Reductions

Shifting Gears

- from individual problems ro problem-solving models
- from linear/quadratic to polynomial/exponential scale
- from details of implementation o conceptual framework

Introduction

> Suppose we could (couldn't) solve problem X efficiently

> What else could (couldn't) we solve efficiently?

Definition

- 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 pf reduction

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?

- Ex 1. --- finding the median reduces to sorting
  - sort N items
  - return item in the middle
  - $NlogN + 1$
- Ex 2. ---element distinctness reduces to sorting
  - sort N items
  - check adjacent pairs for equality
  - $NlogN + N$

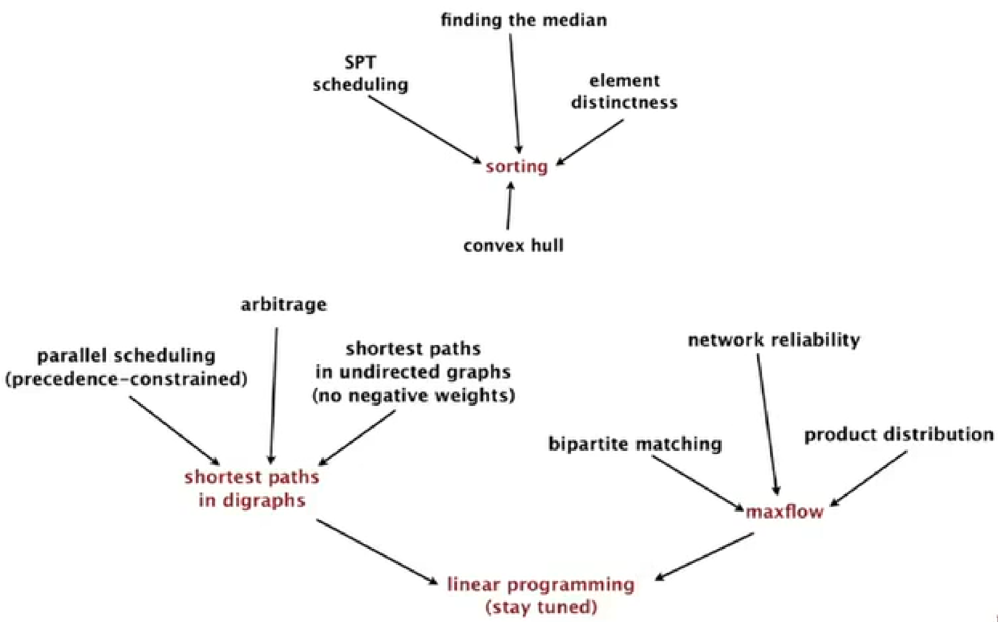
Designing Algorithms

> Given algorithm for Y, can also solve X

- Ex

- 3-collinear reduces to sorting.
- 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]
- ...

- Convex hull reduces to sorting---$NlogN + N$
- Shortest-path on edge-weighted graphs and digraphs---$ElogV + E$



Establishing Lower bounds

> Prove that a problem requires a certin number of steps

Linear-time reductions

- 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
- Prove that two problems X and Y have the same complexity
  - first, show that problem X linear-time reduces to Y
  - second, show that problem Y linear-time reduces to X
  - conclude that X and Y have the same complexity

Sorting and Convex Hull

Classifying Problems

Linear Arithmetic Reductions

- complexity by reduction

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)

- complexity in history

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

Matrix Multiplication

- complexity by reduction

problem	linear algebra	order of growth
matrix multiplication	$A \times B$	MM(N)
matrix inversion	A^{-1}	MM(N)
determinant	$\lvert A \rvert$	MM(N)
system of linear equations	$Ax = b$	MM(N)
LU decomposition	$A = LU$	MM(N)
least squares	$\min \lVert Ax - b \rVert_2$	MM(N)

- complexity in history

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	$N^{2.3727}$
?	?	$N^{2+\varepsilon}$

Complexity class

> set of problems sharing some computational property

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.
  - use exact algorithm for tractable problems
  - use heuristics for intractable problems