Reductions

Some of these lecture slides are adapted from CLRS Chapter 31.5 and Kozen Chapter 30.

Contents

Contents.

- "Linear-time reductions."
- Undirected and directed shortest path.
- Matrix inversion and multiplication.
- Integer division and multiplication.
- Sorting and convex hull.

Reduction

Intuitively, decision problem X reduces to problem Y if:

- Any instance of X can be "rephrased" as an instance of Y.
- The solution to instance of Y provides solution to instance of X.

Consequences:

- Used to establish relative difficulty between two problems.
- Given algorithm for Y, we can also solve X. (design algorithms)
- If X is hard, then so is Y. (prove intractability)

Reduction

Problem X linearly reduces to problem Y if, given a black box that solves Y in O(f(N)) time, we can devise an O(f(N)) algorithm for X.

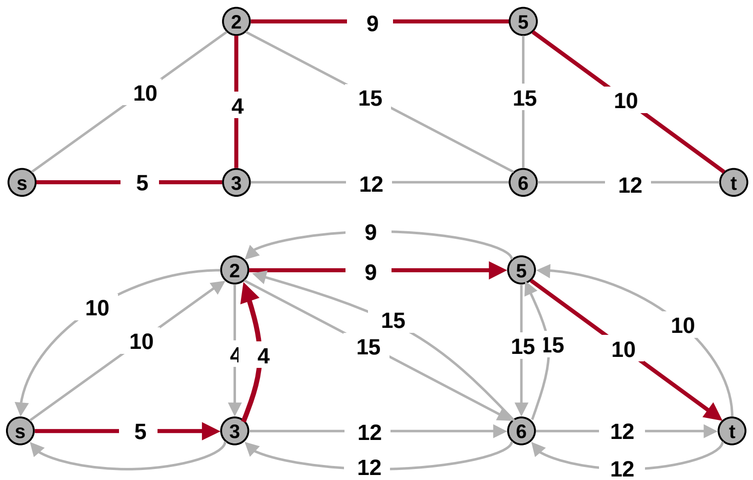
Ex 1. X = PRIME linearly reduces to Y = COMPOSITE.

- PRIME(x): Is x prime?
- COMPOSITE(x): Is x composite?
- To compute PRIME(x), call COMPOSITE(x) and return opposite answer.

Reduction: Undirected to Directed Shortest Path

Ex 2. Undirected shortest path (with nonnegative weights) linearly reduces to directed shortest path.

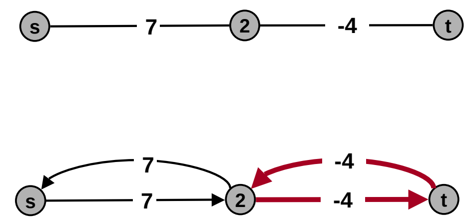
- Replace each directed arc by two undirected arcs.
- Shortest directed path will use each arc at most once.



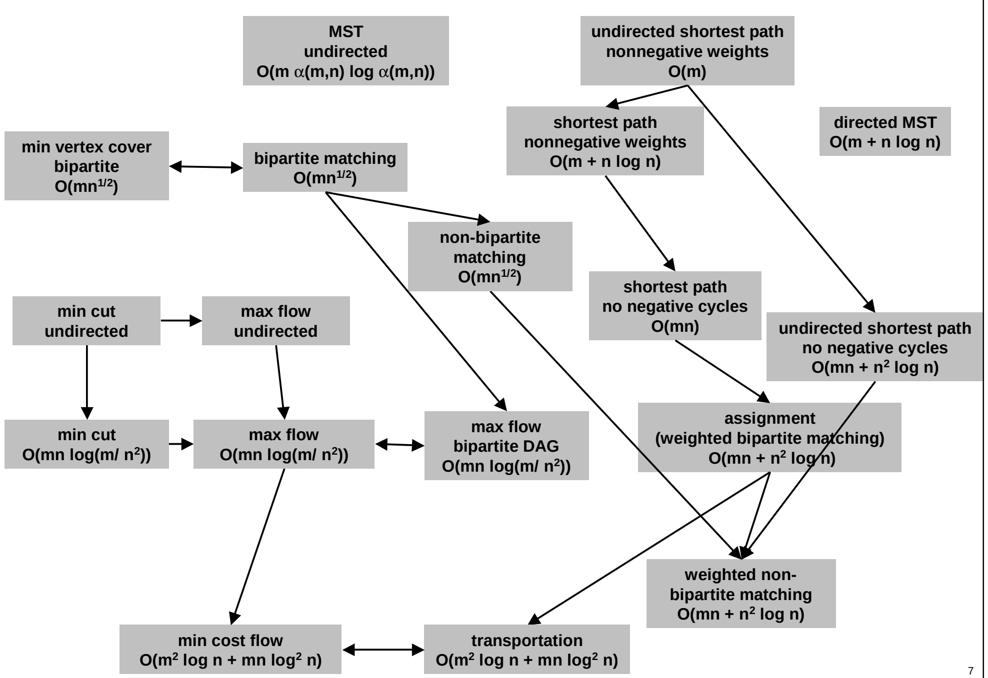
Reduction: Undirected to Directed Shortest Path

Ex 2. Undirected shortest path (with nonnegative weights) linearly reduces to directed shortest path.

- Replace each directed arc by two undirected arcs.
- Shortest directed path will use each arc at most once.
- Note: reduction invalid in networks with negative cost arcs, even if no negative cycles.



Network Flow Running Times and Linear Time Reductions



Matrix Inversion

Fundamental problem in numerical analysis.

- Intimately tied to solving system of linear equations.
- Note: avoid explicitly taking inverses in practice.

$$
\begin{array}{rcrcrcr}
1x_1 & + & 5x_2 & + & 4x_3 & = & 4 \\
2x_1 & + & 0x_2 & + & 2x_3 & = & 6 \\
5x_1 & + & 1x_2 & + & 2x_3 & = & 12
\end{array}
$$

$$\Rightarrow \quad x_1 = \frac{19}{9}, \; x_2 = -\frac{1}{3}, \; x_3 = \frac{8}{9}.$$

$$A = \begin{pmatrix} 1 & 5 & 4 \\ 2 & 0 & 2 \\ 5 & 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 4 \\ 6 \\ 12 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} -1/18 & -1/6 & 5/18 \\ 1/6 & -1/2 & 1/6 \\ 1/18 & 2/3 & -5/18 \end{pmatrix}, \quad x = A^{-1}b = \begin{pmatrix} 19/9 \\ -1/3 \\ 8/9 \end{pmatrix}$$

Matrix Multiplication vs. Matrix Inversion (CLR 31.5)

Matrix multiplication and inversion have same asymptotic complexity.

- M(N) = time to multiply to N x N matrices.
- I(N) = time to invert N x N matrix.
- Note: we don't know asymptotic complexity of either!

Proof (matrix multiplication linearly reduces to inversion).

- Regularity assumption: I(3N) = O(I(N)).
  - Holds if I(N) = N^α, since then I(3N) = $(3N)^\alpha = 3^\alpha$ I(N).
  - Holds if if I(N) = $\Theta(N^\alpha \log^\beta N)$.
- To compute C = AB, define 3N x 3N matrix D.

$$D = \begin{pmatrix} I_N & A & 0 \\ 0 & I_N & B \\ 0 & 0 & I_N \end{pmatrix} \qquad D^{-1} = \begin{pmatrix} I_N & -A & AB \\ 0 & I_N & -B \\ 0 & 0 & I_N \end{pmatrix}$$

Matrix Multiplication vs. Matrix Inversion

Proof (matrix inversion linearly reduces to multiplication).

- Regularity assumption: M(N + k) = O(M(N)) for $0 \le k < N$.
  - Holds if M(N) = $\Theta(N^\alpha \log^\beta N)$ for some $\alpha \ge 2$, $\beta \ge 0$.
- WLOG: assume N is a power of 2.
  - Pad with 0s.

$$\begin{pmatrix} A & 0 \\ 0 & I_k \end{pmatrix} = \begin{pmatrix} A^{-1} & 0 \\ 0 & I_k \end{pmatrix}$$

- WLOG: assume A is symmetric positive definite.
  - if A is invertible, then A^TA is symmetric positive definite.
  - $A^{-1} = (A^TA)^{-1} A^T$.
  - Only two extra matrix multiplications.

Matrix Multiplication vs. Matrix Inversion

Proof (matrix inversion linearly reduces to multiplication).

- To invert N x N symmetric positive definite matrix A, partition into 4 N/2 x N/2 submatrices.
  - Note: B and S (Schur complement) are symmetric positive definite since A is.

$$A = \begin{pmatrix} B & C^T \\ C & D \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^TS^{-1}CB^{-1} & -B^{-1}C^TS^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix}$$

$$S = D - CB^{-1}C^T$$

$$\begin{aligned} P_1 &= CB^{-1} &&= C \times B^{-1} \\ P_2 &= CB^{-1}C^T &&= C \times P_1 \\ S &= D - CB^{-1}C^T &&= D - P_1 \\ P_2 &= S^{-1}CB^{-1} &&= S^{-1} \times P_1 \\ P_3 &= B^{-1}C^TS^{-1}CB^{-1} &&= P_1^T \times P_2 \end{aligned}$$

Matrix Multiplication vs. Matrix Inversion

Proof (matrix inversion linearly reduces to multiplication).

- Running time.
  - 4 half-size matrix multiplications.
  - 2 half-size matrix inversions.
  - 2 half-size matrix addition, subtraction.

$$\begin{aligned} I(N) &= 2I(N/2) + 4M(N/2) + O(N^2) \\ &= 2I(N/2) + O(M(N)) \\ &= O(M(N)) \end{aligned}$$

Integer Arithmetic

Fundamental questions.

- Is integer addition easier than integer multiplication?
- Is integer multiplication easier than integer division?
- Is integer division easier than integer multiplication?

Operation	Upper Bound	Lower Bound
Addition	O(N)	Ω(N)
Multiplication	O(N log N log log N)	Ω(N)
Division	O(N log N log log N)	Ω(N)

Warmup: Squaring vs. Multiplication

Integer multiplication: given two N-digit integer s and t, compute st.

Integer squaring: given an N-digit integer s, compute s^2.

Theorem. Integer squaring and integer multiplication have the same asymptotic complexity.

Proof.

- Squaring linearly reduces to multiplication.
  - trivial: multiply s and s
- Multiplication linearly reduces to squaring.
  - regularity assumption: S(N+1) = O(S(N))

$$st = \tfrac{1}{2}((s+t)^2 - s^2 - t^2)$$

Integer Division (See Kozen, Chapter 30)

Integer division: given two integers s and t of at most N digits each, compute the quotient q and remainder r:

- $q = \lfloor s / t \rfloor$, r = s mod t.
- Alternatively, s = qt +r, $0 \le r < t$.

Example.

- s = 1000, t = 110 $\Rightarrow$ q = 9, r = 10.
- s = 4905648605986590685, t = 100 $\Rightarrow$ r = 85.

We show integer division linearly reduces to integer multiplication.

Integer Division: "Grade-School"

Divide two integers, each is N bits or less.

- $q = \lfloor s / t \rfloor$
- r = s mod t.

(q, r) = IntegerDivision (s, t)

```
IF (s < t)
   RETURN (0, t)

(q', r') ← IntegerDivision(s, 2t)

IF (r' < t)
   RETURN(2q', r')
ELSE
   RETURN (2q' + 1, r' - t)
```

Running time. O(N^2).

- O(N) per iteration + recursive calls.
- Denominator increases by factor of 2 each iteration.
  - $s < 2^N$ and does not change
  - $1 \le t \le s$ throughout
    $\Rightarrow$ O(N) recursive calls

Integer Division: "Grade-School"

The algorithm correctly compute $q = \lfloor s / t \rfloor$, $r = s \bmod t$.

Proof by reverse induction.

- Base case: $t > s$.
- Inductive step: algorithm computes q', r' such that
  - $q' = \lfloor s / 2t \rfloor$, $r' = s \bmod 2t$.
  - $s = q' (2t) + r'$, $0 \le r' < 2t$.
- Goal: show $\left\lfloor \frac{s}{t} \right\rfloor = \begin{cases} 2q' & \text{if } r' < t \\ 2q' + 1 & \text{otherwise} \end{cases}$

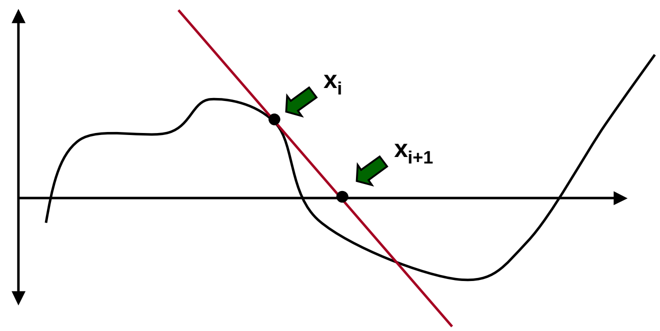
$$\left\lfloor \frac{s}{t} \right\rfloor = \left\lfloor \frac{q'(2t) + r'}{t} \right\rfloor = 2q' + \left\lfloor \frac{r'}{t} \right\rfloor$$

Newton's Method

Given a differentiable function f(x), find a value x* such that f(x*) = 0.

Newton's method.

- Start with initial guess x_0.
- Compute a sequence of approximations: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$.
- Equivalent to finding line of tangent to curve y = f(x) at x_i and taking x_{i+1} to be point where line crosses x-axis.



Newton's Method

Convergence of Newton's method.

- Not guaranteed to converge to a root x*.
- If function is well-behaved, and x_0 sufficiently close to x* then Newton's method converges quadratically.
  - number of bits of accuracy doubles at each iteration

Applications.

- Computing square roots: $f(x) = t - x^2$, $x_{i+1} = \frac{1}{2}\left(x_i + \frac{t}{x_i}\right)$
- Finding min / max of function.
  - ✎ Extends to multivariate case.
- Cornerstone problem in continuous optimization.
- Interior point methods for linear programming.

Integer Division: Newton's Method

Our application of Newton's method.

- We will use exact binary arithmetic and obtain exact solution.
- Approximately compute x = 1 / t using Newton's method.
- We'll show exact answer is either $\lfloor s\,x \rfloor$ or $\lceil s\,x \rceil$.

$$f(x) = t - \frac{1}{x}$$
$$x_{i+1} = 2x_i - t x_i^2$$

Theorem: given a O(M(N)) algorithm for multiplying two N-digit integers, there exists an O(M(N)) algorithm for dividing two integers, each of which is at most N-digits.

Integer Division: Newton's Method Example

Compute: 1 / 7.

$$f(x) = t - \frac{1}{x}$$
$$x_{i+1} = 2x_i - tx_i^2$$

- 1 — x_0 = 0.1
- 1 — x_1 = 0.13
- 2 — x_2 = 0.1417
- 4 — x_3 = 0.142847770
- 9 — x_4 = 0.14285714224218970
- 17 — x_5 = 0.1428571428571428544956854444 9737
- 34 — x_6 = 0.14285714285714285714285714285714280809023113867839307631644158170
- 67 — x_7 = 0.1428571428571428571428571428571428571428571428571428571428571428571260140220318020240406844673408393

Compute $\lfloor 123456 / 7 \rfloor$.

- 123456 * x_5 = 17636.5714285714282446193422358673107 2000
- Correct answer is either 17636 or 17637.

Integer Division: Newton's Method

(q, r) = NewtonIntegerDivision (s, t)

```
Arbitrary precision rational x.

Choose x to be unique fractional power of
2 in interval (1/2t, 1/t].

WHILE ( s – s x t ≥ t )
   x ← 2x – tx²


IF ( s - ⌊s x⌋ t < t )

   q = ⌊s x⌋

ELSE
   q = ⌈s x⌉, r = s - qt

r = s - qt
```


Analysis

L1: $\frac{1}{2t} < x_0 \le x_1 \le x_2 \le \cdots \le \frac{1}{t}.$

Proof by induction on i.

- Base case:

$$\frac{1}{2t} < x_0 \le \frac{1}{t}.$$

- Inductive hypothesis:

$$\frac{1}{2t} < x_0 \le x_1 \le \cdots \le x_i \le \frac{1}{t}.$$

$$\begin{aligned} x_{i+1} &= x_i(2 - t\,x_i) \\ &\ge x_i(2 - t\,(1/t)) \\ &= x_i \end{aligned}$$

$$\begin{aligned} x_{i+1} &= 2x_i - t\,x_i^2 \\ &= 2x_i - t\,x_i^2 - 1/t + 1/t \\ &= -t(x_i - 1/t)^2 + 1/t \\ &\le 1/t \end{aligned}$$

Analysis

L2: Sequence of Newton iterations converges quadratically to 1 / t. Iterate x_i is approximates 1 / t to 2^i significant bits of accuracy.

$$\boxed{1 - tx_i < \frac{1}{2^{2^i}}.}$$

Proof by induction on i.

- Base case: $\frac{1}{2t} < x_0$
- Inductive hypothesis: $1 - t\,x_i < \frac{1}{2^{2^i}}$

$$\begin{aligned} 1 - t\,x_{i+1} &= 1 - t\,(2x_i - t\,x_i^2) \\ &= (1 - t\,x_i)^2 \\ &< \left(\frac{1}{2^{2^i}}\right)^2 \\ &= \frac{1}{2^{2^{i+1}}} \end{aligned}$$

Analysis

L3: Algorithm terminates after O(log N) steps.

- By L2, after $k = \lceil \log_2 \log_2 (s / t) \rceil$ steps, we have: $1 - tx_k < \frac{1}{2^{2^k}} \leq \frac{t}{s}$.
  Note: $2^k = O(N)$, $k = O(\log N)$.

L4: Algorithm returns correct answer.

- By L1, $x_k \leq 1 / t$.
- Combining with proof of L3: $0 \leq \frac{s}{t} - sx_k < 1$
- This implies, $\lfloor s / t \rfloor$ is either $\lfloor s\, x_k \rfloor$ or $\lceil s\, x_k \rceil$; the remainder can be found by subtraction.

Analysis

Theorem: Newton's method does integer division in O(M(N)) time, where M(N) is the time to do multiply two N-digit integers.

- By L3, $2^k = O(N)$, and the number of iterations is O(log N).
- Each Newton iteration involves two multiplications, one addition, and one subtraction.

$$f(x) = t - \frac{1}{x}$$
$$x_{i+1} = 2x_i - tx_i^2$$

- Technical fact (not proved here): algorithm still works if we only keep track of 2^i significant digits in iteration i.
  - Bottleneck operation = multiplications.
  - $2M(1) + 2M(2) + 2M(4) + \ldots + 2M(2^k) = O(M(N))$.

Integer Arithmetic

Theorem: The following integer operations have the same asymptotic bit complexity.

- Multiplication.
- Squaring.
- Division.
- Reciprocal: N-significant bit approximation of 1/s. $\left\lfloor \frac{2^{2N-1}}{s} \right\rfloor$

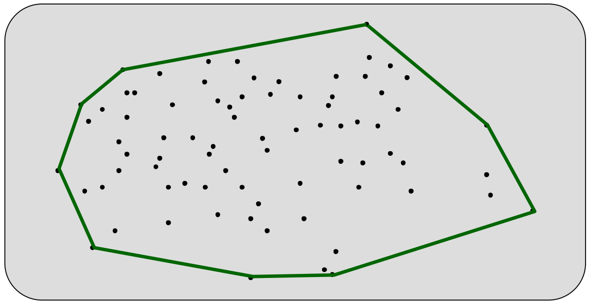
Sorting and Convex Hull

Sorting.

- Given N distinct integers, rearrange in increasing order.

Convex hull.

- Given N points in the plane, find their convex hull in counter-clockwise order.
  - Find shortest fence enclosing N points.



Sorting and Convex Hull

Sorting.
- Given N distinct integers, rearrange in increasing order.

Convex hull.
- Given N points in the plane, find their convex hull in counter-clockwise order.

Lower bounds.
- Recall, under comparison-based model of computation, sorting N items requires Ω(N log N) comparisons.
- We show sorting linearly reduces to convex hull.
- Hence, finding convex hull of N points requires Ω(N log N) comparisons.

Sorting Reduces to Convex Hull

Sorting instance:

$$x_1,\ x_2,\ \ldots,\ x_N$$

Convex hull instance.

$$(x_1,\ x_1^2),\ (x_2,\ x_2^2),\ \ldots,\ (x_N,\ x_N^2)$$

Key observation.
- Region $\{x : x^2 \geq x\}$ is convex $\Rightarrow$ all points are on hull.
- Counter-clockwise order of convex hull (starting at point with most negative x) yields items in sorted order.