

https://algs4.cs.princeton.edu

Reductions

- *efficient algorithms*
- *computational problems*
- *linear-time reductions*
- *poly-time reductions*
- *lower bounds*

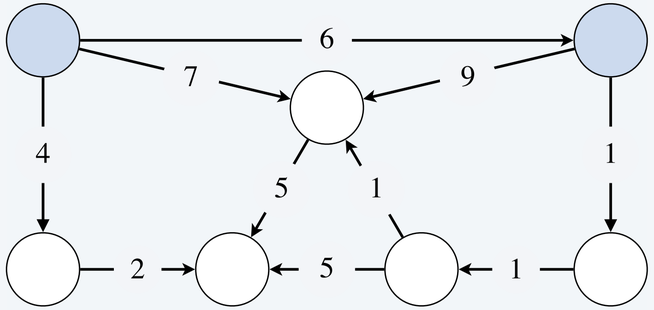
Reductions (for algorithm design)

Scenario. We worked hard to design efficient algorithm for Problem Y (and prove it correct).
Now we want to solve related, but different, Problem X.

A solution. Reduction, the following three-step process.

- Step 1: Transform input of X into input of Y.
- Step 2: Run algorithm for Y.
- Step 3: Convert solution of Y into solution of X.

Example. X: Multi-source shortest paths
Y: Single-source shortest paths



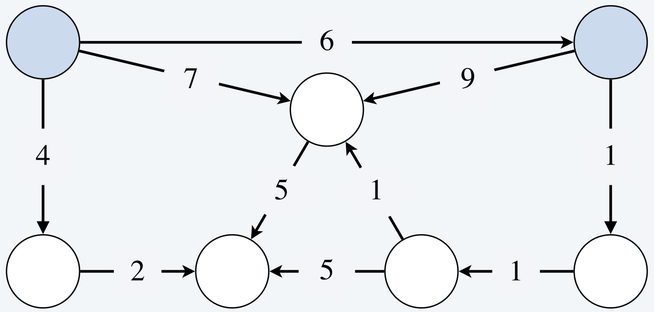
Reductions (for algorithm design)

Reduction. The following three-step process.

- Step 1: Transform input of X into input of Y.
- Step 2: Run algorithm for Y.
- Step 3: Convert solution of Y into solution of X.

Example. X: Multi-source shortest paths

Y: Single-source shortest paths



Reductions (for algorithm design)

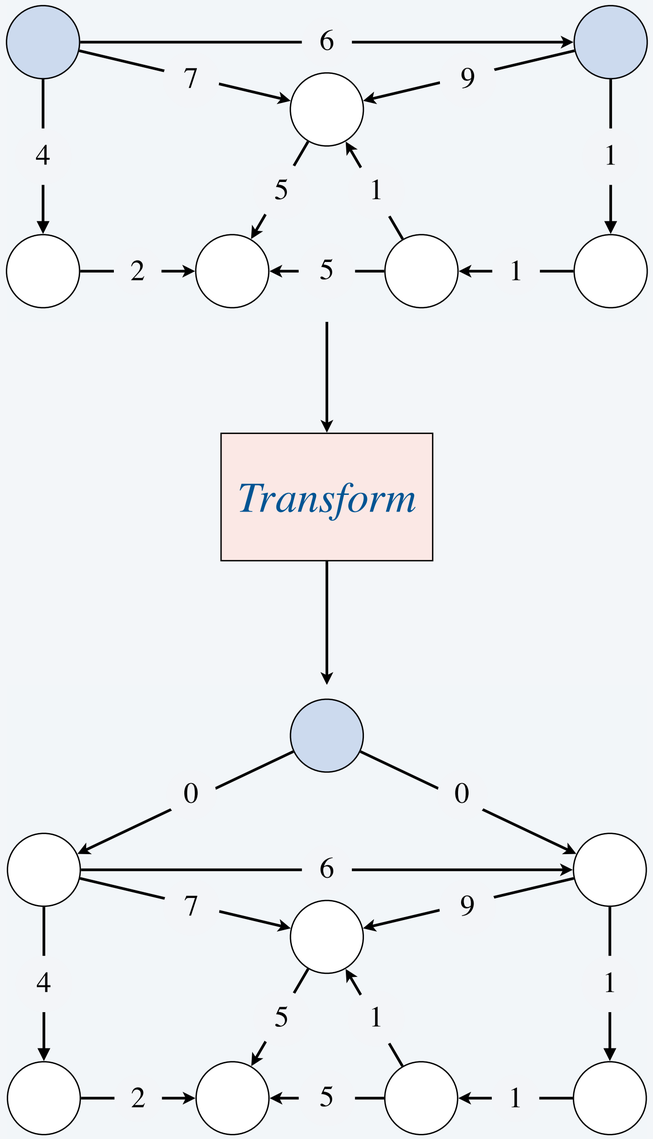
Reduction. The following three-step process.

- Step 1: Transform input of X into input of Y.
- Step 2: Run algorithm for Y.
- Step 3: Convert solution of Y into solution of X.

Example. X: Multi-source shortest paths

Y: Single-source shortest paths

- Add virtual vertex with zero-weight edges to all sources.



Reductions (for algorithm design)

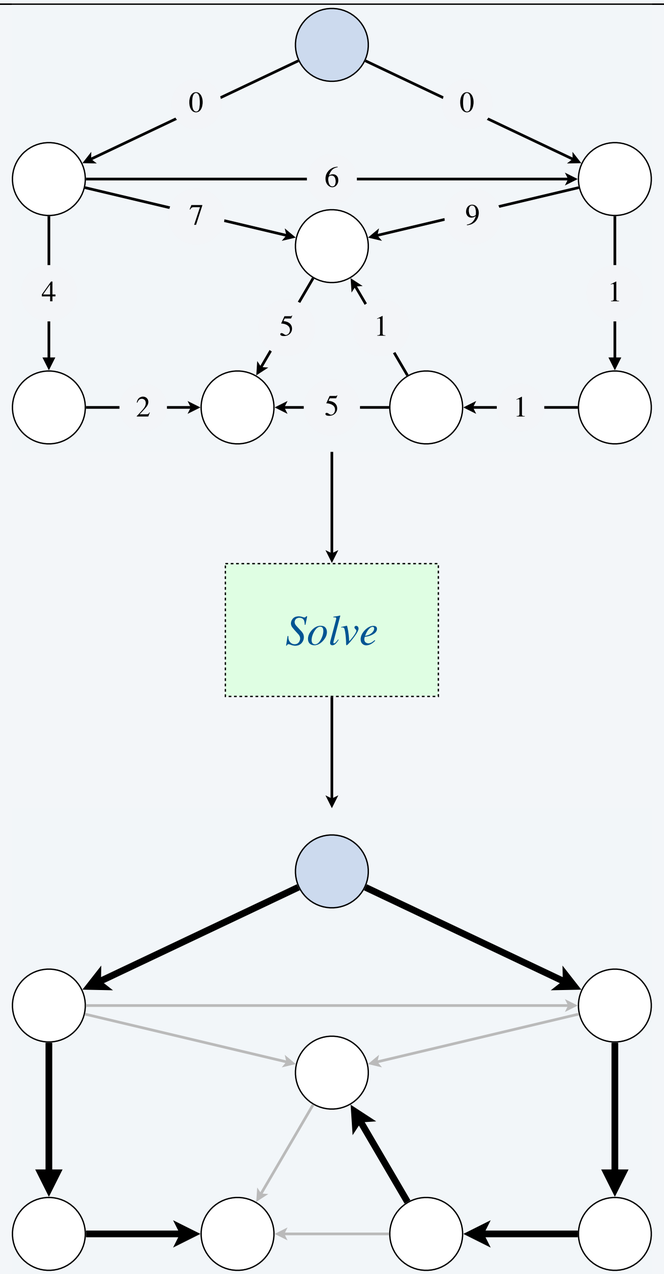
Reduction. The following three-step process.

- Step 1: Transform input of X into input of Y.
- Step 2: Run algorithm for Y.
- Step 3: Convert solution of Y into solution of X.

Example. X: Multi-source shortest paths

Y: Single-source shortest paths

- Add virtual vertex with zero-weight edges to all sources.
- Run Dijkstra's algorithm (or Bellman-Ford).



Reductions (for algorithm design)

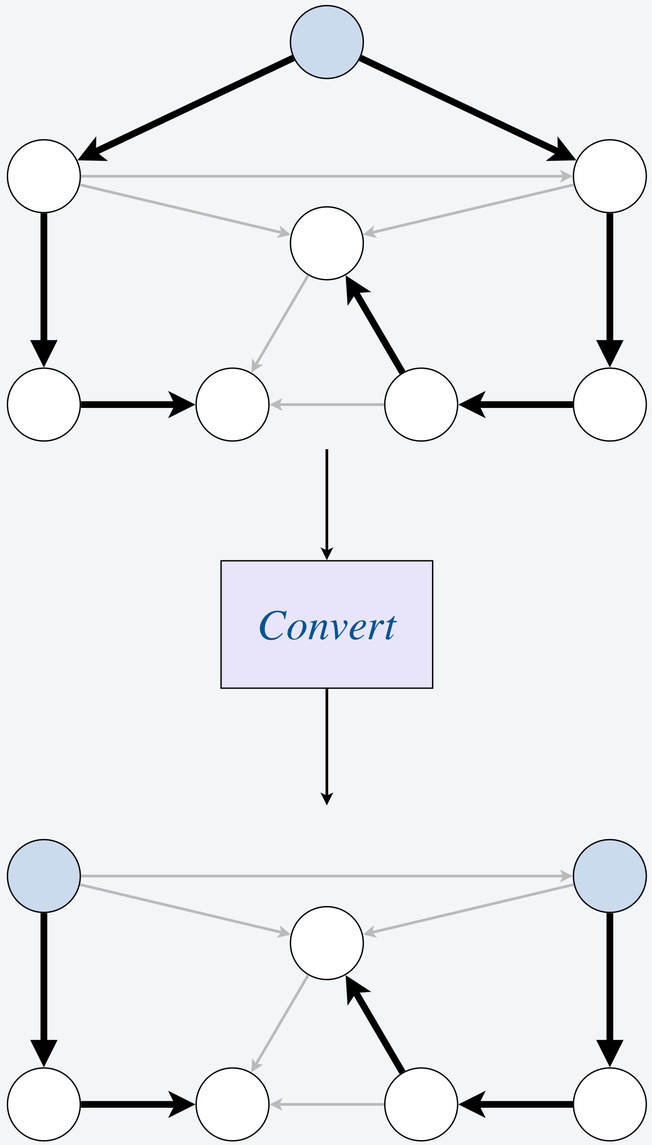
Reduction. The following three-step process.

- Step 1: Transform input of X into input of Y.
- Step 2: Run algorithm for Y.
- Step 3: Convert solution of Y into solution of X.

Example. X: Multi-source shortest paths

Y: Single-source shortest paths

- Add virtual vertex with zero-weight edges to all sources.
- Run Dijkstra's algorithm (or Bellman-Ford).
- Remove virtual source and its outgoing edges.





https://algs4.cs.princeton.edu

REDUCTIONS

- *efficient algorithms*
- *computational problems*
- *linear-time reductions*
- *poly-time reductions*
- *lower bounds*

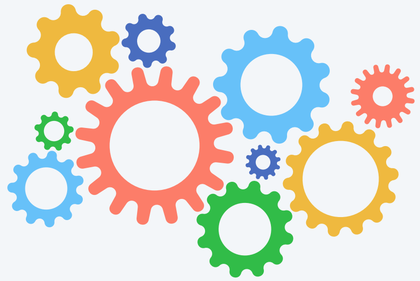
Overview: introduction to advanced topics

Main topics. [final two weeks] ⟵ *plus machine learning!*

- Reductions: solve new problems with algorithms for "old" problems.
- Intractability: barriers to designing efficient algorithms.
- Algorithm design: general paradigms for solving computational problems.

Shifting gears.

- From individual problems to problem-solving models.
- From linear/quadratic to poly-time/exponential scale.
- From implementation details to conceptual frameworks.



Goals.

- Introduce you to essential ideas.
- Place algorithms and techniques we've studied in a larger context.

Guiding questions

Q1. What is an algorithm?

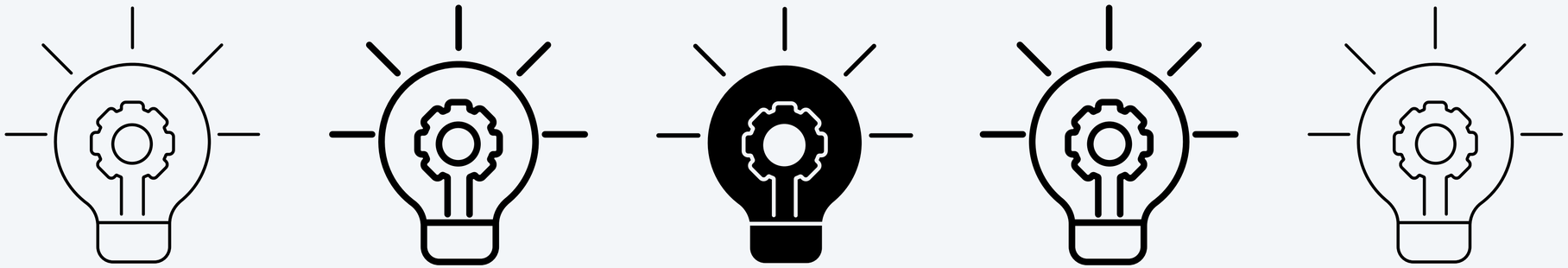
Q2. What is an efficient algorithm?

Q3. Which problems are tractable (efficiently solvable)?

Q4. Which problems are intractable?

Q5. How can we cope with intractable problems?

Q6. How can we benefit from intractability?



Guiding questions

Q1. What is an algorithm?

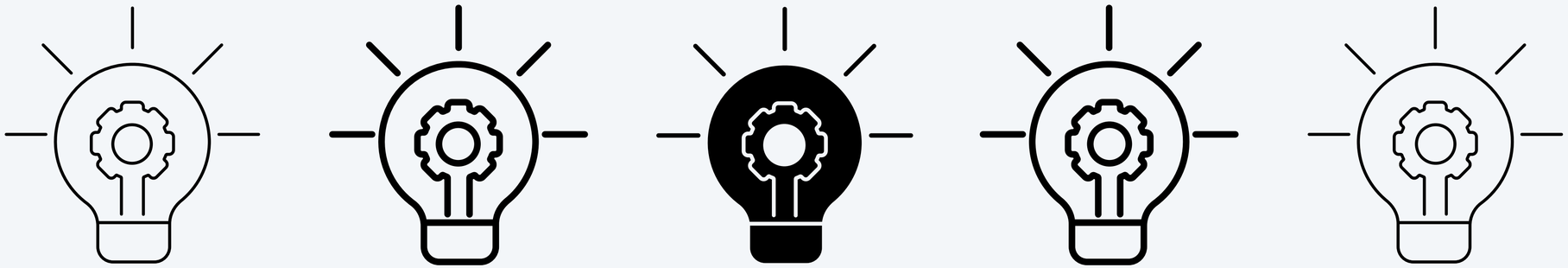
Q2. What is an efficient algorithm?

Q3. Which problems are tractable (efficiently solvable)?

Q4. Which problems are intractable?

Q5. How can we cope with intractable problems?

Q6. How can we benefit from intractability?



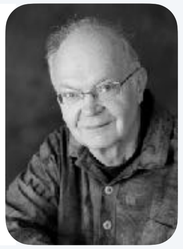
Algorithms

Q1. What is an algorithm?

Informal definition. [Knuth]

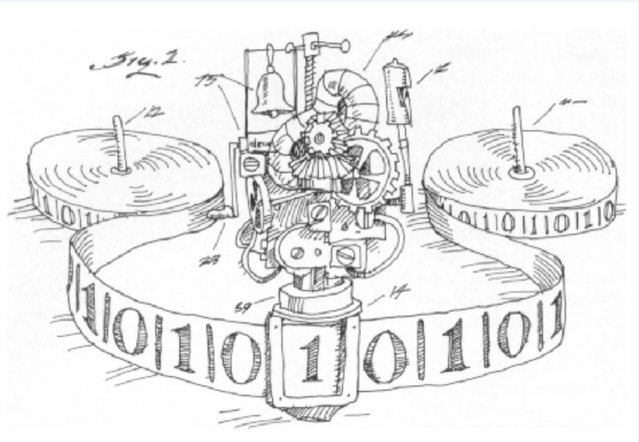
- Finite: always halts after a finite number of steps.
- Definite: each step is precisely specified (no ambiguity).
- Effective: each step is mechanically executable (no "magic").
- Procedure: a step-by-step process.

> *" An algorithm is a finite, definite, effective procedure, with some input and some output. "*
>
> *— Donald Knuth*



Formal definition. An algorithm is modeled as a Turing (or RAM) machine.

In practice: Think of an algorithm as a program in Java or Python.


Fig. 1
1 0 1 1 0 1 1 0 1

Efficient algorithms

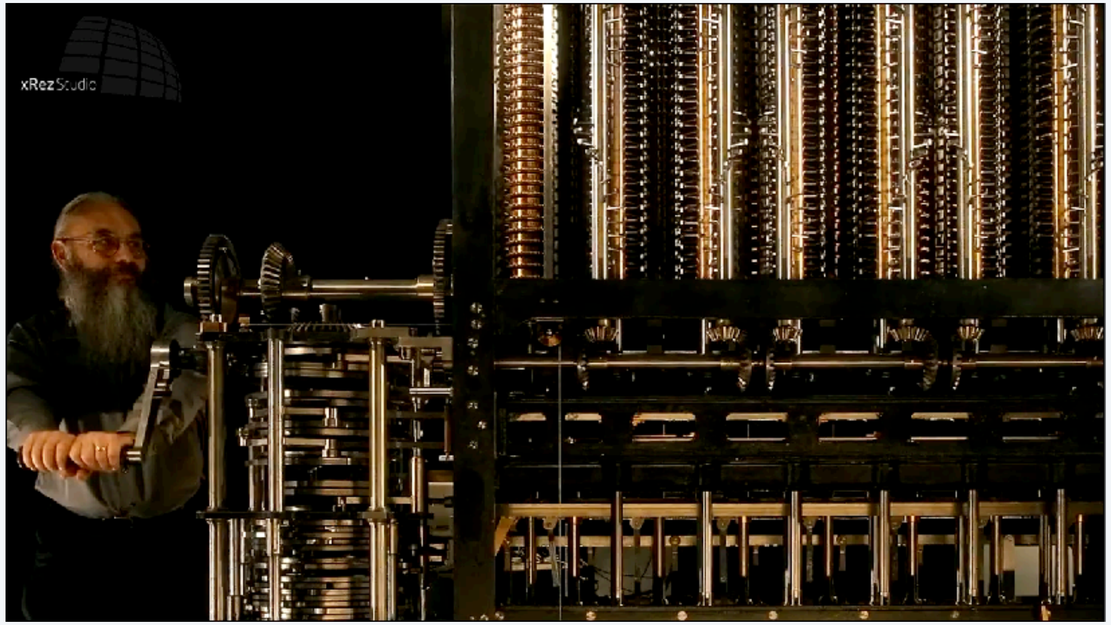
Q2. What is an efficient algorithm?

A2. One that runs in polynomial time.

Poly-time algorithm. An algorithm whose worst-case running time on inputs of size n is $T(n) \leq c\, n^k$ for some positive constants c and k.

n = # of bits in input

Equivalent. Worst-case $T(n) = O(n^k)$ with $k > 0$ constant.



https://vimeo.com/49080293

Why polynomial time? [stay tuned for next lecture!]



https://algs4.cs.princeton.edu

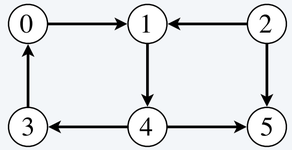
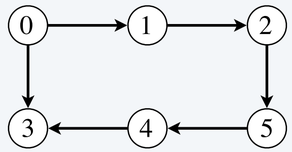
REDUCTIONS

- *efficient algorithms*
- *computational problems*
- *linear-time reductions*
- *poly-time reductions*
- *lower bounds*

Computational problems

Computational problem. A problem that specifies

- a set of valid inputs (instances), and
- for each valid input, the set of acceptable outputs (solutions).

problem type	output format	example problem	an instance	a solution
decision	*YES or NO*	Given a digraph G, is there a path that visits every vertex exactly once?		*NO*
search	*a feasible solution* (*or none*)	Given a digraph G, find a path that visits every vertex exactly once (or report none exists).		0→1→2→5→4→3

decision and search versions of the Hamiltonian path problem

Remark. Search generalizes decision as well as other natural problem types. ← *optimization (find path with most edges), function (compute its length)…*

Decision problems

Decision problem. A computational problem whose solution to an instance x is either *YES* or *NO*.

Ex. 3-SUM: Given a length-n integer array a[] with distinct elements, are there $i < j < k$ such that a[i] + a[j] + a[k] == 0 ?



an algorithm for 3-SUM

instance	solution
[-1, 1, 3, 4]	*NO*
[-1, 0, 1, 3]	*YES*
[-2, 0, 2]	*YES*
[-3, 1, 2]	*YES*
[-3, 0, 1]	*NO*
[-3, 2, 3]	*NO*
[-5, 2, 3]	*YES*
⋮	⋮

Solving a decision problem. An algorithm that, on every instance x, outputs the correct *YES* or *NO* answer.

Search problems

Search problem. A computational problem whose solutions to an instance x are a set S_x satisfying specified conditions.

Ex. 3-SUM: Given a length-n integer array a[] with distinct elements, find $i < j < k$ such that a[i] + a[j] + a[k] == 0 (or report none exist).

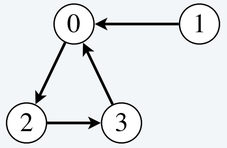
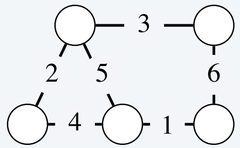
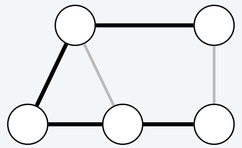
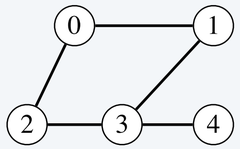


an algorithm for 3-SUM

instance	solution
[-1, 1, 3, 4]	*NONE*
[-1, 0, 1, 3]	*(0, 1, 2)*
[-2, 0, 2]	*(0, 1, 2)*
[-3, 1, 2]	*(0, 1, 2)*
[-3, 0, 1]	*NONE*
[-3, 2, 3]	*NONE*
[-5, 2, 3]	*(0, 1, 2)*
⋮	⋮

Solving a search problem. An algorithm that, on every instance x, outputs a solution in S_x if one exists, and *NONE* otherwise.

Some search problems

problem	description	an instance	a solution	efficient algorithm	runtime
3-SUM (*triple that sums to* 0)	Given an array a of integers, find a triple of entries that sums to 0.	`[-5, 1, 2, 3]`	$(0, 2, 3)$	*triple nested loop*	$\Theta(n^3)$ $O(n^{1.9992})$ *recently proved!*
DIRECTED-CYCLE (*cyclic graph*)	Given a digraph G, find a directed cycle.		$0 \to 2 \to 3$	*DFS*	$\Theta(n)$
MST (*minimum spanning tree*)	Given a connected weighted graph G, find a spanning tree of minimal weight.			*Kruskal/Prim*	$n = \Theta(E + V)$ $\Theta(n \log n)$
BIPARTITE (*bipartite graph*)	Find a bipartition of a graph G.		$\{0, 3\}$	*BFS/DFS*	$\Theta(n)$
FACTOR (*integer factorization*)	Given an n-bit integer x, find a nontrivial divisor d.	$x = 143$	$d = 11$		



https://algs4.cs.princeton.edu

REDUCTIONS

- *efficient algorithms*
- *computational problems*
- *linear-time reductions*
- *poly-time reductions*
- *lower bounds*

Bird's-eye view

Challenge. Difficult to design efficient algorithms.

Design strategy. Suppose we can solve problem Y efficiently. What else can we solve efficiently?

Design tool. Reductions.

> *" Give me a lever long enough and a fulcrum on which to place it, and I shall move the world."* — *Archimedes*

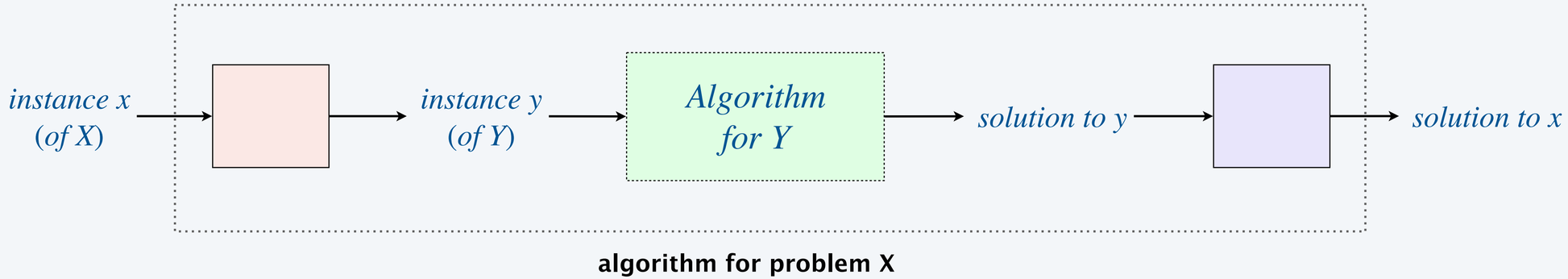


Linear-time reductions

Def. Problem X linear-time reduces to problem Y if there is an algorithm for X that:

- transforms any instance of X into an instance of Y in linear time;
- calls an algorithm for Y as a subroutine on this new instance;
- transforms the resulting solution for Y into a solution for X in linear time.

Levin reduction: one call to subroutine for Y



Algorithm design. An $O(n^k)$-time algorithm for Y yields an $O(n^k)$-time algorithm for X, for $k \geq 1$.

Reductions: poll 1



Suppose that Problem X linear-time reduces to Problem Y. Which must be true?

A. If X can be solved in $\Theta(n^3)$ time, then so can Y.

B. If Y can be solved in $\Theta(n^3)$ time, then so can X.

C. Both A and B.

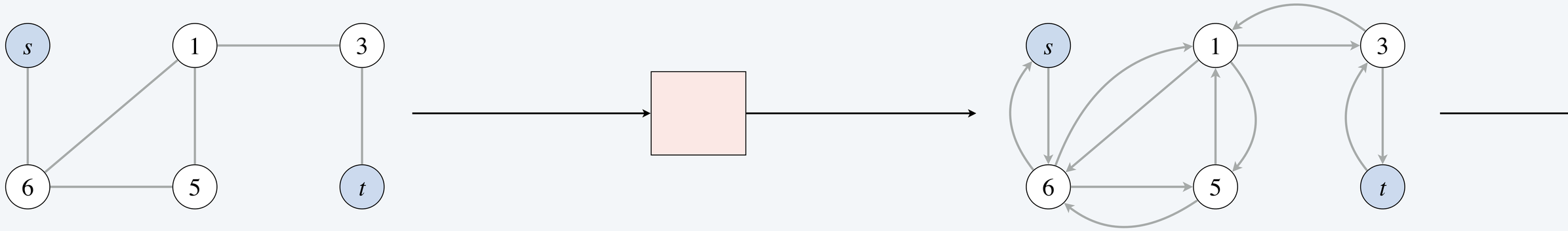
D. Neither A nor B.

UNDIRECTED-ST-PATH linear-time reduces to DIRECTED-ST-PATH

U-ST-PATH: Given a graph G, and two vertices s and t, find an s–t path (or report none).

ST-PATH: Given a digraph G, and two vertices s and t, find an $s \rightsquigarrow t$ path (or report none).

Example 1. U-ST-PATH linear-time reduces to ST-PATH.



instance of U-ST-PATH

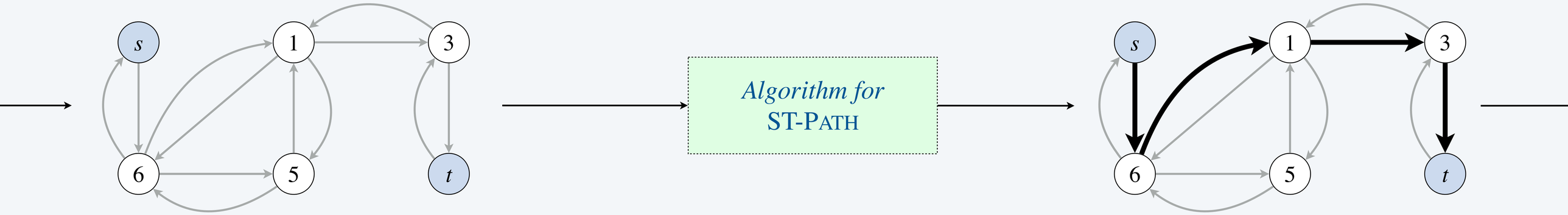
instance of ST-PATH

UNDIRECTED-ST-PATH linear-time reduces to DIRECTED-ST-PATH

U-ST-PATH: Given a graph G, and two vertices s and t, find an s–t path (or report none).

ST-PATH: Given a digraph G, and two vertices s and t, find an $s \rightsquigarrow t$ path (or report none).

Example 1. U-ST-PATH linear-time reduces to ST-PATH.



instance of ST-PATH

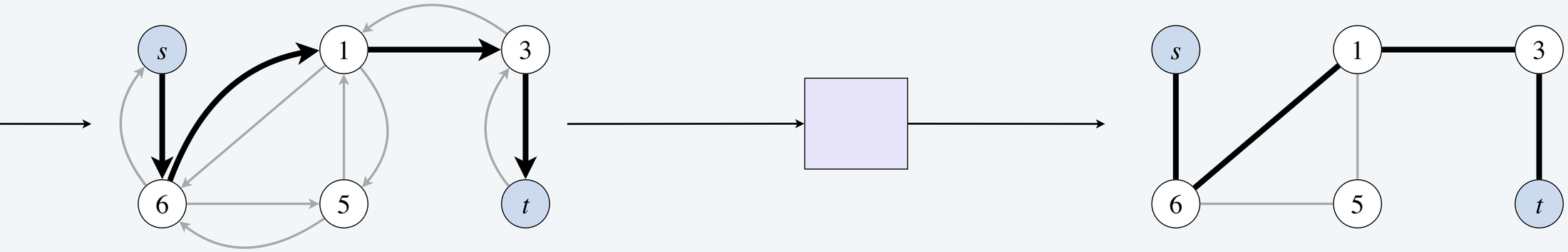
solution of ST-PATH

UNDIRECTED-ST-PATH linear-time reduces to DIRECTED-ST-PATH

U-ST-PATH: Given a graph G, and two vertices s and t, find an s–t path (or report none).

ST-PATH: Given a digraph G, and two vertices s and t, find an $s \rightsquigarrow t$ path (or report none).

Example 1. U-ST-PATH linear-time reduces to ST-PATH.



solution of ST-PATH

solution of U-ST-PATH

Linear-time reductions via antiparallel edges

Remark. Many undirected graph problems linear-time reduce to their directed counterparts (by replacing each undirected edge $v-w$ with two antiparallel edges $v \to w$ and $w \to v$). Digraph G' has $V' = V$ vertices and $E' = 2E$ edges.

Problem X		Problem Y
U-ST-PATH (*path between s and t*)	**linear-time reduces to**	ST-PATH (*directed path from s to t*)
U-ST-SHORTEST-PATH (*shortest path between s and t*)	**linear-time reduces to**	ST-SHORTEST-PATH (*shortest directed path from s to t*)
U-HAM-PATH (*Hamiltonian path*)	**linear-time reduces to**	HAM-PATH (*Hamiltonian directed path*)
⋮	⋮	⋮

Reductions: poll 2



UNDIRECTED-CYCLE does not linear-time reduce to DIRECTED-CYCLE via antiparallel edges.

What error(s) does the attempt make?

A. A cyclic undirected G is mapped to an acyclic digraph G'.

B. An acyclic undirected G is mapped to a cyclic digraph G'.

C. The mapping from G to G' is not linear-time.

D. All of the above.

Linear-time reductions speedrun



Problem 1. Solve MULTI-SOURCE-SHORTEST-PATHS via linear-time reduction to SSSP (single-source).

- Step 1: Add virtual vertex s to G with zero-weight edges to all sources, obtaining G'.
- Step 2: Run SSSP algorithm on G' from s, obtaining shortest-path tree T'.
- Step 3: Remove s and its edges from T', obtaining T; output T.

Problem 2. Solve MEDIAN via linear-time reduction to SORTING.

- Step 1: Do nothing: sorting input is the same array `a[]`.
- Step 2: Run SORTING algorithm on `a[]`, obtaining sorted array `s[]`.
- Step 3: Output `s[s.length / 2]`.

Problem 3. Solve SINGLE-DESTINATION-SHORTEST-PATHS via linear-time reduction to SSSP (single-source).

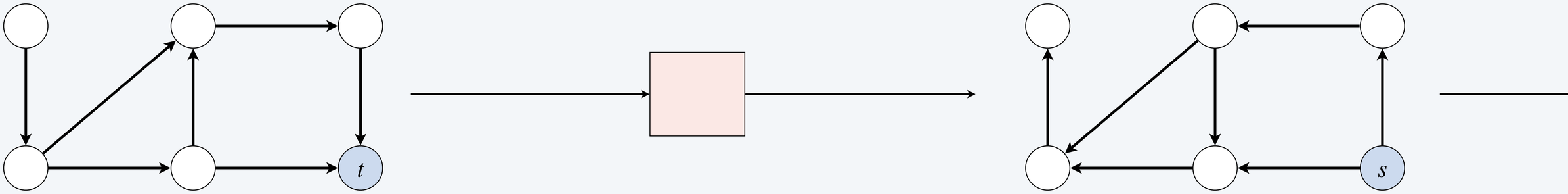
- Step 1: Reverse edges of G to obtain G'.
- Step 2: Run SSSP algorithm on G' from destination t, obtaining shortest-path tree T'.
- Step 3: Reverse edges of T', obtaining T; output T.

SINGLE-DESTINATION-SHORTEST-PATHS linear-time reduces to SINGLE-SOURCE-SHORTEST-PATHS

SDSP: Given a digraph G and a target t, find the shortest $v \rightsquigarrow t$ paths for all v that reach t.

SSSP: Given a digraph G and a source s, find the shortest $s \rightsquigarrow v$ paths for all v that s reaches.

Example 2. SDSP linear-time reduces to SSSP.



instance of SDSP

instance of SSSP

SINGLE-DESTINATION-SHORTEST-PATHS linear-time reduces to SINGLE-SOURCE-SHORTEST-PATHS

SDSP: Given a digraph G and a target t, find the shortest $v \rightsquigarrow t$ paths for all v that reach t.

SSSP: Given a digraph G and a source s, find the shortest $s \rightsquigarrow v$ paths for all v that s reaches.

Example 2. SDSP linear-time reduces to SSSP.



instance of SSSP

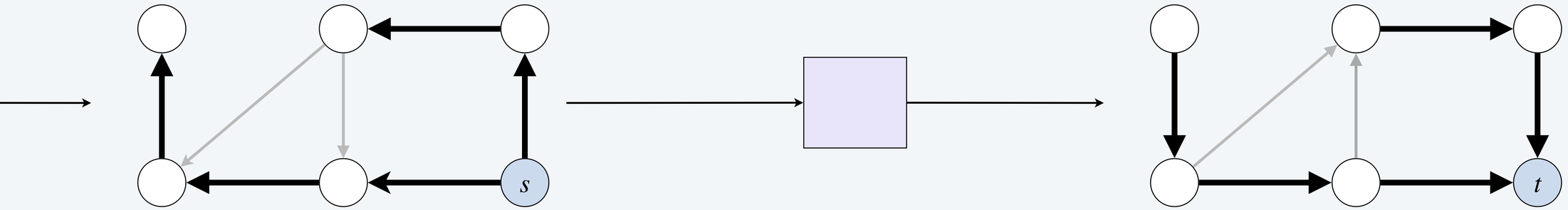
solution of SSSP

SINGLE-DESTINATION-SHORTEST-PATHS linear-time reduces to SINGLE-SOURCE-SHORTEST-PATHS

SDSP: Given a digraph G and a target t, find the shortest $v \rightsquigarrow t$ paths for all v that reach t.

SSSP: Given a digraph G and a source s, find the shortest $s \rightsquigarrow v$ paths for all v that s reaches.

Example 2. SDSP linear-time reduces to SSSP.



solution of SSSP

solution of SDSP

Linear-time reductions speedrun



Problem 4. Solve MAXIMUM-SPANNING-TREE via linear-time reduction to MST (MINIMUM-SPANNING-TREE).

- Step 1: Negate edge weights of G to obtain G'.
- Step 2: Run MST algorithm on G', obtaining tree T'.
- Step 3: Negate weights of T', obtaining T; output T.

Problem 5. Solve SHORTEST-ST-TELEPORT-PATH via linear-time reduction to SSSP.

- Step 1: Construct G' as two copies of G with zero-weight $v_1 \to w_2$ edges for every $v \to w$ in G.
- Step 2: Run SSSP algorithm on G' from s_1, obtaining $s_1 \rightsquigarrow t_2$ path P'.
- Step 3: For each $v_1 \to w_1$, $v_2 \to w_2$ and $v_1 \to w_2$ edge in P', add $v \to w$ to P; output P.

Problem 6. Solve DORMS-AND-ROUTERS via linear-time reduction to MST.

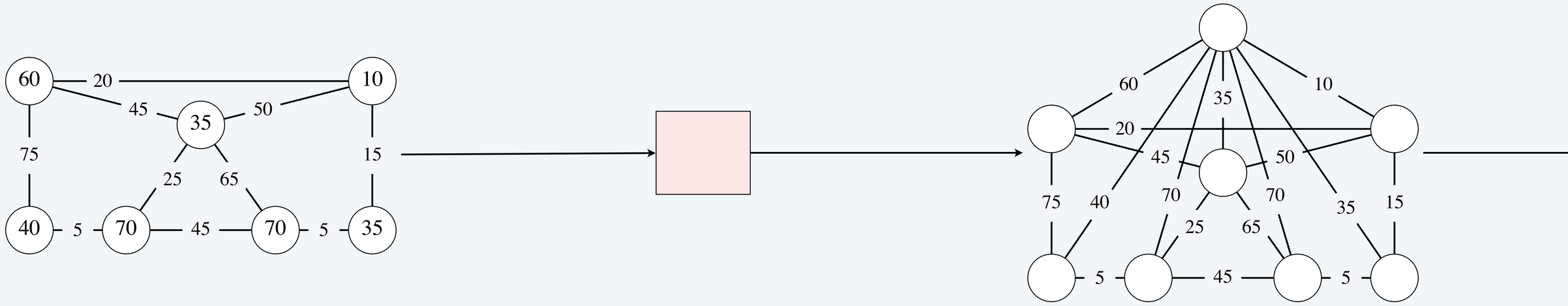
- Step 1: Add virtual vertex s to G with an edge $s-v$ with the same weight as v, obtaining G'.
- Step 2: Run MST algorithm on G' obtaining tree T'.
- Step 3: For each $s-v$ in T', place router in v; for every $u-v$ in T', connect dorms u and v.

DORMS-AND-ROUTERS linear-time reduces to MST

DORMS-AND-ROUTERS: Given a connected graph G with both vertex and edge weights, find a minimum-weight set of vertices R and edges C such that every vertex is connected to R via edges in C.

MST: Given a connected graph G with edge weights, find a minimum-weight spanning tree T.

Example 3. DORMS-AND-ROUTERS linear-time reduces to MST.



instance of DORMS-AND-ROUTERS

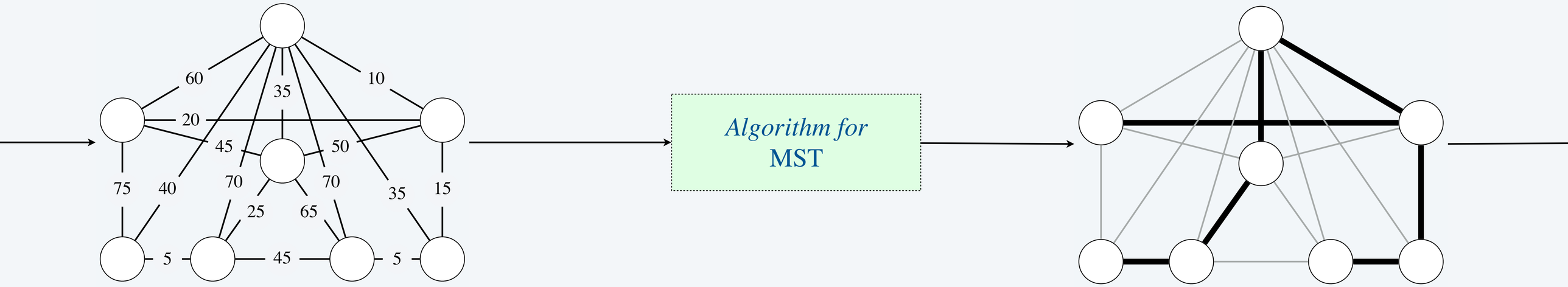
instance of MST

DORMS-AND-ROUTERS linear-time reduces to MST

DORMS-AND-ROUTERS: Given a connected graph G with both vertex and edge weights, find a minimum-weight set of vertices R and edges C such that every vertex is connected to R via edges in C.

MST: Given a connected graph G with edge weights, find a minimum-weight spanning tree T.

Example 3. DORMS-AND-ROUTERS linear-time reduces to MST.



instance of MST

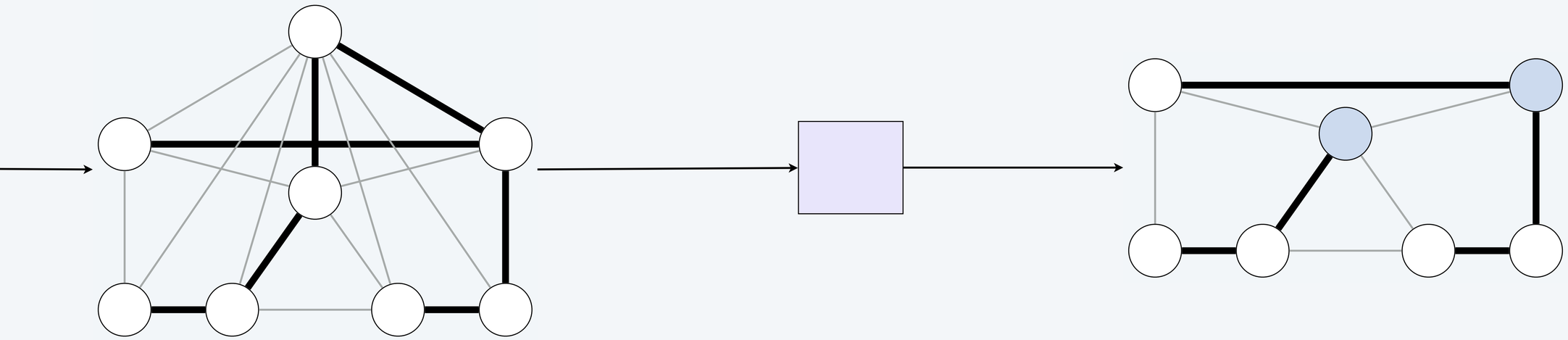
solution of MST

DORMS-AND-ROUTERS linear-time reduces to MST

DORMS-AND-ROUTERS: Given a connected graph G with both vertex and edge weights, find a minimum-weight set of vertices R and edges C such that every vertex is connected to R via edges in C.

MST: Given a connected graph G with edge weights, find a minimum-weight spanning tree T.

Example 3. DORMS-AND-ROUTERS linear-time reduces to MST.



solution of MST

solution of DORMS-AND-ROUTERS

Recap

Reductions. A familiar technique, now with a name!

- Step 1: Transform input x of X into input y of Y (in linear time).
- Step 2: Run algorithm for Y on y, getting solution s_y.
- Step 3: Convert s_y into solution s_x of x (in linear time).

Problem X		Problem Y
2-SUM (*opposite pairs*)	**linear-time reduces to**	SORTING (*sort an array*)
PERCOLATION (*full site in last row?*)	**linear-time reduces to**	U-ST-PATH (*s-t connectivity*)
HOUSE-PAINTING (*maximum profit set of houses*)	**linear-time reduces to**	LONGEST-PATH-DAG (*longest path in a DAG*)
⋮	⋮	⋮



https://algs4.cs.princeton.edu

REDUCTIONS

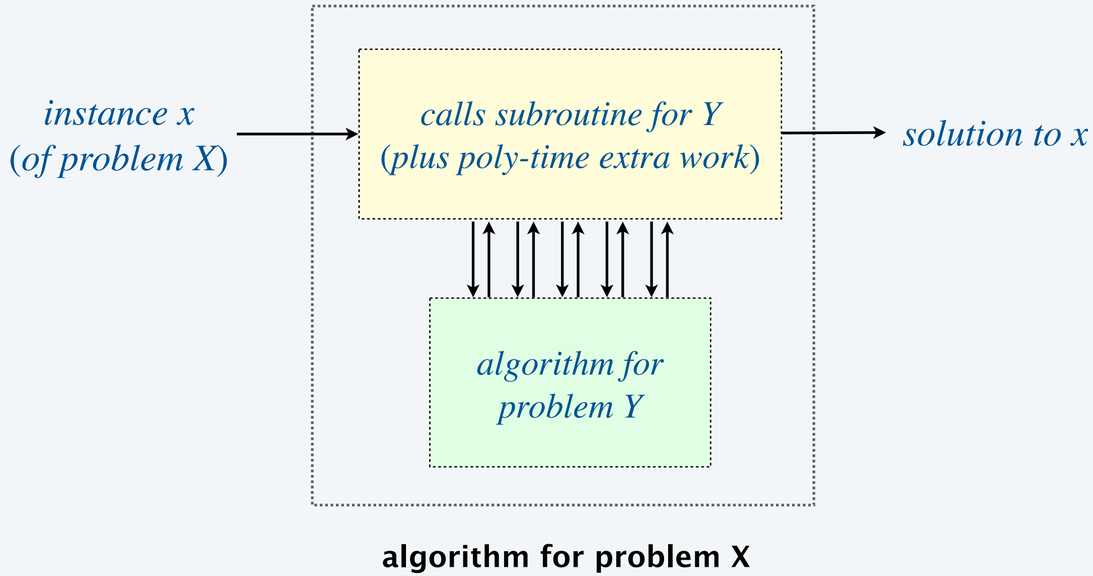
- *efficient algorithms*
- *computational problems*
- *linear-time reductions*
- *poly-time reductions*
- *lower bounds*

Polynomial-time reductions

Def. Problem X poly-time reduces to problem Y if there is an algorithm for X that:

- makes a polynomial number of calls to an algorithm for Y as a subroutine, and
- performs at most a polynomial amount of additional work outside those calls.

← *Cook reduction* ($X \preceq Y$)



Algorithm design. An efficient (poly-time) algorithm for Y yields an efficient algorithm for X.

Remark. Generalizes linear-time: linear vs. poly extra work; single vs. poly many calls.

Reductions: poll 3



Suppose that Problem X poly-time reduces to Problem Y. Which must be true?

A. If Y can be solved in poly-time, then so can X.

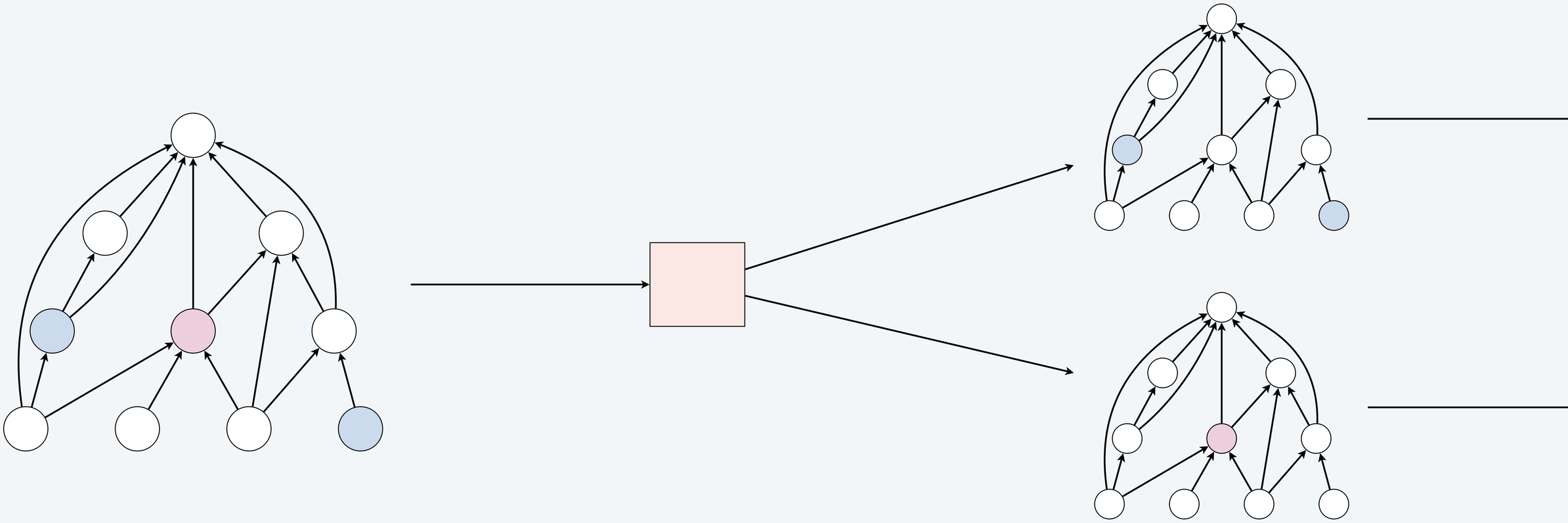
B. If Y can be solved in $\Theta(n^3)$ time, then so can X.

C. Both A and B.

D. Neither A nor B.

SHORTEST-ANCESTRAL-PATH poly-time reduces to MULTI-SOURCE-MIN-CARDINALITY-PATHS

Example 4. SHORTEST-ANCESTRAL-PATH poly–time reduces to MULTI-SOURCE-MIN-CARDINALITY-PATHS (MSMCP).

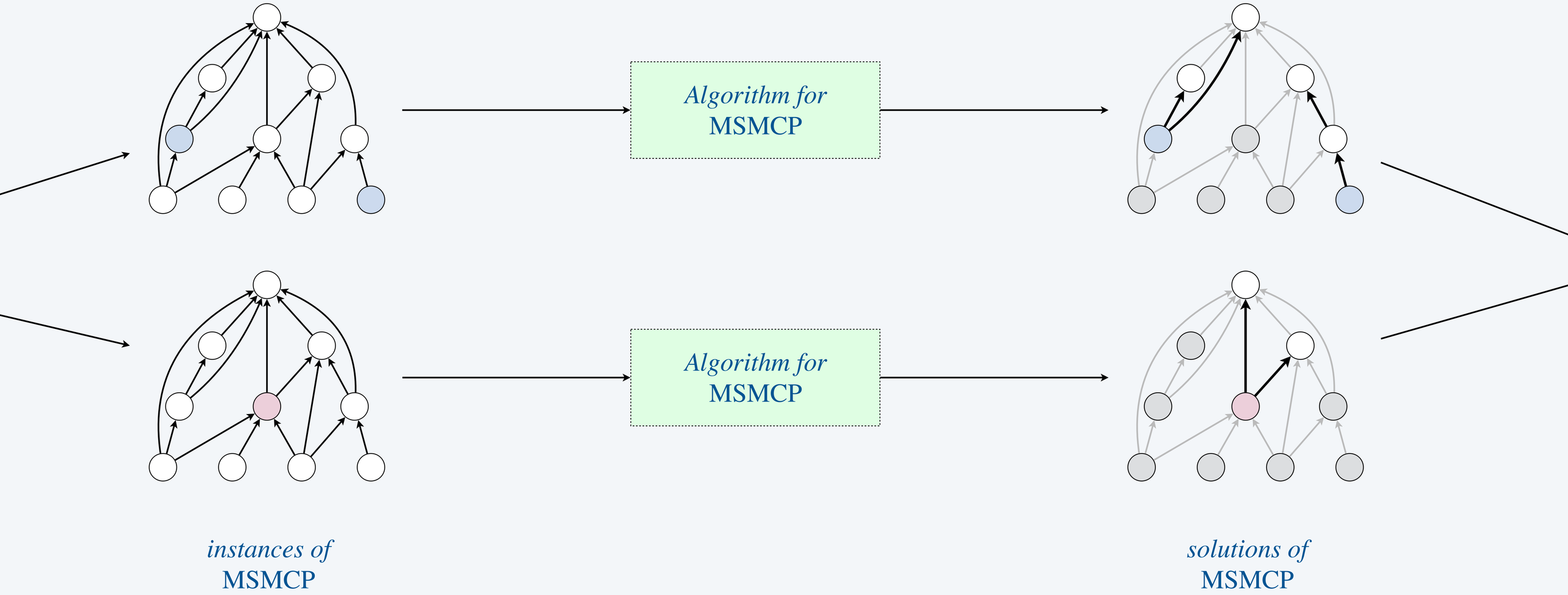


instance of SHORTEST-ANCESTRAL-PATH

instances of MSMCP

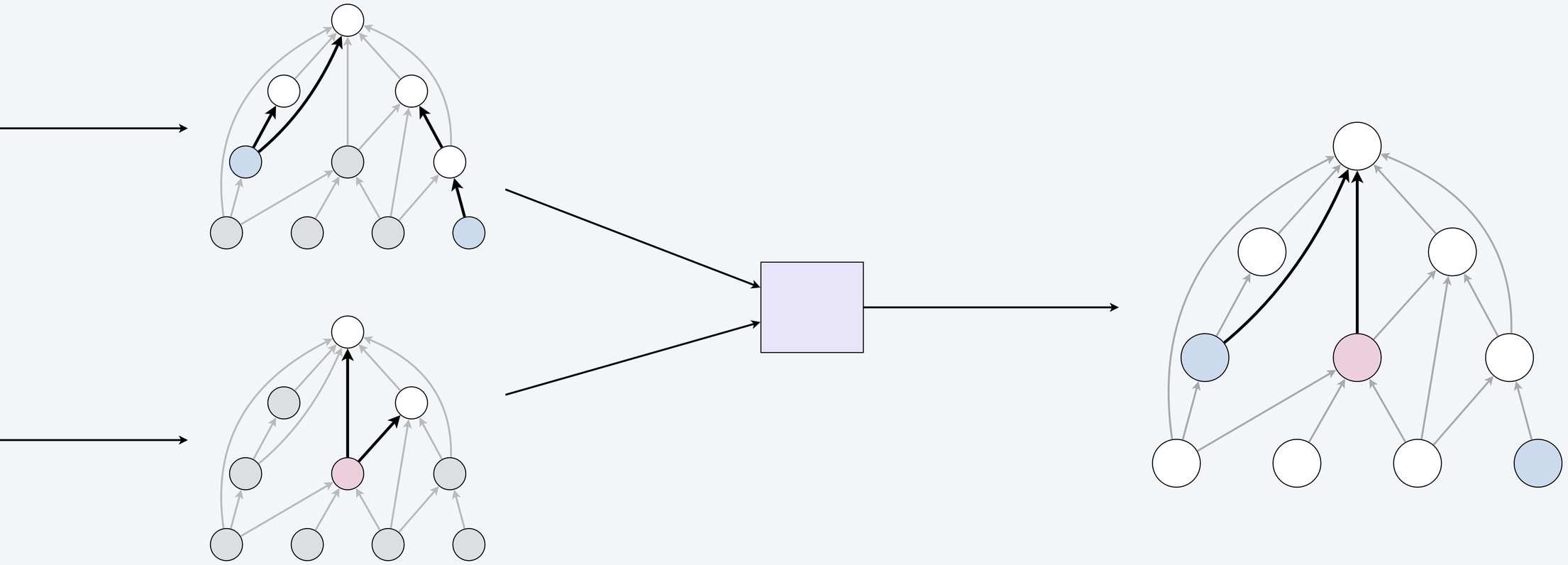
SHORTEST-ANCESTRAL-PATH poly-time reduces to MULTI-SOURCE-MIN-CARDINALITY-PATHS

Example 4. SHORTEST-ANCESTRAL-PATH poly–time reduces to MULTI-SOURCE-MIN-CARDINALITY-PATHS (MSMCP).



SHORTEST-ANCESTRAL-PATH poly-time reduces to MULTI-SOURCE-MIN-CARDINALITY-PATHS

Example 4. SHORTEST-ANCESTRAL-PATH poly–time reduces to MULTI-SOURCE-MIN-CARDINALITY-PATHS (MSMCP).



solutions of MSMCP

solution of SHORTEST-ANCESTRAL-PATH



https://algs4.cs.princeton.edu

REDUCTIONS

- *efficient algorithms*
- *computational problems*
- *linear-time reductions*
- *poly-time reductions*
- *lower bounds*

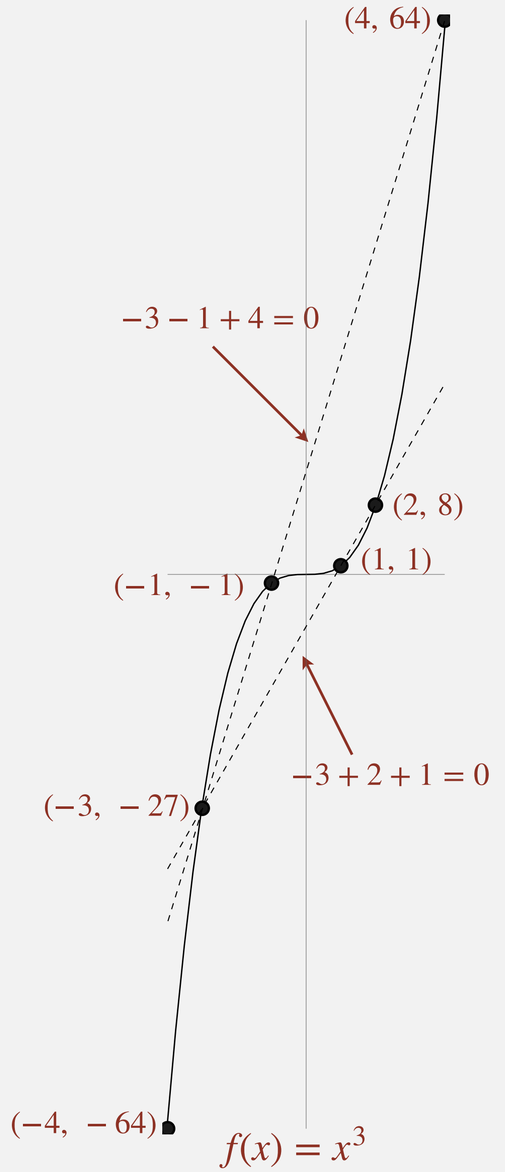
3-SUM linear-time reduces to 3-COLLINEAR

3-SUM: Given distinct integers $x_1, x_2, \ldots, x_n$, are there $i < j < k$ with $x_i + x_j + x_k = 0$?

3-COLLINEAR: Given distinct 2D points $p_1, p_2, \ldots, p_n$, with integer coordinates, are there $i < j < k$ with p_i, p_j, p_k on the same line?

Example 5. 3-SUM linear-time reduces to 3-COLLINEAR. ⟵ *both search and decision versions*

- 3-SUM instance: $x_1, x_2, \ldots, x_n$.
- 3-COLLINEAR instance: $(x_1, x_1^3), (x_2, x_2^3), \ldots, (x_n, x_n^3)$.



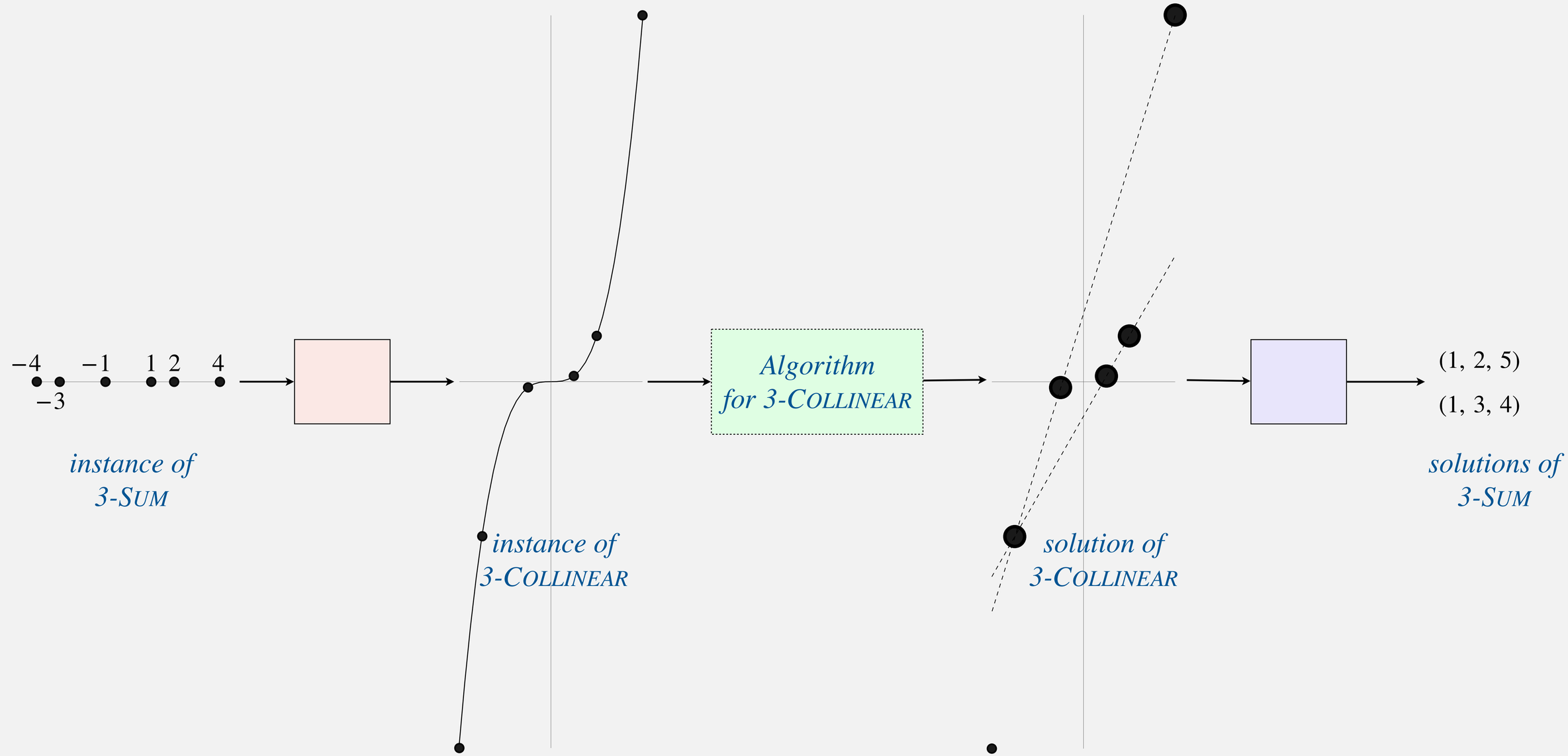
Lemma. If a, b, and c are distinct, then $a + b + c = 0$ if and only if (a, a^3), (b, b^3), and (c, c^3) are collinear.

Pf. Slopes of lines $(a, a^3) \to (b, b^3)$ and $(a, a^3) \to (c, c^3)$ equal iff

$$\frac{b^3 - a^3}{b - a} = \frac{c^3 - a^3}{c - a} \iff (b - c)(a + b + c) = 0 \quad \blacksquare$$

3-SUM linear-time reduces to 3-COLLINEAR

Example 5. 3-SUM linear-time reduces to 3-COLLINEAR.



Reductions: poll 4



Which of the following options must be false?

A. There exists a $\Theta(n^{1.8})$–time algorithm for 3-COLLINEAR but no $\Theta(n^{1.8})$–time algorithm for 3-SUM.

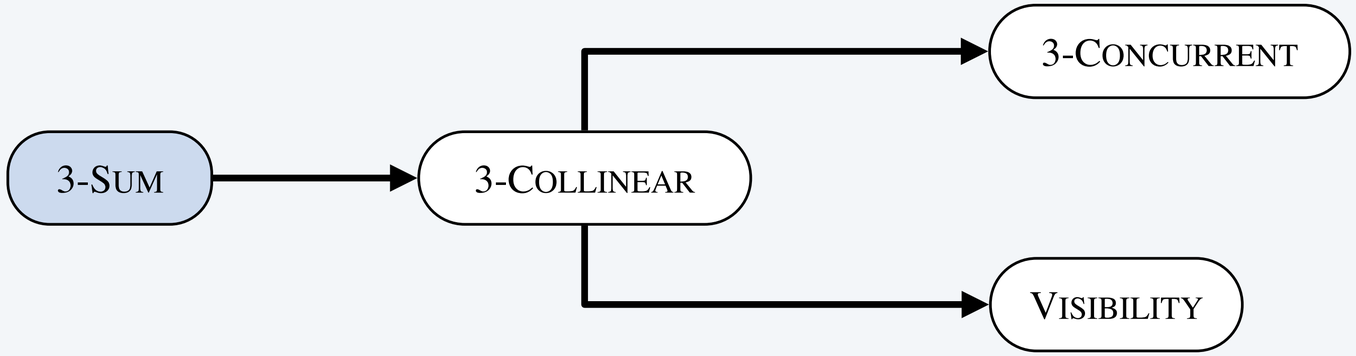
B. There exists a $\Theta(n^{1.8})$–time algorithm for 3-SUM but no $\Theta(n^{1.8})$–time algorithm for 3-COLLINEAR.

C. There exist $\Theta(n^{1.8})$–time algorithms for both 3-SUM and 3-COLLINEAR.

D. There do not exist $\Theta(n^{1.8})$–time algorithms for either 3-SUM or 3-COLLINEAR.

Transitivity

Transitivity. If X linear-time reduces to Y and Y linear-time reduces to Z, then X linear-time reduces to Z.



3-CONCURRENT: Given 2D lines $L_1, \ldots, L_n$, do any three contain the same point?

VISIBILITY: Given 2D points $p_1, p_2, \ldots, p_n$, is the line of sight between every pair free?

Summary

Efficient algorithm. Finite, definite, effective procedure with polynomially many steps.

X reduces to Y.

- Algorithm design. Efficient algorithm for Y yields efficient algorithm for X.
- Hardness. If X has no efficient algorithm, then neither does Y.

Q. Which problems can we use to prove hardness?

A. See next lecture.

Credits

image	source	license
Gears	Adobe Stock	Education License
Light Bulb	Adobe Stock	Education License
Cartoon of Turing Machine	Tom Dunne	
Donald Knuth	Hector Garcia-Molina	
Babbage Engine in Operation	xRez Studio	
Question Marks	pikpng.com	non-commercial use
Archimedes, Lever, and Fulcrum	unknown	

Lecture Slides © Copyright 2026 Marcel Dall'Agnol, Gillat Kol, Robert Sedgewick, and Kevin Wayne