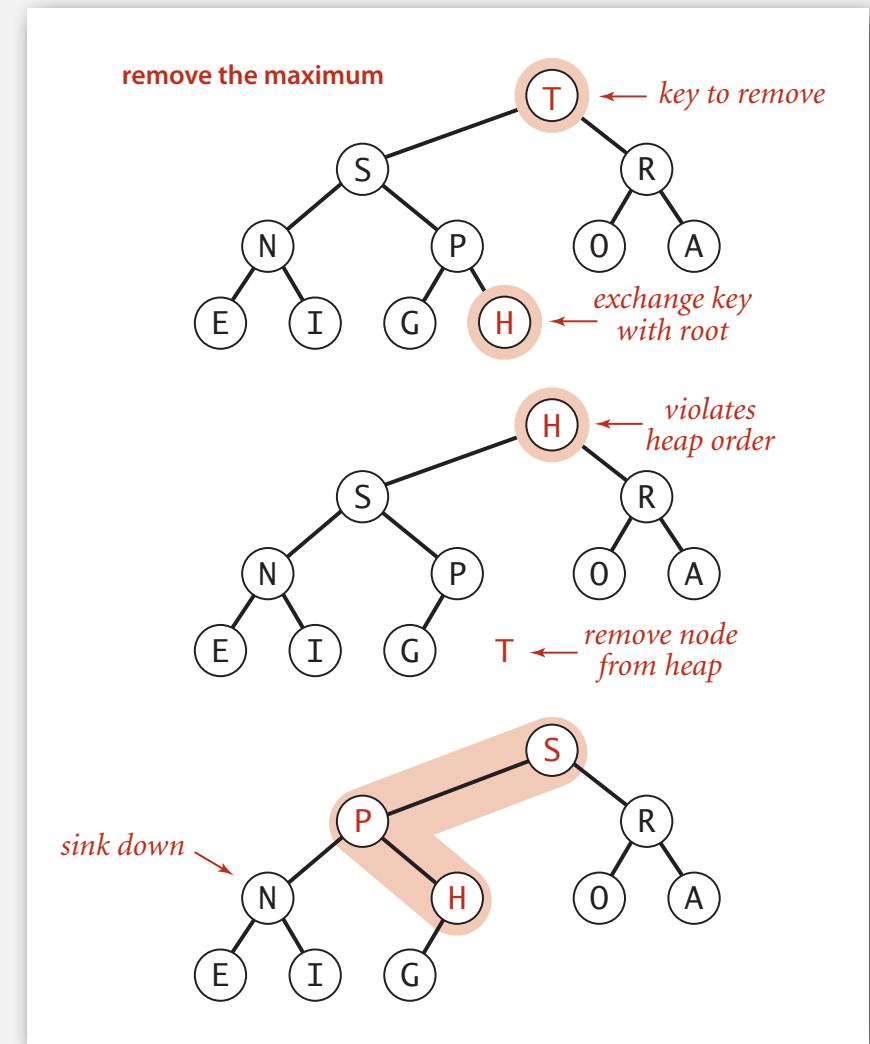
Power struggle. Better subordinate promoted.

Delete the maximum in a heap

Delete max. Exchange root with node at end, then sink it down.

Cost. At most $2 \lg N$ compares.

```
public Key delMax()
{
    Key max = pq[1];
    exch(1, N--);
    sink(1);
    pq[N+1] = null; ← prevent loitering
    return max;
}
```

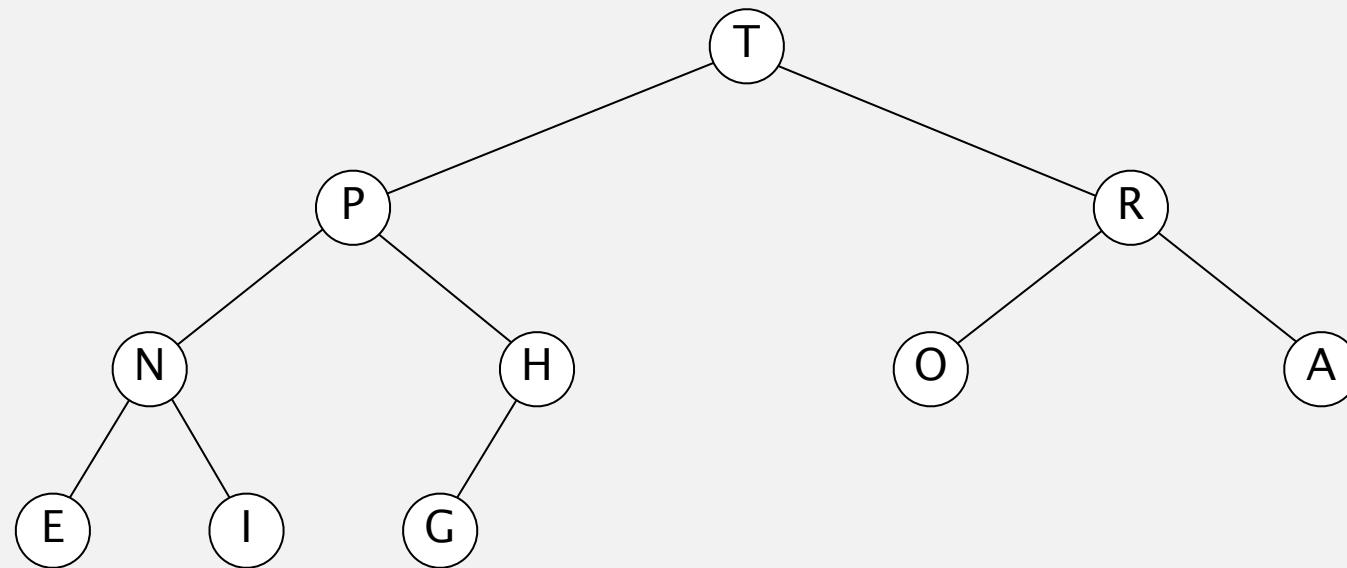


Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



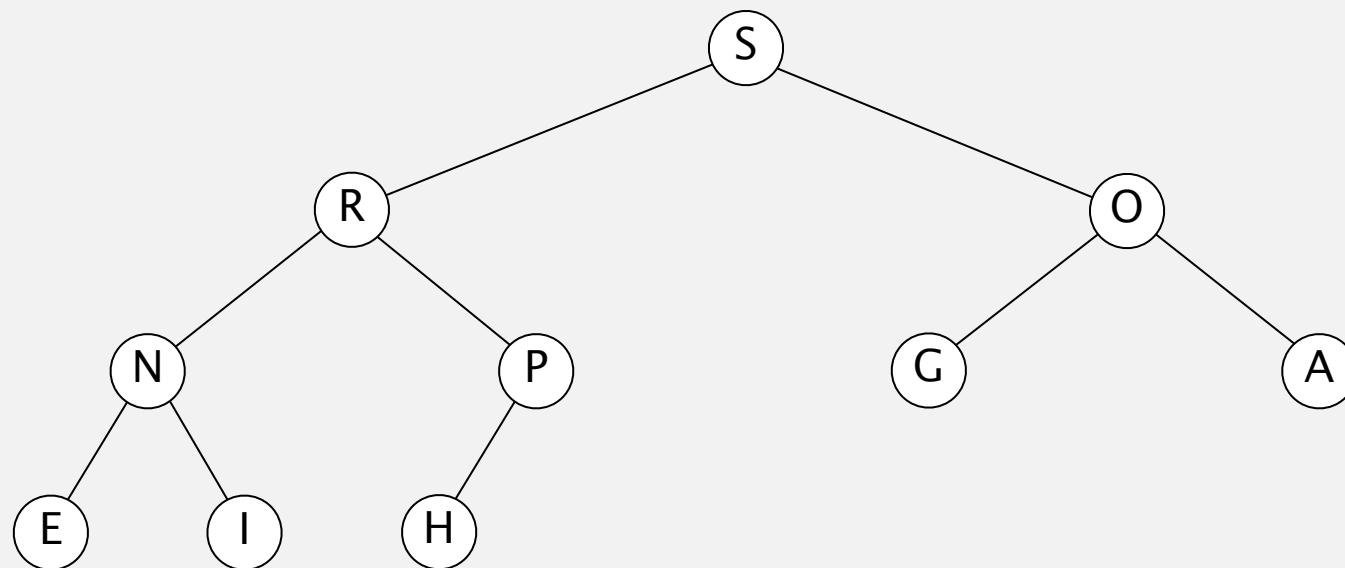
T	P	R	N	H	O	A	E	I	G
---	---	---	---	---	---	---	---	---	---

Binary heap demo

Insert. Add node at end, then swim it up.

Remove the maximum. Exchange root with node at end, then sink it down.

heap ordered



S	R	O	N	P	G	A	E	I	H
---	---	---	---	---	---	---	---	---	---

Binary heap: Java implementation

```
public class MaxPQ<Key extends Comparable<Key>>
{
    private Key[] pq;
    private int N;

    public MaxPQ(int capacity)
    {   pq = (Key[]) new Comparable[capacity+1]; }

    public boolean isEmpty()
    {   return N == 0;   }
    public void insert(Key key)
    public Key delMax()
    {   /* see previous code */ }

    private void swim(int k)
    private void sink(int k)
    {   /* see previous code */ }

    private boolean less(int i, int j)
    {   return pq[i].compareTo(pq[j]) < 0;   }
    private void exch(int i, int j)
    {   Key t = pq[i]; pq[i] = pq[j]; pq[j] = t;   }
}
```

fixed capacity
(for simplicity)

PQ ops

heap helper functions

array helper functions

Priority queues implementation cost summary

order-of-growth of running time for priority queue with N items

implementation	insert	del max	max
unordered array	1	N	N
ordered array	N	1	1
binary heap	$\log N$	$\log N$	1
d-ary heap	$\log_d N$	$d \log_d N$	1
Fibonacci	1	$\log N$ †	1
impossible	1	1	1

← why impossible?

† amortized

Binary heap considerations

Immutability of keys.

- Assumption: client does not change keys while they're on the PQ.
- Best practice: use immutable keys.

Underflow and overflow.

- Underflow: throw exception if deleting from empty PQ.
- Overflow: add no-arg constructor and use resizing array.

leads to log N
amortized time per op
(how to make worst case?)

Minimum-oriented priority queue.

- Replace `less()` with `greater()`.
- Implement `greater()`.

Other operations.

- Remove an arbitrary item.
- Change the priority of an item.

can implement with `sink()` and `swim()` [stay tuned]

Immutability: implementing in Java

Data type. Set of values and operations on those values.

Immutable data type. Can't change the data type value once created.

```
public final class Vector {  
    private final int N;  
    private final double[] data; | ← can't override instance methods  
  
    public Vector(double[] data) {  
        this.N = data.length;  
        this.data = new double[N];  
        for (int i = 0; i < N; i++) ← all instance variables private and final  
            this.data[i] = data[i];  
    }  
    ...  
}
```

← defensive copy of mutable instance variables

← instance methods don't change instance variables

Immutable. String, Integer, Double, Color, Vector, Transaction, Point2D.

Mutable. StringBuilder, Stack, Counter, Java array.

Immutability: properties

Data type. Set of values and operations on those values.

Immutable data type. Can't change the data type value once created.

Advantages.

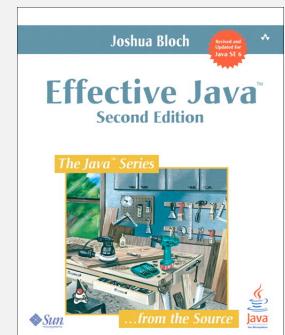
- Simplifies debugging.
- Safer in presence of hostile code.
- Simplifies concurrent programming.
- Safe to use as key in priority queue or symbol table.



Disadvantage. Must create new object for each data type value.

“Classes should be immutable unless there's a very good reason to make them mutable.... If a class cannot be made immutable, you should still limit its mutability as much as possible.”

— Joshua Bloch (Java architect)



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ ***binary heaps***
- ▶ *heapsort*
- ▶ *event-driven simulation*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

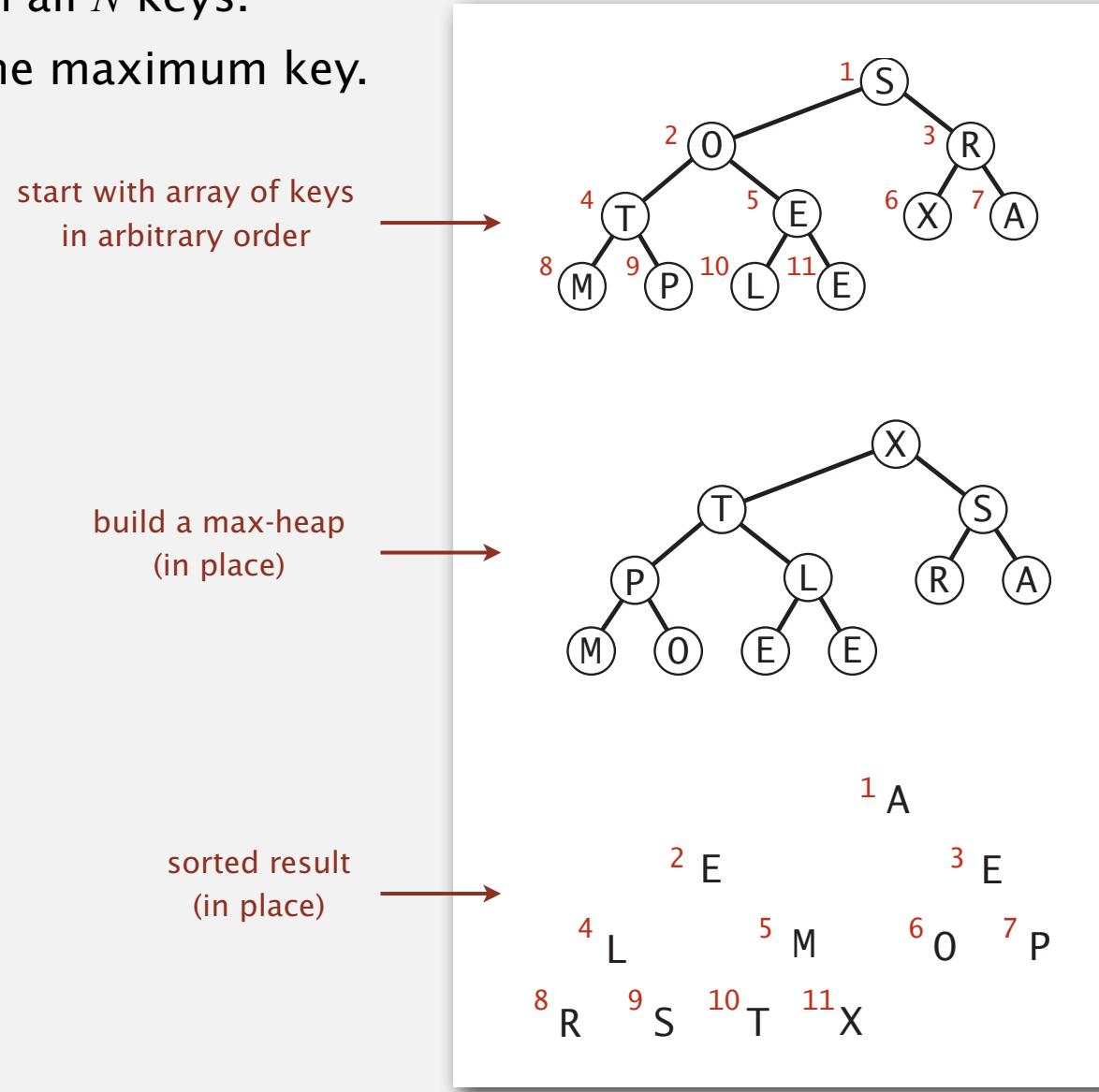
2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Heapsort

Basic plan for in-place sort.

- Create max-heap with all N keys.
- Repeatedly remove the maximum key.

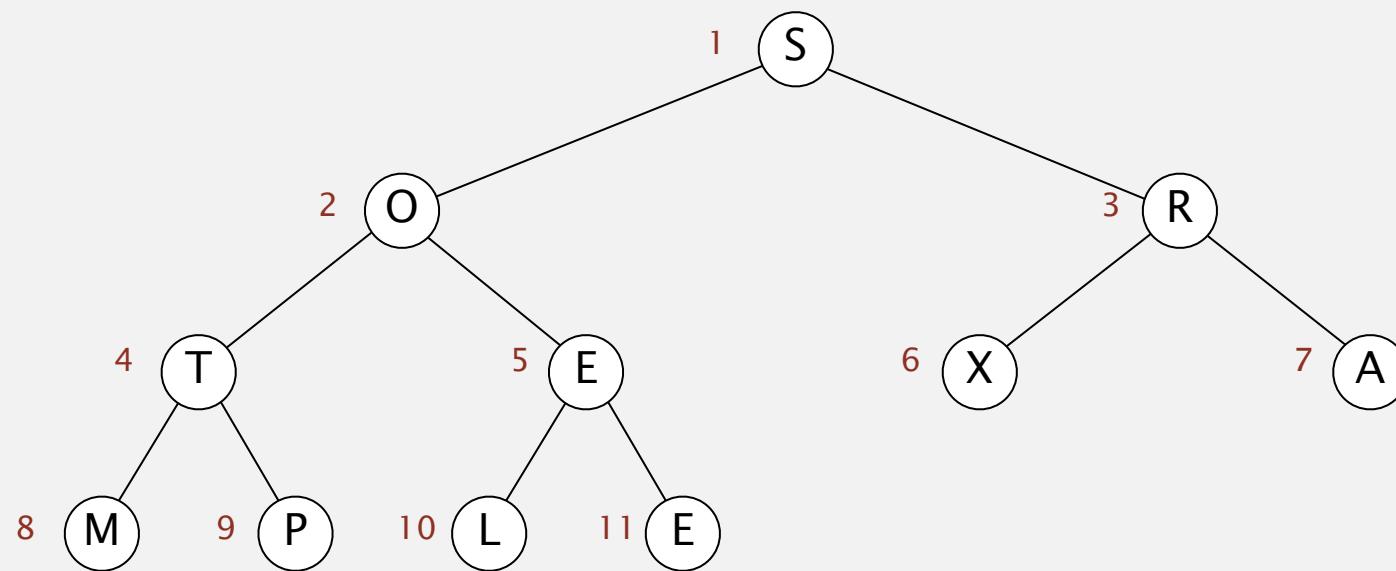


Heapsort demo

Heap construction. Build max heap using bottom-up method.

we assume array entries are indexed 1 to N

array in arbitrary order



S O R T E X A M P L E
1 2 3 4 5 6 7 8 9 10 11

Heapsort demo

Sortdown. Repeatedly delete the largest remaining item.

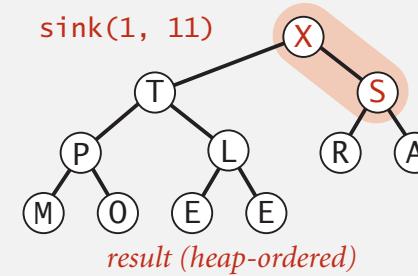
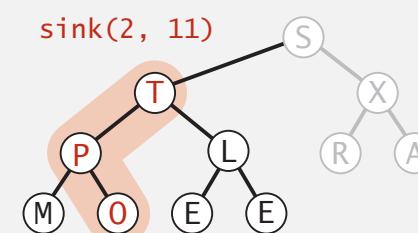
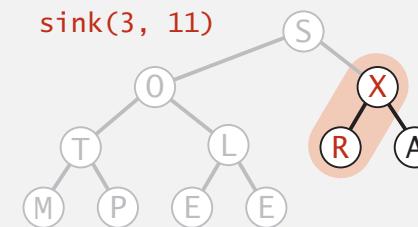
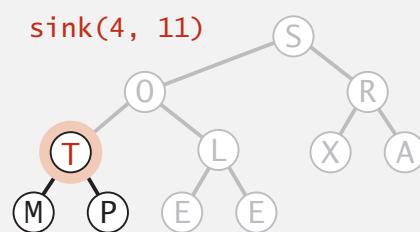
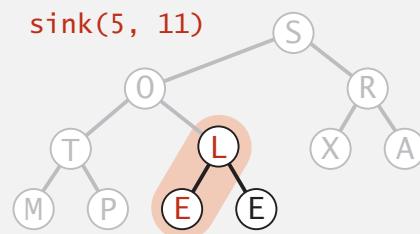
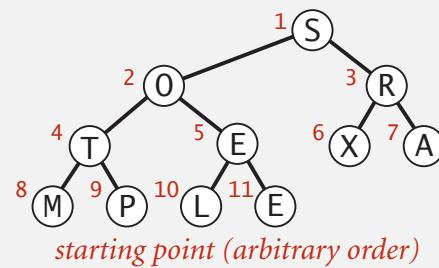
array in sorted order



Heapsort: heap construction

First pass. Build heap using bottom-up method.

```
for (int k = N/2; k >= 1; k--)  
    sink(a, k, N);
```

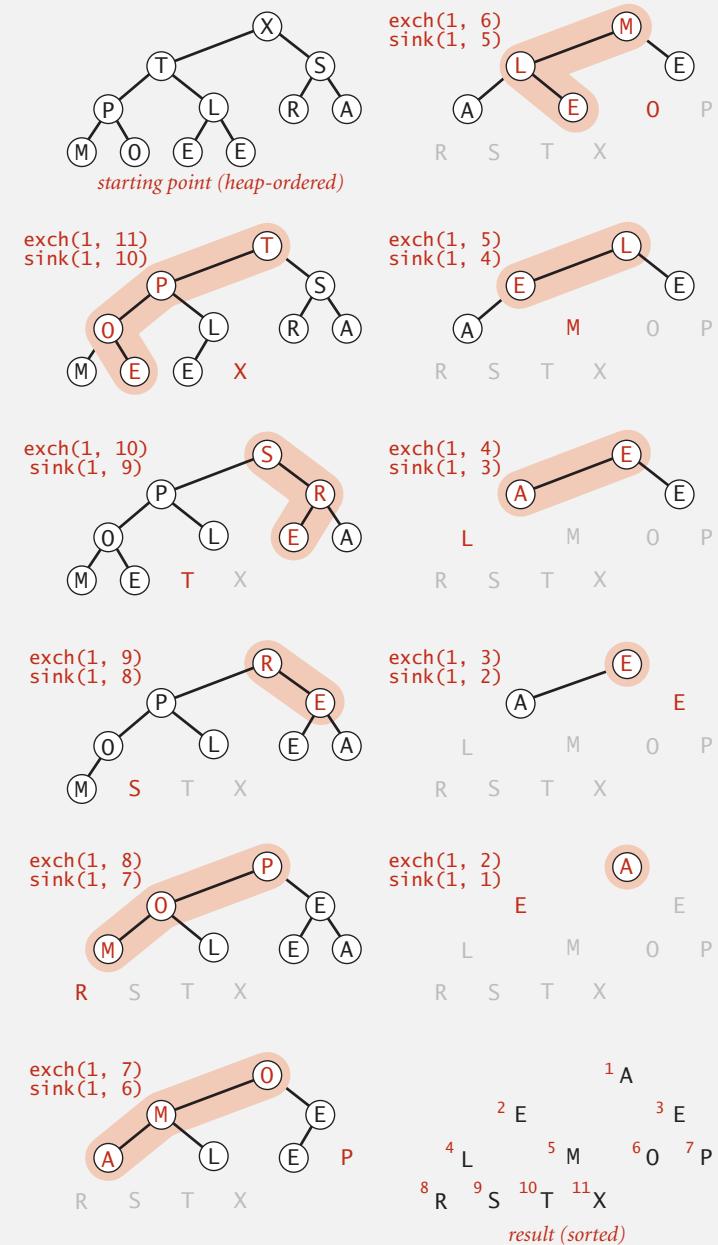


Heapsort: sortdown

Second pass.

- Remove the maximum, one at a time.
- Leave in array, instead of nulling out.

```
while (N > 1)
{
    exch(a, 1, N--);
    sink(a, 1, N);
}
```



Heapsort: Java implementation

```
public class Heap
{
    public static void sort(Comparable[] pq)
    {
        int N = pq.length;
        for (int k = N/2; k >= 1; k--)
            sink(pq, k, N);
        while (N > 1)
        {
            exch(pq, 1, N);
            sink(pq, 1, --N);
        }
    }

    private static void sink(Comparable[] pq, int k, int N)
    { /* as before */ }

    private static boolean less(Comparable[] pq, int i, int j)
    { /* as before */ }

    private static void exch(Comparable[] pq, int i, int j)
    { /* as before */ }
}
```

but convert from
1-based indexing to
0-base indexing

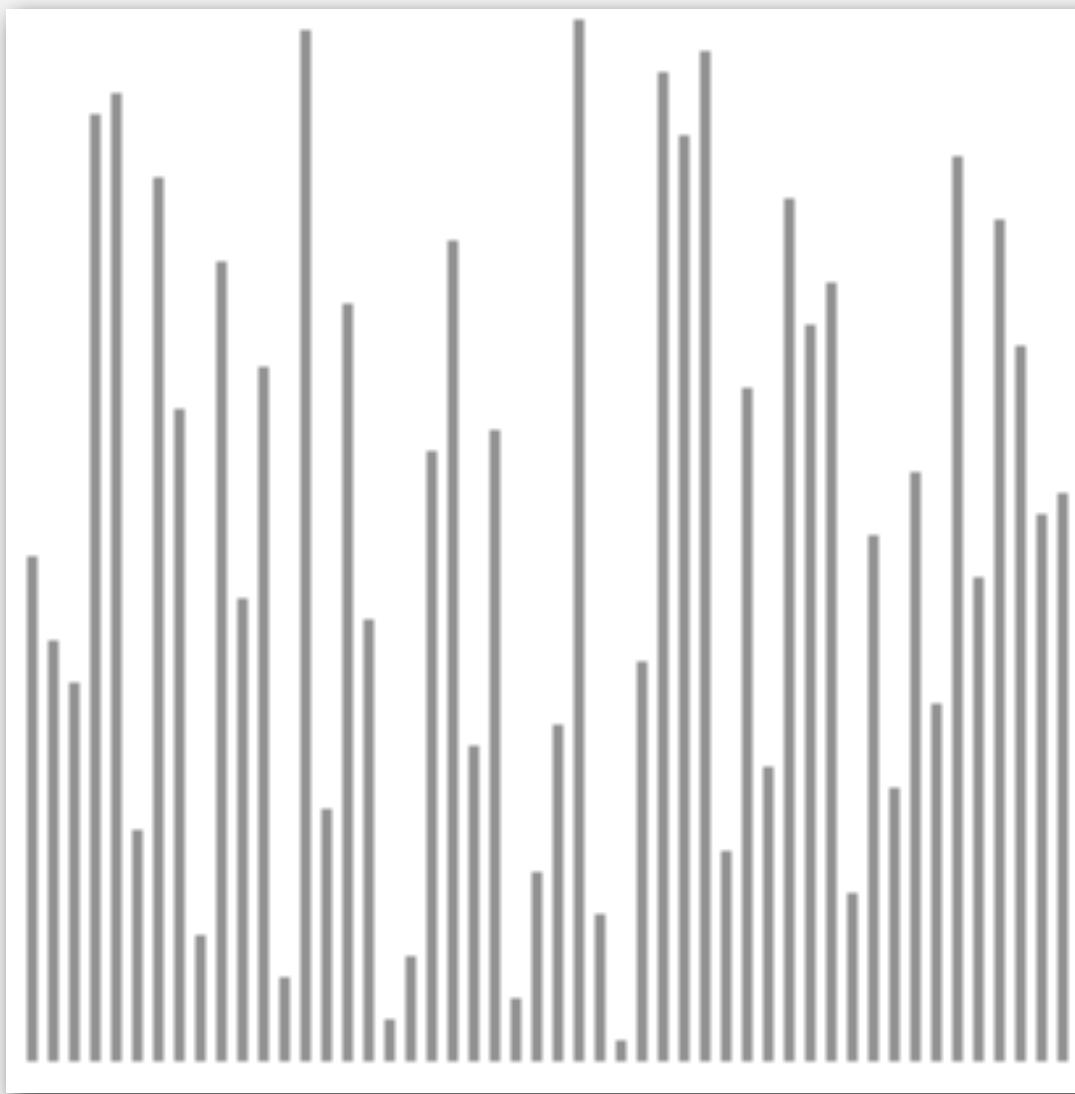
Heapsort: trace

		a[i]											
N	k	0	1	2	3	4	5	6	7	8	9	10	11
<i>initial values</i>		S	O	R	T	E	X	A	M	P	L	E	
11	5	S	O	R	T	L	X	A	M	P	E	E	
11	4	S	O	R	T	L	X	A	M	P	E	E	
11	3	S	O	X	T	L	R	A	M	P	E	E	
11	2	S	T	X	P	L	R	A	M	O	E	E	
11	1	X	T	S	P	L	R	A	M	O	E	E	
<i>heap-ordered</i>		X	T	S	P	L	R	A	M	O	E	E	
10	1	T	P	S	O	L	R	A	M	E	E	X	
9	1	S	P	R	O	L	E	A	M	E	T	X	
8	1	R	P	E	O	L	E	A	M	S	T	X	
7	1	P	O	E	M	L	E	A	R	S	T	X	
6	1	O	M	E	A	L	E	P	R	S	T	X	
5	1	M	L	E	A	E	O	P	R	S	T	X	
4	1	L	E	E	A	M	O	P	R	S	T	X	
3	1	E	A	E	L	M	O	P	R	S	T	X	
2	1	E	A	E	L	M	O	P	R	S	T	X	
1	1	A	E	E	L	M	O	P	R	S	T	X	
<i>sorted result</i>		A	E	E	L	M	O	P	R	S	T	X	

Heapsort trace (array contents just after each sink)

Heapsort animation

50 random items



<http://www.sorting-algorithms.com/heap-sort>

Heapsort: mathematical analysis

Proposition. Heap construction uses $\leq 2N$ compares and exchanges.

Proposition. Heapsort uses $\leq 2N \lg N$ compares and exchanges.

Significance. In-place sorting algorithm with $N \log N$ worst-case.

- Mergesort: no, linear extra space. ← in-place merge possible, not practical
- Quicksort: no, quadratic time in worst case. ← $N \log N$ worst-case quicksort possible, not practical
- Heapsort: yes!

Bottom line. Heapsort is optimal for both time and space, **but**:

- Inner loop longer than quicksort's.
- Makes poor use of cache memory.
- Not stable.

Sorting algorithms: summary

	inplace?	stable?	worst	average	best	remarks
selection	x		$N^2 / 2$	$N^2 / 2$	$N^2 / 2$	N exchanges
insertion	x	x	$N^2 / 2$	$N^2 / 4$	N	use for small N or partially ordered
shell	x		?	?	N	tight code, subquadratic
quick	x		$N^2 / 2$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
3-way quick	x		$N^2 / 2$	$2N \ln N$	N	improves quicksort in presence of duplicate keys
merge		x	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable
heap	x		$2N \lg N$	$2N \lg N$	$N \lg N$	$N \log N$ guarantee, in-place
???	x	x	$N \lg N$	$N \lg N$	$N \lg N$	holy sorting grail

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

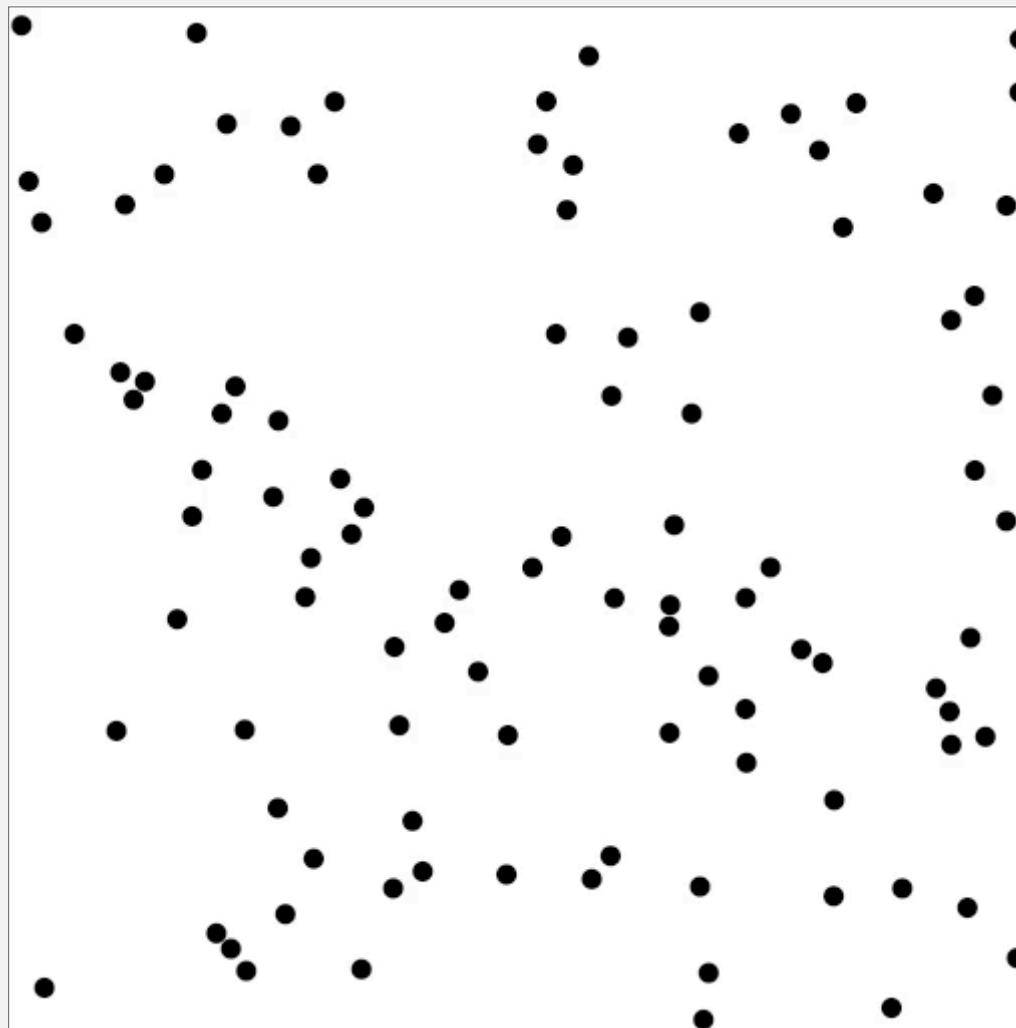
<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Molecular dynamics simulation of hard discs

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.

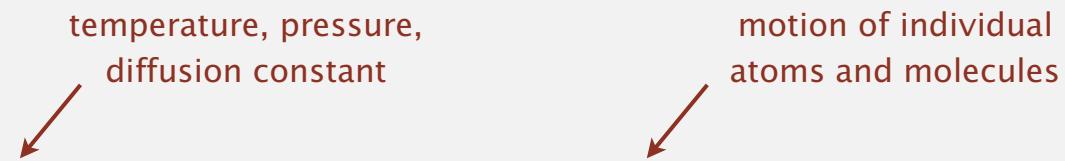


Molecular dynamics simulation of hard discs

Goal. Simulate the motion of N moving particles that behave according to the laws of elastic collision.

Hard disc model.

- Moving particles interact via elastic collisions with each other and walls.
- Each particle is a disc with known position, velocity, mass, and radius.
- No other forces.



Significance. Relates macroscopic observables to microscopic dynamics.

- Maxwell-Boltzmann: distribution of speeds as a function of temperature.
- Einstein: explain Brownian motion of pollen grains.

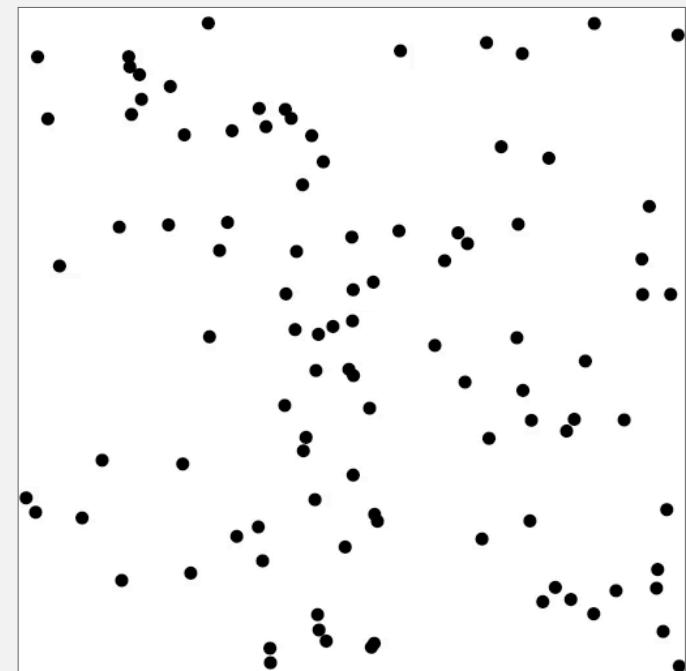
Warmup: bouncing balls

Time-driven simulation. N bouncing balls in the unit square.

```
public class BouncingBalls
{
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        Ball[] balls = new Ball[N];
        for (int i = 0; i < N; i++)
            balls[i] = new Ball();
        while(true)
        {
            StdDraw.clear();
            for (int i = 0; i < N; i++)
            {
                balls[i].move(0.5);
                balls[i].draw();
            }
            StdDraw.show(50);
        }
    }
}
```

main simulation loop

```
% java BouncingBalls 100
```



Warmup: bouncing balls

```
public class Ball
{
    private double rx, ry;          // position
    private double vx, vy;          // velocity
    private final double radius;    // radius
    public Ball(...)
    { /* initialize position and velocity */ }

    public void move(double dt)
    {
        if ((rx + vx*dt < radius) || (rx + vx*dt > 1.0 - radius)) { vx = -vx; }
        if ((ry + vy*dt < radius) || (ry + vy*dt > 1.0 - radius)) { vy = -vy; }
        rx = rx + vx*dt;
        ry = ry + vy*dt;
    }

    public void draw()
    { StdDraw.filledCircle(rx, ry, radius); }
}
```

check for collision with walls

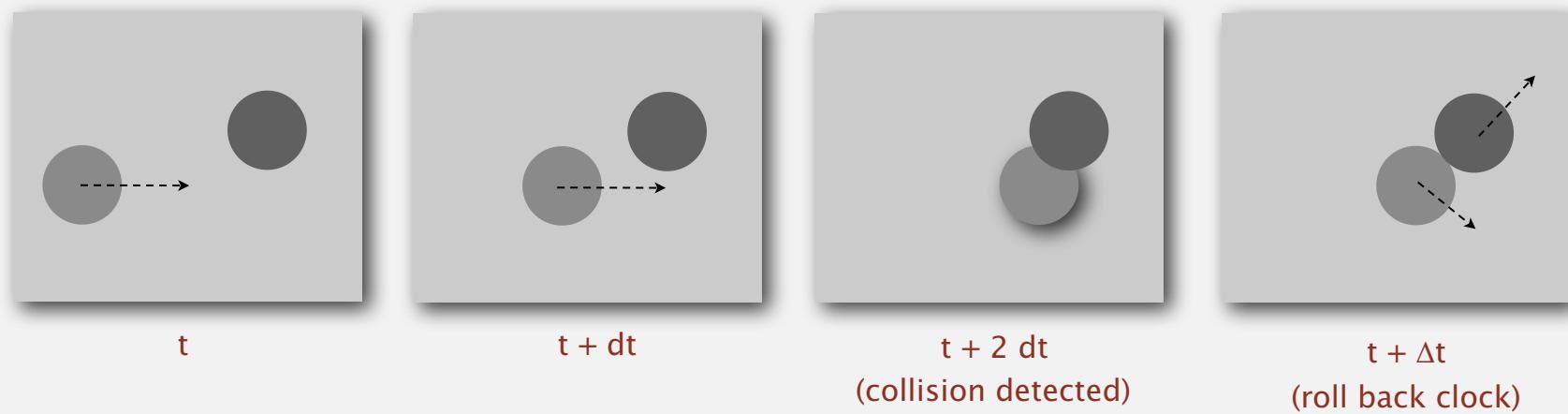


Missing. Check for balls colliding with **each other**.

- Physics problems: when? what effect?
- CS problems: which object does the check? too many checks?

Time-driven simulation

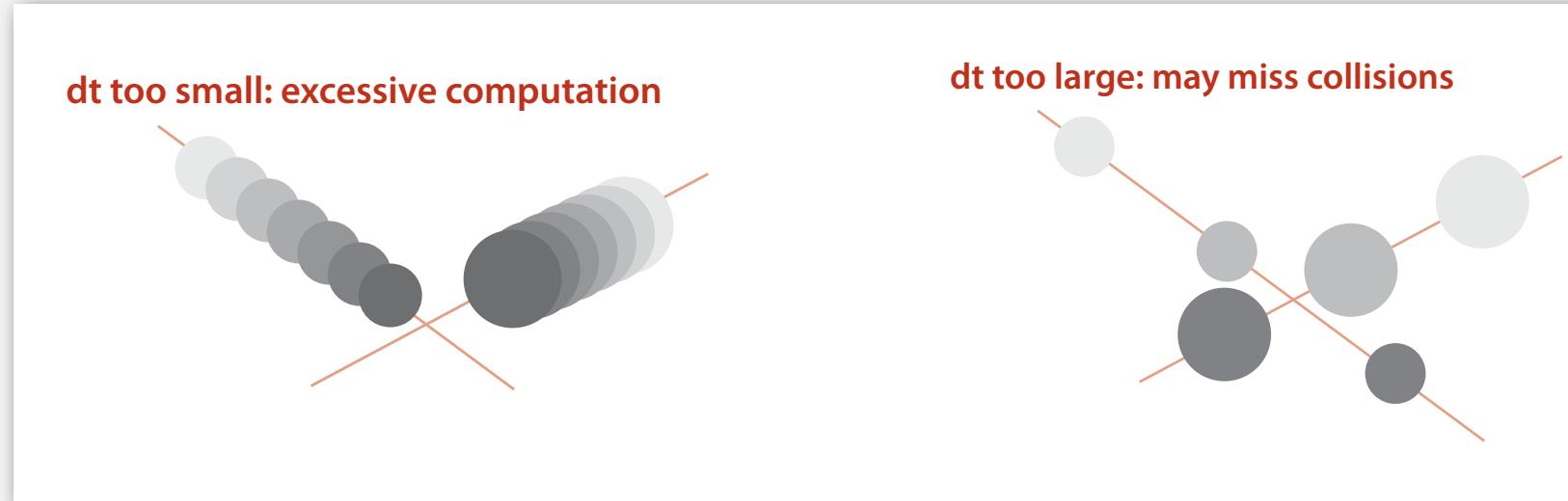
- Discretize time in quanta of size dt .
- Update the position of each particle after every dt units of time, and check for overlaps.
- If overlap, roll back the clock to the time of the collision, update the velocities of the colliding particles, and continue the simulation.



Time-driven simulation

Main drawbacks.

- $\sim N^2 / 2$ overlap checks per time quantum.
- Simulation is too slow if dt is very small.
- May miss collisions if dt is too large.
(if colliding particles fail to overlap when we are looking)



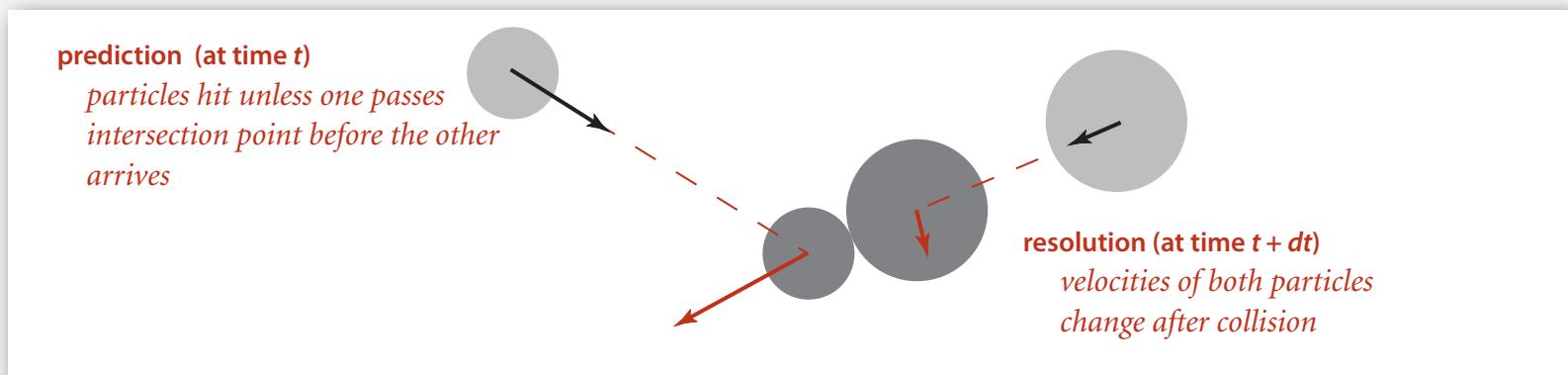
Event-driven simulation

Change state only when something happens.

- Between collisions, particles move in straight-line trajectories.
- Focus only on times when collisions occur.
- Maintain **PQ** of collision events, prioritized by time.
- Remove the min = get next collision.

Collision prediction. Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?

Collision resolution. If collision occurs, update colliding particle(s) according to laws of elastic collisions.



Particle-wall collision

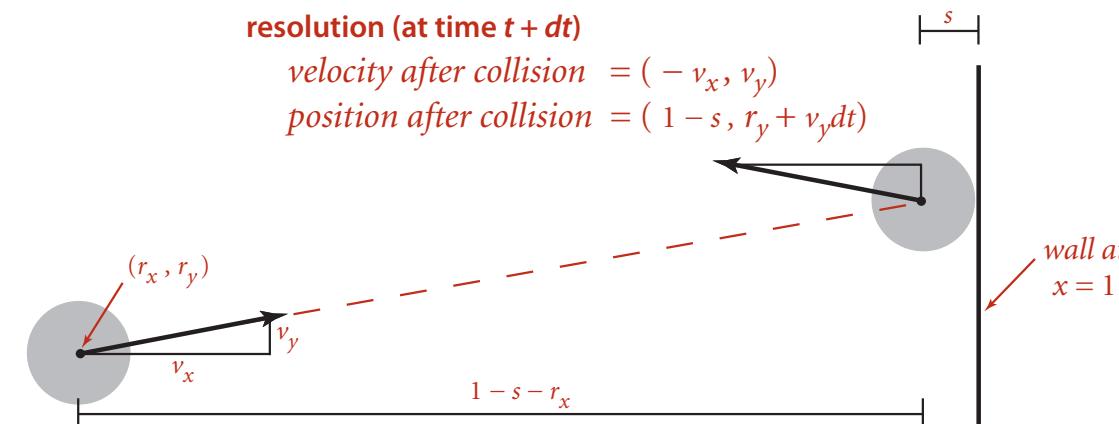
Collision prediction and resolution.

- Particle of radius s at position (r_x, r_y) .
- Particle moving in unit box with velocity (v_x, v_y) .
- Will it collide with a vertical wall? If so, when?

prediction (at time t)

$$dt \equiv \text{time to hit wall}$$
$$= \text{distance}/\text{velocity}$$
$$= (1 - s - r_x)/v_x$$

resolution (at time $t + dt$)

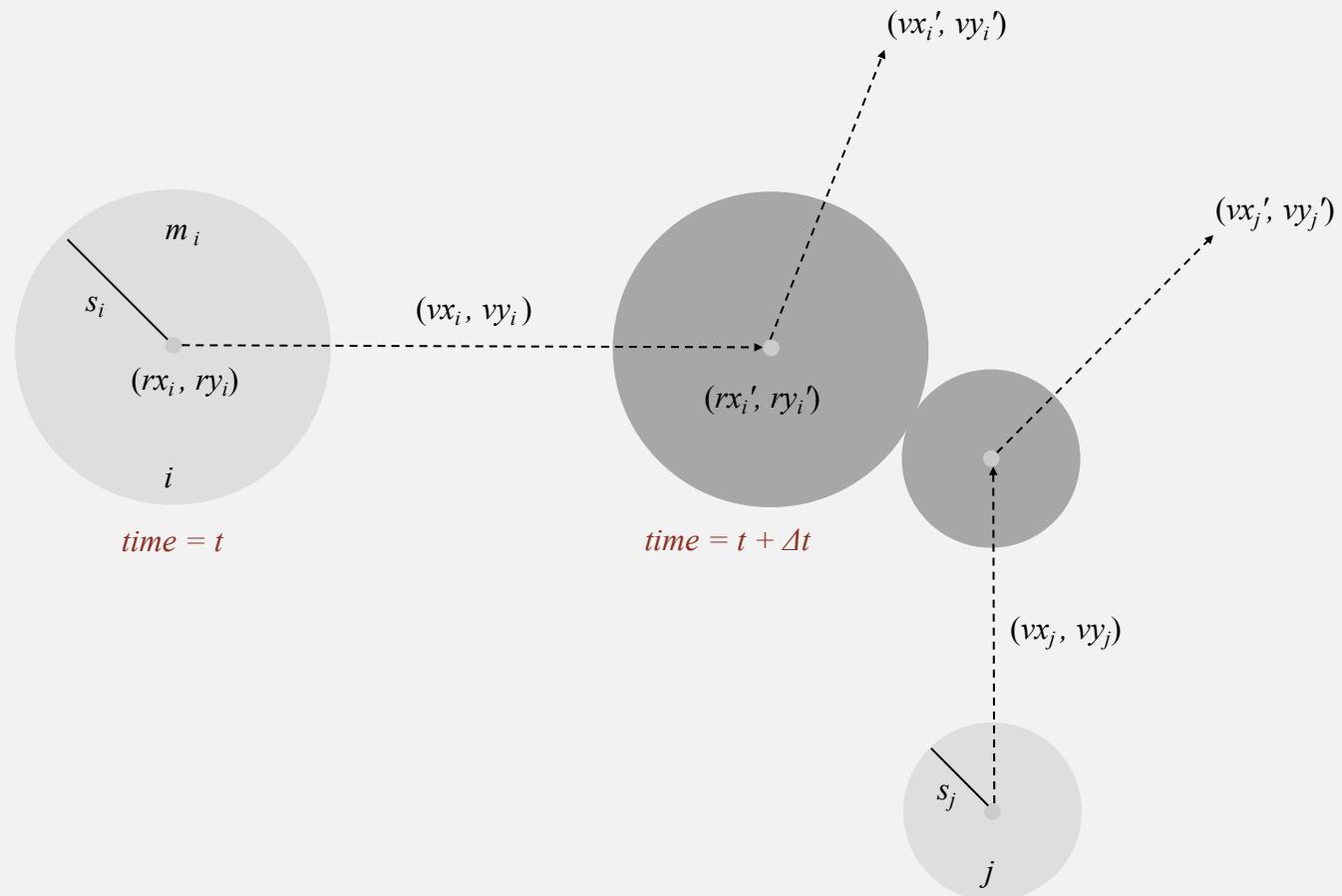
$$\text{velocity after collision} = (-v_x, v_y)$$
$$\text{position after collision} = (1 - s, r_y + v_y dt)$$


Predicting and resolving a particle-wall collision

Particle-particle collision prediction

Collision prediction.

- Particle i : radius s_i , position (rx_i, ry_i) , velocity (vx_i, vy_i) .
- Particle j : radius s_j , position (rx_j, ry_j) , velocity (vx_j, vy_j) .
- Will particles i and j collide? If so, when?



Particle-particle collision prediction

Collision prediction.

- Particle i : radius s_i , position (rx_i, ry_i) , velocity (vx_i, vy_i) .
- Particle j : radius s_j , position (rx_j, ry_j) , velocity (vx_j, vy_j) .
- Will particles i and j collide? If so, when?

$$\Delta t = \begin{cases} \infty & \text{if } \Delta v \cdot \Delta r \geq 0 \\ \infty & \text{if } d < 0 \\ -\frac{\Delta v \cdot \Delta r + \sqrt{d}}{\Delta v \cdot \Delta v} & \text{otherwise} \end{cases}$$

$$d = (\Delta v \cdot \Delta r)^2 - (\Delta v \cdot \Delta v) (\Delta r \cdot \Delta r - \sigma^2) \quad \sigma = \sigma_i + \sigma_j$$

$$\begin{aligned}\Delta v &= (\Delta vx, \Delta vy) = (vx_i - vx_j, vy_i - vy_j) \\ \Delta r &= (\Delta rx, \Delta ry) = (rx_i - rx_j, ry_i - ry_j)\end{aligned}$$

$$\begin{aligned}\Delta v \cdot \Delta v &= (\Delta vx)^2 + (\Delta vy)^2 \\ \Delta r \cdot \Delta r &= (\Delta rx)^2 + (\Delta ry)^2 \\ \Delta v \cdot \Delta r &= (\Delta vx)(\Delta rx) + (\Delta vy)(\Delta ry)\end{aligned}$$

Important note: This is high-school physics, so we won't be testing you on it!

Particle-particle collision resolution

Collision resolution. When two particles collide, how does velocity change?

$$\begin{aligned} v{x_i}' &= v{x_i} + Jx / m_i \\ v{y_i}' &= v{y_i} + Jy / m_i \\ v{x_j}' &= v{x_j} - Jx / m_j \\ v{y_j}' &= v{y_j} - Jy / m_j \end{aligned}$$

Newton's second law
(momentum form)

$$Jx = \frac{J \Delta rx}{\sigma}, \quad Jy = \frac{J \Delta ry}{\sigma}, \quad J = \frac{2 m_i m_j (\Delta v \cdot \Delta r)}{\sigma(m_i + m_j)}$$

impulse due to normal force

(conservation of energy, conservation of momentum)

Important note: This is high-school physics, so we won't be testing you on it!

Particle data type skeleton

```
public class Particle
{
    private double rx, ry;          // position
    private double vx, vy;          // velocity
    private final double radius;    // radius
    private final double mass;      // mass
    private int count;              // number of collisions

    public Particle(...) { }

    public void move(double dt) { }
    public void draw() { }

    public double timeToHit(Particle that) { }
    public double timeToHitVerticalWall() { }
    public double timeToHitHorizontalWall() { }

    public void bounceOff(Particle that) { }
    public void bounceOffVerticalWall() { }
    public void bounceOffHorizontalWall() { }

}
```

predict collision
with particle or wall

resolve collision
with particle or wall

Particle-particle collision and resolution implementation

```
public double timeToHit(Particle that)
{
    if (this == that) return INFINITY;
    double dx = that.rx - this.rx, dy = that.ry - this.ry;
    double dvx = that.vx - this.vx; dvy = that.vy - this.vy;
    double dvdr = dx*dvx + dy*dvy;
    if( dvdr > 0) return INFINITY; ← no collision
    double dvdv = dvx*dvx + dvy*dvy;
    double drdr = dx*dx + dy*dy;
    double sigma = this.radius + that.radius;
    double d = (dvdr*dvdr) - dvdv * (drdr - sigma*sigma);
    if (d < 0) return INFINITY;
    return -(dvdr + Math.sqrt(d)) / dvdv;
}
```

```
public void bounceOff(Particle that)
{
    double dx = that.rx - this.rx, dy = that.ry - this.ry;
    double dvx = that.vx - this.vx, dvy = that.vy - this.vy;
    double dvdr = dx*dvx + dy*dvy;
    double dist = this.radius + that.radius;
    double J = 2 * this.mass * that.mass * dvdr / ((this.mass + that.mass) * dist);
    double Jx = J * dx / dist;
    double Jy = J * dy / dist;
    this.vx += Jx / this.mass;
    this.vy += Jy / this.mass;
    that.vx -= Jx / that.mass;
    that.vy -= Jy / that.mass;
    this.count++;
    that.count++;     Important note: This is high-school physics, so we won't be testing you on it!
}
```

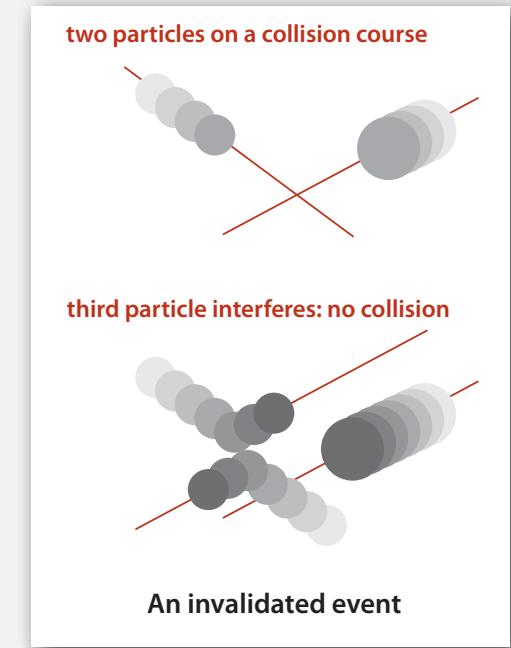
Collision system: event-driven simulation main loop

Initialization.

- Fill PQ with all potential particle-wall collisions.
- Fill PQ with all potential particle-particle collisions.



“potential” since collision may not happen if some other collision intervenes



Main loop.

- Delete the impending event from PQ (min priority = t).
- If the event has been invalidated, ignore it.
- Advance all particles to time t , on a straight-line trajectory.
- Update the velocities of the colliding particle(s).
- Predict future particle-wall and particle-particle collisions involving the colliding particle(s) and insert events onto PQ.

Event data type

Conventions.

- Neither particle null \Rightarrow particle-particle collision.
- One particle null \Rightarrow particle-wall collision.
- Both particles null \Rightarrow redraw event.

```
private class Event implements Comparable<Event>
{
    private double time;          // time of event
    private Particle a, b;        // particles involved in event
    private int countA, countB;   // collision counts for a and b

    public Event(double t, Particle a, Particle b) { }           ← create event

    public int compareTo(Event that)
    {   return this.time - that.time;   }                           ← ordered by time

    public boolean isValid()
    {   }
}
```

invalid if
intervening
collision

Collision system implementation: skeleton

```
public class CollisionSystem
{
    private MinPQ<Event> pq;          // the priority queue
    private double t = 0.0;             // simulation clock time
    private Particle[] particles;      // the array of particles

    public CollisionSystem(Particle[] particles) { }

    private void predict(Particle a)      add to PQ all particle-wall and particle-
    {                                     -particle collisions involving this particle
        if (a == null) return;
        for (int i = 0; i < N; i++)
        {
            double dt = a.timeToHit(particles[i]);
            pq.insert(new Event(t + dt, a, particles[i]));
        }
        pq.insert(new Event(t + a.timeToHitVerticalWall() , a, null));
        pq.insert(new Event(t + a.timeToHitHorizontalWall(), null, a));
    }

    private void redraw() { }

    public void simulate() { /* see next slide */ }
}
```

Collision system implementation: main event-driven simulation loop

```
public void simulate()
{
    pq = new MinPQ<Event>();
    for(int i = 0; i < N; i++) predict(particles[i]);
    pq.insert(new Event(0, null, null));
```

initialize PQ with collision events and redraw event

```
    while(!pq.isEmpty())
    {
        Event event = pq.delMin();
        if(!event.isValid()) continue;
        Particle a = event.a;
        Particle b = event.b;
```

get next event

```
        for(int i = 0; i < N; i++)
            particles[i].move(event.time - t);
        t = event.time;
```

update positions and time

```
        if      (a != null && b != null) a.bounceOff(b);
        else if (a != null && b == null) a.bounceOffVerticalWall();
        else if (a == null && b != null) b.bounceOffHorizontalWall();
        else if (a == null && b == null) redraw();
```

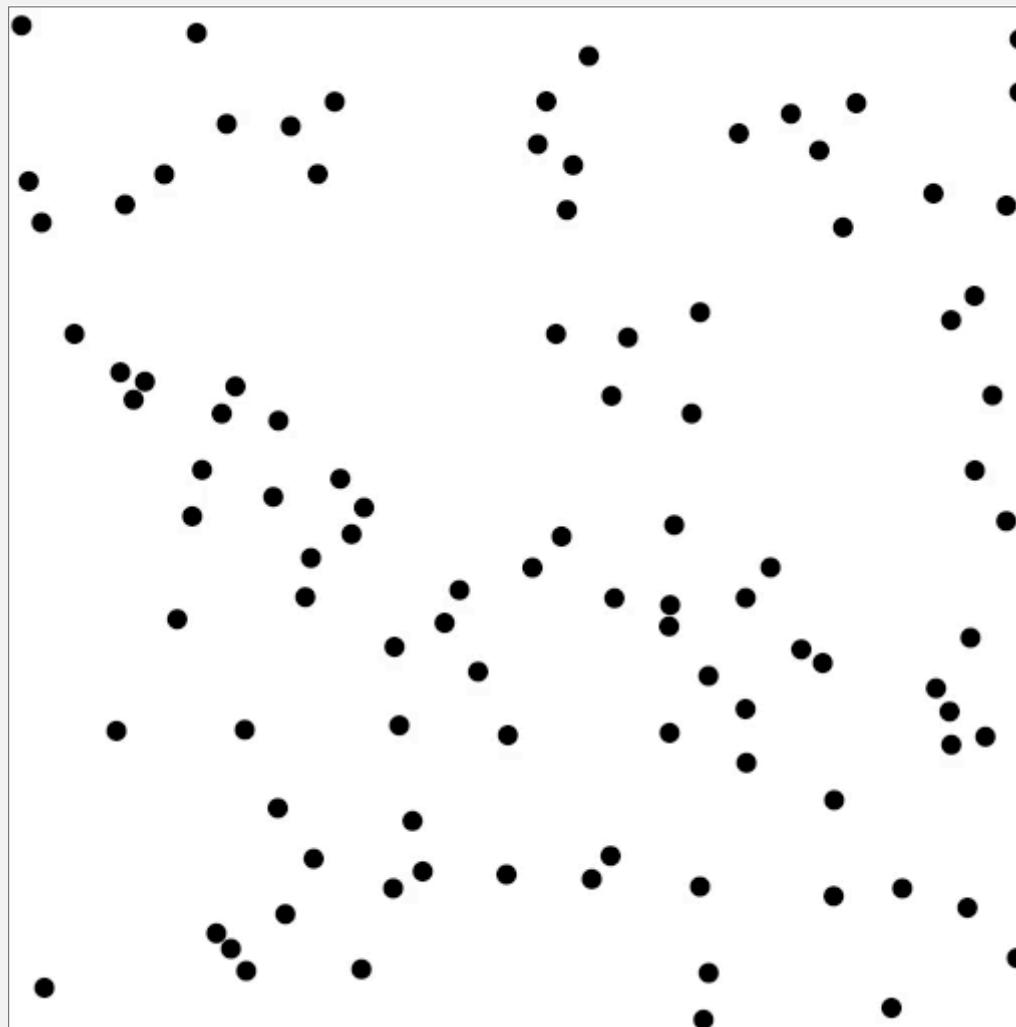
process event

```
        predict(a);
        predict(b);
    }
}
```

predict new events based on changes

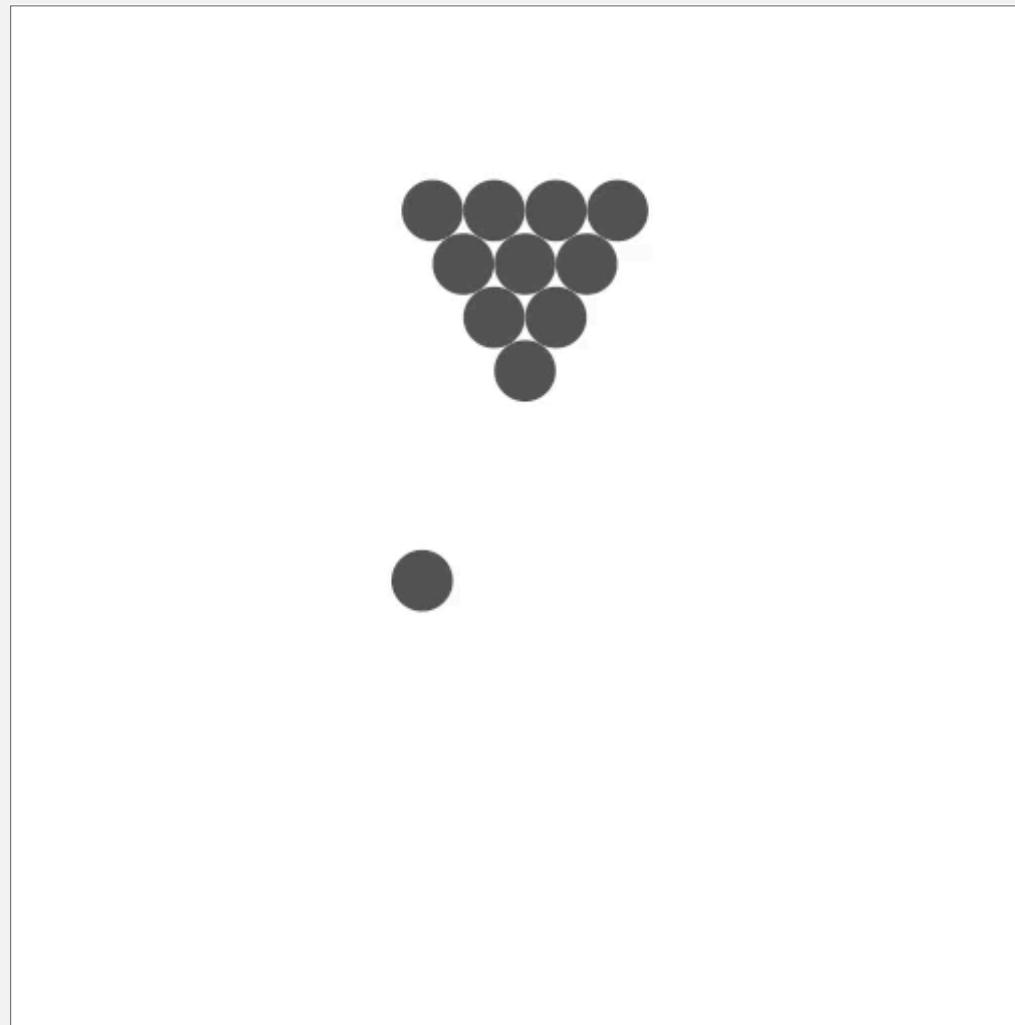
Particle collision simulation example 1

```
% java CollisionSystem 100
```



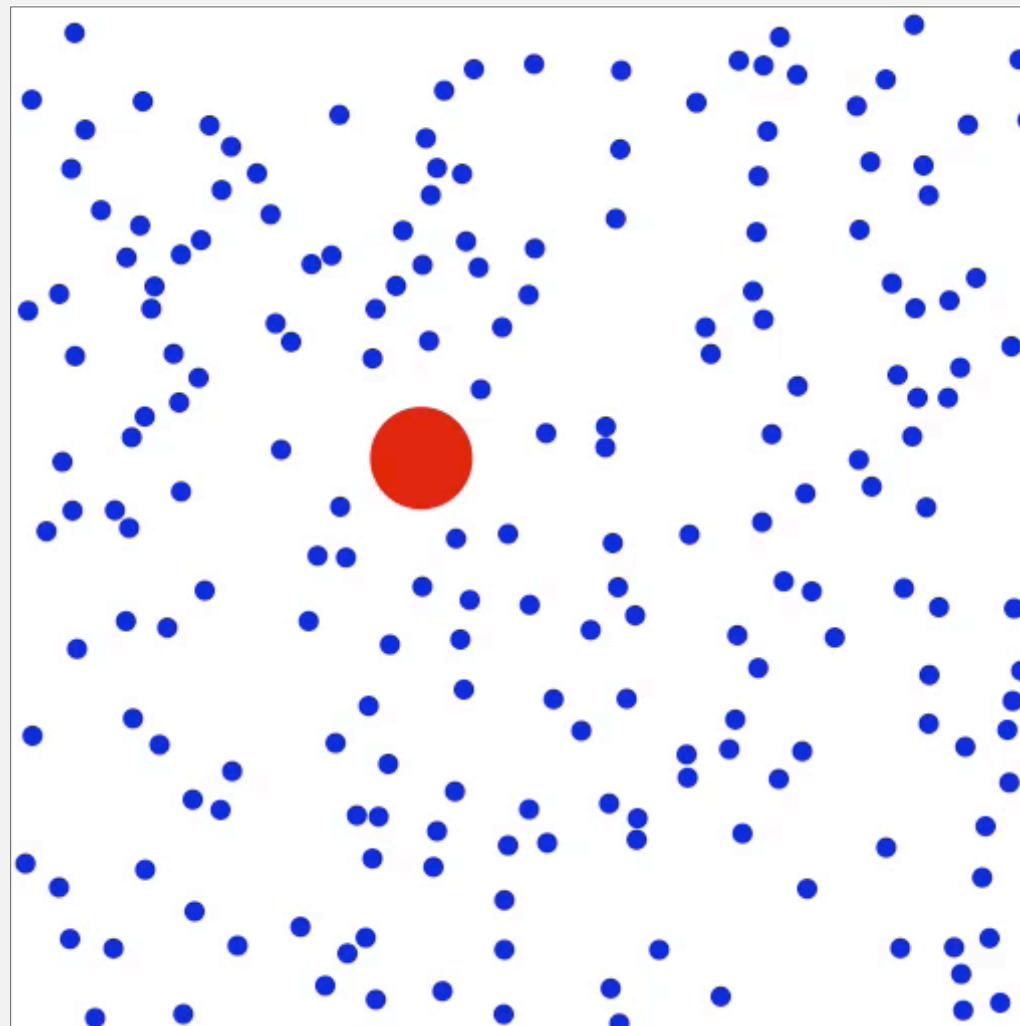
Particle collision simulation example 2

```
% java CollisionSystem < billiards.txt
```



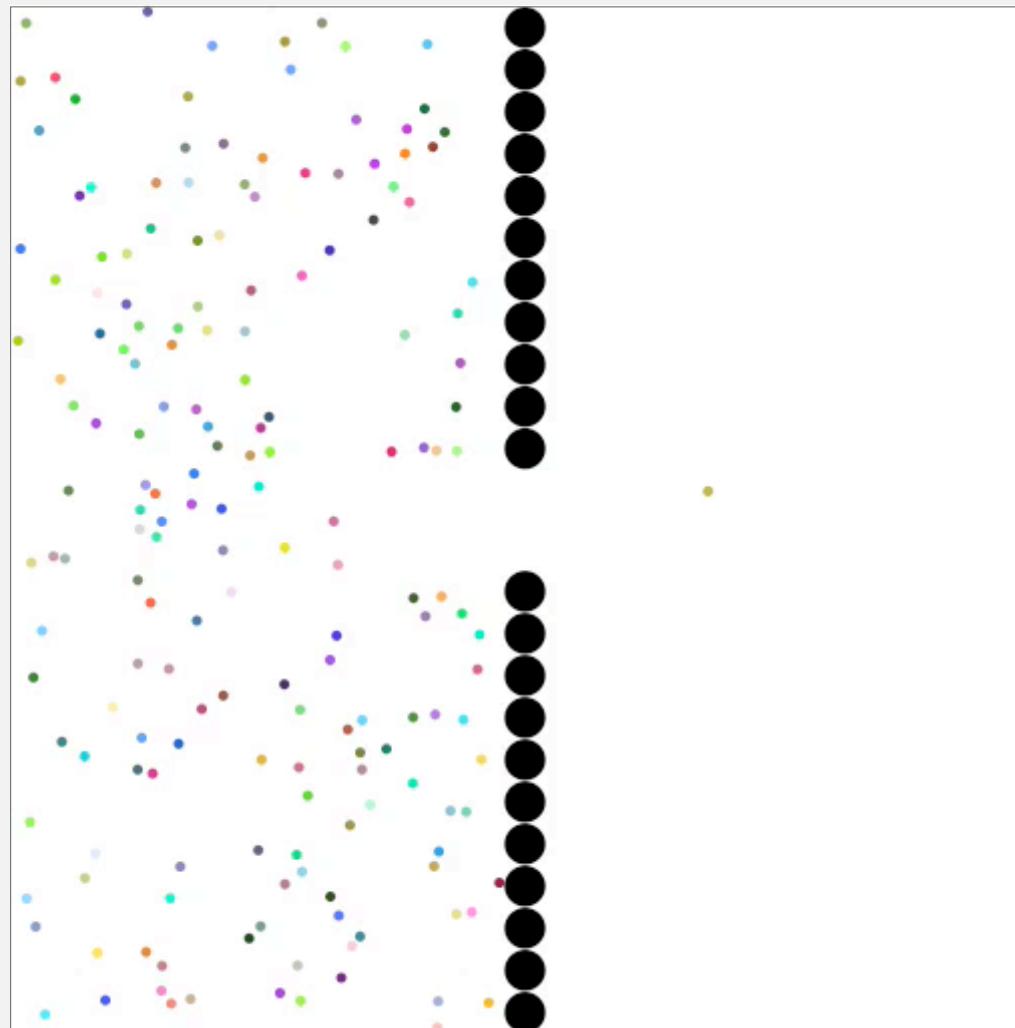
Particle collision simulation example 3

```
% java CollisionSystem < brownian.txt
```



Particle collision simulation example 4

```
% java CollisionSystem < diffusion.txt
```



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



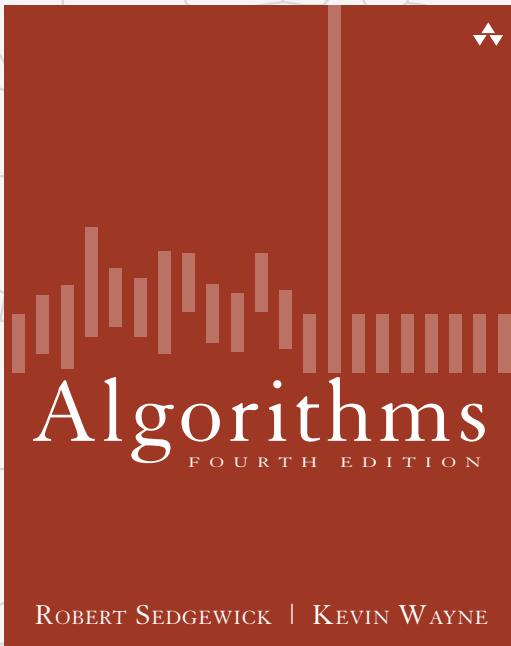
<http://algs4.cs.princeton.edu>

2.4 PRIORITY QUEUES

- ▶ *API and elementary implementations*
- ▶ *binary heaps*
- ▶ *heapsort*
- ▶ *event-driven simulation*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

- ▶ API
- ▶ *elementary implementations*
- ▶ *ordered operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

▶ API

▶ *elementary implementations*

▶ *ordered operations*

Symbol tables

Key-value pair abstraction.

- **Insert** a value with specified key.
- Given a key, **search** for the corresponding value.

Ex. DNS lookup.

- Insert URL with specified IP address.
- Given URL, find corresponding IP address.

URL	IP address
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60

key

value

Symbol table applications

application	purpose of search	key	value
dictionary	find definition	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
financial account	process transactions	account number	transaction details
web search	find relevant web pages	keyword	list of page names
compiler	find properties of variables	variable name	type and value
routing table	route Internet packets	destination	best route
DNS	find IP address given URL	URL	IP address
reverse DNS	find URL given IP address	IP address	URL
genomics	find markers	DNA string	known positions
file system	find file on disk	filename	location on disk

Basic symbol table API

Associative array abstraction. Associate one value with each key.

public class ST<Key, Value>	
ST()	<i>create a symbol table</i>
void put(Key key, Value val)	<i>put key-value pair into the table (remove key from table if value is null)</i>
Value get(Key key)	<i>value paired with key (null if key is absent)</i>
void delete(Key key)	<i>remove key (and its value) from table</i>
boolean contains(Key key)	<i>is there a value paired with key?</i>
boolean isEmpty()	<i>is the table empty?</i>
int size()	<i>number of key-value pairs in the table</i>
Iterable<Key> keys()	<i>all the keys in the table</i>

Conventions

- Values are not null.
- Method get() returns null if key not present.
- Method put() overwrites old value with new value.

Intended consequences.

- Easy to implement contains().

```
public boolean contains(Key key)
{   return get(key) != null; }
```

- Can implement lazy version of delete().

```
public void delete(Key key)
{   put(key, null); }
```

Keys and values

Value type. Any generic type.

Key type: several natural assumptions.

- Assume keys are Comparable, use compareTo().
- Assume keys are any generic type, use equals() to test equality.
- Assume keys are any generic type, use equals() to test equality; use hashCode() to scramble key.

specify Comparable in API.

built-in to Java
(stay tuned)

Best practices. Use immutable types for symbol table keys.

- Immutable in Java: Integer, Double, String, java.io.File, ...
- Mutable in Java: StringBuilder, java.net.URL, arrays, ...

Equality test

All Java classes inherit a method `equals()`.

Java requirements. For any references x , y and z :

- Reflexive: $x.equals(x)$ is true.
 - Symmetric: $x.equals(y)$ iff $y.equals(x)$.
 - Transitive: if $x.equals(y)$ and $y.equals(z)$, then $x.equals(z)$.
 - Non-null: $x.equals(null)$ is false.
-  equivalence relation

Default implementation. ($x == y$)

do x and y refer to
the same object?



Customized implementations. `Integer`, `Double`, `String`, `java.io.File`, ...

User-defined implementations. Some care needed.

Implementing equals for user-defined types

Seems easy.

```
public class Date implements Comparable<Date>
{
    private final int month;
    private final int day;
    private final int year;
    ...

    public boolean equals(Date that)
    {

        if (this.day != that.day) return false;
        if (this.month != that.month) return false;
        if (this.year != that.year) return false;
        return true;
    }
}
```

check that all significant
fields are the same

Implementing equals for user-defined types

Seems easy, but requires some care.

typically unsafe to use equals() with inheritance
(would violate symmetry)

```
public final class Date implements Comparable<Date>
{
    private final int month;
    private final int day;
    private final int year;
    ...
    public boolean equals(Object y)
    {
        if (y == this) return true;           ← optimize for true object equality
        if (y == null) return false;          ← check for null
        if (y.getClass() != this.getClass()) ← objects must be in the same class
            return false;                  (religion: getClass() vs. instanceof)
        Date that = (Date) y;
        if (this.day != that.day) return false; ← cast is guaranteed to succeed
        if (this.month != that.month) return false; ← check that all significant
        if (this.year != that.year) return false;   fields are the same
        return true;
    }
}
```

must be Object.
Why? Experts still debate.

Equals design

"Standard" recipe for user-defined types.

- Optimization for reference equality.
- Check against null.
- Check that two objects are of the same type and cast.
- Compare each significant field:
 - if field is a primitive type, use `==`
 - if field is an object, use `equals()` ← apply rule recursively
 - if field is an array, apply to each entry ← alternatively, use `Arrays.equals(a, b)` or `Arrays.deepEquals(a, b)`, but not `a.equals(b)`

Best practices.

- No need to use calculated fields that depend on other fields.
- Compare fields mostly likely to differ first.
- Make `compareTo()` consistent with `equals()`.

`x.equals(y)` if and only if `(x.compareTo(y) == 0)`

ST test client for traces

Build ST by associating value i with i^{th} string from standard input.

```
public static void main(String[] args)
{
    ST<String, Integer> st = new ST<String, Integer>();
    for (int i = 0; !StdIn.isEmpty(); i++)
    {
        String key = StdIn.readString();
        st.put(key, i);
    }
    for (String s : st.keys())
        StdOut.println(s + " " + st.get(s));
}
```

output

A	8
C	4
E	12
H	5
L	11
M	9
P	10
R	3
S	0
X	7

keys	S	E	A	R	C	H	E	X	A	M	P	L	E
values	0	1	2	3	4	5	6	7	8	9	10	11	12

ST test client for analysis

Frequency counter. Read a sequence of strings from standard input and print out one that occurs with highest frequency.

```
% more tinyTale.txt  
it was the best of times  
it was the worst of times  
it was the age of wisdom  
it was the age of foolishness  
it was the epoch of belief  
it was the epoch of incredulity  
it was the season of light  
it was the season of darkness  
it was the spring of hope  
it was the winter of despair
```

```
% java FrequencyCounter 1 < tinyTale.txt  
it 10
```

```
% java FrequencyCounter 8 < tale.txt  
business 122
```

```
% java FrequencyCounter 10 < leipzig1M.txt  
government 24763
```

- ← tiny example
(60 words, 20 distinct)
- ← real example
(135,635 words, 10,769 distinct)
- ← real example
(21,191,455 words, 534,580 distinct)

Frequency counter implementation

```
public class FrequencyCounter
{
    public static void main(String[] args)
    {
        int minlen = Integer.parseInt(args[0]);
        ST<String, Integer> st = new ST<String, Integer>(); ← create ST
        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString(); ← read string and update frequency
            if (word.length() < minlen) continue; ← ignore short strings
            if (!st.contains(word)) st.put(word, 1);
            else st.put(word, st.get(word) + 1);
        }
        String max = "";
        st.put(max, 0);
        for (String word : st.keys())
            if (st.get(word) > st.get(max))
                max = word; ← print a string with max freq
        StdOut.println(max + " " + st.get(max));
    }
}
```

create ST

read string and update frequency

print a string with max freq

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

► API

► *elementary implementations*

► *ordered operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

▶ API

▶ *elementary implementations*

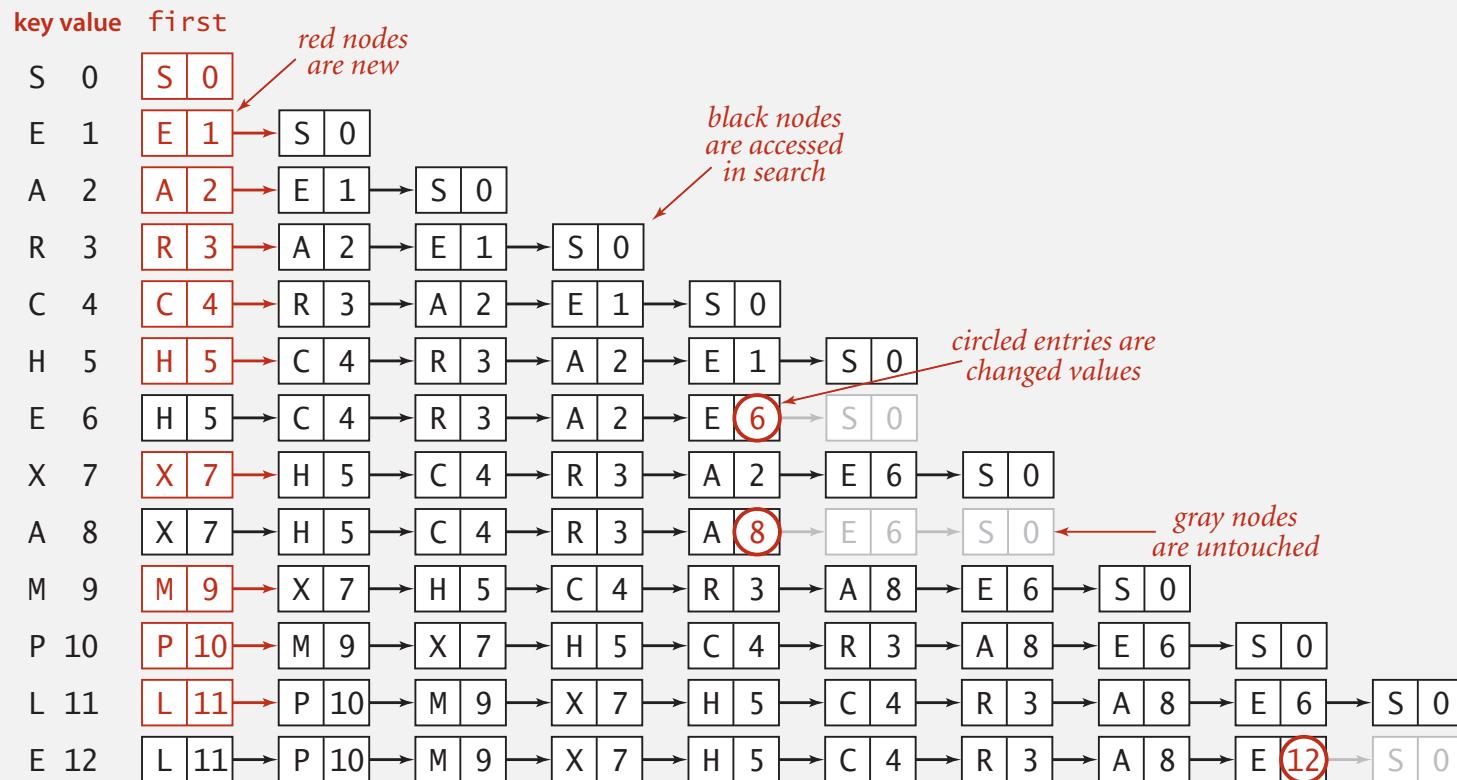
▶ *ordered operations*

Sequential search in a linked list

Data structure. Maintain an (unordered) linked list of key-value pairs.

Search. Scan through all keys until find a match.

Insert. Scan through all keys until find a match; if no match add to front.



Trace of linked-list ST implementation for standard indexing client

Elementary ST implementations: summary

ST implementation	worst-case cost (after N inserts)		average case (after N random inserts)		ordered iteration?	key interface
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	N / 2	N	no	equals()

Challenge. Efficient implementations of both search and insert.

Binary search in an ordered array

Data structure. Maintain an ordered array of key-value pairs.

Rank helper function. How many keys $< k$?

keys[]										
successful search for P	0	1	2	3	4	5	6	7	8	9
lo hi m	0 9 4	A C E H L M P R S X								
	5 9 7	A C E H L M P R S X								
	5 6 5	A C E H L M P R S X								
	6 6 6	A C E H L M P R S X								
entries in black are $a[lo..hi]$										
entry in red is $a[m]$										
loop exits with $keys[m] = P$: return 6										
unsuccessful search for Q	0 9 4	A C E H L M P R S X								
	5 9 7	A C E H L M P R S X								
	5 6 5	A C E H L M P R S X								
	7 6 6	A C E H L M P R S X								
loop exits with $lo > hi$: return 7										

Trace of binary search for rank in an ordered array

Binary search: Java implementation

```
public Value get(Key key)
{
    if (isEmpty()) return null;
    int i = rank(key);
    if (i < N && keys[i].compareTo(key) == 0) return vals[i];
    else return null;
}
```

```
private int rank(Key key)                                number of keys < key
{
    int lo = 0, hi = N-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        int cmp = key.compareTo(keys[mid]);
        if (cmp < 0) hi = mid - 1;
        else if (cmp > 0) lo = mid + 1;
        else if (cmp == 0) return mid;
    }
    return lo;
}
```

Binary search: trace of standard indexing client

Problem. To insert, need to shift all greater keys over.

		keys[]										vals[]										
key	value	0	1	2	3	4	5	6	7	8	9	N	0	1	2	3	4	5	6	7	8	9
S	0	S										1	0									
E	1	E	S									2	1	0								
A	2	A	E	S								3	2	1	0							
R	3	A	E	R	S							4	2	1	3	0						
C	4	A	C	E	R	S						5	2	4	1	3	0					
H	5	A	C	E	H	R	S					6	2	4	1	5	3	0				
E	6	A	C	E	H	R	S					6	2	4	6	5	3	0				
X	7	A	C	E	H	R	S	X				7	2	4	6	5	3	0	7			
A	8	A	C	E	H	R	S	X				7	8	4	6	5	3	0	7			
M	9	A	C	E	H	M	R	S	X			8	8	4	6	5	9	3	0	7		
P	10	A	C	E	H	M	P	R	S	X		9	8	4	6	5	9	10	3	0	7	
L	11	A	C	E	H	L	M	P	R	S	X	10	8	4	6	5	11	9	10	3	0	7
E	12	A	C	E	H	L	M	P	R	S	X	10	8	4	12	5	11	9	10	3	0	7
		A	C	E	H	L	M	P	R	S	X		8	4	12	5	11	9	10	3	0	7

Elementary ST implementations: summary

ST implementation	worst-case cost (after N inserts)		average case (after N random inserts)		ordered iteration?	key interface
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	N / 2	N	no	equals()
binary search (ordered array)	log N	N	log N	N / 2	yes	compareTo()

Challenge. Efficient implementations of both search and insert.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

▶ API

▶ *elementary implementations*

▶ *ordered operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

- ▶ API
- ▶ *elementary implementations*
- ▶ *ordered operations*

Examples of ordered symbol table API

	keys	values
min()	09:00:00	Chicago
	09:00:03	Phoenix
	09:00:13	Houston
get(09:00:13)	09:00:59	Chicago
	09:01:10	Houston
floor(09:05:00)	09:03:13	Chicago
	09:10:11	Seattle
select(7)	09:10:25	Seattle
	09:14:25	Phoenix
	09:19:32	Chicago
	09:19:46	Chicago
keys(09:15:00, 09:25:00)	09:21:05	Chicago
	09:22:43	Seattle
	09:22:54	Seattle
	09:25:52	Chicago
ceiling(09:30:00)	09:35:21	Chicago
	09:36:14	Seattle
max()	09:37:44	Phoenix
size(09:15:00, 09:25:00) is 5		
rank(09:10:25) is 7		

Ordered symbol table API

public class ST<Key extends Comparable<Key>, Value>	
ST()	<i>create an ordered symbol table</i>
void put(Key key, Value val)	<i>put key-value pair into the table (remove key from table if value is null)</i>
Value get(Key key)	<i>value paired with key (null if key is absent)</i>
void delete(Key key)	<i>remove key (and its value) from table</i>
boolean contains(Key key)	<i>is there a value paired with key?</i>
boolean isEmpty()	<i>is the table empty?</i>
int size()	<i>number of key-value pairs</i>
Key min()	<i>smallest key</i>
Key max()	<i>largest key</i>
Key floor(Key key)	<i>largest key less than or equal to key</i>
Key ceiling(Key key)	<i>smallest key greater than or equal to key</i>
int rank(Key key)	<i>number of keys less than key</i>
Key select(int k)	<i>key of rank k</i>
void deleteMin()	<i>delete smallest key</i>
void deleteMax()	<i>delete largest key</i>
int size(Key lo, Key hi)	<i>number of keys in [lo..hi]</i>
Iterable<Key> keys(Key lo, Key hi)	<i>keys in [lo..hi], in sorted order</i>
Iterable<Key> keys()	<i>all keys in the table, in sorted order</i>

Binary search: ordered symbol table operations summary

	sequential search	binary search
search	N	$\lg N$
insert / delete	N	N
min / max	N	1
floor / ceiling	N	$\lg N$
rank	N	$\lg N$
select	N	1
ordered iteration	$N \log N$	N

order of growth of the running time for ordered symbol table operations

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

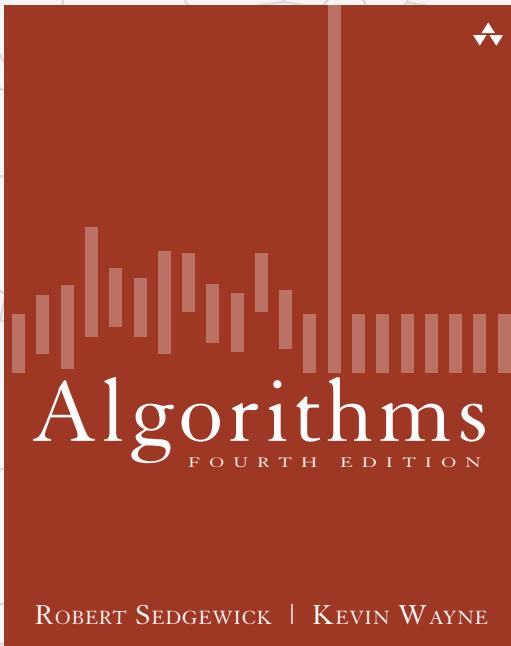
<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

- ▶ API
- ▶ *elementary implementations*
- ▶ *ordered operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



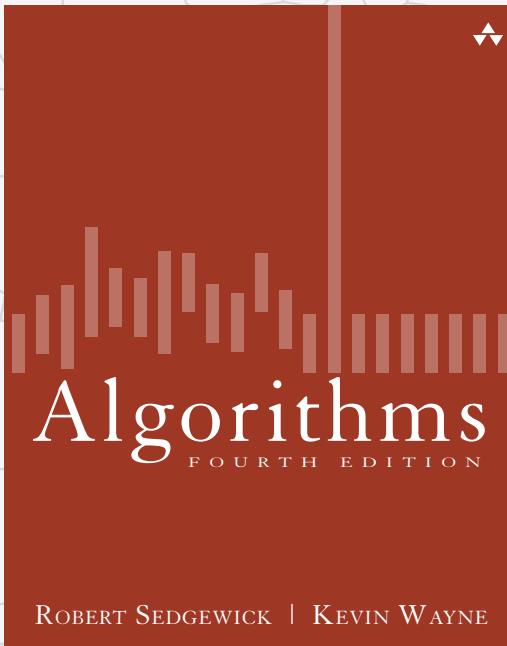
<http://algs4.cs.princeton.edu>

3.1 SYMBOL TABLES

- ▶ API
- ▶ *elementary implementations*
- ▶ *ordered operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

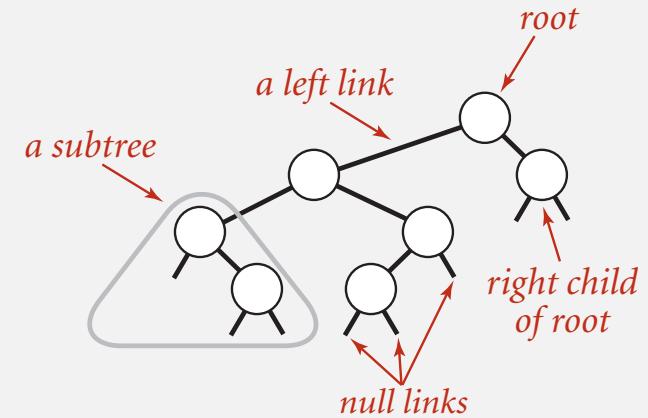
- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

Binary search trees

Definition. A BST is a binary tree in symmetric order.

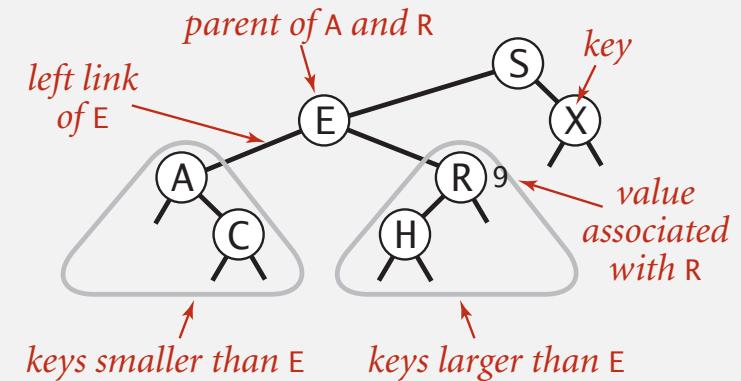
A binary tree is either:

- Empty.
- Two disjoint binary trees (left and right).



Symmetric order. Each node has a key, and every node's key is:

- Larger than all keys in its left subtree.
- Smaller than all keys in its right subtree.

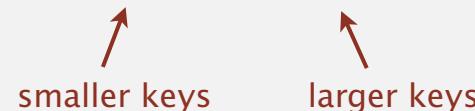


BST representation in Java

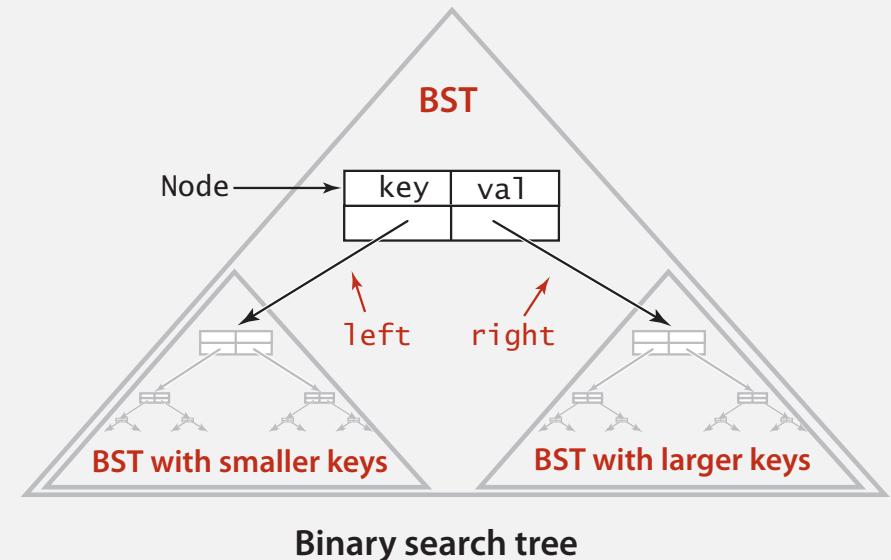
Java definition. A BST is a reference to a root Node.

A Node is comprised of four fields:

- A Key and a Value.
- A reference to the left and right subtree.



```
private class Node
{
    private Key key;
    private Value val;
    private Node left, right;
    public Node(Key key, Value val)
    {
        this.key = key;
        this.val = val;
    }
}
```



Key and Value are generic types; Key is Comparable

BST implementation (skeleton)

```
public class BST<Key extends Comparable<Key>, Value>
{
    private Node root;                                     ← root of BST

    private class Node
    { /* see previous slide */ }

    public void put(Key key, Value val)
    { /* see next slides */ }

    public Value get(Key key)
    { /* see next slides */ }

    public void delete(Key key)
    { /* see next slides */ }

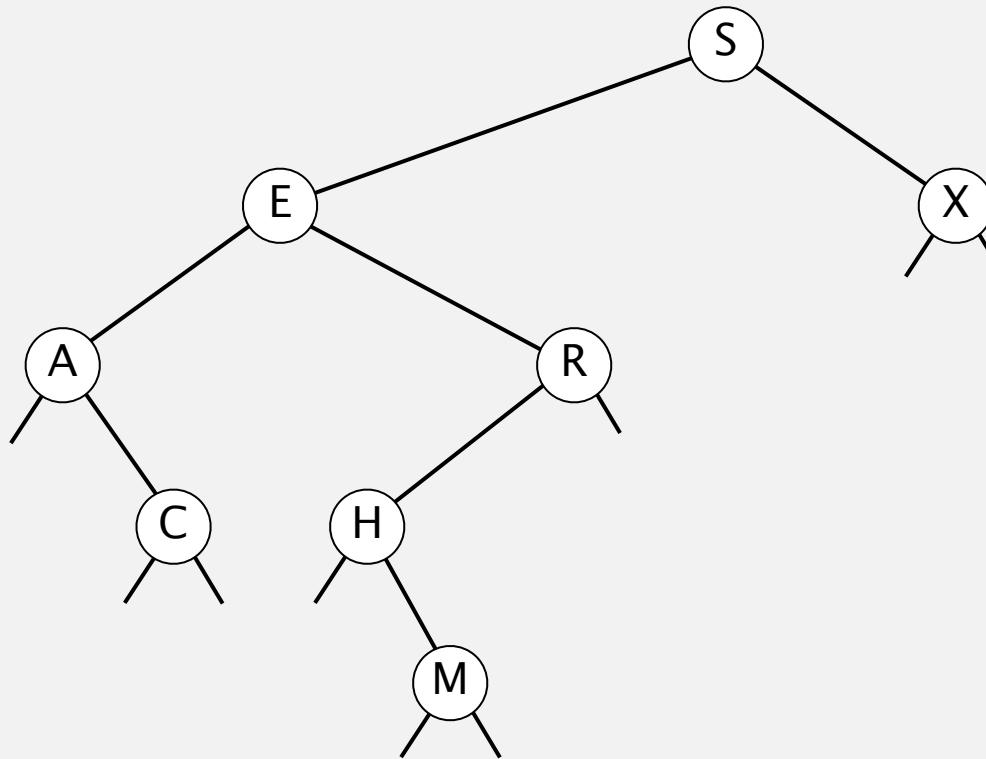
    public Iterable<Key> iterator()
    { /* see next slides */ }

}
```

Binary search tree demo

Search. If less, go left; if greater, go right; if equal, search hit.

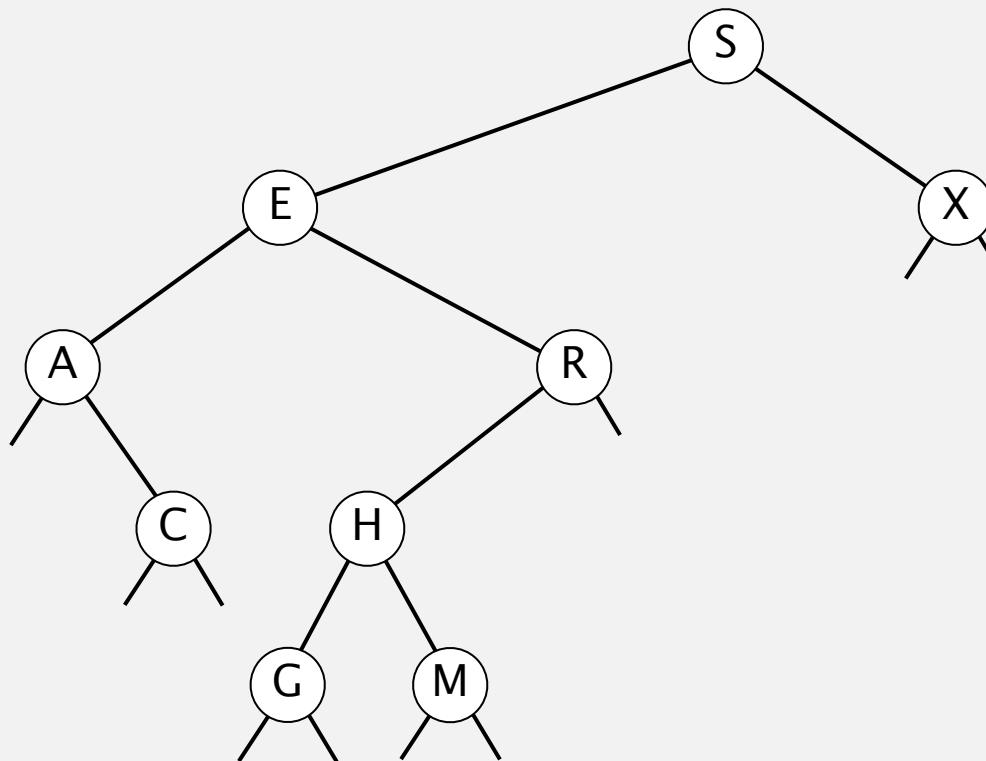
successful search for H



Binary search tree demo

Insert. If less, go left; if greater, go right; if null, insert.

insert G



BST search: Java implementation

Get. Return value corresponding to given key, or null if no such key.

```
public Value get(Key key)
{
    Node x = root;
    while (x != null)
    {
        int cmp = key.compareTo(x.key);
        if      (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
        else if (cmp == 0) return x.val;
    }
    return null;
}
```

Cost. Number of compares is equal to 1 + depth of node.

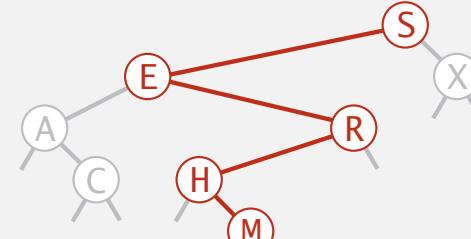
BST insert

Put. Associate value with key.

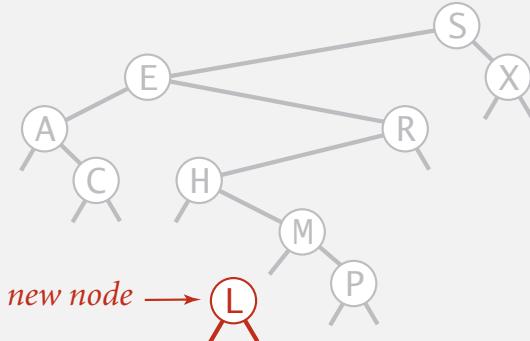
Search for key, then two cases:

- Key in tree \Rightarrow reset value.
- Key not in tree \Rightarrow add new node.

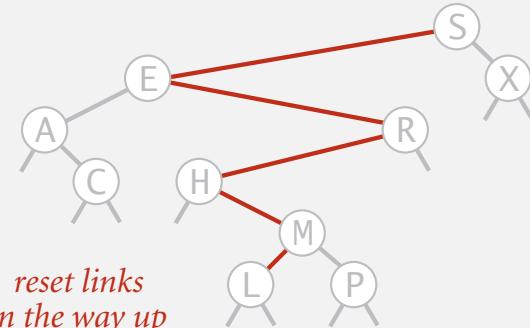
inserting L



search for L ends
at this null link



create new node → (L)



reset links
on the way up

Insertion into a BST

BST insert: Java implementation

Put. Associate value with key.

```
public void put(Key key, Value val)
{   root = put(root, key, val); }

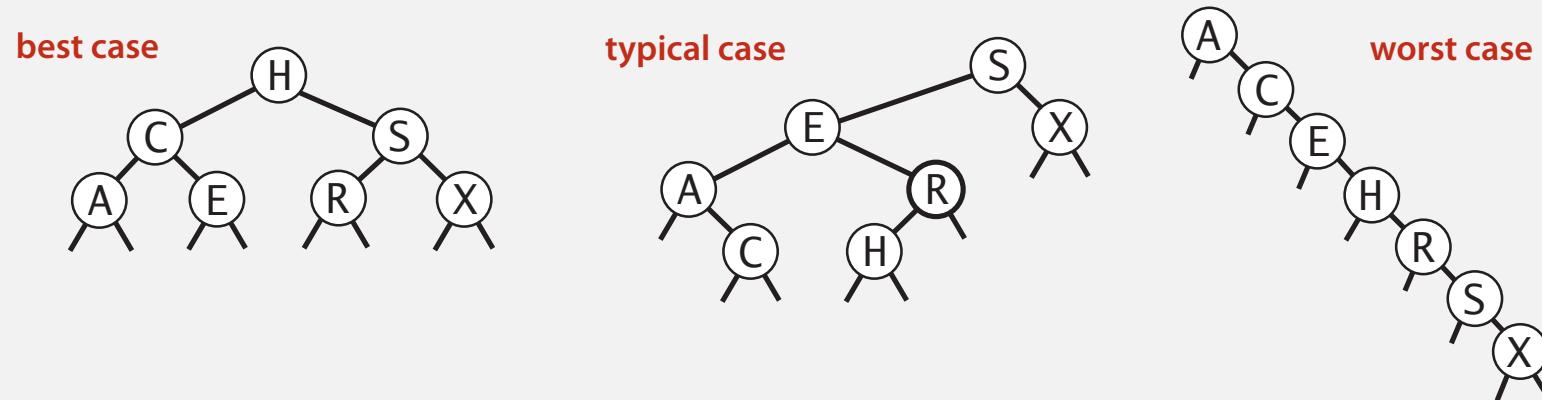
private Node put(Node x, Key key, Value val)
{
    if (x == null) return new Node(key, val);
    int cmp = key.compareTo(x.key);
    if      (cmp  < 0)
        x.left  = put(x.left,  key, val);
    else if (cmp  > 0)
        x.right = put(x.right, key, val);
    else if (cmp == 0)
        x.val = val;
    return x;
}
```

concise, but tricky,
recursive code;
read carefully!

Cost. Number of compares is equal to 1 + depth of node.

Tree shape

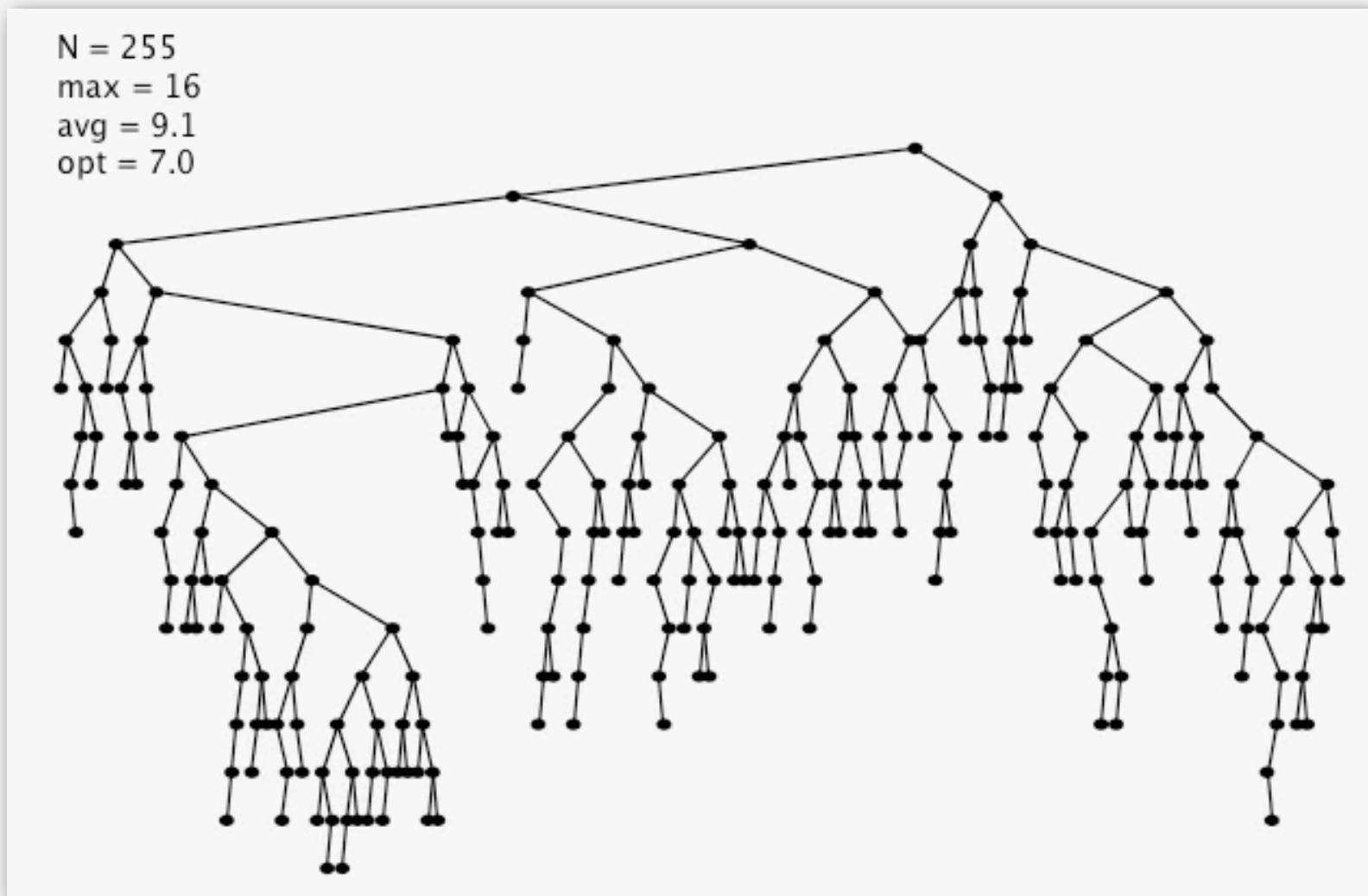
- Many BSTs correspond to same set of keys.
- Number of compares for search/insert is equal to $1 + \text{depth of node}$.



Remark. Tree shape depends on order of insertion.

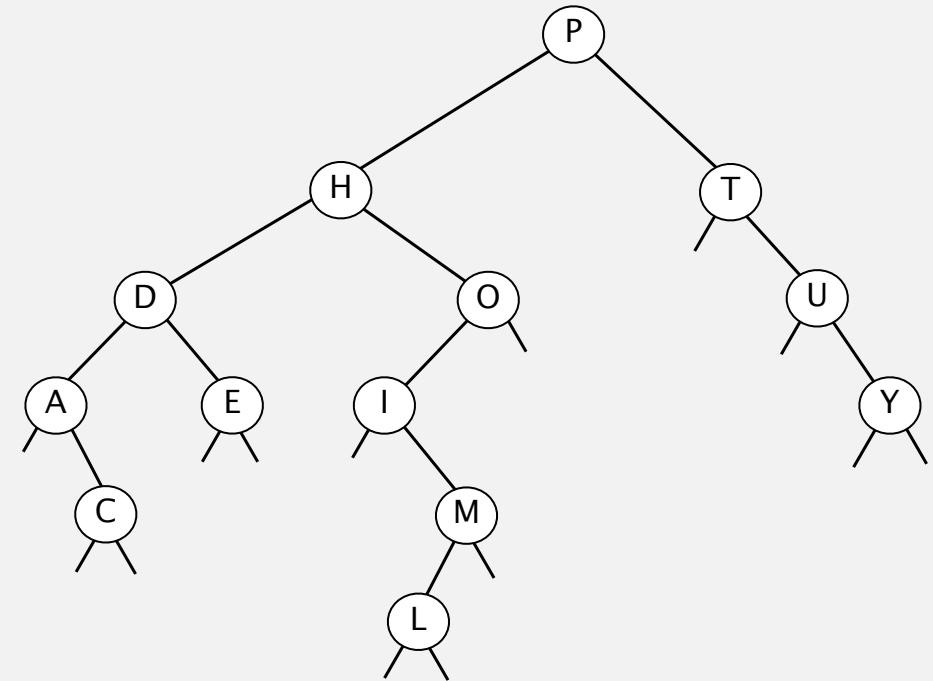
BST insertion: random order visualization

Ex. Insert keys in random order.



Correspondence between BSTs and quicksort partitioning

0	1	2	3	4	5	6	7	8	9	10	11	12	13
P	S	E	U	D	O	M	Y	T	H	I	C	A	L
P	S	E	U	D	O	M	Y	T	H	I	C	A	L
H	L	E	A	D	O	M	C	I	P	T	Y	U	S
D	C	E	A	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	O	M	L	I	P	T	Y	U	S
A	C	D	E	H	I	M	L	O	P	T	Y	U	S
A	C	D	E	H	I	M	L	O	P	T	Y	U	S
A	C	D	E	H	I	L	M	O	P	T	Y	U	S
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y
A	C	D	E	H	I	L	M	O	P	S	T	U	Y



Remark. Correspondence is 1-1 if array has no duplicate keys.

BSTs: mathematical analysis

Proposition. If N distinct keys are inserted into a BST in **random** order, the expected number of compares for a search/insert is $\sim 2 \ln N$.

Pf. 1-1 correspondence with quicksort partitioning.

Proposition. [Reed, 2003] If N distinct keys are inserted in random order, expected height of tree is $\sim 4.311 \ln N$.

How Tall is a Tree?

Bruce Reed
CNRS, Paris, France
reed@moka.ccr.jussieu.fr

ABSTRACT

Let H_n be the height of a random binary search tree on n nodes. We show that there exists constants $\alpha = 4.31107\dots$ and $\beta = 1.95\dots$ such that $\mathbf{E}(H_n) = \alpha \log n - \beta \log \log n + O(1)$. We also show that $\text{Var}(H_n) = O(1)$.

But... Worst-case height is N .
(exponentially small chance when keys are inserted in random order)

ST implementations: summary

implementation	guarantee		average case		ordered ops?	operations on keys
	search	insert	search hit	insert		
sequential search (unordered list)	N	N	N/2	N	no	<code>equals()</code>
binary search (ordered array)	$\lg N$	N	$\lg N$	N/2	yes	<code>compareTo()</code>
BST	N	N	$1.39 \lg N$	$1.39 \lg N$	next	<code>compareTo()</code>

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

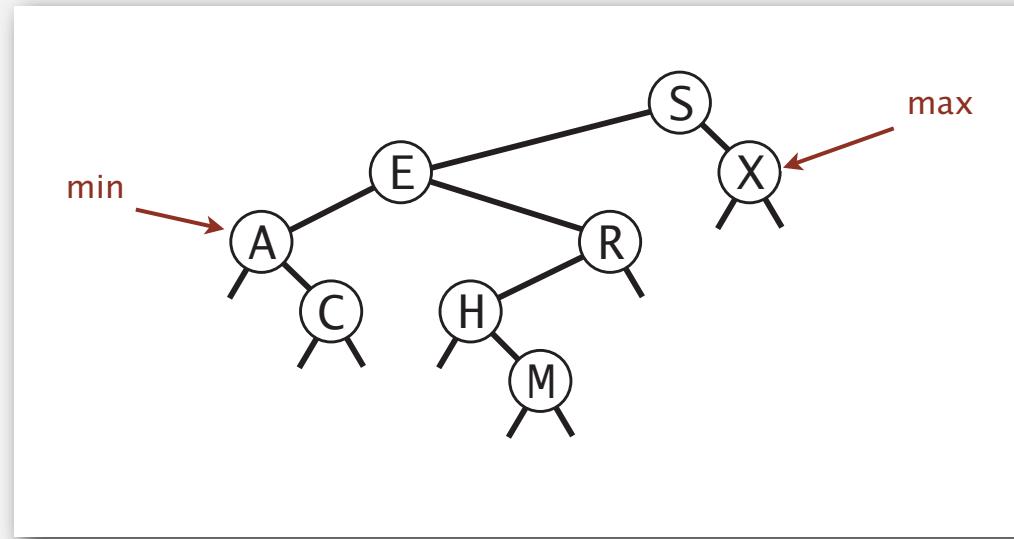
3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

Minimum and maximum

Minimum. Smallest key in table.

Maximum. Largest key in table.

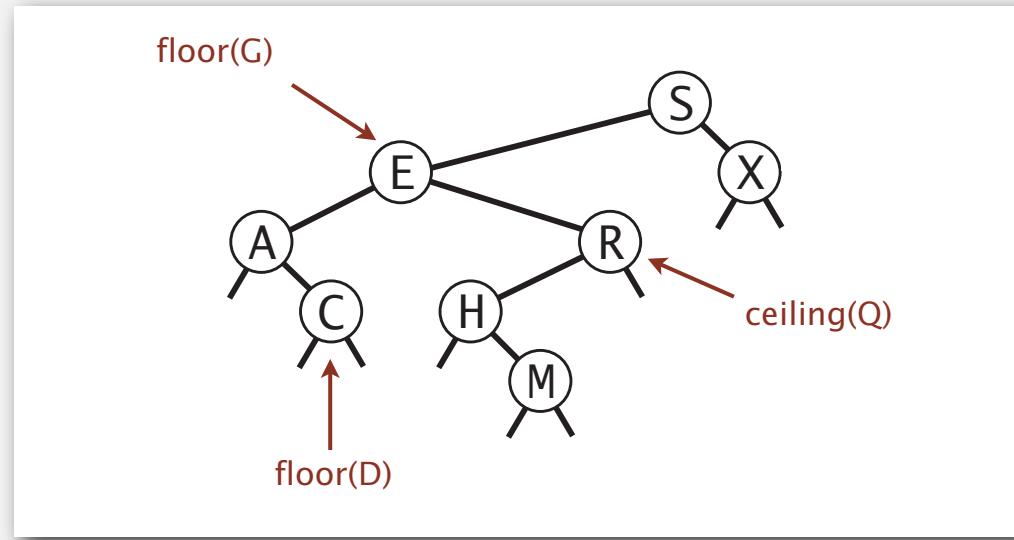


Q. How to find the min / max?

Floor and ceiling

Floor. Largest key \leq to a given key.

Ceiling. Smallest key \geq to a given key.



Q. How to find the floor /ceiling?

Computing the floor

Case 1. [k equals the key at root]

The floor of k is k .

Case 2. [k is less than the key at root]

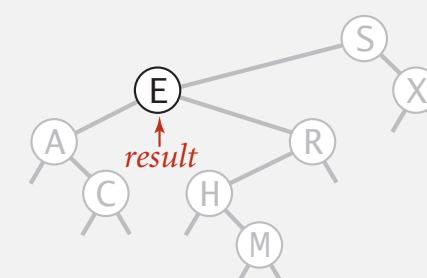
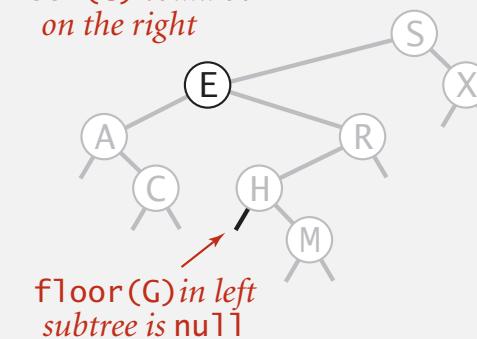
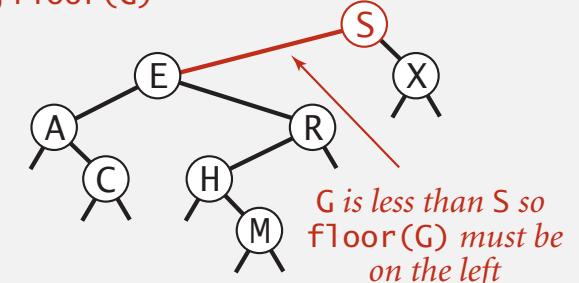
The floor of k is in the left subtree.

Case 3. [k is greater than the key at root]

The floor of k is in the right subtree

(if there is **any** key $\leq k$ in right subtree);
otherwise it is the key in the root.

finding $\text{floor}(G)$



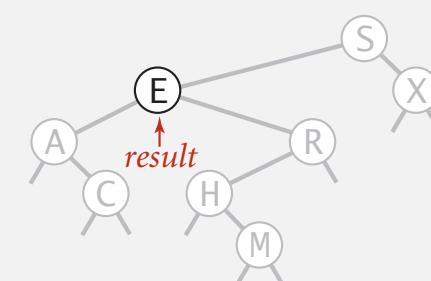
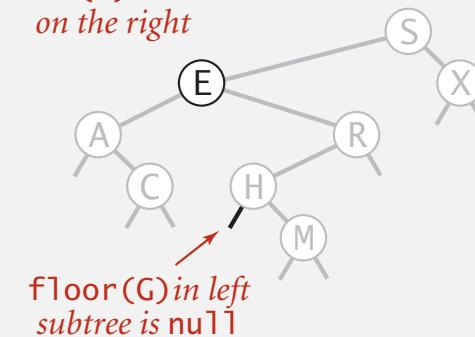
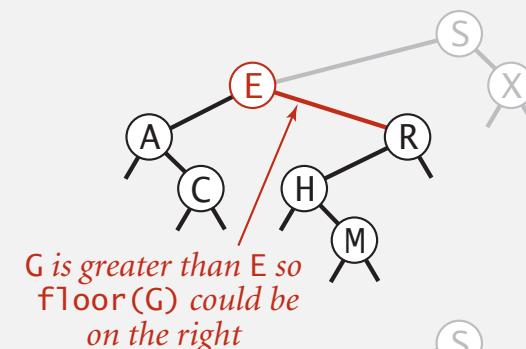
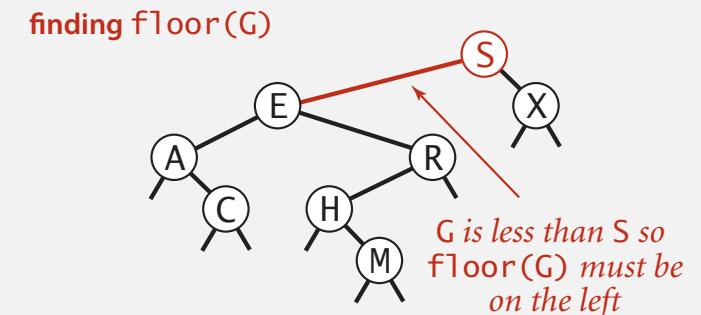
Computing the floor

```
public Key floor(Key key)
{
    Node x = floor(root, key);
    if (x == null) return null;
    return x.key;
}
private Node floor(Node x, Key key)
{
    if (x == null) return null;
    int cmp = key.compareTo(x.key);

    if (cmp == 0) return x;

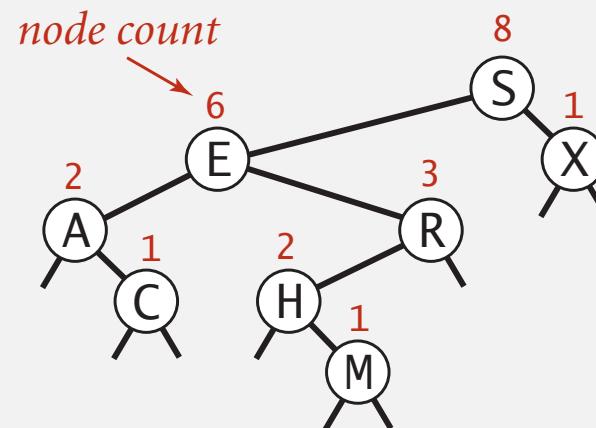
    if (cmp < 0) return floor(x.left, key);

    Node t = floor(x.right, key);
    if (t != null) return t;
    else           return x;
}
```



Subtree counts

In each node, we store the number of nodes in the subtree rooted at that node; to implement `size()`, return the count at the root.



Remark. This facilitates efficient implementation of `rank()` and `select()`.

BST implementation: subtree counts

```
private class Node
{
    private Key key;
    private Value val;
    private Node left;
    private Node right;
    private int count;
}
```

number of nodes
in subtree

```
public int size()
{   return size(root); }
```

```
private int size(Node x)
{
    if (x == null) return 0;
    return x.count; }
```

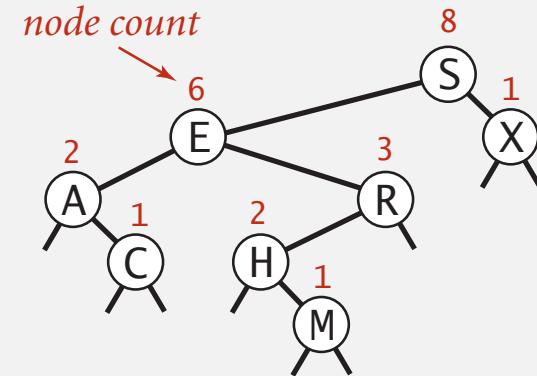
ok to call
when x is null

```
private Node put(Node x, Key key, Value val)
{
    if (x == null) return new Node(key, val);
    int cmp = key.compareTo(x.key);
    if      (cmp < 0) x.left  = put(x.left,  key, val);
    else if (cmp > 0) x.right = put(x.right, key, val);
    else if (cmp == 0) x.val   = val;
    x.count = 1 + size(x.left) + size(x.right);
    return x;
}
```

Rank

Rank. How many keys $< k$?

Easy recursive algorithm (3 cases!)



```
public int rank(Key key)
{   return rank(key, root);  }

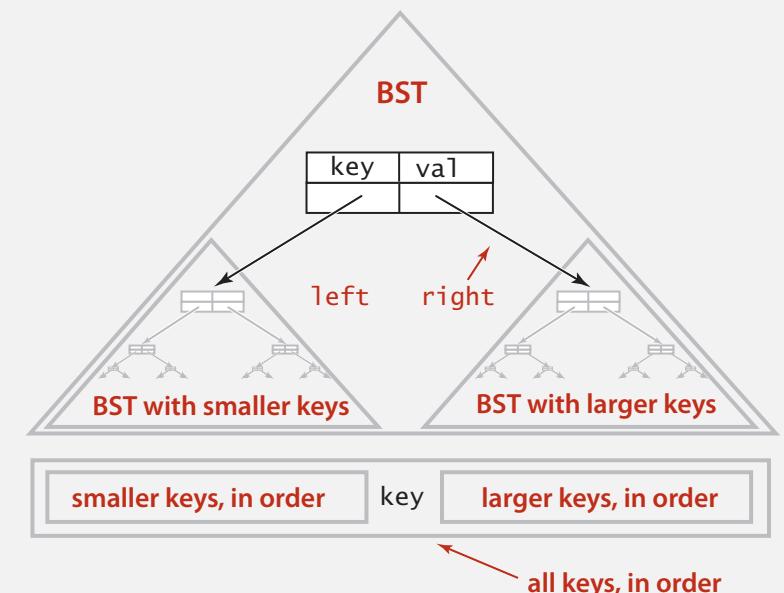
private int rank(Key key, Node x)
{
    if (x == null) return 0;
    int cmp = key.compareTo(x.key);
    if      (cmp < 0) return rank(key, x.left);
    else if (cmp > 0) return 1 + size(x.left) + rank(key, x.right);
    else if (cmp == 0) return size(x.left);
}
```

Inorder traversal

- Traverse left subtree.
- Enqueue key.
- Traverse right subtree.

```
public Iterable<Key> keys()
{
    Queue<Key> q = new Queue<Key>();
    inorder(root, q);
    return q;
}

private void inorder(Node x, Queue<Key> q)
{
    if (x == null) return;
    inorder(x.left, q);
    q.enqueue(x.key);
    inorder(x.right, q);
}
```



Property. Inorder traversal of a BST yields keys in ascending order.

BST: ordered symbol table operations summary

	sequential search	binary search	BST
search	N	$\lg N$	h
insert	N	N	h
min / max	N	1	h
floor / ceiling	N	$\lg N$	h
rank	N	$\lg N$	h
select	N	1	h
ordered iteration	$N \log N$	N	N

$h =$ height of BST
(proportional to $\log N$
if keys inserted in random order)

order of growth of running time of ordered symbol table operations

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

ST implementations: summary

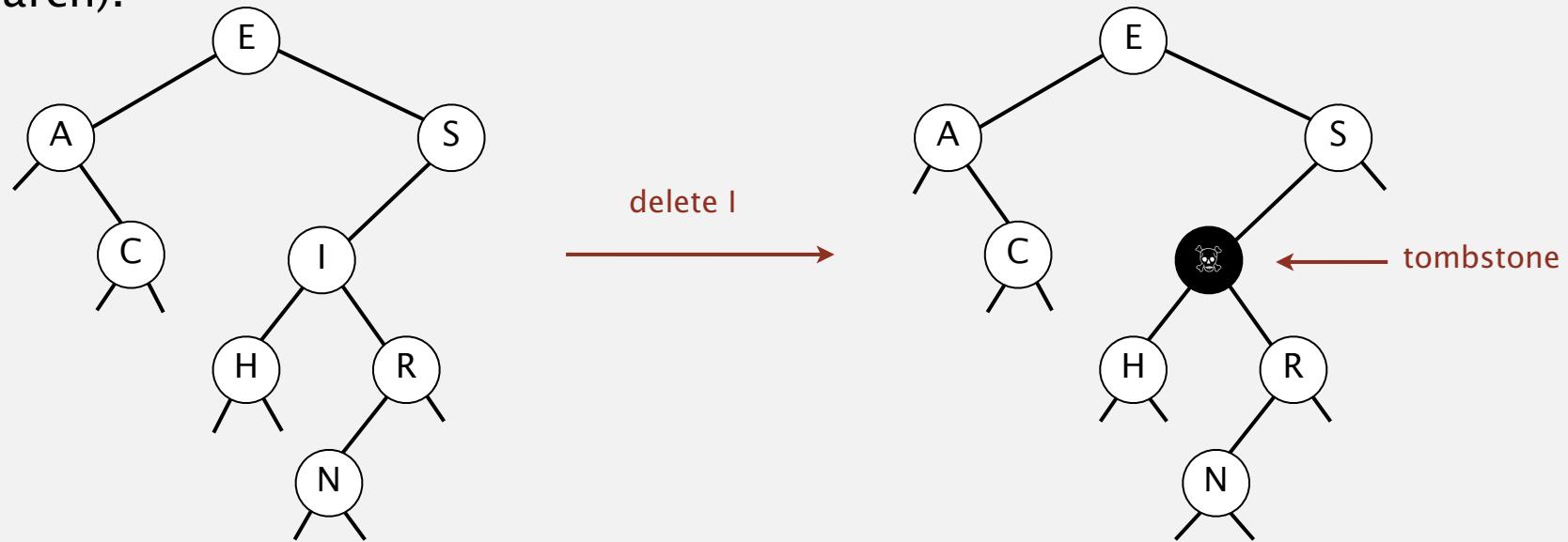
implementation	guarantee			average case			ordered iteration?	operations on keys
	search	insert	delete	search hit	insert	delete		
sequential search (linked list)	N	N	N	N/2	N	N/2	no	<code>equals()</code>
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	<code>compareTo()</code>
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$???	yes	<code>compareTo()</code>

Next. Deletion in BSTs.

BST deletion: lazy approach

To remove a node with a given key:

- Set its value to null.
- Leave key in tree to guide searches (but don't consider it equal in search).



Cost. $\sim 2 \ln N'$ per insert, search, and delete (if keys in random order), where N' is the number of key-value pairs ever inserted in the BST.

Unsatisfactory solution. Tombstone (memory) overload.

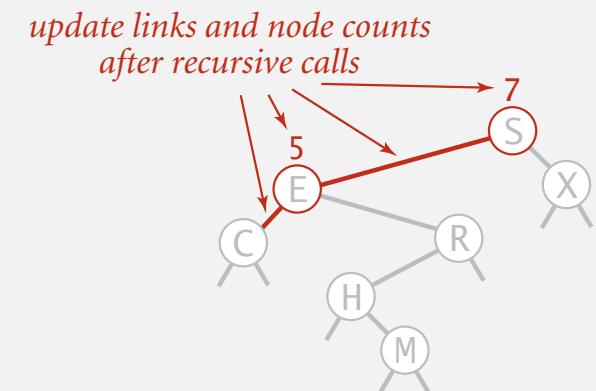
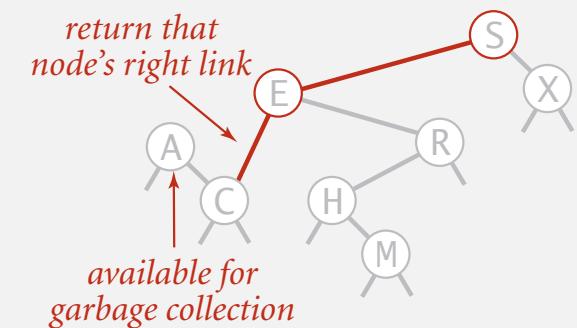
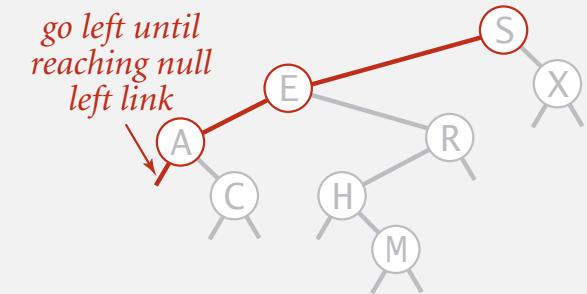
Deleting the minimum

To delete the minimum key:

- Go left until finding a node with a null left link.
- Replace that node by its right link.
- Update subtree counts.

```
public void deleteMin()
{   root = deleteMin(root);  }

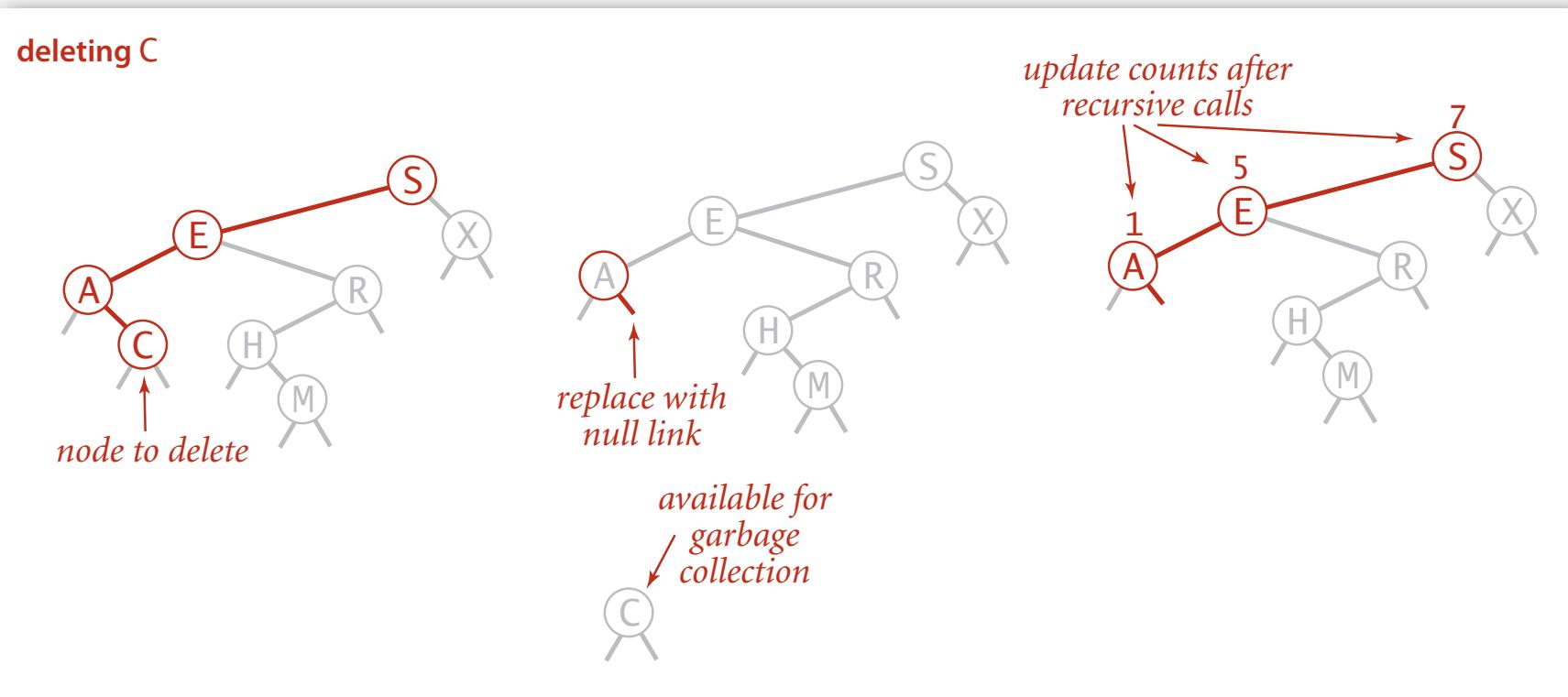
private Node deleteMin(Node x)
{
    if (x.left == null) return x.right;
    x.left = deleteMin(x.left);
    x.count = 1 + size(x.left) + size(x.right);
    return x;
}
```



Hibbard deletion

To delete a node with key k: search for node t containing key k.

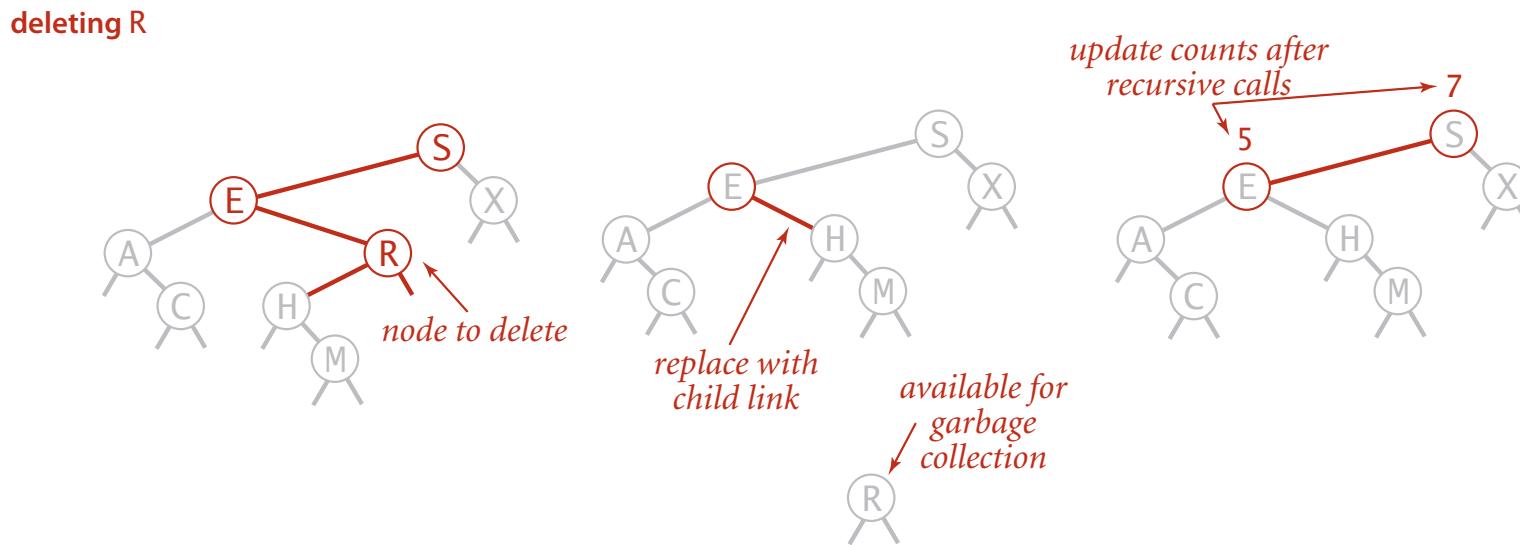
Case 0. [0 children] Delete t by setting parent link to null.



Hibbard deletion

To delete a node with key k: search for node t containing key k.

Case 1. [1 child] Delete t by replacing parent link.



Hibbard deletion

To delete a node with key k : search for node t containing key k .

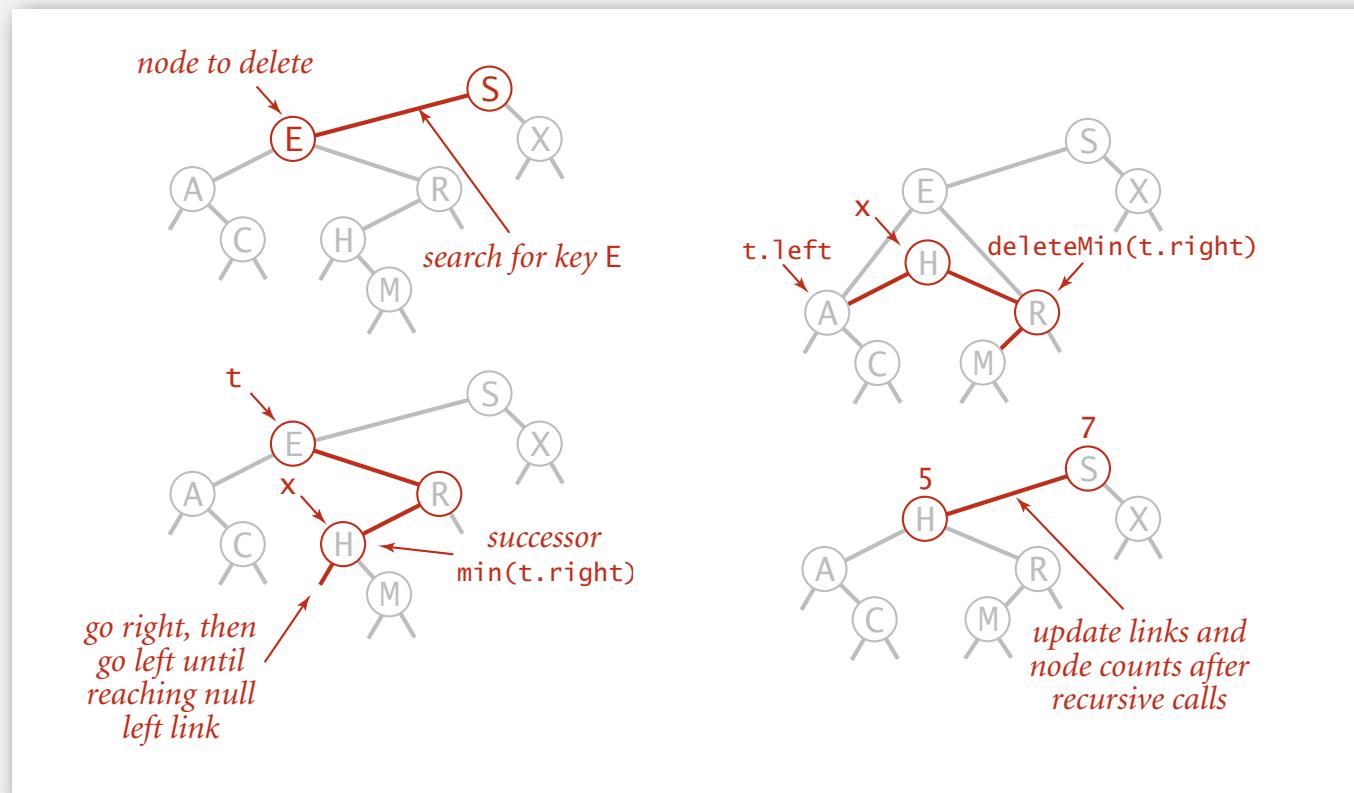
Case 2. [2 children]

- Find successor x of t .
- Delete the minimum in t 's right subtree.
- Put x in t 's spot.

← x has no left child

← but don't garbage collect x

← still a BST



Hibbard deletion: Java implementation

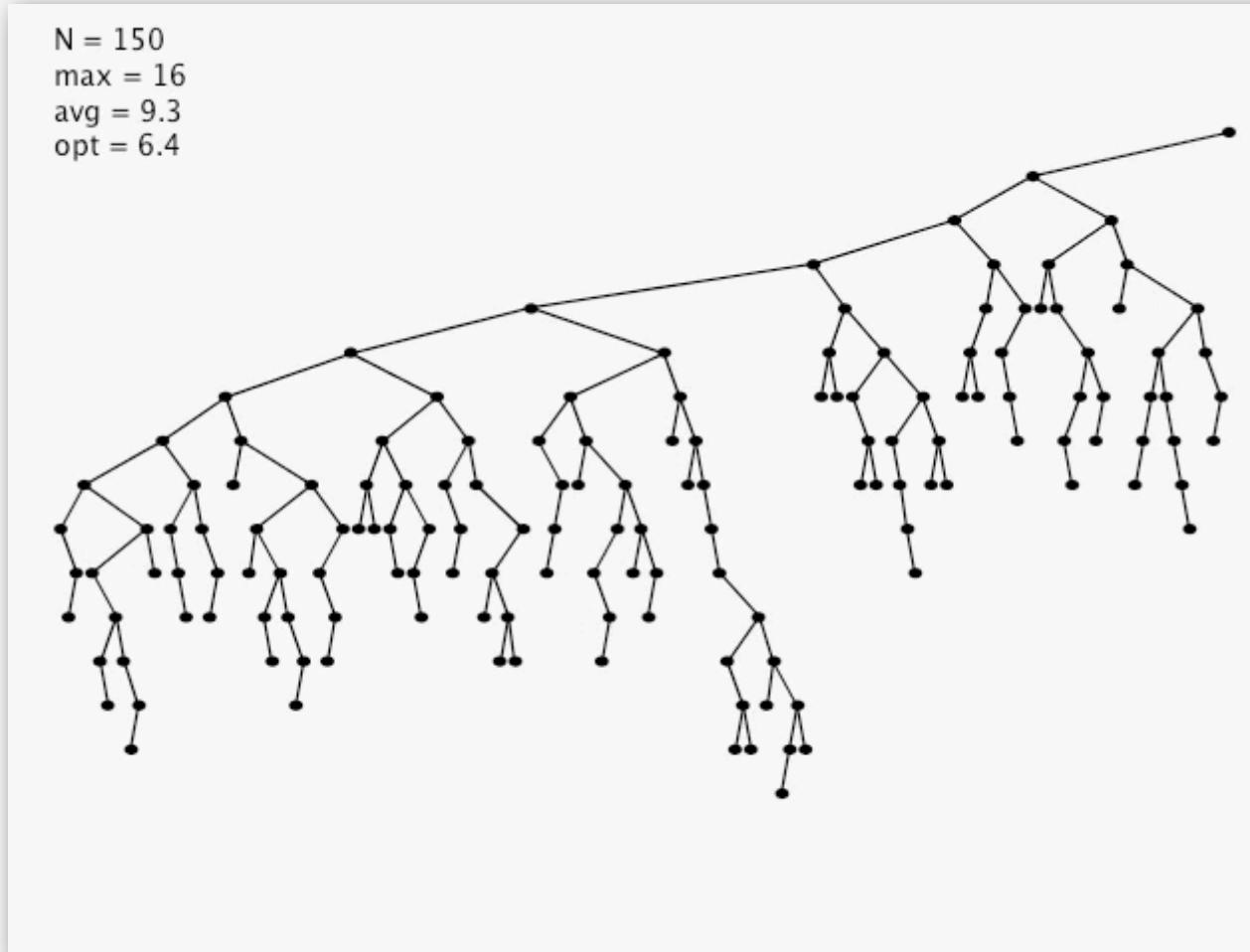
```
public void delete(Key key)
{  root = delete(root, key);  }

private Node delete(Node x, Key key) {
    if (x == null) return null;
    int cmp = key.compareTo(x.key);
    if      (cmp < 0) x.left  = delete(x.left,  key); ← search for key
    else if (cmp > 0) x.right = delete(x.right, key);
    else {
        if (x.right == null) return x.left; ← no right child

        Node t = x;
        x = min(t.right);
        x.right = deleteMin(t.right); ← replace with successor
        x.left = t.left;
    }
    x.count = size(x.left) + size(x.right) + 1; ← update subtree
    return x;                                     counts
}
```

Hibbard deletion: analysis

Unsatisfactory solution. Not symmetric.



Surprising consequence. Trees not random (!) $\Rightarrow \sqrt{N}$ per op.

Longstanding open problem. Simple and efficient delete for BSTs.

ST implementations: summary

implementation	guarantee			average case			ordered iteration?	operations on keys
	search	insert	delete	search hit	insert	delete		
sequential search (linked list)	N	N	N	N/2	N	N/2	no	<code>equals()</code>
binary search (ordered array)	$\lg N$	N	N	$\lg N$	$N/2$	$N/2$	yes	<code>compareTo()</code>
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$	\sqrt{N}	yes	<code>compareTo()</code>

other operations also become \sqrt{N}
if deletions allowed

Red-black BST. Guarantee logarithmic performance for all operations.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

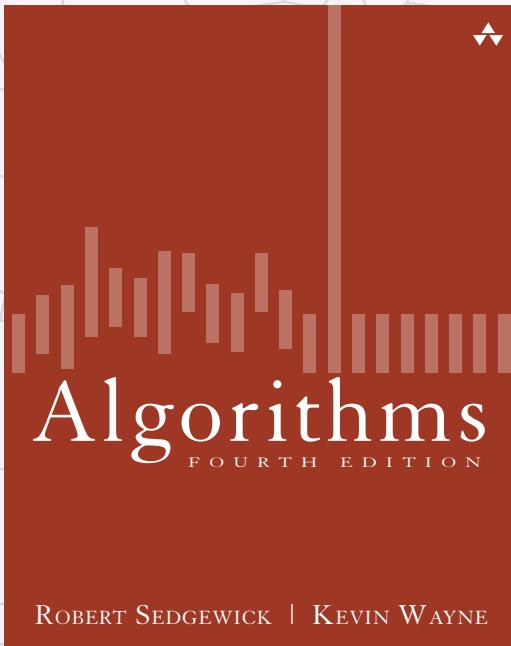
<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*

Algorithms

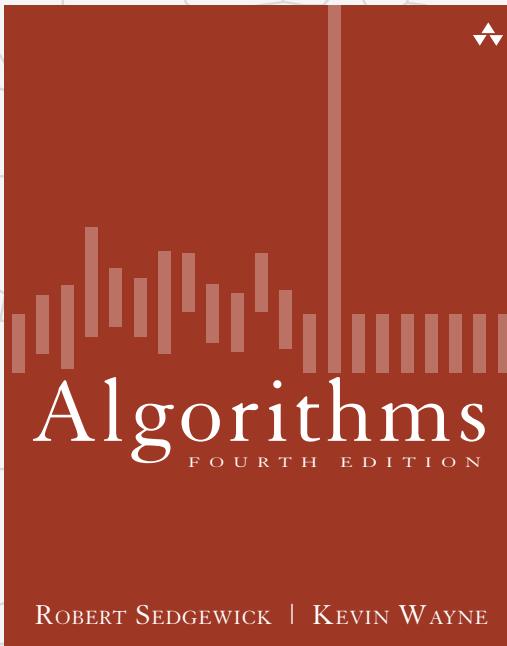
ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

3.2 BINARY SEARCH TREES

- ▶ *BSTs*
- ▶ *ordered operations*
- ▶ *deletion*



<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ 2-3 *search trees*
- ▶ *red-black BSTs*
- ▶ *B-trees*

Symbol table review

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compareTo()
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$?	yes	compareTo()
goal	$\log N$	$\log N$	$\log N$	$\log N$	$\log N$	$\log N$	yes	compareTo()

Challenge. Guarantee performance.

This lecture. 2-3 trees, left-leaning red-black BSTs, B-trees.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

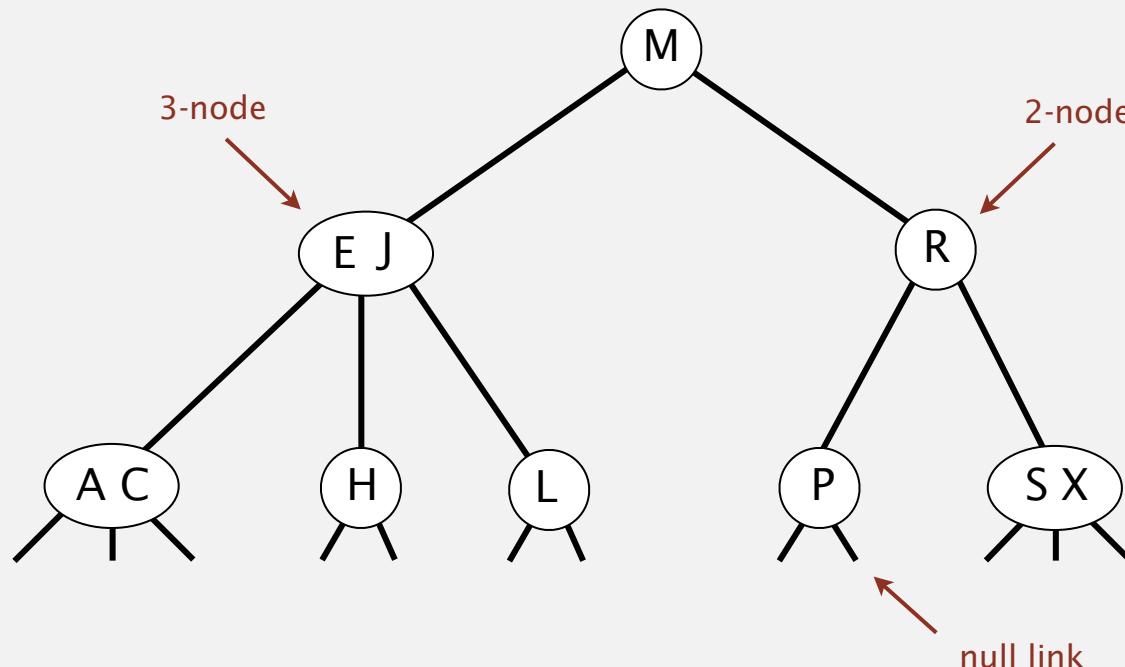
- ▶ 2-3 search trees
- ▶ red-black BSTs
- ▶ B-trees

2-3 tree

Allow 1 or 2 keys per node.

- 2-node: one key, two children.
- 3-node: two keys, three children.

Perfect balance. Every path from root to null link has same length.



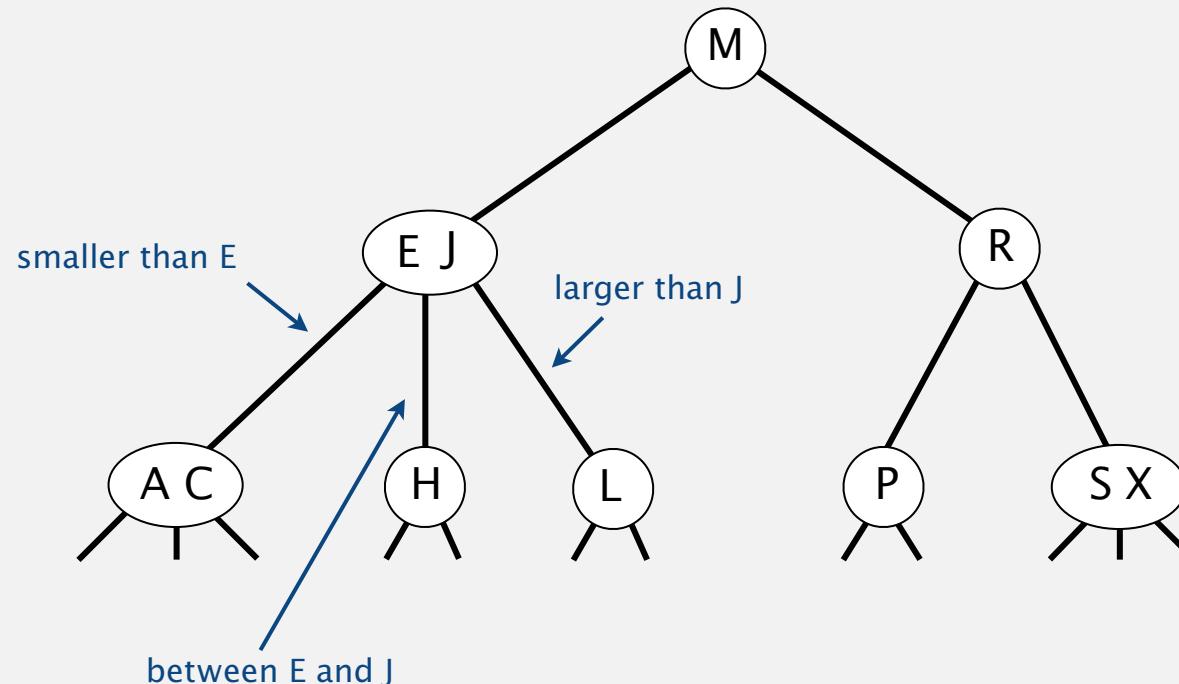
2-3 tree

Allow 1 or 2 keys per node.

- 2-node: one key, two children.
- 3-node: two keys, three children.

Perfect balance. Every path from root to null link has same length.

Symmetric order. Inorder traversal yields keys in ascending order.



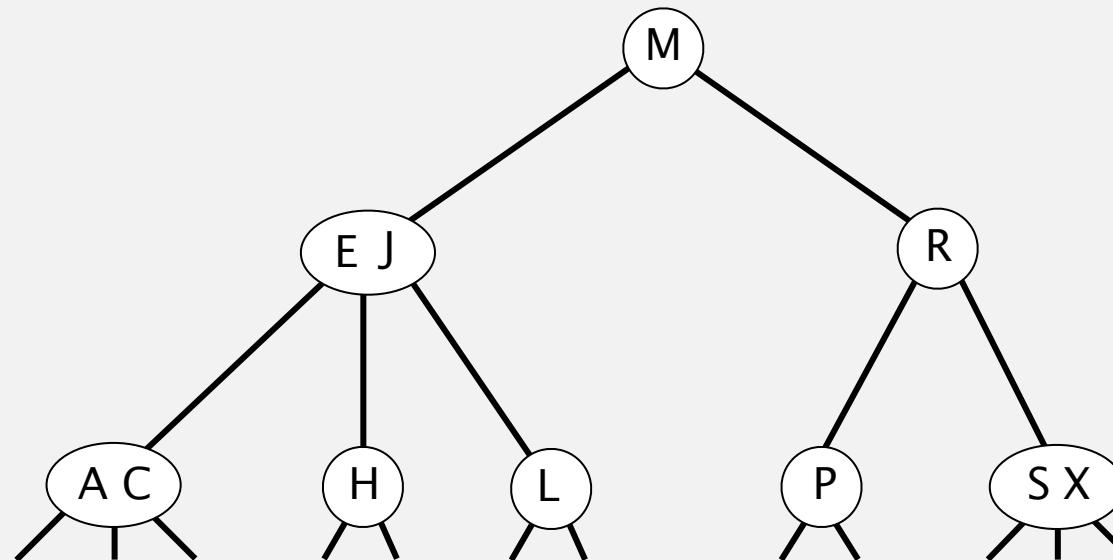
2-3 tree demo

Search.

- Compare search key against keys in node.
- Find interval containing search key.
- Follow associated link (recursively).



search for H

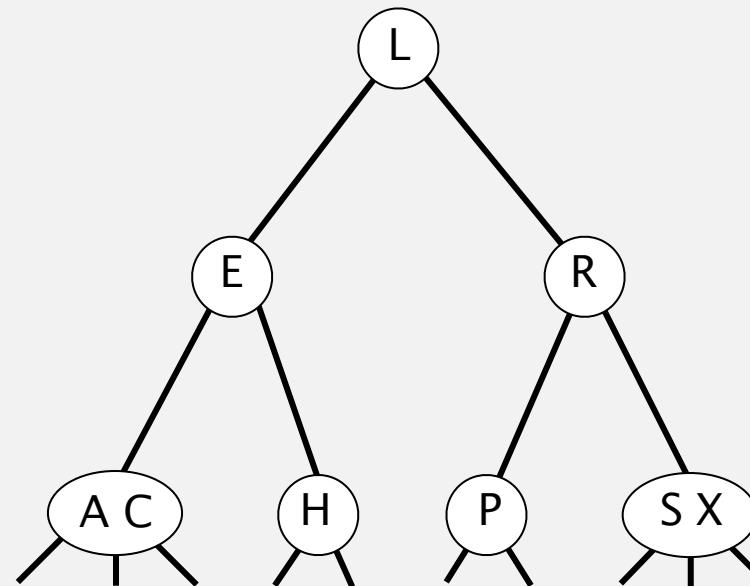


2-3 tree demo

Insertion into a 3-node at bottom.

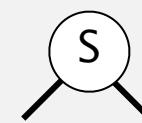
- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it's a 4-node, split it into three 2-nodes.

insert L



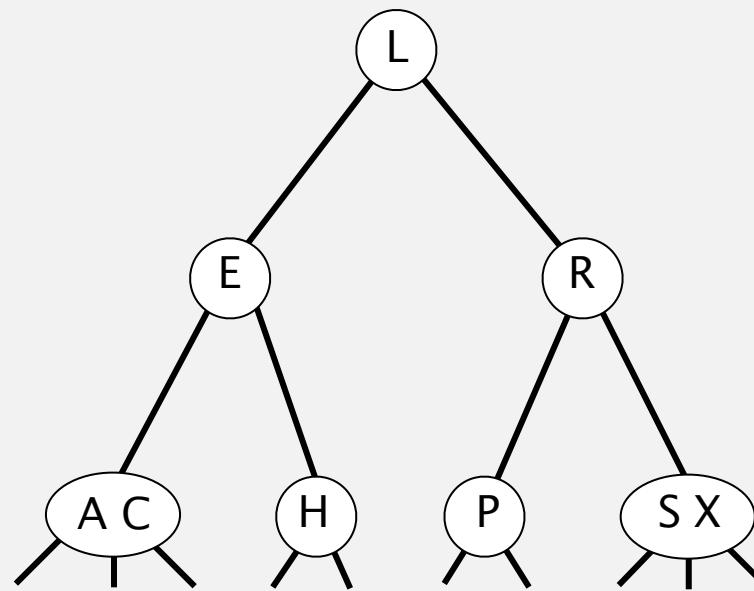
2-3 tree construction demo

insert S



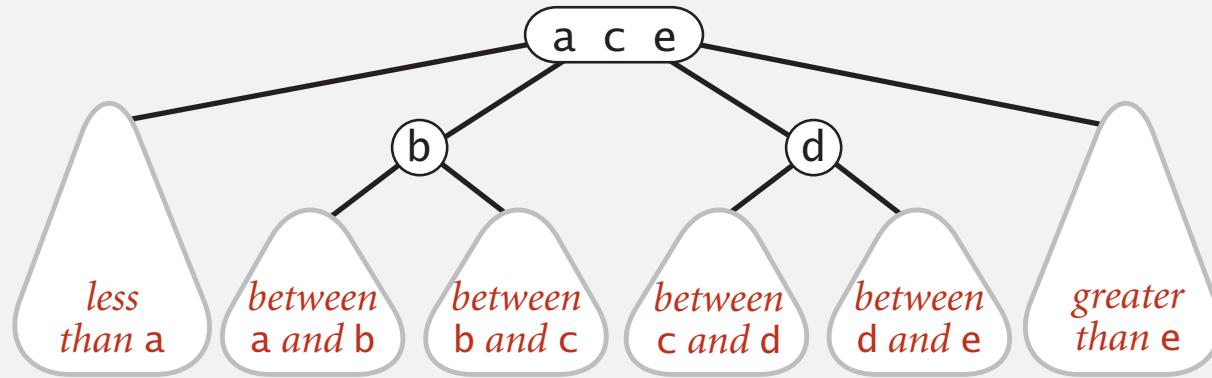
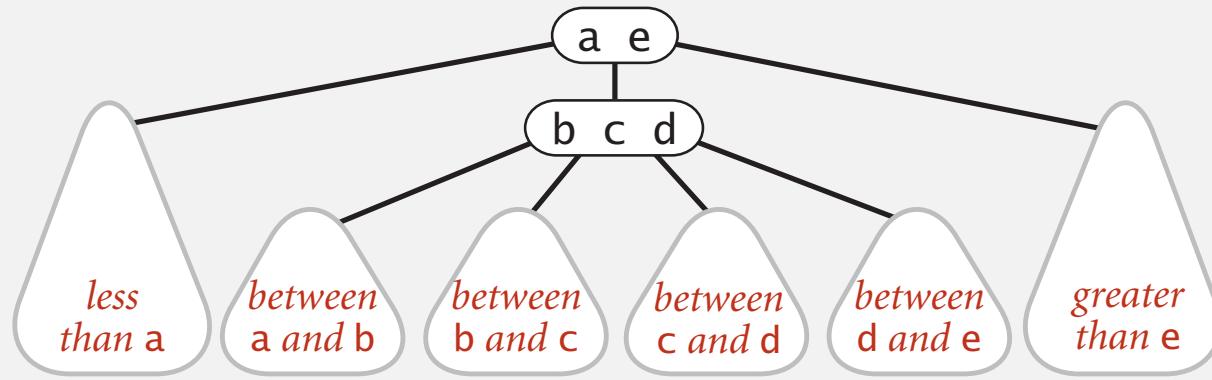
2-3 tree construction demo

2-3 tree



Local transformations in a 2-3 tree

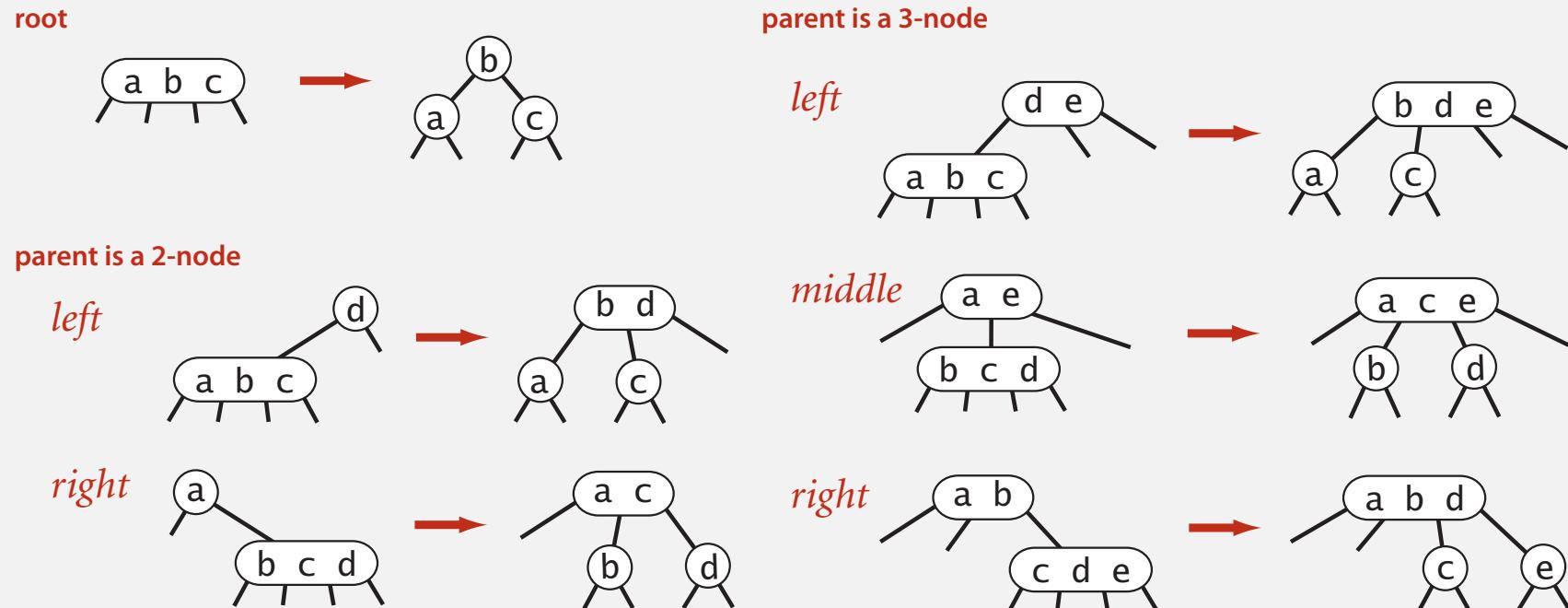
Splitting a 4-node is a **local** transformation: constant number of operations.



Global properties in a 2-3 tree

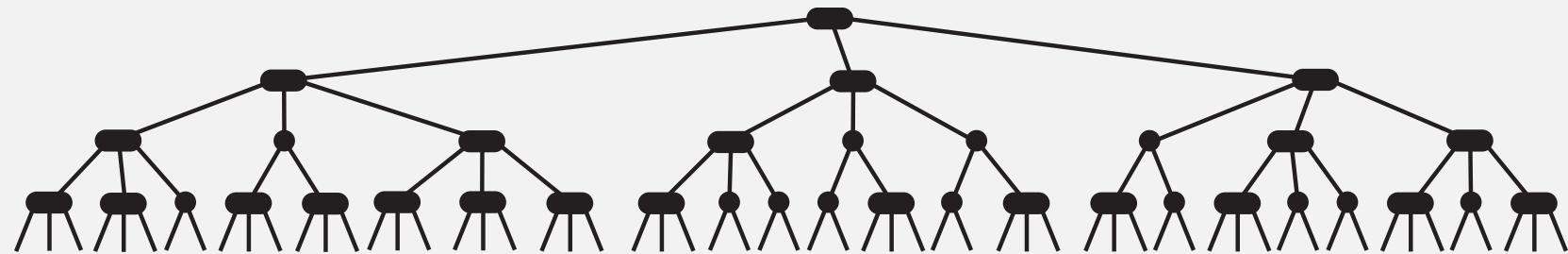
Invariants. Maintains symmetric order and perfect balance.

Pf. Each transformation maintains symmetric order and perfect balance.



2-3 tree: performance

Perfect balance. Every path from root to null link has same length.

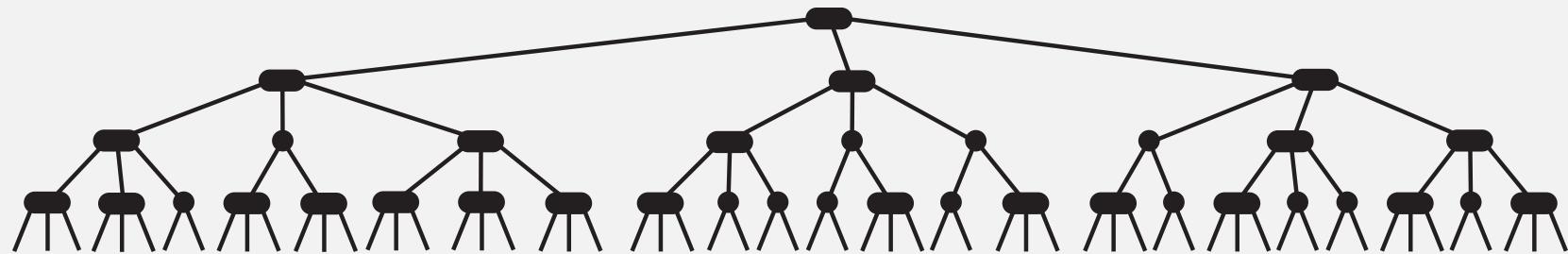


Tree height.

- Worst case:
- Best case:

2-3 tree: performance

Perfect balance. Every path from root to null link has same length.



Tree height.

- Worst case: $\lg N$. [all 2-nodes]
- Best case: $\log_3 N \approx .631 \lg N$. [all 3-nodes]
- Between 12 and 20 for a million nodes.
- Between 18 and 30 for a billion nodes.

Guaranteed **logarithmic** performance for search and insert.

ST implementations: summary

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compareTo()
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$?	yes	compareTo()
2-3 tree	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	yes	compareTo()



 constants depend upon implementation

2-3 tree: implementation?

Direct implementation is complicated, because:

- Maintaining multiple node types is cumbersome.
- Need multiple compares to move down tree.
- Need to move back up the tree to split 4-nodes.
- Large number of cases for splitting.

Bottom line. Could do it, but there's a better way.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ 2-3 search trees
- ▶ red-black BSTs
- ▶ B-trees

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

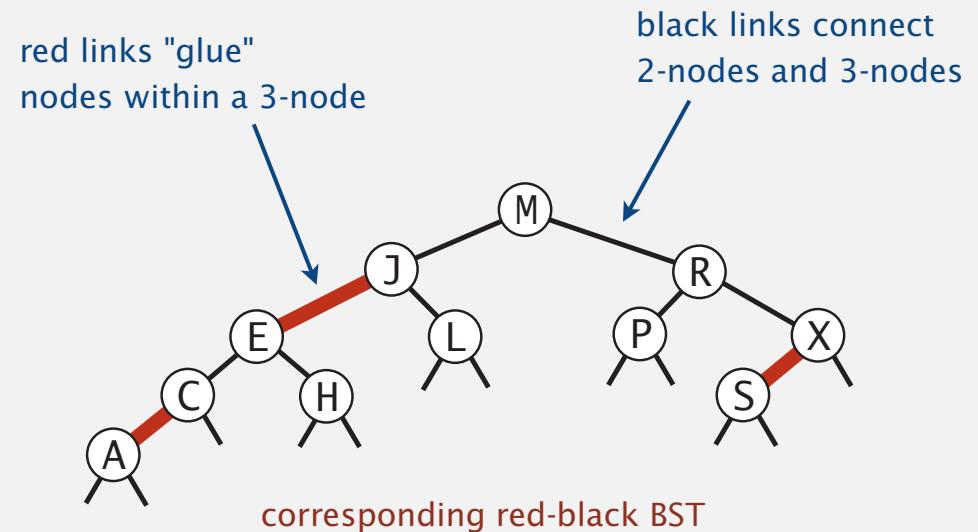
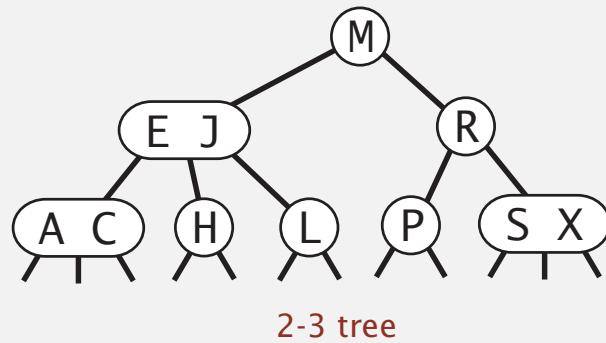
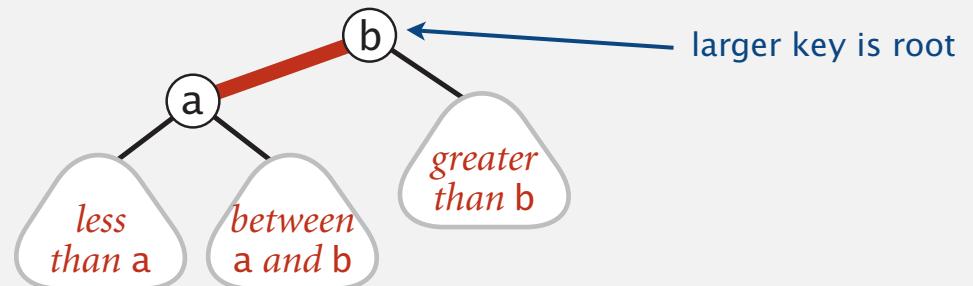
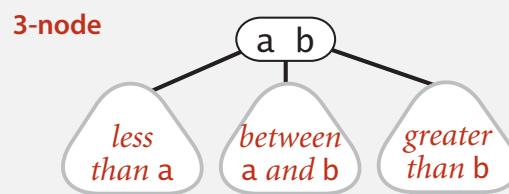
<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ *2-3 search trees*
- ▶ *red-black BSTs*
- ▶ *B-trees*

Left-leaning red-black BSTs (Guibas-Sedgewick 1979 and Sedgewick 2007)

1. Represent 2–3 tree as a BST.
2. Use "internal" left-leaning links as "glue" for 3-nodes.

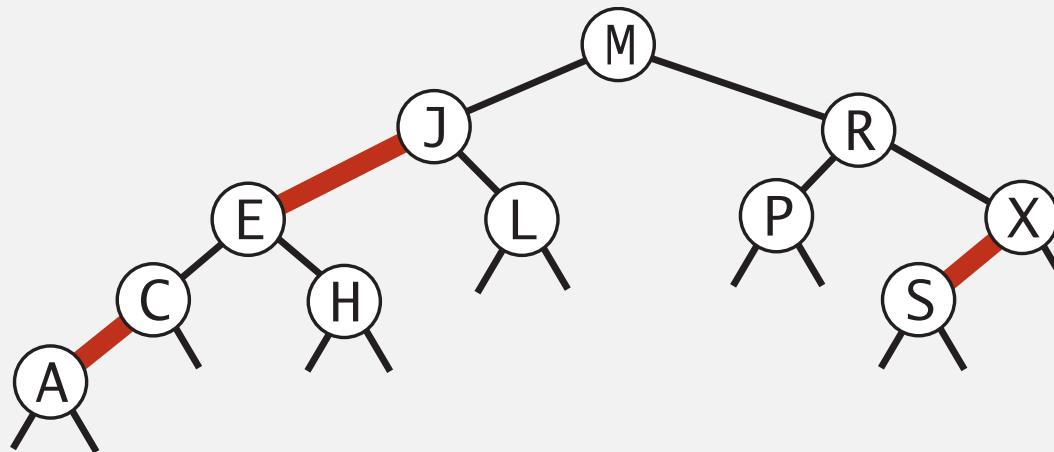


An equivalent definition

A BST such that:

- No node has two red links connected to it.
- Every path from root to null link has the same number of black links.
- Red links lean left.

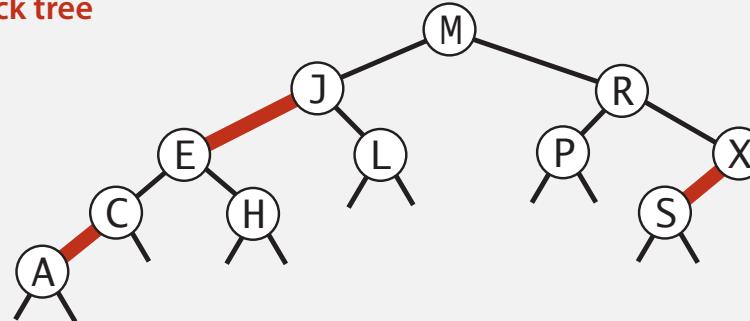
"perfect black balance"



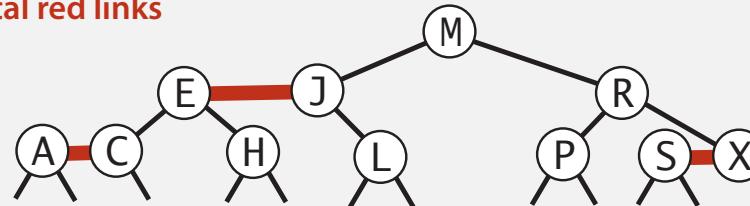
Left-leaning red-black BSTs: 1-1 correspondence with 2-3 trees

Key property. 1–1 correspondence between 2–3 and LLRB.

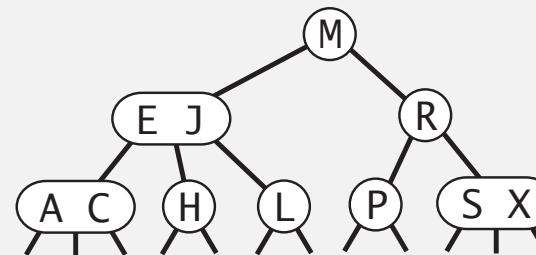
red–black tree



horizontal red links



2-3 tree

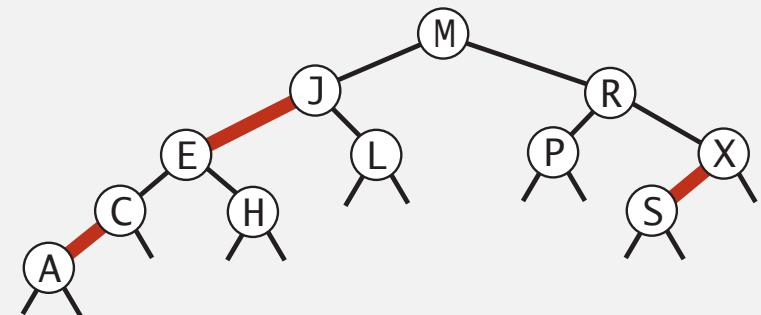


Search implementation for red-black BSTs

Observation. Search is the same as for elementary BST (ignore color).

↑
but runs faster because of better balance

```
public Val get(Key key)
{
    Node x = root;
    while (x != null)
    {
        int cmp = key.compareTo(x.key);
        if      (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
        else if (cmp == 0) return x.val;
    }
    return null;
}
```



Remark. Most other ops (e.g., floor, iteration, selection) are also identical.

Red-black BST representation

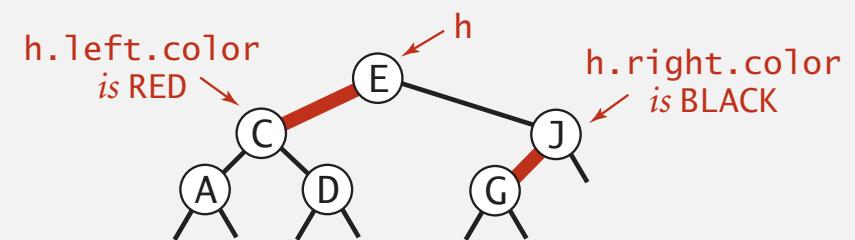
Each node is pointed to by precisely one link (from its parent) \Rightarrow
can encode color of links in nodes.

```
private static final boolean RED  = true;
private static final boolean BLACK = false;
```

```
private class Node
{
    Key key;
    Value val;
    Node left, right;
    boolean color; // color of parent link
}
```

```
private boolean isRed(Node x)
{
    if (x == null) return false;
    return x.color == RED;
}
```

null links are black

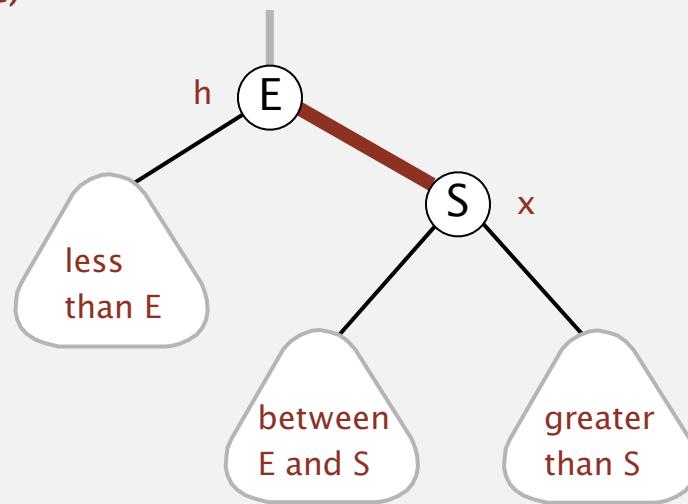


Elementary red-black BST operations

Left rotation. Orient a (temporarily) right-leaning red link to lean left.

rotate E left

(before)



```
private Node rotateLeft(Node h)
{
    assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

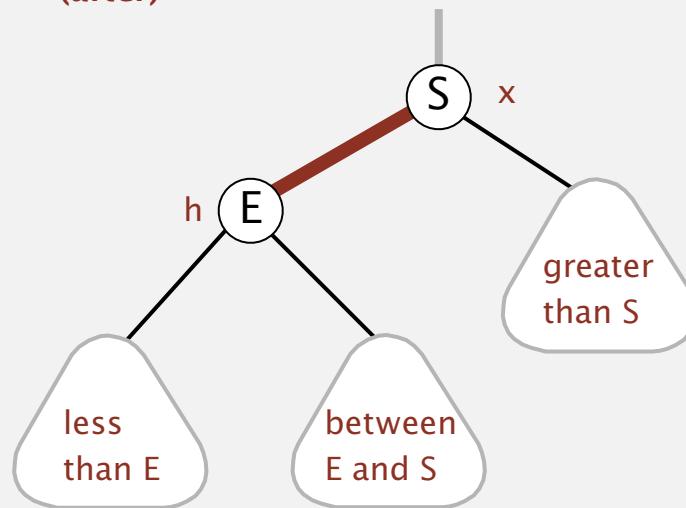
Invariants. Maintains symmetric order and perfect black balance.

Elementary red-black BST operations

Left rotation. Orient a (temporarily) right-leaning red link to lean left.

rotate E left

(after)



```
private Node rotateLeft(Node h)
{
    assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

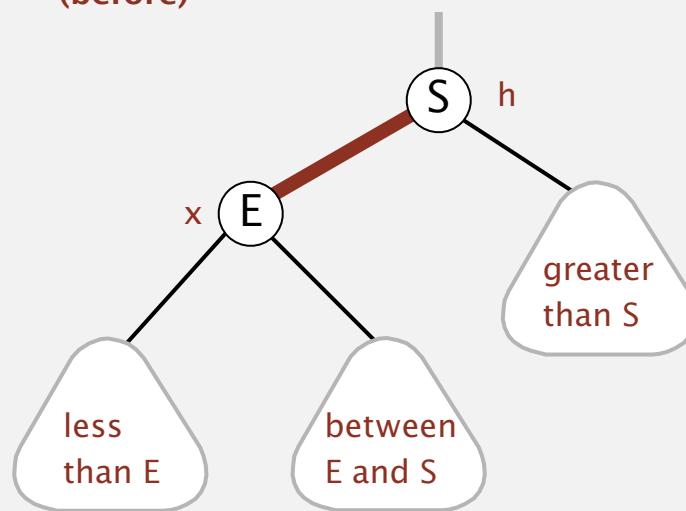
Invariants. Maintains symmetric order and perfect black balance.

Elementary red-black BST operations

Right rotation. Orient a left-leaning red link to (temporarily) lean right.

rotate S right

(before)



```
private Node rotateRight(Node h)
{
    assert isRed(h.left);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

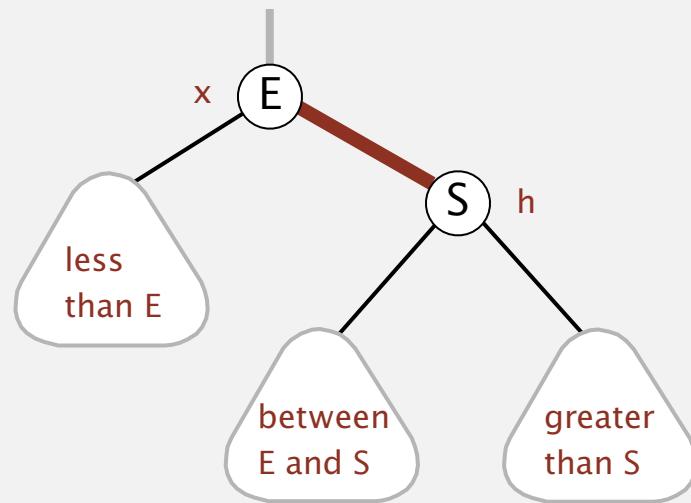
Invariants. Maintains symmetric order and perfect black balance.

Elementary red-black BST operations

Right rotation. Orient a left-leaning red link to (temporarily) lean right.

rotate S right

(after)

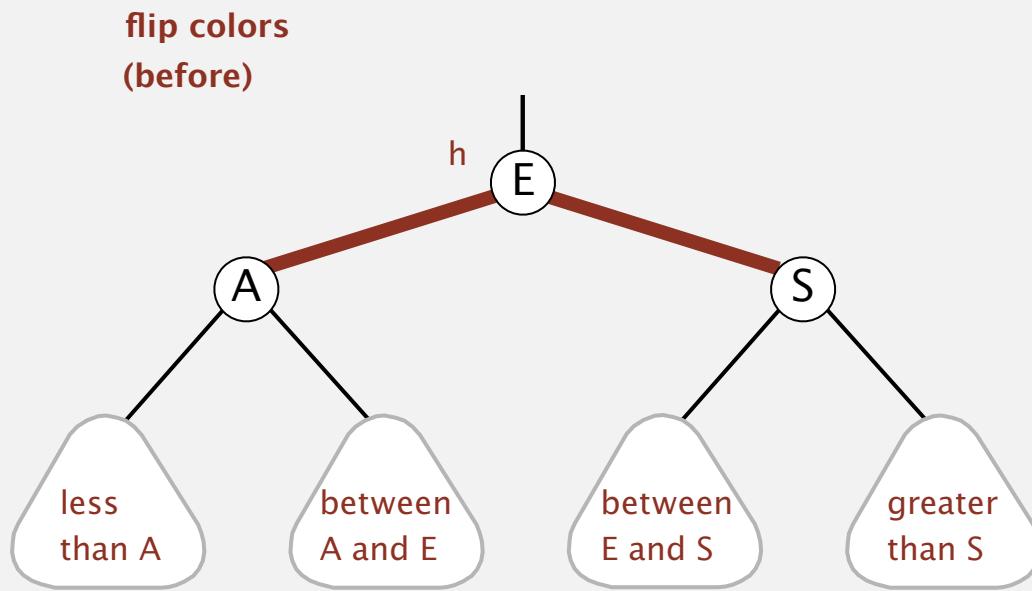


```
private Node rotateRight(Node h)
{
    assert isRed(h.left);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

Invariants. Maintains symmetric order and perfect black balance.

Elementary red-black BST operations

Color flip. Recolor to split a (temporary) 4-node.

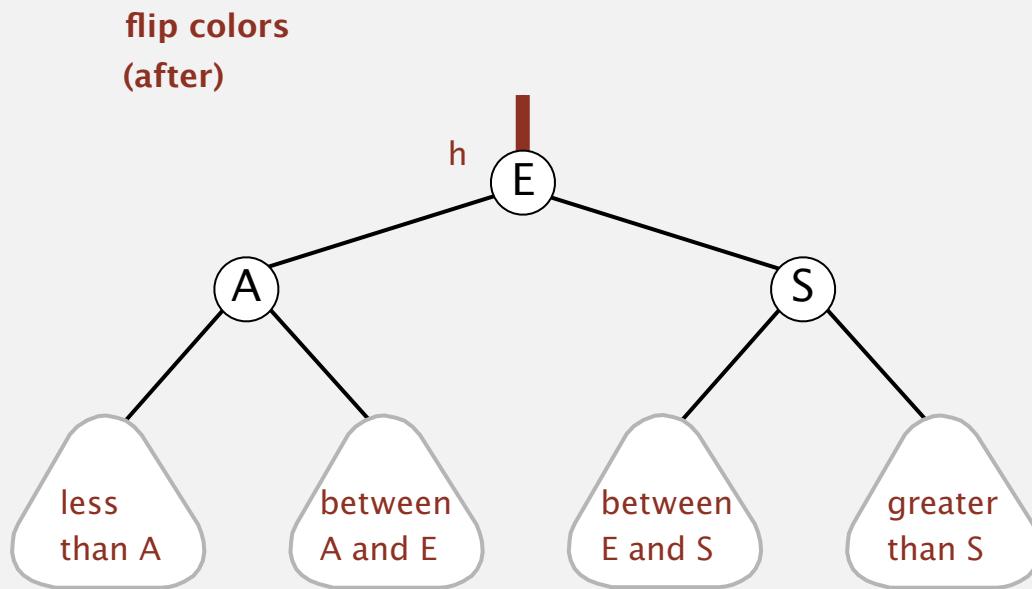


```
private void flipColors(Node h)
{
    assert !isRed(h);
    assert isRed(h.left);
    assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```

Invariants. Maintains symmetric order and perfect black balance.

Elementary red-black BST operations

Color flip. Recolor to split a (temporary) 4-node.

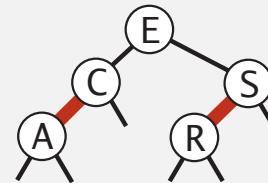
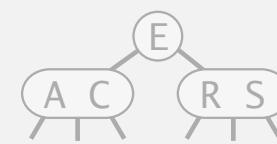
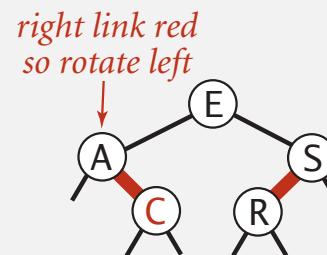
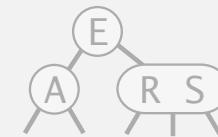
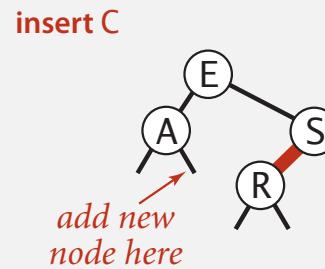


```
private void flipColors(Node h)
{
    assert !isRed(h);
    assert isRed(h.left);
    assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```

Invariants. Maintains symmetric order and perfect black balance.

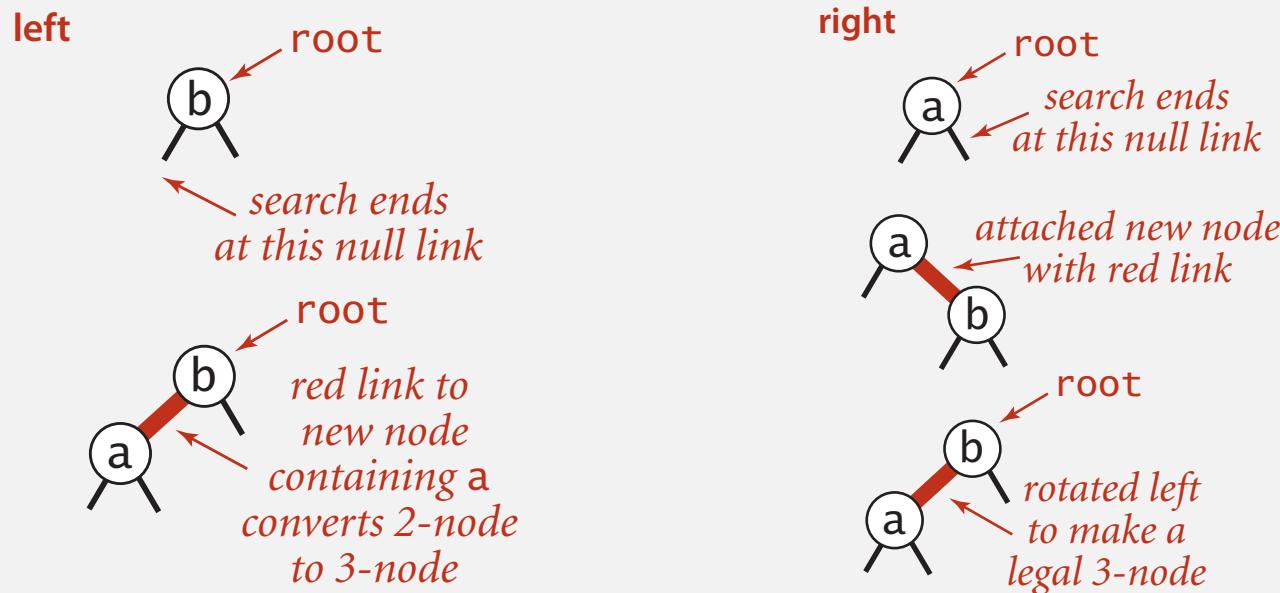
Insertion in a LLRB tree: overview

Basic strategy. Maintain 1-1 correspondence with 2-3 trees by applying elementary red-black BST operations.



Insertion in a LLRB tree

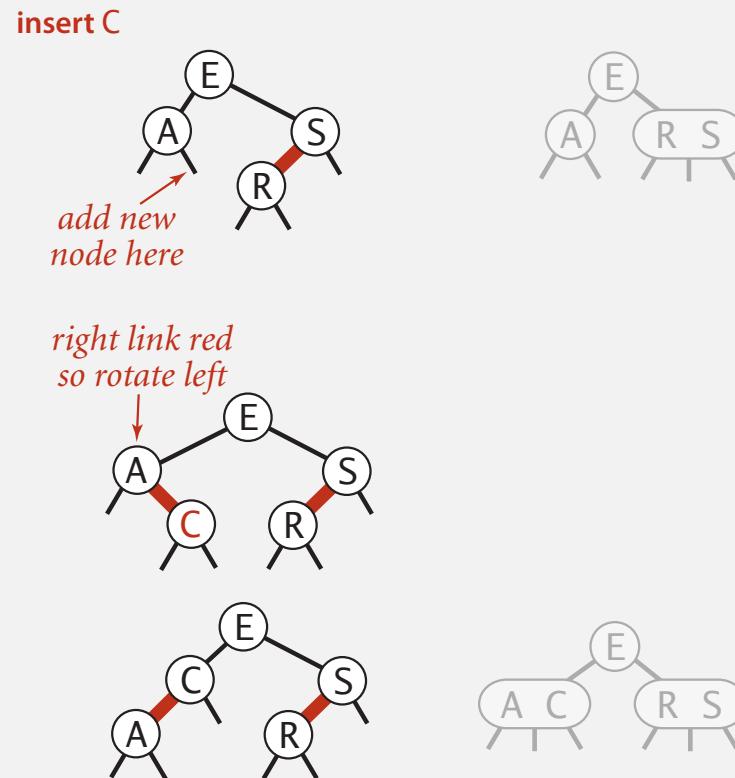
Warmup 1. Insert into a tree with exactly 1 node.



Insertion in a LLRB tree

Case 1. Insert into a 2-node at the bottom.

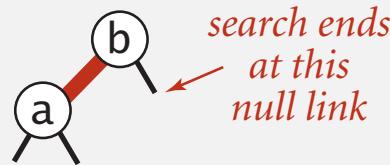
- Do standard BST insert; color new link red.
- If new red link is a right link, rotate left.



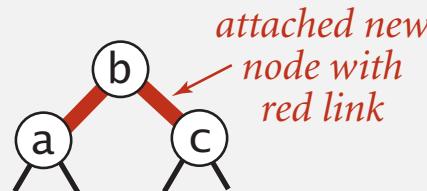
Insertion in a LLRB tree

Warmup 2. Insert into a tree with exactly 2 nodes.

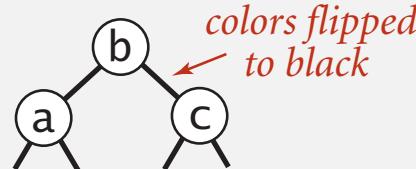
larger



search ends
at this
null link

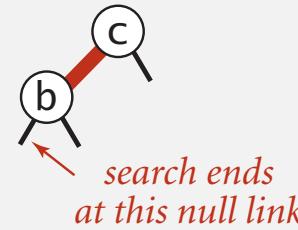


attached new
node with
red link

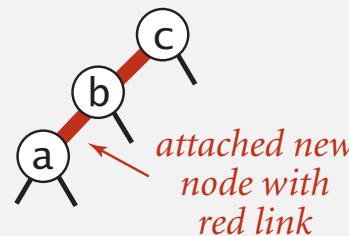


colors flipped
to black

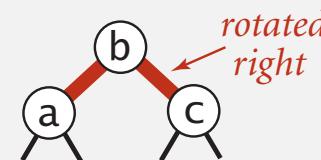
smaller



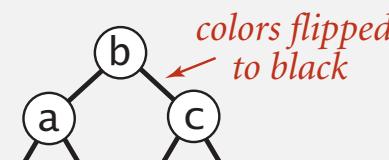
search ends
at this null link



attached new
node with
red link



rotated
right

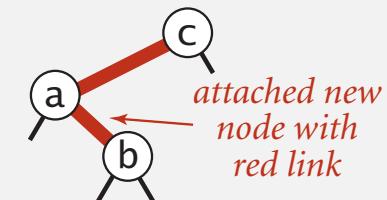


colors flipped
to black

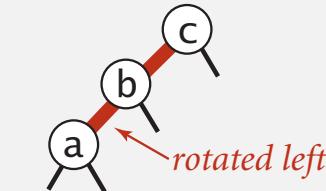
between



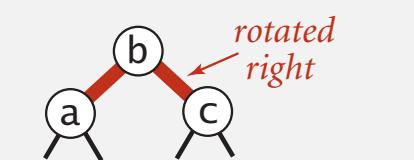
search ends
at this null link



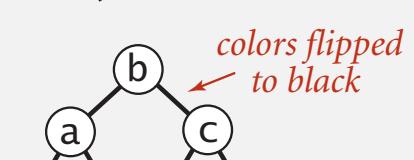
attached new
node with
red link



rotated left



rotated
right

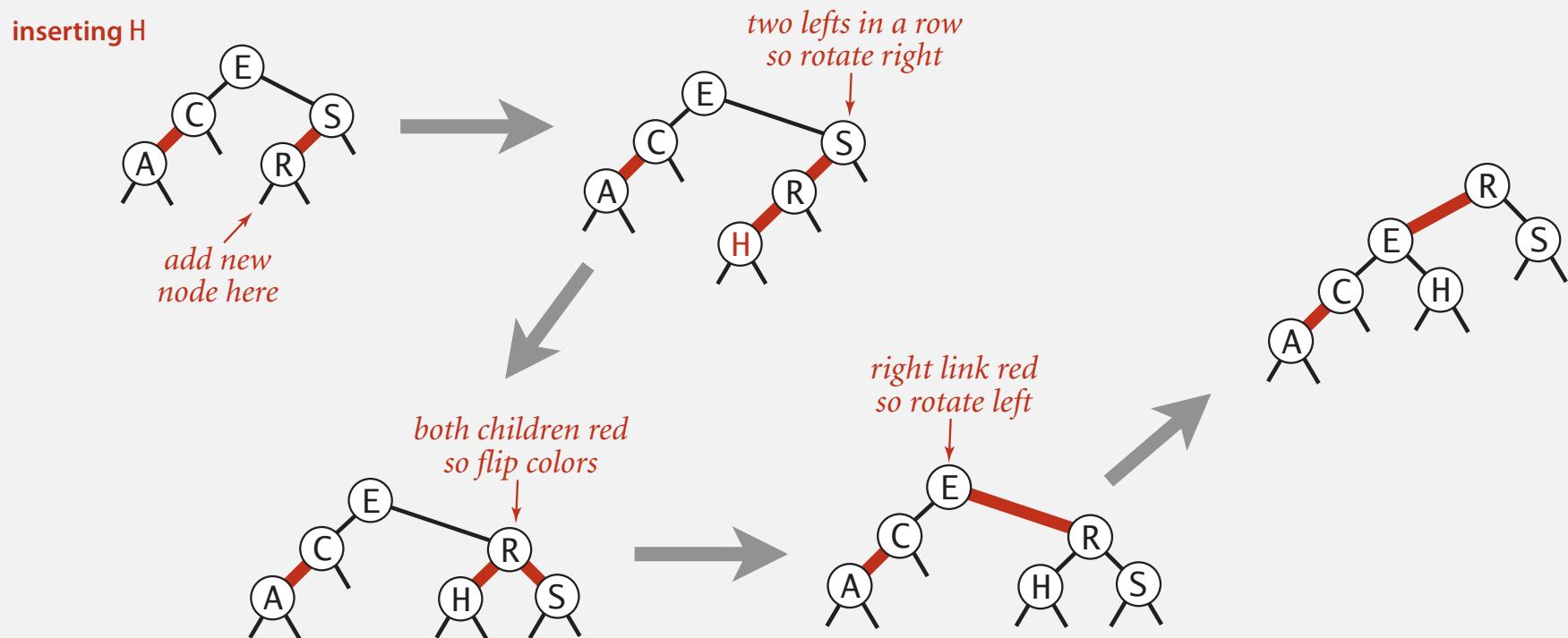


colors flipped
to black

Insertion in a LLRB tree

Case 2. Insert into a 3-node at the bottom.

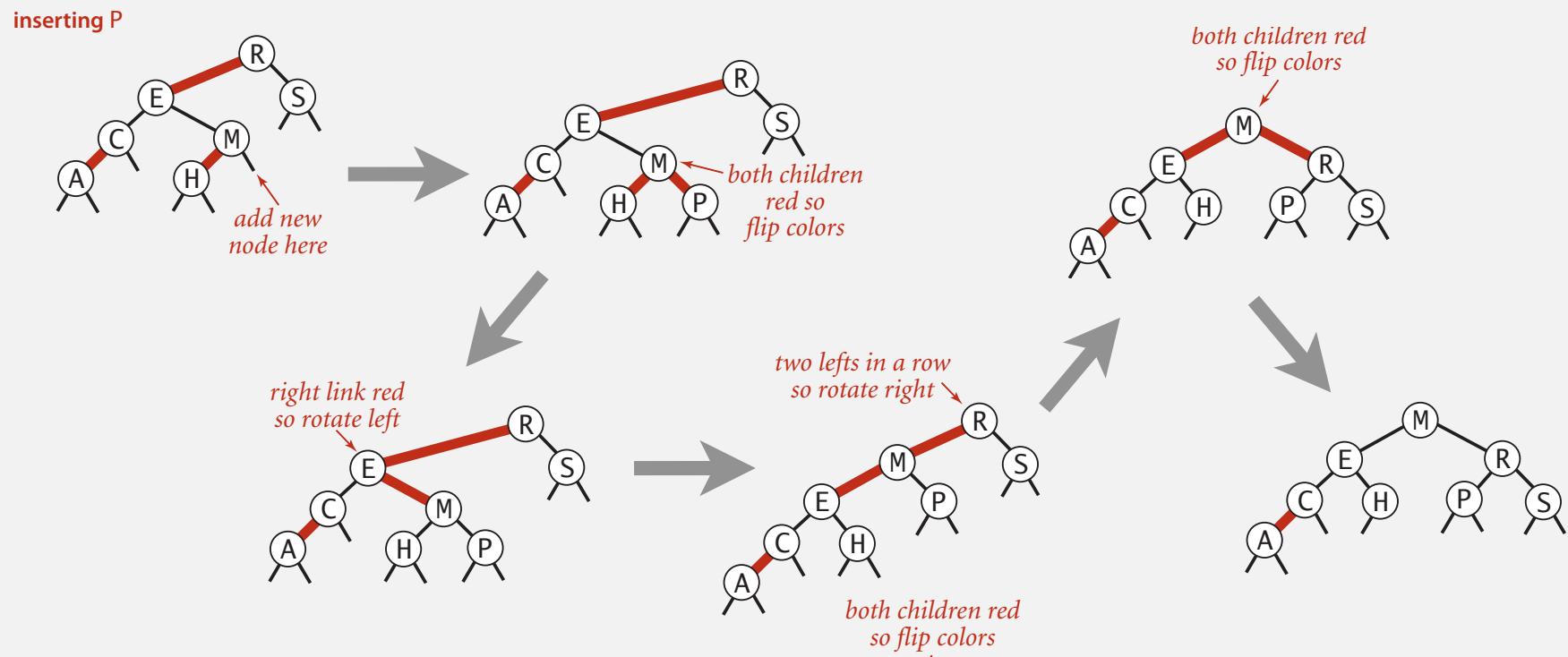
- Do standard BST insert; color new link red.
- Rotate to balance the 4-node (if needed).
- Flip colors to pass red link up one level.
- Rotate to make lean left (if needed).



Insertion in a LLRB tree: passing red links up the tree

Case 2. Insert into a 3-node at the bottom.

- Do standard BST insert; color new link red.
- Rotate to balance the 4-node (if needed).
- Flip colors to pass red link up one level.
- Rotate to make lean left (if needed).
- Repeat case 1 or case 2 up the tree (if needed).



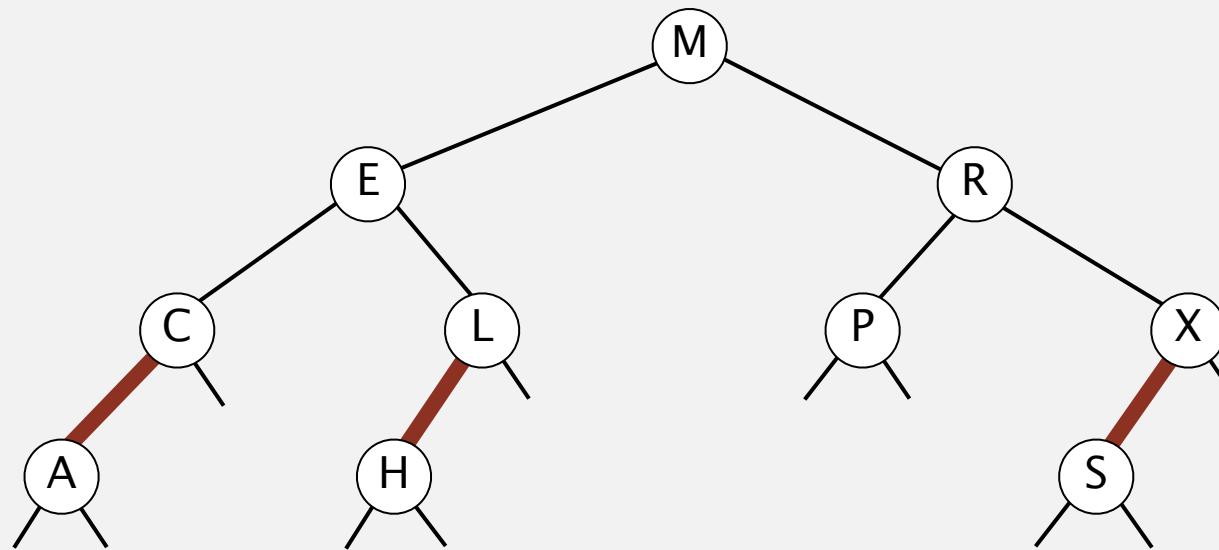
Red-black BST construction demo

insert S



Red-black BST construction demo

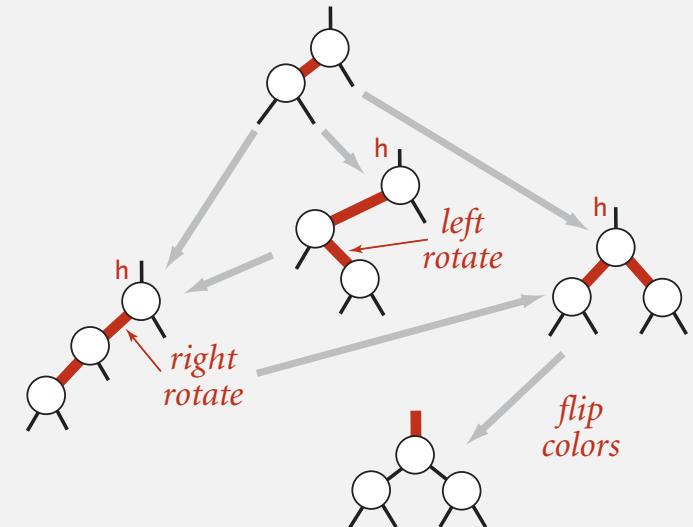
red-black BST



Insertion in a LLRB tree: Java implementation

Same code for all cases.

- Right child red, left child black: **rotate left**.
- Left child, left-left grandchild red: **rotate right**.
- Both children red: **flip colors**.



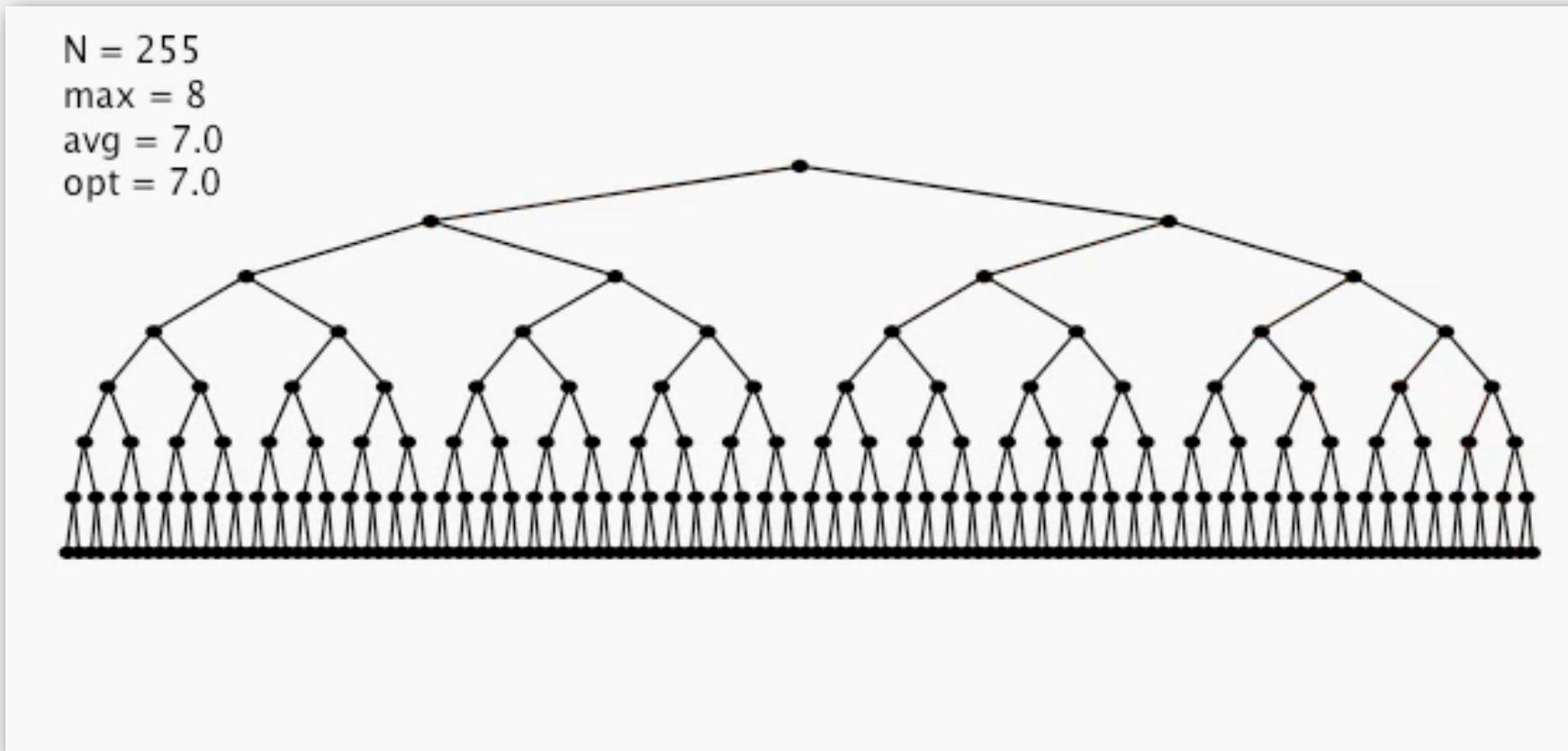
```
private Node put(Node h, Key key, Value val)
{
    if (h == null) return new Node(key, val, RED);           ← insert at bottom
    int cmp = key.compareTo(h.key);                          (and color it red)
    if      (cmp < 0) h.left  = put(h.left,  key, val);
    else if (cmp > 0) h.right = put(h.right, key, val);
    else if (cmp == 0) h.val  = val;

    if (isRed(h.right) && !isRed(h.left))     h = rotateLeft(h);   ← lean left
    if (isRed(h.left)  && isRed(h.left.left)) h = rotateRight(h);  ← balance 4-node
    if (isRed(h.left)  && isRed(h.right))       flipColors(h);    ← split 4-node

    return h;
}
```

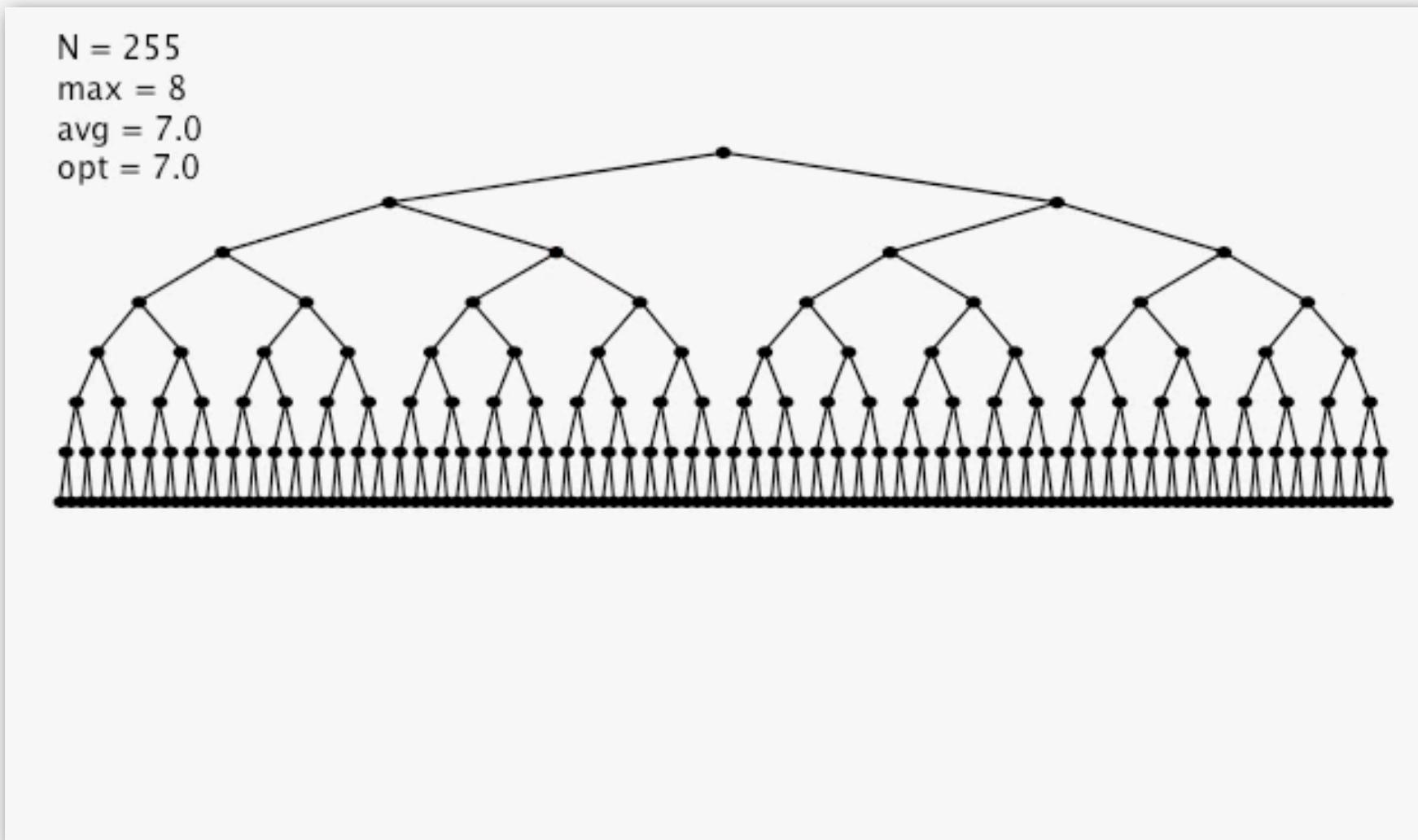
only a few extra lines of code provides near-perfect balance

Insertion in a LLRB tree: visualization



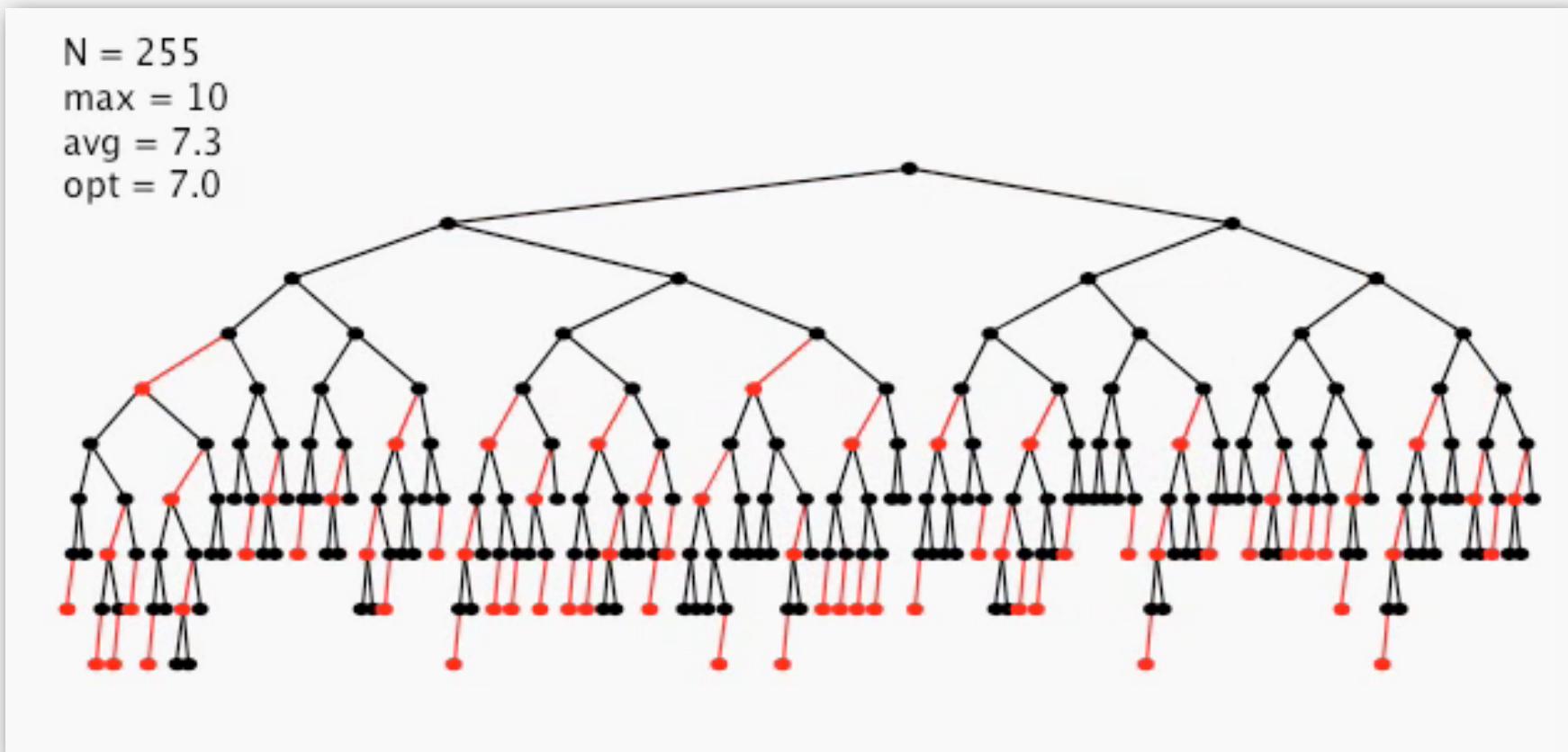
255 insertions in ascending order

Insertion in a LLRB tree: visualization



255 insertions in descending order

Insertion in a LLRB tree: visualization



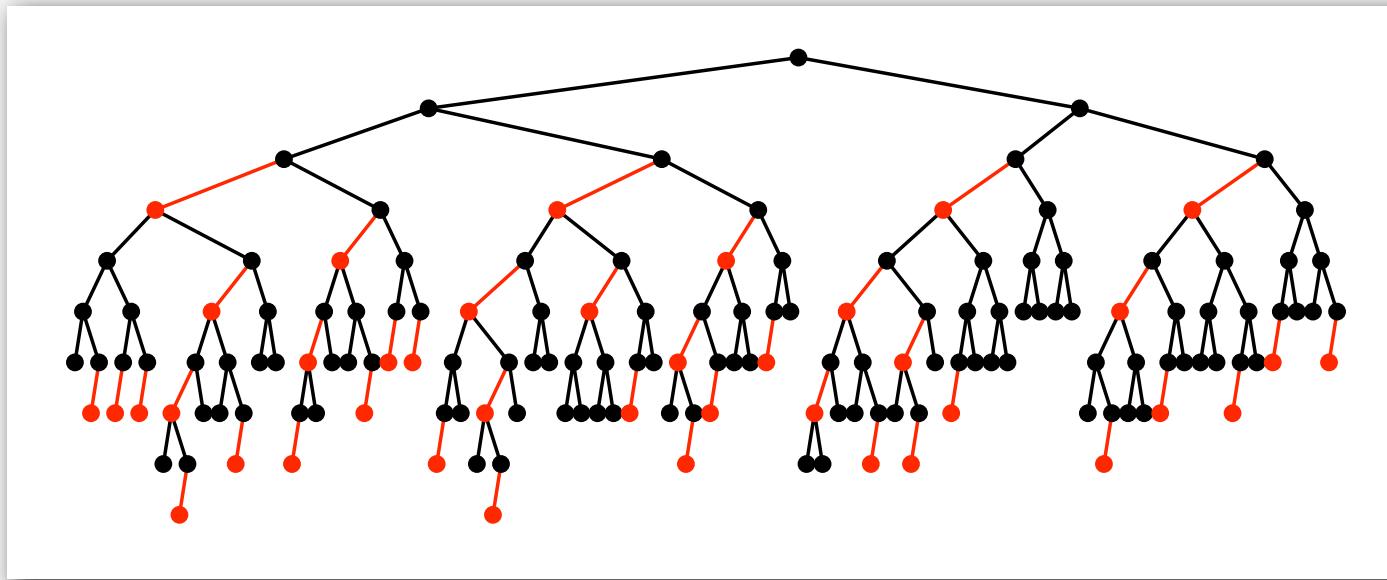
255 random insertions

Balance in LLRB trees

Proposition. Height of tree is $\leq 2 \lg N$ in the worst case.

Pf.

- Every path from root to null link has same number of black links.
- Never two red links in-a-row.



Property. Height of tree is $\sim 1.00 \lg N$ in typical applications.

ST implementations: summary

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compareTo()
BST	N	N	N	$1.39 \lg N$	$1.39 \lg N$?	yes	compareTo()
2-3 tree	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	$c \lg N$	yes	compareTo()
red-black BST	$2 \lg N$	$2 \lg N$	$2 \lg N$	$1.00 \lg N^*$	$1.00 \lg N^*$	$1.00 \lg N^*$	yes	compareTo()

* exact value of coefficient unknown but extremely close to 1

War story: why red-black?

Xerox PARC innovations. [1970s]

- Alto.
- GUI.
- Ethernet.
- Smalltalk.
- InterPress.
- Laser printing.
- Bitmapped display.
- WYSIWYG text editor.
- ...

XEROX®

PARC



Xerox Alto

A DICHROMATIC FRAMEWORK FOR BALANCED TREES

Leo J. Guibas
*Xerox Palo Alto Research Center,
Palo Alto, California, and
Carnegie-Mellon University*

and

Robert Sedgewick*
*Program in Computer Science
Brown University
Providence, R. I.*

ABSTRACT

In this paper we present a uniform framework for the implementation and study of balanced tree algorithms. We show how to imbed in this

the way down towards a leaf. As we will see, this has a number of significant advantages over the older methods. We shall examine a number of variations on a common theme and exhibit full implementations which are notable for their brevity. One implementation is examined carefully, and some properties about its

War story: red-black BSTs

Telephone company contracted with database provider to build real-time database to store customer information.

Database implementation.

- Red-black BST search and insert; Hibbard deletion.
- Exceeding height limit of 80 triggered error-recovery process.

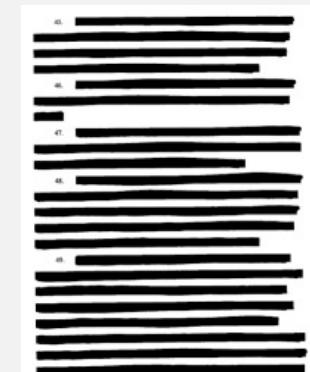
allows for up to 2^{40} keys

Extended telephone service outage.

- Main cause = height bounded exceeded!
- Telephone company sues database provider.
- Legal testimony:

“If implemented properly, the height of a red-black BST with N keys is at most $2 \lg N$. ” — expert witness

Hibbard deletion
was the problem



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ *2-3 search trees*
- ▶ *red-black BSTs*
- ▶ *B-trees*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ *2-3 search trees*
- ▶ *red-black BSTs*
- ▶ *B-trees*

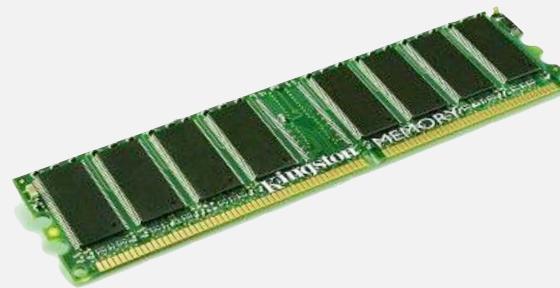
File system model

Page. Contiguous block of data (e.g., a file or 4,096-byte chunk).

Probe. First access to a page (e.g., from disk to memory).



slow



fast

Property. Time required for a probe is much larger than time to access data within a page.

Cost model. Number of probes.

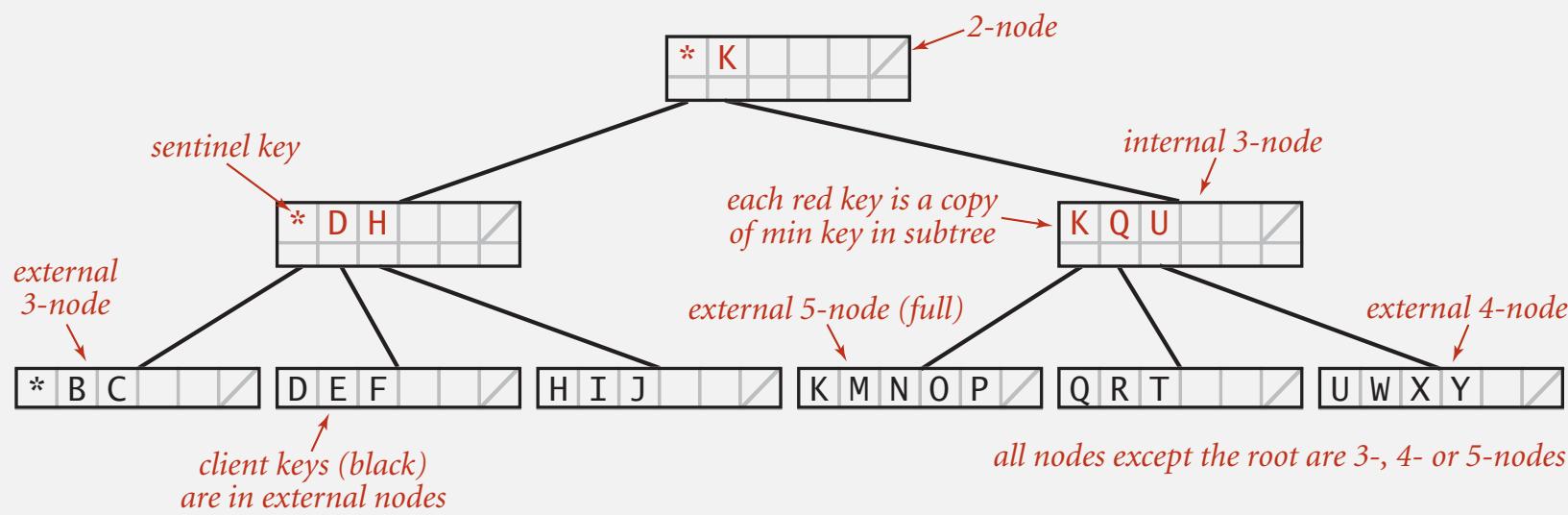
Goal. Access data using minimum number of probes.

B-trees (Bayer-McCreight, 1972)

B-tree. Generalize 2-3 trees by allowing up to $M - 1$ key-link pairs per node.

- At least 2 key-link pairs at root.
- At least $M/2$ key-link pairs in other nodes.
- External nodes contain client keys.
- Internal nodes contain copies of keys to guide search.

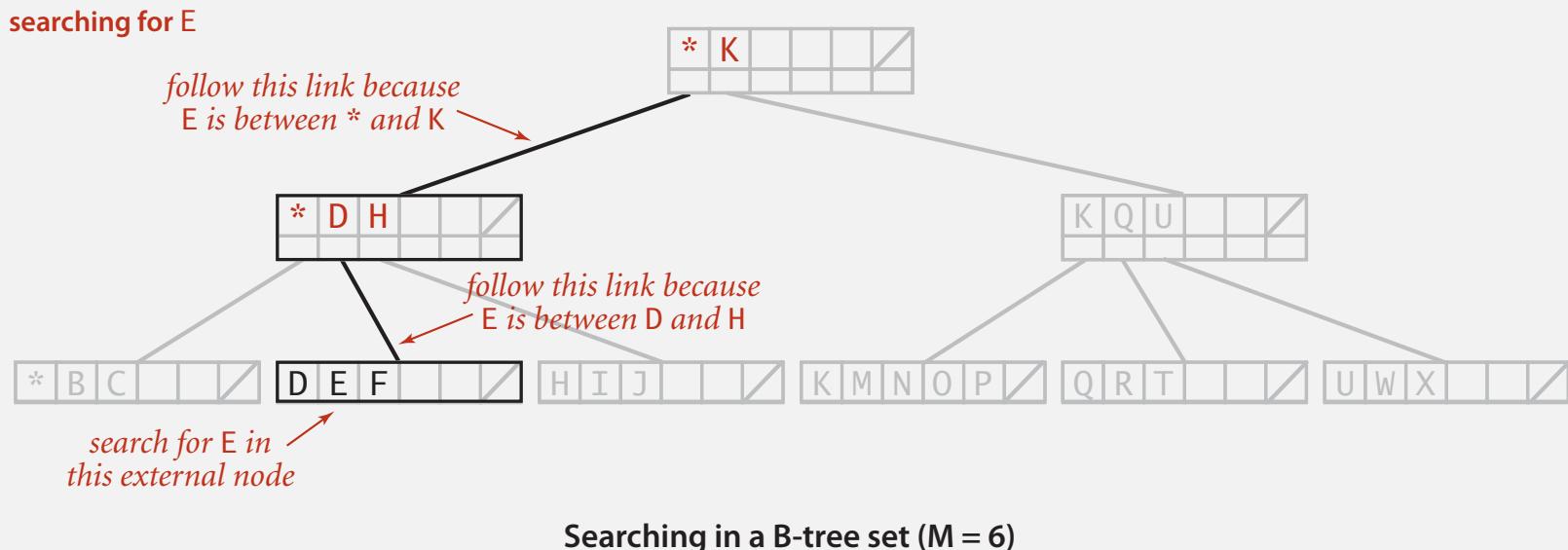
choose M as large as possible so that M links fit in a page, e.g., $M = 1024$



Anatomy of a B-tree set ($M = 6$)

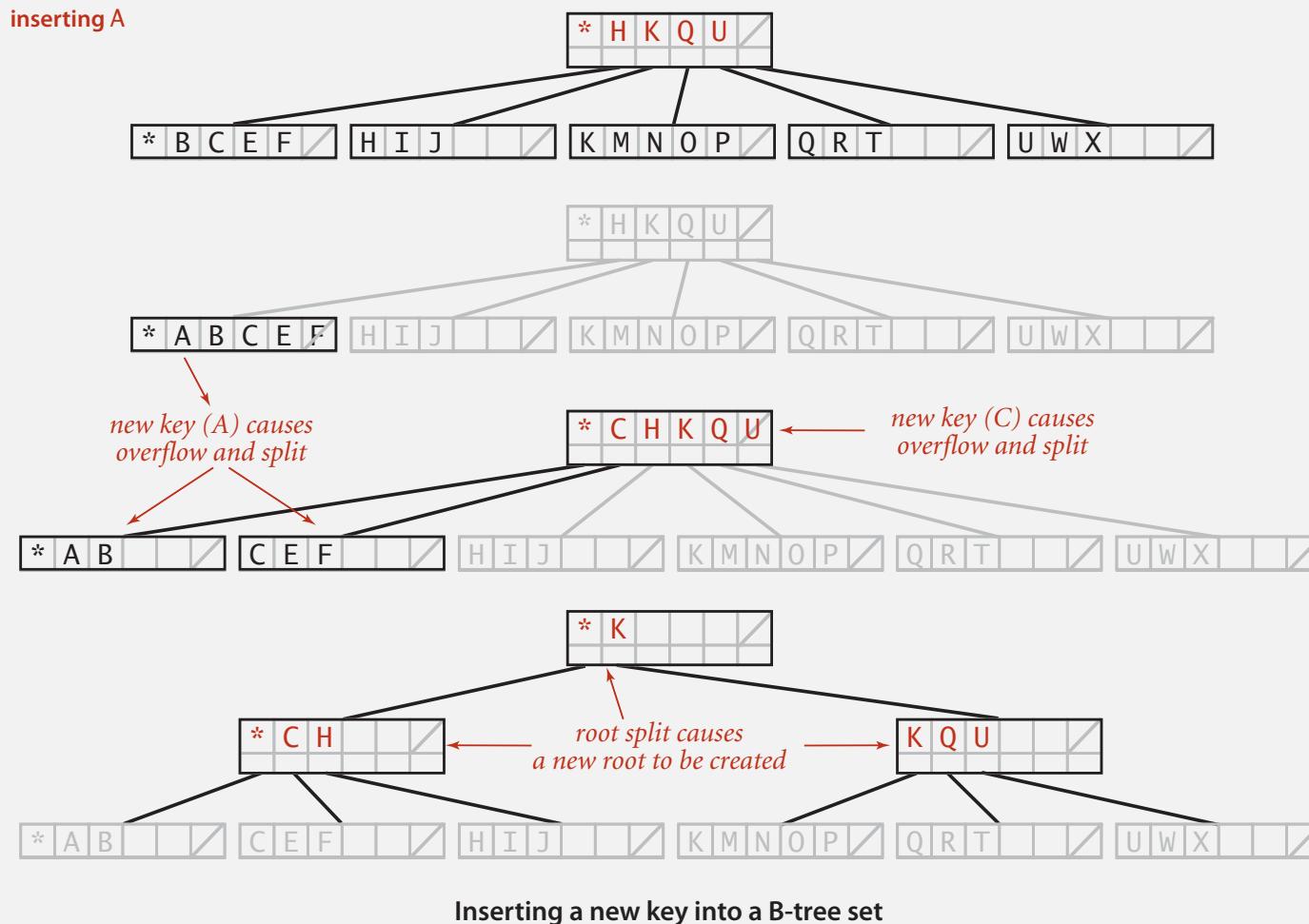
Searching in a B-tree

- Start at root.
- Find interval for search key and take corresponding link.
- Search terminates in external node.



Insertion in a B-tree

- Search for new key.
- Insert at bottom.
- Split nodes with M key-link pairs on the way up the tree.



Balance in B-tree

Proposition. A search or an insertion in a B-tree of order M with N keys requires between $\log_{M-1} N$ and $\log_{M/2} N$ probes.

Pf. All internal nodes (besides root) have between $M/2$ and $M - 1$ links.

In practice. Number of probes is at most 4. $\leftarrow M = 1024; N = 62 \text{ billion}$
 $\log_{M/2} N \leq 4$

Optimization. Always keep root page in memory.

Building a large B tree



Balanced trees in the wild

Red-black trees are widely used as system symbol tables.

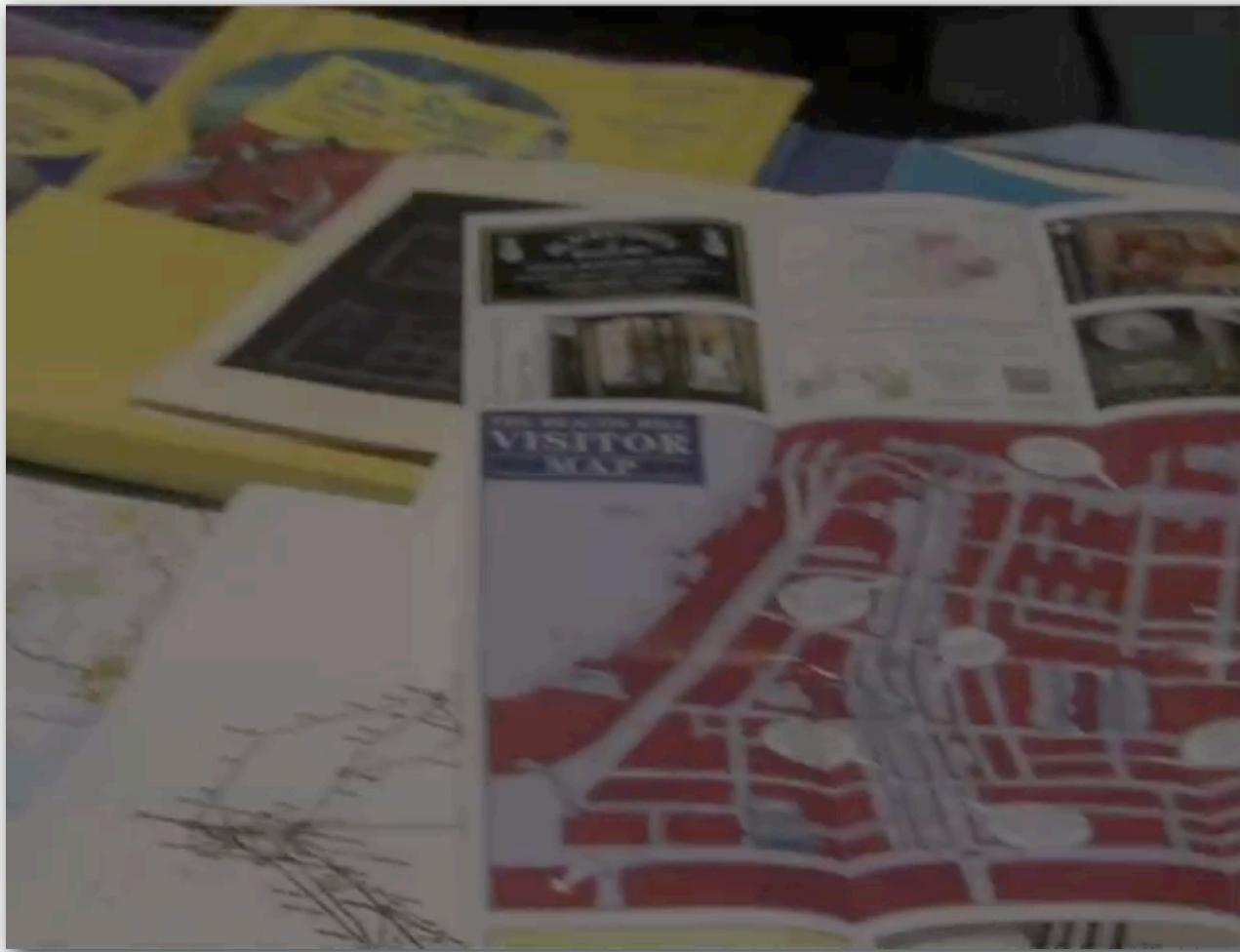
- Java: `java.util.TreeMap`, `java.util.TreeSet`.
- C++ STL: `map`, `multimap`, `multiset`.
- Linux kernel: completely fair scheduler, `linux/rbtree.h`.

B-tree variants. B+ tree, B*tree, B# tree, ...

B-trees (and variants) are widely used for file systems and databases.

- Windows: HPFS.
- Mac: HFS, HFS+.
- Linux: ReiserFS, XFS, Ext3FS, JFS.
- Databases: ORACLE, DB2, INGRES, SQL, PostgreSQL.

Red-black BSTs in the wild



*Common sense. Sixth sense.
Together they're the
FBI's newest team.*

Red-black BSTs in the wild

ACT FOUR

FADE IN:

48 INT. FBI HQ - NIGHT

48

Antonio is at THE COMPUTER as Jess explains herself to Nicole and Pollock. The CONFERENCE TABLE is covered with OPEN REFERENCE BOOKS, TOURIST GUIDES, MAPS and REAMS OF PRINTOUTS.

JESS

It was the red door again.

POLLOCK

I thought the red door was the storage container.

JESS

But it wasn't red anymore. It was black.

ANTONIO

So red turning to black means... what?

POLLOCK

Budget deficits? Red ink, black ink?

NICOLE

Yes. I'm sure that's what it is. But maybe we should come up with a couple other options, just in case.

Antonio refers to his COMPUTER SCREEN, which is filled with mathematical equations.

ANTONIO

It could be an algorithm from a binary search tree. A red-black tree tracks every simple path from a node to a descendant leaf with the same number of black nodes.

JESS

Does that help you with girls?

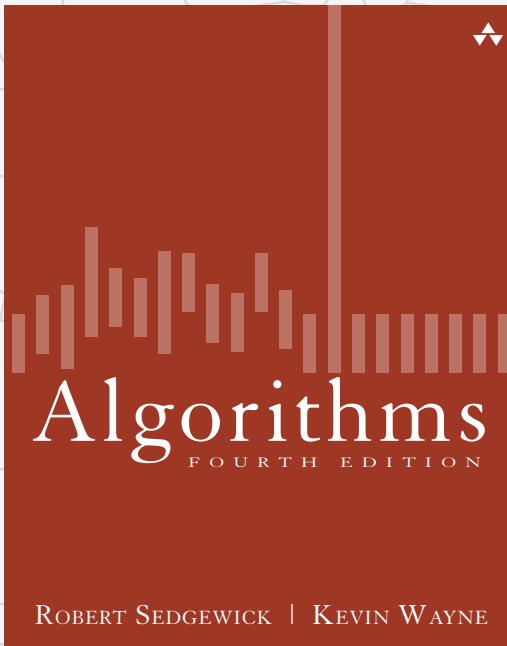
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ *2-3 search trees*
- ▶ *red-black BSTs*
- ▶ *B-trees*



<http://algs4.cs.princeton.edu>

3.3 BALANCED SEARCH TREES

- ▶ 2-3 *search trees*
- ▶ *red-black BSTs*
- ▶ *B-trees*



<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

ST implementations: summary

implementation	worst-case cost (after N inserts)			average-case cost (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compareTo()
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes	compareTo()
red-black BST	$2 \lg N$	$2 \lg N$	$2 \lg N$	$1.00 \lg N$	$1.00 \lg N$	$1.00 \lg N$	yes	compareTo()

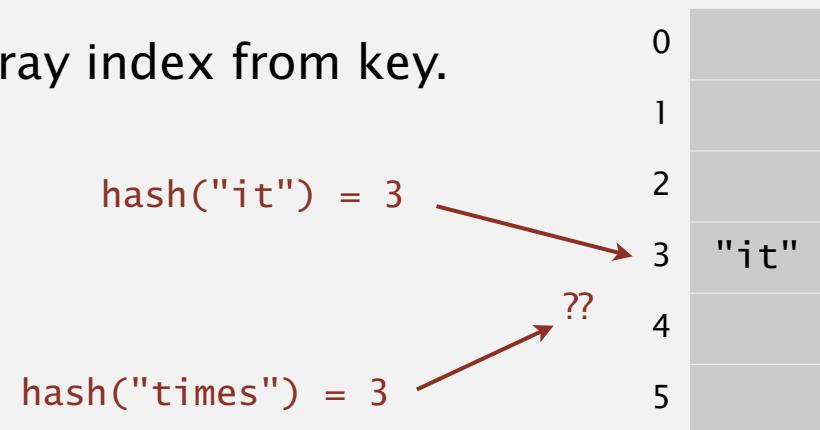
Q. Can we do better?

A. Yes, but with different access to the data.

Hashing: basic plan

Save items in a **key-indexed table** (index is a function of the key).

Hash function. Method for computing array index from key.



Issues.

- Computing the hash function.
- Equality test: Method for checking whether two keys are equal.
- Collision resolution: Algorithm and data structure to handle two keys that hash to the same array index.

Classic space-time tradeoff.

- No space limitation: trivial hash function with key as index.
- No time limitation: trivial collision resolution with sequential search.
- Space and time limitations: hashing (the real world).

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

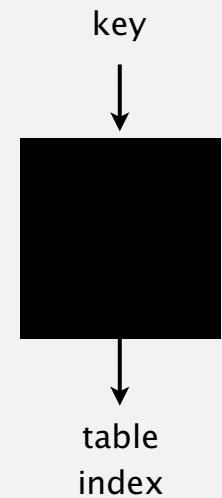
- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Computing the hash function

Idealistic goal. Scramble the keys uniformly to produce a table index.

- Efficiently computable.
- Each table index equally likely for each key.

thoroughly researched problem,
still problematic in practical applications



Ex 1. Phone numbers.

- Bad: first three digits.
- Better: last three digits.

Ex 2. Social Security numbers.

- Bad: first three digits. ← 573 = California, 574 = Alaska
(assigned in chronological order within geographic region)
- Better: last three digits.

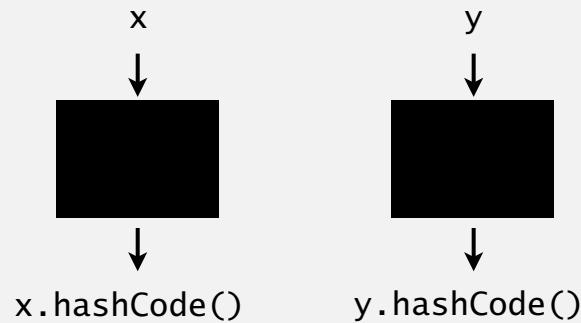
Practical challenge. Need different approach for each key type.

Java's hash code conventions

All Java classes inherit a method `hashCode()`, which returns a 32-bit `int`.

Requirement. If `x.equals(y)`, then `(x.hashCode() == y.hashCode())`.

Highly desirable. If `!x.equals(y)`, then `(x.hashCode() != y.hashCode())`.



Default implementation. Memory address of `x`.

Legal (but poor) implementation. Always return 17.

Customized implementations. `Integer`, `Double`, `String`, `File`, `URL`, `Date`, ...

User-defined types. Users are on their own.

Implementing hash code: integers, booleans, and doubles

Java library implementations

```
public final class Integer
{
    private final int value;
    ...
    public int hashCode()
    { return value; }
}
```

```
public final class Boolean
{
    private final boolean value;
    ...
    public int hashCode()
    {
        if (value) return 1231;
        else      return 1237;
    }
}
```

```
public final class Double
{
    private final double value;
    ...
    public int hashCode()
    {
        long bits = doubleToLongBits(value);
        return (int) (bits ^ (bits >>> 32));
    }
}
```

convert to IEEE 64-bit representation;
xor most significant 32-bits
with least significant 32-bits

Implementing hash code: strings

Java library implementation

```
public final class String
{
    private final char[] s;
    ...
    public int hashCode()
    {
        int hash = 0;
        for (int i = 0; i < length(); i++)
            hash = s[i] + (31 * hash);
        return hash;
    }
}
```

char	Unicode
...	...
'a'	97
'b'	98
'c'	99
...	...

- Horner's method to hash string of length L : L multiplies/adds.
- Equivalent to $h = s[0] \cdot 31^{L-1} + \dots + s[L-3] \cdot 31^2 + s[L-2] \cdot 31^1 + s[L-1] \cdot 31^0$.

Ex.

```
String s = "call";
int code = s.hashCode();
```

$3045982 = 99 \cdot 31^3 + 97 \cdot 31^2 + 108 \cdot 31^1 + 108 \cdot 31^0$
 $= 108 + 31 \cdot (108 + 31 \cdot (97 + 31 \cdot (99)))$
(Horner's method)

Implementing hash code: strings

Performance optimization.

- Cache the hash value in an instance variable.
- Return cached value.

```
public final class String
{
    private int hash = 0;           ← cache of hash code
    private final char[] s;
    ...

    public int hashCode()
    {
        int h = hash;             ← return cached value
        if (h != 0) return h;
        for (int i = 0; i < length(); i++)
            h = s[i] + (31 * hash);
        hash = h;                 ← store cache of hash code
        return h;
    }
}
```

Implementing hash code: user-defined types

```
public final class Transaction implements Comparable<Transaction>
{
    private final String who;
    private final Date when;
    private final double amount;

    public Transaction(String who, Date when, double amount)
    { /* as before */ }

    ...

    public boolean equals(Object y)
    { /* as before */ }

    public int hashCode()
    {
        int hash = 17;           ← nonzero constant
        hash = 31*hash + who.hashCode(); ← for reference types,
                                         use hashCode()
        hash = 31*hash + when.hashCode(); ← for primitive types,
                                         use hashCode()
        hash = 31*hash + ((Double) amount).hashCode(); ← of wrapper type
        return hash;
    }
}
```

typically a small prime

Hash code design

"Standard" recipe for user-defined types.

- Combine each significant field using the $31x + y$ rule.
- If field is a primitive type, use wrapper type `hashCode()`.
- If field is null, return 0.
- If field is a reference type, use `hashCode()`. ← applies rule recursively
- If field is an array, apply to each entry. ← or use `Arrays.deepHashCode()`

In practice. Recipe works reasonably well; used in Java libraries.

In theory. Keys are bitstring; "universal" hash functions exist.

Basic rule. Need to use the whole key to compute hash code;
consult an expert for state-of-the-art hash codes.

Modular hashing

Hash code. An int between -2^{31} and $2^{31} - 1$.

Hash function. An int between 0 and $M - 1$ (for use as array index).

typically a prime or power of 2

```
private int hash(Key key)
{   return key.hashCode() % M; }
```

bug

```
private int hash(Key key)
{   return Math.abs(key.hashCode()) % M; }
```

1-in-a-billion bug

hashCode() of "polygenelubricants" is -2^{31}

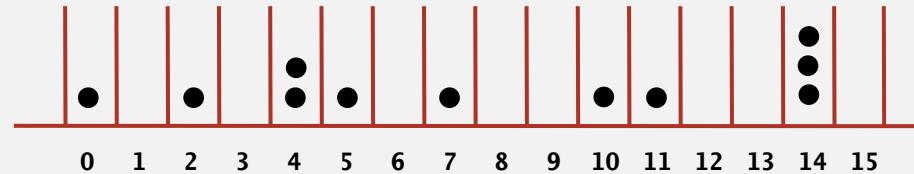
```
private int hash(Key key)
{   return (key.hashCode() & 0xffffffff) % M; }
```

correct

Uniform hashing assumption

Uniform hashing assumption. Each key is equally likely to hash to an integer between 0 and $M - 1$.

Bins and balls. Throw balls uniformly at random into M bins.



Birthday problem. Expect two balls in the same bin after $\sim \sqrt{\pi M / 2}$ tosses.

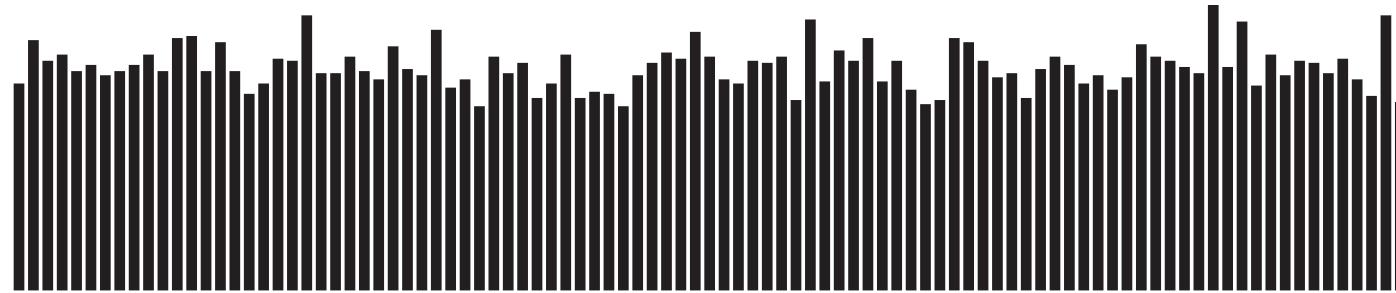
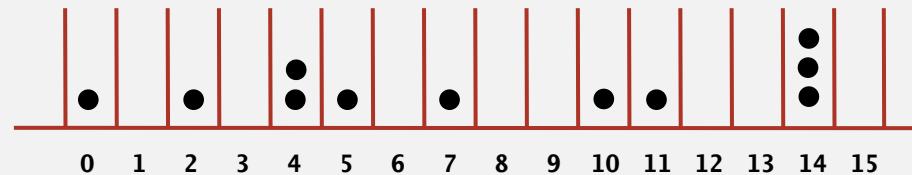
Coupon collector. Expect every bin has ≥ 1 ball after $\sim M \ln M$ tosses.

Load balancing. After M tosses, expect most loaded bin has $\Theta(\log M / \log \log M)$ balls.

Uniform hashing assumption

Uniform hashing assumption. Each key is equally likely to hash to an integer between 0 and $M - 1$.

Bins and balls. Throw balls uniformly at random into M bins.



Hash value frequencies for words in Tale of Two Cities ($M = 97$)

Java's String data uniformly distribute the keys of Tale of Two Cities

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

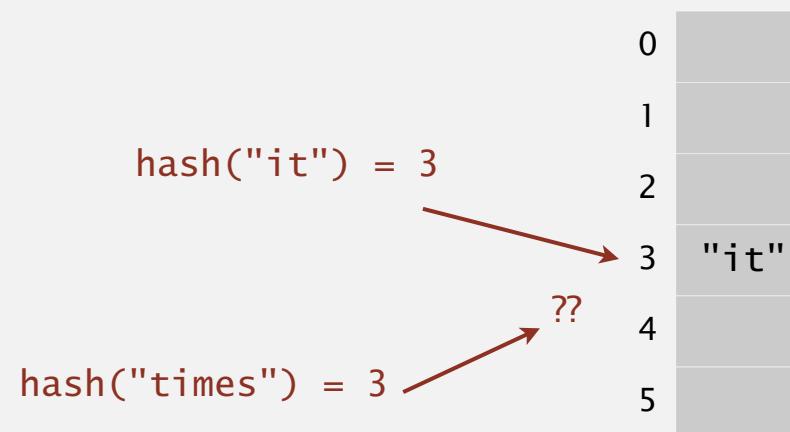
- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Collisions

Collision. Two distinct keys hashing to same index.

- Birthday problem \Rightarrow can't avoid collisions unless you have a ridiculous (quadratic) amount of memory.
- Coupon collector + load balancing \Rightarrow collisions will be evenly distributed.

Challenge. Deal with collisions efficiently.



Separate chaining symbol table

Use an array of $M < N$ linked lists. [H. P. Luhn, IBM 1953]

- Hash: map key to integer i between 0 and $M - 1$.
- Insert: put at front of i^{th} chain (if not already there).
- Search: need to search only i^{th} chain.

key hash value

S 2 0

E 0 1

A 0 2

R 4 3

C 4 4

H 4 5

E 0 6

X 2 7

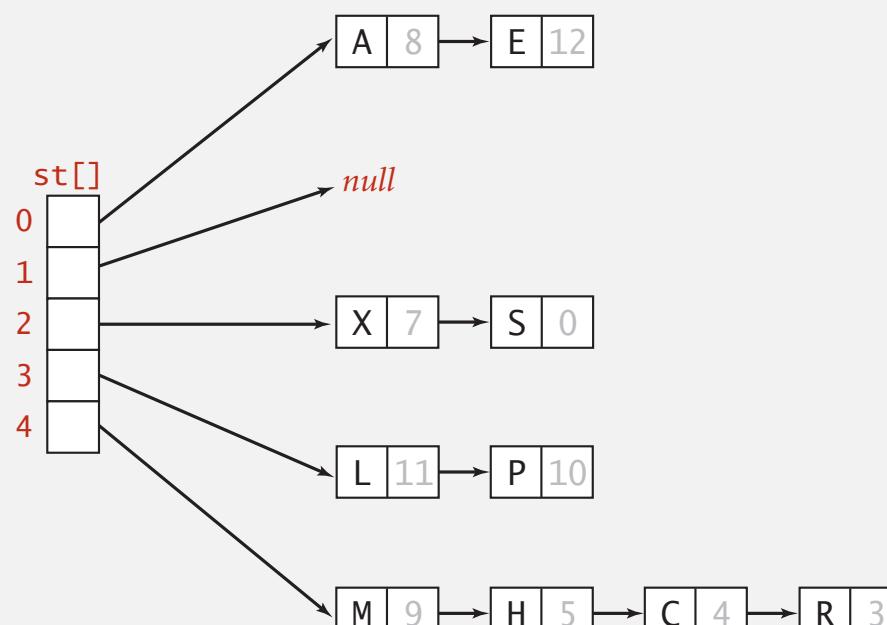
A 0 8

M 4 9

P 3 10

L 3 11

E 0 12



Separate chaining ST: Java implementation

```
public class SeparateChainingHashST<Key, Value>
{
    private int M = 97;                      // number of chains
    private Node[] st = new Node[M]; // array of chains

    private static class Node
    {
        private Object key; ← no generic array creation
        private Object val; ← (declare key and value of type Object)
        private Node next;
        ...
    }

    private int hash(Key key)
    {   return (key.hashCode() & 0xffffffff) % M;   }

    public Value get(Key key) {
        int i = hash(key);
        for (Node x = st[i]; x != null; x = x.next)
            if (key.equals(x.key)) return (Value) x.val;
        return null;
    }
}
```

array doubling and
halving code omitted

Separate chaining ST: Java implementation

```
public class SeparateChainingHashST<Key, Value>
{
    private int M = 97;                      // number of chains
    private Node[] st = new Node[M]; // array of chains

    private static class Node
    {
        private Object key;
        private Object val;
        private Node next;
        ...
    }

    private int hash(Key key)
    {   return (key.hashCode() & 0x7fffffff) % M;   }

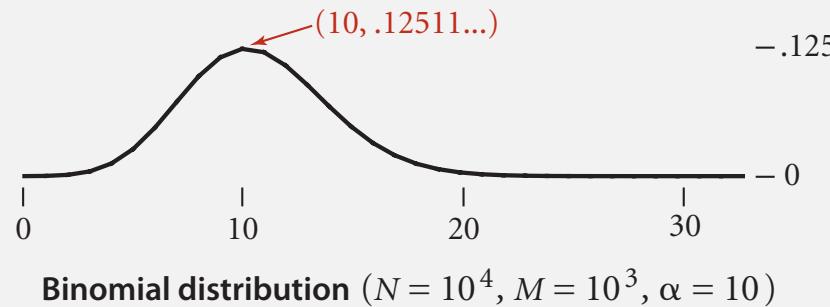
    public void put(Key key, Value val) {
        int i = hash(key);
        for (Node x = st[i]; x != null; x = x.next)
            if (key.equals(x.key)) { x.val = val; return; }
        st[i] = new Node(key, val, st[i]);
    }

}
```

Analysis of separate chaining

Proposition. Under uniform hashing assumption, prob. that the number of keys in a list is within a constant factor of N/M is extremely close to 1.

Pf sketch. Distribution of list size obeys a binomial distribution.



Consequence. Number of probes for search/insert is proportional to N/M .

- M too large \Rightarrow too many empty chains.
- M too small \Rightarrow chains too long.
- Typical choice: $M \sim N/5 \Rightarrow$ constant-time ops.

equals() and hashCode()

\uparrow
M times faster than
sequential search

ST implementations: summary

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compareTo()
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes	compareTo()
red-black tree	$2 \lg N$	$2 \lg N$	$2 \lg N$	$1.00 \lg N$	$1.00 \lg N$	$1.00 \lg N$	yes	compareTo()
separate chaining	$\lg N^*$	$\lg N^*$	$\lg N^*$	3-5 *	3-5 *	3-5 *	no	equals() hashCode()

* under uniform hashing assumption

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

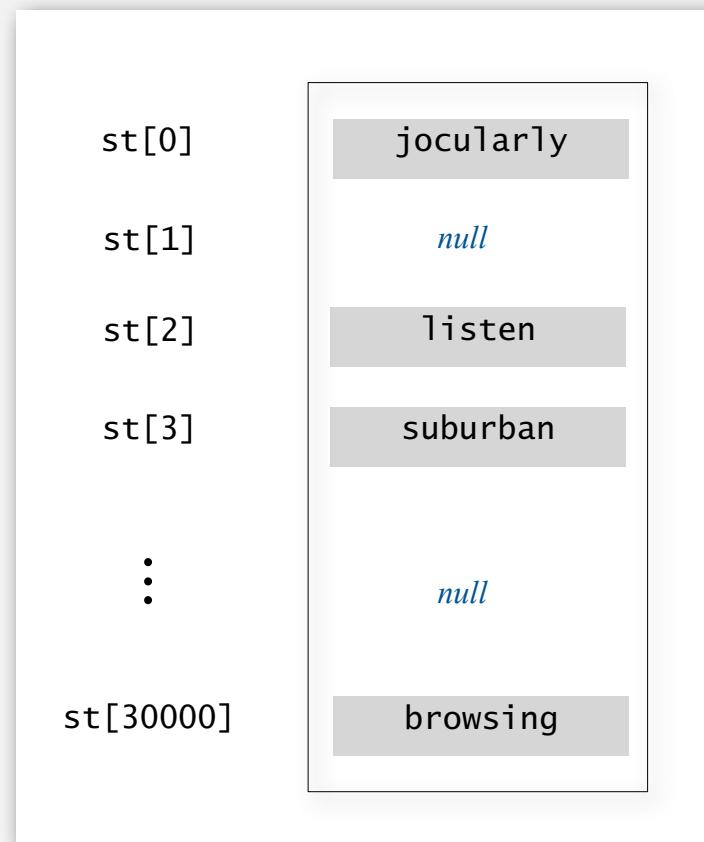
3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Collision resolution: open addressing

Open addressing. [Amdahl-Boehme-Rochester-Samuel, IBM 1953]

When a new key collides, find next empty slot, and put it there.



linear probing ($M = 30001$, $N = 15000$)

Linear probing hash table demo

Hash. Map key to integer i between 0 and $M-1$.

Insert. Put at table index i if free; if not try $i+1, i+2$, etc.

linear probing hash table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]																

$M = 16$



Linear probing hash table demo

Hash. Map key to integer i between 0 and $M-1$.

Search. Search table index i ; if occupied but no match, try $i+1$, $i+2$, etc.

search K

hash(K) = 5

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L	E					R	X
M = 16											K					

search miss
(return null)

Linear probing hash table summary

Hash. Map key to integer i between 0 and $M-1$.

Insert. Put at table index i if free; if not try $i+1, i+2$, etc.

Search. Search table index i ; if occupied but no match, try $i+1, i+2$, etc.

Note. Array size M **must be** greater than number of key-value pairs N .

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st[]	P	M			A	C	S	H	L		E				R	X

$M = 16$

Linear probing ST implementation

```
public class LinearProbingHashST<Key, Value>
{
    private int M = 30001;
    private Value[] vals = (Value[]) new Object[M];
    private Key[] keys = (Key[]) new Object[M];
```

array doubling and
halving code omitted ←

```
    private int hash(Key key) { /* as before */ }
```

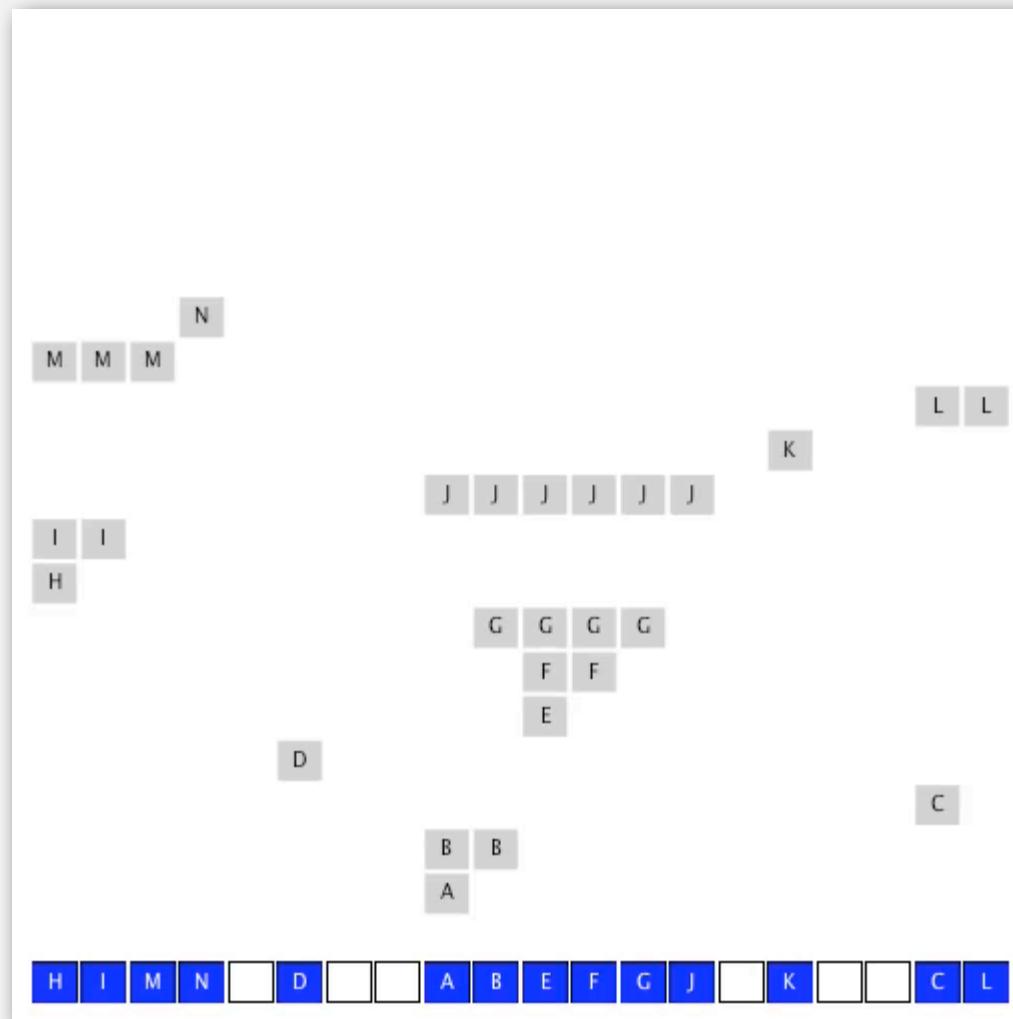
```
    public void put(Key key, Value val)
    {
        int i;
        for (i = hash(key); keys[i] != null; i = (i+1) % M)
            if (keys[i].equals(key))
                break;
        keys[i] = key;
        vals[i] = val;
    }
```

```
    public Value get(Key key)
    {
        for (int i = hash(key); keys[i] != null; i = (i+1) % M)
            if (key.equals(keys[i]))
                return vals[i];
        return null;
    }
}
```

Clustering

Cluster. A contiguous block of items.

Observation. New keys likely to hash into middle of big clusters.

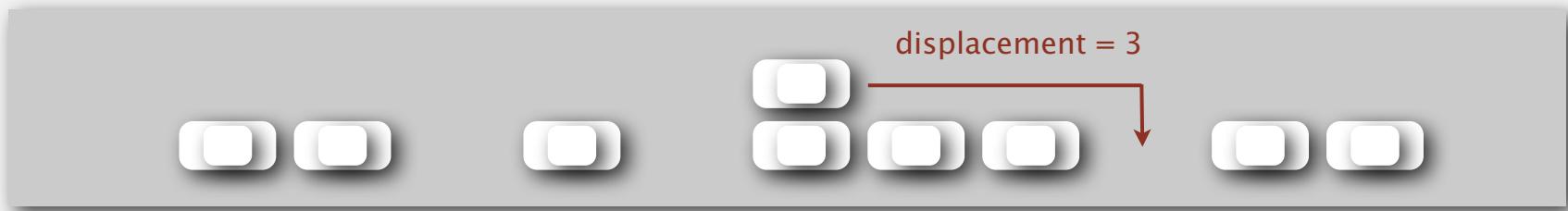


Knuth's parking problem

Model. Cars arrive at one-way street with M parking spaces.

Each desires a random space i : if space i is taken, try $i + 1, i + 2$, etc.

Q. What is mean displacement of a car?



Half-full. With $M/2$ cars, mean displacement is $\sim 3/2$.

Full. With M cars, mean displacement is $\sim \sqrt{\pi M / 8}$.

Analysis of linear probing

Proposition. Under uniform hashing assumption, the average # of probes in a linear probing hash table of size M that contains $N = \alpha M$ keys is:

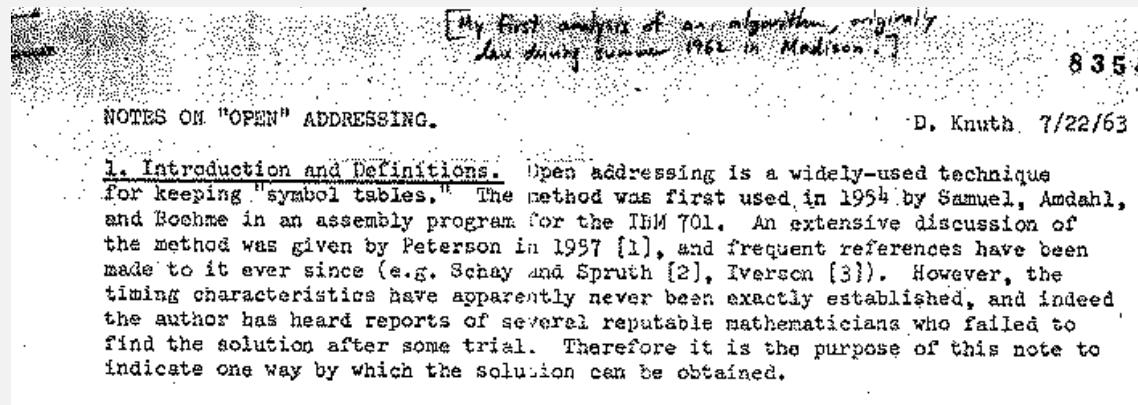
$$\sim \frac{1}{2} \left(1 + \frac{1}{1 - \alpha} \right)$$

search hit

$$\sim \frac{1}{2} \left(1 + \frac{1}{(1 - \alpha)^2} \right)$$

search miss / insert

Pf.



Parameters.

- M too large \Rightarrow too many empty array entries.
- M too small \Rightarrow search time blows up.
- Typical choice: $\alpha = N/M \sim \frac{1}{2}$. \leftarrow # probes for search hit is about 3/2
probes for search miss is about 5/2

ST implementations: summary

implementation	worst-case cost (after N inserts)			average case (after N random inserts)			ordered iteration?	key interface
	search	insert	delete	search hit	insert	delete		
sequential search (unordered list)	N	N	N	N/2	N	N/2	no	equals()
binary search (ordered array)	$\lg N$	N	N	$\lg N$	N/2	N/2	yes	compareTo()
BST	N	N	N	$1.38 \lg N$	$1.38 \lg N$?	yes	compareTo()
red-black tree	$2 \lg N$	$2 \lg N$	$2 \lg N$	$1.00 \lg N$	$1.00 \lg N$	$1.00 \lg N$	yes	compareTo()
separate chaining	$\lg N^*$	$\lg N^*$	$\lg N^*$	$3-5^*$	$3-5^*$	$3-5^*$	no	equals() hashCode()
linear probing	$\lg N^*$	$\lg N^*$	$\lg N^*$	$3-5^*$	$3-5^*$	$3-5^*$	no	equals() hashCode()

* under uniform hashing assumption

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ **context**

War story: String hashing in Java

String hashCode() in Java 1.1.

- For long strings: only examine 8-9 evenly spaced characters.
- Benefit: saves time in performing arithmetic.

```
public int hashCode()
{
    int hash = 0;
    int skip = Math.max(1, length() / 8);
    for (int i = 0; i < length(); i += skip)
        hash = s[i] + (37 * hash);
    return hash;
}
```

- Downside: great potential for bad collision patterns.

<http://www.cs.princeton.edu/introcs/13loop>Hello.java>
<http://www.cs.princeton.edu/introcs/13loop>Hello.class>
<http://www.cs.princeton.edu/introcs/13loop>Hello.html>
<http://www.cs.princeton.edu/introcs/12type/index.html>

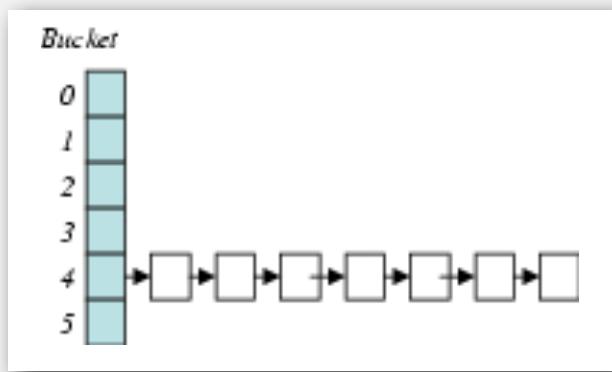


War story: algorithmic complexity attacks

Q. Is the uniform hashing assumption important in practice?

A. Obvious situations: aircraft control, nuclear reactor, pacemaker.

A. Surprising situations: **denial-of-service** attacks.



malicious adversary learns your hash function
(e.g., by reading Java API) and causes a big pile-up
in single slot that grinds performance to a halt

Real-world exploits. [Crosby-Wallach 2003]

- Bro server: send carefully chosen packets to DOS the server, using less bandwidth than a dial-up modem.
- Perl 5.8.0: insert carefully chosen strings into associative array.
- Linux 2.4.20 kernel: save files with carefully chosen names.

Algorithmic complexity attack on Java

Goal. Find family of strings with the same hash code.

Solution. The base 31 hash code is part of Java's string API.

key	hashCode()
"Aa"	2112
"BB"	2112

key	hashCode()
"AaAaAaAa"	-540425984
"AaAaAaBB"	-540425984
"AaAaBBAa"	-540425984
"AaAaBBBB"	-540425984
"AaBBAaAa"	-540425984
"AaBBAaBB"	-540425984
"AaBBBBAa"	-540425984
"AaBBBBBB"	-540425984

key	hashCode()
"BBAaAaAa"	-540425984
"BBAaAaBB"	-540425984
"BBAaBBAa"	-540425984
"BBAaBBBB"	-540425984
"BBBBAaAa"	-540425984
"BBBBAaBB"	-540425984
"BBBBBBAA"	-540425984
"BBBBBBBB"	-540425984

2^N strings of length $2N$ that hash to same value!

Diversion: one-way hash functions

One-way hash function. "Hard" to find a key that will hash to a desired value (or two keys that hash to same value).

Ex. MD4, MD5, SHA-0, SHA-1, SHA-2, WHIRLPOOL, RIPEMD-160,

known to be insecure

```
String password = args[0];
MessageDigest sha1 = MessageDigest.getInstance("SHA1");
byte[] bytes = sha1.digest(password);

/* prints bytes as hex string */
```

Applications. Digital fingerprint, message digest, storing passwords.

Caveat. Too expensive for use in ST implementations.

Separate chaining vs. linear probing

Separate chaining.

- Easier to implement delete.
- Performance degrades gracefully.
- Clustering less sensitive to poorly-designed hash function.

Linear probing.

- Less wasted space.
- Better cache performance.

Q. How to delete?

Q. How to resize?

Hashing: variations on the theme

Many improved versions have been studied.

Two-probe hashing. (separate-chaining variant)

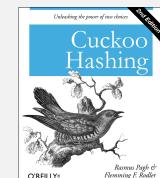
- Hash to two positions, insert key in shorter of the two chains.
- Reduces expected length of the longest chain to $\log \log N$.

Double hashing. (linear-probing variant)

- Use linear probing, but skip a variable amount, not just 1 each time.
- Effectively eliminates clustering.
- Can allow table to become nearly full.
- More difficult to implement delete.

Cuckoo hashing. (linear-probing variant)

- Hash key to two positions; insert key into either position; if occupied, reinsert displaced key into its alternative position (and recur).
- Constant worst case time for search.



Hash tables vs. balanced search trees

Hash tables.

- Simpler to code.
- No effective alternative for unordered keys.
- Faster for simple keys (a few arithmetic ops versus $\log N$ compares).
- Better system support in Java for strings (e.g., cached hash code).

Balanced search trees.

- Stronger performance guarantee.
- Support for ordered ST operations.
- Easier to implement `compareTo()` correctly than `equals()` and `hashCode()`.

Java system includes both.

- Red-black BSTs: `java.util.TreeMap`, `java.util.TreeSet`.
- Hash tables: `java.util.HashMap`, `java.util.IdentityHashMap`.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ **context**



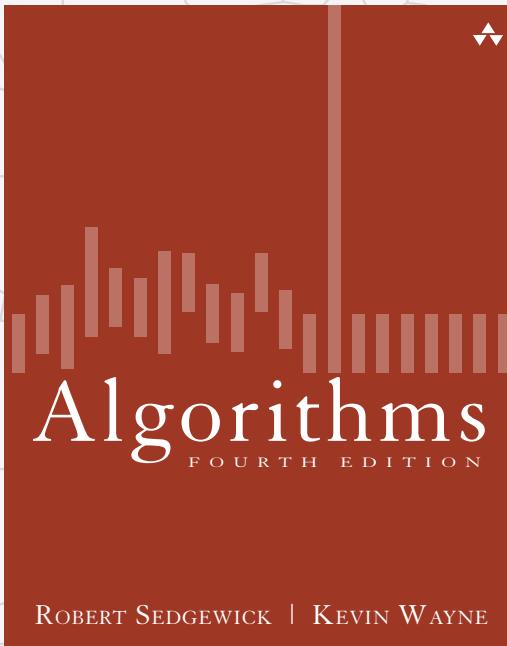
<http://algs4.cs.princeton.edu>

3.4 HASH TABLES

- ▶ *hash functions*
- ▶ *separate chaining*
- ▶ *linear probing*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Set API

Mathematical set. A collection of distinct keys.

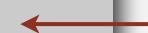
public class SET<Key extends Comparable<Key>>	
SET()	<i>create an empty set</i>
void add(Key key)	<i>add the key to the set</i>
boolean contains(Key key)	<i>is the key in the set?</i>
void remove(Key key)	<i>remove the key from the set</i>
int size()	<i>return the number of keys in the set</i>
Iterator<Key> iterator()	<i>iterator through keys in the set</i>

Q. How to implement?

Exception filter

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

```
% more list.txt  
was it the of  
  
% java WhiteList list.txt < tinyTale.txt  
it was the of it was the of  
  
% java BlackList list.txt < tinyTale.txt  
best times worst times  
age wisdom age foolishness  
epoch belief epoch incredulity  
season light season darkness  
spring hope winter despair
```



list of exceptional words

Exception filter applications

- Read in a list of words from one file.
- Print out all words from standard input that are { in, not in } the list.

application	purpose	key	in list
spell checker	identify misspelled words	word	dictionary words
browser	mark visited pages	URL	visited pages
parental controls	block sites	URL	bad sites
chess	detect draw	board	positions
spam filter	eliminate spam	IP address	spam addresses
credit cards	check for stolen cards	number	stolen cards

Exception filter: Java implementation

- Read in a list of words from one file.
- Print out all words from standard input that are in the list.

```
public class WhiteList
{
    public static void main(String[] args)
    {
        SET<String> set = new SET<String>(); ← create empty set of strings

        In in = new In(args[0]);
        while (!in.isEmpty())
            set.add(in.readString()); ← read in whitelist

        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString();
            if (set.contains(word))
                StdOut.println(word); ← print words not in list
        }
    }
}
```

Exception filter: Java implementation

- Read in a list of words from one file.
- Print out all words from standard input that are **not** in the list.

```
public class BlackList
{
    public static void main(String[] args)
    {
        SET<String> set = new SET<String>(); ← create empty set of strings

        In in = new In(args[0]);
        while (!in.isEmpty())
            set.add(in.readString()); ← read in whitelist

        while (!StdIn.isEmpty())
        {
            String word = StdIn.readString();
            if (!set.contains(word))
                StdOut.println(word); ← print words not in list
        }
    }
}
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 1. DNS lookup.

```
URL is key   IP is value  
% java LookupCSV ip.csv 0 1  
adobe.com  
192.150.18.60  
www.princeton.edu  
128.112.128.15  
ebay.edu  
Not found  
  
IP is key   URL is value  
% java LookupCSV ip.csv 1 0  
128.112.128.15  
www.princeton.edu  
999.999.999.99  
Not found
```

```
% more ip.csv  
www.princeton.edu,128.112.128.15  
www.cs.princeton.edu,128.112.136.35  
www.math.princeton.edu,128.112.18.11  
www.cs.harvard.edu,140.247.50.127  
www.harvard.edu,128.103.60.24  
www.yale.edu,130.132.51.8  
www.econ.yale.edu,128.36.236.74  
www.cs.yale.edu,128.36.229.30  
espn.com,199.181.135.201  
yahoo.com,66.94.234.13  
msn.com,207.68.172.246  
google.com,64.233.167.99  
baidu.com,202.108.22.33  
yahoo.co.jp,202.93.91.141  
sina.com.cn,202.108.33.32  
ebay.com,66.135.192.87  
adobe.com,192.150.18.60  
163.com,220.181.29.154  
passport.net,65.54.179.226  
tom.com,61.135.158.237  
nate.com,203.226.253.11  
cnn.com,64.236.16.20  
daum.net,211.115.77.211  
blogger.com,66.102.15.100  
fastclick.com,205.180.86.4  
wikipedia.org,66.230.200.100  
rakuten.co.jp,202.72.51.22  
...
```

Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 2. Amino acids.

```
codon is key name is value  
% java LookupCSV amino.csv 0 3  
ACT  
Threonine  
TAG  
Stop  
CAT  
Histidine
```

```
% more amino.csv  
TTT,Phe,F,Phenylalanine  
TTC,Phe,F,Phenylalanine  
TTA,Leu,L,Leucine  
TTG,Leu,L,Leucine  
TCT,Ser,S,Serine  
TCC,Ser,S,Serine  
TCA,Ser,S,Serine  
TCG,Ser,S,Serine  
TAT,Tyr,Y,Tyrosine  
TAC,Tyr,Y,Tyrosine  
TAA,Stop,Stop,Stop  
TAG,Stop,Stop,Stop  
TGT,Cys,C,Cysteine  
TGC,Cys,C,Cysteine  
TGA,Stop,Stop,Stop  
TGG,Trp,W,Tryptophan  
CTT,Leu,L,Leucine  
CTC,Leu,L,Leucine  
CTA,Leu,L,Leucine  
CTG,Leu,L,Leucine  
CCT,Pro,P,Proline  
CCC,Pro,P,Proline  
CCA,Pro,P,Proline  
CCG,Pro,P,Proline  
CAT,His,H,Histidine  
CAC,His,H,Histidine  
CAA,Gln,Q,Glutamine  
CAG,Gln,Q,Glutamine  
CGT,Arg,R,Arginine  
CGC,Arg,R,Arginine  
...
```

Dictionary lookup

Command-line arguments.

- A comma-separated value (CSV) file.
- Key field.
- Value field.

Ex 3. Class list.

```
% java LookupCSV classlist.csv 4 1  
eberl  
Ethan  
nwebb  
Natalie
```

first name
login is key is value

```
% java LookupCSV classlist.csv 4 3  
dpan  
P01
```

section
login is key is value

```
% more classlist.csv  
13,Berl,Ethan Michael,P01,eberl  
12,Cao,Phillips Minghua,P01,pcao  
11,Chehoud,Christel,P01,cchehoud  
10,Douglas,Malia Morioka,P01,malia  
12,Haddock,Sara Lynn,P01,shaddock  
12,Hantman,Nicole Samantha,P01,nhantman  
11,Hesterberg,Adam Classen,P01,ahesterb  
13,Hwang,Roland Lee,P01,rhwang  
13,Hyde,Gregory Thomas,P01,ghyde  
13,Kim,Hyunmoon,P01,hktwo  
12,Korac,Damjan,P01,dkorac  
11,MacDonald,Graham David,P01,gmacdona  
10,Michal,Brian Thomas,P01,bmichal  
12,Nam,Seung Hyeon,P01,seungnam  
11,Nastasescu,Maria Monica,P01,mnastase  
11,Pan,Di,P01,dpan  
12,Partridge,Brenton Alan,P01,bpartrid  
13,Rilee,Alexander,P01,arilee  
13,Roopakalu,Ajay,P01,aroopaka  
11,Sheng,Ben C,P01,bsheng  
12,Webb,Natalie Sue,P01,nwebb  
:
```

Dictionary lookup: Java implementation

```
public class LookupCSV
{
    public static void main(String[] args)
    {
        In in = new In(args[0]);
        int keyField = Integer.parseInt(args[1]);
        int valField = Integer.parseInt(args[2]);
```

← process input file

```
        ST<String, String> st = new ST<String, String>();
        while (!in.isEmpty())
        {
            String line = in.readLine();
            String[] tokens = database[i].split(",");
            String key = tokens[keyField];
            String val = tokens[valField];
            st.put(key, val);
        }
```

← build symbol table

```
        while (!StdIn.isEmpty())
        {
            String s = StdIn.readString();
            if (!st.contains(s)) StdOut.println("Not found");
            else                 StdOut.println(st.get(s));
        }
    }
}
```

← process lookups
with standard I/O

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

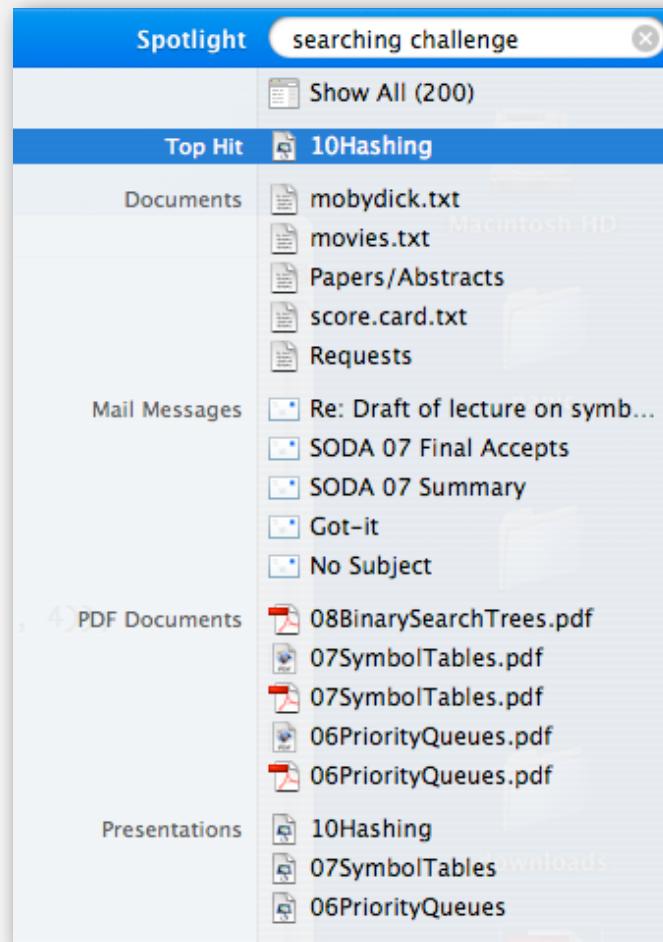
<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

File indexing

Goal. Index a PC (or the web).



File indexing

Goal. Given a list of files specified, create an index so that you can efficiently find all files containing a given query string.

```
% ls *.txt  
aesop.txt magna.txt moby.txt  
sawyer.txt tale.txt
```

```
% java FileIndex *.txt  
  
freedom  
magna.txt moby.txt tale.txt
```

```
whale  
moby.txt
```

```
lamb  
sawyer.txt aesop.txt
```

```
% ls *.java  
BlackList.java Concordance.java  
DeDup.java FileIndex.java ST.java  
SET.java WhiteList.java
```

```
% java FileIndex *.java
```

```
import  
FileIndex.java SET.java ST.java
```

```
Comparator  
null
```

Solution. Key = query string; value = set of files containing that string.

File indexing

```
import java.io.File;
public class FileIndex
{
    public static void main(String[] args)
    {
        ST<String, SET<File>> st = new ST<String, SET<File>>(); ← symbol table

        for (String filename : args) {
            File file = new File(filename);
            In in = new In(file);
            while (!in.isEmpty())
            {
                String key = in.readString();
                if (!st.contains(key))
                    st.put(key, new SET<File>());
                SET<File> set = st.get(key);
                set.add(file);
            }
        }

        while (!StdIn.isEmpty())
        {
            String query = StdIn.readString();
            StdOut.println(st.get(query));
        }
    }
}
```

list of file names from command line

for each word in file, add file to corresponding set

process queries

Book index

Goal. Index for an e-book.

The image shows a screenshot of an e-book index. The title "Index" is centered at the top. Below it, the index is presented in two columns. The left column contains entries starting with "Abstract data type (ADT)", while the right column contains entries starting with "stack of int (intStack)". The entries are listed in a standard monospaced font, with some terms underlined as links. The background of the page is white, and the overall layout is clean and organized.

Abstract data type (ADT), 127-195	stack of int (intStack), 140
abstract classes, 163	symbol table (ST), 503
classes, 129-136	text index (TI), 525
collections of items, 137-139	union-find (UF), 159
creating, 157-164	Abstract in-place merging, 351-353
defined, 128	Abstract operation, 10
duplicate items, 173-176	Access control state, 131
equivalence-relations, 159-162	Actual data, 31
FIFO queues, 165-171	Adapter class, 155-157
first-class, 177-186	Adaptive sort, 268
generic operations, 273	Address, 84-85
index items, 177	Adjacency list, 120-123
insert/remove operations, 138-139	depth-first search, 251-256
modular programming, 135	Adjacency matrix, 120-122
polynomial, 188-192	Ajtai, M., 464
priority queues, 375-376	Algorithm, 4-6, 27-64
pushdown stack, 138-156	abstract operations, 10, 31, 34-35
stubs, 135	analysis of, 6
symbol table, 497-506	average-worst-case performance, 35, 60-62
ADT interfaces	big-Oh notation, 44-47
array (<code>myArray</code>), 274	binary search, 56-59
complex number (<code>Complex</code>), 181	computational complexity, 62-64
existence table (ET), 663	efficiency, 6, 30, 32
full priority queue (<code>PQfull</code>), 397	empirical analysis, 30-32, 58
indirect priority queue (<code>PQi</code>), 403	exponential-time, 219
item (<code>myItem</code>), 273, 498	implementation, 28-30
key (<code>myKey</code>), 498	logarithm function, 40-43
polynomial (<code>Poly</code>), 189	mathematical analysis, 33-36, 58
point (<code>Point</code>), 134	primary parameter, 36
priority queue (<code>PQ</code>), 375	probabilistic, 331
queue of int (<code>intQueue</code>), 166	recurrences, 49-52, 57
	recursive, 198
	running time, 34-40
	search, 53-56, 498
	steps in, 22-23
	<i>See also</i> Randomized algorithm
	Amortization approach, 557, 627
	Arithmetic operator, 177-179, 188, 191
	Array, 12, 83
	binary search, 57
	dynamic allocation, 87
	and linked lists, 92, 94-95
	merging, 349-350
	multidimensional, 117-118
	references, 86-87, 89
	sorting, 265-267, 273-276
	and strings, 119
	two-dimensional, 117-118, 120-124
	vectors, 87
	visualizations, 295
	<i>See also</i> Index, array
	Array representation
	binary tree, 381
	FIFO queue, 168-169
	linked lists, 110
	polynomial ADT, 191-192
	priority queue, 377-378, 403, 406
	pushdown stack, 148-150
	random queue, 170
	symbol table, 508, 511-512, 521
	Asymptotic expression, 45-46
	Average deviation, 80-81
	Average-case performance, 35, 60-61
	AVL tree, 583
	B tree, 584, 692-704
	external/internal pages, 695
	4-5-6-7-8 tree, 693-704
	Markov chain, 701
	remove, 701-703
	search/insert, 697-701
	select/sort, 701
	Balanced tree, 238, 555-598
	B tree, 584
	bottom-up, 576, 584-585
	height-balanced, 583
	indexed sequential access, 690-692
	performance, 575-576, 581-582, 595-598
	randomized, 559-564
	red-black, 577-585
	skip lists, 587-594
	splay, 566-571

Concordance

Goal. Preprocess a text corpus to support concordance queries: given a word, find all occurrences with their immediate contexts.

```
% java Concordance tale.txt  
cities  
tongues of the two *cities* that were blended in  
  
majesty  
their turnkeys and the *majesty* of the law fired  
me treason against the *majesty* of the people in  
of his most gracious *majesty* king george the third  
  
princeton  
no matches
```

Concordance

```
public class Concordance
{
    public static void main(String[] args)
    {
        In in = new In(args[0]);
        String[] words = StdIn.readStrings();
        ST<String, SET<Integer>> st = new ST<String, SET<Integer>>();
        for (int i = 0; i < words.length; i++)
        {
            String s = words[i];
            if (!st.contains(s))
                st.put(s, new SET<Integer>());
            SET<Integer> set = st.get(s);
            set.put(i);
        }
    }

    while (!StdIn.isEmpty())
    {
        String query = StdIn.readString();
        SET<Integer> set = st.get(query);
        for (int k : set)
            // print words[k-4] to words[k+4]
        }
    }
}
```

← read text and build index

← process queries and print concordances

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ ***sparse vectors***

Matrix-vector multiplication (standard implementation)

$$\begin{array}{c} \text{a[][]} & \text{x[]} & \text{b[]} \\ \left[\begin{array}{ccccc} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ -.47 & 0 & .47 & 0 & 0 \end{array} \right] & \left[\begin{array}{c} .05 \\ .04 \\ .36 \\ .37 \\ .19 \end{array} \right] & = \left[\begin{array}{c} .036 \\ .297 \\ .333 \\ .045 \\ .1927 \end{array} \right] \end{array}$$

```
...
double[][] a = new double[N][N];
double[] x = new double[N];
double[] b = new double[N];

...
// initialize a[][] and x[]

for (int i = 0; i < N; i++)
{
    sum = 0.0;
    for (int j = 0; j < N; j++)
        sum += a[i][j]*x[j];
    b[i] = sum;
}
```

nested loops
(N^2 running time)

Sparse matrix-vector multiplication

Problem. Sparse matrix-vector multiplication.

Assumptions. Matrix dimension is 10,000; average nonzeros per row ~ 10 .

A sparse matrix A is shown as a grid of dots. Most dots are blue, representing non-zero elements, while others are black, representing zeros. To the right of A , a vector x is represented by a vertical column of blue dots. Below A and x , the equation $A \cdot x = b$ is written in red, where b is represented by a vertical column of black dots.

$$A \cdot x = b$$

Vector representations

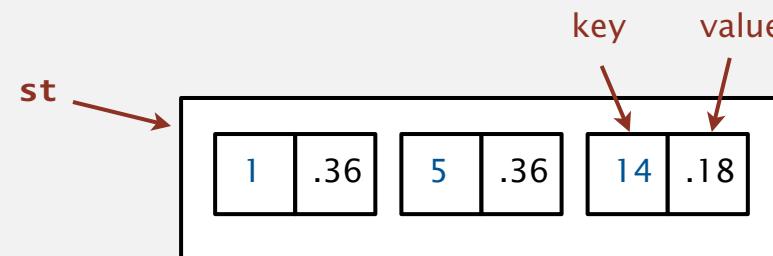
1d array (standard) representation.

- Constant time access to elements.
- Space proportional to N.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	.36	0	0	0	.36	0	0	0	0	0	0	0	0	.18	0	0	0	0	0

Symbol table representation.

- Key = index, value = entry.
- Efficient iterator.
- Space proportional to number of nonzeros.



Sparse vector data type

```
public class SparseVector
{
    private HashST<Integer, Double> v; ← HashST because order not important

    public SparseVector()
    { v = new HashST<Integer, Double>(); } ← empty ST represents all 0s vector

    public void put(int i, double x)
    { v.put(i, x); } ← a[i] = value

    public double get(int i)
    {
        if (!v.contains(i)) return 0.0;
        else return v.get(i); } ← return a[i]

    public Iterable<Integer> indices()
    { return v.keys(); }

    public double dot(double[] that)
    {
        double sum = 0.0; ← dot product is constant
        for (int i : indices())
            sum += that[i]*this.get(i);
        return sum;
    }
}
```

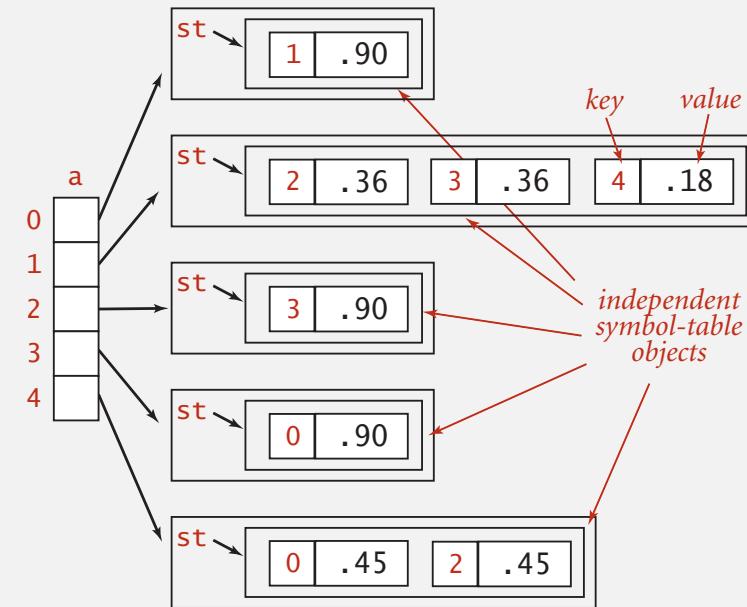
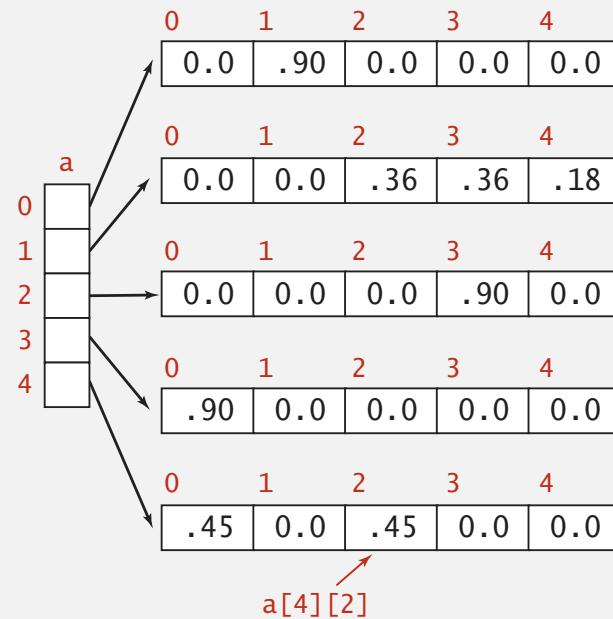
Matrix representations

2D array (standard) matrix representation: Each row of matrix is an **array**.

- Constant time access to elements.
- Space proportional to N^2 .

Sparse matrix representation: Each row of matrix is a **sparse vector**.

- Efficient access to elements.
- Space proportional to number of nonzeros (plus N).



Sparse matrix-vector multiplication

$$\begin{array}{c} \text{a[][]} \\ \left[\begin{array}{ccccc} 0 & .90 & 0 & 0 & 0 \\ 0 & 0 & .36 & .36 & .18 \\ 0 & 0 & 0 & .90 & 0 \\ .90 & 0 & 0 & 0 & 0 \\ .47 & 0 & .47 & 0 & 0 \end{array} \right] \end{array} \quad \begin{array}{c} \text{x[]} \\ \left[\begin{array}{c} .05 \\ .04 \\ .36 \\ .37 \\ .19 \end{array} \right] \end{array} \quad = \quad \begin{array}{c} \text{b[]} \\ \left[\begin{array}{c} .036 \\ .297 \\ .333 \\ .045 \\ .1927 \end{array} \right] \end{array}$$

```
...
SparseVector[] a = new SparseVector[N];
double[] x = new double[N];
double[] b = new double[N];
...
// Initialize a[] and x[]
...
for (int i = 0; i < N; i++)
    b[i] = a[i].dot(x);
```



linear running time
for sparse matrix

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

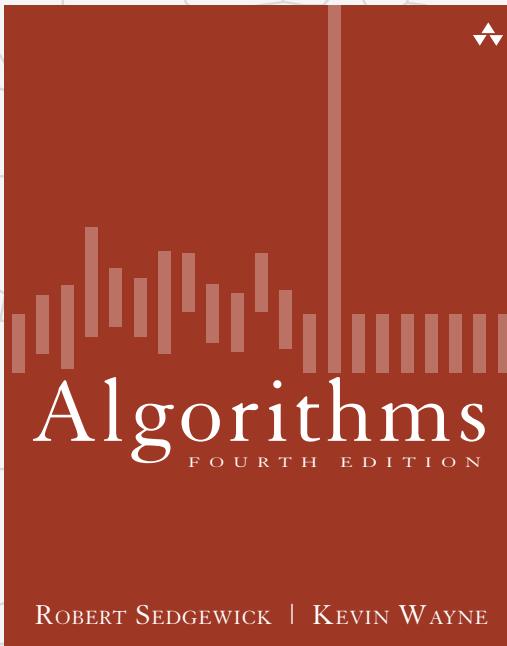
<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ ***sparse vectors***

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

3.5 SYMBOL TABLE APPLICATIONS

- ▶ *sets*
- ▶ *dictionary clients*
- ▶ *indexing clients*
- ▶ *sparse vectors*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



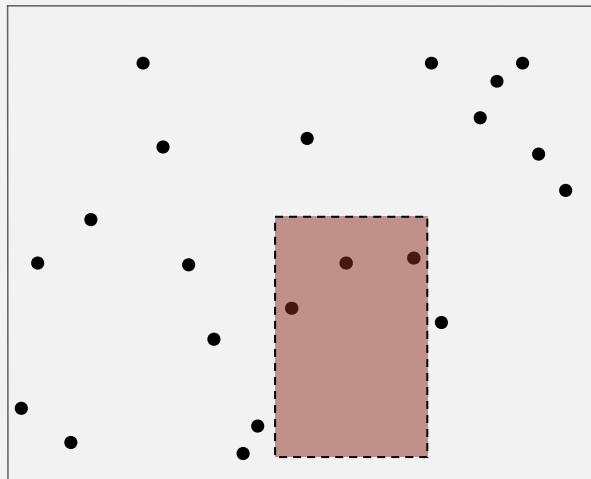
<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

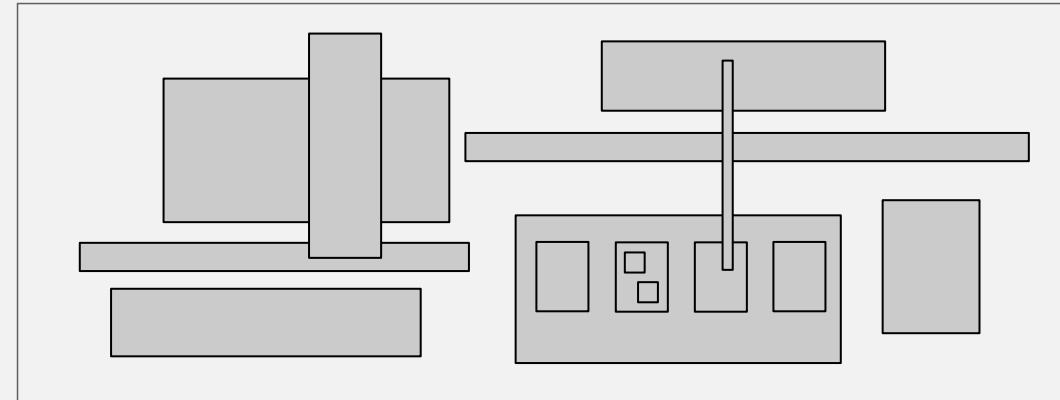
- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Overview

This lecture. Intersections among **geometric objects**.



2d orthogonal range search



orthogonal rectangle intersection

Applications. CAD, games, movies, virtual reality, databases, GIS,

Efficient solutions. **Binary search trees** (and extensions).

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

1d range search

Extension of ordered symbol table.

- Insert key-value pair.
- Search for key k .
- Delete key k .
- Range search: find all keys between k_1 and k_2 .
- Range count: number of keys between k_1 and k_2 .

Application. Database queries.

Geometric interpretation.

- Keys are point on a line.
- Find/count points in a given 1d interval.

• • • • [• • •] • • • •

insert B	B
insert D	B D
insert A	A B D
insert I	A B D I
insert H	A B D H I
insert F	A B D F H I
insert P	A B D F H I P
count G to K	2
search G to K	H I

1d range search: implementations

Unordered list. Fast insert, slow range search.

Ordered array. Slow insert, binary search for k_1 and k_2 to do range search.

order of growth of running time for 1d range search

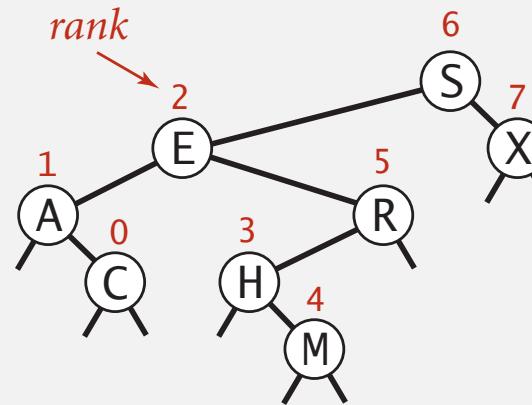
data structure	insert	range count	range search
unordered list	1	N	N
ordered array	N	log N	R + log N
goal	log N	log N	R + log N

N = number of keys

R = number of keys that match

1d range count: BST implementation

1d range count. How many keys between l_o and h_i ?



```
public int size(Key lo, Key hi)
{
    if (contains(hi)) return rank(hi) - rank(lo) + 1;
    else               return rank(hi) - rank(lo);
}
```

number of keys < hi

Proposition. Running time proportional to $\log N$.

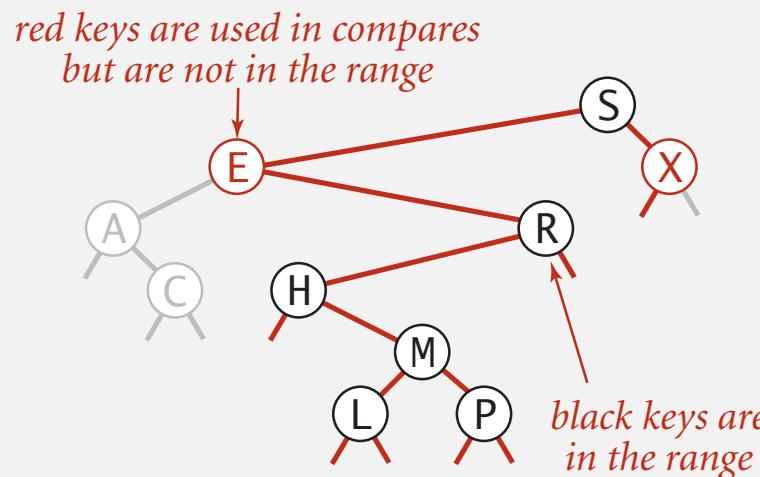
Pf. Nodes examined = search path to l_o + search path to h_i .

1d range search: BST implementation

1d range search. Find all keys between l_0 and h_i .

- Recursively find all keys in left subtree (if any could fall in range).
- Check key in current node.
- Recursively find all keys in right subtree (if any could fall in range).

searching in the range $[F \dots T]$



Proposition. Running time proportional to $R + \log N$.

Pf. Nodes examined = search path to l_0 + search path to h_i + matches.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

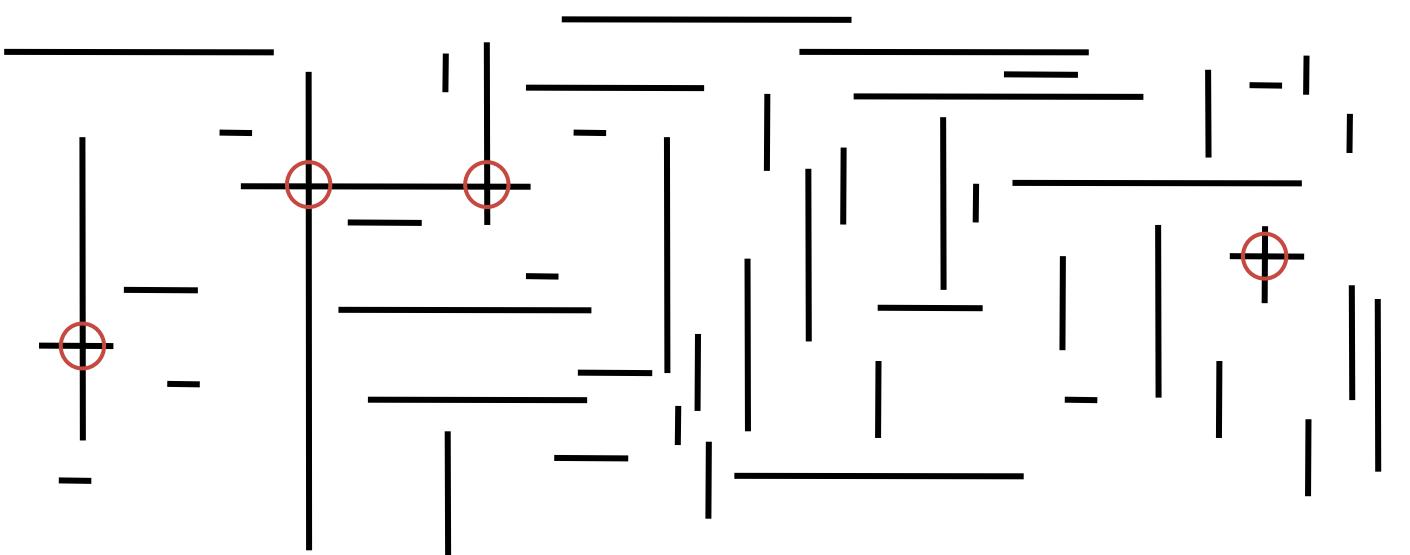
<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Orthogonal line segment intersection

Given N horizontal and vertical line segments, find all intersections.



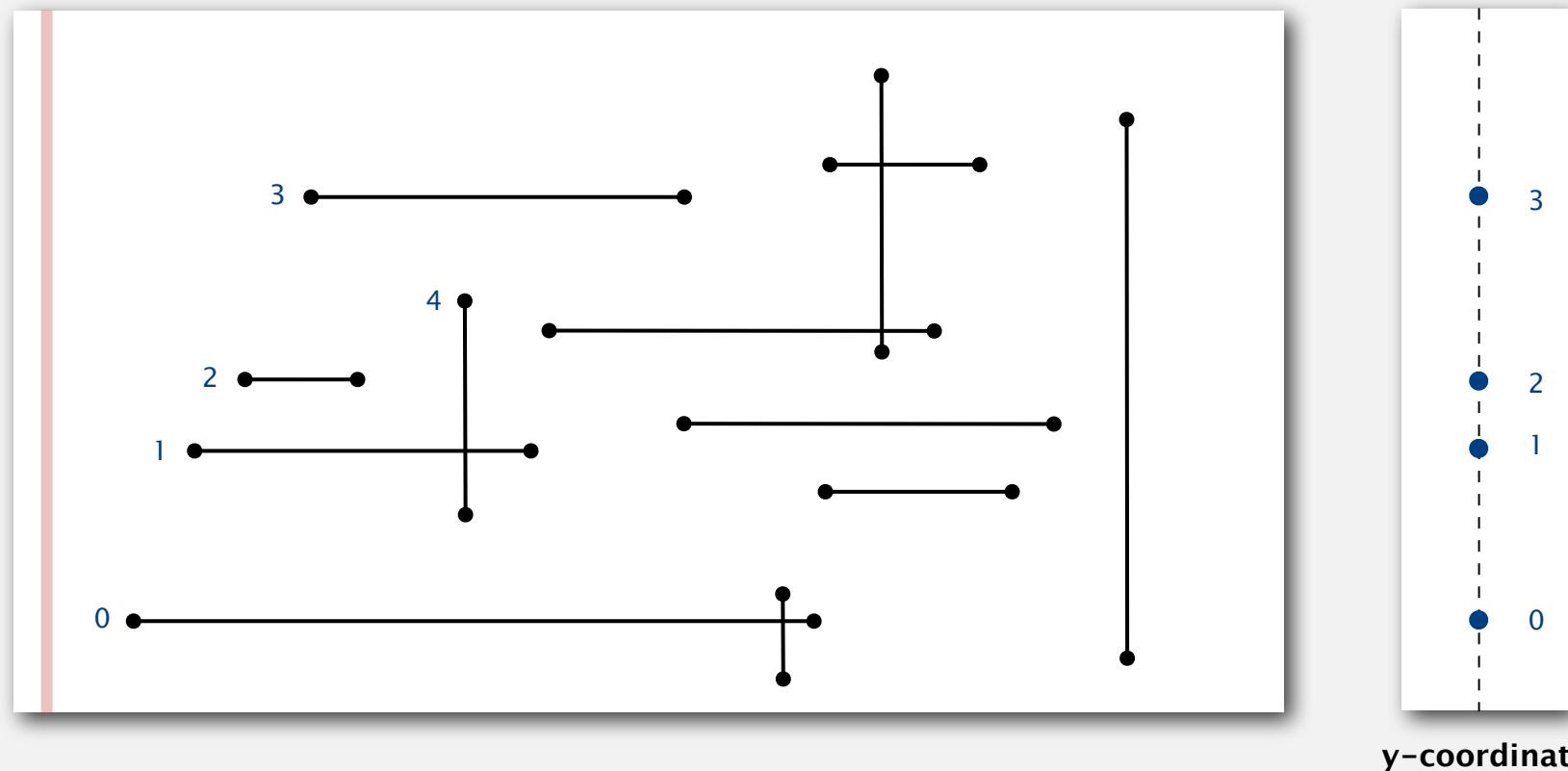
Quadratic algorithm. Check all pairs of line segments for intersection.

Nondegeneracy assumption. All x - and y -coordinates are distinct.

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x -coordinates define events.
- h -segment (left endpoint): insert y -coordinate into BST.

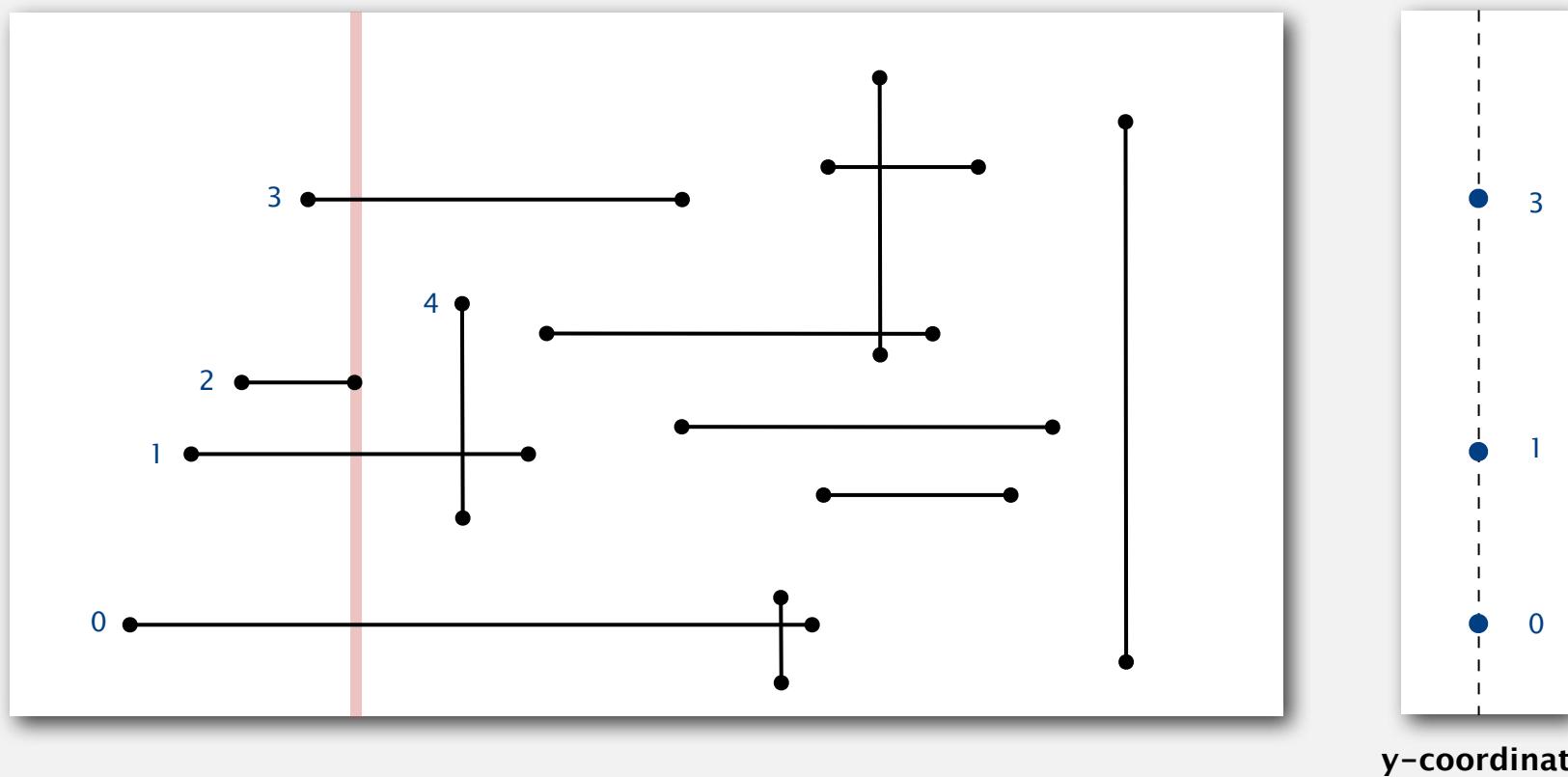


y-coordinates

Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

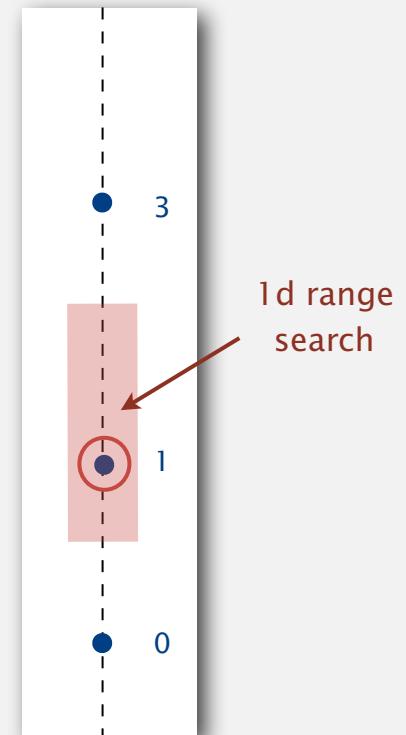
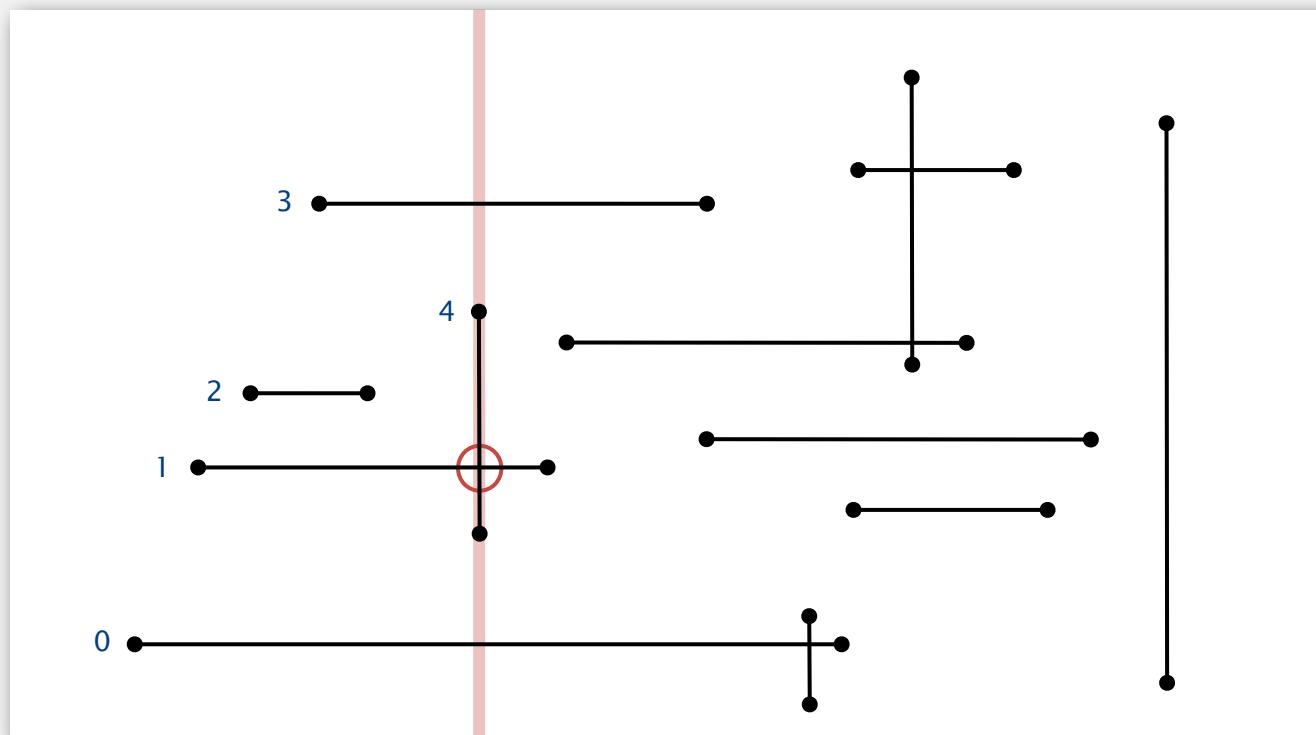
- x -coordinates define events.
- h -segment (left endpoint): insert y -coordinate into BST.
- h -segment (right endpoint): remove y -coordinate from BST.



Orthogonal line segment intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x -coordinates define events.
- h -segment (left endpoint): insert y -coordinate into BST.
- h -segment (right endpoint): remove y -coordinate from BST.
- v -segment: range search for interval of y -endpoints.



y-coordinates

Orthogonal line segment intersection: sweep-line analysis

Proposition. The sweep-line algorithm takes time proportional to $N \log N + R$ to find all R intersections among N orthogonal line segments.

Pf.

- Put x -coordinates on a PQ (or sort). $\leftarrow N \log N$
- Insert y -coordinates into BST. $\leftarrow N \log N$
- Delete y -coordinates from BST. $\leftarrow N \log N$
- Range searches in BST. $\leftarrow N \log N + R$

Bottom line. Sweep line reduces 2d orthogonal line segment intersection search to 1d range search.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

2-d orthogonal range search

Extension of ordered symbol-table to 2d keys.

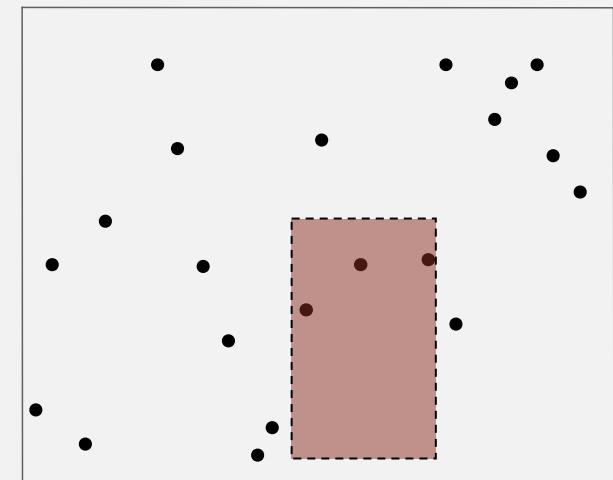
- Insert a 2d key.
- Delete a 2d key.
- Search for a 2d key.
- Range search: find all keys that lie in a 2d range.
- Range count: number of keys that lie in a 2d range.

Applications. Networking, circuit design, databases, ...

Geometric interpretation.

- Keys are point in the plane.
- Find/count points in a given *h-v* rectangle

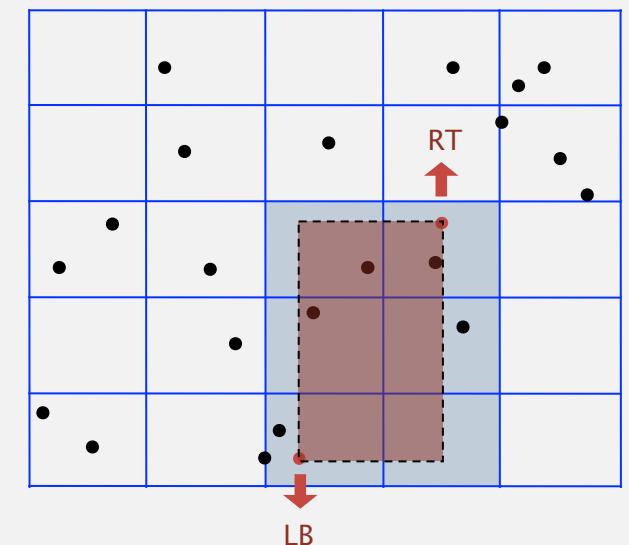
↑
rectangle is axis-aligned



2d orthogonal range search: grid implementation

Grid implementation.

- Divide space into M -by- M grid of squares.
- Create list of points contained in each square.
- Use 2d array to directly index relevant square.
- Insert: add (x, y) to list for corresponding square.
- Range search: examine only squares that intersect 2d range query.



2d orthogonal range search: grid implementation analysis

Space-time tradeoff.

- Space: $M^2 + N$.
- Time: $1 + N/M^2$ per square examined, on average.

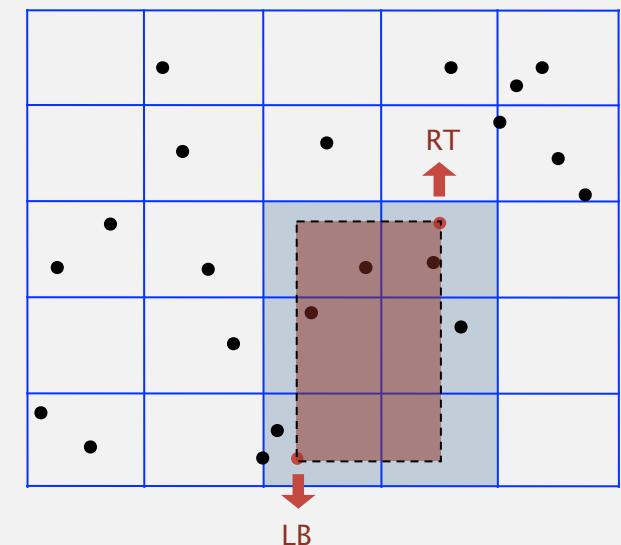
Choose grid square size to tune performance.

- Too small: wastes space.
- Too large: too many points per square.
- Rule of thumb: \sqrt{N} -by- \sqrt{N} grid.

Running time. [if points are evenly distributed]

- Initialize data structure: N .
- Insert point: 1.
- Range search: 1 per point in range.

choose $M \sim \sqrt{N}$

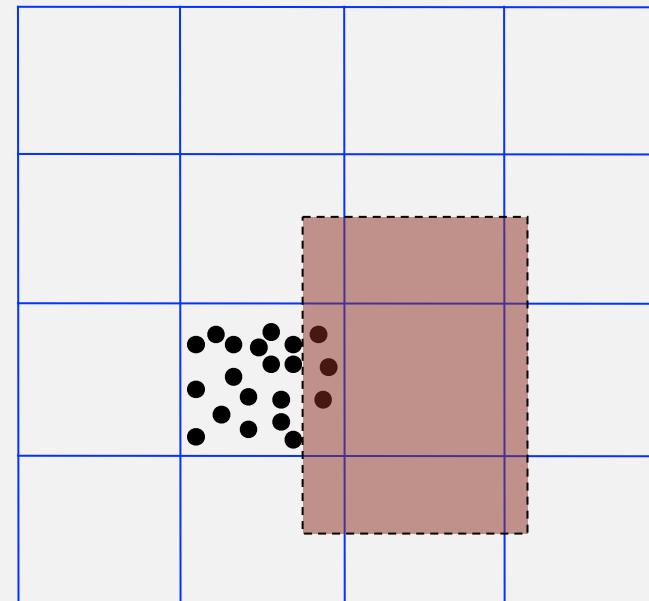


Clustering

Grid implementation. Fast, simple solution for evenly-distributed points.

Problem. Clustering a well-known phenomenon in geometric data.

- Lists are too long, even though average length is short.
- Need data structure that adapts gracefully to data.



Clustering

Grid implementation. Fast, simple solution for evenly-distributed points.

Problem. Clustering a well-known phenomenon in geometric data.

Ex. USA map data.



13,000 points, 1000 grid squares



half the squares are empty

half the points are
in 10% of the squares

Space-partitioning trees

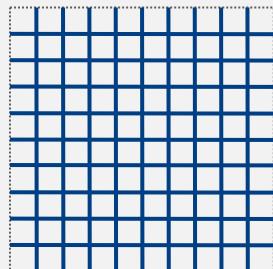
Use a **tree** to represent a recursive subdivision of 2d space.

Grid. Divide space uniformly into squares.

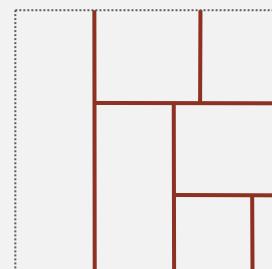
2d tree. Recursively divide space into two halfplanes.

Quadtree. Recursively divide space into four quadrants.

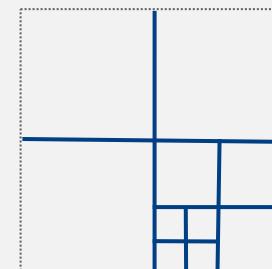
BSP tree. Recursively divide space into two regions.



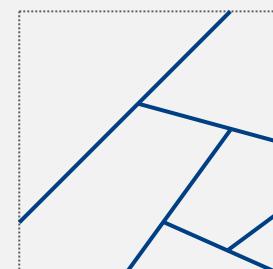
Grid



2d tree



Quadtree

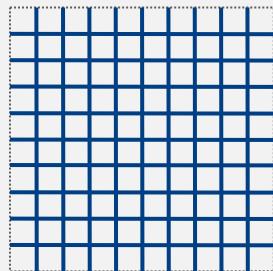


BSP tree

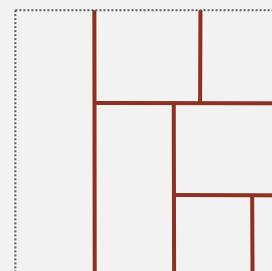
Space-partitioning trees: applications

Applications.

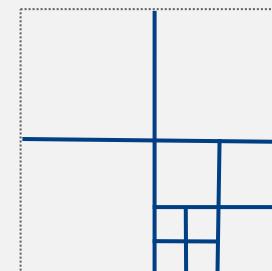
- Ray tracing.
- **2d range search.**
- Flight simulators.
- N-body simulation.
- Collision detection.
- Astronomical databases.
- **Nearest neighbor search.**
- Adaptive mesh generation.
- Accelerate rendering in Doom.
- Hidden surface removal and shadow casting.



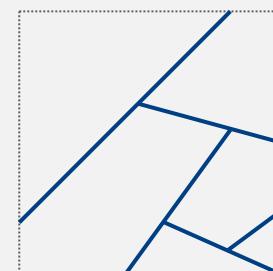
Grid



2d tree



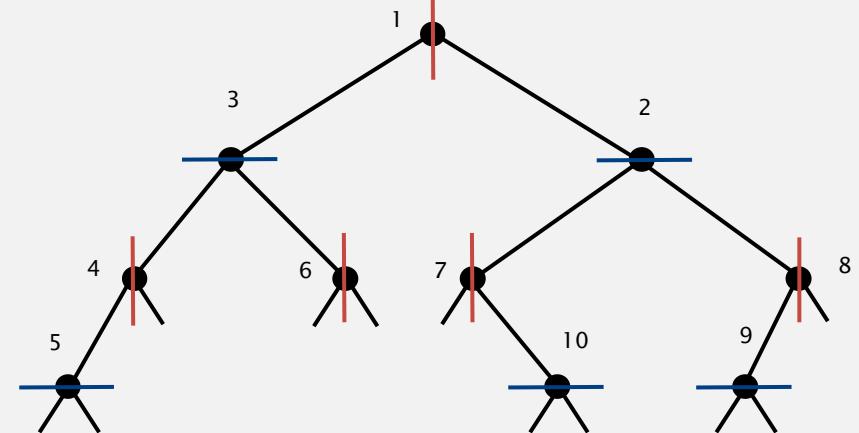
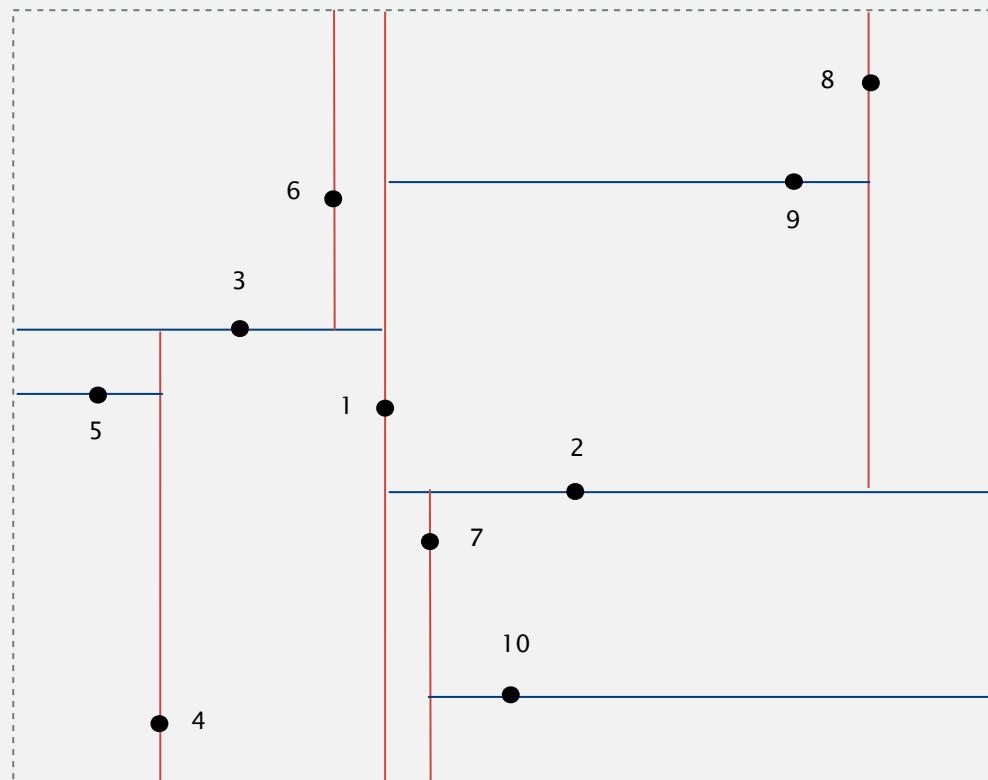
Quadtree



BSP tree

2d tree construction

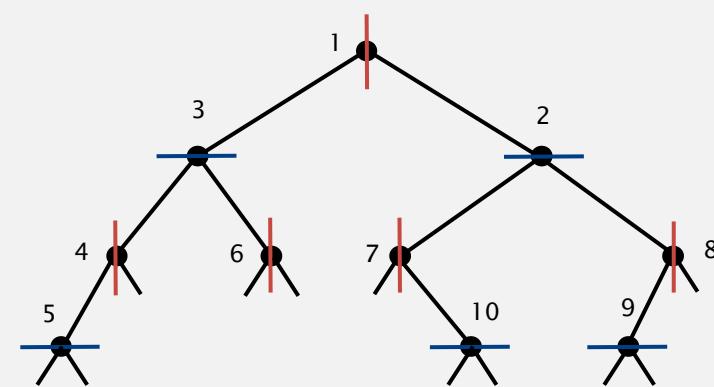
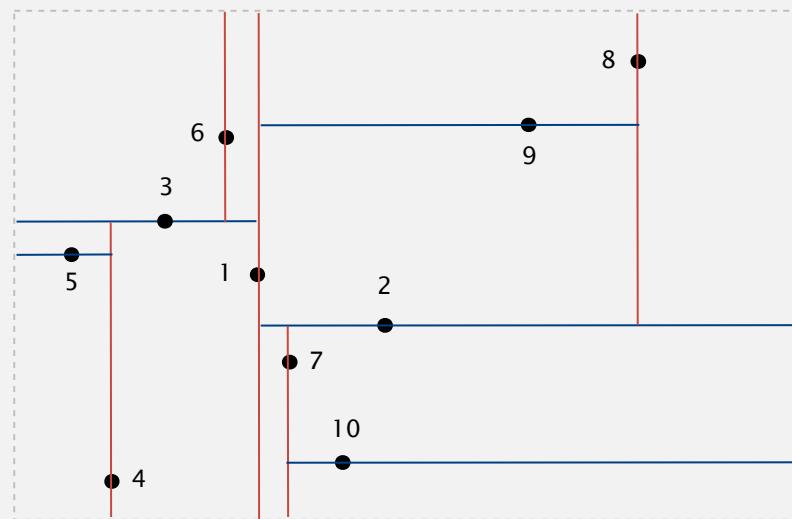
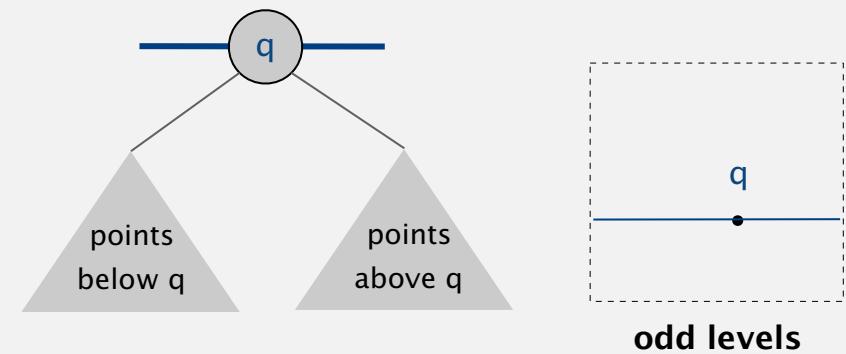
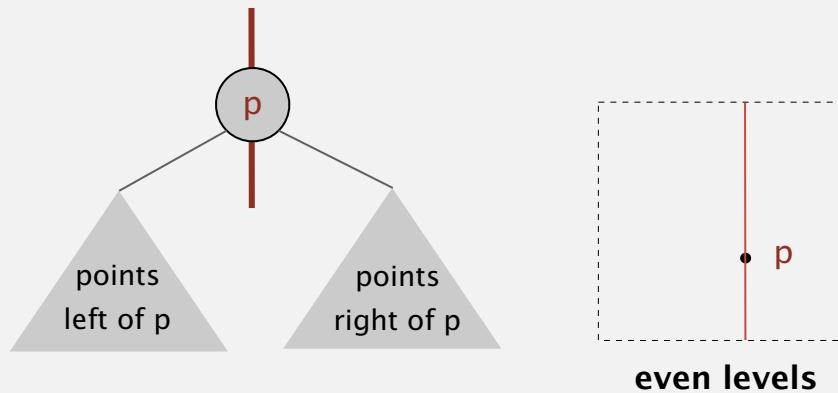
Recursively partition plane into two halfplanes.



2d tree implementation

Data structure. BST, but alternate using x - and y -coordinates as key.

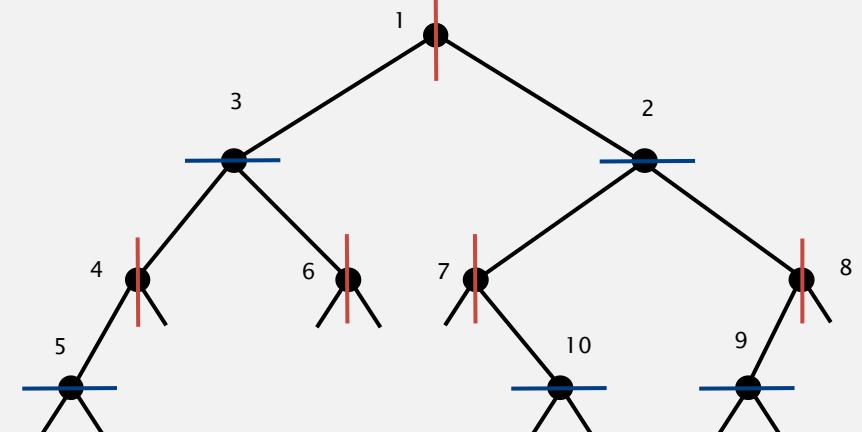
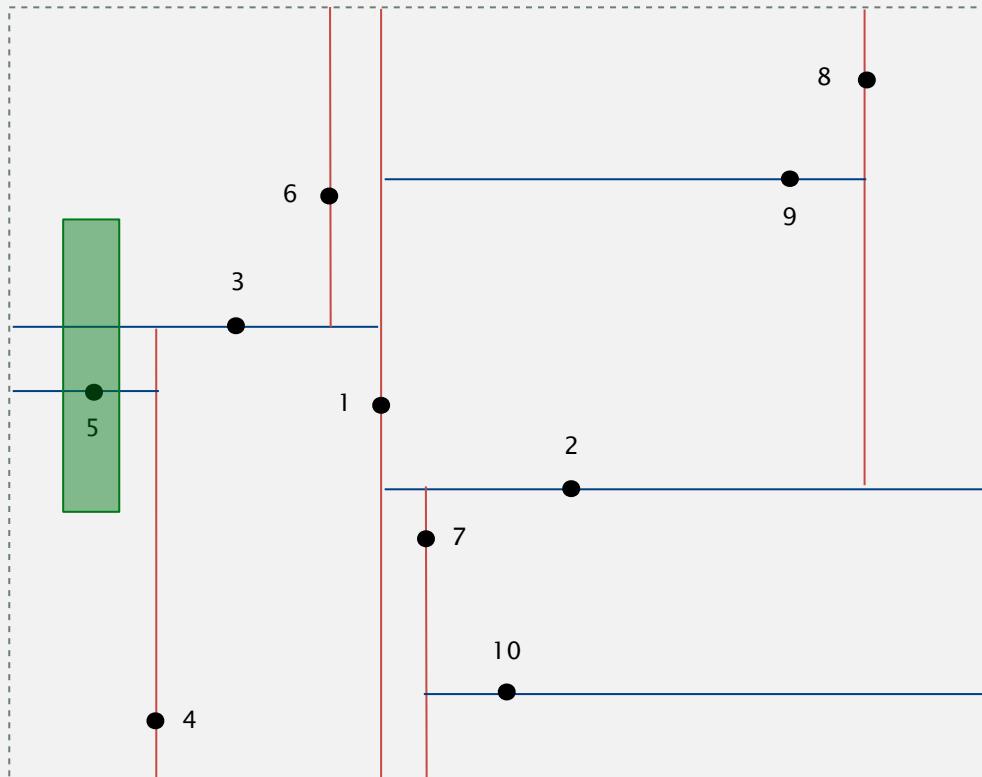
- Search gives rectangle containing point.
- Insert further subdivides the plane.



Range search in a 2d tree demo

Goal. Find all points in a query axis-aligned rectangle.

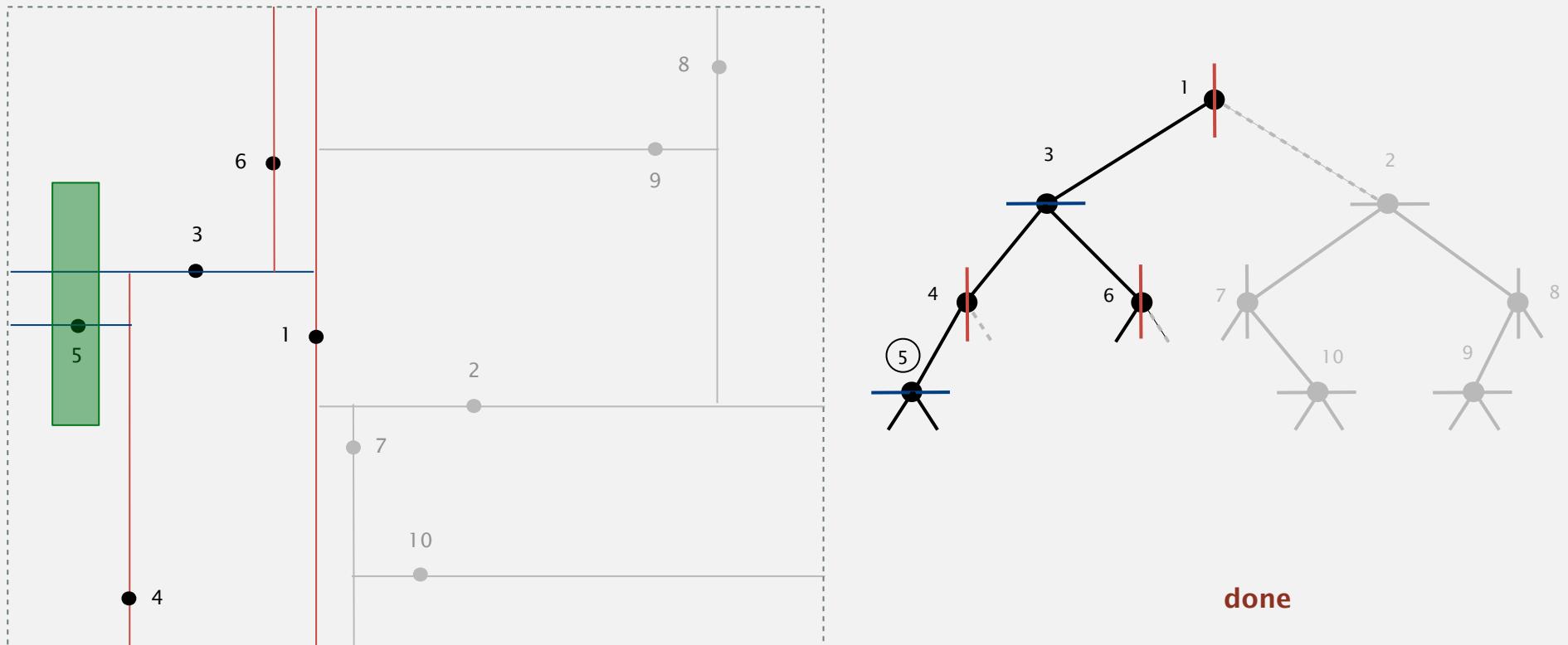
- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



Range search in a 2d tree demo

Goal. Find all points in a query axis-aligned rectangle.

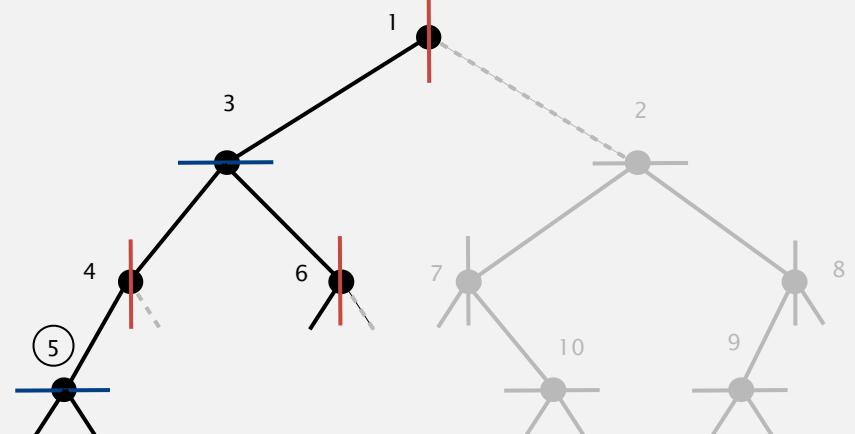
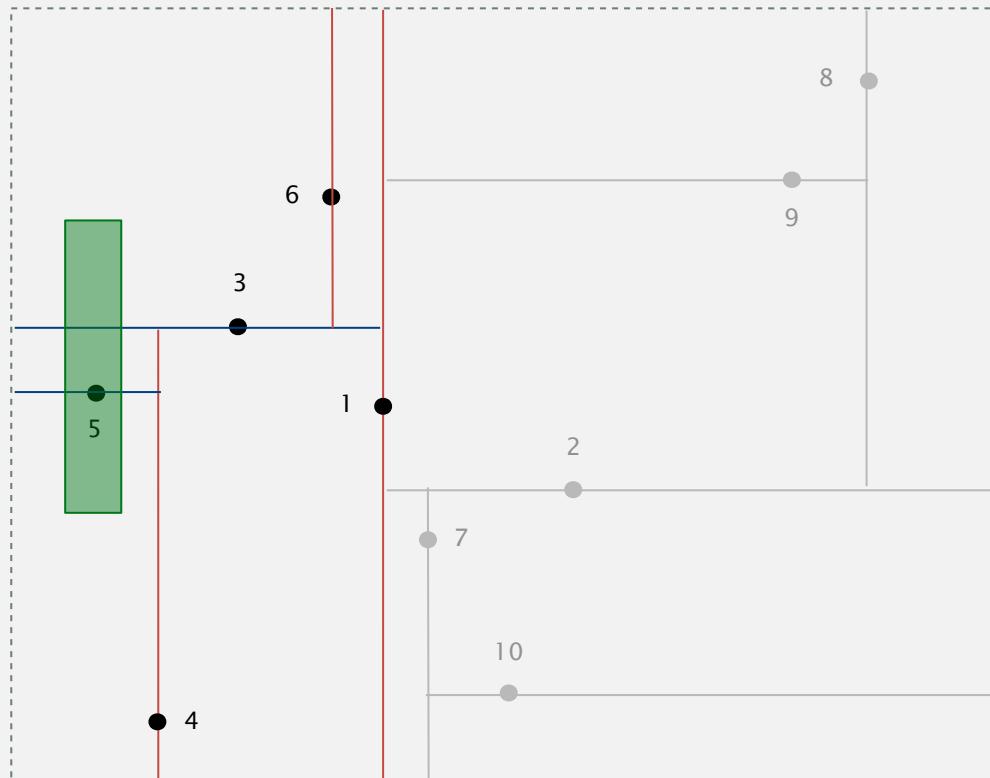
- Check if point in node lies in given rectangle.
- Recursively search left/bottom (if any could fall in rectangle).
- Recursively search right/top (if any could fall in rectangle).



Range search in a 2d tree analysis

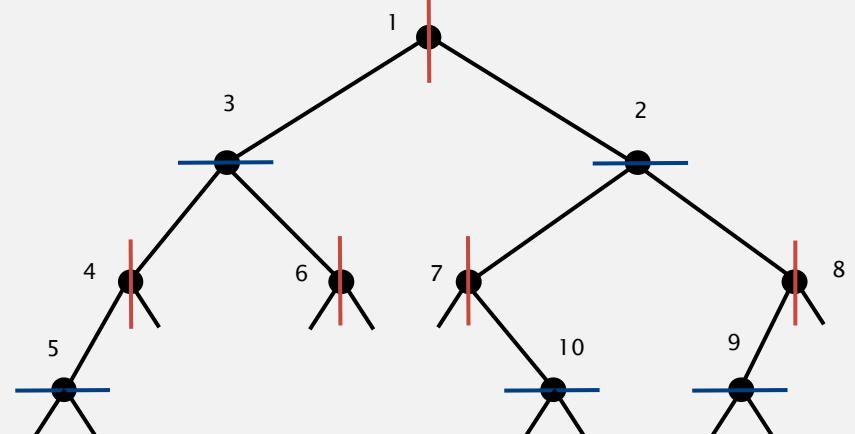
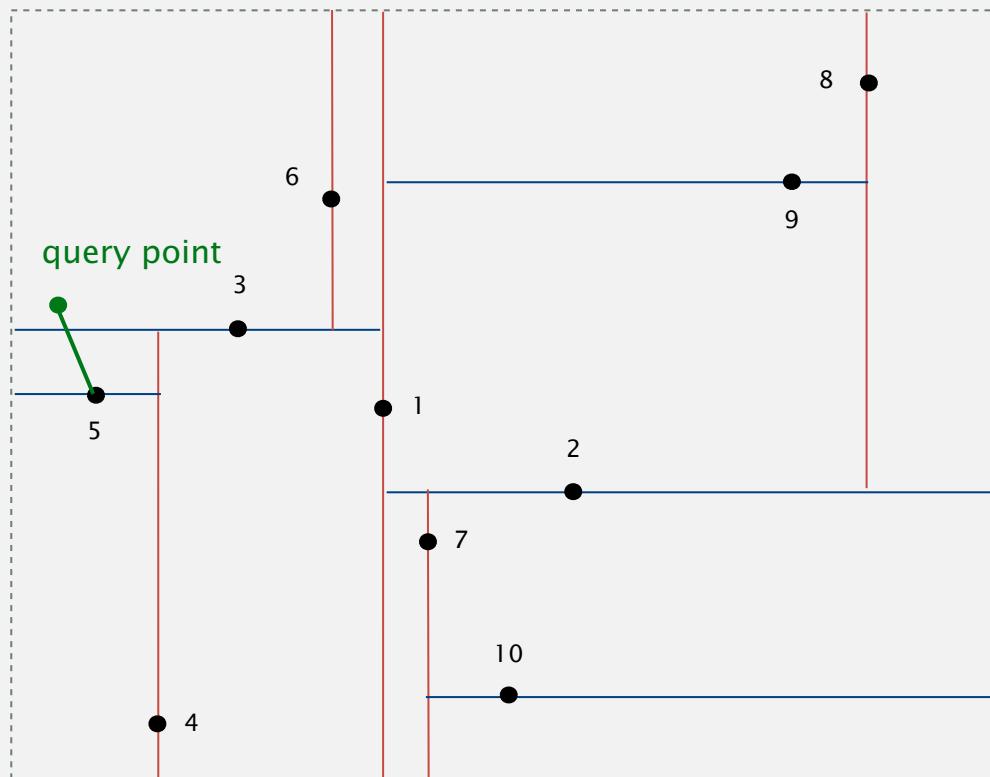
Typical case. $R + \log N$.

Worst case (assuming tree is balanced). $R + \sqrt{N}$.



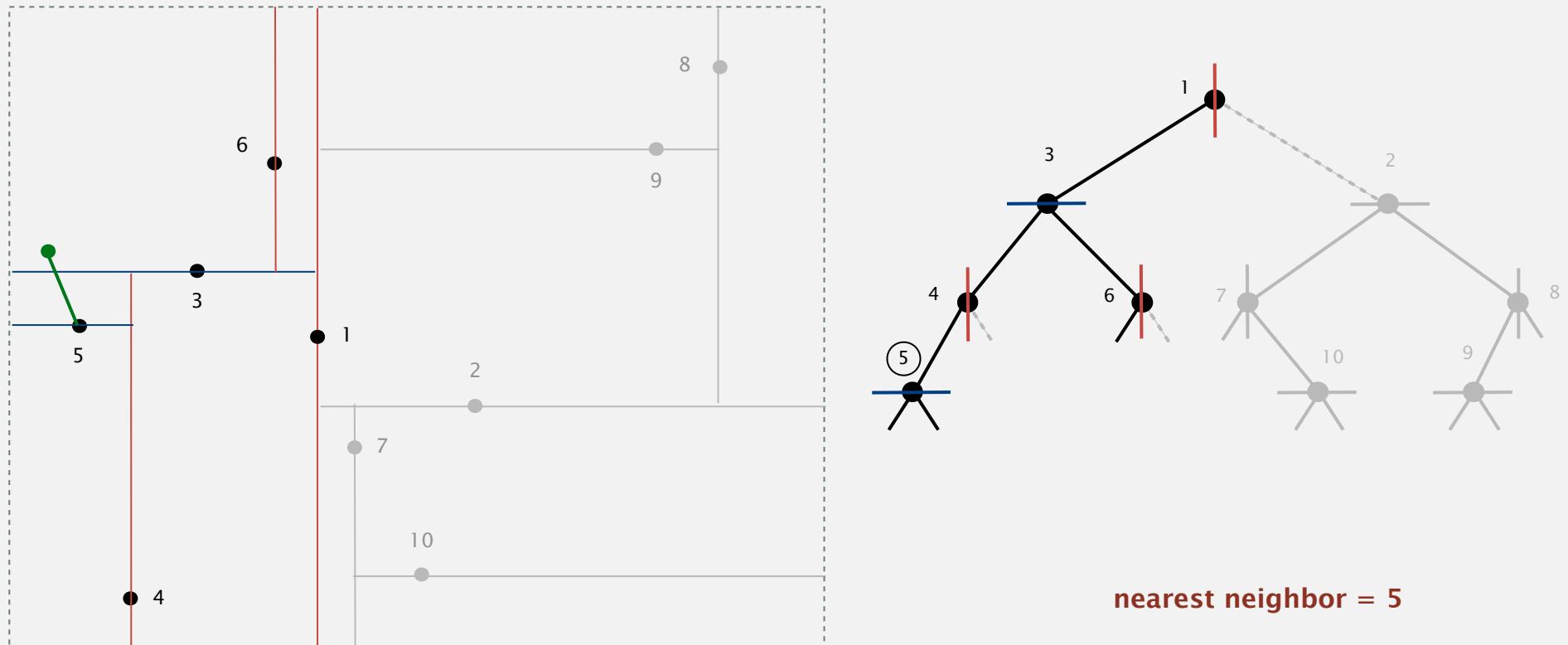
Nearest neighbor search in a 2d tree demo

Goal. Find closest point to query point.



Nearest neighbor search in a 2d tree demo

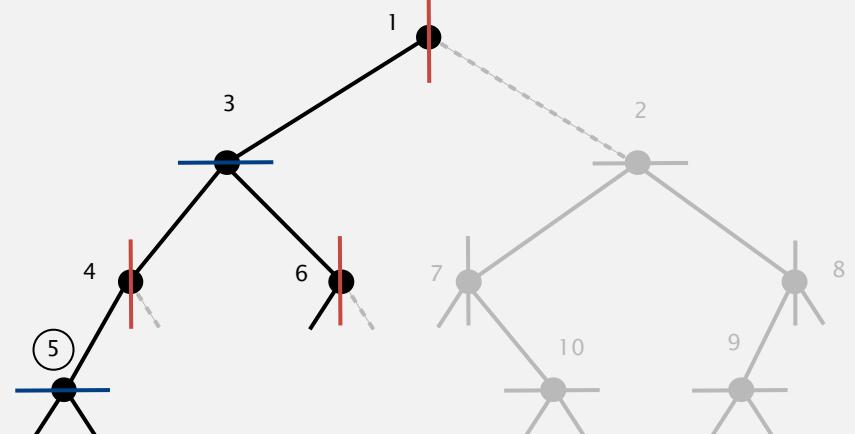
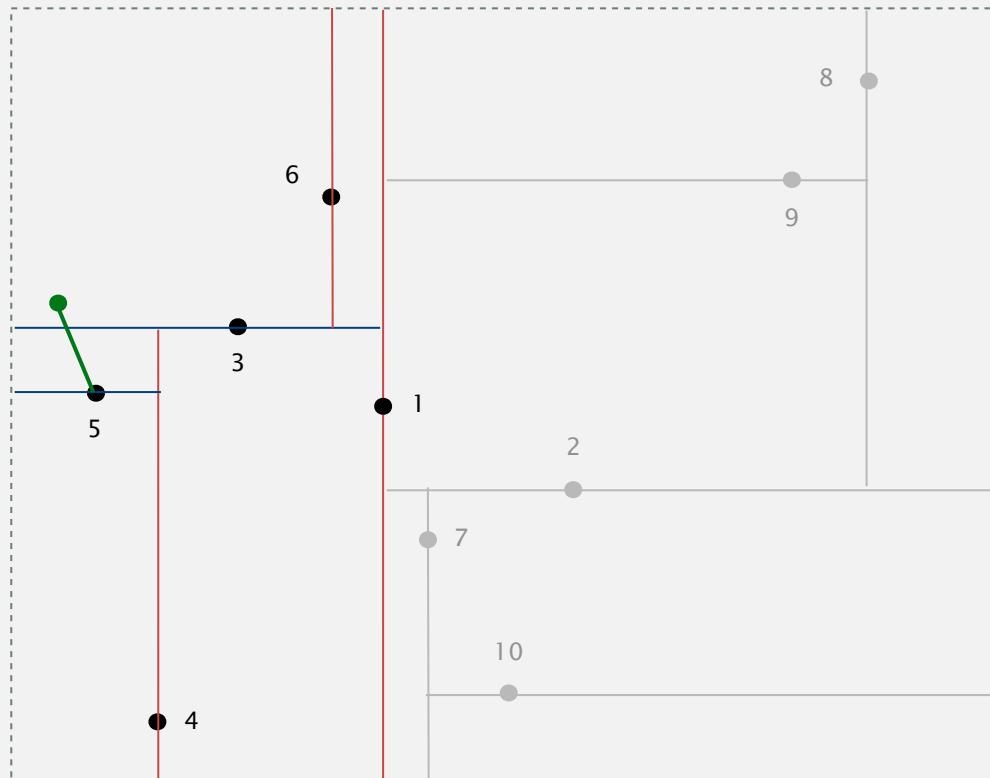
- Check distance from point in node to query point.
- Recursively search left/bottom (if it could contain a closer point).
- Recursively search right/top (if it could contain a closer point).
- Organize method so that it begins by searching for query point.



Nearest neighbor search in a 2d tree analysis

Typical case. $\log N$.

Worst case (even if tree is balanced). N .



Flocking birds

Q. What "natural algorithm" do starlings, migrating geese, starlings, cranes, bait balls of fish, and flashing fireflies use to flock?

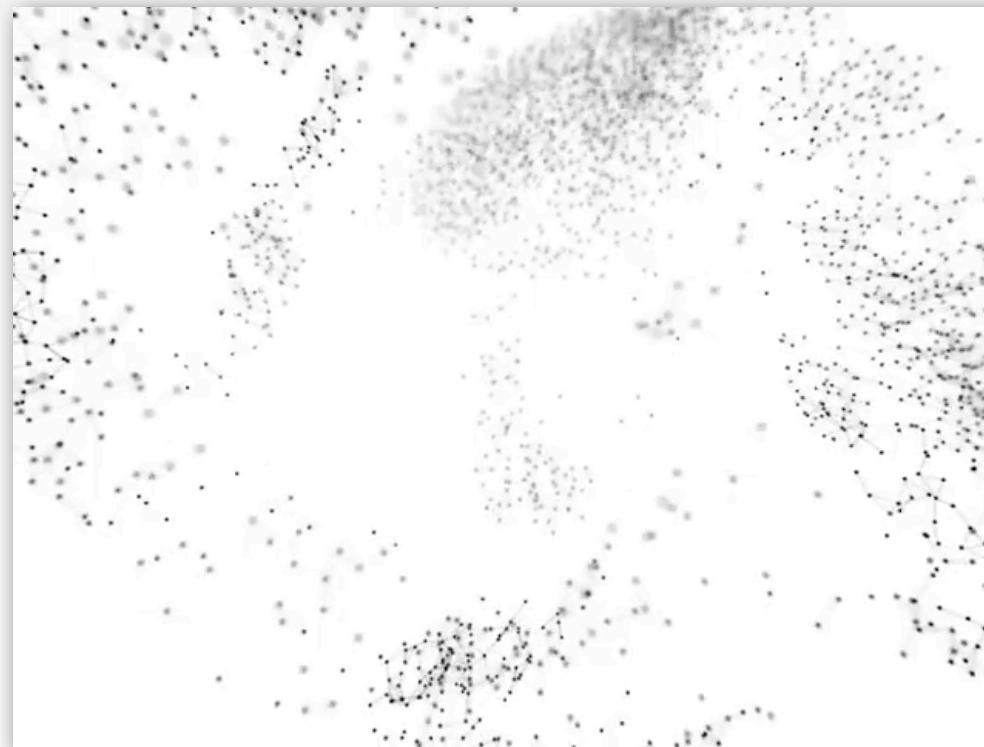


<http://www.youtube.com/watch?v=XH-groCeKbE>

Flocking boids [Craig Reynolds, 1986]

Boids. Three simple rules lead to complex emergent flocking behavior:

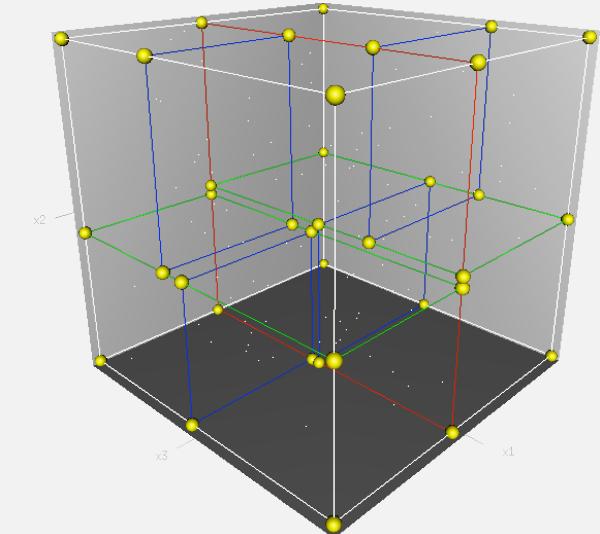
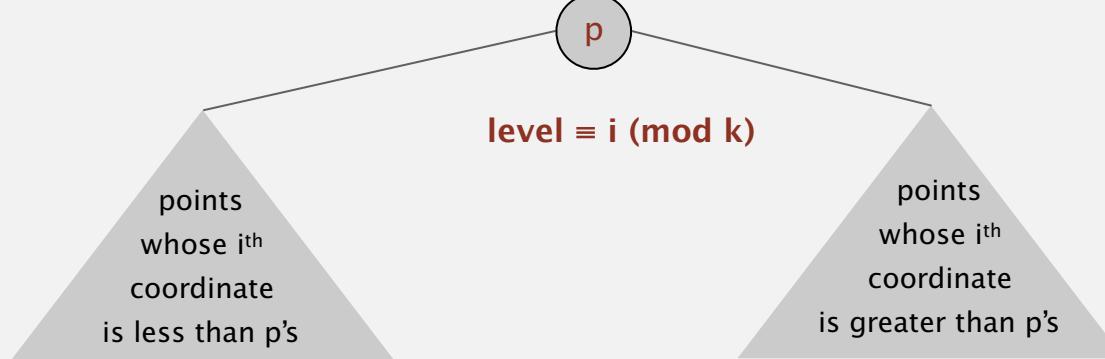
- Collision avoidance: point away from **k nearest** boids.
- Flock centering: point towards the center of mass of **k nearest** boids.
- Velocity matching: update velocity to the average of **k nearest** boids.



Kd tree

Kd tree. Recursively partition k -dimensional space into 2 halfspaces.

Implementation. BST, but cycle through dimensions ala 2d trees.



Efficient, simple data structure for processing k -dimensional data.

- Widely used.
- Adapts well to high-dimensional and clustered data.
- Discovered by an undergrad in an algorithms class!



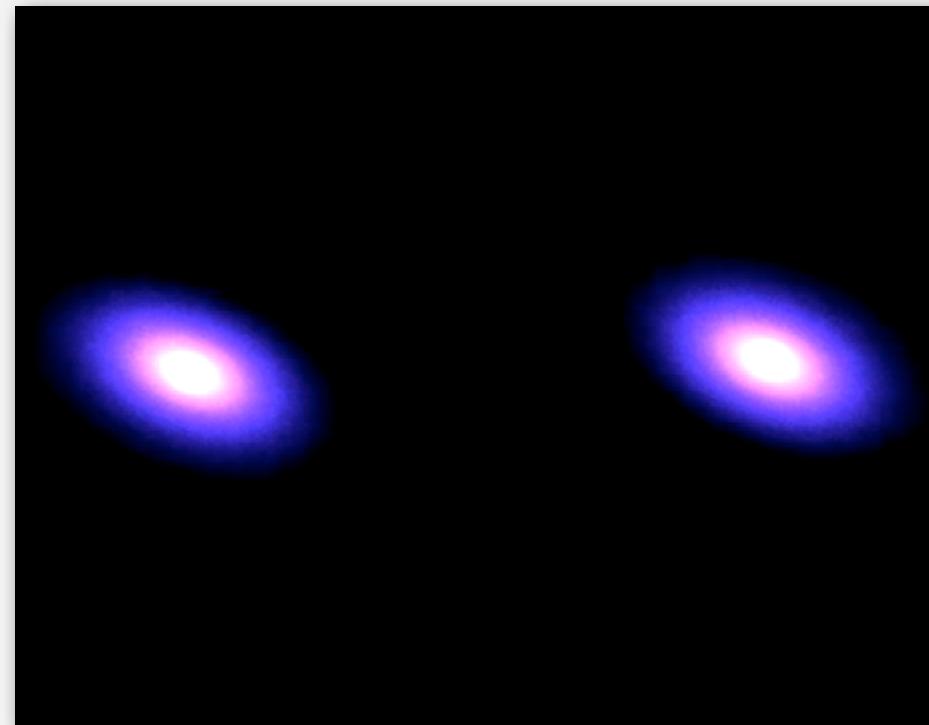
Jon Bentley

N-body simulation

Goal. Simulate the motion of N particles, mutually affected by gravity.

Brute force. For each pair of particles, compute force: $F = \frac{G m_1 m_2}{r^2}$

Running time. Time per step is N^2 .



http://www.youtube.com/watch?v=ua7YIN4eL_w

Appel's algorithm for N-body simulation

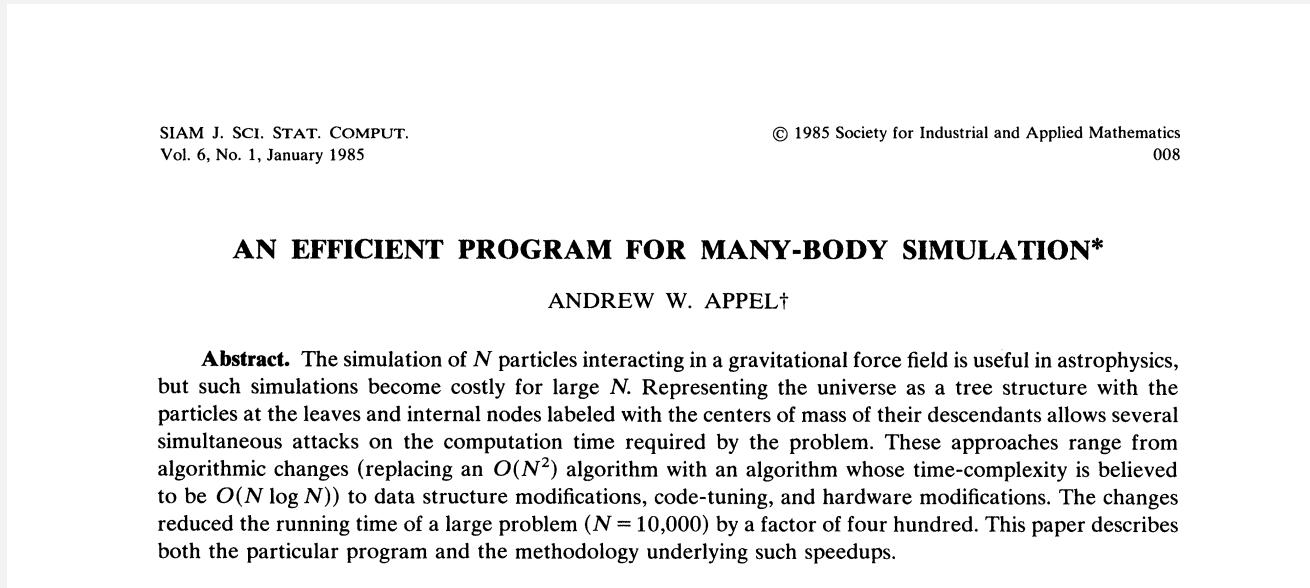
Key idea. Suppose particle is far, far away from cluster of particles.

- Treat cluster of particles as a single aggregate particle.
- Compute force between particle and **center of mass** of aggregate.



Appel's algorithm for N-body simulation

- Build 3d-tree with N particles as nodes.
- Store center-of-mass of subtree in each node.
- To compute total force acting on a particle, traverse tree, but stop as soon as distance from particle to subdivision is sufficiently large.



Impact. Running time per step is $N \log N \Rightarrow$ enables new research.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

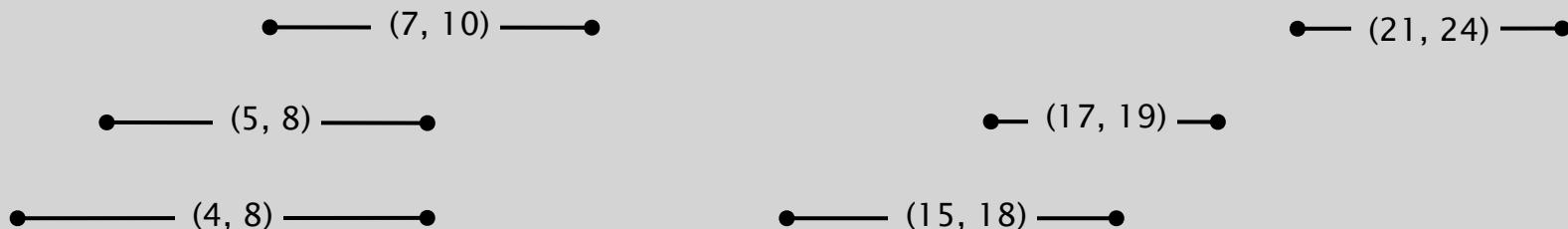
1d interval search

1d interval search. Data structure to hold set of (overlapping) intervals.

- Insert an interval (lo, hi).
- Search for an interval (lo, hi).
- Delete an interval (lo, hi).
- **Interval intersection query:** given an interval (lo, hi), find all intervals in data structure overlapping (lo, hi).

Q. Which intervals intersect (9, 14)?

A. Which intervals intersect (7, 10) and (15, 18).



1d interval search API

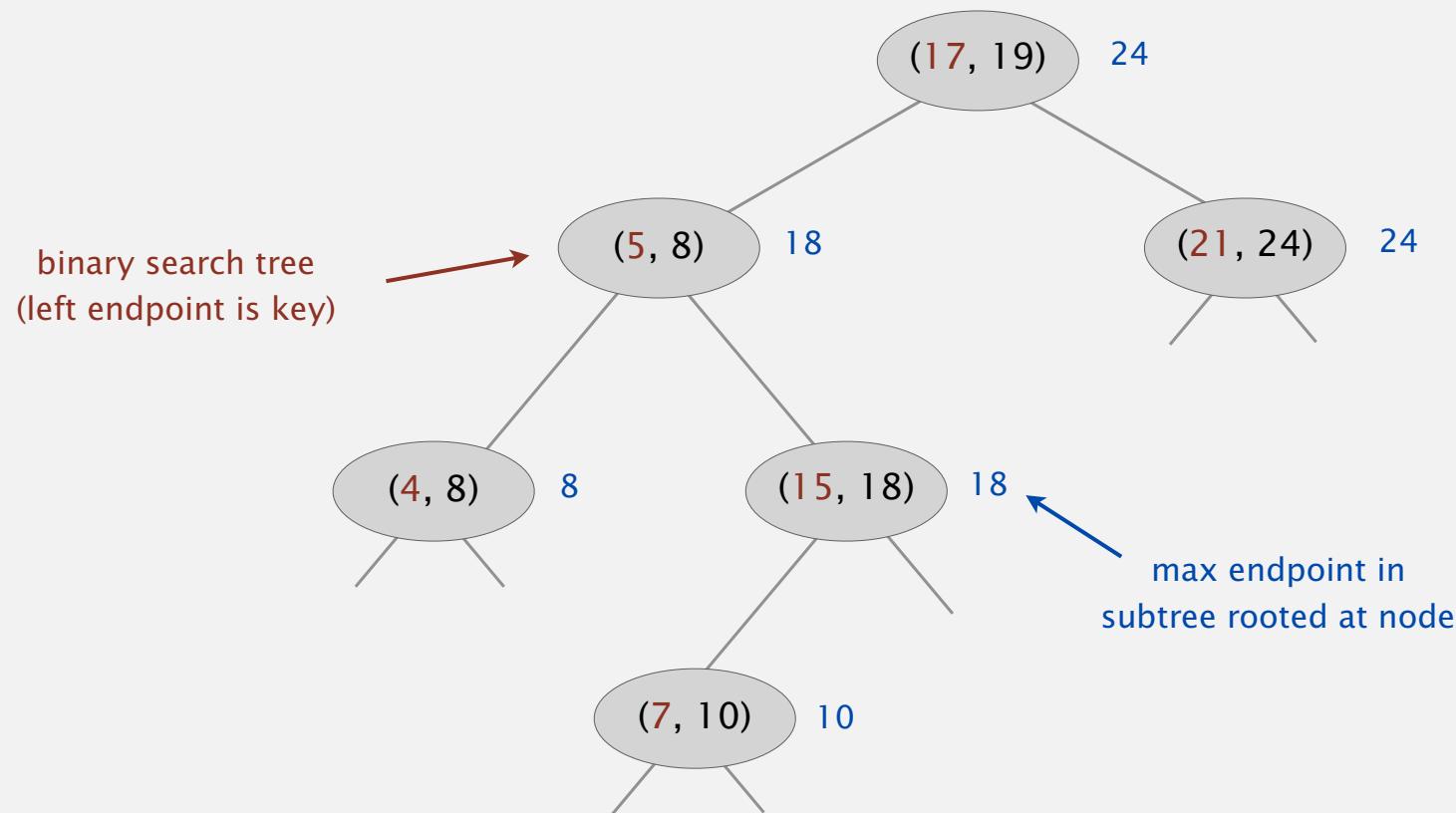
public class IntervalST<Key extends Comparable<Key>, Value>	
IntervalST()	<i>create interval search tree</i>
void put(Key lo, Key hi, Value val)	<i>put interval-value pair into ST</i>
Value get(Key lo, Key hi)	<i>value paired with given interval</i>
void delete(Key lo, Key hi)	<i>delete the given interval</i>
Iterable<Value> intersects(Key lo, Key hi)	<i>all intervals that intersect the given interval</i>

Nondegeneracy assumption. No two intervals have the same left endpoint.

Interval search trees

Create BST, where each node stores an interval (lo, hi).

- Use left endpoint as BST **key**.
- Store **max endpoint** in subtree rooted at node.



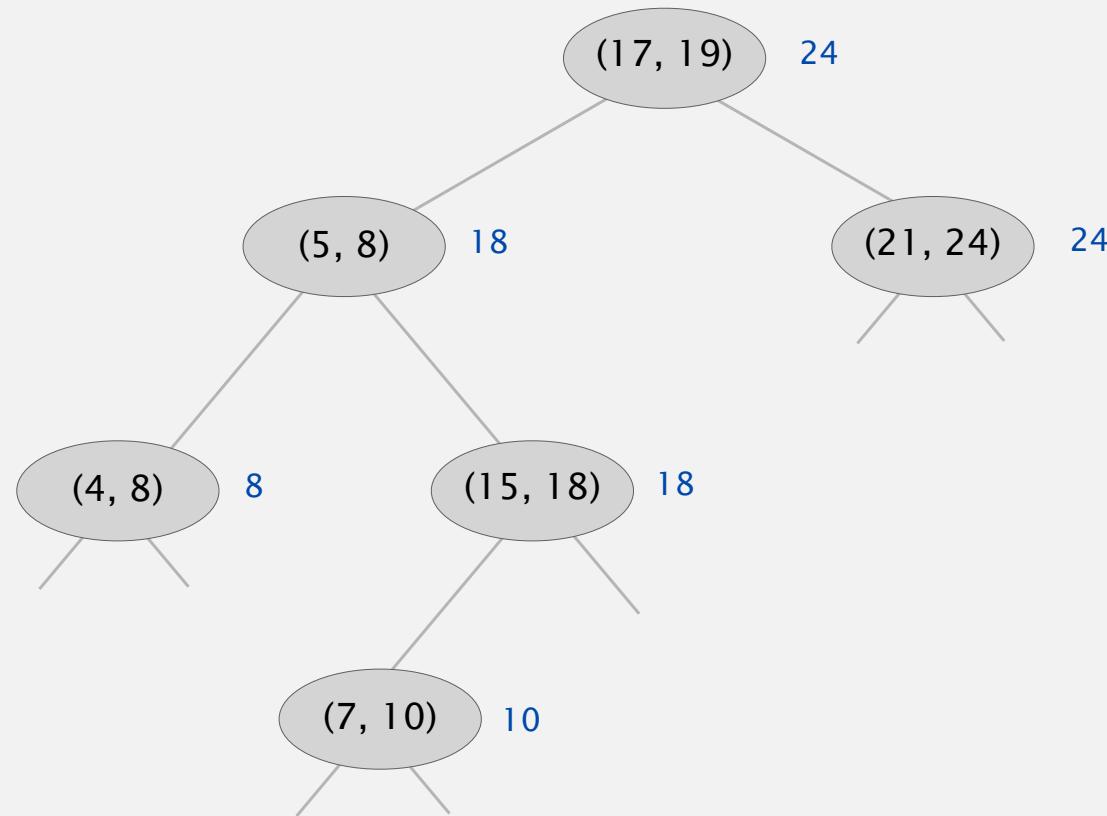
Interval search tree demo

To insert an interval (lo, hi) :

- Insert into BST, using lo as the key.
- Update max in each node on search path.



insert interval (16, 22)

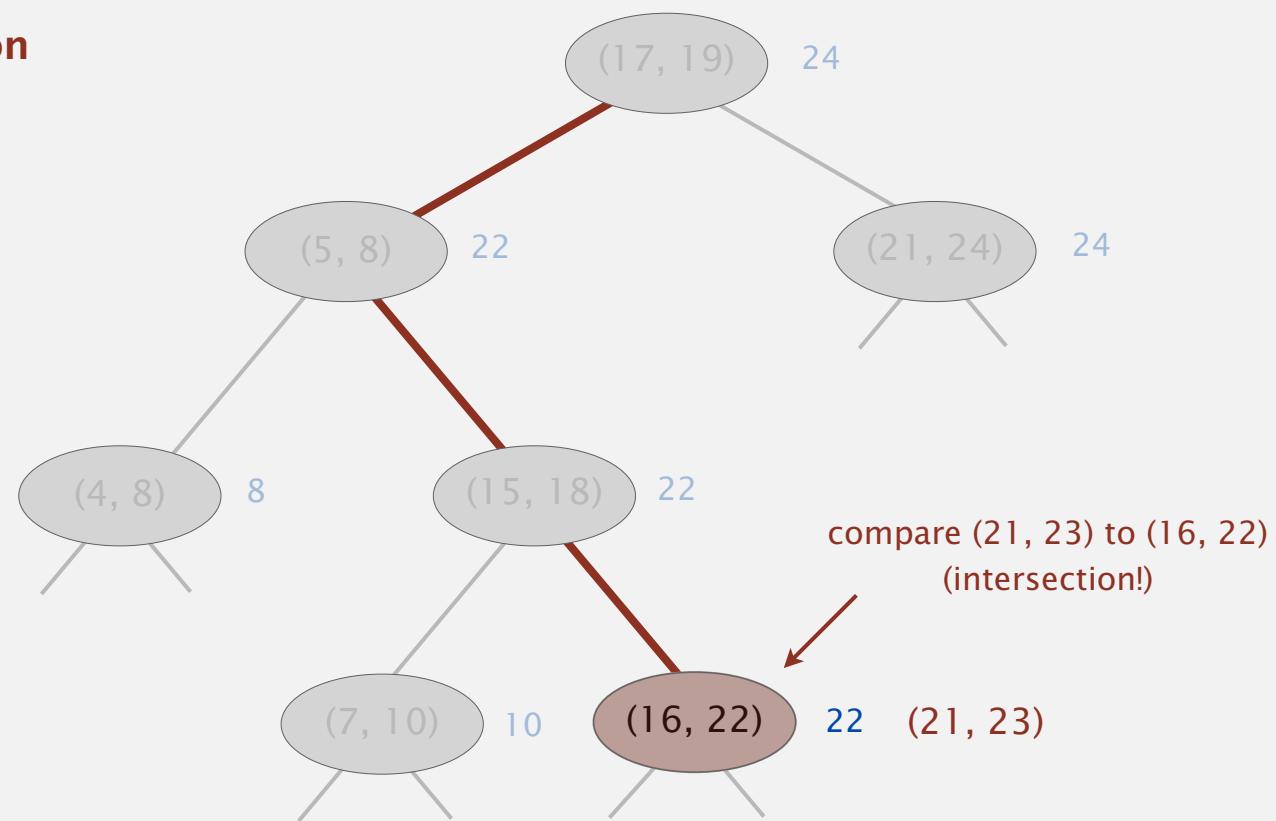


Interval search tree demo

To search for any one interval that intersects query interval (lo, hi):

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo , go right.
- Else go left.

interval intersection
search for (21, 23)



Search for an intersecting interval implementation

To search for any one interval that intersects query interval (lo, hi):

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo , go right.
- Else go left.

```
Node x = root;
while (x != null)
{
    if      (x.interval.intersects(lo, hi)) return x.interval;
    else if (x.left == null)                  x = x.right;
    else if (x.left.max < lo)                x = x.right;
    else                                      x = x.left;
}
return null;
```

Search for an intersecting interval analysis

To search for any one interval that intersects query interval (lo, hi):

- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo , go right.
- Else go left.

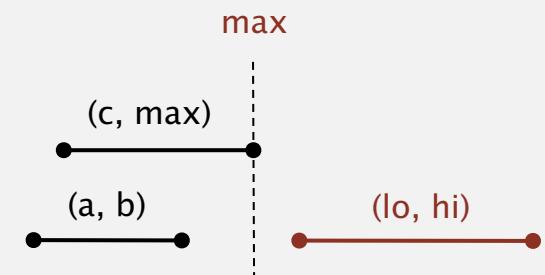
Case 1. If search goes **right**, then no intersection in left.

Pf.

- Left subtree is empty \Rightarrow trivial.
- Max endpoint max in left subtree is less than $lo \Rightarrow$ for any interval (a, b) in left subtree of x , we have $b \leq max < lo$.

definition of max

reason for going right



left subtree of x

right subtree of x

Search for an intersecting interval analysis

To search for any one interval that intersects query interval (lo, hi) :

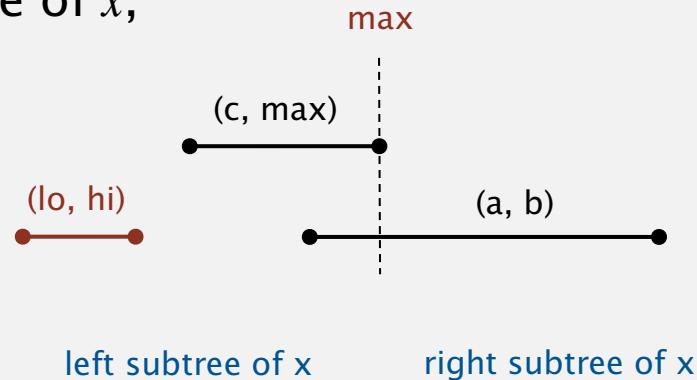
- If interval in node intersects query interval, return it.
- Else if left subtree is null, go right.
- Else if max endpoint in left subtree is less than lo , go right.
- Else go left.

Case 2. If search goes **left**, then there is either an intersection in left subtree or no intersections in either.

Pf. Suppose no intersection in left.

- Since went left, we have $lo \leq max$.
- Then for any interval (a, b) in right subtree of x ,
 $hi < c \leq a \Rightarrow$ no intersection in right.

no intersections in left subtree intervals sorted by left endpoint



Interval search tree: analysis

Implementation. Use a red-black BST to guarantee performance.

easy to maintain auxiliary information
using $\log N$ extra work per op

operation	brute	interval search tree	best in theory
insert interval	1	$\log N$	$\log N$
find interval	N	$\log N$	$\log N$
delete interval	N	$\log N$	$\log N$
find any one interval that intersects (lo, hi)	N	$\log N$	$\log N$
find all intervals that intersects (lo, hi)	N	$R \log N$	$R + \log N$

order of growth of running time for N intervals

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

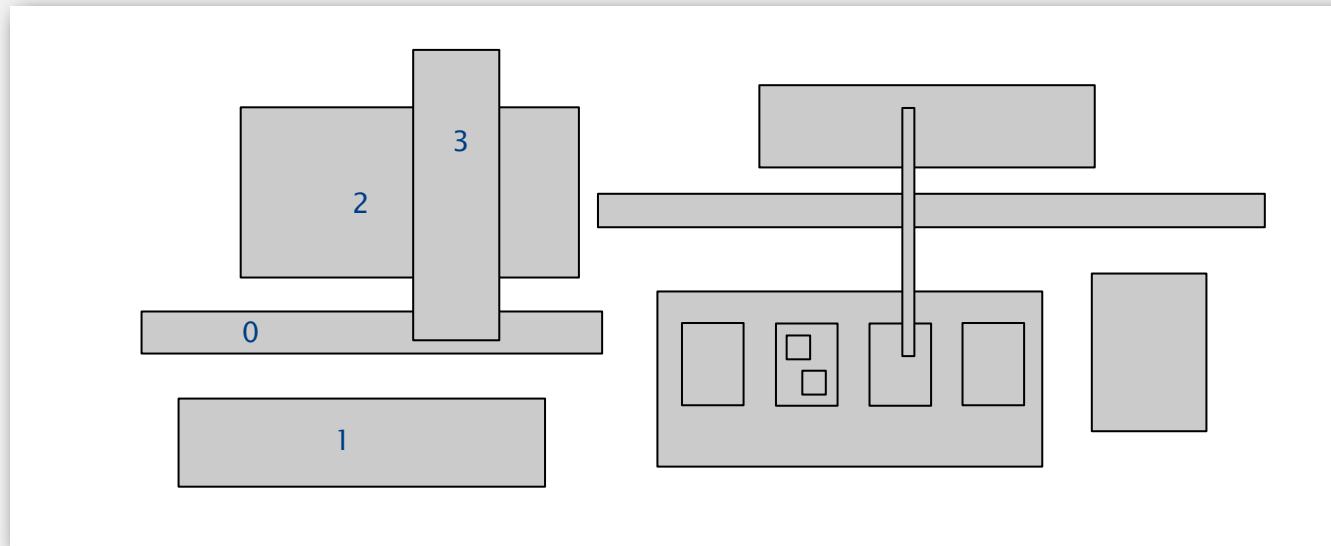
GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Orthogonal rectangle intersection

Goal. Find all intersections among a set of N orthogonal rectangles.

Quadratic algorithm. Check all pairs of rectangles for intersection.



Non-degeneracy assumption. All x - and y -coordinates are distinct.

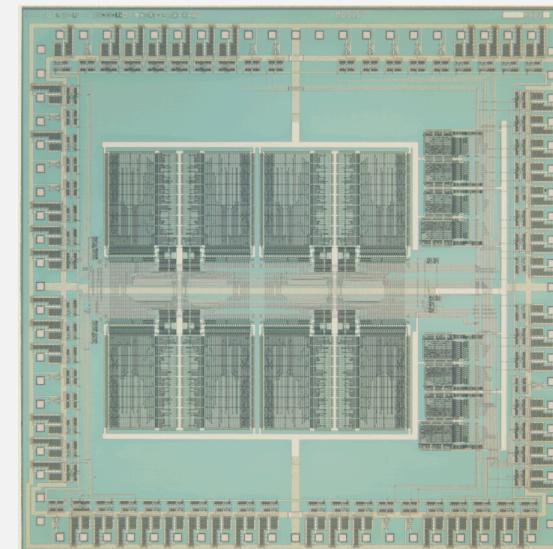
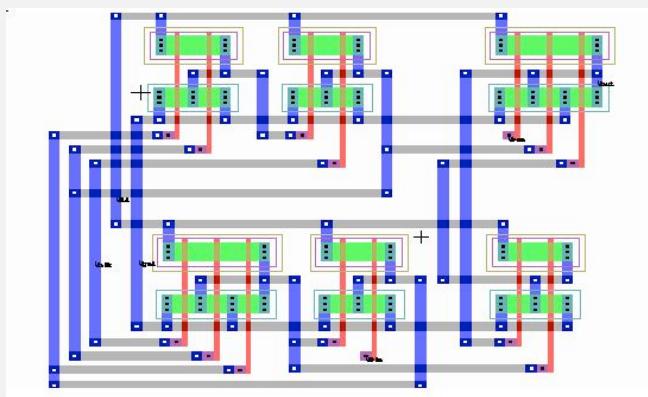
Microprocessors and geometry

Early 1970s. microprocessor design became a **geometric** problem.

- Very Large Scale Integration (VLSI).
- Computer-Aided Design (CAD).

Design-rule checking.

- Certain wires cannot intersect.
- Certain spacing needed between different types of wires.
- Debugging = orthogonal rectangle intersection search.



Algorithms and Moore's law

"Moore's law." Processing power doubles every 18 months.

- $197x$: check N rectangles.
- $197(x+1.5)$: check $2N$ rectangles on a $2x$ -faster computer.



Gordon Moore

Bootstrapping. We get to use the faster computer for bigger circuits.

But bootstrapping is not enough if using a quadratic algorithm:

- $197x$: takes M days.
- $197(x+1.5)$: takes $(4M)/2 = 2M$ days. (!)

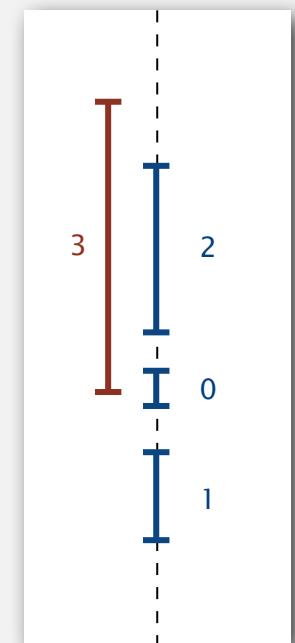
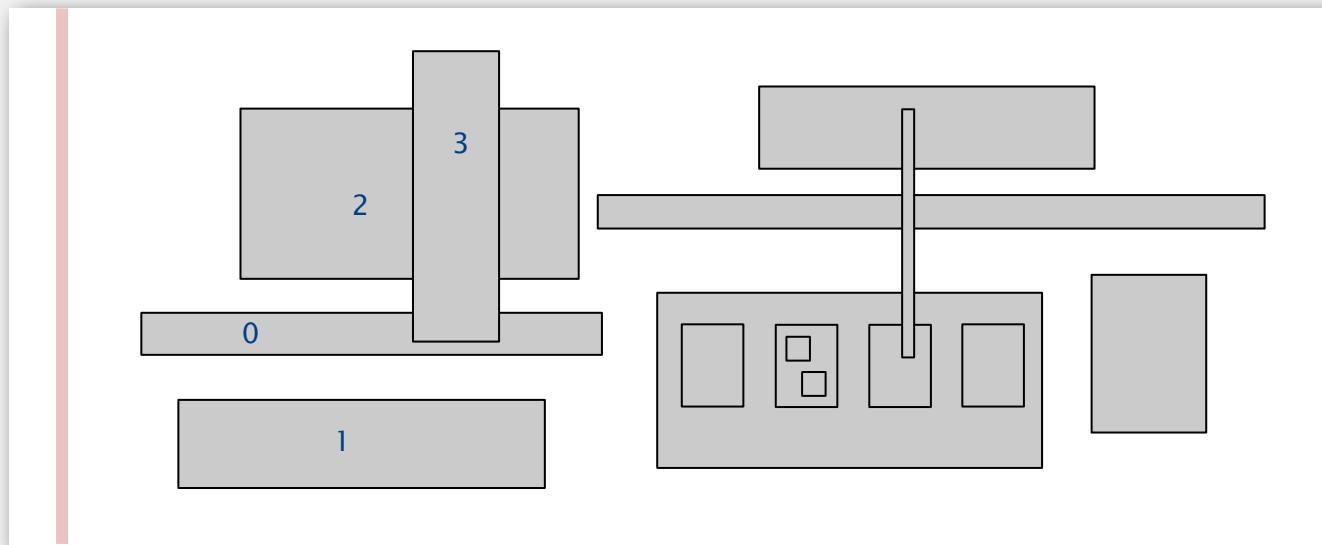


Bottom line. Linearithmic algorithm is **necessary** to sustain Moore's Law.

Orthogonal rectangle intersection: sweep-line algorithm

Sweep vertical line from left to right.

- x -coordinates of left and right endpoints define events.
- Maintain set of rectangles that intersect the sweep line in an interval search tree (using y -intervals of rectangle).
- Left endpoint: interval search for y -interval of rectangle; insert y -interval.
- Right endpoint: remove y -interval.



y-coordinates

Orthogonal rectangle intersection: sweep-line analysis

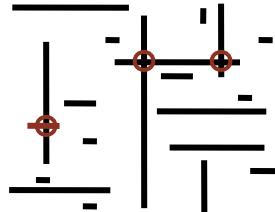
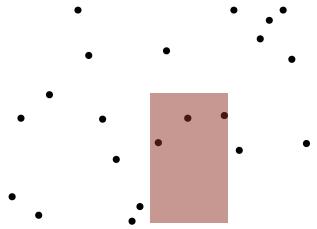
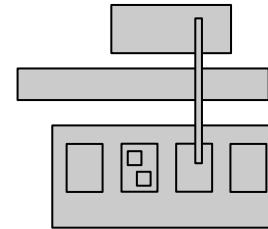
Proposition. Sweep line algorithm takes time proportional to $N \log N + R \log N$ to find R intersections among a set of N rectangles.

Pf.

- Put x -coordinates on a PQ (or sort). $\leftarrow N \log N$
- Insert y -intervals into ST. $\leftarrow N \log N$
- Delete y -intervals from ST. $\leftarrow N \log N$
- Interval searches for y -intervals. $\leftarrow N \log N + R \log N$

Bottom line. Sweep line reduces 2d orthogonal rectangle intersection search to 1d interval search.

Geometric applications of BSTs

problem	example	solution
1d range search	BST
2d orthogonal line segment intersection		sweep line reduces to 1d range search
kd range search		kd tree
1d interval search		interval search tree
2d orthogonal rectangle intersection		sweep line reduces to 1d interval search

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*

Algorithms

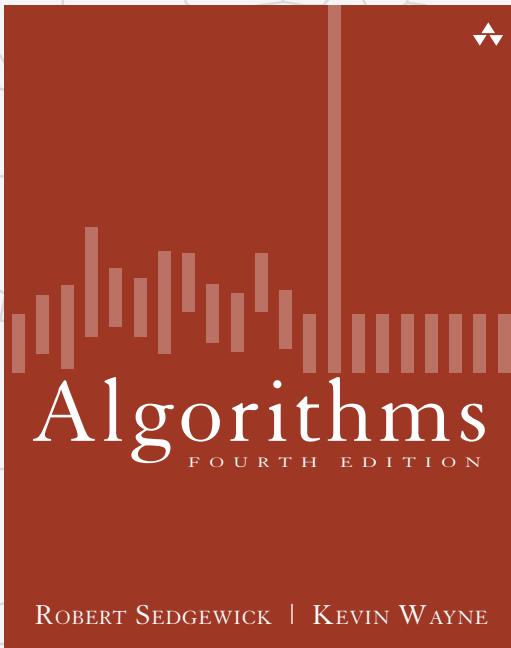
ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

GEOMETRIC APPLICATIONS OF BSTs

- ▶ *1d range search*
- ▶ *line segment intersection*
- ▶ *kd trees*
- ▶ *interval search trees*
- ▶ *rectangle intersection*



<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

String processing

String. Sequence of characters.

Important fundamental abstraction.

- Information processing.
- Genomic sequences.
- Communication systems (e.g., email).
- Programming systems (e.g., Java programs).
- ...

“ The digital information that underlies biochemistry, cell biology, and development can be represented by a simple string of G's, A's, T's and C's. This string is the root data structure of an organism's biology. ” — M. V. Olson

The char data type

C char data type. Typically an 8-bit integer.

- Supports 7-bit ASCII.
- Can represent only 256 characters.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	\$	%	&	‘	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Hexadecimal to ASCII conversion table

A á ð ö

U+0041 U+00E1 U+2202 U+1D50A

Unicode characters

Java char data type. A 16-bit unsigned integer.

- Supports original 16-bit Unicode.
- Supports 21-bit Unicode 3.0 (awkwardly).

I (heart) Unicode



The String data type

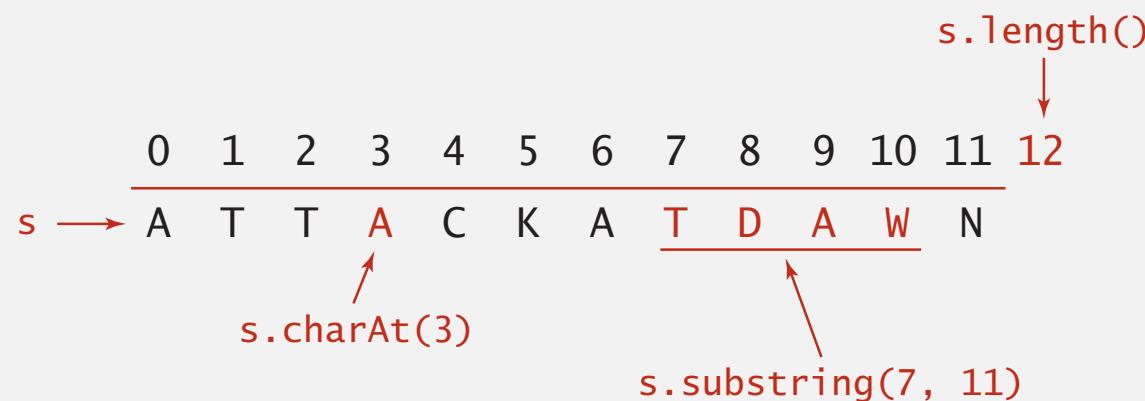
String data type in Java. Sequence of characters (immutable).

Length. Number of characters.

Indexing. Get the i^{th} character.

Substring extraction. Get a contiguous subsequence of characters.

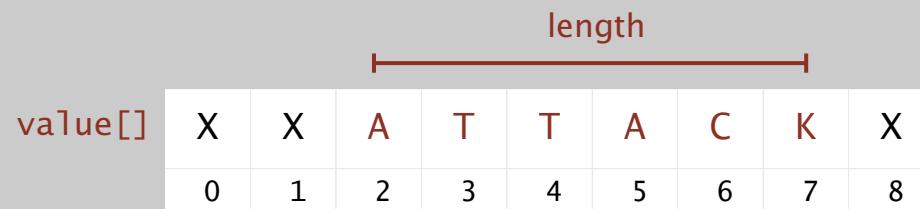
String concatenation. Append one character to end of another string.



The String data type: Java implementation

```
public final class String implements Comparable<String>
{
    private char[] value; // characters
    private int offset; // index of first char in array
    private int length; // length of string
    private int hash; // cache of hashCode()
```

```
public int length()
{ return length; }
```



```
public char charAt(int i)
{ return value[i + offset]; }
```

```
private String(int offset, int length, char[] value)
```

```
{  
    this.offset = offset;  
    this.length = length;  
    this.value = value;  
}
```

copy of reference to
original char array

```
public String substring(int from, int to)
{ return new String(offset + from, to - from, value); }
```

```
...
```

The String data type: performance

String data type (in Java). Sequence of characters (immutable).

Underlying implementation. Immutable char[] array, offset, and length.

String		
operation	guarantee	extra space
length()	1	1
charAt()	1	1
substring()	1	1
concat()	N	N

Memory. $40 + 2N$ bytes for a virgin String of length N .

can use byte[] or char[] instead of String to save space
(but lose convenience of String data type)

The StringBuilder data type

StringBuilder data type. Sequence of characters (mutable).

Underlying implementation. Resizing char[] array and length.

	String		StringBuilder	
operation	guarantee	extra space	guarantee	extra space
length()	1	1	1	1
charAt()	1	1	1	1
substring()	1	1	N	N
concat()	N	N	1 *	1 *

* amortized

Remark. StringBuffer data type is similar, but thread safe (and slower).

String vs. StringBuilder

Q. How to efficiently reverse a string?

A.

```
public static String reverse(String s)
{
    String rev = "";
    for (int i = s.length() - 1; i >= 0; i--)
        rev += s.charAt(i);
    return rev;
}
```

← quadratic time

B.

```
public static String reverse(String s)
{
    StringBuilder rev = new StringBuilder();
    for (int i = s.length() - 1; i >= 0; i--)
        rev.append(s.charAt(i));
    return rev.toString();
}
```

← linear time

String challenge: array of suffixes

Q. How to efficiently form array of suffixes?

input string

a	a	c	a	a	g	t	t	t	a	c	a	a	g	c
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

suffixes

0	a	a	c	a	a	g	t	t	t	a	c	a	a	g	c
1	a	c	a	a	g	t	t	t	a	c	a	a	g	c	
2	c	a	a	g	t	t	t	a	c	a	a	g	c		
3	a	a	g	t	t	t	a	c	a	a	g	c			
4	a	g	t	t	t	a	c	a	a	g	c				
5	g	t	t	t	a	c	a	a	g	c					
6	t	t	t	a	c	a	a	g	c						
7	t	t	a	c	a	a	g	c							
8	t	a	c	a	a	g	c								
9	a	c	a	a	g	c									
10	c	a	a	g	c										
11	a	a	g	c											
12	a	g	c												
13	g	c													
14	c														

String vs. StringBuilder

Q. How to efficiently form array of suffixes?

A.

```
public static String[] suffixes(String s)
{
    int N = s.length();
    String[] suffixes = new String[N];
    for (int i = 0; i < N; i++)
        suffixes[i] = s.substring(i, N);
    return suffixes;
}
```

linear time and
linear space

B.

```
public static String[] suffixes(String s)
{
    int N = s.length();
    StringBuilder sb = new StringBuilder(s);
    String[] suffixes = new String[N];
    for (int i = 0; i < N; i++)
        suffixes[i] = sb.substring(i, N);
    return suffixes;
}
```

quadratic time and
quadratic space

Longest common prefix

Q. How long to compute length of longest common prefix?

p	r	e	f	e	t	c	h
0	1	2	3	4	5	6	7
p	r	e	f	i	x		

```
public static int lcp(String s, String t)
{
    int N = Math.min(s.length(), t.length());
    for (int i = 0; i < N; i++)
        if (s.charAt(i) != t.charAt(i))
            return i;
    return N;
}
```

linear time (worst case)
sublinear time (typical case)

Running time. Proportional to length D of longest common prefix.

Remark. Also can compute compareTo() in sublinear time.

Alphabets

Digital key. Sequence of digits over fixed alphabet.

Radix. Number of digits R in alphabet.

name	R()	IgR()	characters
BINARY	2	1	01
OCTAL	8	3	01234567
DECIMAL	10	4	0123456789
HEXADECIMAL	16	4	0123456789ABCDEF
DNA	4	2	ACTG
LOWERCASE	26	5	abcdefghijklmnopqrstuvwxyz
UPPERCASE	26	5	ABCDEFGHIJKLMNOPQRSTUVWXYZ
PROTEIN	20	5	ACDEFGHIJKLMNOPQRSTUVWXYZ
BASE64	64	6	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
ASCII	128	7	<i>ASCII characters</i>
EXTENDED_ASCII	256	8	<i>extended ASCII characters</i>
UNICODE16	65536	16	<i>Unicode characters</i>

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Review: summary of the performance of sorting algorithms

Frequency of operations = key compares.

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	yes	compareTo()
mergesort	$N \lg N$	$N \lg N$	N	yes	compareTo()
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$	no	compareTo()
heapsort	$2 N \lg N$	$2 N \lg N$	1	no	compareTo()

* probabilistic

Lower bound. $\sim N \lg N$ compares required by any compare-based algorithm.

Q. Can we do better (despite the lower bound)?

A. Yes, if we don't depend on key compares.

Key-indexed counting: assumptions about keys

Assumption. Keys are integers between 0 and $R - 1$.

Implication. Can use key as an array index.

Applications.

- Sort string by first letter.
- Sort class roster by section.
- Sort phone numbers by area code.
- Subroutine in a sorting algorithm. [stay tuned]

Remark. Keys may have associated data \Rightarrow
can't just count up number of keys of each value.

	input		sorted result (by section)	
	name	section		
Anderson	2		Harris	1
Brown	3		Martin	1
Davis	3		Moore	1
Garcia	4		Anderson	2
Harris	1		Martinez	2
Jackson	3		Miller	2
Johnson	4		Robinson	2
Jones	3		White	2
Martin	1		Brown	3
Martinez	2		Davis	3
Miller	2		Jackson	3
Moore	1		Jones	3
Robinson	2		Taylor	3
Smith	4		Williams	3
Taylor	3		Garcia	4
Thomas	4		Johnson	4
Thompson	4		Smith	4
White	2		Thomas	4
Williams	3		Thompson	4
Wilson	4		Wilson	4

*↑
keys are
small integers*

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.



$R = 6$

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	
0	d	
1	a	use a for 0
2	c	b for 1
3	f	c for 2
4	f	d for 3
5	b	e for 4
6	d	f for 5
7	b	
8	f	
9	b	
10	e	
11	a	

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

count frequencies → for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	offset by 1 [stay tuned]
0	d	
1	a	
2	c	
3	f	
4	f	
5	b	
6	d	
7	b	
8	f	
9	b	
10	e	
11	a	

↓

r count[r]

0 2 3 1 2 1 3

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

compute
cumulates → for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	r	count[r]
0	d		
1	a		
2	c		
3	f	a	0
4	f	b	2
5	b	c	5
6	d	d	6
7	b	e	8
8	f	f	9
9	b		
10	e		
11	a		

6 keys < d, 8 keys < e
so d's go in a[6] and a[7]

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

move items → for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

i	a[i]	r	count[r]	i	aux[i]
0	d			0	a
1	a			1	a
2	c			2	b
3	f	a	2	3	b
4	f	b	5	4	b
5	b	c	6	5	c
6	d	d	8	6	d
7	b	e	9	7	d
8	f	f	12	8	e
9	b	-	12	9	f
10	e			10	f
11	a			11	f

Key-indexed counting demo

Goal. Sort an array $a[]$ of N integers between 0 and $R - 1$.

- Count frequencies of each letter using key as index.
- Compute frequency cumulates which specify destinations.
- Access cumulates using key as index to move items.
- Copy back into original array.

```
int N = a.length;
int[] count = new int[R+1];

for (int i = 0; i < N; i++)
    count[a[i]+1]++;

for (int r = 0; r < R; r++)
    count[r+1] += count[r];

for (int i = 0; i < N; i++)
    aux[count[a[i]]++] = a[i];

for (int i = 0; i < N; i++)
    a[i] = aux[i];
```

copy
back

i	a[i]	r	count[r]	i	aux[i]
0	a			0	a
1	a			1	a
2	b			2	b
3	b	a	2	3	b
4	b	b	5	4	b
5	c	c	6	5	c
6	d	d	8	6	d
7	d	e	9	7	d
8	e	f	12	8	e
9	f	-	12	9	f
10	f			10	f
11	f			11	f

Key-indexed counting: analysis

Proposition. Key-indexed counting uses $\sim 11N + 4R$ array accesses to sort N items whose keys are integers between 0 and $R - 1$.

Proposition. Key-indexed counting uses extra space proportional to $N + R$.

Stable? ✓

a[0]	Anderson	2	Harris	1	aux[0]
a[1]	Brown	3	Martin	1	aux[1]
a[2]	Davis	3	Moore	1	aux[2]
a[3]	Garcia	4	Anderson	2	aux[3]
a[4]	Harris	1	Martinez	2	aux[4]
a[5]	Jackson	3	Miller	2	aux[5]
a[6]	Johnson	4	Robinson	2	aux[6]
a[7]	Jones	3	White	2	aux[7]
a[8]	Martin	1	Brown	3	aux[8]
a[9]	Martinez	2	Davis	3	aux[9]
a[10]	Miller	2	Jackson	3	aux[10]
a[11]	Moore	1	Jones	3	aux[11]
a[12]	Robinson	2	Taylor	3	aux[12]
a[13]	Smith	4	Williams	3	aux[13]
a[14]	Taylor	3	Garcia	4	aux[14]
a[15]	Thomas	4	Johnson	4	aux[15]
a[16]	Thompson	4	Smith	4	aux[16]
a[17]	White	2	Thomas	4	aux[17]
a[18]	Williams	3	Thompson	4	aux[18]
a[19]	Wilson	4	Wilson	4	aux[19]

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

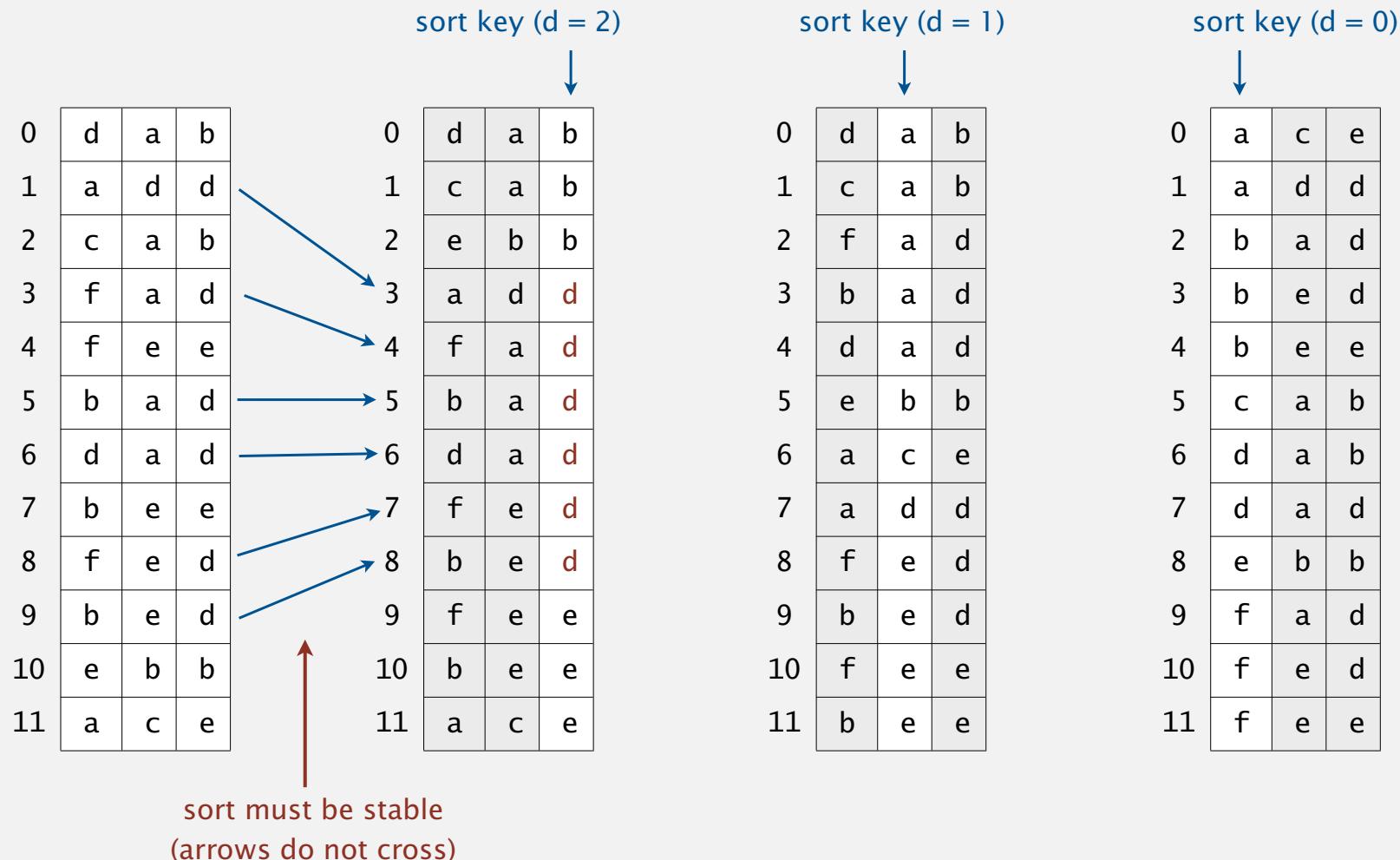
5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Least-significant-digit-first string sort

LSD string (radix) sort.

- Consider characters from right to left.
- Stably sort using d^{th} character as the key (using key-indexed counting).



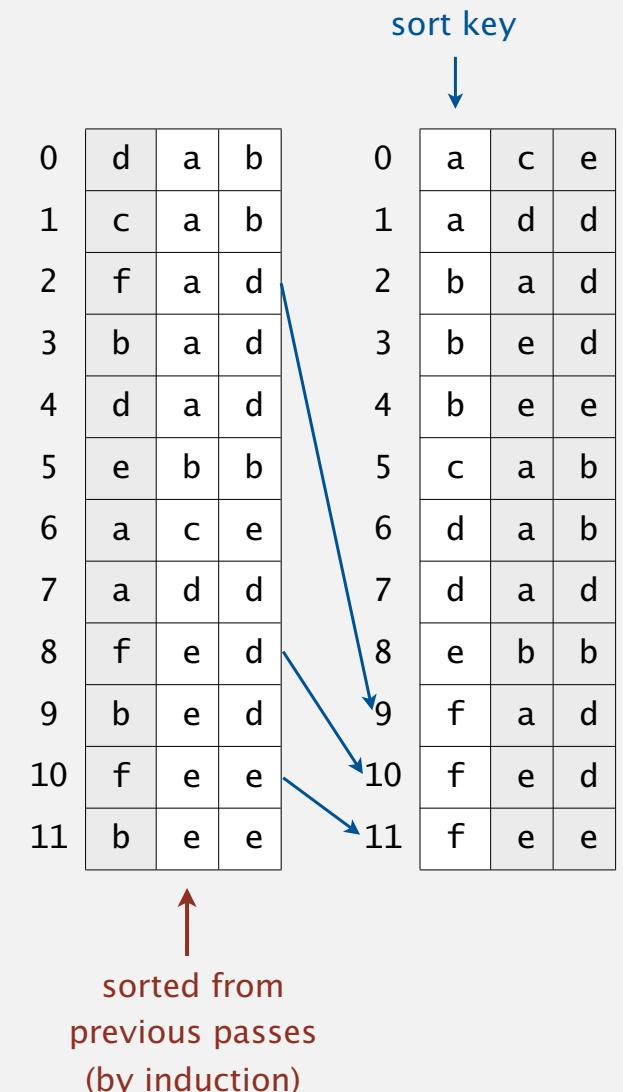
LSD string sort: correctness proof

Proposition. LSD sorts fixed-length strings in ascending order.

Pf. [by induction on i]

After pass i , strings are sorted by last i characters.

- If two strings differ on sort key, key-indexed sort puts them in proper relative order.
- If two strings agree on sort key, stability keeps them in proper relative order.



Proposition. LSD sort is stable.

LSD string sort: Java implementation

```
public class LSD
{
    public static void sort(String[] a, int W) ← fixed-length W strings
    {
        int R = 256; ← radix R
        int N = a.length;
        String[] aux = new String[N];

        for (int d = W-1; d >= 0; d--) ← do key-indexed counting
        {
            int[] count = new int[R+1];
            for (int i = 0; i < N; i++)
                count[a[i].charAt(d) + 1]++;
            for (int r = 0; r < R; r++)
                count[r+1] += count[r]; ← key-indexed counting
            for (int i = 0; i < N; i++)
                aux[count[a[i].charAt(d)]++] = a[i];
            for (int i = 0; i < N; i++)
                a[i] = aux[i];
        }
    }
}
```

Summary of the performance of sorting algorithms

Frequency of operations.

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	yes	compareTo()
mergesort	$N \lg N$	$N \lg N$	N	yes	compareTo()
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$	no	compareTo()
heapsort	$2 N \lg N$	$2 N \lg N$	1	no	compareTo()
LSD †	$2 W N$	$2 W N$	$N + R$	yes	charAt()

* probabilistic

† fixed-length W keys

Q. What if strings do not have same length?

String sorting interview question

Problem. Sort one million 32-bit integers.

Ex. Google (or presidential) interview.

Which sorting method to use?

- Insertion sort.
- Mergesort.
- Quicksort.
- Heapsort.
- LSD string sort.



How to take a census in 1900s?

1880 Census. Took 1,500 people 7 years to manually process data.

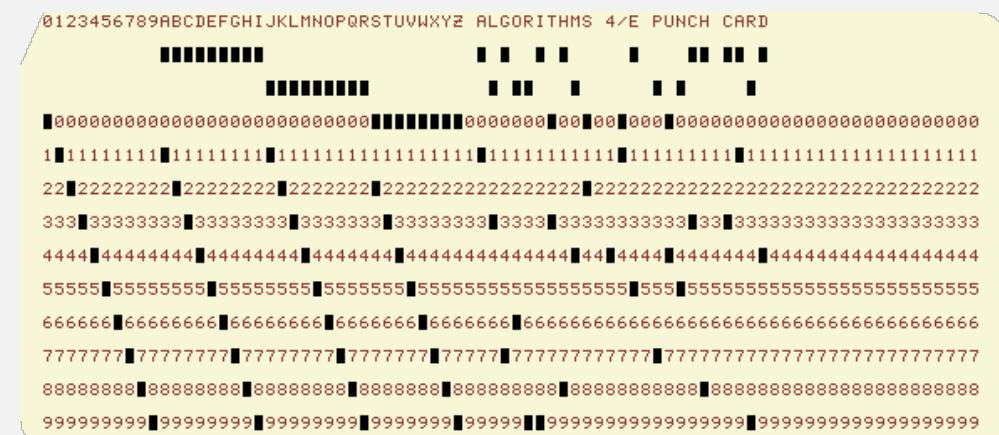


Herman Hollerith. Developed counting and sorting machine to automate.

- Use punch cards to record data (e.g., gender, age).
 - Machine sorts one column at a time (into one of 12 bins).
 - Typical question: how many women of age 20 to 30?



Hollerith tabulating machine and sorter



punch card (12 holes per column)

1890 Census. Finished months early and under budget!

How to get rich sorting in 1900s?

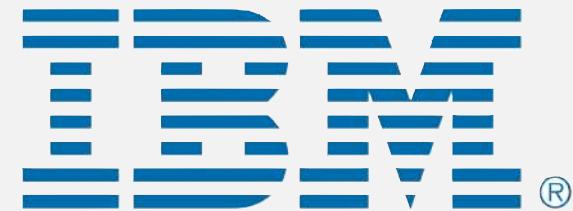
Punch cards. [1900s to 1950s]

- Also useful for accounting, inventory, and business processes.
- Primary medium for data entry, storage, and processing.

Hollerith's company later merged with 3 others to form Computing Tabulating Recording Corporation (CTR); company renamed in 1924.



IBM 80 Series Card Sorter (650 cards per minute)



LSD string sort: a moment in history (1960s)



card punch



punched cards



card reader



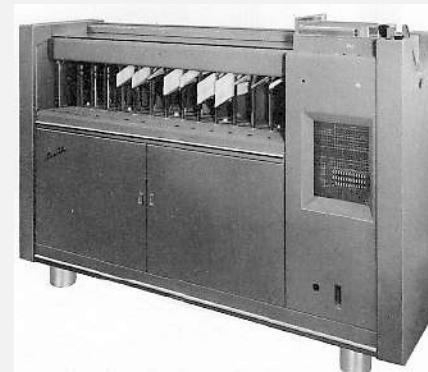
mainframe



line printer

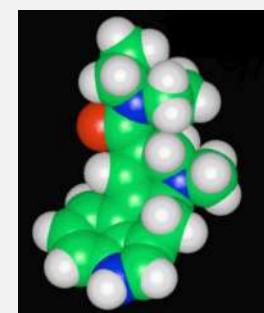
To sort a card deck

- start on right column
- put cards into hopper
- machine distributes into bins
- pick up cards (stable)
- move left one column
- continue until sorted



card sorter

not related to sorting



Lysergic Acid Diethylamide
(Lucy in the Sky with Diamonds)

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ **MSD radix sort**
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Most-significant-digit-first string sort

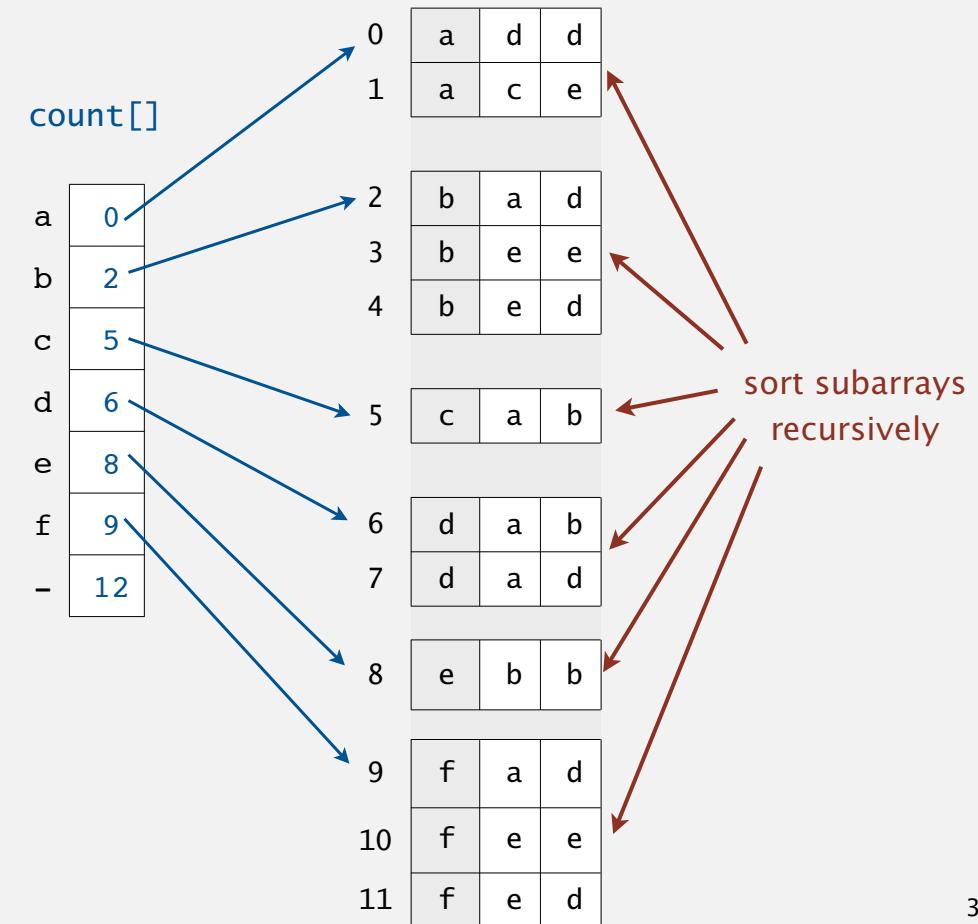
MSD string (radix) sort.

- Partition array into R pieces according to first character (use key-indexed counting).
- Recursively sort all strings that start with each character (key-indexed counts delineate subarrays to sort).

0	d	a	b
1	a	d	d
2	c	a	b
3	f	a	d
4	f	e	e
5	b	a	d
6	d	a	d
7	b	e	e
8	f	e	d
9	b	e	d
10	e	b	b
11	a	c	e

0	a	d	d
1	a	c	e
2	b	a	d
3	b	e	e
4	b	e	d
5	c	a	b
6	d	a	b
7	d	a	d
8	e	b	b
9	f	a	d
10	f	e	e
11	f	e	d

sort key



MSD string sort: example

input		d								
she	are									
sells	by									
seashells	she	seells	seashells	sea	seashells	sea	seashells	seashells	seas	sea
by	sells	seashells	sea	seashells						
the	seashells	sea	seashells							
sea	sea	sells	seells	sells						
shore	shore	seashells	seells	sells						
the	shells	she								
shells	she	shore	shore	shore	shore	shore	shore	shells	shells	shells
she	sells	shells	shells	shells	shells	shells	shells	shore	shore	shore
sells	surely	she								
are	seashells	surely								
surely	the	hi	the							
seashells	the									

need to examine every character in equal keys	end-of-string goes before any char value	output
are	are	are
by	by	by
sea	sea	sea
seashells	seashells	seashells
seashells	seashells	seashells
sells	sells	sells
sells	sells	sells
she	she	she
shells	shells	shells
she	she	she
shore	shore	shore
surely	surely	surely
the	the	the
the	the	the

Trace of recursive calls for MSD string sort (no cutoff for small subarrays, subarrays of size 0 and 1 omitted)

Variable-length strings

Treat strings as if they had an extra char at end (smaller than any char).

0	s	e	a	-1						
1	s	e	a	s	h	e	l	l	s	-1
2	s	e	l	l	s	-1				
3	s	h	e	-1						
4	s	h	e	-1						
5	s	h	e	l	l	s	-1			
6	s	h	o	r	e	-1				
7	s	u	r	e	l	y	-1			

why smaller?

she before shells

```
private static int charAt(String s, int d)
{
    if (d < s.length()) return s.charAt(d);
    else return -1;
}
```

C strings. Have extra char '\0' at end \Rightarrow no extra work needed.

MSD string sort: Java implementation

```
public static void sort(String[] a)
{
    aux = new String[a.length]; ←
    sort(a, aux, 0, a.length, 0);
}

private static void sort(String[] a, String[] aux, int lo, int hi, int d)
{
    if (hi <= lo) return; ←
    int[] count = new int[R+2];                                key-indexed counting
    for (int i = lo; i <= hi; i++)
        count[charAt(a[i], d) + 2]++;
    for (int r = 0; r < R+1; r++)
        count[r+1] += count[r];
    for (int i = lo; i <= hi; i++)
        aux[count[charAt(a[i], d) + 1]++] = a[i];
    for (int i = lo; i <= hi; i++)
        a[i] = aux[i - lo];

    for (int r = 0; r < R; r++)                                sort R subarrays recursively
        sort(a, aux, lo + count[r], lo + count[r+1] - 1, d+1);
}
```

can recycle aux[] array
but not count[] array

key-indexed counting

sort R subarrays recursively

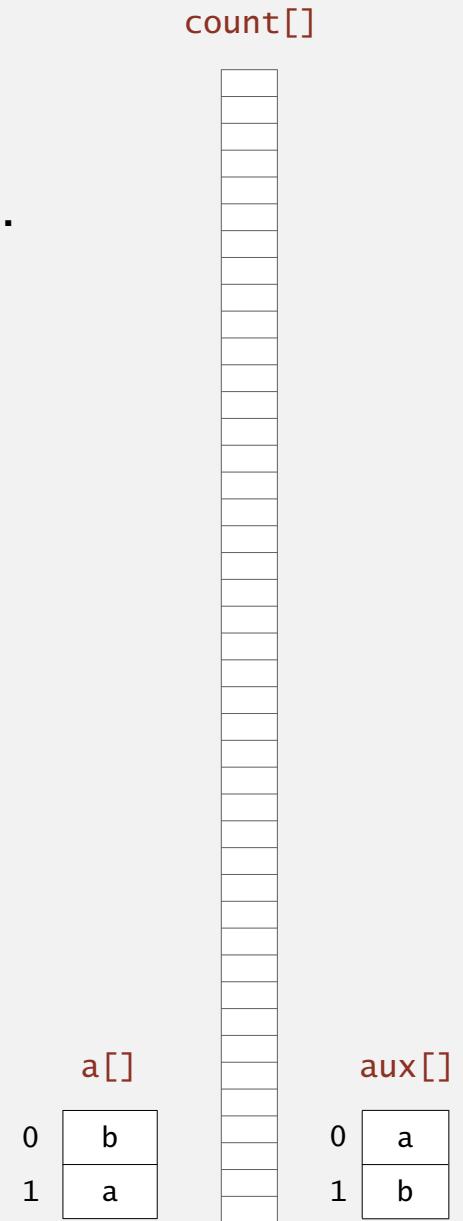
MSD string sort: potential for disastrous performance

Observation 1. Much too slow for small subarrays.

- Each function call needs its own count[] array.
- ASCII (256 counts): 100x slower than copy pass for $N=2$.
- Unicode (65,536 counts): 32,000x slower for $N=2$.

Observation 2. Huge number of small subarrays

because of recursion.



Cutoff to insertion sort

Solution. Cutoff to insertion sort for small subarrays.

- Insertion sort, but start at d^{th} character.
- Implement `less()` so that it compares starting at d^{th} character.

```
public static void sort(String[] a, int lo, int hi, int d)
{
    for (int i = lo; i <= hi; i++)
        for (int j = i; j > lo && less(a[j], a[j-1], d); j--)
            exch(a, j, j-1);
}
```

```
private static boolean less(String v, String w, int d)
{ return v.substring(d).compareTo(w.substring(d)) < 0; }
```

in Java, forming and comparing
substrings is faster than directly
comparing chars with `charAt()`

MSD string sort: performance

Number of characters examined.

- MSD examines just enough characters to sort the keys.
- Number of characters examined depends on keys.
- Can be sublinear in input size!

↑
compareTo() based sorts
can also be sublinear!

Random (sublinear)	Non-random with duplicates (nearly linear)	Worst case (linear)
1E I 0402	are	1DNB377
1H Y L490	by	1DNB377
1R O Z572	sea	1DNB377
2H X E734	seashells	1DNB377
2I Y E230	seashells	1DNB377
2X O R846	sells	1DNB377
3CDB573	sells	1DNB377
3CVP720	she	1DNB377
3IGJ319	she	1DNB377
3KNA382	shells	1DNB377
3TAV879	shore	1DNB377
4CQP781	surely	1DNB377
4QGI284	the	1DNB377
4YHV229	the	1DNB377

Characters examined by MSD string sort

Summary of the performance of sorting algorithms

Frequency of operations.

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	yes	<code>compareTo()</code>
mergesort	$N \lg N$	$N \lg N$	N	yes	<code>compareTo()</code>
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$	no	<code>compareTo()</code>
heapsort	$2 N \lg N$	$2 N \lg N$	1	no	<code>compareTo()</code>
LSD †	$2 N W$	$2 N W$	$N + R$	yes	<code>charAt()</code>
MSD ‡	$2 N W$	$N \log_R N$	$N + D R$	yes	<code>charAt()</code>

D = function-call stack depth
(length of longest prefix match)



- * probabilistic
- † fixed-length W keys
- ‡ average-length W keys

MSD string sort vs. quicksort for strings

Disadvantages of MSD string sort.

- Extra space for aux[].
- Extra space for count[].
- Inner loop has a lot of instructions.
- Accesses memory "randomly" (cache inefficient).

Disadvantage of quicksort.

- Linearithmic number of string compares (not linear).
- Has to rescan many characters in keys with long prefix matches.

Goal. Combine advantages of MSD and quicksort.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ **MSD radix sort**
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

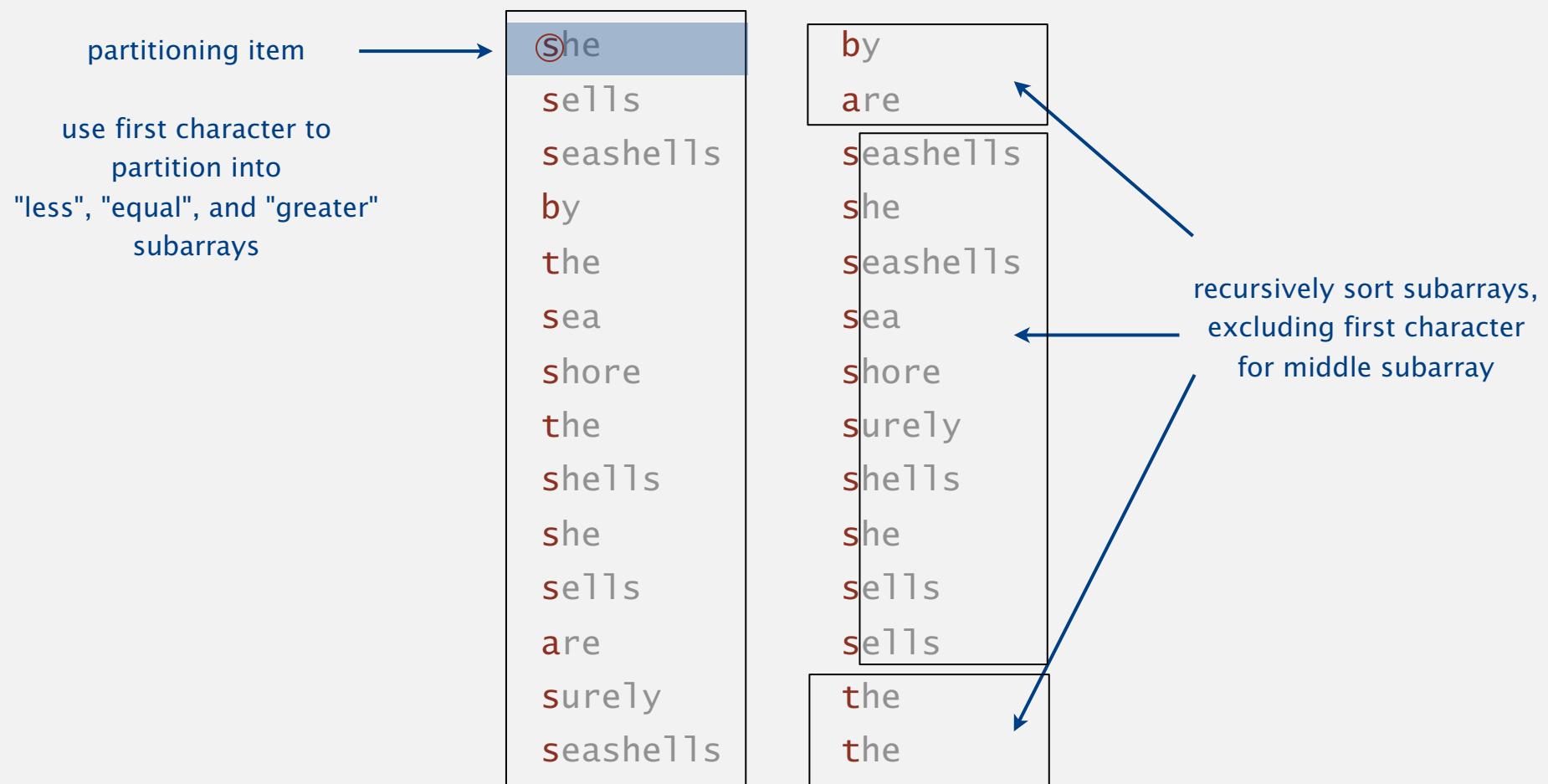
5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ ***3-way radix quicksort***
- ▶ *suffix arrays*

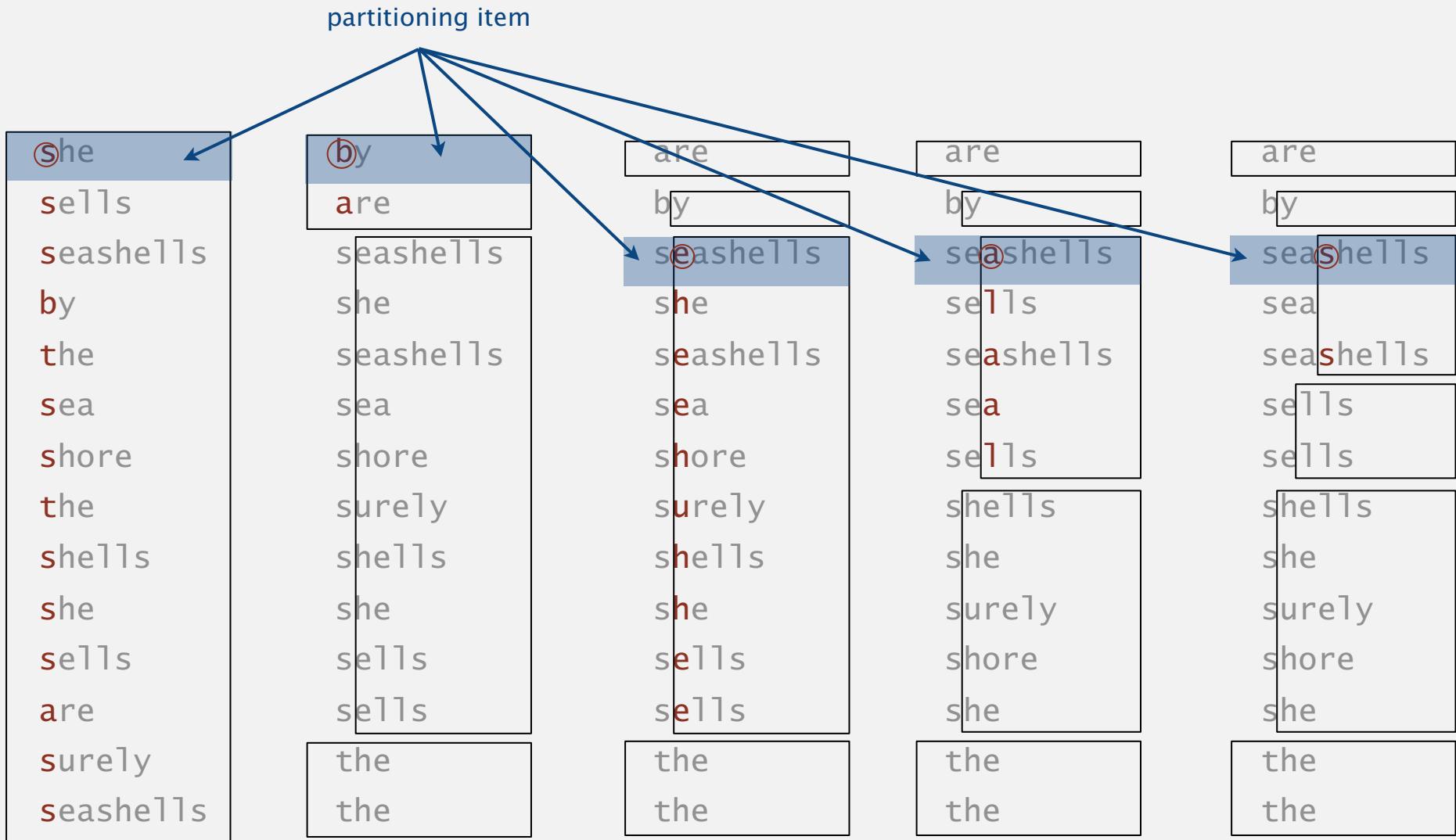
3-way string quicksort (Bentley and Sedgewick, 1997)

Overview. Do 3-way partitioning on the d^{th} character.

- Less overhead than R -way partitioning in MSD string sort.
- Does not re-examine characters equal to the partitioning char
(but does re-examine characters not equal to the partitioning char).



3-way string quicksort: trace of recursive calls



Trace of first few recursive calls for 3-way string quicksort (subarrays of size 1 not shown)

3-way string quicksort: Java implementation

```
private static void sort(String[] a)
{  sort(a, 0, a.length - 1, 0); }

private static void sort(String[] a, int lo, int hi, int d)
{
    if (hi <= lo) return;
    int lt = lo, gt = hi;
    int v = charAt(a[lo], d);
    int i = lo + 1;
    while (i <= gt)
    {
        int t = charAt(a[i], d);
        if      (t < v) exch(a, lt++, i++);
        else if (t > v) exch(a, i, gt--);
        else              i++;
    }
    sort(a, lo, lt-1, d);
    if (v >= 0) sort(a, lt, gt, d+1); ← sort 3 subarrays recursively
    sort(a, gt+1, hi, d);
}
```

3-way partitioning
(using d^{th} character)

to handle variable-length strings

3-way string quicksort vs. standard quicksort

Standard quicksort.

- Uses $\sim 2N \ln N$ **string compares** on average.
- Costly for keys with long common prefixes (and this is a common case!)

3-way string (radix) quicksort.

- Uses $\sim 2N \ln N$ **character compares** on average for random strings.
- Avoids re-comparing long common prefixes.

Fast Algorithms for Sorting and Searching Strings

Jon L. Bentley* Robert Sedgewick#

Abstract

We present theoretical algorithms for sorting and searching multikey data, and derive from them practical C implementations for applications in which keys are character strings. The sorting algorithm blends Quicksort and radix sort; it is competitive with the best known C sort codes. The searching algorithm blends tries and binary

that is competitive with the most efficient string sorting programs known. The second program is a symbol table implementation that is faster than hashing, which is commonly regarded as the fastest symbol table implementation. The symbol table implementation is much more space-efficient than multiway trees, and supports more advanced searches.

3-way string quicksort vs. MSD string sort

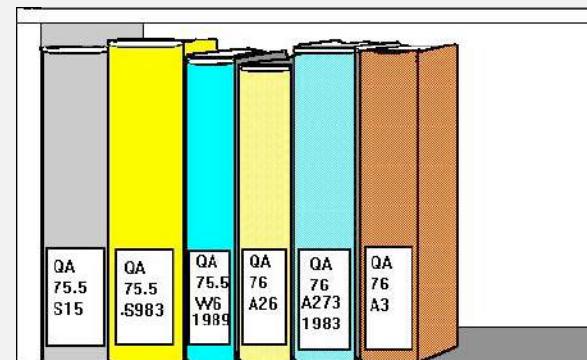
MSD string sort.

- Is cache-inefficient.
- Too much memory storing count[].
- Too much overhead reinitializing count[] and aux[].

3-way string quicksort.

- Has a short inner loop.
- Is cache-friendly.
- Is in-place.

library of Congress call numbers



Bottom line. 3-way string quicksort is method of choice for sorting strings.

Summary of the performance of sorting algorithms

Frequency of operations.

algorithm	guarantee	random	extra space	stable?	operations on keys
insertion sort	$\frac{1}{2} N^2$	$\frac{1}{4} N^2$	1	yes	compareTo()
mergesort	$N \lg N$	$N \lg N$	N	yes	compareTo()
quicksort	$1.39 N \lg N$ *	$1.39 N \lg N$	$c \lg N$	no	compareTo()
heapsort	$2 N \lg N$	$2 N \lg N$	1	no	compareTo()
LSD †	$2 N W$	$2 N W$	$N + R$	yes	charAt()
MSD ‡	$2 N W$	$N \log_R N$	$N + D R$	yes	charAt()
3-way string quicksort	$1.39 W N \lg N$ *	$1.39 N \lg N$	$\log N + W$	no	charAt()

* probabilistic

† fixed-length W keys

‡ average-length W keys

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ ***3-way radix quicksort***
- ▶ *suffix arrays*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ ***suffix arrays***

Keyword-in-context search

Given a text of N characters, preprocess it to enable fast substring search (find all occurrences of query string context).

```
% more tale.txt
it was the best of times
it was the worst of times
it was the age of wisdom
it was the age of foolishness
it was the epoch of belief
it was the epoch of incredulity
it was the season of light
it was the season of darkness
it was the spring of hope
it was the winter of despair
:
```

Applications. Linguistics, databases, web search, word processing,

Keyword-in-context search

Given a text of N characters, preprocess it to enable fast substring search
(find all occurrences of query string context).

```
% java KWIC tale.txt 15 ← characters of  
search surrounding context
```

```
o st giless to search for contraband  
her unavailing search for your fathe  
le and gone in search of her husband  
t provinces in search of impoverishe  
dispersing in search of other carri  
n that bed and search the straw hold
```

```
better thing  
t is a far far better thing that i do than  
some sense of better things else forgotte  
was capable of better things mr carton ent
```

Applications. Linguistics, databases, web search, word processing,

Suffix sort

input string															
	i	t	w	a	s	b	e	s	t	i	t	w	a	s	w
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
form suffixes															
0	i	t	w	a	s	b	e	s	t	i	t	w	a	s	w
1	t	w	a	s	b	e	s	t	i	t	w	a	s	w	
2	w	a	s	b	e	s	t	i	t	w	a	s	w		
3	a	s	b	e	s	t	i	t	w	a	s	w			
4	s	b	e	s	t	i	t	w	a	s	w				
5	b	e	s	t	i	t	w	a	s	w					
6	e	s	t	i	t	w	a	s	w						
7	s	t	i	t	w	a	s	w							
8	t	i	t	w	a	s	w								
9	i	t	w	a	s	w									
10	t	w	a	s	w										
11	w	a	s	w											
12	a	s	w												
13	s	w													
14	w														
sort suffixes to bring repeated substrings together															
3	a	s	b	e	s	t									
12	a	s	w												
5	b	e	s	t	i	t	w	a	s	w					
6	e	s	t	i	t	w	a	s	w						
0	i	t	w	a	s	b	e	s	t	i	t	w	a	s	w
9	i	t	w	a	s	w									
4	s	b	e	s	t	i	t	w	a	s	w				
7	s	t	i	t	w	a	s	w							
13	s	w													
8	t	i	t	w	a	s	w								
1	t	w	a	s	b	e	s	t	i	t	w	a	s	w	
10	t	w	a	s	w										
14	w														
2	w	a	s	b	e	s	t	i	t	w	a	s	w		
11	w	a	s	w											

Keyword-in-context search: suffix-sorting solution

- Preprocess: **suffix sort** the text.
- Query: **binary search** for query; scan until mismatch.

KWIC search for "search" in Tale of Two Cities

		:
632698	s e a l e d _ m y _ l e t t e r _ a n d _ ...	
713727	s e a m s t r e s s _ i s _ l i f t e d _ ...	
660598	s e a m s t r e s s _ o f _ t w e n t y _ ...	
67610	s e a m s t r e s s _ w h o _ w a s _ w i ...	
4430	s e a r c h _ f o r _ c o n t r a b a n d ...	
42705	s e a r c h _ f o r _ y o u r _ f a t h e ...	
499797	s e a r c h _ o f _ h e r _ h u s b a n d ...	
182045	s e a r c h _ o f _ i m p o v e r i s h e ...	
143399	s e a r c h _ o f _ o t h e r _ c a r r i ...	
411801	s e a r c h _ t h e _ s t r a w _ h o l d ...	
158410	s e a r e d _ m a r k i n g _ a b o u t _ ...	
691536	s e a s e _ a n d _ m a d a m e _ d e f a r ...	
536569	s e a s e _ a _ t e r r i b l e _ p a s s ...	
484763	s e a s e _ t h a t _ h a d _ b r o u g h ...	
	:	

Longest repeated substring

Given a string of N characters, find the longest repeated substring.

```
a a c a a g t t t a c a a g c a t g a t g c t g t a c t a  
g g a g a g t t a t a c t g g t c g t c a a a c c t g a a  
c c t a a t c c t t g t g t g t a c a c a c a c a c t a c t a  
c t g t c g t c g t c a t a t a t c g a g a t c a t c g a  
a c c g g a a g g c c g g a c a a g g c g g g g g t a t  
a g a t a g a t a g a c c c c t a g a t a c a c a t a c a  
t a g a t c t a g c t a g c t a g c t c a t c g a t a c a  
c a c t c t c a c a c t c a a g a g t t a t a c t g g t c  
a a c a c a c t a c t a c g a c a g a c g a c c a a c c a  
g a c a g a a a a a a a a c t c t a t a c t a t a a a a a
```

Applications. Bioinformatics, cryptanalysis, data compression, ...

Longest repeated substring: a musical application

Visualize repetitions in music. <http://www.bewitched.com>

Mary Had a Little Lamb



Bach's Goldberg Variations



Longest repeated substring

Given a string of N characters, find the longest repeated substring.

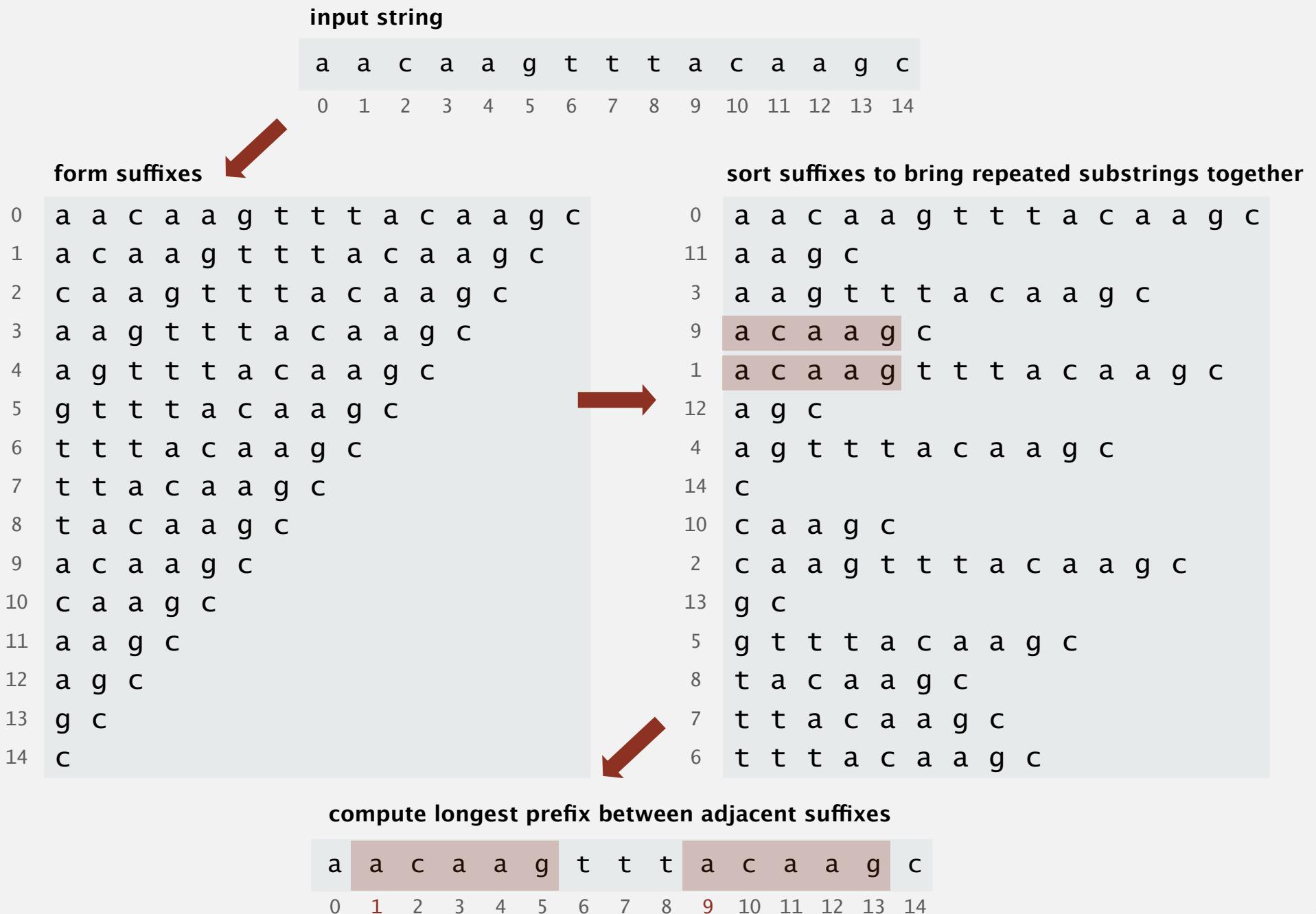
Brute-force algorithm.

- Try all indices i and j for start of possible match.
- Compute longest common prefix (LCP) for each pair.



Analysis. Running time $\leq D N^2$, where D is length of longest match.

Longest repeated substring: a sorting solution



Longest repeated substring: Java implementation

```
public String lrs(String s)
{
    int N = s.length();

    String[] suffixes = new String[N];
    for (int i = 0; i < N; i++)
        suffixes[i] = s.substring(i, N);

    Arrays.sort(suffixes);

    String lrs = "";
    for (int i = 0; i < N-1; i++)
    {
        int len = lcp(suffixes[i], suffixes[i+1]);
        if (len > lrs.length())
            lrs = suffixes[i].substring(0, len);
    }
    return lrs;
}
```

create suffixes
(linear time and space)

sort suffixes

find LCP between
adjacent suffixes in
sorted order

```
% java LRS < moby dick.txt
,- Such a funny, sporty, gamy, jesty, joky, hoky-poky lad, is the Ocean, oh! Th
```

Sorting challenge

Problem. Five scientists A , B , C , D , and E are looking for long repeated substring in a genome with over 1 billion nucleotides.

- A has a grad student do it by hand.
- B uses brute force (check all pairs).
- C uses suffix sorting solution with insertion sort.
- D uses suffix sorting solution with LSD string sort.
- ✓ • E uses suffix sorting solution with 3-way string quicksort.



but only if LRS is not long (!)

Q. Which one is more likely to lead to a cure cancer?

Longest repeated substring: empirical analysis

input file	characters	brute	suffix sort	length of LRS
LRS.java	2,162	0.6 sec	0.14 sec	73
amendments.txt	18,369	37 sec	0.25 sec	216
aesop.txt	191,945	1.2 hours	1.0 sec	58
mobydick.txt	1.2 million	43 hours †	7.6 sec	79
chromosome11.txt	7.1 million	2 months †	61 sec	12,567
pi.txt	10 million	4 months †	84 sec	14
pipi.txt	20 million	forever †	???	10 million

† estimated

Suffix sorting: worst-case input

Bad input: longest repeated substring very long.

- Ex: same letter repeated N times.
- Ex: two copies of the same Java codebase.

	form suffixes	sorted suffixes
0	t w i n s t w i n s	i n s
1	w i n s t w i n s	i n s t w i n s
2	i n s t w i n s	n s
3	n s t w i n s	n s t w i n s
4	s t w i n s	s
5	t w i n s	s t w i n s
6	w i n s	t w i n s
7	i n s	t w i n s t w i n s
8	n s	w i n s
9	s	w i n s t w i n s

LRS needs at least $1 + 2 + 3 + \dots + D$ character compares,
where $D = \text{length of longest match}$.

Running time. Quadratic (or worse) in D for LRS (and also for sort).

Suffix sorting challenge

Problem. Suffix sort an arbitrary string of length N .

Q. What is worst-case running time of best algorithm for problem?

- Quadratic.
- ✓ • Linearithmic. ← Manber-Myers algorithm
- ✓ • Linear. ← suffix trees (beyond our scope)
- Nobody knows.

Suffix sorting in linearithmic time

Manber-Myers MSD algorithm overview.

- Phase 0: sort on first character using key-indexed counting sort.
- Phase i : given array of suffixes sorted on first 2^{i-1} characters, create array of suffixes sorted on first 2^i characters.

Worst-case running time. $N \lg N$.

- Finishes after $\lg N$ phases.
- Can perform a phase in linear time. (!) [ahead]

Linearithmic suffix sort example: phase 0

original suffixes

0	b a b a a a a b c b a b a a a a a 0
1	a b a a a a b c b a b a a a a a 0
2	b a a a a b c b a b a a a a a 0
3	a a a a b c b a b a a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
6	a b c b a b a a a a a 0
7	b c b a b a a a a a 0
8	c b a b a a a a a 0
9	b a b a a a a a 0
10	a b a a a a a a 0
11	b a a a a a a 0
12	a a a a a a 0
13	a a a a a 0
14	a a a a 0
15	a a a 0
16	a 0
17	0

key-indexed counting sort (first character)

17	0
1	a b a a a a b c b a b a a a a a 0
16	a 0
3	a a a a b c b a b a a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
6	a b c b a b a a a a a 0
15	a a 0
14	a a a 0
13	a a a a 0
12	a a a a a 0
10	a b a a a a a 0
0	b a b a a a a b c b a b a a a a a 0
9	b a b a a a a a 0
11	b a a a a a a 0
7	b c b a b a a a a a 0
2	b a a a a b c b a b a a a a a 0
8	c b a b a a a a a 0

↑
sorted

Linearithmic suffix sort example: phase 1

original suffixes

0	b a b a a a a b c b a b a a a a a 0
1	a b a a a a b c b a b a a a a a 0
2	b a a a a b c b a b a a a a a 0
3	a a a a b c b a b a a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
6	a b c b a b a a a a a 0
7	b c b a b a a a a a 0
8	c b a b a a a a a 0
9	b a b a a a a a 0
10	a b a a a a a a 0
11	b a a a a a a 0
12	a a a a a a 0
13	a a a a a 0
14	a a a a 0
15	a a a 0
16	a 0
17	0

index sort (first two characters)

17	0
16	a 0
12	a a a a a 0
3	a a a a b c b a b a a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
13	a a a a 0
15	a a 0
14	a a a 0
6	a b c b a b a a a a a 0
1	a b a a a a b c b a b a a a a 0
10	a b a a a a a 0
0	b a b a a a a b c b a b a a a a a 0
9	b a b a a a a a 0
11	b a a a a a 0
2	b a a a a b c b a b a a a a a 0
7	b c b a b a a a a 0
8	c b a b a a a a a 0

↑
sorted

Linearithmic suffix sort example: phase 2

original suffixes

0	b a b a a a a b c b a b a a a a a 0
1	a b a a a a b c b a b a a a a a 0
2	b a a a a b c b a b a a a a a 0
3	a a a a b c b a b a a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
6	a b c b a b a a a a a 0
7	b c b a b a a a a a 0
8	c b a b a a a a a 0
9	b a b a a a a a 0
10	a b a a a a a 0
11	b a a a a a 0
12	a a a a a 0
13	a a a a 0
14	a a a 0
15	a a 0
16	a 0
17	0

index sort (first four characters)

17	0
16	a 0
15	a a 0
14	a a a 0
3	a a a a b c b a b a a a a a 0
12	a a a a a 0
13	a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
1	a b a a a a b c b a b a a a a 0
10	a b a a a a a 0
6	a b c b a b a a a a a 0
2	b a a a a b c b a b a a a a a 0 a 0
11	b a a a a a 0
0	b a b a a a a b c b a b a a a a a 0
9	b a b a a a a a 0
7	b c b a b a a a a a 0
8	c b a b a a a a a 0

↑
sorted

Linearithmic suffix sort example: phase 3

original suffixes

0	b a b a a a a b c b a b a a a a a 0
1	a b a a a a b c b a b a a a a a 0
2	b a a a a b c b a b a a a a a 0
3	a a a a a b c b a b a a a a a 0
4	a a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
6	a b c b a b a a a a a 0
7	b c b a b a a a a a 0
8	c b a b a a a a a 0
9	b a b a a a a a 0
10	a b a a a a a a 0
11	b a a a a a a 0
12	a a a a a a 0
13	a a a a a 0
14	a a a a 0
15	a a a 0
16	a a 0
17	0

index sort (first eight characters)

17	0
16	a 0
15	a a 0
14	a a a 0
13	a a a a 0
12	a a a a a 0
3	a a a a b c b a b a a a a a 0
4	a a a b c b a b a a a a a 0
5	a a b c b a b a a a a a 0
10	a b a a a a a 0
1	a b a a a a b c b a b a a a a a 0
6	a b c b a b a a a a a 0
11	b a a a a a a 0
2	b a a a a b c b a b a a a a a 0 a 0
9	b a b a a a a a 0
0	b a b a a a a b c b a b a a a a a 0
7	b c b a b a a a a a 0
8	c b a b a a a a a 0

finished (no equal keys)

Constant-time string compare by indexing into inverse

	original suffixes	index sort (first four characters)	inverse[]
0	b a b a a a a b c b a b a a a a a 0	17 0	0 14
1	a b a a a a b c b a b a a a a a 0	16 a 0	1 9
2	b a a a a b c b a b a a a a a 0	15 a a 0	2 12
3	a a a a b c b a b a a a a a 0	14 a a a 0	3 4
4	a a a b c b a b a a a a a 0	3 a a a a b c b a b a a a a a 0	4 7
5	a a b c b a b a a a a a 0	12 a a a a a 0	5 8
6	a b c b a b a a a a a 0	13 a a a a 0	6 11
7	b c b a b a a a a a 0	4 a a a b c b a b a a a a a 0	7 16
8	c b a b a a a a a 0	5 a a b c b a b a a a a a 0	8 17
9	b a b a a a a a 0	1 a b a a a a b c b a b a a a a a 0	9 15
10	a b a a a a a 0	10 a b a a a a a 0	10 10
11	b a a a a a a 0	6 a b c b a b a a a a a 0	11 13
12	a a a a a a 0	2 b a a a a b c b a b a a a a a a 0 a 0	12 5
13	a a a a a 0	11 b a a a a a a 0	13 6
14	a a a a 0	0 b a b a a a a b c b a b a a a a a 0	14 3
15	a a 0	9 b a b a a a a a 0	15 2
16	a 0	7 b c b a b a a a a a 0	16 1
17	0	8 c b a b a a a a a 0	17 0

$\text{suffixes}_4[13] \leq \text{suffixes}_4[4]$ (because $\text{inverse}[13] < \text{inverse}[4]$)
 so $\text{suffixes}_8[9] \leq \text{suffixes}_8[0]$

String sorting summary

We can develop linear-time sorts.

- Key compares not necessary for string keys.
- Use characters as index in an array.

We can develop sublinear-time sorts.

- Input size is amount of data in keys (not number of keys).
- Not all of the data has to be examined.

3-way string quicksort is asymptotically optimal.

- $1.39 N \lg N$ chars for random data.

Long strings are rarely random in practice.

- Goal is often to learn the structure!
- May need specialized algorithms.

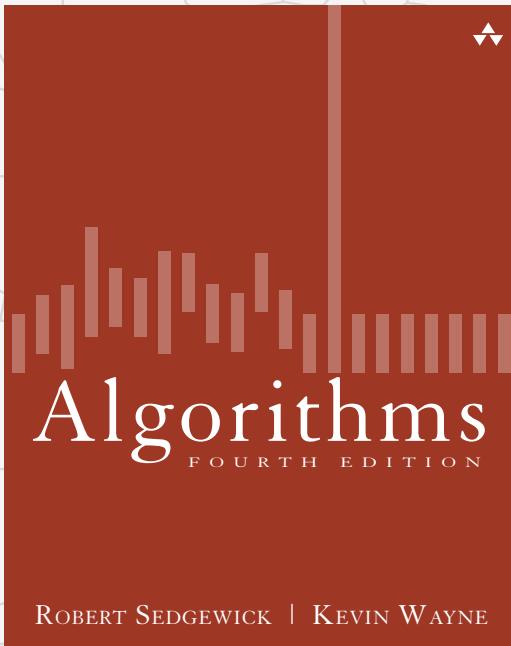
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

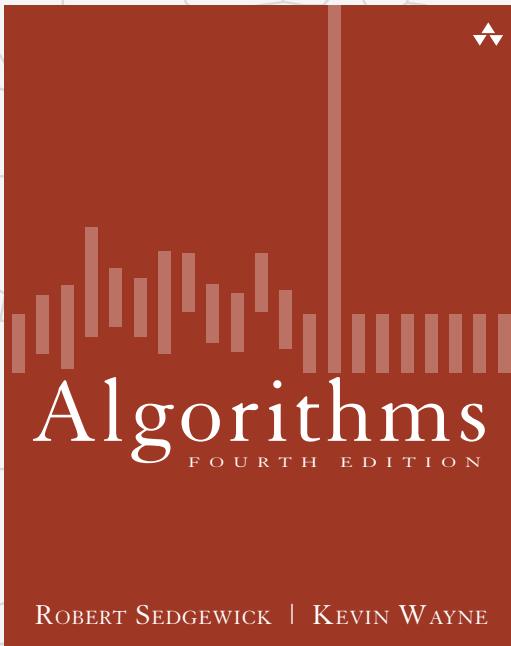
- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ ***suffix arrays***



<http://algs4.cs.princeton.edu>

5.1 STRING SORTS

- ▶ *strings in Java*
- ▶ *key-indexed counting*
- ▶ *LSD radix sort*
- ▶ *MSD radix sort*
- ▶ *3-way radix quicksort*
- ▶ *suffix arrays*



<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

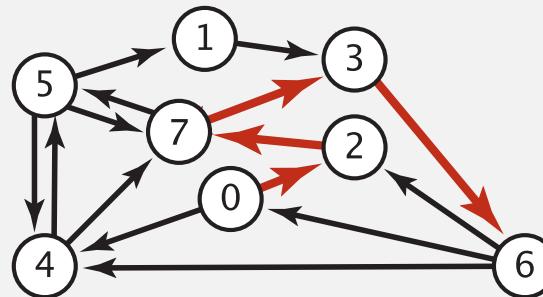
- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Shortest paths in an edge-weighted digraph

Given an edge-weighted digraph, find the shortest path from s to t .

edge-weighted digraph

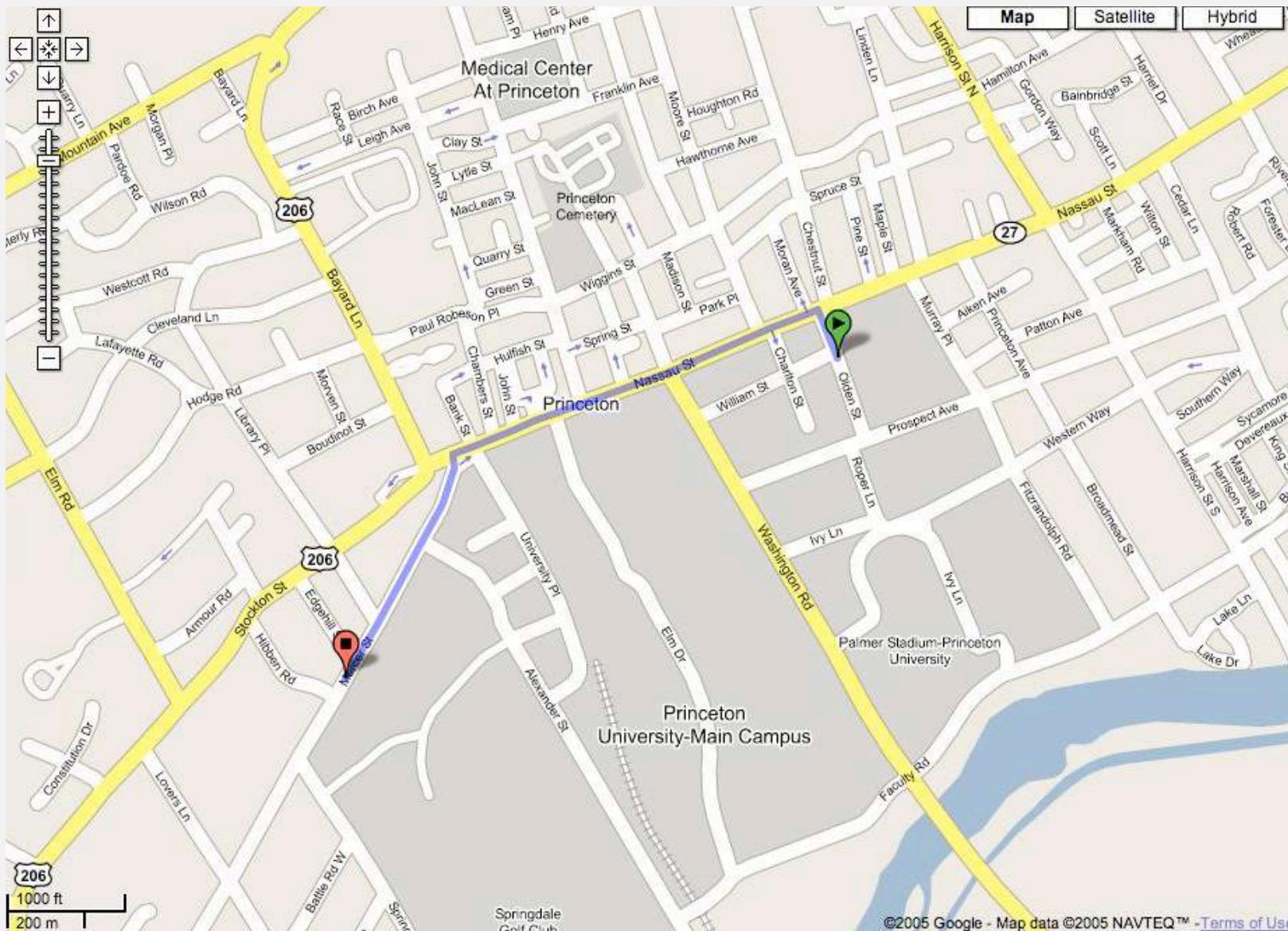
4->5	0.35
5->4	0.35
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



shortest path from 0 to 6

0->2	0.26
2->7	0.34
7->3	0.39
3->6	0.52
6->0	0.58

Google maps



Car navigation



Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in TeX.
- Urban traffic planning.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Exploiting arbitrage opportunities in currency exchange.
- Optimal truck routing through given traffic congestion pattern.



http://en.wikipedia.org/wiki/Seam_carving



Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

Shortest path variants

Which vertices?

- Source-sink: from one vertex to another.
- Single source: from one vertex to every other.
- All pairs: between all pairs of vertices.

Restrictions on edge weights?

- Nonnegative weights.
- Arbitrary weights.
- Euclidean weights.

Cycles?

- No directed cycles.
- No "negative cycles."

Simplifying assumption. Shortest paths from s to each vertex v exist.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

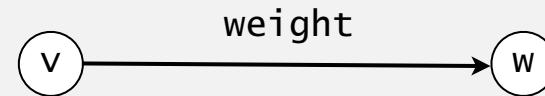
<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Weighted directed edge API

```
public class DirectedEdge  
  
    DirectedEdge(int v, int w, double weight)      weighted edge v→w  
  
    int from()                                     vertex v  
  
    int to()                                       vertex w  
  
    double weight()                                weight of this edge  
  
    String toString()                             string representation
```



Idiom for processing an edge e: `int v = e.from(), w = e.to();`

Weighted directed edge: implementation in Java

Similar to Edge for undirected graphs, but a bit simpler.

```
public class DirectedEdge
{
    private final int v, w;
    private final double weight;

    public DirectedEdge(int v, int w, double weight)
    {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int from()
    { return v; }

    public int to()
    { return w; }

    public int weight()
    { return weight; }
}
```

from() and to() replace
either() and other()

Edge-weighted digraph API

```
public class EdgeWeightedDigraph
```

EdgeWeightedDigraph(int V)	<i>edge-weighted digraph with V vertices</i>
----------------------------	--

EdgeWeightedDigraph(In in)	<i>edge-weighted digraph from input stream</i>
----------------------------	--

void addEdge(DirectedEdge e)	<i>add weighted directed edge e</i>
------------------------------	-------------------------------------

Iterable<DirectedEdge> adj(int v)	<i>edges pointing from v</i>
-----------------------------------	------------------------------

int V()	<i>number of vertices</i>
---------	---------------------------

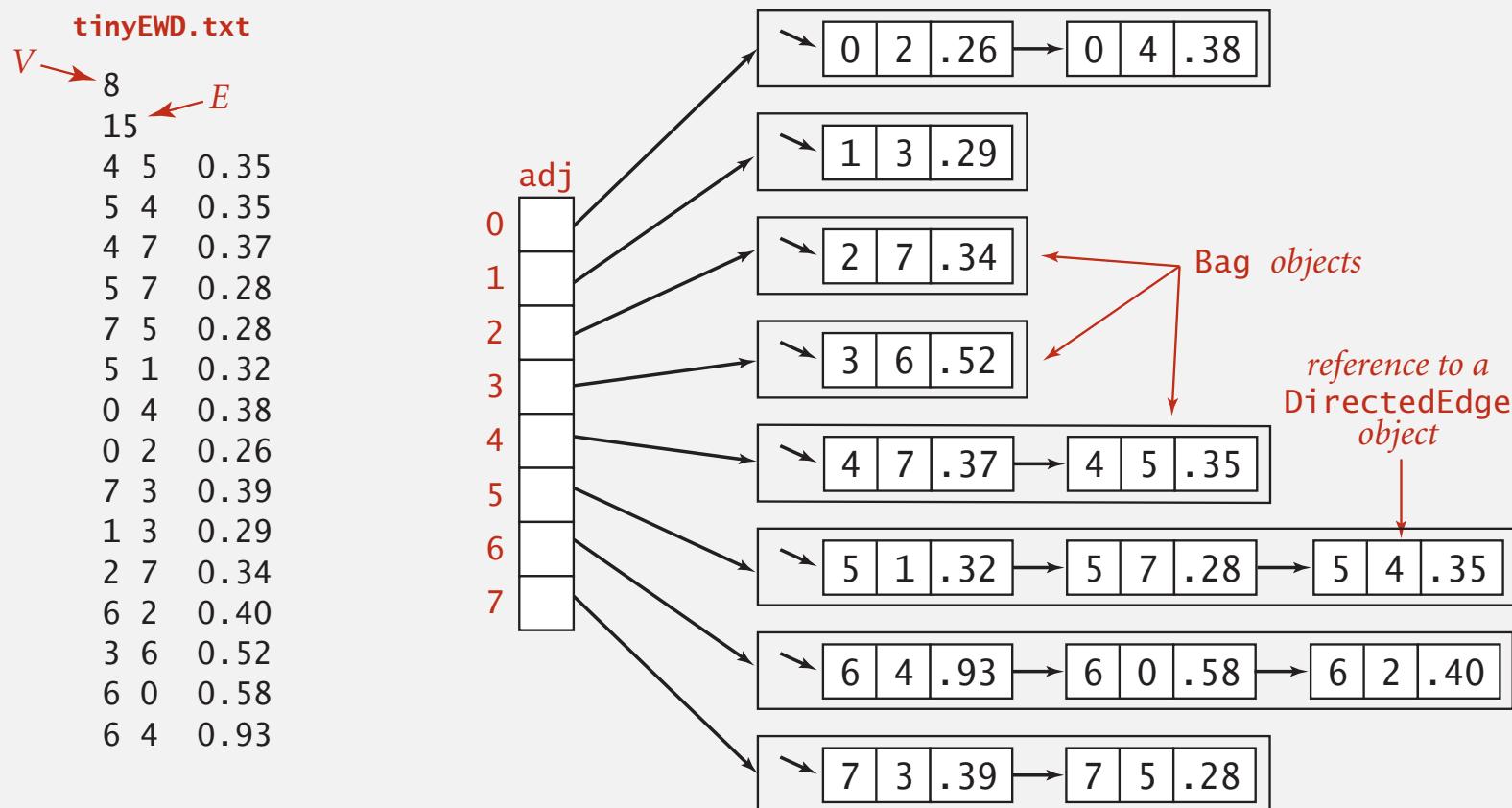
int E()	<i>number of edges</i>
---------	------------------------

Iterable<DirectedEdge> edges()	<i>all edges</i>
--------------------------------	------------------

String toString()	<i>string representation</i>
-------------------	------------------------------

Conventions. Allow self-loops and parallel edges.

Edge-weighted digraph: adjacency-lists representation



Edge-weighted digraph: adjacency-lists implementation in Java

Same as EdgeWeightedGraph except replace Graph with Digraph.

```
public class EdgeWeightedDigraph
{
    private final int V;
    private final Bag<Edge>[] adj;

    public EdgeWeightedDigraph(int V)
    {
        this.V = V;
        adj = (Bag<DirectedEdge>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<DirectedEdge>();
    }

    public void addEdge(DirectedEdge e)
    {
        int v = e.from();
        adj[v].add(e);
    }

    public Iterable<DirectedEdge> adj(int v)
    { return adj[v]; }
}
```

add edge $e = v \rightarrow w$ to
only v 's adjacency list

Single-source shortest paths API

Goal. Find the shortest path from s to every other vertex.

```
public class SP
```

```
    SP(EdgeWeightedDigraph G, int s) shortest paths from s in graph G
```

```
    double distTo(int v) length of shortest path from s to v
```

```
    Iterable <DirectedEdge> pathTo(int v) shortest path from s to v
```

```
    boolean hasPathTo(int v) is there a path from s to v?
```

```
SP sp = new SP(G, s);
for (int v = 0; v < G.V(); v++)
{
    StdOut.printf("%d to %d (%.2f): ", s, v, sp.distTo(v));
    for (DirectedEdge e : sp.pathTo(v))
        StdOut.print(e + " ");
    StdOut.println();
}
```

Single-source shortest paths API

Goal. Find the shortest path from s to every other vertex.

```
public class SP
```

```
    SP(EdgeWeightedDigraph G, int s) shortest paths from s in graph G
```

```
    double distTo(int v) length of shortest path from s to v
```

```
    Iterable <DirectedEdge> pathTo(int v) shortest path from s to v
```

```
    boolean hasPathTo(int v) is there a path from s to v?
```

```
% java SP tinyEWD.txt 0
0 to 0 (0.00):
0 to 1 (1.05): 0->4 0.38  4->5 0.35  5->1 0.32
0 to 2 (0.26): 0->2 0.26
0 to 3 (0.99): 0->2 0.26  2->7 0.34  7->3 0.39
0 to 4 (0.38): 0->4 0.38
0 to 5 (0.73): 0->4 0.38  4->5 0.35
0 to 6 (1.51): 0->2 0.26  2->7 0.34  7->3 0.39  3->6 0.52
0 to 7 (0.60): 0->2 0.26  2->7 0.34
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

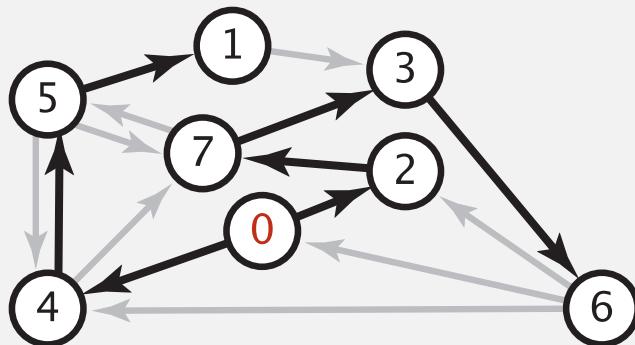
Data structures for single-source shortest paths

Goal. Find the shortest path from s to every other vertex.

Observation. A **shortest-paths tree** (SPT) solution exists. Why?

Consequence. Can represent the SPT with two vertex-indexed arrays:

- $\text{distTo}[v]$ is length of shortest path from s to v .
- $\text{edgeTo}[v]$ is last edge on shortest path from s to v .



shortest-paths tree from 0

	edgeTo[]	distTo[]
0	null	0
1	5->1	0.32
2	0->2	0.26
3	7->3	0.37
4	0->4	0.38
5	4->5	0.35
6	3->6	0.52
7	2->7	0.60

parent-link representation

Data structures for single-source shortest paths

Goal. Find the shortest path from s to every other vertex.

Observation. A **shortest-paths tree** (SPT) solution exists. Why?

Consequence. Can represent the SPT with two vertex-indexed arrays:

- $\text{distTo}[v]$ is length of shortest path from s to v .
- $\text{edgeTo}[v]$ is last edge on shortest path from s to v .

```
public double distTo(int v)
{   return distTo[v];  }

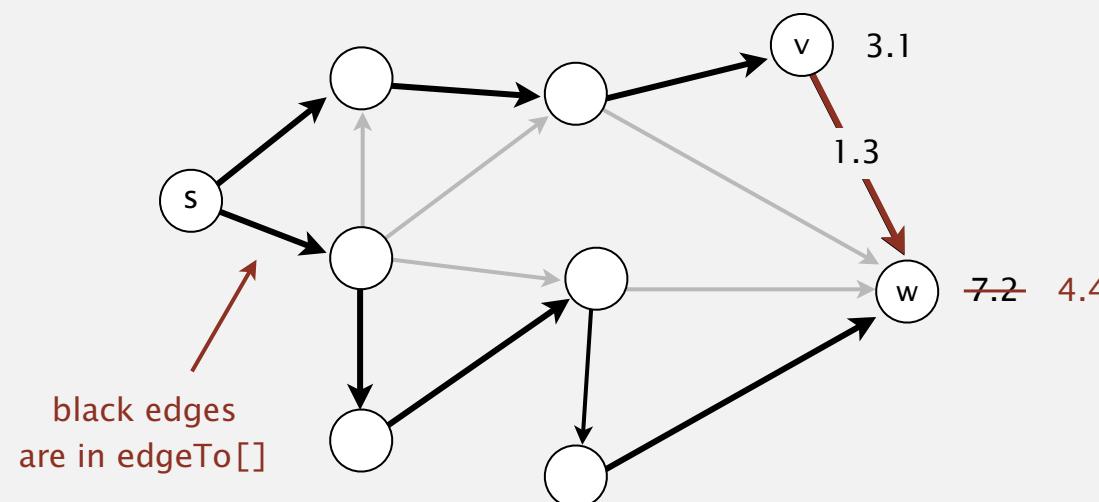
public Iterable<DirectedEdge> pathTo(int v)
{
    Stack<DirectedEdge> path = new Stack<DirectedEdge>();
    for (DirectedEdge e = edgeTo[v]; e != null; e = edgeTo[e.from()])
        path.push(e);
    return path;
}
```

Edge relaxation

Relax edge $e = v \rightarrow w$.

- $\text{distTo}[v]$ is length of shortest **known** path from s to v .
- $\text{distTo}[w]$ is length of shortest **known** path from s to w .
- $\text{edgeTo}[w]$ is last edge on shortest **known** path from s to w .
- If $e = v \rightarrow w$ gives shorter path to w through v ,
update both $\text{distTo}[w]$ and $\text{edgeTo}[w]$.

$v \rightarrow w$ successfully relaxes



Edge relaxation

Relax edge $e = v \rightarrow w$.

- $\text{distTo}[v]$ is length of shortest **known** path from s to v .
- $\text{distTo}[w]$ is length of shortest **known** path from s to w .
- $\text{edgeTo}[w]$ is last edge on shortest **known** path from s to w .
- If $e = v \rightarrow w$ gives shorter path to w through v ,
update both $\text{distTo}[w]$ and $\text{edgeTo}[w]$.

```
private void relax(DirectedEdge e)
{
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight())
    {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
    }
}
```

Shortest-paths optimality conditions

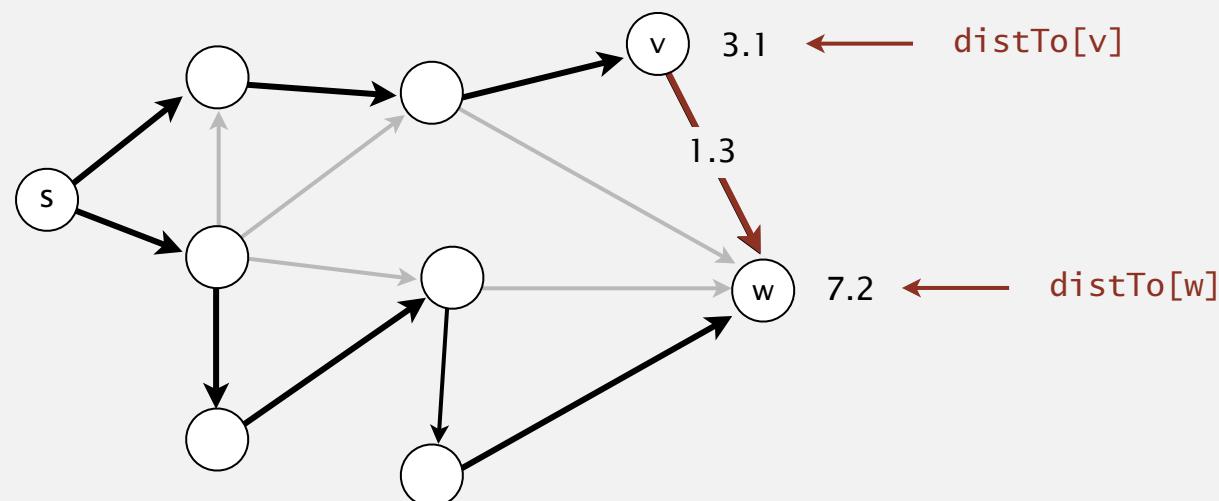
Proposition. Let G be an edge-weighted digraph.

Then $\text{distTo}[]$ are the shortest path distances from s iff:

- For each vertex v , $\text{distTo}[v]$ is the length of some path from s to v .
- For each edge $e = v \rightarrow w$, $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight}()$.

Pf. \Leftarrow [necessary]

- Suppose that $\text{distTo}[w] > \text{distTo}[v] + e.\text{weight}()$ for some edge $e = v \rightarrow w$.
- Then, e gives a path from s to w (through v) of length less than $\text{distTo}[w]$.



Shortest-paths optimality conditions

Proposition. Let G be an edge-weighted digraph.

Then $\text{distTo}[]$ are the shortest path distances from s iff:

- For each vertex v , $\text{distTo}[v]$ is the length of some path from s to v .
- For each edge $e = v \rightarrow w$, $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight}()$.

Pf. \Rightarrow [sufficient]

- Suppose that $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k = w$ is a shortest path from s to w .
- Then, $\text{distTo}[v_1] \leq \text{distTo}[v_0] + e_1.\text{weight}()$
 $\text{distTo}[v_2] \leq \text{distTo}[v_1] + e_2.\text{weight}()$
 \dots
 $\text{distTo}[v_k] \leq \text{distTo}[v_{k-1}] + e_k.\text{weight}()$
- Add inequalities; simplify; and substitute $\text{distTo}[v_0] = \text{distTo}[s] = 0$:
$$\text{distTo}[w] = \text{distTo}[v_k] \leq \underbrace{e_1.\text{weight}() + e_2.\text{weight}() + \dots + e_k.\text{weight}()}_{\text{weight of shortest path from } s \text{ to } w}$$
- Thus, $\text{distTo}[w]$ is the weight of shortest path to w . ■

weight of some path from s to w

$e_i = i^{\text{th}}$ edge on shortest path from s to w

Generic shortest-paths algorithm

Generic algorithm (to compute SPT from s)

Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.

Repeat until optimality conditions are satisfied:

- Relax any edge.

Proposition. Generic algorithm computes SPT (if it exists) from s .

Pf sketch.

- Throughout algorithm, $\text{distTo}[v]$ is the length of a simple path from s to v (and $\text{edgeTo}[v]$ is last edge on path).
- Each successful relaxation decreases $\text{distTo}[v]$ for some v .
- The entry $\text{distTo}[v]$ can decrease at most a finite number of times. ■

Generic shortest-paths algorithm

Generic algorithm (to compute SPT from s)

Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.

Repeat until optimality conditions are satisfied:

- Relax any edge.
-

Efficient implementations. How to choose which edge to relax?

Ex 1. Dijkstra's algorithm (nonnegative weights).

Ex 2. Topological sort algorithm (no directed cycles).

Ex 3. Bellman-Ford algorithm (no negative cycles).

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ *APIs*
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Edsger W. Dijkstra: select quotes

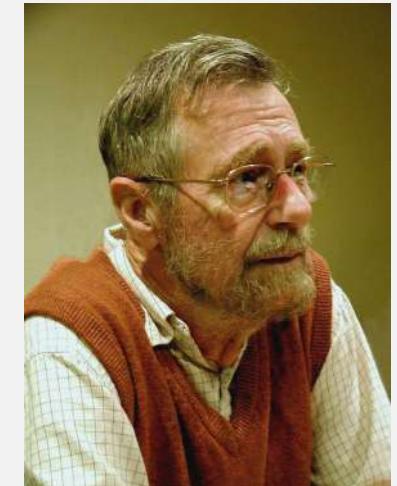
“Do only what only you can do.”

“In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.”

“The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.”

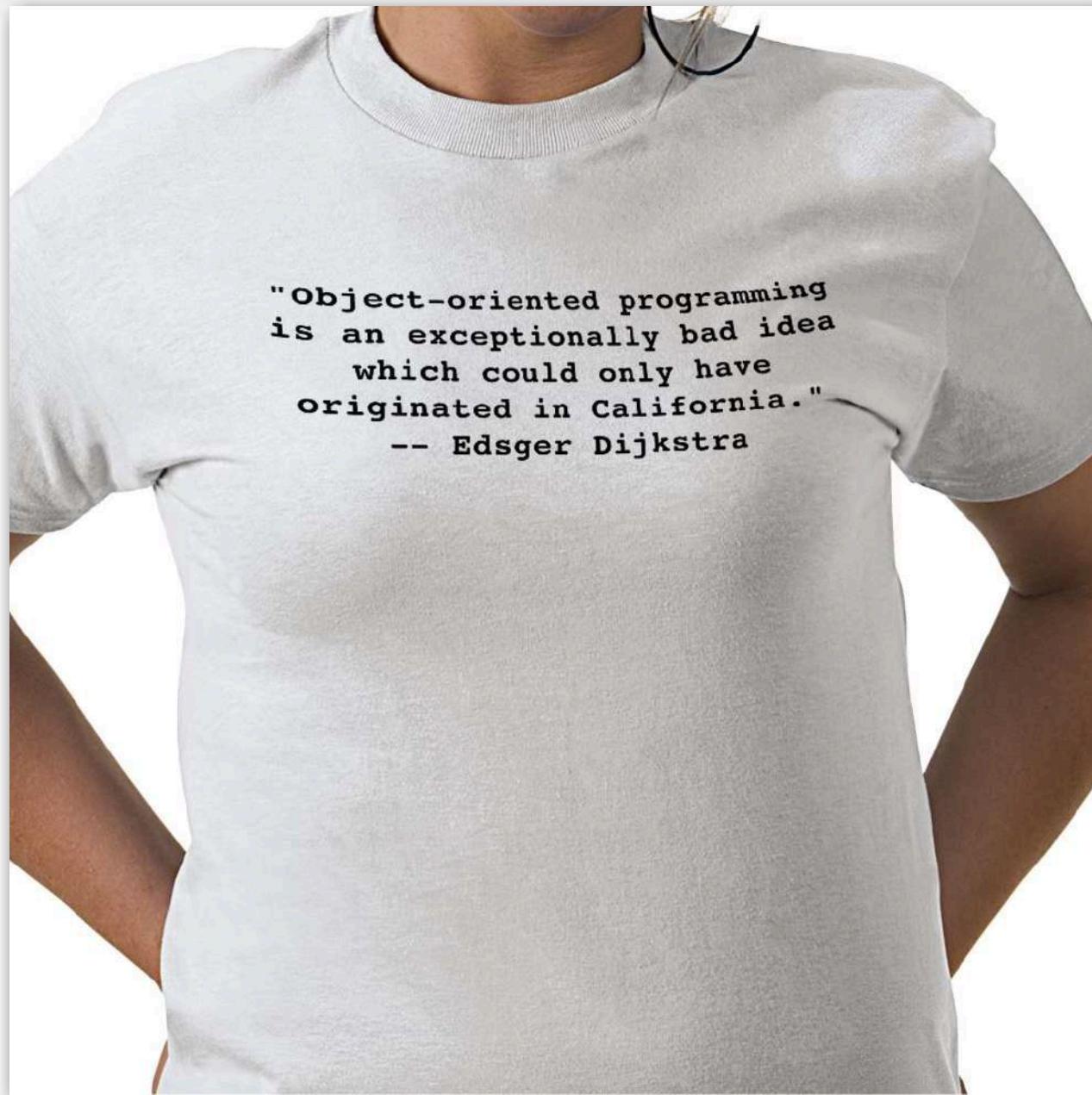
“It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration.”

“APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.”



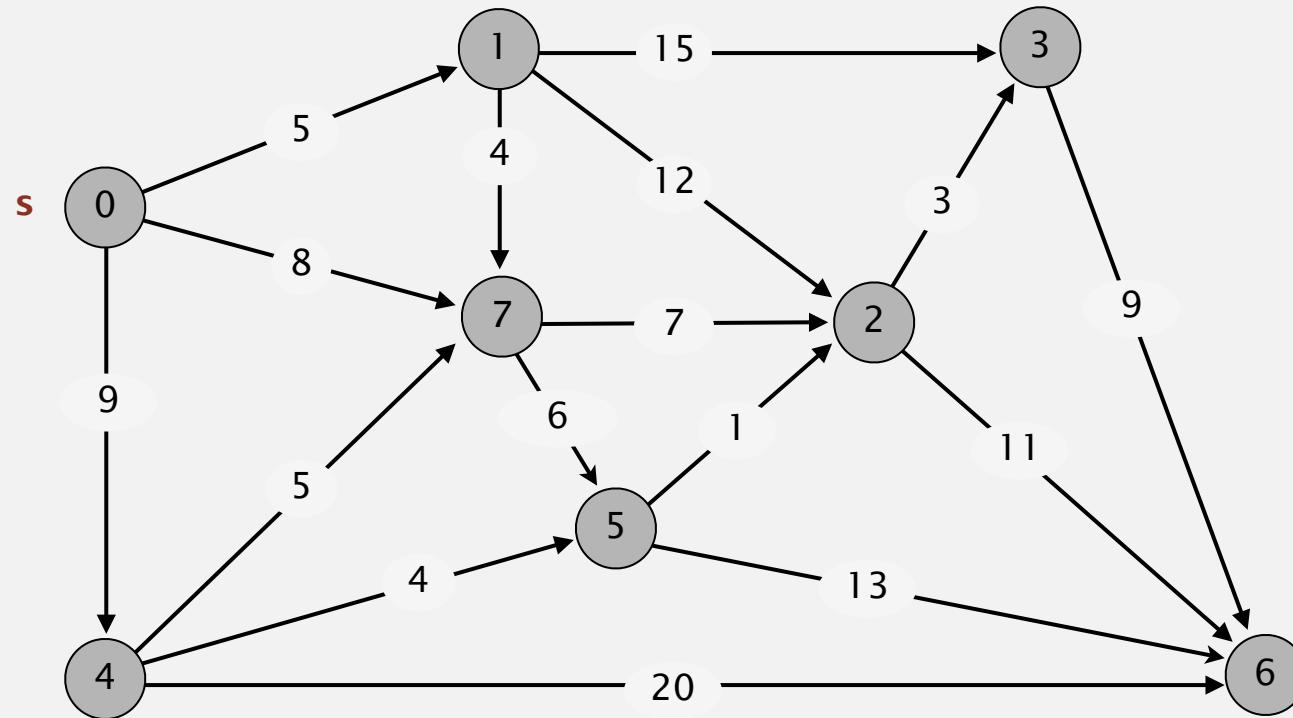
Edsger W. Dijkstra
Turing award 1972

Edsger W. Dijkstra: select quotes



Dijkstra's algorithm demo

- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.

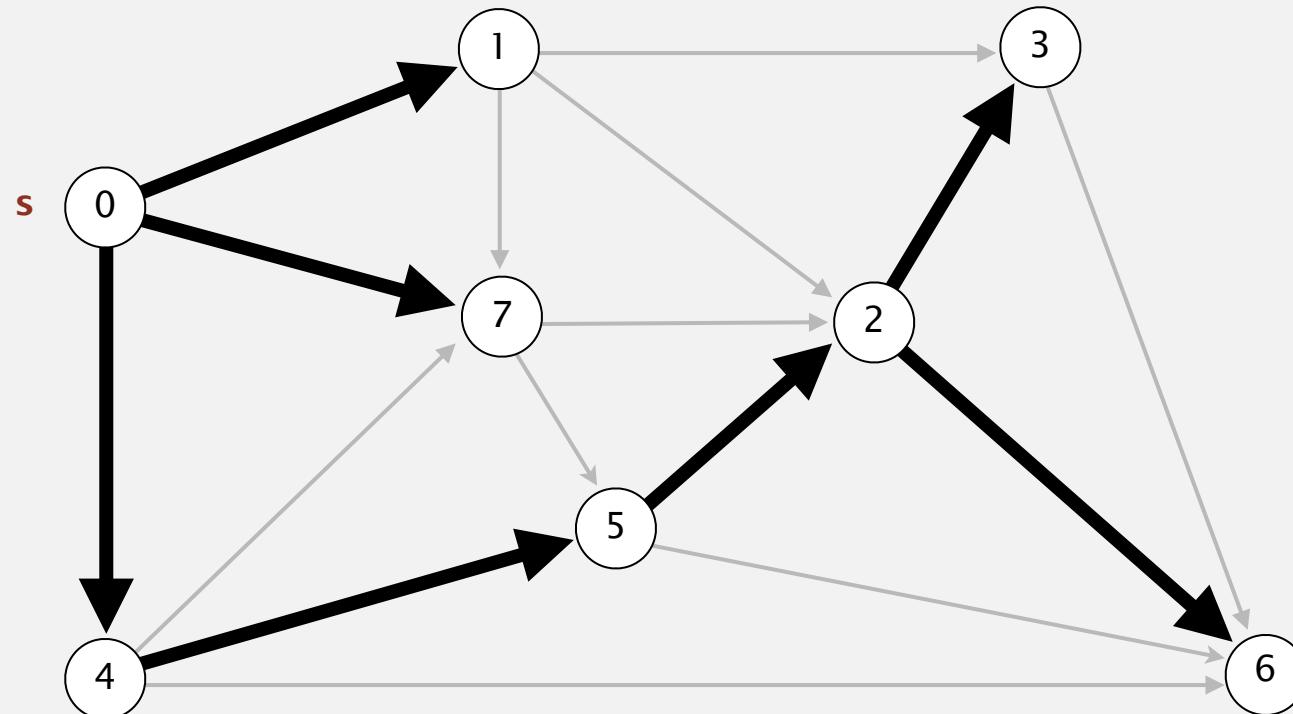


an edge-weighted digraph

0→1	5.0
0→4	9.0
0→7	8.0
1→2	12.0
1→3	15.0
1→7	4.0
2→3	3.0
2→6	11.0
3→6	9.0
4→5	4.0
4→6	20.0
4→7	5.0
5→2	1.0
5→6	13.0
7→5	6.0
7→2	7.0

Dijkstra's algorithm demo

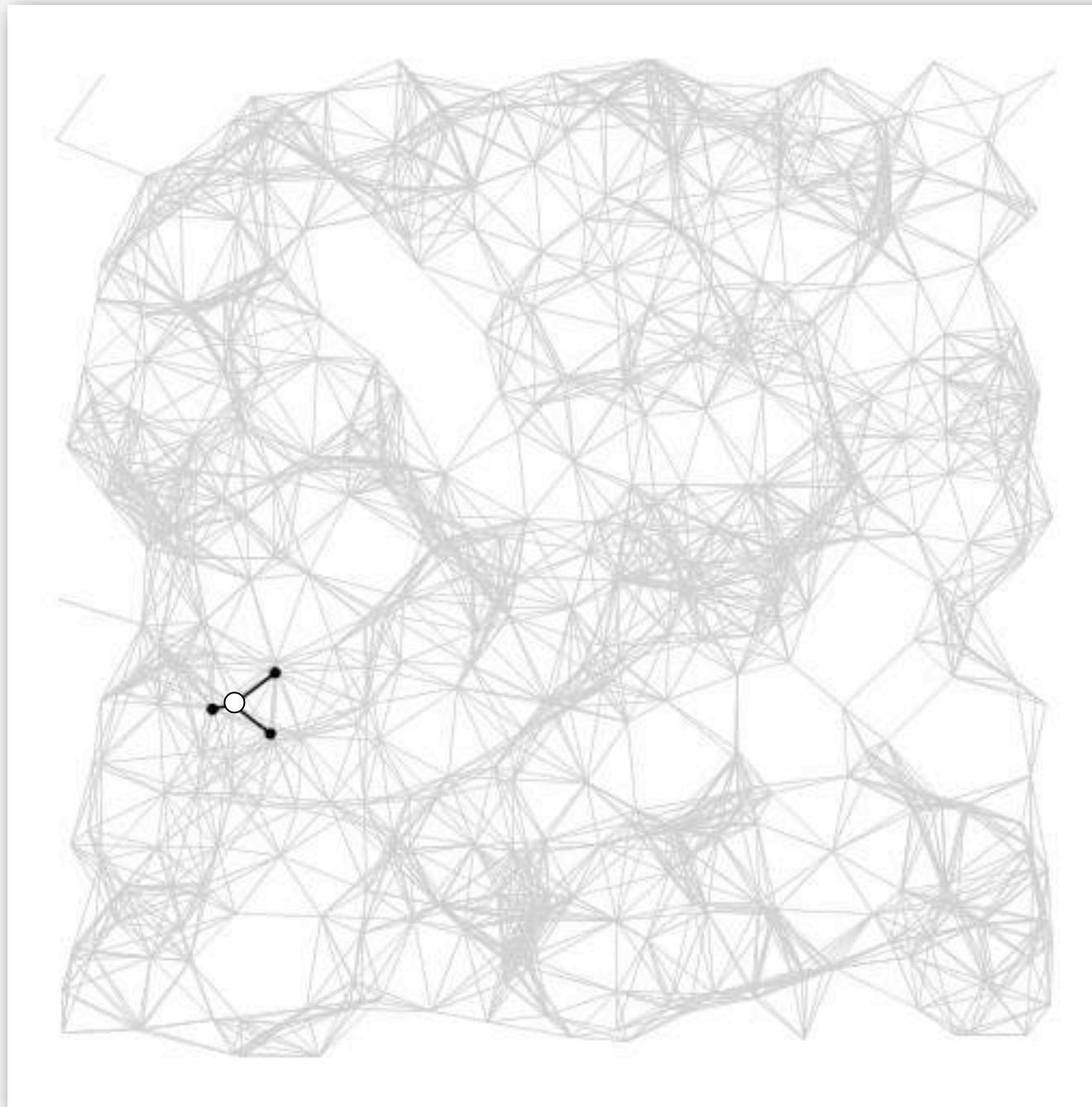
- Consider vertices in increasing order of distance from s (non-tree vertex with the lowest $\text{distTo}[]$ value).
- Add vertex to tree and relax all edges pointing from that vertex.



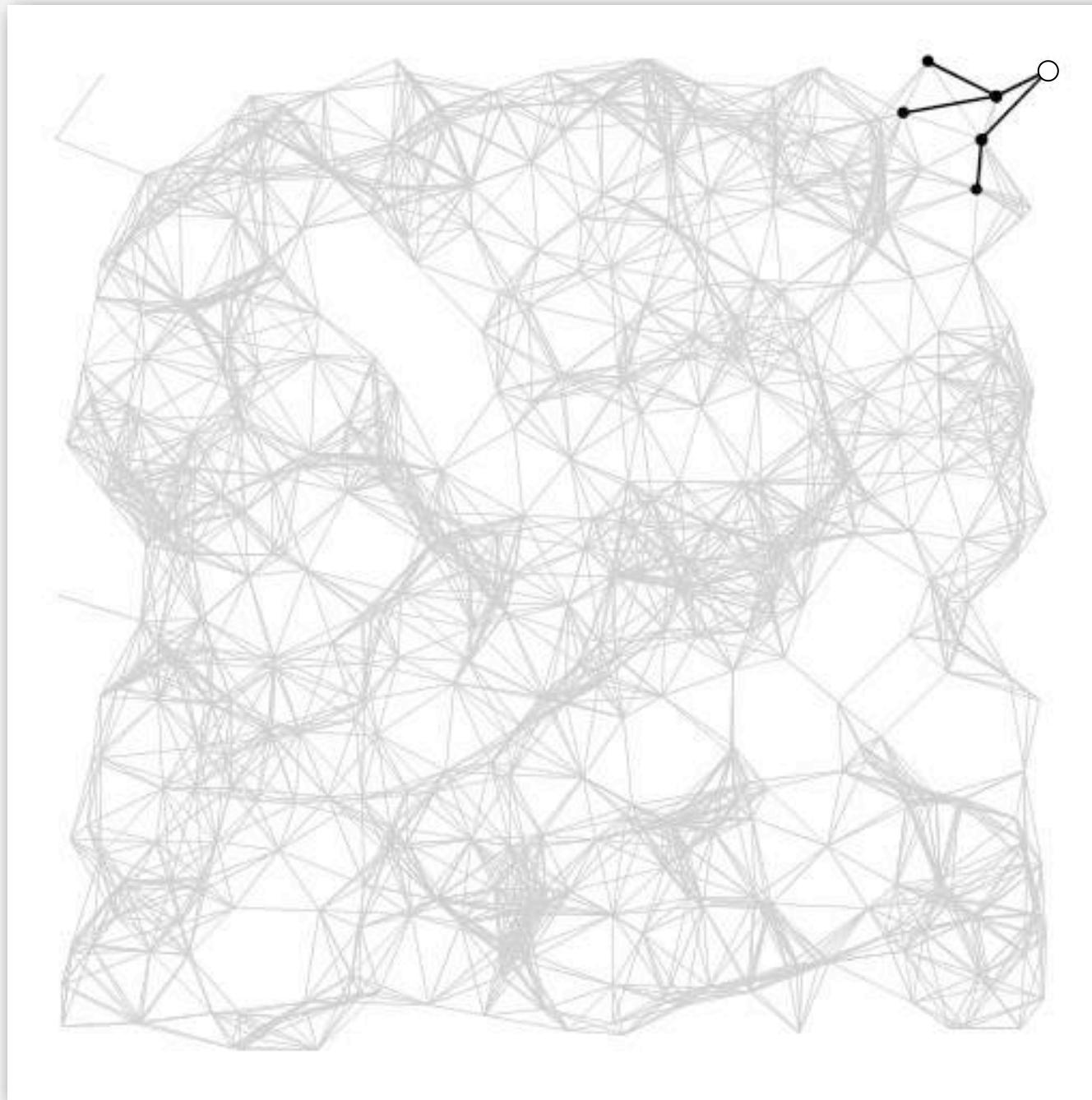
shortest-paths tree from vertex s

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

Dijkstra's algorithm visualization



Dijkstra's algorithm visualization



Dijkstra's algorithm: correctness proof

Proposition. Dijkstra's algorithm computes a SPT in any edge-weighted digraph with nonnegative weights.

Pf.

- Each edge $e = v \rightarrow w$ is relaxed exactly once (when v is relaxed), leaving $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight}()$.
- Inequality holds until algorithm terminates because:
 - $\text{distTo}[w]$ cannot increase ← distTo[] values are monotone decreasing
 - $\text{distTo}[v]$ will not change ← edge weights are nonnegative and we choose lowest distTo[] value at each step
- Thus, upon termination, shortest-paths optimality conditions hold. ■

Dijkstra's algorithm: Java implementation

```
public class DijkstraSP
{
    private DirectedEdge[] edgeTo;
    private double[] distTo;
    private IndexMinPQ<Double> pq;

    public DijkstraSP(EdgeWeightedDigraph G, int s)
    {
        edgeTo = new DirectedEdge[G.V()];
        distTo = new double[G.V()];
        pq = new IndexMinPQ<Double>(G.V());

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        pq.insert(s, 0.0);
        while (!pq.isEmpty())
        {
            int v = pq.delMin();
            for (DirectedEdge e : G.adj(v))
                relax(e);
        }
    }
}
```

←
relax vertices in order
of distance from s

Dijkstra's algorithm: Java implementation

```
private void relax(DirectedEdge e)
{
    int v = e.from(), w = e.to();
    if (distTo[w] > distTo[v] + e.weight())
    {
        distTo[w] = distTo[v] + e.weight();
        edgeTo[w] = e;
        if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
        else                  pq.insert      (w, distTo[w]);
    }
}
```

← update PQ

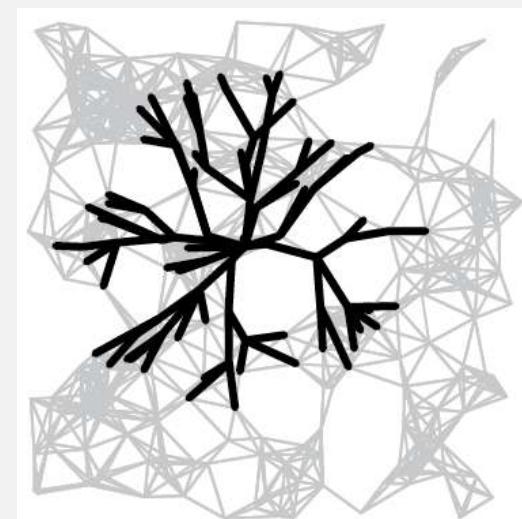
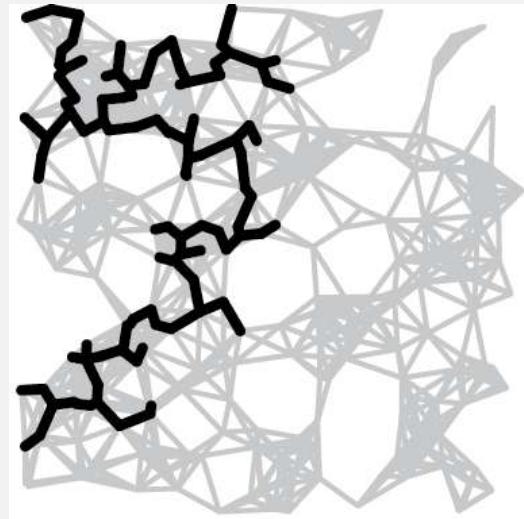
Computing spanning trees in graphs

Dijkstra's algorithm seem familiar?

- Prim's algorithm is essentially the same algorithm.
- Both are in a family of algorithms that compute a graph's spanning tree.

Main distinction: Rule used to choose next vertex for the tree.

- Prim's: Closest vertex to the **tree** (via an undirected edge).
- Dijkstra's: Closest vertex to the **source** (via a directed path).



Note: DFS and BFS are also in this family of algorithms.

Dijkstra's algorithm: which priority queue?

Depends on PQ implementation: V insert, V delete-min, E decrease-key.

PQ implementation	insert	delete-min	decrease-key	total
unordered array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap (Johnson 1975)	$d \log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap (Fredman-Tarjan 1984)	1^\dagger	$\log V^\dagger$	1^\dagger	$E + V \log V$

\dagger amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- d-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ *APIs*
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

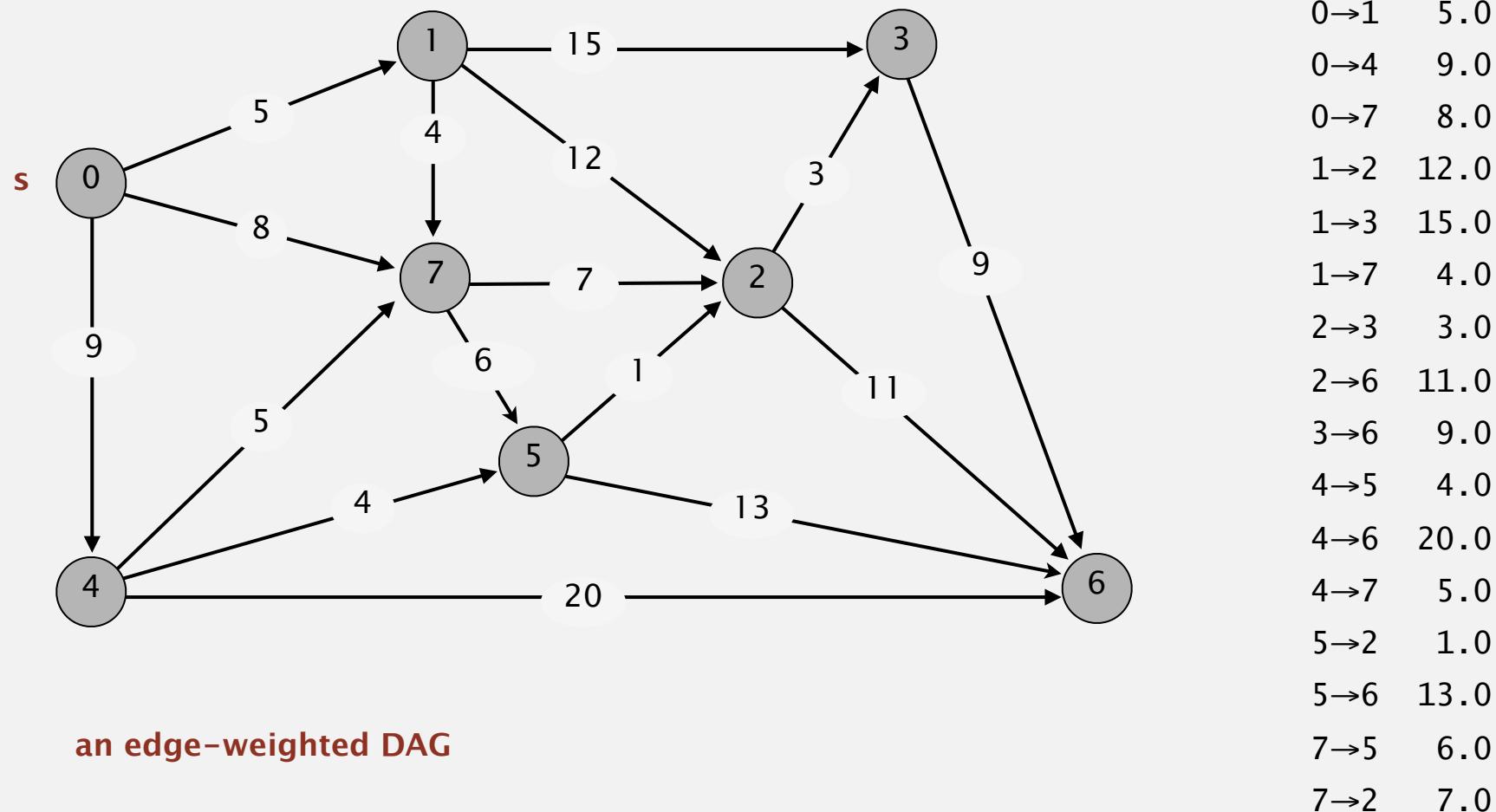
Acyclic edge-weighted digraphs

Q. Suppose that an edge-weighted digraph has no directed cycles.
Is it easier to find shortest paths than in a general digraph?

A. Yes!

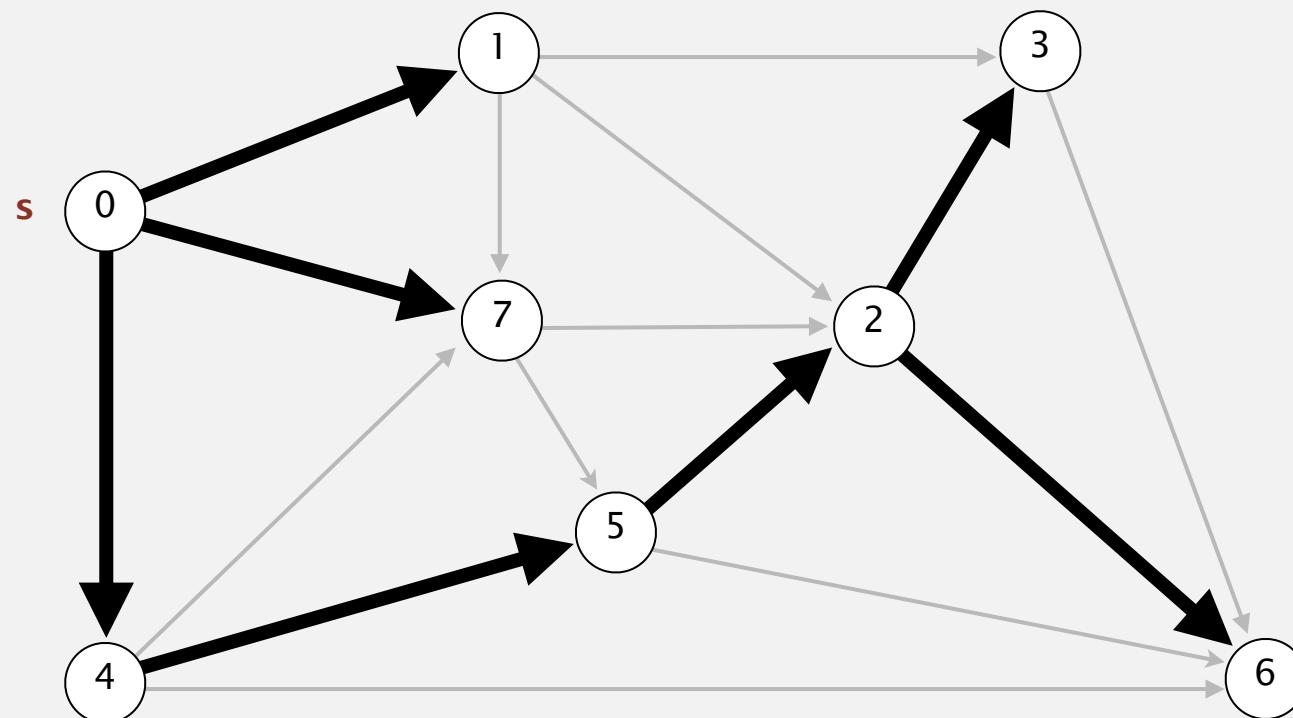
Acyclic shortest paths demo

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.



Acyclic shortest paths demo

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.



shortest-paths tree from vertex s

0	1	4	7	5	2	3	6
v	distTo[]	edgeTo[]					
0	0.0	-					
1	5.0	0→1					
2	14.0	5→2					
3	17.0	2→3					
4	9.0	0→4					
5	13.0	4→5					
6	25.0	2→6					
7	8.0	0→7					

Shortest paths in edge-weighted DAGs

Proposition. Topological sort algorithm computes SPT in any edge-weighted DAG in time proportional to $E + V$.

edge weights
can be negative!

Pf.

- Each edge $e = v \rightarrow w$ is relaxed exactly once (when v is relaxed), leaving $\text{distTo}[w] \leq \text{distTo}[v] + e.\text{weight}()$.
- Inequality holds until algorithm terminates because:
 - $\text{distTo}[w]$ cannot increase ← distTo[] values are monotone decreasing
 - $\text{distTo}[v]$ will not change ← because of topological order, no edge pointing to v will be relaxed after v is relaxed
- Thus, upon termination, shortest-paths optimality conditions hold. ■

Shortest paths in edge-weighted DAGs

```
public class AcyclicSP
{
    private DirectedEdge[] edgeTo;
    private double[] distTo;

    public AcyclicSP(EdgeWeightedDigraph G, int s)
    {
        edgeTo = new DirectedEdge[G.V()];
        distTo = new double[G.V()];

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        distTo[s] = 0.0;

        Topological topological = new Topological(G); ← topological order
        for (int v : topological.order())
            for (DirectedEdge e : G.adj(v))
                relax(e);
    }
}
```

Content-aware resizing

Seam carving. [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.



<http://www.youtube.com/watch?v=vIFCV2spKtg>

Content-aware resizing

Seam carving. [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.



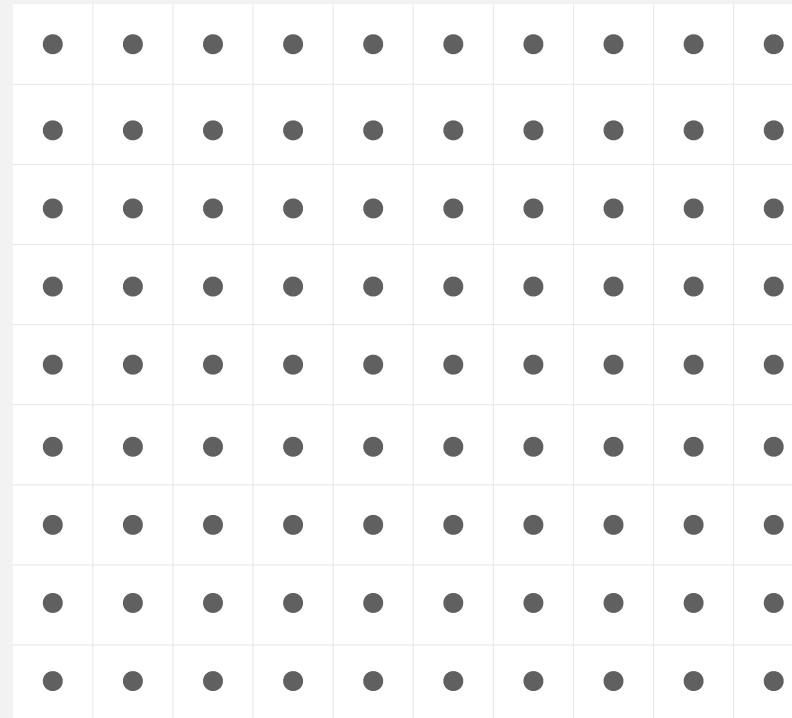
In the wild. Photoshop CS 5, Imagemagick, GIMP, ...



Content-aware resizing

To find vertical seam:

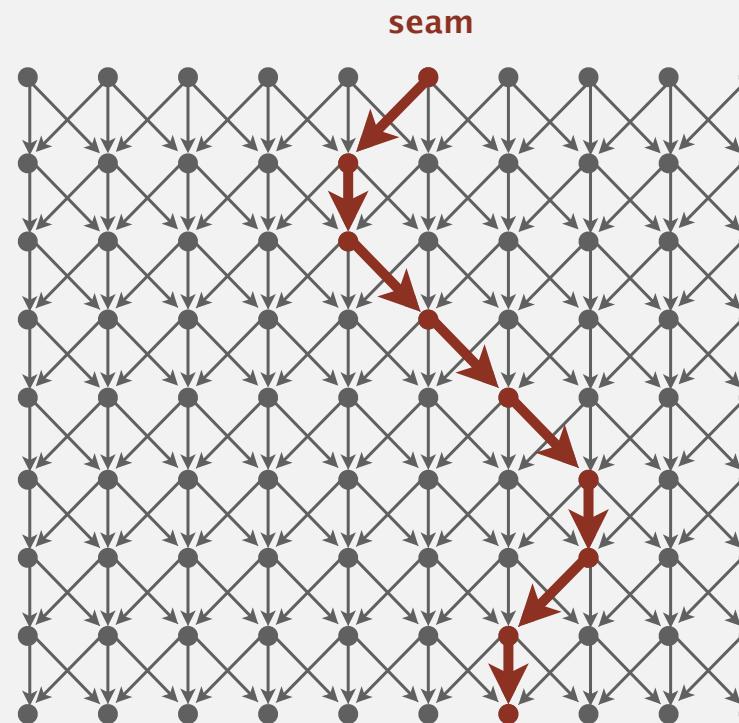
- Grid DAG: vertex = pixel; edge = from pixel to 3 downward neighbors.
- Weight of pixel = energy function of 8 neighboring pixels.
- Seam = shortest path (sum of vertex weights) from top to bottom.



Content-aware resizing

To find vertical seam:

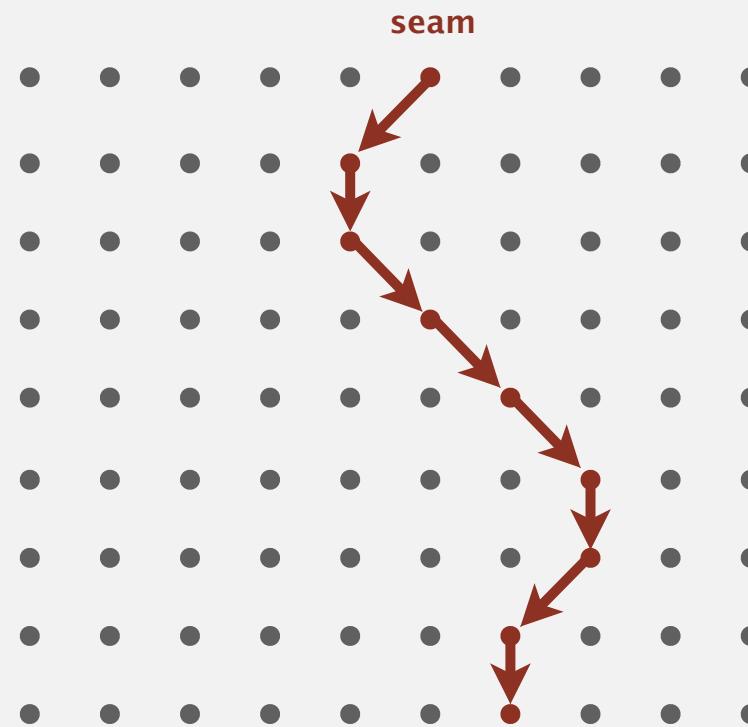
- Grid DAG: vertex = pixel; edge = from pixel to 3 downward neighbors.
- Weight of pixel = energy function of 8 neighboring pixels.
- Seam = shortest path (sum of vertex weights) from top to bottom.



Content-aware resizing

To remove vertical seam:

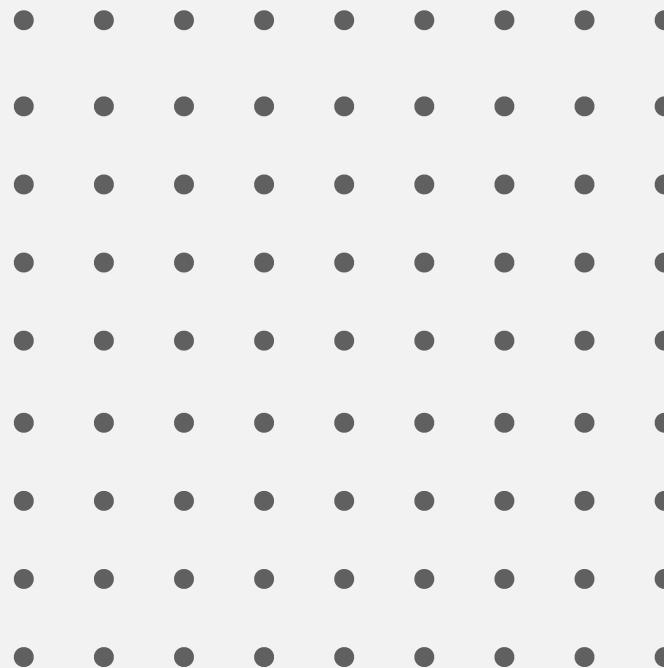
- Delete pixels on seam (one in each row).



Content-aware resizing

To remove vertical seam:

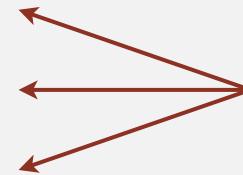
- Delete pixels on seam (one in each row).



Longest paths in edge-weighted DAGs

Formulate as a shortest paths problem in edge-weighted DAGs.

- Negate all weights.
- Find shortest paths.
- Negate weights in result.



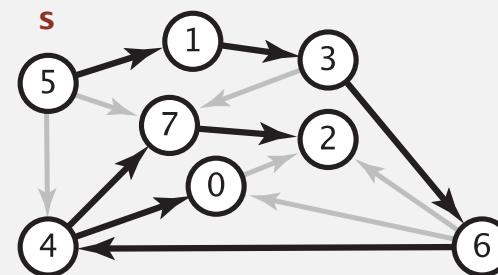
equivalent: reverse sense of equality in `relax()`

longest paths input

5->4	0.35
4->7	0.37
5->7	0.28
5->1	0.32
4->0	0.38
0->2	0.26
3->7	0.39
1->3	0.29
7->2	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93

shortest paths input

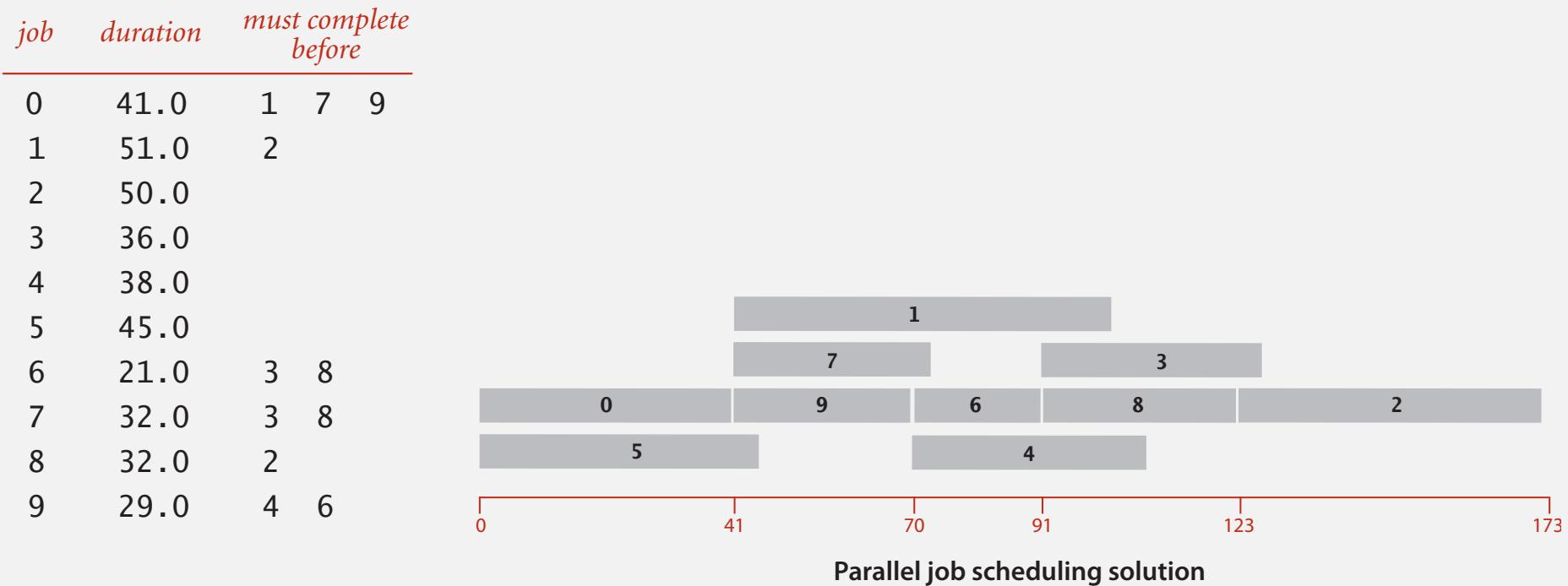
5->4	-0.35
4->7	-0.37
5->7	-0.28
5->1	-0.32
4->0	-0.38
0->2	-0.26
3->7	-0.39
1->3	-0.29
7->2	-0.34
6->2	-0.40
3->6	-0.52
6->0	-0.58
6->4	-0.93



Key point. Topological sort algorithm works even with negative weights.

Longest paths in edge-weighted DAGs: application

Parallel job scheduling. Given a set of jobs with durations and precedence constraints, schedule the jobs (by finding a start time for each) so as to achieve the minimum completion time, while respecting the constraints.

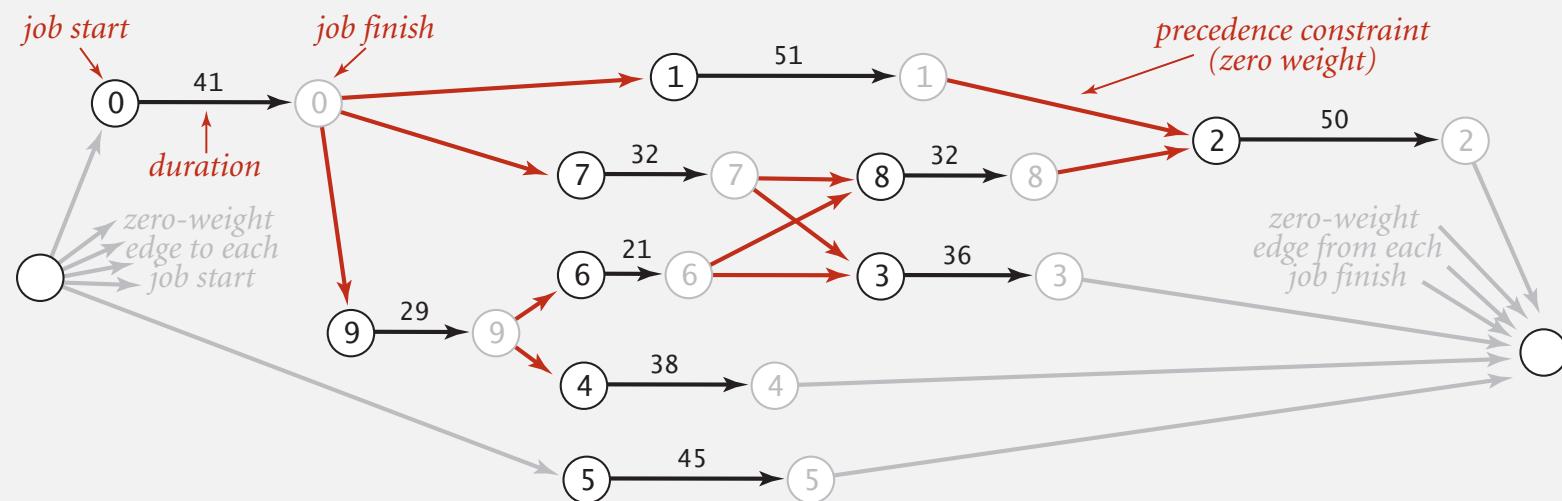


Critical path method

CPM. To solve a parallel job-scheduling problem, create edge-weighted DAG:

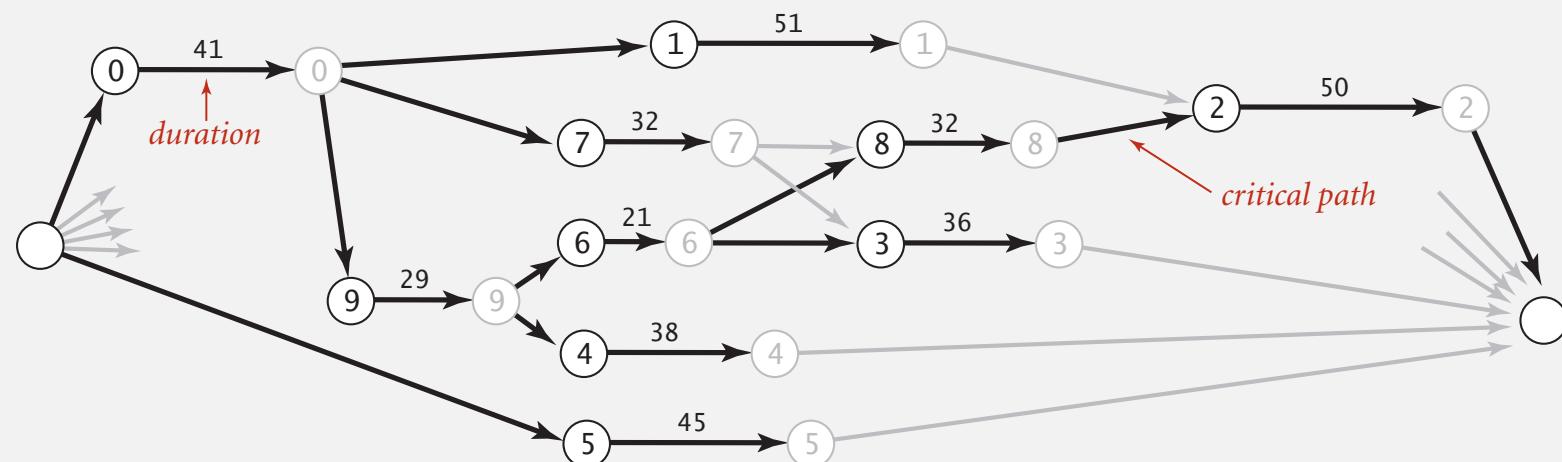
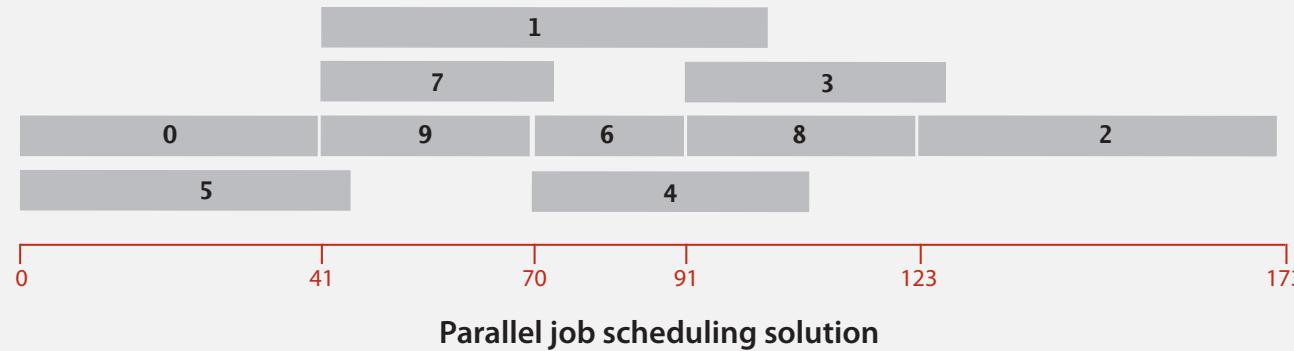
- Source and sink vertices.
- Two vertices (begin and end) for each job.
- Three edges for each job.
 - begin to end (weighted by duration)
 - source to begin (0 weight)
 - end to sink (0 weight)
- One edge for each precedence constraint (0 weight).

job	duration	must complete before		
0	41.0	1	7	9
1	51.0	2		
2	50.0			
3	36.0			
4	38.0			
5	45.0			
6	21.0	3	8	
7	32.0	3	8	
8	32.0	2		
9	29.0	4	6	



Critical path method

CPM. Use **longest path** from the source to schedule each job.



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ *APIs*
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

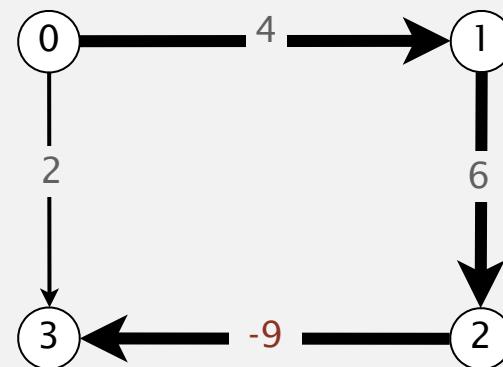
<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ *APIs*
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ ***negative weights***

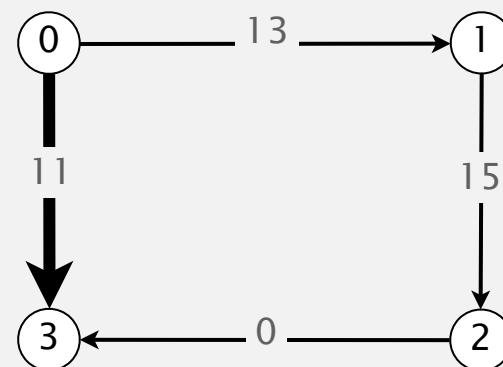
Shortest paths with negative weights: failed attempts

Dijkstra. Doesn't work with negative edge weights.



Dijkstra selects vertex 3 immediately after 0.
But shortest path from 0 to 3 is $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$.

Re-weighting. Add a constant to every edge weight doesn't work.



Adding 9 to each edge weight changes the shortest path from $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ to $0 \rightarrow 3$.

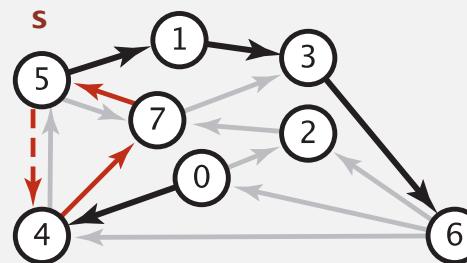
Conclusion. Need a different algorithm.

Negative cycles

Def. A negative cycle is a directed cycle whose sum of edge weights is negative.

digraph

4->5	0.35
5->4	-0.66
4->7	0.37
5->7	0.28
7->5	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
6->4	0.93



negative cycle (-0.66 + 0.37 + 0.28)

5->4->7->5

shortest path from 0 to 6

0->4->7->5->4->7->5...->1->3->6

Proposition. A SPT exists iff no negative cycles.

assuming all vertices reachable from s

Bellman-Ford algorithm

Bellman-Ford algorithm

Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.

Repeat V times:

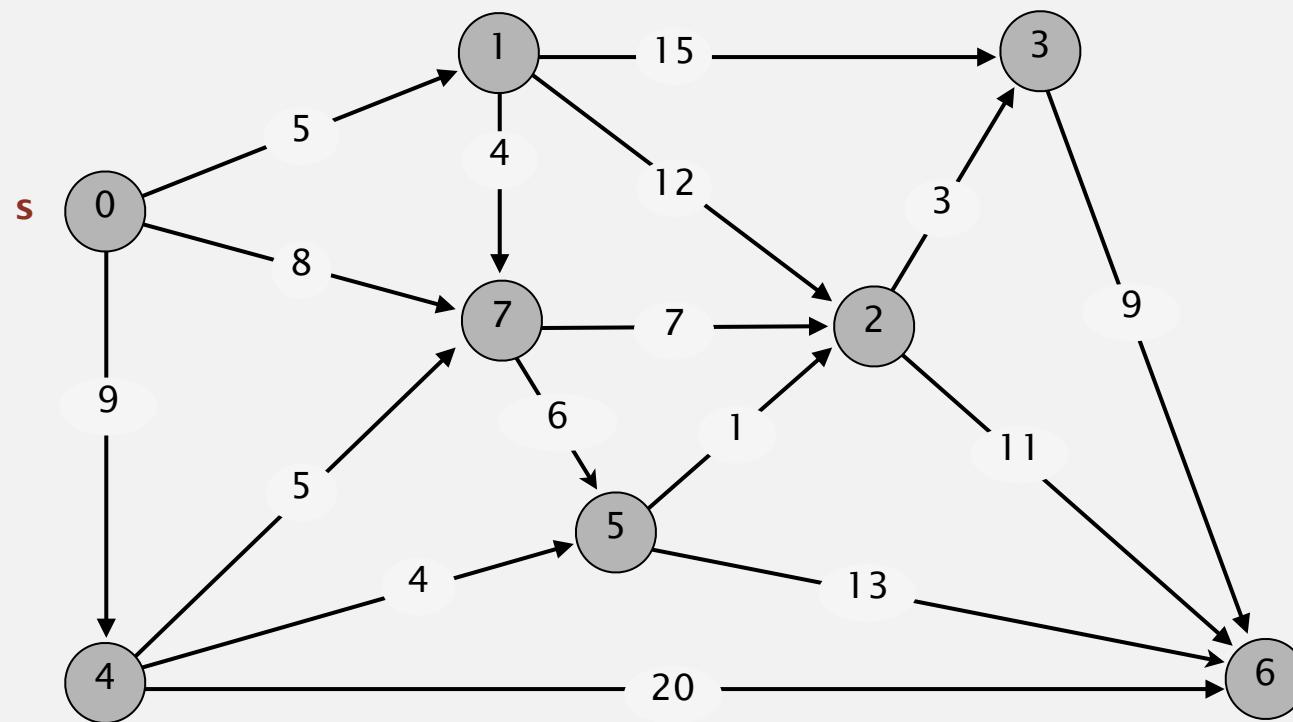
- Relax each edge.

```
for (int i = 0; i < G.V(); i++)
    for (int v = 0; v < G.V(); v++)
        for (DirectedEdge e : G.adj(v))
            relax(e);
```

pass i (relax each edge)

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.

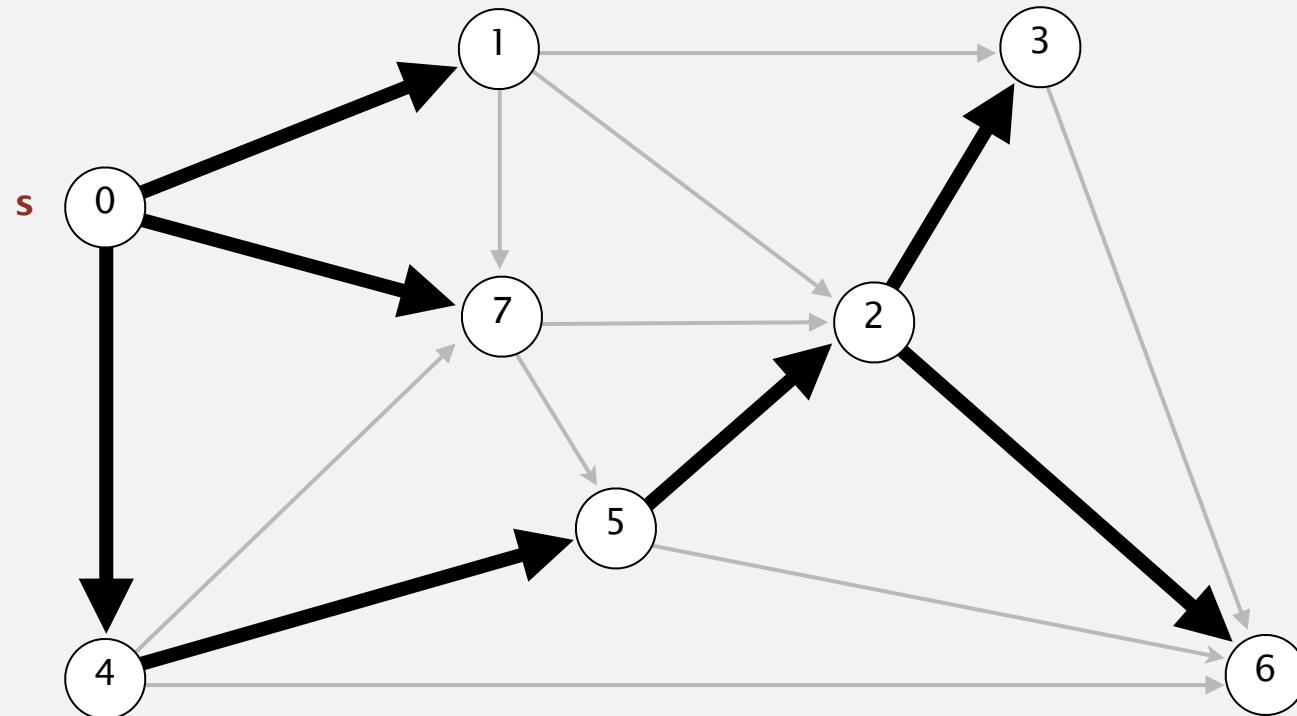


an edge-weighted digraph

0→1	5.0
0→4	9.0
0→7	8.0
1→2	12.0
1→3	15.0
1→7	4.0
2→3	3.0
2→6	11.0
3→6	9.0
4→5	4.0
4→6	20.0
4→7	5.0
5→2	1.0
5→6	13.0
5→7	6.0
6→2	7.0

Bellman-Ford algorithm demo

Repeat V times: relax all E edges.



shortest-paths tree from vertex s

v	distTo[]	edgeTo[]
0	0.0	-
1	5.0	0→1
2	14.0	5→2
3	17.0	2→3
4	9.0	0→4
5	13.0	4→5
6	25.0	2→6
7	8.0	0→7

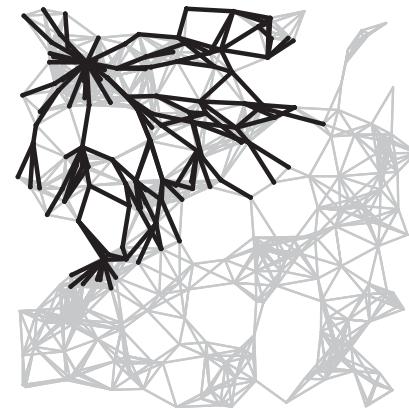
Bellman-Ford algorithm visualization

passes

4



7



10



13



SPT



Bellman-Ford algorithm: analysis

Bellman-Ford algorithm

Initialize $\text{distTo}[s] = 0$ and $\text{distTo}[v] = \infty$ for all other vertices.

Repeat V times:

- Relax each edge.**
-

Proposition. Dynamic programming algorithm computes SPT in any edge-weighted digraph with no negative cycles in time proportional to $E \times V$.

Pf idea. After pass i , found shortest path containing at most i edges.

Bellman-Ford algorithm: practical improvement

Observation. If $\text{distTo}[v]$ does not change during pass i , no need to relax any edge pointing from v in pass $i+1$.

FIFO implementation. Maintain **queue** of vertices whose $\text{distTo}[]$ changed.

↑
be careful to keep at most one copy
of each vertex on queue (why?)

Overall effect.

- The running time is still proportional to $E \times V$ in worst case.
- But much faster than that in practice.

Single source shortest-paths implementation: cost summary

algorithm	restriction	typical case	worst case	extra space
topological sort	no directed cycles	$E + V$	$E + V$	V
Dijkstra (binary heap)	no negative weights	$E \log V$	$E \log V$	V
Bellman-Ford	no negative cycles	$E V$	$E V$	V
Bellman-Ford (queue-based)		$E + V$	$E V$	V

Remark 1. Directed cycles make the problem harder.

Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

Finding a negative cycle

Negative cycle. Add two method to the API for SP.

boolean hasNegativeCycle()

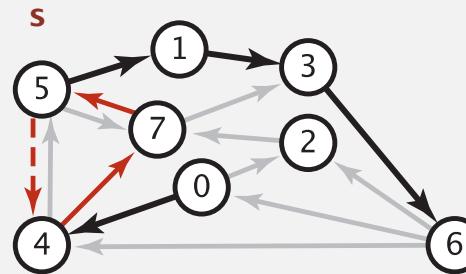
is there a negative cycle?

Iterable <DirectedEdge> negativeCycle()

negative cycle reachable from s

digraph

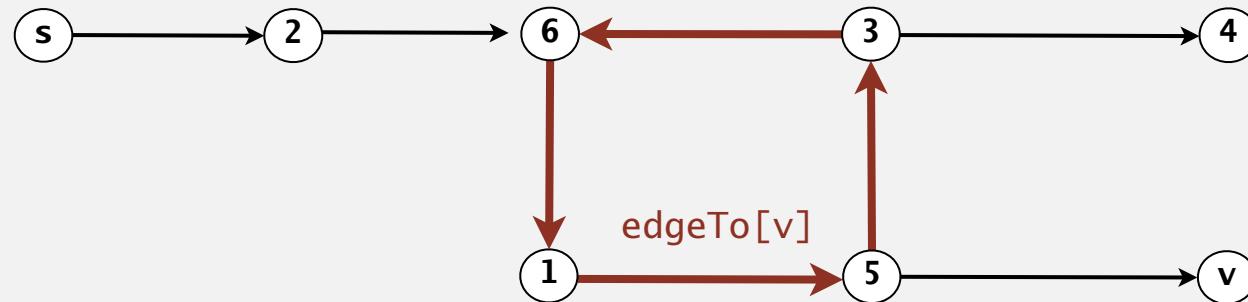
```
4->5  0.35
5->4 -0.66
4->7  0.37
5->7  0.28
7->5  0.28
5->1  0.32
0->4  0.38
0->2  0.26
7->3  0.39
1->3  0.29
2->7  0.34
6->2  0.40
3->6  0.52
6->0  0.58
6->4  0.93
```



negative cycle (-0.66 + 0.37 + 0.28)
5->4->7->5

Finding a negative cycle

Observation. If there is a negative cycle, Bellman-Ford gets stuck in loop, updating `distTo[]` and `edgeTo[]` entries of vertices in the cycle.



Proposition. If any vertex v is updated in phase v , there exists a negative cycle (and can trace back `edgeTo[v]` entries to find it).

In practice. Check for negative cycles more frequently.

Negative cycle application: arbitrage detection

Problem. Given table of exchange rates, is there an arbitrage opportunity?

	USD	EUR	GBP	CHF	CAD
USD	1	0.741	0.657	1.061	1.011
EUR	1.350	1	0.888	1.433	1.366
GBP	1.521	1.126	1	1.614	1.538
CHF	0.943	0.698	0.620	1	0.953
CAD	0.995	0.732	0.650	1.049	1

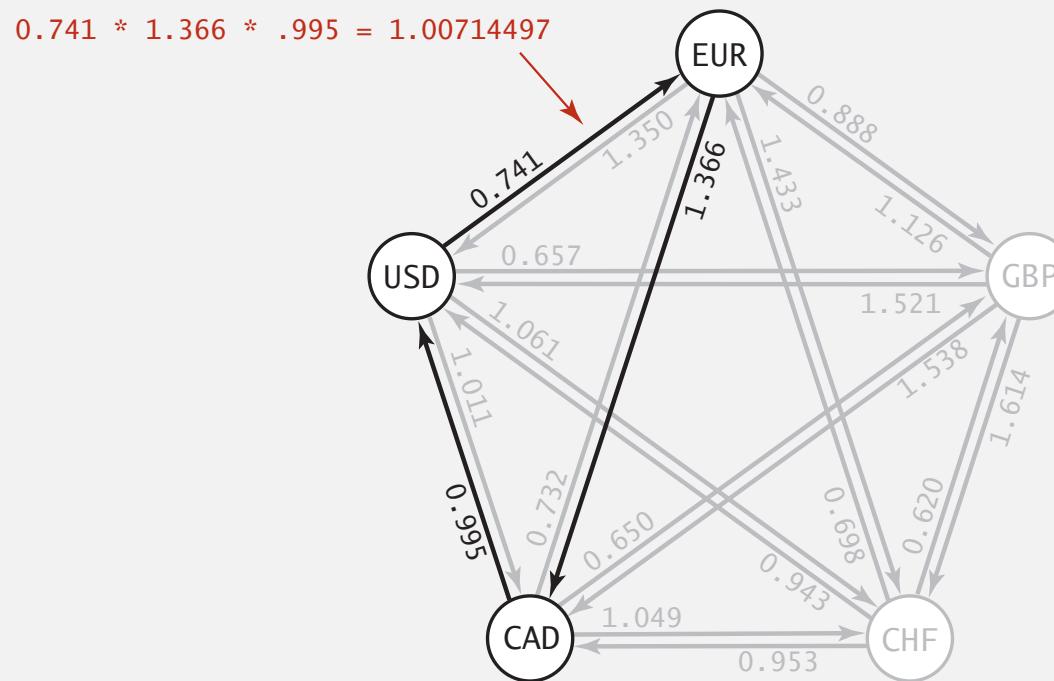
Ex. \$1,000 \Rightarrow 741 Euros \Rightarrow 1,012.206 Canadian dollars \Rightarrow \$1,007.14497.

$$1000 \times 0.741 \times 1.366 \times 0.995 = 1007.14497$$

Negative cycle application: arbitrage detection

Currency exchange graph.

- Vertex = currency.
- Edge = transaction, with weight equal to exchange rate.
- Find a directed cycle whose product of edge weights is > 1 .

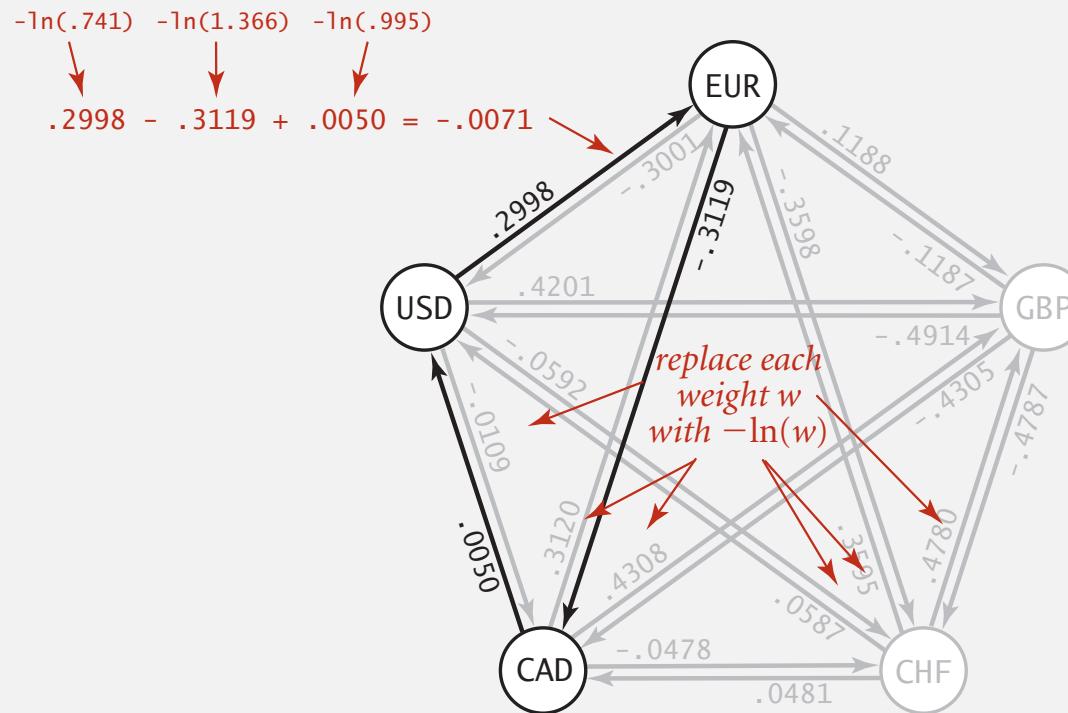


Challenge. Express as a negative cycle detection problem.

Negative cycle application: arbitrage detection

Model as a negative cycle detection problem by taking logs.

- Let weight of edge $v \rightarrow w$ be $-\ln$ (exchange rate from currency v to w).
- Multiplication turns to addition; > 1 turns to < 0 .
- Find a directed cycle whose sum of edge weights is < 0 (negative cycle).



Remark. Fastest algorithm is extraordinarily valuable!

Shortest paths summary

Dijkstra's algorithm.

- Nearly linear-time when weights are nonnegative.
- Generalization encompasses DFS, BFS, and Prim.

Acyclic edge-weighted digraphs.

- Arise in applications.
- Faster than Dijkstra's algorithm.
- Negative weights are no problem.

Negative weights and negative cycles.

- Arise in applications.
- If no negative cycles, can find shortest paths via Bellman-Ford.
- If negative cycles, can find one via Bellman-Ford.

Shortest-paths is a broadly useful problem-solving model.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ *APIs*
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ ***negative weights***



<http://algs4.cs.princeton.edu>

4.4 SHORTEST PATHS

- ▶ APIs
- ▶ *shortest-paths properties*
- ▶ *Dijkstra's algorithm*
- ▶ *edge-weighted DAGs*
- ▶ *negative weights*



<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *REs and NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

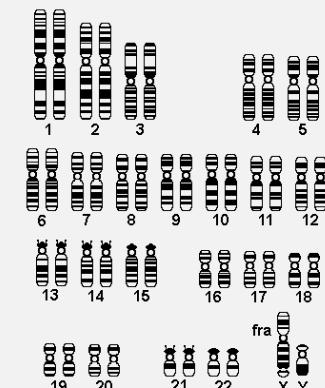
Pattern matching

Substring search. Find a single string in text.

Pattern matching. Find one of a **specified set** of strings in text.

Ex. [genomics]

- Fragile X syndrome is a common cause of mental retardation.
- A human's genome is a string.
- It contains triplet repeats of CGG or AGG, bracketed by CCG at the beginning and CTG at the end.
- Number of repeats is variable and is correlated to syndrome.



pattern **GCG(CGG|AGG)*CTG**

text **GC GG CG TG TG CG AG AG AG TG GG TT AA AG CT G**GC GCG AGG CGG CT G**GC GCG AGG CT G**

Syntax highlighting

```
/*
 * Compilation: javac NFA.java
 * Execution:   java NFA regexp text
 * Dependencies: Stack.java Bag.java Digraph.java DirectedDFS.java
 *
 * % java NFA "(A*B|AC)D" AAAABD
 * true
 *
 * % java NFA "(A*B|AC)D" AAAAC
 * false
 */
public class NFA {
    private Digraph G;           // digraph of epsilon transitions
    private String regexp;       // regular expression
    private int M;               // number of characters in regular expression

    // Create the NFA for the given RE
    public NFA(String regexp) {
        this.regexp = regexp;
        M = regexp.length();
        Stack<Integer> ops = new Stack<Integer>();
        G = new Digraph(M+1);
    }
}
```

input	output
Ada	HTML
Asm	XHTML
Applescript	LATEX
Awk	MediaWiki
Bat	ODF
Bib	TEXINFO
Bison	ANSI
C/C++	DocBook
C#	
Cobol	
Caml	
Changelog	
Css	
D	
Erlang	
Flex	
Fortran	
GLSL	
Haskell	
Html	
Java	
Javalog	
Javascript	
Latex	
Lisp	
Lua	
:	

Google code search

Search public source code

Search Code

Search via regular expression, e.g. ^java/.*\\.java\$

Search Options	In Search Box
Package	package:linux-2.6
Language	lang:c++
File Path	file:(code [^or]g)search
Class	class:HashMap
Function	function:toString
License	license:mozilla
Case Sensitive	case:yes

<http://code.google.com/p/chromium/source/search>

Pattern matching: applications

Test if a string matches some pattern.

- Scan for virus signatures.
- Process natural language.
- Specify a programming language.
- Access information in digital libraries.
- Search genome using PROSITE patterns.
- Filter text (spam, NetNanny, Carnivore, malware).
- Validate data-entry fields (dates, email, URL, credit card).

...



Form Validation

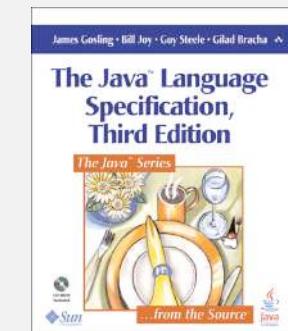
First name:	<input type="text"/>
Last name:	<input type="text"/>
Username:	<input type="text"/>
E-mail:	<input type="text"/>
Password:	<input type="text"/>
Phone:	<input type="text"/>
Date:	<input type="text"/>
Address:	<input type="text"/> Some thing

Powered by QuiteWeb.com ©

Parse text files.

- Compile a Java program.
- Crawl and index the Web.
- Read in data stored in ad hoc input file format.
- Create Java documentation from Javadoc comments.

...



Regular expressions

A **regular expression** is a notation to specify a set of strings.

↑
possibly infinite

operation	order	example RE	matches	does not match
concatenation	3	AABAAB	AABAAB	every other string
or	4	AA BAAB	AA BAAB	every other string
closure	2	AB*A	AA BBBBBBBBBA	AB ABABA
parentheses	1	A(A B)AAB	AAAAB ABAAB	every other string
		(AB)*A	A ABABABABABA	AA ABBA

Regular expression shortcuts

Additional operations are often added for convenience.

operation	example RE	matches	does not match
wildcard	.U.U.U.	CUMULUS JUGULUM	SUCCUBUS TUMULTUOUS
character class	[A-Za-z] [a-z]*	word Capitalized	camelCase 4illegal
at least 1	A(BC)+DE	ABCDE ABCBCDE	ADE BCDE
exactly k	[0-9]{5}-[0-9]{4}	08540-1321 19072-5541	111111111 166-54-111

Ex. $[A-E]^+$ is shorthand for $(A|B|C|D|E)(A|B|C|D|E)^*$

Regular expression examples

RE notation is surprisingly expressive.

regular expression	matches	does not match
$\cdot^* \text{SPB} \cdot^*$ <i>(substring search)</i>	RASPBERRY CRISPBREAD	SUBSPACE SUBSPECIES
$[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}$ <i>(U. S. Social Security numbers)</i>	166-11-4433 166-45-1111	11-55555555 8675309
$[\text{a-z}]^+ @ ([\text{a-z}]^+ \cdot) + (\text{edu} \text{com})$ <i>(simplified email addresses)</i>	wayne@princeton.edu rs@princeton.edu	spam@nowhere
$[\$\text{A-Z}\text{a-z}] [\$\text{A-Z}\text{a-z}0-9]^*$ <i>(Java identifiers)</i>	ident3 PatternMatcher	3a ident#3

REs play a well-understood role in the theory of computation.

Illegally screening a job candidate

“ [First name]! and pre/2 [last name] w/7
bush or gore or republican! or democrat! or charg!
or accus! or criticiz! or blam! or defend! or iran contra
or clinton or spotted owl or florida recount or sex!
or controvers! or fraud! or investigat! or bankrupt!
or layoff! or downsiz! or PNTR or NAFTA or outsourc!
or indict! or enron or kerry or iraq or wmd! or arrest!
or intox! or fired or racis! or intox! or slur!
or controvers! or abortion! or gay! or homosexual!
or gun! or firearm! ”

*— LexisNexis search string used by Monica Goodling
to illegally screen candidates for DOJ positions*



LexisNexis™

<http://www.justice.gov/oig/special/s0807/final.pdf>

Can the average web surfer learn to use REs?

Google. Supports * for full word wildcard and | for union.

The screenshot shows a Mozilla browser window with the title "Google Search: 'the * of seville' - Mozilla". The search bar contains the query "the * of seville". The results page is titled "Web" and shows 10 results out of approximately 60,100. The results include links to news articles about the Barber of Seville opera, information about the City of Sevilla, the Universidad de Sevilla website, and the Catholic Encyclopedia entry for St. Isidore of Seville. Each result includes a snippet of text and links to the full page, a cached version, and similar pages.

Results 1 - 10 of about 60,100 for "[the * of seville](#)". (0.31 seconds)

[News results for "the * of seville"](#) - View all the latest headlines

[Opera: Barber of Seville/ Marriage of Figaro](#) - Financial Times - 3 hours ago

[Information about the City of Sevilla \(Seville\), Andalucía ...](#)
... Post a request on our Notice Board. Promote your business on this website;
email sales@andalucia.com. Information about **the City of Seville**. ...
[www.andalucia.com/cities/sevilla.htm](#) - 22k - [Cached](#) - [Similar pages](#)

[Universidad de Sevilla](#) - [[Translate this page](#)]
INICIO | ESTUDIANTES | PROFESORES | PAS | INDICES | BUSCADOR | COMENTARIOS,
Complemento Autonómico, Estatuto, Espacio Europeo de Educación ...
[www.us.es/](#) - 15k - Apr 18, 2004 - [Cached](#) - [Similar pages](#)

[CATHOLIC ENCYCLOPEDIA: St. Isidore of Seville](#)
... On the death of Leander, Isidore succeeded to **the See of Seville**. His long incumbency
to this office was spent in a period of disintegration and transition. ...
[www.newadvent.org/cathen/08186a.htm](#) - 32k - [Cached](#) - [Similar pages](#)

[The Trickster of Seville and the Stone Guest](#)
Commentary and analysis of Tirso de Molina's "**The Trickster of Seville**", one of the seventeenth century's...
[www.modlang.fsu.edu/darst/trickster.htm](#) - [Similar pages](#)

Regular expressions to the rescue



<http://xkcd.com/208>

Can the average programmer learn to use REs?

Perl RE for valid RFC822 email addresses

Regular expression caveat

Writing a RE is like writing a program.

- Need to understand programming model.
- Can be easier to write than read.
- Can be difficult to debug.



“Some people, when confronted with a problem, think 'I know I'll use regular expressions.' Now they have two problems.”

— Jamie Zawinski (flame war on alt.religion.emacs)

Bottom line. REs are amazingly powerful and expressive, but using them in applications can be amazingly complex and error-prone.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ **NFAs**
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Duality between REs and DFAs

RE. Concise way to describe a set of strings.

DFA. Machine to recognize whether a given string is in a given set.

Kleene's theorem.

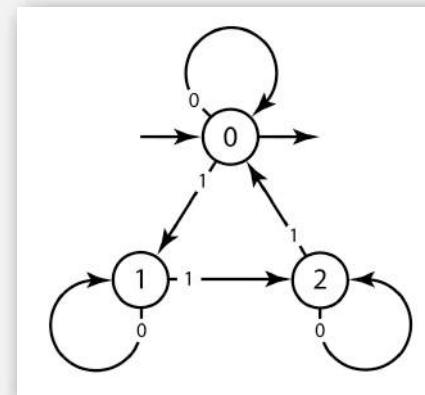
- For any DFA, there exists a RE that describes the same set of strings.
- For any RE, there exists a DFA that recognizes the same set of strings.

RE

$0^* \mid (0^*10^*10^*)^*$

number of 1's is a multiple of 3

DFA



number of 1's is a multiple of 3

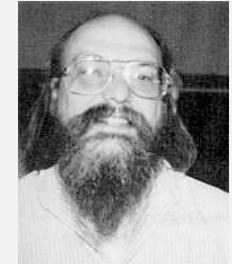


Stephen Kleene
Princeton Ph.D. 1934

Pattern matching implementation: basic plan (first attempt)

Overview is the same as for KMP.

- No backup in text input stream.
- Linear-time guarantee.

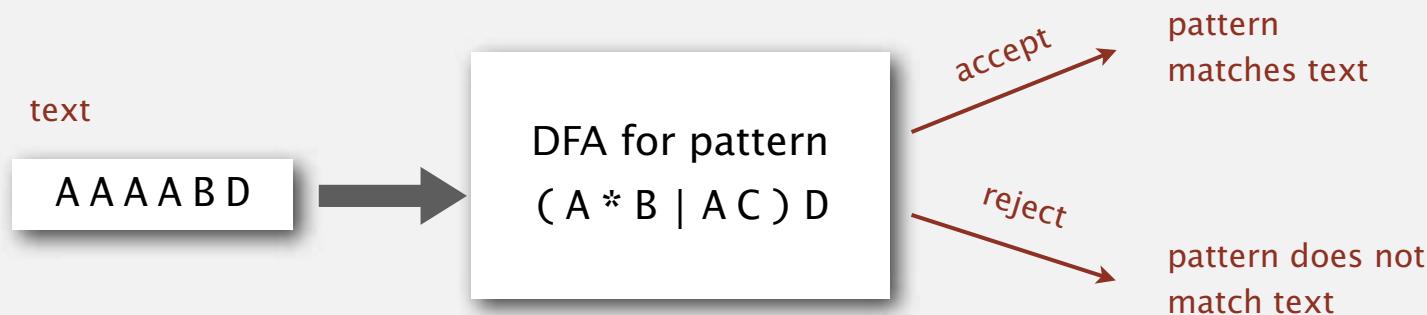


Ken Thompson
Turing Award '83

Underlying abstraction. Deterministic finite state automata (DFA).

Basic plan. [apply Kleene's theorem]

- Build DFA from RE.
- Simulate DFA with text as input.

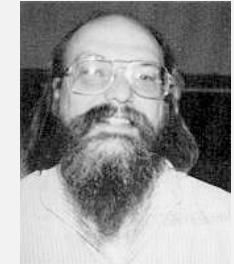


Bad news. Basic plan is infeasible (DFA may have exponential # of states).

Pattern matching implementation: basic plan (revised)

Overview is similar to KMP.

- No backup in text input stream.
- Quadratic-time guarantee (linear-time typical).

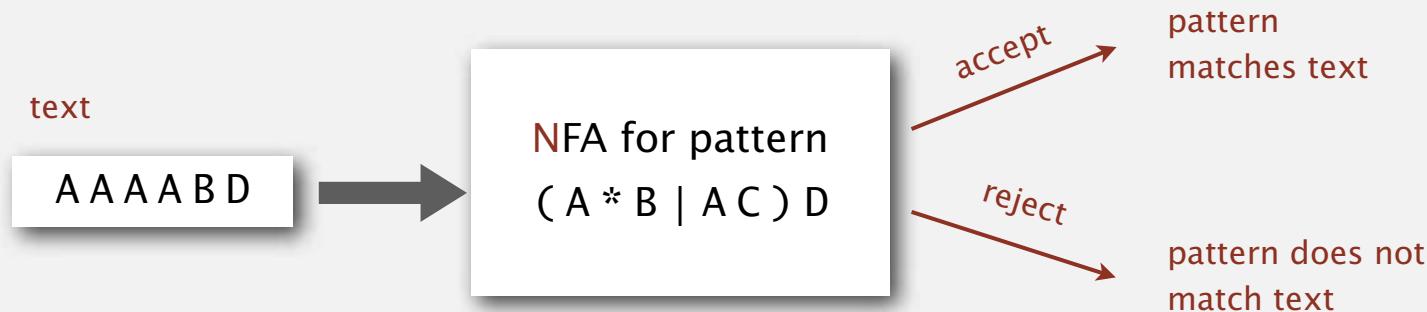


Ken Thompson
Turing Award '83

Underlying abstraction. Nondeterministic finite state automata (NFA).

Basic plan. [apply Kleene's theorem]

- Build NFA from RE.
- Simulate NFA with text as input.



Q. What is an NFA?

Nondeterministic finite-state automata

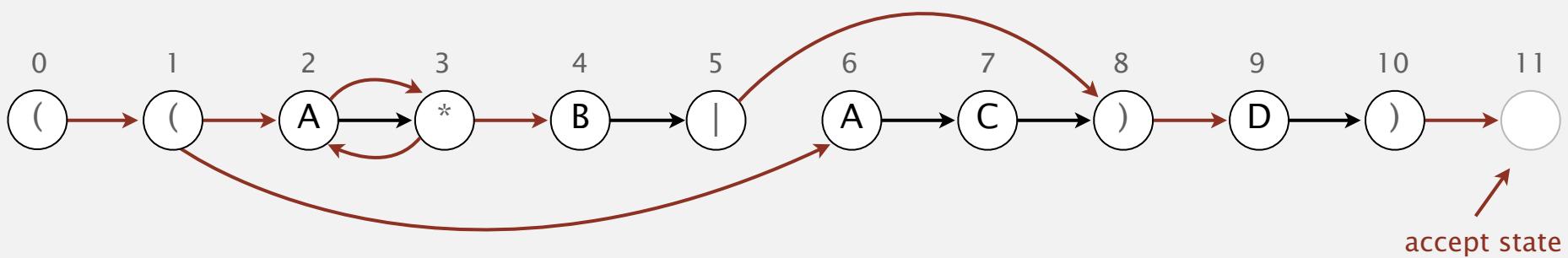
Regular-expression-matching NFA.

- RE enclosed in parentheses.
- One state per RE character (start = 0, accept = M).
- Red ϵ -transition (change state, but don't scan text).
- Black match transition (change state and scan to next text char).
- Accept if **any** sequence of transitions ends in accept state.

after scanning all text characters

Nondeterminism.

- One view: machine can guess the proper sequence of state transitions.
- Another view: sequence is a proof that the machine accepts the text.

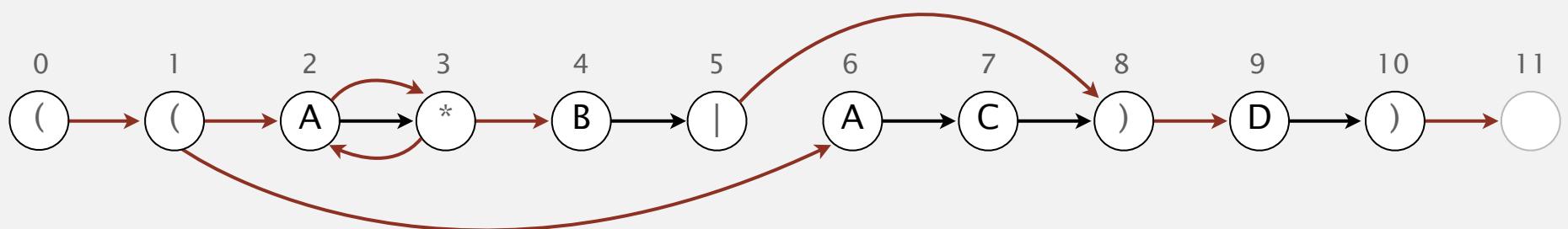
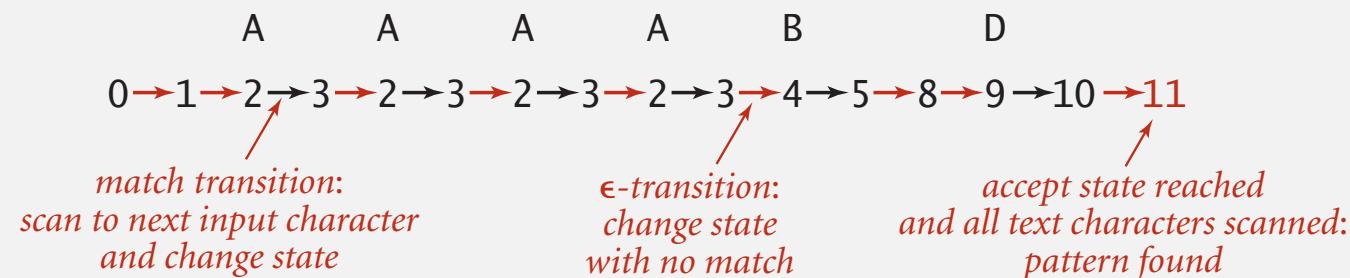


NFA corresponding to the pattern $((A^*B|AC)D)$

Nondeterministic finite-state automata

Q. Is AAAABD matched by NFA?

A. Yes, because **some** sequence of legal transitions ends in state 11.

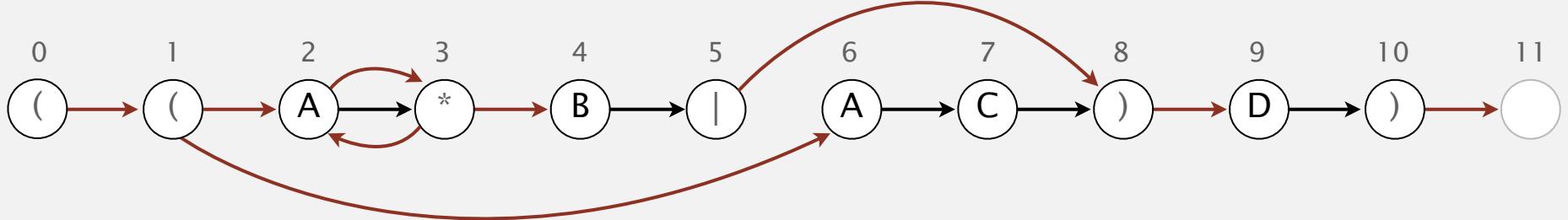
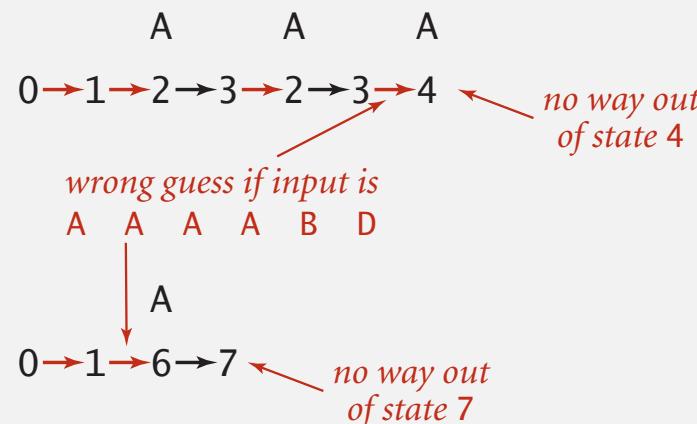


NFA corresponding to the pattern $((A^* B \mid A C) D)$

Nondeterministic finite-state automata

Q. Is AAAABD matched by NFA?

A. Yes, because **some** sequence of legal transitions ends in state 11.
[even though some sequences end in wrong state or stall]

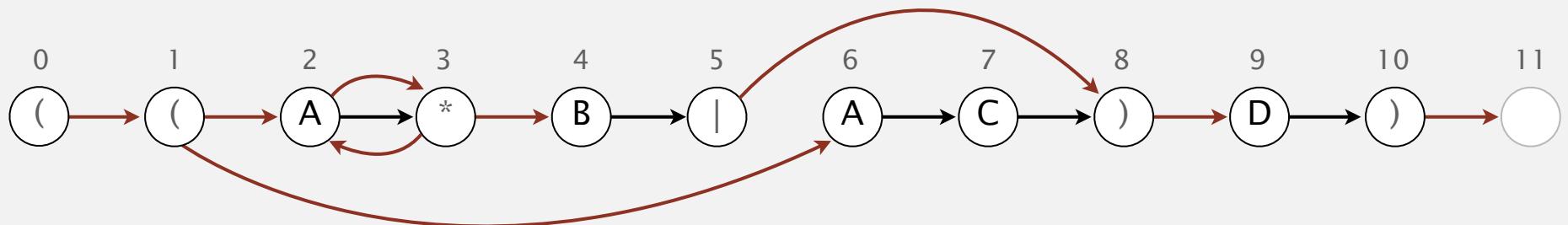
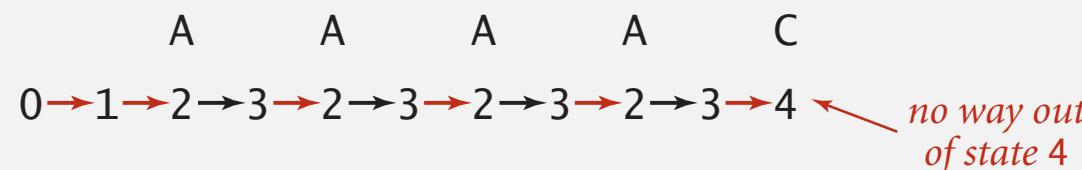


NFA corresponding to the pattern $((A^* B \mid A C) D)$

Nondeterministic finite-state automata

Q. Is AAAC matched by NFA?

A. No, because no sequence of legal transitions ends in state 11.
[but need to argue about all possible sequences]



NFA corresponding to the pattern $((A^* B \mid A C) D)$

Nondeterminism

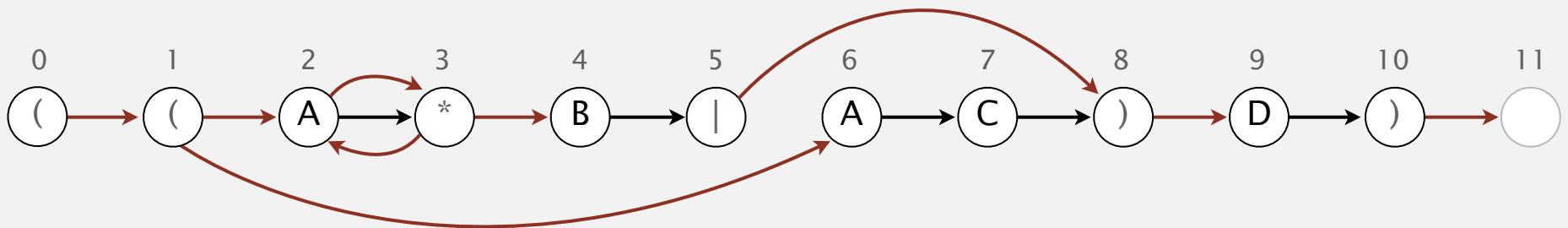
Q. How to determine whether a string is matched by an automaton?

DFA. Deterministic \Rightarrow easy because exactly one applicable transition.

NFA. Nondeterministic \Rightarrow can be several applicable transitions;
need to select the right one!

Q. How to simulate NFA?

A. Systematically consider all possible transition sequences.



NFA corresponding to the pattern $((A^* B \mid A C) D)$

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ **NFAs**
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

NFA representation

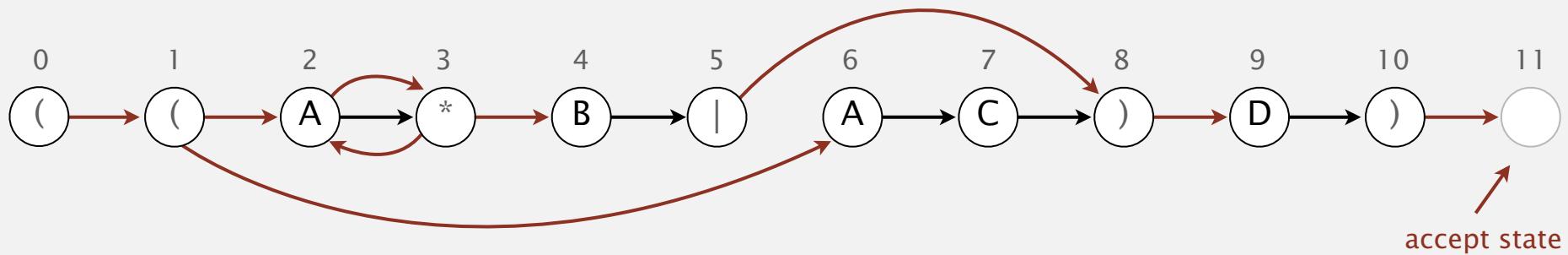
State names. Integers from 0 to M .

number of symbols in RE

Match-transitions. Keep regular expression in array `re[]`.

ϵ -transitions. Store in a digraph G .

$0 \rightarrow 1, 1 \rightarrow 2, 1 \rightarrow 6, 2 \rightarrow 3, 3 \rightarrow 2, 3 \rightarrow 4, 5 \rightarrow 8, 8 \rightarrow 9, 10 \rightarrow 11$

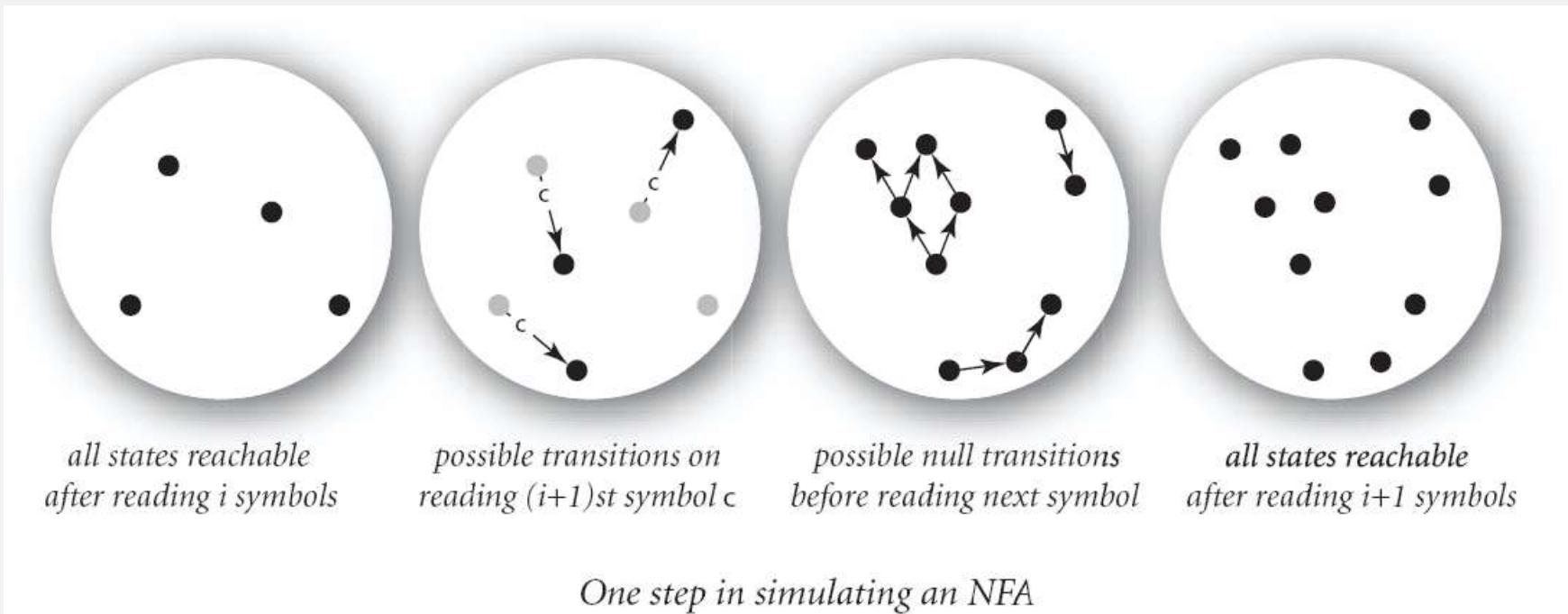


NFA corresponding to the pattern $((A^* B \mid A C) D)$

NFA simulation

Q. How to efficiently simulate an NFA?

A. Maintain set of **all** possible states that NFA could be in after reading in the first i text characters.



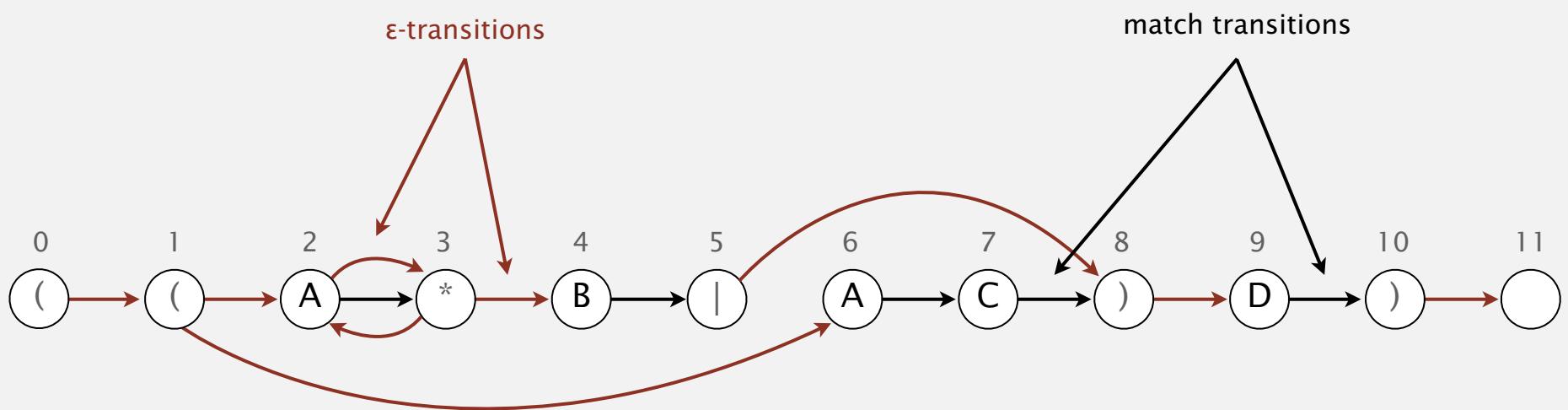
Q. How to perform reachability?

NFA simulation demo

Goal. Check whether input matches pattern.



input A A B D

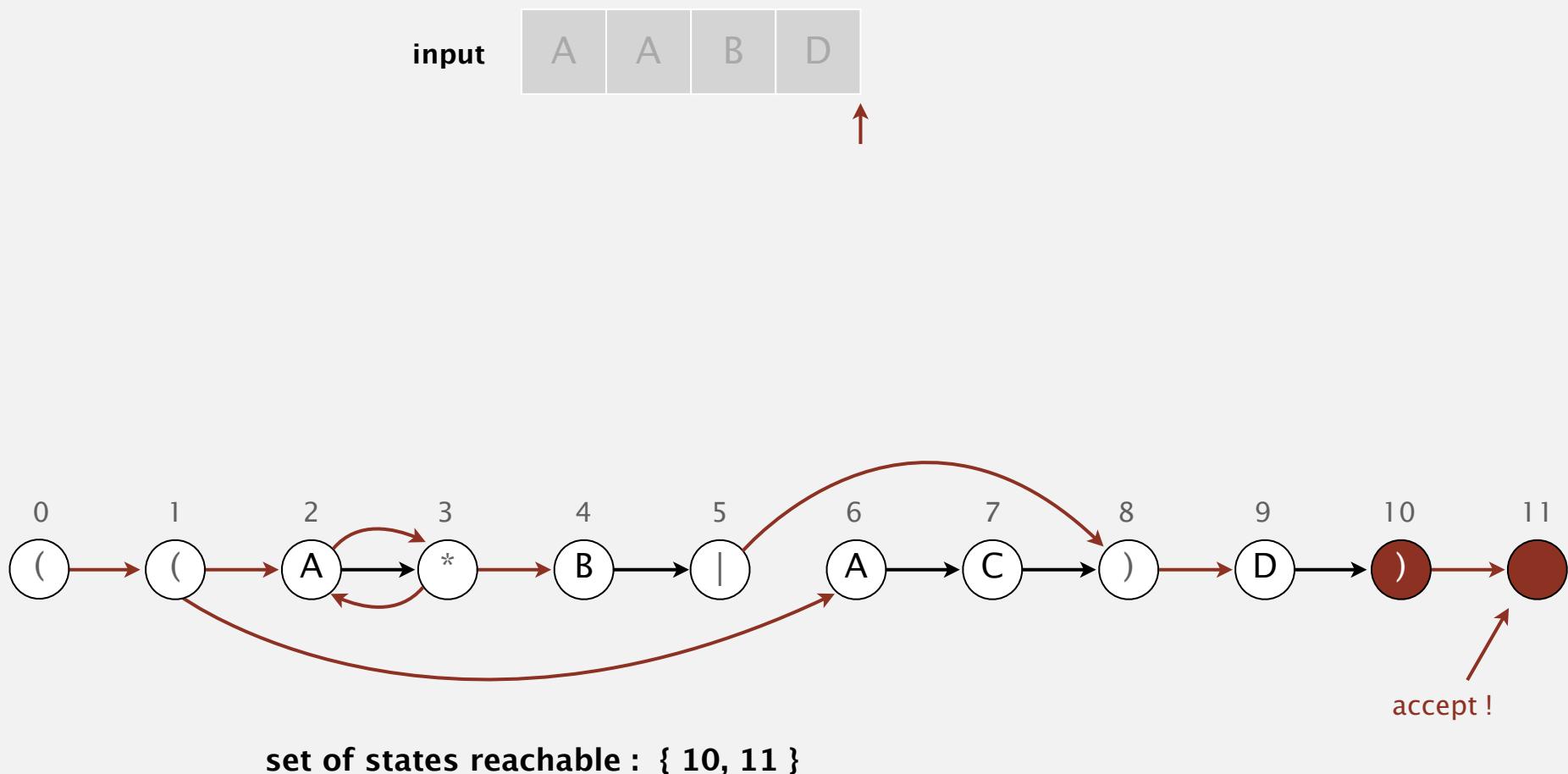


NFA corresponding to the pattern $((A^* B \mid A C) D)$

NFA simulation demo

When no more input characters:

- Accept if any state reachable is an accept state.
- Reject otherwise.



Digraph reachability

Digraph reachability. Find all vertices reachable from a given source or **set** of vertices.

recall Section 4.2

```
public class DirectedDFS
```

```
    DirectedDFS(Digraph G, int s)
```

find vertices reachable from s

```
    DirectedDFS(Digraph G, Iterable<Integer> s)
```

find vertices reachable from sources

```
    boolean marked(int v)
```

is v reachable from source(s)?

Solution. Run DFS from each source, without unmarking vertices.

Performance. Runs in time proportional to $E + V$.

NFA simulation: Java implementation

```
public class NFA
{
    private char[] re;          // match transitions
    private Digraph G;          // epsilon transition digraph
    private int M;              // number of states

    public NFA(String regexp)
    {
        M = regexp.length();
        re = regexp.toCharArray();
        G = buildEpsilonTransitionsDigraph();           ← stay tuned (next segment)
    }

    public boolean recognizes(String txt)
    { /* see next slide */ }

    public Digraph buildEpsilonTransitionDigraph()
    { /* stay tuned */ }

}
```

NFA simulation: Java implementation

```
public boolean recognizes(String txt)
{
    Bag<Integer> pc = new Bag<Integer>();
    DirectedDFS dfs = new DirectedDFS(G, 0);
    for (int v = 0; v < G.V(); v++)
        if (dfs.marked(v)) pc.add(v);

    for (int i = 0; i < txt.length(); i++)
    {
        Bag<Integer> match = new Bag<Integer>();
        for (int v : pc)
        {
            if (v == M) continue;
            if ((re[v] == txt.charAt(i)) || re[v] == '.')
                match.add(v+1);
        }

        dfs = new DirectedDFS(G, match);
        pc = new Bag<Integer>();
        for (int v = 0; v < G.V(); v++)
            if (dfs.marked(v)) pc.add(v);
    }

    for (int v : pc)
        if (v == M) return true;
    return false;
}
```

states reachable from start by ϵ -transitions

states reachable after scanning past `txt.charAt(i)`

follow ϵ -transitions

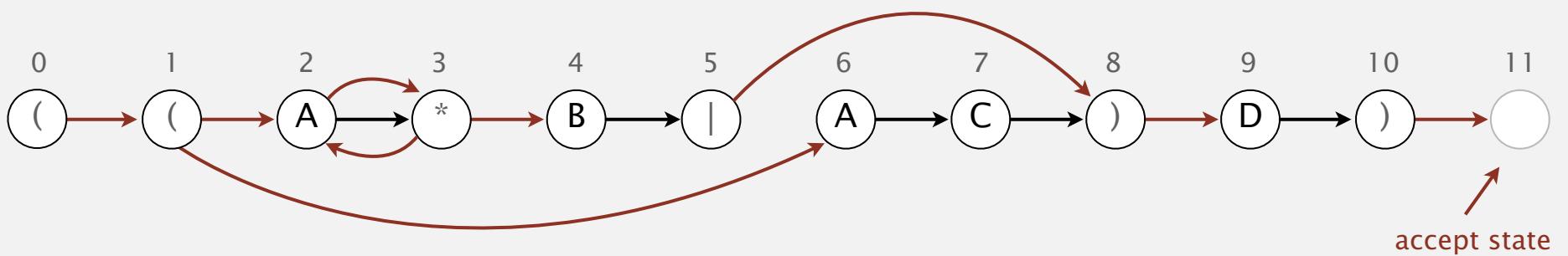
accept if can end in state M

NFA simulation: analysis

Proposition. Determining whether an N -character text is recognized by the NFA corresponding to an M -character pattern takes time proportional to MN in the worst case.

Pf. For each of the N text characters, we iterate through a set of states of size no more than M and run DFS on the graph of ϵ -transitions.

[The NFA construction we will consider ensures the number of edges $\leq 3M$.]



NFA corresponding to the pattern $((A^* B \mid A C) D)$

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

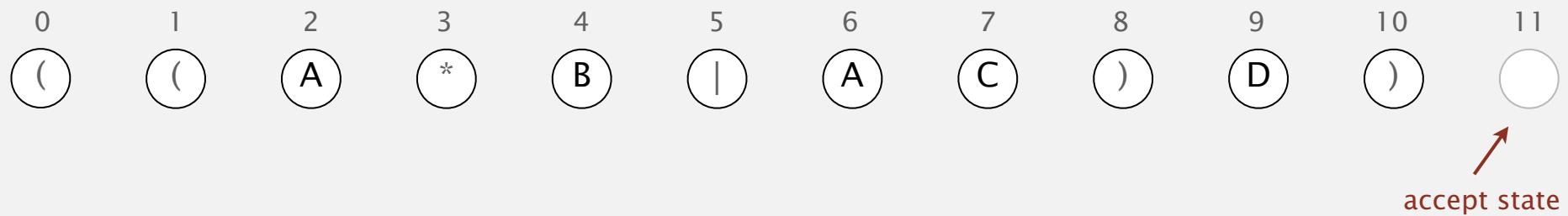
<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Building an NFA corresponding to an RE

States. Include a state for each symbol in the RE, plus an accept state.



NFA corresponding to the pattern $((A^* B \mid A C) D)$

Building an NFA corresponding to an RE

Concatenation. Add match-transition edge from state corresponding to characters in the alphabet to next state.

Alphabet. A B C D

Metacharacters. () . * |



NFA corresponding to the pattern $((A^* B \mid A C) D)$

Building an NFA corresponding to an RE

Parentheses. Add ϵ -transition edge from parentheses to next state.

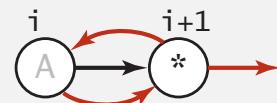


NFA corresponding to the pattern $((A^* B \mid A C) D)$

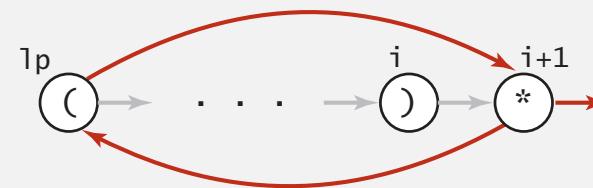
Building an NFA corresponding to an RE

Closure. Add three ϵ -transition edges for each * operator.

single-character closure



closure expression

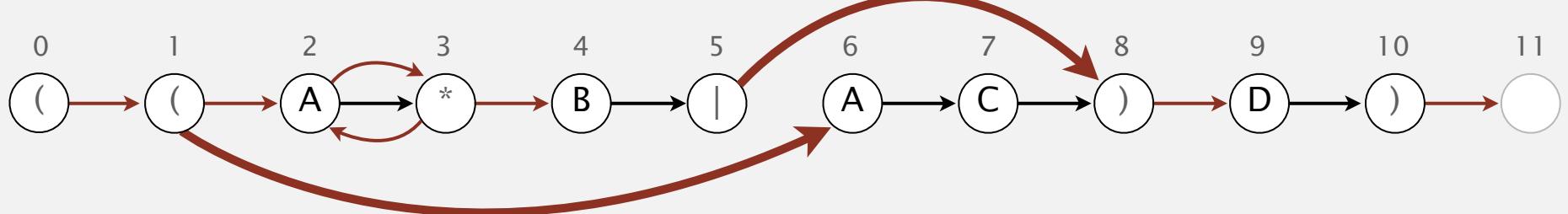
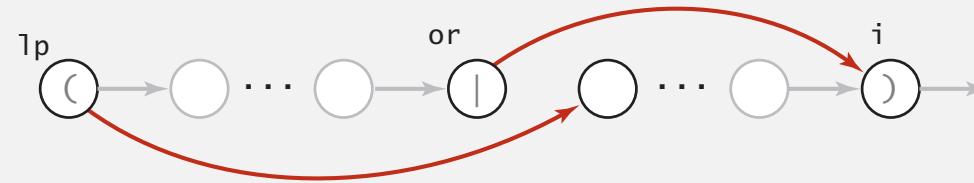


NFA corresponding to the pattern $((A^* B \mid A C) D)$

Building an NFA corresponding to an RE

Or. Add two ϵ -transition edges for each | operator.

or expression



NFA corresponding to the pattern $((A^* B | A C) D)$

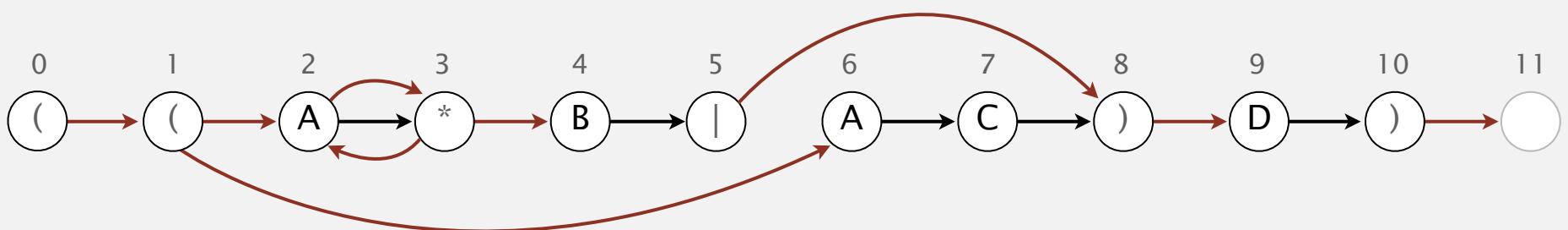
NFA construction: implementation

Goal. Write a program to build the ϵ -transition digraph.

Challenges. Remember left parentheses to implement closure and or;
remember | to implement or.

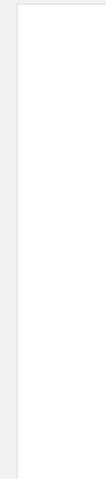
Solution. Maintain a stack.

- (symbol: push (onto stack.
- | symbol: push | onto stack.
-) symbol: pop corresponding (and any intervening |;
add ϵ -transition edges for closure/or.



NFA corresponding to the pattern $((A^* B \mid A C) D)$

NFA construction demo



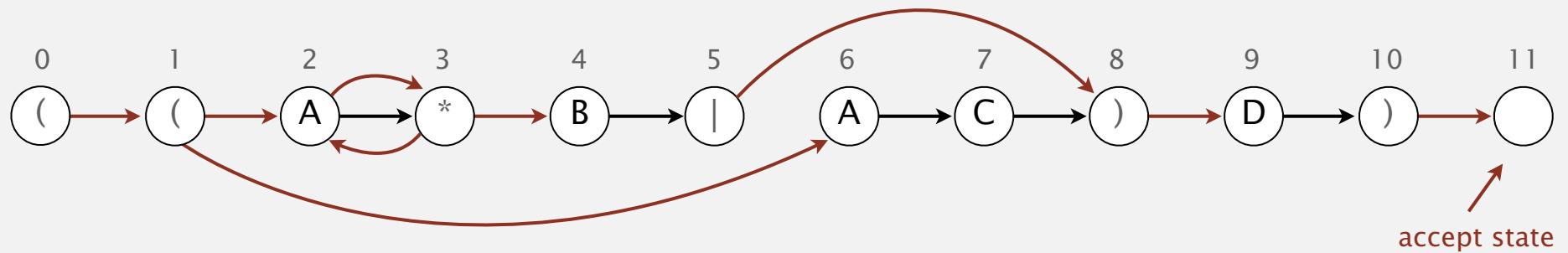
stack

((A * B | A C) D)

NFA construction demo



stack



NFA corresponding to the pattern $((A^* B \mid A C) D)$

NFA construction: Java implementation

```
private Digraph buildEpsilonTransitionDigraph() {
    Digraph G = new Digraph(M+1);
    Stack<Integer> ops = new Stack<Integer>();
    for (int i = 0; i < M; i++) {
        int lp = i;

        if (re[i] == '(' || re[i] == '|') ops.push(i); ← left parentheses and |

        else if (re[i] == ')') {
            int or = ops.pop();
            if (re[or] == '|') {
                lp = ops.pop();
                G.addEdge(lp, or+1);
                G.addEdge(or, i);
            }
            else lp = or;
        }

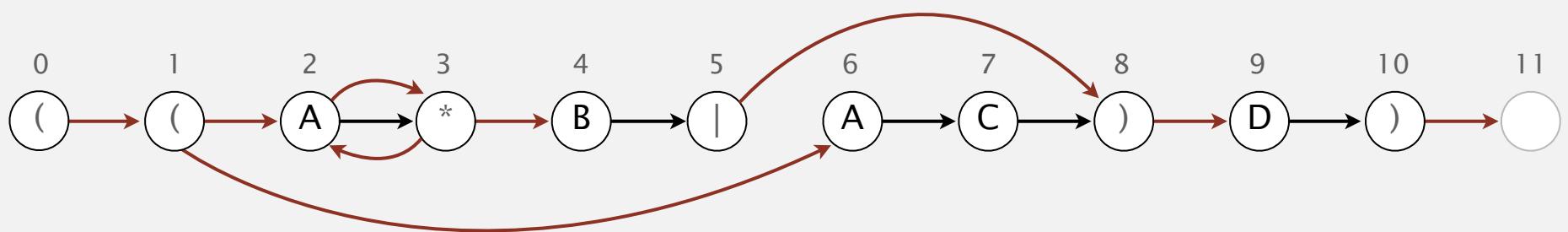
        if (i < M-1 && re[i+1] == '*') { ← closure
            G.addEdge(lp, i+1);
            G.addEdge(i+1, lp);
        }

        if (re[i] == '(' || re[i] == '*' || re[i] == ')') ← metasymbols
            G.addEdge(i, i+1);
    }
    return G;
}
```

NFA construction: analysis

Proposition. Building the NFA corresponding to an M -character RE takes time and space proportional to M .

Pf. For each of the M characters in the RE, we add at most three ϵ -transitions and execute at most two stack operations.



NFA corresponding to the pattern $((A^* B \mid A C) D)$

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Generalized regular expression print

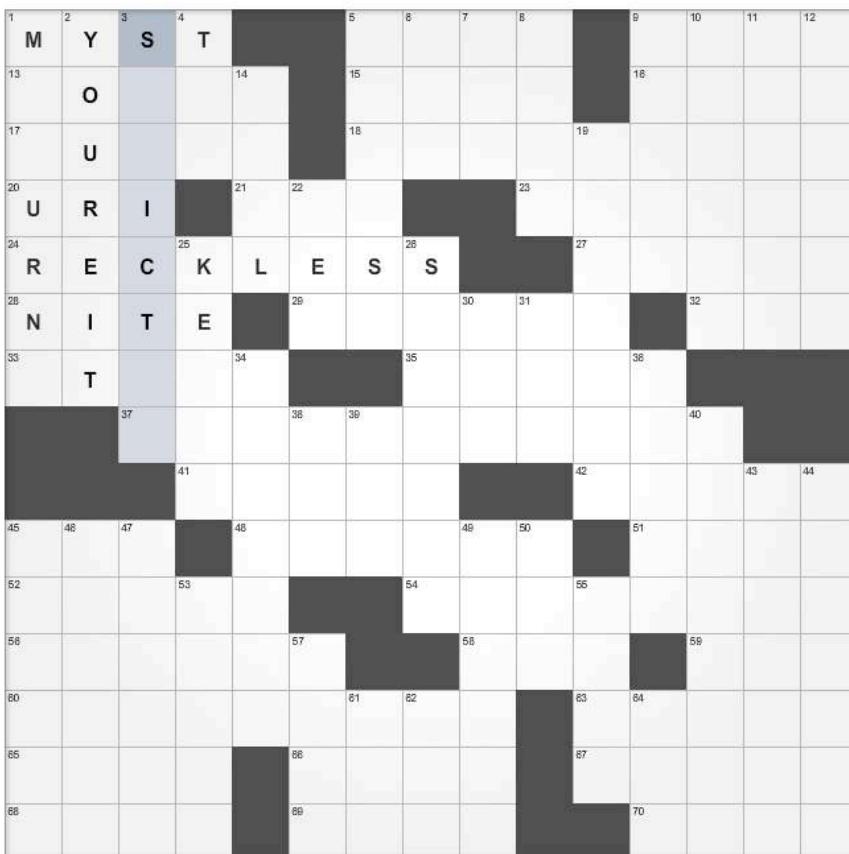
Grep. Take a RE as a command-line argument and print the lines from standard input having some substring that is matched by the RE.

```
public class GREP
{
    public static void main(String[] args)
    {
        String re = ".*" + args[0] + ".*";
        NFA nfa = new NFA(re);
        while (StdIn.hasNextLine())
        {
            String line = StdIn.readLine();
            if (nfa.recognizes(line))
                StdOut.println(line);
        }
    }
}
```

contains RE
as a substring

Bottom line. Worst-case for grep (proportional to MN) is the same as for brute-force substring search.

Typical grep application: crossword puzzles



```
% more words.txt
```

```
a  
aback  
abacus  
abalone  
abandon
```

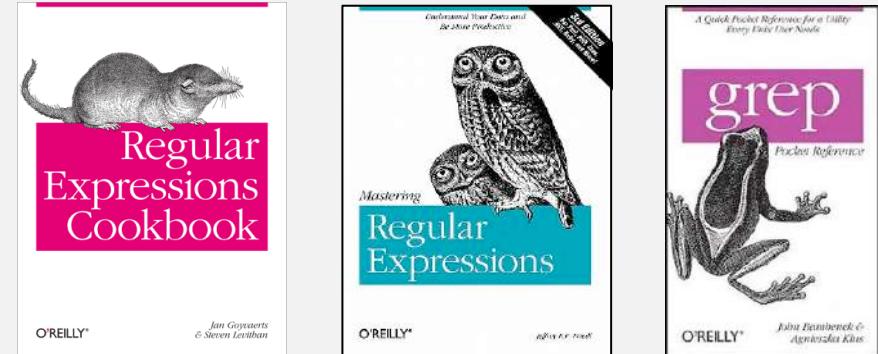
dictionary
(standard in Unix)
also on booksite

```
***  
  
% grep "s..ict.." words.txt  
constrictor  
stricter  
stricture
```

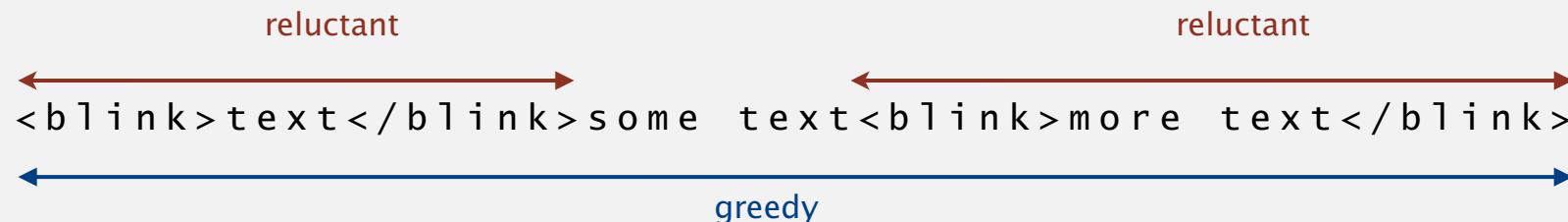
Industrial-strength grep implementation

To complete the implementation:

- Add character classes.
- Handle metacharacters.
- Add capturing capabilities.
- Extend the closure operator.
- Error checking and recovery.
- Greedy vs. reluctant matching.



Ex. Which substring(s) should be matched by the RE `<blink>.*</blink>` ?



Regular expressions in other languages

Broadly applicable programmer's tool.

- Originated in Unix in the 1970s.
- Many languages support extended regular expressions.
- Built into grep, awk, emacs, Perl, PHP, Python, JavaScript, ...

```
% grep 'NEWLINE' */*.java
```

← print all lines containing NEWLINE which occurs in any file with a .java extension

```
% egrep '^[qwertyuiop]*[zxcvbnm]*$' words.txt | egrep '.....'
```

← typewritten

PERL. Practical Extraction and Report Language.

```
% perl -p -i -e 's|from|to|g' input.txt
```

← replace all occurrences of from with to in the file input.txt

```
% perl -n -e 'print if /^[A-Z][A-Za-z]*$/' words.txt
```

← print all words that start with uppercase letter



do for each line

Regular expressions in Java

Validity checking. Does the input match the re?

Java string library. Use `input.matches(re)` for basic RE matching.

```
public class Validate
{
    public static void main(String[] args)
    {
        String regexp = args[0];
        String input = args[1];
        StdOut.println(input.matches(re));
    }
}
```

```
% java Validate "[$_A-Za-z][$_A-Za-z0-9]*" ident123
```

legal Java identifier

```
% java Validate "[a-z]+@[a-z]+\.(edu|com)" rs@cs.princeton.edu
```

valid email address
(simplified)

```
% java Validate "[0-9]{3}-[0-9]{2}-[0-9]{4}" 166-11-4433
```

Social Security number

Harvesting information

Goal. Print all substrings of input that match a RE.

```
% java Harvester "gcg(cgg|agg)*ctg" chromosomeX.txt
gcgcggcggcggcggcggctg
gcgctg
gcgctg
gcgcggcggcggaggcggaggcggctg
                           ↑
                           harvest patterns from DNA
                           ↓
                           harvest links from website
% java Harvester "http://(\w+\.\w+)*(\w+)" http://www.cs.princeton.edu
http://www.princeton.edu
http://www.google.com
http://www.cs.princeton.edu/news
```

Harvesting information

RE pattern matching is implemented in Java's `java.util.regex.Pattern` and `java.util.regex.Matcher` classes.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class Harvester
{
    public static void main(String[] args)
    {
        String regexp    = args[0];
        In in          = new In(args[1]);
        String input    = in.readAll();
        Pattern pattern = Pattern.compile(regexp);
        Matcher matcher = pattern.matcher(input);
        while (matcher.find())
        {
            StdOut.println(matcher.group());
        }
    }
}
```

The diagram shows the `Harvester` class code with several red arrows pointing from specific lines to explanatory text:

- An arrow points from `Pattern.compile(regexp);` to the text: `compile()` creates a `Pattern` (NFA) from RE
- An arrow points from `pattern.matcher(input);` to the text: `matcher()` creates a `Matcher` (NFA simulator) from NFA and text
- An arrow points from `while (matcher.find())` to the text: `find()` looks for the next match
- An arrow points from `StdOut.println(matcher.group());` to the text: `group()` returns the substring most recently found by `find()`

Algorithmic complexity attacks

Warning. Typical implementations do **not** guarantee performance!

Unix grep, Java, Perl

```
% java Validate "(a|aa)*b"aaaaaaaaaaaaaaaaaaaaaaaac      1.6 seconds
% java Validate "(a|aa)*b"aaaaaaaaaaaaaaaaaaaaaaaac      3.7 seconds
% java Validate "(a|aa)*b"aaaaaaaaaaaaaaaaaaaaaaaac      9.7 seconds
% java Validate "(a|aa)*b"aaaaaaaaaaaaaaaaaaaaaaaac     23.2 seconds
% java Validate "(a|aa)*b"aaaaaaaaaaaaaaaaaaaaaaaac    62.2 seconds
% java Validate "(a|aa)*b"aaaaaaaaaaaaaaaaaaaaaaaac 161.6 seconds
```

SpamAssassin regular expression.

```
% java RE "[a-z]+@[a-z]+([a-z\.]+\.)+[a-z]+" spammer@x.....
```

- Takes exponential time on pathological email addresses.
- Troublemaker can use such addresses to DOS a mail server.

Not-so-regular expressions

Back-references.

- \1 notation matches subexpression that was matched earlier.
- Supported by typical RE implementations.

```
(.+)\1          // beriberi couscous  
1?$_|^((11+?)\1+) // 1111 111111 1111111111
```

Some non-regular languages.

- Strings of the form ww for some string w : beriberi.
- Unary strings with a composite number of 1s: 111111.
- Bitstrings with an equal number of 0s and 1s: 01110100.
- Watson-Crick complemented palindromes: atttcggaaat.

Remark. Pattern matching with back-references is intractable.

Context

Abstract machines, languages, and nondeterminism.

- Basis of the theory of computation.
- Intensively studied since the 1930s.
- Basis of programming languages.

Compiler. A program that translates a program to machine code.

- KMP string \Rightarrow DFA.
- grep RE \Rightarrow NFA.
- javac Java language \Rightarrow Java byte code.

	KMP	grep	Java
pattern	string	RE	program
parser	unnecessary	check if legal	check if legal
compiler output	DFA	NFA	byte code
simulator	DFA simulator	NFA simulator	JVM

Summary of pattern-matching algorithms

Programmer.

- Implement substring search via DFA simulation.
- Implement RE pattern matching via NFA simulation.



Theoretician.

- RE is a compact description of a set of strings.
- NFA is an abstract machine equivalent in power to RE.
- DFAs, NFAs, and REs have limitations.



You. Practical application of core computer science principles.

Example of essential paradigm in computer science.

- Build intermediate abstractions.
- Pick the right ones!
- Solve important practical problems.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*



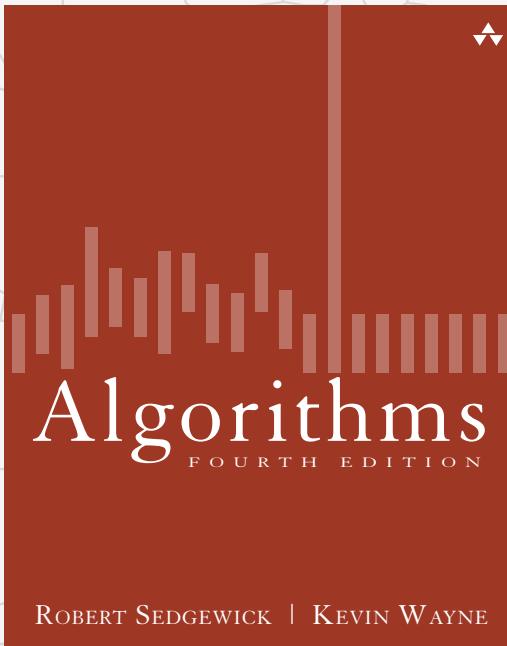
<http://algs4.cs.princeton.edu>

5.4 REGULAR EXPRESSIONS

- ▶ *regular expressions*
- ▶ *REs and NFAs*
- ▶ *NFA simulation*
- ▶ *NFA construction*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

Summary of the performance of symbol-table implementations

Order of growth of the frequency of operations.

implementation	typical case			ordered operations	operations on keys
	search	insert	delete		
red-black BST	$\log N$	$\log N$	$\log N$	yes	<code>compareTo()</code>
hash table	$1 \ddagger$	$1 \ddagger$	$1 \ddagger$	no	<code>equals()</code> <code>hashCode()</code>

\ddagger under uniform hashing assumption

Q. Can we do better?

A. Yes, if we can avoid examining the entire key, as with string sorting.

String symbol table basic API

String symbol table. Symbol table specialized to string keys.

```
public class StringST<Value>
```

```
    StringST()
```

create an empty symbol table

```
    void put(String key, Value val)
```

put key-value pair into the symbol table

```
    Value get(String key)
```

return value paired with given key

```
    void delete(String key)
```

delete key and corresponding value

```
    :
```

Goal. Faster than hashing, more flexible than BSTs.

String symbol table implementations cost summary

implementation	character accesses (typical case)					dedup	
	search hit	search miss	insert	space (references)	moby.txt	actors.txt	
red-black BST	$L + c \lg^2 N$	$c \lg^2 N$	$c \lg^2 N$	$4N$	1.40	97.4	
hashing (linear probing)	L	L	L	$4N$ to $16N$	0.76	40.6	

Parameters

- N = number of strings
- L = length of string
- R = radix

file	size	words	distinct
moby.txt	1.2 MB	210 K	32 K
actors.txt	82 MB	11.4 M	900 K

Challenge. Efficient performance for string keys.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.2 TRIES

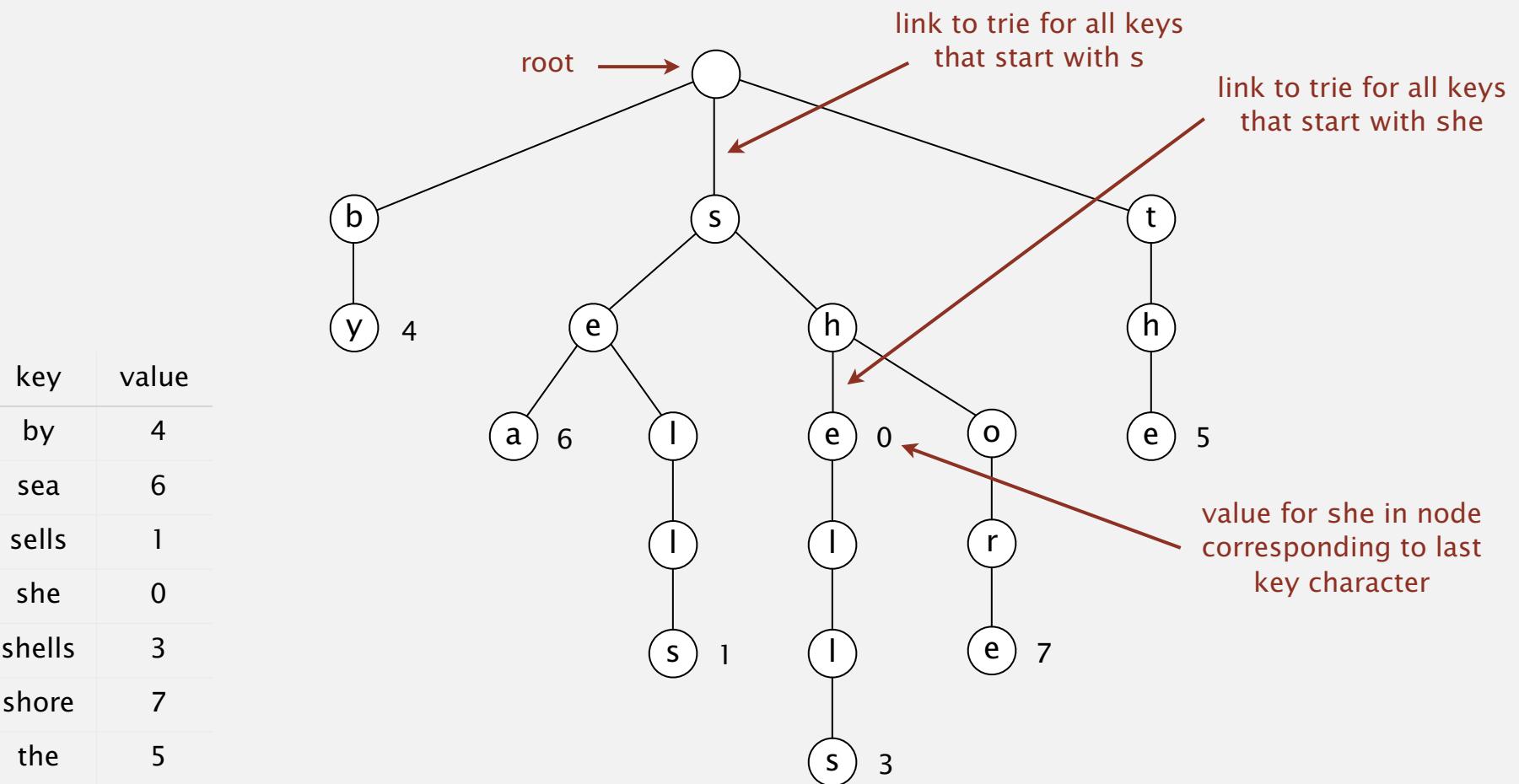
- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

Tries

Tries. [from **retrieval**, but pronounced "try"]

- Store characters in nodes (not keys).
- Each node has R children, one for each possible character.
- For now, we do not draw null links.

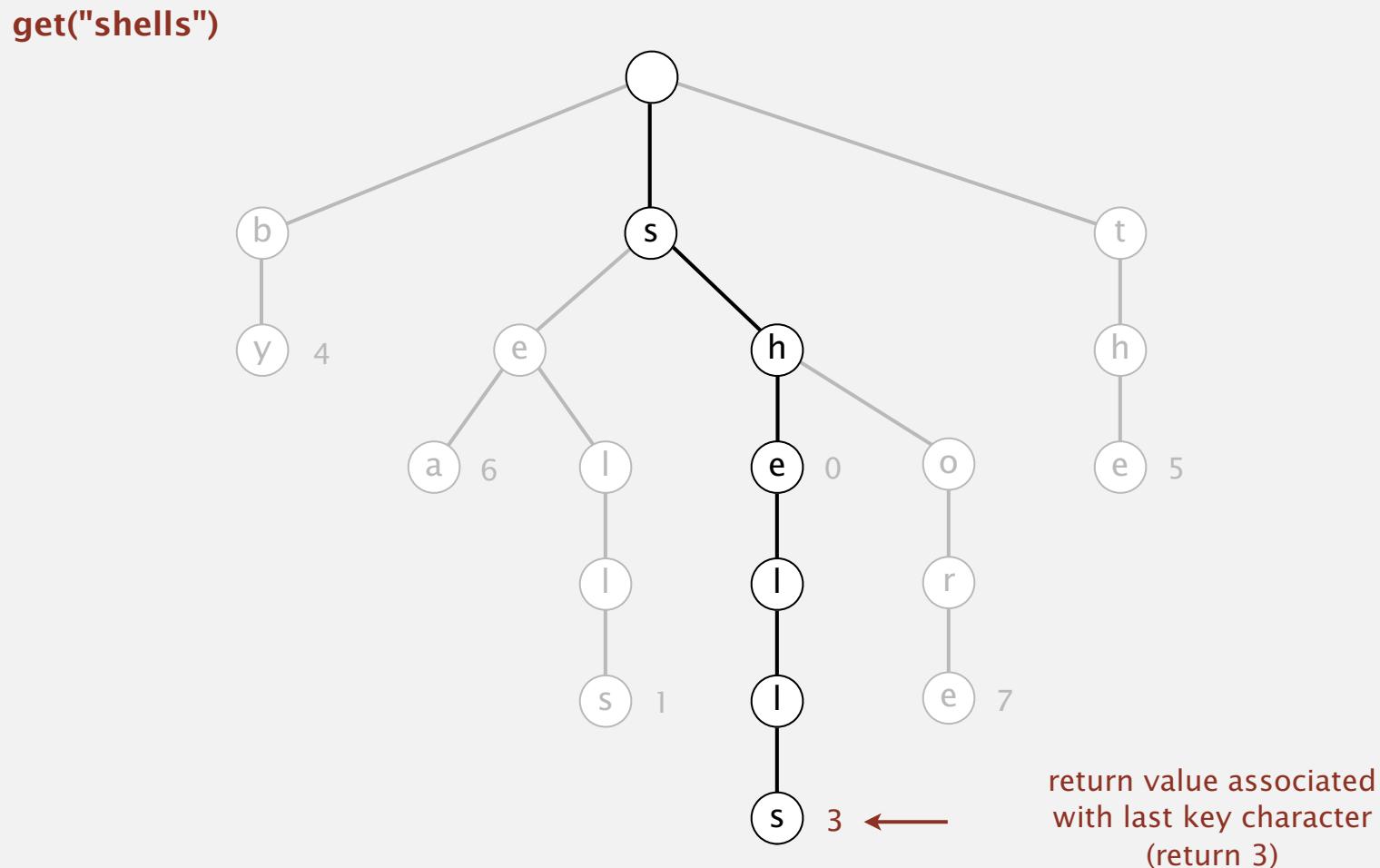
for now we do not
draw null links



Search in a trie

Follow links corresponding to each character in the key.

- **Search hit:** node where search ends has a non-null value.
- **Search miss:** reach null link or node where search ends has null value.

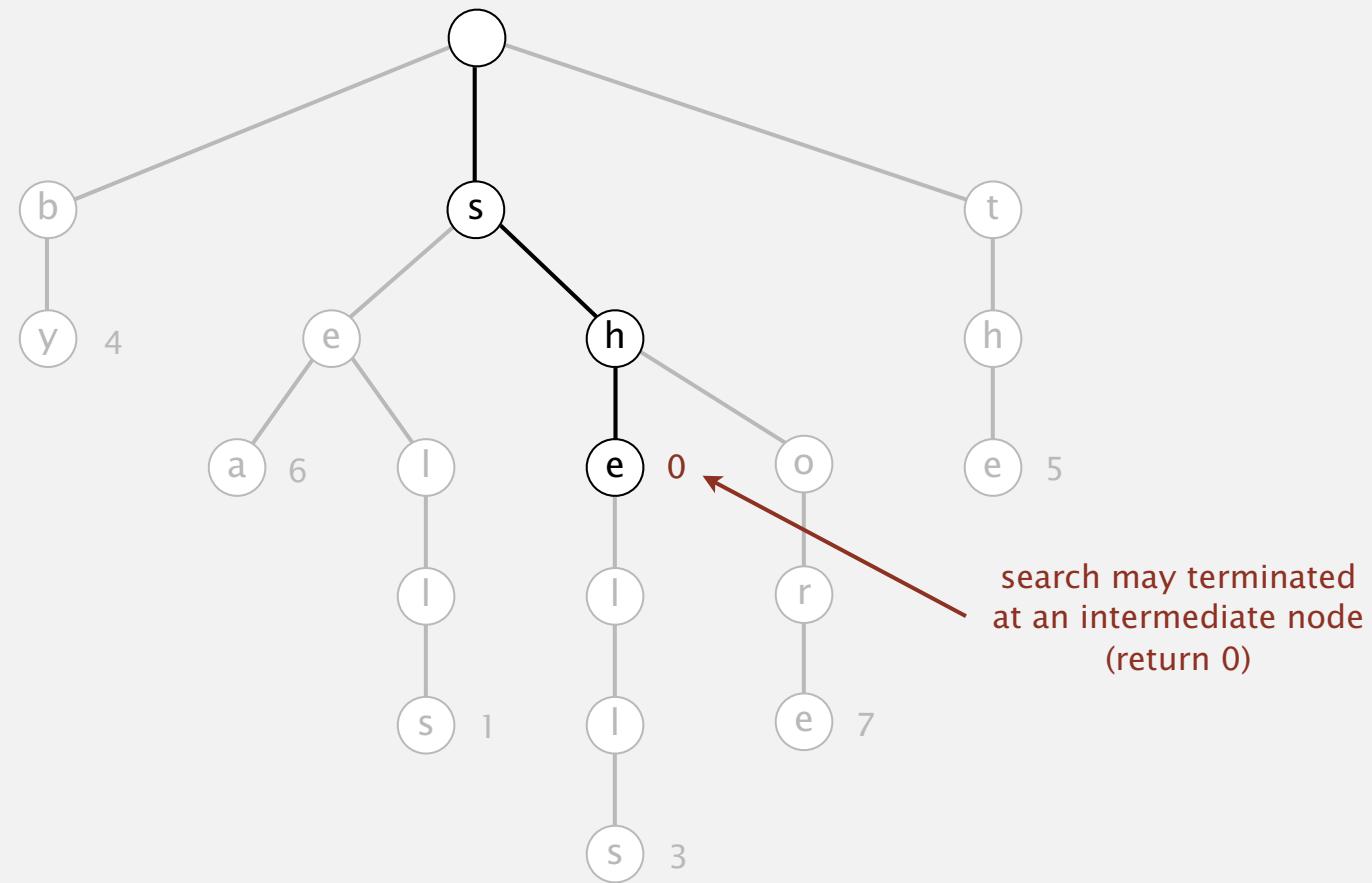


Search in a trie

Follow links corresponding to each character in the key.

- **Search hit:** node where search ends has a non-null value.
- **Search miss:** reach null link or node where search ends has null value.

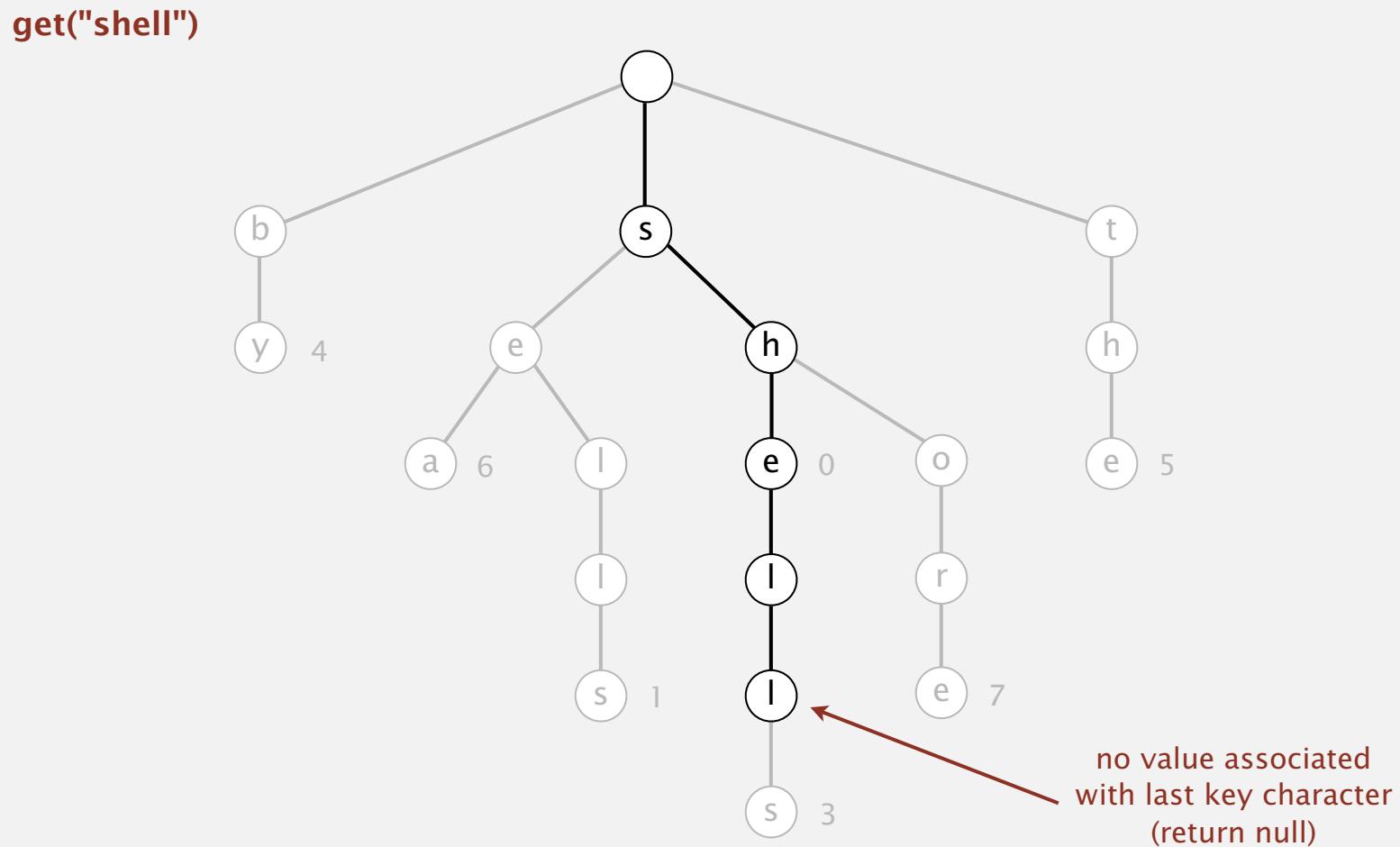
get("she")



Search in a trie

Follow links corresponding to each character in the key.

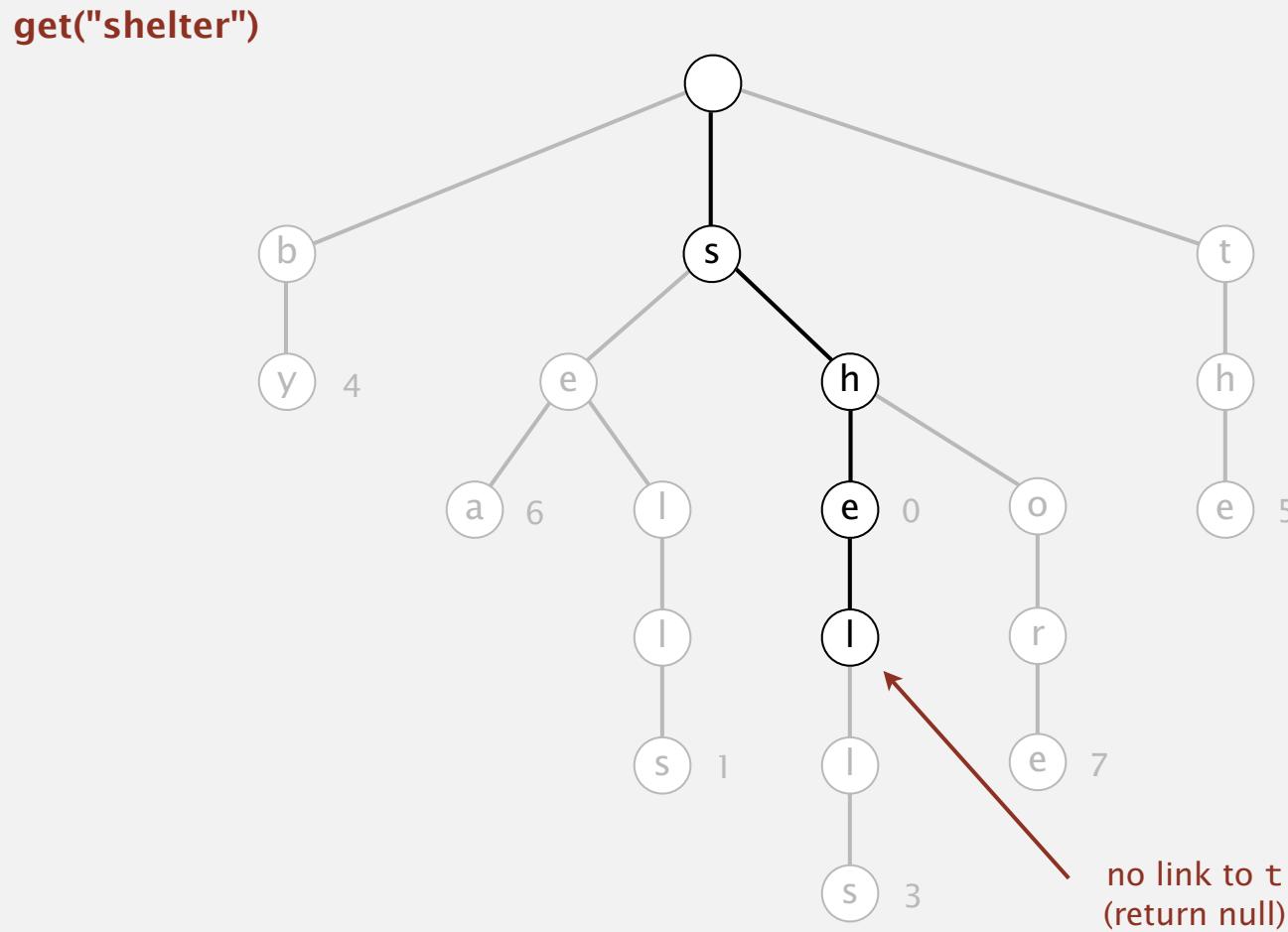
- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.



Search in a trie

Follow links corresponding to each character in the key.

- Search hit: node where search ends has a non-null value.
- Search miss: reach null link or node where search ends has null value.

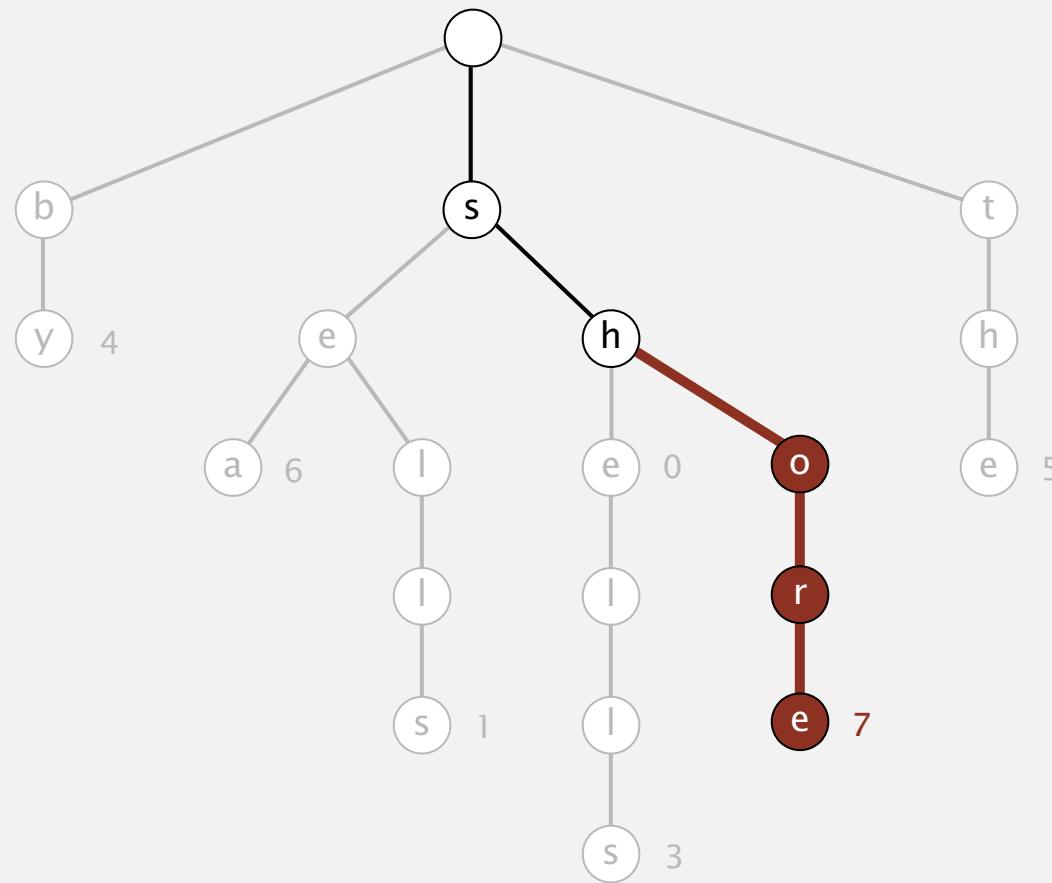


Insertion into a trie

Follow links corresponding to each character in the key.

- Encounter a null link: create new node.
- Encounter the last character of the key: set value in that node.

`put("shore", 7)`



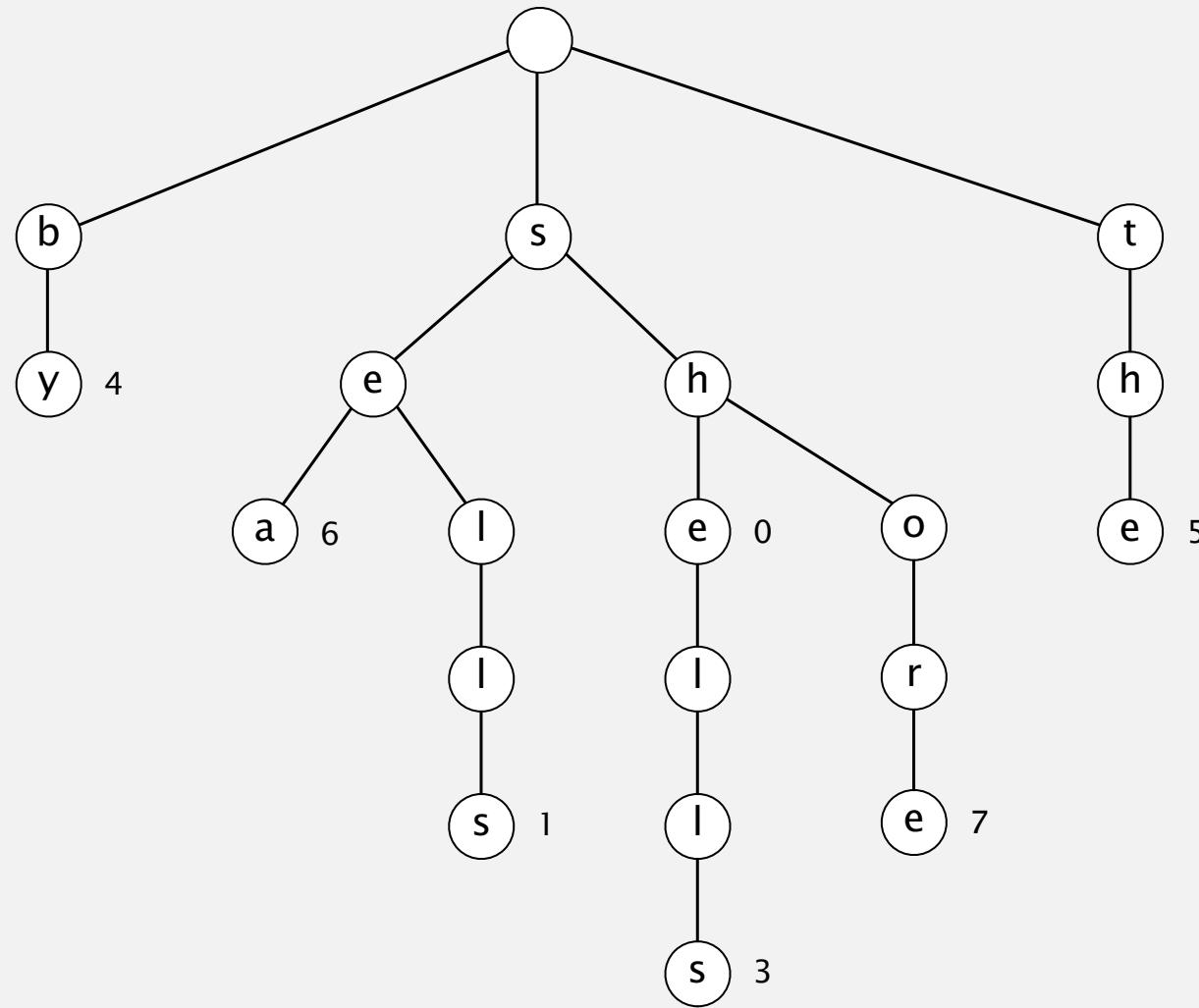
Trie construction demo

trie



Trie construction demo

trie

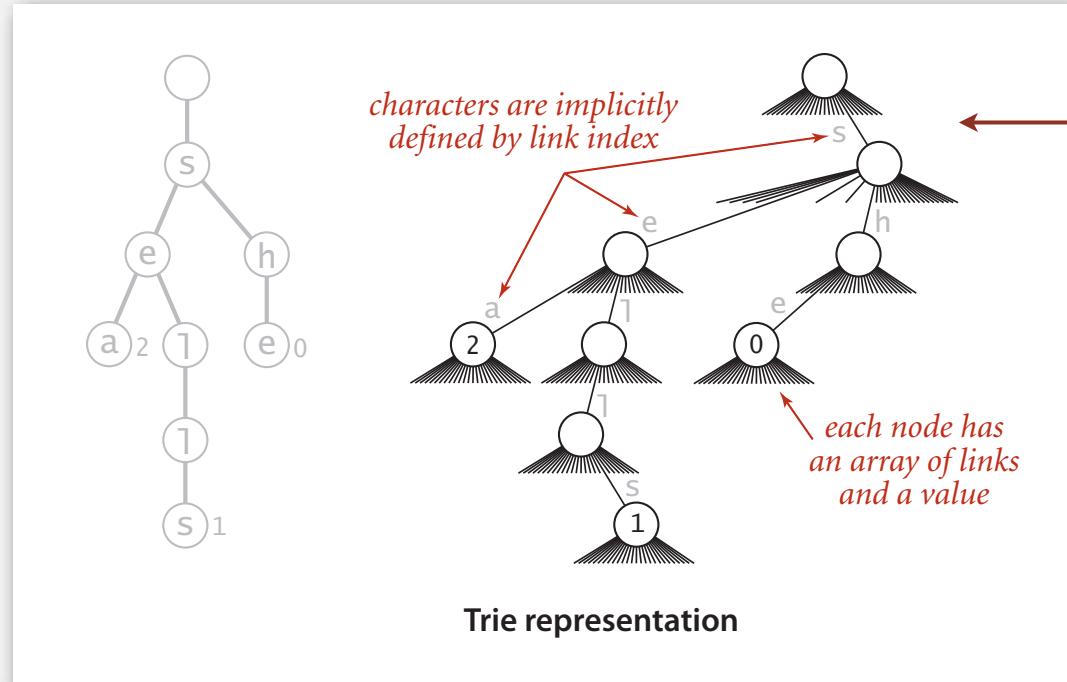


Trie representation: Java implementation

Node. A value, plus references to R nodes.

```
private static class Node
{
    private Object value;
    private Node[] next = new Node[R];
}
```

use Object instead of Value since
no generic array creation in Java



R-way trie: Java implementation

```
public class TrieST<Value>
{
    private static final int R = 256;      ← extended ASCII
    private Node root = new Node();

    private static class Node
    { /* see previous slide */ }

    public void put(String key, Value val)
    { root = put(root, key, val, 0); }

    private Node put(Node x, String key, Value val, int d)
    {
        if (x == null) x = new Node();
        if (d == key.length()) { x.val = val; return x; }
        char c = key.charAt(d);
        x.next[c] = put(x.next[c], key, val, d+1);
        return x;
    }

    ...
}
```

R-way trie: Java implementation (continued)

```
:
public boolean contains(String key)
{  return get(key) != null;  }

public Value get(String key)
{
    Node x = get(root, key, 0);
    if (x == null) return null;
    return (Value) x.val; ← cast needed
}

private Node get(Node x, String key, int d)
{
    if (x == null) return null;
    if (d == key.length()) return x;
    char c = key.charAt(d);
    return get(x.next[c], key, d+1);
}

}
```

Trie performance

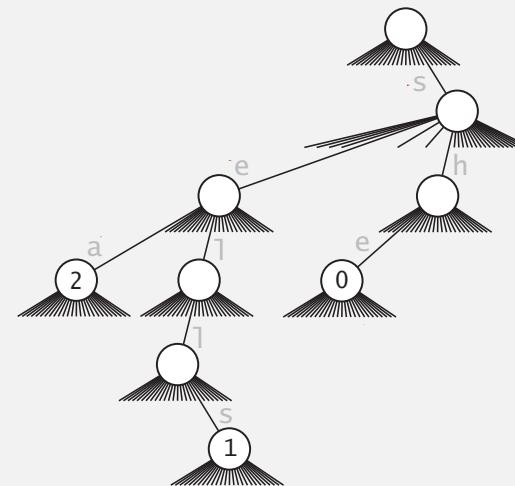
Search hit. Need to examine all L characters for equality.

Search miss.

- Could have mismatch on first character.
- Typical case: examine only a few characters (sublinear).

Space. R null links at each leaf.

(but sublinear space possible if many short strings share common prefixes)



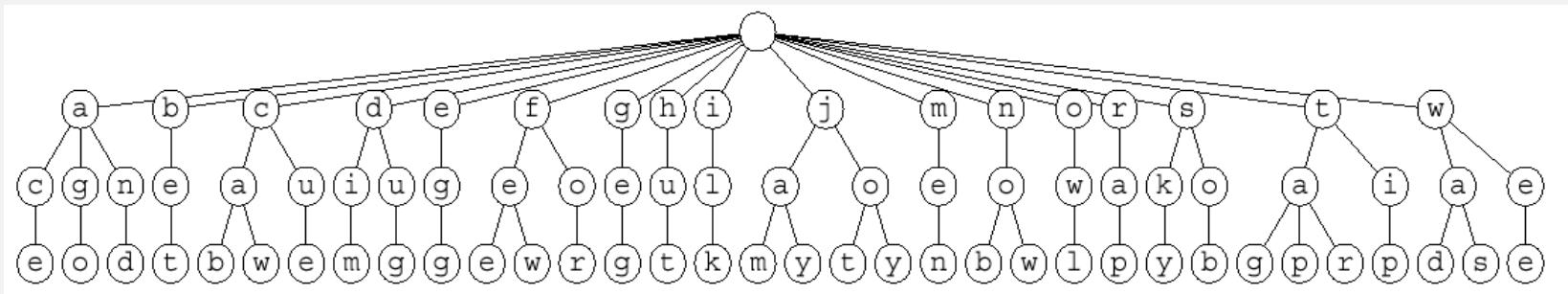
Bottom line. Fast search hit and even faster search miss, but wastes space.

Popular interview question

Goal. Design a data structure to perform efficient spell checking.

Solution. Build a 26-way trie (key = word, value = bit).

English-language text



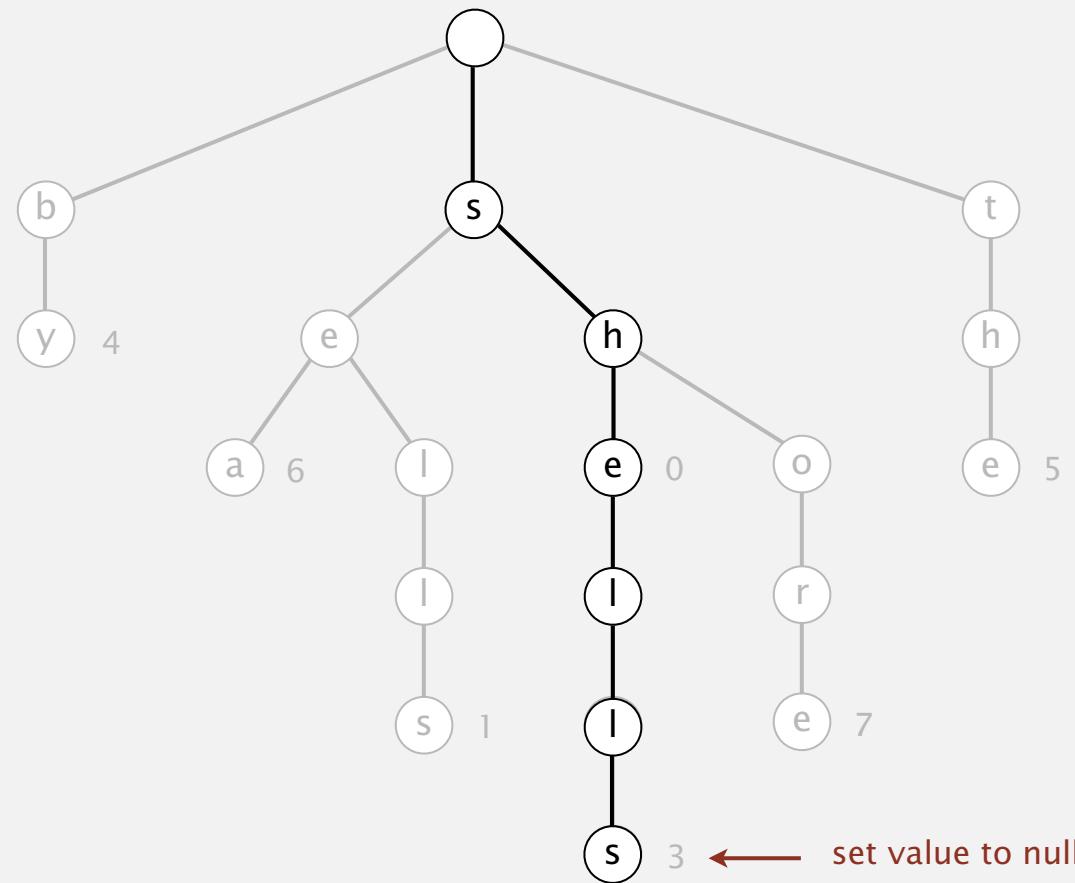
ace
ago
and
bet
cab
caw
cue
dim
dug
egg
fee
few
for
gig
hut
ilk
jam
jay
jot
joy
men
nob
now
owl
rap
sky
sob
tag
tap
tar
tip
wad
was
wee

Deletion in an R-way trie

To delete a key-value pair:

- Find the node corresponding to key and set value to null.
- If node has null value and all null links, remove that node (and recur).

`delete("shells")`

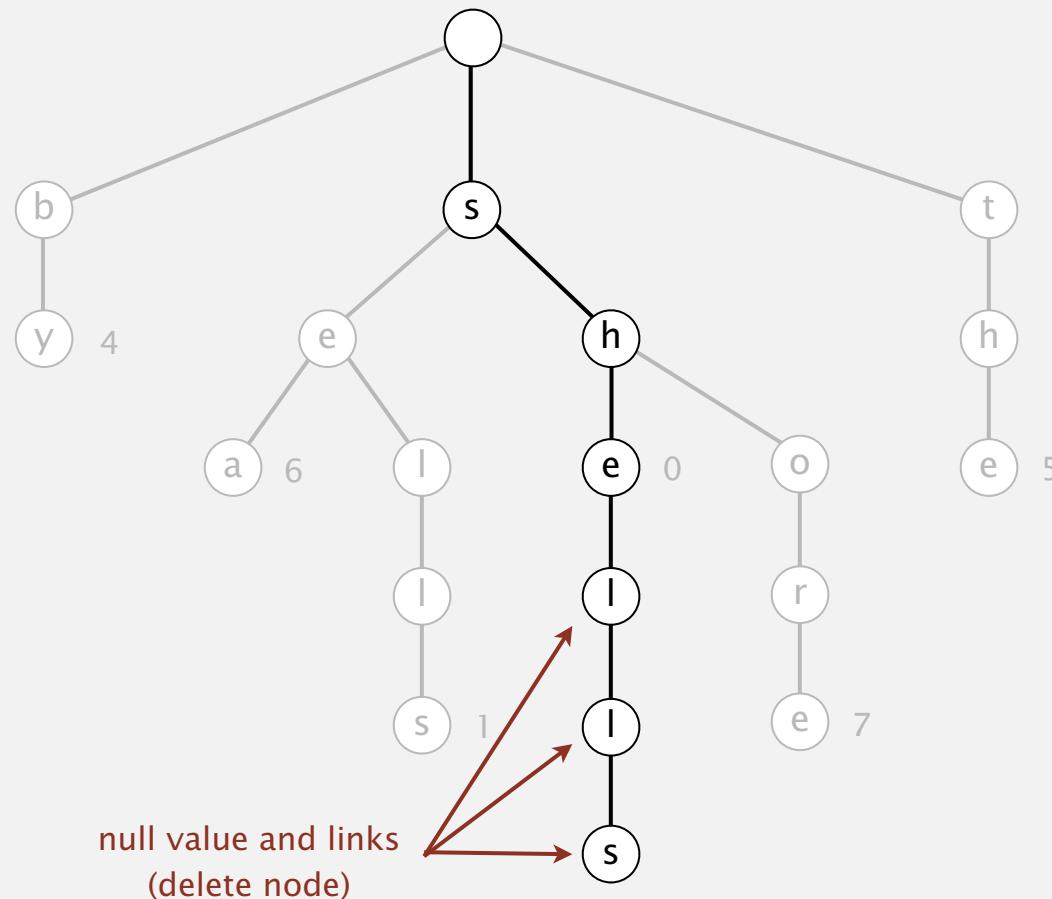


Deletion in an R-way trie

To delete a key-value pair:

- Find the node corresponding to key and set value to null.
- If node has null value and all null links, remove that node (and recur).

delete("shells")



String symbol table implementations cost summary

implementation	character accesses (typical case)				dedup	
	search hit	search miss	insert	space (references)	moby.txt	actors.txt
red-black BST	$L + c \lg^2 N$	$c \lg^2 N$	$c \lg^2 N$	$4N$	1.40	97.4
hashing (linear probing)	L	L	L	$4N$ to $16N$	0.76	40.6
R-way trie	L	$\log_R N$	L	$(R+1) N$	1.12	out of memory

R-way trie.

- Method of choice for small R .
- Too much memory for large R .

Challenge. Use less memory, e.g., 65,536-way trie for Unicode!

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

Ternary search tries

- Store characters and values in nodes (not keys).
- Each node has 3 children: smaller (left), equal (middle), larger (right).

Fast Algorithms for Sorting and Searching Strings

Jon L. Bentley* Robert Sedgewick#

Abstract

We present theoretical algorithms for sorting and searching multikey data, and derive from them practical C implementations for applications in which keys are character strings. The sorting algorithm blends Quicksort and radix sort; it is competitive with the best known C sort codes. The searching algorithm blends tries and binary search trees; it is faster than hashing and other commonly used search methods. The basic ideas behind the algo-

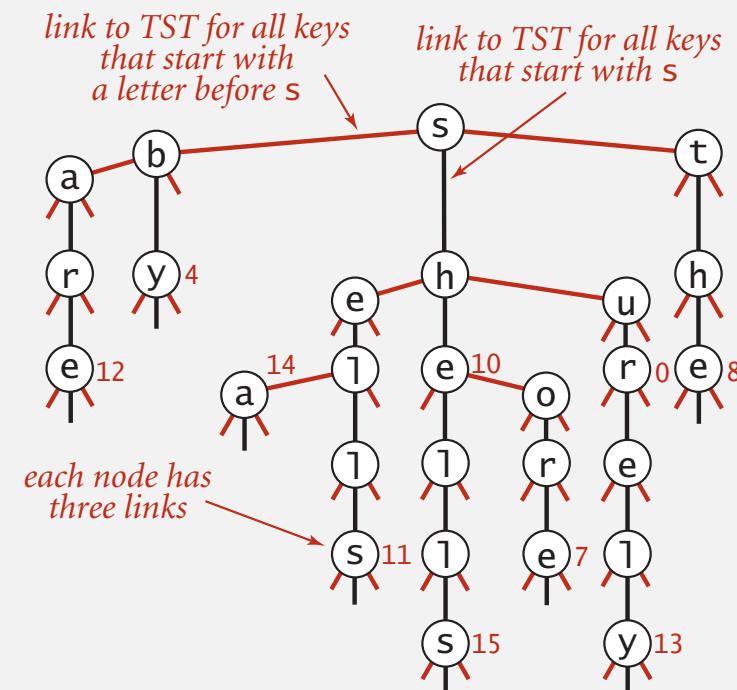
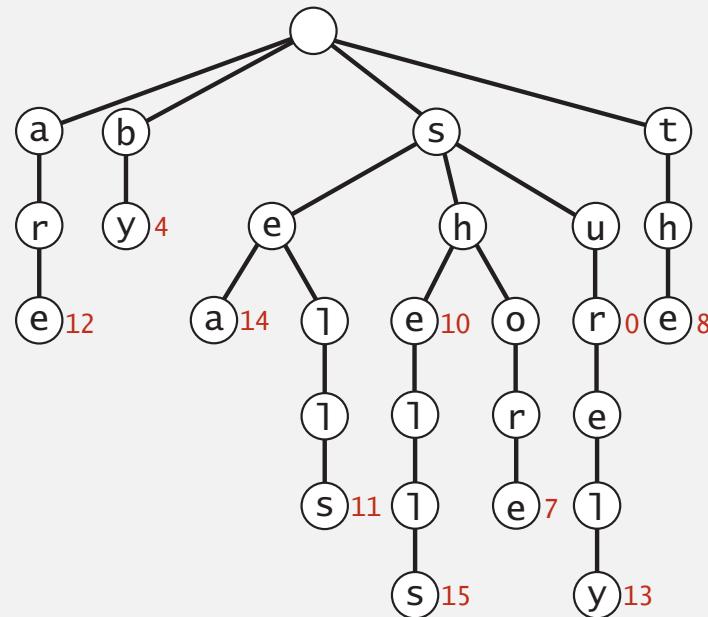
that is competitive with the most efficient string sorting programs known. The second program is a symbol table implementation that is faster than hashing, which is commonly regarded as the fastest symbol table implementation. The symbol table implementation is much more space-efficient than multiway trees, and supports more advanced searches.

In many application programs, sorts use a Quicksort implementation based on an abstract compare operation,



Ternary search tries

- Store characters and values in nodes (not keys).
- Each node has 3 children: smaller (left), equal (middle), larger (right).



TST representation of a trie

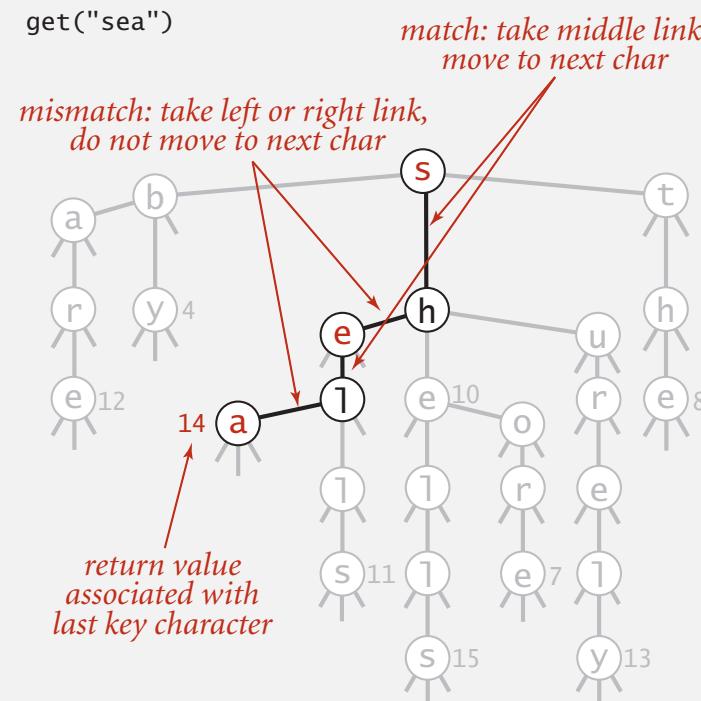
Search in a TST

Follow links corresponding to each character in the key.

- If less, take left link; if greater, take right link.
- If equal, take the middle link and move to the next key character.

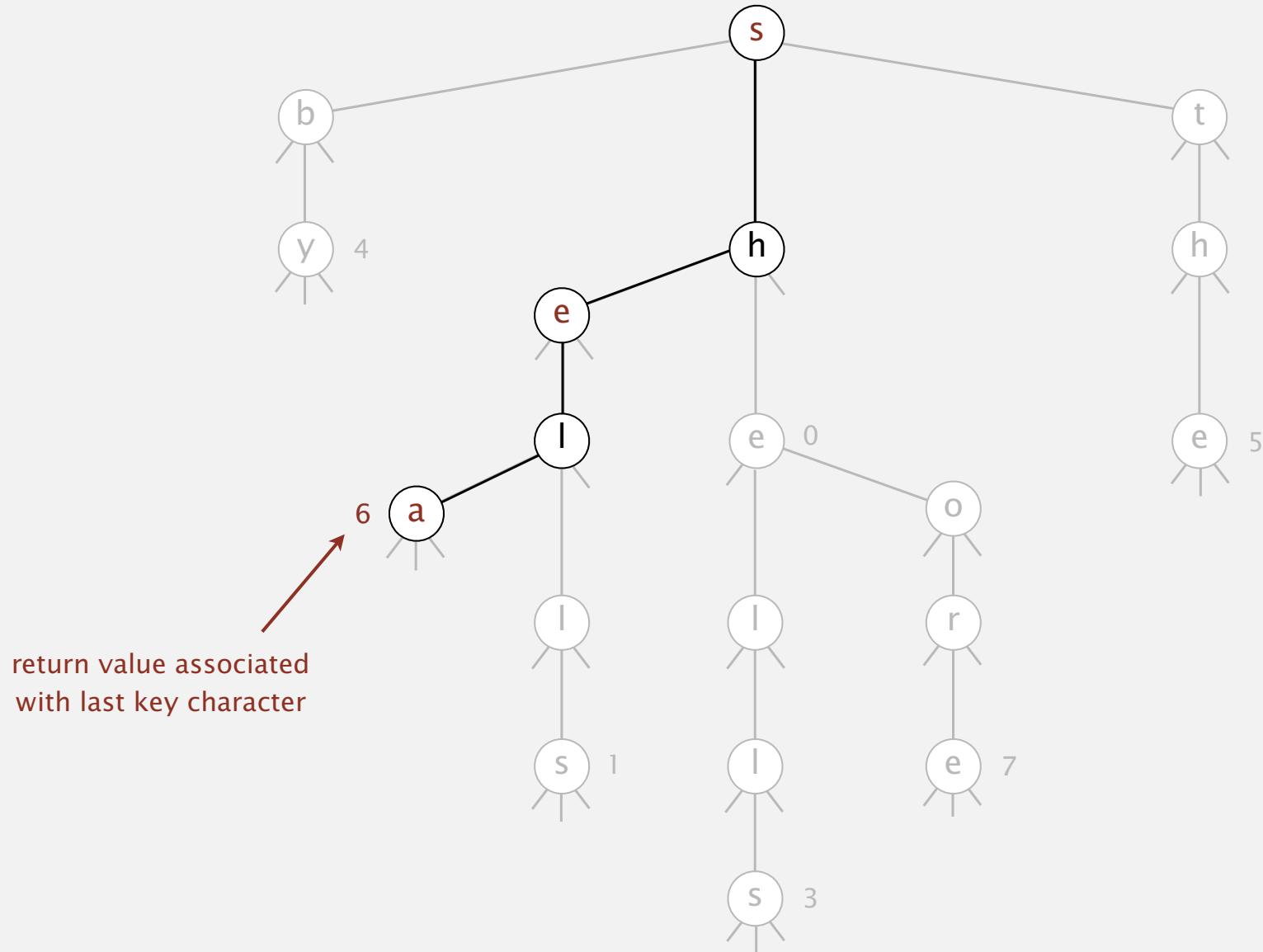
Search hit. Node where search ends has a non-null value.

Search miss. Reach a null link or node where search ends has null value.



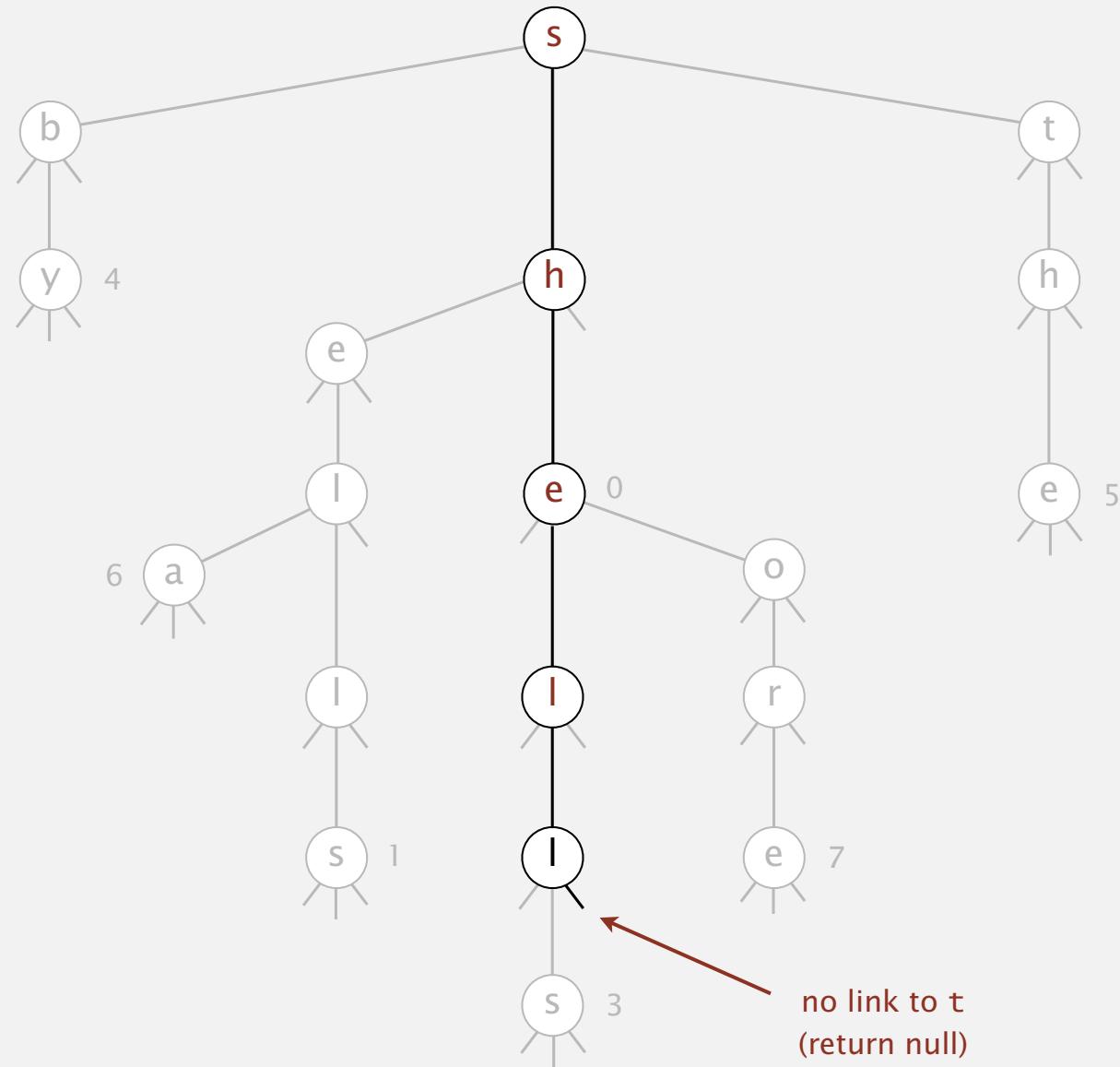
Search hit in a TST

get("sea")



Search miss in a TST

get("shelter")



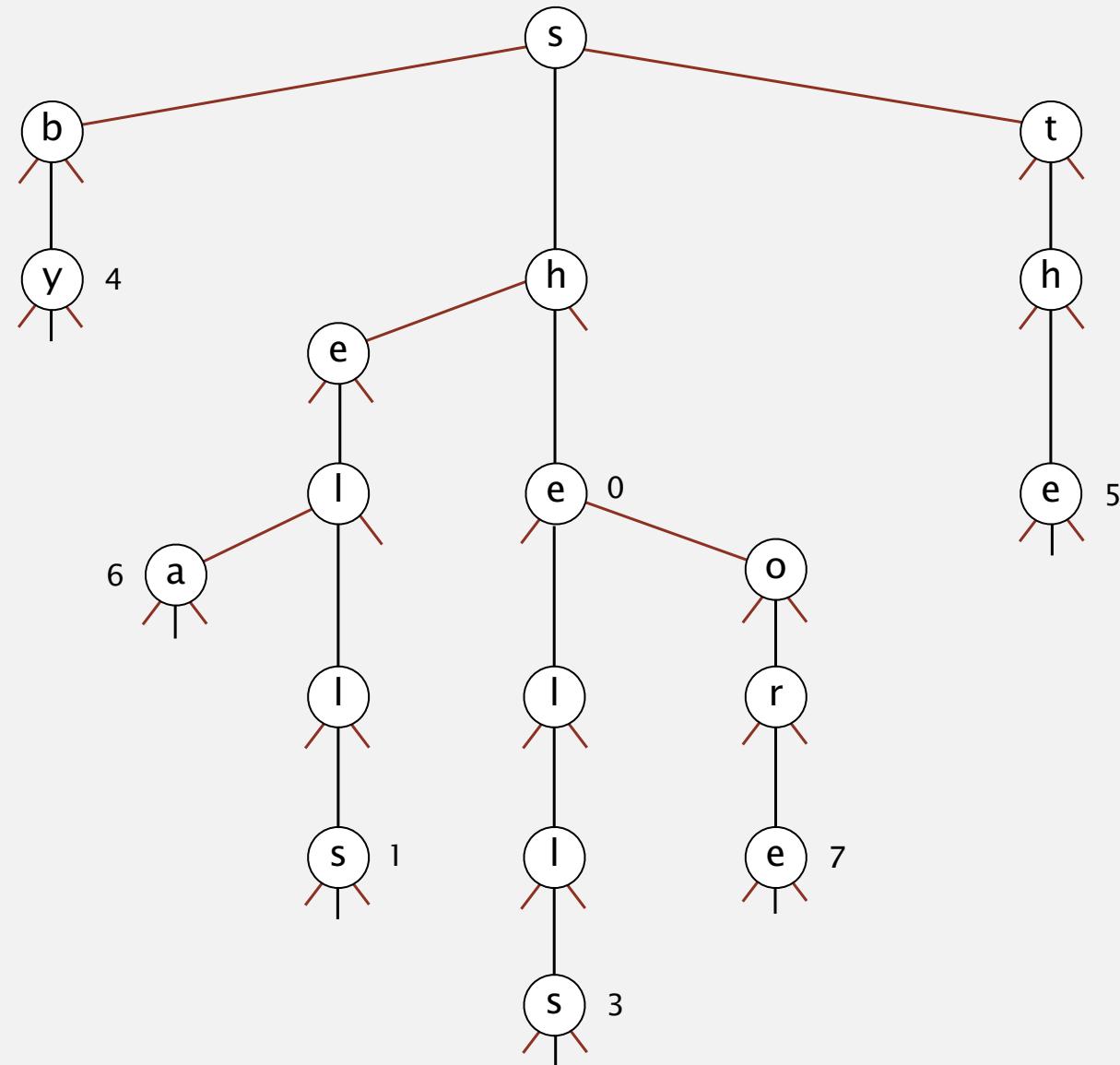
Ternary search trie construction demo

ternary search trie



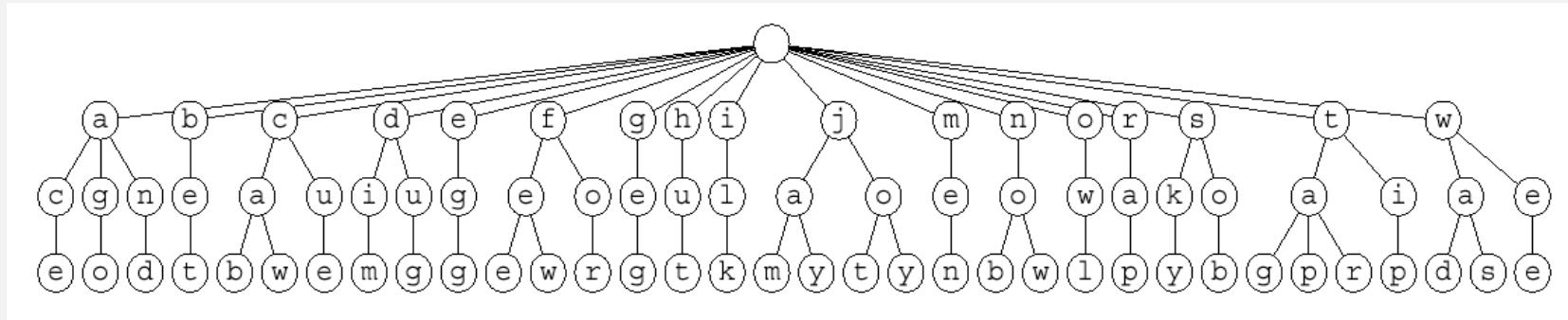
Ternary search trie construction demo

ternary search trie



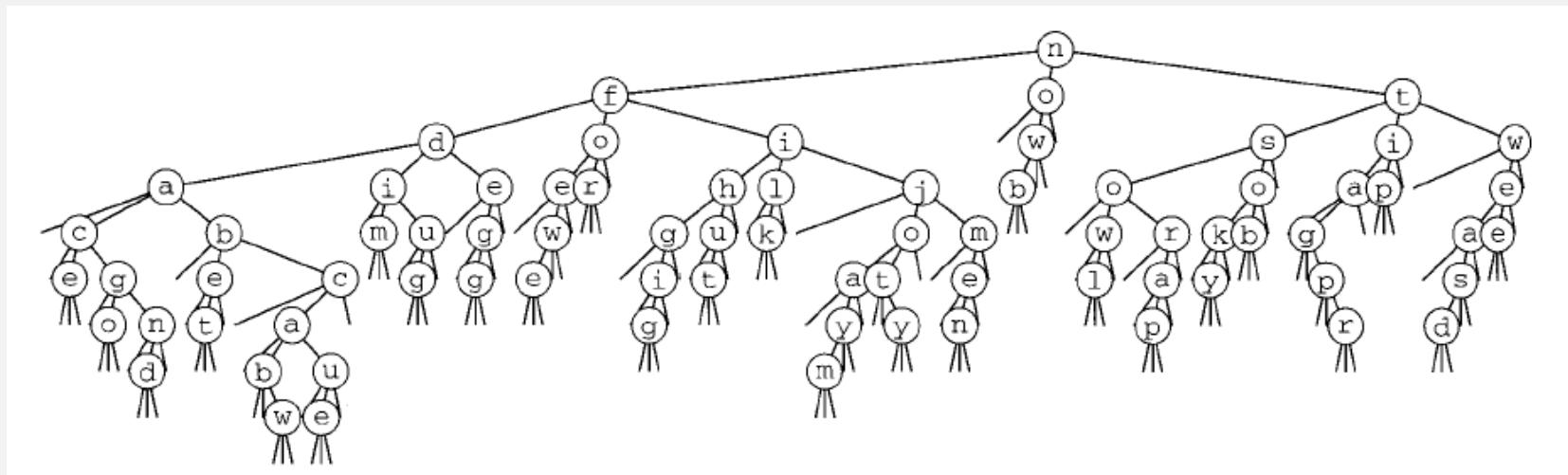
26-way trie vs. TST

26-way trie. 26 null links in each leaf.



26-way trie (1035 null links, not shown)

TST. 3 null links in each leaf.



TST (155 null links)

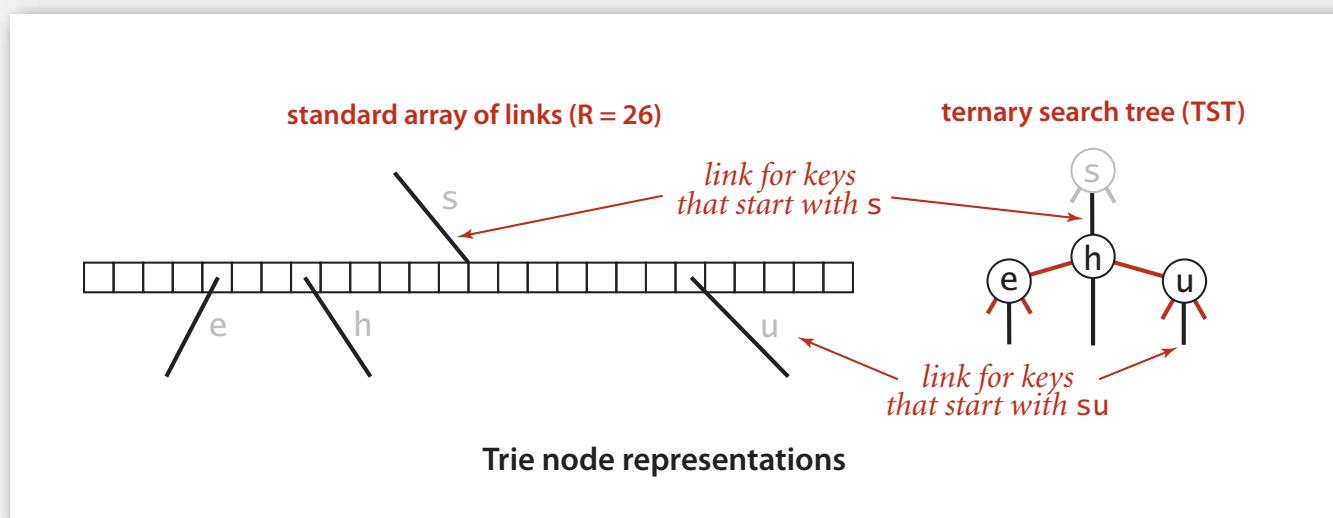
now
for
tip
ilk
dim
tag
jot
sob
nob
sky
hut
ace
bet
men
egg
few
jay
owl
joy
rap
gig
wee
was
cab
wad
caw
cue
fee
tap
ago
tar
jam
dug
and

TST representation in Java

A TST node is five fields:

- A value.
- A character c .
- A reference to a left TST.
- A reference to a middle TST.
- A reference to a right TST.

```
private class Node
{
    private Value val;
    private char c;
    private Node left, mid, right;
}
```



TST: Java implementation

```
public class TST<Value>
{
    private Node root;

    private class Node
    { /* see previous slide */ }

    public void put(String key, Value val)
    { root = put(root, key, val, 0); }

    private Node put(Node x, String key, Value val, int d)
    {
        char c = key.charAt(d);
        if (x == null) { x = new Node(); x.c = c; }
        if (c < x.c)             x.left  = put(x.left,  key, val, d);
        else if (c > x.c)       x.right = put(x.right, key, val, d);
        else if (d < key.length() - 1) x.mid   = put(x.mid,   key, val, d+1);
        else                      x.val   = val;
        return x;
    }

    ...
}
```

TST: Java implementation (continued)

```
:
public boolean contains(String key)
{  return get(key) != null;  }

public Value get(String key)
{
    Node x = get(root, key, 0);
    if (x == null) return null;
    return x.val;
}

private Node get(Node x, String key, int d)
{
    if (x == null) return null;
    char c = key.charAt(d);
    if      (c < x.c)                  return get(x.left,  key, d);
    else if (c > x.c)                  return get(x.right, key, d);
    else if (d < key.length() - 1)    return get(x.mid,   key, d+1);
    else                                return x;
}
```

String symbol table implementation cost summary

implementation	character accesses (typical case)					dedup	
	search hit	search miss	insert	space (references)	moby.txt	actors.txt	
red-black BST	$L + c \lg^2 N$	$c \lg^2 N$	$c \lg^2 N$	$4 N$	1.40	97.4	
hashing (linear probing)	L	L	L	$4 N$ to $16 N$	0.76	40.6	
R-way trie	L	$\log_R N$	L	$(R + 1) N$	1.12	out of memory	
TST	$L + \ln N$	$\ln N$	$L + \ln N$	4 N	0.72	38.7	

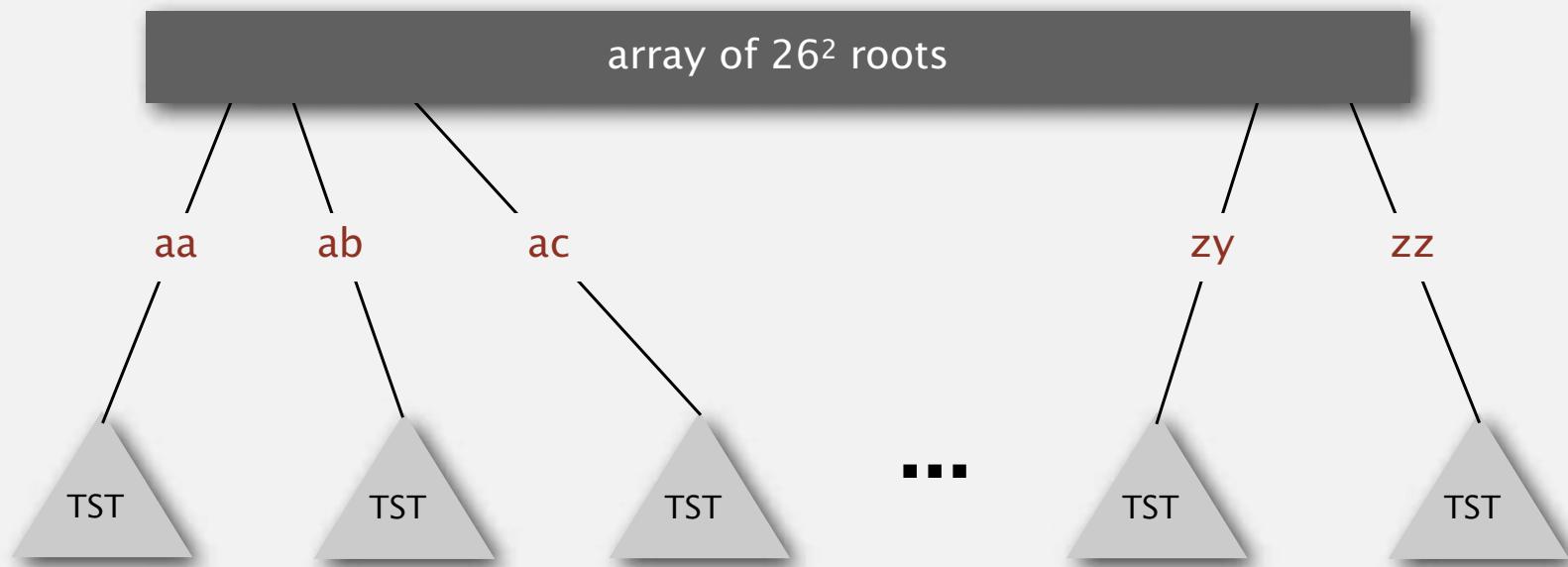
Remark. Can build balanced TSTs via rotations to achieve $L + \log N$ worst-case guarantees.

Bottom line. TST is as fast as hashing (for string keys), space efficient.

TST with R^2 branching at root

Hybrid of R-way trie and TST.

- Do R^2 -way branching at root.
- Each of R^2 root nodes points to a TST.



Q. What about one- and two-letter words?

String symbol table implementation cost summary

implementation	character accesses (typical case)					dedup	
	search hit	search miss	insert	space (references)	moby.txt	actors.txt	
red-black BST	$L + c \lg^2 N$	$c \lg^2 N$	$c \lg^2 N$	$4 N$	1.40	97.4	
hashing (linear probing)	L	L	L	$4 N$ to $16 N$	0.76	40.6	
R-way trie	L	$\log_R N$	L	$(R + 1) N$	1.12	out of memory	
TST	$L + \ln N$	$\ln N$	$L + \ln N$	$4 N$	0.72	38.7	
TST with R^2	$L + \ln N$	$\ln N$	$L + \ln N$	$4 N + R^2$	0.51	32.7	

Bottom line. Faster than hashing for our benchmark client.

TST vs. hashing

Hashing.

- Need to examine entire key.
- Search hits and misses cost about the same.
- Performance relies on hash function.
- Does not support ordered symbol table operations.

TSTs.

- Works only for strings (or digital keys).
- Only examines just enough key characters.
- Search miss may involve only a few characters.
- Supports ordered symbol table operations (plus others!).

Bottom line. TSTs are:

- Faster than hashing (especially for search misses).
- More flexible than red-black BSTs. [stay tuned]

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

String symbol table API

Character-based operations. The string symbol table API supports several useful character-based operations.

key	value
by	4
sea	6
sells	1
she	0
shells	3
shore	7
the	5

Prefix match. Keys with prefix sh: she, sells, and shore.

Wildcard match. Keys that match .he: she and the.

Longest prefix. Key that is the longest prefix of shellsort: shells.

String symbol table API

```
public class StringST<Value>
```

```
    StringST()
```

create a symbol table with string keys

```
    void put(String key, Value val)
```

put key-value pair into the symbol table

```
    Value get(String key)
```

value paired with key

```
    void delete(String key)
```

delete key and corresponding value

```
    :
```

```
Iterable<String> keys()
```

all keys

```
Iterable<String> keysWithPrefix(String s)
```

keys having s as a prefix

```
Iterable<String> keysThatMatch(String s)
```

keys that match s (where . is a wildcard)

```
    String longestPrefixOf(String s)
```

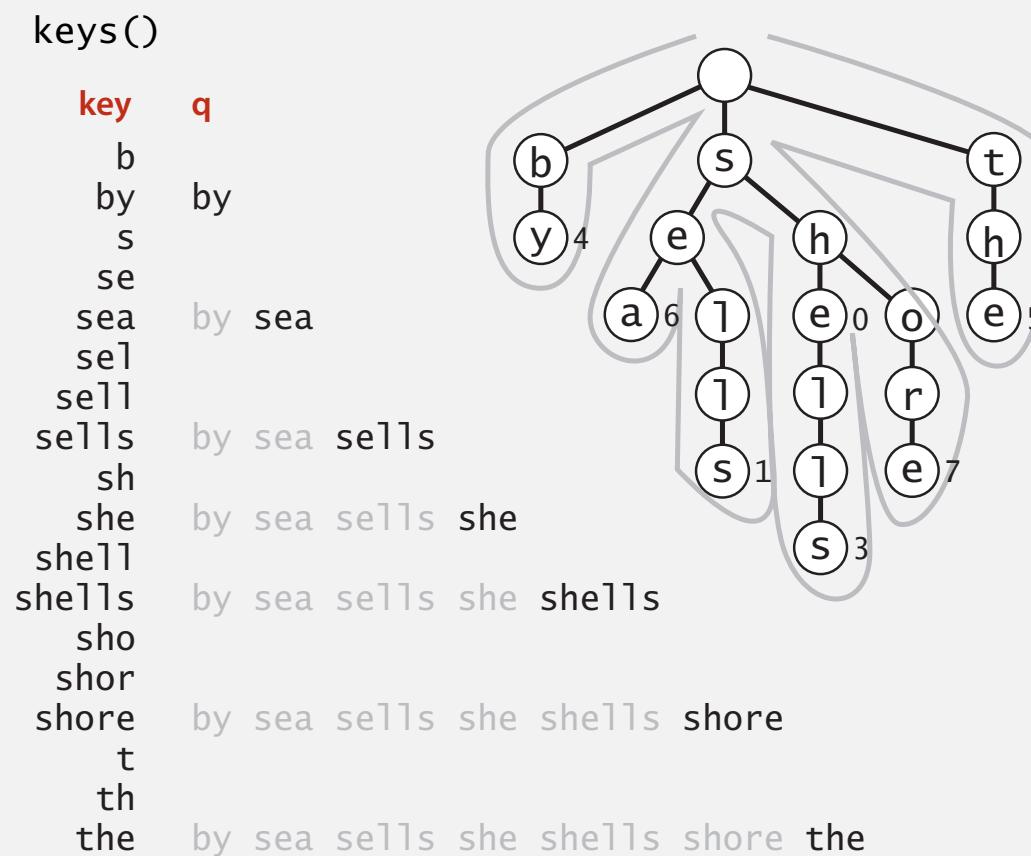
longest key that is a prefix of s

Remark. Can also add other ordered ST methods, e.g., floor() and rank().

Warmup: ordered iteration

To iterate through all keys in sorted order:

- Do inorder traversal of trie; add keys encountered to a queue.
 - Maintain sequence of characters on path from root to node.



Ordered iteration: Java implementation

To iterate through all keys in sorted order:

- Do inorder traversal of trie; add keys encountered to a queue.
- Maintain sequence of characters on path from root to node.

```
public Iterable<String> keys()
{
    Queue<String> queue = new Queue<String>();
    collect(root, "", queue);
    return queue;
}

private void collect(Node x, String prefix, Queue<String> q)
{
    if (x == null) return;
    if (x.val != null) q.enqueue(prefix);
    for (char c = 0; c < R; c++)
        collect(x.next[c], prefix + c, q);
}
```

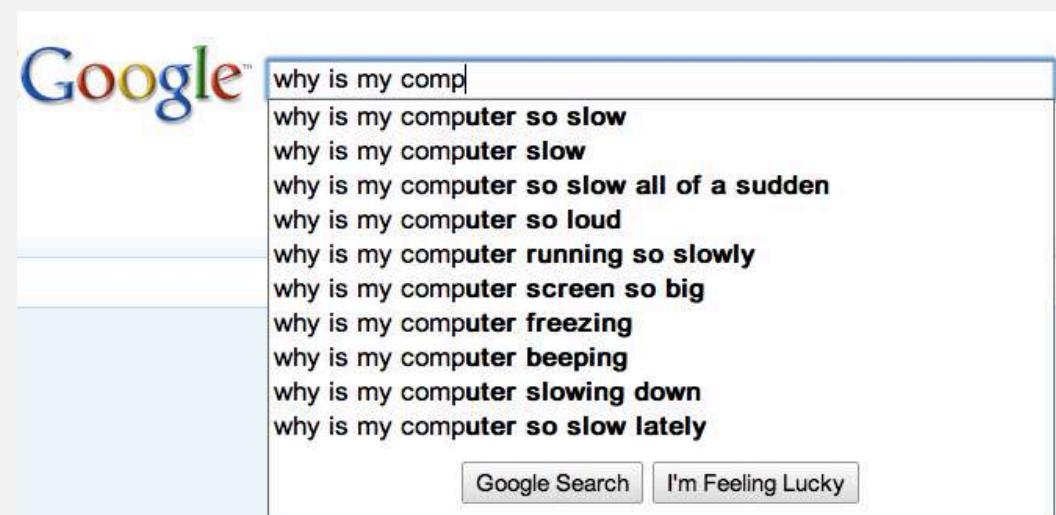
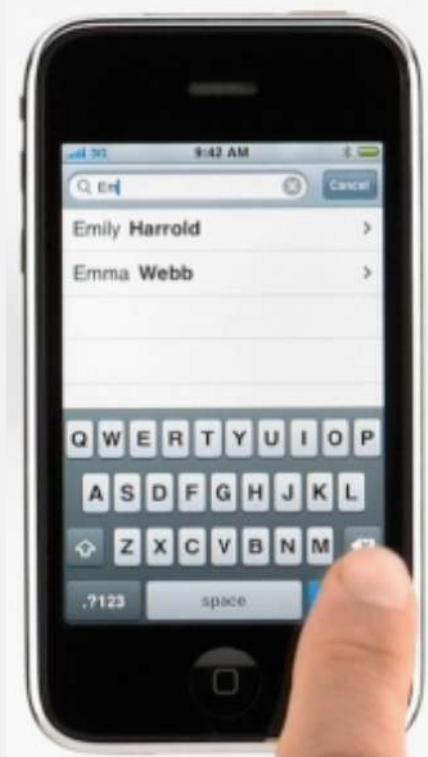
sequence of characters
on path from root to x

Prefix matches

Find all keys in a symbol table starting with a given prefix.

Ex. Autocomplete in a cell phone, search bar, text editor, or shell.

- User types characters one at a time.
- System reports all matching strings.



why is my comp

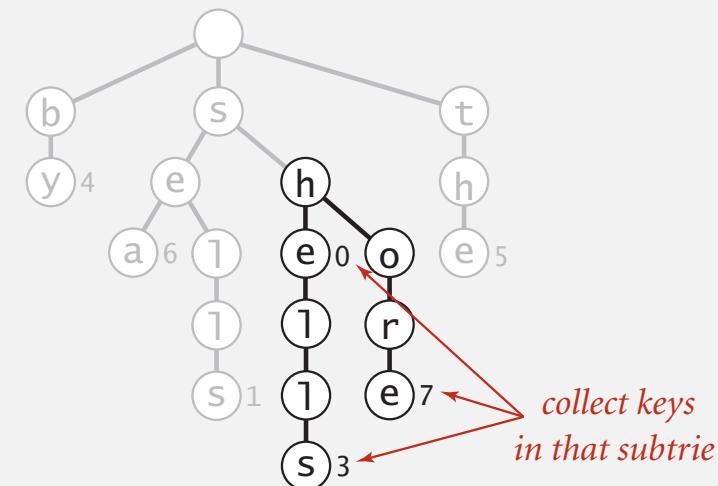
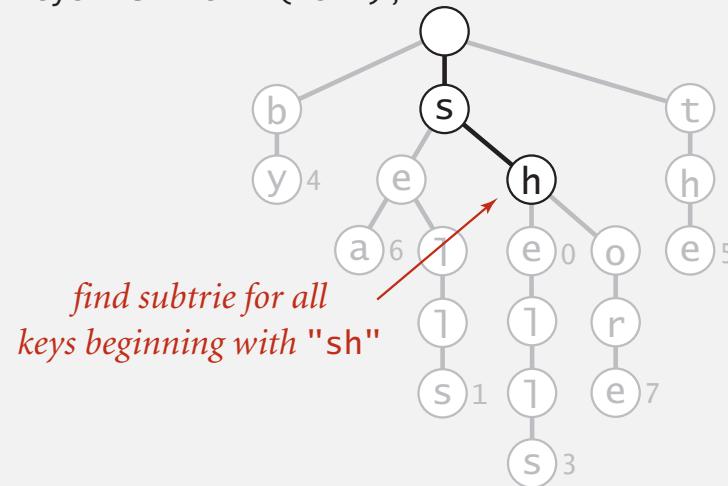
- why is my computer so slow
- why is my computer slow
- why is my computer so slow all of a sudden
- why is my computer so loud
- why is my computer running so slowly
- why is my computer screen so big
- why is my computer freezing
- why is my computer beeping
- why is my computer slowing down
- why is my computer so slow lately

Google Search I'm Feeling Lucky

Prefix matches in an R-way trie

Find all keys in a symbol table starting with a given prefix.

keysWithPrefix("sh");



```
public Iterable<String> keysWithPrefix(String prefix)
{
    Queue<String> queue = new Queue<String>();
    Node x = get(root, prefix, 0);
    collect(x, prefix, queue);
    return queue;
}
```

root of subtrie for all strings
beginning with given prefix

key	queue
sh	she
she	
shel	
shell	
shells	she shells
sho	
shor	
shore	she shells shore

Longest prefix

Find longest key in symbol table that is a prefix of query string.

Ex. To send packet toward destination IP address, router chooses IP address in routing table that is longest prefix match.

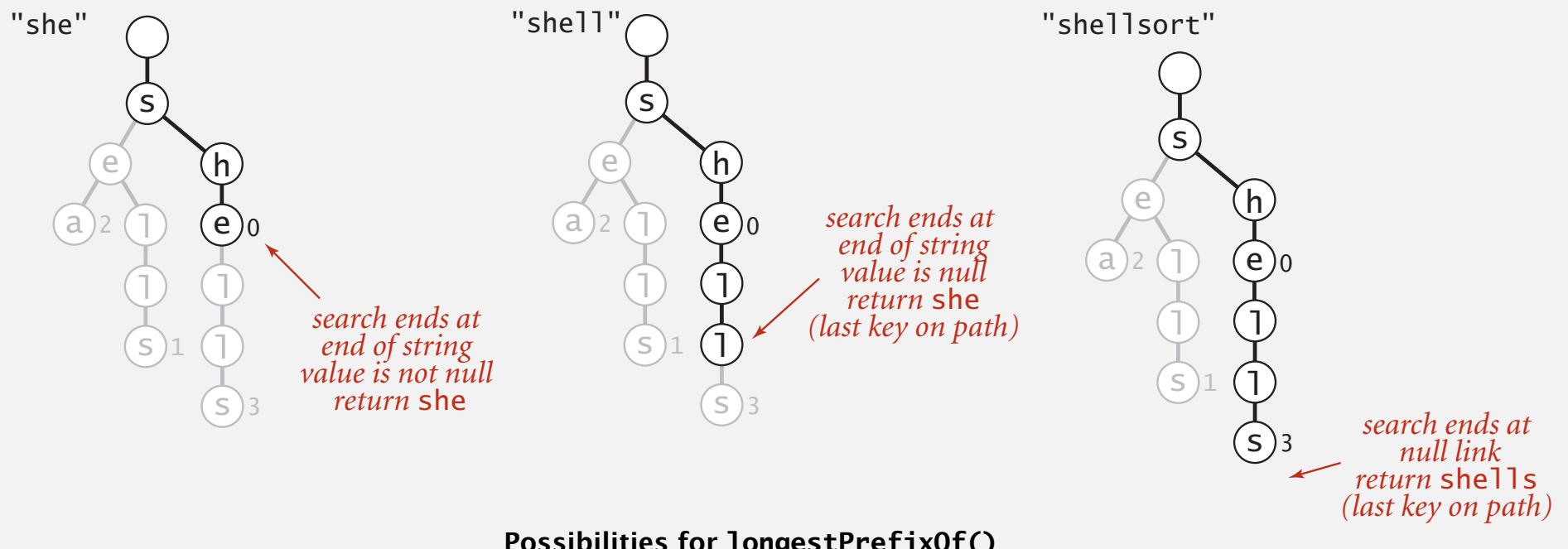
"128"	represented as 32-bit binary number for IPv4 (instead of string)
"128.112"	
"128.112.055"	
"128.112.055.15"	
"128.112.136"	<code>longestPrefixOf("128.112.136.11") = "128.112.136"</code>
"128.112.155.11"	<code>longestPrefixOf("128.112.100.16") = "128.112"</code>
"128.112.155.13"	<code>longestPrefixOf("128.166.123.45") = "128"</code>
"128.222"	
"128.222.136"	

Note. Not the same as floor: `floor("128.112.100.16") = "128.112.055.15"`

Longest prefix in an R-way trie

Find longest key in symbol table that is a prefix of query string.

- Search for query string.
- Keep track of longest key encountered.



Longest prefix in an R-way trie: Java implementation

Find longest key in symbol table that is a prefix of query string.

- Search for query string.
- Keep track of longest key encountered.

```
public String longestPrefixOf(String query)
{
    int length = search(root, query, 0, 0);
    return query.substring(0, length);
}

private int search(Node x, String query, int d, int length)
{
    if (x == null) return length;
    if (x.val != null) length = d;
    if (d == query.length()) return length;
    char c = query.charAt(d);
    return search(x.next[c], query, d+1, length);
}
```

T9 texting

Goal. Type text messages on a phone keypad.

Multi-tap input. Enter a letter by repeatedly pressing a key until the desired letter appears.

T9 text input.

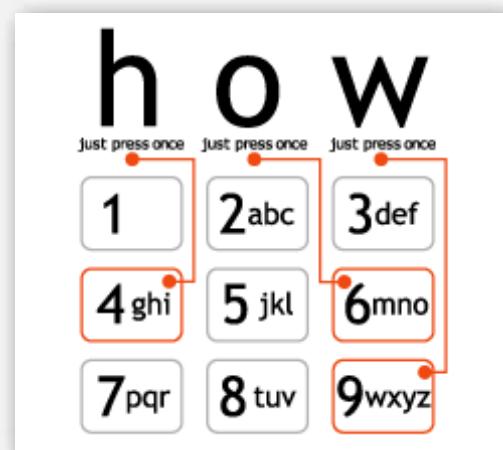
"a much faster and more fun way to enter text"



- Find all words that correspond to given sequence of numbers.
- Press 0 to see all completion options.

Ex. hello

- Multi-tap: 4 4 3 3 5 5 5 5 5 6 6 6
- T9: 4 3 5 5 6



www.t9.com

Q. How to implement?

A letter to t9.com

To: info@t9support.com

Date: Tue, 25 Oct 2005 14:27:21 -0400 (EDT)

Dear T9 texting folks,

I enjoyed learning about the T9 text system from your webpage, and used it as an example in my data structures and algorithms class. However, one of my students noticed a bug in your phone keypad

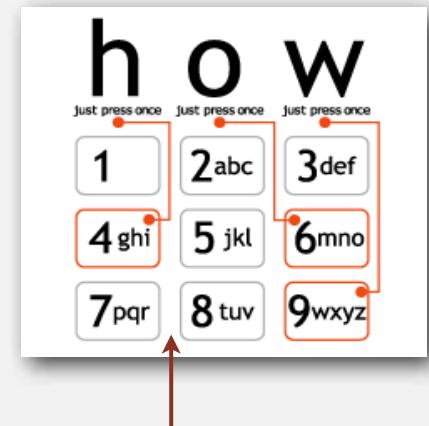
<http://www.t9.com/images/how.gif>

Somehow, it is missing the letter s. (!)

Just wanted to bring this information to your attention and thank you for your website.

Regards,

Kevin



where the @#\$% is the "s" ???

A world without 's' ?

To: "'Kevin Wayne'" <wayne@CS.Princeton.EDU>

Date: Tue, 25 Oct 2005 12:44:42 -0700

Thank you Kevin.

I am glad that you find T9 o valuable for your cla. I had not noticed thi before. Thank for writing in and letting u know.

Take care,

Brooke nyder
OEM Dev upport
AOL/Tegic Communication
1000 Dexter Ave N. uite 300
eattle, WA 98109

ALL INFORMATION CONTAINED IN THIS EMAIL IS CONSIDERED
CONFIDENTIAL AND PROPERTY OF AOL/TEGIC COMMUNICATIONS

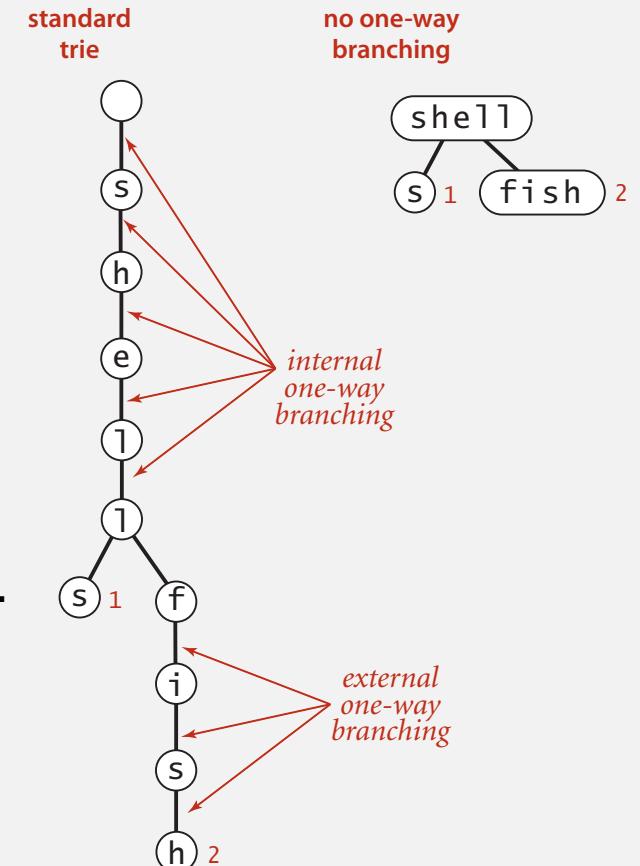
Patricia trie

Patricia trie. [Practical Algorithm to Retrieve Information Coded in Alphanumeric]

- Remove one-way branching.
- Each node represents a sequence of characters.
- Implementation: one step beyond this course.

```
put("shells", 1);
put("shellfish", 2);
```

standard
trie



Applications.

- Database search.
- P2P network search.
- IP routing tables: find longest prefix match.
- Compressed quad-tree for N-body simulation.
- Efficiently storing and querying XML documents.

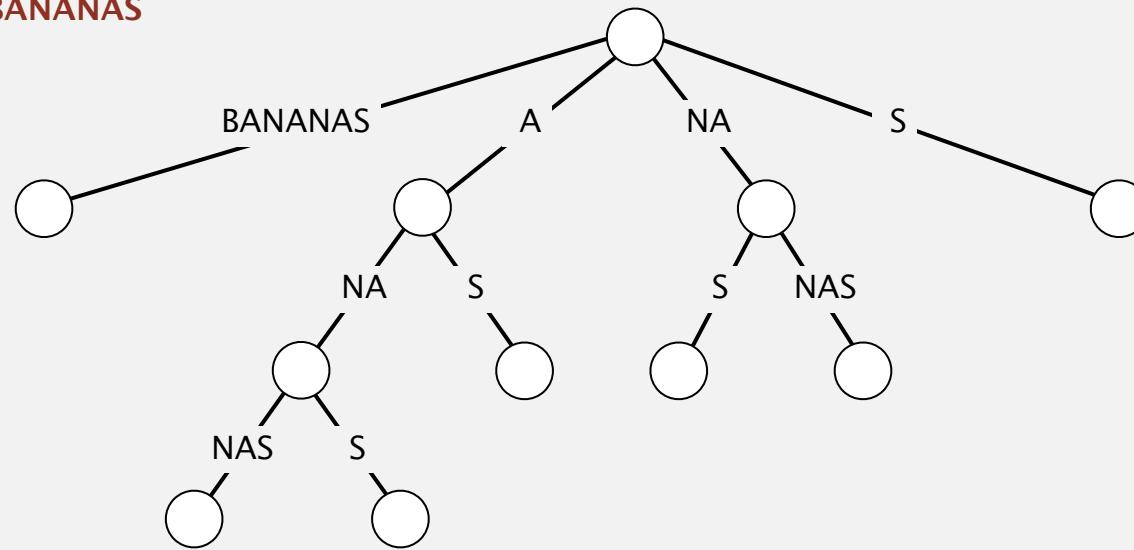
Also known as: crit-bit tree, radix tree.

Suffix tree

Suffix tree.

- Patricia trie of suffixes of a string.
- Linear-time construction: beyond this course.

suffix tree for BANANAS



Applications.

- Linear-time: longest repeated substring, longest common substring, longest palindromic substring, substring search, tandem repeats,
- Computational biology databases (BLAST, FASTA).

String symbol tables summary

A success story in algorithm design and analysis.

Red-black BST.

- Performance guarantee: $\log N$ key compares.
- Supports ordered symbol table API.

Hash tables.

- Performance guarantee: constant number of probes.
- Requires good hash function for key type.

Tries. R-way, TST.

- Performance guarantee: $\log N$ **characters** accessed.
- Supports character-based operations.

Bottom line. You can get at anything by examining 50-100 bits (!!?)

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

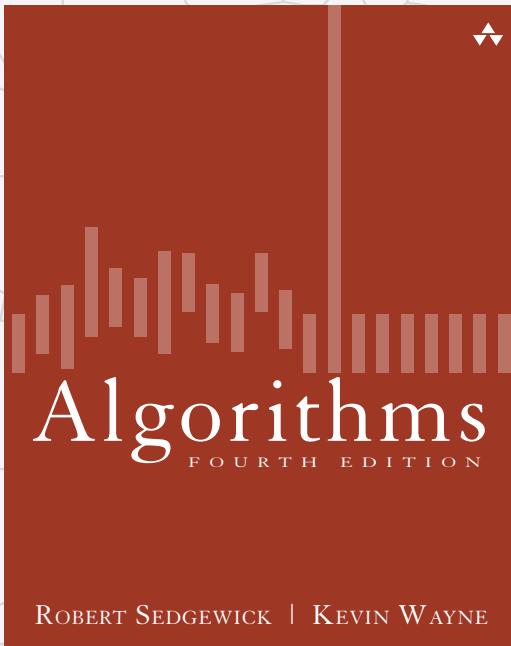
<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*

Algorithms

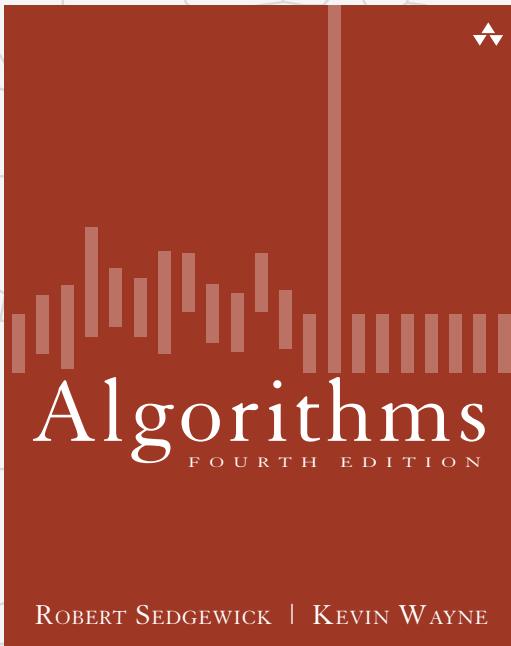
ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

5.2 TRIES

- ▶ *R-way tries*
- ▶ *ternary search tries*
- ▶ *character-based operations*



<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Substring search

Goal. Find pattern of length M in a text of length N .

 typically $N \gg M$

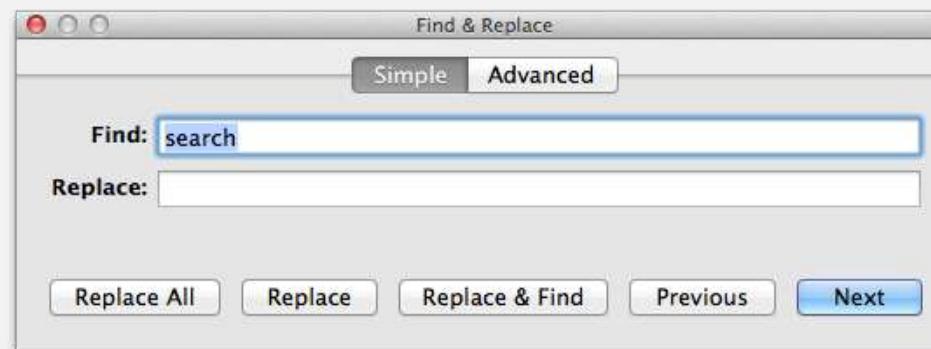
pattern → N E E D L E

Substring search applications

Goal. Find pattern of length M in a text of length N .

typically $N \gg M$

pattern → N E E D L E



Substring search applications

Goal. Find pattern of length M in a text of length N .

 typically $N \gg M$

pattern → N E E D L E

text → I N A H A Y S T A C K N E E D L E I N A
↑
match

Computer forensics. Search memory or disk for signatures, e.g., all URLs or RSA keys that the user has entered.



<http://citp.princeton.edu/memory>

Substring search applications

Goal. Find pattern of length M in a text of length N .

 typically $N \gg M$

pattern → N E E D L E

Identify patterns indicative of spam.

- PROFITS
 - LOSE WEIGHT
 - herbal Viagra
 - There is no catch.
 - This is a one-time mailing.
 - This message is sent in compliance with spam regulations.

SpamAssassin



Substring search applications

Electronic surveillance.



Need to monitor all
internet traffic.
(security)



Well, we're mainly
interested in
“ATTACK AT DAWN”

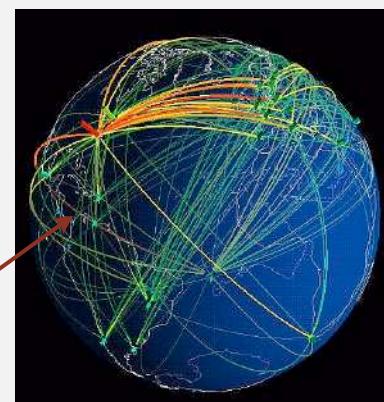
No way!
(privacy)



OK. Build a
machine that just
looks for that.



“ATTACK AT DAWN”
substring search
machine
found



Substring search applications

Screen scraping. Extract relevant data from web page.

Ex. Find string delimited by and after first occurrence of pattern Last Trade::



<http://finance.yahoo.com/q?s=goog>

```
...
<tr>
<td class= "yfnc_tablehead1"
width= "48%">
Last Trade:
</td>
<td class= "yfnc_tabledata1">
<big><b>452.92</b></big>
</td></tr>
<td class= "yfnc_tablehead1"
width= "48%">
Trade Time:
</td>
<td class= "yfnc_tabledata1">
...

```

Screen scraping: Java implementation

Java library. The `indexOf()` method in Java's string library returns the index of the first occurrence of a given string, starting at a given offset.

```
public class StockQuote
{
    public static void main(String[] args)
    {
        String name = "http://finance.yahoo.com/q?s=";
        In in = new In(name + args[0]);
        String text = in.readAll();
        int start      = text.indexOf("Last Trade:", 0);
        int from       = text.indexOf("<b>", start);
        int to         = text.indexOf("</b>", from);
        String price = text.substring(from + 3, to);
        StdOut.println(price);
    }
}
```

```
% java StockQuote goog  
582.93
```

```
% java StockQuote msft  
24.84
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ ***brute force***
- ▶ ***Knuth-Morris-Pratt***
- ▶ ***Boyer-Moore***
- ▶ ***Rabin-Karp***

Brute-force substring search

Check for pattern starting at each text position.

i	j	i+j	0	1	2	3	4	5	6	7	8	9	10
			A	B	A	C	A	D	A	B	R	A	C
0	2	2	A	B	R	A							
1	0	1		A	B	R	A						
2	1	3			A	B	R	A					
3	0	3				A	B	R	A				
4	1	5					A	B	R	A			
5	0	5						A	B	R	A		
6	4	10							A	B	R	A	

txt → A B A C A D A B R A C

entries in red are mismatches

entries in gray are for reference only

entries in black match the text

return i when j is M

match

Brute-force substring search: Java implementation

Check for pattern starting at each text position.

i	j	i + j	0	1	2	3	4	5	6	7	8	9	10
			A	B	A	C	A	D	A	B	R	A	C
4	3	7					A	D	A	C	R		
5	0	5						A	D	A	C	R	

```
public static int search(String pat, String txt)
{
    int M = pat.length();
    int N = txt.length();
    for (int i = 0; i <= N - M; i++)
    {
        int j;
        for (j = 0; j < M; j++)
            if (txt.charAt(i+j) != pat.charAt(j))
                break;
        if (j == M) return i; ← index in text where
                           pattern starts
    }
    return N; ← not found
}
```

Brute-force substring search: worst case

Brute-force algorithm can be slow if text and pattern are repetitive.

i	j	$i+j$	0	1	2	3	4	5	6	7	8	9
			txt →	A	A	A	A	A	A	A	A	B
0	4	4	A	A	A	A	B	← pat				
1	4	5		A	A	A	A	B				
2	4	6			A	A	A	A	B			
3	4	7				A	A	A	A	B		
4	4	8					A	A	A	A	B	
5	5	10						<u>A</u>	<u>A</u>	<u>A</u>	<u>A</u>	B

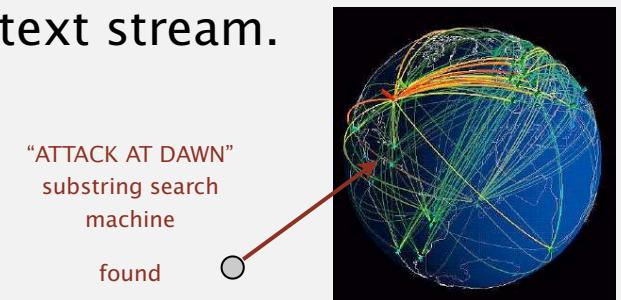
\uparrow
match

Worst case. $\sim MN$ char compares.

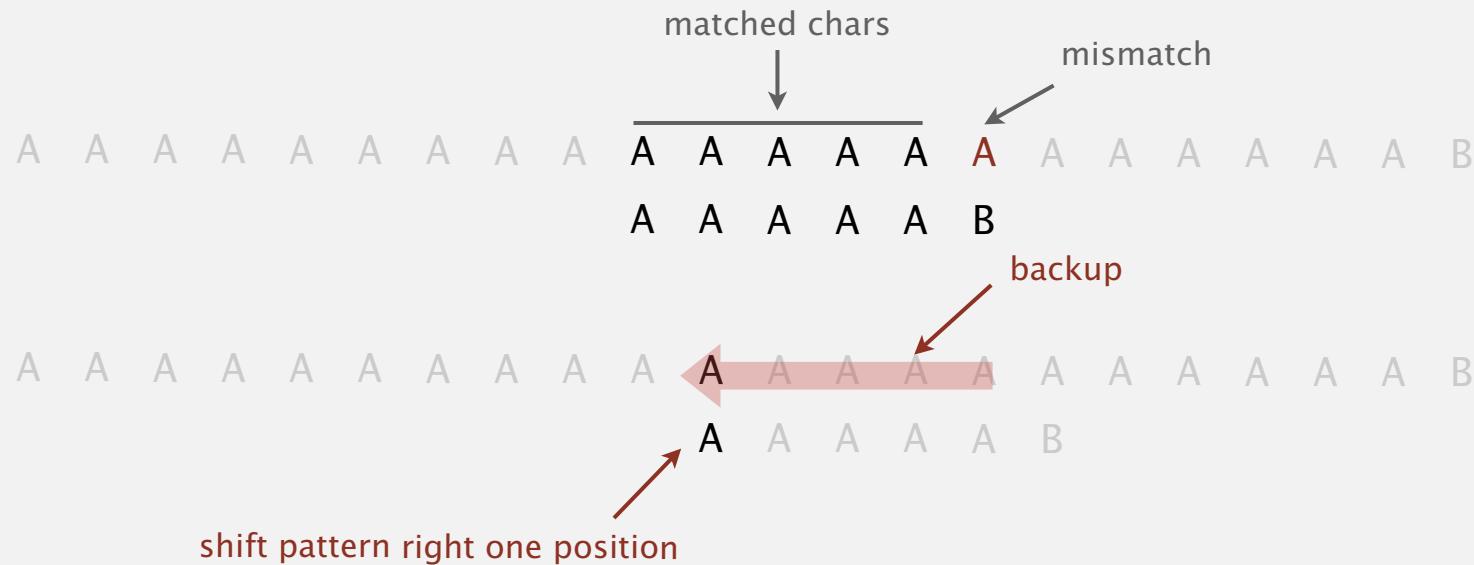
Backup

In many applications, we want to avoid **backup** in text stream.

- Treat input as stream of data.
- Abstract model: standard input.



Brute-force algorithm needs backup for every mismatch.



Approach 1. Maintain buffer of last M characters.

Approach 2. Stay tuned.

Brute-force substring search: alternate implementation

Same sequence of char compares as previous implementation.

- i points to end of sequence of already-matched chars in text.
- j stores # of already-matched chars (end of sequence in pattern).

<u>i</u>	<u>j</u>	0	1	2	3	4	5	6	7	8	9	10
		A	B	A	C	A	D	A	B	R	A	C
7	3				A	D	A	C	R			
5	0				A	D	A	C	R			

```
public static int search(String pat, String txt)
{
    int i, N = txt.length();
    int j, M = pat.length();
    for (i = 0, j = 0; i < N && j < M; i++)
    {
        if (txt.charAt(i) == pat.charAt(j)) j++;
        else { i -= j; j = 0; } ← explicit backup
    }
    if (j == M) return i - M;
    else return N;
}
```

Algorithmic challenges in substring search

Brute-force is not always good enough.

Theoretical challenge. Linear-time guarantee. ← fundamental algorithmic problem

Practical challenge. Avoid backup in text stream. ← often no room or time to save text

Now is the time for all people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for a lot of good people to come to the aid of their party. Now is the time for all of the good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for each good person to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many or all good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for a lot of good people to come to the aid of their party. Now is the time for all of the good people to come to the aid of their party. Now is the time for all good people to come to the aid of their **attack at dawn** party. Now is the time for each person to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many or all good people to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ ***brute force***
- ▶ ***Knuth-Morris-Pratt***
- ▶ ***Boyer-Moore***
- ▶ ***Rabin-Karp***

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

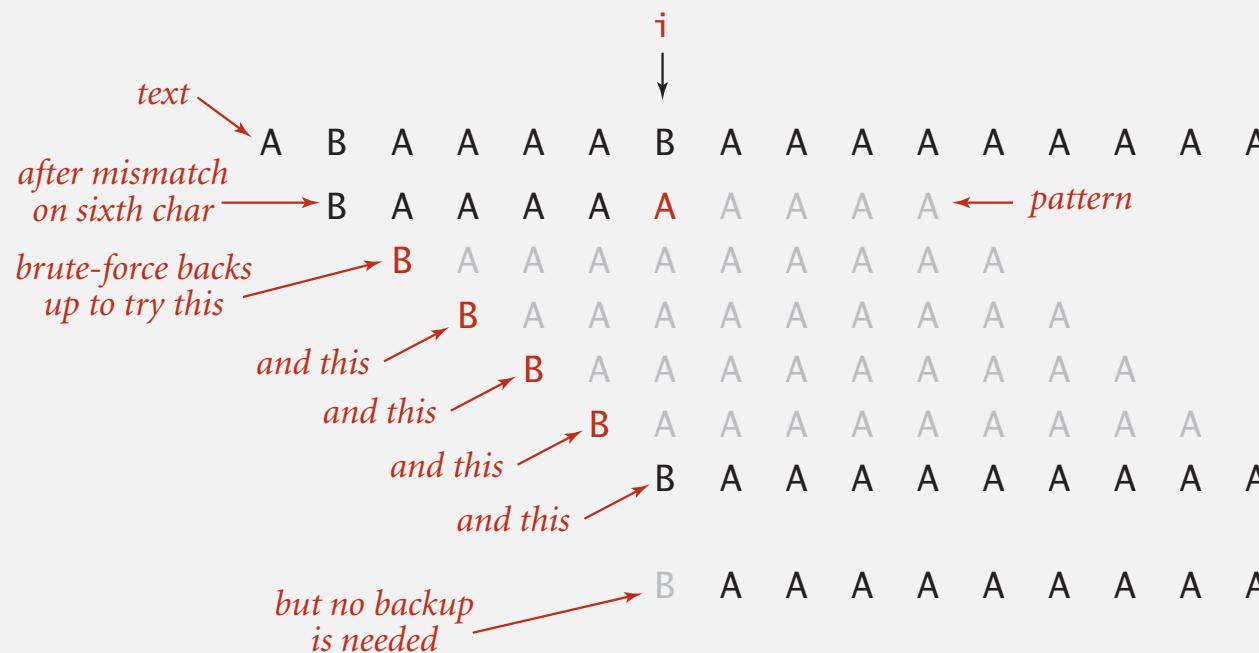
- ▶ *introduction*
- ▶ *brute force*
- ▶ ***Knuth-Morris-Pratt***
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Knuth-Morris-Pratt substring search

Intuition. Suppose we are searching in text for pattern BAAAAAAAAAA.

- Suppose we match 5 chars in pattern, with mismatch on 6th char.
- We know previous 6 chars in text are BAAAAB.
- Don't need to back up text pointer!

assuming { A, B } alphabet



Knuth-Morris-Pratt algorithm. Clever method to always avoid backup. (!)

Deterministic finite state automaton (DFA)

DFA is abstract string-searching machine.

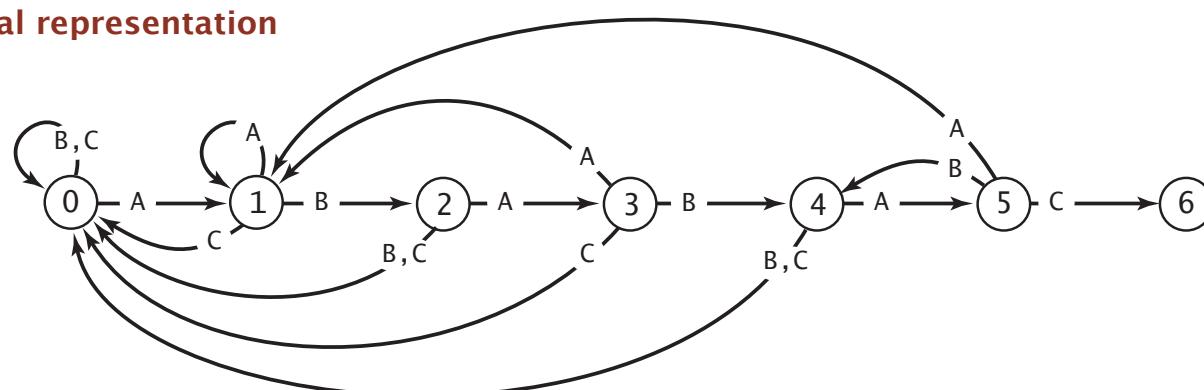
- Finite number of states (including start and halt).
- Exactly one transition for each char in alphabet.
- Accept if sequence of transitions leads to halt state.

internal representation

j	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	1	1	3	1	5	1
	0	2	0	4	0	4
	0	0	0	0	0	6

If in state j reading char C:
if j is 6 halt and accept
else move to state dfa[c][j]

graphical representation

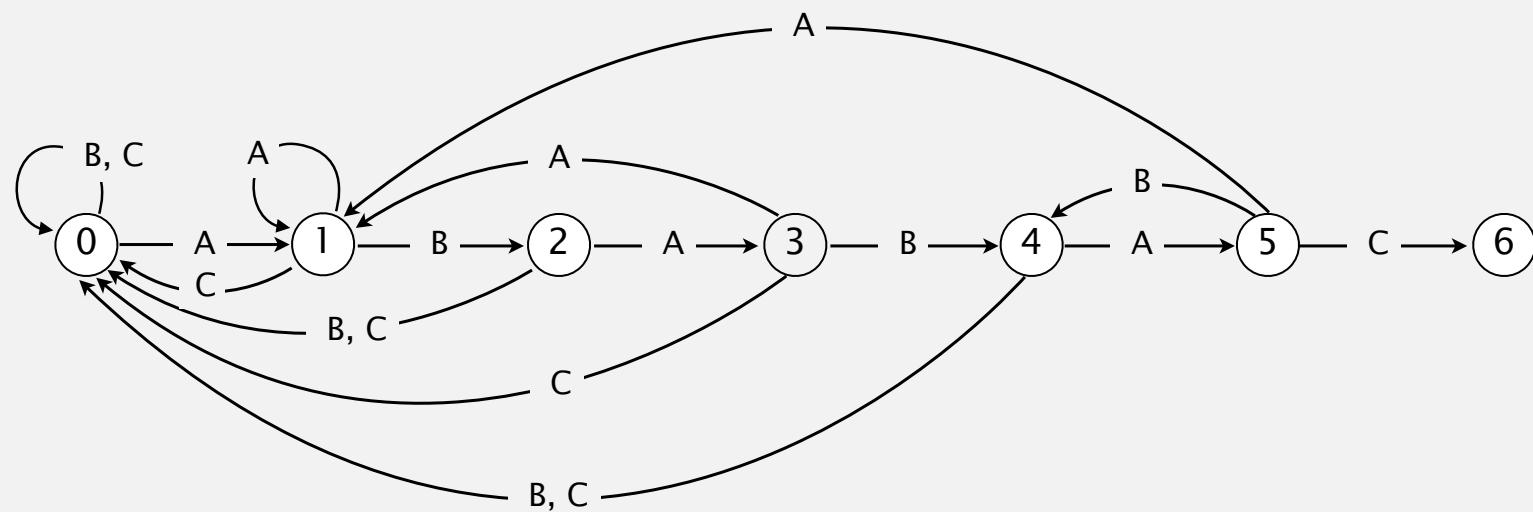


DFA simulation demo

A A B A C A A B A B A C A A



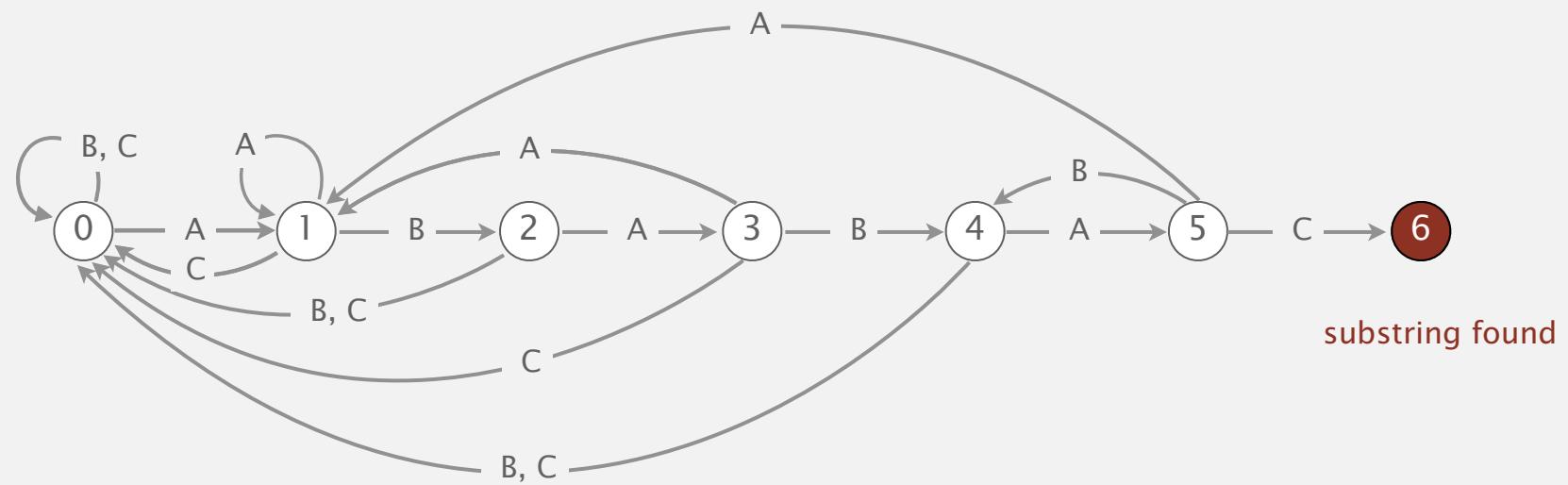
		0	1	2	3	4	5
pat.charAt(j)	A	A	B	A	B	A	C
	B	1	1	3	1	5	1
dfa[][][j]	C	0	2	0	4	0	4
	C	0	0	0	0	0	6



DFA simulation demo

A A B A C A A B A B A C A A
↑

pat.charAt(j)	0	1	2	3	4	5
A	A	B	A	B	A	C
dfa[][][j]	1	1	3	1	5	1
B	0	2	0	4	0	4
C	0	0	0	0	0	6



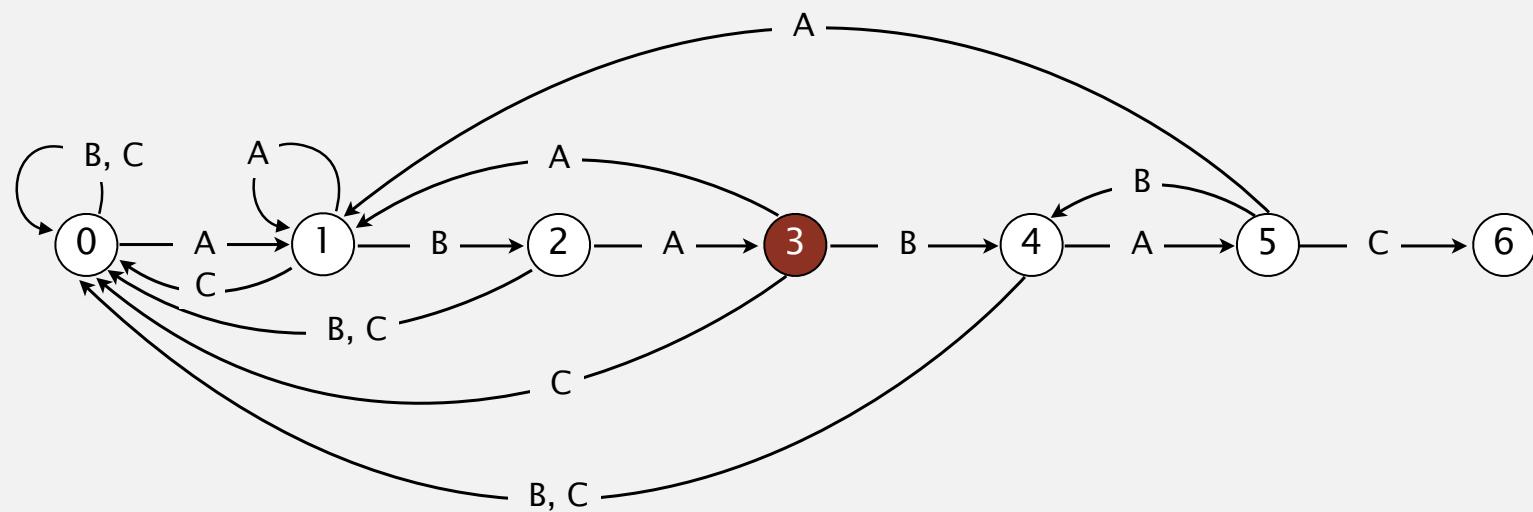
Interpretation of Knuth-Morris-Pratt DFA

Q. What is interpretation of DFA state after reading in $\text{txt}[i]$?

A. State = number of characters in pattern that have been matched.

length of longest prefix of $\text{pat}[]$
that is a suffix of $\text{txt}[0..i]$

Ex. DFA is in state 3 after reading in $\text{txt}[0..6]$.



Knuth-Morris-Pratt substring search: Java implementation

Key differences from brute-force implementation.

- Need to precompute $\text{dfa}[][]$ from pattern.
- Text pointer i never decrements.

```
public int search(String txt)
{
    int i, j, N = txt.length();
    for (i = 0, j = 0; i < N && j < M; i++)
        j = dfa[txt.charAt(i)][j];           ← no backup
    if (j == M) return i - M;
    else         return N;
}
```

Running time.

- Simulate DFA on text: at most N character accesses.
- Build DFA: how to do efficiently? [warning: tricky algorithm ahead]

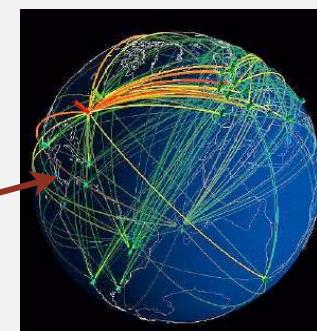
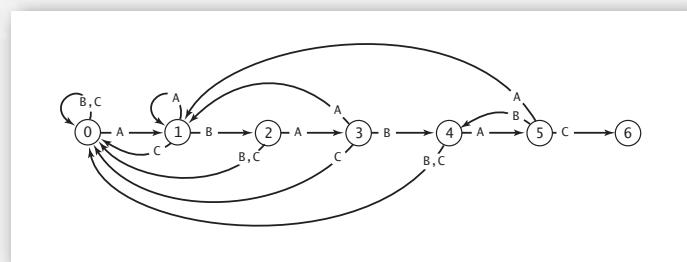
Knuth-Morris-Pratt substring search: Java implementation

Key differences from brute-force implementation.

- Need to precompute `dfa[][]` from pattern.
- Text pointer `i` never decrements.
- Could use **input stream**.

```
public int search(In in)
{
    int i, j;
    for (i = 0, j = 0; !in.isEmpty() && j < M; i++)
        j = dfa[in.readChar()][j];
    if (j == M) return i - M;
    else         return NOT_FOUND;
}
```

no backup



Knuth-Morris-Pratt construction demo

Include one state for each character in pattern (plus accept state).



pat.charAt(j)	0	1	2	3	4	5
A	A	B	A	B	A	C
dfa[][][j]	A	B				
C						

Constructing the DFA for KMP substring search for A B A B A C

0

1

2

3

4

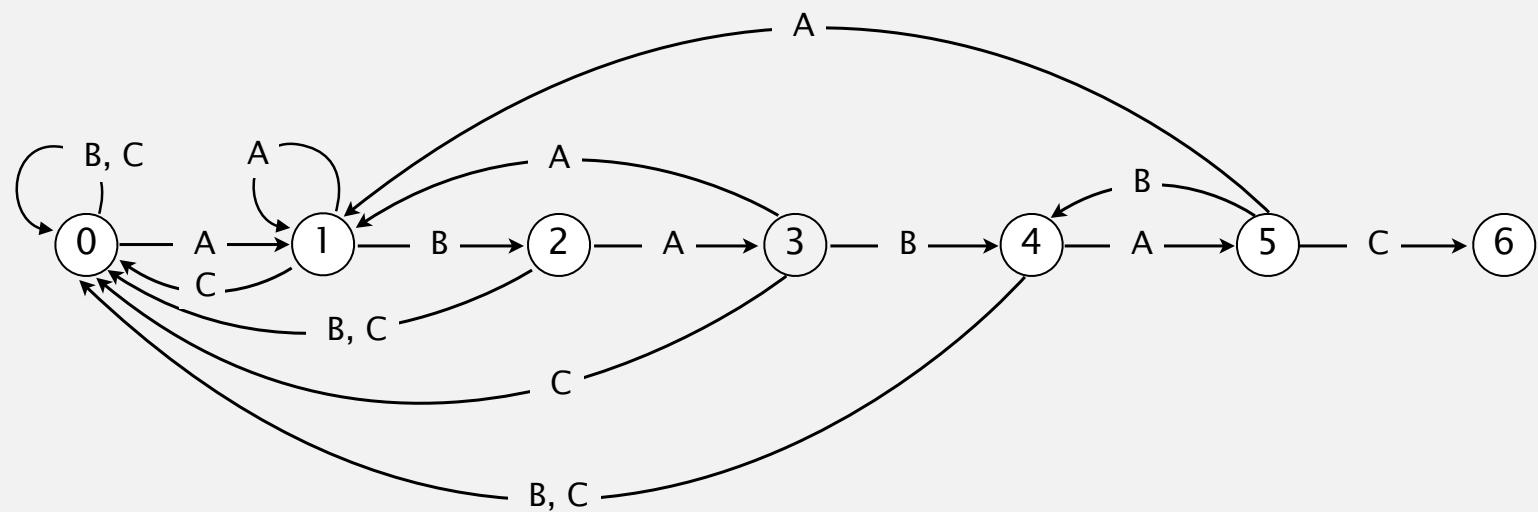
5

6

Knuth-Morris-Pratt construction demo

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
A	1	1	3	1	5	1
dfa[][][j]	B	0	2	0	4	0
C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



How to build DFA from pattern?

Include one state for each character in pattern (plus accept state).



How to build DFA from pattern?

Match transition. If in state j and next char $c == \text{pat.charAt}(j)$, go to $j+1$.

↑
first j characters of pattern
have already been matched ↑
next char matches ↑
now first $j+1$ characters of
pattern have been matched

		0	1	2	3	4	5
pat.charAt(j)	A	A	B	A	B	A	C
	B						
	C						6



How to build DFA from pattern?

Mismatch transition. If in state j and next char $c \neq \text{pat.charAt}(j)$, then the last $j-1$ characters of input are $\text{pat}[1..j-1]$, followed by c .

To compute $\text{dfa}[c][j]$: Simulate $\text{pat}[1..j-1]$ on DFA and take transition c .

Running time. Seems to require j steps.

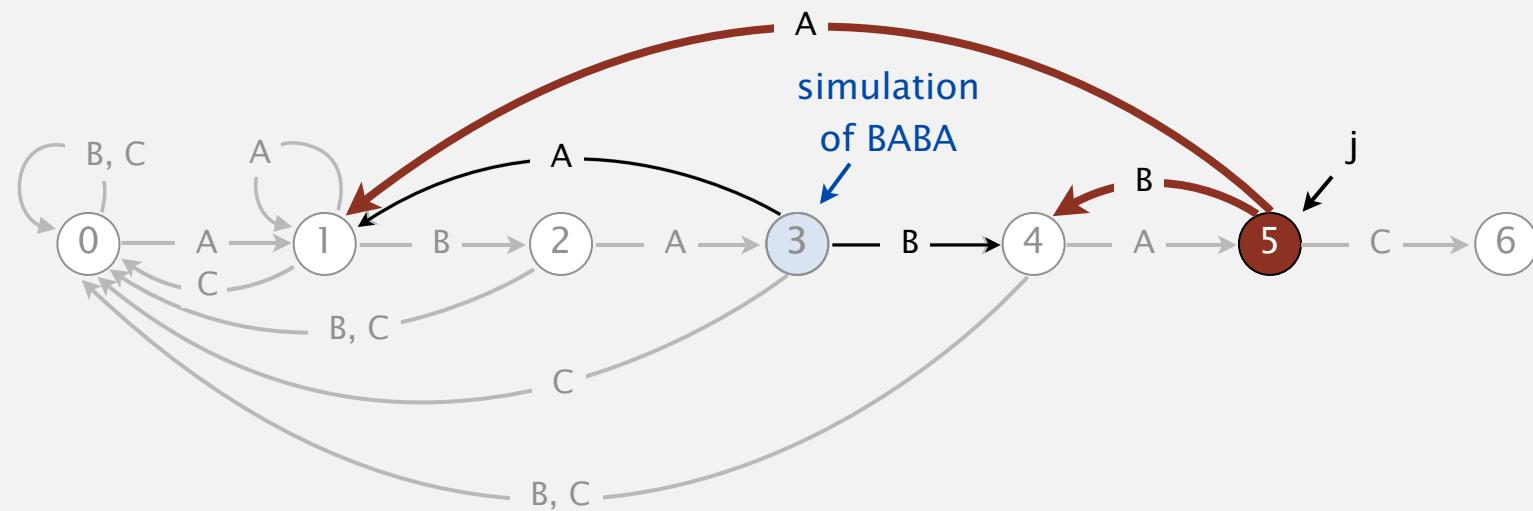
still under construction (!)

Ex. $\text{dfa}['A'][5] = 1$; $\text{dfa}['B'][5] = 4$

simulate BABA;
take transition 'A'
 $= \text{dfa}['A'][3]$

simulate BABA;
take transition 'B'
 $= \text{dfa}['B'][3]$

	j	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C	



How to build DFA from pattern?

Mismatch transition. If in state j and next char $c \neq \text{pat.charAt}(j)$, then the last $j-1$ characters of input are $\text{pat}[1..j-1]$, followed by c .

To compute $\text{dfa}[c][j]$: Simulate $\text{pat}[1..j-1]$ on DFA and take transition c .
Running time. Takes only constant time if we maintain state X .

Ex. $\text{dfa}['A'][5] = 1$; $\text{dfa}['B'][5] = 4$;

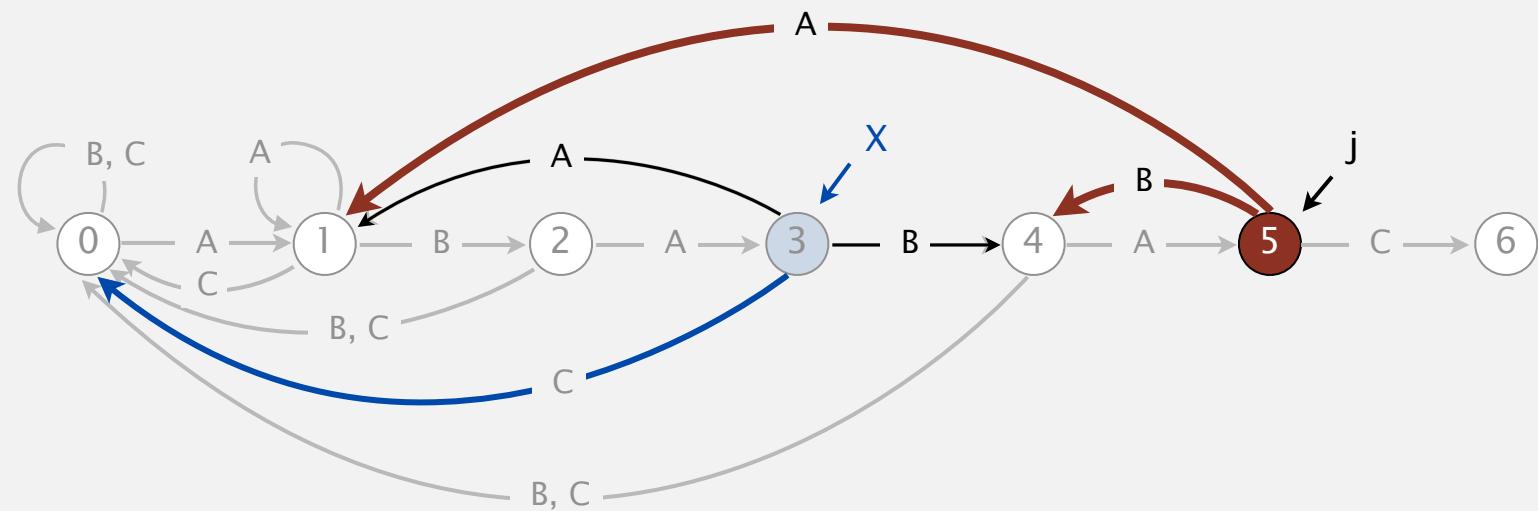
from state X ,
take transition 'A'
 $= \text{dfa}['A'][X]$

from state X ,
take transition 'B'
 $= \text{dfa}['B'][X]$

$X' = 0$

from state X ,
take transition 'C'
 $= \text{dfa}['C'][X]$

0	1	2	3	4	5
A	B	A	B	A	C



Knuth-Morris-Pratt construction demo (in linear time)

Include one state for each character in pattern (plus accept state).



pat.charAt(j)	0	1	2	3	4	5
A	A	B	A	B	A	C
dfa[][][j]	A	B				
	C					

Constructing the DFA for KMP substring search for A B A B A C

0

1

2

3

4

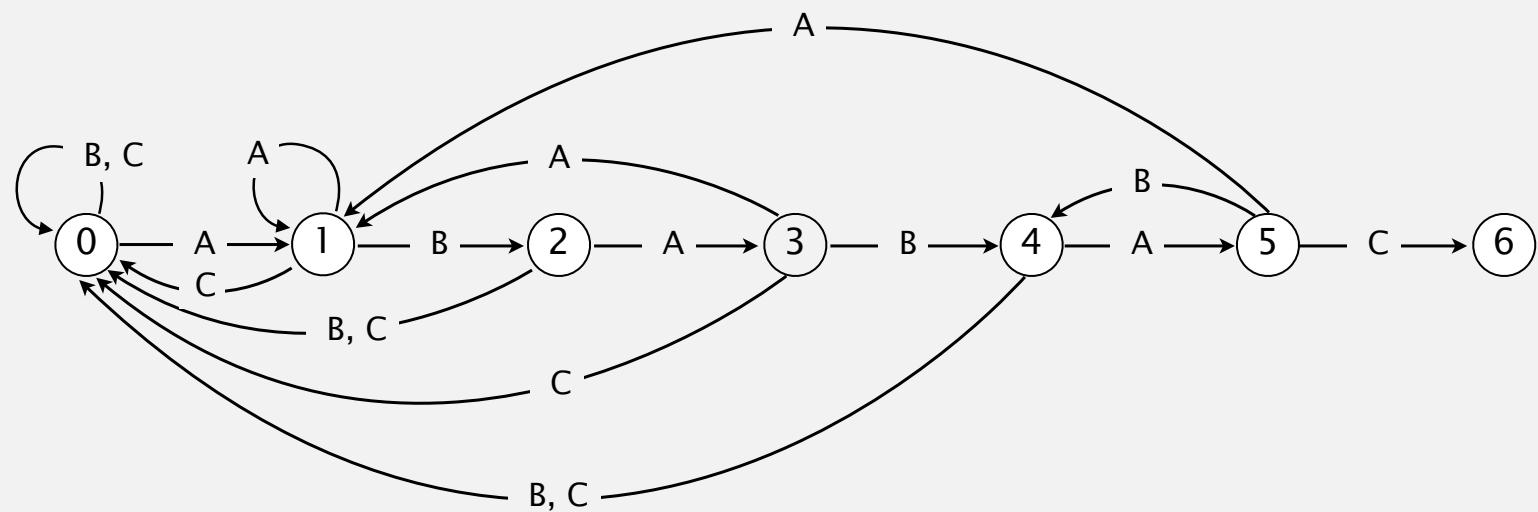
5

6

Knuth-Morris-Pratt construction demo (in linear time)

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
A	1	1	3	1	5	1
dfa[][][j]	B	0	2	0	4	0
C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



Constructing the DFA for KMP substring search: Java implementation

For each state j :

- Copy $\text{dfa}[][\text{X}]$ to $\text{dfa}[][\text{j}]$ for mismatch case.
- Set $\text{dfa}[\text{pat.charAt(j)}][\text{j}]$ to $\text{j}+1$ for match case.
- Update X .

```
public KMP(String pat)
{
    this.pat = pat;
    M = pat.length();
    dfa = new int[R][M];
    dfa[pat.charAt(0)][0] = 1;
    for (int X = 0, j = 1; j < M; j++)
    {
        for (int c = 0; c < R; c++)
            dfa[c][j] = dfa[c][X]; ← copy mismatch cases
        dfa[pat.charAt(j)][j] = j+1; ← set match case
        X = dfa[pat.charAt(j)][X]; ← update restart state
    }
}
```

Running time. M character accesses (but space/time proportional to $R M$).

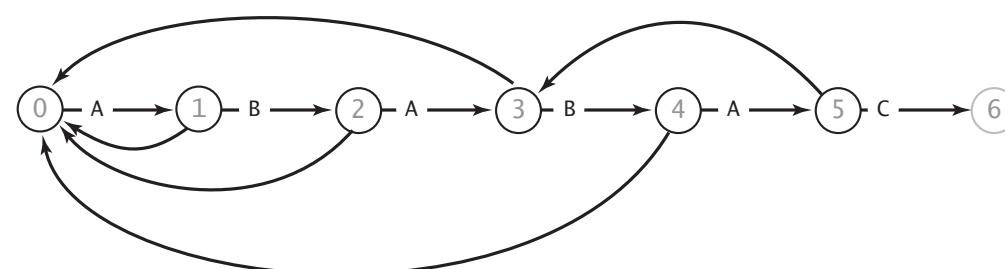
KMP substring search analysis

Proposition. KMP substring search accesses no more than $M + N$ chars to search for a pattern of length M in a text of length N .

Pf. Each pattern char accessed once when constructing the DFA; each text char accessed once (in the worst case) when simulating the DFA.

Proposition. KMP constructs `dfa[][]` in time and space proportional to $R M$.

Larger alphabets. Improved version of KMP constructs `nfa[]` in time and space proportional to M .



Knuth-Morris-Pratt: brief history

- Independently discovered by two theoreticians and a hacker.
 - Knuth: inspired by esoteric theorem, discovered linear algorithm
 - Pratt: made running time independent of alphabet size
 - Morris: built a text editor for the CDC 6400 computer
- Theory meets practice.

SIAM J. COMPUT.
Vol. 6, No. 2, June 1977

FAST PATTERN MATCHING IN STRINGS*

DONALD E. KNUTH†, JAMES H. MORRIS, JR.‡ AND VAUGHAN R. PRATT¶

Abstract. An algorithm is presented which finds all occurrences of one given string within another, in running time proportional to the sum of the lengths of the strings. The constant of proportionality is low enough to make this algorithm of practical use, and the procedure can also be extended to deal with some more general pattern-matching problems. A theoretical application of the algorithm shows that the set of concatenations of even palindromes, i.e., the language $\{\alpha\alpha^R\}^*$, can be recognized in linear time. Other algorithms which run even faster on the average are also considered.



Don Knuth



Jim Morris



Vaughan Pratt

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ ***Knuth-Morris-Pratt***
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

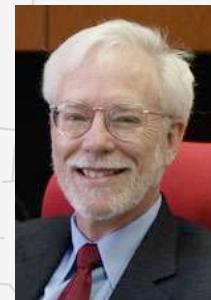
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*



Robert Boyer J. Strother Moore

Boyer-Moore: mismatched character heuristic

Intuition.

- Scan characters in pattern from right to left.
- Can skip as many as M text chars when finding one not in the pattern.

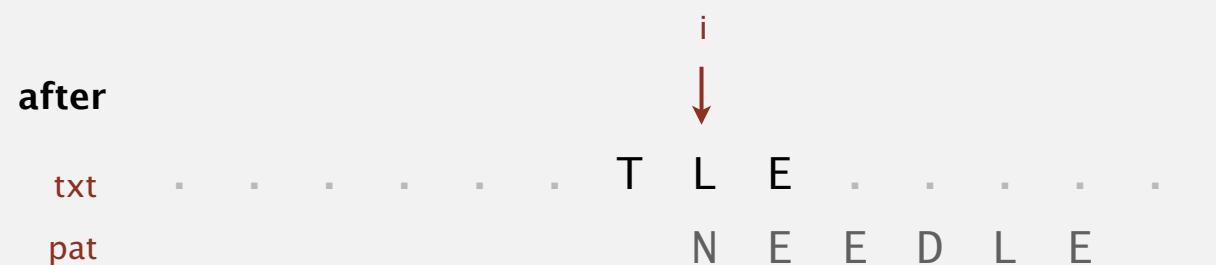
i	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		F	I	N	D	I	N	A	H	A	Y	S	T	A	C	K	N	E	E	D	L	E	I	N	A
		text →																							
0	5	N	E	E	D	L	E	← pattern																	
5	5								N	E	E	D	L	E											
11	4														N	E	E	D	L	E					
15	0																	N	E	E	D	L	E		

return i = 15

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 1. Mismatch character not in pattern.

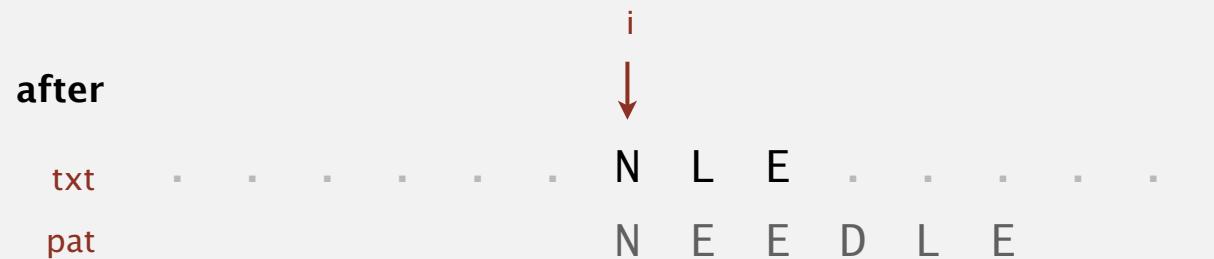


mismatch character 'T' not in pattern: increment i one character beyond 'T'

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2a. Mismatch character in pattern.



mismatch character 'N' in pattern: align text 'N' with rightmost pattern 'N'

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2b. Mismatch character in pattern (but heuristic no help).



mismatch character 'E' in pattern: align text 'E' with rightmost pattern 'E' ?

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2b. Mismatch character in pattern (but heuristic no help).

											i		
before	E	L	E
txt
pat			N	E	E	D	L	E					

											i		
after	E	L	E
txt
pat			N	E	E	D	L	E					

mismatch character 'E' in pattern: increment i by 1

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

A. Precompute index of rightmost occurrence of character c in pattern
(-1 if character not in pattern).

```
right = new int[R];
for (int c = 0; c < R; c++)
    right[c] = -1;
for (int j = 0; j < M; j++)
    right[pat.charAt(j)] = j;
```

c	N	E	E	D	L	E	right[c]
	0	1	2	3	4	5	
A	-1	-1	-1	-1	-1	-1	-1
B	-1	-1	-1	-1	-1	-1	-1
C	-1	-1	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	3	3	3
E	-1	-1	1	2	2	5	5
...							-1
L	-1	-1	-1	-1	-1	4	4
M	-1	-1	-1	-1	-1	-1	-1
N	-1	0	0	0	0	0	0
...							-1

Boyer-Moore skip table computation

Boyer-Moore: Java implementation

```
public int search(String txt)
{
    int N = txt.length();
    int M = pat.length();
    int skip;
    for (int i = 0; i <= N-M; i += skip)
    {
        skip = 0;
        for (int j = M-1; j >= 0; j--)
        {
            if (pat.charAt(j) != txt.charAt(i+j))
            {
                skip = Math.max(1, j - right[txt.charAt(i+j)]);
                break;
            }
        }
        if (skip == 0) return i; ← match
    }
    return N;
}
```

compute
skip value

in case other term is nonpositive

Boyer-Moore: analysis

Property. Substring search with the Boyer-Moore mismatched character heuristic takes about $\sim N/M$ character compares to search for a pattern of length M in a text of length N .

sublinear!

Worst-case. Can be as bad as $\sim MN$.

i	skip	0	1	2	3	4	5	6	7	8	9
	txt →	B	B	B	B	B	B	B	B	B	B
0	0	A	B	B	B	B	B	B	B	B	B
1	1		A	B	B	B	B				
2	1			A	B	B	B	B			
3	1				A	B	B	B	B		
4	1					A	B	B	B	B	
5	1						A	B	B	B	B

Boyer-Moore variant. Can improve worst case to $\sim 3N$ character compares by adding a KMP-like rule to guard against repetitive patterns.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ **Boyer-Moore**
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ ***Rabin-Karp***



Michael Rabin, Turing Award '76

Dick Karp, Turing Award '85

Rabin-Karp fingerprint search

Basic idea = modular hashing.

- Compute a hash of pattern characters 0 to $M - 1$.
- For each i , compute a hash of text characters i to $M + i - 1$.
- If pattern hash = text substring hash, check for a match.

pat.charAt(i)																			
i	0	1	2	3	4														
	2	6	5	3	5														
$\% 997 = 613$																			
txt.charAt(i)																			
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13					
	3	1	4	1	5	9	2	6	5	3	5	8	9	7					
0	3	1	4	1	5	$\% 997 = 508$													
1		1	4	1	5	9	$\% 997 = 201$												
2			4	1	5	9	2	$\% 997 = 715$											
3				1	5	9	2	6	$\% 997 = 971$										
4					5	9	2	6	5	$\% 997 = 442$									
5						9	2	6	5	3	$\% 997 = 929$								
6	\leftarrow	$\text{return } i = 6$				2	6	5	3	5	$\% 997 = 613$								

Efficiently computing the hash function

Modular hash function. Using the notation t_i for `txt.charAt(i)`, we wish to compute

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0 \pmod{Q}$$

Intuition. M -digit, base- R integer, modulo Q .

Horner's method. Linear-time method to evaluate degree- M polynomial.

pat.charAt()					
i	0	1	2	3	4
	2	6	5	3	5
0	2	% 997 = 2			
1	2	6	% 997 = (2*10 + 6) % 997 = 26	<i>R</i>	<i>Q</i>
2	2	6	5 % 997 = (26*10 + 5) % 997 = 265		
3	2	6	5 3 % 997 = (265*10 + 3) % 997 = 659		
4	2	6	5 3 5 % 997 = (659*10 + 5) % 997 = 613		

```
// Compute hash for M-digit key
private long hash(String key, int M)
{
    long h = 0;
    for (int j = 0; j < M; j++)
        h = (R * h + key.charAt(j)) % Q;
    return h;
}
```

Efficiently computing the hash function

Challenge. How to efficiently compute x_{i+1} given that we know x_i .

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0$$

$$x_{i+1} = t_{i+1} R^{M-1} + t_{i+2} R^{M-2} + \dots + t_{i+M} R^0$$

Key property. Can update hash function in constant time!

$$x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M}$$

↑ ↑ ↑ ↑
current subtract multiply add new
value leading digit by radix trailing digit (can precompute R^{M-1})

i	...	2	3	4	5	6	7	...
current value		1	4	1	5	9	2	6 5
new value		4	1	5	9	2	6	5

\Rightarrow text

4	1	5	9	2	current value
-	4	0	0	0	
1	5	9	2		subtract leading digit
*	1	0			multiply by radix
1	5	9	2	0	
			+	6	add new trailing digit
1	5	9	2	6	new value

Rabin-Karp substring search example

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3	
0	3	1	%	997	=	3											
1	3	1	%	997	=	(3*10 + 1) % 997	=	31									
2	3	1	4	%	997	=	(31*10 + 4) % 997	=	314								
3	3	1	4	1	%	997	=	(314*10 + 1) % 997	=	150							
4	3	1	4	1	5	%	997	=	(150*10 + 5) % 997	=	508						
5		1	4	1	5	9	%	997	=	((508 + 3*(997 - 30))*10 + 9) % 997	=	201					
6		4	1	5	9	2	%	997	=	((201 + 1*(997 - 30))*10 + 2) % 997	=	715					
7			1	5	9	2	6	%	997	=	((715 + 4*(997 - 30))*10 + 6) % 997	=	971				
8				5	9	2	6	5	%	997	=	((971 + 1*(997 - 30))*10 + 5) % 997	=	442			
9					9	2	6	5	3	%	997	=	((442 + 5*(997 - 30))*10 + 3) % 997	=	929		
10	←	return	i-M+1	=	6		2	6	5	3	5	%	997	=	((929 + 9*(997 - 30))*10 + 5) % 997	=	613

Rabin-Karp: Java implementation

```
public class RabinKarp
{
    private long patHash;          // pattern hash value
    private int M;                 // pattern length
    private long Q;                // modulus
    private int R;                 // radix
    private long RM;               //  $R^{M-1} \bmod Q$ 

    public RabinKarp(String pat) {
        M = pat.length();
        R = 256;
        Q = longRandomPrime();           ← a large prime
                                         (but avoid overflow)

        RM = 1;
        for (int i = 1; i <= M-1; i++)
            RM = (R * RM) % Q;
        patHash = hash(pat, M);
    }

    private long hash(String key, int M)
    { /* as before */ }

    public int search(String txt)
    { /* see next slide */ }
}
```

a large prime
(but avoid overflow)

← precompute $R^{M-1} \bmod Q$

Rabin-Karp: Java implementation (continued)

Monte Carlo version. Return match if hash match.

```
public int search(String txt)
{
    int N = txt.length();
    int txtHash = hash(txt, M);
    if (patHash == txtHash) return 0;
    for (int i = M; i < N; i++)
    {
        txtHash = (txtHash + Q - RM*txt.charAt(i-M) % Q) % Q;
        txtHash = (txtHash*R + txt.charAt(i)) % Q;
        if (patHash == txtHash) return i - M + 1;
    }
    return N;
}
```

check for hash collision
using rolling hash function

Las Vegas version. Check for substring match if hash match;
continue search if false collision.

Rabin-Karp analysis

Theory. If Q is a sufficiently large random prime (about MN^2), then the probability of a false collision is about $1/N$.

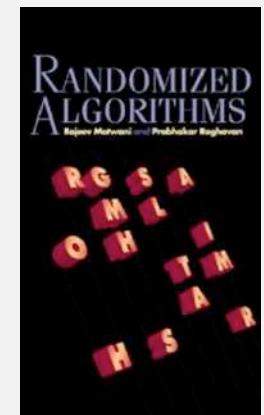
Practice. Choose Q to be a large prime (but not so large to cause overflow). Under reasonable assumptions, probability of a collision is about $1/Q$.

Monte Carlo version.

- Always runs in linear time.
- Extremely likely to return correct answer (but not always!).

Las Vegas version.

- Always returns correct answer.
- Extremely likely to run in linear time (but worst case is MN).



Rabin-Karp fingerprint search

Advantages.

- Extends to 2d patterns.
- Extends to finding multiple patterns.

Disadvantages.

- Arithmetic ops slower than char compares.
- Las Vegas version requires backup.
- Poor worst-case guarantee.

Q. How would you extend Rabin-Karp to efficiently search for any one of P possible patterns in a text of length N ?



Substring search cost summary

Cost of searching for an M -character pattern in an N -character text.

algorithm	version	operation count		backup in input?	correct?	extra space
		guarantee	typical			
brute force	—	MN	$1.1 N$	yes	yes	1
Knuth-Morris-Pratt	<i>full DFA</i> (Algorithm 5.6)	$2N$	$1.1 N$	no	yes	MR
	<i>mismatch transitions only</i>	$3N$	$1.1 N$	no	yes	M
Boyer-Moore	<i>full algorithm</i>	$3N$	N/M	yes	yes	R
	<i>mismatched char heuristic only</i> (Algorithm 5.7)	MN	N/M	yes	yes	R
Rabin-Karp [†]	<i>Monte Carlo</i> (Algorithm 5.8)	$7N$	$7N$	no	yes^{\dagger}	1
	<i>Las Vegas</i>	$7N^{\dagger}$	$7N$	yes	yes	1

[†] probabilistic guarantee, with uniform hash function

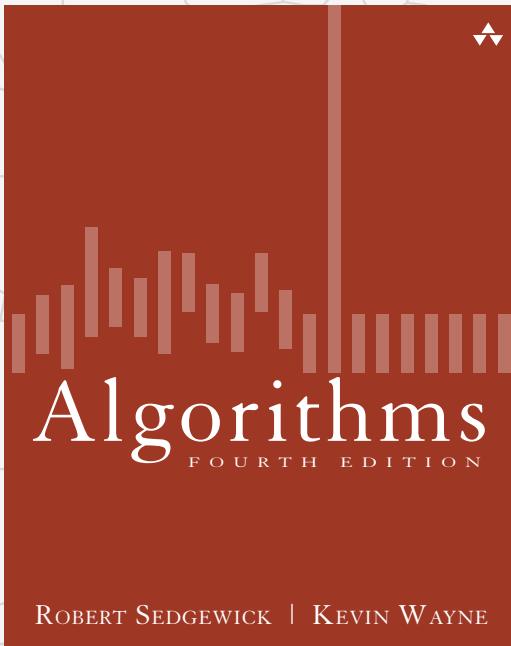
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ ***Rabin-Karp***



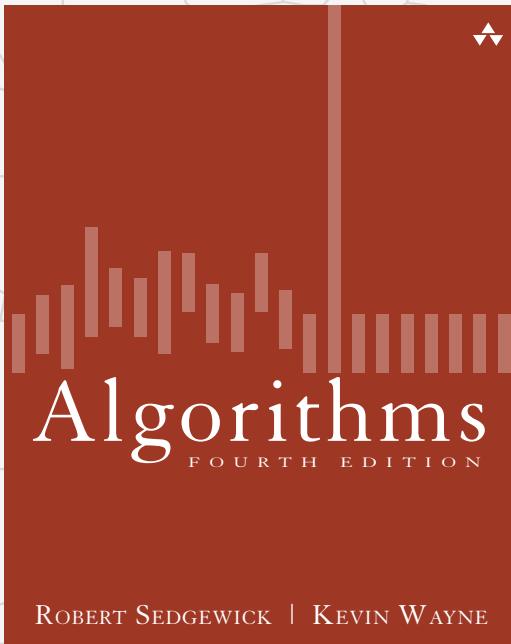
<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Overview: introduction to advanced topics

Main topics. [next 3 lectures]

- Reduction: design algorithms, establish lower bounds, classify problems.
- Linear programming: the ultimate practical problem-solving model.
- Intractability: problems beyond our reach.

Shifting gears.

- From individual problems to problem-solving models.
- From linear/quadratic to polynomial/exponential scale.
- From details of implementation to conceptual framework.

Goals.

- Place algorithms we've studied in a larger context.
- Introduce you to important and essential ideas.
- Inspire you to learn more about algorithms!

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Bird's-eye view

Desiderata. Classify **problems** according to computational requirements.

complexity	order of growth	examples
linear	N	min, max, median, Burrows-Wheeler transform, ...
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, ...
quadratic	N^2	?
:	:	:
exponential	c^N	?

Frustrating news. Huge number of problems have defied classification.

Bird's-eye view

Desiderata. Classify **problems** according to computational requirements.

Desiderata'.

Suppose we could (could not) solve problem X efficiently.

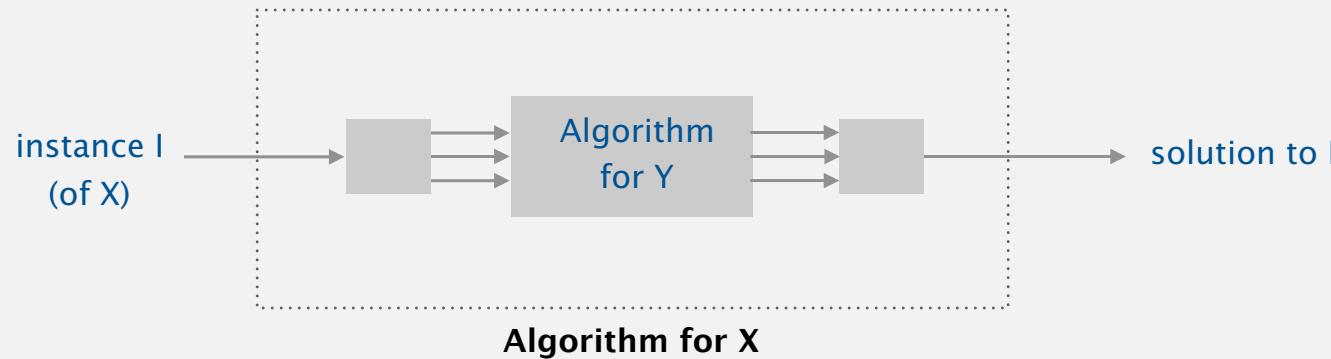
What else could (could not) we solve efficiently?



“Give me a lever long enough and a fulcrum on which to place it, and I shall move the world.” — Archimedes

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X .



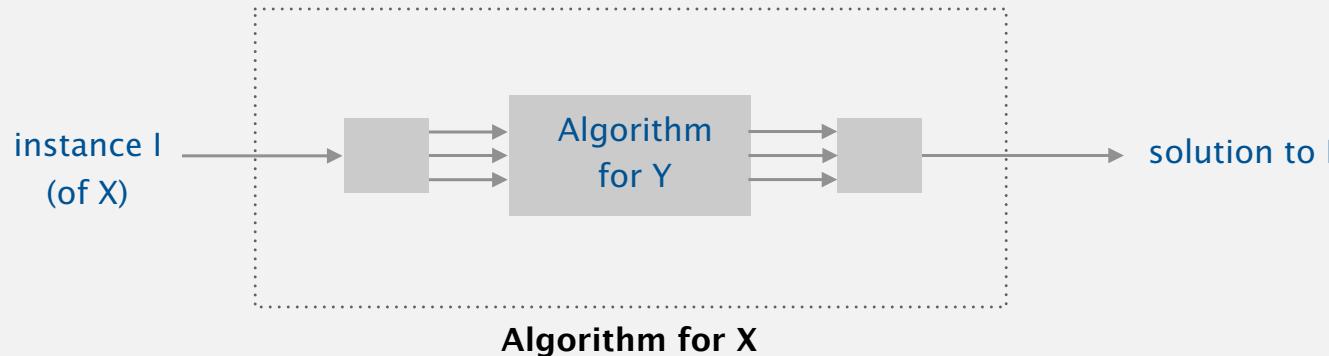
Cost of solving X = total cost of solving Y + cost of reduction.

perhaps many calls to Y
on problems of different sizes

↑
preprocessing and postprocessing

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X .



Ex 1. [finding the median reduces to sorting]

To find the median of N items:

- Sort N items.
 - Return item in the middle.

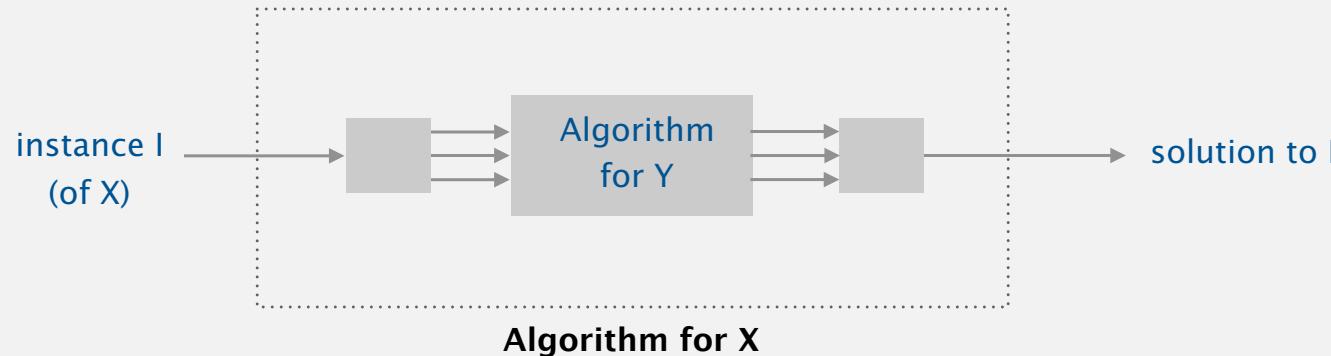
Cost of solving finding the median. $N \log N + 1$.

cost of sorting

cost of reduction

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X .



Ex 2. [element distinctness reduces to sorting]

To solve element distinctness on N items:

- Sort N items.
 - Check adjacent pairs for equality.

Cost of solving element distinctness. $N \log N + N$.

cost of sorting cost of reduction

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Reduction: design algorithms

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X .

Design algorithm. Given algorithm for Y , can also solve X .

Ex.

- 3-collinear reduces to sorting. [assignment]
- Finding the median reduces to sorting.
- Element distinctness reduces to sorting.
- CPM reduces to topological sort. [shortest paths lecture]
- Arbitrage reduces to shortest paths. [shortest paths lecture]
- Burrows-Wheeler transform reduces to suffix sort. [assignment]
- ...

Mentality. Since I know how to solve Y , can I use that algorithm to solve X ?

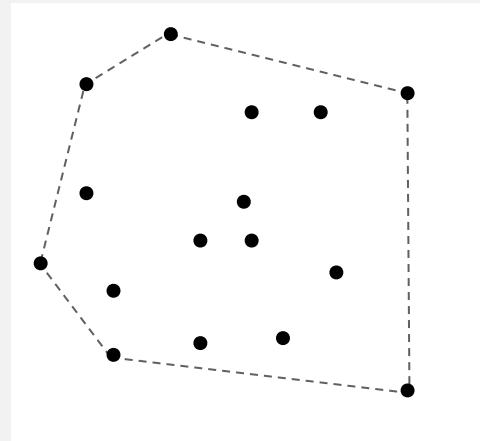


programmer's version: I have code for Y . Can I use it for X ?

Convex hull reduces to sorting

Sorting. Given N distinct integers, rearrange them in ascending order.

Convex hull. Given N points in the plane, identify the extreme points of the convex hull (in counterclockwise order).



convex hull

1251432
2861534
3988818
4190745
8111033
13546464
89885444
43434213
34435312

sorting

Proposition. Convex hull reduces to sorting.

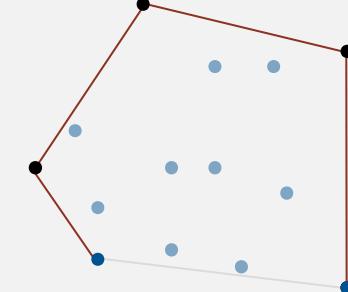
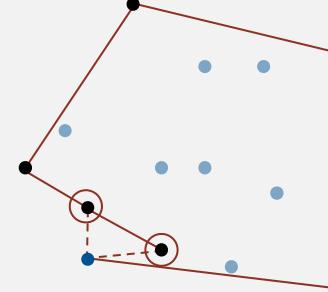
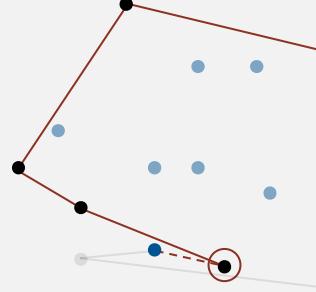
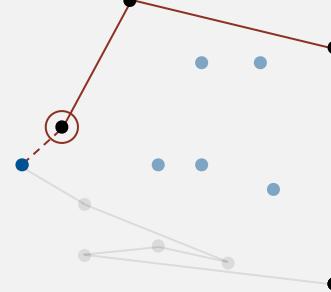
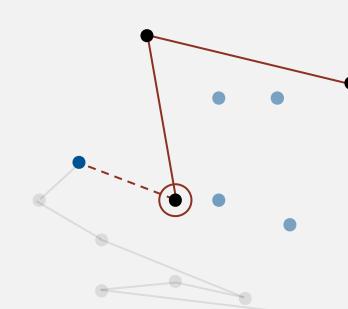
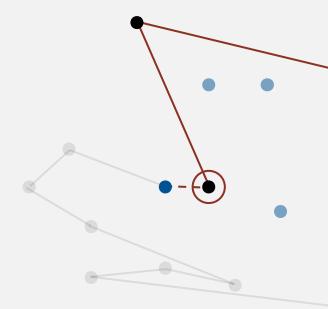
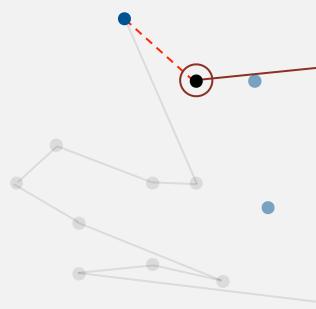
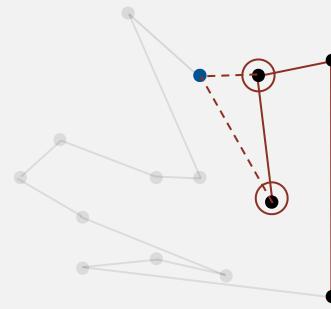
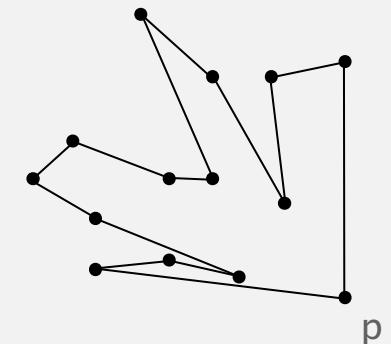
Pf. Graham scan algorithm (see next slide).

cost of sorting cost of reduction
Cost of convex hull. $N \log N + N$.

Graham scan algorithm

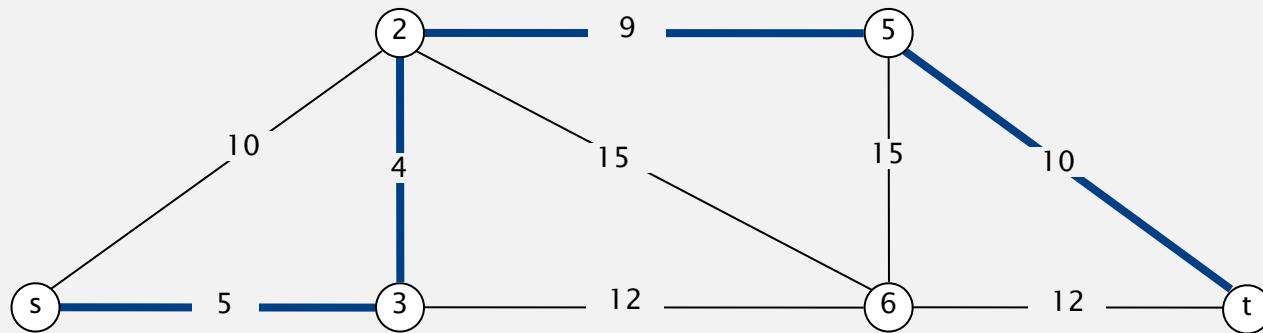
Graham scan.

- Choose point p with smallest (or largest) y-coordinate.
- Sort points by polar angle with p to get simple polygon.
- Consider points in order, and discard those that would create a clockwise turn.

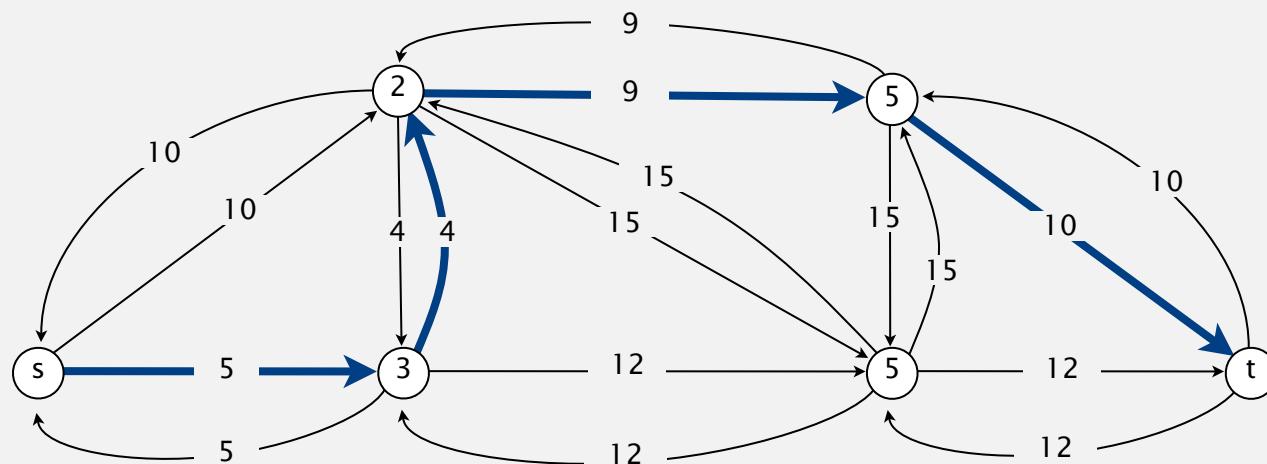


Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

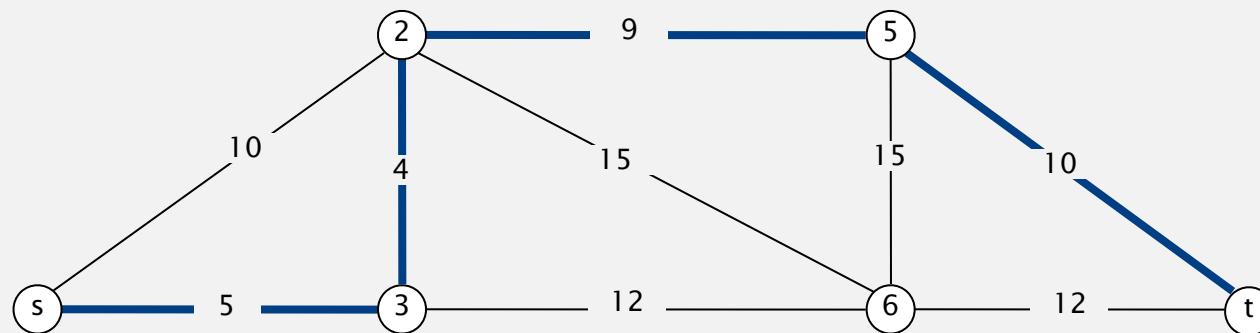


Pf. Replace each undirected edge by two directed edges.



Shortest paths on edge-weighted graphs and digraphs

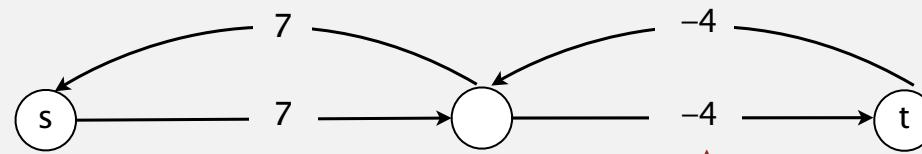
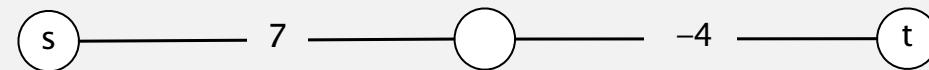
Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.



cost of shortest paths in digraph cost of reduction
↓ ↓
Cost of undirected shortest paths. $E \log V + E$.

Shortest paths with negative weights

Caveat. Reduction is invalid for edge-weighted graphs with negative weights (even if no negative cycles).

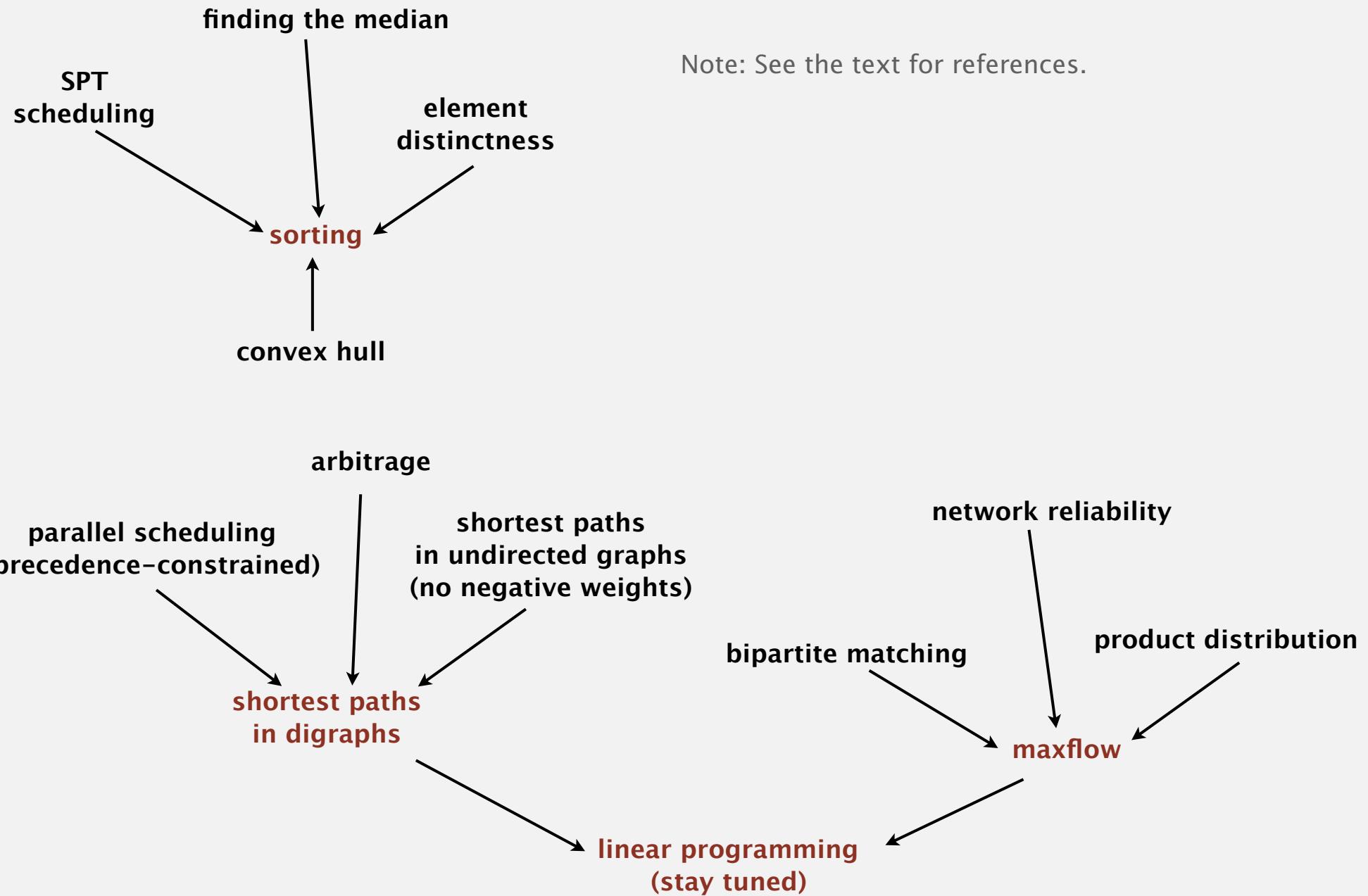


reduction creates
negative cycles

Remark. Can still solve shortest-paths problem in undirected graphs (if no negative cycles), but need more sophisticated techniques.

reduces to weighted
non-bipartite matching (!)

Linear-time reductions involving familiar problems



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

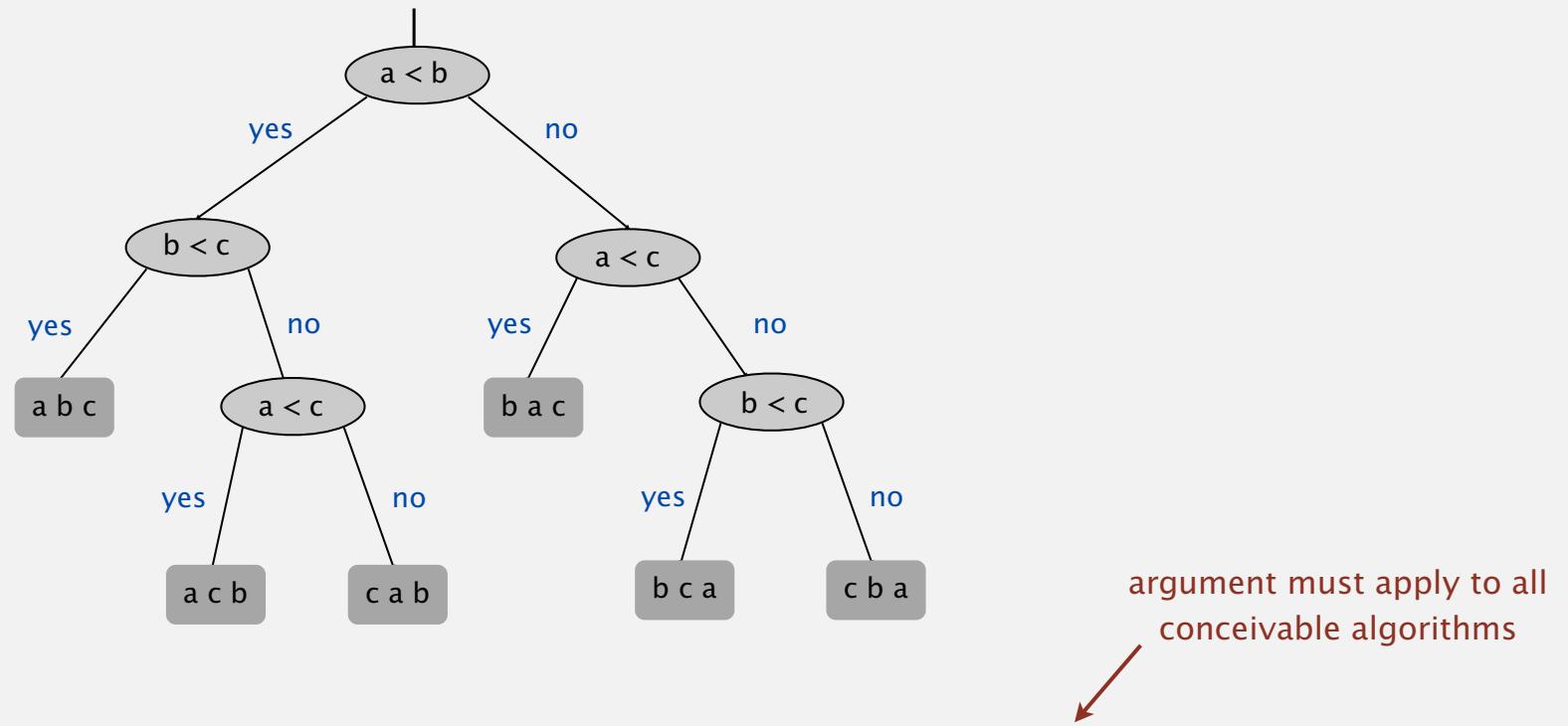
6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Bird's-eye view

Goal. Prove that a problem requires a certain number of steps.

Ex. In decision tree model, any compare-based sorting algorithm requires $\Omega(N \log N)$ compares in the worst case.



Bad news. Very difficult to establish lower bounds from scratch.

Good news. Spread $\Omega(N \log N)$ lower bound to Y by reducing sorting to Y .

assuming cost of reduction is not too high

Linear-time reductions

Def. Problem X **linear-time reduces** to problem Y if X can be solved with:

- Linear number of standard computational steps.
- Constant number of calls to Y .

Ex. Almost all of the reductions we've seen so far. [Which ones weren't?]

Establish lower bound:

- If X takes $\Omega(N \log N)$ steps, then so does Y .
- If X takes $\Omega(N^2)$ steps, then so does Y .

Mentality.

- If I could easily solve Y , then I could easily solve X .
- I can't easily solve X .
- Therefore, I can't easily solve Y .

Lower bound for convex hull

Proposition. In quadratic decision tree model, any algorithm for sorting N integers requires $\Omega(N \log N)$ steps.

allows linear or quadratic tests:

$$\underline{x}_i < \underline{x}_j \text{ or } (x_j - x_i)(x_k - x_i) - (x_j)(\underline{x}_j - x_i) < 0$$

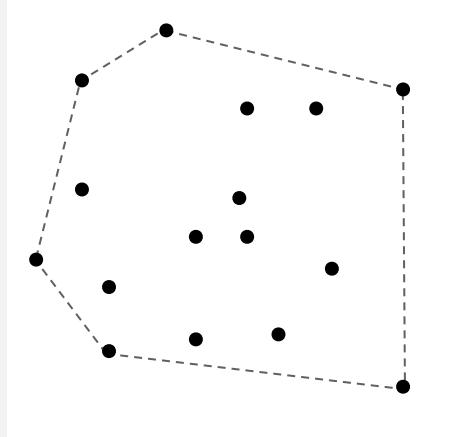
Proposition. Sorting linear-time reduces to convex hull.

Pf. [see next slide]

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently

1251432
2861534
3988818
4190745
8111033
13546464
89885444
43434213
34435312

sorting



convex hull

linear or
quadratic tests

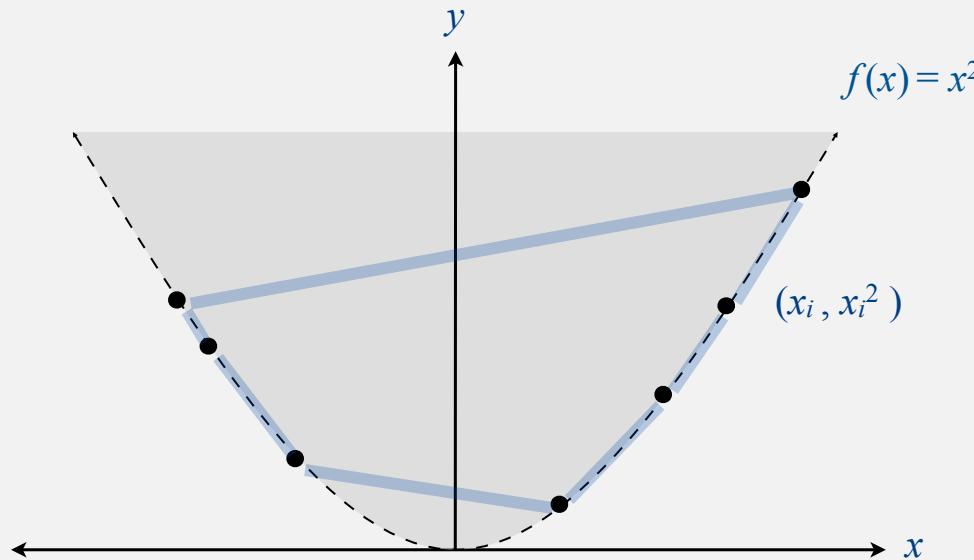
Implication. Any ccw-based convex hull algorithm requires $\Omega(N \log N)$ ops.

Sorting linear-time reduces to convex hull

Proposition. Sorting linear-time reduces to convex hull.

- Sorting instance: x_1, x_2, \dots, x_N .
- Convex hull instance: $(x_1, x_1^2), (x_2, x_2^2), \dots, (x_N, x_N^2)$.

lower-bound mentality:
if I can solve convex hull
efficiently, I can sort efficiently



Pf.

- Region $\{x : x^2 \geq x\}$ is convex \Rightarrow all points are on hull.
- Starting at point with most negative x , counterclockwise order of hull points yields integers in ascending order.

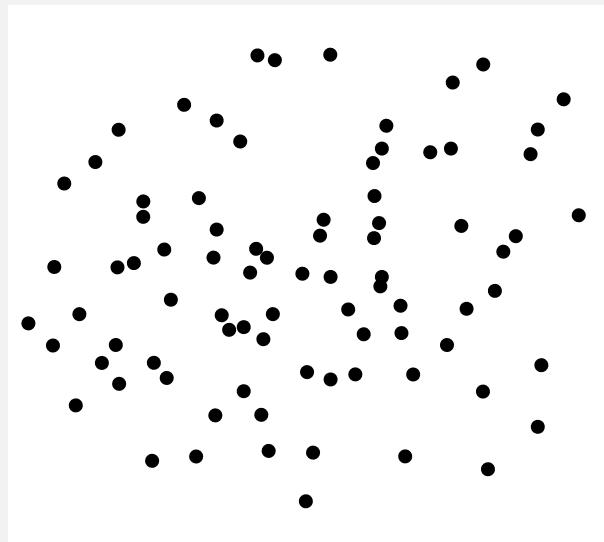
Establishing lower bounds: summary

Establishing lower bounds through reduction is an important tool in guiding algorithm design efforts.

Q. How to convince yourself no linear-time convex hull algorithm exists?

A1. [hard way] Long futile search for a linear-time algorithm.

A2. [easy way] Linear-time reduction from sorting.



convex hull



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Classifying problems: summary

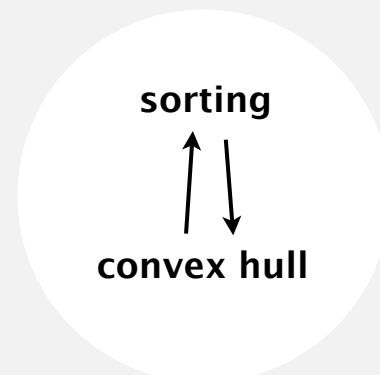
Desiderata. Problem with algorithm that matches lower bound.

Ex. Sorting and convex hull have complexity $N \log N$.

Desiderata'. Prove that two problems X and Y have the same complexity.

- First, show that problem X linear-time reduces to Y .
- Second, show that Y linear-time reduces to X .
- Conclude that X and Y have the same complexity.

even if we don't know what it is!



Caveat

SORT. Given N distinct integers, rearrange them in ascending order.

CONVEX HULL. Given N points in the plane, identify the extreme points of the convex hull (in counterclockwise order).

Proposition. SORT linear-time reduces to CONVEX HULL.

Proposition. CONVEX HULL linear-time reduces to SORT.

Conclusion. SORT and CONVEX HULL have the same complexity.

A possible real-world scenario.

- System designer specs the APIs for project.
- Alice implements `sort()` using `convexHull()`.
- Bob implements `convexHull()` using `sort()`.
- Infinite reduction loop!
- Who's fault?

well, maybe not so realistic

Integer arithmetic reductions

Integer multiplication. Given two N -bit integers, compute their product.

Brute force. N^2 bit operations.

Integer arithmetic reductions

Integer multiplication. Given two N -bit integers, compute their product.

Brute force. N^2 bit operations.

problem	arithmetic	order of growth
integer multiplication	$a \times b$	$M(N)$
integer division	$a / b, a \bmod b$	$M(N)$
integer square	a^2	$M(N)$
integer square root	$\lfloor \sqrt{a} \rfloor$	$M(N)$

integer arithmetic problems with the same complexity as integer multiplication

Q. Is brute-force algorithm optimal?

History of complexity of integer multiplication

year	algorithm	order of growth
?	brute force	N^2
1962	Karatsuba-Ofman	$N^{1.585}$
1963	Toom-3, Toom-4	$N^{1.465}, N^{1.404}$
1966	Toom-Cook	$N^{1+\varepsilon}$
1971	Schönhage–Strassen	$N \log N \log \log N$
2007	Fürer	$N \log N 2^{\log^* N}$
?	?	N

number of bit operations to multiply two N -bit integers

Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

used in Maple, Mathematica, gcc, cryptography, ...



Linear algebra reductions

Matrix multiplication. Given two N -by- N matrices, compute their product.

Brute force. N^3 flops.

$$\begin{array}{c|cccc} & & \text{column j} & & \\ & & 0.4 & 0.3 & 0.1 & 0.1 \\ \text{row i} & 0.5 & 0.3 & 0.9 & 0.6 & \\ \hline & 0.1 & 0.0 & 0.7 & 0.4 & \\ & 0.0 & 0.3 & 0.3 & 0.1 & \end{array} \times \begin{array}{c|cccc} & & \text{column j} & & \\ & & 0.4 & 0.3 & 0.1 & 0.1 \\ & & 0.2 & 0.2 & 0.0 & 0.6 \\ & & 0.0 & 0.0 & 0.4 & 0.5 \\ & & 0.8 & 0.4 & 0.1 & 0.9 \\ \hline & & \text{j} & & \\ & & 0.16 & 0.11 & 0.34 & 0.62 \\ & & 0.74 & 0.45 & 0.47 & 1.22 \\ & & 0.36 & 0.19 & 0.33 & 0.72 \\ & & 0.14 & 0.10 & 0.13 & 0.42 \\ \hline & & = & & \\ & & i & & \end{array} = \begin{array}{c} \\ \\ \\ \\ \end{array}$$

$0.5 \cdot 0.1 + 0.3 \cdot 0.0 + 0.9 \cdot 0.4 + 0.6 \cdot 0.1 = 0.47$

Linear algebra reductions

Matrix multiplication. Given two N -by- N matrices, compute their product.

Brute force. N^3 flops.

problem	linear algebra	order of growth
matrix multiplication	$A \times B$	$MM(N)$
matrix inversion	A^{-1}	$MM(N)$
determinant	$ A $	$MM(N)$
system of linear equations	$Ax = b$	$MM(N)$
LU decomposition	$A = L U$	$MM(N)$
least squares	$\min Ax - b _2$	$MM(N)$

numerical linear algebra problems with the same complexity as matrix multiplication

Q. Is brute-force algorithm optimal?

History of complexity of matrix multiplication

year	algorithm	order of growth
?	brute force	N^3
1969	Strassen	$N^{2.808}$
1978	Pan	$N^{2.796}$
1979	Bini	$N^{2.780}$
1981	Schönhage	$N^{2.522}$
1982	Romani	$N^{2.517}$
1982	Coppersmith-Winograd	$N^{2.496}$
1986	Strassen	$N^{2.479}$
1989	Coppersmith-Winograd	$N^{2.376}$
2010	Strother	$N^{2.3737}$
2011	Williams	$N^{2.3727}$
?	?	$N^{2+\varepsilon}$

number of floating-point operations to multiply two N-by-N matrices

Birds-eye view: review

Desiderata. Classify **problems** according to computational requirements.

complexity	order of growth	examples
linear	N	min, max, median, Burrows-Wheeler transform, ...
linearithmic	$N \log N$	sorting, convex hull, closest pair, farthest pair, ...
quadratic	N^2	?
:	:	:
exponential	c^N	?

Frustrating news. Huge number of problems have defied classification.

Birds-eye view: revised

Desiderata. Classify **problems** according to computational requirements.

complexity	order of growth	examples
linear	N	min, max, median,
linearithmic	$N \log N$	sorting, convex hull,
$M(N)$?	integer multiplication, division, square root, ...
$MM(N)$?	matrix multiplication, $Ax = b$, least square, determinant, ...
:	:	:
NP-complete	probably not N^b	SAT, IND-SET, ILP, ...

↑
STAY TUNED!

Good news. Can put many problems into equivalence classes.

Complexity zoo

Complexity class. Set of problems sharing some computational property.



http://qwiki.stanford.edu/index.php/Complexity_Zoo

Bad news. Lots of complexity classes.

Summary

Reductions are important in theory to:

- Design algorithms.
- Establish lower bounds.
- Classify problems according to their computational requirements.

Reductions are important in practice to:

- Design algorithms.
- Design reusable software modules.
 - stacks, queues, priority queues, symbol tables, sets, graphs
 - sorting, regular expressions, Delaunay triangulation
 - MST, shortest path, maxflow, linear programming
- Determine difficulty of your problem and choose the right tool.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

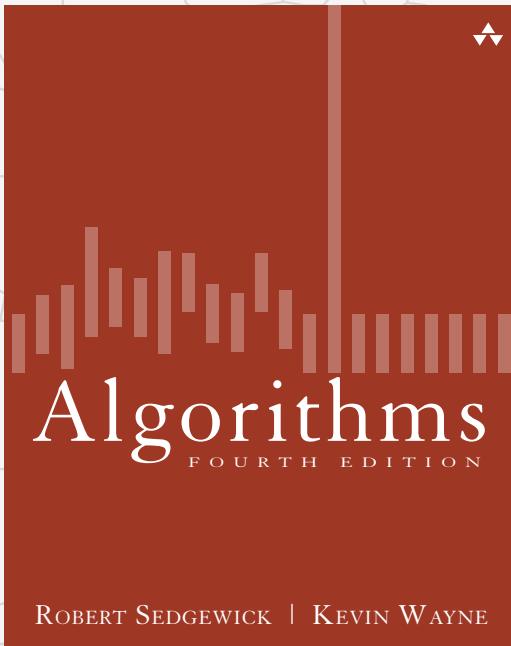
<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*

Algorithms

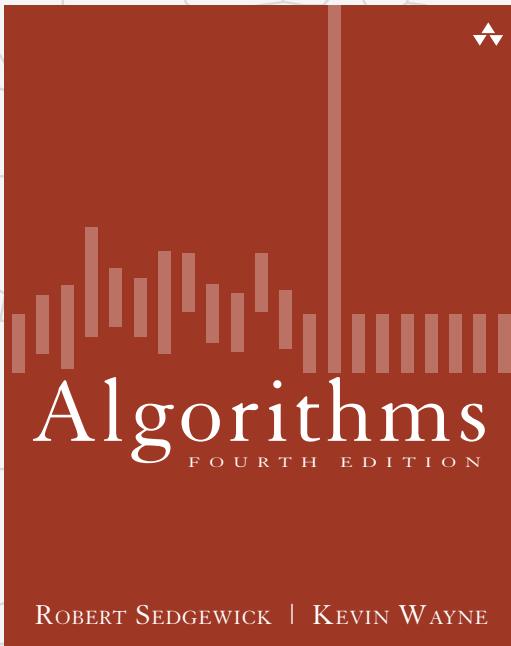
ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

6.5 REDUCTIONS

- ▶ *introduction*
- ▶ *designing algorithms*
- ▶ *establishing lower bounds*
- ▶ *classifying problems*



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

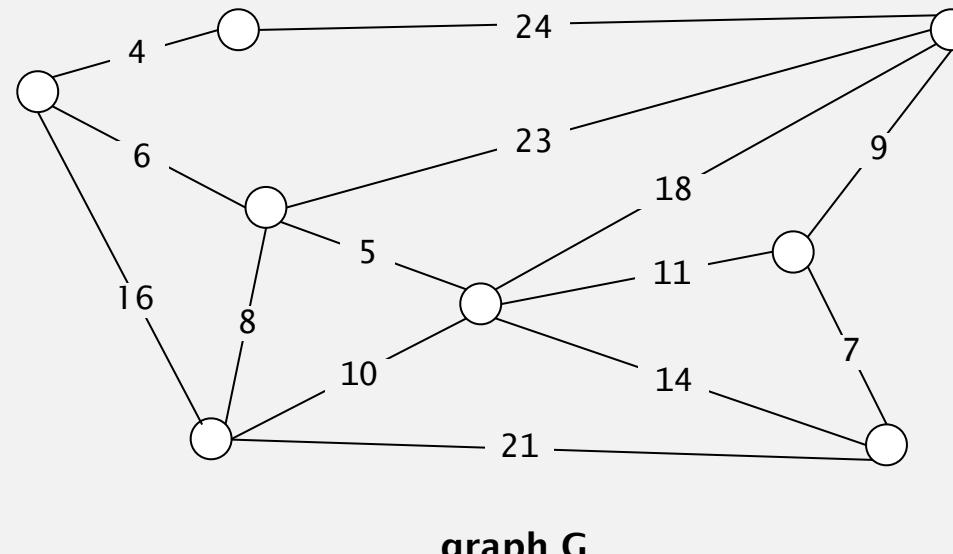
- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).

Def. A **spanning tree** of G is a subgraph T that is connected and acyclic.

Goal. Find a min weight spanning tree.

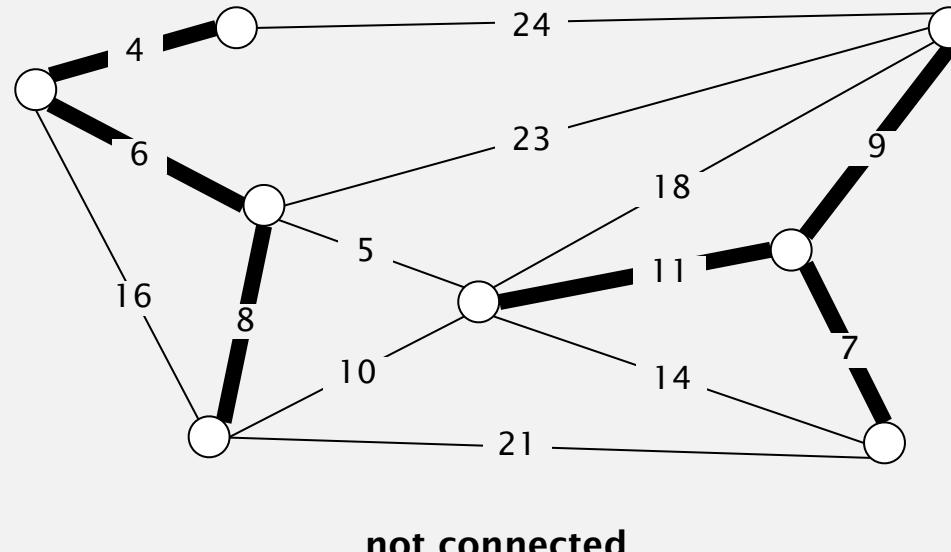


Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).

Def. A **spanning tree** of G is a subgraph T that is connected and acyclic.

Goal. Find a min weight spanning tree.

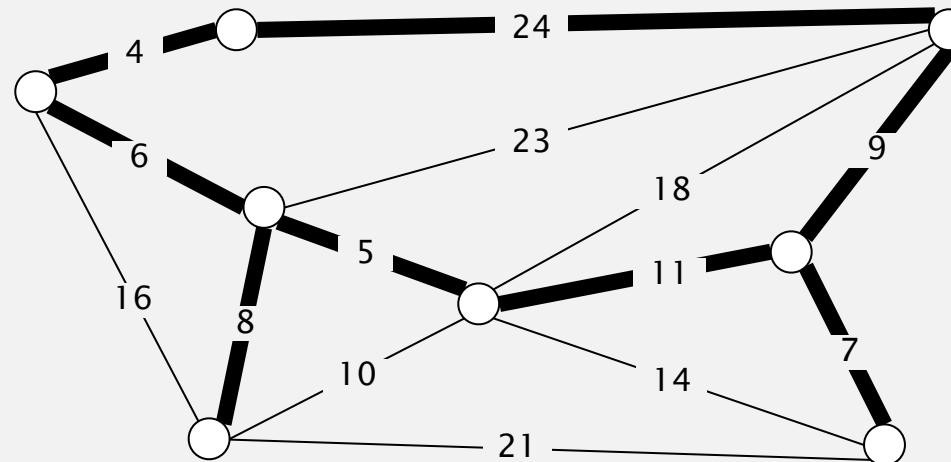


Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).

Def. A **spanning tree** of G is a subgraph T that is connected and acyclic.

Goal. Find a min weight spanning tree.



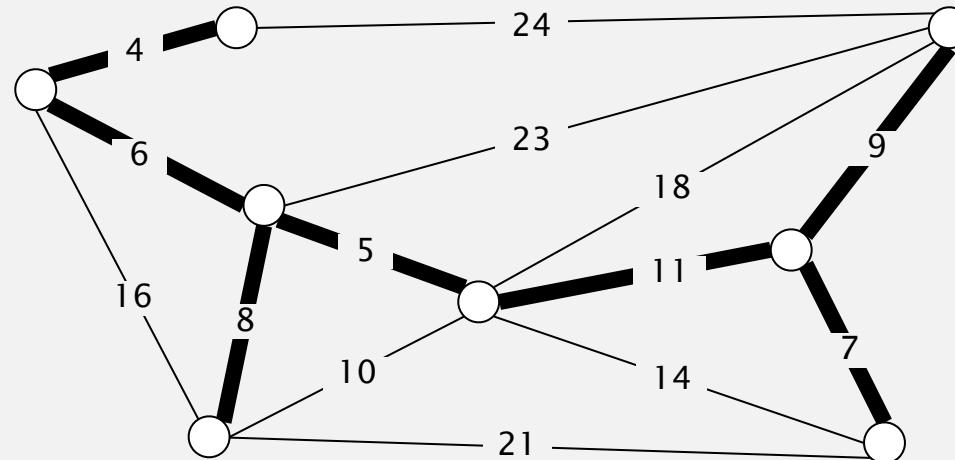
not acyclic

Minimum spanning tree

Given. Undirected graph G with positive edge weights (connected).

Def. A **spanning tree** of G is a subgraph T that is connected and acyclic.

Goal. Find a min weight spanning tree.

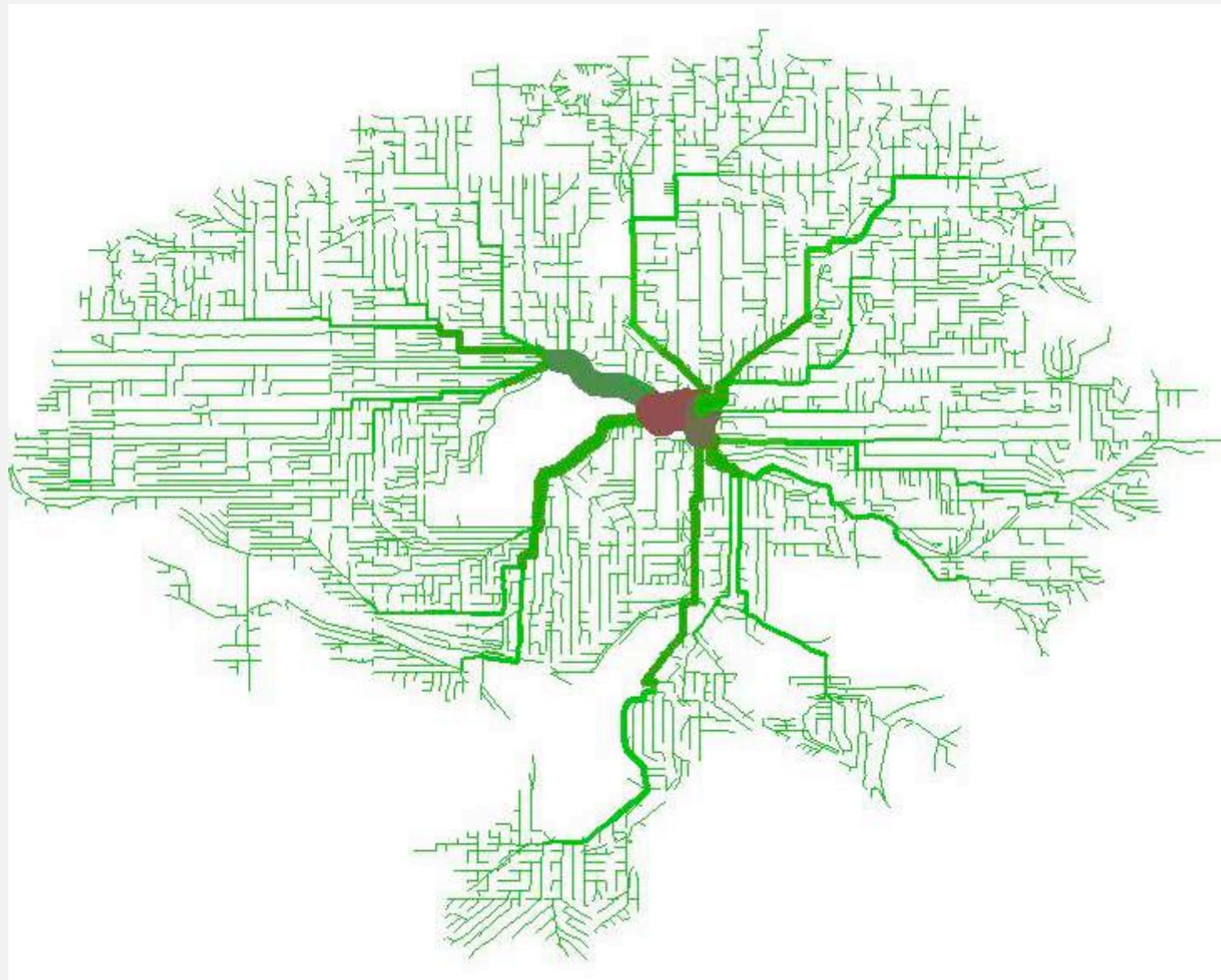


spanning tree T: cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Brute force. Try all spanning trees?

Network design

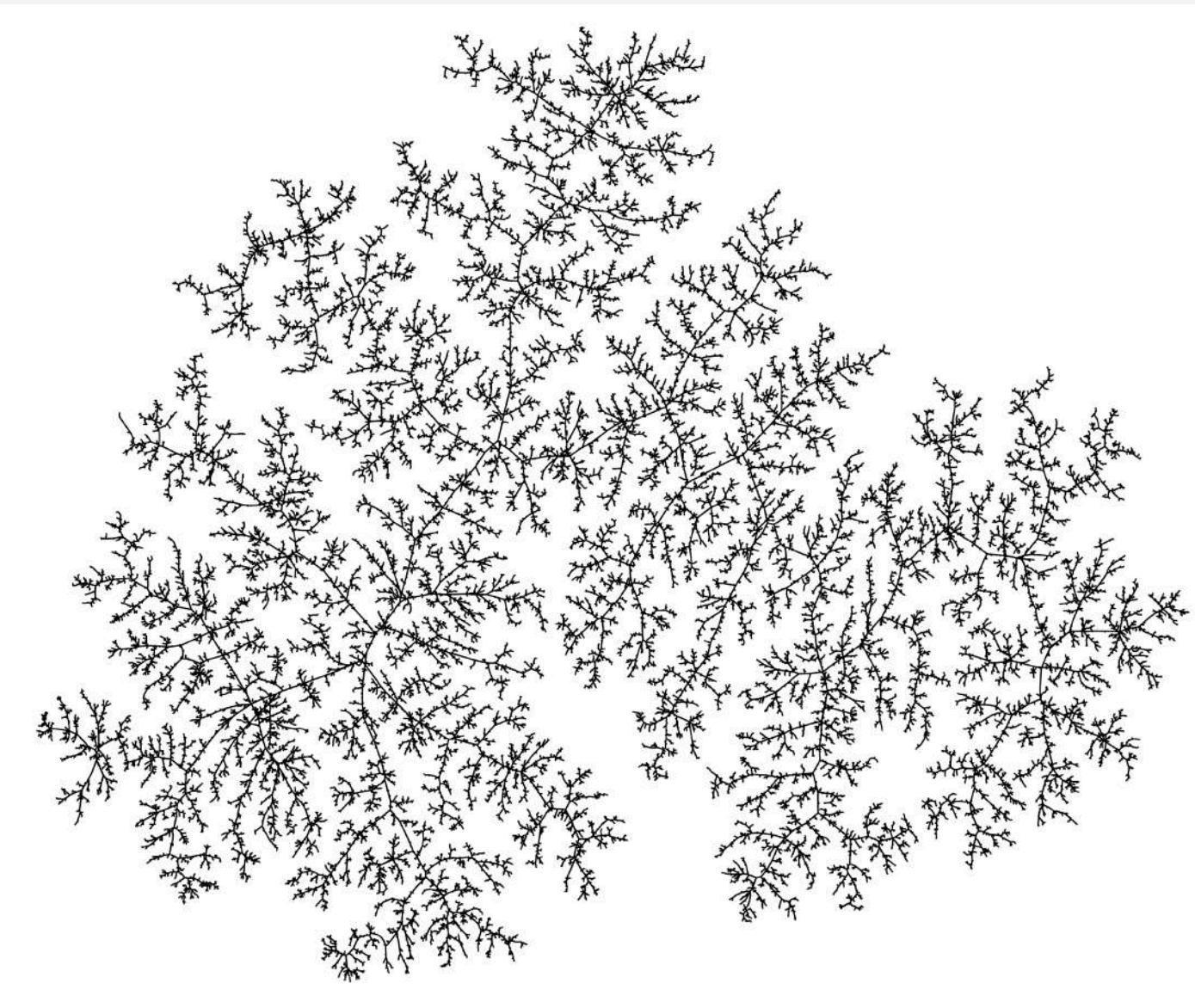
MST of bicycle routes in North Seattle



<http://www.flickr.com/photos/ewedistrict/21980840>

Models of nature

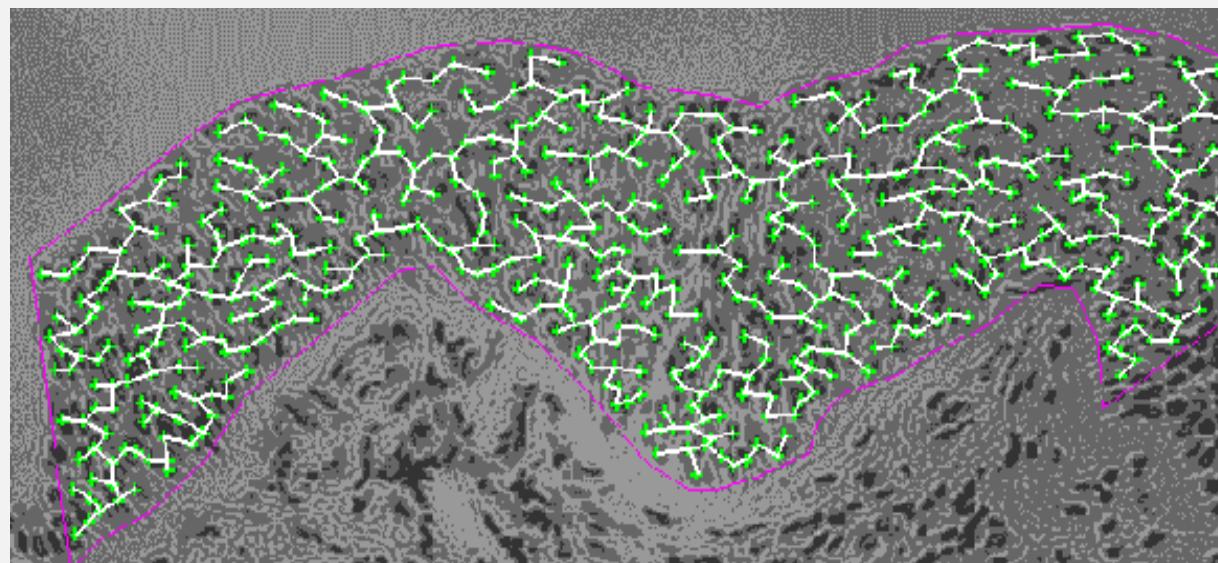
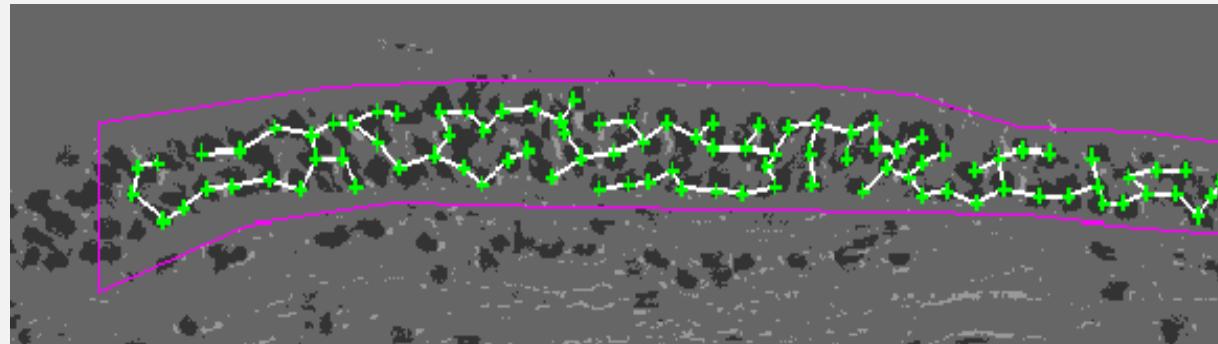
MST of random graph



<http://algo.inria.fr;broutin/gallery.html>

Medical image processing

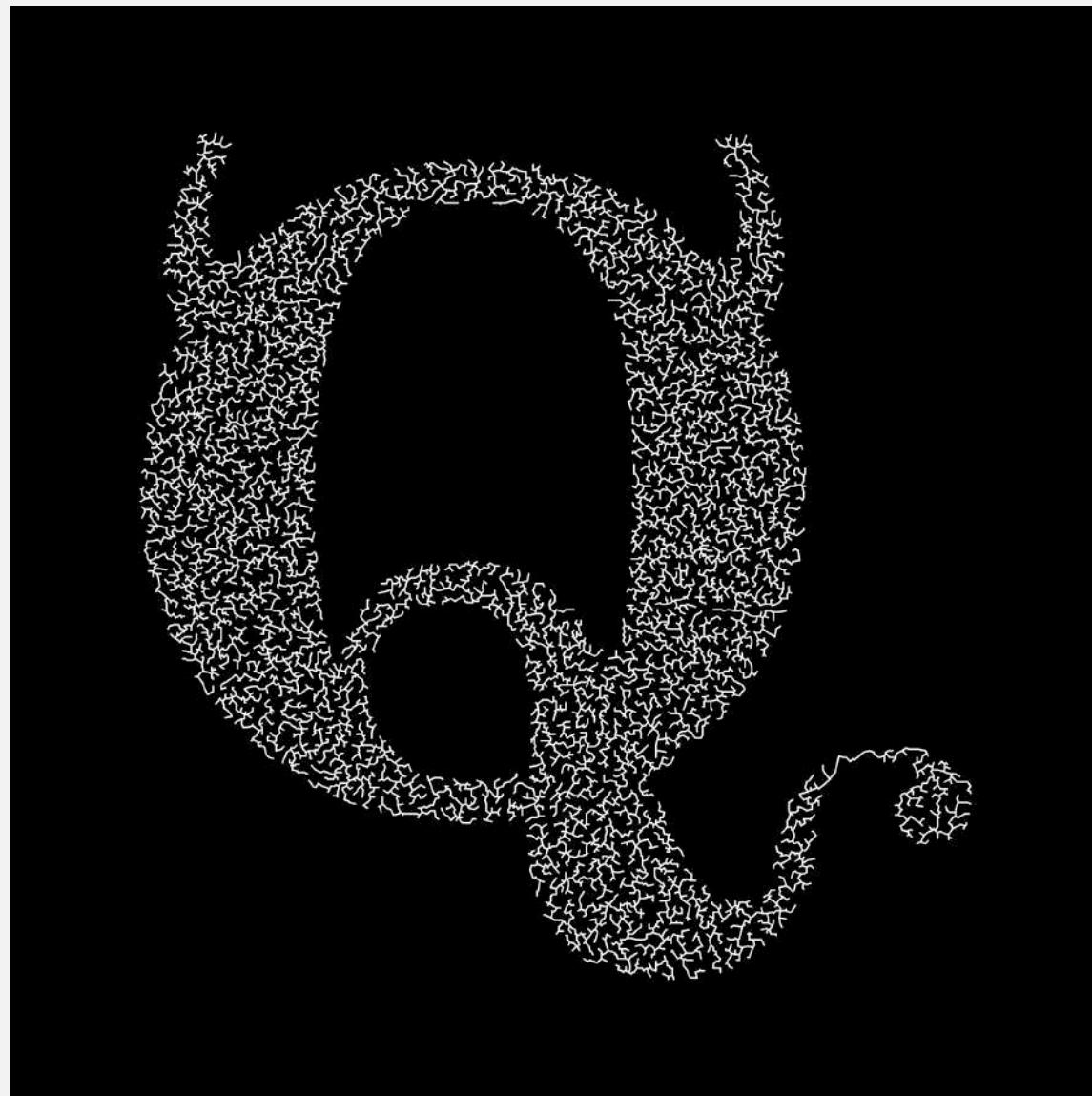
MST describes arrangement of nuclei in the epithelium for cancer research



http://www.bccrc.ca/ci/ta01_archlevel.html

Medical image processing

MST dithering



<http://www.flickr.com/photos/quasimondo/2695389651>

Applications

MST is fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Reducing data storage in sequencing amino acids in a protein.
- Model locality of particle interactions in turbulent fluid flows.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).

<http://www.ics.uci.edu/~eppstein/gina/mst.html>

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

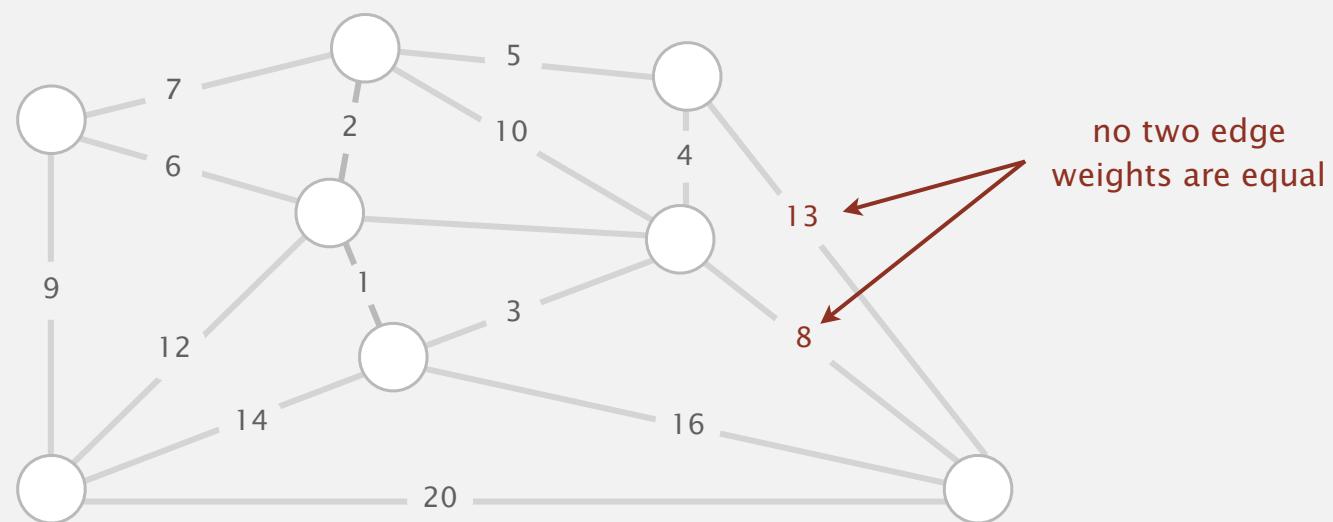
- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Simplifying assumptions

Simplifying assumptions.

- Edge weights are distinct.
- Graph is connected.

Consequence. MST exists and is unique.

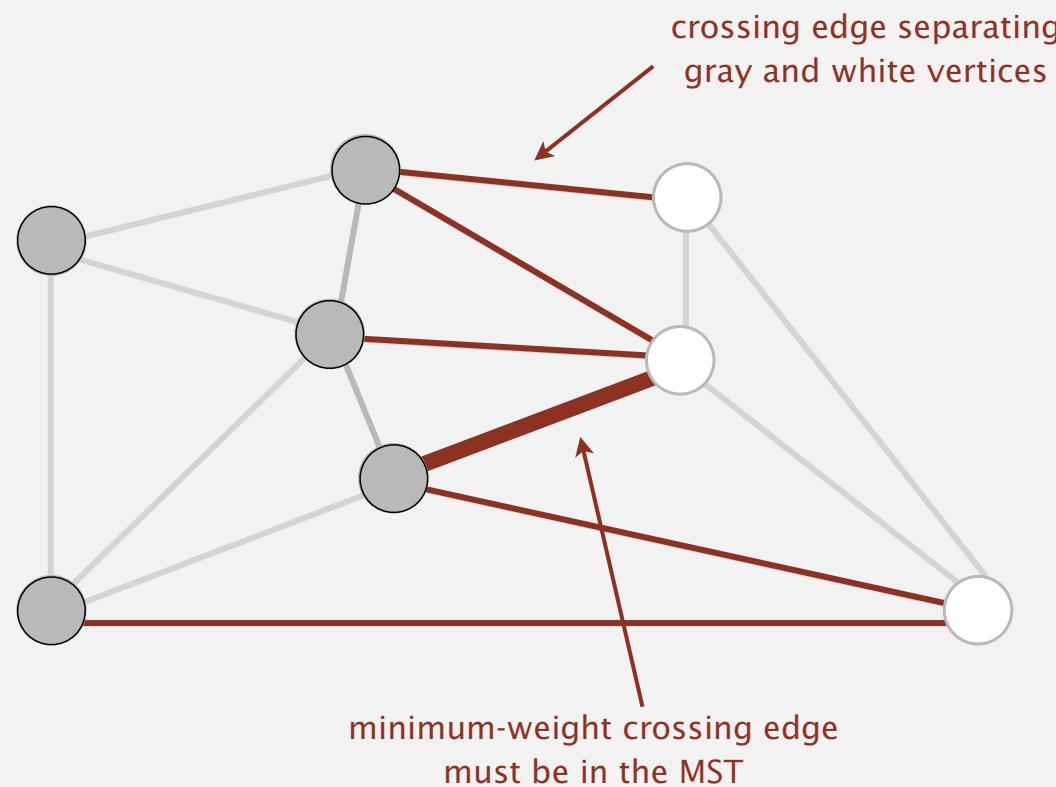


Cut property

Def. A **cut** in a graph is a partition of its vertices into two (nonempty) sets.

Def. A **crossing edge** connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.



Cut property: correctness proof

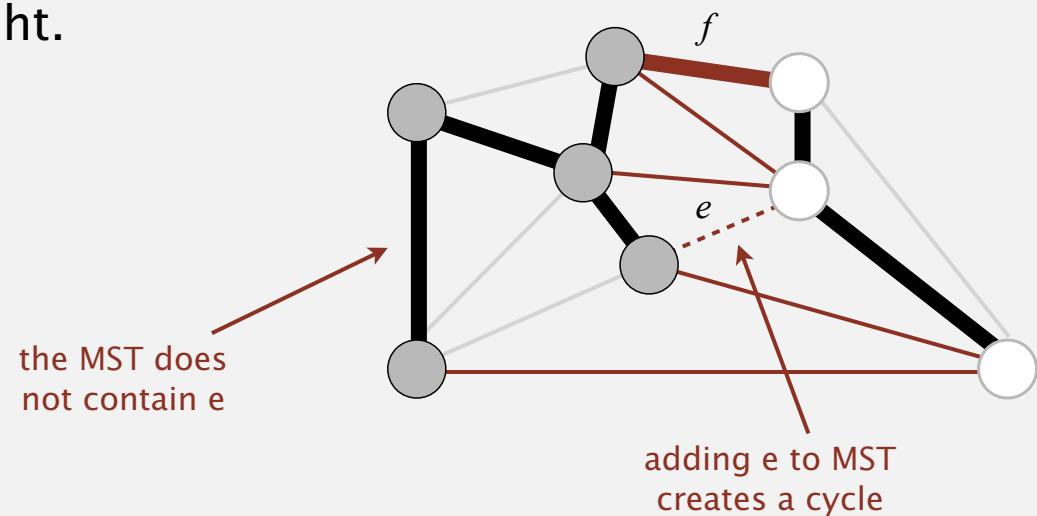
Def. A **cut** in a graph is a partition of its vertices into two (nonempty) sets.

Def. A **crossing edge** connects a vertex in one set with a vertex in the other.

Cut property. Given any cut, the crossing edge of min weight is in the MST.

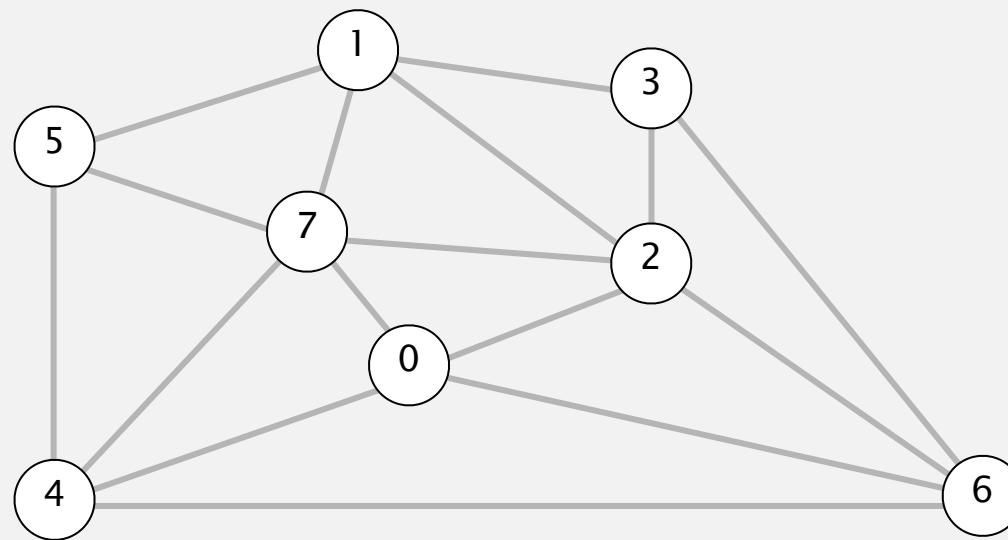
Pf. Suppose min-weight crossing edge e is not in the MST.

- Adding e to the MST creates a cycle.
- Some other edge f in cycle must be a crossing edge.
- Removing f and adding e is also a spanning tree.
- Since weight of e is less than the weight of f ,
that spanning tree is lower weight.
- Contradiction. ▀



Greedy MST algorithm demo

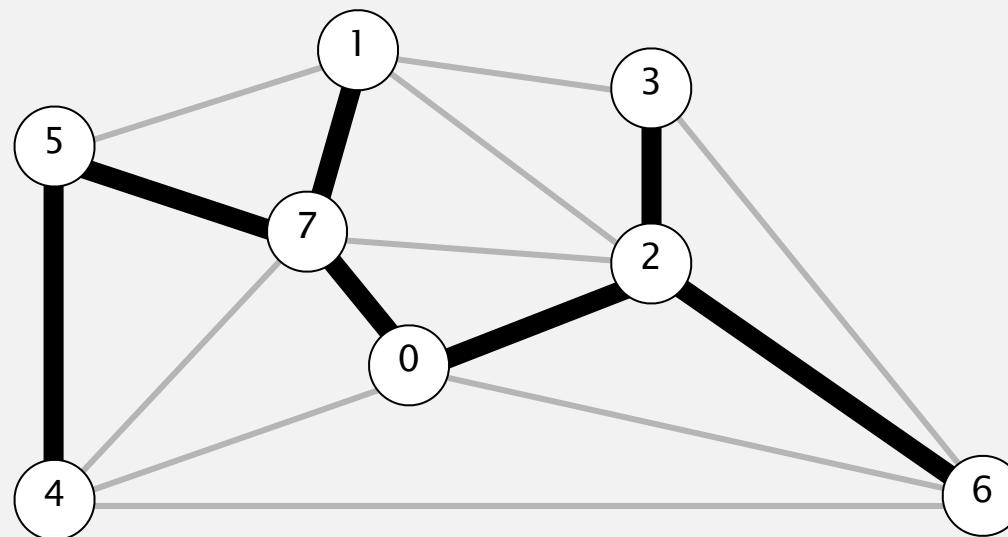
- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Greedy MST algorithm demo

- Start with all edges colored gray.
- Find cut with no black crossing edges; color its min-weight edge black.
- Repeat until $V - 1$ edges are colored black.



MST edges

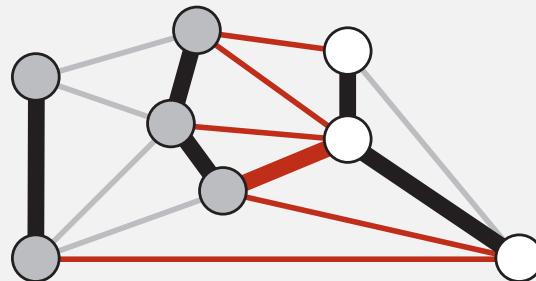
0-2 5-7 6-2 0-7 2-3 1-7 4-5

Greedy MST algorithm: correctness proof

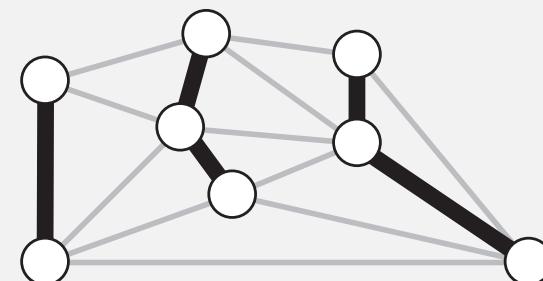
Proposition. The greedy algorithm computes the MST.

Pf.

- Any edge colored black is in the MST (via cut property).
- Fewer than $V - 1$ black edges \Rightarrow cut with no black crossing edges.
(consider cut whose vertices are one connected component)



a cut with no black crossing edges



fewer than $V-1$ edges colored black

Greedy MST algorithm: efficient implementations

Proposition. The greedy algorithm computes the MST.

Efficient implementations. Choose cut? Find min-weight edge?

Ex 1. Kruskal's algorithm. [stay tuned]

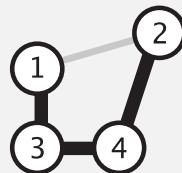
Ex 2. Prim's algorithm. [stay tuned]

Ex 3. Borüvka's algorithm.

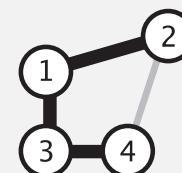
Removing two simplifying assumptions

Q. What if edge weights are not all distinct?

A. Greedy MST algorithm still correct if equal weights are present!
(our correctness proof fails, but that can be fixed)



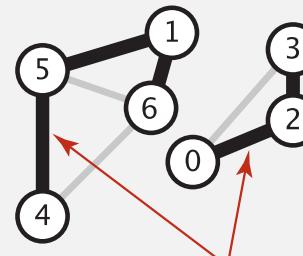
1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50



1	2	1.00
1	3	0.50
2	4	1.00
3	4	0.50

Q. What if graph is not connected?

A. Compute minimum spanning forest = MST of each component.



*can independently compute
MSTs of components*

4	5	0.61
4	6	0.62
5	6	0.88
1	5	0.11
2	3	0.35
0	3	0.6
1	6	0.10
0	2	0.22

Greed is good



Gordon Gecko (Michael Douglas) address to Teldar Paper Stockholders in Wall Street (1986)

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

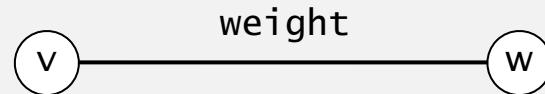
4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Weighted edge API

Edge abstraction needed for weighted edges.

public class Edge implements Comparable<Edge>	
Edge(int v, int w, double weight)	<i>create a weighted edge v-w</i>
int either()	<i>either endpoint</i>
int other(int v)	<i>the endpoint that's not v</i>
int compareTo(Edge that)	<i>compare this edge to that edge</i>
double weight()	<i>the weight</i>
String toString()	<i>string representation</i>



Idiom for processing an edge e: `int v = e.either(), w = e.other(v);`

Weighted edge: Java implementation

```
public class Edge implements Comparable<Edge>
{
    private final int v, w;
    private final double weight;

    public Edge(int v, int w, double weight)
    {
        this.v = v;
        this.w = w;
        this.weight = weight;
    }

    public int either()
    {   return v;   }

    public int other(int vertex)
    {
        if (vertex == v) return w;
        else return v;
    }

    public int compareTo(Edge that)
    {
        if      (this.weight < that.weight) return -1;
        else if (this.weight > that.weight) return +1;
        else                                return 0;
    }
}
```

constructor

either endpoint

other endpoint

compare edges by weight

Edge-weighted graph API

```
public class EdgeWeightedGraph
```

```
    EdgeWeightedGraph(int V)
```

create an empty graph with V vertices

```
    EdgeWeightedGraph(In in)
```

create a graph from input stream

```
    void addEdge(Edge e)
```

add weighted edge e to this graph

```
    Iterable<Edge> adj(int v)
```

edges incident to v

```
    Iterable<Edge> edges()
```

all edges in this graph

```
    int V()
```

number of vertices

```
    int E()
```

number of edges

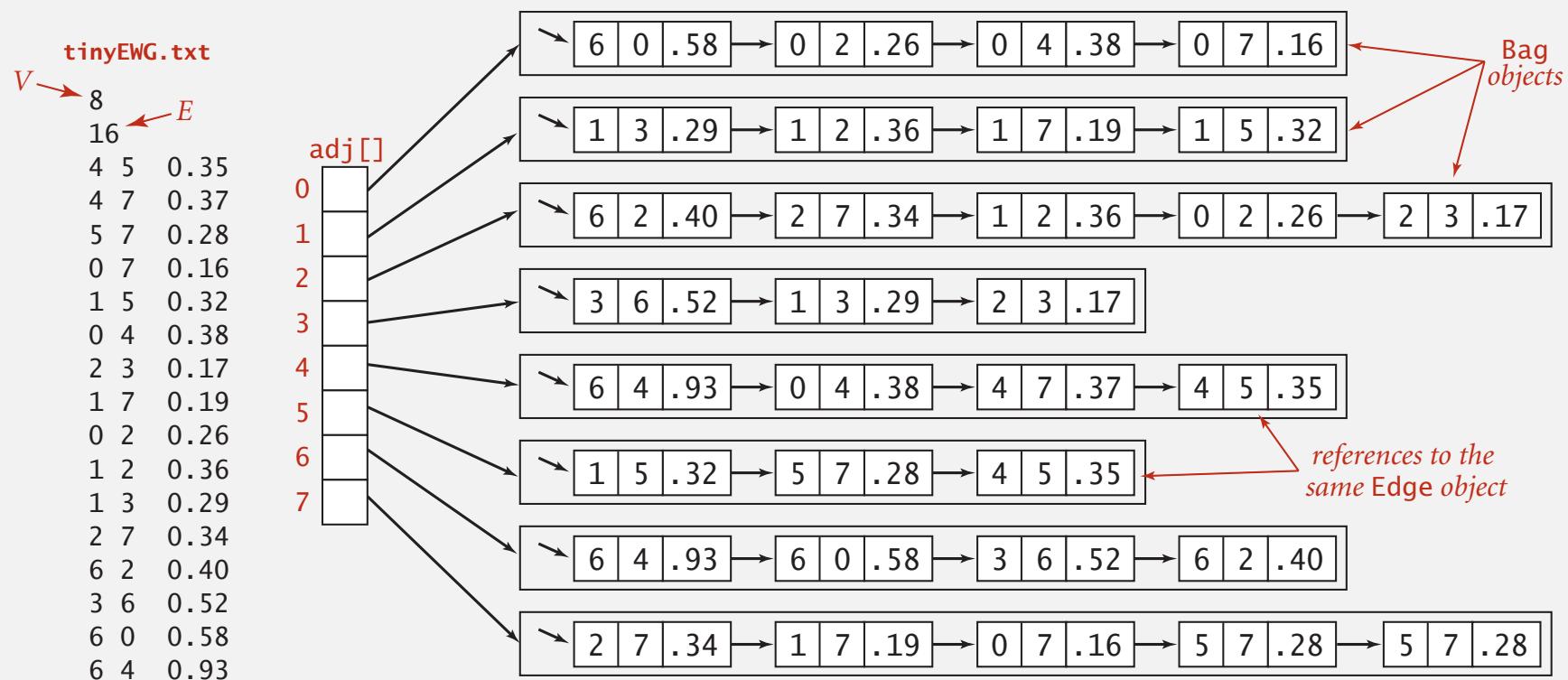
```
    String toString()
```

string representation

Conventions. Allow self-loops and parallel edges.

Edge-weighted graph: adjacency-lists representation

Maintain vertex-indexed array of Edge lists.



Edge-weighted graph: adjacency-lists implementation

```
public class EdgeWeightedGraph
{
```

```
    private final int V;
```

```
    private final Bag<Edge>[] adj;
```



same as Graph, but adjacency lists of Edges instead of integers

```
    public EdgeWeightedGraph(int V)
```

```
    {
```

```
        this.V = V;
```

```
        adj = (Bag<Edge>[]) new Bag[V];
```

```
        for (int v = 0; v < V; v++)
```

```
            adj[v] = new Bag<Edge>();
```

```
}
```



constructor

```
    public void addEdge(Edge e)
```

```
    {
```

```
        int v = e.either(), w = e.other(v);
```

```
        adj[v].add(e);
```

```
        adj[w].add(e);
```

```
}
```



add edge to both adjacency lists

```
    public Iterable<Edge> adj(int v)
```

```
    { return adj[v]; }
```

```
}
```

Minimum spanning tree API

Q. How to represent the MST?

```
public class MST
```

```
MST(EdgeWeightedGraph G)
```

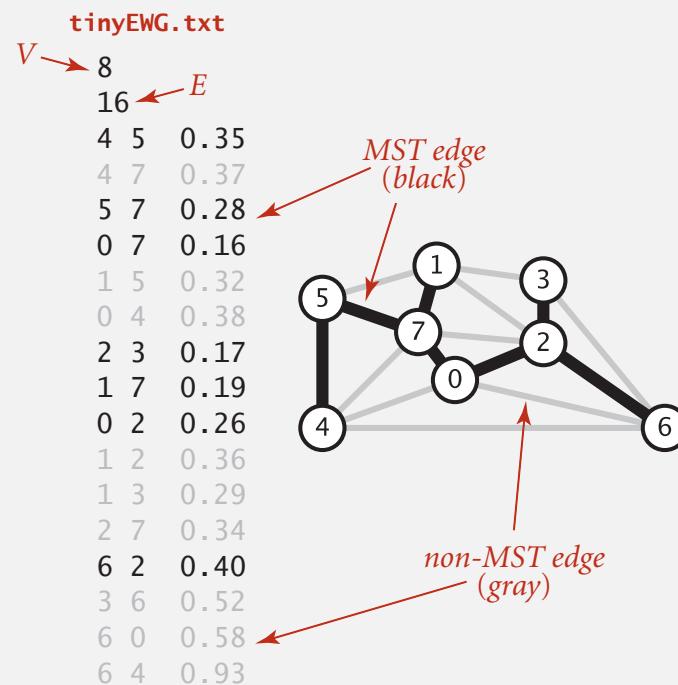
constructor

```
Iterable<Edge> edges()
```

edges in MST

```
double weight()
```

weight of MST



```
% java MST tinyEWG.txt  
0-7 0.16  
1-7 0.19  
0-2 0.26  
2-3 0.17  
5-7 0.28  
4-5 0.35  
6-2 0.40  
1.81
```

Minimum spanning tree API

Q. How to represent the MST?

```
public class MST
```

```
    MST(EdgeWeightedGraph G)
```

constructor

```
    Iterable<Edge> edges()
```

edges in MST

```
    double weight()
```

weight of MST

```
public static void main(String[] args)
{
    In in = new In(args[0]);
    EdgeWeightedGraph G = new EdgeWeightedGraph(in);
    MST mst = new MST(G);
    for (Edge e : mst.edges())
        StdOut.println(e);
    StdOut.printf("%.2f\n", mst.weight());
}
```

```
% java MST tinyEWG.txt
0-7 0.16
1-7 0.19
0-2 0.26
2-3 0.17
5-7 0.28
4-5 0.35
6-2 0.40
1.81
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

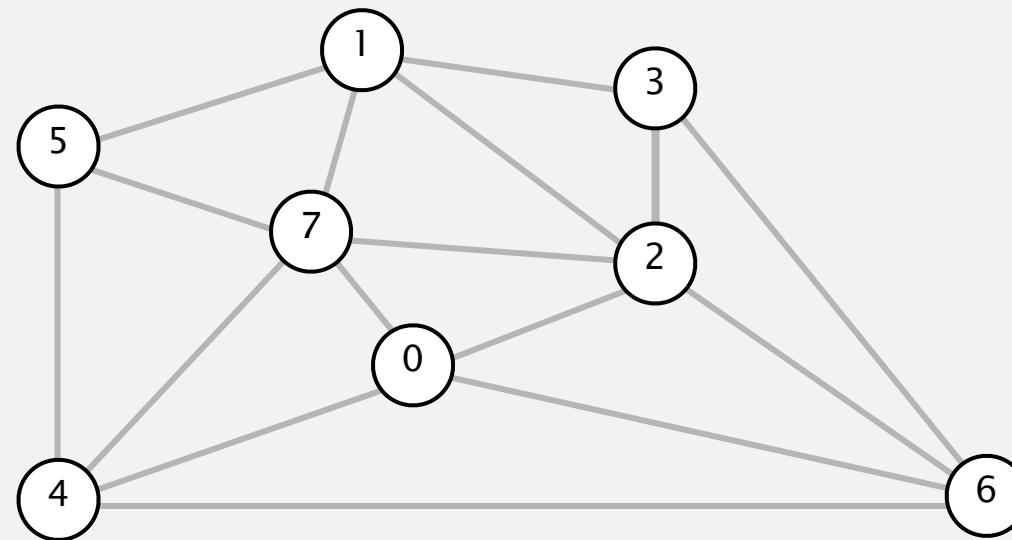
4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ ***Kruskal's algorithm***
- ▶ ***Prim's algorithm***
- ▶ *context*

Kruskal's algorithm demo

Consider edges in ascending order of weight.

- Add next edge to tree T unless doing so would create a cycle.



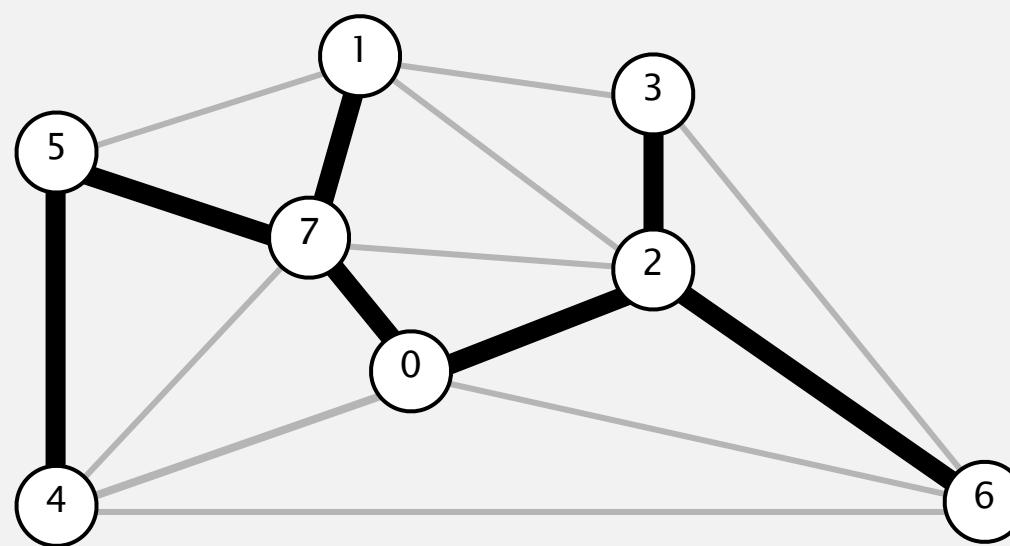
an edge-weighted graph

graph edges sorted by weight	
0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Kruskal's algorithm demo

Consider edges in ascending order of weight.

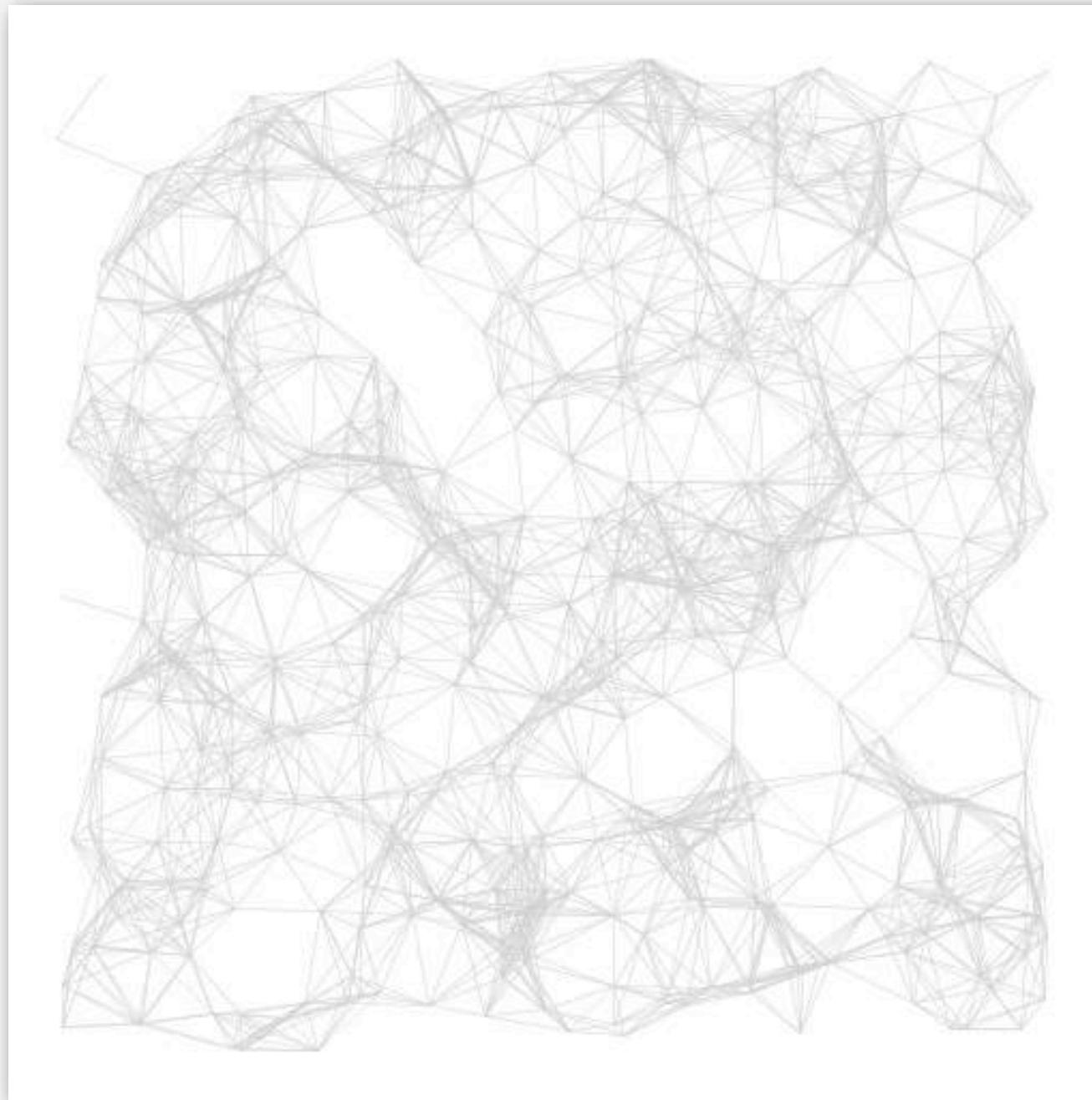
- Add next edge to tree T unless doing so would create a cycle.



a minimum spanning tree

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Kruskal's algorithm: visualization

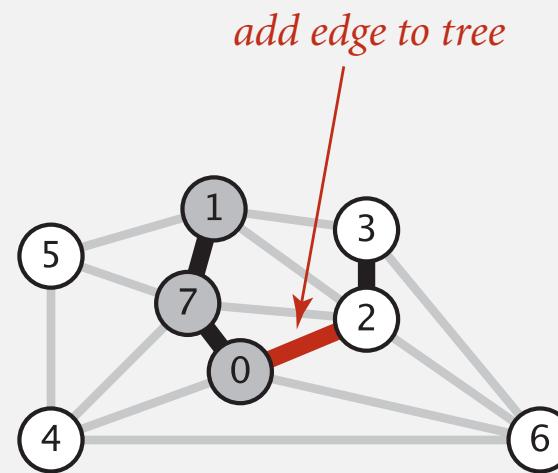


Kruskal's algorithm: correctness proof

Proposition. [Kruskal 1956] Kruskal's algorithm computes the MST.

Pf. Kruskal's algorithm is a special case of the greedy MST algorithm.

- Suppose Kruskal's algorithm colors the edge $e = v-w$ black.
- Cut = set of vertices connected to v in tree T .
- No crossing edge is black.
- No crossing edge has lower weight. Why?

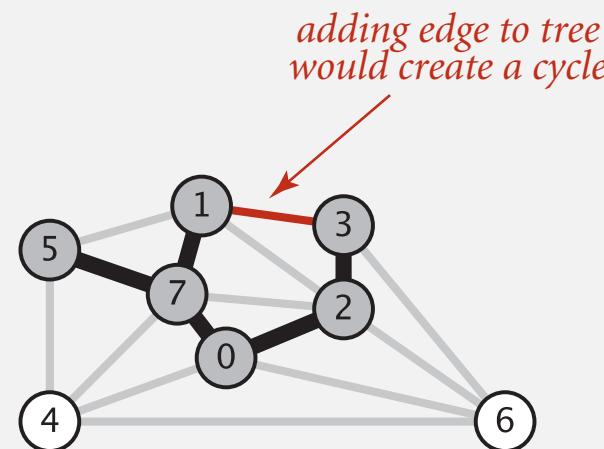
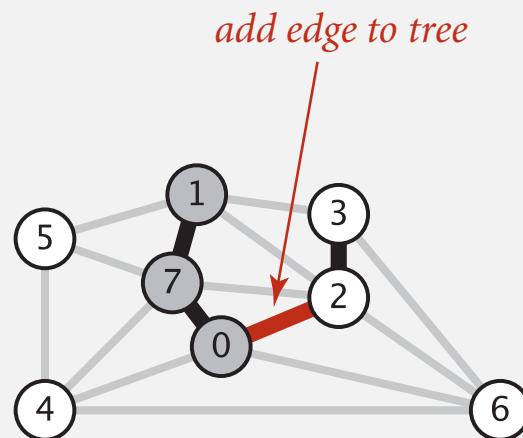


Kruskal's algorithm: implementation challenge

Challenge. Would adding edge $v-w$ to tree T create a cycle? If not, add it.

How difficult?

- $E + V$
- V ← run DFS from v , check if w is reachable
(T has at most $V - 1$ edges)
- $\log V$
- $\log^* V$ ← use the union-find data structure !
- 1

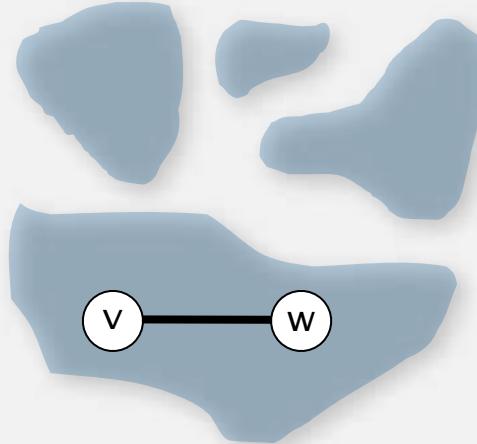


Kruskal's algorithm: implementation challenge

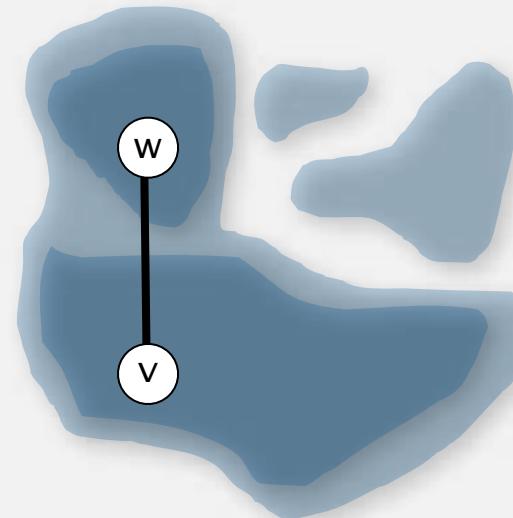
Challenge. Would adding edge $v-w$ to tree T create a cycle? If not, add it.

Efficient solution. Use the **union-find** data structure.

- Maintain a set for each connected component in T .
- If v and w are in same set, then adding $v-w$ would create a cycle.
- To add $v-w$ to T , merge sets containing v and w .



Case 1: adding $v-w$ creates a cycle



Case 2: add $v-w$ to T and merge sets containing v and w

Kruskal's algorithm: Java implementation

```
public class KruskalMST
{
    private Queue<Edge> mst = new Queue<Edge>();

    public KruskalMST(EdgeWeightedGraph G)
    {
        MinPQ<Edge> pq = new MinPQ<Edge>();
        for (Edge e : G.edges())
            pq.insert(e);

        UF uf = new UF(G.V());
        while (!pq.isEmpty() && mst.size() < G.V()-1)
        {
            Edge e = pq.delMin();
            int v = e.either(), w = e.other(v);
            if (!uf.connected(v, w))
            {
                uf.union(v, w);
                mst.enqueue(e);
            }
        }
    }

    public Iterable<Edge> edges()
    {   return mst;   }
}
```

← build priority queue

← greedily add edges to MST

← edge v-w does not create cycle

← merge sets

← add edge to MST

Kruskal's algorithm: running time

Proposition. Kruskal's algorithm computes MST in time proportional to $E \log E$ (in the worst case).

Pf.

operation	frequency	time per op
build pq	1	$E \log E$
delete-min	E	$\log E$
union	V	$\log^* V \dagger$
connected	E	$\log^* V \dagger$

\dagger amortized bound using weighted quick union with path compression

recall: $\log^* V \leq 5$ in this universe



Remark. If edges are already sorted, order of growth is $E \log^* V$.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ ***Kruskal's algorithm***
- ▶ ***Prim's algorithm***
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

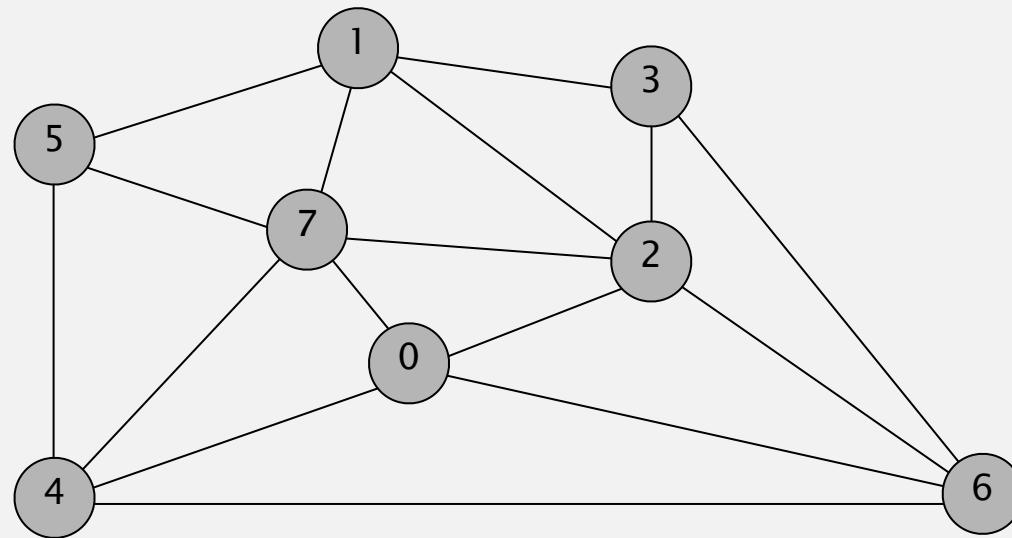
<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ ***Prim's algorithm***
- ▶ *context*

Prim's algorithm demo

- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.

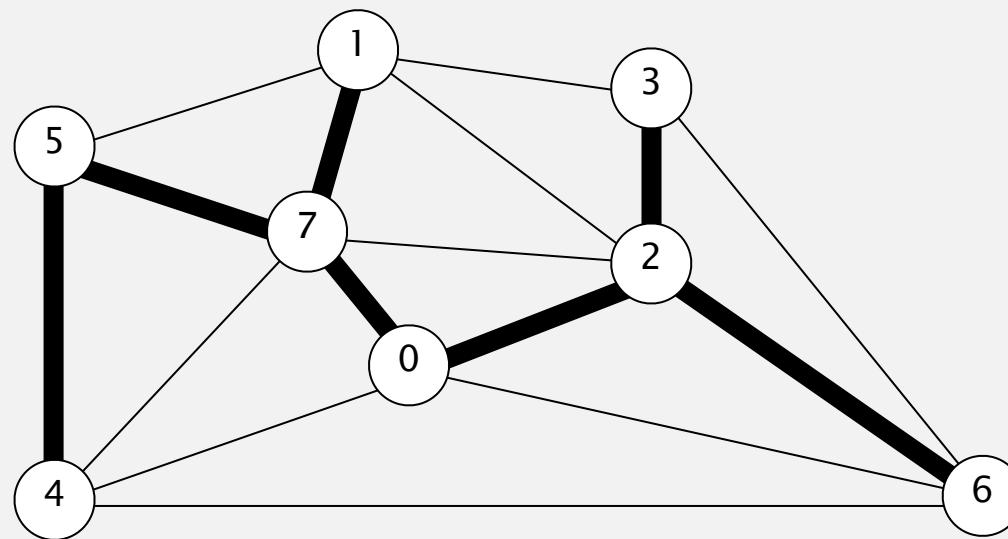


an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm demo

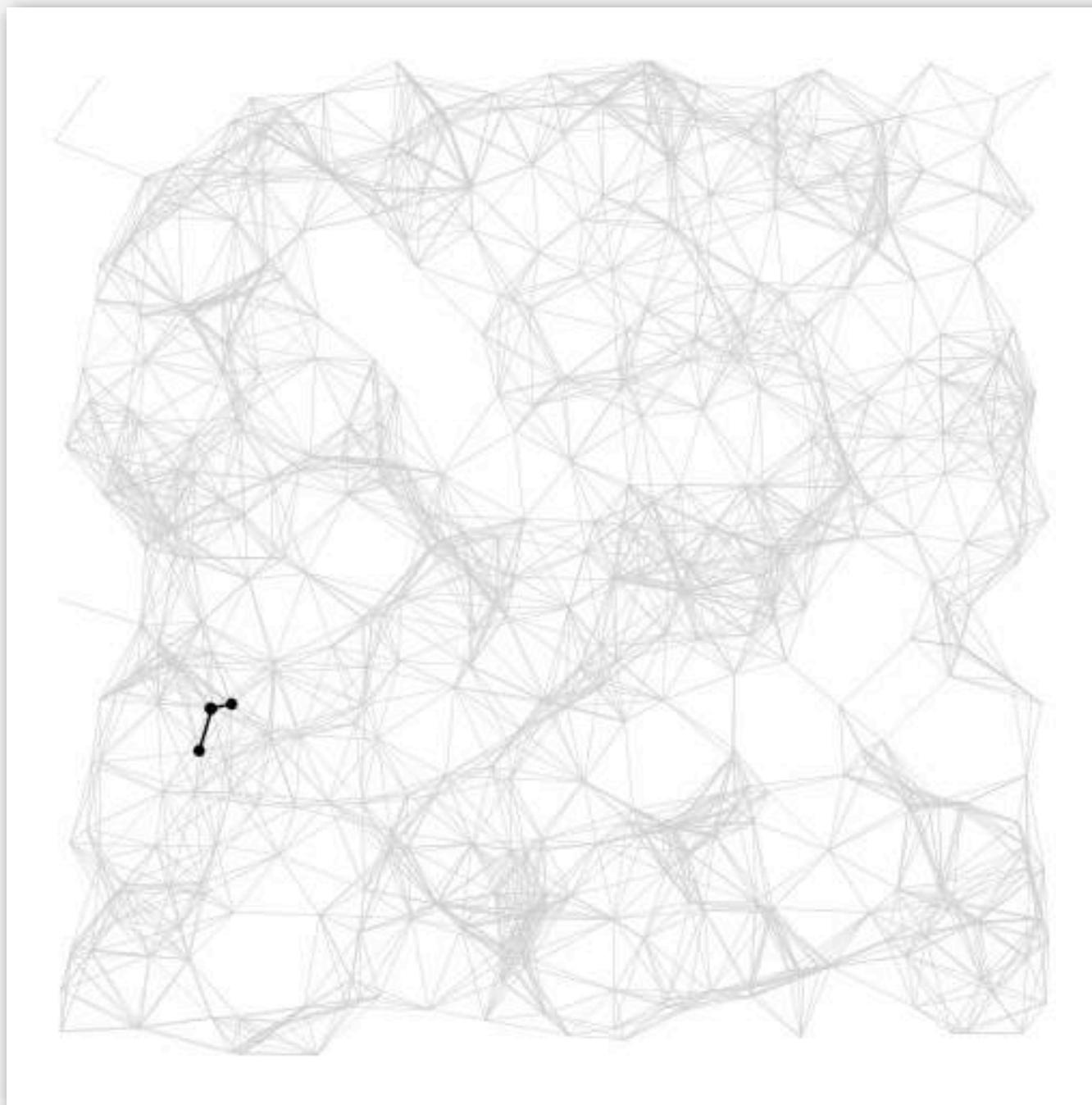
- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.



MST edges

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

Prim's algorithm: visualization



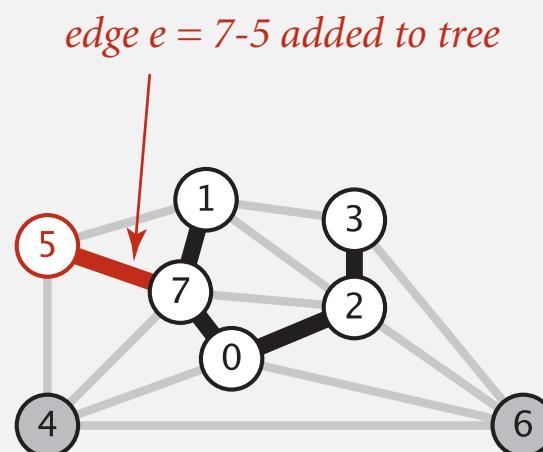
Prim's algorithm: proof of correctness

Proposition. [Jarník 1930, Dijkstra 1957, Prim 1959]

Prim's algorithm computes the MST.

Pf. Prim's algorithm is a special case of the greedy MST algorithm.

- Suppose edge $e = \min$ weight edge connecting a vertex on the tree to a vertex not on the tree.
- Cut = set of vertices connected on tree.
- No crossing edge is black.
- No crossing edge has lower weight.



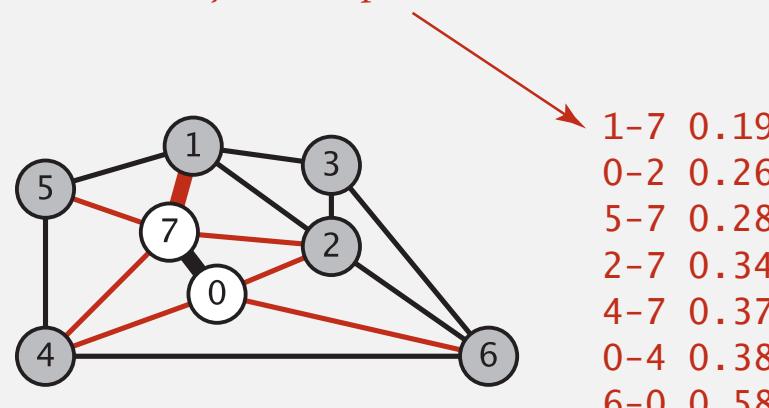
Prim's algorithm: implementation challenge

Challenge. Find the min weight edge with exactly one endpoint in T .

How difficult?

- E ← try all edges
- V
- $\log E$ ← use a priority queue!
- $\log^* E$
- 1

1-7 is min weight edge with
exactly one endpoint in T

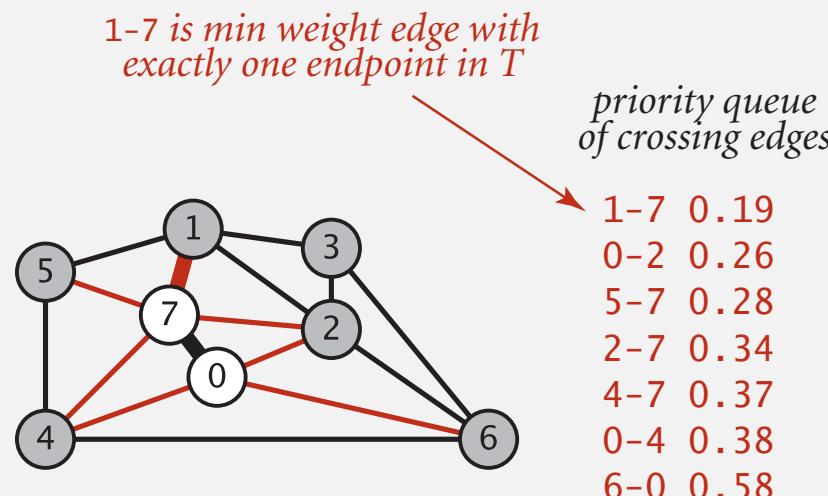


Prim's algorithm: lazy implementation

Challenge. Find the min weight edge with exactly one endpoint in T .

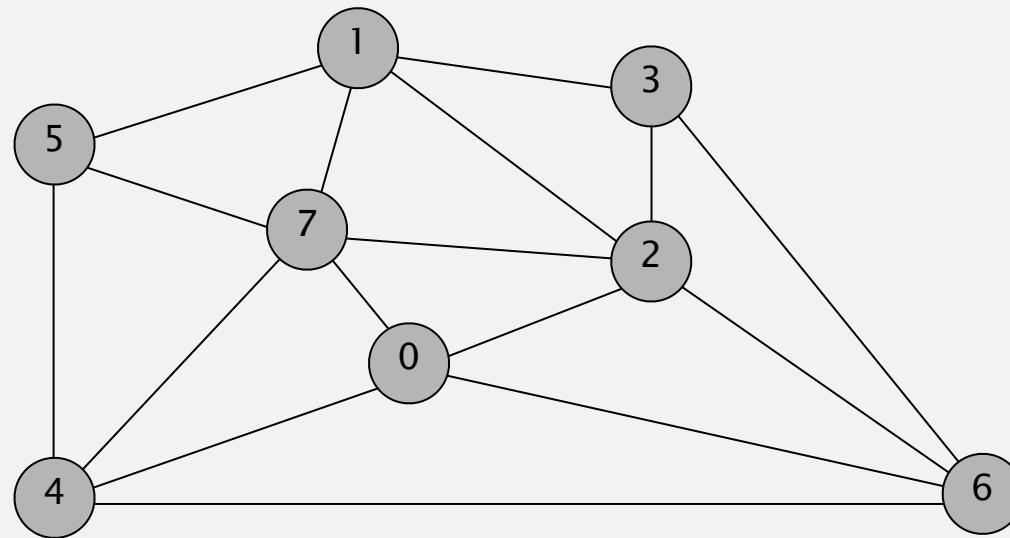
Lazy solution. Maintain a PQ of edges with (at least) one endpoint in T .

- Key = edge; priority = weight of edge.
- Delete-min to determine next edge $e = v-w$ to add to T .
- Disregard if both endpoints v and w are in T .
- Otherwise, let w be the vertex not in T :
 - add to PQ any edge incident to w (assuming other endpoint not in T)
 - add w to T



Prim's algorithm (lazy) demo

- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.

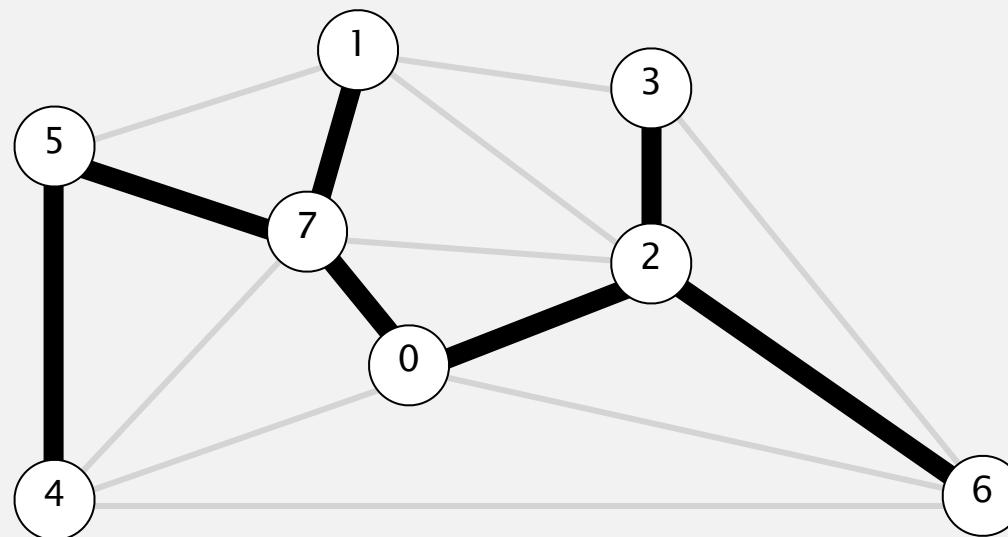


an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm (lazy) demo

- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.



MST edges

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

Prim's algorithm: lazy implementation

```
public class LazyPrimMST
{
    private boolean[] marked;      // MST vertices
    private Queue<Edge> mst;      // MST edges
    private MinPQ<Edge> pq;       // PQ of edges

    public LazyPrimMST(WeightedGraph G)
    {
        pq = new MinPQ<Edge>();
        mst = new Queue<Edge>();
        marked = new boolean[G.V()];
        visit(G, 0);
    }

    while (!pq.isEmpty() && mst.size() < G.V() - 1)
    {
        Edge e = pq.delMin();
        int v = e.either(), w = e.other(v);
        if (marked[v] && marked[w]) continue;
        mst.enqueue(e);
        if (!marked[v]) visit(G, v);
        if (!marked[w]) visit(G, w);
    }
}
```

assume G is connected

repeatedly delete the min weight edge $e = v-w$ from PQ

ignore if both endpoints in T

add edge e to tree

add v or w to tree

Prim's algorithm: lazy implementation

```
private void visit(WeightedGraph G, int v)
{
    marked[v] = true;
    for (Edge e : G.adj(v))
        if (!marked[e.other(v)])
            pq.insert(e);
}

public Iterable<Edge> mst()
{   return mst; }
```

← add v to T

← for each edge $e = v-w$, add to PQ if w not already in T

Lazy Prim's algorithm: running time

Proposition. Lazy Prim's algorithm computes the MST in time proportional to $E \log E$ and extra space proportional to E (in the worst case).

Pf.

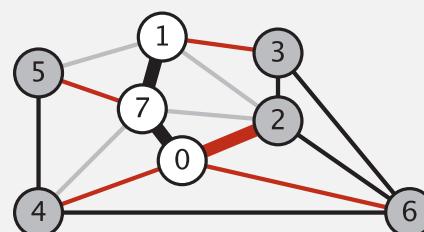
operation	frequency	binary heap
delete min	E	$\log E$
insert	E	$\log E$

Prim's algorithm: eager implementation

Challenge. Find min weight edge with exactly one endpoint in T .

Eager solution. Maintain a PQ of vertices connected by an edge to T , where priority of vertex v = weight of shortest edge connecting v to T .

- Delete min vertex v and add its associated edge $e = v-w$ to T .
- Update PQ by considering all edges $e = v-x$ incident to v
 - ignore if x is already in T
 - add x to PQ if not already on it
 - decrease priority of x if $v-x$ becomes shortest edge connecting x to T

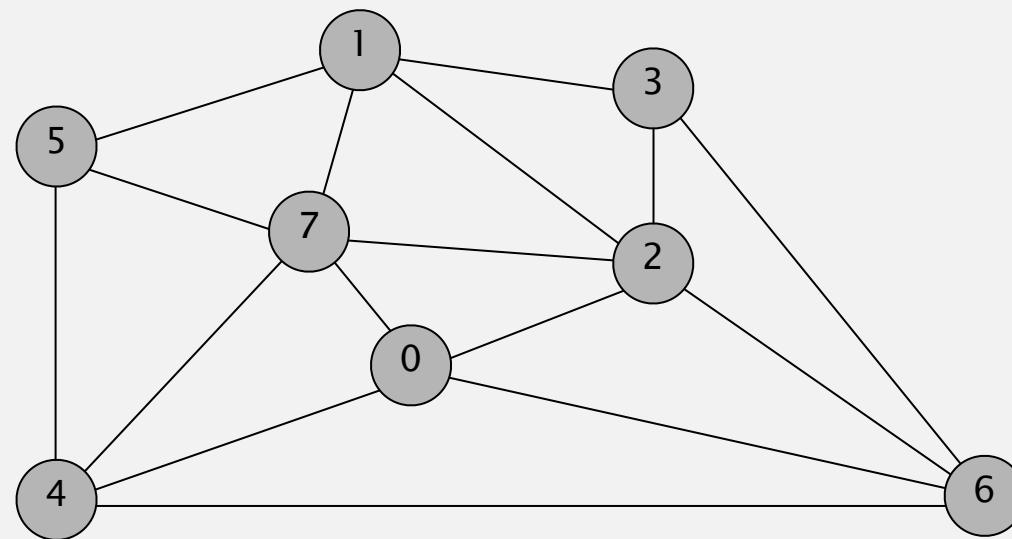


0		
1	1-7	0.19
2	0-2	0.26
3	1-3	0.29
4	0-4	0.38
5	5-7	0.28
6	6-0	0.58
7	0-7	0.16

black: on MST
red: on PQ

Prim's algorithm (eager) demo

- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.

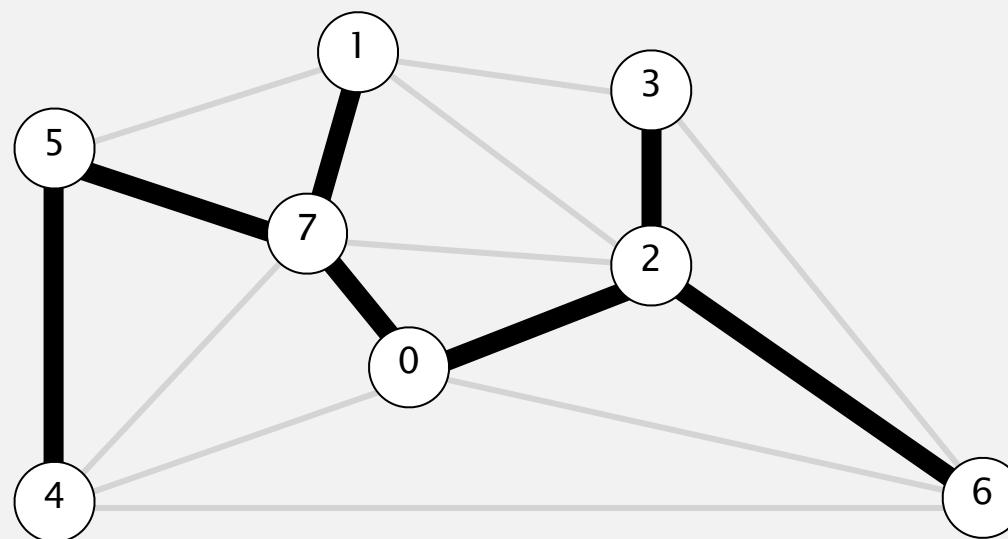


an edge-weighted graph

0-7	0.16
2-3	0.17
1-7	0.19
0-2	0.26
5-7	0.28
1-3	0.29
1-5	0.32
2-7	0.34
4-5	0.35
1-2	0.36
4-7	0.37
0-4	0.38
6-2	0.40
3-6	0.52
6-0	0.58
6-4	0.93

Prim's algorithm (eager) demo

- Start with vertex 0 and greedily grow tree T .
- Add to T the min weight edge with exactly one endpoint in T .
- Repeat until $V - 1$ edges.



v	edgeTo[]	distTo[]
0	-	-
7	0-7	0.16
1	1-7	0.19
2	0-2	0.26
3	2-3	0.17
5	5-7	0.28
4	4-5	0.35
6	6-2	0.40

MST edges

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

Indexed priority queue

Associate an index between 0 and $N - 1$ with each key in a priority queue.

- Client can insert and delete-the-minimum.
- Client can change the key by specifying the index.

public class IndexMinPQ<Key extends Comparable<Key>>	
IndexMinPQ(int N)	<i>create indexed priority queue with indices 0, 1, ..., N-1</i>
void insert(int i, Key key)	<i>associate key with index i</i>
void decreaseKey(int i, Key key)	<i>decrease the key associated with index i</i>
boolean contains(int i)	<i>is i an index on the priority queue?</i>
int delMin()	<i>remove a minimal key and return its associated index</i>
boolean isEmpty()	<i>is the priority queue empty?</i>
int size()	<i>number of entries in the priority queue</i>

Indexed priority queue implementation

Implementation.

- Start with same code as MinPQ.
- Maintain parallel arrays `keys[]`, `pq[]`, and `qp[]` so that:
 - `keys[i]` is the priority of `i`
 - `pq[i]` is the index of the key in heap position `i`
 - `qp[i]` is the heap position of the key with index `i`
- Use `swim(qp[i])` implement `decreaseKey(i, key)`.

i	0	1	2	3	4	5	6	7	8
keys[i]	A	S	O	R	T	I	N	G	-
pq[i]	-	0	6	7	2	1	5	4	3
qp[i]	1	5	4	8	7	6	2	3	-

Diagram of a binary heap structure:

```
graph TD; A((A)) -- 1 --> N((N)); A -- 1 --> G((G)); N((N)) -- 2 --> O((O)); N((N)) -- 2 --> S((S)); G((G)) -- 3 --> I((I)); G((G)) -- 3 --> T((T)); O((O)) -- 4 --> R((R));
```

The heap has 9 nodes labeled A through T. Node N is highlighted with a red oval and has a red number 2 above it, indicating it is the parent of nodes O and S. Node A is at index 6 in the array, which corresponds to pq[6] = 0 and qp[0] = 6. Node N is at index 2 in the array, which corresponds to pq[2] = 6 and qp[6] = 2. Node A is at index 1 in the array, which corresponds to pq[1] = - and qp[-] = 1.

Prim's algorithm: running time

Depends on PQ implementation: V insert, V delete-min, E decrease-key.

PQ implementation	insert	delete-min	decrease-key	total
array	1	V	1	V^2
binary heap	$\log V$	$\log V$	$\log V$	$E \log V$
d-way heap (Johnson 1975)	$d \log_d V$	$d \log_d V$	$\log_d V$	$E \log_{E/V} V$
Fibonacci heap (Fredman-Tarjan 1984)	1^\dagger	$\log V^\dagger$	1^\dagger	$E + V \log V$

\dagger amortized

Bottom line.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci heap best in theory, but not worth implementing.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ ***Prim's algorithm***
- ▶ *context*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ **context**

Does a linear-time MST algorithm exist?

deterministic compare-based MST algorithms

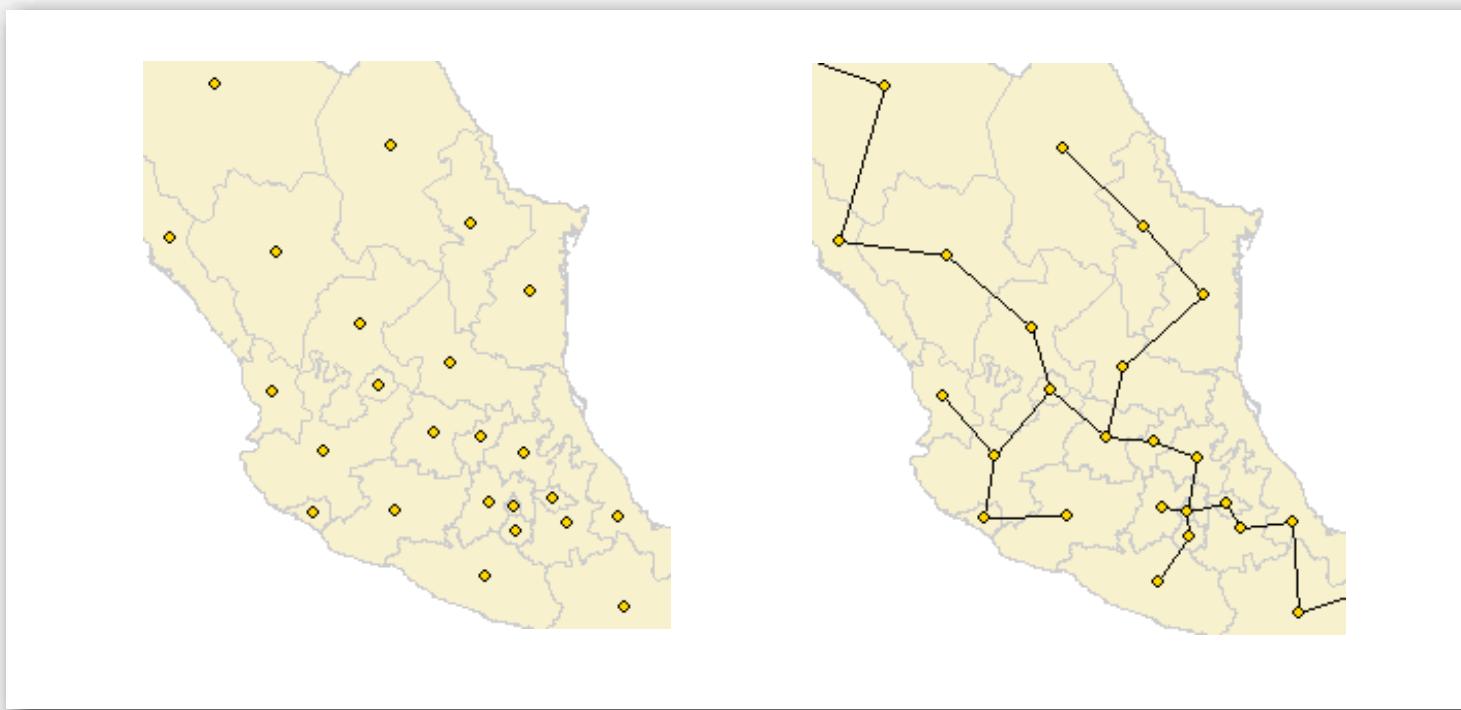
year	worst case	discovered by
1975	$E \log \log V$	Yao
1976	$E \log \log V$	Cheriton-Tarjan
1984	$E \log^* V, E + V \log V$	Fredman-Tarjan
1986	$E \log (\log^* V)$	Gabow-Galil-Spencer-Tarjan
1997	$E \alpha(V) \log \alpha(V)$	Chazelle
2000	$E \alpha(V)$	Chazelle
2002	optimal	Pettie-Ramachandran
20xx	E	???



Remark. Linear-time randomized MST algorithm (Karger-Klein-Tarjan 1995).

Euclidean MST

Given N points in the plane, find MST connecting them, where the distances between point pairs are their **Euclidean** distances.



Brute force. Compute $\sim N^2 / 2$ distances and run Prim's algorithm.

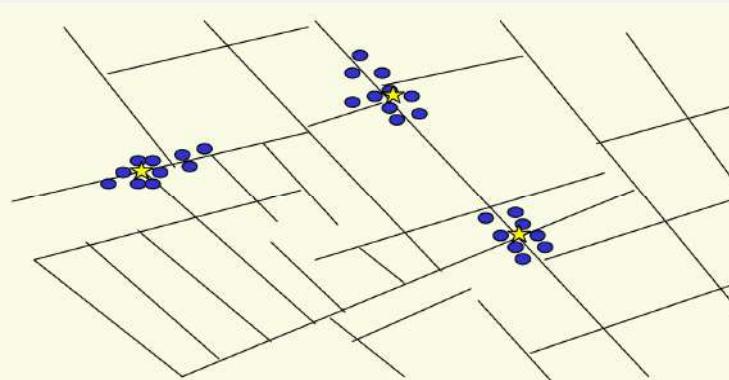
Ingenuity. Exploit geometry and do it in $\sim c N \log N$.

Scientific application: clustering

k-clustering. Divide a set of objects classify into k coherent groups.

Distance function. Numeric value specifying "closeness" of two objects.

Goal. Divide into clusters so that objects in different clusters are far apart.



outbreak of cholera deaths in London in 1850s (Nina Mishra)

Applications.

- Routing in mobile ad hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases.
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

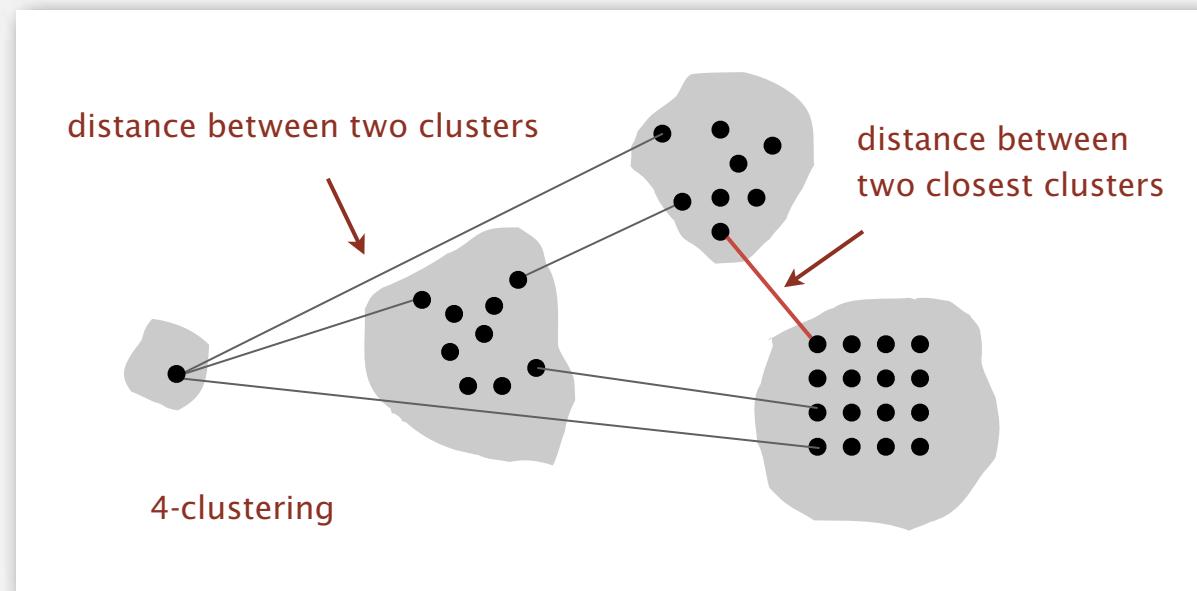
Single-link clustering

k-clustering. Divide a set of objects classify into k coherent groups.

Distance function. Numeric value specifying "closeness" of two objects.

Single link. Distance between two clusters equals the distance between the two closest objects (one in each cluster).

Single-link clustering. Given an integer k, find a k-clustering that maximizes the distance between two closest clusters.

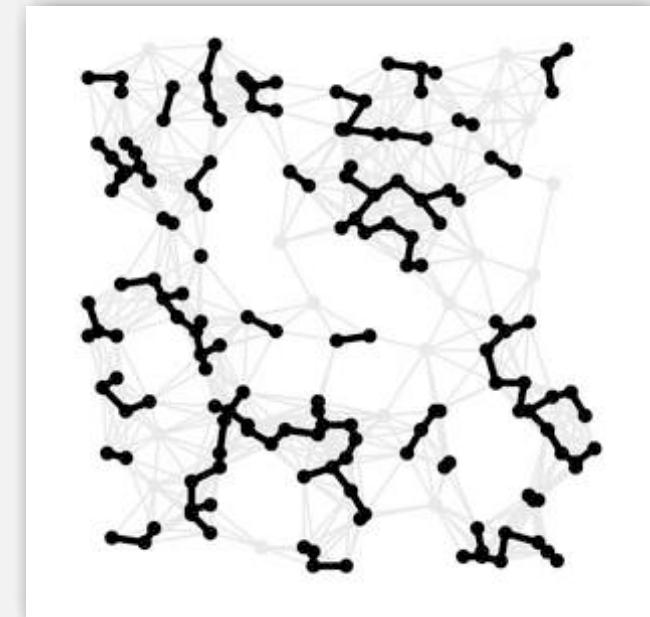


Single-link clustering algorithm

“Well-known” algorithm in science literature for single-link clustering:

- Form V clusters of one object each.
- Find the closest pair of objects such that each object is in a different cluster, and merge the two clusters.
- Repeat until there are exactly k clusters.

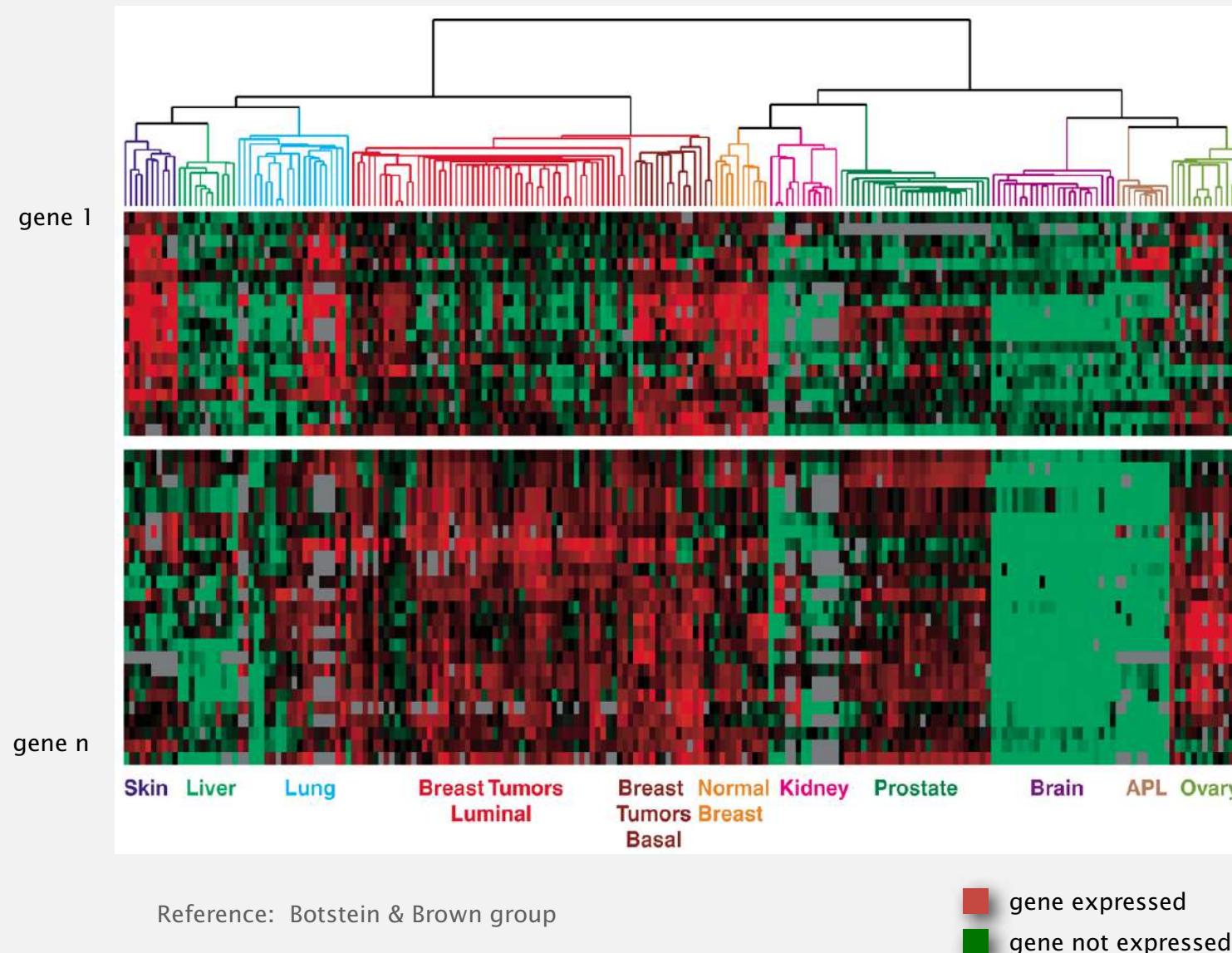
Observation. This is Kruskal's algorithm
(stop when k connected components).



Alternate solution. Run Prim's algorithm and delete $k-1$ max weight edges.

Dendrogram of cancers in human

Tumors in similar tissues cluster together.



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.3 MINIMUM SPANNING TREES

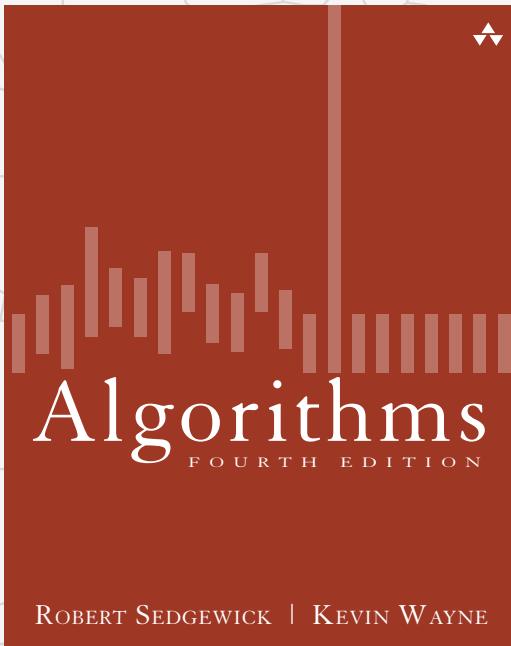
- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ **context**



http://algs4.cs.princeton.edu

4.3 MINIMUM SPANNING TREES

- ▶ *introduction*
- ▶ *greedy algorithm*
- ▶ *edge-weighted graph API*
- ▶ *Kruskal's algorithm*
- ▶ *Prim's algorithm*
- ▶ *context*



<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

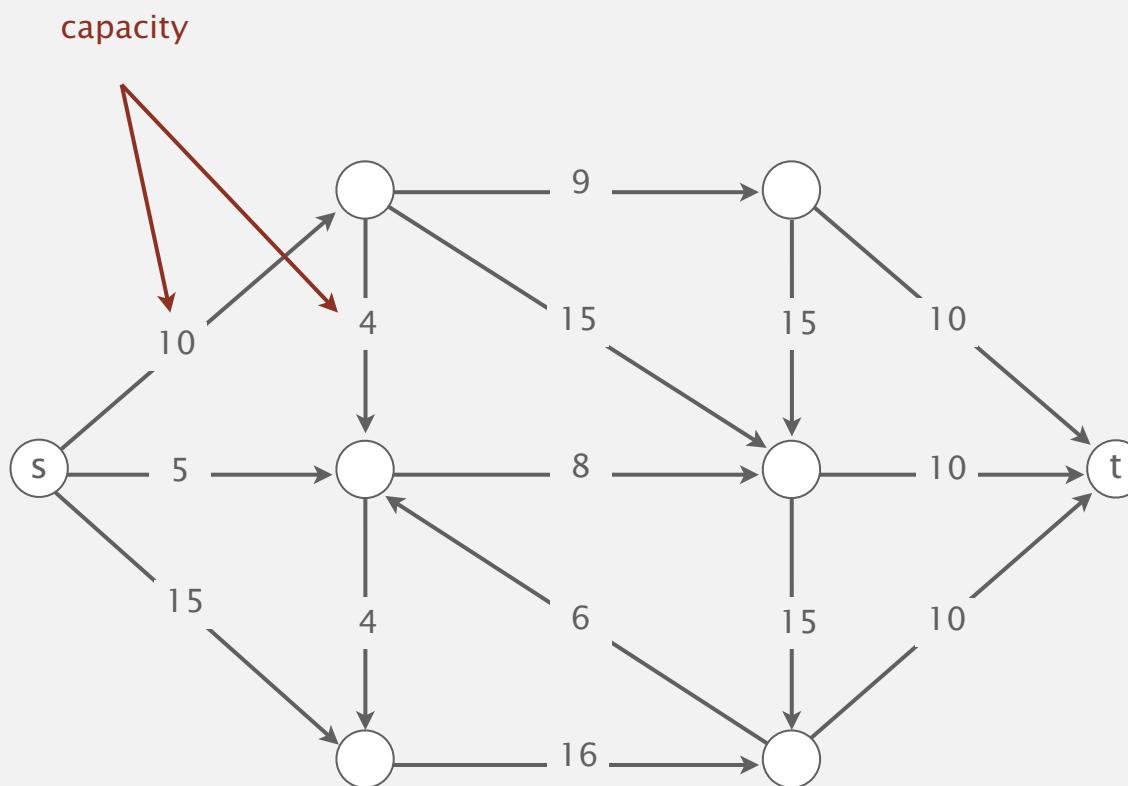
- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Mincut problem

Input. An edge-weighted digraph, source vertex s , and target vertex t .



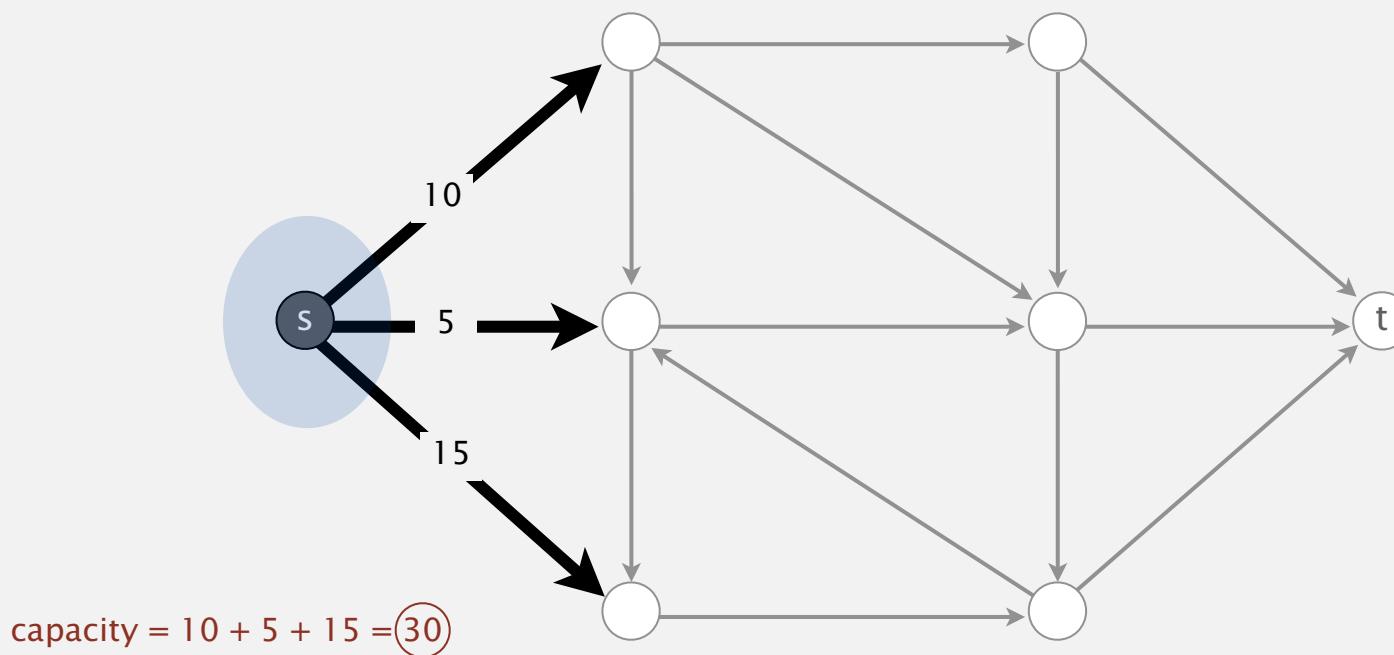
each edge has a
positive capacity



Mincut problem

Def. A *st-cut (cut)* is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B .

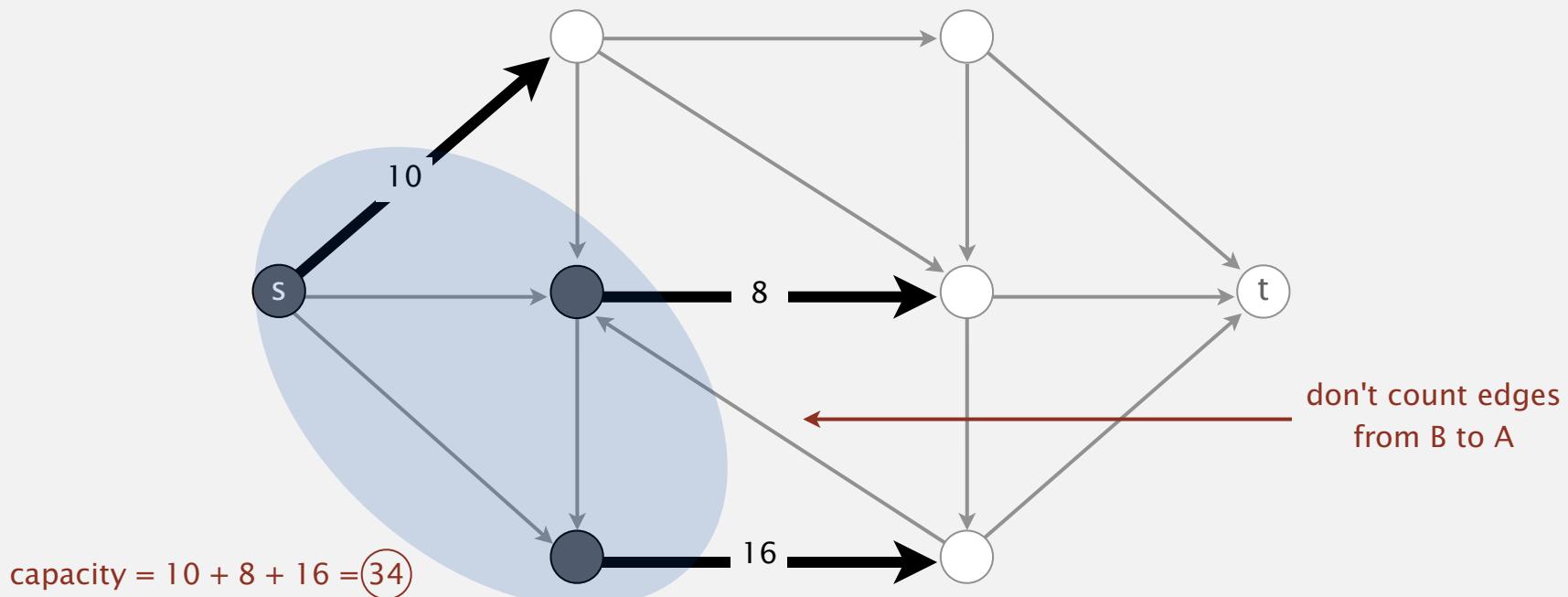
Def. Its **capacity** is the sum of the capacities of the edges from A to B .



Mincut problem

Def. A *st-cut* (cut) is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B .

Def. Its **capacity** is the sum of the capacities of the edges from A to B .

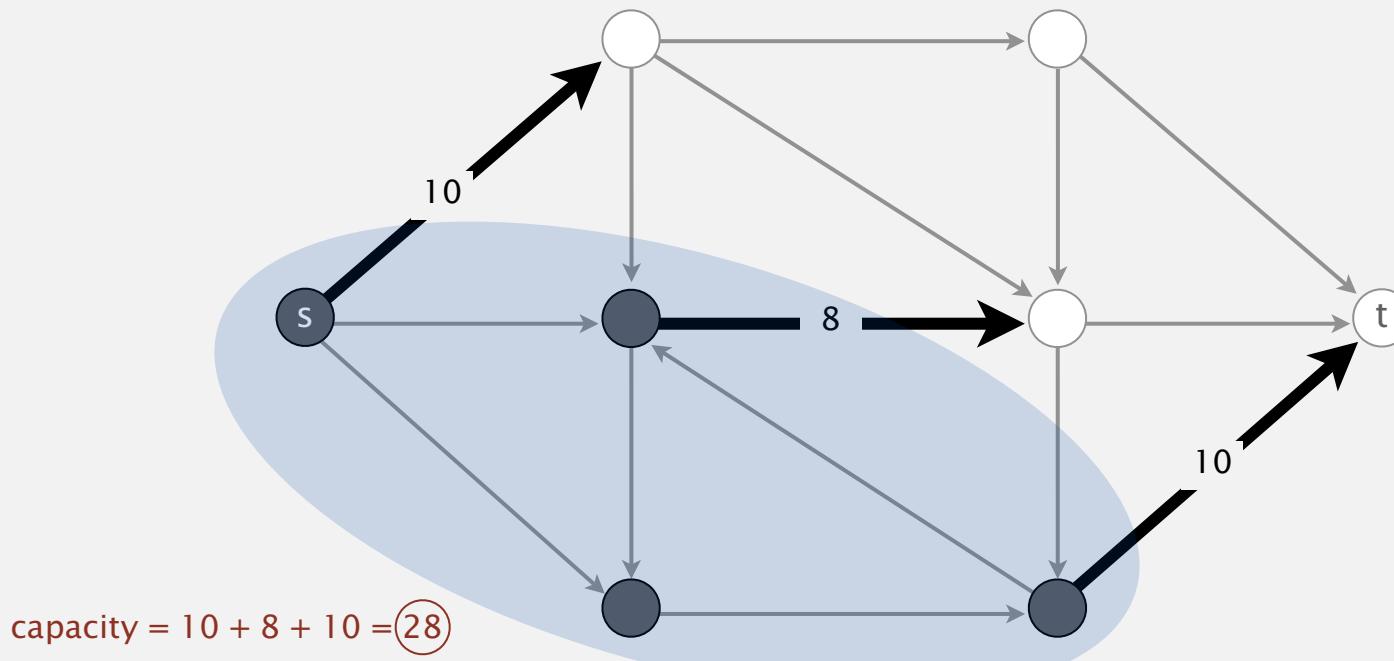


Mincut problem

Def. A *st-cut (cut)* is a partition of the vertices into two disjoint sets, with s in one set A and t in the other set B .

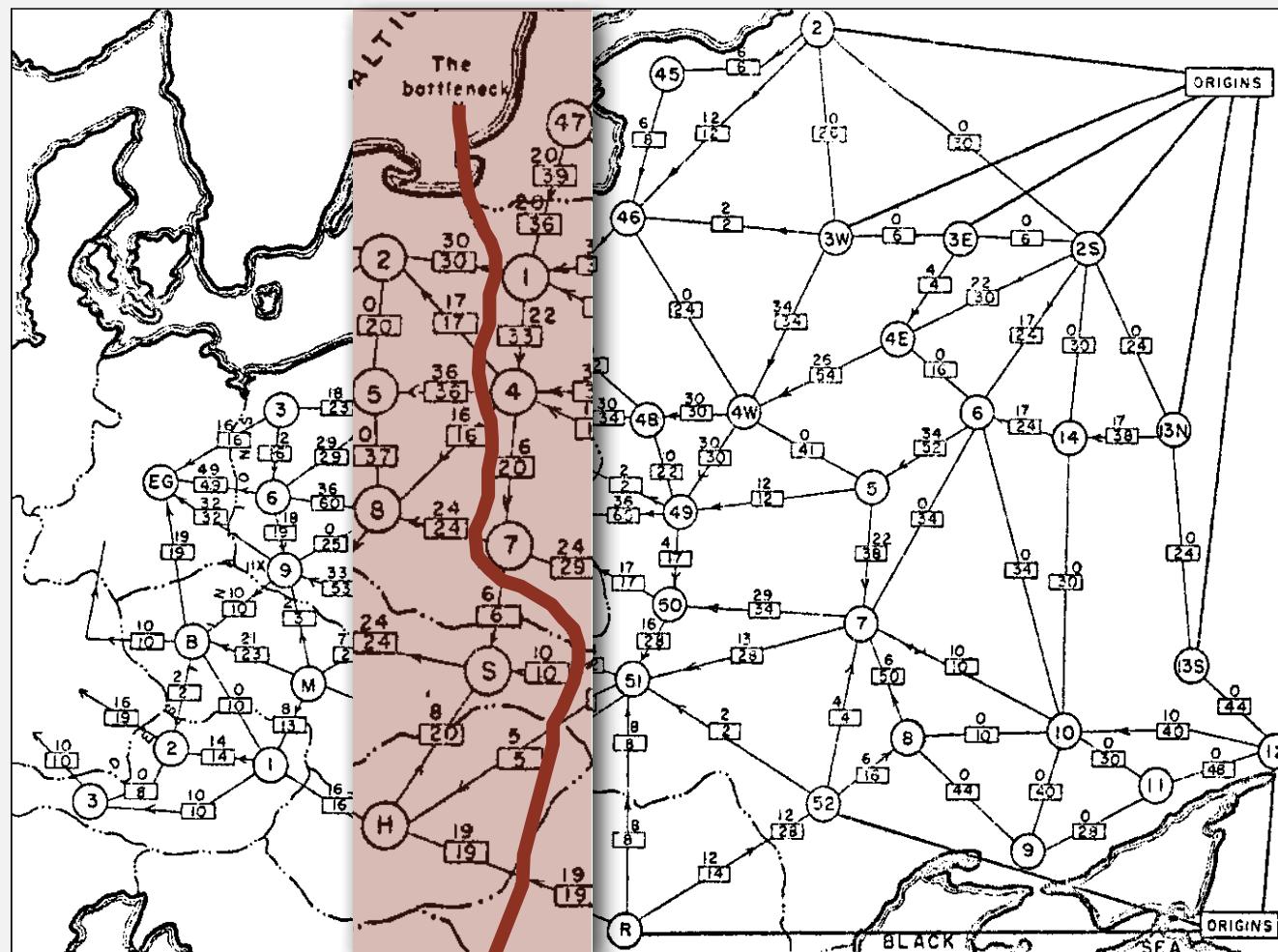
Def. Its **capacity** is the sum of the capacities of the edges from A to B .

Minimum st-cut (mincut) problem. Find a cut of minimum capacity.



Mincut application (1950s)

"Free world" goal. Cut supplies (if cold war turns into real war).



rail network connecting Soviet Union with Eastern European countries (map declassified by Pentagon in 1999)

Potential mincut application (2010s)

Government-in-power's goal. Cut off communication to set of people.

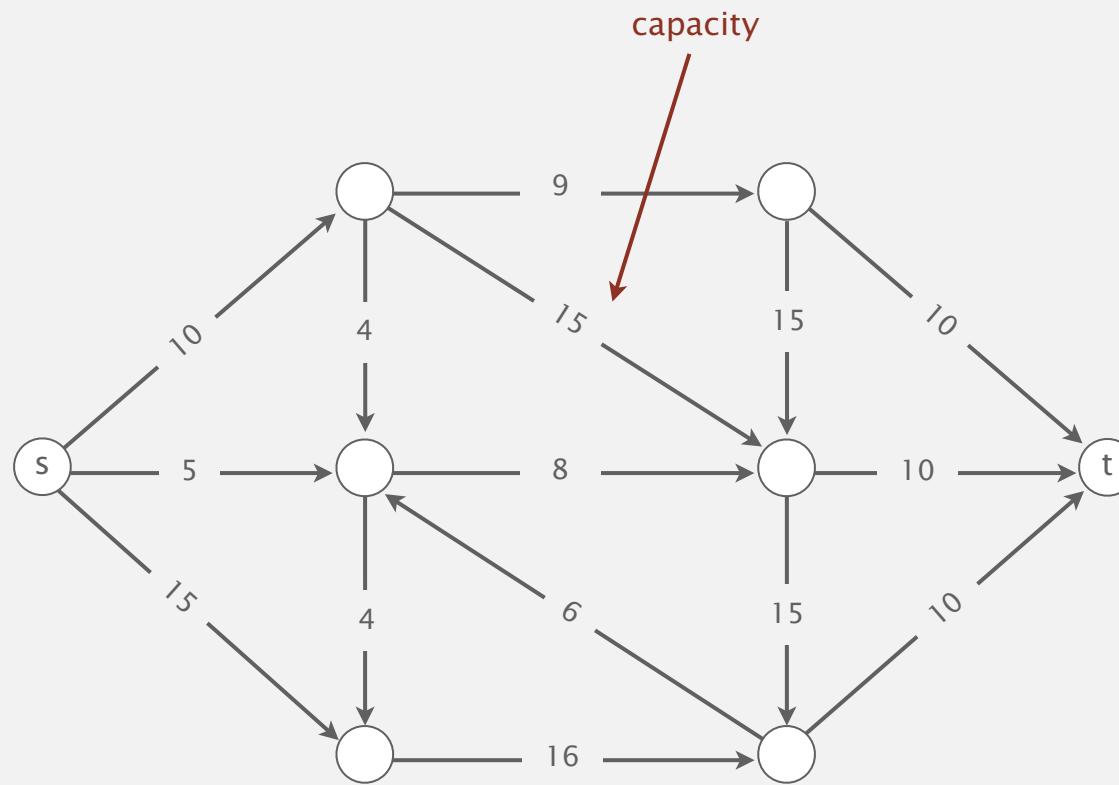


Maxflow problem

Input. An edge-weighted digraph, source vertex s , and target vertex t .



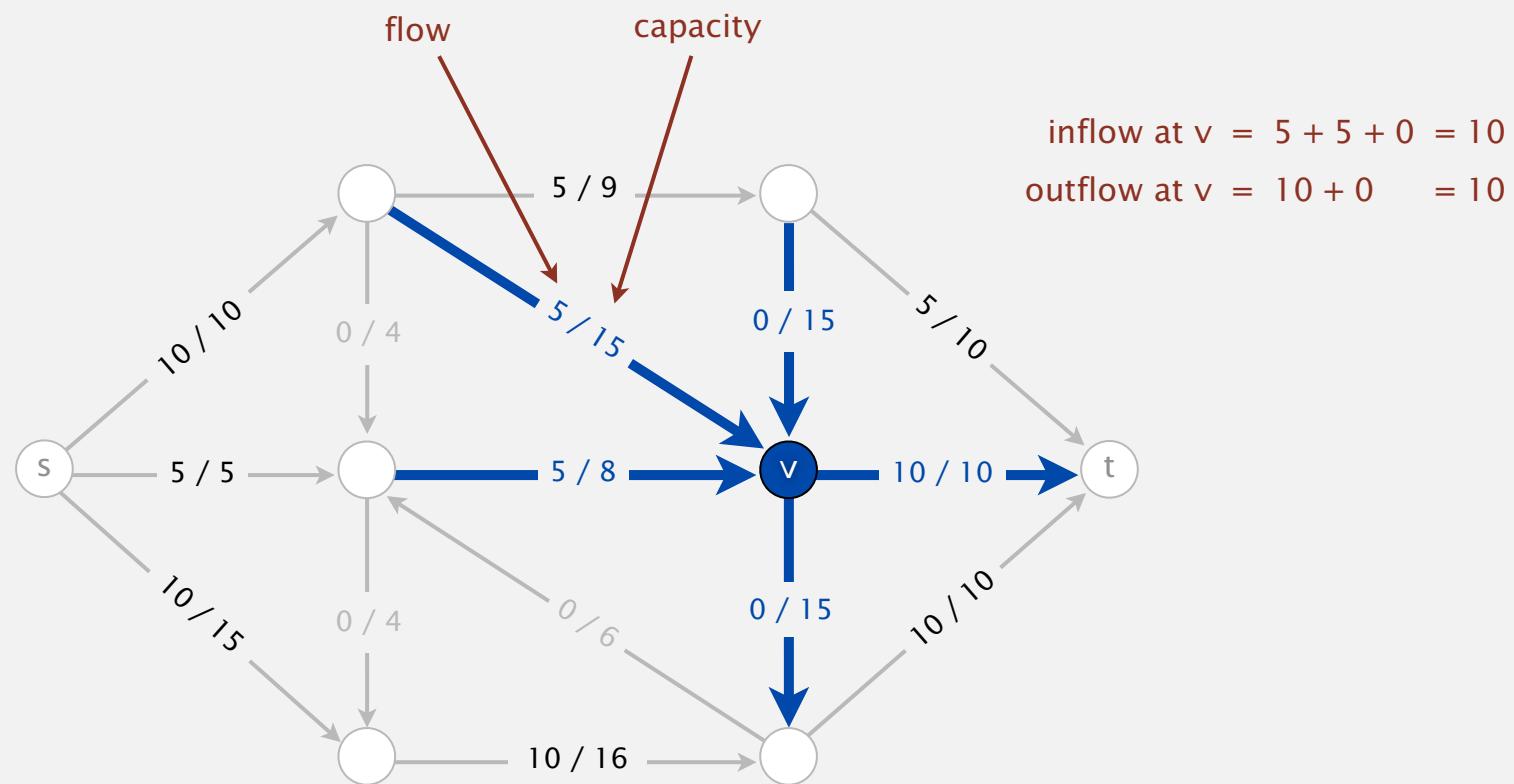
each edge has a
positive capacity



Maxflow problem

Def. An *st-flow* (*flow*) is an assignment of values to the edges such that:

- Capacity constraint: $0 \leq \text{edge's flow} \leq \text{edge's capacity}$.
 - Local equilibrium: $\text{inflow} = \text{outflow}$ at every vertex (except s and t).



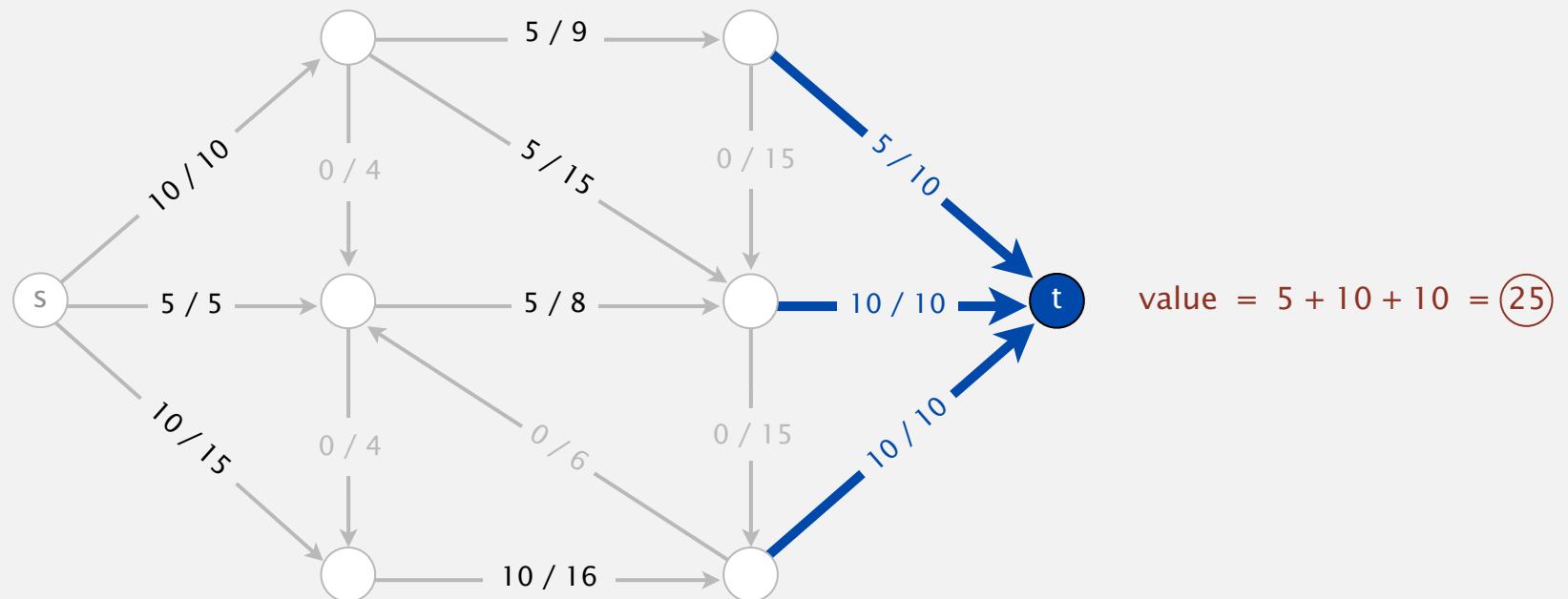
Maxflow problem

Def. An *st-flow (flow)* is an assignment of values to the edges such that:

- Capacity constraint: $0 \leq$ edge's flow \leq edge's capacity.
- Local equilibrium: inflow = outflow at every vertex (except s and t).

Def. The **value** of a flow is the inflow at t .

we assume no edge points to s or from t



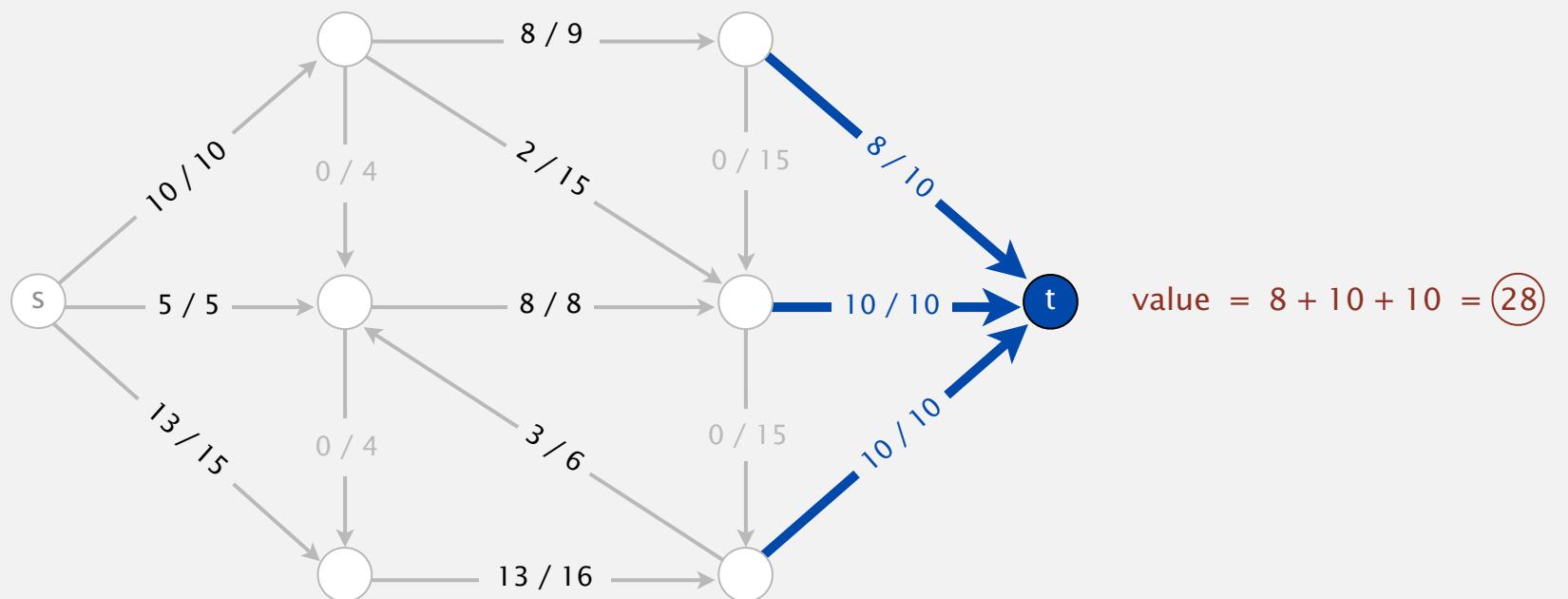
Maxflow problem

Def. An *st-flow (flow)* is an assignment of values to the edges such that:

- Capacity constraint: $0 \leq$ edge's flow \leq edge's capacity.
- Local equilibrium: inflow = outflow at every vertex (except s and t).

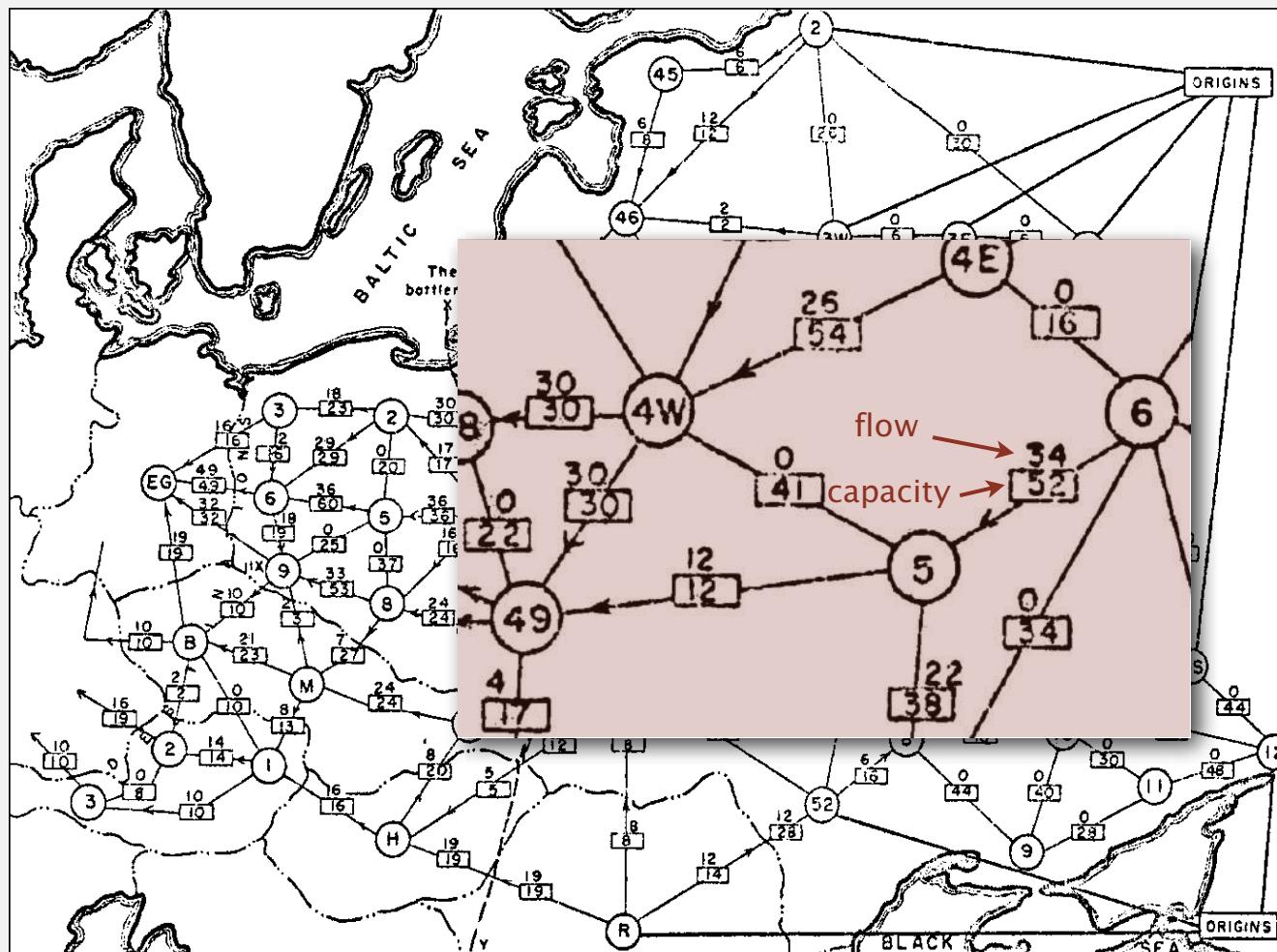
Def. The **value** of a flow is the inflow at t .

Maximum st-flow (maxflow) problem. Find a flow of maximum value.



Maxflow application (1950s)

Soviet Union goal. Maximize flow of supplies to Eastern Europe.



rail network connecting Soviet Union with Eastern European countries

(map declassified by Pentagon in 1999)

Potential maxflow application (2010s)

"Free world" goal. Maximize flow of information to specified set of people.



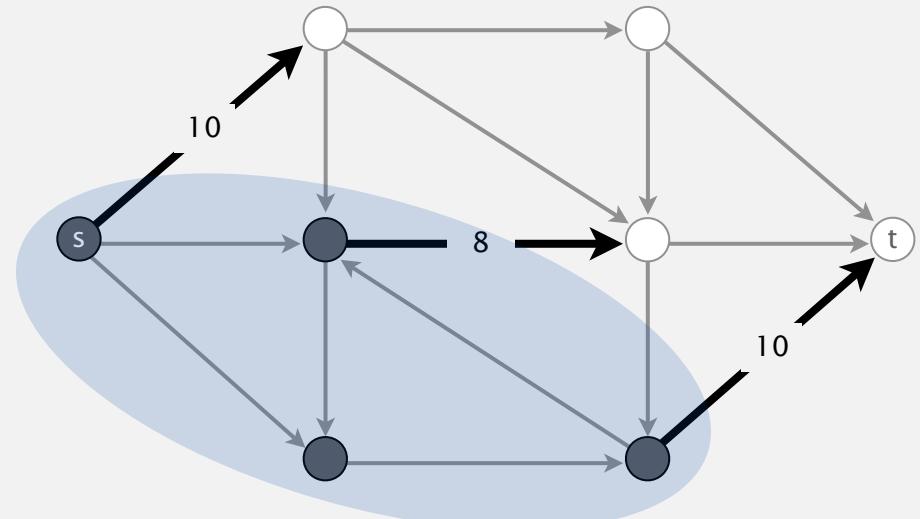
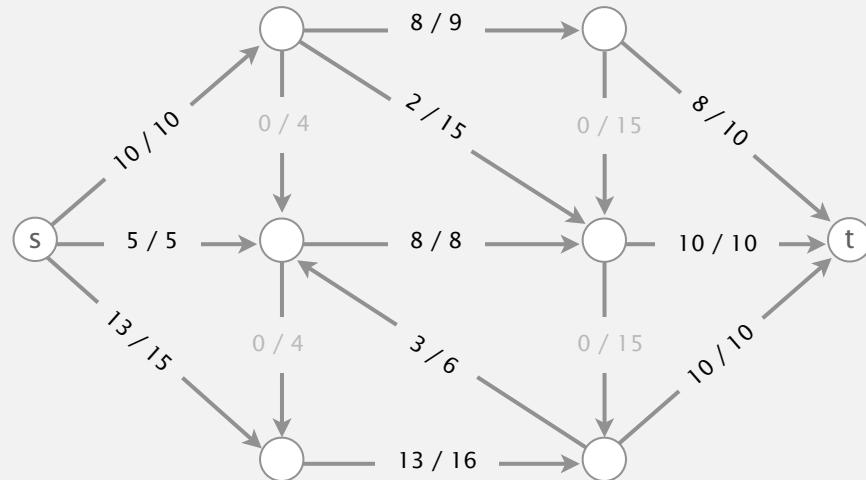
facebook graph

Summary

Input. A weighted digraph, source vertex s , and target vertex t .

Mincut problem. Find a cut of minimum capacity.

Maxflow problem. Find a flow of maximum value.



Remarkable fact. These two problems are dual!

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

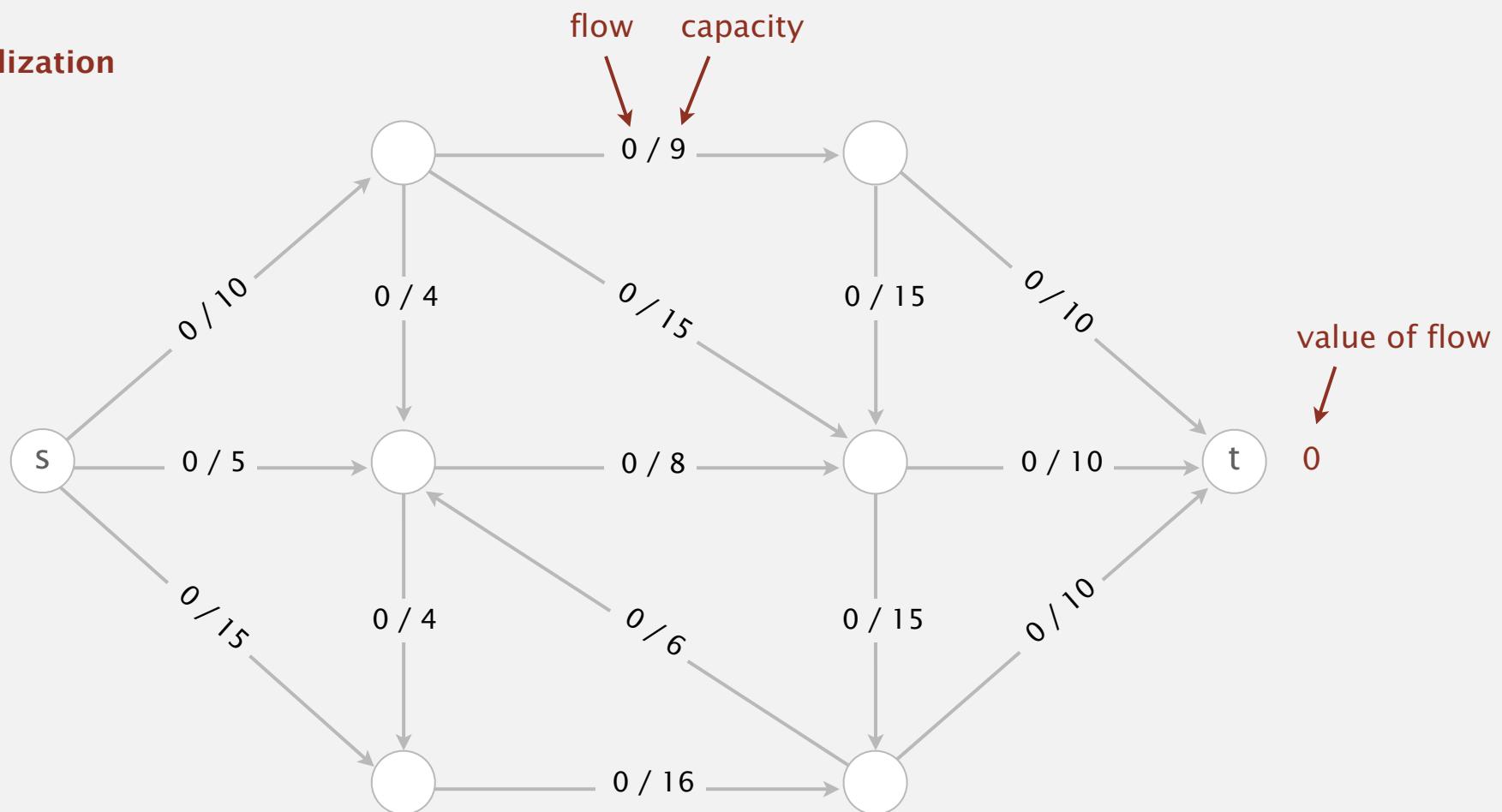
6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ ***Ford-Fulkerson algorithm***
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Ford-Fulkerson algorithm

Initialization. Start with 0 flow.

initialization

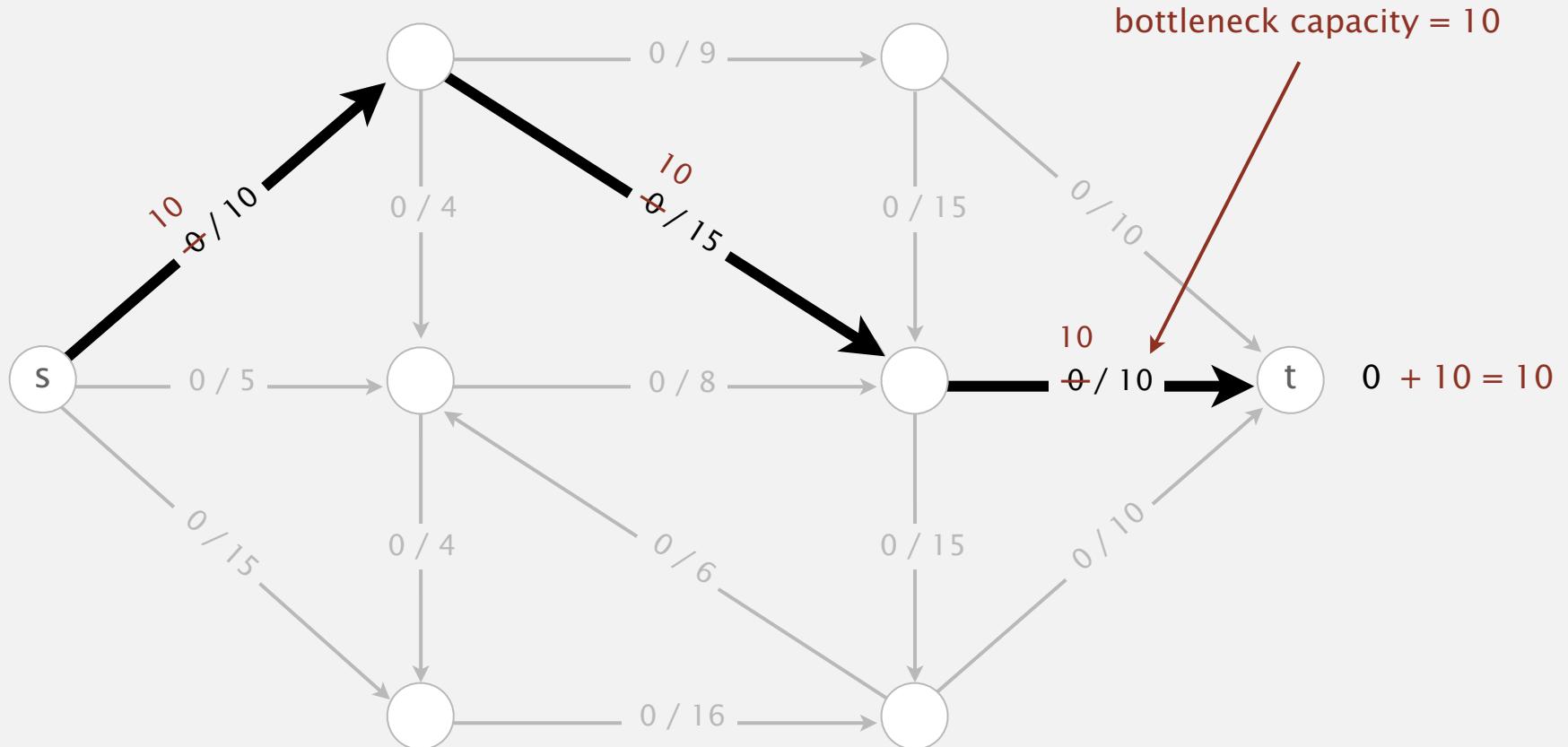


Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

1st augmenting path

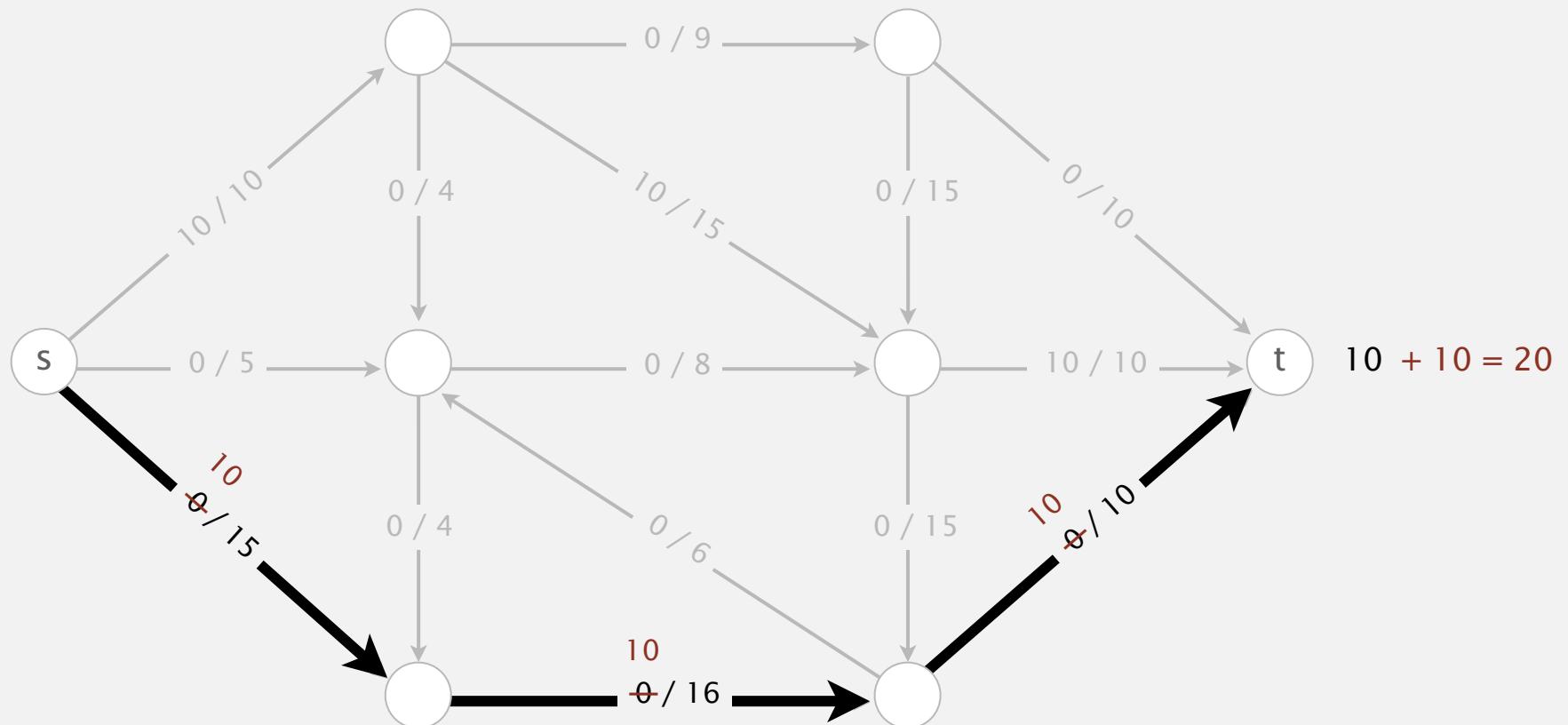


Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

2nd augmenting path

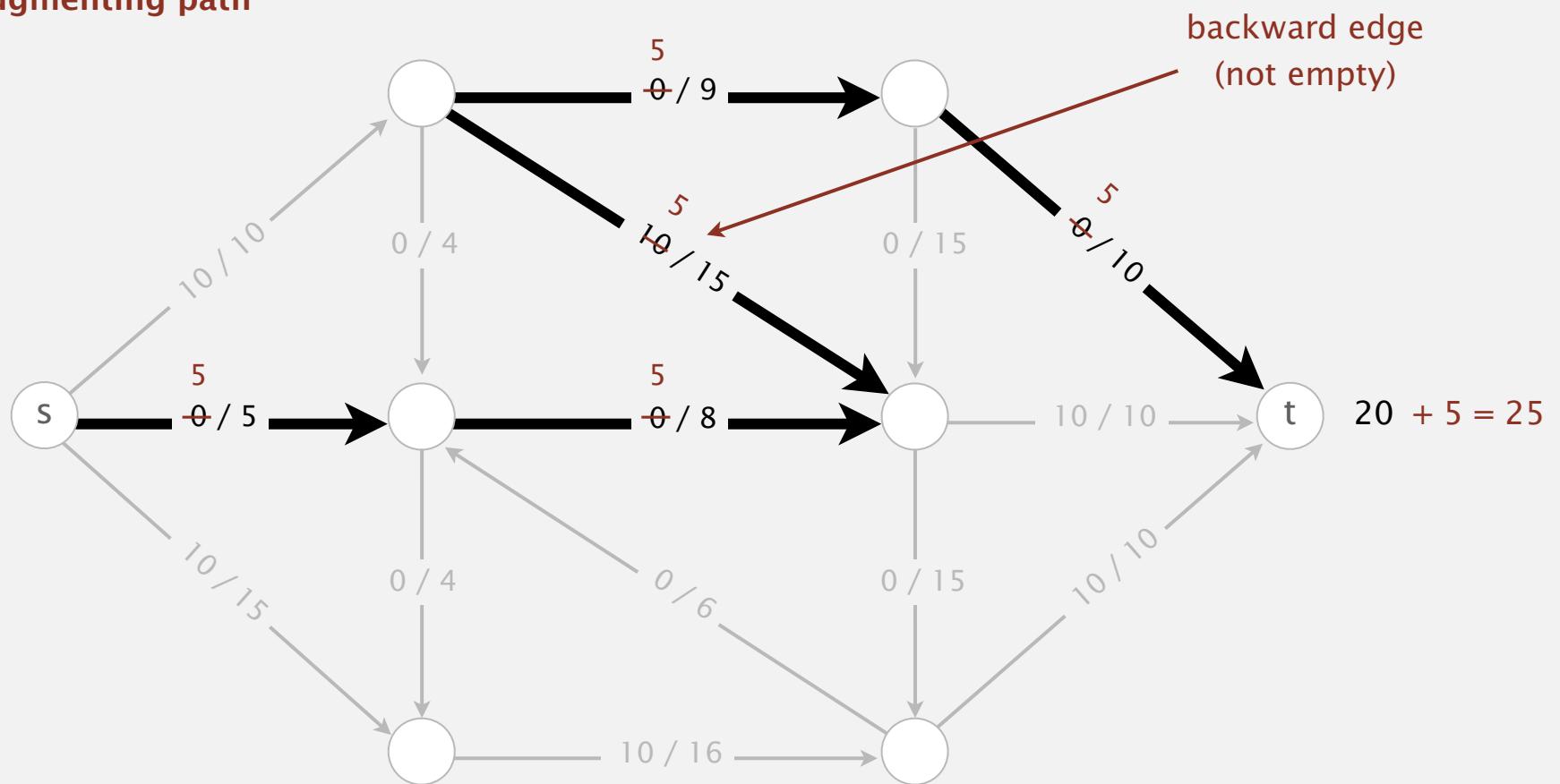


Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

3rd augmenting path

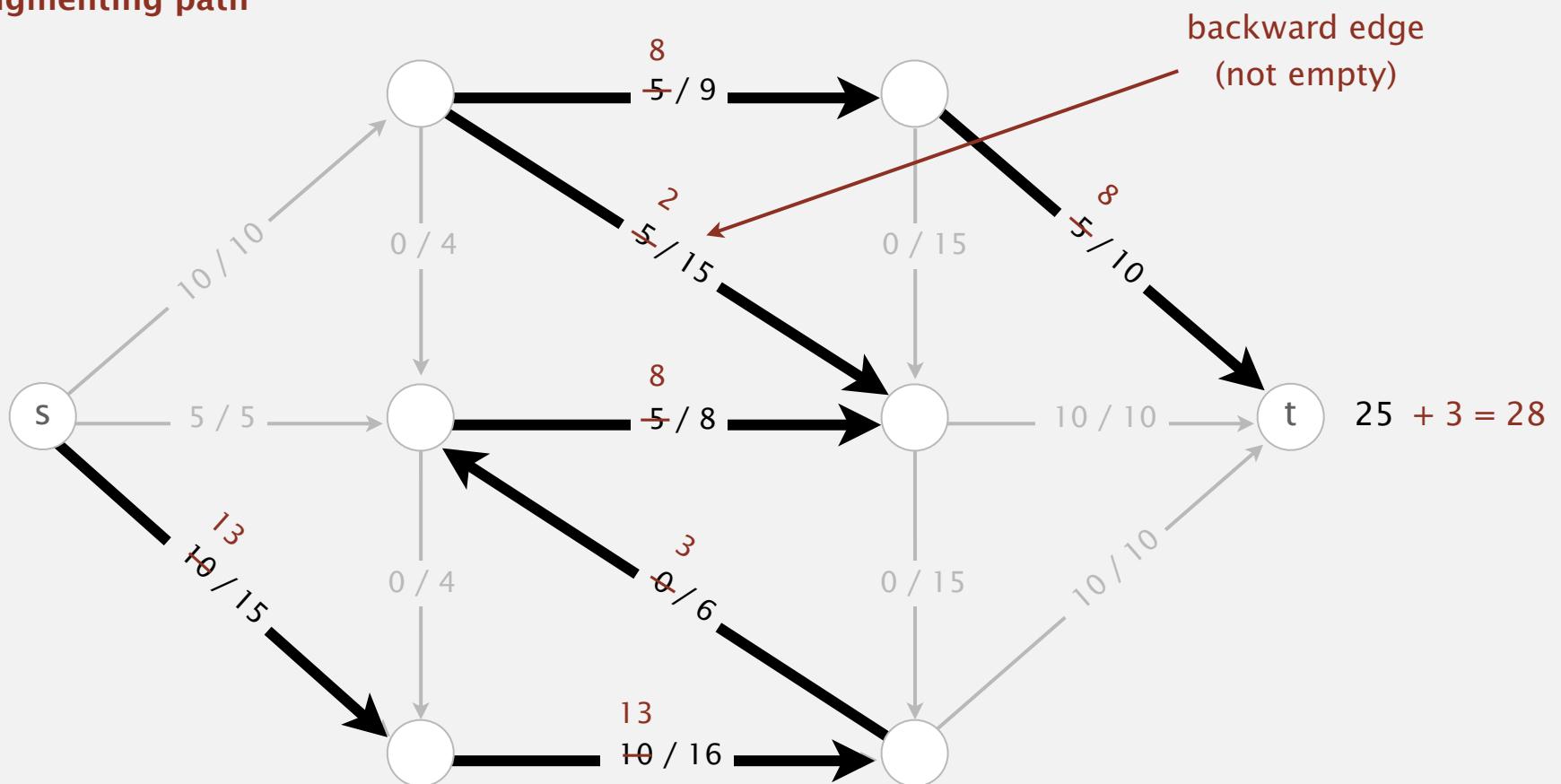


Idea: increase flow along augmenting paths

Augmenting path. Find an undirected path from s to t such that:

- Can increase flow on forward edges (not full).
- Can decrease flow on backward edge (not empty).

4th augmenting path

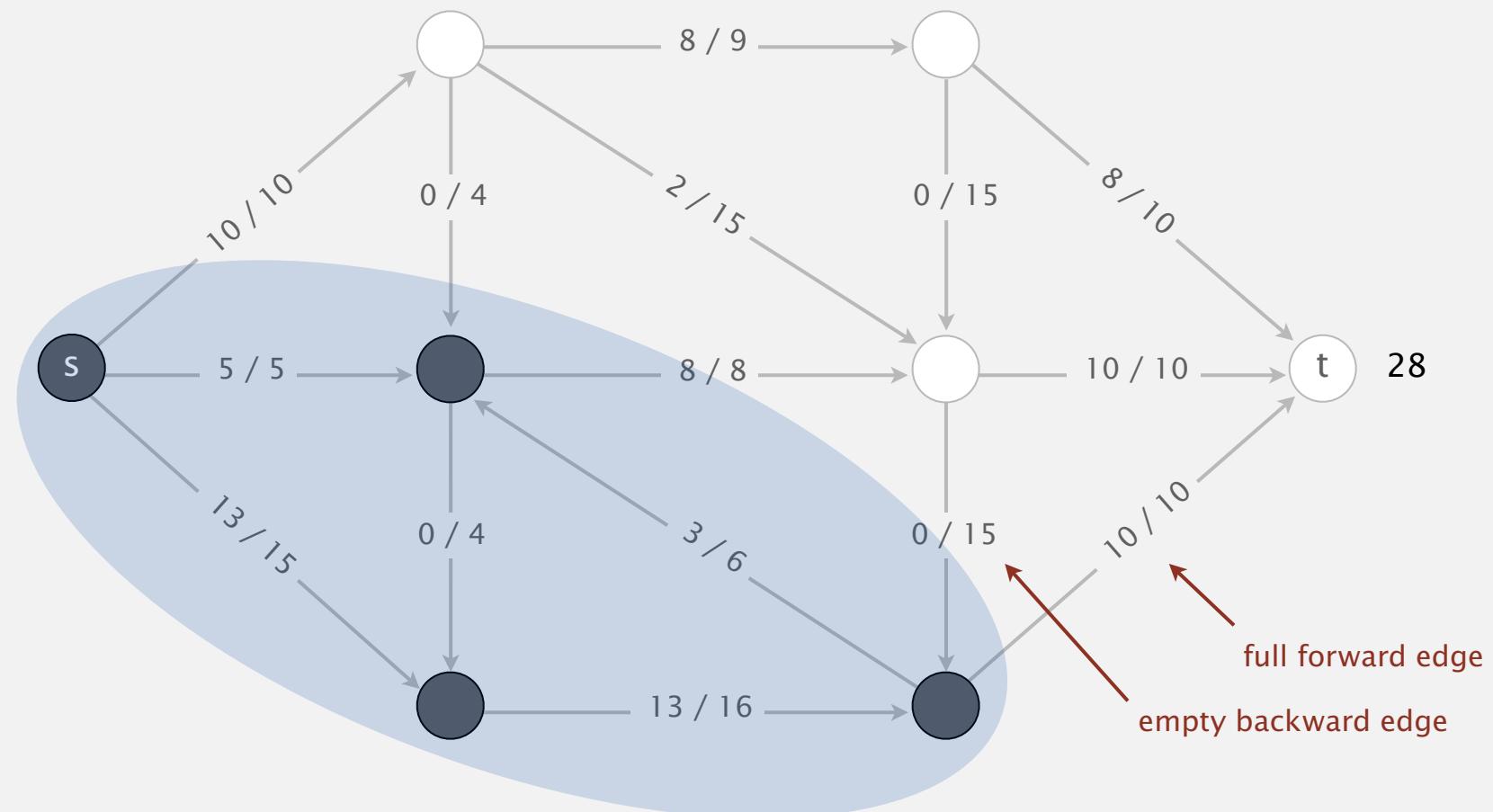


Idea: increase flow along augmenting paths

Termination. All paths from s to t are blocked by either a

- Full forward edge.
- Empty backward edge.

no more augmenting paths



Ford-Fulkerson algorithm

Ford-Fulkerson algorithm

Start with 0 flow.

While there exists an augmenting path:

- **find an augmenting path**
 - **compute bottleneck capacity**
 - **increase flow on that path by bottleneck capacity**
-

Questions.

- How to compute a mincut?
- How to find an augmenting path?
- If FF terminates, does it always compute a maxflow?
- Does FF always terminate? If so, after how many augmentations?

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ ***Ford-Fulkerson algorithm***
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

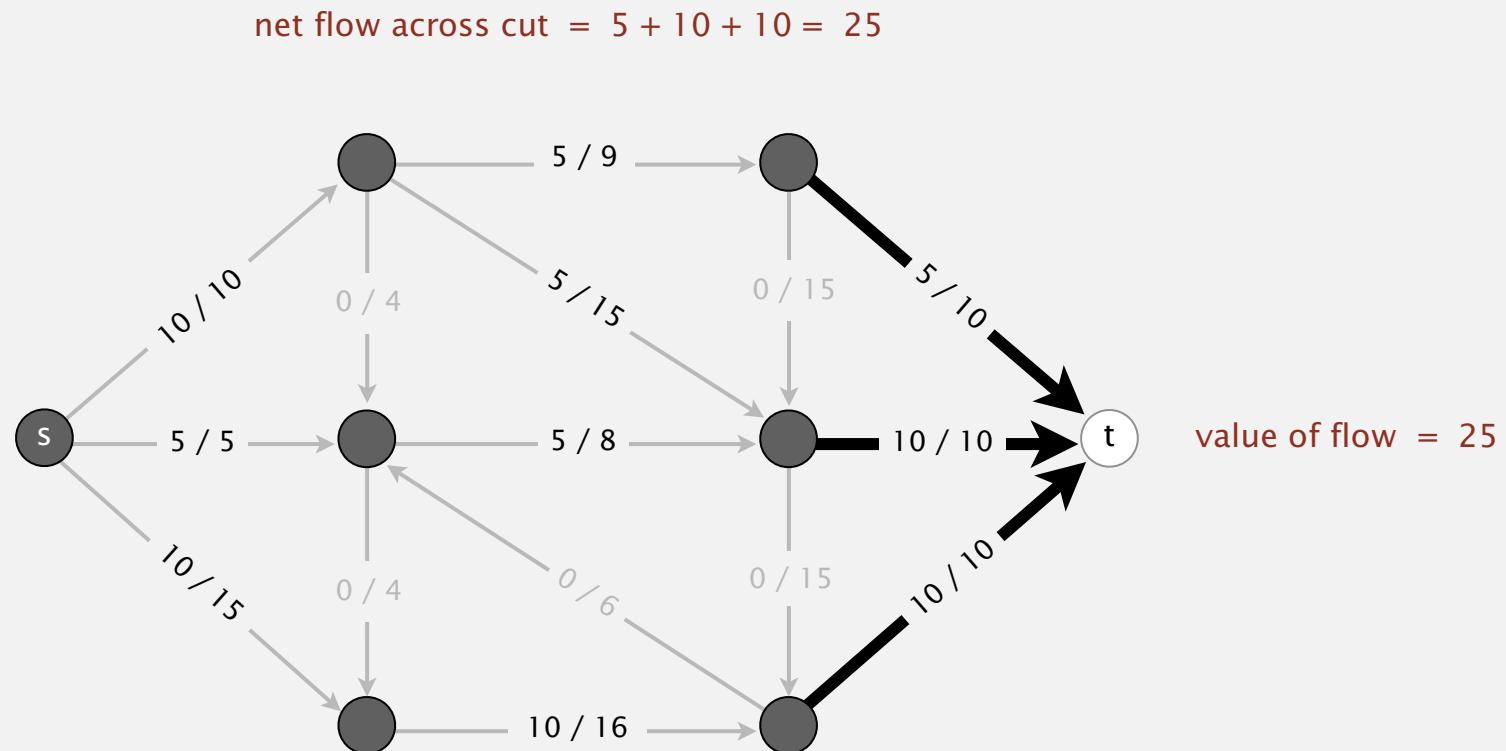
6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Relationship between flows and cuts

Def. The **net flow across** a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A .

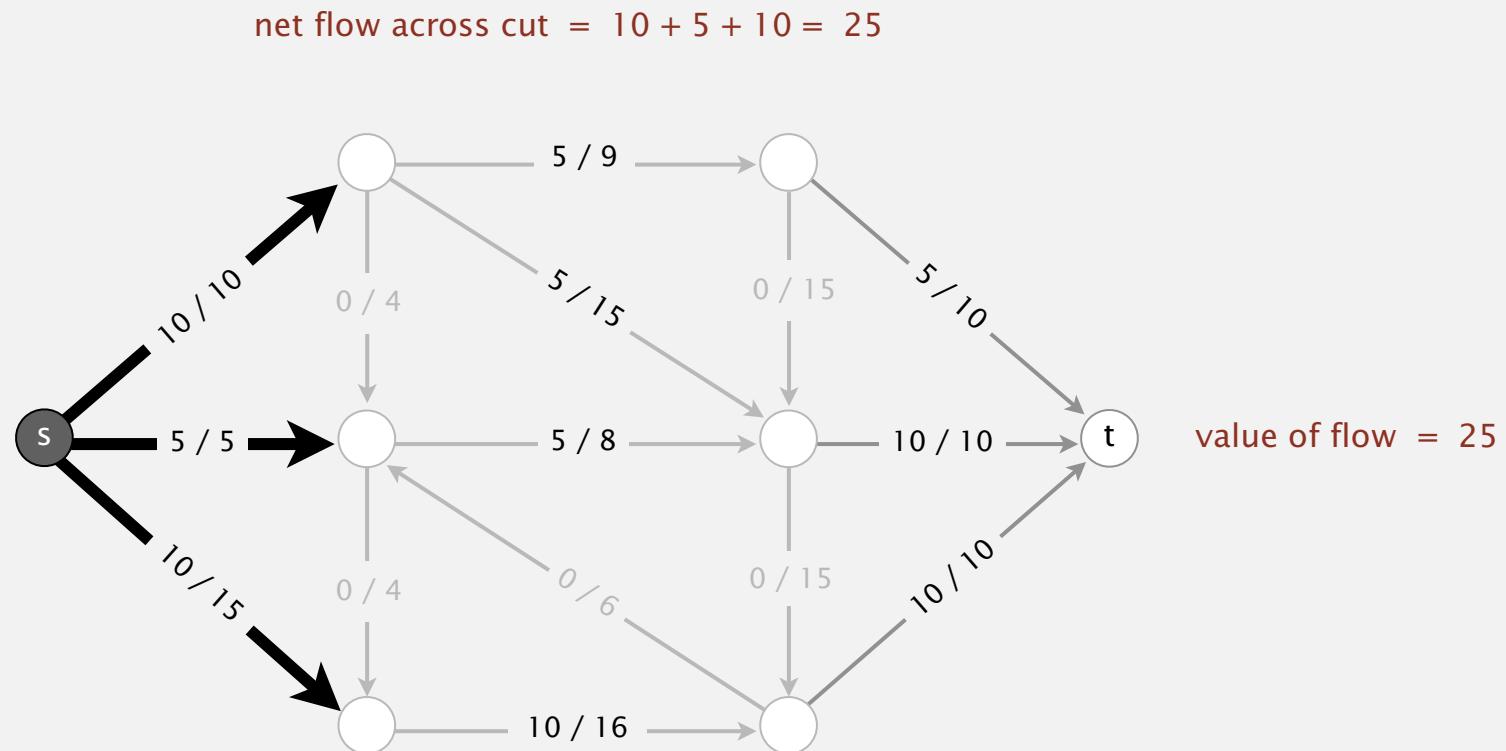
Flow-value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f .



Relationship between flows and cuts

Def. The **net flow across** a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A .

Flow-value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f .

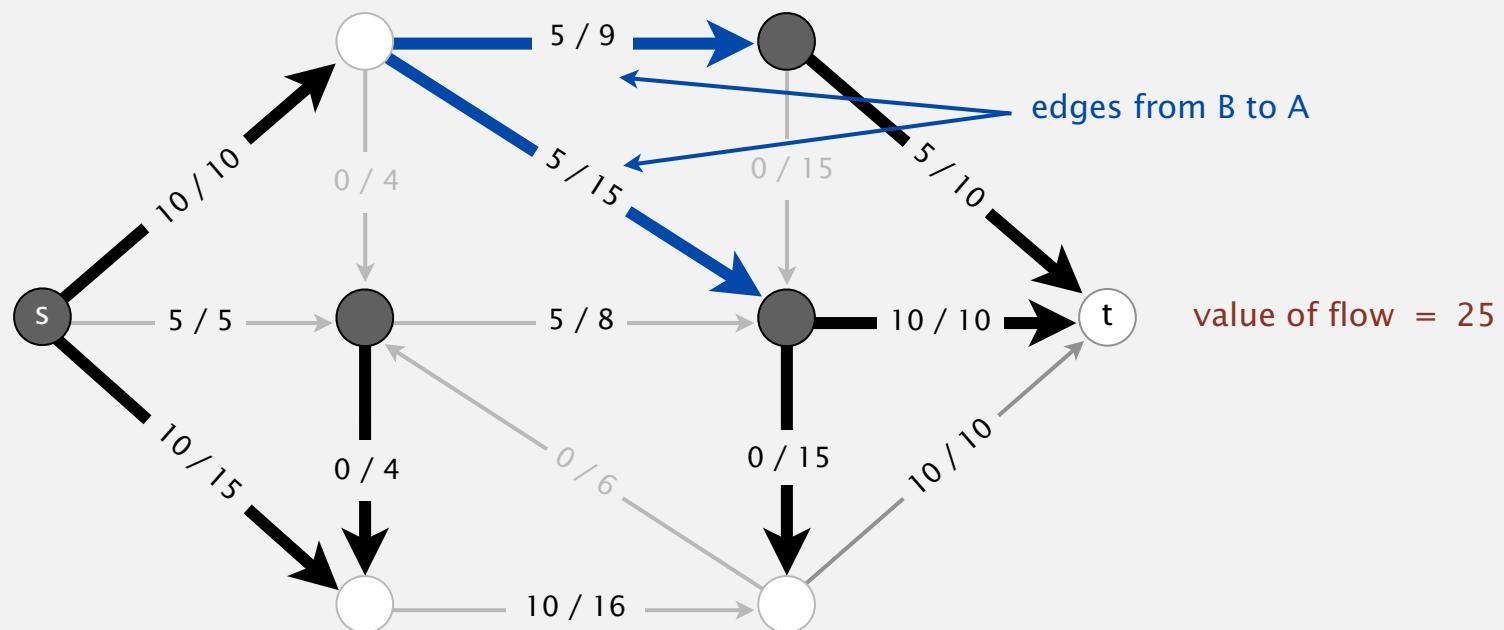


Relationship between flows and cuts

Def. The **net flow across** a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A .

Flow-value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f .

$$\text{net flow across cut} = (10 + 10 + 5 + 10 + 0 + 0) - (5 + 5) = 25$$



Relationship between flows and cuts

Def. The **net flow across** a cut (A, B) is the sum of the flows on its edges from A to B minus the sum of the flows on its edges from B to A .

Flow-value lemma. Let f be any flow and let (A, B) be any cut. Then, the net flow across (A, B) equals the value of f .

Pf. By induction on the size of B .

- Base case: $B = \{t\}$.
- Induction step: remains true by local equilibrium when moving any vertex from A to B .

Corollary. Outflow from s = inflow to t = value of flow.

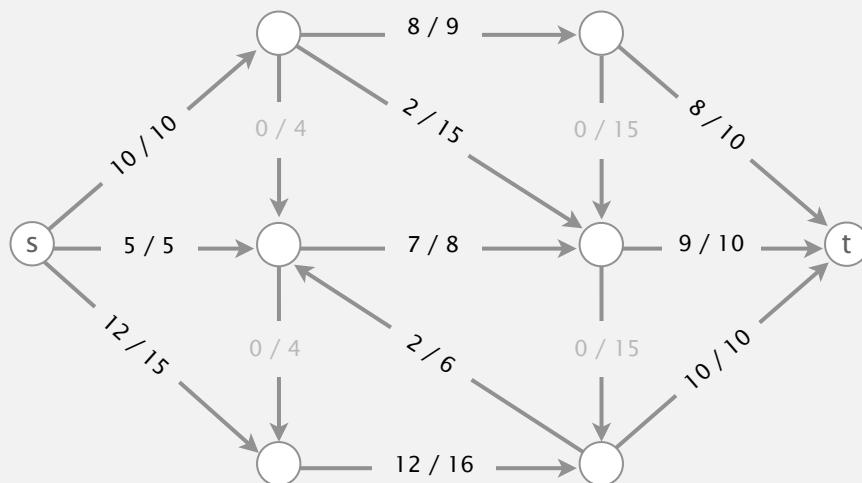
Relationship between flows and cuts

Weak duality. Let f be any flow and let (A, B) be any cut.
Then, the value of the flow \leq the capacity of the cut.

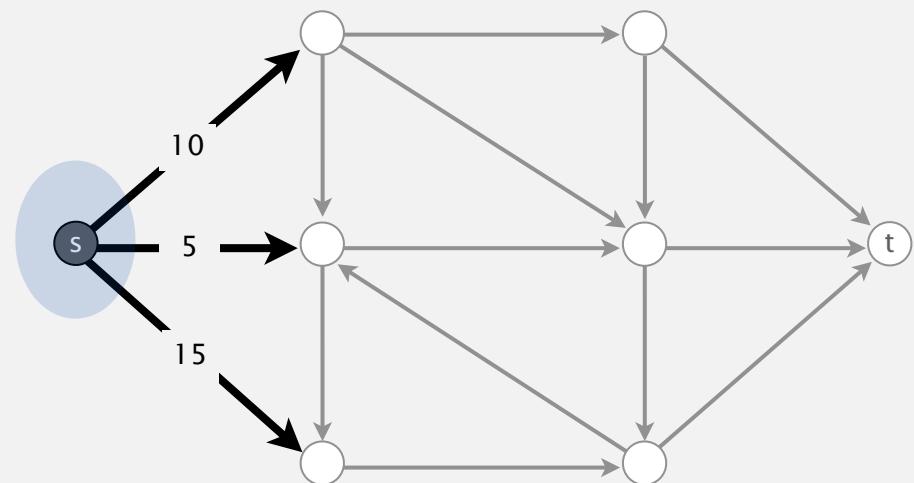
Pf. Value of flow f = net flow across cut $(A, B) \leq$ capacity of cut (A, B) .

↑
flow-value lemma

↑
flow bounded by capacity



value of flow = 27



capacity of cut = 30

Maxflow-mincut theorem

Augmenting path theorem. A flow f is a maxflow iff no augmenting paths.

Maxflow-mincut theorem. Value of the maxflow = capacity of mincut.

Pf. The following three conditions are equivalent for any flow f :

- i. There exists a cut whose capacity equals the value of the flow f .
- ii. f is a maxflow.
- iii. There is no augmenting path with respect to f .

[i \Rightarrow ii]

- Suppose that (A, B) is a cut with capacity equal to the value of f .
- Then, the value of any flow $f' \leq$ capacity of $(A, B) =$ value of f .
- Thus, f is a maxflow.

↑
weak duality

↑
by assumption

Maxflow-mincut theorem

Augmenting path theorem. A flow f is a maxflow iff no augmenting paths.

Maxflow-mincut theorem. Value of the maxflow = capacity of mincut.

Pf. The following three conditions are equivalent for any flow f :

- i. There exists a cut whose capacity equals the value of the flow f .
- ii. f is a maxflow.
- iii. There is no augmenting path with respect to f .

[ii \Rightarrow iii] We prove contrapositive: \sim iii \Rightarrow \sim ii.

- Suppose that there is an augmenting path with respect to f .
- Can improve flow f by sending flow along this path.
- Thus, f is not a maxflow.

Maxflow-mincut theorem

Augmenting path theorem. A flow f is a maxflow iff no augmenting paths.

Maxflow-mincut theorem. Value of the maxflow = capacity of mincut.

Pf. The following three conditions are equivalent for any flow f :

- i. There exists a cut whose capacity equals the value of the flow f .
- ii. f is a maxflow.
- iii. There is no augmenting path with respect to f .

[iii \Rightarrow i]

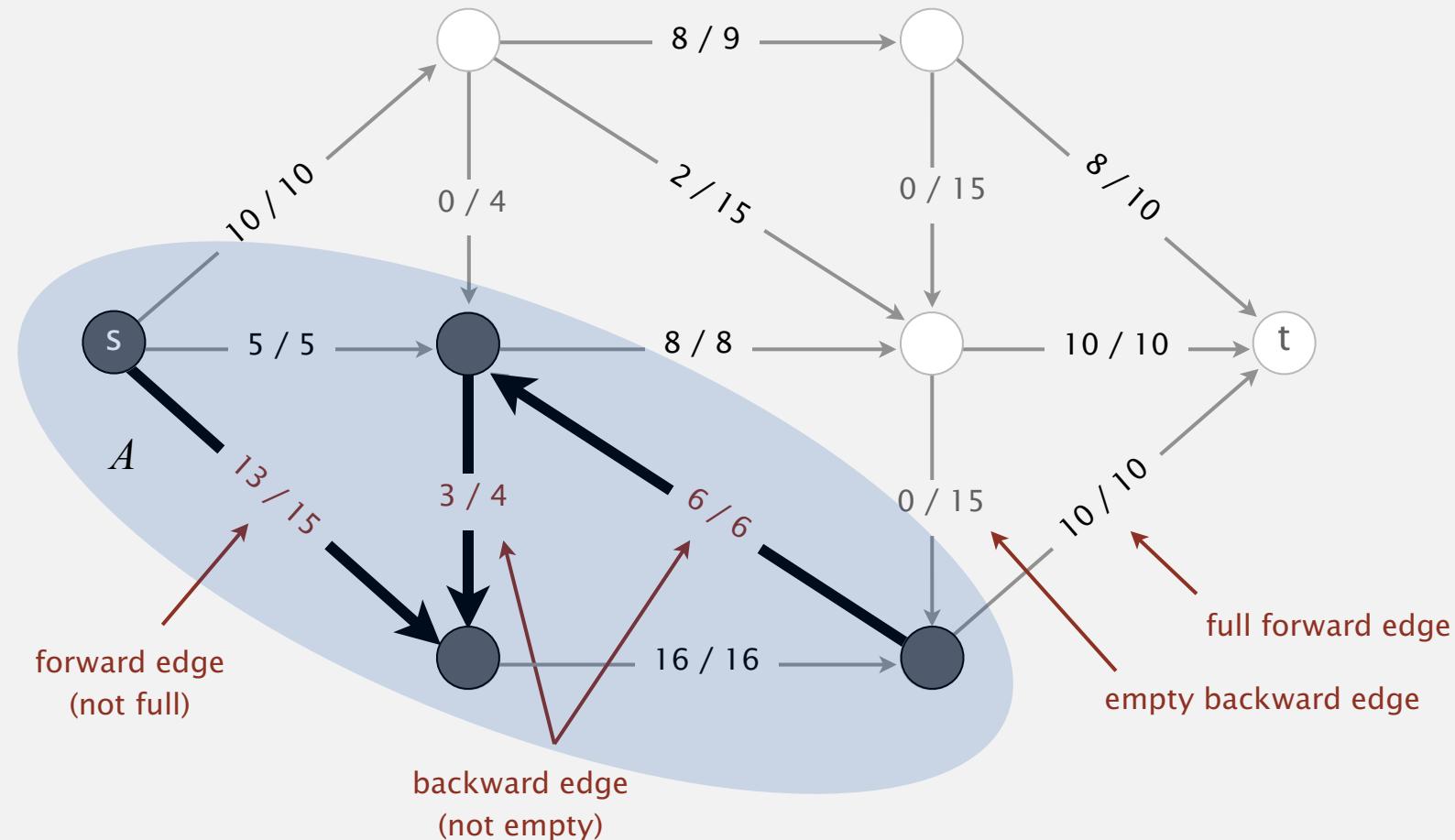
Suppose that there is no augmenting path with respect to f .

- Let (A, B) be a cut where A is the set of vertices connected to s by an undirected path with no full forward or empty backward edges.
- By definition, s is in A ; since no augmenting path, t is in B .
- Capacity of cut = net flow across cut \leftarrow forward edges full; backward edges empty
= value of flow f . \leftarrow flow-value lemma

Computing a mincut from a maxflow

To compute mincut (A, B) from maxflow f :

- By augmenting path theorem, no augmenting paths with respect to f .
- Compute $A = \text{set of vertices connected to } s \text{ by an undirected path}$ with no full forward or empty backward edges.



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Ford-Fulkerson algorithm

Ford-Fulkerson algorithm

Start with 0 flow.

While there exists an augmenting path:

- find an augmenting path
- compute bottleneck capacity
- increase flow on that path by bottleneck capacity

Questions.

- How to compute a mincut? Easy. ✓
- How to find an augmenting path? BFS works well.
- If FF terminates, does it always compute a maxflow? Yes. ✓
- Does FF always terminate? If so, after how many augmentations?

yes, provided edge capacities are integers
(or augmenting paths are chosen carefully)

requires clever analysis

Ford-Fulkerson algorithm with integer capacities

Important special case. Edge capacities are integers between 1 and U .

Invariant. The flow is integer-valued throughout Ford-Fulkerson.

Pf. [by induction]

- Bottleneck capacity is an integer.
- Flow on an edge increases/decreases by bottleneck capacity.

Proposition. Number of augmentations \leq the value of the maxflow.

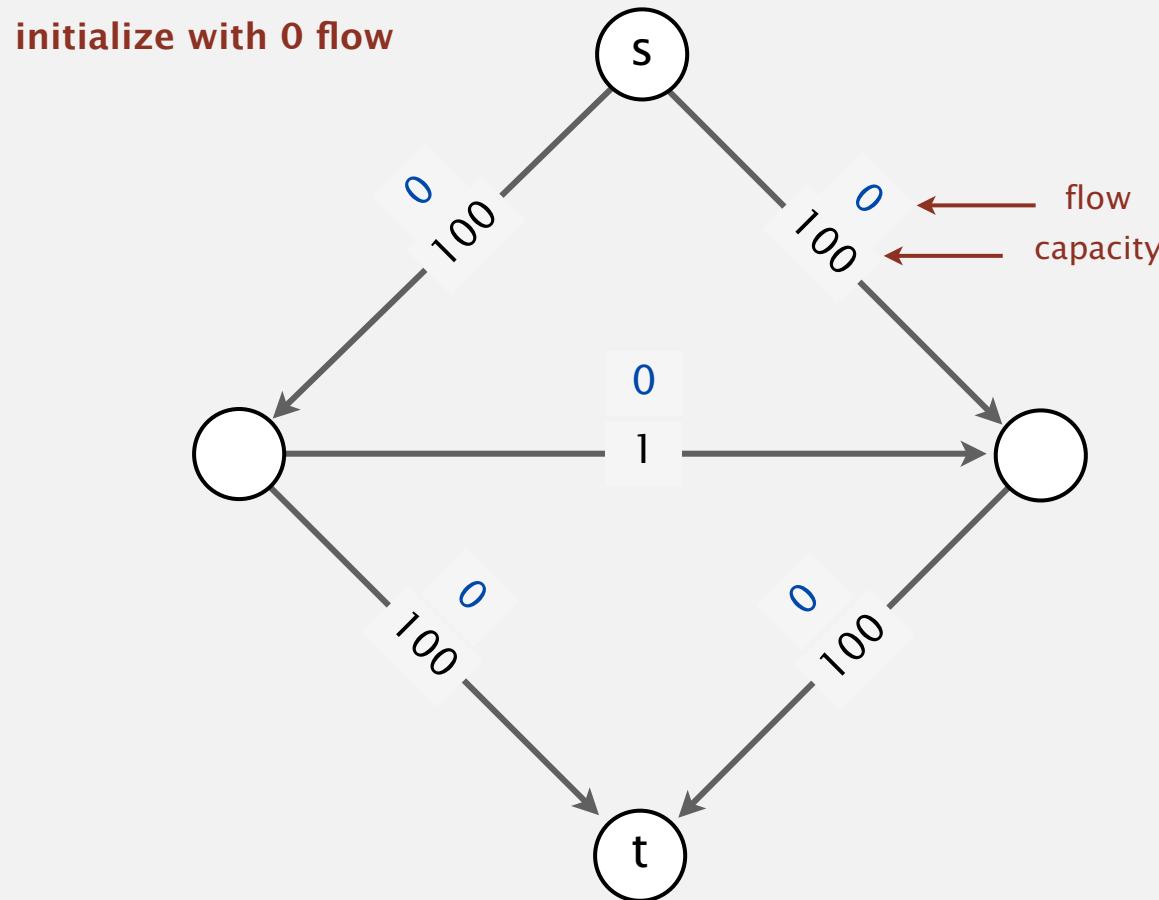
Pf. Each augmentation increases the value by at least 1.

Integrality theorem. There exists an integer-valued maxflow.

Pf. Ford-Fulkerson terminates and maxflow that it finds is integer-valued.

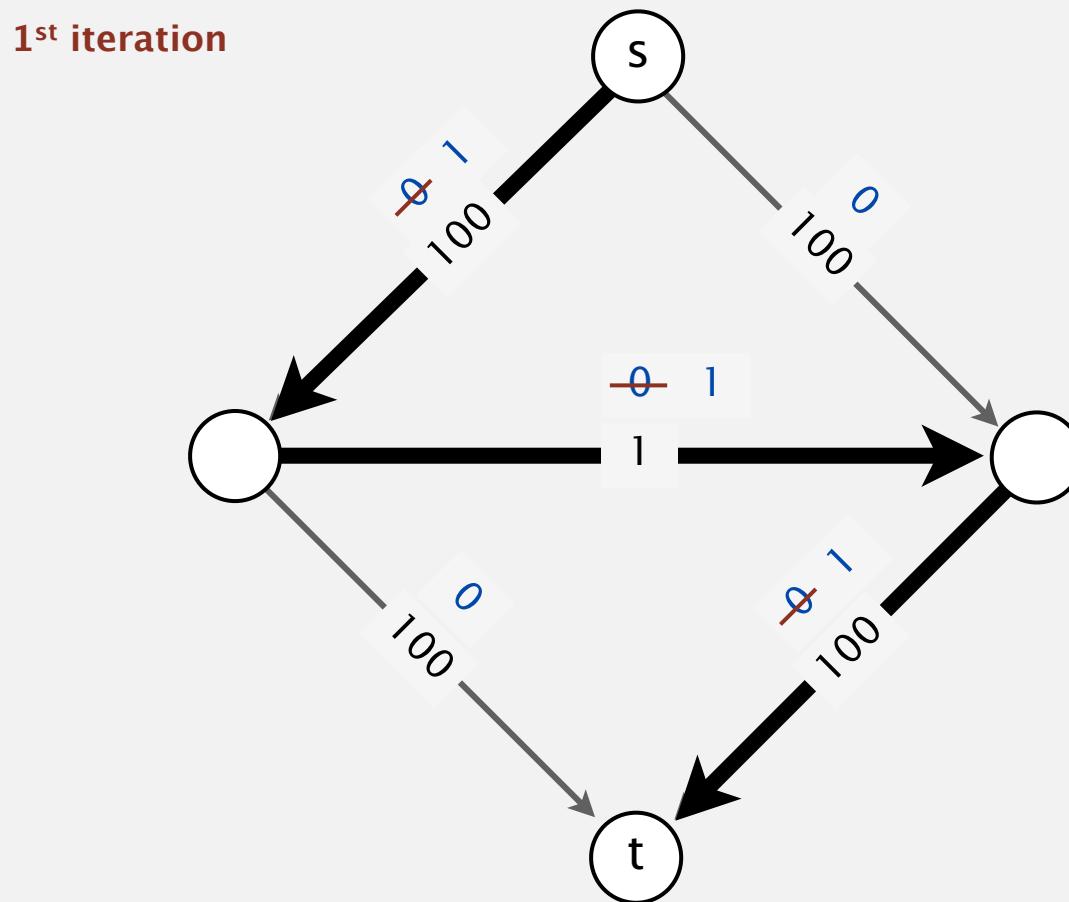
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



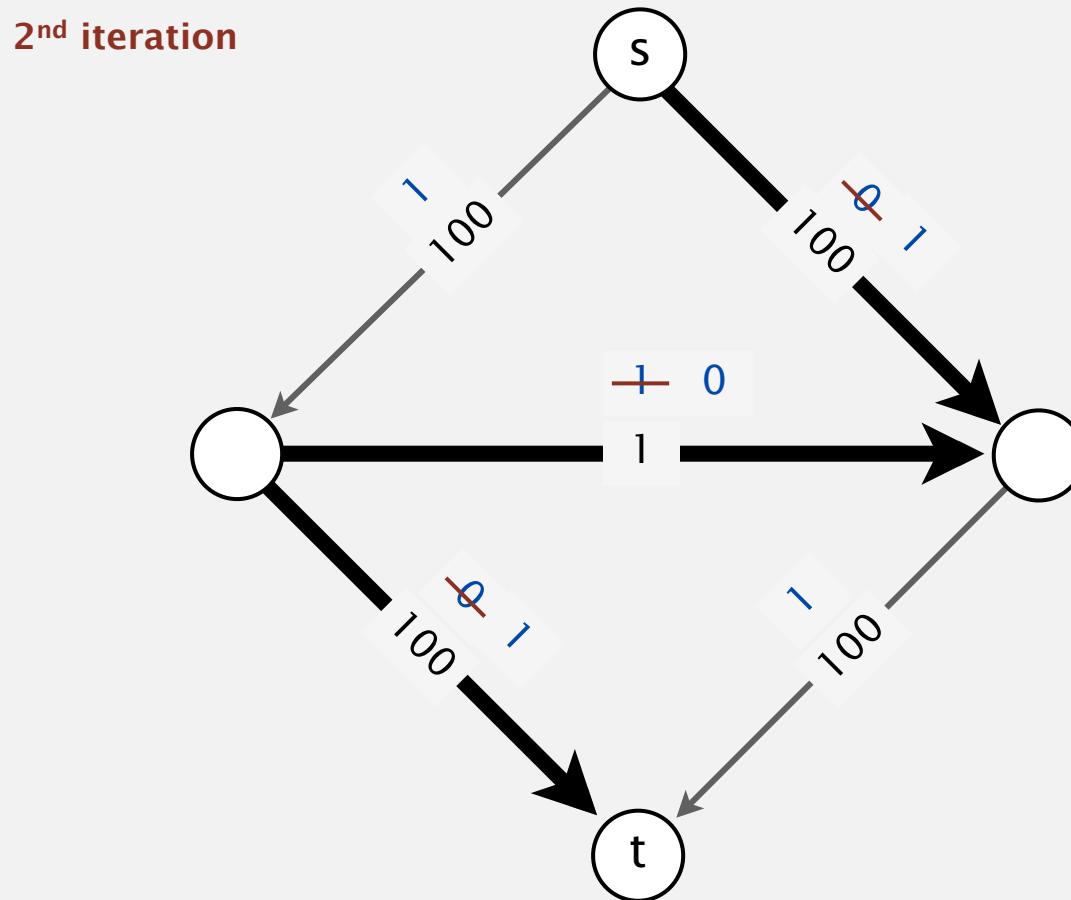
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



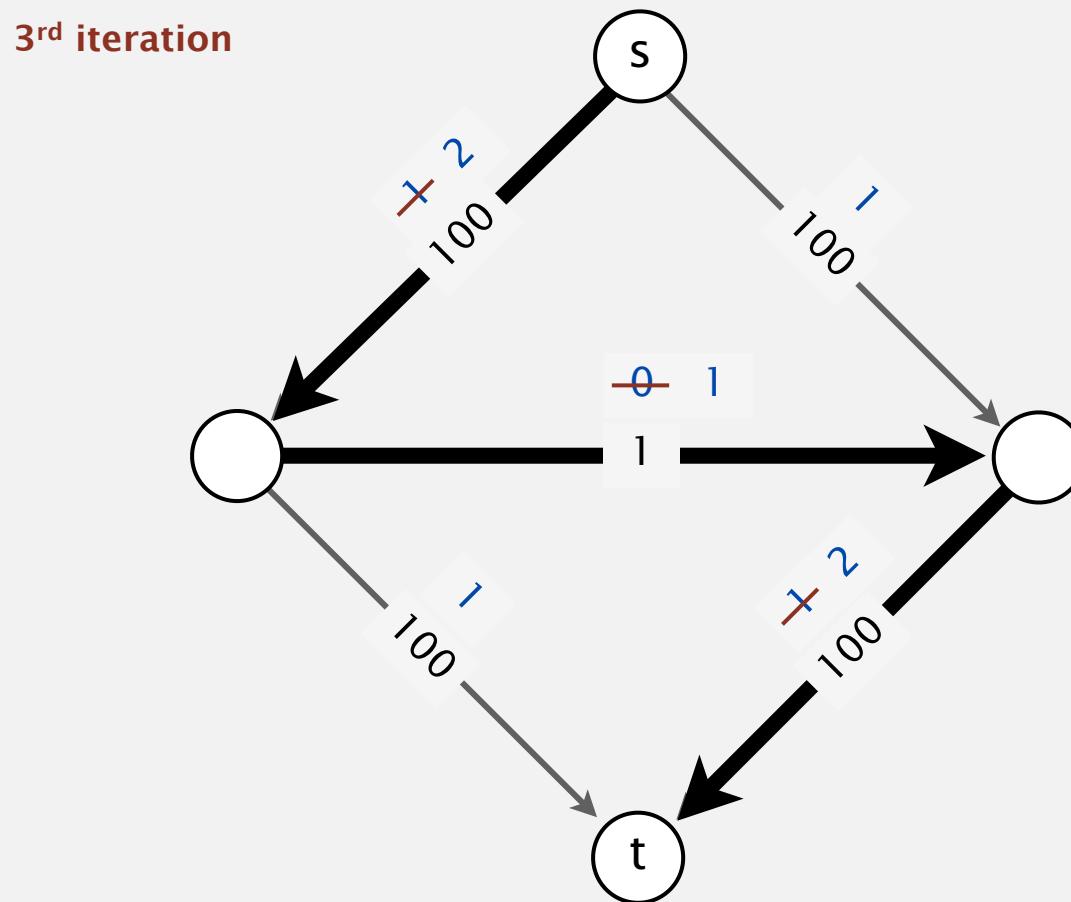
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



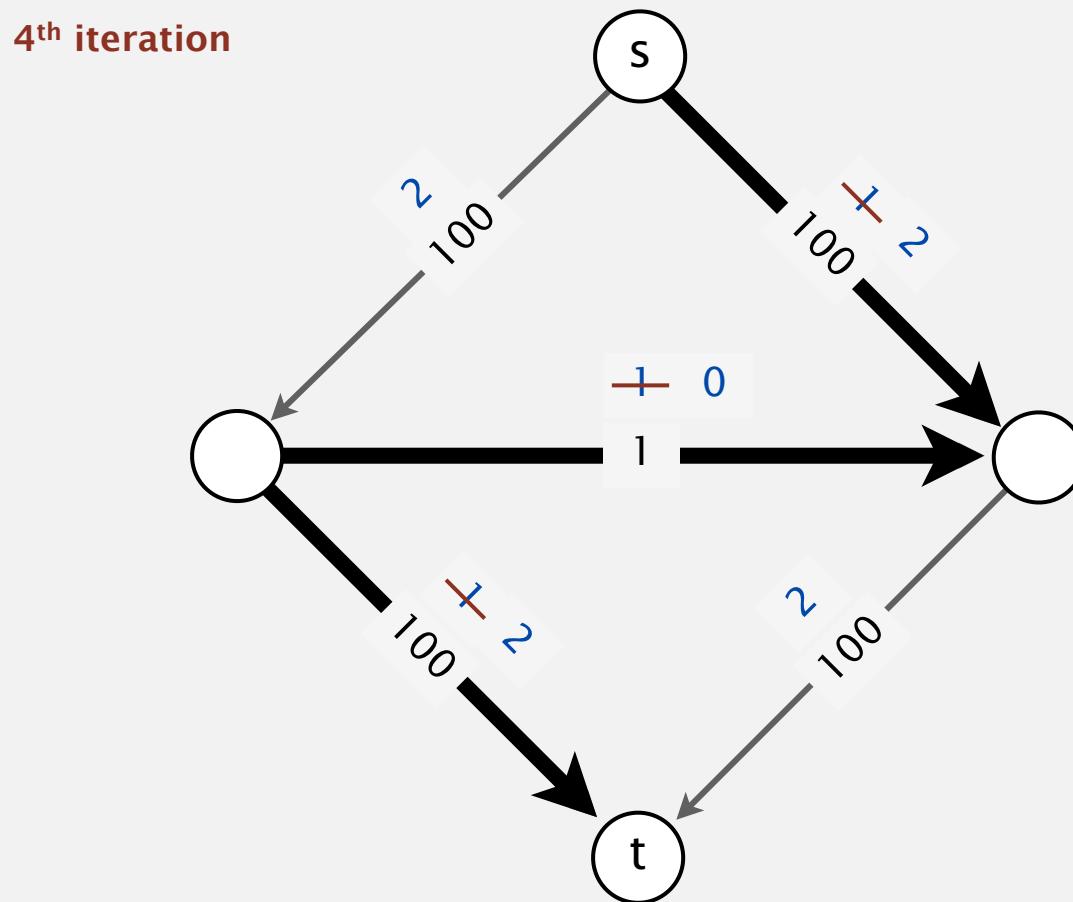
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



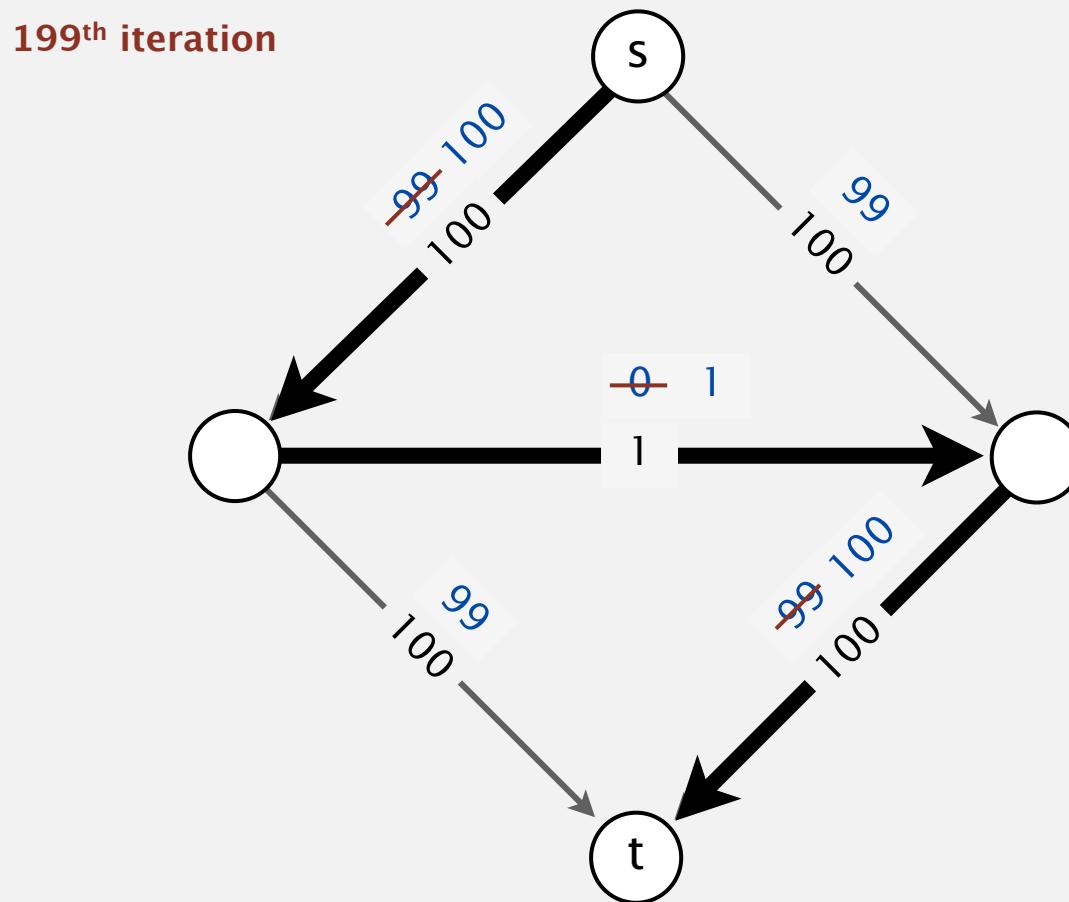
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



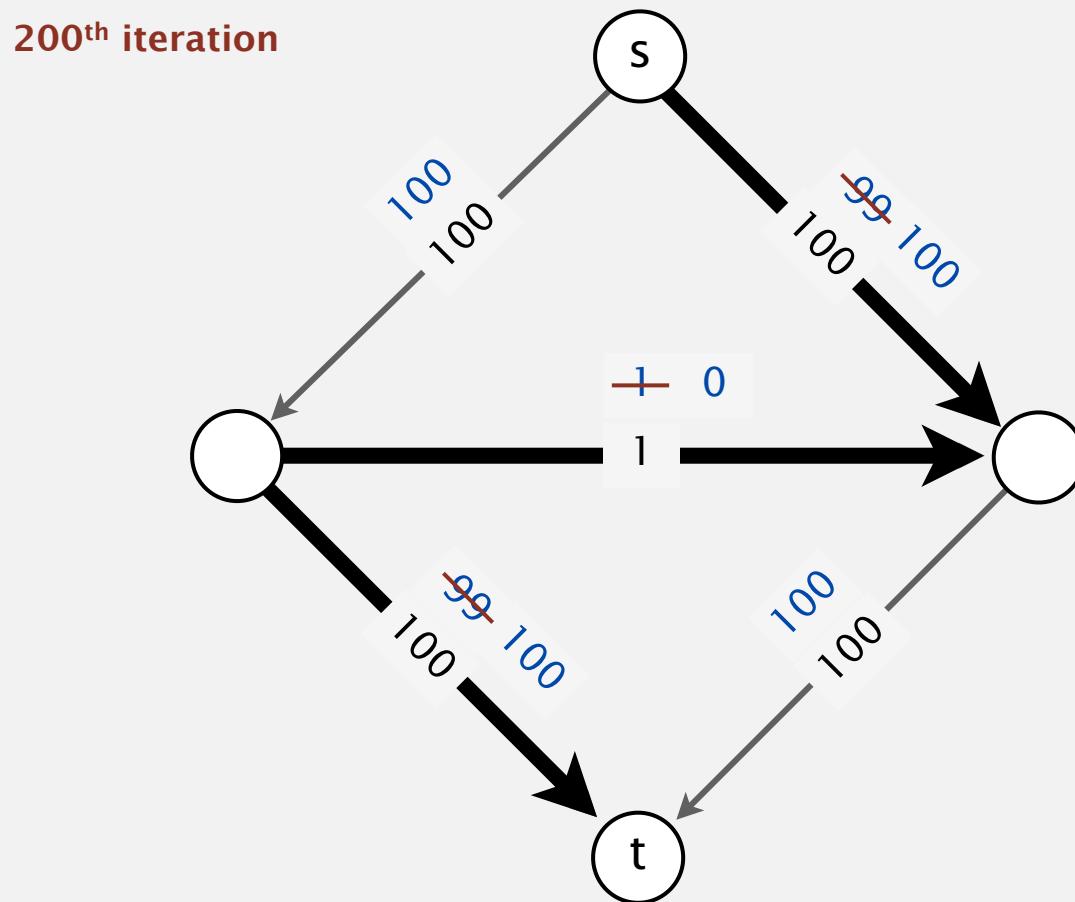
Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.



Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

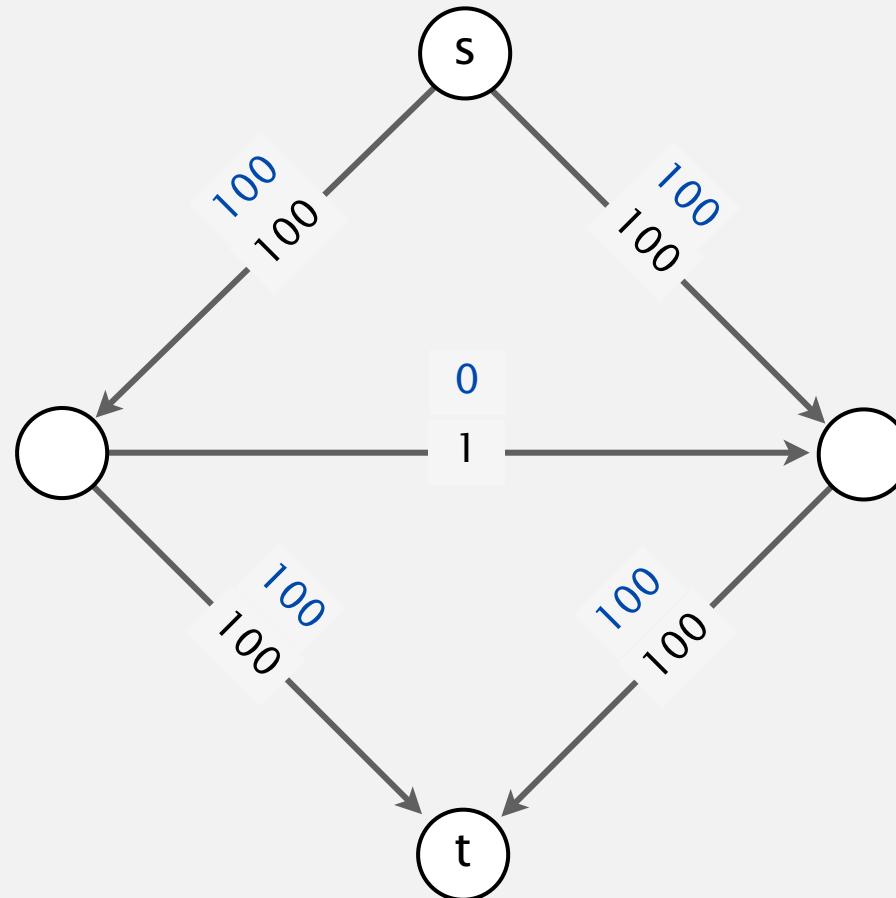


Bad case for Ford-Fulkerson

Bad news. Even when edge capacities are integers, number of augmenting paths could be equal to the value of the maxflow.

can be exponential in input size

Good news. This case is easily avoided. [use shortest/fattest path]

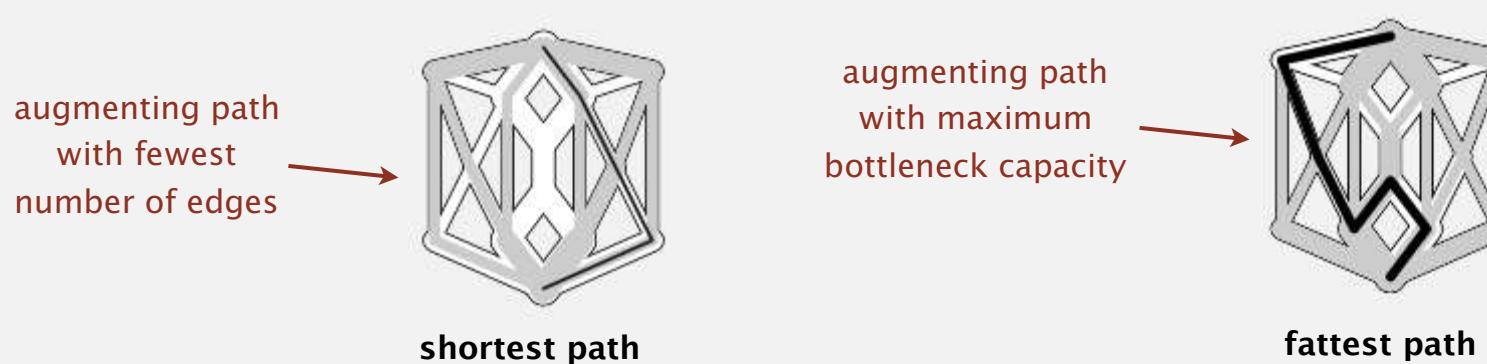


How to choose augmenting paths?

FF performance depends on choice of augmenting paths.

augmenting path	number of paths	implementation
shortest path	$\leq \frac{1}{2} E V$	queue (BFS)
fattest path	$\leq E \ln(E U)$	priority queue
random path	$\leq E U$	randomized queue
DFS path	$\leq E U$	stack (DFS)

digraph with V vertices, E edges, and integer capacities between 1 and U



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

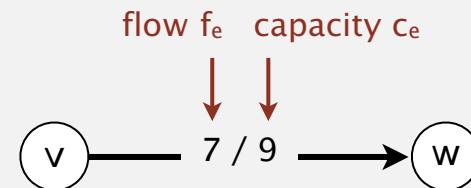
<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ ***Java implementation***
- ▶ *applications*

Flow network representation

Flow edge data type. Associate flow f_e and capacity c_e with edge $e = v \rightarrow w$.



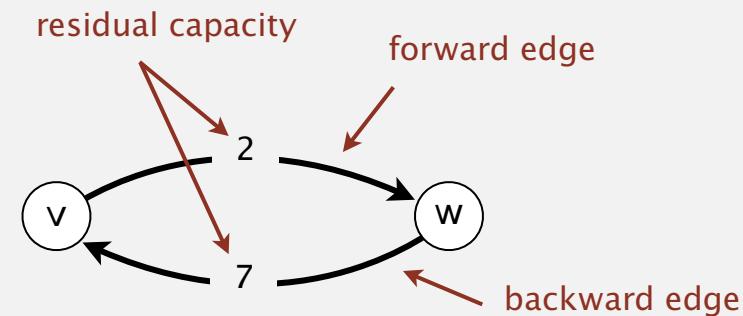
Flow network data type. Need to process edge $e = v \rightarrow w$ in either direction:
Include e in both v and w 's adjacency lists.

Residual capacity.

- Forward edge: residual capacity $= c_e - f_e$.
- Backward edge: residual capacity $= f_e$.

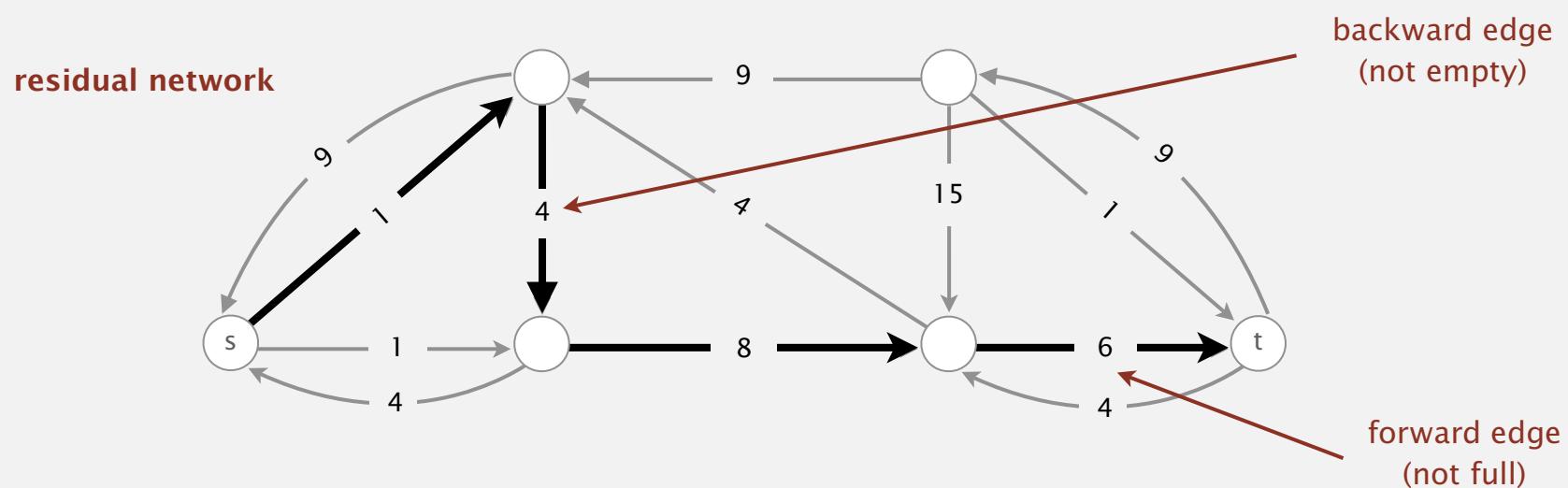
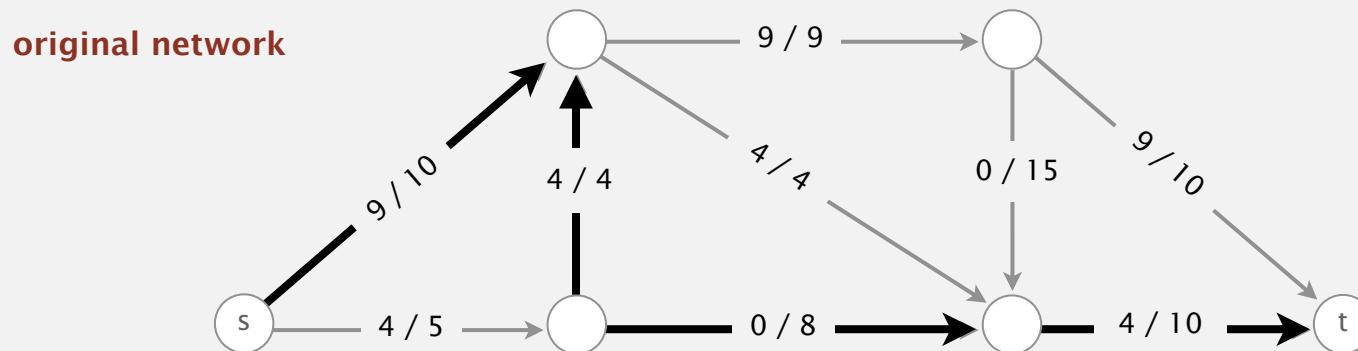
Augment flow.

- Forward edge: add Δ .
- Backward edge: subtract Δ .



Flow network representation

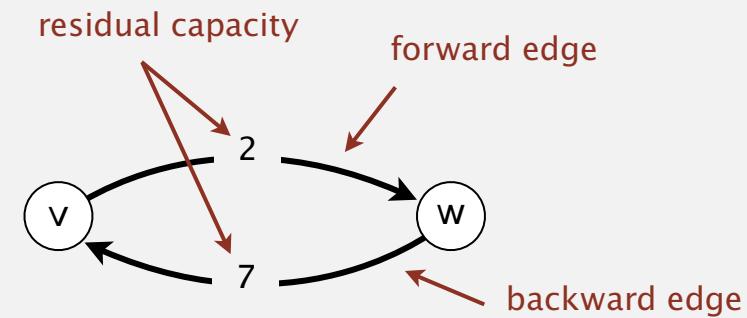
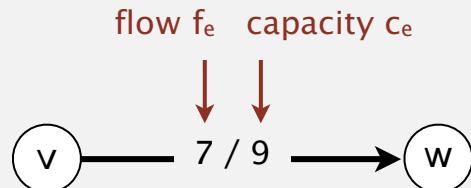
Residual network. A useful view of a flow network.



Key point. Augmenting path in original network is equivalent to directed path in residual network.

Flow edge API

public class FlowEdge	
FlowEdge(int v, int w, double capacity)	<i>create a flow edge v→w</i>
int from()	<i>vertex this edge points from</i>
int to()	<i>vertex this edge points to</i>
int other(int v)	<i>other endpoint</i>
double capacity()	<i>capacity of this edge</i>
double flow()	<i>flow in this edge</i>
double residualCapacityTo(int v)	<i>residual capacity toward v</i>
void addResidualFlowTo(int v, double delta)	<i>add delta flow toward v</i>
String toString()	<i>string representation</i>



Flow edge: Java implementation

```
public class FlowEdge
{
    private final int v, w;          // from and to
    private final double capacity;   // capacity
    private double flow;            // flow
```

← flow variable
(mutable)

```
public FlowEdge(int v, int w, double capacity)
{
    this.v      = v;
    this.w      = w;
    this.capacity = capacity;
}

public int from()      { return v; }
public int to()        { return w; }
public double capacity() { return capacity; }
public double flow()    { return flow; }

public int other(int vertex)
{
    if      (vertex == v) return w;
    else if (vertex == w) return v;
    else throw new RuntimeException("Illegal endpoint");
}

public double residualCapacityTo(int vertex)      {...}
public void addResidualFlowTo(int vertex, double delta) {...}
```

← next slide

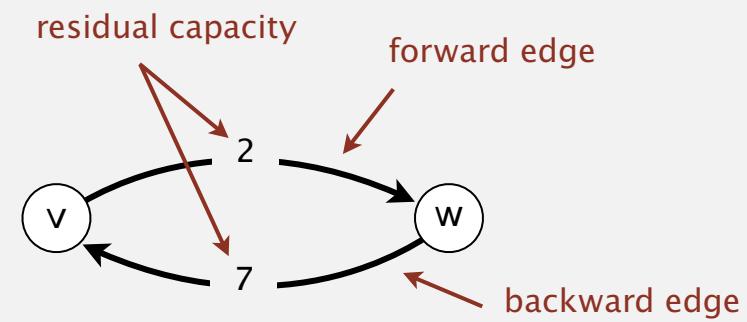
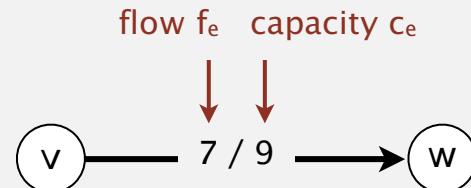
Flow edge: Java implementation

```
public double residualCapacityTo(int vertex)
{
    if (vertex == v) return flow;
    else if (vertex == w) return capacity - flow;
    else throw new IllegalArgumentException();
}
```

← forward edge
← backward edge

```
public void addResidualFlowTo(int vertex, double delta)
{
    if (vertex == v) flow -= delta;
    else if (vertex == w) flow += delta;
    else throw new IllegalArgumentException();
}
```

← forward edge
← backward edge



Flow network API

public class FlowNetwork	
FlowNetwork(int V)	<i>create an empty flow network with V vertices</i>
FlowNetwork(In in)	<i>construct flow network input stream</i>
void addEdge(FlowEdge e)	<i>add flow edge e to this flow network</i>
Iterable<FlowEdge> adj(int v)	<i>forward and backward edges incident to v</i>
Iterable<FlowEdge> edges()	<i>all edges in this flow network</i>
int V()	<i>number of vertices</i>
int E()	<i>number of edges</i>
String toString()	<i>string representation</i>

Conventions. Allow self-loops and parallel edges.

Flow network: Java implementation

```
public class FlowNetwork
{
    private final int V;
    private Bag<FlowEdge>[] adj;

    public FlowNetwork(int V)
    {
        this.V = V;
        adj = (Bag<FlowEdge>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<FlowEdge>();
    }

    public void addEdge(FlowEdge e)
    {
        int v = e.from();
        int w = e.to();
        adj[v].add(e);
        adj[w].add(e);
    }

    public Iterable<FlowEdge> adj(int v)
    { return adj[v]; }
}
```

same as EdgeWeightedGraph,
but adjacency lists of
FlowEdges instead of Edges

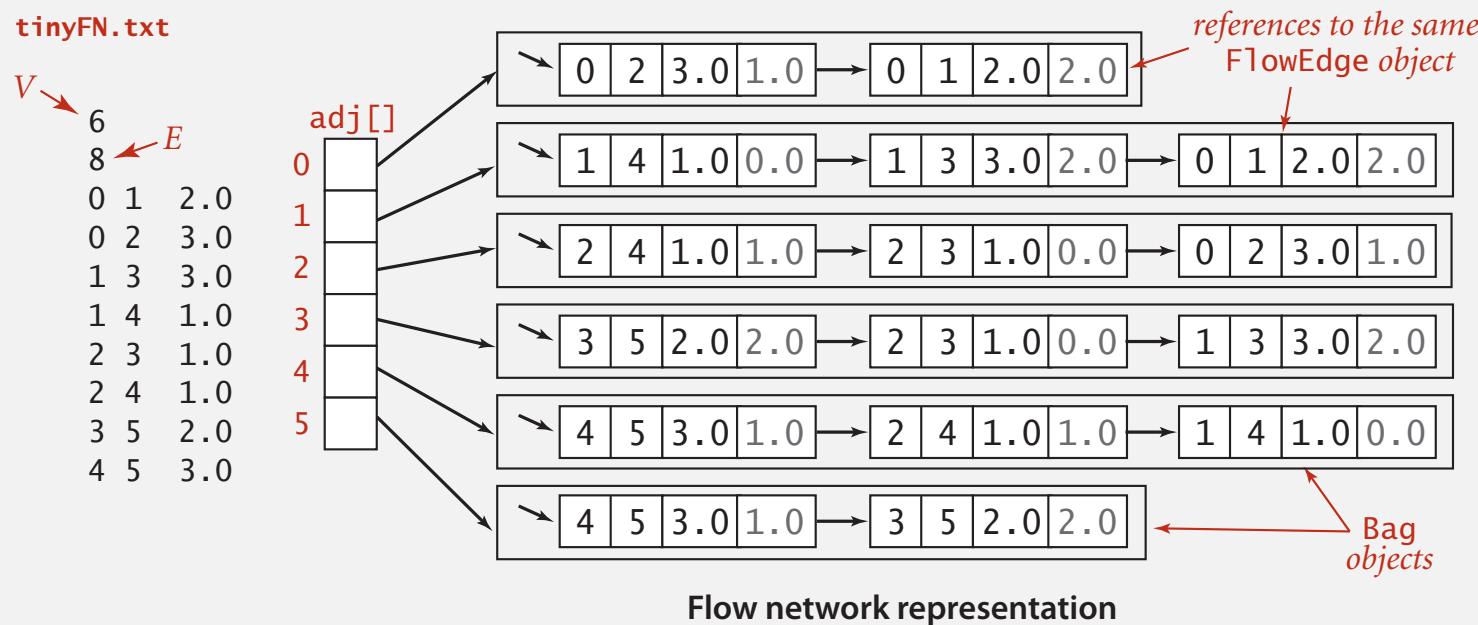


add forward edge
add backward edge



Flow network: adjacency-lists representation

Maintain vertex-indexed array of FlowEdge lists (use Bag abstraction).



Ford-Fulkerson: Java implementation

```
public class FordFulkerson
{
    private boolean[] marked;      // true if s->v path in residual network
    private FlowEdge[] edgeTo;     // last edge on s->v path
    private double value;         // value of flow

    public FordFulkerson(FlowNetwork G, int s, int t)
    {
        value = 0.0;
        while (hasAugmentingPath(G, s, t))
        {
            double bottle = Double.POSITIVE_INFINITY;
            for (int v = t; v != s; v = edgeTo[v].other(v))
                bottle = Math.min(bottle, edgeTo[v].residualCapacityTo(v)); ← compute bottleneck capacity

            for (int v = t; v != s; v = edgeTo[v].other(v))
                edgeTo[v].addResidualFlowTo(v, bottle); ← augment flow

            value += bottle;
        }
    }

    public double hasAugmentingPath(FlowNetwork G, int s, int t)
    { /* See next slide. */ }

    public double value()
    { return value; }

    public boolean inCut(int v) ← is v reachable from s in residual network?
    { return marked[v]; }
}
```

Finding a shortest augmenting path (cf. breadth-first search)

```
private boolean hasAugmentingPath(FlowNetwork G, int s, int t)
{
    edgeTo = new FlowEdge[G.V()];
    marked = new boolean[G.V()];

    Queue<Integer> queue = new Queue<Integer>();
    queue.enqueue(s);
    marked[s] = true;
    while (!queue.isEmpty())
    {
        int v = queue.dequeue();

        for (FlowEdge e : G.adj(v))          found path from s to w
        {                                     in the residual network?
            int w = e.other(v);
            if (e.residualCapacityTo(w) > 0 && !marked[w])
            {
                edgeTo[w] = e;           save last edge on path to w;
                marked[w] = true;        ← mark w;
                queue.enqueue(w);       add w to the queue
            }
        }
        return marked[t];      ← is t reachable from s in residual network?
    }
}
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ ***Java implementation***
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

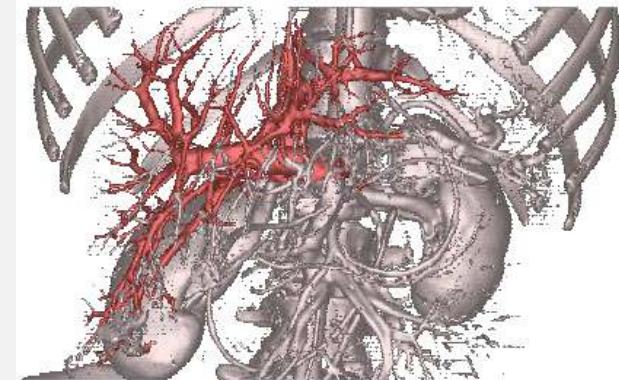
6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ ***applications***

Maxflow and mincut applications

Maxflow/mincut is a widely applicable problem-solving model.

- Data mining.
- Open-pit mining.
- Bipartite matching.
- Network reliability.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Distributed computing.
- Security of statistical data.
- Egalitarian stable matching.
- Multi-camera scene reconstruction.
- Sensor placement for homeland security.
- Many, many, more.



liver and hepatic vascularization segmentation

Bipartite matching problem

N students apply for N jobs.



Each gets several offers.



Is there a way to match all students to jobs?



bipartite matching problem

1	Alice	6	Adobe
	Adobe		Alice
	Amazon		Bob
	Google		Carol
2	Bob	7	Amazon
	Adobe		Alice
	Amazon		Bob
3	Carol		Dave
	Adobe		Eliza
	Facebook	8	Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

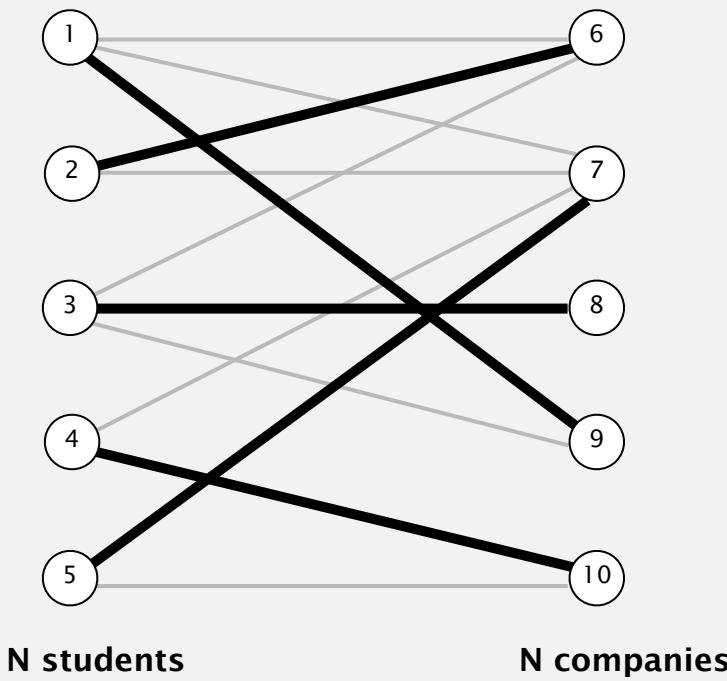
Bipartite matching problem

Given a bipartite graph, find a perfect matching.

perfect matching (solution)

Alice	— Google
Bob	— Adobe
Carol	— Facebook
Dave	— Yahoo
Eliza	— Amazon

bipartite graph

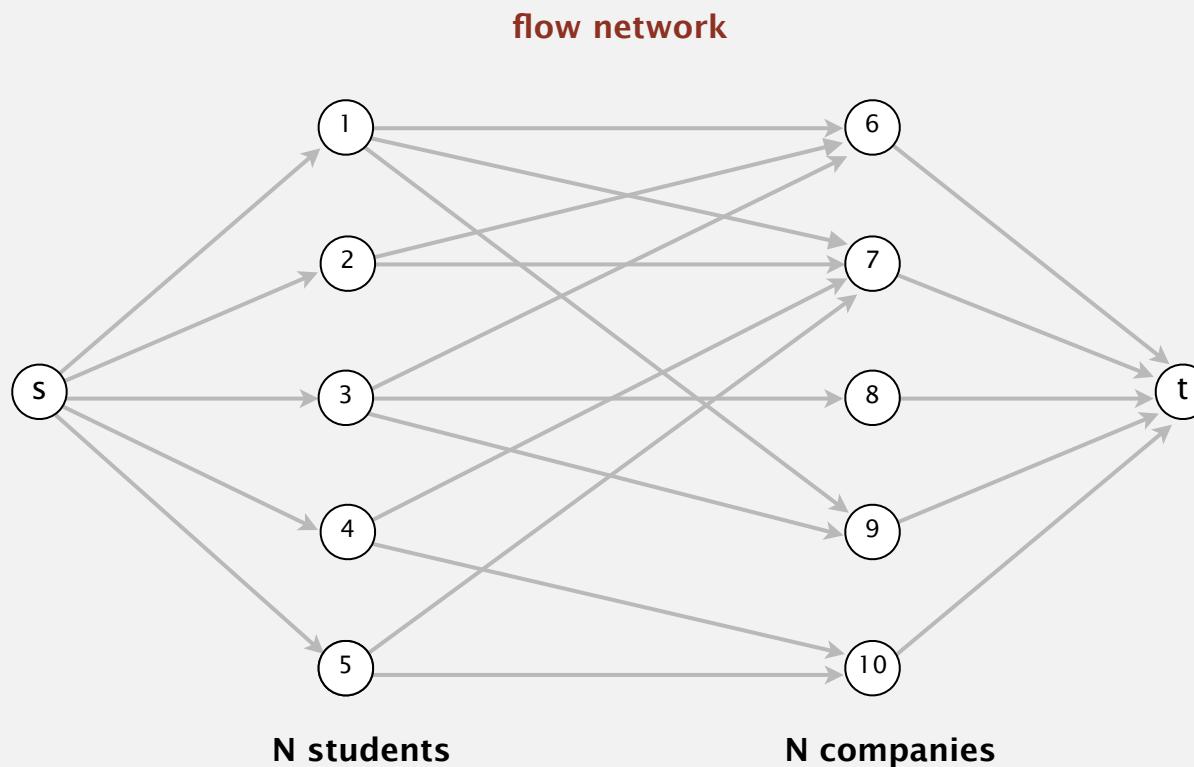


bipartite matching problem

1	Alice	6	Adobe
		7	Alice
2	Bob	Adobe	Adobe
		8	Amazon
3	Carol	Amazon	Amazon
		9	Facebook
4	Dave	Facebook	Facebook
		10	Google
5	Eliza	Google	Google
		6	Alice
		7	Bob
		8	Carol
		9	Dave
		10	Eliza

Network flow formulation of bipartite matching

- Create s, t , one vertex for each student, and one vertex for each job.
- Add edge from s to each student (capacity 1).
- Add edge from each job to t (capacity 1).
- Add edge from student to each job offered (infinite capacity).

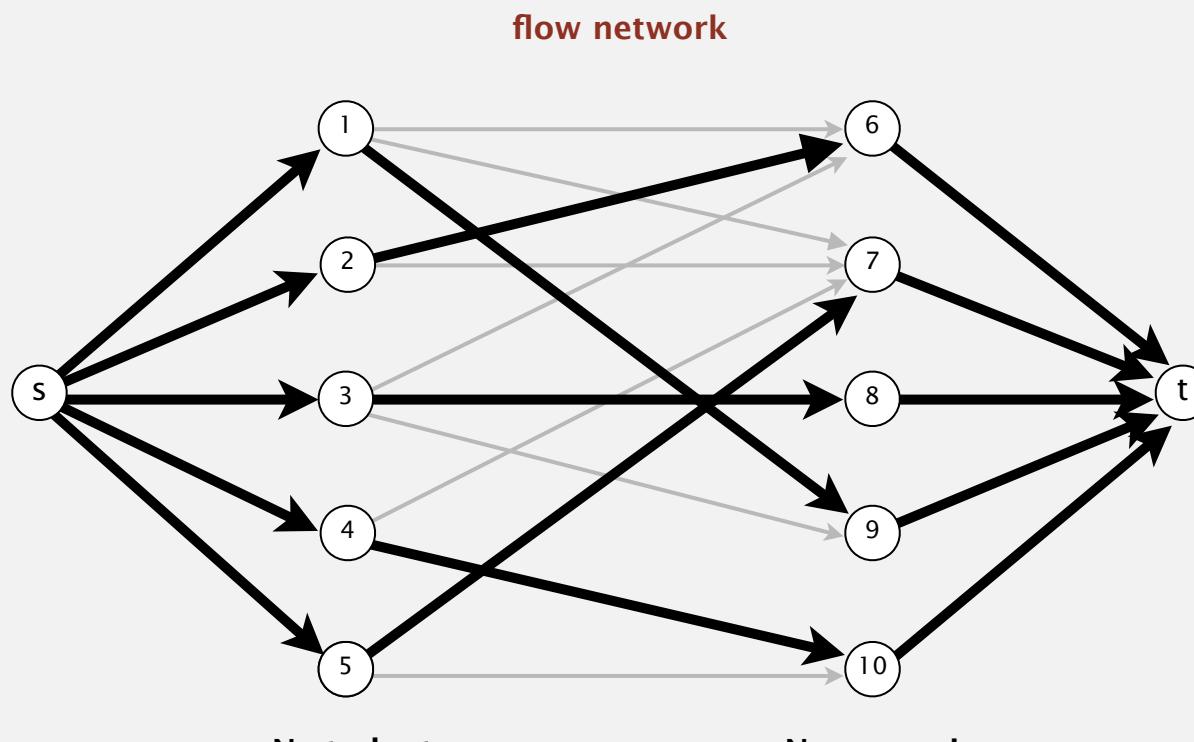


bipartite matching problem

1	Alice	6	Adobe
	Adobe		Alice
	Amazon		Bob
	Google		Carol
2	Bob	7	Amazon
	Adobe		Alice
	Amazon		Bob
3	Carol	8	Dave
	Adobe		Eliza
	Facebook		Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

Network flow formulation of bipartite matching

1-1 correspondence between perfect matchings in bipartite graph and integer-valued maxflows of value N .

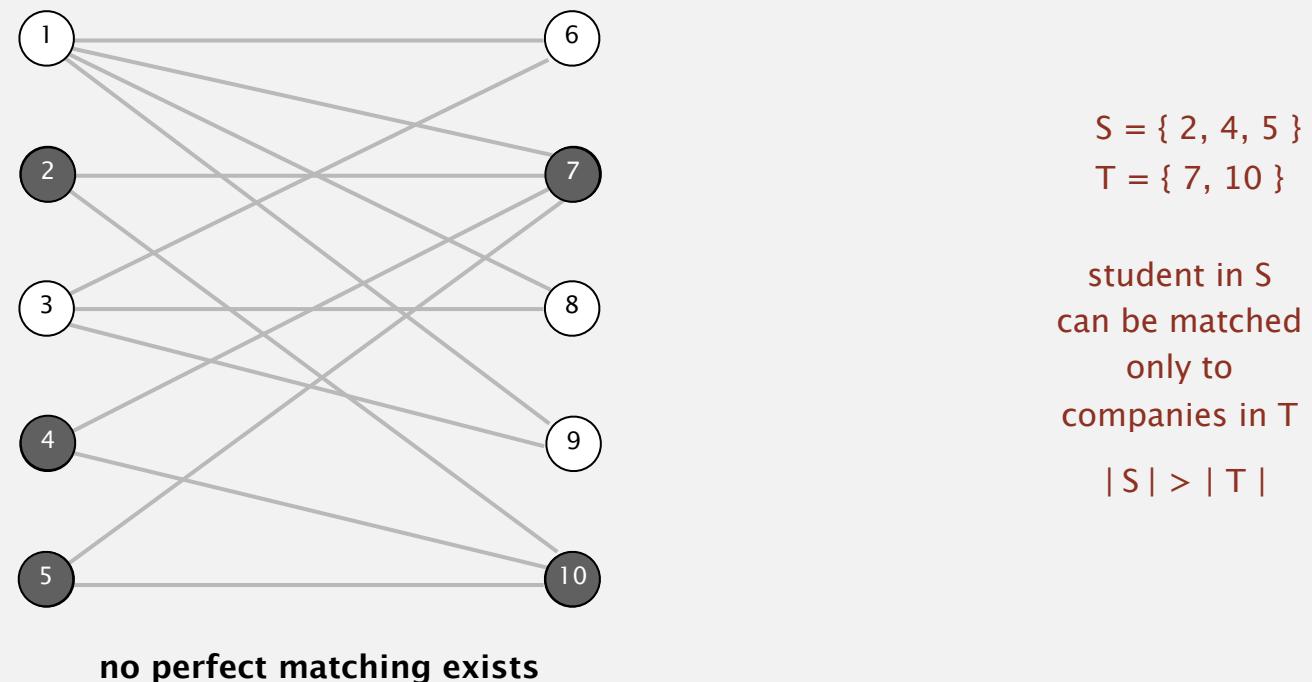


bipartite matching problem

1	Alice	6	Adobe
	Adobe		Alice
	Amazon		Bob
	Google		Carol
2	Bob	7	Amazon
	Adobe		Alice
	Amazon		Bob
3	Carol	8	Dave
	Adobe		Eliza
	Facebook		Facebook
	Google		Carol
4	Dave	9	Google
	Amazon		Alice
	Yahoo		Carol
5	Eliza	10	Yahoo
	Amazon		Dave
	Yahoo		Eliza

What the mincut tells us

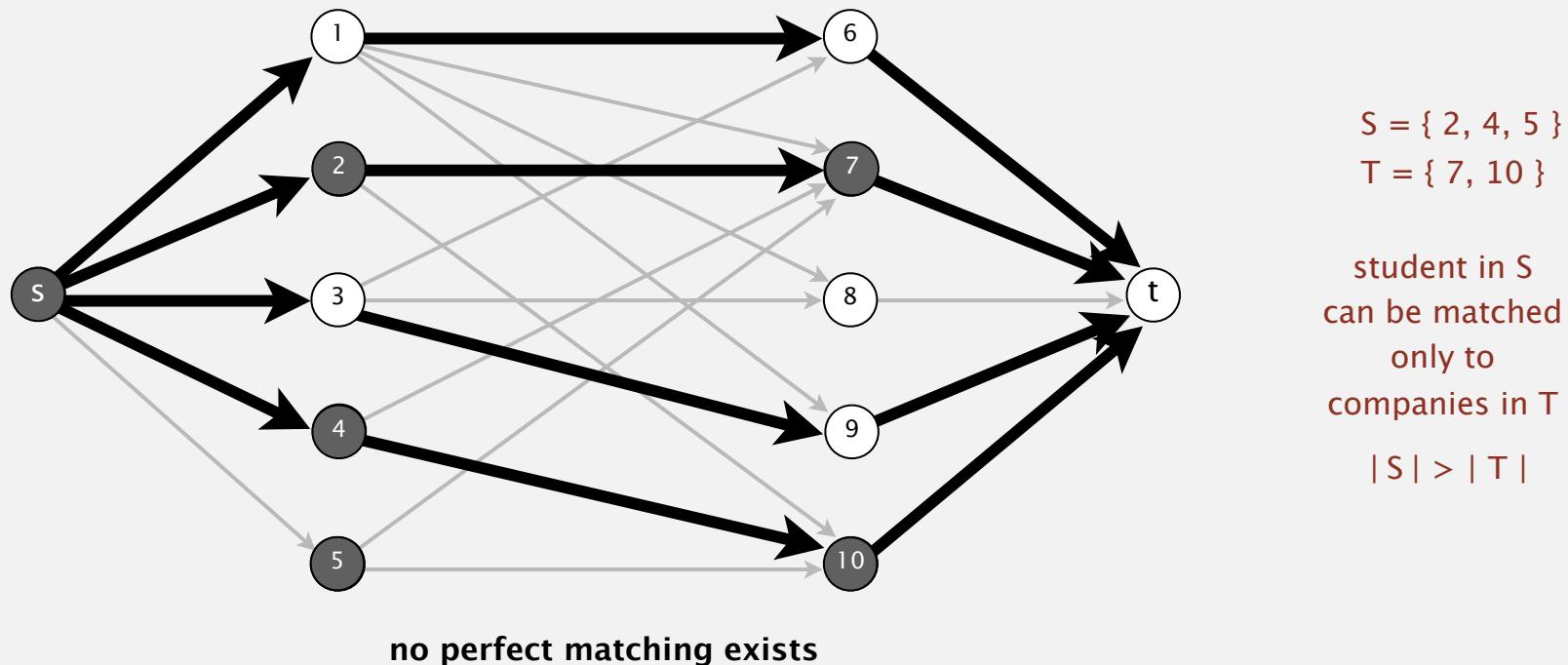
Goal. When no perfect matching, explain why.



What the mincut tells us

Mincut. Consider mincut (A, B) .

- Let S = students on s side of cut.
- Let T = companies on s side of cut.
- Fact: $|S| > |T|$; students in S can be matched only to companies in T .



Bottom line. When no perfect matching, mincut explains why.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	ATL	PHI	NYM	MON	
0		Atlanta	83	71	8	-	1	6	1
1		Philly	80	79	3	1	-	0	2
2		New York	78	78	6	6	0	-	0
3		Montreal	77	82	3	1	2	0	-

Montreal is mathematically eliminated.

- Montreal finishes with ≤ 80 wins.
- Atlanta already has 83 wins.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	ATL	PHI	NYM	MON	
0		Atlanta	83	71	8	-	1	6	1
1		Philly	80	79	3	1	-	0	2
2		New York	78	78	6	6	0	-	0
3		Montreal	77	82	3	1	2	0	-

Philadelphia is mathematically eliminated.

- Philadelphia finishes with ≤ 83 wins.
- Either New York or Atlanta will finish with ≥ 84 wins.

Observation. Answer depends not only on how many games already won and left to play, but on **whom** they're against.

Baseball elimination problem

Q. Which teams have a chance of finishing the season with the most wins?

i	team	wins	losses	to play	NYY	BAL	BOS	TOR	DET	
0		New York	75	59	28	-	3	8	7	3
1		Baltimore	71	63	28	3	-	2	7	4
2		Boston	69	66	27	8	2	-	0	0
3		Toronto	63	72	27	7	7	0	-	0
4		Detroit	49	86	27	3	4	0	0	-

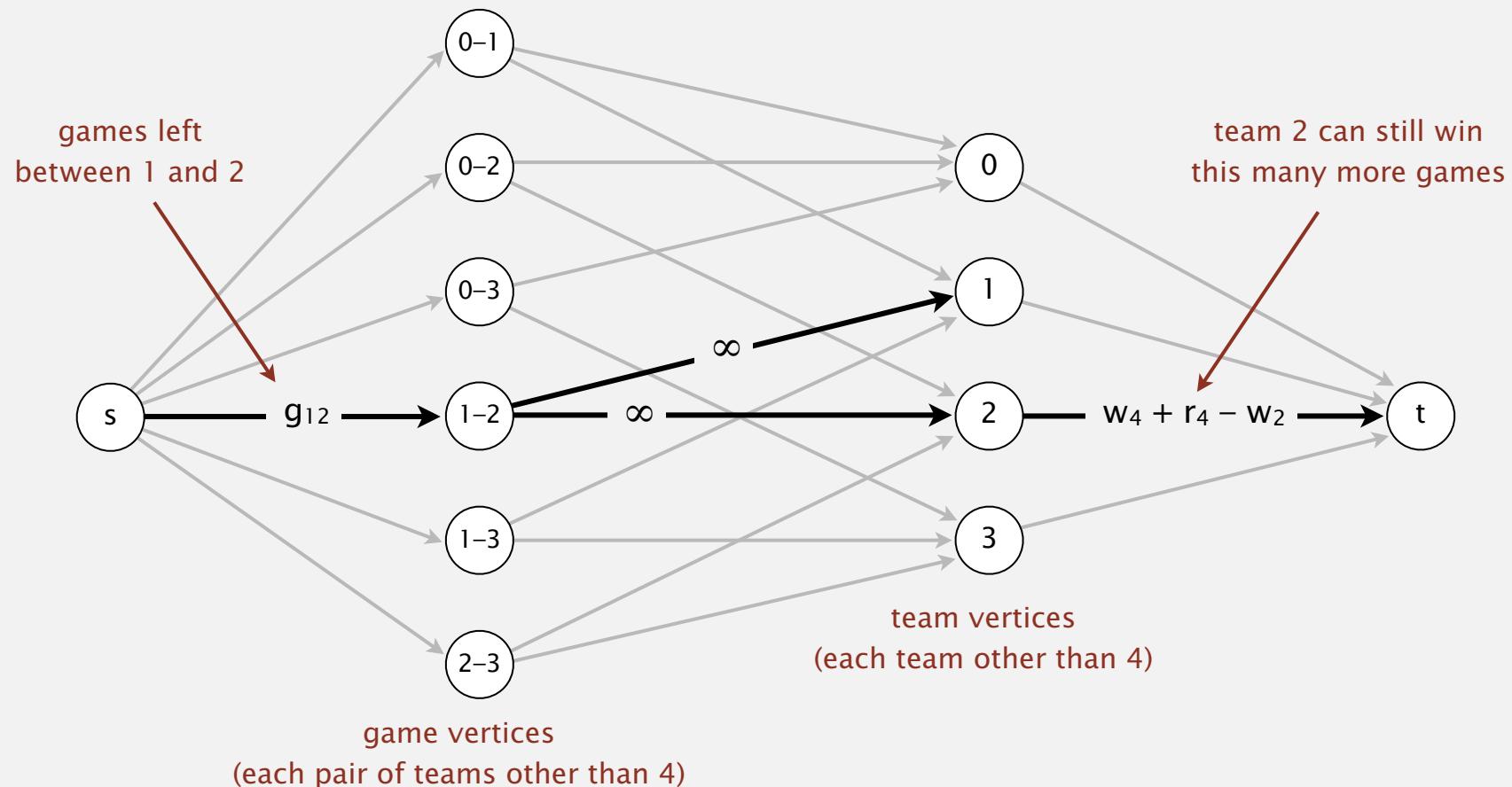
AL East (August 30, 1996)

Detroit is mathematically eliminated.

- Detroit finishes with ≤ 76 wins.
- Wins for $R = \{ \text{NYY}, \text{BAL}, \text{BOS}, \text{TOR} \} = 278$.
- Remaining games among $\{ \text{NYY}, \text{BAL}, \text{BOS}, \text{TOR} \} = 3 + 8 + 7 + 2 + 7 = 27$.
- Average team in R wins $305/4 = 76.25$ games.

Baseball elimination problem: maxflow formulation

Intuition. Remaining games flow from s to t .



Fact. Team 4 not eliminated iff all edges pointing from s are full in maxflow.

Maximum flow algorithms: theory

(Yet another) holy grail for theoretical computer scientists.

year	method	worst case	discovered by
1951	simplex	$E^3 U$	Dantzig
1955	augmenting path	$E^2 U$	Ford-Fulkerson
1970	shortest augmenting path	E^3	Dinitz, Edmonds-Karp
1970	fattest augmenting path	$E^2 \log E \log(EU)$	Dinitz, Edmonds-Karp
1977	blocking flow	$E^{5/2}$	Cherkasky
1978	blocking flow	$E^{7/3}$	Galil
1983	dynamic trees	$E^2 \log E$	Sleator-Tarjan
1985	capacity scaling	$E^2 \log U$	Gabow
1997	length function	$E^{3/2} \log E \log U$	Goldberg-Rao
2012	compact network	$E^2 / \log E$	Orlin
?	?	E	?

maxflow algorithms for sparse digraphs with E edges, integer capacities between 1 and U

Maximum flow algorithms: practice

Warning. Worst-case order-of-growth is generally not useful for predicting or comparing maxflow algorithm performance in practice.

Best in practice. Push-relabel method with gap relabeling: $E^{3/2}$.

On Implementing Push-Relabel Method for the Maximum Flow Problem

Boris V. Cherkassky¹ and Andrew V. Goldberg²

¹ Central Institute for Economics and Mathematics,
Krasikova St. 32, 117418, Moscow, Russia
cher@cemi.msk.su

² Computer Science Department, Stanford University
Stanford, CA 94305, USA
goldberg@cs.stanford.edu

Abstract. We study efficient implementations of the push-relabel method for the maximum flow problem. The resulting codes are faster than the previous codes, and much faster on some problem families. The speedup is due to the combination of heuristics used in our implementations. We also exhibit a family of problems for which the running time of all known methods seem to have a roughly quadratic growth rate.



European Journal of Operational Research 97 (1997) 509–542

EUROPEAN
JOURNAL
OF OPERATIONAL
RESEARCH

Theory and Methodology Computational investigations of maximum flow algorithms

Ravindra K. Ahuja ^a, Murali Kodialam ^b, Ajay K. Mishra ^c, James B. Orlin ^{d,*}

^a Department of Industrial and Management Engineering, Indian Institute of Technology, Kanpur, 208 016, India

^b AT & T Bell Laboratories, Holmdel, NJ 07733, USA

^c Katz Graduate School of Business, University of Pittsburgh, Pittsburgh, PA 15260, USA

^d Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Received 30 August 1995; accepted 27 June 1996

Summary

Mincut problem. Find an st -cut of minimum capacity.

Maxflow problem. Find an st -flow of maximum value.

Duality. Value of the maxflow = capacity of mincut.

Proven successful approaches.

- Ford-Fulkerson (various augmenting-path strategies).
- Preflow-push (various versions).

Open research challenges.

- Practice: solve real-word maxflow/mincut problems in linear time.
- Theory: prove it for worst-case inputs.
- Still much to be learned!

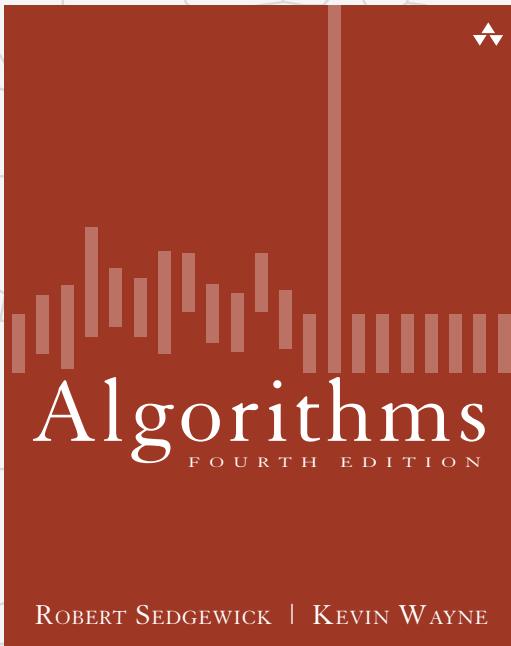
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ ***applications***



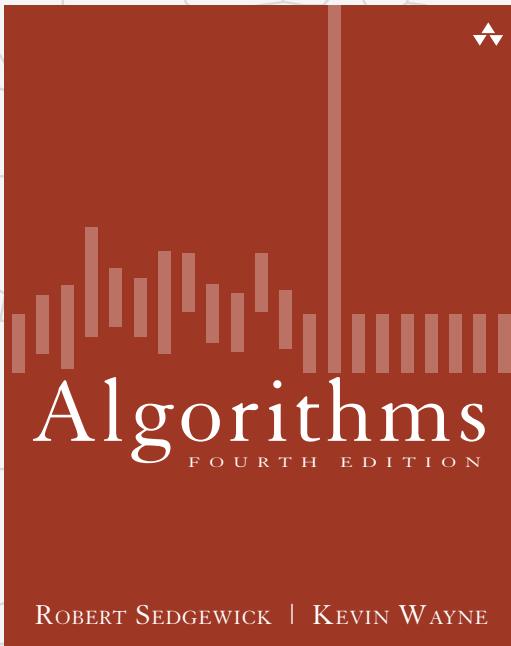
<http://algs4.cs.princeton.edu>

6.4 MAXIMUM FLOW

- ▶ *introduction*
- ▶ *Ford-Fulkerson algorithm*
- ▶ *maxflow-mincut theorem*
- ▶ *running time analysis*
- ▶ *Java implementation*
- ▶ *applications*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Questions about computation

- Q. What is a general-purpose computer?
- Q. Are there limits on the power of digital computers?
- Q. Are there limits on the power of machines we can build?



David Hilbert



Kurt Gödel



Alan Turing



Alonzo Church



John von Neumann

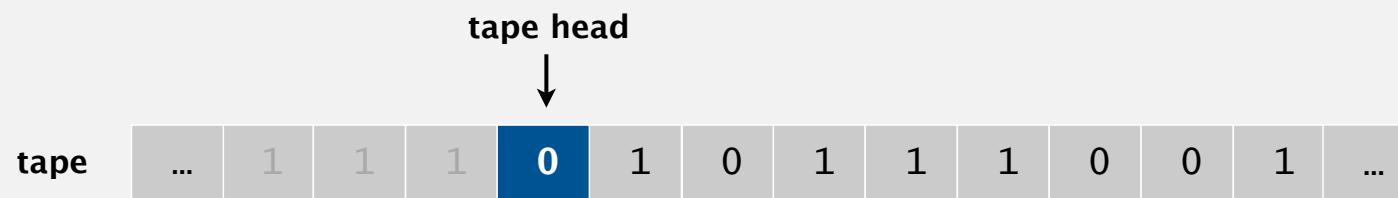
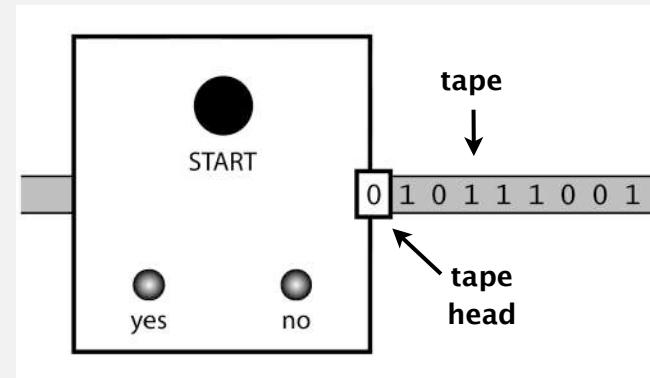
A simple model of computation: DFAs

Tape.

- Stores input.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- Moves one cell at a time.



Q. Is there a more powerful model of computation?

A. Yes.

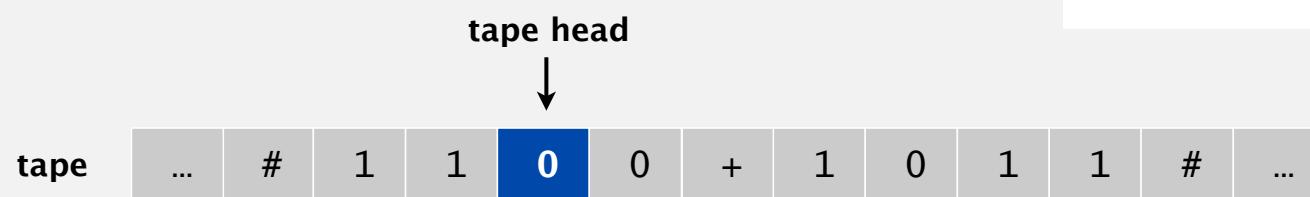
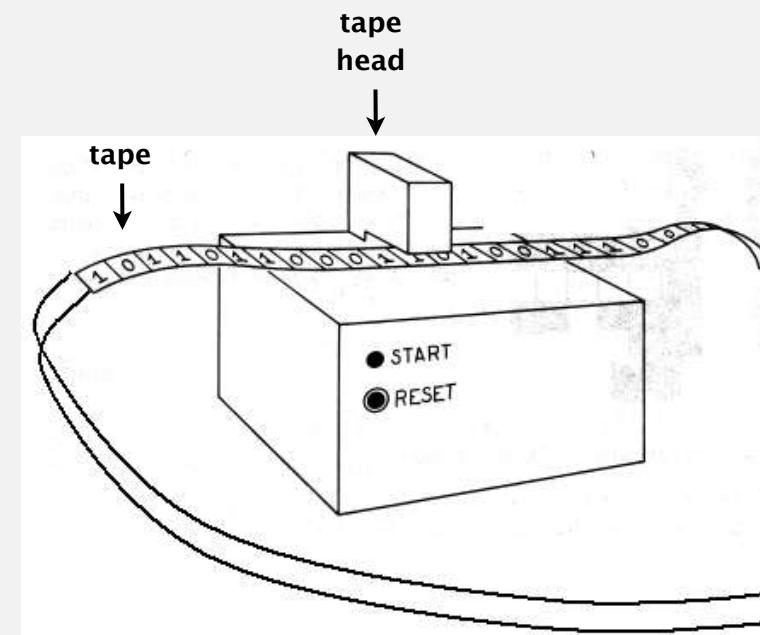
A universal model of computation: Turing machines

Tape.

- Stores input, **output**, and **intermediate results**.
- One arbitrarily long strip, divided into cells.
- Finite alphabet of symbols.

Tape head.

- Points to one cell of tape.
- Reads a symbol from active cell.
- **Writes** a symbol to active cell.
- Moves one cell at a time.



Q. Is there a more powerful model of computation?

A. No! ← most important scientific result of 20th century?

Church-Turing thesis (1936)

Turing machines can compute any function that can be computed by a physically harnessable process of the natural world.

Remark. "Thesis" and not a mathematical theorem because it's a statement about the physical world and not subject to proof.

but can be falsified

Use simulation to prove models equivalent.

- Android simulator on iPhone.
- iPhone simulator on Android.

Implications.

- No need to seek more powerful machines or languages.
- Enables rigorous study of computation (in this universe).

Bottom line. Turing machine is a **simple** and **universal** model of computation.

Church-Turing thesis: evidence

- 8 decades without a counterexample.
- Many, many models of computation that turned out to be equivalent.

"universal"
↓

model of computation	description
enhanced Turing machines	multiple heads, multiple tapes, 2D tape, nondeterminism
untyped lambda calculus	method to define and manipulate functions
recursive functions	functions dealing with computation on integers
unrestricted grammars	iterative string replacement rules used by linguists
extended L-systems	parallel string replacement rules that model plant growth
programming languages	Java, C, C++, Perl, Python, PHP, Lisp, PostScript, Excel
random access machines	registers plus main memory, e.g., TOY, Pentium
cellular automata	cells which change state based on local interactions
quantum computer	compute using superposition of quantum states
DNA computer	compute using biological operations on DNA

A question about algorithms

Q. Which algorithms are useful in practice?

- Measure running time as a function of input size N .
- Useful in practice ("efficient") = polynomial time for all inputs.

$$a N^b$$



von Neumann
(1953)



Nash
(1955)



Gödel
(1956)



Cobham
(1964)



Edmonds
(1965)



Rabin
(1966)

Ex 1. Sorting N items takes $N \log N$ compares using mergesort.

Ex 2. Finding best TSP tour on N points takes $N!$ steps using brute search.

Theory. Definition is broad and robust.

constants a and b tend to be small, e.g., $3 N^2$

Practice. Poly-time algorithms scale to huge problems.

Exponential growth

Exponential growth dwarfs technological change.

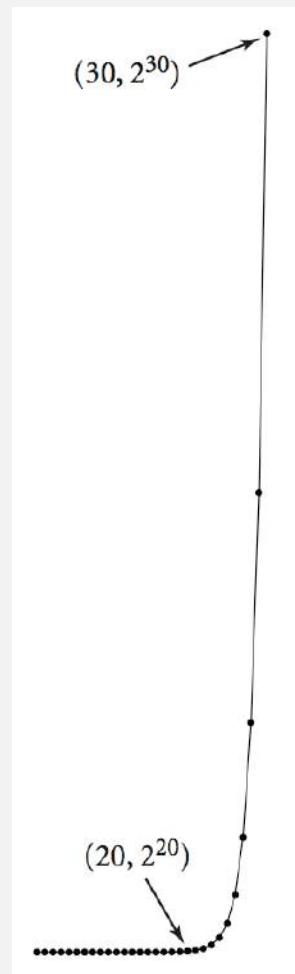
- Suppose you have a giant parallel computing device...
- With as many processors as electrons in the universe...
- And each processor has power of today's supercomputers...
- And each processor works for the life of the universe...

quantity	value
electrons in universe †	10^{79}
supercomputer instructions per second	10^{13}
age of universe in seconds	10^{17}

† estimated

- Will not help solve 1,000 city TSP problem via brute force.

$$1000! \gg 10^{1000} \gg 10^{79} \times 10^{13} \times 10^{17}$$



Questions about problems

Q. Which problems can we solve in practice?

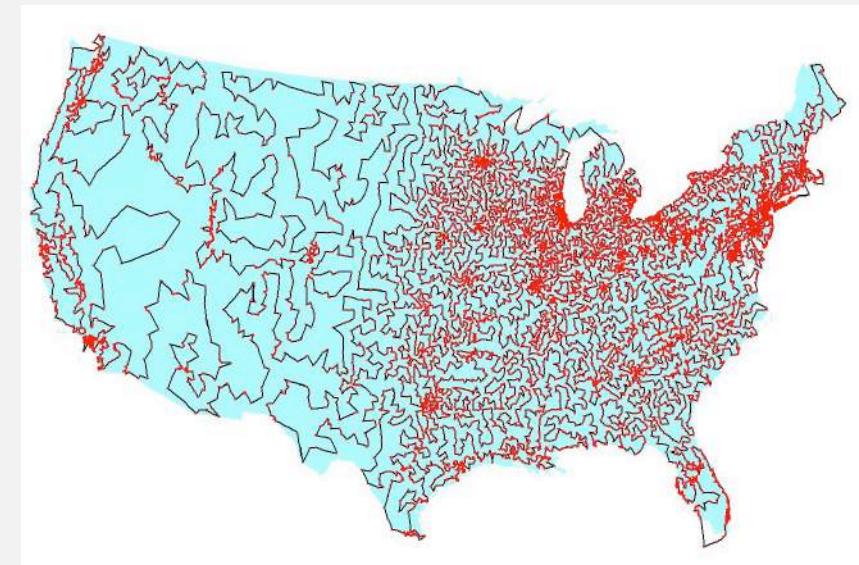
A. Those with poly-time algorithms.

Q. Which problems have poly-time algorithms?

A. Not so easy to know. Focus of today's lecture.



many known poly-time algorithms for sorting



no known poly-time algorithm for TSP

Bird's-eye view

Def. A problem is **intractable** if it can't be solved in polynomial time.

Desiderata. Prove that a problem is intractable.

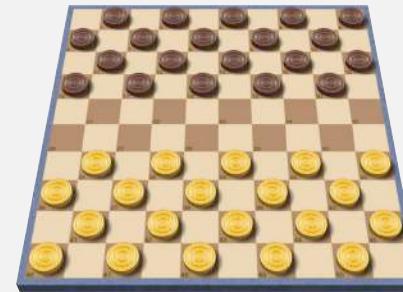
Two problems that provably require exponential time.

- Given a constant-size program, does it halt in at most K steps?
- Given N -by- N checkers board position, can the first player force a win?

input size = $c + \lg K$



using forced capture rule



Frustrating news. Very few successes.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Four fundamental problems

LSOLVE. Given a system of **linear equations**, find a solution.

$$\begin{array}{rcl} 0x_0 + 1x_1 + 1x_2 & = & 4 \\ 2x_0 + 4x_1 - 2x_2 & = & 2 \\ 0x_0 + 3x_1 + 15x_2 & = & 36 \end{array}$$

$$\begin{array}{rcl} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$$

variables are
real numbers

LP. Given a system of **linear inequalities**, find a solution.

$$\begin{array}{rcl} 48x_0 + 16x_1 + 119x_2 & \leq & 88 \\ 5x_0 + 4x_1 + 35x_2 & \geq & 13 \\ 15x_0 + 4x_1 + 20x_2 & \geq & 23 \\ x_0, x_1, x_2 & \geq & 0 \end{array}$$

$$\begin{array}{rcl} x_0 & = & 1 \\ x_1 & = & 1 \\ x_2 & = & \frac{1}{5} \end{array}$$

variables are
real numbers

ILP. Given a system of **linear inequalities**, find a 0-1 solution.

$$\begin{array}{rcl} x_1 + x_2 & \geq & 1 \\ x_0 + x_1 + x_2 & \geq & 1 \\ x_0 + x_1 + x_2 & \leq & 2 \end{array}$$

$$\begin{array}{rcl} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$$

variables are
0 or 1

SAT. Given a system of **boolean equations**, find a binary solution.

$$\begin{array}{l} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) = \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) = \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) = \text{true} \end{array}$$

$$\begin{array}{rcl} x_0 & = & \text{false} \\ x_1 & = & \text{false} \\ x_2 & = & \text{true} \end{array}$$

variables are
true or false

Four fundamental problems

LSOLVE. Given a system of linear equations, find a solution.

LP. Given a system of linear inequalities, find a solution.

ILP. Given a system of linear inequalities, find a 0-1 solution.

SAT. Given a system of boolean equations, find a binary solution.

Q. Which of these problems have **poly-time** algorithms?

- LSOLVE. Yes. Gaussian elimination solves N -by- N system in N^3 time.
- LP. Yes. Ellipsoid algorithm is poly-time. ← but was open problem for decades
- ILP, SAT. No poly-time algorithm known or believed to exist!

but we still don't know for sure

Search problems

Search problem. Given an instance I of a problem, find a solution S .

Requirement. Must be able to efficiently check that S is a solution.

poly-time in size of instance I

or report
none exists



Search problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

LSOLVE. Given a system of linear equations, find a solution.

$$\begin{array}{rcl} 0x_0 + 1x_1 + 1x_2 & = & 4 \\ 2x_0 + 4x_1 - 2x_2 & = & 2 \\ 0x_0 + 3x_1 + 15x_2 & = & 36 \end{array}$$

$$\begin{array}{rcl} x_0 & = & -1 \\ x_1 & = & 2 \\ x_2 & = & 2 \end{array}$$

instance I

solution S

To check solution S , plug in values and verify each equation.

Search problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

LP. Given a system of linear inequalities, find a solution.

$$\begin{array}{lllll} 48x_0 & + 16x_1 & + 119x_2 & \leq & 88 \\ 5x_0 & + 4x_1 & + 35x_2 & \geq & 13 \\ 15x_0 & + 4x_1 & + 20x_2 & \geq & 23 \\ x_0, & x_1, & x_2 & \geq & 0 \end{array}$$

instance I

$$\begin{array}{ll} x_0 & = 1 \\ x_1 & = 1 \\ x_2 & = \frac{1}{5} \end{array}$$

solution S

To check solution S , plug in values and verify each inequality.

Search problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

ILP. Given a system of linear inequalities, find a binary solution.

$$\begin{array}{rcl} x_1 & + & x_2 \geq 1 \\ x_0 & + & x_2 \geq 1 \\ x_0 & + & x_1 + x_2 \leq 2 \end{array}$$

$$\begin{array}{rcl} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$$

instance I

solution S

To check solution S , plug in values and verify each inequality.

Search problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

SAT. Given a system of boolean equations, find a boolean solution.

$$\begin{array}{lll} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) & = \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) & = \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) & = \text{true} \end{array}$$

instance I

$$\begin{array}{l} x_0 = \text{false} \\ x_1 = \text{false} \\ x_2 = \text{true} \end{array}$$

solution S

To check solution S , plug in values and verify each equation.

Search problems

Search problem. Given an instance I of a problem, **find** a solution S .

Requirement. Must be able to efficiently **check** that S is a solution.

FACTOR. Given an n -bit integer x , find a nontrivial factor.

input size = number of bits

147573952589676412927

instance I

193707721

solution S

To check solution S , long divide 193707721 into 147573952589676412927.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

NP

Def. NP is the class of all search problems.

Note: classic definition limits
NP to yes-no problems

problem	description	poly-time algorithm	instance I	solution S
LSOLVE (A, b)	Find a vector x that satisfies $Ax = b$	Gaussian elimination	$\begin{aligned} 0x_0 + 1x_1 + 1x_2 &= 4 \\ 2x_0 + 4x_1 - 2x_2 &= 2 \\ 0x_0 + 3x_1 + 15x_2 &= 36 \end{aligned}$	$\begin{aligned} x_0 &= -1 \\ x_1 &= 2 \\ x_2 &= 2 \end{aligned}$
LP (A, b)	Find a vector x that satisfies $Ax \leq b$	ellipsoid	$\begin{aligned} 48x_0 + 16x_1 + 119x_2 &\leq 88 \\ 5x_0 + 4x_1 + 35x_2 &\geq 13 \\ 15x_0 + 4x_1 + 20x_2 &\geq 23 \\ x_0, x_1, x_2 &\geq 0 \end{aligned}$	$\begin{aligned} x_0 &= 1 \\ x_1 &= 1 \\ x_2 &= \frac{1}{5} \end{aligned}$
ILP (A, b)	Find a binary vector x that satisfies $Ax \leq b$???	$\begin{aligned} x_1 + x_2 &\geq 1 \\ x_0 + x_2 &\geq 1 \\ x_0 + x_1 + x_2 &\leq 2 \end{aligned}$	$\begin{aligned} x_0 &= 0 \\ x_1 &= 1 \\ x_2 &= 1 \end{aligned}$
SAT (Φ, b)	Find a boolean vector x that satisfies $\Phi(x) = b$???	$\begin{aligned} (x'_1 \text{ or } x'_2) \text{ and } (x_0 \text{ or } x_2) &= \text{true} \\ (x_0 \text{ or } x_1) \text{ and } (x_1 \text{ or } x'_2) &= \text{false} \\ (x_0 \text{ or } x_2) \text{ and } (x'_0) &= \text{true} \end{aligned}$	$\begin{aligned} x_0 &= \text{false} \\ x_1 &= \text{false} \\ x_2 &= \text{true} \end{aligned}$
FACTOR (x)	Find a nontrivial factor of the integer x	???	147573952589676412927	193707721

Significance. What scientists and engineers **aspire to compute** feasibly.

P

Def. P is the class of search problems solvable in poly-time.

Note: classic definition limits
P to yes-no problems

problem	description	poly-time algorithm	instance I	solution S
LSOLVE (A, b)	Find a vector x that satisfies $Ax = b$	Gaussian elimination (Edmonds 1967)	$\begin{array}{l} 0x_0 + 1x_1 + 1x_2 = 4 \\ 2x_0 + 4x_1 - 2x_2 = 2 \\ 0x_0 + 3x_1 + 15x_2 = 36 \end{array}$	$\begin{array}{l} x_0 = -1 \\ x_1 = 2 \\ x_2 = 2 \end{array}$
LP (A, b)	Find a vector x that satisfies $Ax \leq b$	ellipsoid (Khachiyan 1979)	$\begin{array}{l} 48x_0 + 16x_1 + 119x_2 \leq 88 \\ 5x_0 + 4x_1 + 35x_2 \geq 13 \\ 15x_0 + 4x_1 + 20x_2 \geq 23 \\ x_0, x_1, x_2 \geq 0 \end{array}$	$\begin{array}{l} x_0 = 1 \\ x_1 = 1 \\ x_2 = \frac{1}{5} \end{array}$
SORT (a)	Find a permutation that puts array a in order	mergesort (von Neumann 1945)	$\begin{array}{cccccc} 2 & 3 & 8 & 5 & 1 & 2 \\ 9 & 1 & 2 & 2 & 0 & 3 \end{array}$	5 2 4 0 1 3
STCONN (G, s, t)	Find a path in a graph G from s to t	depth-first search (Theseus)		

Significance. What scientists and engineers **do compute** feasibly.

Nondeterminism

Nondeterministic machine can **guess** the desired solution.

Ex. `int[] a = new int[N];`

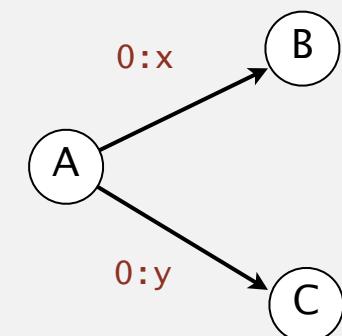
- Java: initializes entries to 0.
- Nondeterministic machine: initializes entries to the solution!

recall NFA implementation

ILP. Given a system of linear inequalities, **guess** a 0-1 solution.

$$\begin{array}{rcl} x_1 & + & x_2 \geq 1 \\ x_0 & + & x_2 \geq 1 \\ x_0 & + & x_1 + x_2 \leq 2 \end{array}$$

$$\begin{array}{rcl} x_0 & = & 0 \\ x_1 & = & 1 \\ x_2 & = & 1 \end{array}$$



Ex. Turing machine.

- Deterministic: state, input determines next state.
- Nondeterministic: more than one possible next state.

NP. Search problems solvable in poly time on a nondeterministic TM.

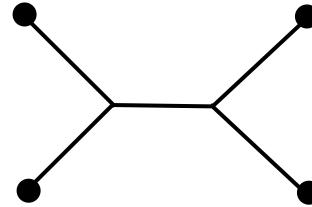
Extended Church-Turing thesis

P = search problems solvable in poly-time **in the natural world.**

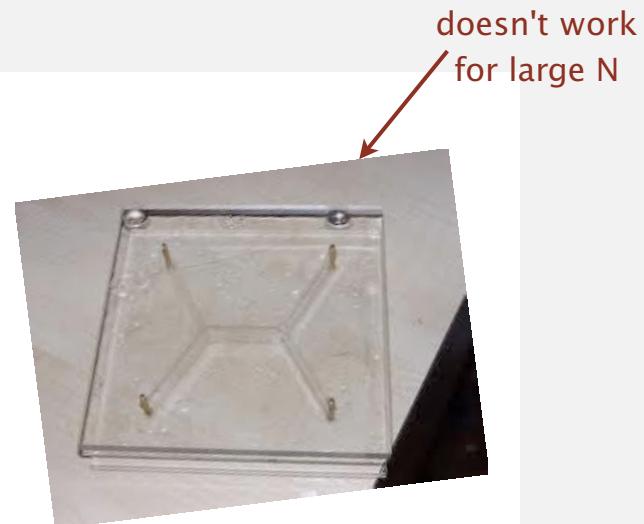
Evidence supporting thesis. True for all physical computers.

Natural computers? No successful attempts (yet).

Ex. Computing Steiner trees with soap bubbles



STEINER: Find set of lines of minimal length connecting N given points



Implication. To make future computers more efficient, suffices to focus on improving implementation of existing designs.

P vs. NP

Does P = NP ?



Copyright © 1990, Matt Groening



Copyright © 2000, Twentieth Century Fox

Automating creativity

Q. Being creative vs. appreciating creativity?

Ex. Mozart composes a piece of music; our neurons appreciate it.

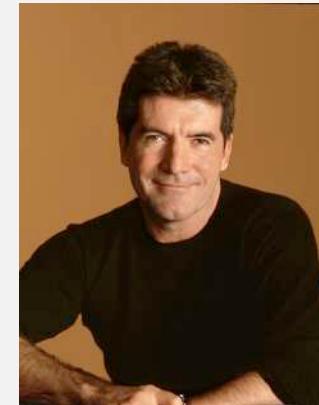
Ex. Wiles proves a deep theorem; a colleague referees it.

Ex. Boeing designs an efficient airfoil; a simulator verifies it.

Ex. Einstein proposes a theory; an experimentalist validates it.



creative



ordinary

Computational analog. Does P = NP?

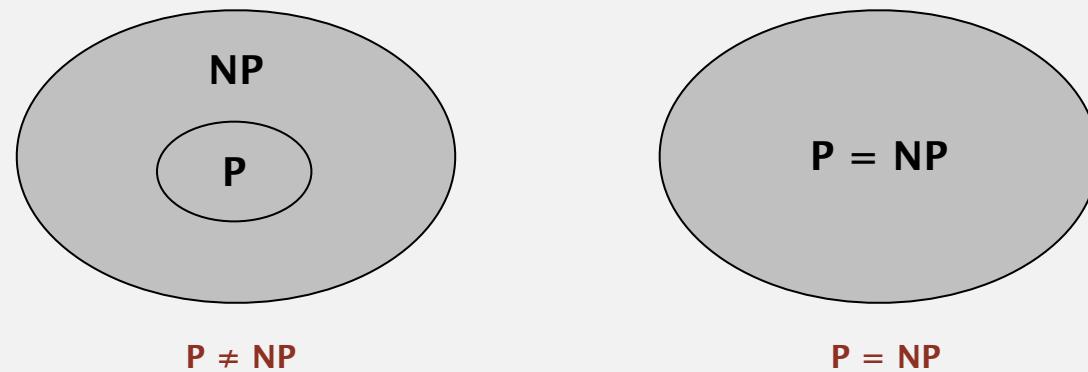
The central question

P. Class of search problems solvable in poly-time.

NP. Class of all search problems.

Does $P = NP$? [Can you always avoid brute-force searching and do better]

Two worlds.



If $P = NP$... Poly-time algorithms for SAT, ILP, TSP, FACTOR, ...

If $P \neq NP$... Would learn something fundamental about our universe.

Overwhelming consensus. $P \neq NP$.

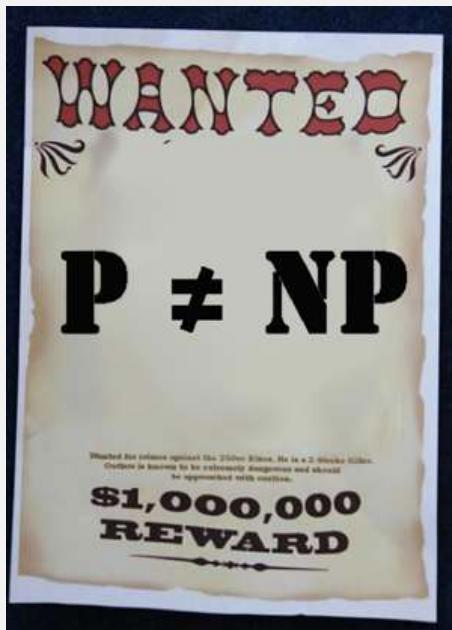
The central question

P. Class of search problems solvable in poly-time.

NP. Class of all search problems.

Does P = NP ? [Can you always avoid brute-force searching and do better]

Millennium prize. \$1 million for resolution of P = NP problem.



Clay Mathematics Institute
Dedicated to increasing and disseminating mathematical knowledge

HOME | ABOUT CMI | PROGRAMS | NEWS & EVENTS | AWARDS | SCHOLARS | PUBLICATIONS

Millennium Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven *Prize Problems*. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a \$7 million prize fund for the solution to these problems, with \$1 million allocated to each. During the *Millennium Meeting* held on May 24, 2000 at the Collège de France, Timothy Gowers presented a lecture entitled *The Importance of Mathematics*, aimed for the general public, while John Tate and Michael Atiyah spoke on the problems. The CMI invited specialists to formulate each problem.

► [Birch and Swinnerton-Dyer Conjecture](#)
► [Hodge Conjecture](#)
► [Navier-Stokes Equations](#)
► [P vs NP](#)
► [Poincaré Conjecture](#)
► [Riemann Hypothesis](#)
► [Yang-Mills Theory](#)
► [Rules](#)
► [Millennium Meeting Videos](#)

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*

Periodic table of the elements

Periodic Table of the Elements																			
																		O	
1	IA	1 H	IIA															2 He	
2		3 Li	4 Be															10 Ne	
3		11 Na	12 Mg	IIIB	IVB	VB	VIB	VIIB	— VII —	IB	IIB	5 B	6 C	7 N	8 O	9 F			
4		19 K	20 Ca	21 Sc	22 Ti	23 V	24 Cr	25 Mn	26 Fe	27 Co	28 Ni	29 Cu	30 Zn	13 Al	14 Si	15 P	16 S	17 Cl	18 Ar
5		37 Rb	38 Sr	39 Y	40 Zr	41 Nb	42 Mo	43 Tc	44 Ru	45 Rh	46 Pd	47 Ag	48 Cd	49 In	50 Sn	51 Sb	52 Te	53 I	54 Xe
6		55 Cs	56 Ba	57 *La	72 Hf	73 Ta	74 W	75 Re	76 Os	77 Ir	78 Pt	79 Au	80 Hg	81 Tl	82 Pb	83 Bi	84 Po	85 At	86 Rn
7		87 Fr	88 Ra	+Ac	Rf	Ha	Sg	Ns	Hs	Mt	110	111	112	113					
* Lanthanide Series		58 Ce	59 Pr	60 Nd	61 Pm	62 Sm	63 Eu	64 Gd	65 Tb	66 Dy	67 Ho	68 Er	69 Tm	70 Yb	71 Lu				
+ Actinide Series		90 Th	91 Pa	92 U	93 Np	94 Pu	95 Am	96 Cm	97 Bk	98 Cf	99 Es	100 Fm	101 Md	102 No	103 Lr				

A key problem: satisfiability

SAT. Given a system of boolean equations, find a solution.

$$x'_1 \text{ or } x_2 \text{ or } x_3 = \text{true}$$

$$x_1 \text{ or } x'_2 \text{ or } x_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x'_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x_4 = \text{true}$$

Key applications.

- Automatic verification systems for software.
- Electronic design automation (EDA) for hardware.
- Mean field diluted spin glass model in physics.
- ...

Exhaustive search

Q. How to solve an instance of SAT with n variables?

A. Exhaustive search: try all 2^n truth assignments.

Q. Can we do anything substantially more clever?

Conjecture. No poly-time algorithm for SAT.

"intractable"



www.jplyon.co.uk

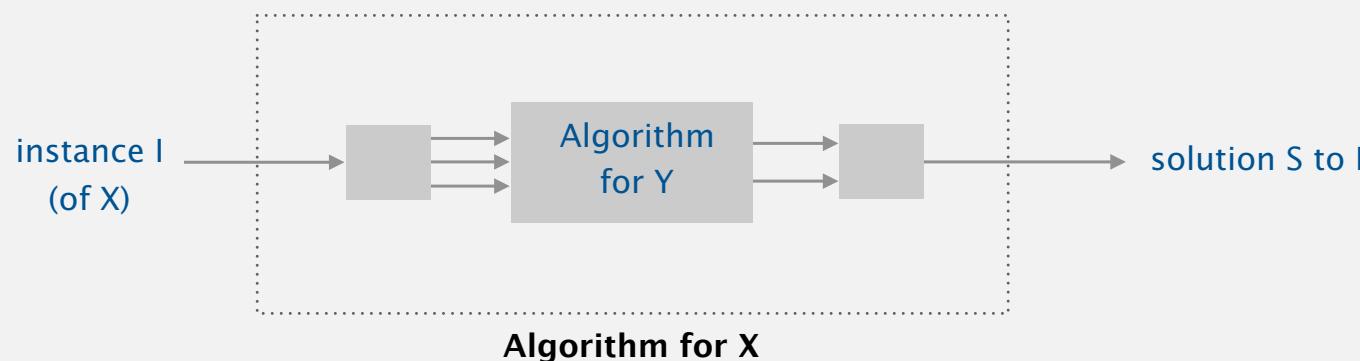
Classifying problems

Q. Which search problems are in P?

A. No easy answers (we don't even know whether $P = NP$).

Problem X **poly-time reduces** to problem Y if X can be solved with:

- Polynomial number of standard computational steps.
- Polynomial number of calls to Y .



Consequence. If SAT poly-time reduces to Y , then we conclude that Y is (probably) intractable.

SAT poly-time reduces to ILP

SAT. Given a system of boolean equations, find a solution.

$$x'_1 \text{ or } x_2 \text{ or } x_3 = \text{true}$$

$$x_1 \text{ or } x'_2 \text{ or } x_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x'_3 = \text{true}$$

$$x'_1 \text{ or } x'_2 \text{ or } x_4 = \text{true}$$

← can to reduce any SAT problem to this form

ILP. Given a system of linear inequalities, find a 0-1 solution.

$C_i = 1$ iff equation i is satisfied

$$C_1 \geq 1 - x_1$$

$$C_1 \geq x_2$$

$$C_1 \geq x_3$$

$$C_1 \leq (1 - x_1) + x_2 + x_3$$

$$C_2 \geq x_1$$

$$C_2 \geq 1 - x_2$$

$$C_2 \geq x_3$$

$$C_2 \leq 1 + x_1 - x_2 + x_3$$

$$C_3 \geq 1 - x_1$$

$$C_3 \geq 1 - x_2$$

$$C_3 \geq 1 - x_3$$

$$C_3 \leq 3 - x_1 - x_2 - x_3$$

$$C_4 \geq 1 - x_1$$

$$C_4 \geq 1 - x_2$$

$$C_4 \geq x_4$$

$$C_4 \leq 2 - x_1 - x_2 + x_4$$

$\Phi = 1$ iff all equations are satisfied

$$\Phi \leq C_1$$

$$\Phi \leq C_2$$

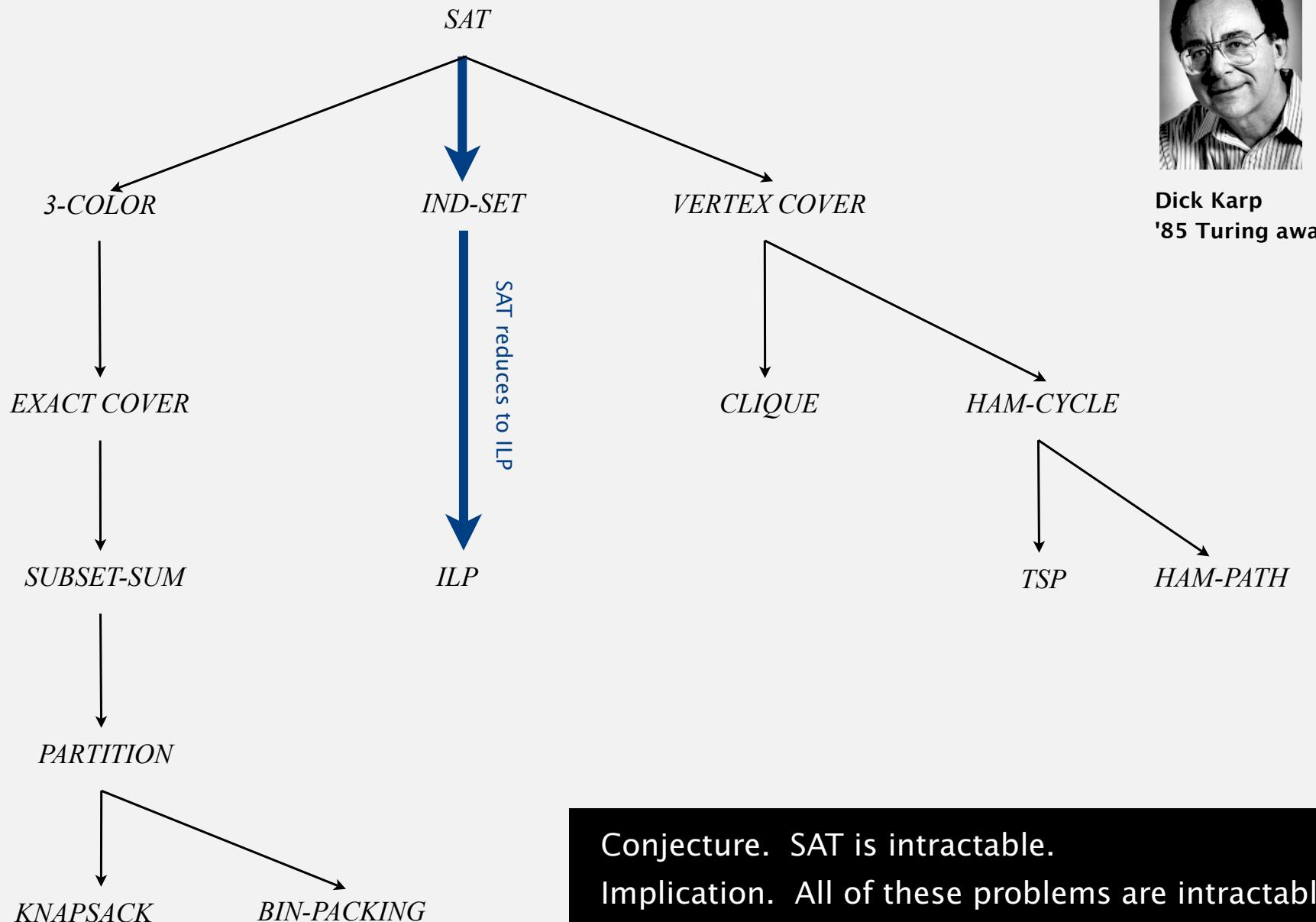
$$\Phi \leq C_3$$

$$\Phi \leq C_4$$

$$\Phi \geq C_1 + C_2 + C_3 + C_4 - 3$$

solution to this ILP instance gives solution to original SAT instance

More poly-time reductions from boolean satisfiability



Still more reductions from SAT

Aerospace engineering. Optimal mesh partitioning for finite elements.

Biology. Phylogeny reconstruction.

Chemical engineering. Heat exchanger network synthesis.

Chemistry. Protein folding.

Civil engineering. Equilibrium of urban traffic flow.

Economics. Computation of arbitrage in financial markets with friction.

Electrical engineering. VLSI layout.

Environmental engineering. Optimal placement of contaminant sensors.

Financial engineering. Minimum risk portfolio of given return.

Game theory. Nash equilibrium that maximizes social welfare.

Mathematics. Given integer a_1, \dots, a_n , compute $\int_0^{2\pi} \cos(a_1\theta) \times \cos(a_2\theta) \times \dots \times \cos(a_n\theta) d\theta$

Mechanical engineering. Structure of turbulence in sheared flows.

Medicine. Reconstructing 3d shape from biplane angiogram.

Operations research. Traveling salesperson problem.

Physics. Partition function of 3d Ising model.

Politics. Shapley-Shubik voting power.

Recreation. Versions of Sudoku, Checkers, Minesweeper, Tetris.

Statistics. Optimal experimental design.

plus over 6,000 scientific papers per year

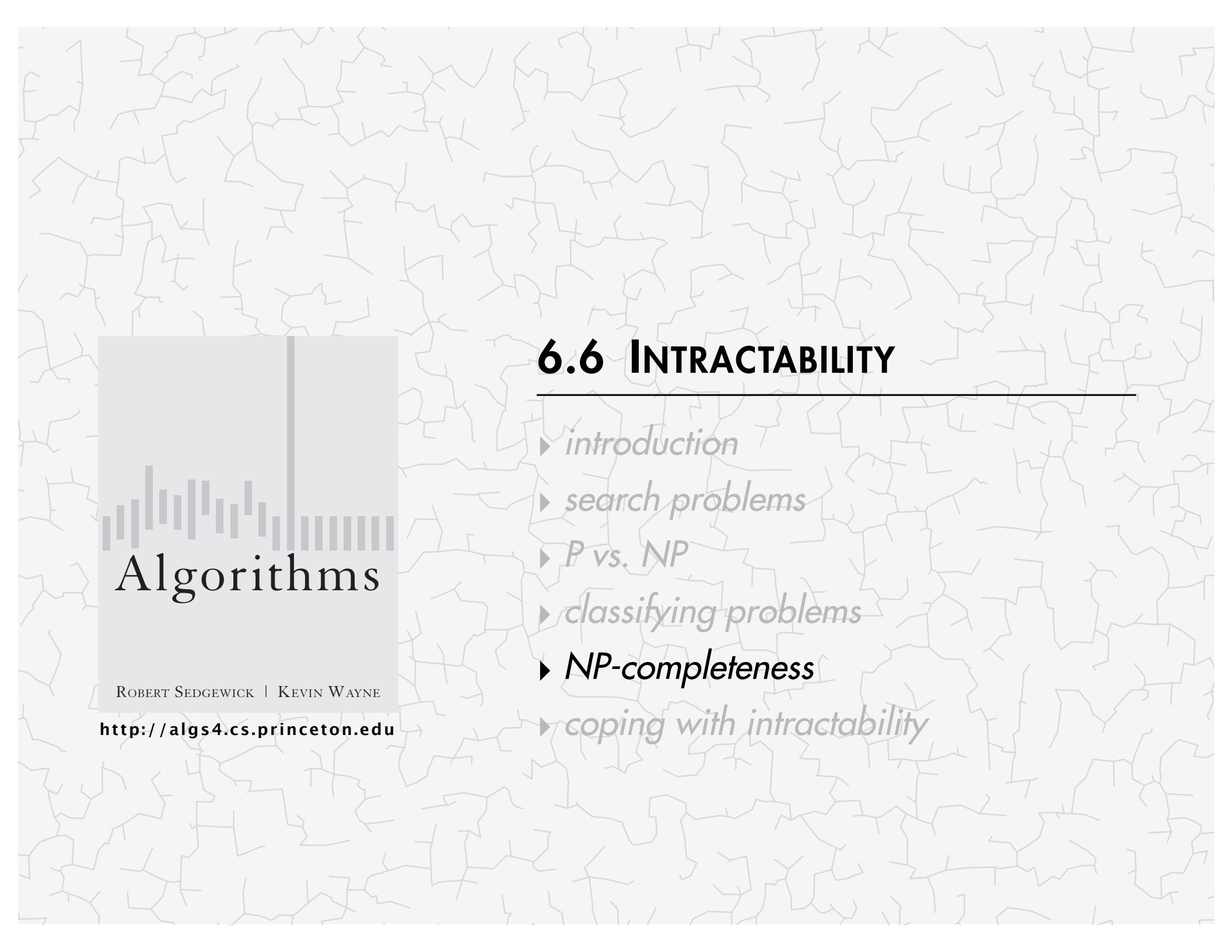
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ ***NP-completeness***
- ▶ *coping with intractability*

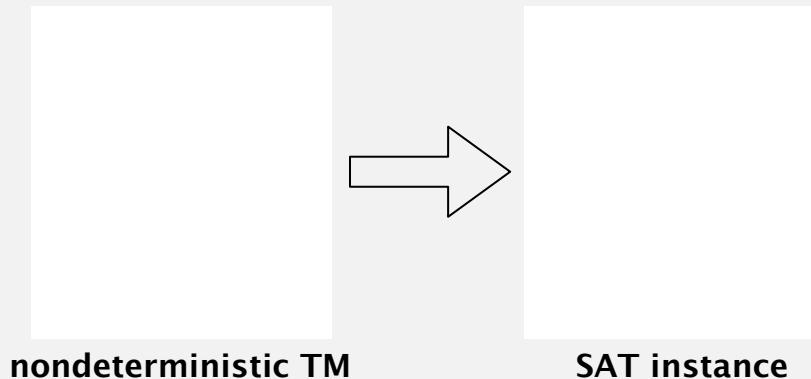
NP-completeness

Def. An NP problem is **NP-complete** if every problem in NP poly-time reduce to it.

Proposition. [Cook 1971, Levin 1973] SAT is NP-complete.

Extremely brief proof sketch:

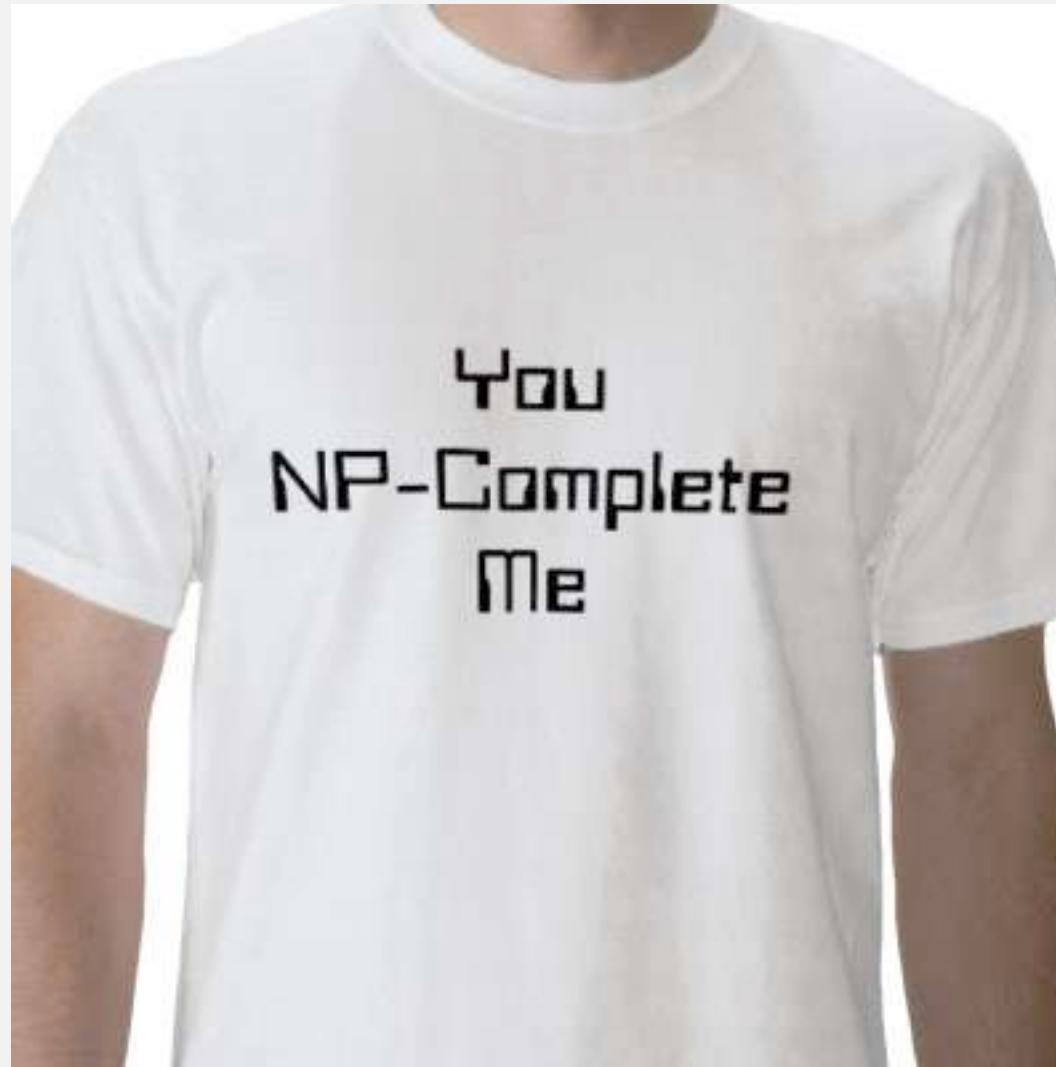
- Convert non-deterministic TM notation to SAT notation.
- If you can solve SAT, you can solve any problem in NP.



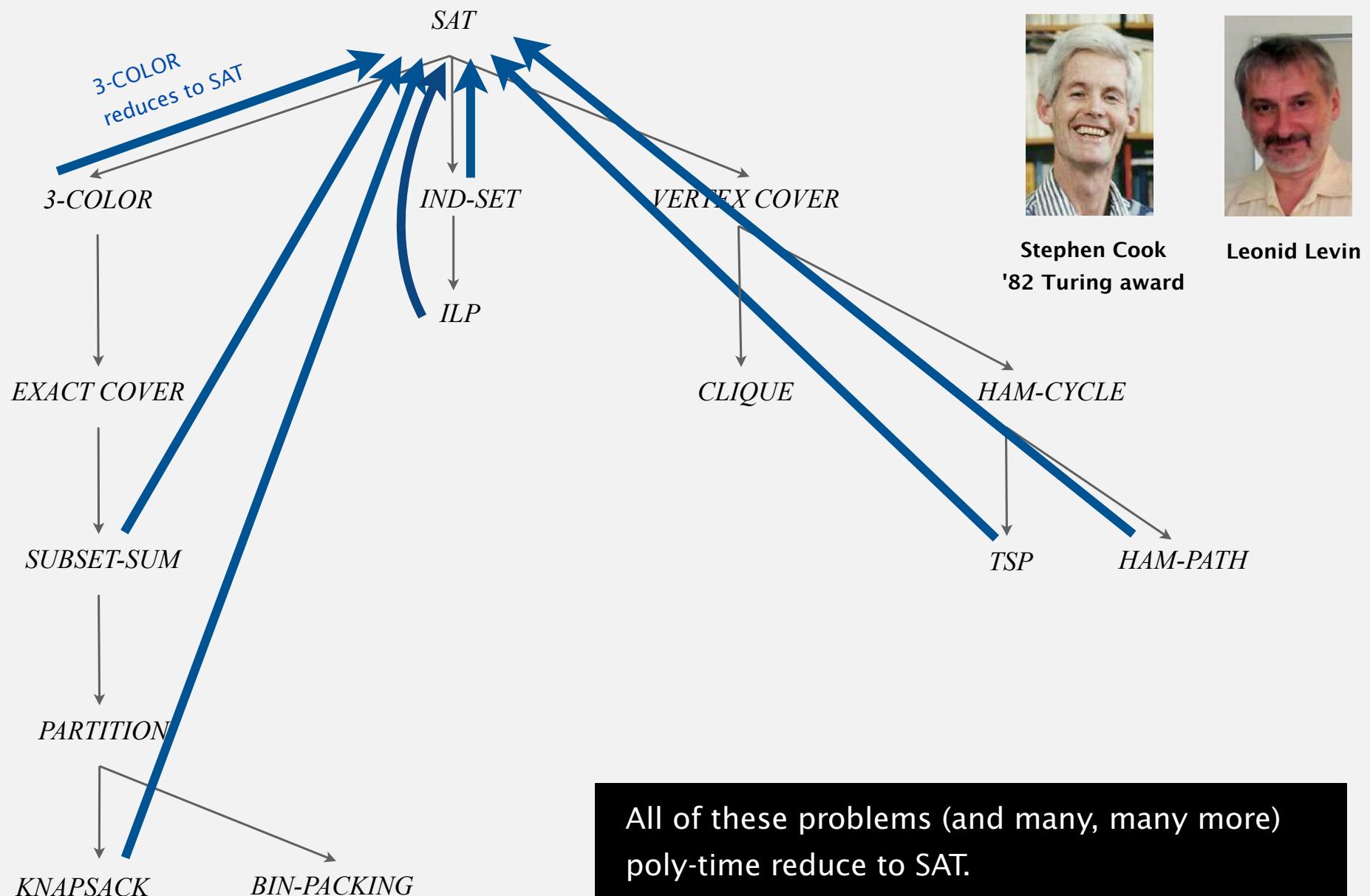
every NP problem is a
SAT problem in disguise

Corollary. Poly-time algorithm for SAT iff $P = NP$.

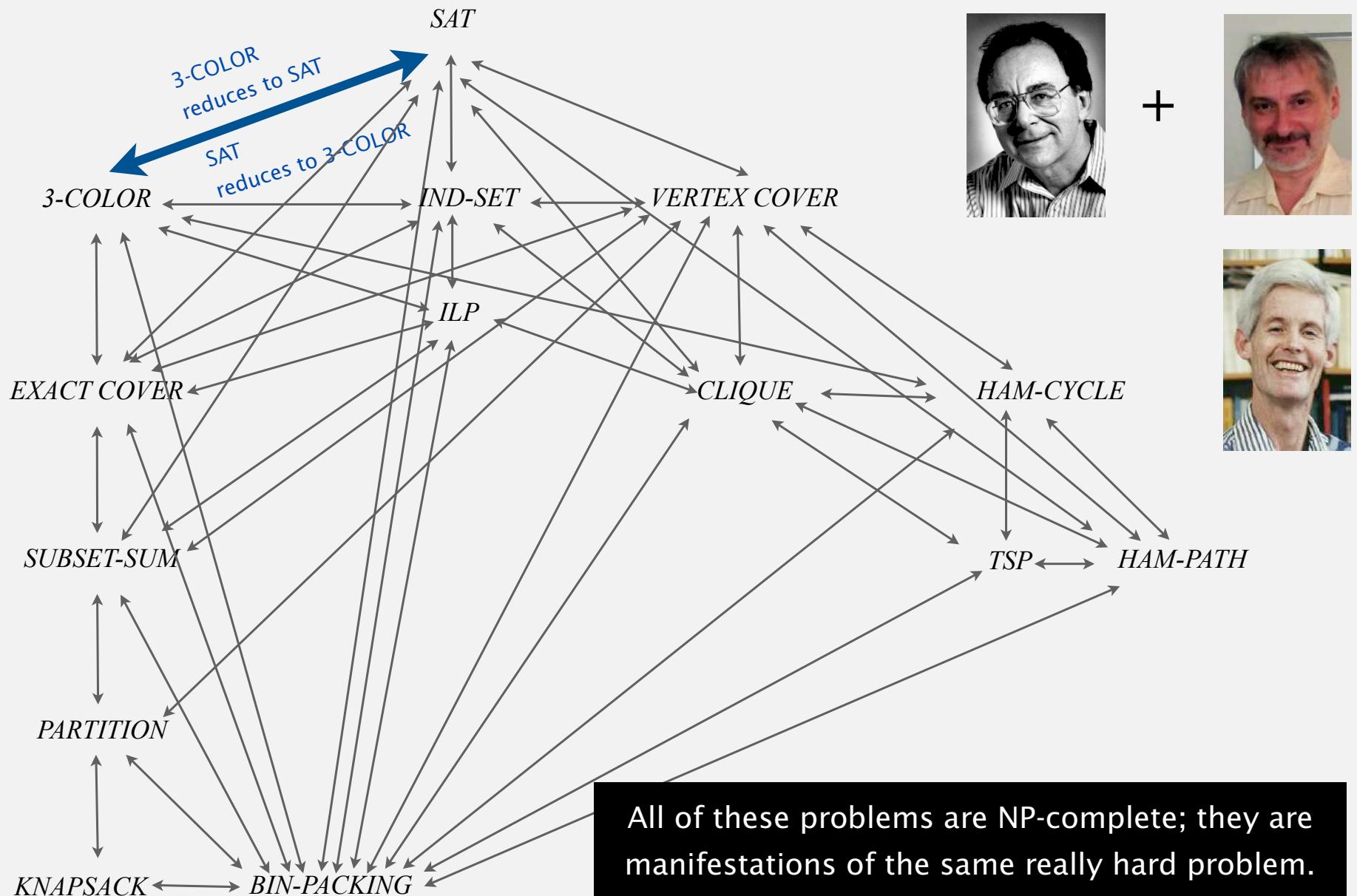
You NP-complete me



Implications of Cook-Levin theorem



Implications of Karp + Cook-Levin



Implications of NP-Completeness

Implication. [SAT captures difficulty of whole class NP]

- Poly-time algorithm for SAT iff $P = NP$.
- No poly-time algorithm for some NP problem \Rightarrow none for SAT.

Remark. Can replace SAT with any of Karp's problems.

Proving a problem NP-complete guides scientific inquiry.

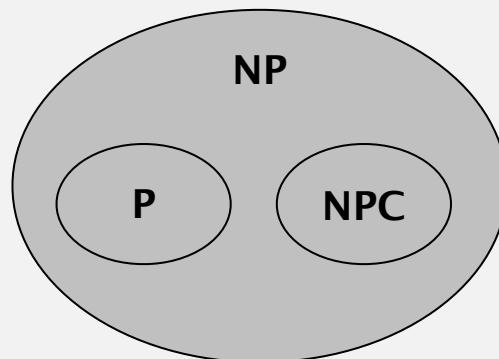
- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager finds closed form solution to 2D version in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: 3D-ISING proved NP-complete.

search for closed formula appears doomed

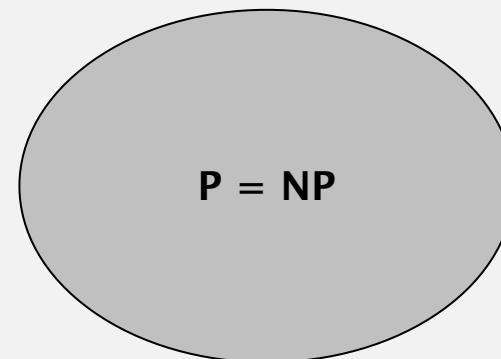
a holy grail of statistical mechanics

Two worlds (more detail)

Overwhelming consensus (still). $P \neq NP$.



$P \neq NP$



$P = NP$

Why we believe $P \neq NP$.

“ We admire Wiles' proof of Fermat's last theorem, the scientific theories of Newton, Einstein, Darwin, Watson and Crick, the design of the Golden Gate bridge and the Pyramids, precisely because they seem to require a leap which cannot be made by everyone, let alone a by simple mechanical device. ” — Avi Wigderson

Summary

P. Class of search problems solvable in poly-time.

NP. Class of all search problems, some of which seem wickedly hard.

NP-complete. Hardest problems in NP.

Intractable. Problem with no poly-time algorithm.

Many fundamental problems are NP-complete.

- SAT, ILP, HAMILTON-PATH, ...
- 3D-ISING, ...

Use theory a guide:

- A poly-time algorithm for an NP-complete problem would be a stunning breakthrough (a proof that $P = NP$).
- You will confront NP-complete problems in your career.
- Safe to assume that $P \neq NP$ and that such problems are intractable.
- Identify these situations and proceed accordingly.

Princeton CS Building, West Wall, Circa 2001



Princeton CS Building, West Wall, Circa 2001

A photograph of a red brick wall with binary digits (0s and 1s) painted on it. The digits are arranged in a grid pattern, representing the ASCII values of the characters 'P', '=', 'N', 'P', and '?'. The grid consists of 5 columns and 8 rows of digits.

char	ASCII	binary
P	80	1010000
=	61	0111101
N	78	1001110
P	80	1010000
?	63	0111111

char	ASCII	binary
P	80	1010000
=	61	0111101
N	78	1001110
P	80	1010000
?	63	0111111

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ ***NP-completeness***
- ▶ *coping with intractability*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ ***coping with intractability***

Exploiting intractability

Modern cryptography.

- Ex. Send your credit card to Amazon.
- Ex. Digitally sign an e-document.
- Enables freedom of privacy, speech, press, political association.

RSA cryptosystem.

- To use: multiply two n -bit integers. [poly-time]
- To break: factor a 2 n -bit integer. [unlikely poly-time]

Multiply = EASY

$$23 \times 67 \longleftrightarrow 1,541$$

Factor = HARD

Exploiting intractability

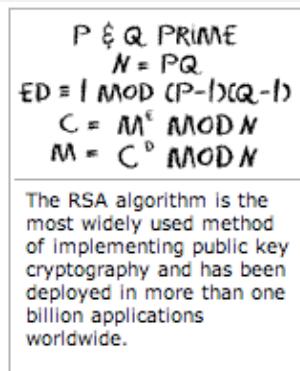
Challenge. Factor this number.

740375634795617128280467960974295731425931888892312890849362
326389727650340282662768919964196251178439958943305021275853
701189680982867331732731089309005525051168770632990723963807
86710086096962537934650563796359

RSA-704

(\$30,000 prize if you can factor)

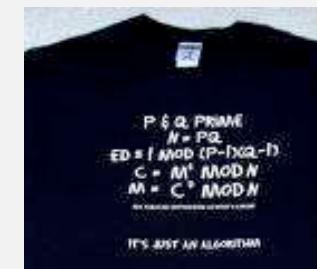
Can't do it? Create a company based on the difficulty of factoring.



RSA algorithm



RSA sold
for \$2.1 billion



or design a t-shirt

Exploiting intractability

FACTOR. Given an n -bit integer x , find a nontrivial factor.

Q. What is complexity of FACTOR?

A. In NP, but not known (or believed) to be in P or NP-complete.

Q. What if $P = NP$?

A. Poly-time algorithm for factoring; modern e-economy collapses.

Proposition. [Shor 1994] Can factor an n -bit integer in n^3 steps on a "quantum computer."

Q. Do we still believe the extended Church-Turing thesis???



Coping with intractability

Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Special cases may be tractable.

- Ex: Linear time algorithm for 2-SAT. ← at most two variables per equation
- Ex: Linear time algorithm for Horn-SAT. ← at most one un-negated variable per equation

Coping with intractability

Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Develop a heuristic, and hope it produces a good solution.

- No guarantees on quality of solution.
- Ex. TSP assignment heuristics.
- Ex. Metropolis algorithm, simulating annealing, genetic algorithms.

Approximation algorithm. Find solution of provably good quality.

- Ex. MAX-3SAT: provably satisfy 87.5% as many clauses as possible.



but if you can guarantee to satisfy 87.51% as many clauses as possible in poly-time, then $P = NP$!

Coping with intractability

Relax one of desired features.

- Solve arbitrary instances of the problem.
- Solve the problem to optimality.
- Solve the problem in poly-time.

Complexity theory deals with worst case behavior.

- Instance(s) you want to solve may be "easy."
- Chaff solves real-world SAT instances with ~ 10K variable.

Chaff: Engineering an Efficient SAT Solver

Matthew W. Moskewicz

Department of EECS
UC Berkeley

moskewcz@alumni.princeton.edu

Conor F. Madigan

Department of EECS
MIT

cmadigan@mit.edu

Ying Zhao, Lintao Zhang, Sharad Malik

Department of Electrical Engineering
Princeton University

{yingzhao, lintaoz, sharad}@ee.princeton.edu

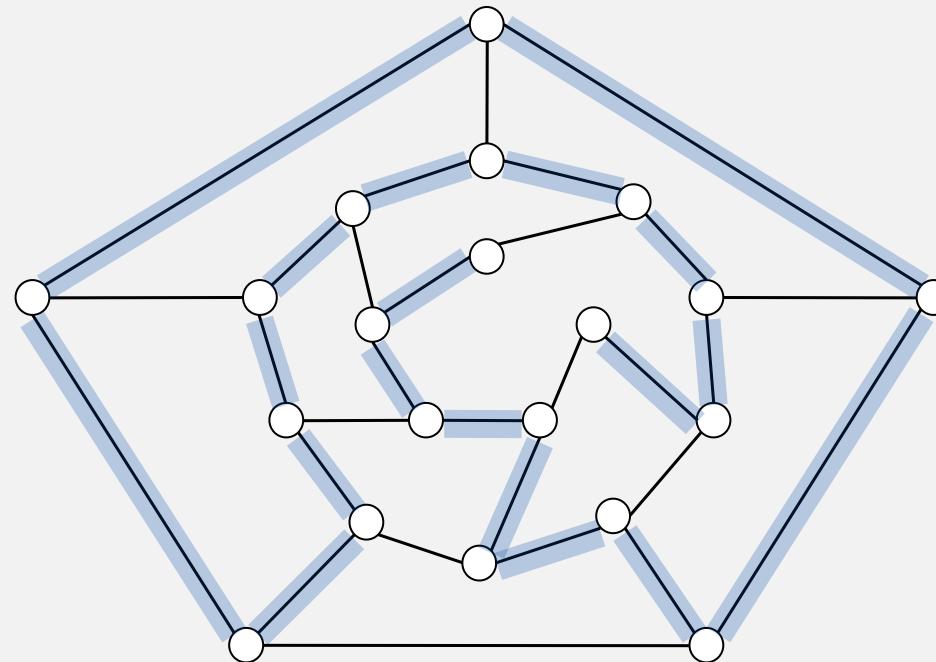
ABSTRACT

Boolean Satisfiability is probably the most studied of combinatorial optimization/search problems. Significant effort has been devoted to trying to provide practical solutions to this problem for problem instances encountered in a range of applications in Electronic Design Automation (EDA), as well as in Artificial Intelligence (AI). This study has culminated in the

Many publicly available SAT solvers (e.g. GRASP [8], POSIT [5], SATO [13], rel_sat [2], WalkSAT [9]) have been developed, most employing some combination of two main strategies: the Davis-Putnam (DP) backtrack search and heuristic local search. Heuristic local search techniques are not guaranteed to be complete (i.e. they are not guaranteed to find a satisfying assignment if one exists or prove unsatisfiability); as a

Hamilton path

Goal. Find a simple path that visits every vertex exactly once.



visit every edge exactly once



Remark. Euler path easy, but Hamilton path is NP-complete.

Hamilton path: Java implementation

```
public class HamiltonPath
{
    private boolean[] marked;      // vertices on current path
    private int count = 0;         // number of Hamiltonian paths

    public HamiltonPath(Graph G)
    {
        marked = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            dfs(G, v, 1);
    }

    private void dfs(Graph G, int v, int depth)
    {
        marked[v] = true;
        if (depth == G.V()) count++;

        found one →
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w, depth+1); ← backtrack if w is
                                                       already part of path
        }                                     marked[v] = false; ← clean up
    }
}
```

length of current path
(depth of recursion)

backtrack if w is
already part of path

The longest path



*Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!*

*If you said P is NP tonight,
There would still be papers left to write.
I have a weakness;
I'm addicted to completeness,
And I keep searching for the longest path.*

*The algorithm I would like to see
Is of polynomial degree.
But it's elusive:
Nobody has found conclusive
Evidence that we can find a longest path.*

*I have been hard working for so long.
I swear it's right, and he marks it wrong.
Some how I'll feel sorry when it's done: GPA 2.1
Is more than I hope for.*

*Garey, Johnson, Karp and other men (and women)
Tried to make it order $N \log N$.
Am I a mad fool
If I spend my life in grad school,
Forever following the longest path?*

*Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path!
Woh-oh-oh-oh, find the longest path.*

**Written by Dan Barrett in 1988 while a student
at Johns Hopkins during a difficult algorithms take-home final**

That's all, folks: keep searching!



**The world's longest path (Sendero de Chile): 9,700 km.
(originally scheduled for completion in 2010; now delayed until 2038)**

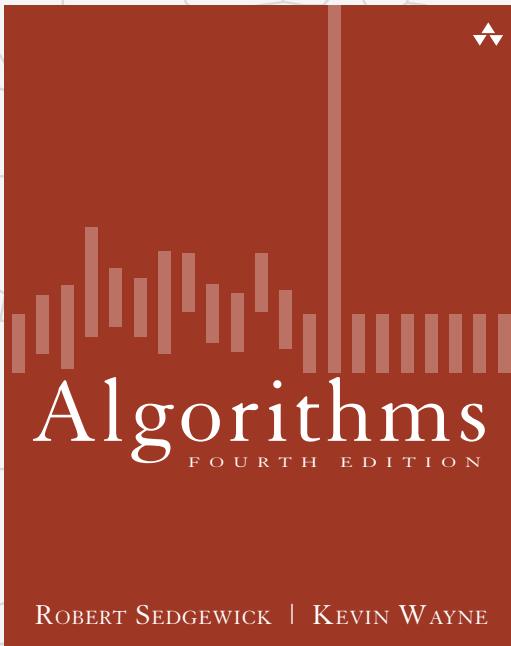
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ ***coping with intractability***

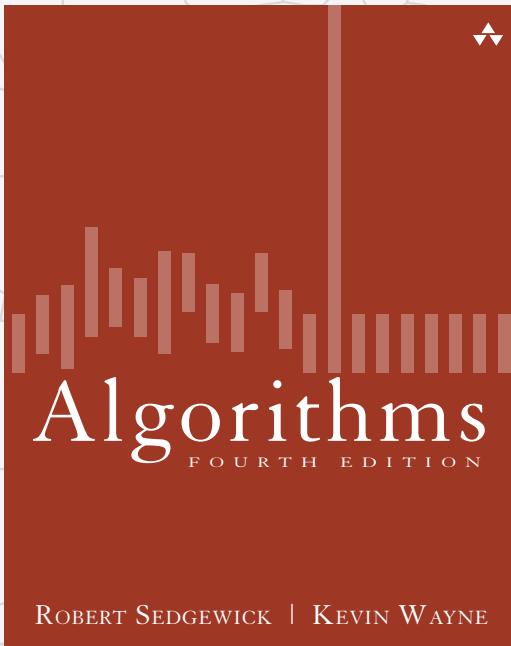


ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

6.6 INTRACTABILITY

- ▶ *introduction*
- ▶ *search problems*
- ▶ *P vs. NP*
- ▶ *classifying problems*
- ▶ *NP-completeness*
- ▶ *coping with intractability*



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

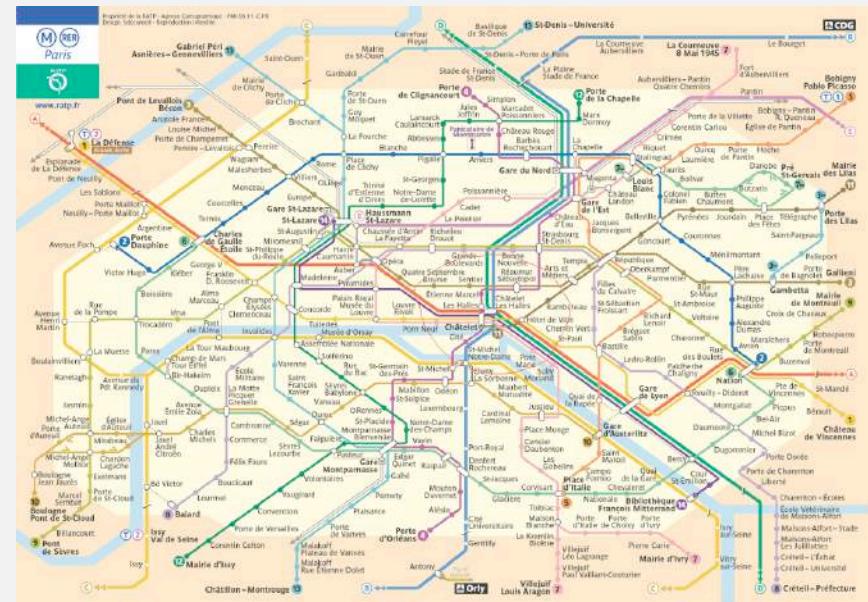
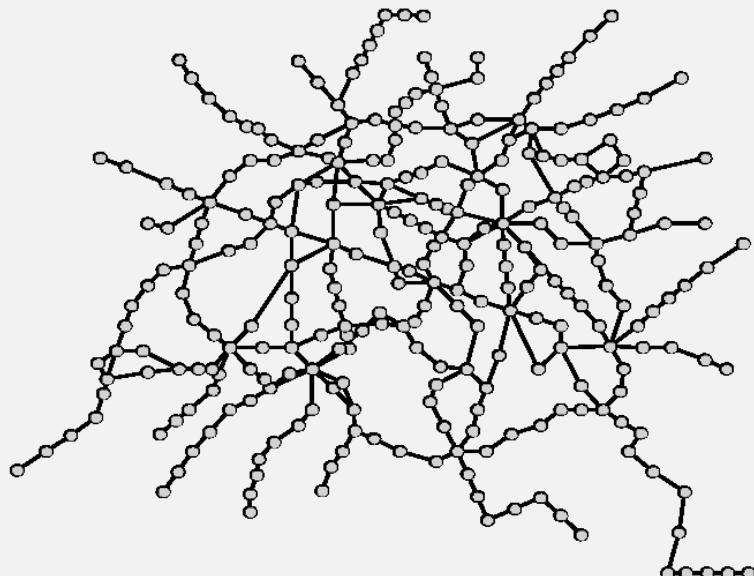
- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Undirected graphs

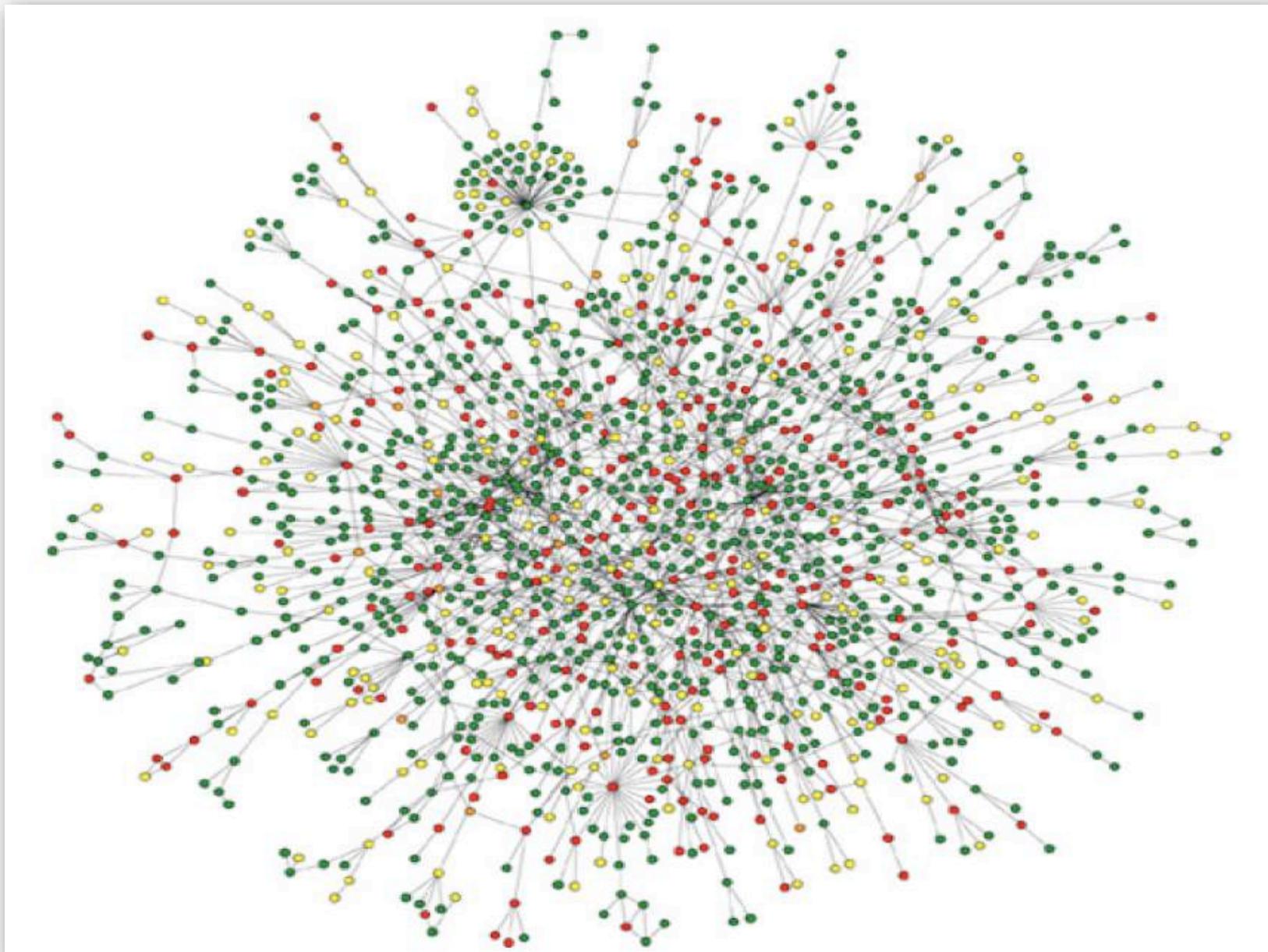
Graph. Set of **vertices** connected pairwise by **edges**.

Why study graph algorithms?

- Thousands of practical applications.
- Hundreds of graph algorithms known.
- Interesting and broadly useful abstraction.
- Challenging branch of computer science and discrete math.

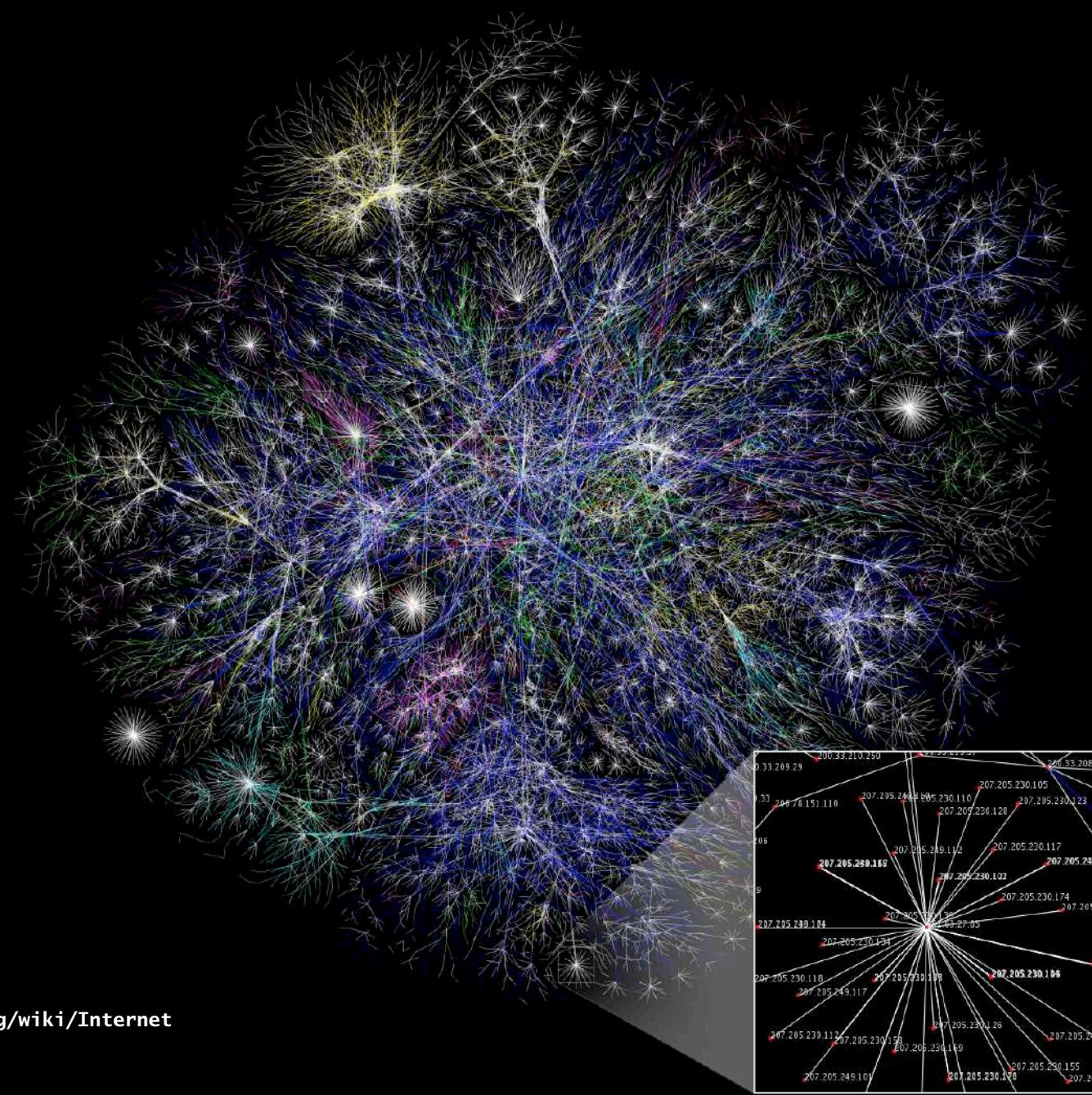


Protein-protein interaction network



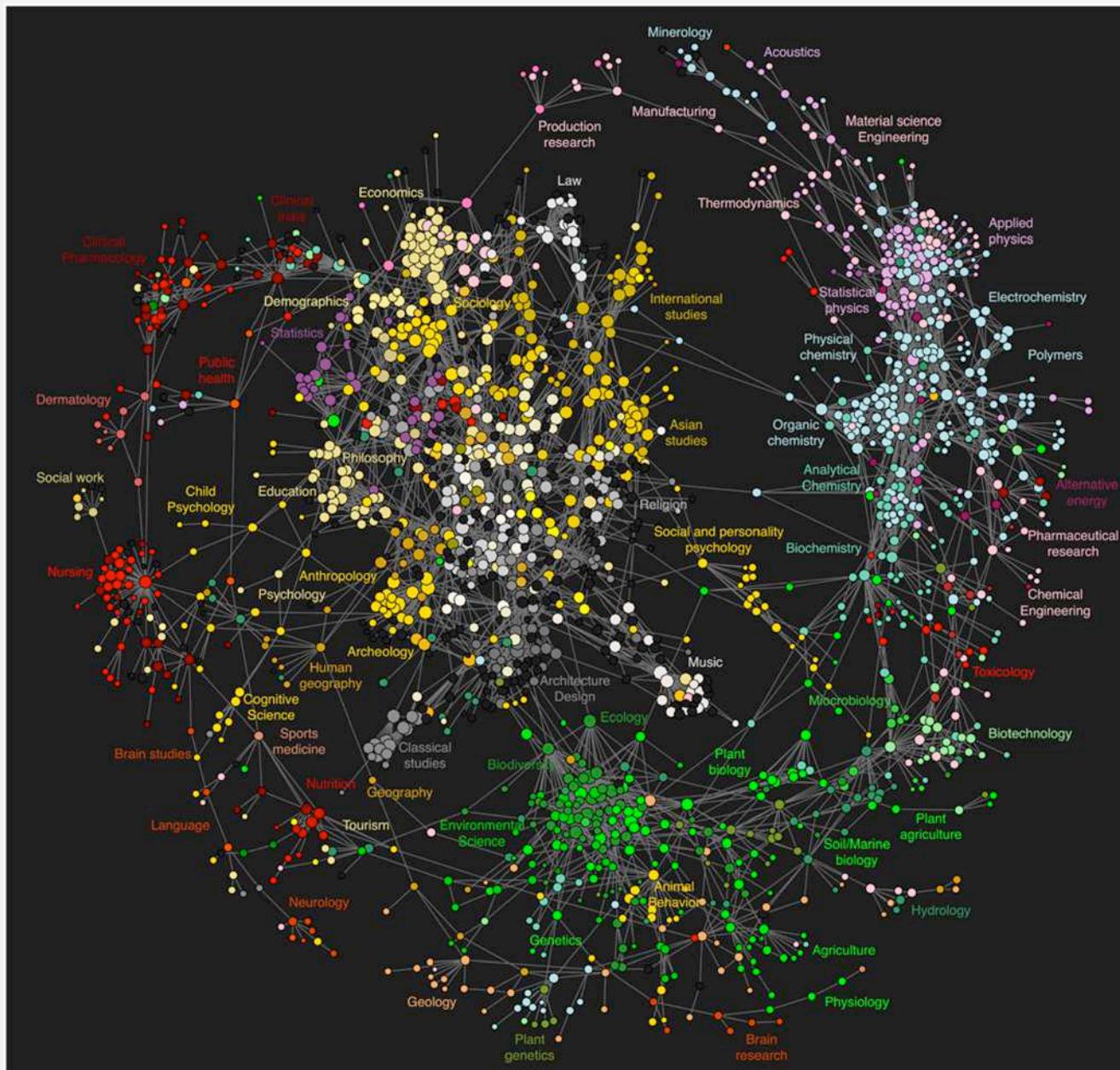
Reference: Jeong et al, Nature Review | Genetics

The Internet as mapped by the Opte Project



<http://en.wikipedia.org/wiki/Internet>

Map of science clickstreams

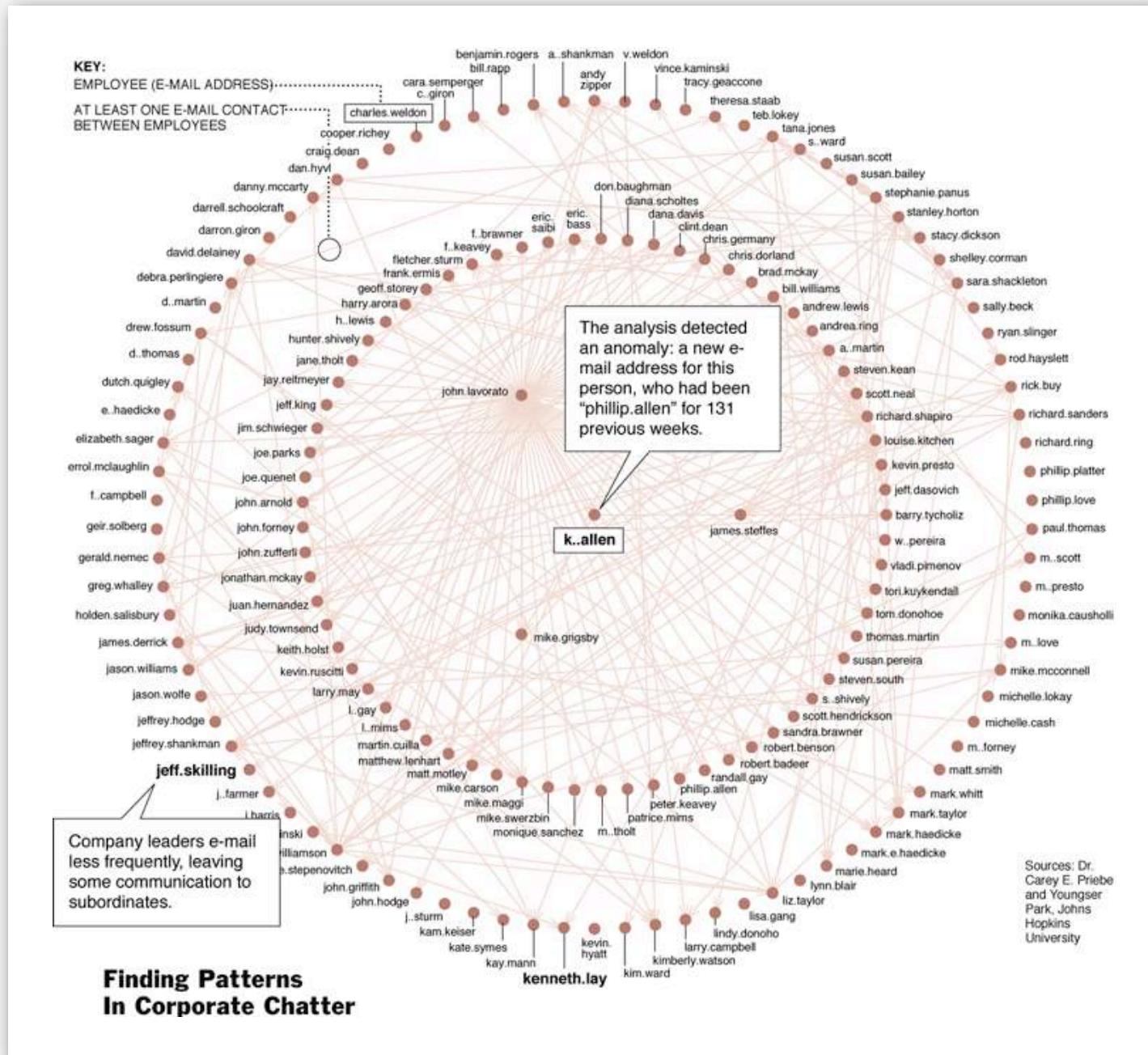


10 million Facebook friends

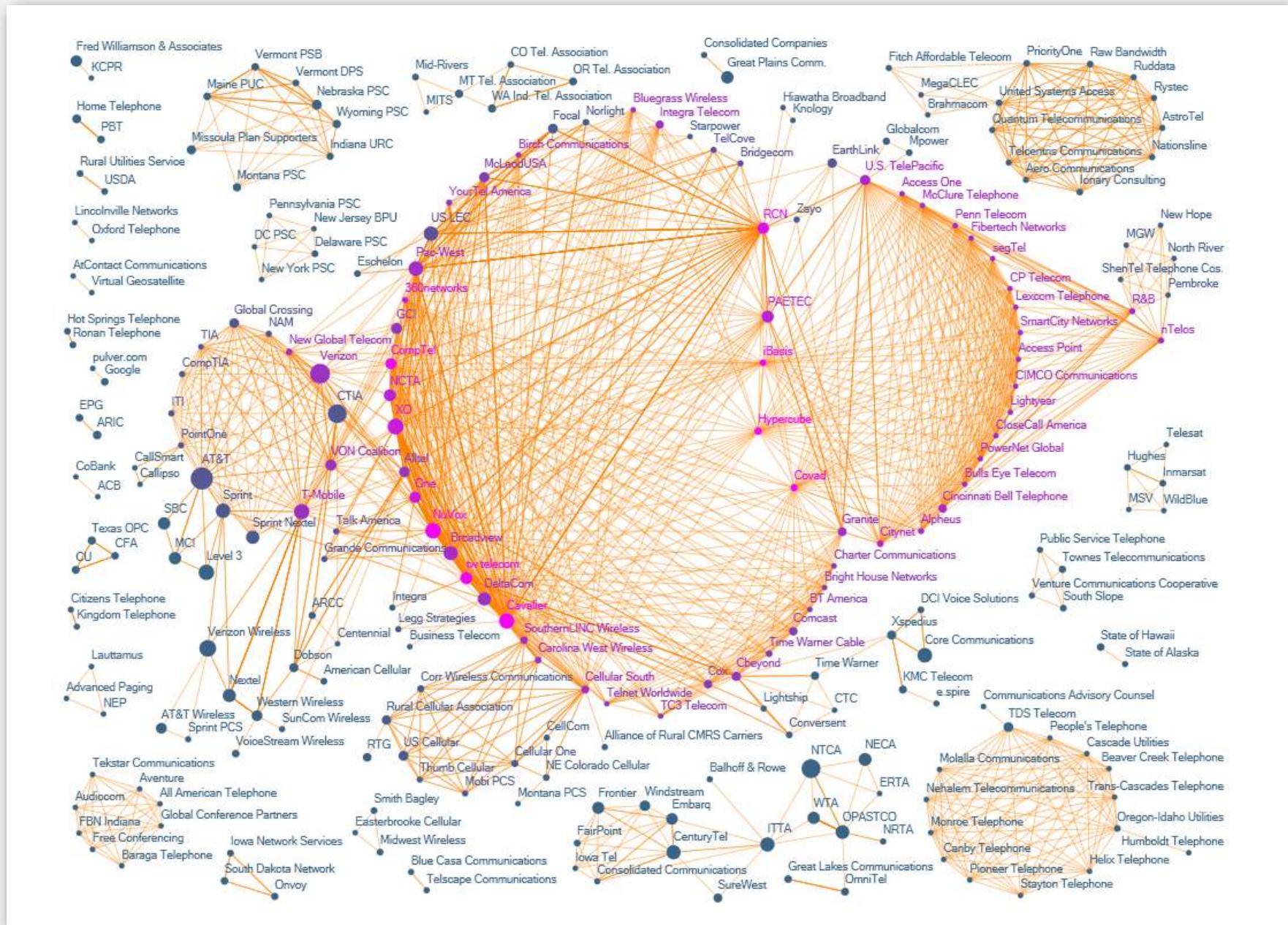


"Visualizing Friendships" by Paul Butler

One week of Enron emails



The evolution of FCC lobbying coalitions



"The Evolution of FCC Lobbying Coalitions" by Pierre de Vries in JoSS Visualization Symposium 2010

Framingham heart study

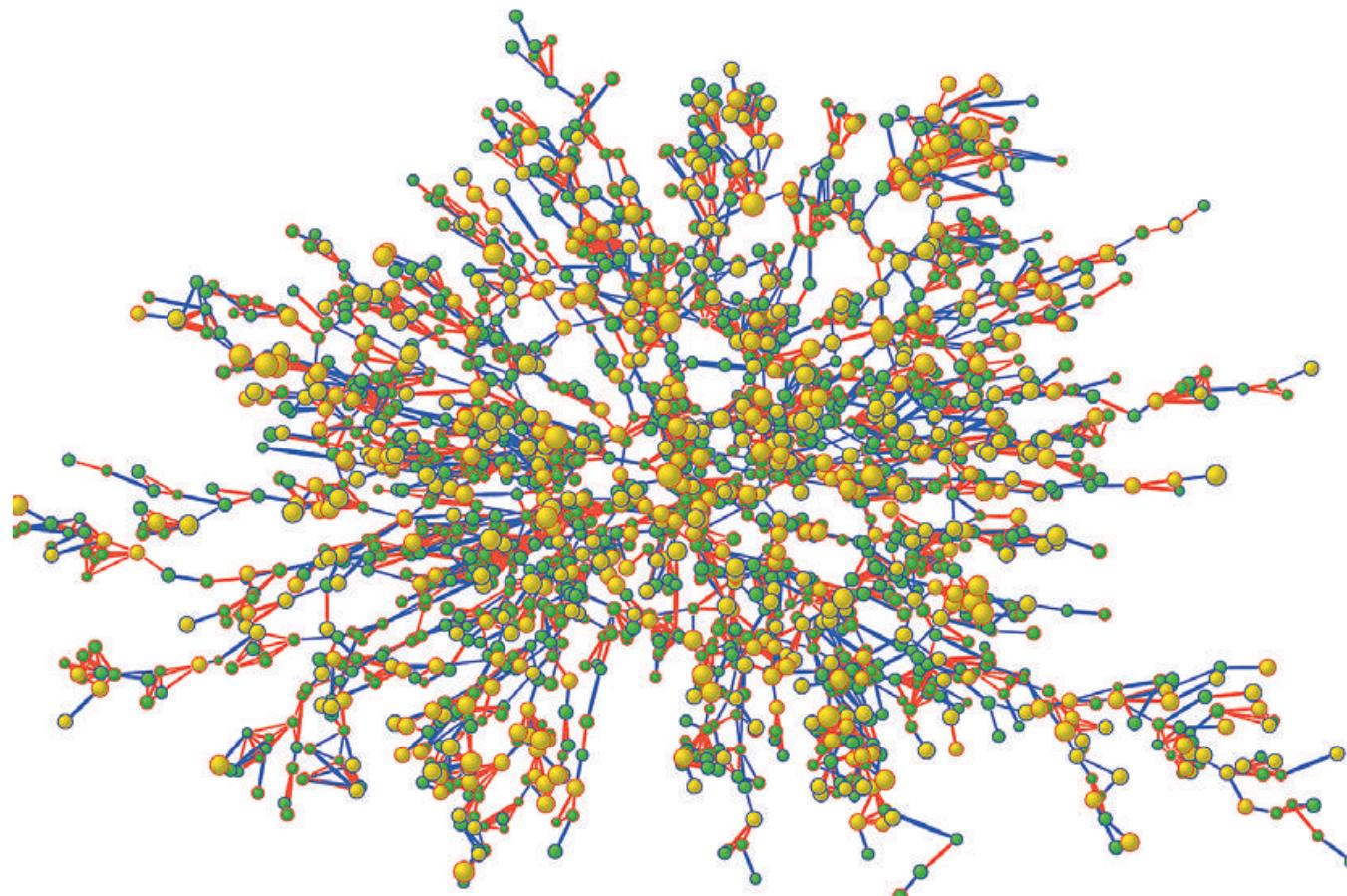


Figure 1. Largest Connected Subcomponent of the Social Network in the Framingham Heart Study in the Year 2000.

Each circle (node) represents one person in the data set. There are 2200 persons in this subcomponent of the social network. Circles with red borders denote women, and circles with blue borders denote men. The size of each circle is proportional to the person's body-mass index. The interior color of the circles indicates the person's obesity status: yellow denotes an obese person (body-mass index, ≥ 30) and green denotes a nonobese person. The colors of the ties between the nodes indicate the relationship between them: purple denotes a friendship or marital tie and orange denotes a familial tie.

Graph applications

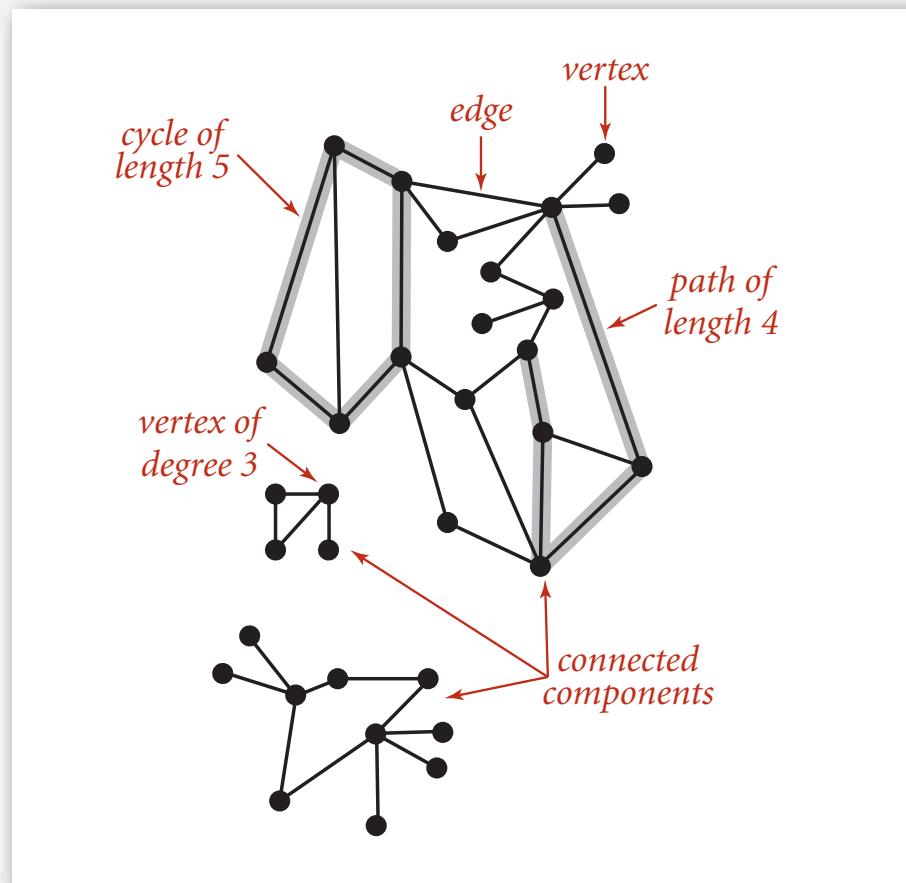
graph	vertex	edge
communication	telephone, computer	fiber optic cable
circuit	gate, register, processor	wire
mechanical	joint	rod, beam, spring
financial	stock, currency	transactions
transportation	street intersection, airport	highway, airway route
internet	class C network	connection
game	board position	legal move
social relationship	person, actor	friendship, movie cast
neural network	neuron	synapse
protein network	protein	protein-protein interaction
chemical compound	molecule	bond

Graph terminology

Path. Sequence of vertices connected by edges.

Cycle. Path whose first and last vertices are the same.

Two vertices are **connected** if there is a path between them.



Some graph-processing problems

Path. Is there a path between s and t ?

Shortest path. What is the shortest path between s and t ?

Cycle. Is there a cycle in the graph?

Euler tour. Is there a cycle that uses each edge exactly once?

Hamilton tour. Is there a cycle that uses each vertex exactly once.

Connectivity. Is there a way to connect all of the vertices?

MST. What is the best way to connect all of the vertices?

Biconnectivity. Is there a vertex whose removal disconnects the graph?

Planarity. Can you draw the graph in the plane with no crossing edges?

Graph isomorphism. Do two adjacency lists represent the same graph?

Challenge. Which of these problems are easy? difficult? intractable?

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

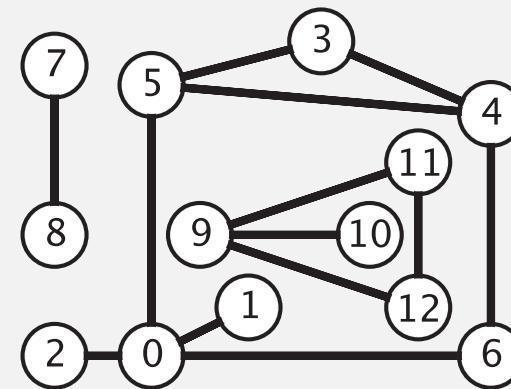
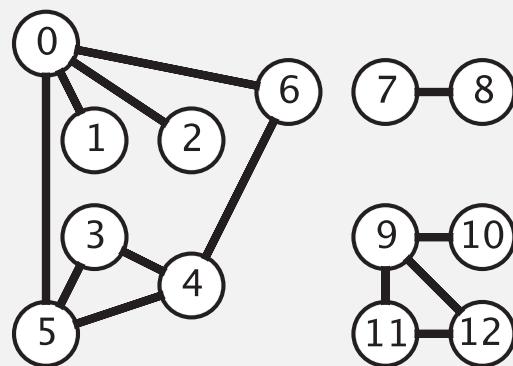
<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Graph representation

Graph drawing. Provides intuition about the structure of the graph.



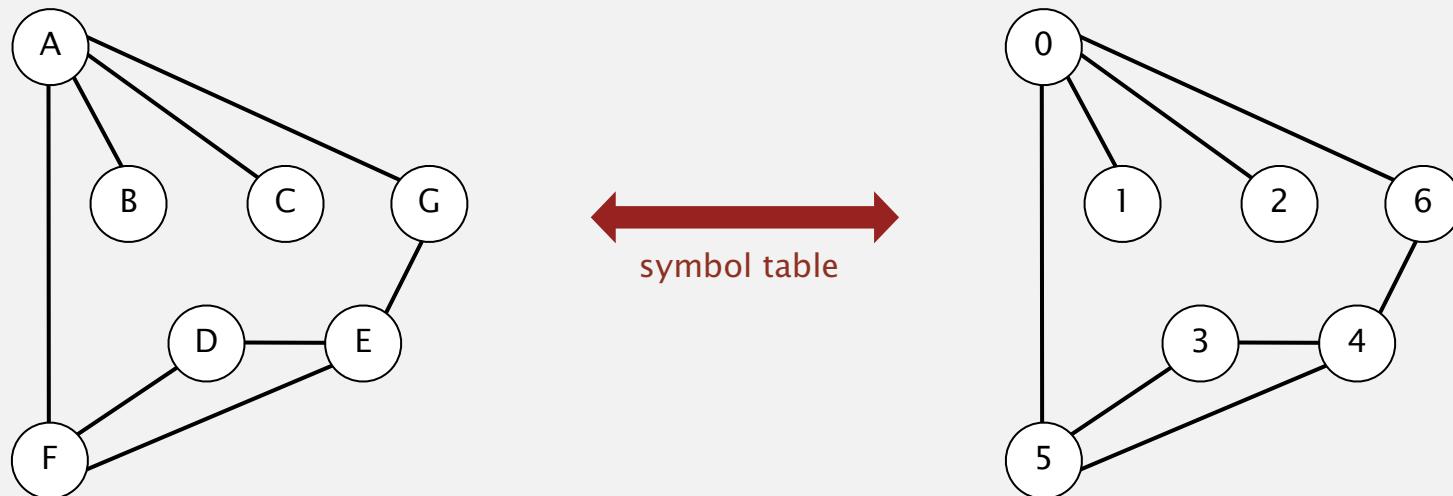
two drawings of the same graph

Caveat. Intuition can be misleading.

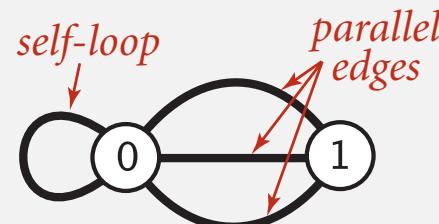
Graph representation

Vertex representation.

- This lecture: use integers between 0 and $V - 1$.
- Applications: convert between names and integers with symbol table.



Anomalies.



Graph API

```
public class Graph
```

```
    Graph(int V)
```

create an empty graph with V vertices

```
    Graph(In in)
```

create a graph from input stream

```
    void addEdge(int v, int w)
```

add an edge v-w

```
    Iterable<Integer> adj(int v)
```

vertices adjacent to v

```
    int V()
```

number of vertices

```
    int E()
```

number of edges

```
    String toString()
```

string representation

```
In in = new In(args[0]);  
Graph G = new Graph(in);
```

read graph from
input stream

```
for (int v = 0; v < G.V(); v++)  
    for (int w : G.adj(v))  
        StdOut.println(v + "-" + w);
```

print out each
edge (twice)

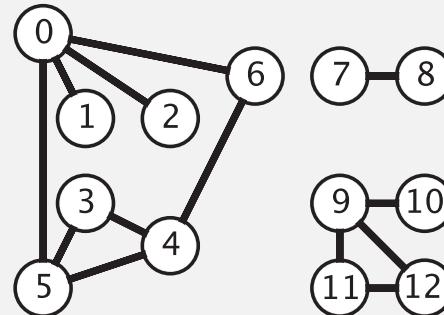
Graph API: sample client

Graph input format.

tinyG.txt

$V \rightarrow 13$
 $E \leftarrow 13$

13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3



% java Test tinyG.txt

0-6
0-2
0-1
0-5
1-0
2-0
3-5
3-4
...
12-11
12-9

```
In in = new In(args[0]);
Graph G = new Graph(in);

for (int v = 0; v < G.V(); v++)
    for (int w : G.adj(v))
        StdOut.println(v + " - " + w);
```

read graph from
input stream

print out each
edge (twice)

Typical graph-processing code

```
public static int degree(Graph G, int v)
{
    int degree = 0;
    for (int w : G.adj(v)) degree++;
    return degree;
}

public static int maxDegree(Graph G)
{
    int max = 0;
    for (int v = 0; v < G.V(); v++)
        if (degree(G, v) > max)
            max = degree(G, v);
    return max;
}

public static double averageDegree(Graph G)
{   return 2.0 * G.E() / G.V(); }

public static int numberOfSelfLoops(Graph G)
{
    int count = 0;
    for (int v = 0; v < G.V(); v++)
        for (int w : G.adj(v))
            if (v == w) count++;
    return count/2; // each edge counted twice
}
```

compute the degree of v

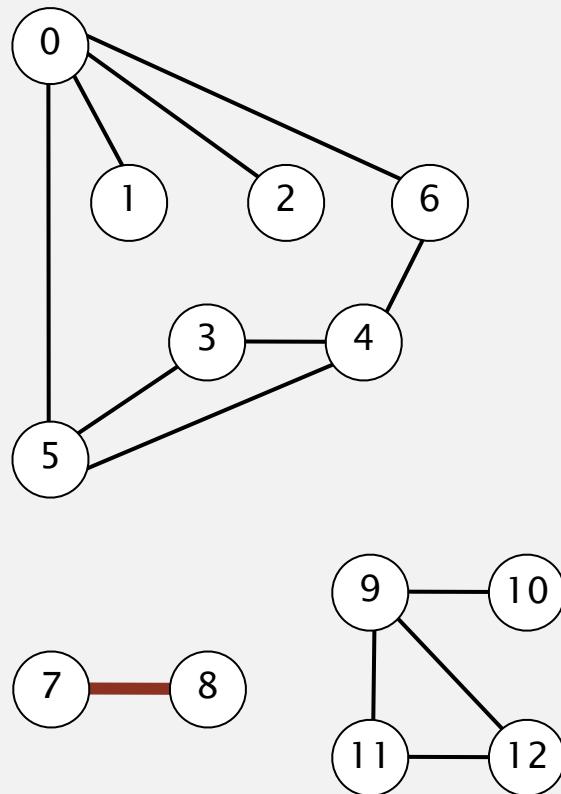
compute maximum degree

compute average degree

count self-loops

Set-of-edges graph representation

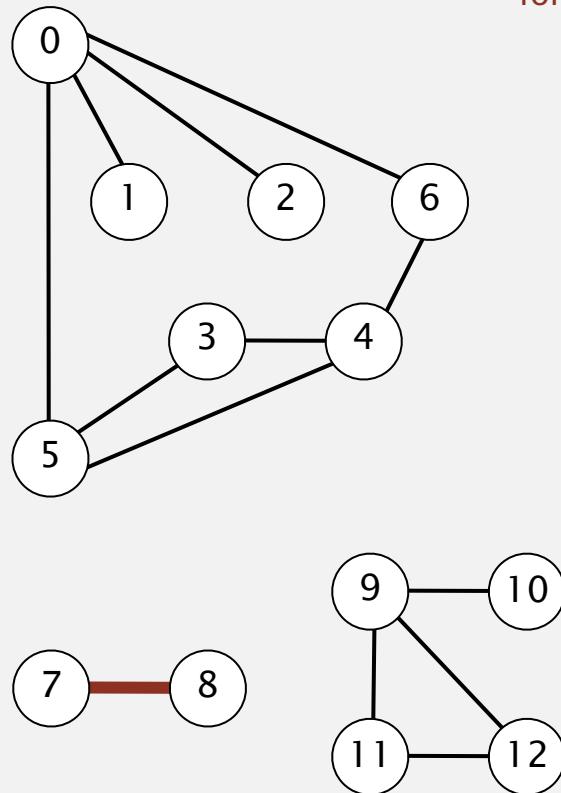
Maintain a list of the edges (linked list or array).



0	1
0	2
0	5
0	6
3	4
3	5
4	5
4	6
7	8
9	10
9	11
9	12
11	12

Adjacency-matrix graph representation

Maintain a two-dimensional V -by- V boolean array;
for each edge $v-w$ in graph: $\text{adj}[v][w] = \text{adj}[w][v] = \text{true}$.

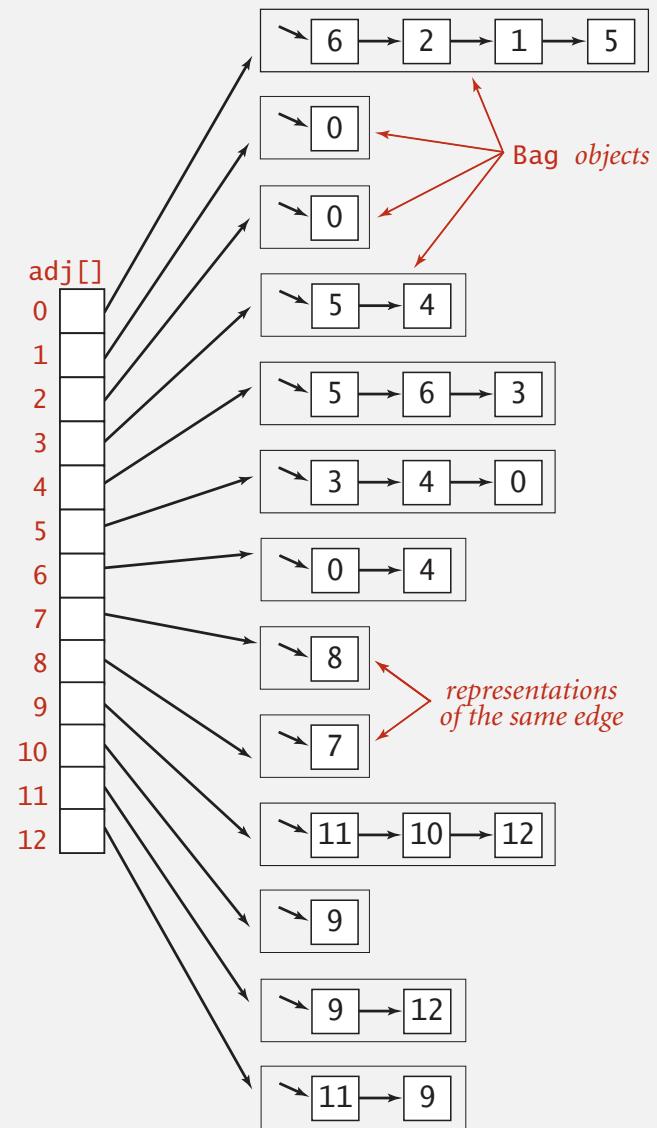
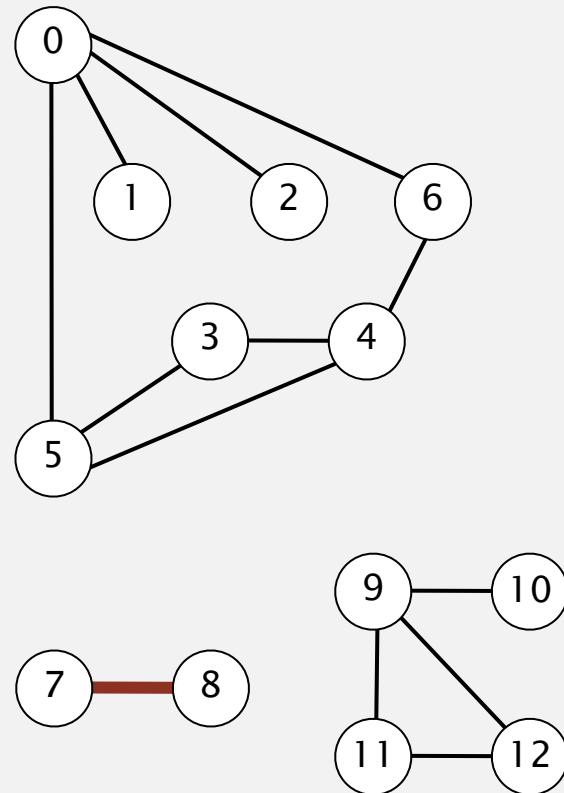


two entries
for each edge

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	1	0	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

Adjacency-list graph representation

Maintain vertex-indexed array of lists.



Adjacency-list graph representation: Java implementation

```
public class Graph
{
    private final int V;
    private Bag<Integer>[] adj;
```

adjacency lists
(using Bag data type)

```
public Graph(int V)
{
    this.V = V;
    adj = (Bag<Integer>[]) new Bag[V];
    for (int v = 0; v < V; v++)
        adj[v] = new Bag<Integer>();
}
```

create empty graph
with V vertices

```
public void addEdge(int v, int w)
{
    adj[v].add(w);
    adj[w].add(v);
}
```

add edge v-w
(parallel edges and
self-loops allowed)

```
public Iterable<Integer> adj(int v)
{ return adj[v]; }
```

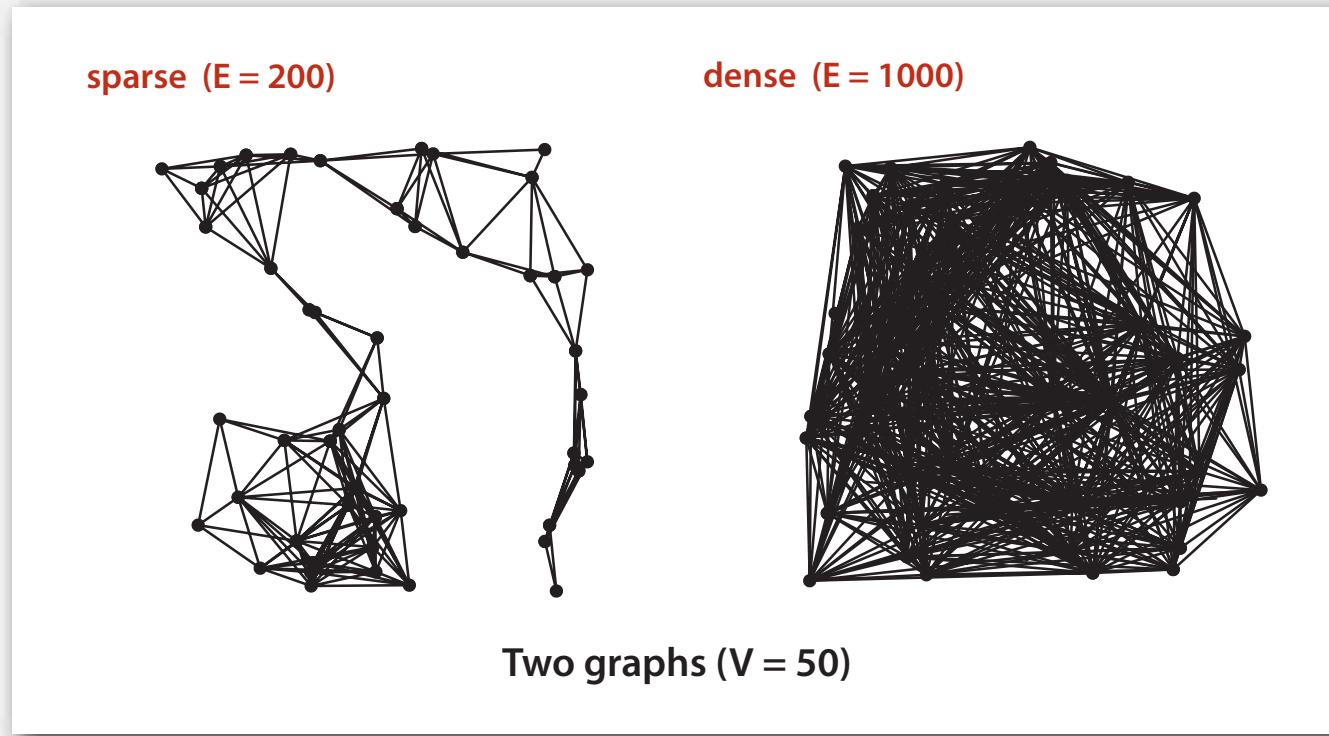
iterator for vertices adjacent to v

Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v .
- Real-world graphs tend to be **sparse**.

huge number of vertices,
small average vertex degree



Graph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices adjacent to v .
- Real-world graphs tend to be **sparse**.

huge number of vertices,
small average vertex degree

representation	space	add edge	edge between v and w ?	iterate over vertices adjacent to v ?
list of edges	E	1	E	E
adjacency matrix	V^2	1^*	1	V
adjacency lists	$E + V$	1	$\text{degree}(v)$	$\text{degree}(v)$

* disallows parallel edges

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

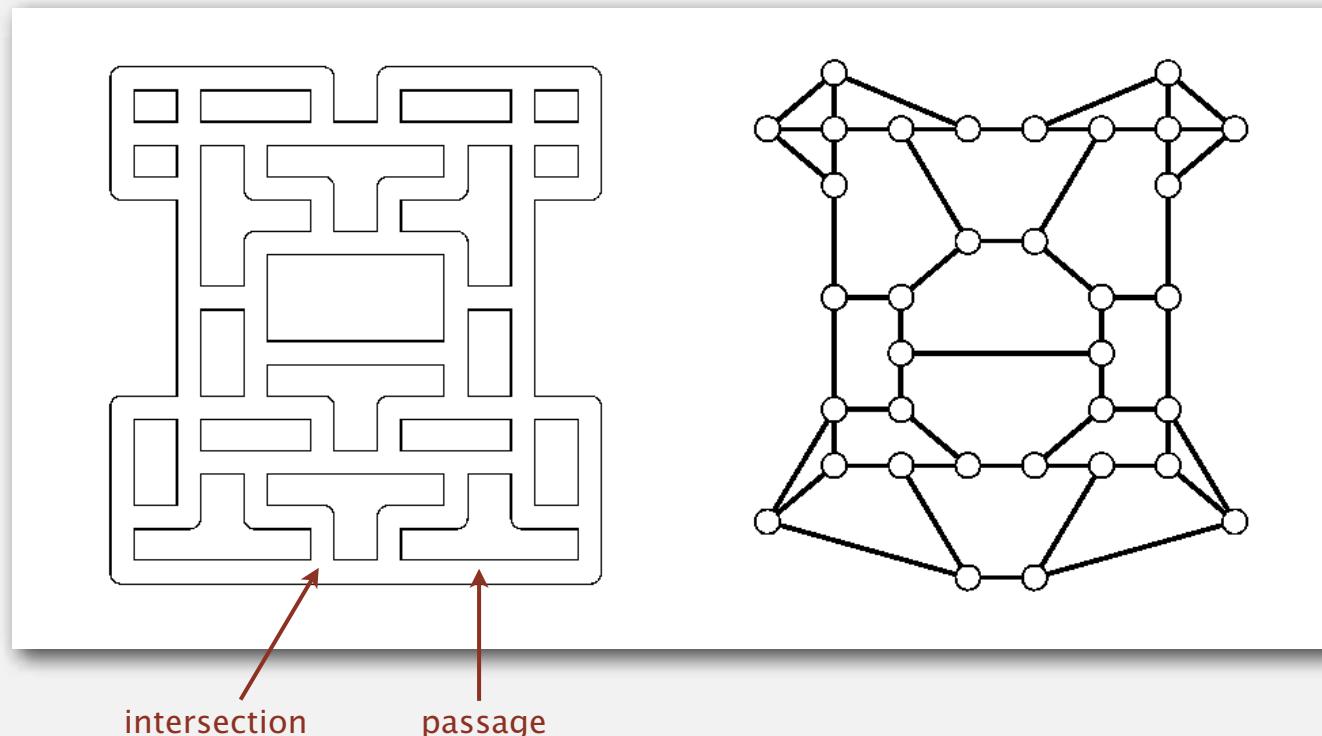
4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Maze exploration

Maze graph.

- Vertex = intersection.
- Edge = passage.

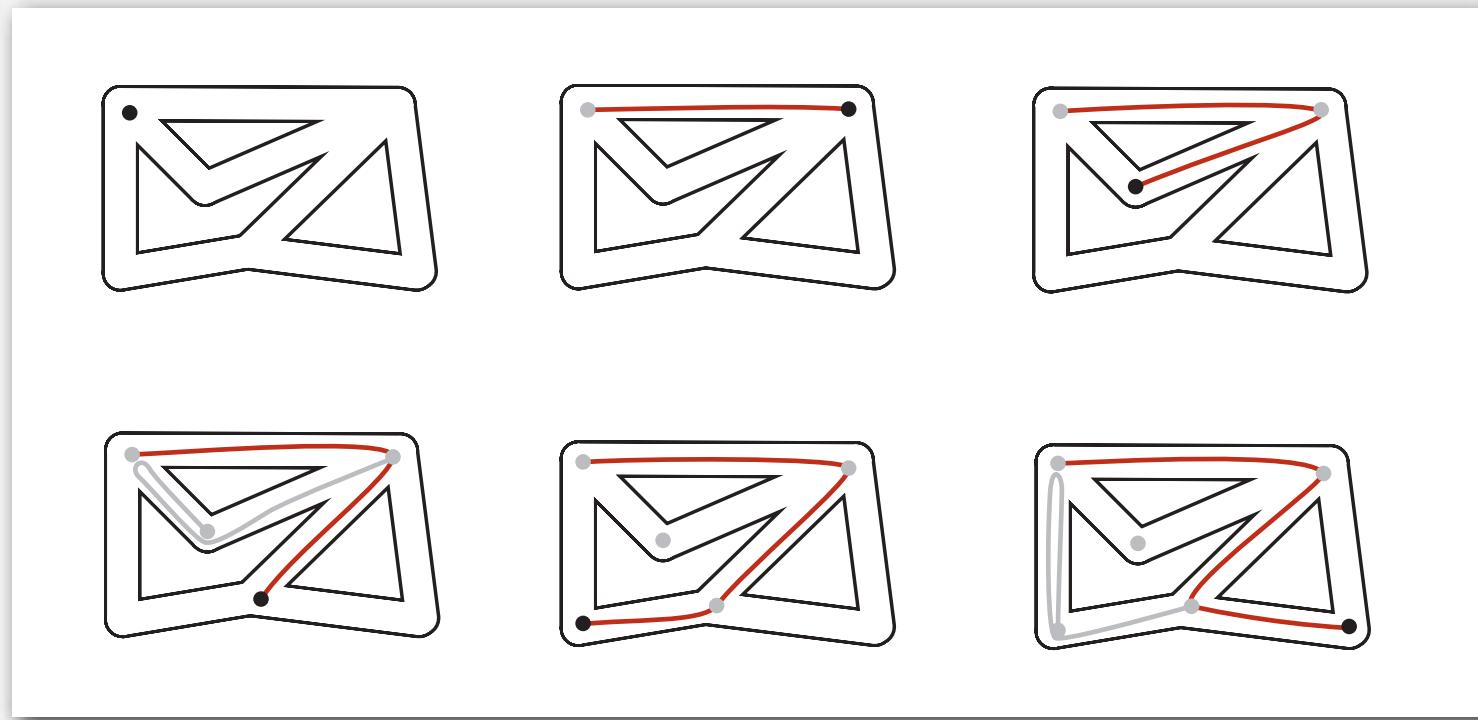


Goal. Explore every intersection in the maze.

Trémaux maze exploration

Algorithm.

- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.



Trémaux maze exploration

Algorithm.

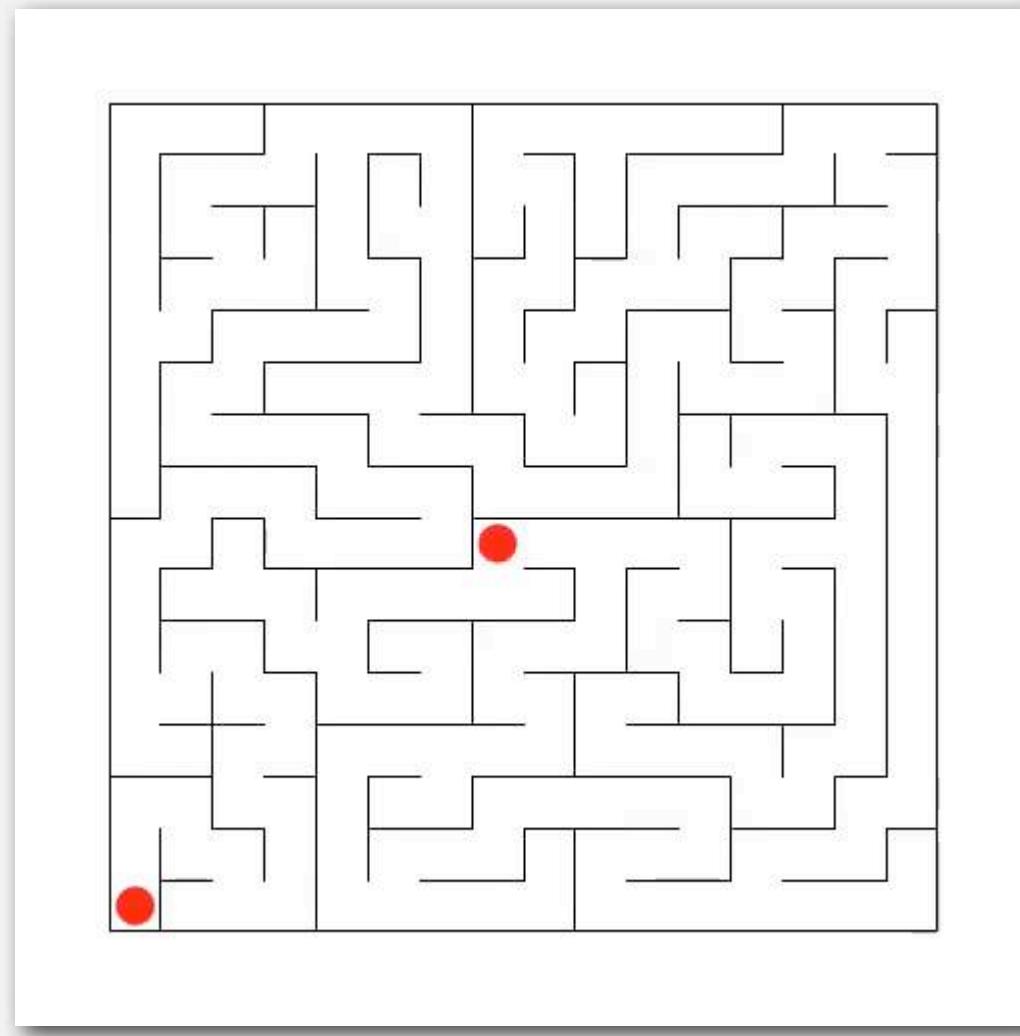
- Unroll a ball of string behind you.
- Mark each visited intersection and each visited passage.
- Retrace steps when no unvisited options.

First use? Theseus entered Labyrinth to kill the monstrous Minotaur; Ariadne instructed Theseus to use a ball of string to find his way back out.

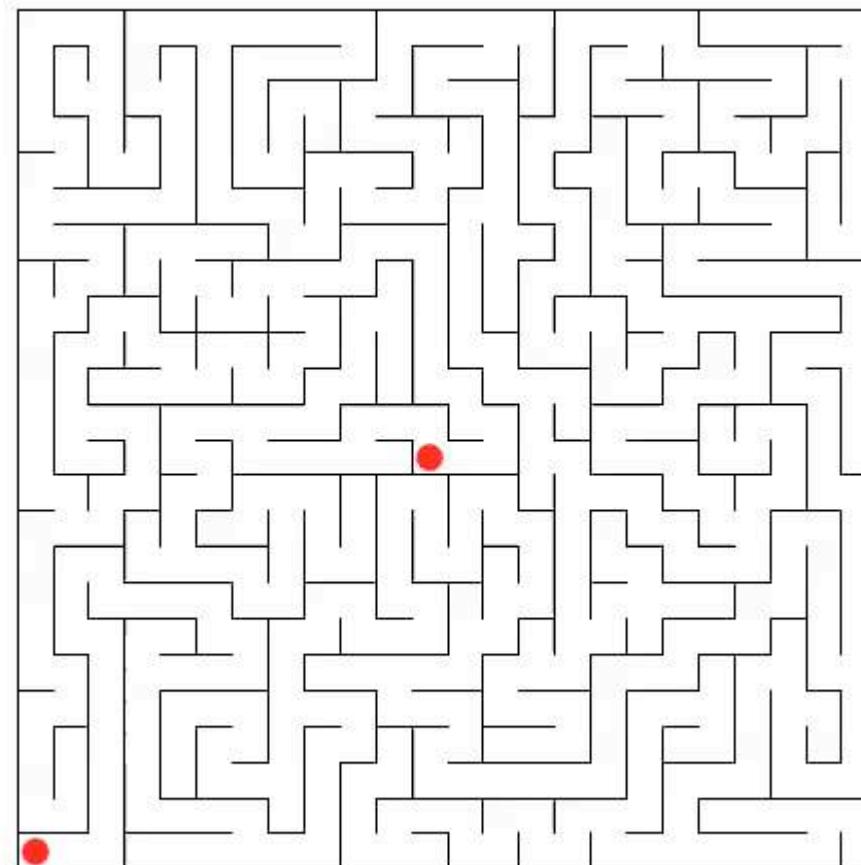


Claude Shannon (with Theseus mouse)

Maze exploration



Maze exploration



Depth-first search

Goal. Systematically search through a graph.

Idea. Mimic maze exploration.

DFS (to visit a vertex v)

Mark v as visited.

**Recursively visit all unmarked
vertices w adjacent to v.**

Typical applications.

- Find all vertices connected to a given source vertex.
- Find a path between two vertices.

Design challenge. How to implement?

Design pattern for graph processing

Design pattern. Decouple graph data type from graph processing.

- Create a Graph object.
- Pass the Graph to a graph-processing routine.
- Query the graph-processing routine for information.

```
public class Paths
```

```
    Paths(Graph G, int s)
```

find paths in G from source s

```
    boolean hasPathTo(int v)
```

is there a path from s to v?

```
    Iterable<Integer> pathTo(int v)
```

path from s to v; null if no such path

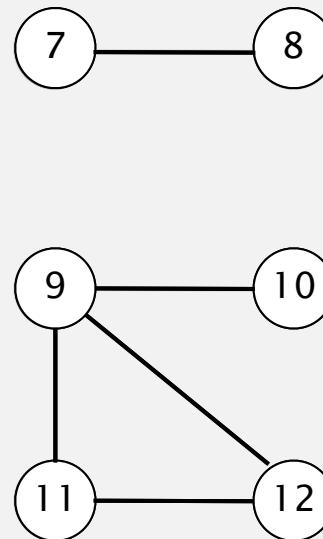
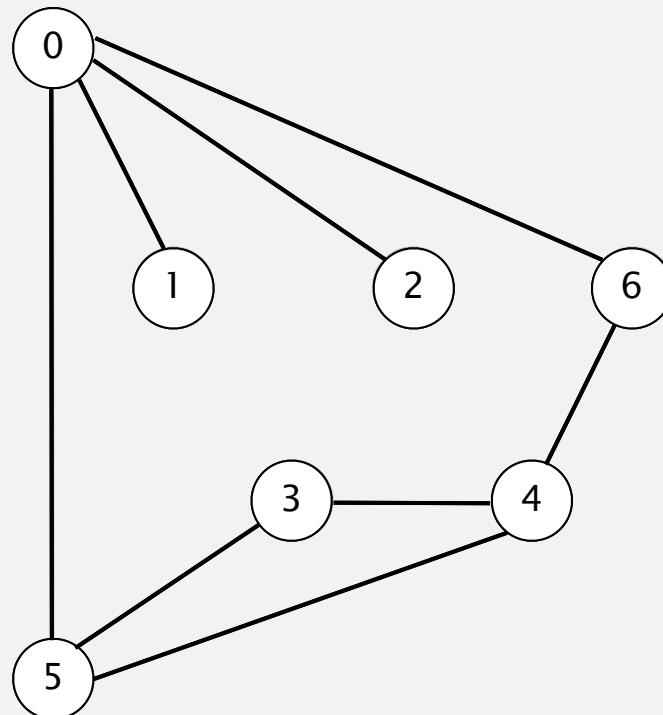
```
Paths paths = new Paths(G, s);
for (int v = 0; v < G.V(); v++)
    if (paths.hasPathTo(v))
        StdOut.println(v);
```

print all vertices
connected to s

Depth-first search demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v .



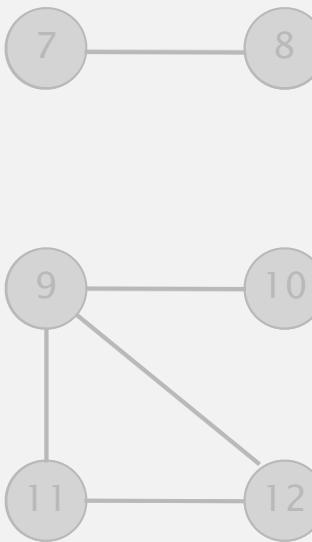
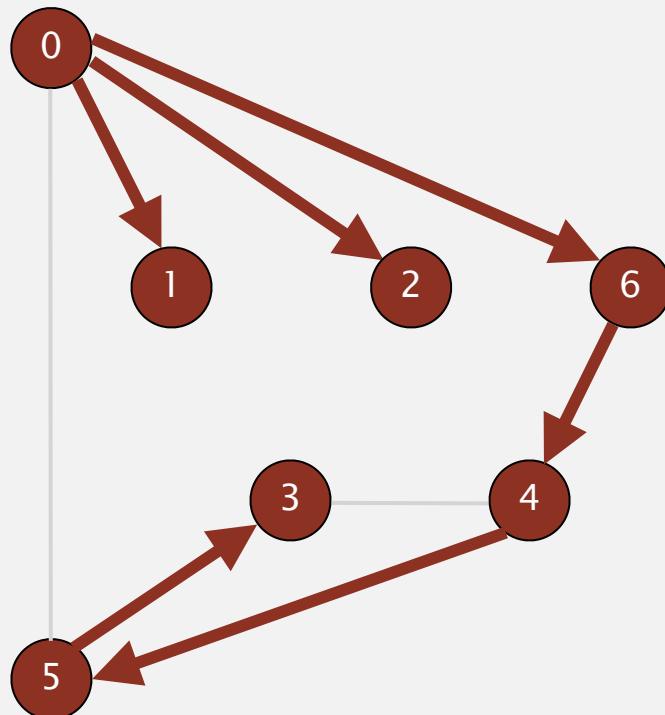
graph G

tinyG.txt
V → 13
E → 13
0 5
4 3
0 1
9 12
6 4
5 4
0 2
11 12
9 10
0 6
7 8
9 11
5 3

Depth-first search demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v .



v	marked[]	edgeTo[v]
0	T	-
1	T	0
2	T	0
3	T	5
4	T	6
5	T	4
6	T	0
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

vertices reachable from 0

Depth-first search

Goal. Find all vertices connected to s (and a corresponding path).

Idea. Mimic maze exploration.

Algorithm.

- Use recursion (ball of string).
- Mark each visited vertex (and keep track of edge taken to visit it).
- Return (retrace steps) when no unvisited options.

Data structures.

- `boolean[] marked` to mark visited vertices.
- `int[] edgeTo` to keep tree of paths.
 $(\text{edgeTo}[w] == v)$ means that edge $v-w$ taken to visit w for first time

Depth-first search

```
public class DepthFirstPaths
{
    private boolean[] marked;
    private int[] edgeTo;
    private int s;
```

marked[v] = true
if v connected to s
edgeTo[v] = previous
vertex on path from s to v

```
public DepthFirstSearch(Graph G, int s)
{
    ...
    dfs(G, s);
}
```

initialize data structures
find vertices connected to s

```
private void dfs(Graph G, int v)
{
    marked[v] = true;
    for (int w : G.adj(v))
        if (!marked[w])
        {
            dfs(G, w);
            edgeTo[w] = v;
        }
}
```

recursive DFS does the work

Depth-first search properties

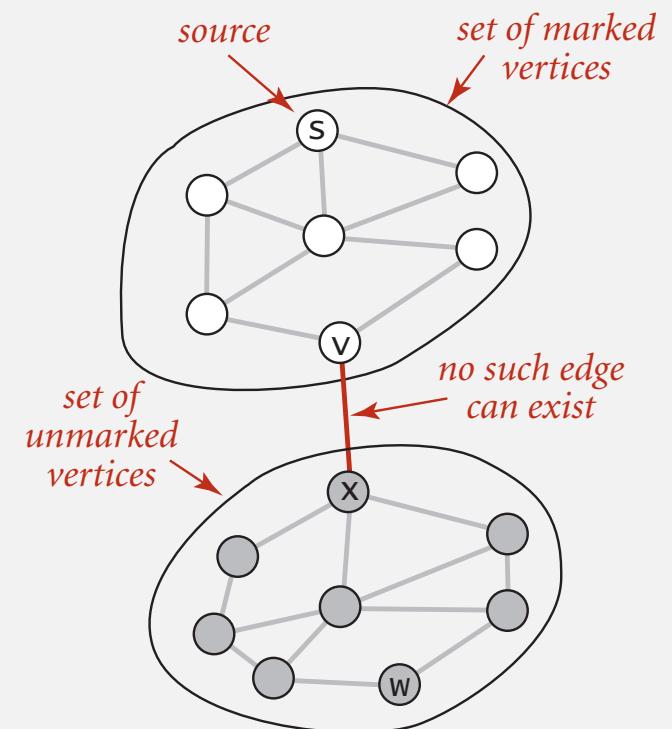
Proposition. DFS marks all vertices connected to s in time proportional to the sum of their degrees.

Pf. [correctness]

- If w marked, then w connected to s (why?)
- If w connected to s , then w marked.
(if w unmarked, then consider last edge on a path from s to w that goes from a marked vertex to an unmarked one).

Pf. [running time]

Each vertex connected to s is visited once.



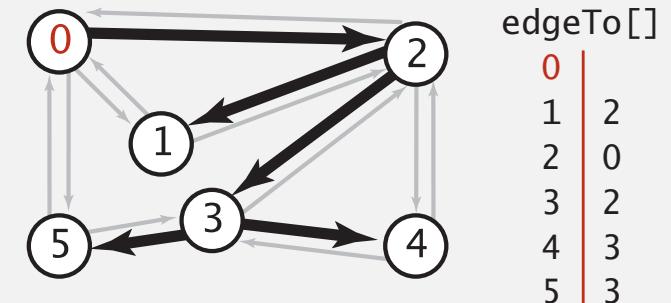
Depth-first search properties

Proposition. After DFS, can find vertices connected to s in constant time and can find a path to s (if one exists) in time proportional to its length.

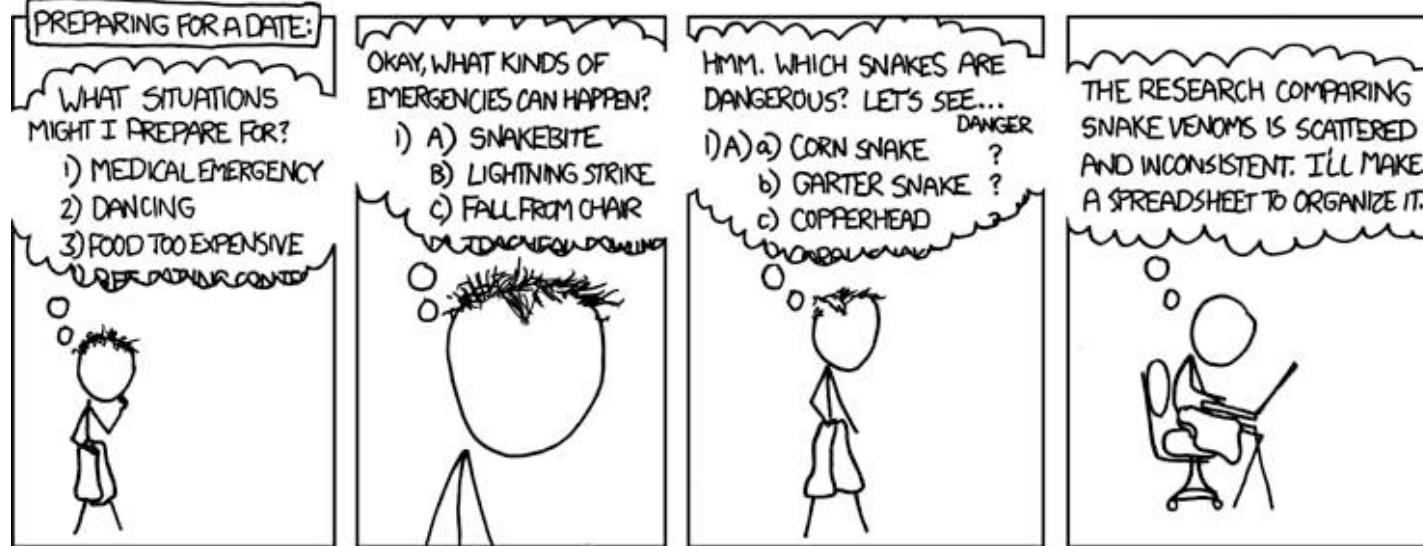
Pf. `edgeTo[]` is parent-link representation of a tree rooted at s .

```
public boolean hasPathTo(int v)
{   return marked[v]; }

public Iterable<Integer> pathTo(int v)
{
    if (!hasPathTo(v)) return null;
    Stack<Integer> path = new Stack<Integer>();
    for (int x = v; x != s; x = edgeTo[x])
        path.push(x);
    path.push(s);
    return path;
}
```



Depth-first search application: preparing for a date



I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

xkcd

<http://xkcd.com/761/>

Depth-first search application: flood fill

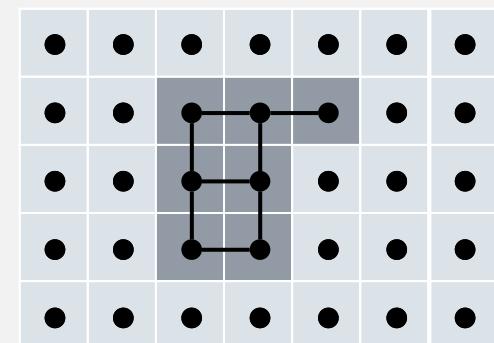
Challenge. Flood fill (Photoshop magic wand).

Assumptions. Picture has millions to billions of pixels.



Solution. Build a grid graph.

- Vertex: pixel.
- Edge: between two adjacent gray pixels.
- Blob: all pixels connected to given pixel.



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

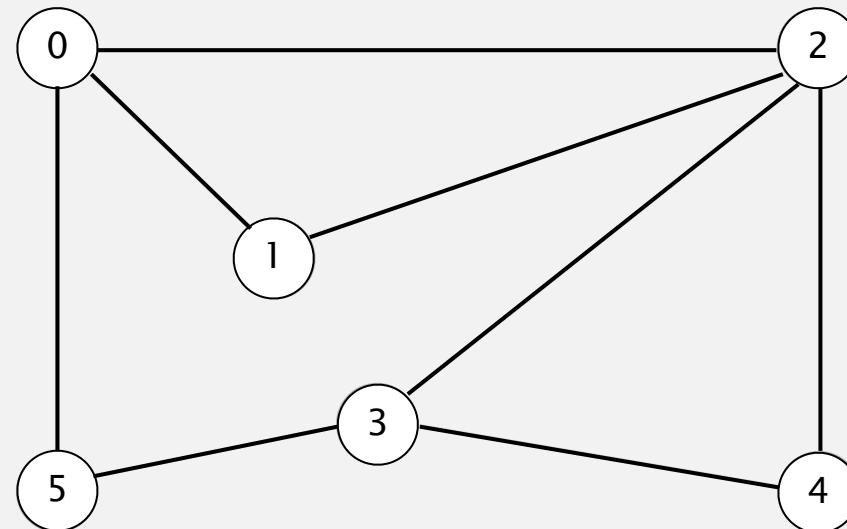
4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ ***breadth-first search***
- ▶ *connected components*
- ▶ *challenges*

Breadth-first search demo

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



tinyCG.txt

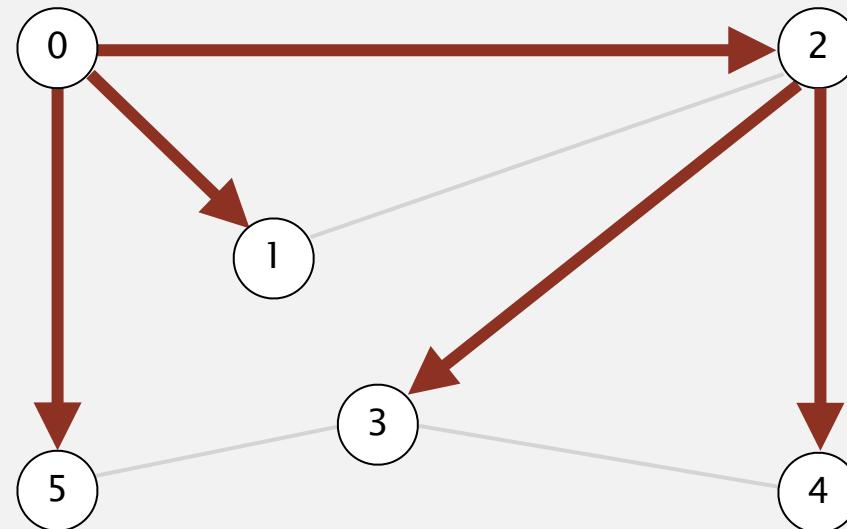
$V \rightarrow$ 6
8
0 5
2 4
2 3
1 2
0 1
3 4
3 5
0 2
 $E \leftarrow$

graph G

Breadth-first search demo

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices adjacent to v and mark them.



<code>v</code>	<code>edgeTo[]</code>	<code>distTo[]</code>
0	-	0
1	0	1
2	0	1
3	2	2
4	2	2
5	0	1

done

Breadth-first search

Depth-first search. Put unvisited vertices on a **stack**.

Breadth-first search. Put unvisited vertices on a **queue**.

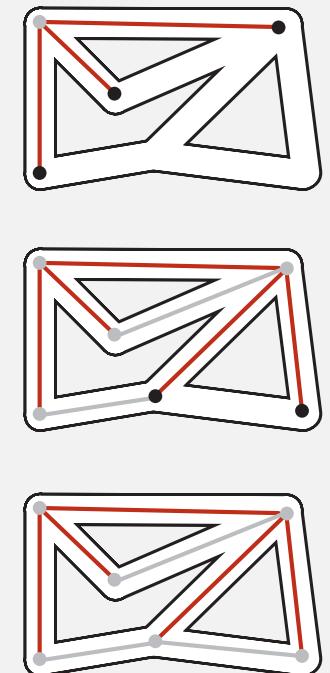
Shortest path. Find path from s to t that uses **fewest number of edges**.

BFS (from source vertex s)

Put s onto a FIFO queue, and mark s as visited.

Repeat until the queue is empty:

- **remove the least recently added vertex v**
 - **add each of v 's unvisited neighbors to the queue,**
and mark them as visited.
-



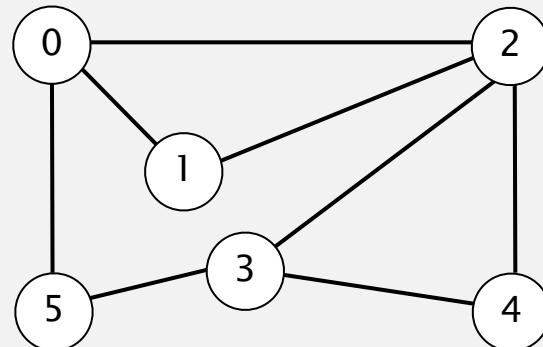
Intuition. BFS examines vertices in increasing distance from s .

Breadth-first search properties

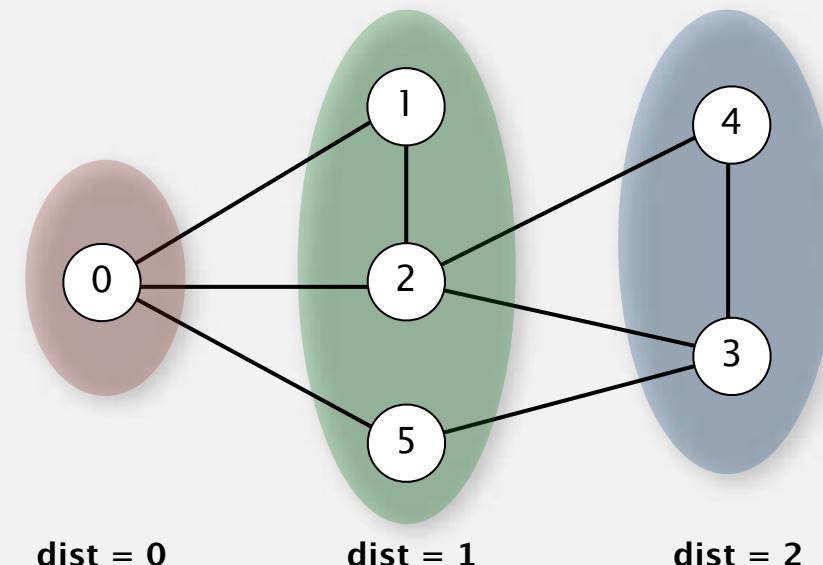
Proposition. BFS computes shortest paths (fewest number of edges) from s to all other vertices in a graph in time proportional to $E + V$.

Pf. [correctness] Queue always consists of zero or more vertices of distance k from s , followed by zero or more vertices of distance $k + 1$.

Pf. [running time] Each vertex connected to s is visited once.



graph



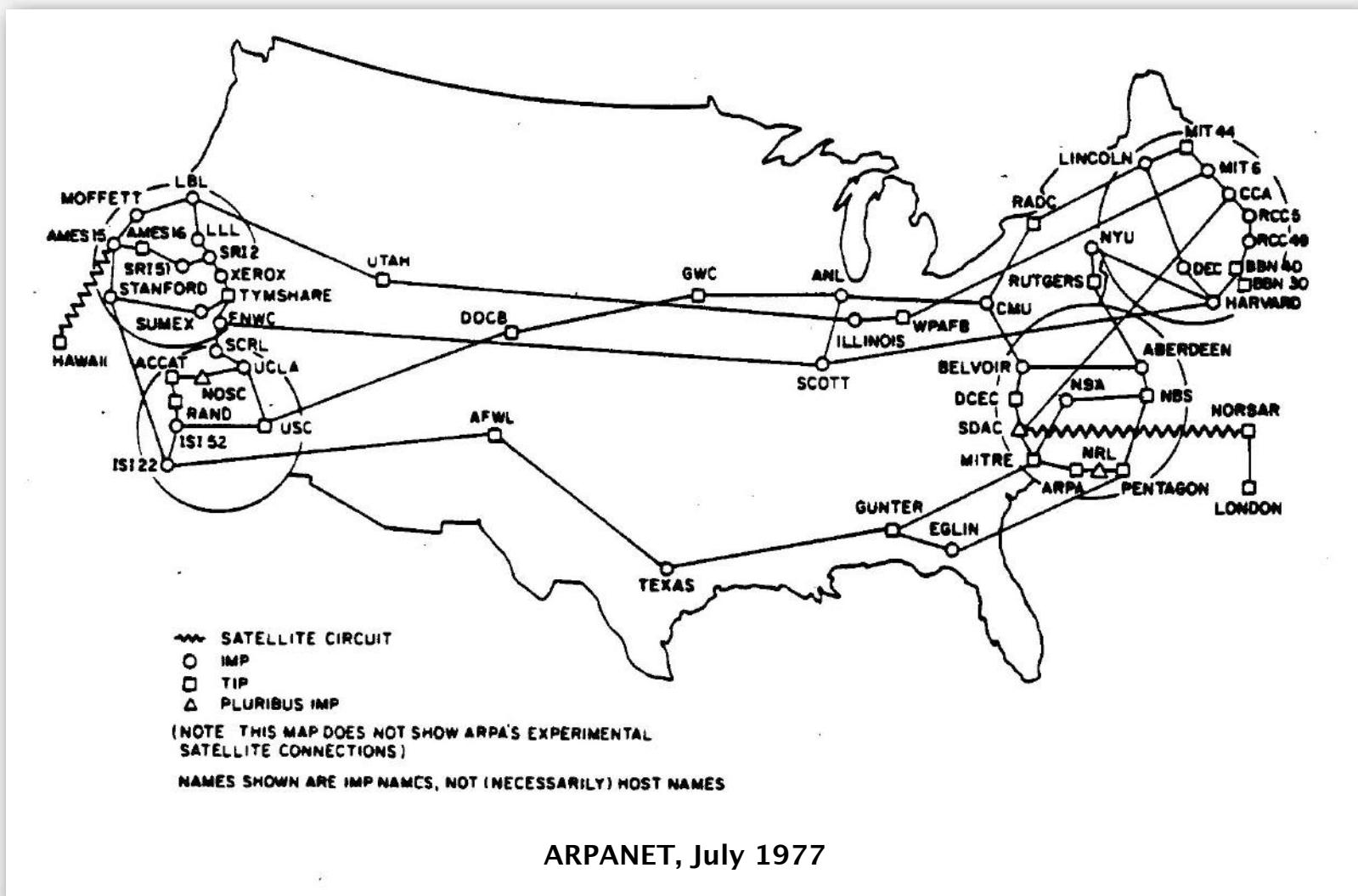
Breadth-first search

```
public class BreadthFirstPaths
{
    private boolean[] marked;
    private int[] edgeTo;
    ...

    private void bfs(Graph G, int s)
    {
        Queue<Integer> q = new Queue<Integer>();
        q.enqueue(s);
        marked[s] = true;
        while (!q.isEmpty())
        {
            int v = q.dequeue();
            for (int w : G.adj(v))
            {
                if (!marked[w])
                {
                    q.enqueue(w);
                    marked[w] = true;
                    edgeTo[w] = v;
                }
            }
        }
    }
}
```

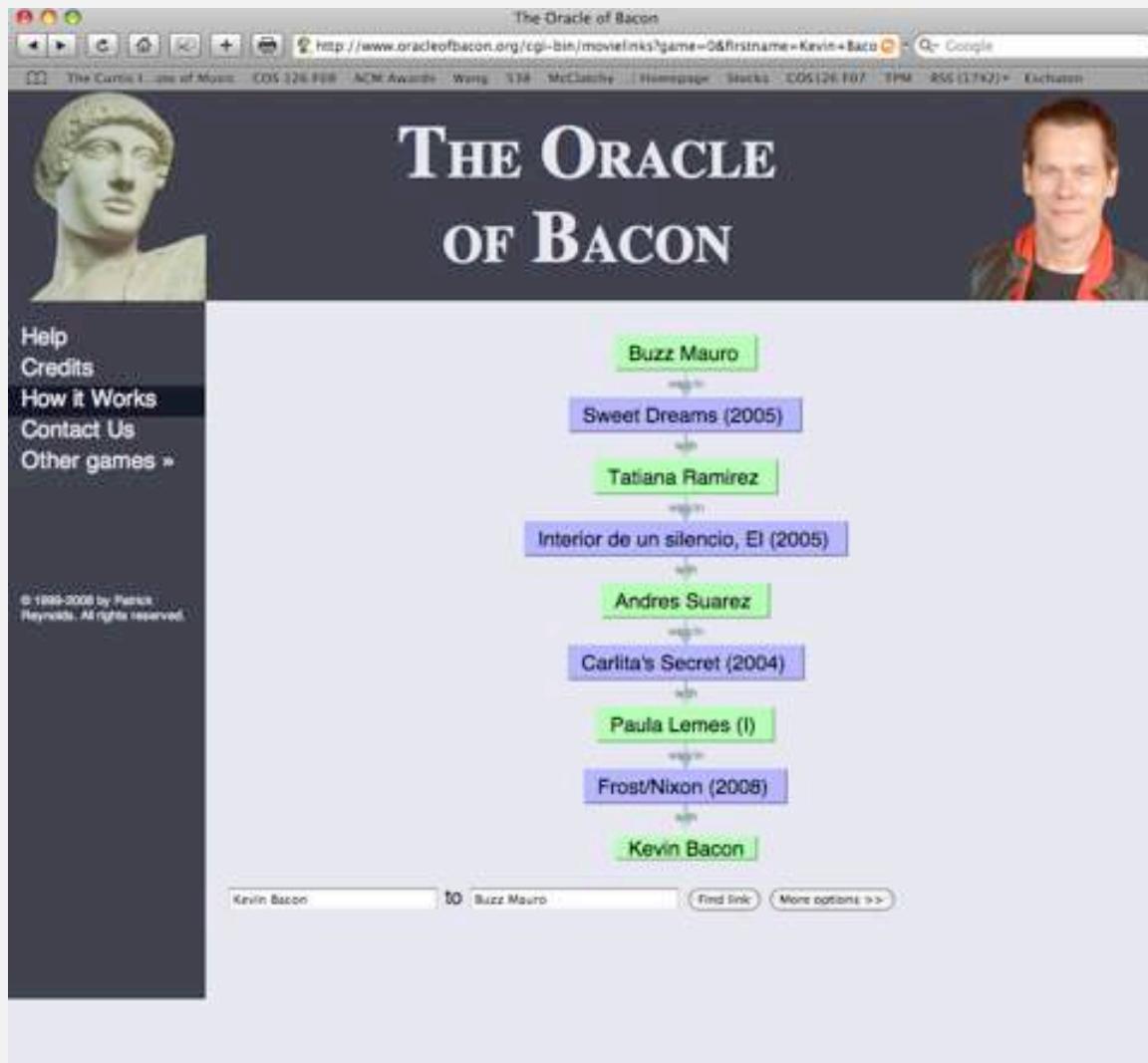
Breadth-first search application: routing

Fewest number of hops in a communication network.



Breadth-first search application: Kevin Bacon numbers

Kevin Bacon numbers.



<http://oracleofbacon.org>



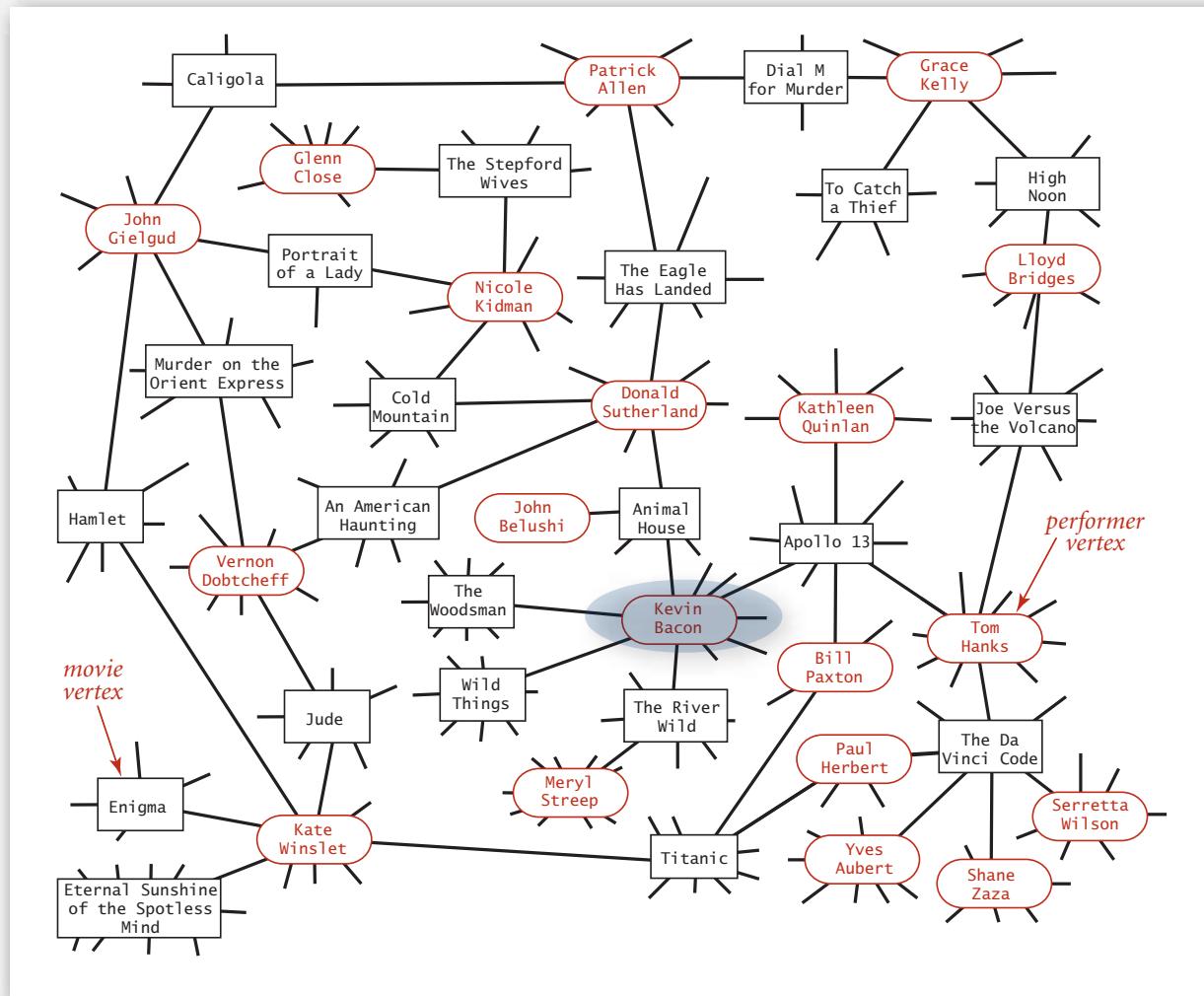
Endless Games board game

The SixDegrees iPhone app interface. At the top, it says "2 Degrees". Below that is a list of actors and their connections: "Uma Thurman acted in Be Cool (2005) with Scott Adsit who acted in The Informant! (2009) with Matt Damon". At the bottom are four navigation icons: "Lookup" (magnifying glass), "Trivia" (question mark), "# Guess Degrees" (hash symbol), and "Scoreboard" (checkmark).

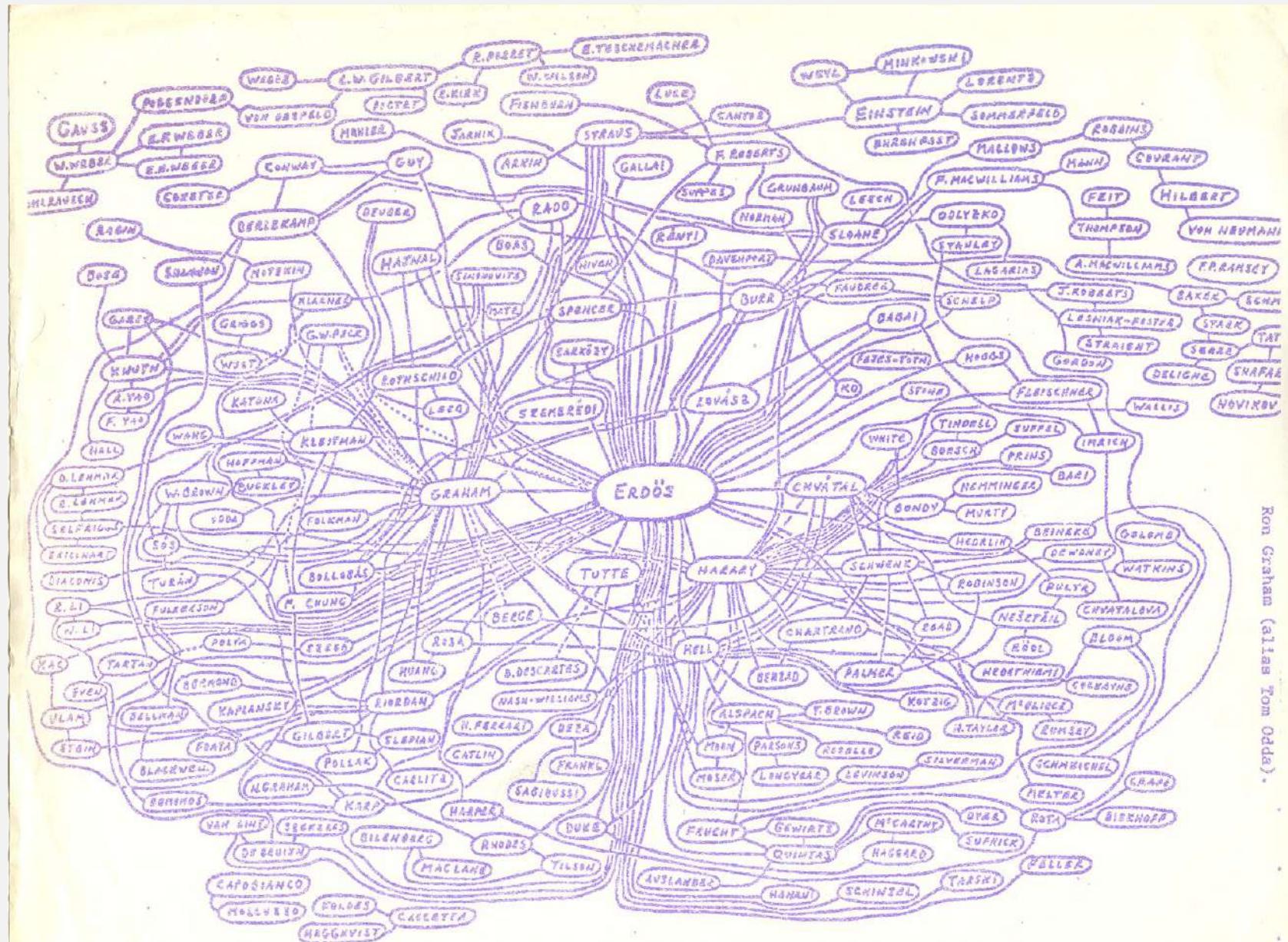
SixDegrees iPhone App

Kevin Bacon graph

- Include one vertex for each performer **and** one for each movie.
- Connect a movie to all performers that appear in that movie.
- Compute shortest path from $s = \text{Kevin Bacon}$.



Breadth-first search application: Erdős numbers



hand-drawing of part of the Erdős graph by Ron Graham

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ ***breadth-first search***
- ▶ *connected components*
- ▶ *challenges*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ ***connected components***
- ▶ *challenges*

Connectivity queries

Def. Vertices v and w are **connected** if there is a path between them.

Goal. Preprocess graph to answer queries of the form *is v connected to w ?* in **constant time**.

public class CC	
CC(Graph G)	<i>find connected components in G</i>
boolean connected(int v, int w)	<i>are v and w connected?</i>
int count()	<i>number of connected components</i>
int id(int v)	<i>component identifier for v</i>

Union-Find? Not quite.

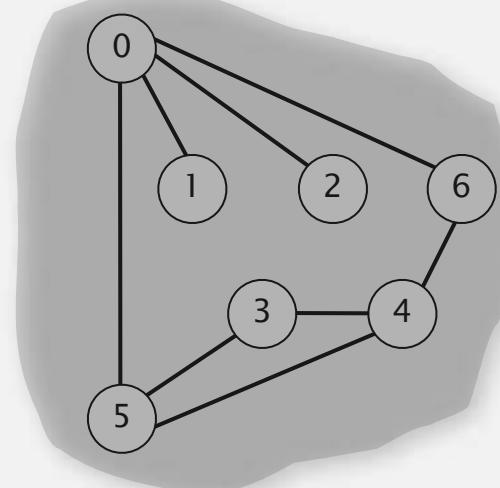
Depth-first search. Yes. [next few slides]

Connected components

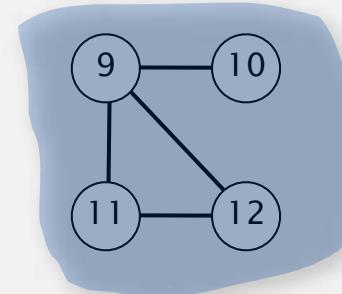
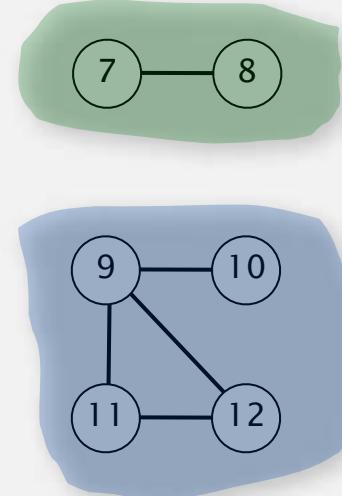
The relation "is connected to" is an **equivalence relation**:

- Reflexive: v is connected to v .
- Symmetric: if v is connected to w , then w is connected to v .
- Transitive: if v connected to w and w connected to x , then v connected to x .

Def. A **connected component** is a maximal set of connected vertices.



3 connected components

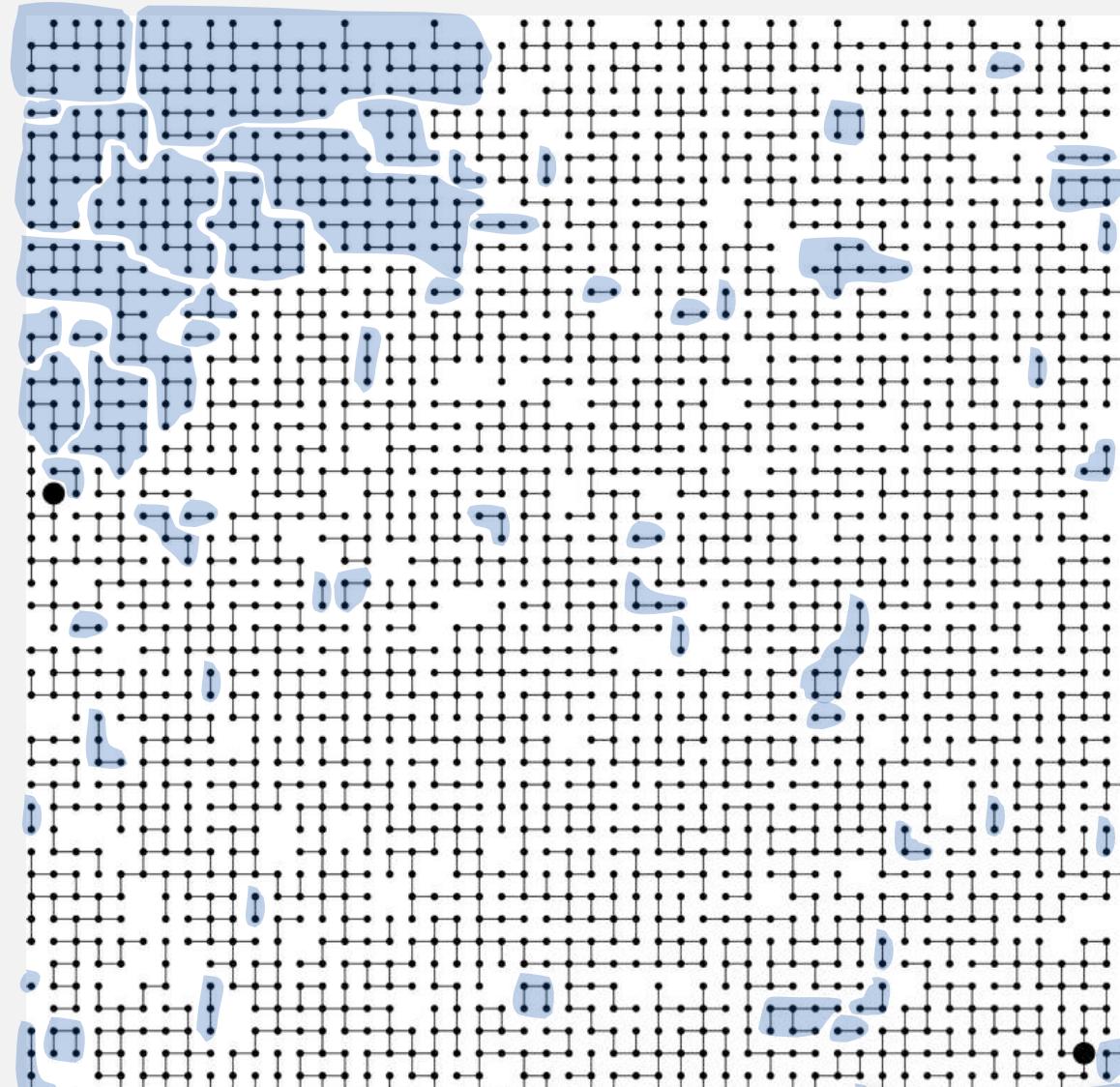


v	id[]
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	1
9	2
10	2
11	2
12	2

Remark. Given connected components, can answer queries in constant time.

Connected components

Def. A **connected component** is a maximal set of connected vertices.



63 connected components

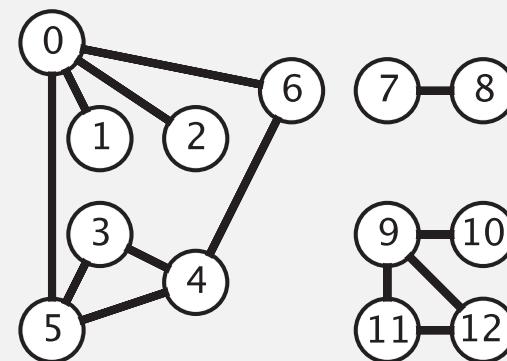
Connected components

Goal. Partition vertices into connected components.

Connected components

Initialize all vertices v as unmarked.

For each unmarked vertex v , run DFS to identify all vertices discovered as part of the same component.



tinyG.txt

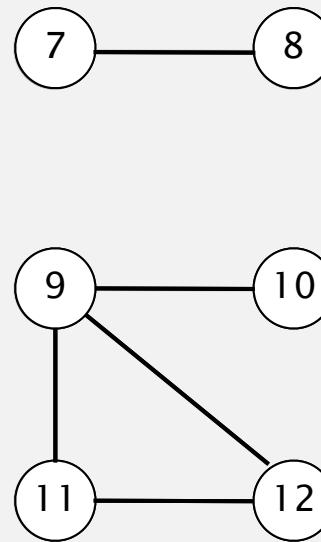
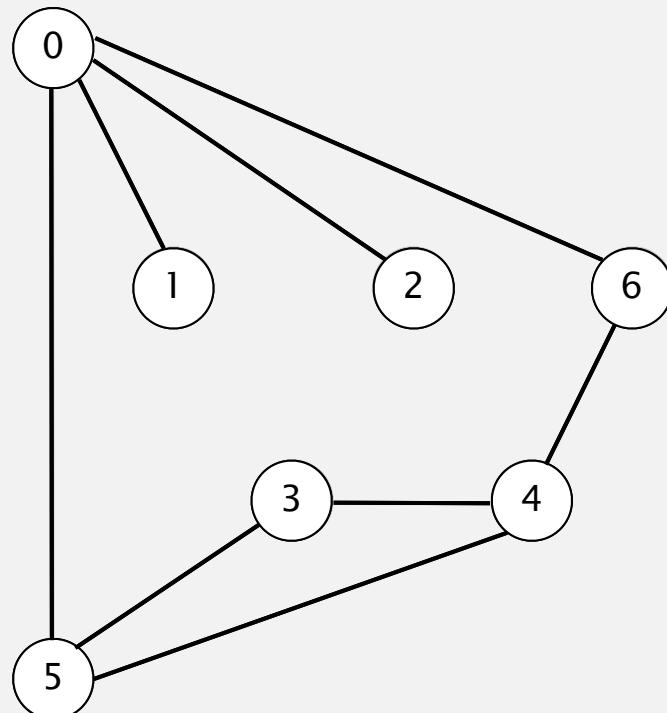
$V \rightarrow 13$ $E \leftarrow$

13	13
0 5	
4 3	
0 1	
9 12	
6 4	
5 4	
0 2	
11 12	
9 10	
0 6	
7 8	
9 11	
5 3	

Connected components demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v .



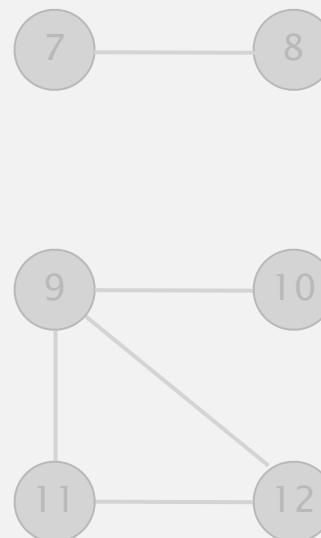
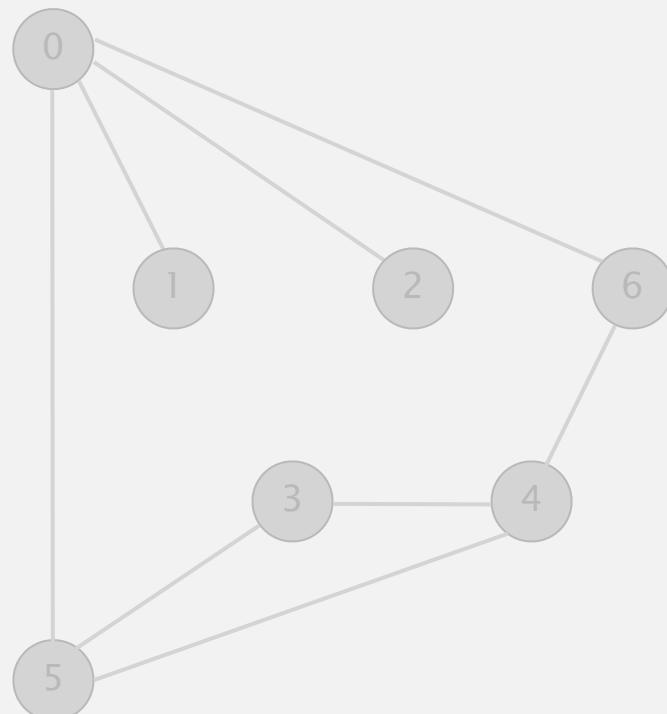
v	marked[]	id[]
0	F	-
1	F	-
2	F	-
3	F	-
4	F	-
5	F	-
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

graph G

Connected components demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices adjacent to v .



v	marked[]	id[]
0	T	0
1	T	0
2	T	0
3	T	0
4	T	0
5	T	0
6	T	0
7	T	1
8	T	1
9	T	2
10	T	2
11	T	2
12	T	2

done

Finding connected components with DFS

```
public class CC
{
    private boolean[] marked;
    private int[] id;
    private int count;

    public CC(Graph G)
    {
        marked = new boolean[G.V()];
        id = new int[G.V()];
        for (int v = 0; v < G.V(); v++)
        {
            if (!marked[v])
            {
                dfs(G, v);
                count++;
            }
        }
    }

    public int count()
    public int id(int v)
    private void dfs(Graph G, int v)
}
```

$\text{id}[v] = \text{id}$ of component containing v
number of components

run DFS from one vertex in
each component

see next slide

Finding connected components with DFS (continued)

```
public int count()  
{  return count;  }
```

number of components

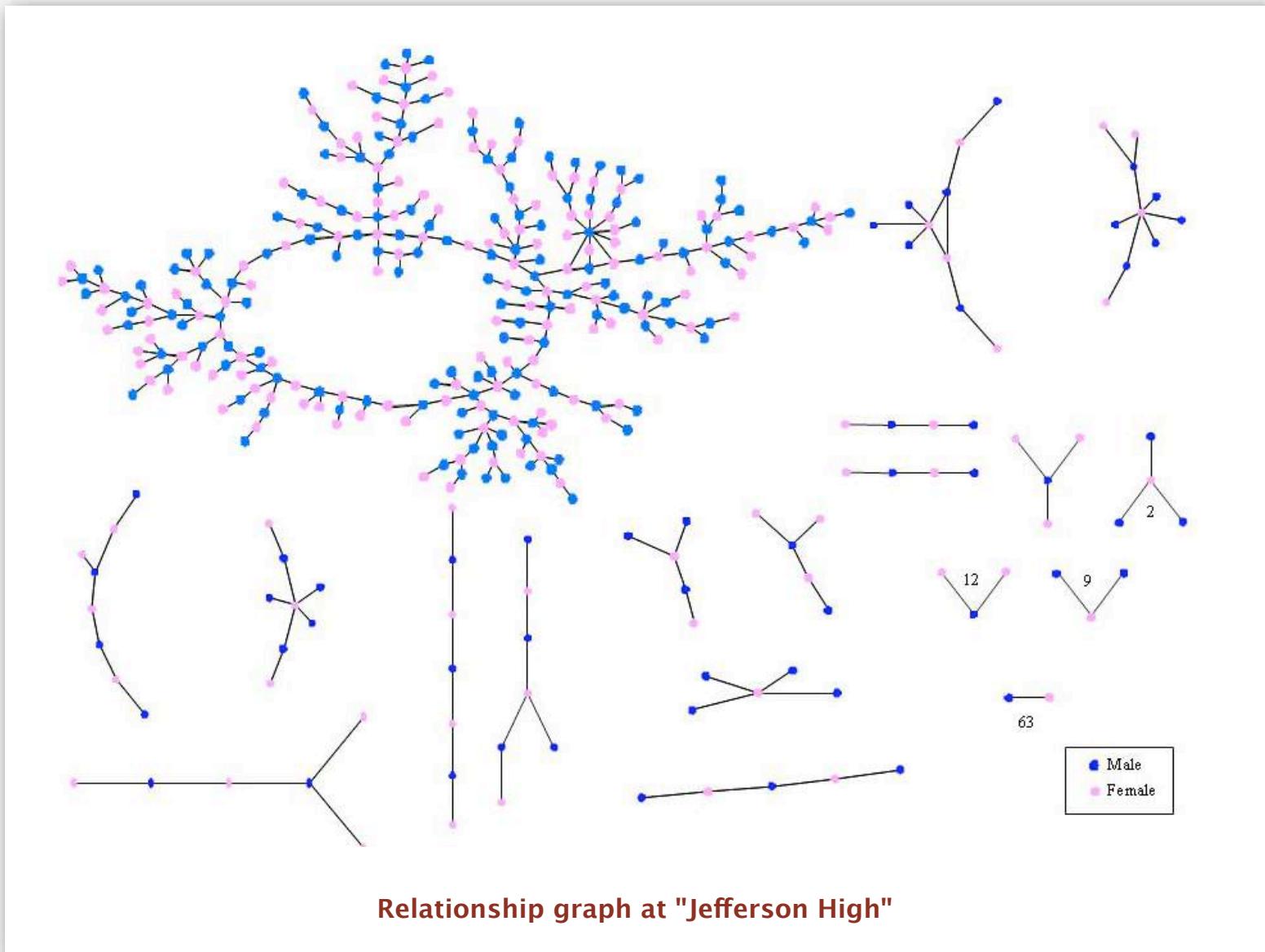
```
public int id(int v)  
{  return id[v];  }
```

id of component containing v

```
private void dfs(Graph G, int v)  
{  
    marked[v] = true;  
    id[v] = count;  
    for (int w : G.adj(v))  
        if (!marked[w])  
            dfs(G, w);  
}
```

all vertices discovered in
same call of dfs have same id

Connected components application: study spread of STDs



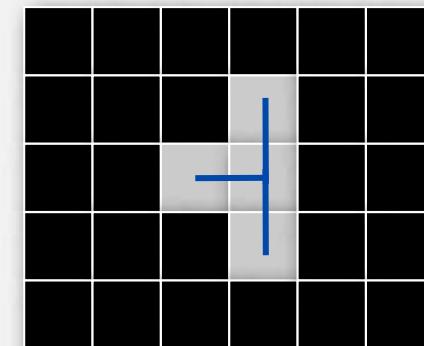
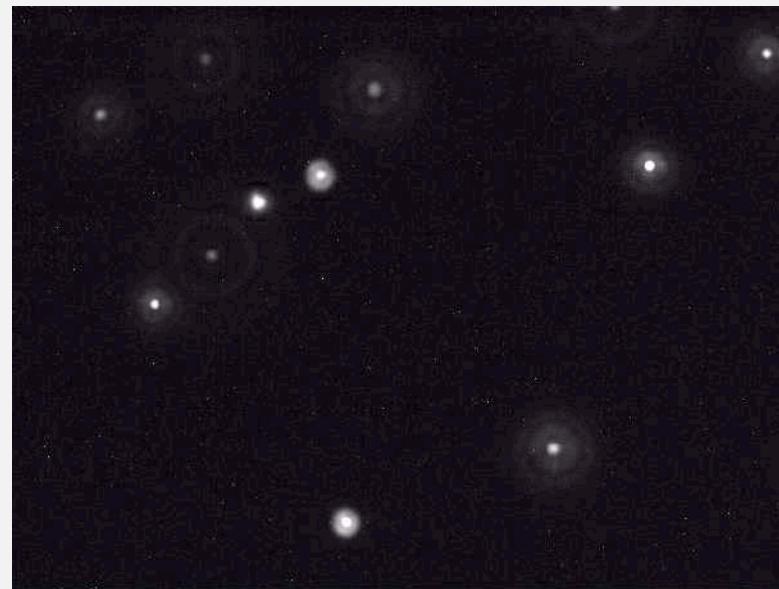
Peter Bearman, James Moody, and Katherine Stovel. Chains of affection: The structure of adolescent romantic and sexual networks. American Journal of Sociology, 110(1): 44–99, 2004.

Connected components application: particle detection

Particle detection. Given grayscale image of particles, identify "blobs."

- Vertex: pixel.
- Edge: between two adjacent pixels with grayscale value ≥ 70 .
- Blob: connected component of 20-30 pixels.

black = 0
white = 255



Particle tracking. Track moving particles over time.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ ***connected components***
- ▶ *challenges*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ ***challenges***

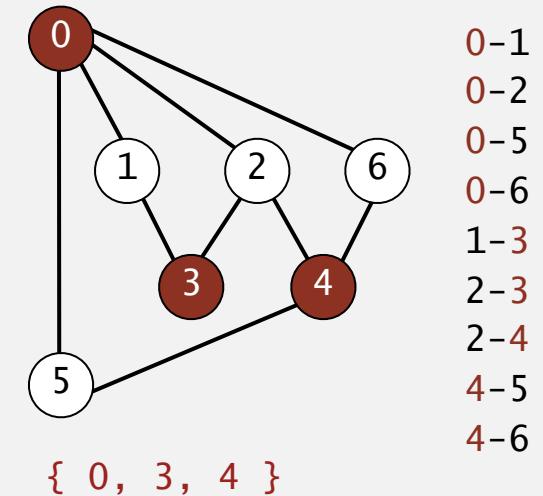
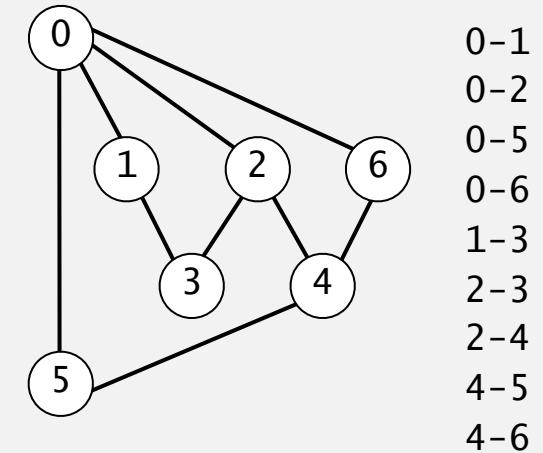
Graph-processing challenge 1

Problem. Is a graph bipartite?

How difficult?

- Any programmer could do it.
- ✓ • Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

simple DFS-based solution
(see textbook)



Bipartiteness application: is dating graph bipartite?

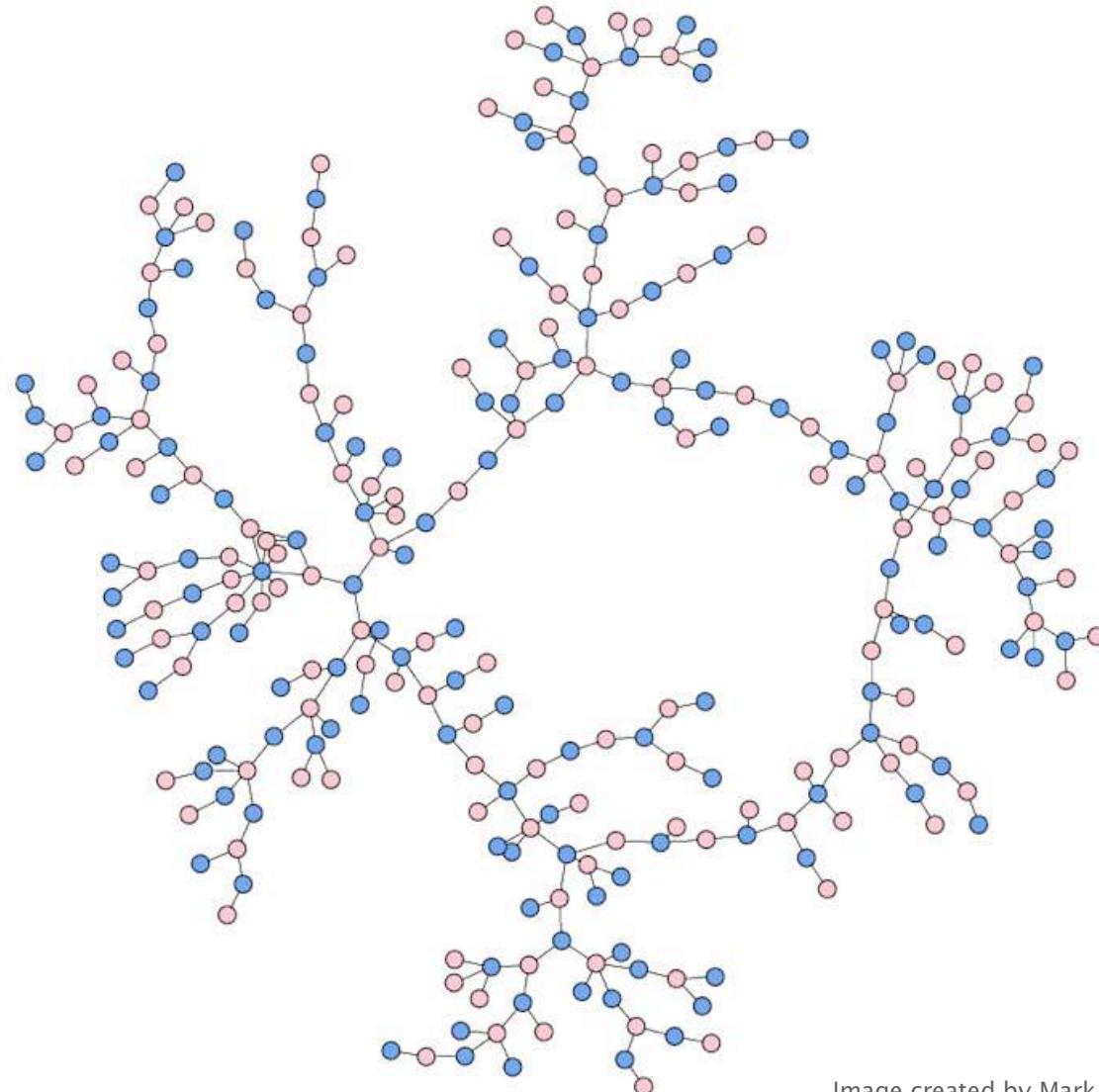


Image created by Mark Newman.

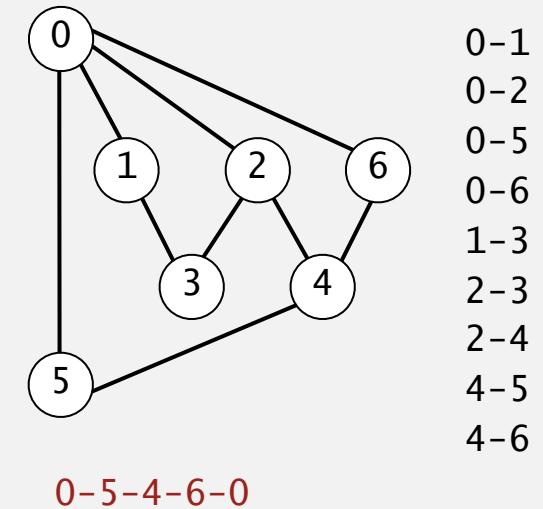
Graph-processing challenge 2

Problem. Find a cycle.

How difficult?

- Any programmer could do it.
- ✓ • Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

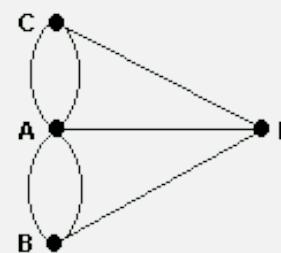
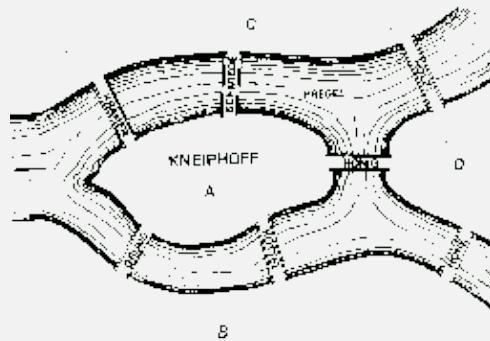
simple DFS-based solution
(see textbook)



Bridges of Königsberg

The Seven Bridges of Königsberg. [Leonhard Euler 1736]

“ ... in Königsberg in Prussia, there is an island A, called the Kneiphof; the river which surrounds it is divided into two branches ... and these branches are crossed by seven bridges. Concerning these bridges, it was asked whether anyone could arrange a route in such a way that he could cross each bridge once and only once.”



Euler tour. Is there a (general) cycle that uses each edge exactly once?

Answer. Yes iff connected and all vertices have **even** degree.

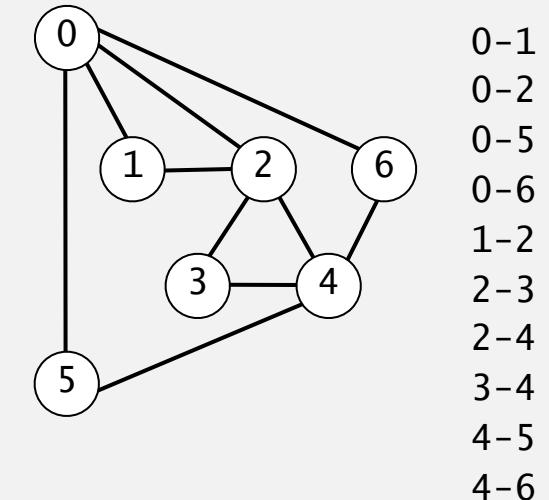
Graph-processing challenge 3

Problem. Find a (general) cycle that uses every edge exactly once.

How difficult?

- Any programmer could do it.
- ✓ • Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- No one knows.
- Impossible.

Eulerian tour
(classic graph-processing problem)



0-1-2-3-4-2-0-6-4-5-0

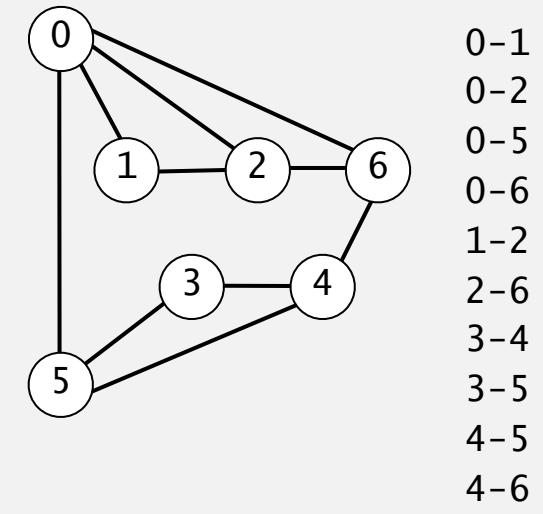
Graph-processing challenge 4

Problem. Find a cycle that visits every vertex exactly once.

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- ✓ • Intractable. ←
• No one knows.
• Impossible.

Hamiltonian cycle
(classical NP-complete problem)



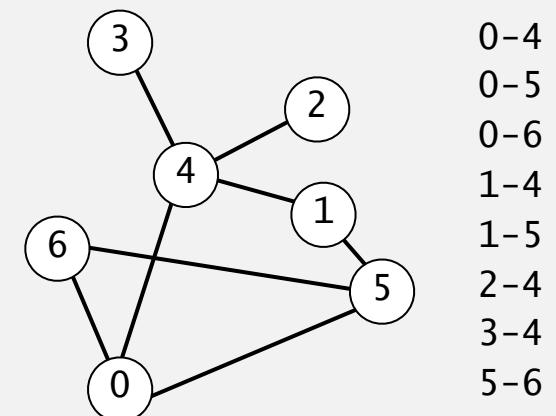
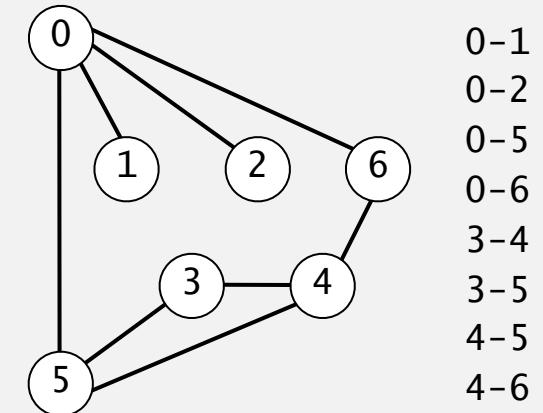
Graph-processing challenge 5

Problem. Are two graphs identical except for vertex names?

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- Hire an expert.
- Intractable.
- ✓ • No one knows.
- Impossible.

graph isomorphism is
longstanding open problem



$0 \leftrightarrow 4, 1 \leftrightarrow 3, 2 \leftrightarrow 2, 3 \leftrightarrow 6, 4 \leftrightarrow 5, 5 \leftrightarrow 0, 6 \leftrightarrow 1$

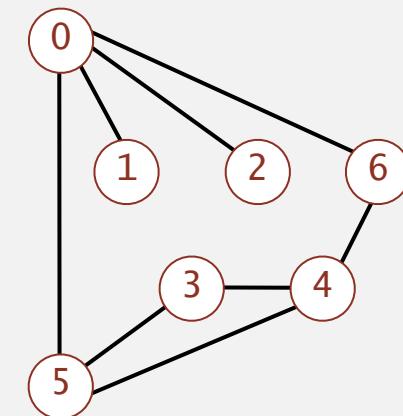
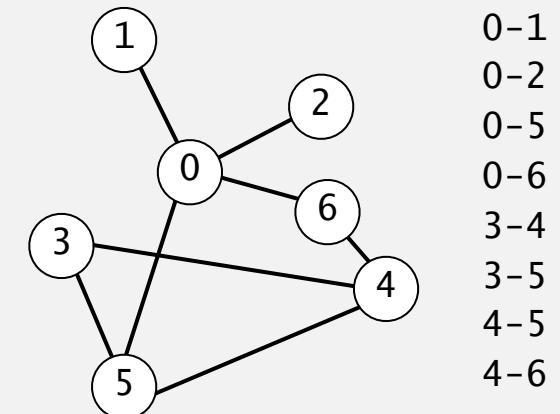
Graph-processing challenge 6

Problem. Lay out a graph in the plane without crossing edges?

How difficult?

- Any programmer could do it.
- Typical diligent algorithms student could do it.
- ✓ • Hire an expert.
- Intractable.
- No one knows.
- Impossible.

linear-time DFS-based planarity algorithm
discovered by Tarjan in 1970s
(too complicated for most practitioners)



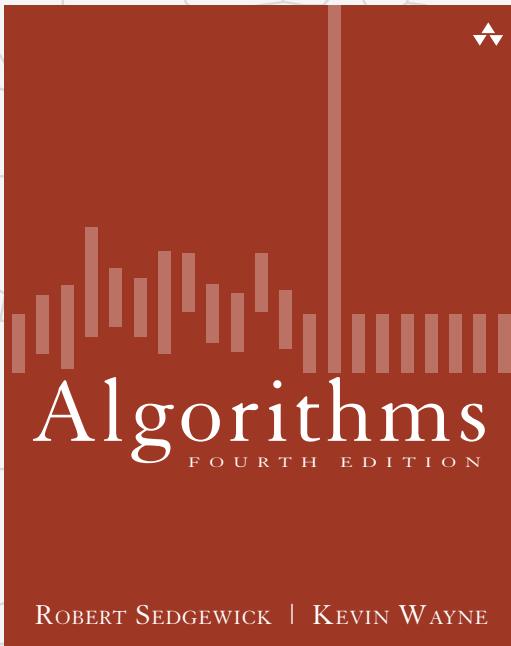
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ ***challenges***



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.1 UNDIRECTED GRAPHS

- ▶ *introduction*
- ▶ *graph API*
- ▶ *depth-first search*
- ▶ *breadth-first search*
- ▶ *connected components*
- ▶ *challenges*



<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

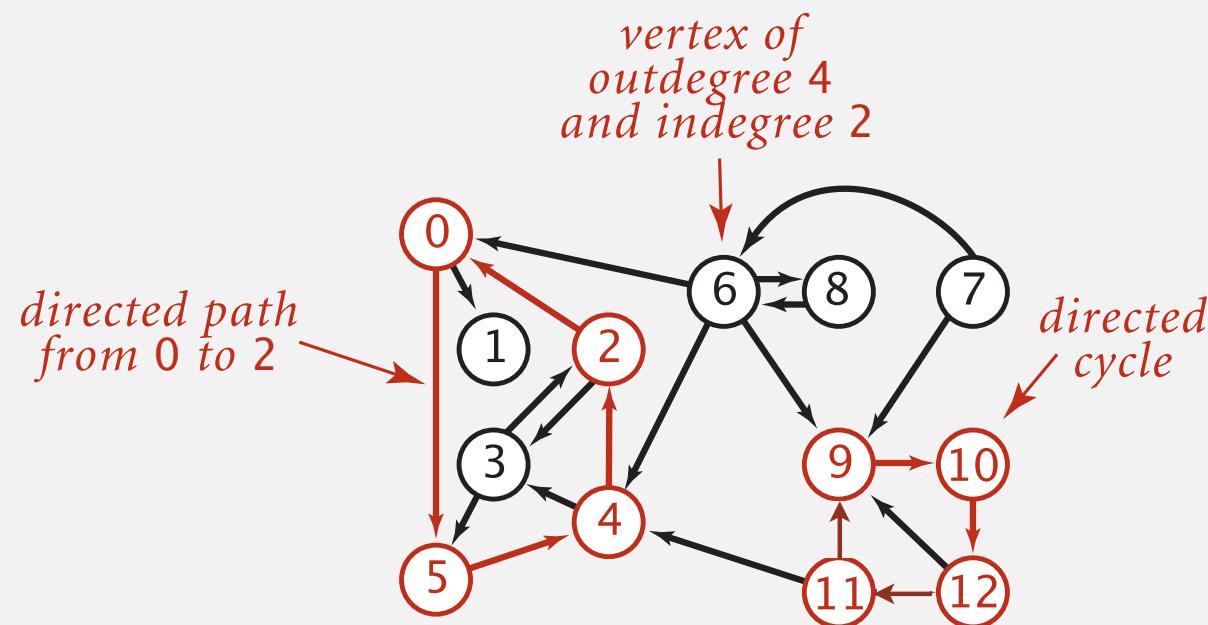
<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

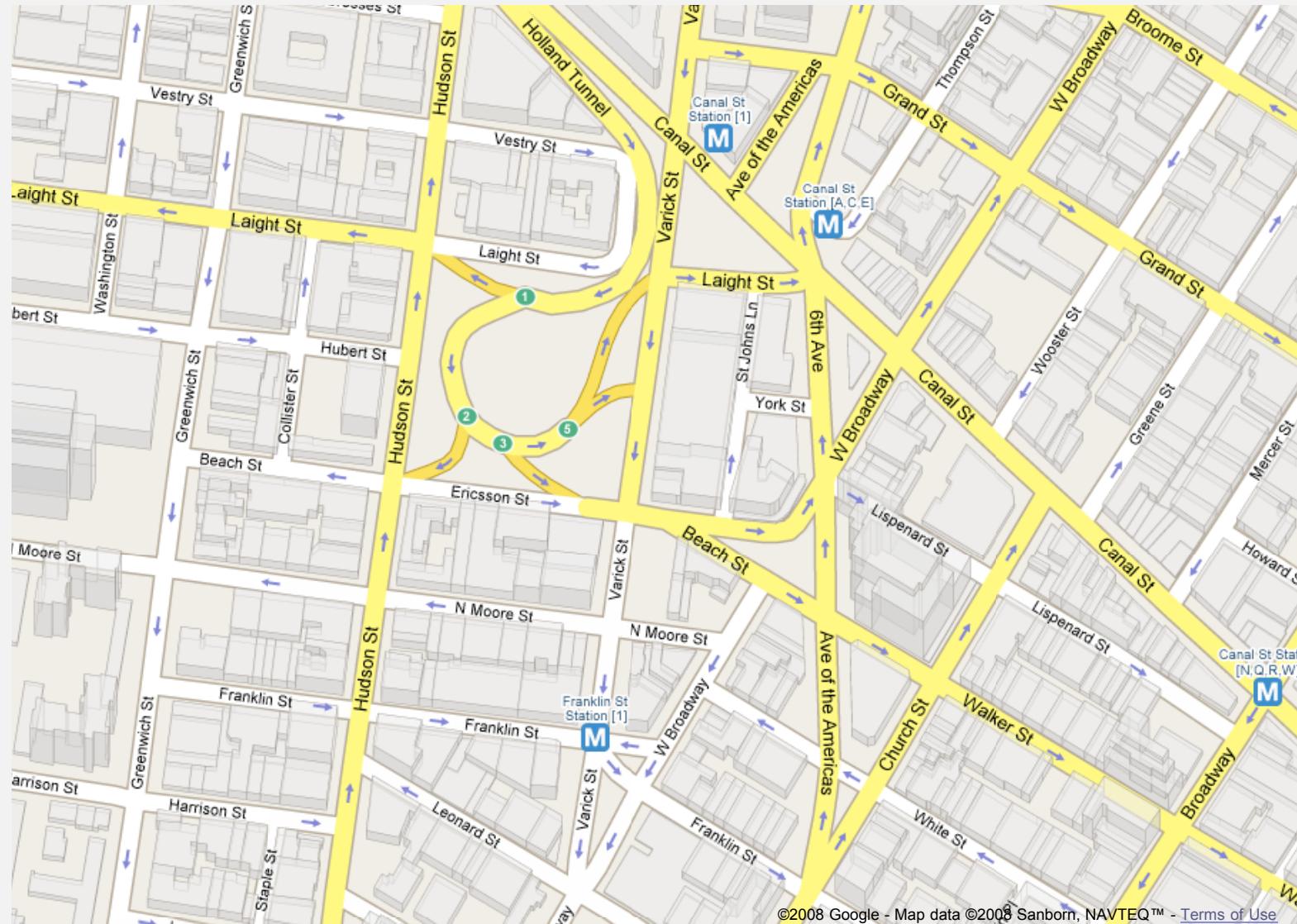
Directed graphs

Digraph. Set of vertices connected pairwise by **directed** edges.



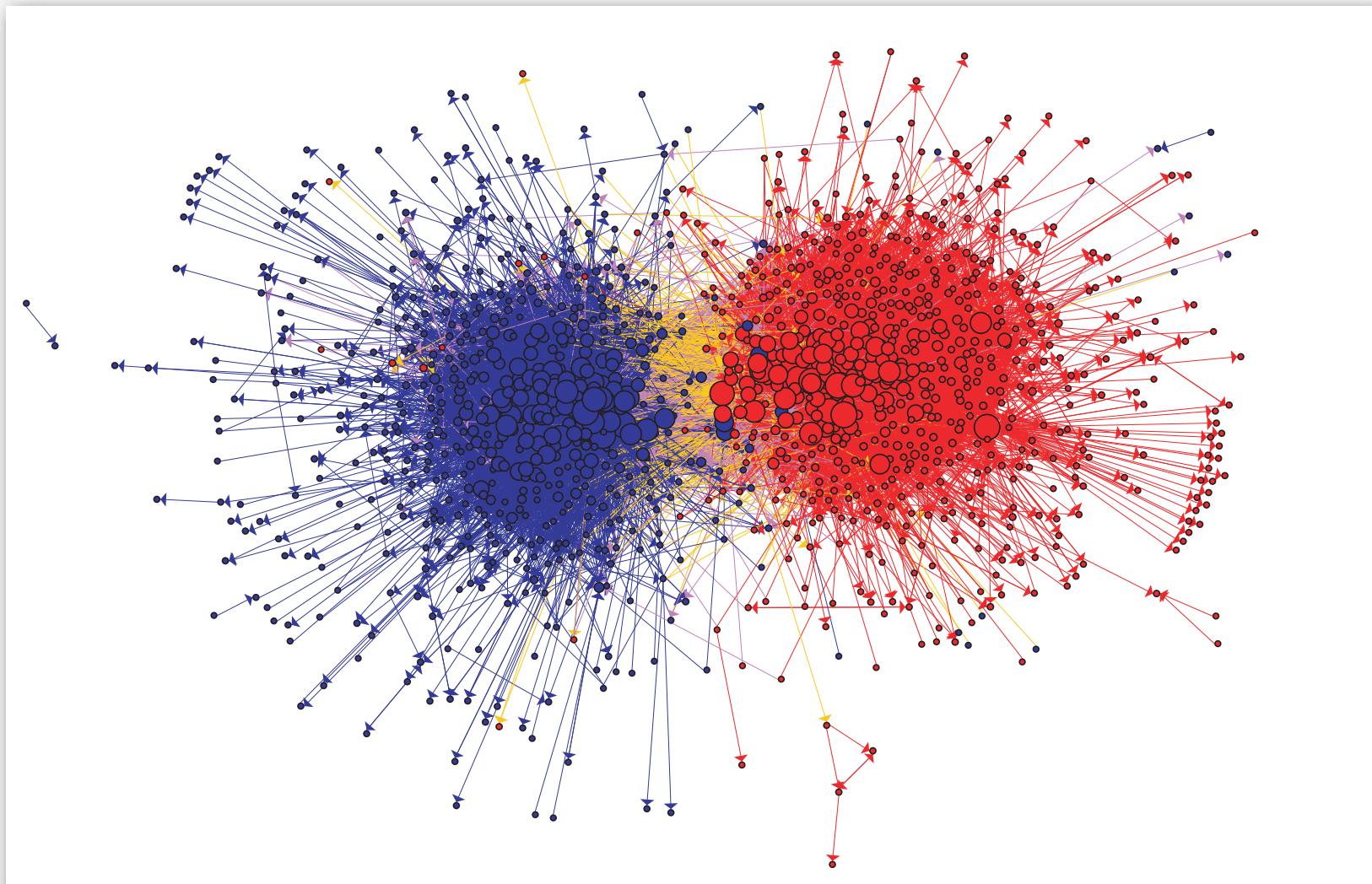
Road network

Vertex = intersection; edge = one-way street.



Political blogosphere graph

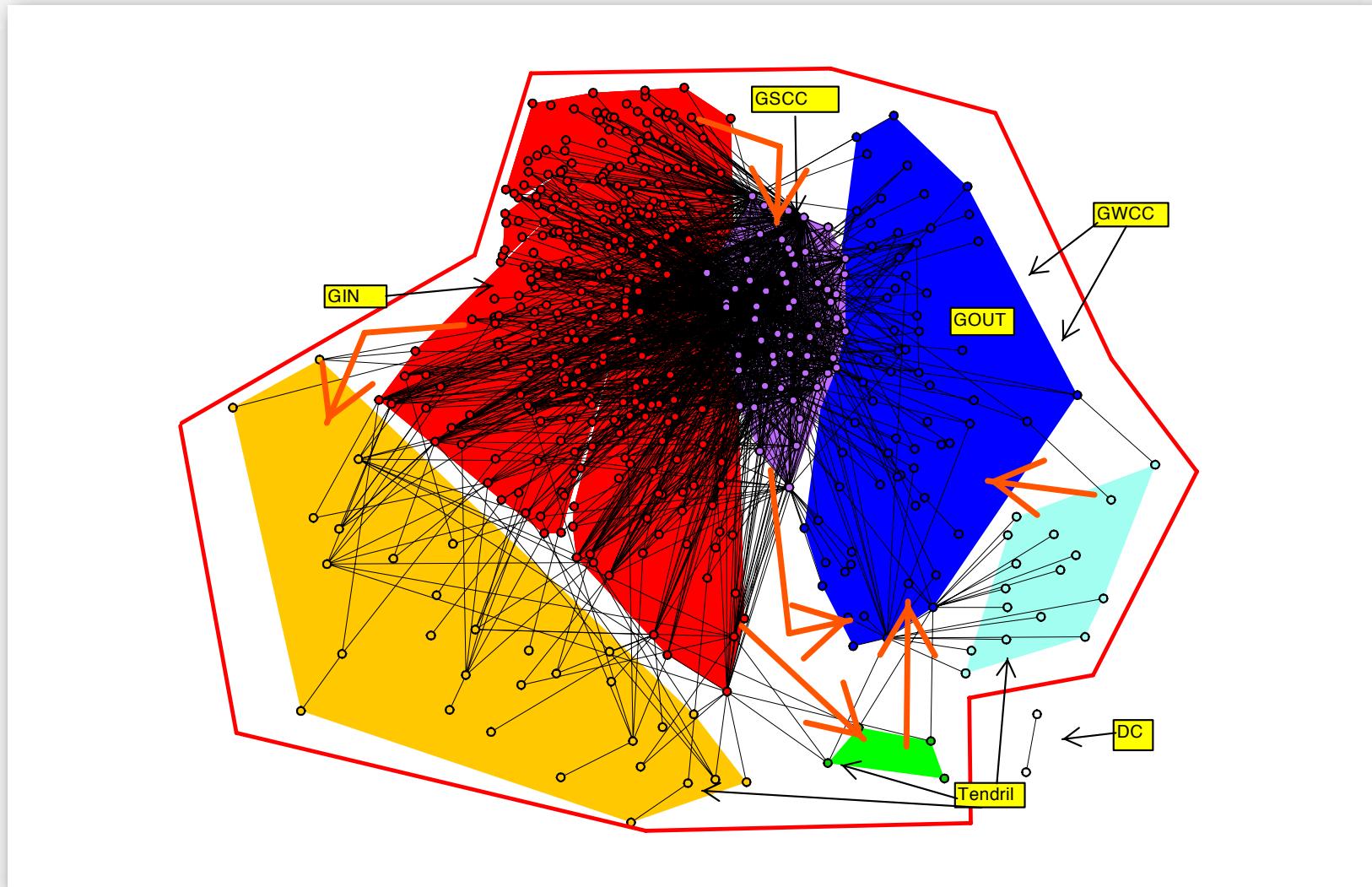
Vertex = political blog; edge = link.



The Political Blogosphere and the 2004 U.S. Election: Divided They Blog, Adamic and Glance, 2005

Overnight interbank loan graph

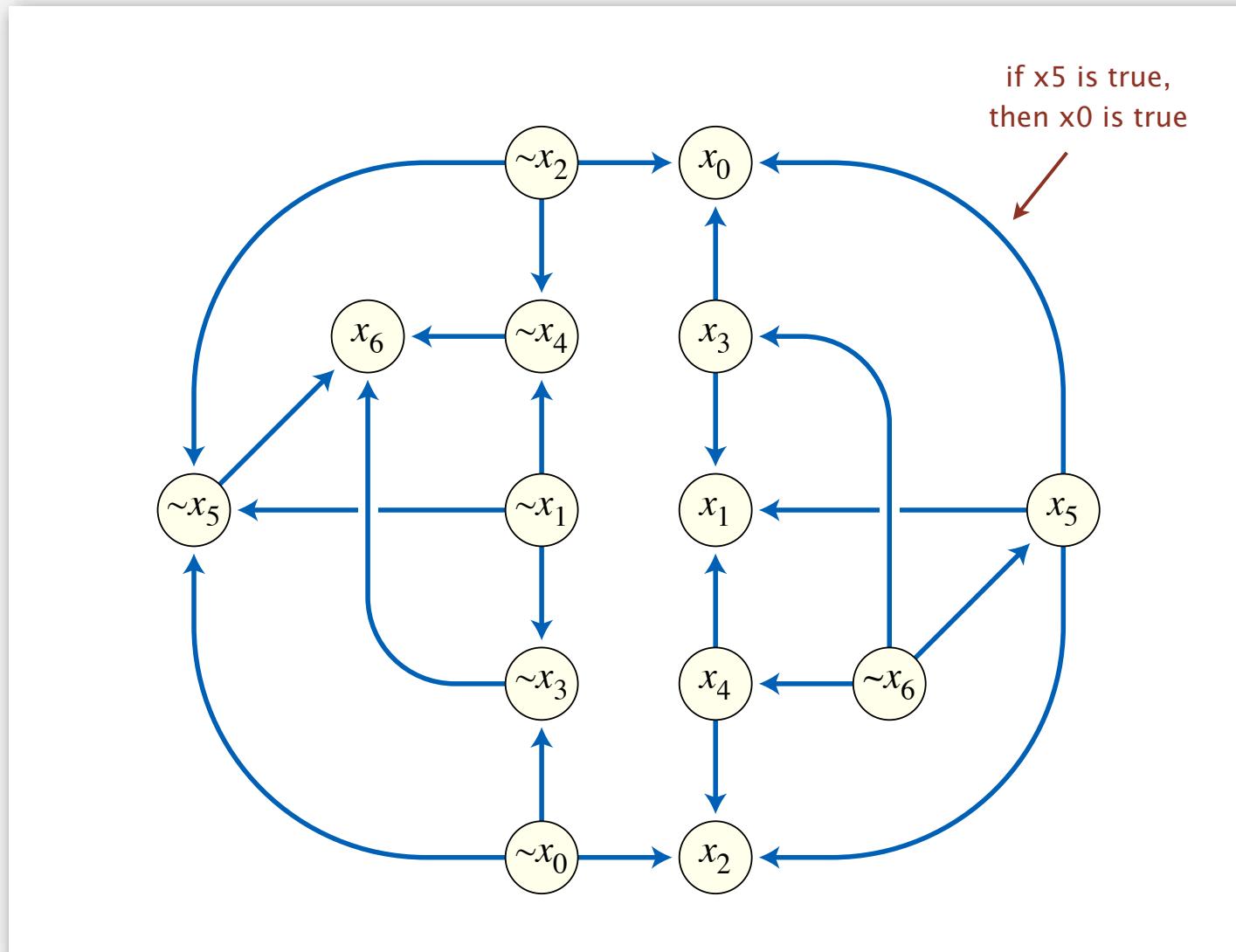
Vertex = bank; edge = overnight loan.



The Topology of the Federal Funds Market, Bech and Atalay, 2008

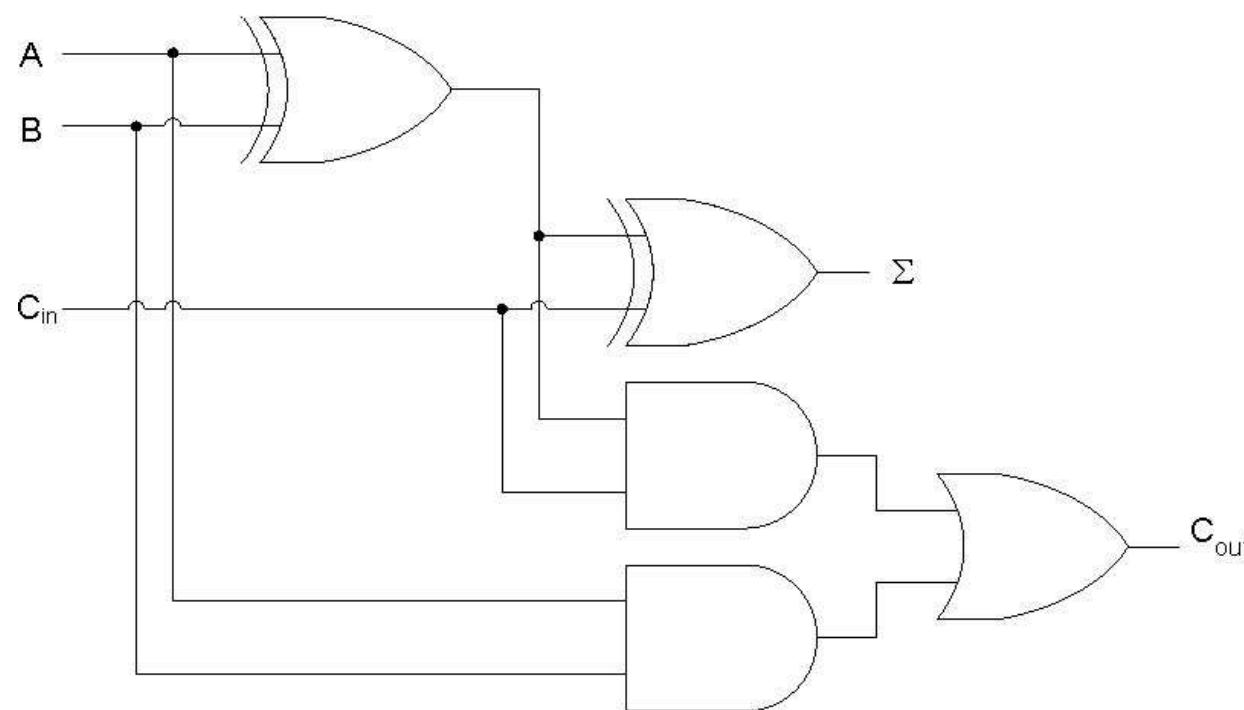
Implication graph

Vertex = variable; edge = logical implication.



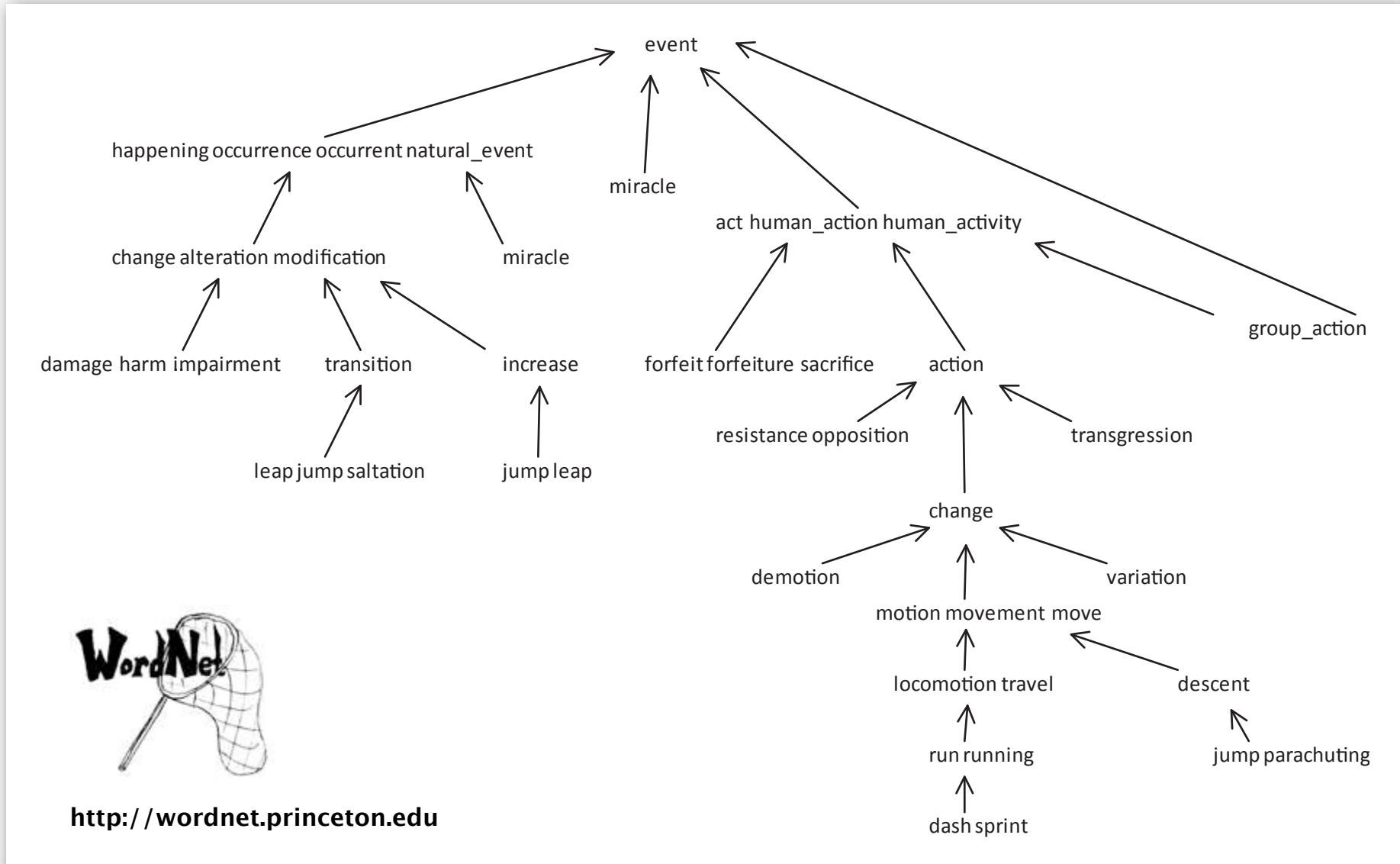
Combinational circuit

Vertex = logical gate; edge = wire.



WordNet graph

Vertex = synset; edge = hypernym relationship.

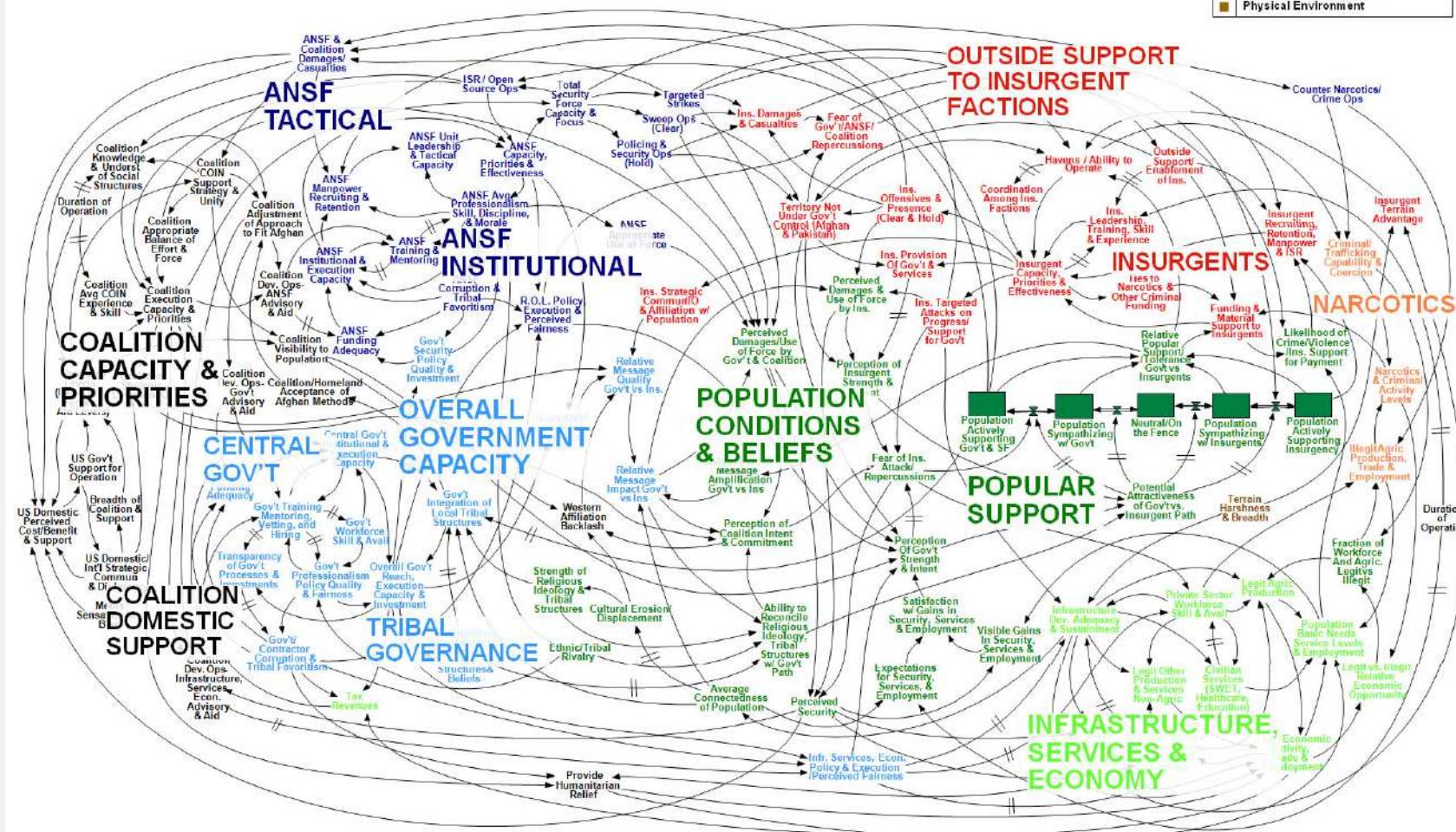


The McChrystal Afghanistan PowerPoint slide

Afghanistan Stability / COIN Dynamics

= Significant Delay

- Population/Popular Support
- Infrastructure, Economy, & Services
- Government
- Afghanistan Security Forces
- Insurgents
- Crime and Narcotics
- Coalition Forces & Actions
- Physical Environment



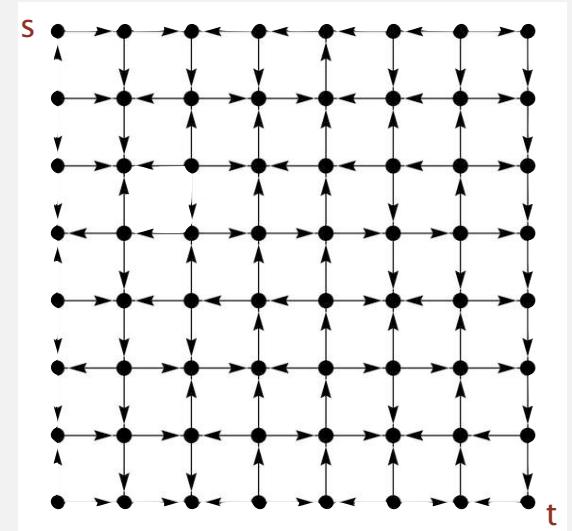
WORKING DRAFT – V3

Digraph applications

digraph	vertex	directed edge
transportation	street intersection	one-way street
web	web page	hyperlink
food web	species	predator-prey relationship
WordNet	synset	hypernym
scheduling	task	precedence constraint
financial	bank	transaction
cell phone	person	placed call
infectious disease	person	infection
game	board position	legal move
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

Some digraph problems

Path. Is there a directed path from s to t ?



Shortest path. What is the shortest directed path from s to t ?

Topological sort. Can you draw a digraph so that all edges point upwards?

Strong connectivity. Is there a directed path between all pairs of vertices?

Transitive closure. For which vertices v and w is there a path from v to w ?

PageRank. What is the importance of a web page?

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Digraph API

```
public class Digraph
```

```
    Digraph(int V)
```

create an empty digraph with V vertices

```
    Digraph(In in)
```

create a digraph from input stream

```
    void addEdge(int v, int w)
```

add a directed edge $v \rightarrow w$

```
    Iterable<Integer> adj(int v)
```

vertices pointing from v

```
    int V()
```

number of vertices

```
    int E()
```

number of edges

```
    Digraph reverse()
```

reverse of this digraph

```
    String toString()
```

string representation

```
In in = new In(args[0]);  
Digraph G = new Digraph(in);
```

read digraph from
input stream

```
for (int v = 0; v < G.V(); v++)  
    for (int w : G.adj(v))  
        StdOut.println(v + "->" + w);
```

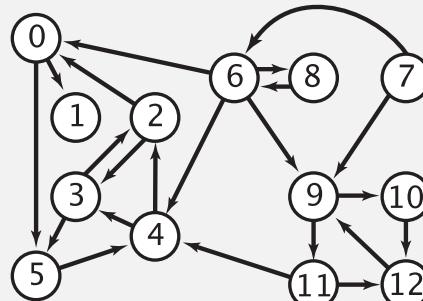
print out each
edge (once)

Digraph API

tinyDG.txt

V → 13
← *E*

```
22  
4 2  
2 3  
3 2  
6 0  
0 1  
2 0  
11 12  
12 9  
9 10  
9 11  
7 9  
10 12  
11 4  
4 3  
3 5  
6 8  
8 6  
:
```



```
% java Digraph tinyDG.txt
```

```
0->5  
0->1  
2->0  
2->3  
3->5  
3->2  
4->3  
4->2  
5->4  
:  
11->4  
11->12  
12->9
```

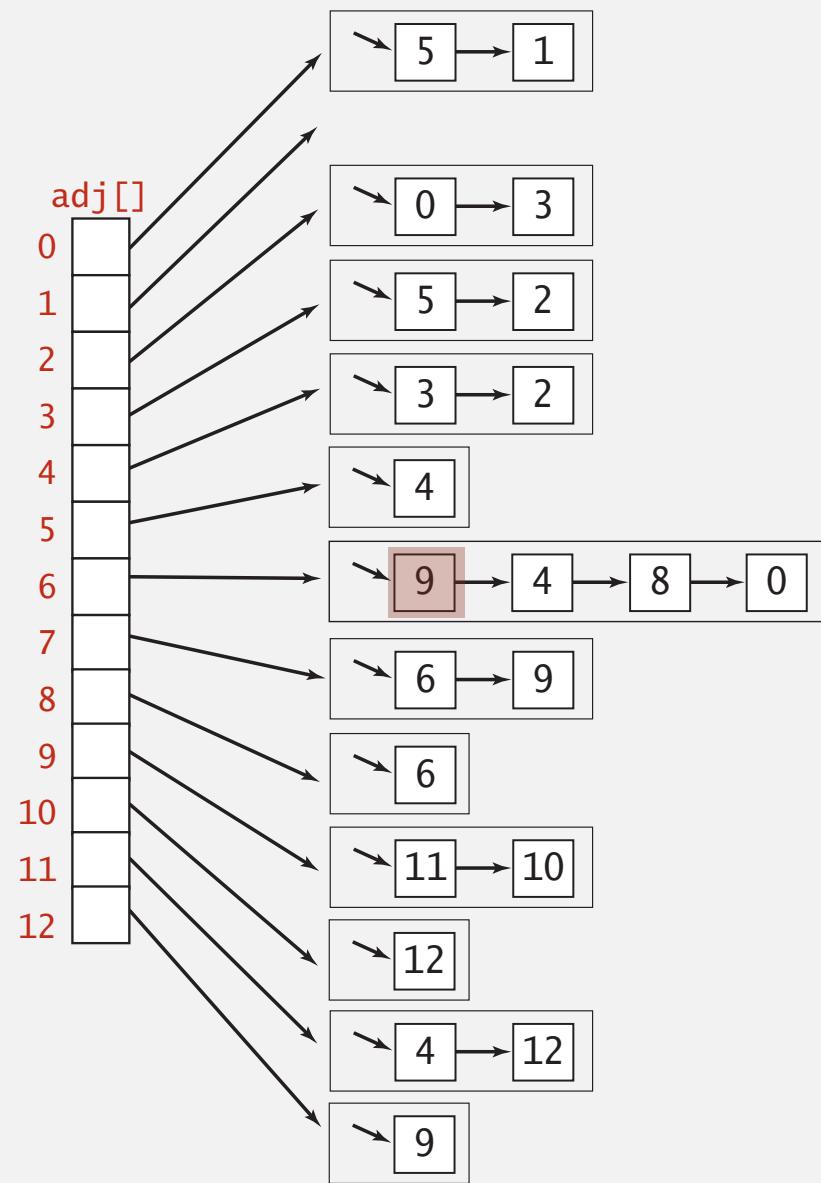
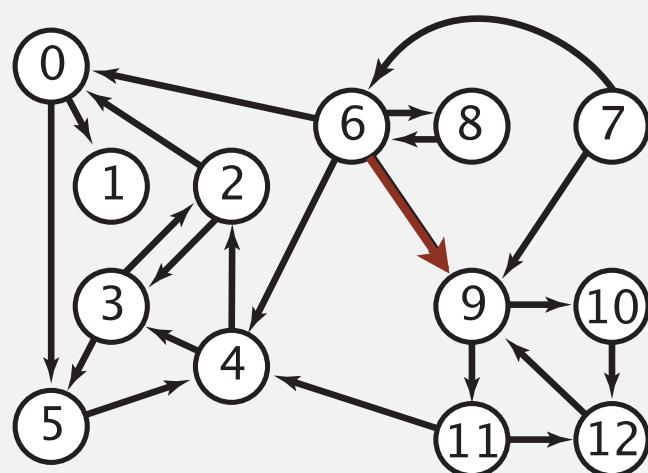
```
In in = new In(args[0]);  
Digraph G = new Digraph(in);  
  
for (int v = 0; v < G.V(); v++)  
    for (int w : G.adj(v))  
        StdOut.println(v + "->" + w);
```

read digraph from
input stream

print out each
edge (once)

Adjacency-lists digraph representation

Maintain vertex-indexed array of lists.



Adjacency-lists graph representation (review): Java implementation

```
public class Graph
{
    private final int V;
    private final Bag<Integer>[] adj;           ← adjacency lists

    public Graph(int V)
    {
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }

    public void addEdge(int v, int w)             ← add edge v-w
    {
        adj[v].add(w);
        adj[w].add(v);
    }

    public Iterable<Integer> adj(int v)          ← iterator for vertices
    {   return adj[v];  }
}
```

Adjacency-lists digraph representation: Java implementation

```
public class Digraph
{
    private final int V;
    private final Bag<Integer>[] adj;           ← adjacency lists

    public Digraph(int V)
    {
        this.V = V;
        adj = (Bag<Integer>[]) new Bag[V];
        for (int v = 0; v < V; v++)
            adj[v] = new Bag<Integer>();
    }

    public void addEdge(int v, int w)             ← add edge v→w
    {
        adj[v].add(w);
    }

    public Iterable<Integer> adj(int v)          ← iterator for vertices
    {   return adj[v];  }                         pointing from v
}
```

Digraph representations

In practice. Use adjacency-lists representation.

- Algorithms based on iterating over vertices pointing from v .
- Real-world digraphs tend to be sparse.

huge number of vertices,
small average vertex degree

representation	space	insert edge from v to w	edge from v to w ?	iterate over vertices pointing from v ?
list of edges	E	1	E	E
adjacency matrix	V^2	1^\dagger	1	V
adjacency lists	$E + V$	1	$\text{outdegree}(v)$	$\text{outdegree}(v)$

\dagger disallows parallel edges

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

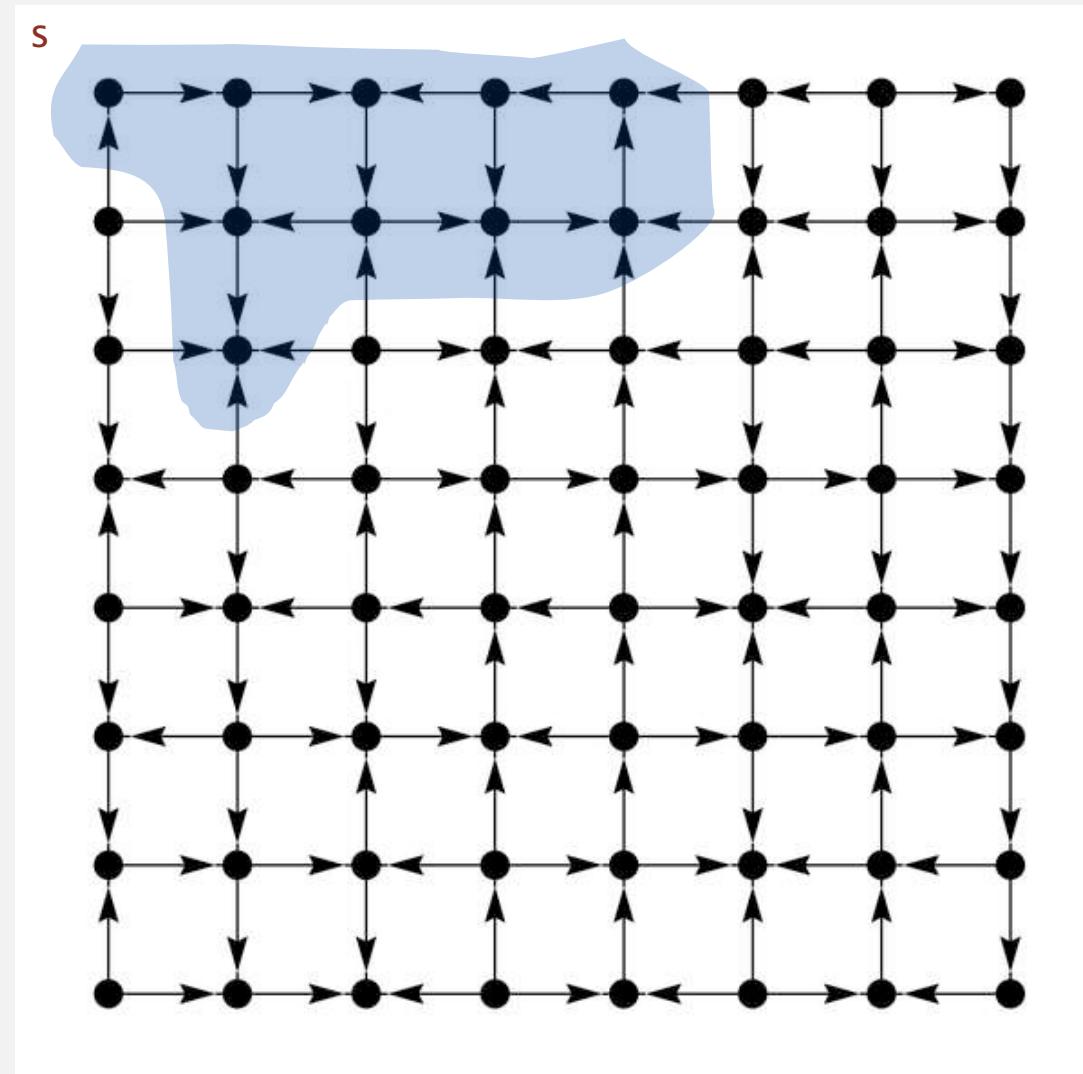
<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Reachability

Problem. Find all vertices reachable from s along a directed path.



Depth-first search in digraphs

Same method as for undirected graphs.

- Every undirected graph is a digraph (with edges in both directions).
- DFS is a **digraph** algorithm.

DFS (to visit a vertex v)

Mark v as visited.

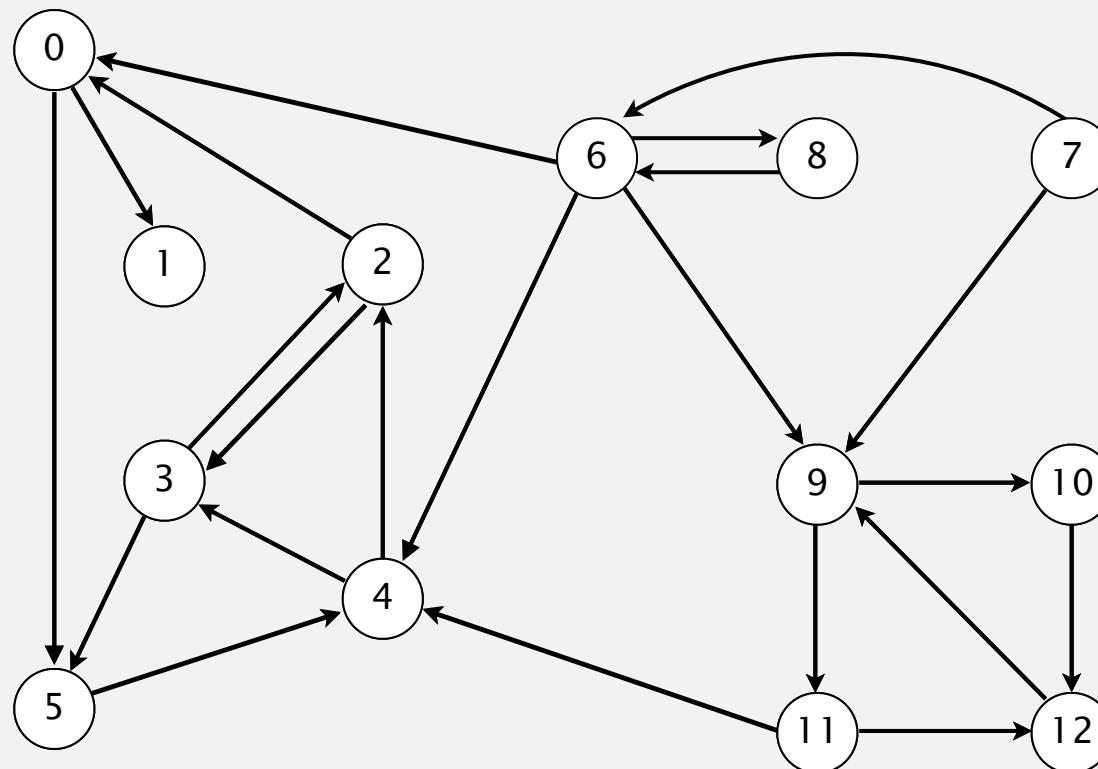
Recursively visit all unmarked

vertices w pointing from v.

Depth-first search demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



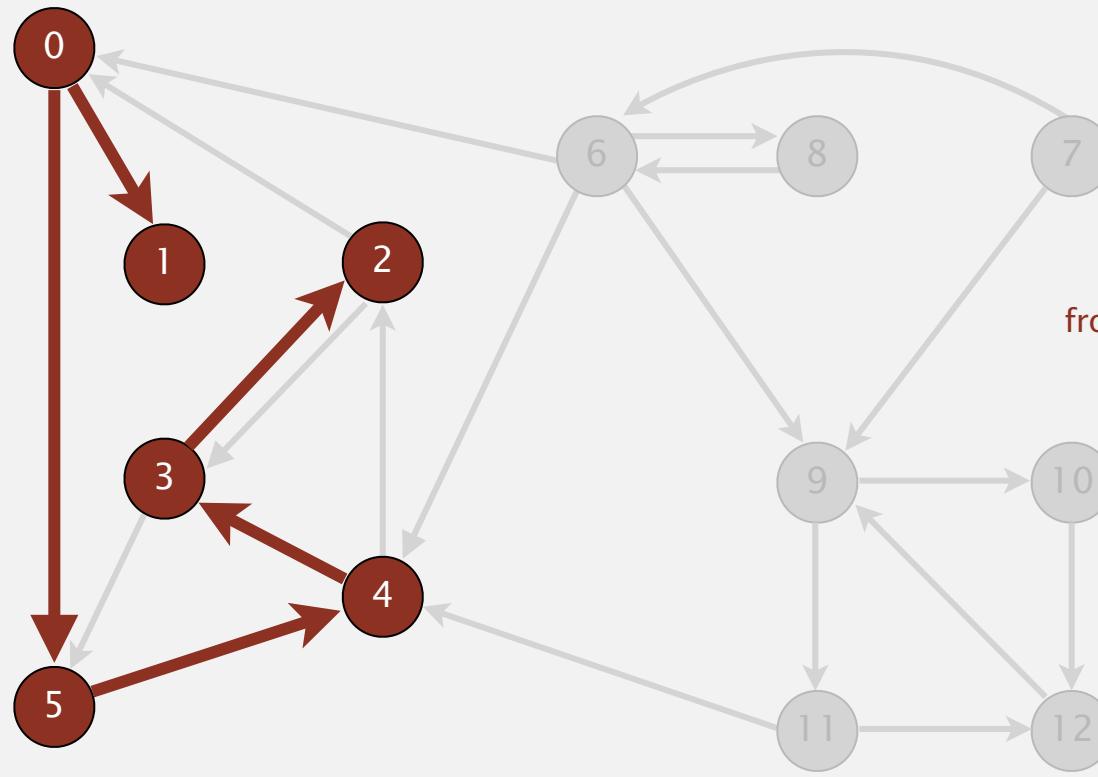
a directed graph

4→2
2→3
3→2
6→0
0→1
2→0
11→12
12→9
9→10
9→11
8→9
10→12
11→4
4→3
3→5
6→8
8→6
5→4
0→5
6→4
6→9
7→6

Depth-first search demo

To visit a vertex v :

- Mark vertex v as visited.
- Recursively visit all unmarked vertices pointing from v .



v	marked[]	edgeTo[]
0	T	-
1	T	0
2	T	3
3	T	4
4	T	5
5	T	0
6	F	-
7	F	-
8	F	-
9	F	-
10	F	-
11	F	-
12	F	-

Depth-first search (in undirected graphs)

Recall code for **undirected** graphs.

```
public class DepthFirstSearch
{
    private boolean[] marked;           ← true if path to s

    public DepthFirstSearch(Graph G, int s)
    {
        marked = new boolean[G.V()];
        dfs(G, s);
    }

    private void dfs(Graph G, int v)     ← recursive DFS does the work
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w);
    }

    public boolean visited(int v)        ← client can ask whether any
    {   return marked[v];   }           vertex is connected to s
}
```

Depth-first search (in directed graphs)

Code for **directed** graphs identical to undirected one.

[substitute Digraph for Graph]

```
public class DirectedDFS
{
    private boolean[] marked;           ← true if path from s

    public DirectedDFS(Digraph G, int s)
    {
        marked = new boolean[G.V()];
        dfs(G, s);
    }

    private void dfs(Digraph G, int v)   ← recursive DFS does the work
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w);
    }

    public boolean visited(int v)       ← client can ask whether any
    { return marked[v]; }             vertex is reachable from s
}
```

Reachability application: program control-flow analysis

Every program is a digraph.

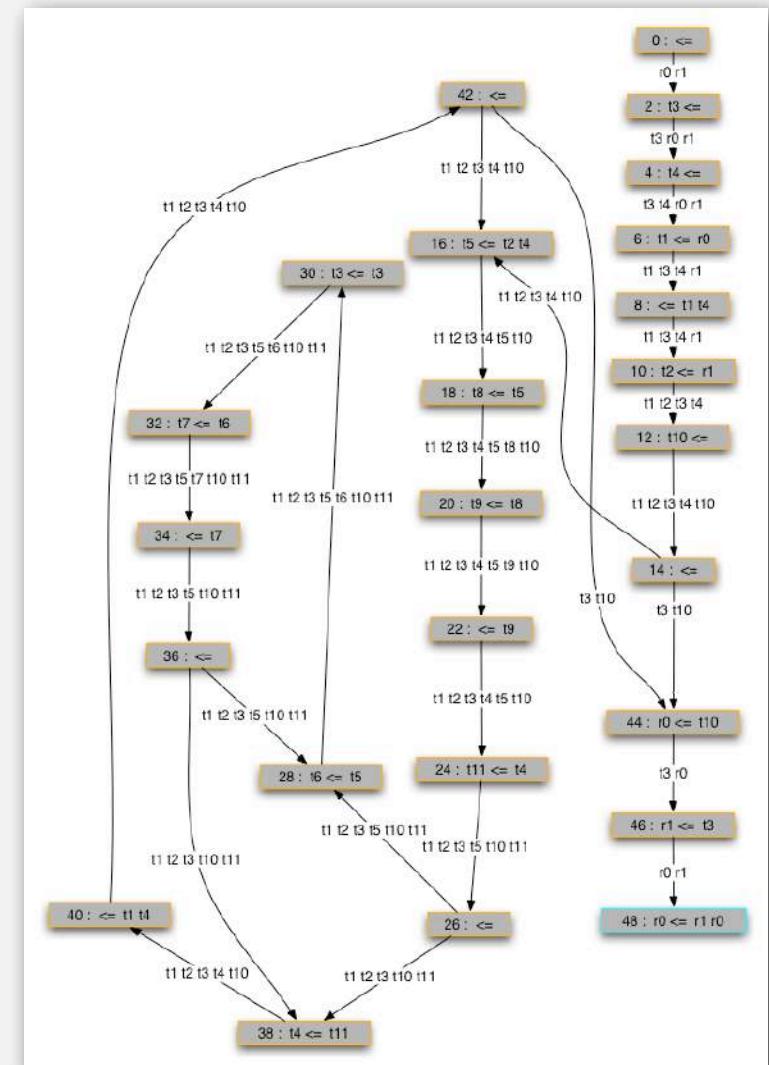
- Vertex = basic block of instructions (straight-line program).
- Edge = jump.

Dead-code elimination.

Find (and remove) unreachable code.

Infinite-loop detection.

Determine whether exit is unreachable.



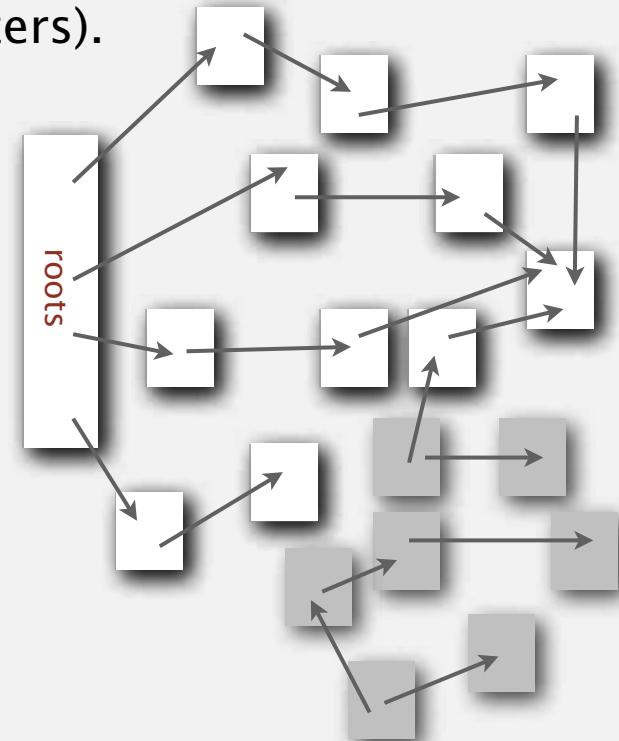
Reachability application: mark-sweep garbage collector

Every data structure is a digraph.

- Vertex = object.
- Edge = reference.

Roots. Objects known to be directly accessible by program (e.g., stack).

Reachable objects. Objects indirectly accessible by program (starting at a root and following a chain of pointers).

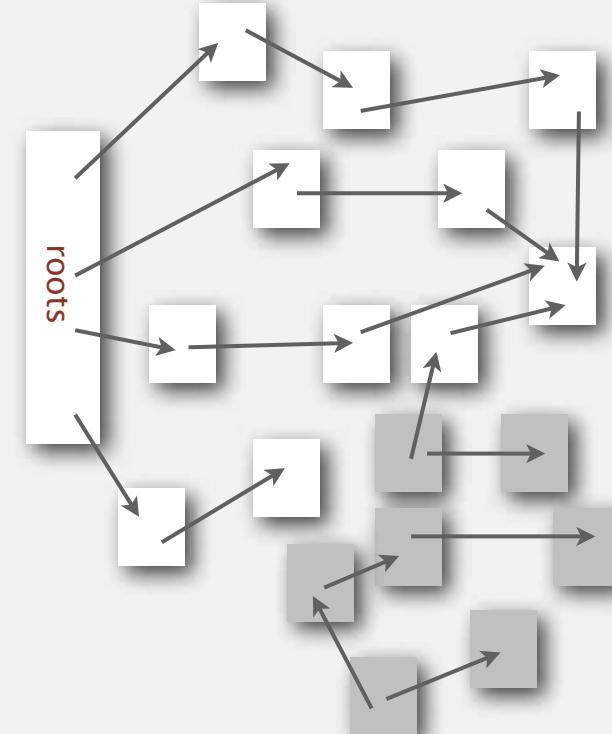


Reachability application: mark-sweep garbage collector

Mark-sweep algorithm. [McCarthy, 1960]

- Mark: mark all reachable objects.
- Sweep: if object is unmarked, it is garbage (so add to free list).

Memory cost. Uses 1 extra mark bit per object (plus DFS stack).



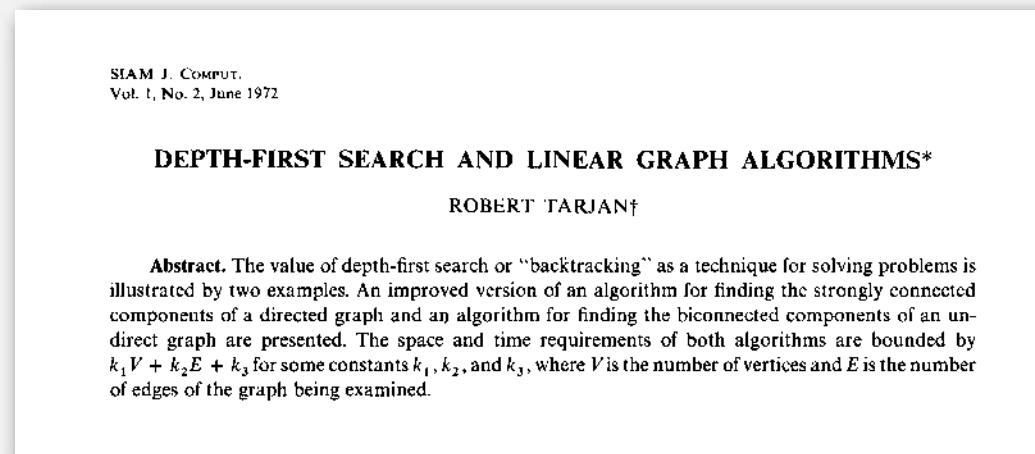
Depth-first search in digraphs summary

DFS enables direct solution of simple digraph problems.

- ✓ • Reachability.
- Path finding.
- Topological sort.
- Directed cycle detection.

Basis for solving difficult digraph problems.

- 2-satisfiability.
- Directed Euler path.
- Strongly-connected components.



Breadth-first search in digraphs

Same method as for undirected graphs.

- Every undirected graph is a digraph (with edges in both directions).
- BFS is a **digraph** algorithm.

BFS (from source vertex s)

Put s onto a FIFO queue, and mark s as visited.

Repeat until the queue is empty:

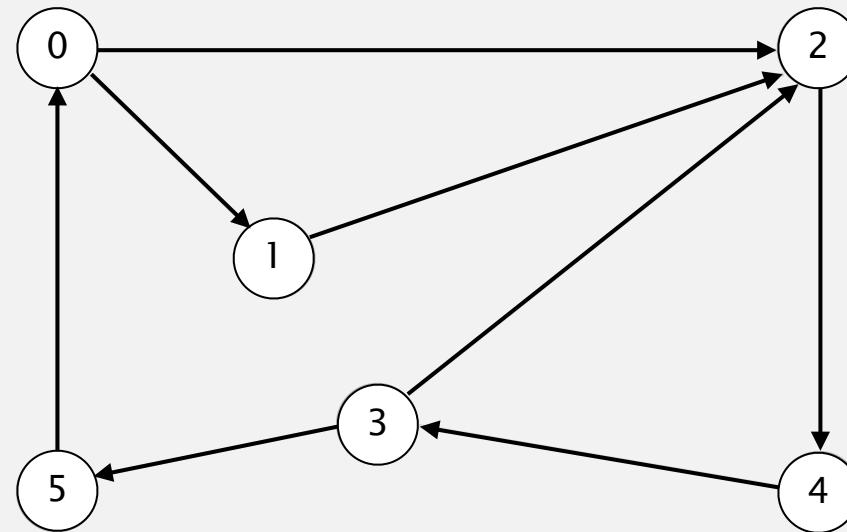
- **remove the least recently added vertex v**
 - **for each unmarked vertex pointing from v:**
add to queue and mark as visited.
-

Proposition. BFS computes shortest paths (fewest number of edges) from s to all other vertices in a digraph in time proportional to $E + V$.

Directed breadth-first search demo

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices pointing from v and mark them.



tinyDG2.txt

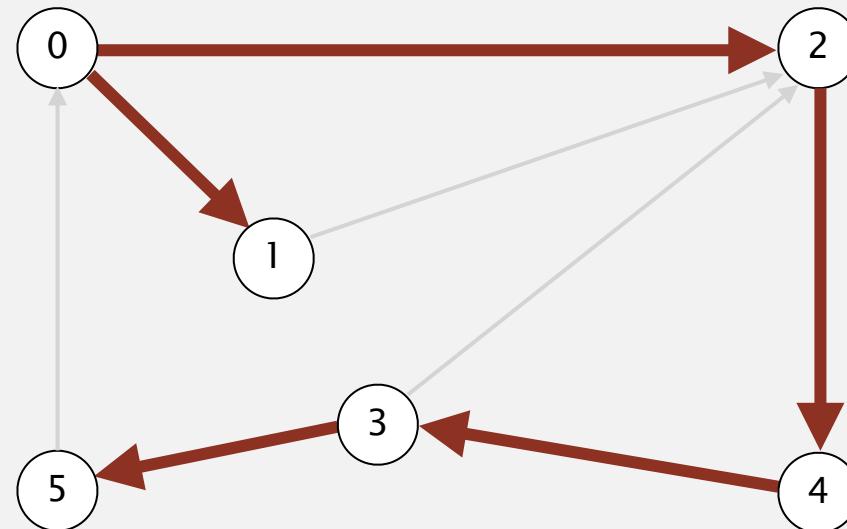
V → 6
E → 8
5 0
2 4
3 2
1 2
0 1
4 3
3 5
0 2

graph G

Directed breadth-first search demo

Repeat until queue is empty:

- Remove vertex v from queue.
- Add to queue all unmarked vertices pointing from v and mark them.



v	edgeTo[]	distTo[]
0	-	0
1	0	1
2	0	1
3	4	3
4	2	2
5	3	4

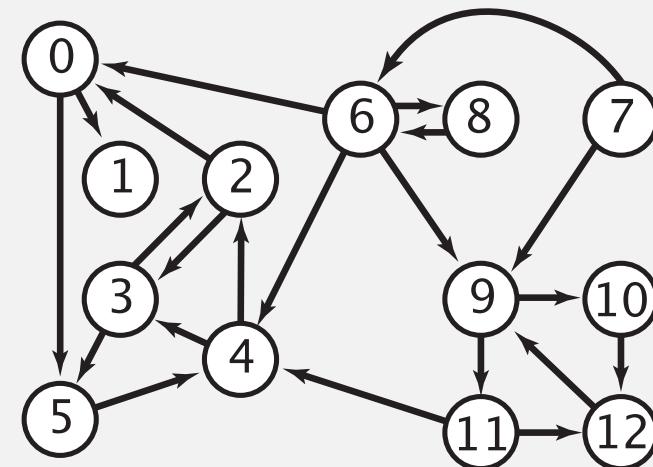
done

Multiple-source shortest paths

Multiple-source shortest paths. Given a digraph and a **set** of source vertices, find shortest path from any vertex in the set to each other vertex.

Ex. $S = \{1, 7, 10\}$.

- Shortest path to 4 is $7 \rightarrow 6 \rightarrow 4$.
- Shortest path to 5 is $7 \rightarrow 6 \rightarrow 0 \rightarrow 5$.
- Shortest path to 12 is $10 \rightarrow 12$.
- ...



Q. How to implement multi-source shortest paths algorithm?

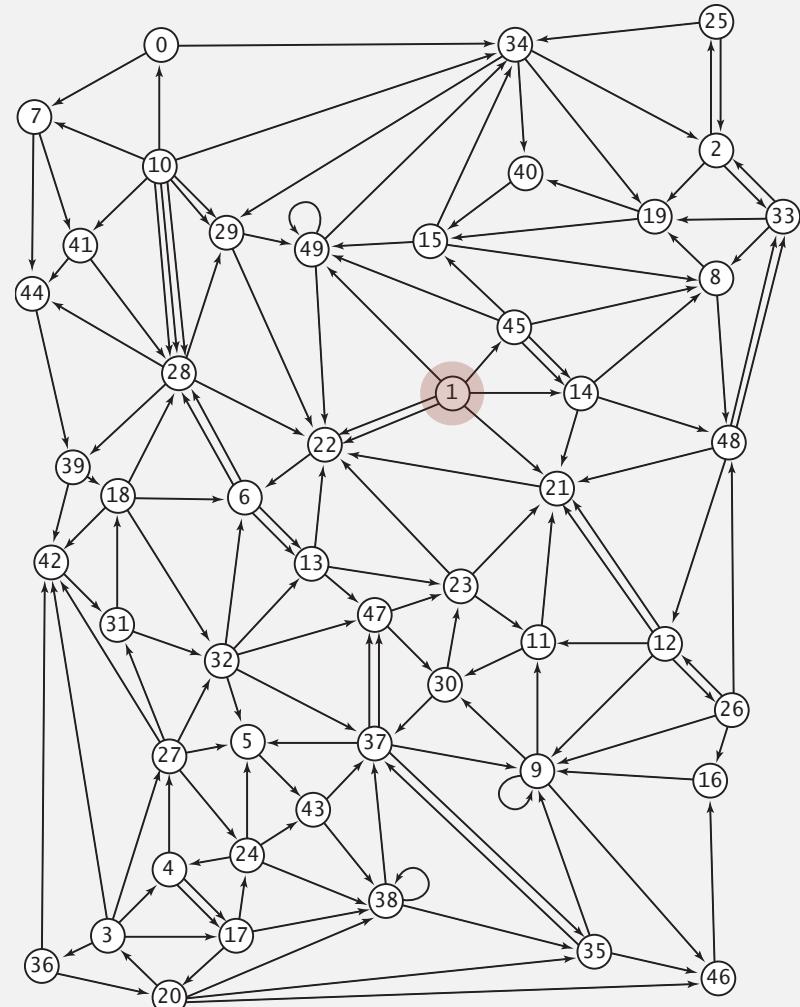
A. Use BFS, but initialize by enqueueing all source vertices.

Breadth-first search in digraphs application: web crawler

Goal. Crawl web, starting from some root web page, say `www.princeton.edu`.

Solution. [BFS with implicit digraph]

- Choose root web page as source s .
- Maintain a Queue of websites to explore.
- Maintain a SET of discovered websites.
- Dequeue the next website and enqueue websites to which it links
(provided you haven't done so before).



Q. Why not use DFS?

Bare-bones web crawler: Java implementation

```
Queue<String> queue = new Queue<String>(); ← queue of websites to crawl
SET<String> marked = new SET<String>(); ← set of marked websites

String root = "http://www.princeton.edu";
queue.enqueue(root);
marked.add(root);

while (!queue.isEmpty())
{
    String v = queue.dequeue();
    StdOut.println(v);
    In in = new In(v);
    String input = in.readAll();

    String regexp = "http://(\w+\.\w+)*(\w+)";
    Pattern pattern = Pattern.compile(regexp);
    Matcher matcher = pattern.matcher(input); ← use regular expression to find all URLs
                                                in website of form http://xxx.yyy.zzz
                                                [crude pattern misses relative URLs]

    while (matcher.find())
    {
        String w = matcher.group();
        if (!marked.contains(w))
        {
            marked.add(w);
            queue.enqueue(w); ← if unmarked, mark it and put
                                on the queue
        }
    }
}
```

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

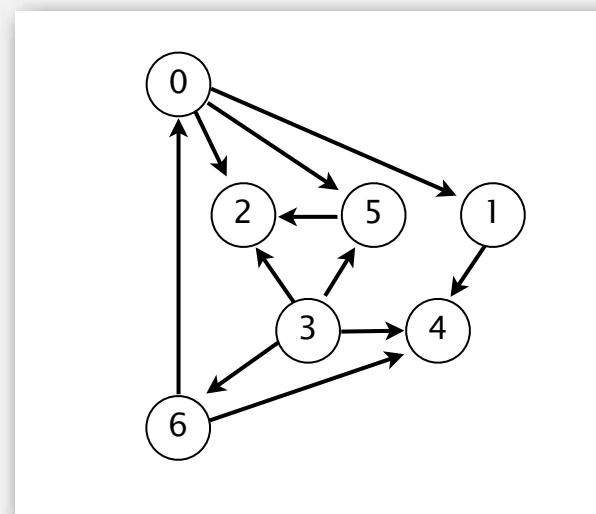
Precedence scheduling

Goal. Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks?

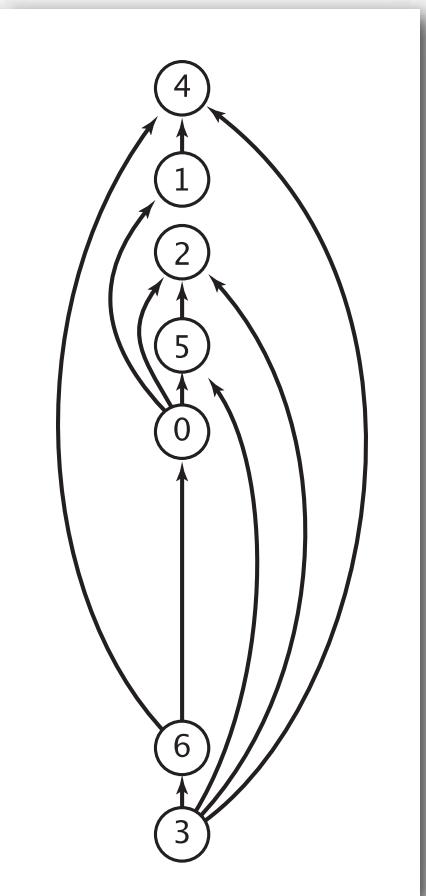
Digraph model. vertex = task; edge = precedence constraint.

- 0. Algorithms
- 1. Complexity Theory
- 2. Artificial Intelligence
- 3. Intro to CS
- 4. Cryptography
- 5. Scientific Computing
- 6. Advanced Programming

tasks



precedence constraint graph



feasible schedule

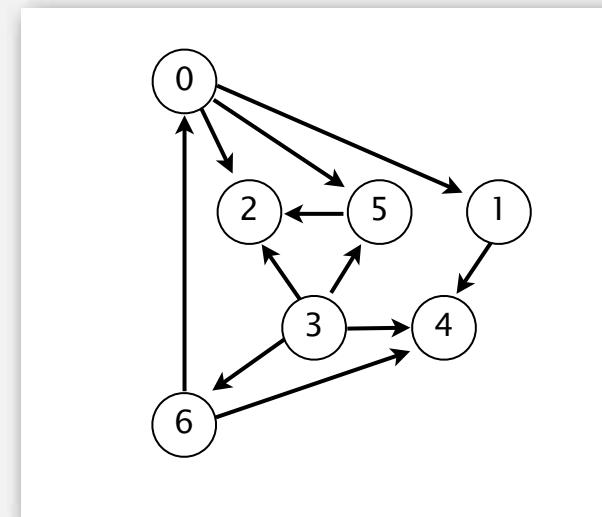
Topological sort

DAG. Directed **acyclic** graph.

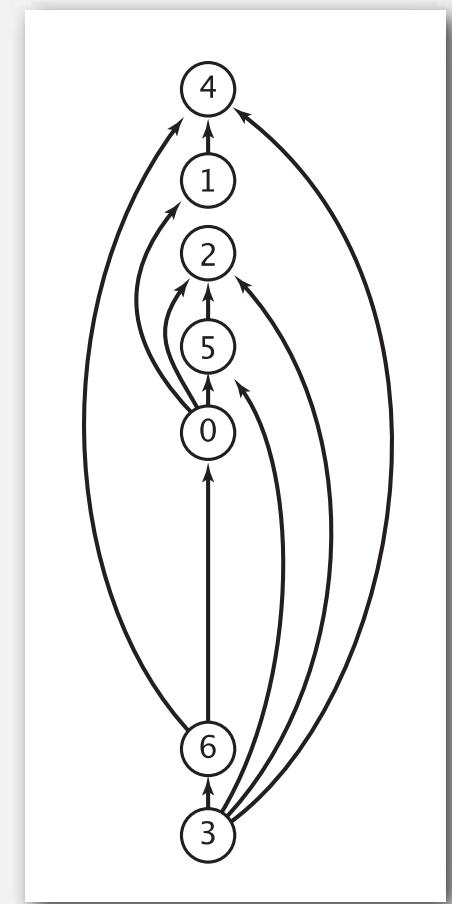
Topological sort. Redraw DAG so all edges point upwards.

0→5	0→2
0→1	3→6
3→5	3→4
5→4	6→4
6→0	3→2
1→4	

directed edges



DAG

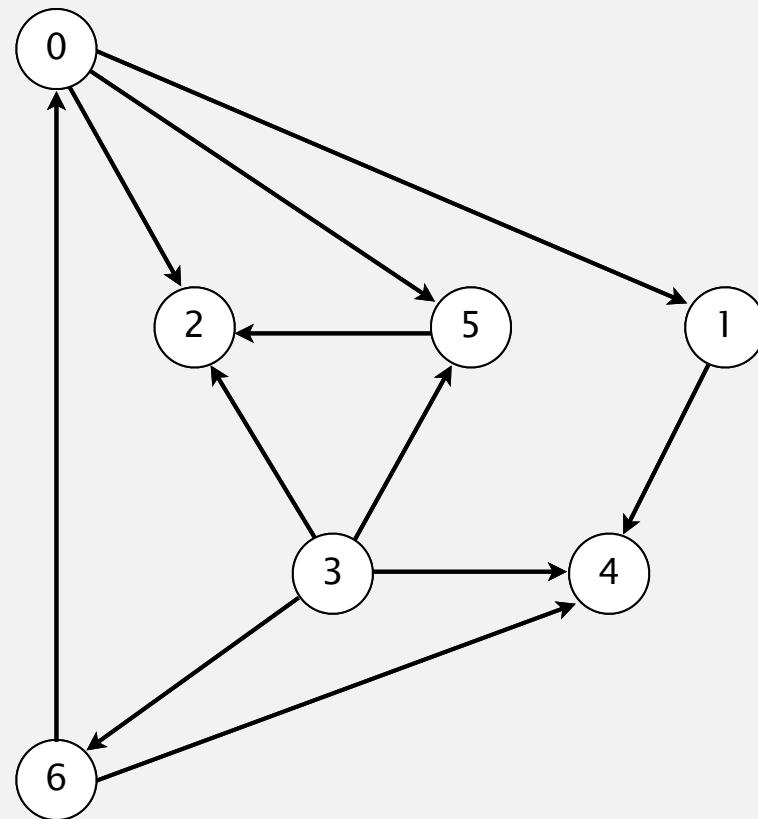


topological order

Solution. DFS. What else?

Topological sort demo

- Run depth-first search.
- Return vertices in reverse postorder.

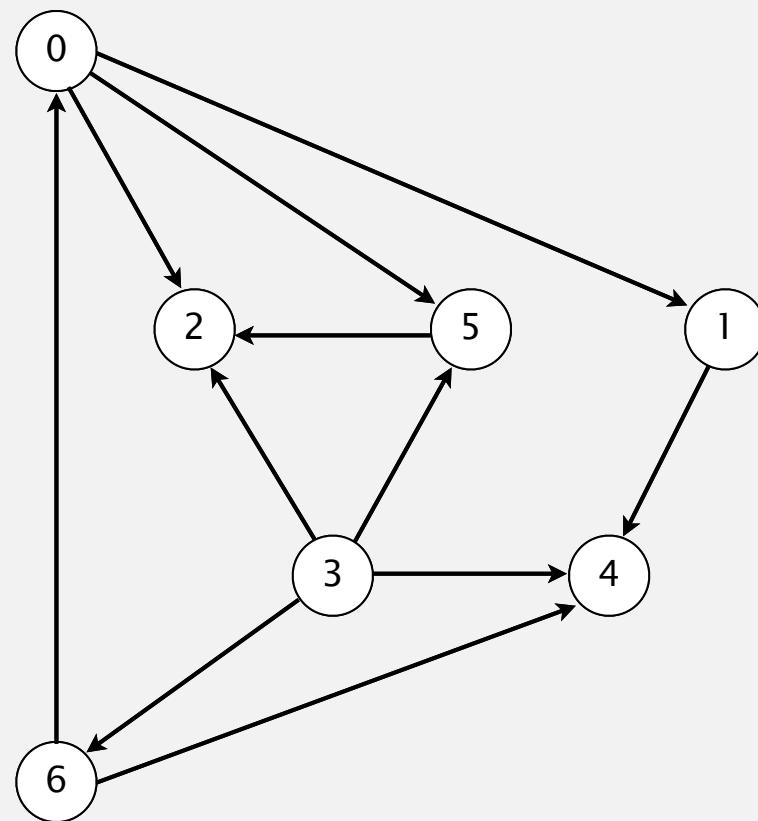


$0 \rightarrow 5$
 $0 \rightarrow 2$
 $0 \rightarrow 1$
 $3 \rightarrow 6$
 $3 \rightarrow 5$
 $3 \rightarrow 4$
 $5 \rightarrow 4$
 $6 \rightarrow 4$
 $6 \rightarrow 0$
 $3 \rightarrow 2$
 $1 \rightarrow 4$

a directed acyclic graph

Topological sort demo

- Run depth-first search.
- Return vertices in reverse postorder.



postorder

4 1 2 5 0 6 3

topological order

3 6 0 5 2 1 4

done

Depth-first search order

```
public class DepthFirstOrder
{
    private boolean[] marked;
    private Stack<Integer> reversePost;

    public DepthFirstOrder(Digraph G)
    {
        reversePost = new Stack<Integer>();
        marked = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) dfs(G, v);
    }

    private void dfs(Digraph G, int v)
    {
        marked[v] = true;
        for (int w : G.adj(v))
            if (!marked[w]) dfs(G, w);
        reversePost.push(v);
    }

    public Iterable<Integer> reversePost()
    { return reversePost; }
}
```

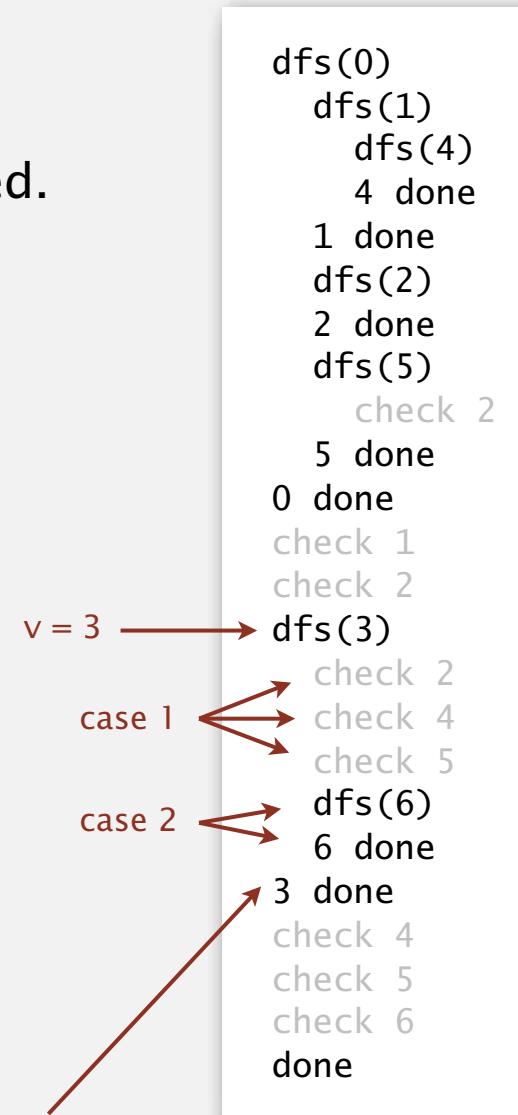
returns all vertices in
“reverse DFS postorder”

Topological sort in a DAG: correctness proof

Proposition. Reverse DFS postorder of a DAG is a topological order.

Pf. Consider any edge $v \rightarrow w$. When $\text{dfs}(v)$ is called:

- Case 1: $\text{dfs}(w)$ has already been called and returned.
Thus, w was done before v .
- Case 2: $\text{dfs}(w)$ has not yet been called.
 $\text{dfs}(w)$ will get called directly or indirectly
by $\text{dfs}(v)$ and will finish before $\text{dfs}(v)$.
Thus, w will be done before v .
- Case 3: $\text{dfs}(w)$ has already been called,
but has not yet returned.
Can't happen in a DAG: function call stack contains
path from w to v , so $v \rightarrow w$ would complete a cycle.



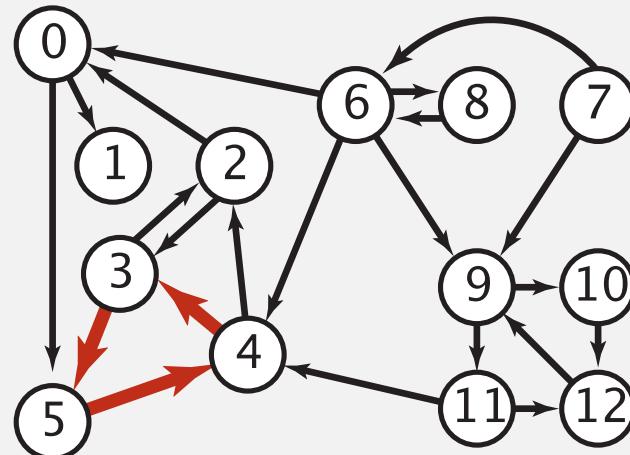
all vertices pointing from 3 are done before 3 is done,
so they appear after 3 in topological order

Directed cycle detection

Proposition. A digraph has a topological order iff no directed cycle.

Pf.

- If directed cycle, topological order impossible.
- If no directed cycle, DFS-based algorithm finds a topological order.



a digraph with a directed cycle

Goal. Given a digraph, find a directed cycle.

Solution. DFS. What else? See textbook.

Directed cycle detection application: precedence scheduling

Scheduling. Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks?

PAGE 3

DEPARTMENT	COURSE	DESCRIPTION	PREREQS
COMPUTER SCIENCE	CPSC 432	INTERMEDIATE COMPILER DESIGN, WITH A FOCUS ON DEPENDENCY RESOLUTION.	CPSC 432

<http://xkcd.com/754>

Remark. A directed cycle implies scheduling problem is infeasible.

Directed cycle detection application: cyclic inheritance

The Java compiler does cycle detection.

```
public class A extends B
{
    ...
}
```

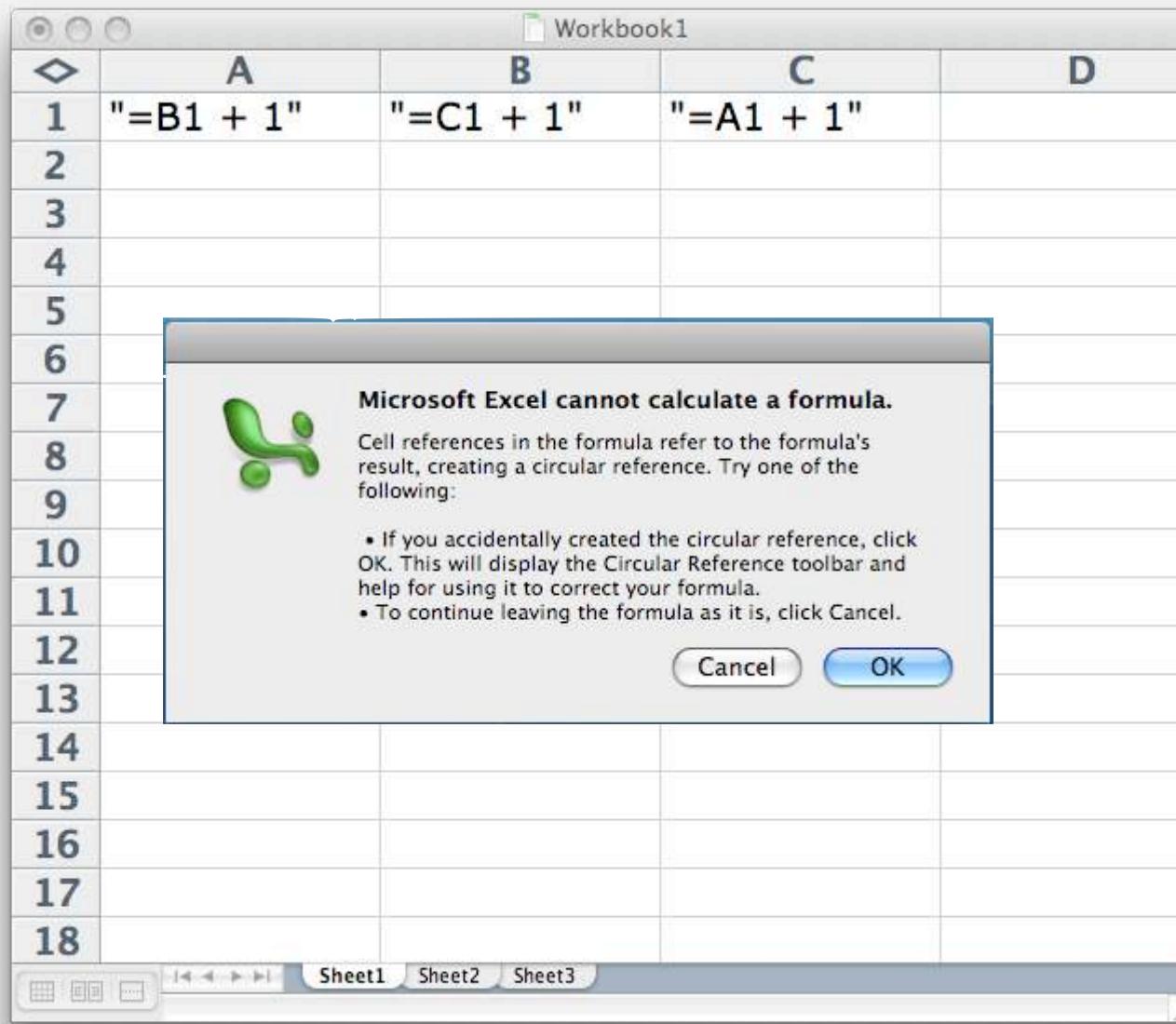
```
public class B extends C
{
    ...
}
```

```
public class C extends A
{
    ...
}
```

```
% javac A.java
A.java:1: cyclic inheritance
involving A
public class A extends B { }
^
1 error
```

Directed cycle detection application: spreadsheet recalculation

Microsoft Excel does cycle detection (and has a circular reference toolbar!)



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ ***strong components***

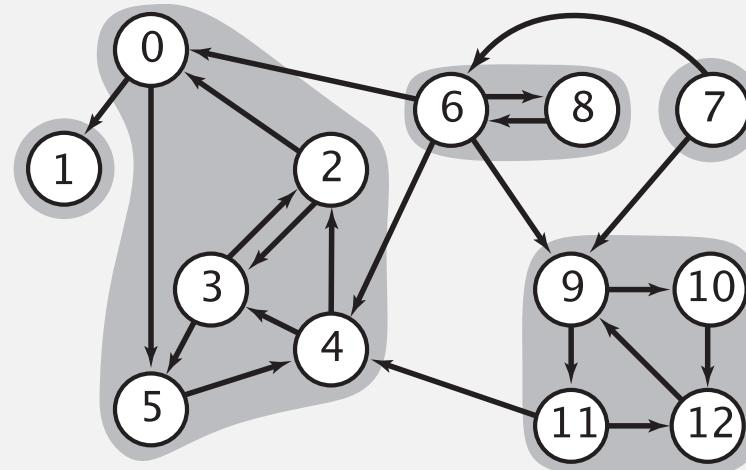
Strongly-connected components

Def. Vertices v and w are **strongly connected** if there is both a directed path from v to w **and** a directed path from w to v .

Key property. Strong connectivity is an **equivalence relation**:

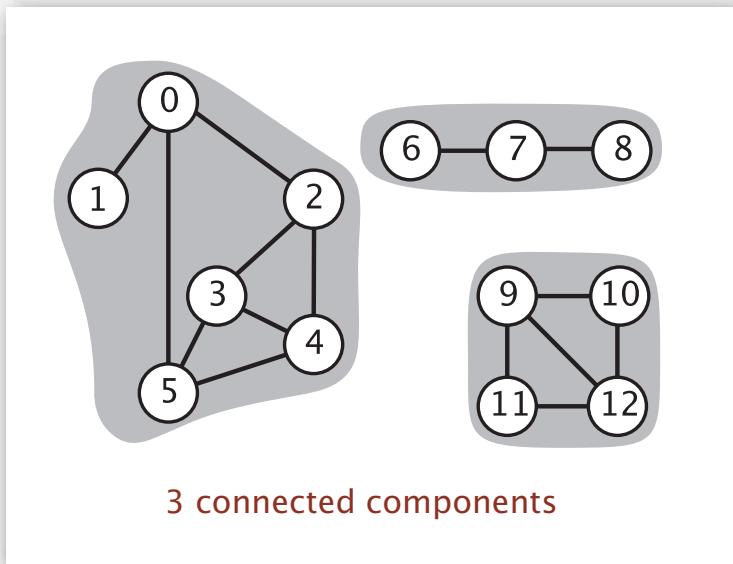
- v is strongly connected to v .
- If v is strongly connected to w , then w is strongly connected to v .
- If v is strongly connected to w and w to x , then v is strongly connected to x .

Def. A **strong component** is a maximal subset of strongly-connected vertices.

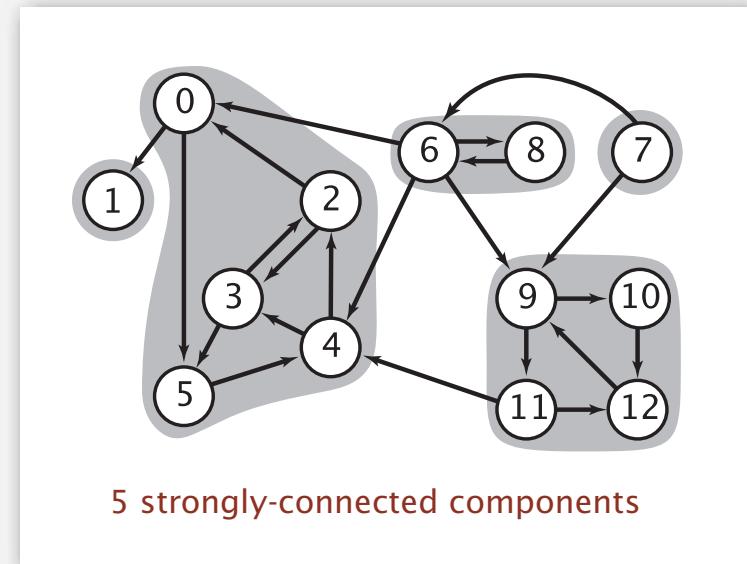


Connected components vs. strongly-connected components

v and w are **connected** if there is a path between v and w



v and w are **strongly connected** if there is both a directed path from v to w and a directed path from w to v



connected component id (easy to compute with DFS)

	0	1	2	3	4	5	6	7	8	9	10	11	12
cc[]	0	0	0	0	0	0	1	1	2	2	2	2	2

```
public int connected(int v, int w)
{   return cc[v] == cc[w]; }
```

constant-time client connectivity query

strongly-connected component id (how to compute?)

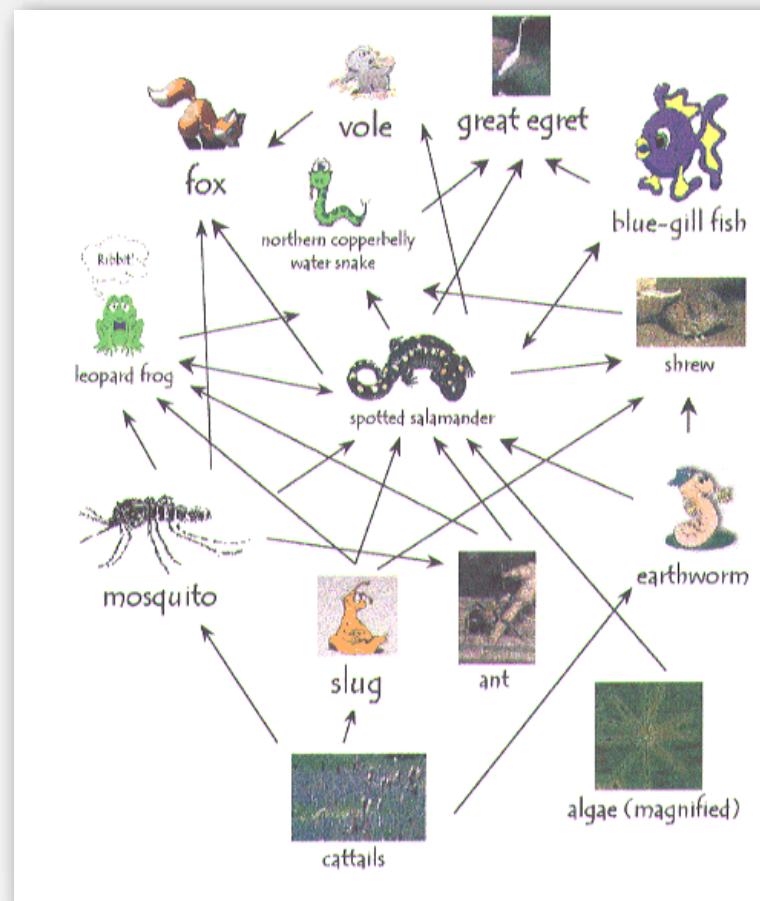
	0	1	2	3	4	5	6	7	8	9	10	11	12
scc[]	1	0	1	1	1	1	3	4	3	2	2	2	2

```
public int stronglyConnected(int v, int w)
{   return scc[v] == scc[w]; }
```

constant-time client strong-connectivity query

Strong component application: ecological food webs

Food web graph. Vertex = species; edge = from producer to consumer.



<http://www.twinkl.com/resource/T2-1000-Wetlands-Salamander-SalGraphics/salfoodweb.gif>

Strong component. Subset of species with common energy flow.

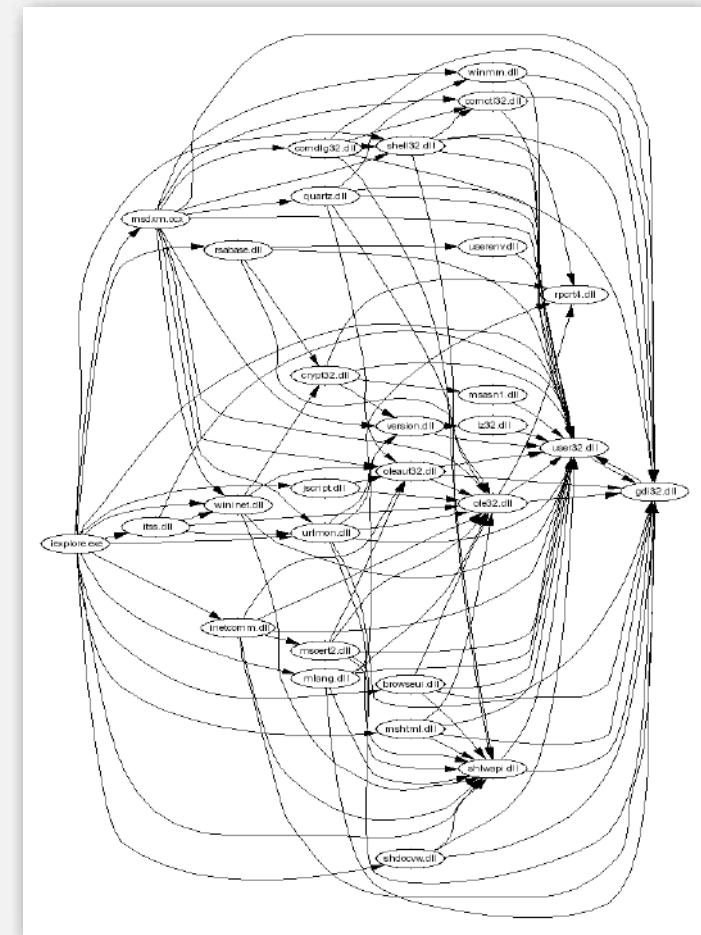
Strong component application: software modules

Software module dependency graph.

- Vertex = software module.
 - Edge: from module to dependency.



Firefox



Internet Explorer

Strong component. Subset of mutually interacting modules.

Approach 1. Package strong components together.

Approach 2. Use to improve design!

Strong components algorithms: brief history

1960s: Core OR problem.

- Widely studied; some practical algorithms.
- Complexity not understood.

1972: linear-time DFS algorithm (Tarjan).

- Classic algorithm.
- Level of difficulty: Algs4++.
- Demonstrated broad applicability and importance of DFS.

1980s: easy two-pass linear-time algorithm (Kosaraju-Sharir).

- Forgot notes for lecture; developed algorithm in order to teach it!
- Later found in Russian scientific literature (1972).

1990s: more easy linear-time algorithms.

- Gabow: fixed old OR algorithm.
- Cheriyan-Mehlhorn: needed one-pass algorithm for LEDA.

Kosaraju-Sharir algorithm: intuition

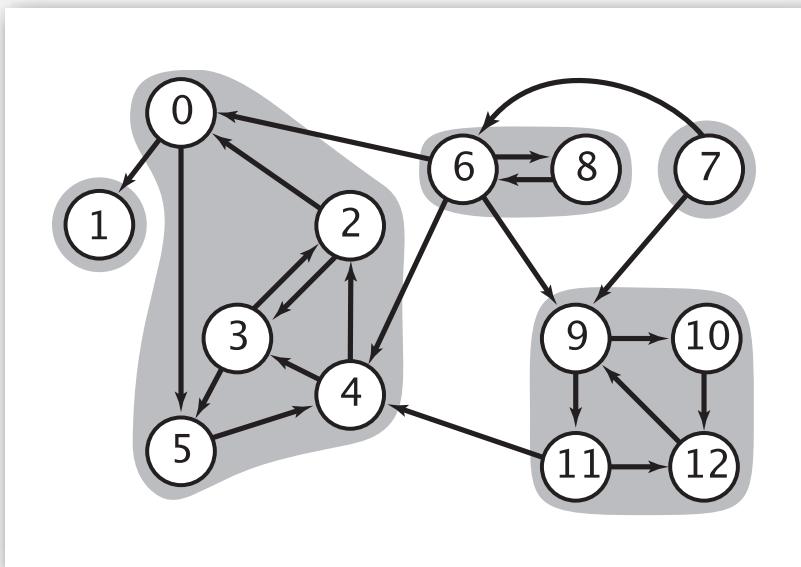
Reverse graph. Strong components in G are same as in G^R .

Kernel DAG. Contract each strong component into a single vertex.

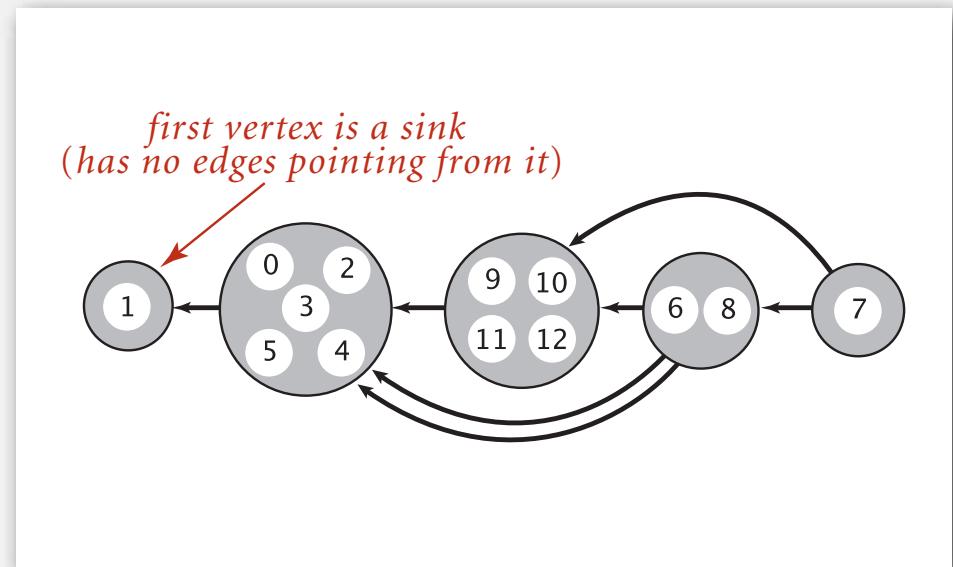
Idea.

- Compute topological order (reverse postorder) in kernel DAG.
- Run DFS, considering vertices in reverse topological order.

how to compute?



digraph G and its strong components

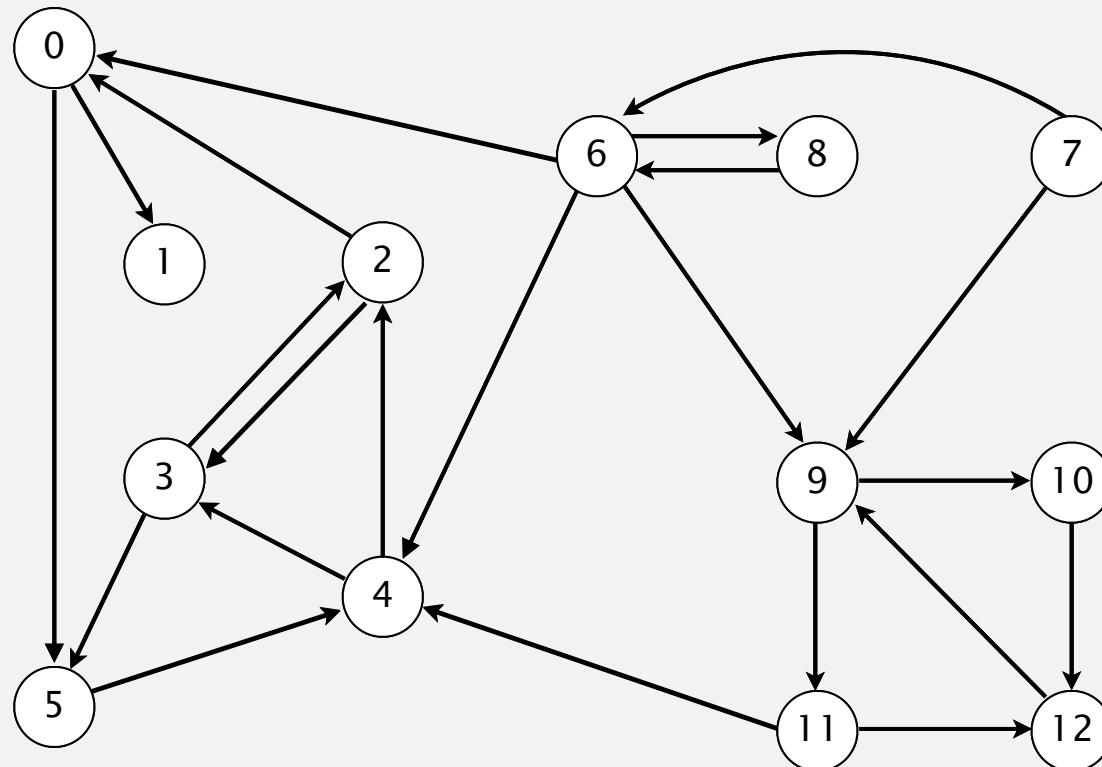


kernel DAG of G (in reverse topological order)

Kosaraju-Sharir algorithm demo

Phase 1. Compute reverse postorder in G^R .

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

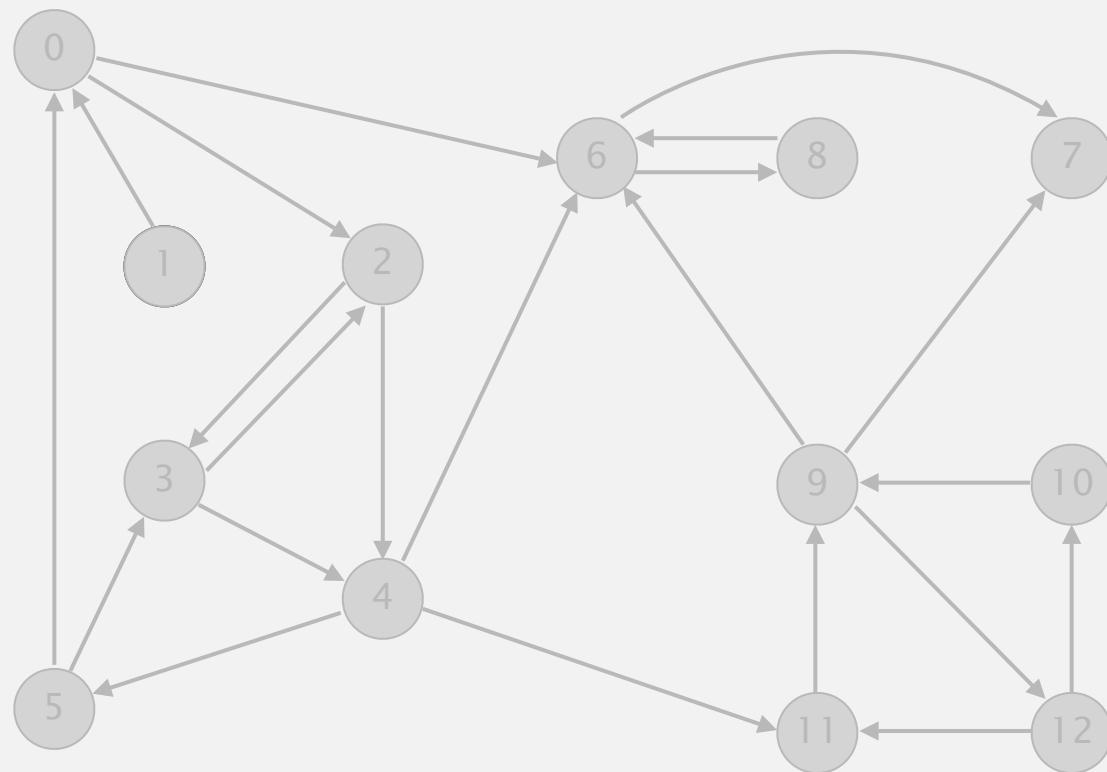


digraph G

Kosaraju-Sharir algorithm demo

Phase 1. Compute reverse postorder in G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8

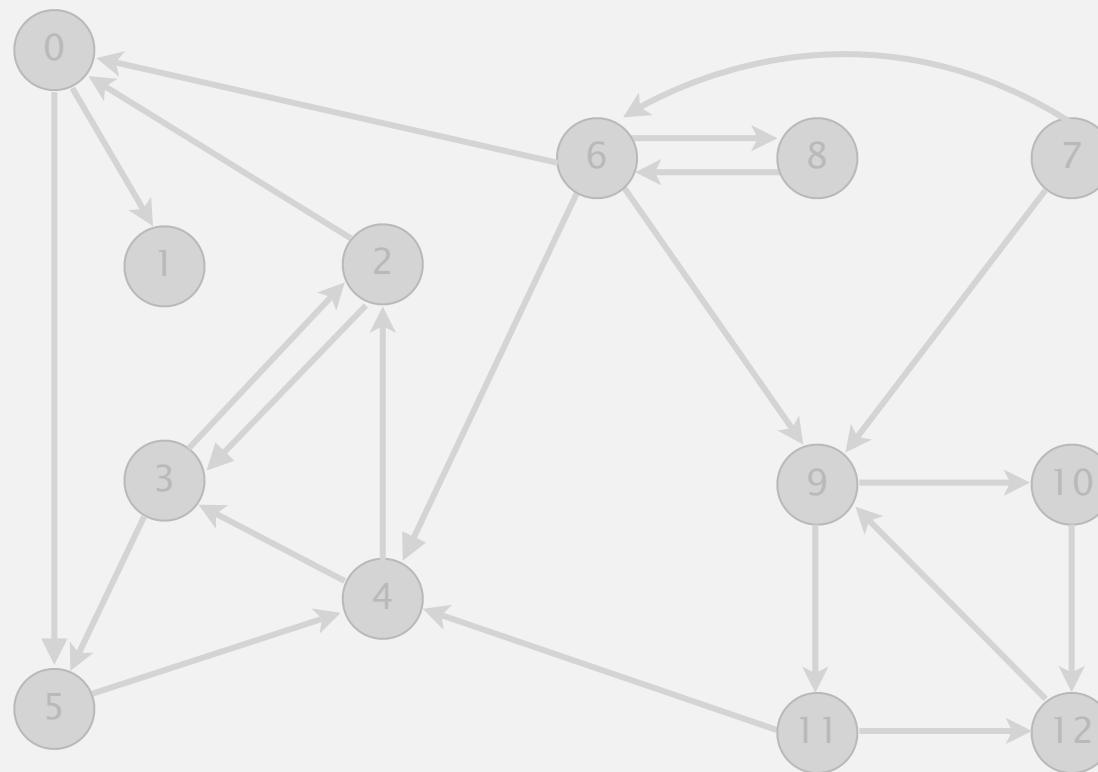


reverse digraph G^R

Kosaraju-Sharir algorithm demo

Phase 2. Run DFS in G , visiting unmarked vertices in reverse postorder of G^R .

1 0 2 4 5 3 11 9 12 10 6 7 8



v	scc[]
0	1
1	0
2	1
3	1
4	1
5	1
6	3
7	4
8	3
9	2
10	2
11	2
12	2

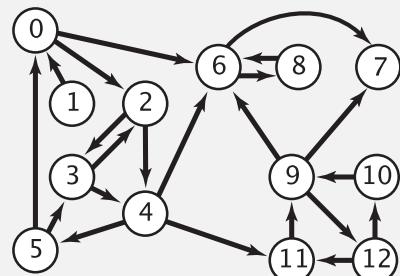
done

Kosaraju-Sharir algorithm

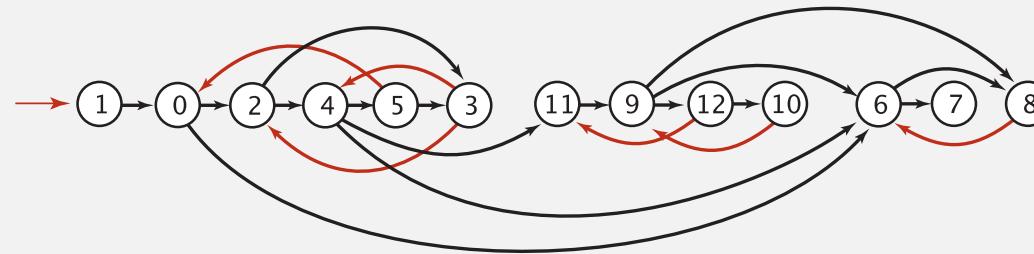
Simple (but mysterious) algorithm for computing strong components.

- Phase 1: run DFS on G^R to compute reverse postorder.
- Phase 2: run DFS on G , considering vertices in order given by first DFS.

DFS in reverse digraph G^R



check unmarked vertices in the order
0 1 2 3 4 5 6 7 8 9 10 11 12



reverse postorder for use in second dfs()
1 0 2 4 5 3 11 9 12 10 6 7 8

```
dfs(0)
  dfs(6)
    dfs(8)
      check 6
      8 done
      dfs(7)
      7 done
    6 done
    dfs(2)
      dfs(4)
        dfs(11)
          dfs(9)
            dfs(12)
              check 11
              dfs(10)
                check 9
              10 done
            12 done
            check 7
          check 6
        ...
      ...
    ...
  ...

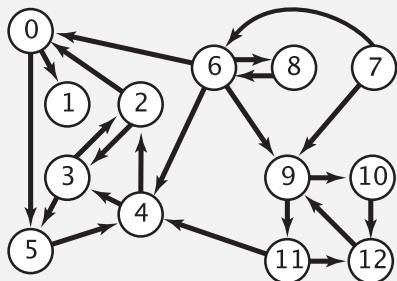
```

Kosaraju-Sharir algorithm

Simple (but mysterious) algorithm for computing strong components.

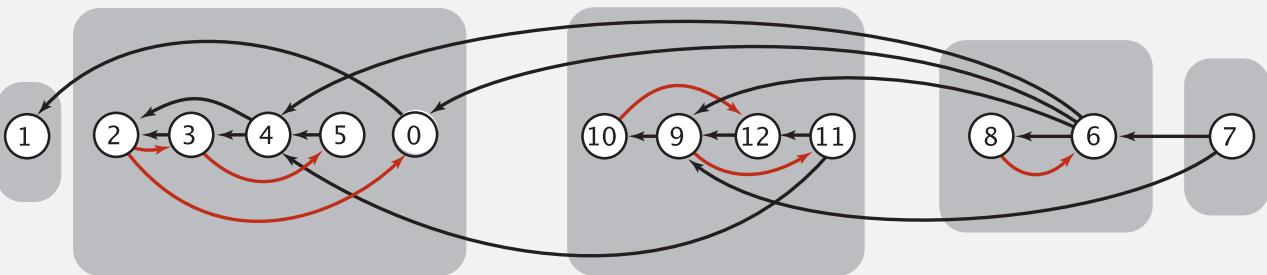
- Phase 1: run DFS on G^R to compute reverse postorder.
- Phase 2: run DFS on G , considering vertices in order given by first DFS.

DFS in original digraph G



check unmarked vertices in the order

1 0 2 4 5 3 11 9 12 10 6 7 8



dfs(1)
1 done

dfs(0)
dfs(5)
dfs(4)
dfs(3)
check 5
dfs(2)
check 0
check 3
2 done
3 done
check 2
4 done
5 done
check 1
0 done
check 2
check 4
check 5
check 3

dfs(11)
check 4
dfs(12)
dfs(9)
check 11
dfs(10)
check 12
10 done
9 done
12 done
11 done
check 9
check 12
check 10

dfs(6)
check 9
check 4
dfs(8)
check 6
8 done
check 0
6 done

dfs(7)
check 6
check 9
7 done
check 8

Kosaraju-Sharir algorithm

Proposition. Kosaraju-Sharir algorithm computes the strong components of a digraph in time proportional to $E + V$.

Pf.

- Running time: bottleneck is running DFS twice (and computing G^R).
- Correctness: tricky, see textbook (2nd printing).
- Implementation: easy!

Connected components in an undirected graph (with DFS)

```
public class CC
{
    private boolean marked[];
    private int[] id;
    private int count;

    public CC(Graph G)
    {
        marked = new boolean[G.V()];
        id = new int[G.V()];

        for (int v = 0; v < G.V(); v++)
        {
            if (!marked[v])
            {
                dfs(G, v);
                count++;
            }
        }
    }

    private void dfs(Graph G, int v)
    {
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }

    public boolean connected(int v, int w)
    {   return id[v] == id[w];   }
}
```

Strong components in a digraph (with two DFSs)

```
public class KosarajuSharirSCC
{
    private boolean marked[];
    private int[] id;
    private int count;

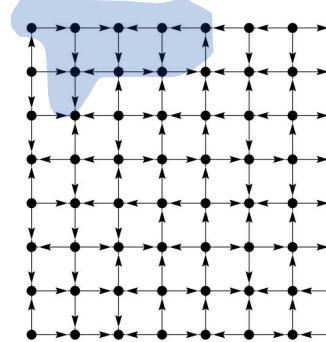
    public KosarajuSharirSCC(Digraph G)
    {
        marked = new boolean[G.V()];
        id = new int[G.V()];
        DepthFirstOrder dfs = new DepthFirstOrder(G.reverse());
        for (int v : dfs.reversePost())
        {
            if (!marked[v])
            {
                dfs(G, v);
                count++;
            }
        }
    }

    private void dfs(Digraph G, int v)
    {
        marked[v] = true;
        id[v] = count;
        for (int w : G.adj(v))
            if (!marked[w])
                dfs(G, w);
    }

    public boolean stronglyConnected(int v, int w)
    {   return id[v] == id[w];  }
}
```

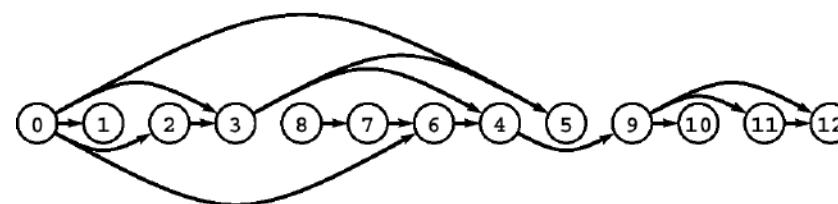
Digraph-processing summary: algorithms of the day

single-source
reachability
in a digraph



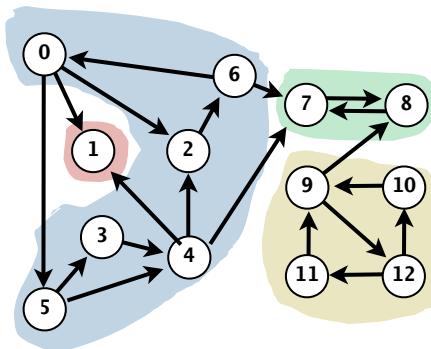
DFS

topological sort
in a DAG



DFS

strong
components
in a digraph



Kosaraju-Sharir
DFS (twice)

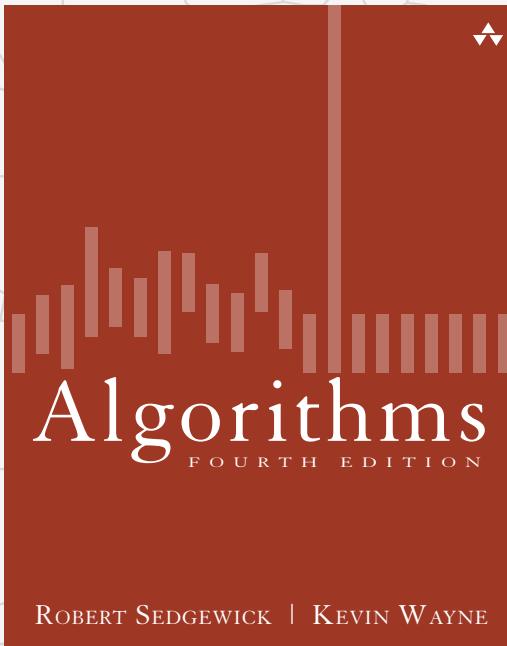
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ ***strong components***



<http://algs4.cs.princeton.edu>

4.2 DIRECTED GRAPHS

- ▶ *introduction*
- ▶ *digraph API*
- ▶ *digraph search*
- ▶ *topological sort*
- ▶ *strong components*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Data compression

Compression reduces the size of a file:

- To save **space** when storing it.
- To save **time** when transmitting it.
- Most files have lots of redundancy.

Who needs compression?

- Moore's law: # transistors on a chip doubles every 18–24 months.
- Parkinson's law: data expands to fill space available.
- Text, images, sound, video, ...

“Everyday, we create 2.5 quintillion bytes of data—so much that 90% of the data in the world today has been created in the last two years alone.” — IBM report on big data (2011)

Basic concepts ancient (1950s), best technology recently developed.

Applications

Generic file compression.

- Files: GZIP, BZIP, 7z.
- Archivers: PKZIP.
- File systems: NTFS, HFS+, ZFS.



Multimedia.

- Images: GIF, JPEG.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.



Communication.

- ITU-T T4 Group 3 Fax.
- V.42bis modem.
- Skype.



Databases. Google, Facebook,



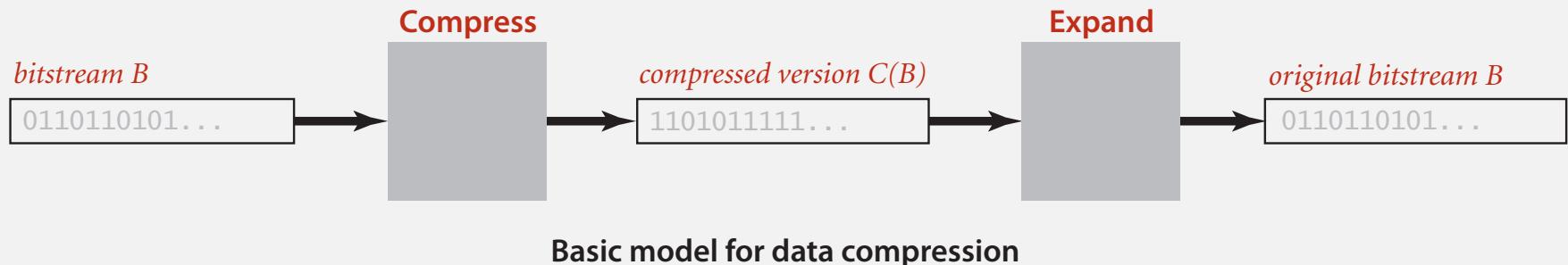
Lossless compression and expansion

Message. Binary data B we want to compress.

Compress. Generates a "compressed" representation $C(B)$.

Expand. Reconstructs original bitstream B .

uses fewer bits (you hope)



Compression ratio. Bits in $C(B)$ / bits in B .

Ex. 50–75% or better compression ratio for natural language.

Food for thought

Data compression has been omnipresent since antiquity:

- Number systems.
- Natural languages.
- Mathematical notation.

$$\cancel{1111} \mid \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$$

has played a central role in communications technology,

- Grade 2 Braille.
- Morse code.
- Telephone system.

b	r	a	i			e
●○ ●○ ○○	●○ ●●○ ○○	●○ ○○ ○○	○● ●○ ○○	●○ ●○ ○○	●○ ●○ ○○	●○ ○● ○○
but	rather	a	I	like	like	every

and is part of modern life.

- MP3.
- MPEG.



Q. What role will it play in the future?

Data representation: genomic code

Genome. String over the alphabet { A, C, T, G }.

Goal. Encode an N -character genome: ATAGATGCCATAG...

Standard ASCII encoding.

- 8 bits per char.
- $8N$ bits.

char	hex	binary
A	41	01000001
C	43	01000011
T	54	01010100
G	47	01000111

Two-bit encoding.

- 2 bits per char.
- $2N$ bits.

char	binary
A	00
C	01
T	10
G	11

Fixed-length code. k -bit code supports alphabet of size 2^k .

Amazing but true. Initial genomic databases in 1990s used ASCII.

Reading and writing binary data

Binary standard input and standard output. Libraries to read and write **bits** from standard input and to standard output.

```
public class BinaryStdIn
    boolean readBoolean()          read 1 bit of data and return as a boolean value
    char readChar()                read 8 bits of data and return as a char value
    char readChar(int r)           read r bits of data and return as a char value
    [similar methods for byte (8 bits); short (16 bits); int (32 bits); long and double (64 bits)]
    boolean isEmpty()              is the bitstream empty?
    void close()                   close the bitstream
```

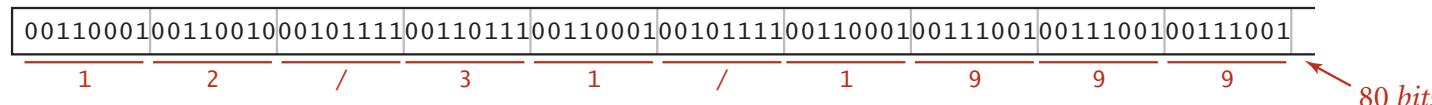
```
public class BinaryStdOut
    void write(boolean b)          write the specified bit
    void write(char c)              write the specified 8-bit char
    void write(char c, int r)        write the r least significant bits of the specified char
    [similar methods for byte (8 bits); short (16 bits); int (32 bits); long and double (64 bits)]
    void close()                   close the bitstream
```

Writing binary data

Date representation. Three different ways to represent 12/31/1999.

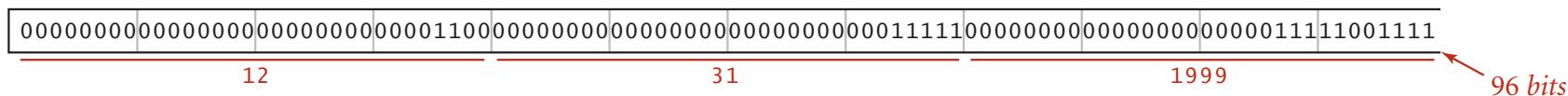
A character stream (StdOut)

```
StdOut.print(month + "/" + day + "/" + year);
```



Three ints (BinaryStdOut)

```
BinaryStdOut.write(month);
BinaryStdOut.write(day);
BinaryStdOut.write(year);
```



A 4-bit field, a 5-bit field, and a 12-bit field (BinaryStdOut)

```
BinaryStdOut.write(month, 4);
BinaryStdOut.write(day, 5);
BinaryStdOut.write(year, 12);
```



Binary dumps

Q. How to examine the contents of a bitstream?

Standard character stream

```
% more abra.txt  
ABRACADABRA!
```

Bitstream represented as 0 and 1 characters

```
% java BinaryDump 16 < abra.txt  
0100000101000010  
0101001001000001  
0100001101000001  
0100010001000001  
0100001001010010  
0100000100100001  
96 bits
```

Bitstream represented with hex digits

```
% java HexDump 4 < abra.txt  
41 42 52 41  
43 41 44 41  
42 52 41 21  
12 bytes
```

Bitstream represented as pixels in a Picture

```
% java PictureDump 16 6 < abra.txt
```



16-by-6 pixel
window, magnified

96 bits

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Hexadecimal to ASCII conversion table

Universal data compression

US Patent 5,533,051 on "Methods for Data Compression", which is capable of compression **all** files.

Slashdot reports of the Zero Space Tuner™ and BinaryAccelerator™.

"ZeoSync has announced a breakthrough in data compression that allows for 100:1 lossless compression of random data. If this is true, our bandwidth problems just got a lot smaller.... "

Universal data compression

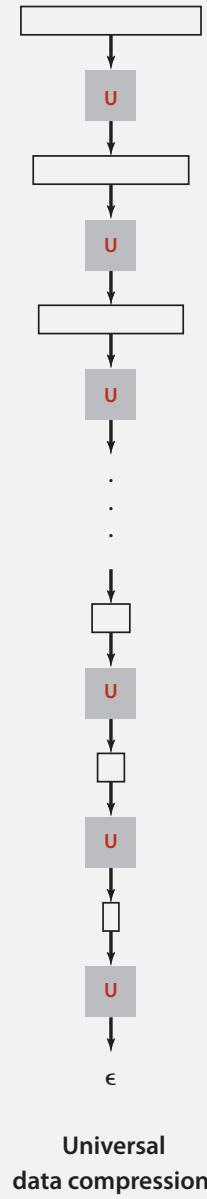
Proposition. No algorithm can compress every bitstring.

Pf 1. [by contradiction]

- Suppose you have a universal data compression algorithm U that can compress every bitstream.
- Given bitstring B_0 , compress it to get smaller bitstring B_1 .
- Compress B_1 to get a smaller bitstring B_2 .
- Continue until reaching bitstring of size 0.
- Implication: all bitstrings can be compressed to 0 bits!

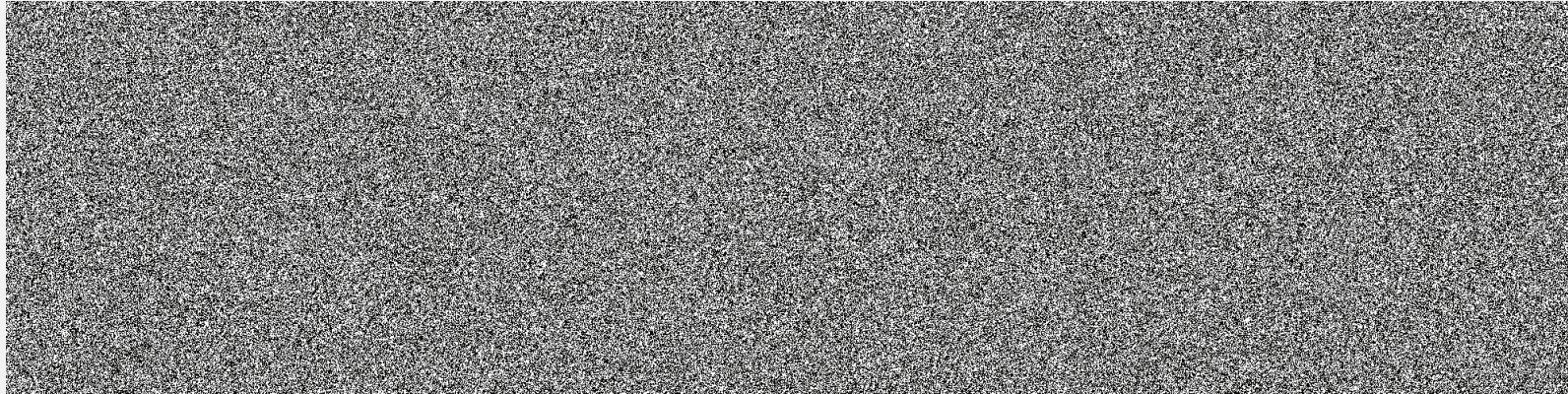
Pf 2. [by counting]

- Suppose your algorithm that can compress all 1,000-bit strings.
- 2^{1000} possible bitstrings with 1,000 bits.
- Only $1 + 2 + 4 + \dots + 2^{998} + 2^{999}$ can be encoded with ≤ 999 bits.
- Similarly, only 1 in 2^{499} bitstrings can be encoded with ≤ 500 bits!



Undecidability

```
% java RandomBits | java PictureDump 2000 500
```



1000000 bits

A difficult file to compress: one million (pseudo-) random bits

```
public class RandomBits
{
    public static void main(String[] args)
    {
        int x = 11111;
        for (int i = 0; i < 1000000; i++)
        {
            x = x * 314159 + 218281;
            BinaryStdOut.write(x > 0);
        }
        BinaryStdOut.close();
    }
}
```

Rdenudcany in Enlgsih Inagugae

Q. How much rdenudcany is in the Enlgsih Inagugae?

“... randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to demonstrate. In a publication of New Scientist you could randomise all the letters, keeping the first two and last two the same, and readability would hardly be affected. My analysis did not come to much because the theory at the time was for shape and sequence recognition. Saberi's work suggests we may have some powerful parallel processors at work. The reason for this is surely that identifying content by parallel processing speeds up recognition. We only need the first and last two letters to spot changes in meaning.” — Graham Rawlinson

A. Quite a bit.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Run-length encoding

Simple type of redundancy in a bitstream. Long runs of repeated bits.

000000000000000011111100000001111111111
↑ 40 bits

Representation. 4-bit counts to represent alternating runs of 0s and 1s:
15 0s, then 7 1s, then 7 0s, then 11 1s.

11110111011110111 ← 16 bits (instead of 40)
— 15 — 7 — 7 — 11 —

- Q. How many bits to store the counts?
 - A. We'll use 8 (but 4 in the example above).

- Q. What to do when run length exceeds max count?
 - A. If longer than 255, intersperse runs of length 0.

Applications. JPEG, ITU-T T4 Group 3 Fax, ...

Run-length encoding: Java implementation

```
public class RunLength
{
    private final static int R    = 256;           ← maximum run-length count
    private final static int lgR = 8;              ← number of bits per count

    public static void compress()
    { /* see textbook */ }

    public static void expand()
    {
        boolean bit = false;
        while (!BinaryStdIn.isEmpty())
        {
            int run = BinaryStdIn.readInt(lgR);   ← read 8-bit count from standard input
            for (int i = 0; i < run; i++)
                BinaryStdOut.write(bit);          ← write 1 bit to standard output
            bit = !bit;
        }
        BinaryStdOut.close();                  ← pad 0s for byte alignment
    }
}
```

An application: compress a bitmap

Typical black-and-white-scanned image.

- 300 pixels/inch.
 - 8.5-by-11 inches.
 - $300 \times 8.5 \times 300 \times 11 = 8.415$ million bits.

Observation. Bits are mostly white.

Typical amount of text on a page.

40 lines × 75 chars per line = 3,000 chars.

A typical bitmap, with run lengths for each row

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ ***Huffman compression***
- ▶ *LZW compression*



David Huffman

Variable-length codes

Use different number of bits to encode different chars.

Ex. Morse code: • • • - - - • • •

Issue. Ambiguity.

SOS ?

V7 ?

IAMIE ?

EEWNI ?

In practice. Use a medium gap to separate codewords.

codeword for S is a prefix of codeword for V

Letters	Numbers
A	•—
B	—•••
C	—•—•
D	—••
E	•
F	••—•
G	——•
H	••••
I	••
J	•— — —
K	—•—
L	•— • •
M	— —
N	— •
O	— — —
P	• — — •
Q	— — • —
R	• — •
S	• • •
T	—
U	• • —
V	• • • —
W	• — —
X	— • • —
Y	— • — —
Z	— — • •

Variable-length codes

Q. How do we avoid ambiguity?

A. Ensure that no codeword is a **prefix** of another.

Ex 1. Fixed-length code.

Ex 2. Append special stop char to each codeword.

Ex 3. General prefix-free code.

Codeword table

<i>key</i>	<i>value</i>
!	101
A	0
B	1111
C	110
D	100
R	1110

Compressed bitstring

0111111001100100011111100101 ← 30 bits
A B RA CA DA B RA !

Codeword table

<i>key</i>	<i>value</i>
!	101
A	11
B	00
C	010
D	100
R	011

Compressed bitstring

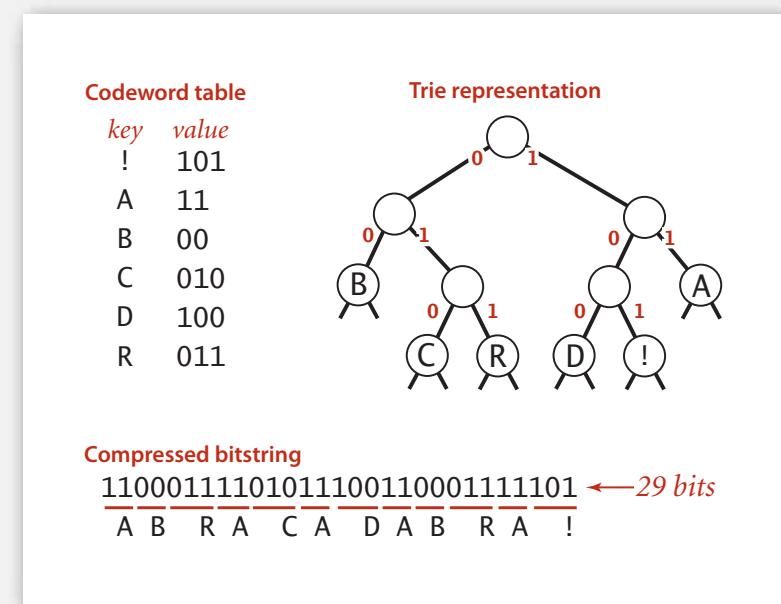
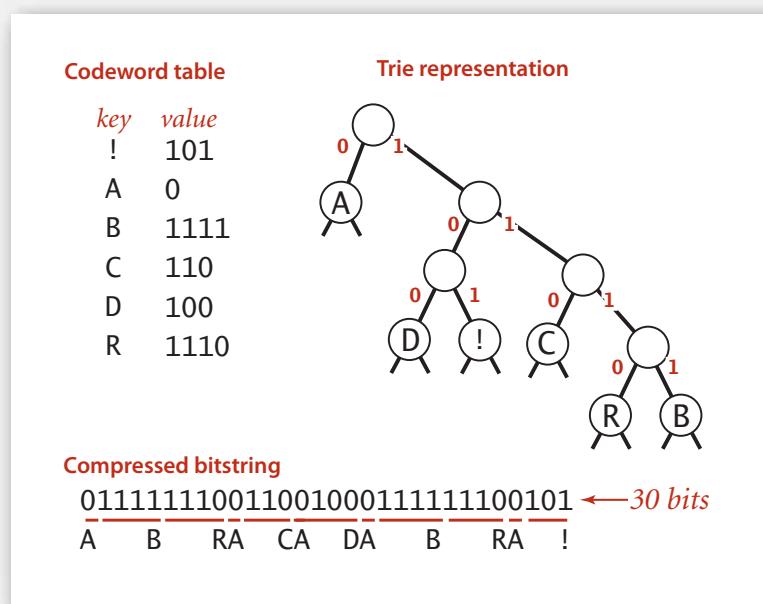
11000111101011100110001111101 ← 29 bits
A B R A C A D A B R A !

Prefix-free codes: trie representation

Q. How to represent the prefix-free code?

A. A binary trie!

- Chars in leaves.
- Codeword is path from root to leaf.



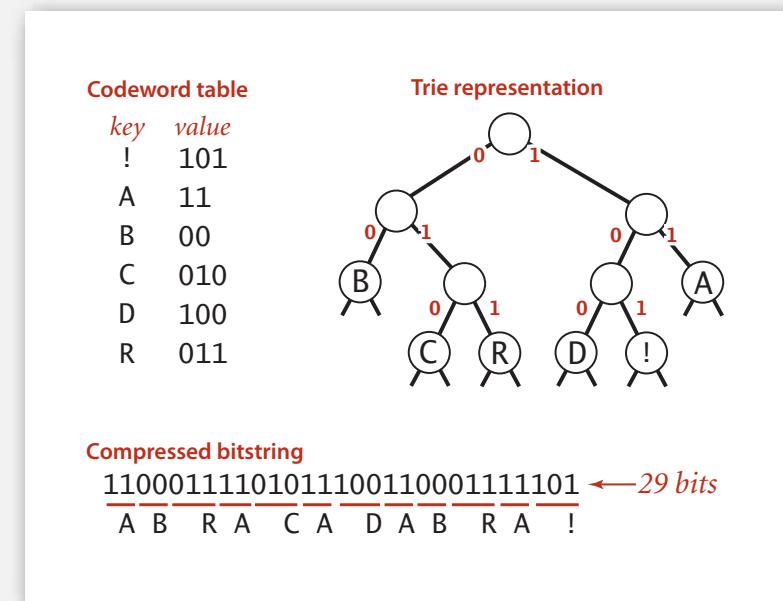
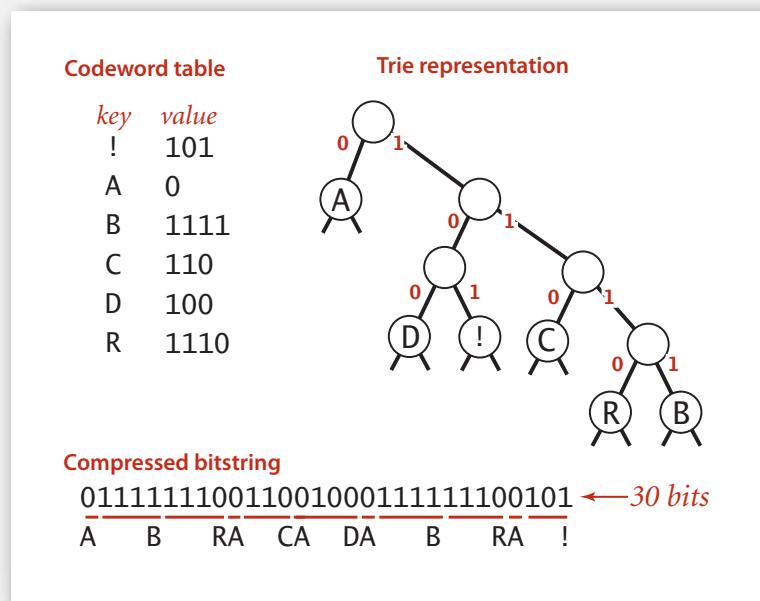
Prefix-free codes: compression and expansion

Compression.

- Method 1: start at leaf; follow path up to the root; print bits in reverse.
- Method 2: create ST of key-value pairs.

Expansion.

- Start at root.
- Go left if bit is 0; go right if 1.
- If leaf node, print char and return to root.



Huffman trie node data type

```
private static class Node implements Comparable<Node>
{
    private final char ch;    // used only for leaf nodes
    private final int freq;   // used only for compress
    private final Node left, right;

    public Node(char ch, int freq, Node left, Node right)
    {
        this.ch      = ch;
        this.freq    = freq;
        this.left    = left;
        this.right   = right;
    }

    public boolean isLeaf()
    { return left == null && right == null; }

    public int compareTo(Node that)
    { return this.freq - that.freq; }
}
```

← initializing constructor

← is Node a leaf?

← compare Nodes by frequency
(stay tuned)

Prefix-free codes: expansion

```
public void expand()
{
    Node root = readTrie();
    int N = BinaryStdIn.readInt();           ← read in encoding trie
                                              ← read in number of chars

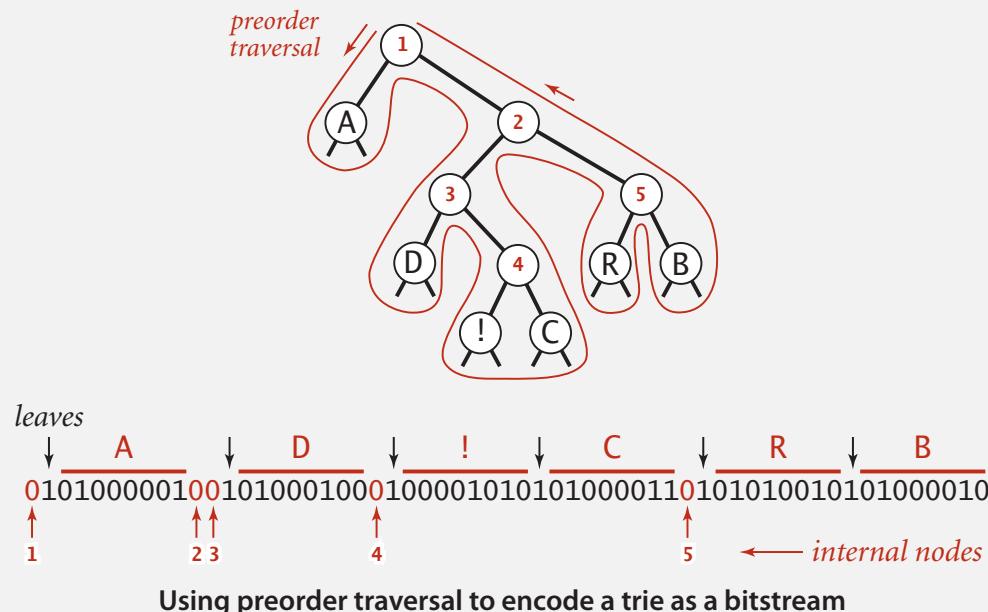
    for (int i = 0; i < N; i++)
    {
        Node x = root;
        while (!x.isLeaf())                  ← expand codeword for ith char
        {
            if (!BinaryStdIn.readBoolean())
                x = x.left;
            else
                x = x.right;
        }
        BinaryStdOut.write(x.ch, 8);
    }
    BinaryStdOut.close();
}
```

Running time. Linear in input size N .

Prefix-free codes: how to transmit

Q. How to write the trie?

A. Write preorder traversal of trie; mark leaf and internal nodes with a bit.



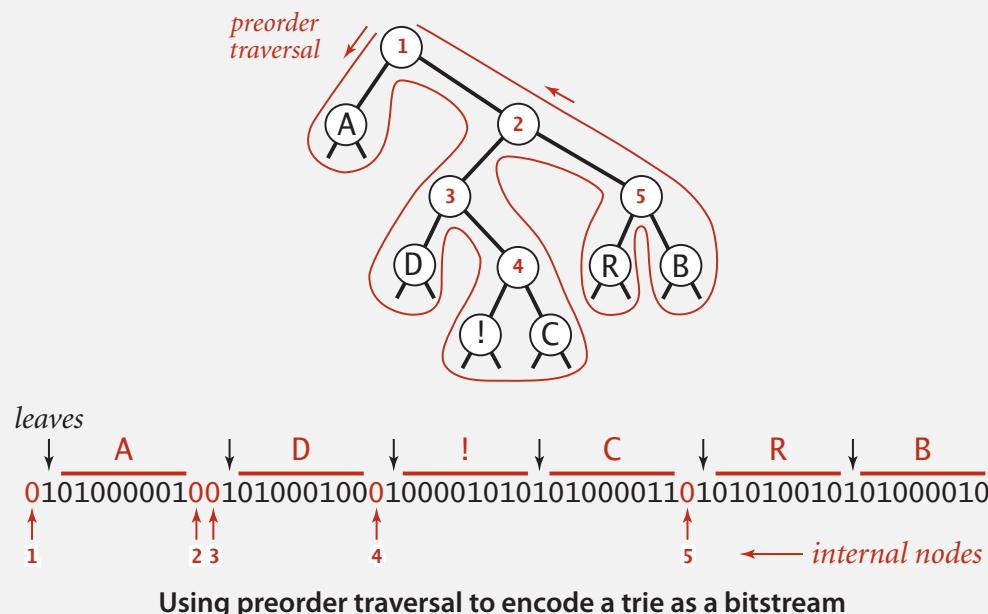
```
private static void writeTrie(Node x)
{
    if (x.isLeaf())
    {
        BinaryStdOut.write(true);
        BinaryStdOut.write(x.ch, 8);
        return;
    }
    BinaryStdOut.write(false);
    writeTrie(x.left);
    writeTrie(x.right);
}
```

Note. If message is long, overhead of transmitting trie is small.

Prefix-free codes: how to transmit

Q. How to read in the trie?

A. Reconstruct from preorder traversal of trie.



```
private static Node readTrie()
{
    if (BinaryStdIn.readBoolean())
    {
        char c = BinaryStdIn.readChar(8);
        return new Node(c, 0, null, null);
    }
    Node x = readTrie();
    Node y = readTrie();
    return new Node('\0', 0, x, y);
}
```

used only for
leaf nodes

Shannon-Fano codes

Q. How to find best prefix-free code?

Shannon-Fano algorithm:

- Partition symbols S into two subsets S_0 and S_1 of (roughly) equal freq.
- Codewords for symbols in S_0 start with 0; for symbols in S_1 start with 1.
- Recur in S_0 and S_1 .

char	freq	encoding
A	5	0...
C	1	0...

$S_0 = \text{codewords starting with 0}$

char	freq	encoding
B	2	1...
D	1	1...
R	2	1...
!	1	1...

$S_1 = \text{codewords starting with 1}$

Problem 1. How to divide up symbols?

Problem 2. Not optimal!

Huffman algorithm demo

- Count frequency for each character in input.

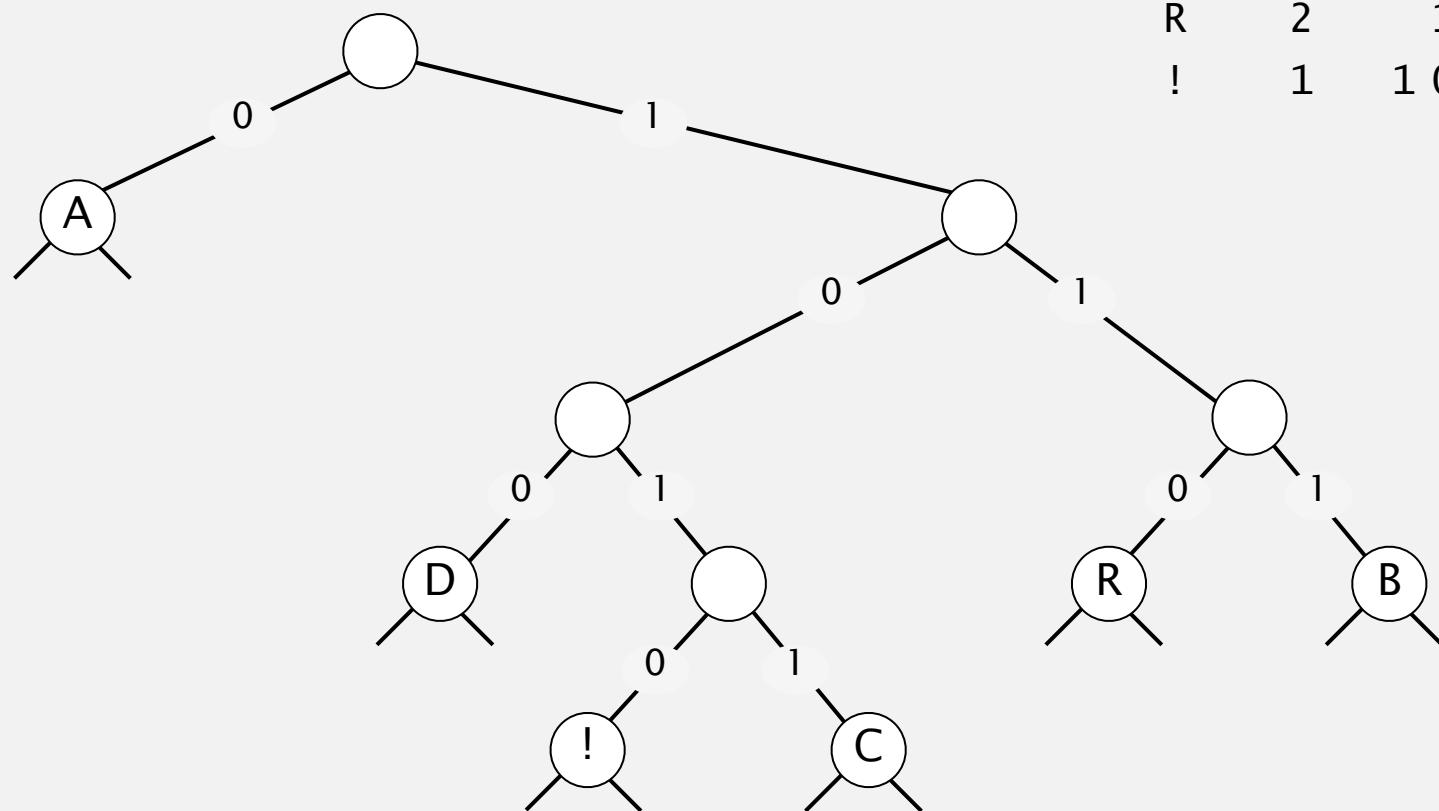


char	freq	encoding
A	5	
B	2	
C	1	
D	1	
R	2	
!	1	

input

A B R A C A D A B R A !

Huffman algorithm demo



char	freq	encoding
A	5	0
B	2	1 1 1
C	1	1 0 1 1
D	1	1 0 0
R	2	1 1 0
!	1	1 0 1 0

Huffman codes

Q. How to find best prefix-free code?

Huffman algorithm:

- Count frequency $\text{freq}[i]$ for each char i in input.
- Start with one node corresponding to each char i (with weight $\text{freq}[i]$).
- Repeat until single trie formed:
 - select two tries with min weight $\text{freq}[i]$ and $\text{freq}[j]$
 - merge into single trie with weight $\text{freq}[i] + \text{freq}[j]$

Applications:



Constructing a Huffman encoding trie: Java implementation

```
private static Node buildTrie(int[] freq)
{
    MinPQ<Node> pq = new MinPQ<Node>();
    for (char i = 0; i < R; i++)
        if (freq[i] > 0)
            pq.insert(new Node(i, freq[i], null, null));

    while (pq.size() > 1)
    {
        Node x = pq.delMin();
        Node y = pq.delMin();
        Node parent = new Node('\0', x.freq + y.freq, x, y);
        pq.insert(parent);
    }

    return pq.delMin();
}
```

initialize PQ with singleton tries

merge two smallest tries

not used for internal nodes

total frequency

two subtrees

Huffman encoding summary

Proposition. [Huffman 1950s] Huffman algorithm produces an optimal prefix-free code.

Pf. See textbook.

↑
no prefix-free code uses fewer bits

Implementation.

- Pass 1: tabulate char frequencies and build trie.
- Pass 2: encode file by traversing trie or lookup table.

Running time. Using a binary heap $\Rightarrow N + R \log R$.

↑ ↑
input size alphabet size

Q. Can we do better? [stay tuned]

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

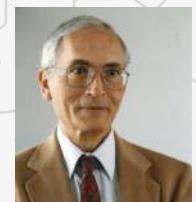
<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*



Abraham Lempel



Jacob Ziv

Statistical methods

Static model. Same model for all texts.

- Fast.
- Not optimal: different texts have different statistical properties.
- Ex: ASCII, Morse code.

Dynamic model. Generate model based on text.

- Preliminary pass needed to generate model.
- Must transmit the model.
- Ex: Huffman code.

Adaptive model. Progressively learn and update model as you read text.

- More accurate modeling produces better compression.
- Decoding must start from beginning.
- Ex: LZW.

LZW compression example

<i>input</i>	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
<i>matches</i>	A	B	R	A	C	A	D	A	B	R	A	B	R	A	B	R	A
<i>value</i>	41	42	52	41	43	41	44	81		83		82		88		41	80

LZW compression for A B R A C A D A B R A B R A B R A

key	value
:	:
A	41
B	42
C	43
D	44
:	:

key	value
AB	81
BR	82
RA	83
AC	84
CA	85
AD	86

key	value
DA	87
ABR	88
RAB	89
BRA	8A
ABRA	8B

codeword table

Lempel-Ziv-Welch compression

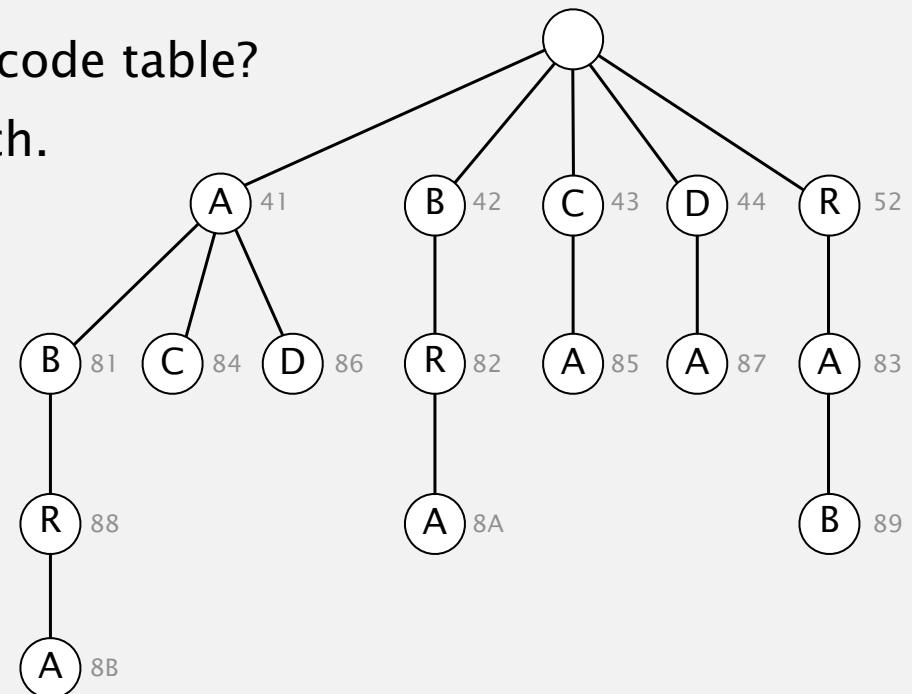
LZW compression.

- Create ST associating W -bit codewords with string keys.
- Initialize ST with codewords for single-char keys.
- Find longest string s in ST that is a prefix of unscanned part of input.
- Write the W -bit codeword associated with s .
- Add $s + c$ to ST, where c is next char in the input.

longest prefix match

Q. How to represent LZW compression code table?

A. A trie to support longest prefix match.



LZW compression: Java implementation

```
public static void compress()
{
    String input = BinaryStdIn.readString();           ← read in input as a string

    TST<Integer> st = new TST<Integer>();
    for (int i = 0; i < R; i++)
        st.put("" + (char) i, i);
    int code = R+1;

    while (input.length() > 0)
    {
        String s = st.longestPrefixOf(input);           ← find longest prefix match s
        BinaryStdOut.write(st.get(s), W);                ← write W-bit codeword for s

        int t = s.length();
        if (t < input.length() && code < L)
            st.put(input.substring(0, t+1), code++);    ← add new codeword
        input = input.substring(t);                      ← scan past s in input
    }

    BinaryStdOut.write(R, W);                          ← write "stop" codeword
    BinaryStdOut.close();                           and close input stream
}
```

LZW expansion example

<i>value</i>	41	42	52	41	43	41	44	81	83	82	88	41	80
<i>output</i>	A	B	R	A	C	A	D	A B	R A	B R	A B R	A	

LZW expansion for 41 42 52 41 43 41 44 81 83 82 88 41 80

key	value
:	:
41	A
42	B
43	C
44	D
:	:

key	value
81	AB
82	BR
83	RA
84	AC
85	CA
86	AD

key	value
87	DA
88	ABR
89	RAB
8A	BRA
8B	ABRA

codeword table

LZW expansion

LZW expansion.

- Create ST associating string values with W -bit keys.
- Initialize ST to contain single-char values.
- Read a W -bit key.
- Find associated string value in ST and write it out.
- Update ST.

key	value
:	:
65	A
66	B
67	C
68	D
:	:
129	AB
130	BR
131	RA
132	AC
133	CA
134	AD
135	DA
136	ABR
137	RAB
138	BRA
139	ABRA
:	:

Q. How to represent LZW expansion code table?

A. An array of size 2^W .

LZW example: tricky case

<i>input</i>	A	B	A	B	A	B	A
<i>matches</i>	A	B	A	B	A	B	A
<i>value</i>	41	42	81		83		80

LZW compression for ABABABA

key	value
:	:
A	41
B	42
C	43
D	44
:	:

key	value
AB	81
BA	82
ABA	83

codeword table

LZW example: tricky case

<i>value</i>	41	42	81	83	80
<i>output</i>	A	B	A B	A B A	←

need to know which
key has value 83
before it is in ST!

LZW expansion for 41 42 81 83 80

key	value
⋮	⋮
41	A
42	B
43	C
44	D
⋮	⋮

key	value
81	AB
82	BA
83	ABA

codeword table

LZW implementation details

How big to make ST?

- How long is message?
- Whole message similar model?
- [many other variations]

What to do when ST fills up?

- Throw away and start over. [GIF]
- Throw away when not effective. [Unix compress]
- [many other variations]

Why not put longer substrings in ST?

- [many variations have been developed]

LZW in the real world

Lempel-Ziv and friends.

- LZ77. LZ77 not patented ⇒ widely used in open source
 - LZ78. LZW patent #4,558,302 expired in U.S. on June 20, 2003
 - LZW.
 - Deflate / zlib = LZ77 variant + Huffman.

United States Patent		[19]	Patent Number:	4,558,302
Welch			[45] Date of Patent:	Dec. 10, 1985
[54] HIGH SPEED DATA COMPRESSION AND DECOMPRESSION APPARATUS AND METHOD				
[75] Inventor:	Terry A. Welch, Concord, Mass.			
[73] Assignee:	Sperry Corporation, New York, N.Y.			
[21] Appl. No.:	505,638			
[22] Filed:	Jun. 20, 1983			
[51] Int. Cl.⁴ G86F 5/00			
[52] U.S. Cl. 340/347 DD; 235/310			
[58] Field of Search 340/347 DD; 235/310, 235/311; 364/200, 900			
[56] References Cited				
U.S. PATENT DOCUMENTS				
4,464,650	8/1984 Eastman	340/347 DD		
OTHER PUBLICATIONS				
Ziv, "IEEE Transactions on Information Theory", IT-24-5, Sep. 1977, pp. 530-537.				
Ziv, "IEEE Transactions on Information Theory", IT-23-3, May 1977, pp. 337-343.				
Primary Examiner—Charles D. Miller				
Attorney, Agent, or Firm—Howard P. Terry; Albert B. Cooper				
[57] ABSTRACT				
A data compressor compresses an input stream of data character signals by storing in a string table strings of data character signals encountered in the input stream. The compressor searches the input stream to determine				
the longest match to a stored string. Each stored string comprises a prefix string and an extension character where the extension character is the last character in the string and the prefix string comprises all but the extension character. Each string has a code signal associated therewith and a string is stored in the string table by, at least implicitly, storing the code signal for the string, the code signal for the string prefix and the extension character. When the longest match between the input data character stream and the stored strings is determined, the code signal for the longest match is transmitted as the compressed code signal for the encountered string of characters and an extension string is stored in the string table. The prefix of the extended string is the longest match and the extension character of the extended string is the next input data character signal following the longest match. Searching through the string table and entering extended strings therein is effected by a limited search hashing procedure. Decompression is effected by a decompressor that receives the compressed code signals and generates a string table similar to that constructed by the compressor to effect lookup of received code signals so as to recover the data character signals comprising a stored string. The decompressor string table is updated by storing a string having a prefix in accordance with a prior received code signal and an extension character in accordance with the first character of the currently recovered string.				
181 Claims, 9 Drawing Figures				



LZW in the real world

Lempel-Ziv and friends.

- LZ77.
- LZ78.
- LZW.
- Deflate / zlib = LZ77 variant + Huffman.



Unix compress, GIF, TIFF, V.42bis modem: LZW.

zip, 7zip, gzip, jar, png, pdf: deflate / zlib.

iPhone, Sony Playstation 3, Apache HTTP server: deflate / zlib.



Lossless data compression benchmarks

year	scheme	bits / char
1967	ASCII	7.00
1950	Huffman	4.70
1977	LZ77	3.94
1984	LZMW	3.32
1987	LZH	3.30
1987	move-to-front	3.24
1987	LZB	3.18
1987	gzip	2.71
1988	PPMC	2.48
1994	SAKDC	2.47
1994	PPM	2.34
1995	Burrows-Wheeler	2.29
1997	BOA	1.99
1999	RK	1.89

← next programming assignment

Data compression summary

Lossless compression.

- Represent fixed-length symbols with variable-length codes. [Huffman]
- Represent variable-length symbols with fixed-length codes. [LZW]

Lossy compression. [not covered in this course]

- JPEG, MPEG, MP3, ...
- FFT, wavelets, fractals, ...

Theoretical limits on compression. Shannon entropy: $H(X) = - \sum_i^n p(x_i) \lg p(x_i)$

Practical compression. Use extra knowledge whenever possible.

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



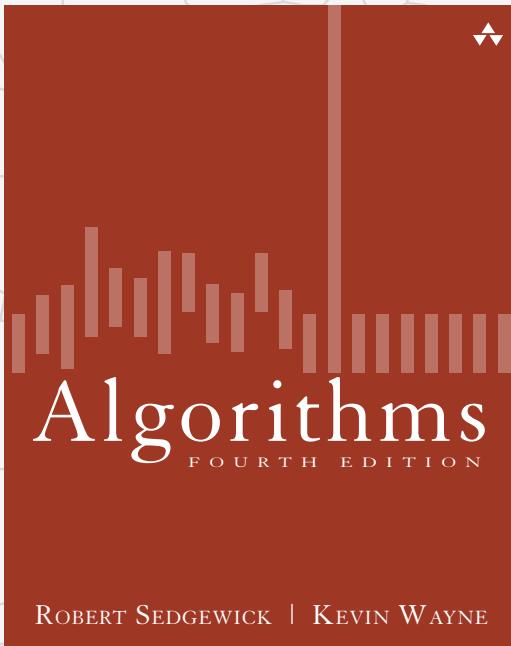
<http://algs4.cs.princeton.edu>

5.5 DATA COMPRESSION

- ▶ *introduction*
- ▶ *run-length coding*
- ▶ *Huffman compression*
- ▶ *LZW compression*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

Linear programming

What is it? Problem-solving model for optimal allocation of scarce resources, among a number of competing activities that encompasses:

- Shortest paths, maxflow, MST, matching, assignment, ...
- $Ax = b$, 2-person zero-sum games, ...

can take an entire course on LP

$$\begin{array}{llllll} \text{maximize} & 13A & + & 23B \\ \text{subject} & 5A & + & 15B & \leq & 480 \\ \text{to the} & 4A & + & 4B & \leq & 160 \\ \text{constraints} & 35A & + & 20B & \leq & 1190 \\ & A, & B & \geq & 0 \end{array}$$

Why significant?

- Fast commercial solvers available.
- Widely applicable problem-solving model.
- Key subroutine for integer programming solvers.

Ex: Delta claims that LP saves \$100 million per year.

Applications

Agriculture. Diet problem.

Computer science. Compiler register allocation, data mining.

Electrical engineering. VLSI design, optimal clocking.

Energy. Blending petroleum products.

Economics. Equilibrium theory, two-person zero-sum games.

Environment. Water quality management.

Finance. Portfolio optimization.

Logistics. Supply-chain management.

Management. Hotel yield management.

Marketing. Direct mail advertising.

Manufacturing. Production line balancing, cutting stock.

Medicine. Radioactive seed placement in cancer treatment.

Operations research. Airline crew assignment, vehicle routing.

Physics. Ground states of 3-D Ising spin glasses.

Telecommunication. Network design, Internet routing.

Sports. Scheduling ACC basketball, handicapping horse races.

Algorithms

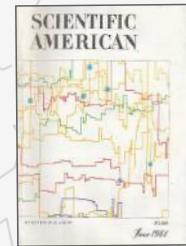
ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

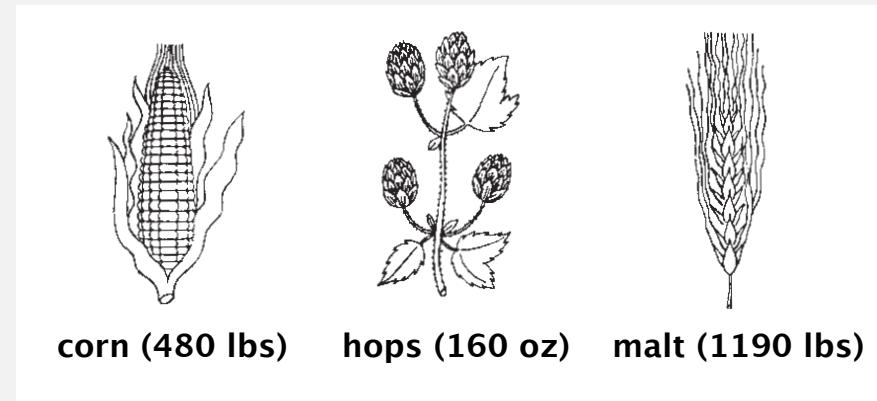
Allocation of Resources by Linear Programming
by Robert Bland
Scientific American, Vol. 244, No. 6, June 1981



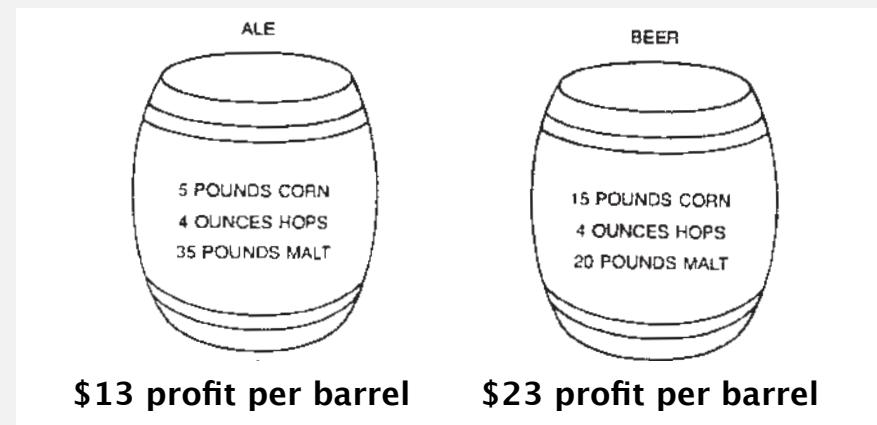
Toy LP example: brewer's problem

Small brewery produces ale and beer.

- Production limited by scarce resources: corn, hops, barley malt.



- Recipes for ale and beer require different proportions of resources.



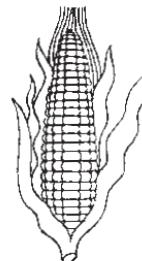
Toy LP example: brewer's problem

Brewer's problem: choose product mix to maximize profits.

34 barrels \times 35 lbs malt = 1190 lbs
[amount of available malt]

ale	beer	corn	hops	malt	profit
34	0	179	136	1190	\$442
0	32	480	128	640	\$736
19.5	20.5	405	160	1092.5	\$725
12	28	480	160	980	\$800
?	?				> \$800 ?

goods are
divisible



corn (480 lbs)



hops (160 oz)



malt (1190 lbs)



\$13 profit per barrel



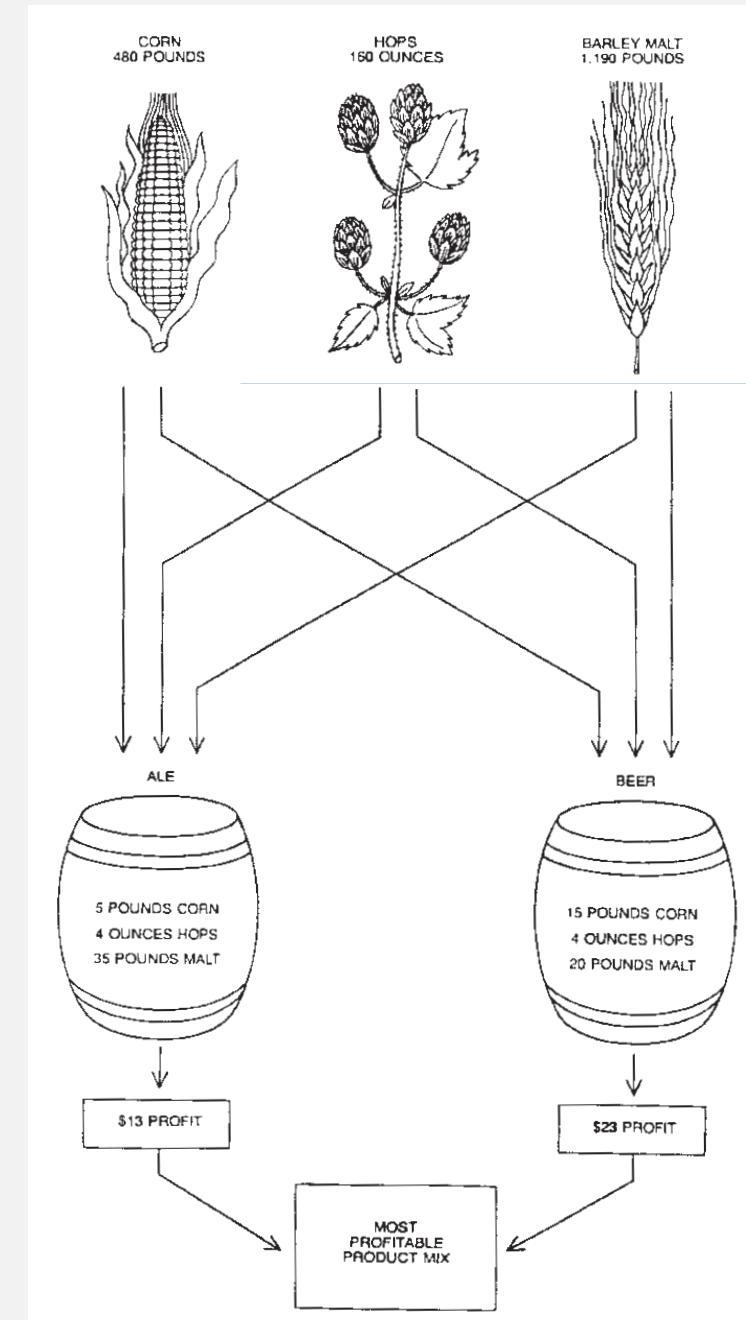
\$23 profit per barrel

Brewer's problem: linear programming formulation

Linear programming formulation.

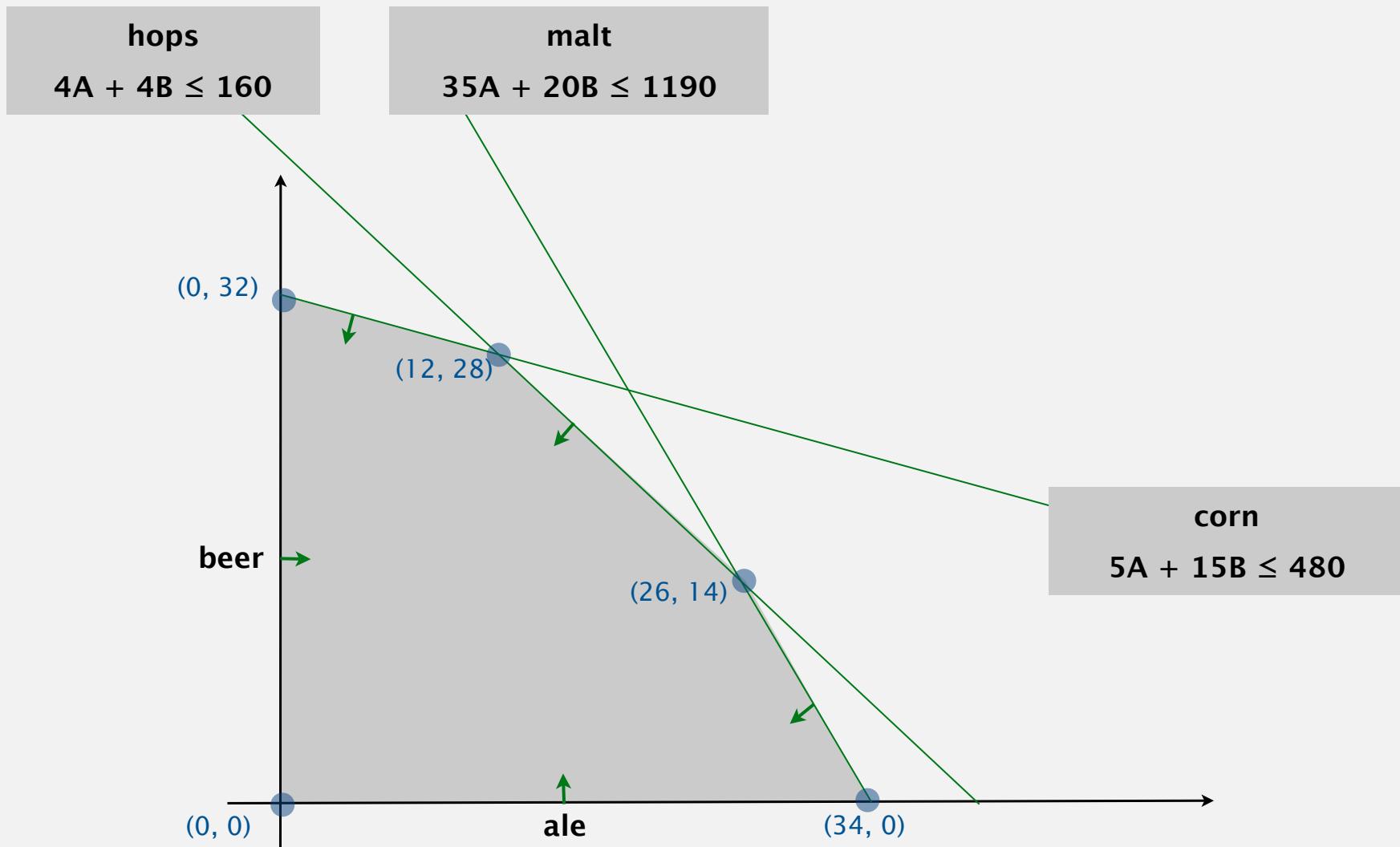
- Let A be the number of barrels of ale.
- Let B be the number of barrels of beer.

	ale	beer		
maximize	$13A$	$+ 23B$		profits
subject to the constraints	$5A$	$+ 15B \leq 480$		corn
	$4A$	$+ 4B \leq 160$		hops
	$35A$	$+ 20B \leq 1190$		malt
	$A, B \geq 0$			

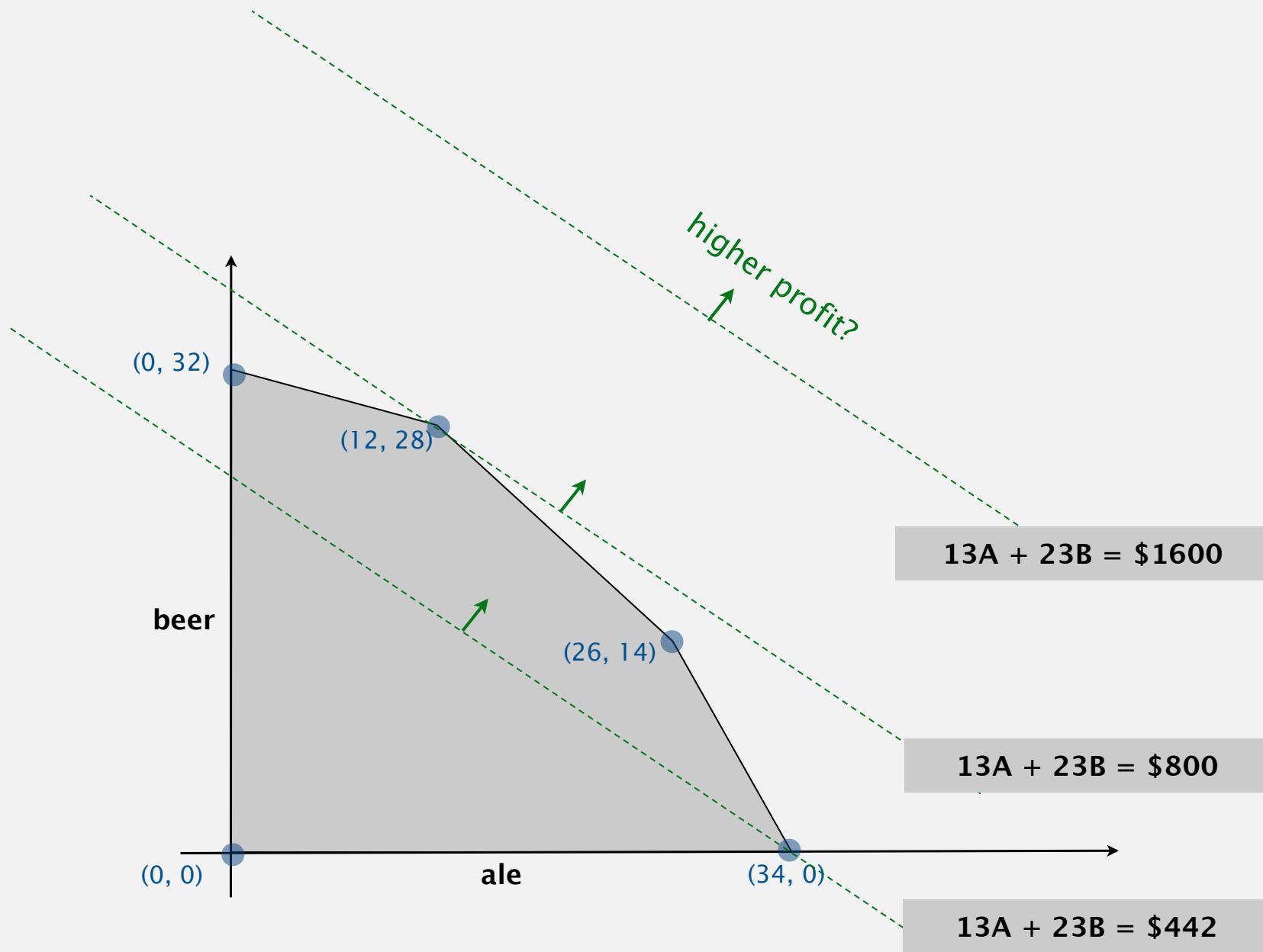


Brewer's problem: feasible region

Inequalities define **halfplanes**; feasible region is a **convex polygon**.



Brewer's problem: objective function

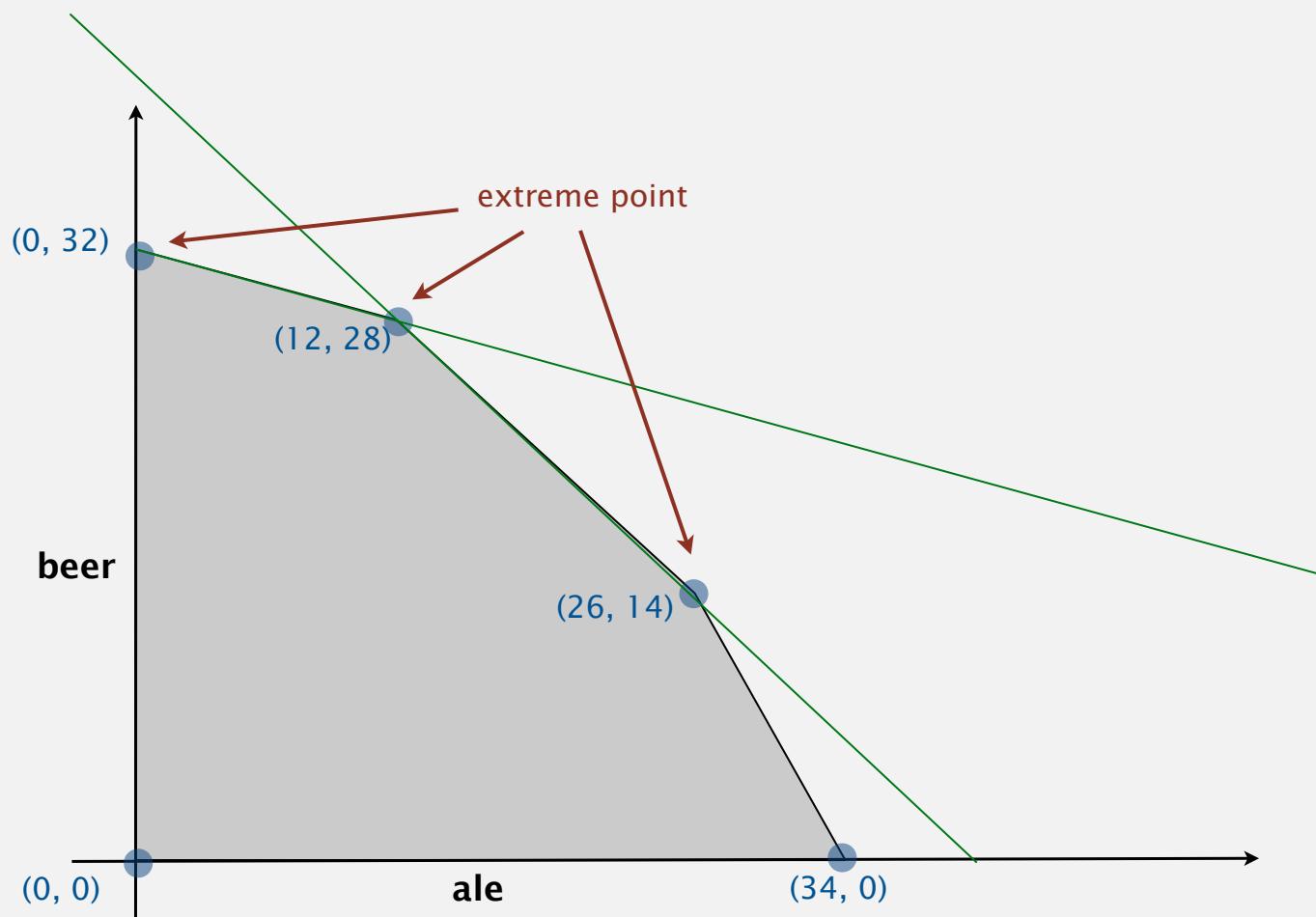


Brewer's problem: geometry

Optimal solution occurs at an **extreme point**.



intersection of 2 constraints in 2d



Standard form linear program

Goal. Maximize linear objective function of n nonnegative variables, subject to m linear equations.

- Input: real numbers a_{ij}, c_j, b_i .
- Output: real numbers x_j .



linear means no x^2 , xy , $\arccos(x)$, etc.

primal problem (P)					matrix version	
maximize	$c_1 x_1 + c_2 x_2 + \dots + c_n x_n$				maximize	$c^T x$
subject to the constraints	$a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n = b_1$				subject to the constraints	$A x = b$
	$a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n = b_2$					
	\vdots	\vdots	\vdots	\vdots		\vdots
	$a_{m1} x_1 + a_{m2} x_2 + \dots + a_{mn} x_n = b_m$					
	$x_1, x_2, \dots, x_n \geq 0$					

Caveat. No widely agreed notion of "standard form."

Converting the brewer's problem to the standard form

Original formulation.

$$\begin{array}{lll} \text{maximize} & 13A + 23B \\ \text{subject} & 5A + 15B \leq 480 \\ \text{to the} & 4A + 4B \leq 160 \\ \text{constraints} & 35A + 20B \leq 1190 \\ & A, B \geq 0 \end{array}$$

Standard form.

- Add variable Z and equation corresponding to objective function.
- Add **slack** variable to convert each inequality to an equality.
- Now a 6-dimensional problem.

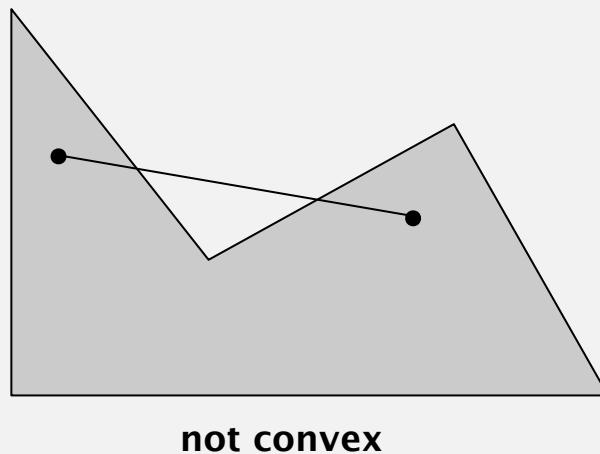
$$\begin{array}{llllll} \text{maximize} & Z & & & & \\ & 13A + 23B & & & -Z = 0 & \\ \hline \text{subject} & 5A + 15B + S_C & & & = 480 & \\ \text{to the} & 4A + 4B + S_H & & & = 160 & \\ \text{constraints} & 35A + 20B + S_M & & & = 1190 & \\ & A, B, S_C, S_H, S_M \geq 0 & & & & \end{array}$$

Geometry

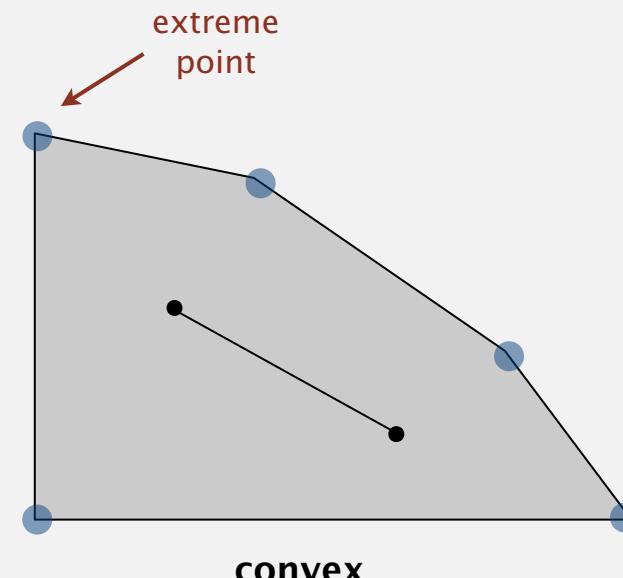
Inequalities define **halfspaces**; feasible region is a **convex polyhedron**.

A set is **convex** if for any two points a and b in the set, so is $\frac{1}{2}(a + b)$.

An **extreme point** of a set is a point in the set that can't be written as $\frac{1}{2}(a + b)$, where a and b are two distinct points in the set.



not convex



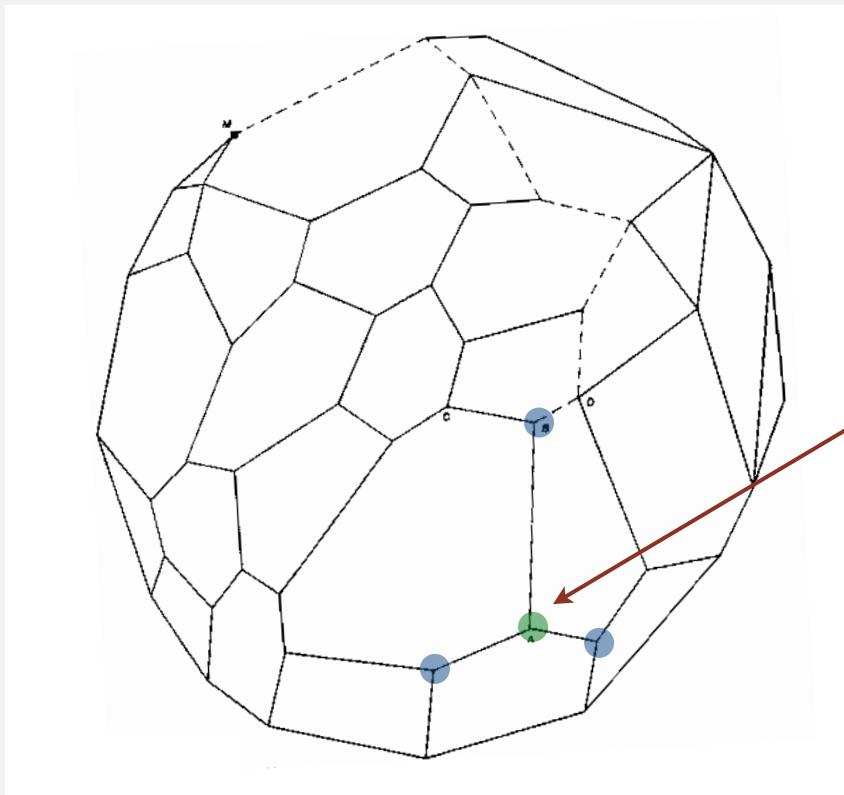
convex

Warning. Don't always trust intuition in higher dimensions.

Geometry (continued)

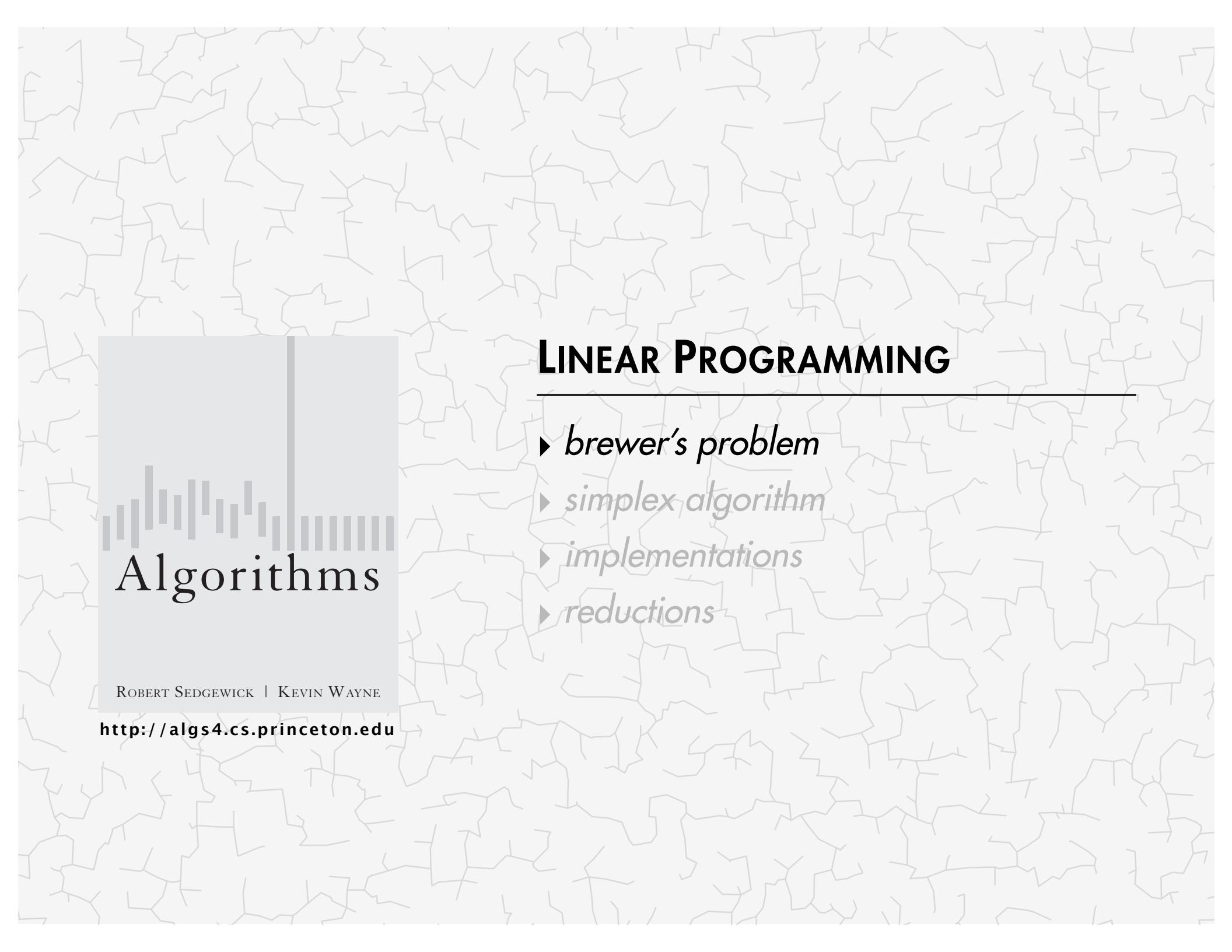
Extreme point property. If there exists an optimal solution to (P), then there exists one that is an extreme point.

- Good news: number of extreme points to consider is **finite**.
- Bad news : number of extreme points can be **exponential!**



local optima are global optima
(follows because objective function is linear
and feasible region is convex)

Greedy property. Extreme point optimal iff no better adjacent extreme point.



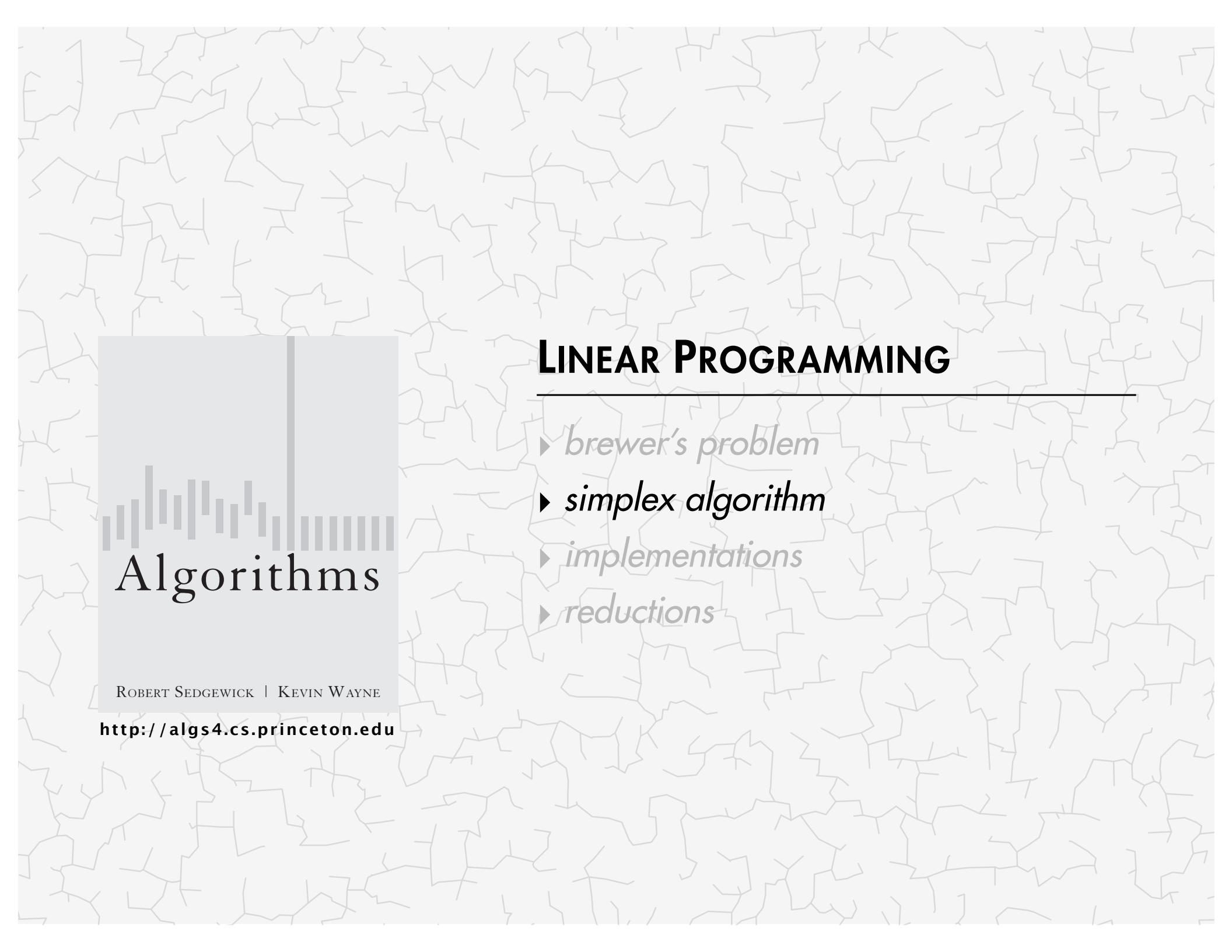
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

Simplex algorithm

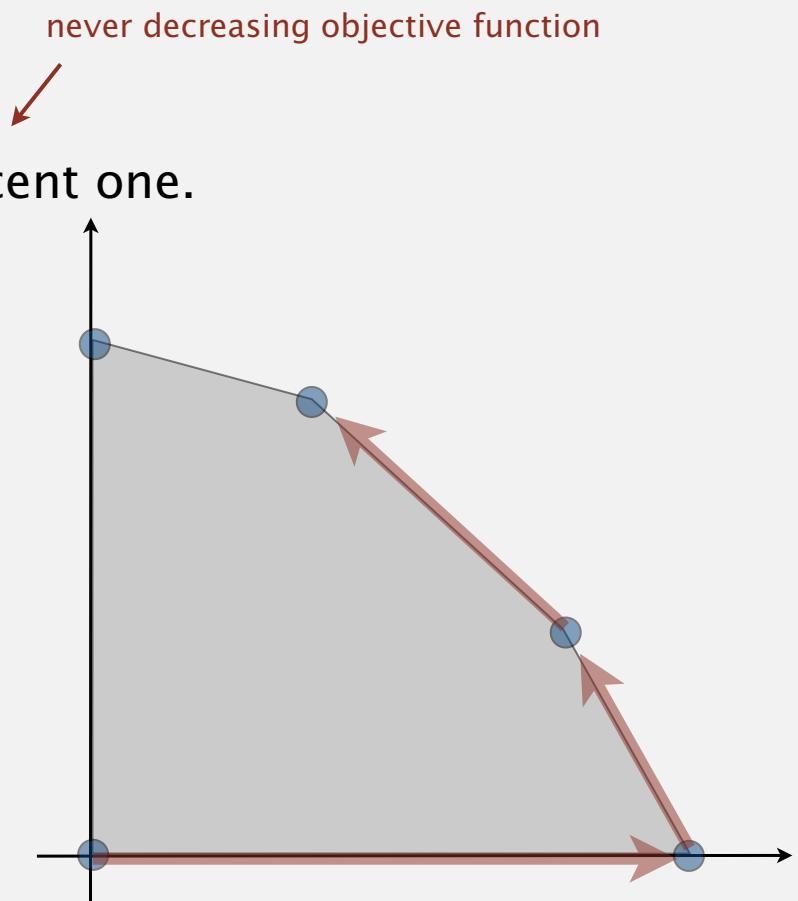
Simplex algorithm. [George Dantzig, 1947]

- Developed shortly after WWII in response to logistical problems, including Berlin airlift.
- Ranked as one of top 10 scientific algorithms of 20th century.

Generic algorithm.

- Start at some extreme point.
- Pivot from one extreme point to an adjacent one.
- Repeat until optimal.

How to implement? Linear algebra.



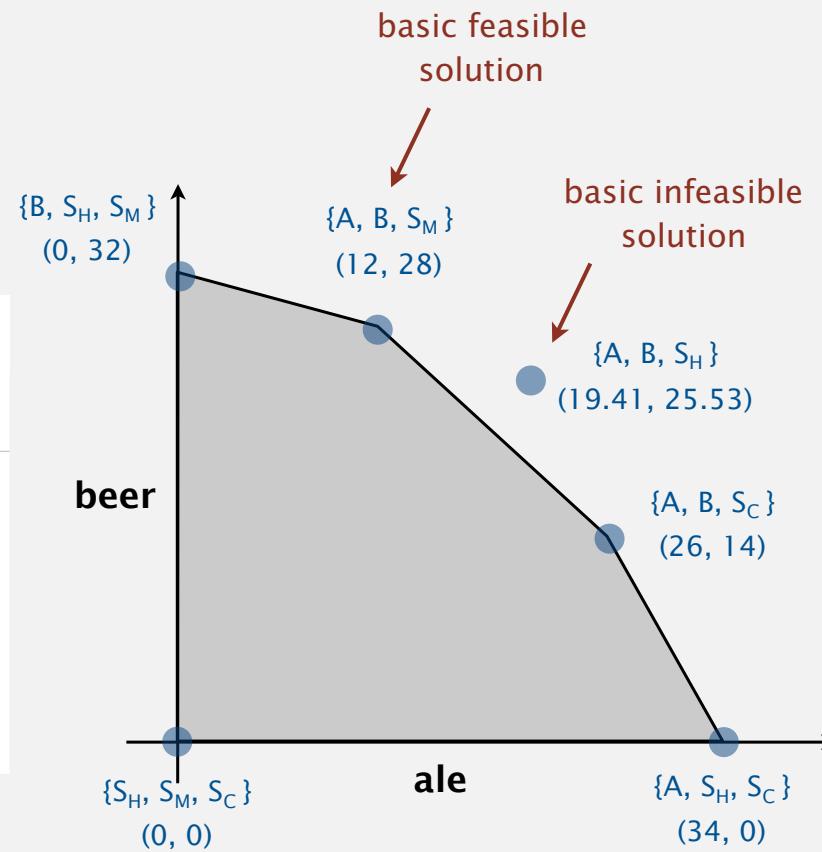
Simplex algorithm: basis

A **basis** is a subset of m of the n variables.

Basic feasible solution (BFS).

- Set $n - m$ nonbasic variables to 0, solve for remaining m variables.
- Solve m equations in m unknowns.
- If unique and feasible \Rightarrow BFS.
- BFS \Leftrightarrow extreme point.

maximize		Z			
		13A	+	23B	$- Z = 0$
<hr/>					
subject to the constraints		5A	+	15B + S_C	$= 480$
		4A	+	4B + S_H	$= 160$
		35A	+	20B + S_M	$= 1190$
		A , B , S_C , S_H , S_M	≥ 0		



Simplex algorithm: initialization

maximize	Z													basis = { S_C, S_H, S_M }
	13A + 23B						-	Z	=	0				A = B = 0
subject to the constraints	5A + 15B + S_C								=	480				$Z = 0$
	4A + 4B + S_H								=	160				$S_C = 480$
	35A + 20B + S_M								=	1190				$S_H = 160$
	A , B , S_C , S_H , S_M								\geq	0				$S_M = 1190$

one basic variable per row

Initial basic feasible solution.



- Start with slack variables $\{S_C, S_H, S_M\}$ as the basis.

- Set non-basic variables A and B to 0.

- 3 equations in 3 unknowns yields $S_C = 480, S_H = 160, S_M = 1190$.

no algebra needed



Simplex algorithm: pivot 1

maximize	Z						
	13A	+	23B				
subject to the constraints	5A	+	15B	+ S_C			
	4A	+	4B		+ S_H		
	35A	+	20B			+ S_M	= 1190
	A	,	B	,	S_C	,	S_H , $S_M \geq 0$

$\text{basis} = \{S_C, S_H, S_M\}$
 $A = B = 0$
 $Z = 0$
 $S_C = 480$
 $S_H = 160$
 $S_M = 1190$

substitute $B = (1/15)(480 - 5A - S_C)$ and add B into the basis
 (rewrite 2nd equation, eliminate B in 1st, 3rd, and 4th equations) ← which basic variable does B replace?

maximize	Z						
	(16/3)A	-	(23/15) S_C				
subject to the constraints	(1/3)A	+	B	+ (1/15) S_C			
	(8/3)A	-	(4/15) S_C	+ S_H			
	(85/3)A	-	(4/3) S_C		+ S_M		= 550
	A	,	B	,	S_C	,	S_H , $S_M \geq 0$

$\text{basis} = \{B, S_H, S_M\}$
 $A = S_C = 0$
 $Z = 736$
 $B = 32$
 $S_H = 32$
 $S_M = 550$

Simplex algorithm: pivot 1

maximize subject to the constraints	$Z = 13A + 23B - Z = 0$						basis = { S_C, S_H, S_M } $A = B = 0$ $Z = 0$ $S_C = 480$ $S_H = 160$ $S_M = 1190$					
	13A	+	23B	-	Z	=						
	5A	+	15B	+ S_C	=	480						
	4A	+	4B	+ S_H	=	160						
	35A	+	20B	+ S_M	=	1190						
	A	,	B	,	S_C	,	S_H	,	S_M	\geq	0	

Q. Why pivot on column 2 (corresponding to variable B)?

- Its objective function coefficient is positive.
(each unit increase in B from 0 increases objective value by \$23)
- Pivoting on column 1 (corresponding to A) also OK.

Q. Why pivot on row 2?

- Preserves feasibility by ensuring $\text{RHS} \geq 0$.
- Minimum ratio rule: $\min \{ 480/15, 160/4, 1190/20 \}$.

Simplex algorithm: pivot 2

maximize	Z						
subject to the constraints	(16/3) A	– (23/15) S _C	– Z	=	-736		
	(1/3) A + B	+ (1/15) S _C		=	32		
	(8/3) A	– (4/15) S _C + S _H		=	32		
	(85/3) A	– (4/3) S _C + S _M		=	550		
	A , B , S _C , S _H , S _M		≥	0			

basis = {B, S_H, S_M}
A = S_C = 0
Z = 736
B = 32
S_H = 32
S_M = 550

substitute A = (3/8)(32 + (4/15)S_C - S_H) and add A into the basis
(rewrite 3rd equation, eliminate A in 1st, 2nd, and 4th equations) ← which basic variable does A replace?

maximize	Z						
subject to the constraints	B	– S _C – 2 S _H	– Z	=	-800		
	A	+ (1/10) S _C + (1/8) S _H		=	28		
		– (1/10) S _C + (3/8) S _H		=	12		
		– (25/6) S _C – (85/8) S _H + S _M		=	110		
	A , B , S _C , S _H , S _M		≥	0			

basis = {A, B, S_M}
S_C = S_H = 0
Z = 800
B = 28
A = 12
S_M = 110

Simplex algorithm: optimality

Q. When to stop pivoting?

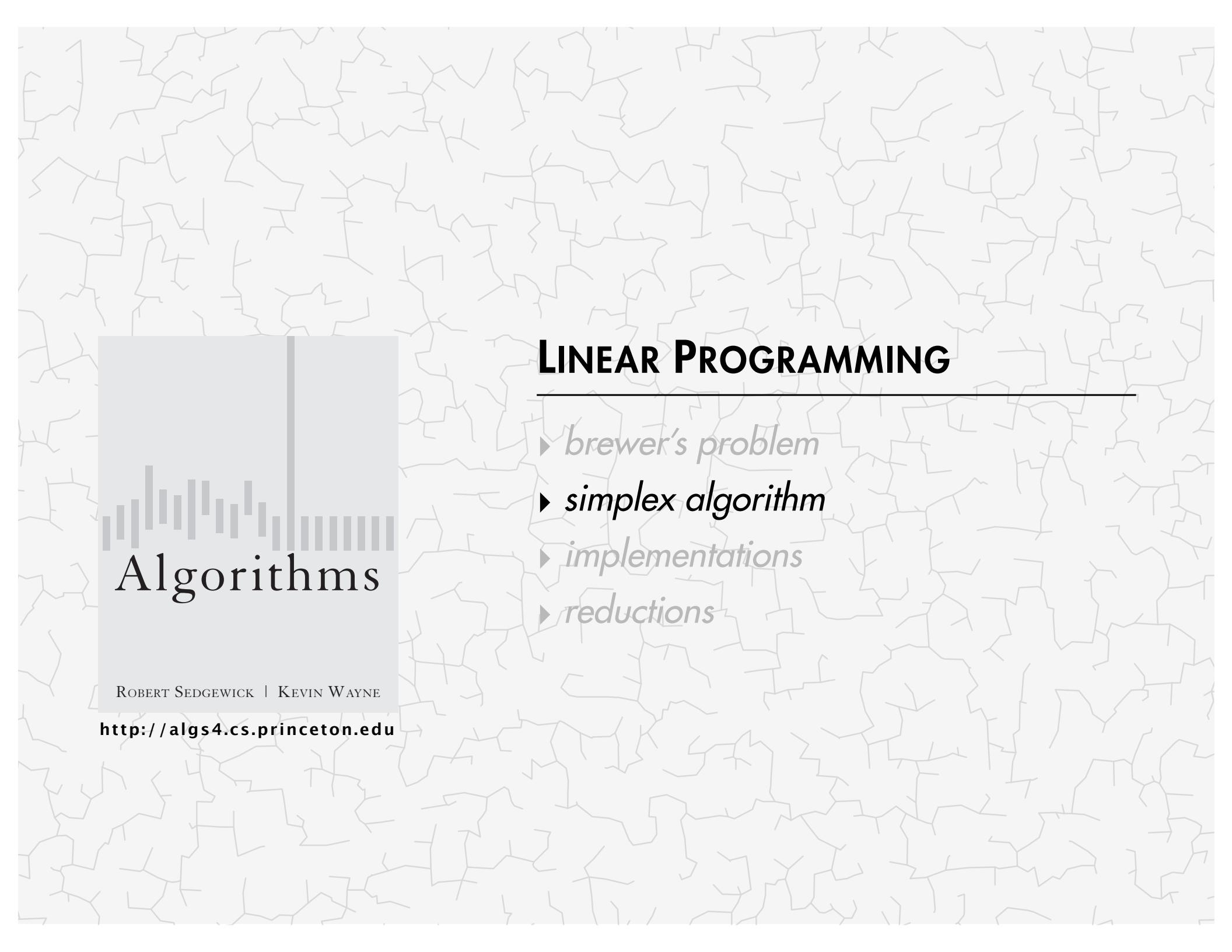
A. When no objective function coefficient is positive.

Q. Why is resulting solution optimal?

A. Any feasible solution satisfies current system of equations.

- In particular: $Z = 800 - S_C - 2 S_H$
- Thus, optimal objective value $Z^* \leq 800$ since $S_C, S_H \geq 0$.
- Current BFS has value 800 \Rightarrow optimal.

maximize		Z						
		- S_C - $2 S_H$ - Z = -800						
subject to the constraints	B	-	(1/10) S_C	+	(1/8) S_H	=	28	basis = {A, B, S_M }
	A	-	(1/10) S_C	+	(3/8) S_H	=	12	$S_C = S_H = 0$
		-	(25/6) S_C	-	(85/8) S_H + S_M	=	110	$Z = 800$ $B = 28$ $A = 12$ $S_M = 110$
A , B ,		S_C ,		S_H ,	S_M	\geq	0	



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

Simplex tableau

Encode standard form LP in a single Java 2D array.

$$\begin{array}{lll} \text{maximize} & Z \\ & 13A + 23B & -Z = 0 \\ \text{subject} & 5A + 15B + S_C & = 480 \\ \text{to the} & 4A + 4B + S_H & = 160 \\ \text{constraints} & 35A + 20B + S_M & = 1190 \\ & A, B, S_C, S_H, S_M & \geq 0 \end{array}$$

5	15	1	0	0	480
4	4	0	1	0	160
35	20	0	0	1	1190
13	23	0	0	0	0

initial simplex tableaux

m	A	I	b
1	c	0	0
n	m	1	

Simplex tableau

Simplex algorithm transforms initial 2D array into solution.

	maximize	Z						
			-	S_C	-	$2 S_H$	-	$Z = -800$
subject to the constraints		B	+	$(1/10) S_C$	+	$(1/8) S_H$	=	28
	A		-	$(1/10) S_C$	+	$(3/8) S_H$	=	12
			-	$(25/6) S_C$	-	$(85/8) S_H + S_M$	=	110
		A , B ,		S_C ,		S_H , S_M	≥ 0	

0	1	1/10	1/8	0	28
1	0	-1/10	3/8	0	12
0	0	-25/6	-85/8	1	110
0	0	-1	-2	0	-800

final simplex tableaux

m			x^*
1	≤ 0	≤ 0	$-Z^*$
n	m	1	

Simplex algorithm: initial simplex tableaux

Construct the initial simplex tableau.

m	A	I	b
1	c	0	0
n	m	m	1

```
public class Simplex
{
    private double[][] a;      // simplex tableaux
    private int m, n;          // M constraints, N variables

    public Simplex(double[][] A, double[] b, double[] c)
    {
        m = b.length;
        n = c.length;
        a = new double[m+1][m+n+1];
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = A[i][j];
        for (int j = n; j < m + n; j++) a[j-n][j] = 1.0;
        for (int j = 0; j < n; j++) a[m][j] = c[j];
        for (int i = 0; i < m; i++) a[i][m+n] = b[i];
    }
}
```

constructor

← put A[][] into tableau

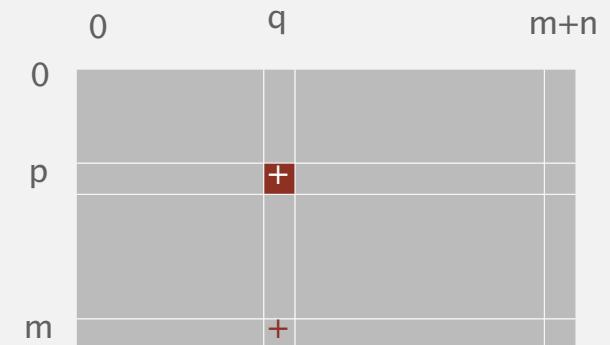
← put I[][] into tableau

← put c[] into tableau

← put b[] into tableau

Simplex algorithm: Bland's rule

Find entering column q using **Bland's rule**:
index of first column whose objective function
coefficient is positive.



```
private int bland()
{
    for (int q = 0; q < m + n; q++)
        if (a[m][j] > 0) return q;

    return -1;
}
```

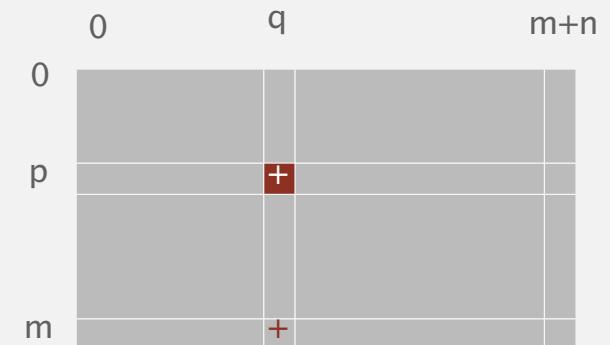
entering column q has positive
objective function coefficient

optimal

Simplex algorithm: min-ratio rule

Find leaving row p using **min ratio rule**.

(Bland's rule: if a tie, choose first such row)



```
private int minRatioRule(int q)
{
    int p = -1;
    for (int i = 0; i < m; i++)
    {
        if (a[i][q] <= 0) continue;
        else if (p == -1) p = i;
        else if (a[i][m+n] / a[i][q] < a[p][m+n] / a[p][q])
            p = i;
    }
    return p;
}
```

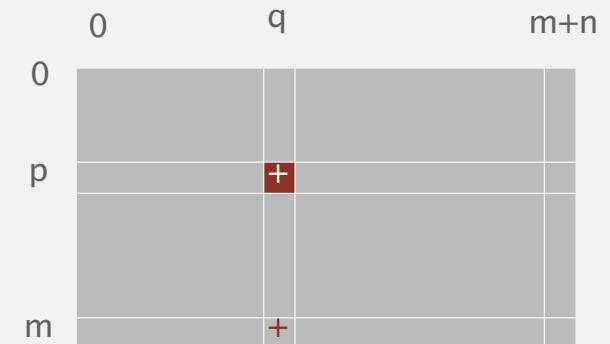
leaving row

consider only positive entries

row p has min ratio so far

Simplex algorithm: pivot

Pivot on element row p , column q .



```
public void pivot(int p, int q)
{
    for (int i = 0; i <= m; i++)
        for (int j = 0; j <= m+n; j++)
            if (i != p && j != q)
                a[i][j] -= a[p][j] * a[i][q] / a[p][q];

    for (int i = 0; i <= m; i++)
        if (i != p) a[i][q] = 0.0;

    for (int j = 0; j <= m+n; j++)
        if (j != q) a[p][j] /= a[p][q];
    a[p][q] = 1.0;
}
```

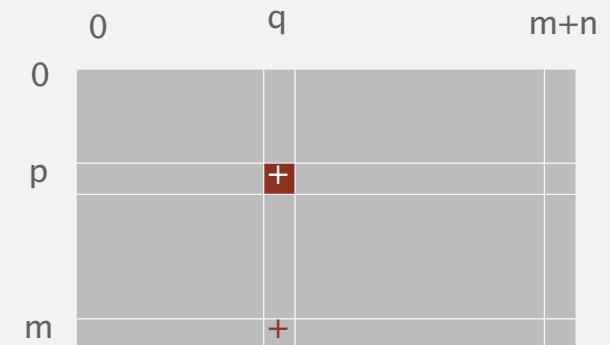
scale all entries but
row p and column q

zero out column q

scale row p

Simplex algorithm: bare-bones implementation

Execute the simplex algorithm.



```
public void solve()
{
    while (true)
    {
        int q = bland();
        if (q == -1) break;

        int p = minRatioRule(q);
        if (p == -1) ...

        pivot(p, q);
    }
}
```

Annotations for the code:

- Annotations point to the following lines:
 - `int q = bland();` ← entering column q (optimal if -1)
 - `int p = minRatioRule(q);` ← leaving row p (unbounded if -1)
 - `pivot(p, q);` ← pivot on row p, column q

Simplex algorithm: running time

Remarkable property. In typical practical applications, simplex algorithm terminates after at most $2(m + n)$ pivots.

Pivoting rules. Carefully balance the cost of finding an entering variable with the number of pivots needed.

- No pivot rule is known that is guaranteed to be polynomial.
- Most pivot rules are known to be exponential (or worse) in worst-case.

Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time

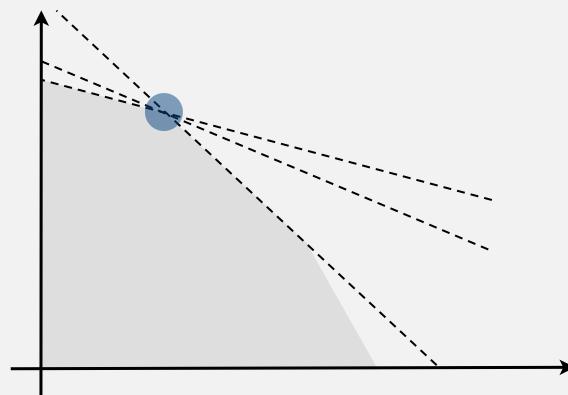
Daniel A. Spielman^{*}
Department of Mathematics
M.I.T.
Cambridge, MA 02139
spielman@mit.edu

Shang-Hua Teng[†]
Akamai Technologies Inc. and
Department of Computer Science
University of Illinois at Urbana-Champaign
steng@cs.uiuc.edu

Simplex algorithm: degeneracy

Degeneracy. New basis, same extreme point.

"stalling" is common in practice



Cycling. Get stuck by cycling through different bases that all correspond to same extreme point.

- Doesn't occur in the wild.
- Bland's rule guarantees finite # of pivots.

choose lowest valid index for
entering and leaving columns

Simplex algorithm: implementation issues

To improve the bare-bones implementation.

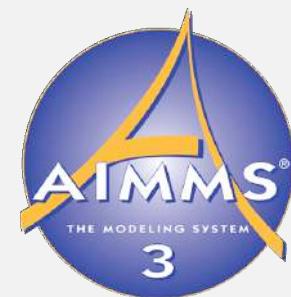
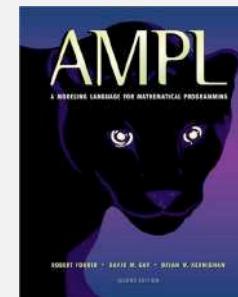
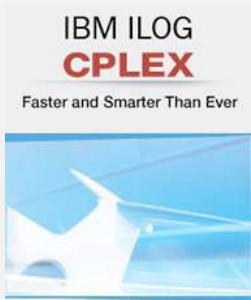
- Avoid stalling. ← requires artful engineering
- Maintain sparsity. ← requires fancy data structures
- Numerical stability. ← requires advanced math
- Detect infeasibility. ← run "phase I" simplex algorithm
- Detect unboundedness. ← no leaving row

Best practice. Don't implement it yourself!

Basic implementations. Available in many programming environments.

Industrial-strength solvers. Routinely solve LPs with millions of variables.

Modeling languages. Simplify task of modeling problem as LP.



LP solvers: industrial strength

“ a benchmark production planning model solved using linear programming would have taken 82 years to solve in 1988, using the computers and the linear programming algorithms of the day. Fifteen years later—in 2003—this same model could be solved in roughly 1 minute, an improvement by a factor of roughly 43 million. Of this, a factor of roughly 1,000 was due to increased processor speed, whereas a factor of roughly 43,000 was due to improvements in algorithms! ”

— *Designing a Digital Future*

(Report to the President and Congress, 2010)

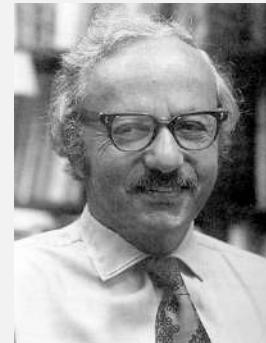


Brief history

- 1939. Production, planning. [Kantorovich]
- 1947. Simplex algorithm. [Dantzig]
- 1947. Duality. [von Neumann, Dantzig, Gale-Kuhn-Tucker]
- 1947. Equilibrium theory. [Koopmans]
- 1948. Berlin airlift. [Dantzig]
- 1975. Nobel Prize in Economics. [Kantorovich and Koopmans]
- 1979. Ellipsoid algorithm. [Khachiyan]
- 1984. Projective-scaling algorithm. [Karmarkar]
- 1990. Interior-point methods. [Nesterov-Nemirovskii, Mehorta, ...]



Kantorovich



George Dantzig



von Neumann



Koopmans



Khachiyan



Karmarkar

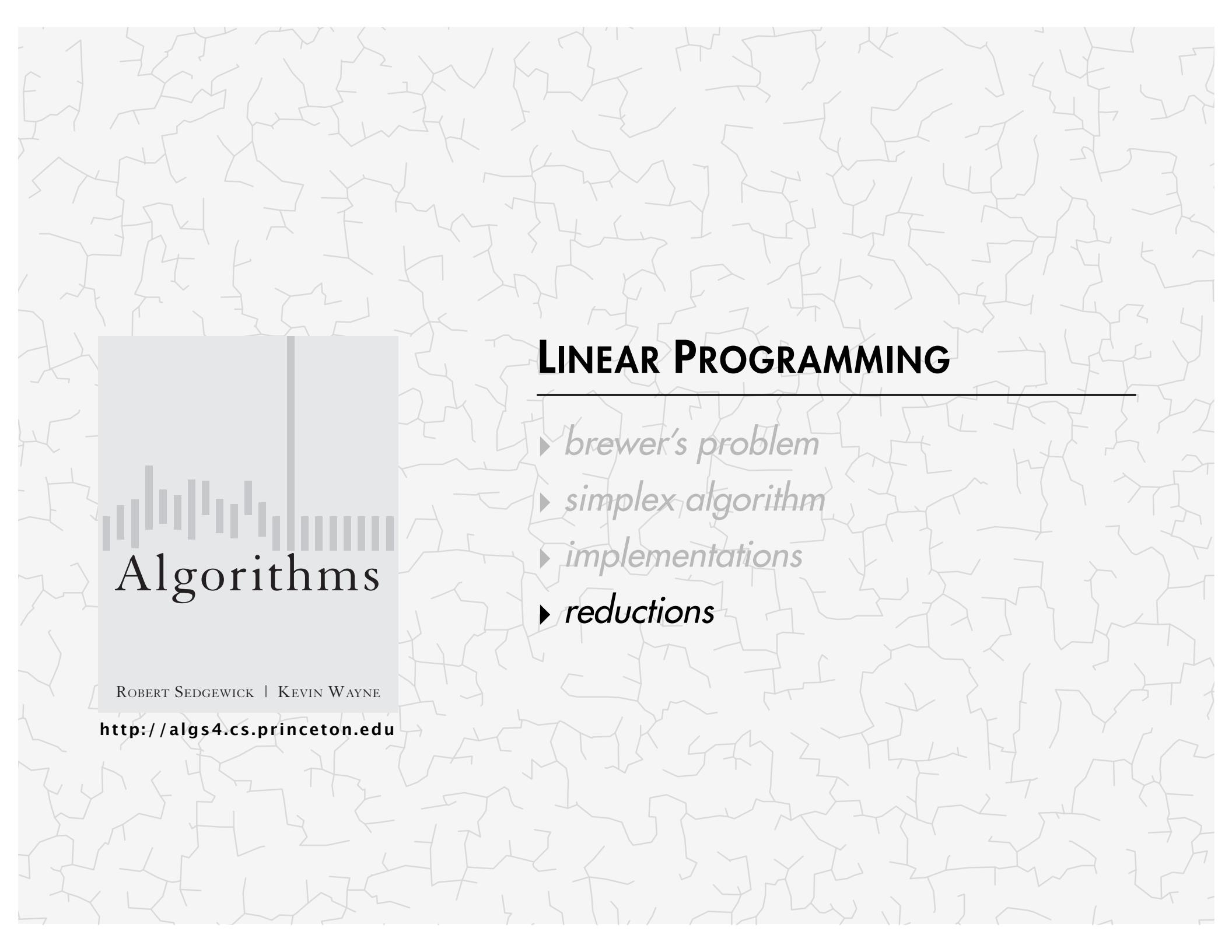
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

Reductions to standard form

Minimization problem. Replace $\min 13A + 15B$ with $\max -13A - 15B$.

\geq constraints. Replace $4A + 4B \geq 160$ with $4A + 4B - S_H = 160, S_H \geq 0$.

Unrestricted variables. Replace B with $B = B_0 - B_1, B_0 \geq 0, B_1 \geq 0$.

nonstandard form

$$\text{minimize} \quad 13A + 15B$$

$$\text{subject to:} \quad 5A + 15B \leq 480$$

$$4A + 4B \geq 160$$

$$35A + 20B = 1190$$

$$A \geq 0$$

B is unrestricted

standard form

$$\text{maximize} \quad -13A - 15B_0 + 15B_1$$

$$\text{subject to:} \quad 5A + 15B_0 - 15B_1 + S_C = 480$$

$$4A + 4B_0 - 4B_1 - S_H = 160$$

$$35A + 20B_0 - 20B_1 = 1190$$

$$A \quad B_0 \quad B_1 \quad S_C \quad S_H \geq 0$$

Modeling

Linear “programming” (1950s term) = reduction to LP (modern term).

- Process of formulating an LP model for a problem.
- Solution to LP for a specific problem gives solution to the problem.

1. Identify **variables**.
2. Define **constraints** (inequalities and equations).
3. Define **objective function**.
4. Convert to standard form. ← software usually performs
this step automatically

Examples.

- Maxflow.
- Shortest paths.
- Bipartite matching.
- Assignment problem.
- 2-person zero-sum games.

...

Maxflow problem (revisited)

Input. Weighted digraph G , single source s and single sink t .

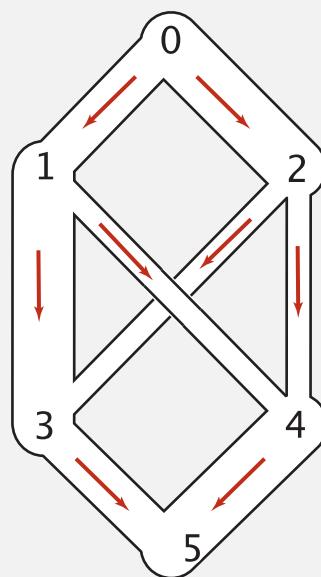
Goal. Find maximum flow from s to t .

maxflow problem

$V \rightarrow$ 6
 $E \rightarrow$ 8

0	1	2.0
0	2	3.0
1	3	3.0
1	4	1.0
2	3	1.0
2	4	1.0
3	5	2.0
4	5	3.0

↑ capacities



Modeling the maxflow problem as a linear program

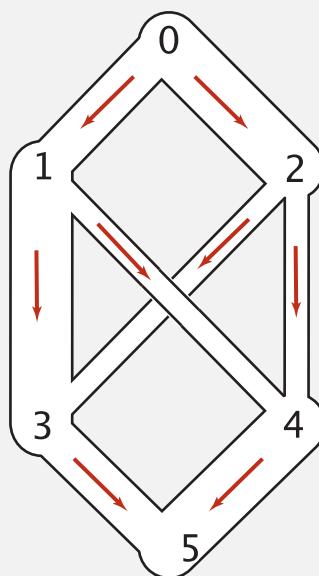
Variables. x_{vw} = flow on edge $v \rightarrow w$.

Constraints. Capacity and flow conservation.

Objective function. Net flow into t .

maxflow problem

V	E	capacities
6		
8		
0 1	2.0	
0 2	3.0	
1 3	3.0	
1 4	1.0	
2 3	1.0	
2 4	1.0	
3 5	2.0	
4 5	3.0	



LP formulation

Maximize $x_{35} + x_{45}$
subject to the constraints

$$\begin{aligned} 0 \leq x_{01} &\leq 2 \\ 0 \leq x_{02} &\leq 3 \\ 0 \leq x_{13} &\leq 3 \\ 0 \leq x_{14} &\leq 1 \\ 0 \leq x_{23} &\leq 1 \\ 0 \leq x_{24} &\leq 1 \\ 0 \leq x_{35} &\leq 2 \\ 0 \leq x_{45} &\leq 3 \end{aligned} \quad \left. \right\} \text{capacity constraints}$$

$$\begin{aligned} x_{01} &= x_{13} + x_{14} \\ x_{02} &= x_{23} + x_{24} \\ x_{13} + x_{23} &= x_{35} \\ x_{14} + x_{24} &= x_{45} \end{aligned} \quad \left. \right\} \text{flow conservation constraints}$$

Maximum cardinality bipartite matching problem

Input. Bipartite graph.

Goal. Find a matching of maximum cardinality.

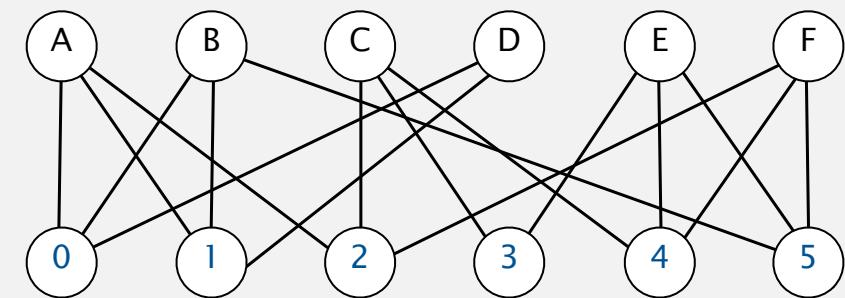
↑
set of edges with no vertex appearing twice

Interpretation. Mutual preference constraints.

- People to jobs.
- Students to writing seminars.

Alice	Adobe
	Alice, Bob, Dave
Bob	Apple
	Alice, Bob, Dave
Carol	Google
	Alice, Carol, Frank
Dave	IBM
	Carol, Eliza
Eliza	Sun
	Carol, Eliza, Frank
Frank	Yahoo
	Bob, Eliza, Frank

Example: job offers



matching of cardinality 6:

A-1, B-5, C-2, D-0, E-3, F-4

Maximum cardinality bipartite matching problem

LP formulation. One variable per pair.

Interpretation. $x_{ij} = 1$ if person i assigned to job j .

maximize	$x_{A0} + x_{A1} + x_{A2} + x_{B0} + x_{B1} + x_{B5} + x_{C2} + x_{C3} + x_{C4}$ $+ x_{D0} + x_{D1} + x_{E3} + x_{E4} + x_{E5} + x_{F2} + x_{F4} + x_{F5}$	
subject to the constraints	at most one job per person	at most one person per job
	$x_{A0} + x_{A1} + x_{A2} \leq 1$	$x_{A0} + x_{B0} + x_{D0} \leq 1$
	$x_{B0} + x_{B1} + x_{B5} \leq 1$	$x_{A1} + x_{B1} + x_{D1} \leq 1$
	$x_{C2} + x_{C3} + x_{C4} \leq 1$	$x_{A2} + x_{C2} + x_{F2} \leq 1$
	$x_{D0} + x_{D1} \leq 1$	$x_{C3} + x_{E3} \leq 1$
	$x_{E3} + x_{E4} + x_{E5} \leq 1$	$x_{C4} + x_{E4} + x_{F4} \leq 1$
	$x_{F2} + x_{F4} + x_{F5} \leq 1$	$x_{B5} + x_{E5} + x_{F5} \leq 1$
	all $x_{ij} \geq 0$	

Theorem. [Birkhoff 1946, von Neumann 1953]

All extreme points of the above polyhedron have integer (0 or 1) coordinates.

Corollary. Can solve matching problem by solving LP. ← not usually so lucky!

Linear programming perspective

Q. Got an optimization problem?

Ex. Maxflow, bipartite matching, shortest paths, ... [many, many, more]

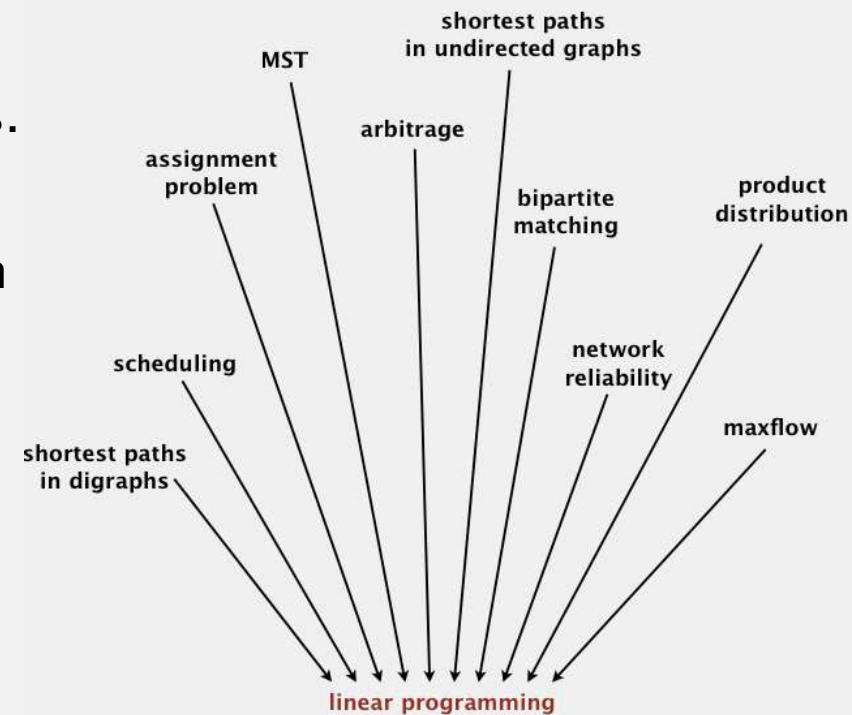
Approach 1: Use a specialized algorithm to solve it.

- Algorithms 4/e.
- Vast literature on algorithms.

Approach 2: Use linear programming.

- Many problems are easily modeled as LPs.
- Commercial solvers can solve those LPs.
- Might be slower than specialized solution
(but you might not care).

Got an LP solver? Learn to use it!



Universal problem-solving model (in theory)

Is there a universal problem-solving model?

- Maxflow.
- Shortest paths.
- Bipartite matching.
- Assignment problem.
- Multicommodity flow.
- ...
- Two-person zero-sum games.
- Linear programming.
- ...

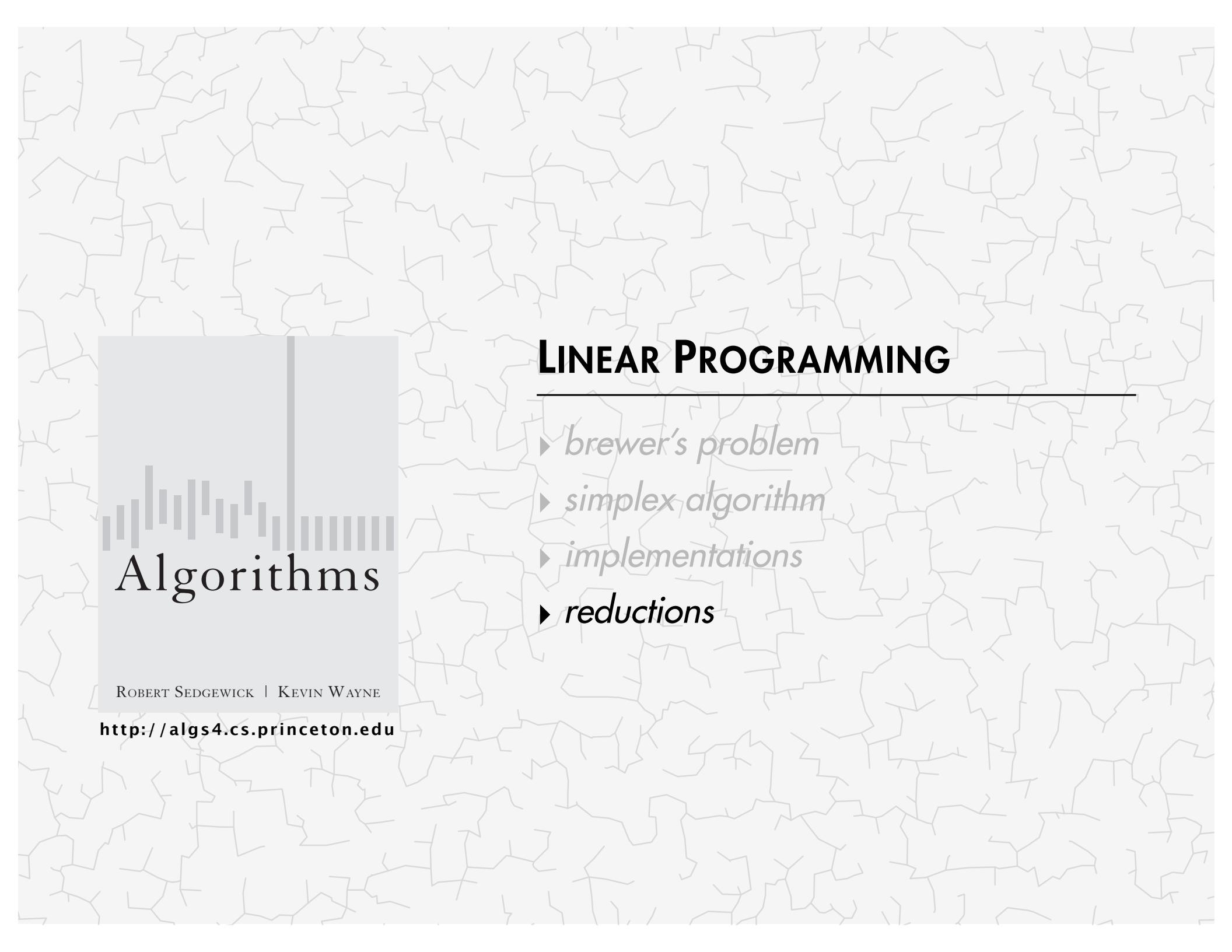
tractable

- Factoring
- NP-complete problems.
- ...

intractable ?

see next lecture

Does $P = NP$? No universal problem-solving model exists unless $P = NP$.



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

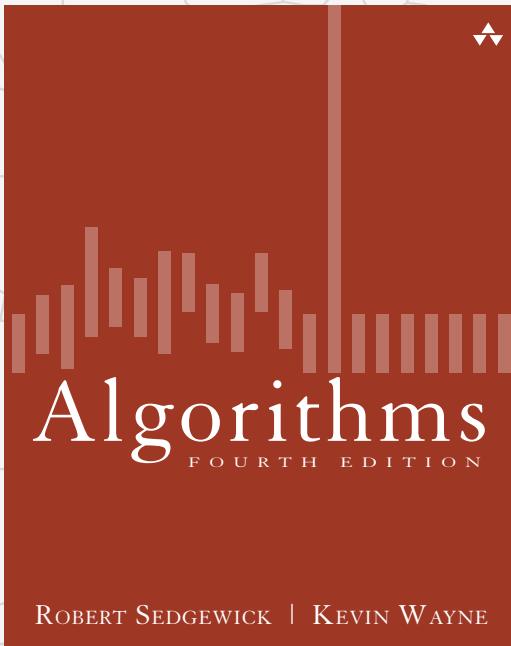
<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



<http://algs4.cs.princeton.edu>

LINEAR PROGRAMMING

- ▶ *brewer's problem*
- ▶ *simplex algorithm*
- ▶ *implementations*
- ▶ *reductions*