CS 226 Lecture 1: Introduction

R. Sedgewick
rs@cs.princeton.edu

Intermediate-level survey course

- prerequisite: COS 126
- programming/problem solving

Algorithm: method for solving a problem
Data Structure: a way to store information

Efficient algorithms use good data structures

Why study algorithms?

Using a computer?

- want it to go faster
- want it to process more data
- want to do something that would otherwise be impossible

Technology improves things by a constant factor
...but might be costly
Good algorithm design can do much better
...and might be cheap

Supercomputer cannot rescue a bad algorithm

Algorithms as a field of study

- old enough that basics are known
- new enough that new discoveries arise
- burgeoning application areas
- philosophical implications

Analysis of algorithms

Compare algorithms by comparing estimated costs

N: size of the input
Typical running times (within constant factor)

- 1
- log N
- N
- N log N
- N^2
- 2^N

Worst Case (guarantee)
Average Case (prediction)

Other functions sometimes arise

- sqrt N
- loglog N [log(log N)]
- log* N number of logs until 1 reached

Sample problem: Online connectivity

Input:

- sequence of pairs of integers (p, q)
- p ``is connected to'' q

Output:

- nothing if p and q are already connected
- (p, q) otherwise

Assume ``is connected to'' is commutative and transitive

- if p is connected to q then q is connected to p
- if (also) q is connected to r then p is connected to r

Output lists previously unknown connections

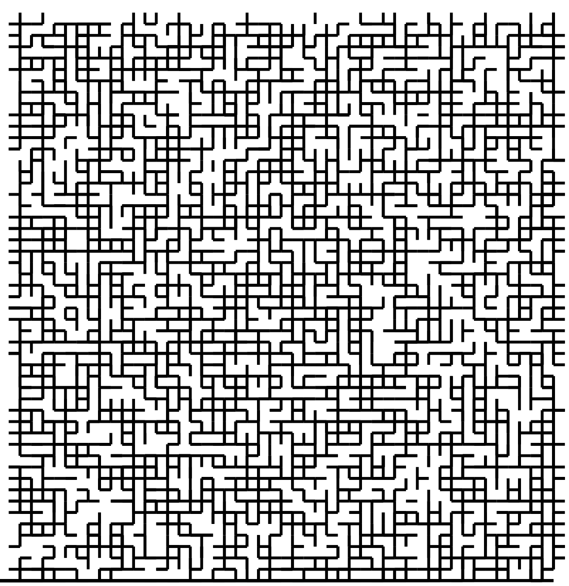
Example of application

- integers represent computers
- pairs represent network connections
- can p and q communicate through network?

Online connectivity example

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	

Network connectivity example



UNION and FIND

Disconnected piece may be hard to spot
...particularly for a computer!

Number of nodes and edges can be huge
- Internet
- computer chip

Need to design data structure and algorithms

Data structure to record connectivity information
Algorithm to use it to test connectivity (FIND)
Algorithm to update data structure (UNION)

Quick-find algorithm

Maintain array with names for components
- if i and j are connected,
- id[i] and id[j] are the same

To maintain this property for p-q connection
- ignore if id[p] = id[q]
- change all entries with p's id to q's id

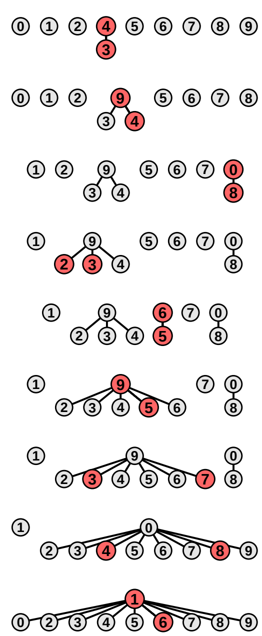
QUICK-FIND name due to constant-time test
to find out if edge makes a new connection
SLOW-UNION?

Quick-find implementation

```
main(int argc, char *argv[])
{ int i, p, q, t, N = atoi(argv[1]);
   int *id = malloc(N*sizeof(int));
   for (i = 0; i < N; i++) id[i] = i;
   while (scanf("%d %d\n", &p, &q) != EOF)
     {
        if (id[p] == id[q]) continue;
        t = id[p];
        for (i = 0; i < N; i++)
           if (id[i] == t) id[i] = id[q];
        printf(" %d %d\n", p, q);
     }
}
```


Quick-find example

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



Problem size and computation time

Rough standard for 2000
- 10^9 operations per second
- 10^9 words of memory
- touch each word in approximately 1 second
  (roughly unchanged since at least 1950)

Ex: huge problem for quick-find
- 10^10 edges connecting 10^9 nodes
  (edges need not fit in memory)
- Quick-find might take 10^20 operations
  (relabel each node (10 ops) for each edge)
- 3000 years of computer time (too much)

Quadratic running time

Quick estimate of running time
- number of edges and nodes both O(N)
- running time of quick-find O(N^2)

(10N)^2/10 = 10N^2

Gap grows as scale increases

new computer may be 10 times faster
...but has 10 times as much memory
so (with quadratic algorithm)
...takes 10 times as long to finish!

Quick-union algorithm

Maintain array with names for components
- if i and j are connected,
- (id[i])* and (id[j])* are the same
- where (id[i])* = id[id[id[...id[i]]]]
  (go until it doesn't change)

To maintain this property for p-q connection
- ignore if (id[p])* = (id[q])*
- set id[i] to j

QUICK-UNION: constant-time for new connection
SLOW-FIND?

Quick-union implementation

```
main(int argc, char *argv[])
{ int i, j, p, q, t, N = atoi(argv[1]);
   int *id = malloc(N*sizeof(int));
   for (i = 0; i < N; i++) id[i] = i;
   while (scanf("%d %d\n", &p, &q) != EOF)
     {
       i = p; j = q;
       while (i != id[i]) i = id[i];
       while (j != id[j]) j = id[j];
       if (i == j) continue;
       id[i] = j;
     }
}
```


Quick-union example

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

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

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

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

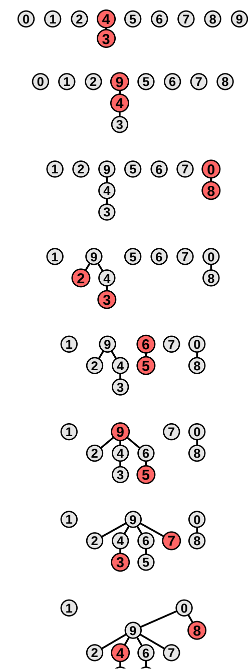
5-6
0 1 9 4 9 6 6 7 0 9

5-9
0 1 9 4 9 6 9 7 0 9

7-3
0 1 9 4 9 6 9 9 0 9

4-8
0 1 9 4 9 6 9 9 0 0

6-1
1 1 9 4 9 6 9 9 0 0



Weighted quick-union algorithm

Quick-find defect:
- UNION could be too expensive
- trees are flat, but too hard to keep them flat

Quick-union defect:
- FIND could be too expensive
- trees could get tall

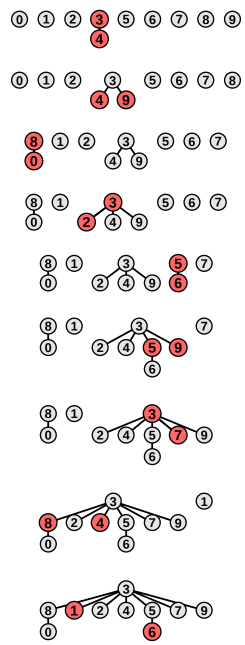
Modify quick-union to avoid tall trees
- keep track of size of each component
- balance by linking small one below large one

Weighted quick-union implementation

```
for (i = 0; i < N; i++) id[i] = i;
for (i = 0; i < N; i++) sz[i] = 1;
while (scanf("%d %d\n", &p, &q) != EOF)
  {
  for (i = p; i != id[i]; i = id[i]) ;
  for (j = q; j != id[j]; j = id[j]) ;
  if (i == j) continue;
  if (sz[i] < sz[j])
    { id[i] = j; sz[j] += sz[i]; }
  else
    { id[j] = i; sz[i] += sz[j]; }
  printf(" %d %d\n", p, q);
  }
```

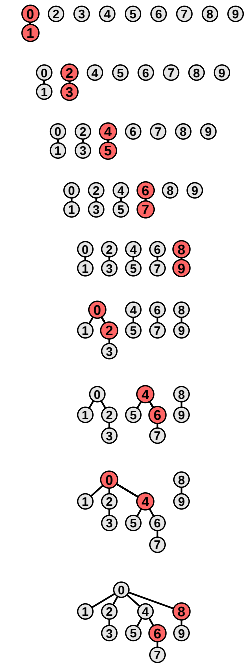

Weighted quick-union example

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



Weighted quick-union worst case

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



Weighted quick union analysis

Is performance improved?

To answer this question, need to:

- run empirical studies
- analyze the algorithm

Good news:

- Worst case is O(lg N) steps per edge

Better news:

- Average case is O(1) steps per edge

Ex: huge practical problem

- 10^10 edges connecting 10^9 nodes
- reduces time from 3000 years to 1 minute

Supercomputer wouldn't help much

Good algorithm makes solution possible

Path compression for weighted quick-union

Stop at guaranteed acceptable performance?
...not hard to improve alg further

Modify weighted quick-union to compress tree
- make every node hit point to the new root

No reason not to!

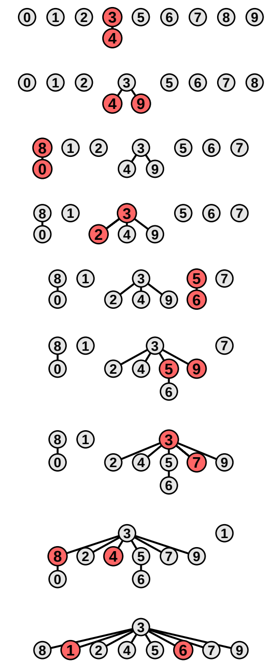
In practice, keeps trees almost completely flat
Same effect as quick-find, without the work

Path compression implementation

```
for (i = 0; i < N; i++) id[i] = i;
for (i = 0; i < N; i++) sz[i] = 1;
while (scanf("%d %d\n", &p, &q) != EOF)
{
   for (i = p; i != id[i]; i = id[i]) ;
   for (j = q; j != id[j]; j = id[j]) ;
   if (i == j) continue;
   if (sz[i] < sz[j])
      { id[i] = j; sz[j] += sz[i]; t = j; }
   else
      { id[j] = i; sz[i] += sz[j]; t = i; }
   for (i = p; i != id[i]; i = id[i])
      id[i] = t;
   for (j = q; j != id[j]; j = id[j])
      id[j] = t;
   printf(" %d %d\n", p, q);
}
```


Path compression example

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



Path compression analysis

THM: Worst-case tree height is O(lg* N)
Proof: Extremely difficult
...but the *algorithm* is still simple!

Note: lg* N is constant in this world

.	N	lg* N
.	2	1
.	4	2
.	16	3
.	65536	4
.	any practical value	5

OPTIMAL algorithm
- cost within a constant factor of cost of gathering data

theory: QFWPC is not optimal
practice: it is (in the real world)

Union-find summary

Worst-case cost per edge is proportional to

quick-find	N
quick-union	N
weighted	lg N
path compression	5

Online algorithm: can solve problem while collecting the data, for "free"

Set-merging abstraction

- FIND: is A in the same set as B?
- UNION: merge A's set and B's set

Lessons

A "trivial" algorithm can be useful
...and nontrivial to study

- start with simple algorithm
- don't use simple algorithm for large problems
- can't use simple algorithm for huge problems
- higher level of abstraction (tree) is helpful
- fast performance on real data OK, but
- strive for worst-case performance guarantees
- identify fundamental abstraction

SORTING

- Elementary algorithms, Shellsort
- Quicksort
- Mergesort
- Priority queues
- Radix sorts

Sort an array that fills memory
Make the union of M spelling dictionaries
Priority queue ADT

SEARCHING

- Tree searching
- Hashing
- Trie searching

Oxford English Dictionary
Internet search engines
DNA subsequence library
Dictionary ADT for other algorithms

STRINGS

- String searching
- Pattern matching
- File compression

file systems, audio and video

GEOMETRIC ALGORITHMS

- Elementary algorithms
- Convex hull
- Multidimensional searching

N-body simulation
World models for games and movies
CAD

GRAPH ALGORITHMS

- Properties of graphs
- Searching in graphs
- Advanced graph algorithms

Connectivity
matching (e. g. students to jobs)
networks

OTHER TOPICS

- Mathematical algorithms
- Dynamic programming
- Parallel algorithms
- Randomized algorithms
- Intractable problems

COURSE MATERIALS

Text

- Algorithms, 3rd edition, in C
  - Parts 1-4 (126 text)
  - Part 5 (graph algorithms)
- Strings and Geometry sections of old book
  - copies available after midterm

Lecture notes

- online

Online course information on homepage

READ HANDOUT ONE

READ ONLINE INFORMATION

COURSEWORK

Programming Assignments

- weekly, eleven in all
- electronic submission
  - programs due Thursdays 11:59PM
  - writeups due Fridays 4:59PM
- first one due NEXT Thursday

Problem Sets

- weekly, nine in all
- due in precept
- first one due NEXT Monday

Exams

- closed book w/ cheat sheet
- midterm in class Wednesday before break
- final at scheduled time