COS 226

Algorithms and Data Structures
Princeton University
Spring 2004

Robert Sedgewick

Overview

What is COS 226?

- Intermediate-level survey course on algorithms and data structures
- Programming and problem-solving with applications

A few applications enabled by good algorithms

Multimedia. CD player, DVD, MP3, JPG, DivX, HDTV.
Internet. Packet routing, Google, Akamai.
Secure communications. Cell phones, e-commerce.
Information processing. Database search, data compression.
Computers. Circuit layout, file system, compilers.
Computer graphics. Hollywood movies, video games.
Biology. Human genome project, protein folding.
Astrophysics. N-body simulation.
Transportation. Airline crew scheduling, map routing.
. . .

Algorithm: method for solving a problem

Data structure: method for storing information

Why Study Algorithms?

Using a computer?

- Want it to go faster? Process more data?
- Want it to do something that would otherwise be impossible?

Approach 1: Buy a supercomputer

- Might be costly.
- Will improve performance by a constant factor at best.

cannot rescue a bad algorithm

Approach 2: Use a good algorithm

- Cheap or free.
- Huge performance-improvement factors available.

might rescue a slow computer

Algorithms as a field of study.

- Old enough that basics are known.
- New enough that new discoveries arise.
- Burgeoning application areas.
- Philosophical implications.

The Usual Suspects

Lectures: Robert Sedgewick

- TTh 11-12:20, CS 105.

Precepts: Sayyen Kale, Seshadri Comandur (Sesh)

- T 1:30, Friend 005.
- T 3:30, Friend 005.
- Clarify programming assignments, exercises, lecture material.
- First precept meets 9/14.

Coursework and Grading

Weekly programming assignments: 45%
- Due Fridays 11:59pm, starting 9/17.

Weekly written exercises: 15%
- Due at beginning of Tuesday lecture, starting 9/14.

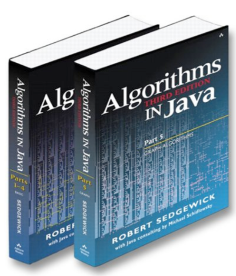
Change: no punts

Exams:
- Closed book with cheatsheet.
- Midterm. 15%
- Final. 25%

Staff discretion. Adjust borderline cases.

Course Materials

Course web page. `http://www.princeton.edu/~cos226`
- Syllabus.
- Programming assignments.
- Exercises.
- Lecture notes.
- Old exams.



Algorithms in Java, 3rd edition.
- Parts 1-4 (COS 126 text).
- Part 5 (graph algorithms).

Algorithms in C, 2nd edition.
- Strings and geometry handouts.

Questionnaire

Please fill out questionnaire so that we can adapt course as needed.
- Let us know who you are.
- One of the precept times will be canceled. Tell us which one.
- Let us know your programming and Java experience.
  - COS 126 in Java: advantage in knowing Java
  - COS 217: advantage in programming experience
  - ELE 101, ORF 201: talk to me after class

Union Find

Quick find

Quick union

Weighted quick union

Path compression

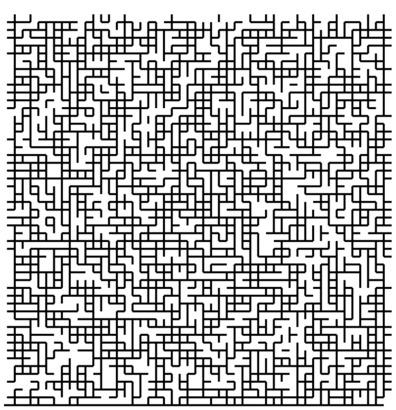
Reference: Chapter 1, Algorithms in Java, 3rd Edition, Robert Sedgewick.

An Example Problem: Network Connectivity

Network connectivity.

- Nodes at grid points.
- Add connections between pairs of nodes.
- Is there a path from node A to node B?

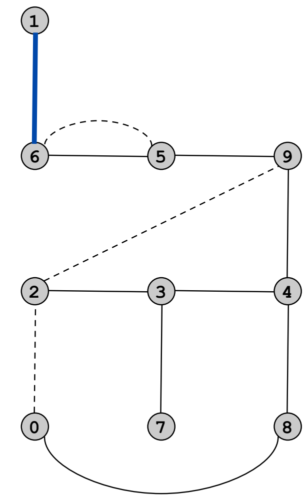
A



B

Network Connectivity

in	out	evidence
3 4	3 4	
4 9	4 9	
8 0	8 0	
2 3	2 3	
5 6	5 6	
2 9		(2–3–4–9)
5 9	5 9	
7 3	7 3	
4 8	4 8	
5 6		(5–6)
0 2		(2–3–4–8–0)
6 1	6 1	



Union-Find Abstraction

What are critical operations we need to support?

- N objects.
  - grid points
- FIND: test whether two objects are in same set.
  - is there a connection between A and B?
- UNION: merge two sets.
  - add a connection

Design efficient data structure to store connectivity information and algorithms for UNION and FIND.

- Number of operations M can be huge.
- Number of objects N can be huge.

Other Applications

More union-find applications.

- Hex.
- Percolation.
- Image processing.
- Minimum spanning tree.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Compiling equivalence statements in Fortran.
- Micali-Vazarani algorithm for nonbipartite matching.
- Weihe's algorithm for edge-disjoint s-t paths in planar graphs.
- Scheduling unit-time tasks to P processors so that each job finishes between its release time and deadline.
- . . .

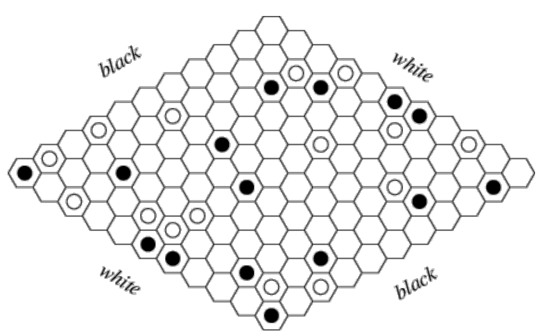
References.

- A Linear Time Algorithm for a Special Case of Disjoint Set Union, Gabow and Tarjan.
- The Design and Analysis of Computer Algorithms, Aho, Hopcroft, and Ullman.

UF Application: Hex

Hex. (Piet Hein 1942, John Nash 1948, Parker Brothers 1962)

- Two players alternate in picking a cell in a hex grid.
- Black: make a black path from upper left to lower right.
- White: make a white path from lower left to upper right.
- How to detect when a player has won?



Objects

Applications involve manipulating objects of all sorts

- Pixels in a digital photo.
- Computers in a network.
- Transistors in a computer chip.
- Web pages on the Internet.
- Metallic sites in a composite system.

When programming, it is convenient to name the objects 0 to N-1

- Details not relevant to union-find.
- Integers allow quick access to object-related info (array indices).

Quick-Find Algorithm

Data structure. (integer between 0 and N-1)

- Maintain array `id[]` with name for each of N elements.
- p and q are connected iff they have the same id.

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

no connections

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

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

Find. To check if p and q are connected, see if they have same id.

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

id[2] = id[4]
so 2 is connected to 4

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

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

union of 3 and 6
connects 2,3,4,5,6, and 9

Quick-Find Algorithm

Data structure. *integer between 0 and N-1*

- Maintain array `id[]` with name for each of N elements.
- If p and q are connected, then they have the same id.
- Initially, set id of each element to itself.

```
for (int i = 0; i < N; i++)
   id[i] = i;
```

N operations

Find. To check if p and q are connected, see if they have same id.

```
return (id[p] == id[q]);
```

1 operation

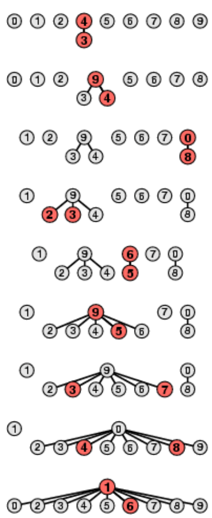
Union. To merge components containing p and q, change all entries with `id[p]` to `id[q]`.

```
int pid = id[p];
for (int i = 0; i < N; i++)
   if (id[i] == pid) id[i] = id[q];
```

N operations

Quick-Find

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



Problem Size and Computation Time

Quick-Find is "Slow-Union"

- MN operations per second.
- When M is proportional to N, time is quadratic

Ex. Huge problem for quick find.

- 10^{10} edges connecting 10^9 nodes.
- Quick-find might take 10^{20} operations. (10 ops per query)
- 3,000 years of computer time! (10^9 ops/sec; 10^9 words of memory)

Paradoxically, quadratic algorithms get worse with newer equipment.

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

Quick-Union

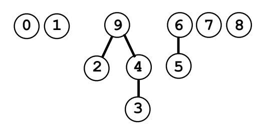
Data structure: forest (set of trees)

- Maintain parent-link array `id[]` for each of N elements.
- p and q are connected if they are in the same tree

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

parent of i in i's tree

Find. Check if p and q have same root.

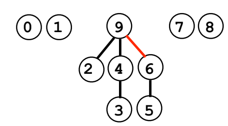


3's root is 9
5's root is 6
3 and 5 not connected

Union. Set the id of q's root to p's root.

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

only one change

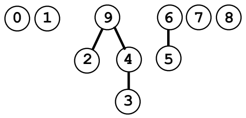


Quick-Union

Data structure: forest (set of trees)

- Maintain array id[] for each of N elements.
- Root of element x = id[id[id[...id[p]...]]]

keep going until it doesn't change




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

time proportional to depth of x

Find. Check if p and q have same root.

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

time proportional to depth of p and q

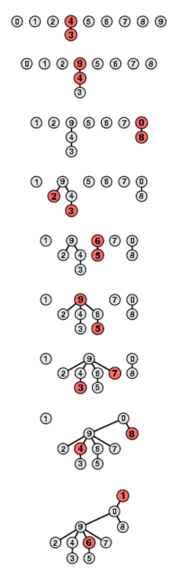
Union. Set the id of p's root to q's root.

```
int i = root(p);
int j = root(q);
id[i] = j;
```

time proportional to depth of p and q

Quick-Union

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



Weighted Quick-Union

Quick-find defect.

- UNION is too expensive.
- Trees are flat, but too much work to keep them flat.

Quick-union defect.

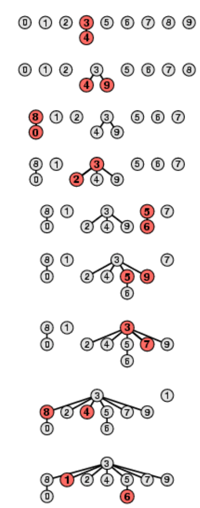
- Finding the root can be expensive.
- Trees could get tall.

Weighted quick-union.

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

Weighted Quick-Union

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



Weighted Quick-Union

Data structure: disjoint forests.

- Also maintain array `sz[i]` that counts the number of elements in the tree rooted at i.

Find. Same as quick union.

Union. Same as quick union, but merge smaller tree into the larger tree and update the `sz[]` array.

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

Analysis.

- FIND takes time proportional to depth of p and q in tree.
- with WQU, depth is guaranteed to be not more than lgN
- UNION takes constant time, given roots.

Needs proof!

Weighted Quick-Union

Is performance improved?

- Theory: lg N per union or find operation (in the worst case).
- Practice: constant time (for typical cases).

Ex. Huge practical problem.

- 10^{10} edges connecting 10^9 nodes.
- Reduces time from 3,000 years to 1 minute.
- Supercomputer wouldn't help much.
- Good algorithm makes solution possible.

Stop at guaranteed acceptable performance?

- Not hard to improve algorithm further.

Weighted Quick-Union with Path Compression

Path compression.

- Add second loop to `root` to compress tree that sets the id of every examined node to the root.
- Simple one-pass variant: make each element point to grandparent.

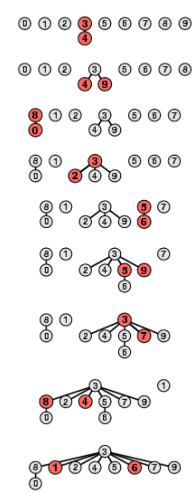
```
public int root(int x) {
   while (x != id[x]) {
      id[x] = id[id[x]];
      x = id[x];
   }
   return x;
}
```

only one extra line of code

- No reason not to!
- In practice, keeps tree almost completely flat.

Weighted Quick-Union with Path Compression

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



Weighted Quick-Union with Path Compression

Theorem. A sequence of M union and find operations on N elements takes O(N + M lg* N) time.

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

Remark. lg* N is a constant in this universe.

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

Linear algorithm?

- Cost within constant factor of reading in the data.
- Theory: WQUPC is not quite linear.
- Practice: WQUPC is linear.

Bottom line:

Cellphone running WQUPC will beat supercomputer running QF!

A Scientific Application: Percolation

Percolation phase-transition.

- Two parallel conducting bars (top and bottom)
- Interior sites intially all insulators
- Electricity flows between neighbors if both are occupied by conductors
- Each interior site is randomly made a conductor with probability p

0	0	0	0	0	0	0	0	0	0	0	0
2	3	4	0	6	0	8	9	10	11	12	0
14	15	0	0	0	0	20	21	22	23	24	0
14	14	28	29	30	31	32	33	34	35	36	0
14	39	40	1	42	43	32	45	46	1	1	49
50	1	52	1	54	55	56	57	58	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1

insulator

Q: What is threshold p* at which electricity flows between top and bottom?
A: ~.592746 for square lattices

constant only known via simulation---enabled by fast algorithm like QFWPC

Lessons

Union-find summary.
Can solve problem for little more than the cost of collecting data

Algorithm	worst-case time
Quick Find	MN
Quick Union	MN
Weighted QU	N + M log N
QU with path compression	N + M log N
WQUPC	5(M+N)

M union-find ops on a set of N elements

Simple algorithms can be very useful.

- Start with brute force approach.
  - don't use for large problems
  - can't use for huge problems
- Strive for worst-case performance guarantees. (might be nontrivial to analyze)
- Identify fundamental abstractions. union-find, disjoint forests