6.006

Rivest

L23.1

11/25/08

Admin: (No recitations tomorrow - Thanksgiving!)

Reading: CLRS §34.3, pages 984-985

Outline: Special lecture on reductions

- ☐ intro
- ☐ VC $\le_p$ clique
- ☐ MSSP $\le_p$ SSSP
- ☐ SAT $\le_p$ VC

Reductions

6.006
Rivest
L23.2
11/25/08

"Problem" maps inputs to outputs (e.g. function or predicate)
it is

E.g. product of two matrices ← function
is graph connected?
is formula satisfiable?
predicate (T/F)

We are interested in relation <u>between</u> problems.

In particular, <u>can we solve problem A using procedure for problem B</u>?

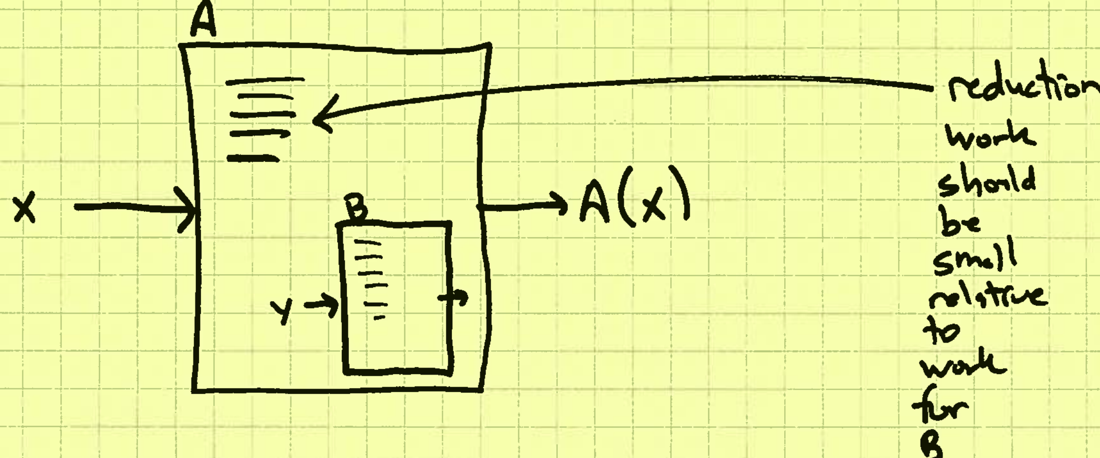
I.e. can we <u>reduce A to B</u>?

<u>Notation:</u> $A \leq B$ for such a reduction

Several different flavors of such reduction...

Suppose A maps x to $A(x)$
B maps y to $B(y)$

<u>Figure:</u>



Considerations:

- Are A & B <u>functions</u> or <u>predicates</u>?
  (works either way, just need to be clear)
- Is there a bound on reduction work (exclusive of calls to B?)
  Typically want this to be "small"...
- Is A = B? ("self-reduction" to smaller problem instances, e.g. mergesort or dyn. programming or gcd...)
  We'll ignore self-reduction today...
- Is B used once, or multiple times, for a single input to A?
- Is reduction a "mapping reduction"? That is, reduction has form

$$A(x) = B(f(x))$$

  We map x to f(x), then apply B once, and take answer from B as final output for A(x).
  Many common reductions are mapping reductions.
- <u>Notation:</u>
  - $A \le_m B$ mapping reduction
  - $A \le_p B$ polynomial time mapping reduction (aka "Karp reduction")
  - $A \le_T B$ "Turing reduction" (can call B many times)
  - $A \le_{cook} B$ "Cook reduction": polynomial time Turing reduction (can call B many times)

Motivations for studying reductions

① **Better upper bounds**

If we show $A \le B$ via an efficient reduction, and we have an efficient algorithm for B, then we have an efficient algorithm for A.

② **Lower bounds**

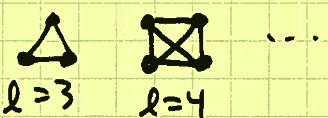
If we show $A \le B$ via an efficient reduction, and we have reason to believe A is hard, then we have reason to believe B is hard.

(Used in computational complexity theory and theory of NP-completeness.)

Example:

A = does input undirected graph G of n vertices contain a vertex cover of size k?
(does $\exists$ set of vertices of size k that touch all edges?)

B = does input graph G of n vertices contain a clique of size ℓ? (clique = ℓ mutally adjacent vertices)

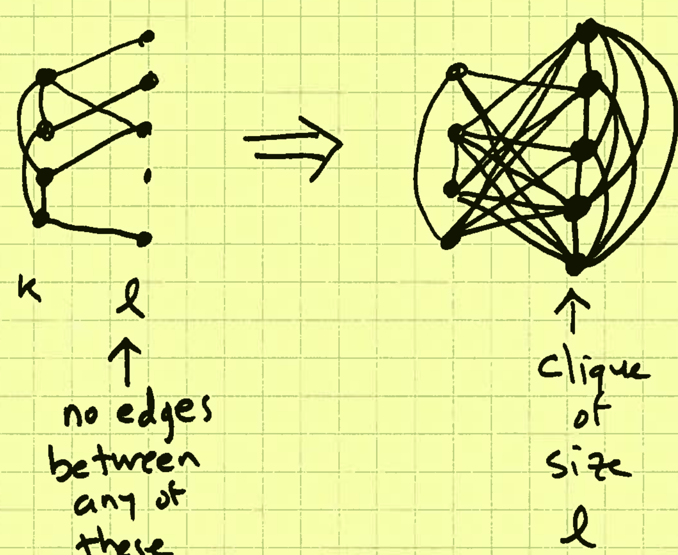


Claim: $A \leq_p B$ (poly-time mapping reduction)

Pf: Given $G = (V, E)$ and k, map to

$$G' = (V, \overline{E}) \text{ and } \ell = n - k$$

where $\overline{E}$ is complement of E.



(Can reverse to show $B \leq_p A$ as well.)

6.006
Rivest
L23.6
11/25/08

Fact: Vertex cover is "NP-hard"

Fact: If A is NP-hard & $A \leq_p B$,
then B is NP-hard.

$\therefore$ Clique problem is NP-hard (actually, NP-complete)

(NP-hard $\equiv$ at least as hard as satisfiability of Boolean formula, SAT. I.e. $SAT \leq_p A$ if A NP-hard.)

(NP-complete $\equiv$ NP-hard & solvable in nondet polynomial time...)

Fact: If $A \leq_p B$ & $B \leq_p C$, then $A \leq_p C$.

Example:

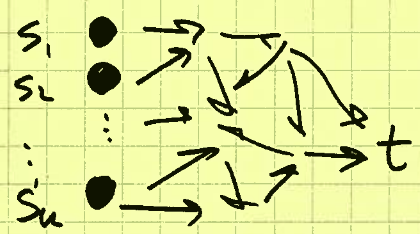
A = multisource shortest path: given $G=(V,E)$ with edge weights w & start vertices $s_1, s_2, \ldots, s_k$ & target vertex t, find shortest path from <u>some</u> start to target t.

B = std SSSP.

$A \leq_p B$

6.006
Rivest
L23.7
11/25/08

Pf:

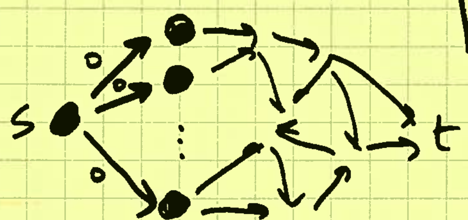


G $\left\{\begin{array}{l}\text{solve } k \\ \text{times \&} \\ \text{take min}\end{array}\right.$

$\boxed{\text{OR}}$

$\Downarrow$

$\Big[$ add new single source s, with edges to each of $s_1, \ldots, s_k$ of weight 0.



G'

shortest path in G' from $s \to t$ yields shortest $s_i \to t$ path in G.

Thus, good alg for SSSP yields good alg for MSSP (since reduction is efficient).

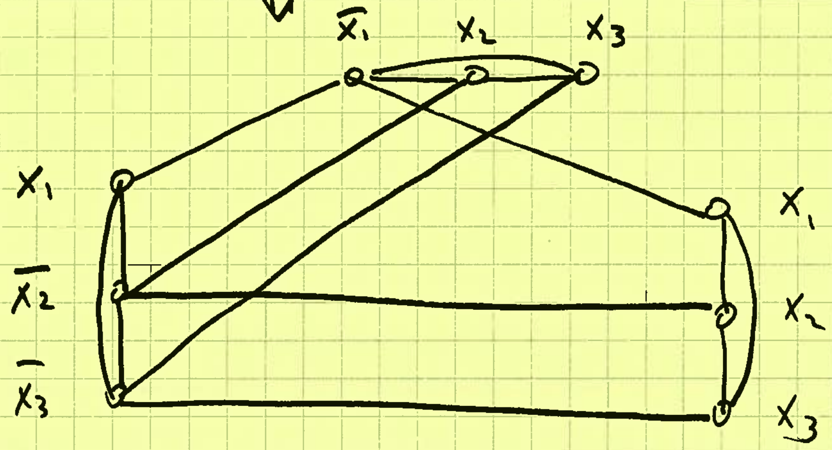
Emphasize: Reduction by itself $A \leq_p B$ doesn't solve A or B, it just relates A to B. You have to know that A is hard, or that B is easy, for this to be meaningful.

Thm SAT $\leq_p$ VC (vertex cover)

Given $\varphi = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

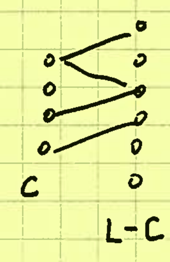
L literals
C clauses

$\Downarrow$ transform to G



draw edge if incompatible or in same clause

Look for v.c. of size L − C



$\therefore$ C vertices with no edges
("anti-clique")

$\Leftrightarrow$ must be one from each clause

$\Leftrightarrow$ satisfying assignment

∎

(This suffices to show VC is NP-hard...)