BBM 202 - ALGORITHMS

HACETTEPE UNIVERSITY

DEPT. OF COMPUTER ENGINEERING

ERKUT ERDEM

REDUCTIONS

May. 15, 2014

Acknowledgement: The course slides are adapted from the slides prepared by R. Sedgewick and K. Wayne of Princeton University.

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	?
⋮	⋮	⋮
exponential	c	?

Frustrating news. Huge number of problems have defied classification.

Bird's-eye view

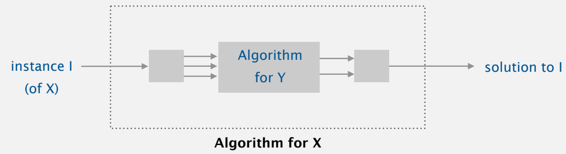
Desiderata. Classify problems according to computational requirements.

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.



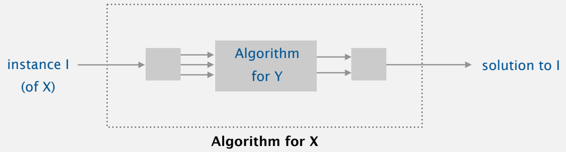
Cost of solving X = total cost of solving Y + cost of reduction.

↑ 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. [element distinctness reduces to sorting]

To solve element distinctness on N items:

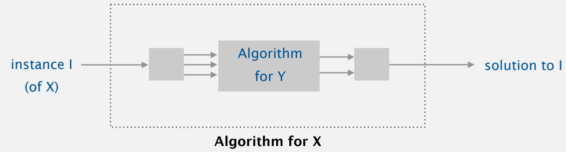
- Sort N items.
- Check adjacent pairs for equality.

Cost of solving element distinctness. $N \log N + N$.

(cost of sorting: $N \log N$; cost of reduction: N)

Reduction

Def. Problem X reduces to problem Y if you can use an algorithm that solves Y to help solve X.



Ex 2. [3-collinear reduces to sorting]

To solve 3-collinear instance on N points in the plane:

- For each point, sort other points by polar angle or slope.
  - check adjacent triples for collinearity

Cost of solving 3-collinear. $N^2 \log N + N^2$.

(cost of sorting: $N^2 \log N$; cost of reduction: N^2)

Reductions

- 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.

- Element distinctness reduces to sorting.
- 3-collinear reduces to sorting.
- CPM reduces to topological sort. [shortest paths lecture]
- h-v line intersection reduces to 1d range searching. [geometric BST lecture]
- Baseball elimination reduces to maxflow.
- Burrows-Wheeler transform reduces to suffix sort.
- …

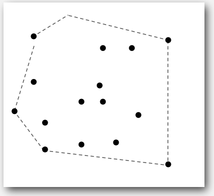
Mentality. Since I know how to solve Y, can I use that algorithm to solve X?

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
13546464
89885444
43434213
34435312

sorting

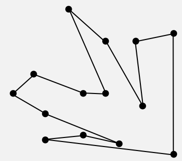
Proposition. Convex hull reduces to sorting.
Pf. Graham scan algorithm.

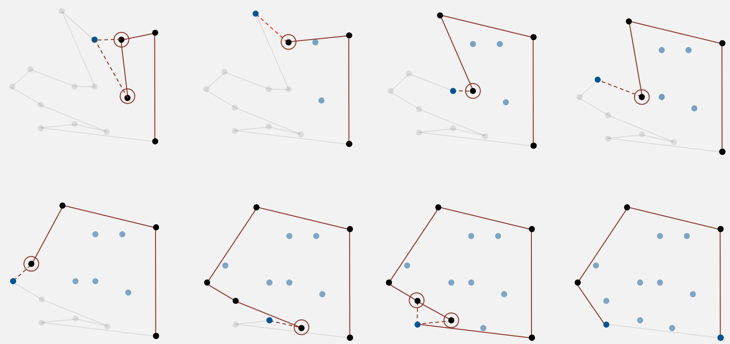
cost of sorting — cost of reduction

Cost of convex hull. $N \log N + N$.

Graham scan algorithm

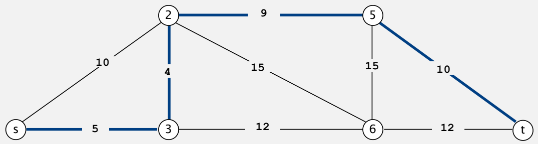
Graham scan.
- Choose point p with smallest (or largest) y-coordinate.
- Sort points by polar angle with p to get simple polygon.
- Consider points in order, and discard those that would create a clockwise turn.





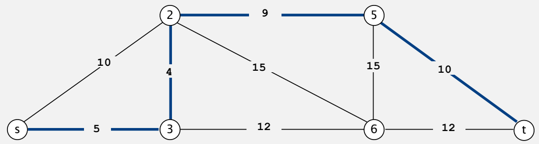
Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

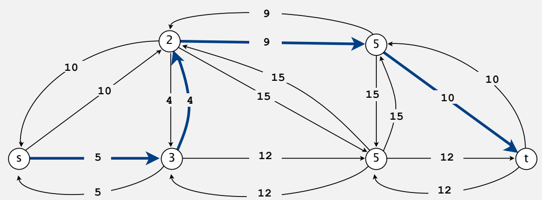


Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.

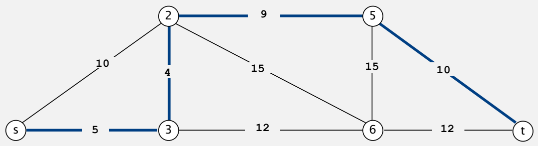


Pf. Replace each undirected edge by two directed edges.



Shortest paths on edge-weighted graphs and digraphs

Proposition. Undirected shortest paths (with nonnegative weights) reduces to directed shortest path.



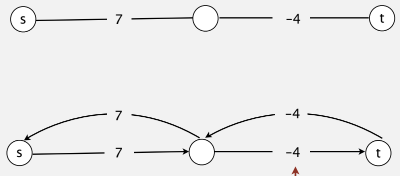
cost of shortest paths in digraph

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

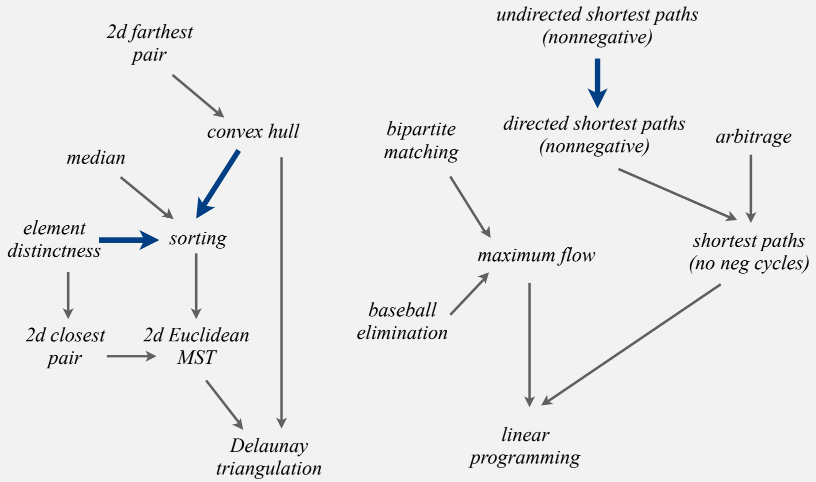
Remark. Can still solve shortest-paths problem in undirected graphs (if no negative cycles), but need more sophisticated techniques.

reduces to weighted non-bipartite matching (!)

Some reductions involving familiar problems

computational geometry

combinatorial optimization

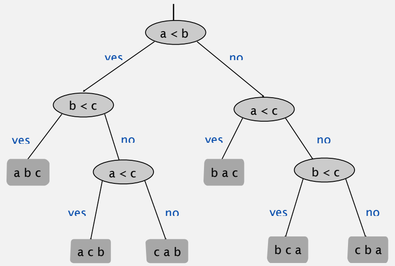


Reductions

- 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.



argument must apply to all conceivable algorithms

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.

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.

Element distinctness linear-time reduces to closest pair

Closest pair. Given N points in the plane, find the closest pair.
Element distinctness. Given N elements, are any two equal?

Proposition. Element distinctness linear-time reduces to closest pair.
Pf.

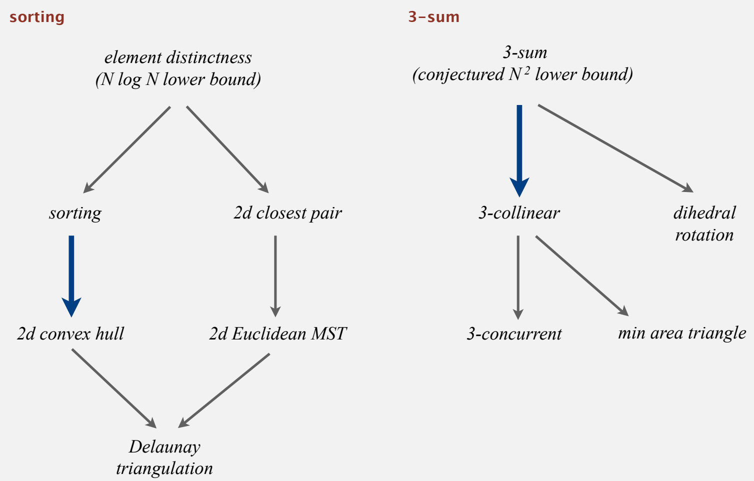
- Element distinctness instance: $x_1, x_2, \ldots, x_N$.
- Closest pair instance: $(x_1, x_1), (x_2, x_2), \ldots, (x_N, x_N)$.
- Two elements are distinct if and only if closest pair = 0.

allows quadratic tests of the form: $x_i < x_j$ or $(x_i - x_k)^2 - (x_j - x_k)^2 < 0$

Element distinctness lower bound. In quadratic decision tree model, any algorithm that solves element distinctness takes $\Omega(N \log N)$ steps.

Implication. In quadratic decision tree model, any algorithm for closest pair takes $\Omega(N \log N)$ steps.

More linear-time reductions and lower bounds



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.

REDUCTIONS

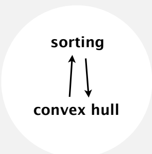
Classifying problems: summary

Desiderata. Problem with algorithm that matches lower bound.
Ex. Sorting, convex hull, and closest pair 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.

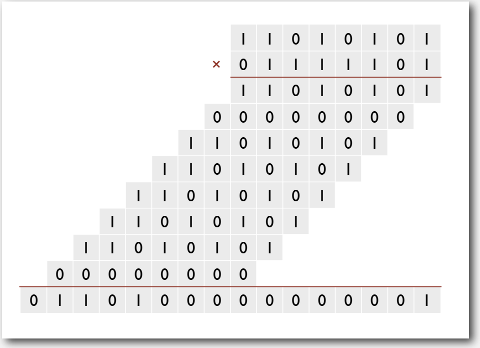
A possible real-world scenario.

- System designer specs the APIs for project.
- Alice implements `sort()` using `convexHull()`. ← well, maybe not so realistic
- Bob implements `convexHull()` using `sort()`.
- Infinite reduction loop!
- Who's fault?

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 × b	M(N)
integer division	a / b, a mod 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
1962	Karatsuba-Ofman	N
1963	Toom-3, Toom-4	N
1966	Toom-Cook	N
1971	Schönhage–Strassen	N log N log log N
2007	Fürer	N log N 2
?	?	N

number of bit operations to multiply two N-bit integers

used in Maple, Mathematica, gcc, cryptography, ...

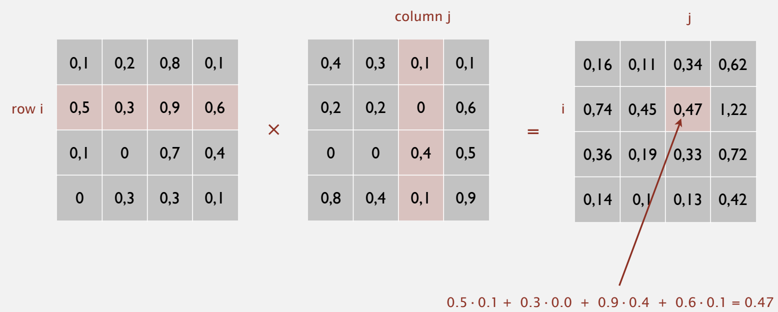
Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

GMP «Arithmetic without limitations»

Linear algebra reductions

Matrix multiplication. Given two N-by-N matrices, compute their product.

Brute force. N^3 flops.



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 × B	MM(N)
matrix inversion	A	MM(N)
determinant	|	MM(N)
system of linear equations	Ax = b	MM(N)
LU decomposition	A = L	MM(N)
least squares	min ||Ax – b||	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
1969	Strassen	N
1978	Pan	N
1979	Bini	N
1981	Schönhage	N
1982	Romani	N
1982	Coppersmith-Winograd	N
1986	Strassen	N
1989	Coppersmith-Winograd	N
2010	Strother	N
2011	Williams	N
?	?	N

number of floating-point operations to multiply two N-by-N matrices

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, closest pair, farthest pair, …
M(N)	?	integer multiplication, division, square root, …
MM(N)	?	matrix multiplication, Ax = b, least square, determinant, …
⋮	⋮	⋮
NP-complete	probably not N	3-SAT, IND-SET, ILP, …

Good news. Can put many problems into equivalence classes.

Summary

Reductions are important in theory to:

- Establish tractability.
- Establish intractability.
- 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