Algorithms and Data Structures
Princeton University
Fall 2005

Kevin Wayne

Overview

What is COS 226?

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

Topic	Data Structures and Algorithms
data types	stack, queue, list, union-find, priority queue
sorting	quicksort, mergesort, heapsort, radix sorts
searching	hash table, BST, red-black tree, B-tree
graphs	DFS, Prim, Kruskal, Dijkstra, Ford-Fulkerson
strings	KMP, Rabin-Karp, TST, Huffman, LZW
geometry	Graham scan, k-d tree, Voronoi diagram

A misperception: *algiros* [painful] + *arithmos* [number].

Impact of Great Algorithms

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

> "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*

Why Study Algorithms?

Using a computer?

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

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. Kevin Wayne (Kevin)
- MW 11-12:20, Bowen 222.

Precepts. Harlan Yu (Harlan), Keith Morley (Keith)
- T 12:30, Friend 110.
- T 3:30, Friend 111.
- Clarify programming assignments, exercises, lecture material.
- First precept meets 9/20.

Coursework and Grading

Regular programming assignments: 45%
- Due 11:59pm, starting 9/26.
- More details next lecture.

Weekly written exercises: 15%
- Due at beginning of Thursday lecture, starting 9/22.

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

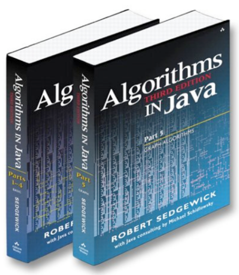
Staff discretion. Adjust borderline cases.

Course Materials

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

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

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

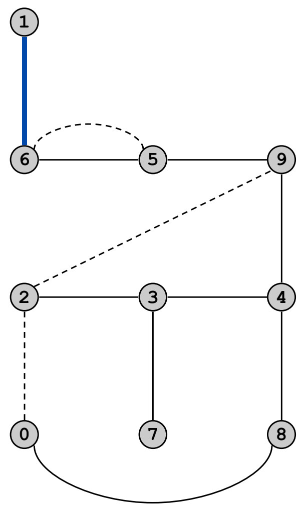


Union Find

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

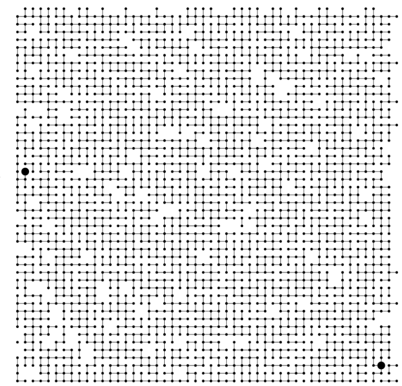
Network Connectivity

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	



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?



Union-Find Abstraction

What are critical operations we need to support?
- Objects.

  `0 1 2 3 4 5 6 7 8 9` grid points

- Disjoint sets of objects.

  `0 1 2-3-9 5-6 7 4-8` subsets of connected grid points

- Find: are objects 2 and 9 in the same set?

  `0 1 2-3-9 5-6 7 4-8` are two grid points connected?

- Union: merge sets containing 3 and 8.

  `0 1 2-3-4-8-9 7 8-4` add a connection between two grid points

Union-Find Abstraction

What are critical operations we need to support?
- Objects.
- Disjoint sets of objects.
- Find: are two objects in the same set?
- Union: replace sets containing two items by their union.

Goal. Design efficient data structure for union and find.
- Number of operations M can be huge.
- Number of objects N can be huge.

Objects

Applications involve manipulating objects of all types.

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

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

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

Quick-Find [eager approach]

Data structure.

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

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

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

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

id[3] = 9; id[6] = 6
3 and 6 not connected

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

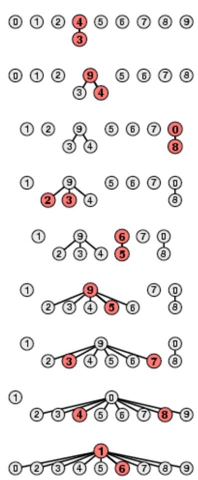
i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	6	6	6	6	6	7	8	6

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

many values can change

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



Quick-Find: Java Implementation

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

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

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

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

set id of each object to itself

1 operation

N operations

Problem Size and Computation Time

Rough standard for 2000.

- 10^9 operations per second.
- 10^9 words of main memory.
- Touch all words in approximately 1 second. [unchanged since 1950!]

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!

Paradoxically, quadratic algorithms get worse with newer equipment.

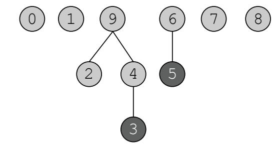
- New computer may be 10x as fast.
- But, has 10x as much memory so problem may be 10x bigger.
- With quadratic algorithm, takes 10x as long!

Quick-Union

Data structure.

- Integer array `id[]` of size N.
- Interpretation: `id[x]` is parent of `x`.
- Root of `x` is `id[id[id[...id[x]...]]]`. ← keep going until it doesn't change

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



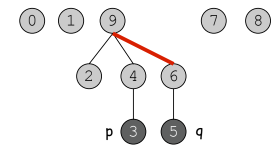
3's root is 9; 5's root is 6
3 and 5 are not connected

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

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

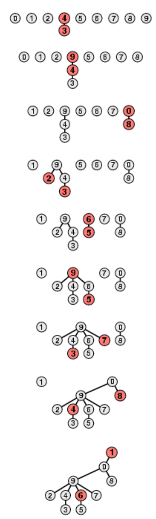
i	0	1	2	3	4	5	6	7	8	9
id[i]	0	1	9	4	9	6	9	7	8	9

↑ only one value changes



Quick-Union

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



Quick-Union: Java Implementation

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

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

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

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

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

root: time proportional to depth of x

find: time proportional to depth of p and q

unite: time proportional to depth of p and q

Summary

Quick-find defect.
- Union too expensive.
- Trees are flat, but too hard to keep them flat.

Quick-union defect.
- Finding the root can be expensive.
- Trees can get tall.

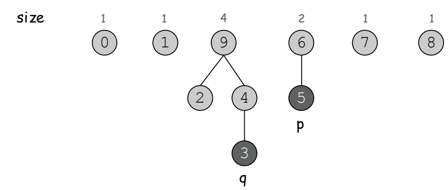
Data Structure	Union	Find
Quick-find	N	1
Quick-union	1 †	N

† union of two root nodes

Weighted Quick-Union

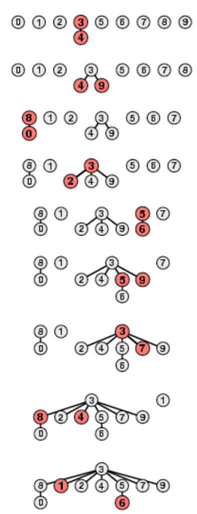
Weighted quick-union.
- Modify quick-union to avoid tall trees.
- Keep track of size of each component.
- Balance by linking small tree below large one.

Ex: union of 5 and 3.
- Quick union: link 9 to 6.
- Weighted quick union: link 6 to 9.



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: Java Implementation

Java implementation.
- Almost identical to quick-union.
- Maintain extra array `sz[]` to count number of elements in the tree rooted at i.

Find. Identical to 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]; }
```

Weighted Quick-Union: Analysis

Analysis.

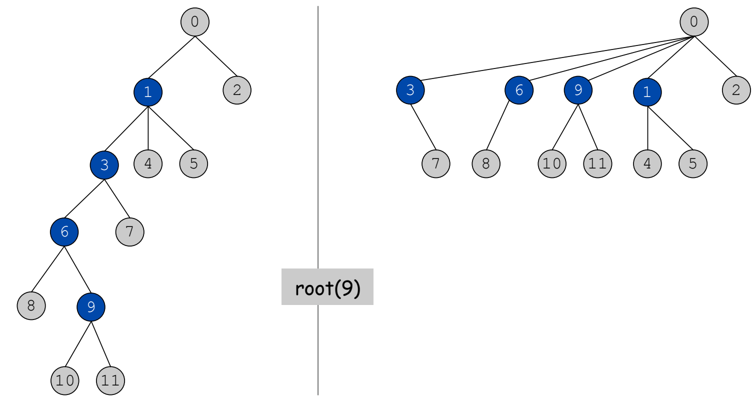
- Find: takes time proportional to depth of p and q.
- Union: takes constant time, given roots.
- Fact: depth is at most 1 + lg N. [needs proof]

Data Structure	Union	Find
Quick-find	N	1
Quick-union	1 †	N
Weighted QU	lg N	lg N

Stop at guaranteed acceptable performance? No, can improve further.

Path Compression

Path compression. Just after computing the root of x, set id of each examined node to root(x).



Weighted Quick-Union with Path Compression

Path compression.

- Standard implementation: add second loop to `root` to set the id of each examined node to the root.
- Simpler one-pass variant: make each examined node point to its grandparent.

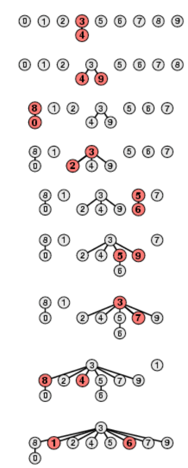
```
public int root(int x) {
   while (x != id[x]) {
      id[x] = id[id[x]];
      x = id[x];
   }
   return x;
}
```

← only one extra line of code !

In practice. No reason not to! 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
4	2
16	3
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.

Context

Ex. Huge practical problem.

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

Bottom line. WQUPC on cell phone beats QF on supercomputer!

M union-find ops on a set of N elements

Algorithm	Time
Quick-find	M N
Quick-union	M N
Weighted QU	N + M log N
Path compression	N + M log N
Weighted + path	5 (M + N)

Applications

Other Applications

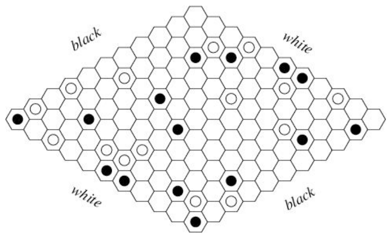
Union-find applications.

- Hex.
- Percolation.
- Connectivity.
- Image processing.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Compiling equivalence statements in Fortran.
- Scheduling unit-time tasks to P processors so that each job finishes between its release time and deadline.

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.



Reference: http://mathworld.wolfram.com/GameofHex.html

Goal. Algorithm to detect when a player has won.

Percolation

Percolation phase-transition.

- Two parallel conducting bars (top and bottom).
- Electricity flows from a site to one of its 4 neighbors if both are occupied by conductors.
- Suppose each site is randomly chosen to be a conductor or insulator with probability p.

top

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

bottom

insulator

Percolation

Q. What is percolation threshold p* at which charge carriers can percolate from top to bottom?

A. ~ 0.592746 for square lattices. [constant only known via simulation]

top

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

bottom

insulator

Summary

Lessons.

- 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.
- Identify fundamental abstractions: union-find.
- Apply to many domains.

might be nontrivial to analyze