CS61B

Lecture 28: Decomposition and Reductions

- Topological Sorting
- Shortest Paths on DAGs
- Longest Paths
- Reductions

Our Many Achievements

We’ve covered a tremendous amount of material so far.

- Programming practice: Java, IntelliJ, JUnit, correctness and timing tests, designing your own data structures that compose other data structures together.
- Asymptotic analysis.
- Tons of different abstract data types and their implementations, e.g. BST to implement a map, heaps to implement a PQ.
- Algorithms on graphs, e.g. shortest paths, minimum spanning trees, etc.

What is the Point of All of This?

A question worth pondering: Why this specific body of knowledge?

- Why is everyone pursuing it?
- Why do companies want to pursue people who have it?

Any thoughts?

- Efficient algorithms can improve performance of code, solve problems that are otherwise intractable.
  - These problems are real world problems.
- Shows an understanding of how computers work. Shibboleth of understanding of computing.
- For fun.
- Provides a common platform for interviewing for technical knowledge.
  - You now have a huge toolset you can use to solve problems.
- I really think we're changing the way you think.

Goals of Today

Today, to practice our problem solving skills, we'll work through some very challenging A-level problems using the tools we've already learned about.

Will focus on graphs, but the ideas today are more general.

Graph Problems So Far

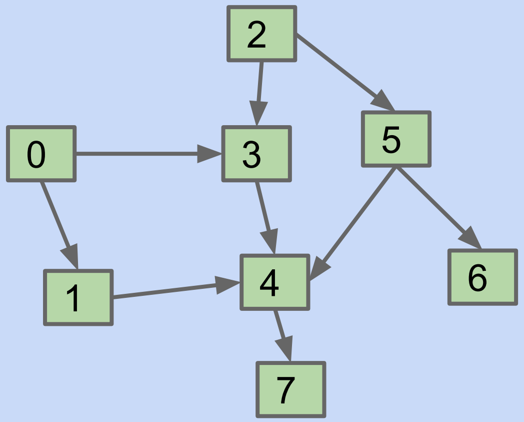
Problem	Problem Description	Solution	Efficiency
paths	Find a path from s to every reachable vertex.	DepthFirstPaths.java Demo	O(V+E) time Θ(V) space
shortest paths	Find the shortest path from s to every reachable vertex.	BreadthFirstPaths.java Demo	O(V+E) time Θ(V) space
shortest weighted paths	Find the shortest path, considering weights, from s to every reachable vertex.	DijkstrasSP.java Demo	O(E log V) time Θ(V) space
shortest weighted path	Find the shortest path, consider weights, from s to some target vertex	A*: Same as Dijkstra's but with h(v, goal) added to priority of each vertex. Demo	Time depends on heuristic. Θ(V) space

Graph Problems So Far

Problem	Problem Description	Solution	Efficiency
minimum spanning tree	Find a minimum spanning tree.	LazyPrimMST.java Demo	O(???) time Θ(???) space
minimum spanning tree	Find a minimum spanning tree.	PrimMST.java Demo	O(E log V) time Θ(V) space
minimum spanning tree	Find a minimum spanning tree.	KruskalMST.java Demo	O(E log E) time Θ(E) space

Topological Sorting

Topological Sort

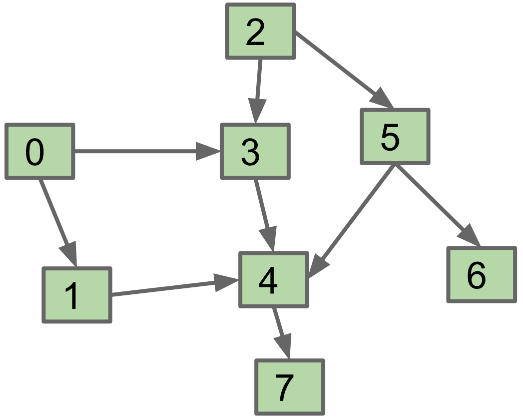


Suppose we have tasks 0 through 7, where an arrow from v to w indicates that v must happen before w.

- What algorithm do we use to find a valid ordering for these tasks?
- Valid orderings include: [0, 2, 1, 3, 5, 4, 7, 6], [2, 0, 3, 5, 1, 4, 6, 7], …

Any suggestions on where we'd start?

Topological Sort



Start at 2:

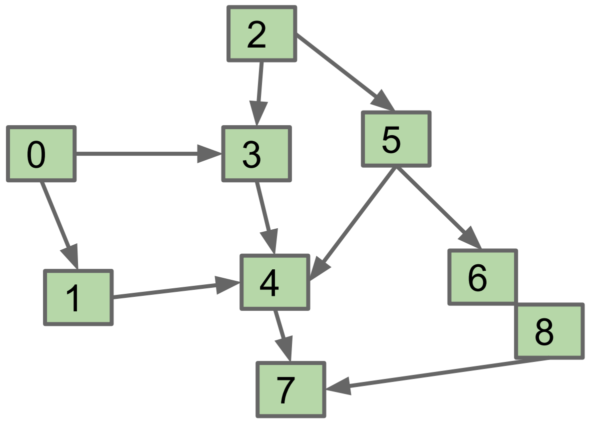
- Go as far as you can get (DFS).
    - 2, 3, 4, 7, 5, 6

Suppose we have tasks 0 through 7, where an arrow from v to w indicates that v must happen before w.

- What algorithm do we use to find a valid ordering for these tasks?
- Valid orderings include: [0, 2, 1, 3, 5, 4, 7, 6], [2, 0, 3, 5, 1, 4, 6, 7], …

Any suggestions on where we'd start?

Topological Sort



Multiple breadth first searches from vertices of in-degree 0.

- Great places to start.

[0/2, 1/3/5, 4/6, 7]

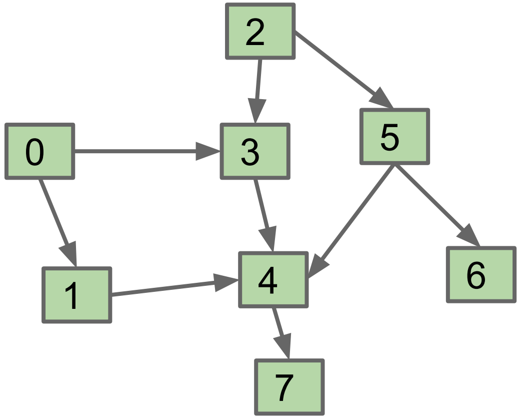
[0/2/8, 1/3/5/7,

Suppose we have tasks 0 through 7, where an arrow from v to w indicates that v must happen before w.

- What algorithm do we use to find a valid ordering for these tasks?
- Valid orderings include: [0, 2, 1, 3, 5, 4, 7, 6], [2, 0, 3, 5, 1, 4, 6, 7], …

Any suggestions on where we'd start?

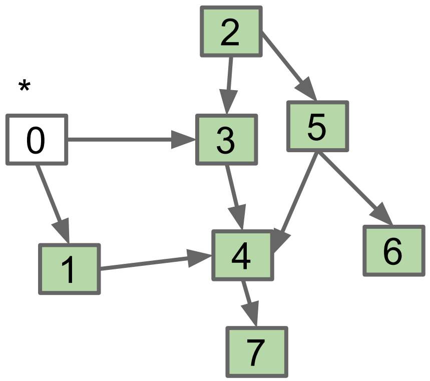
Solution (Spoiler Alert)

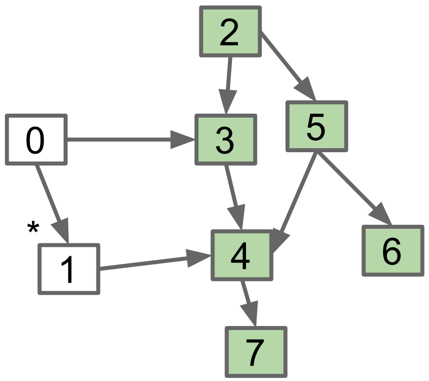


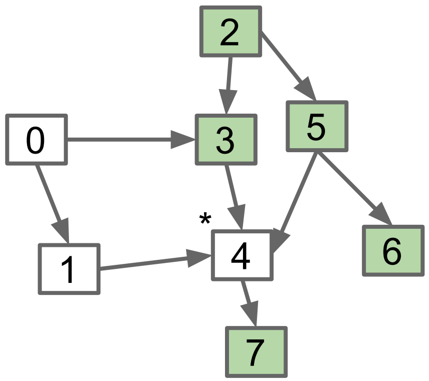
Perform a DFS traversal from every vertex with indegree 0, NOT clearing markings in between traversals.

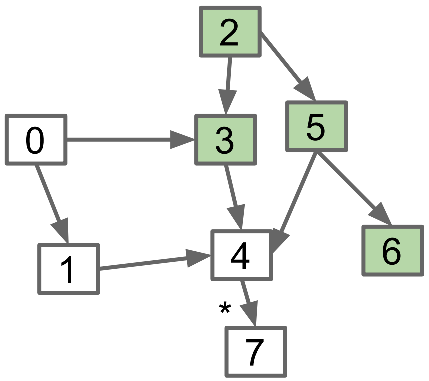
- Record DFS postorder in a list.
- Topological ordering is given by the reverse of that list (reverse postorder).

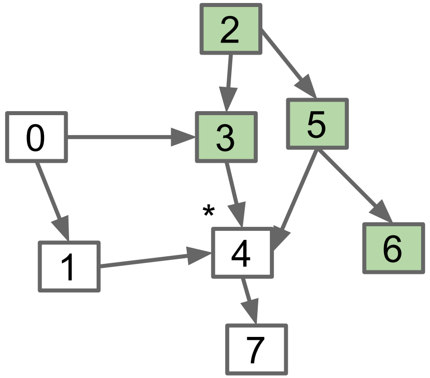
Topological Sort (Demo 1/2)

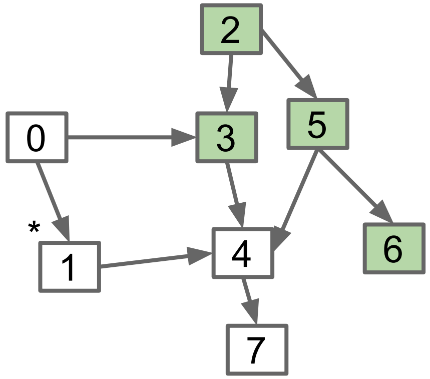

Postorder: []

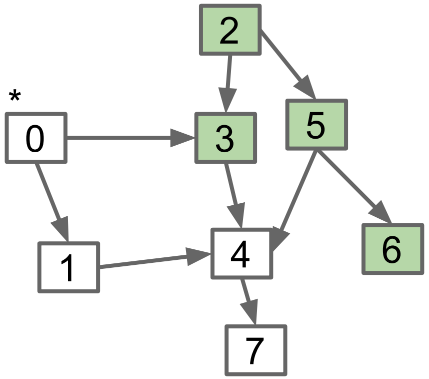

Postorder: []

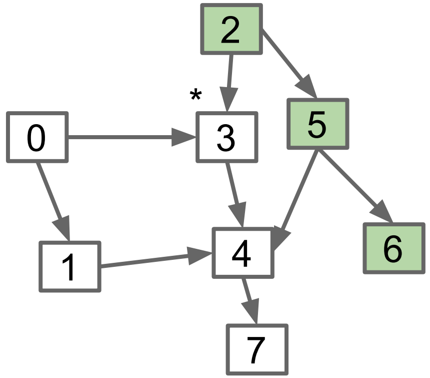

Postorder: []


Postorder: [7]

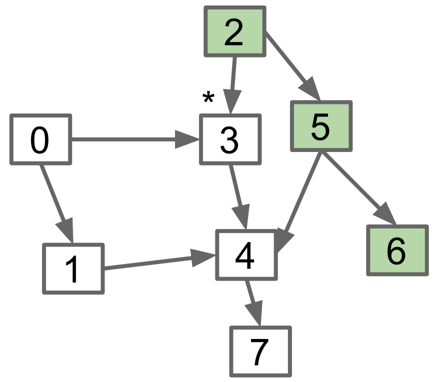

Postorder: [7, 4]


Postorder: [7, 4, 1]

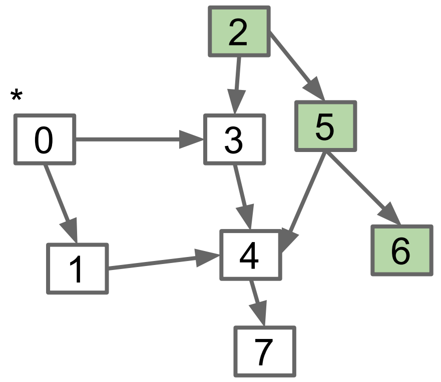

Postorder: [7, 4, 1]


Postorder: [7, 4, 1, 3]

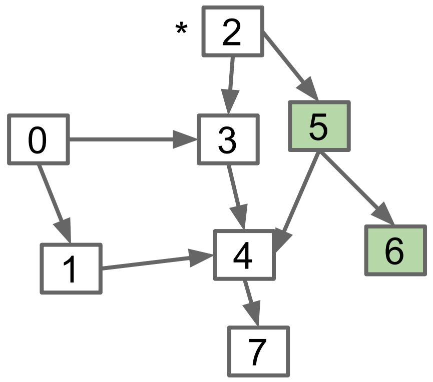
Topological Sort (Demo 2/2)



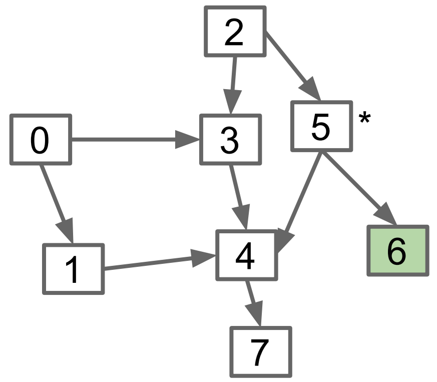
Postorder: [7, 4, 1, 3]



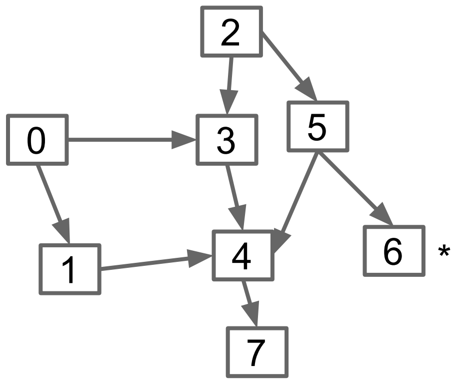
Postorder: [7, 4, 1, 3, 0]



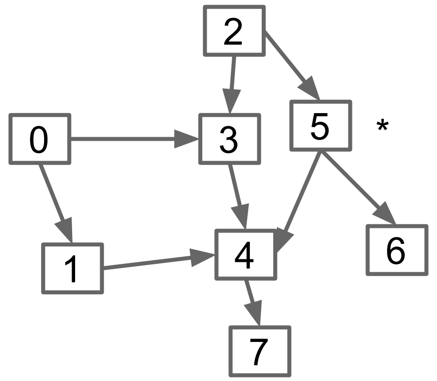
Postorder: [7, 4, 1, 3, 0]



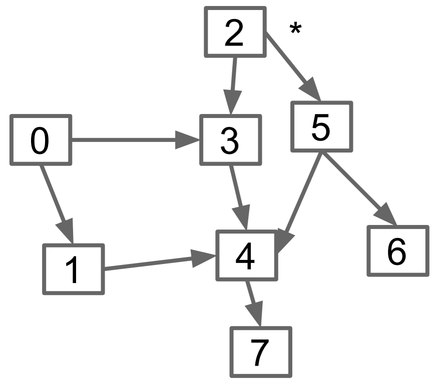
Postorder: [7, 4, 1, 3, 0]



Postorder: [7, 4, 1, 3, 0, 6]

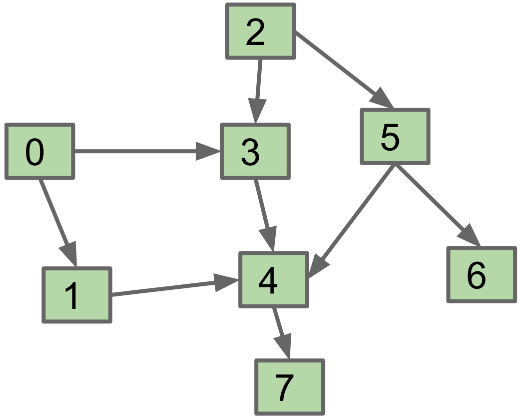


Postorder: [7, 4, 1, 3, 0, 6, 5]



Postorder: [7, 4, 1, 3, 0, 6, 5, 2]

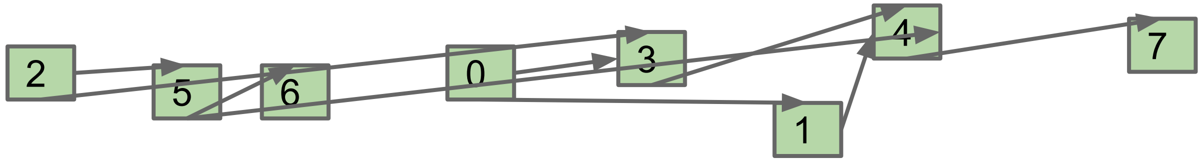
Solution (Spoiler Alert)

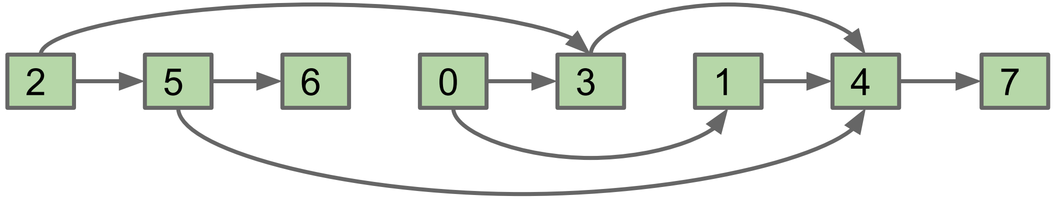

$$\begin{tikzcd}
 & & 2 \arrow[d] \arrow[r] & 5 \arrow[dl] \arrow[dr] & \\
0 \arrow[rr] \arrow[d] & & 3 \arrow[d] & & 6 \\
1 \arrow[rr] & & 4 \arrow[d] & & \\
 & & 7 & &
\end{tikzcd}$$

Perform a DFS traversal from every vertex with indegree 0, NOT clearing markings in between traversals.

- Record DFS postorder in a list: [7, 4, 1, 3, 0, 6, 5, 2]
- Topological ordering is given by the reverse of that list (reverse postorder):
  - [2, 5, 6, 0, 3, 1, 4, 7]

Topological Sort





The reason it's called topological sort: Can think of this process as sorting our nodes so they appear in an order consistent with edges, e.g. [2, 5, 6, 0, 3, 1, 4, 7]

- When nodes are sorted in diagram, arrows all point rightwards.

Depth First Search

Be aware, that when people say “Depth First Search”, they sometimes mean with restarts, and they sometimes mean without.

For example, when we did DepthFirstPaths for reachability, we did not restart.

For Topological Sort, we restarted from every vertex with indegree 0.

Question

What is the runtime to find all vertices of indegree 0?

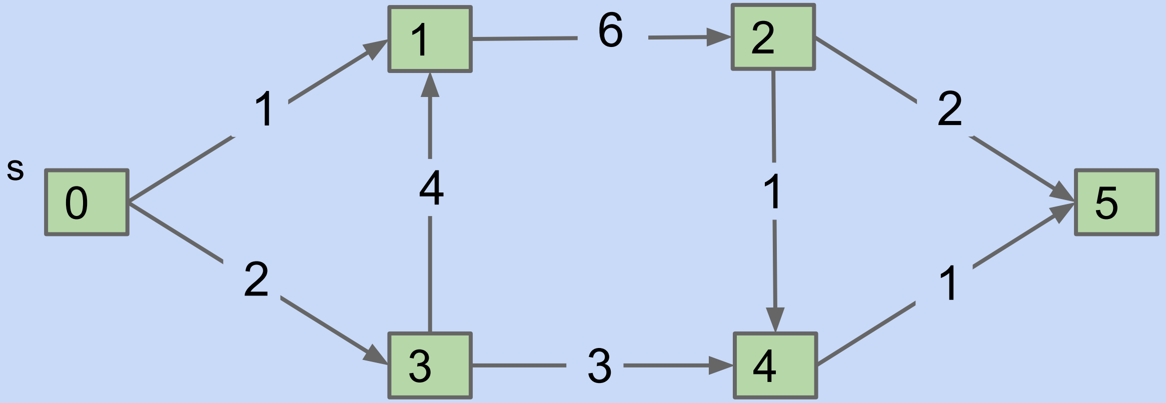
- Interesting thing I did not tell you: You don’t have to.

Another better topological sort algorithm:

- Run DFS from an arbitrary vertex.
- If not all marked, pick an unmarked vertex and do it again.
- Repeat until done.

Test Your Understanding

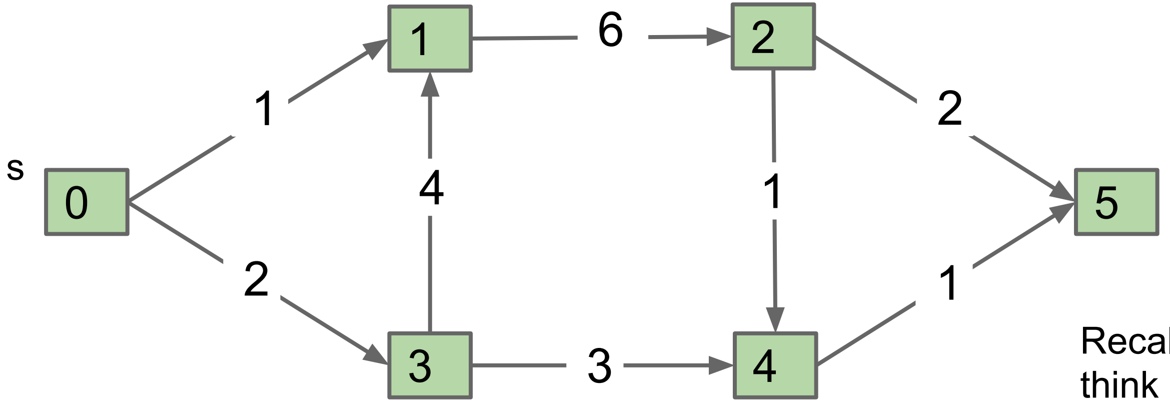
Give a topological ordering for the graph below (a.k.a. topological sort).



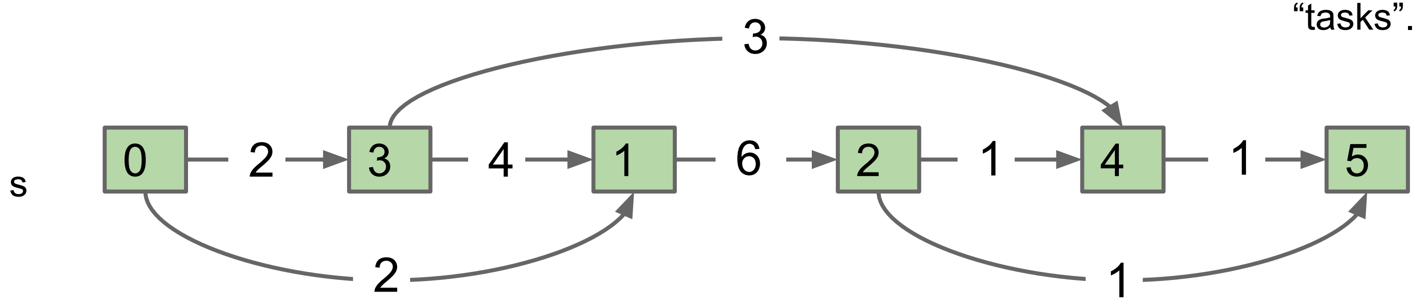
Test Your Understanding

Give a topological ordering for the graph below (a.k.a. topological sort)

- 0, 3, 1, 2, 4, 5 (because DFS postorder was 542130)



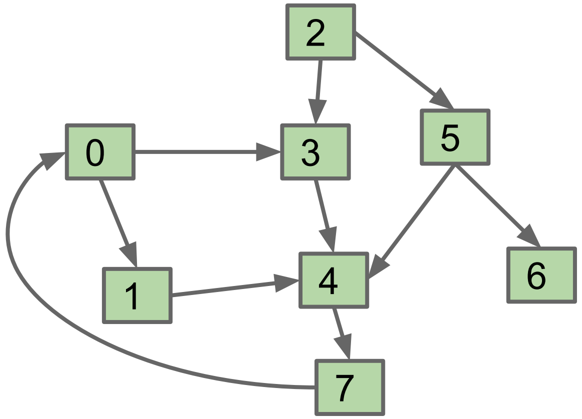
Recall that we can think of topological sort as an ordering of "tasks".



Directed Acyclic Graphs

A topological sort only exists if the graph is a directed acyclic graph (DAG).

- For the graph below, there is NO possible ordering where all arrows are respected.



DAGs appear in many real world applications, and there are many graph algorithms that only work on DAGs.

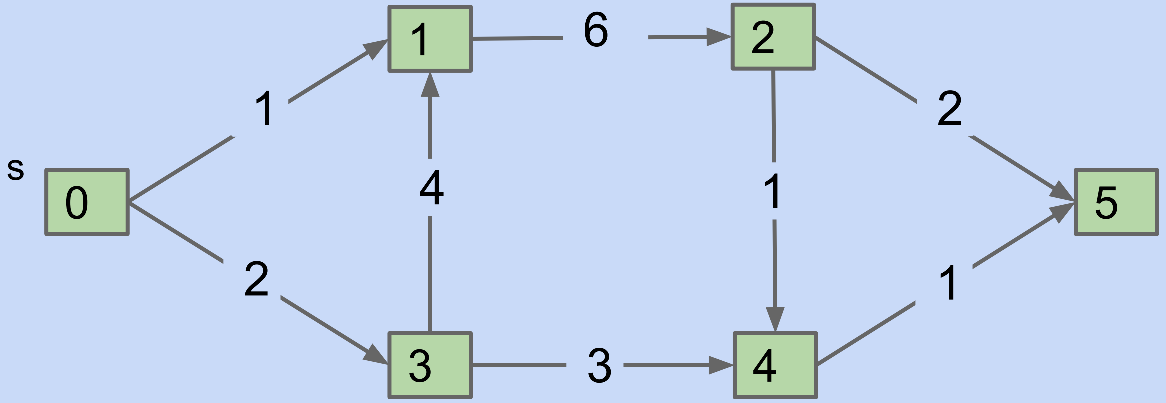
Graph Problems

Problem	Problem Description	Solution	Efficiency
topological sort	Find an ordering of vertices that respects edges of our DAG.	Demo Topological.java	O(V+E) time Θ(V) space

Shortest Paths on DAGs

Shortest Paths Warmup

What is the shortest paths tree for the graph below, using s as the source?
In what order will Dijkstra's algorithm visit the vertices?

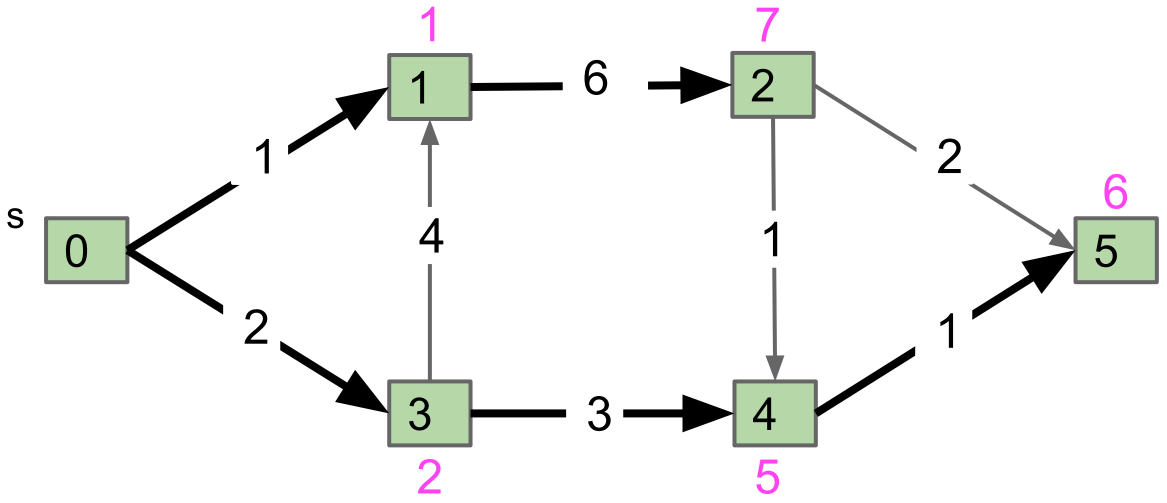


Graph from Algorithms, by Vazirani/Papadimitriou

Shortest Paths Warmup

What is the shortest paths tree for the graph below, using s as the source?
In what order will Dijkstra's algorithm visit the vertices?

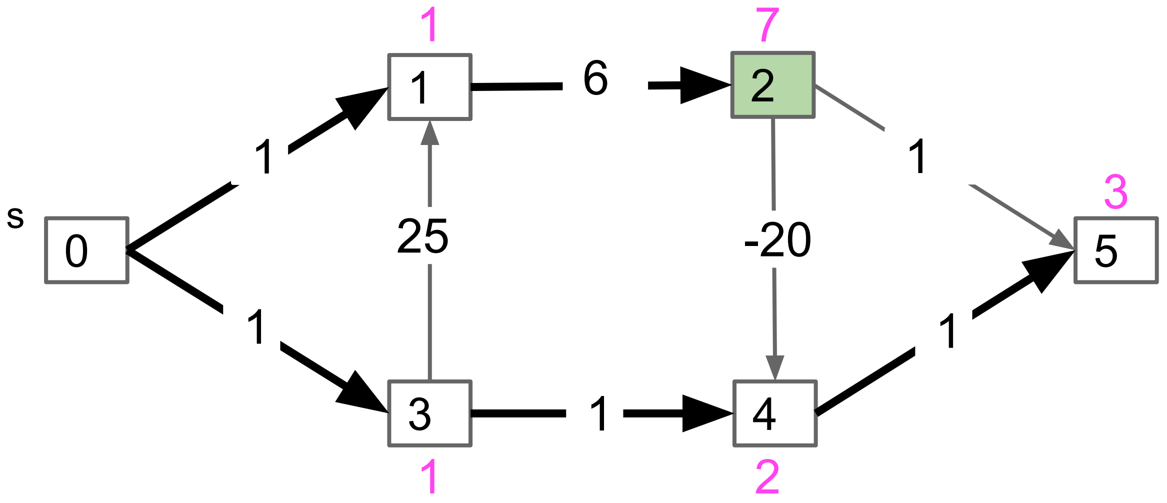
- 0, 1, 3, 4, 5, 2



Shortest Paths Warmup

If we allow negative edges, Dijkstra's algorithm can fail.

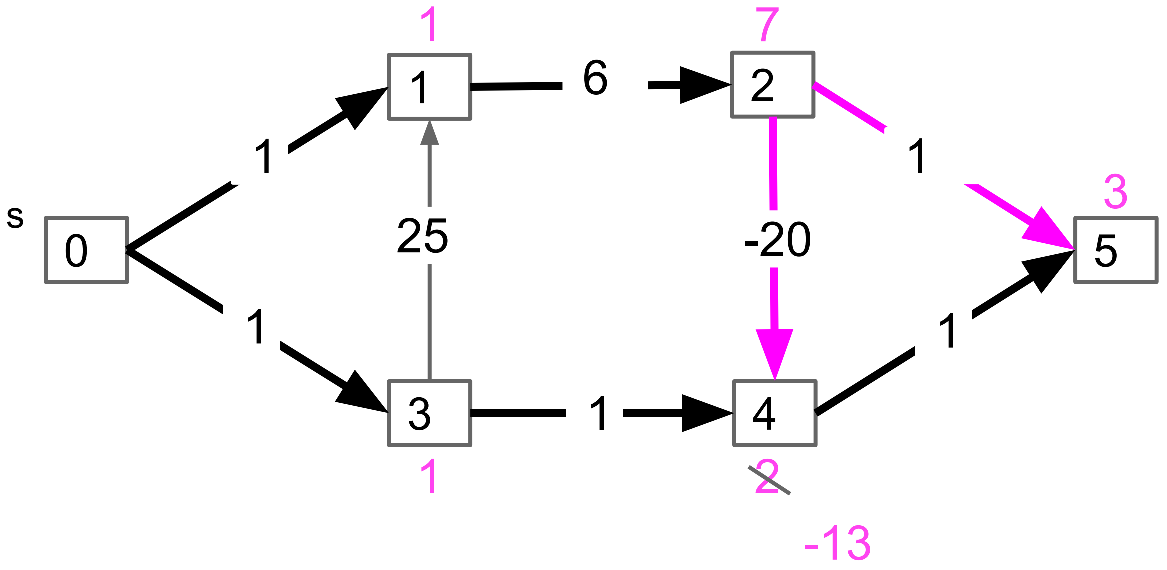
- For example, below we see Dijkstra's just before vertex 2 is visited.



Shortest Paths Warmup

If we allow negative edges, Dijkstra's algorithm can fail.

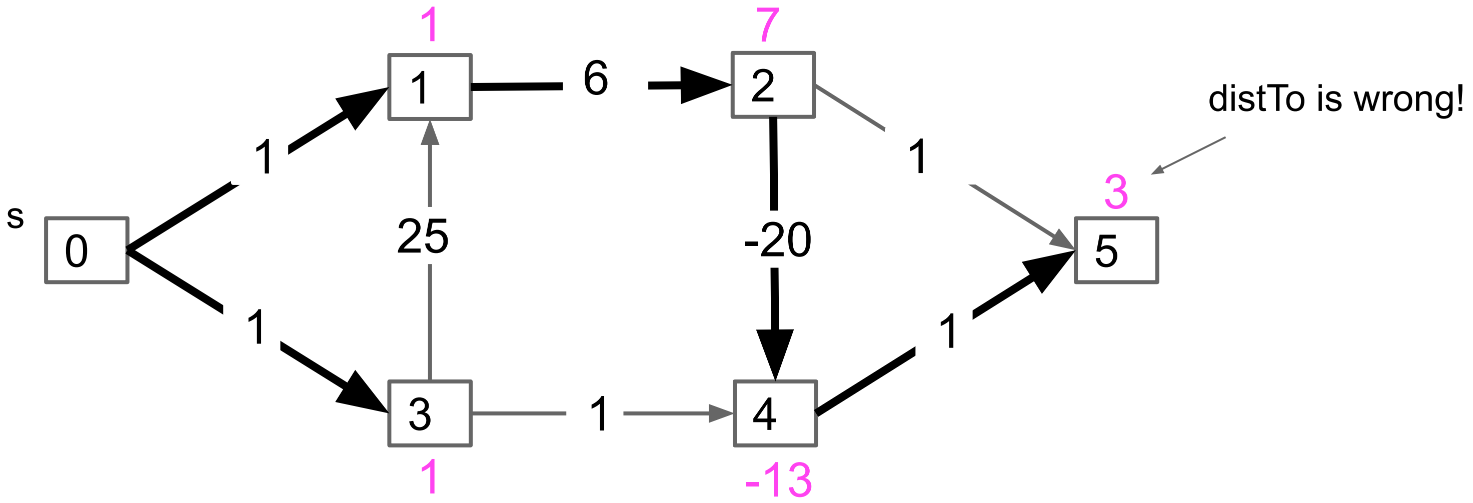
- For example, below we see Dijkstra's just before vertex 2 is visited.
- Relaxation on 4 succeeds, but distance to 5 will never be updated.



Shortest Paths Warmup

If we allow negative edges, Dijkstra’s algorithm can fail.

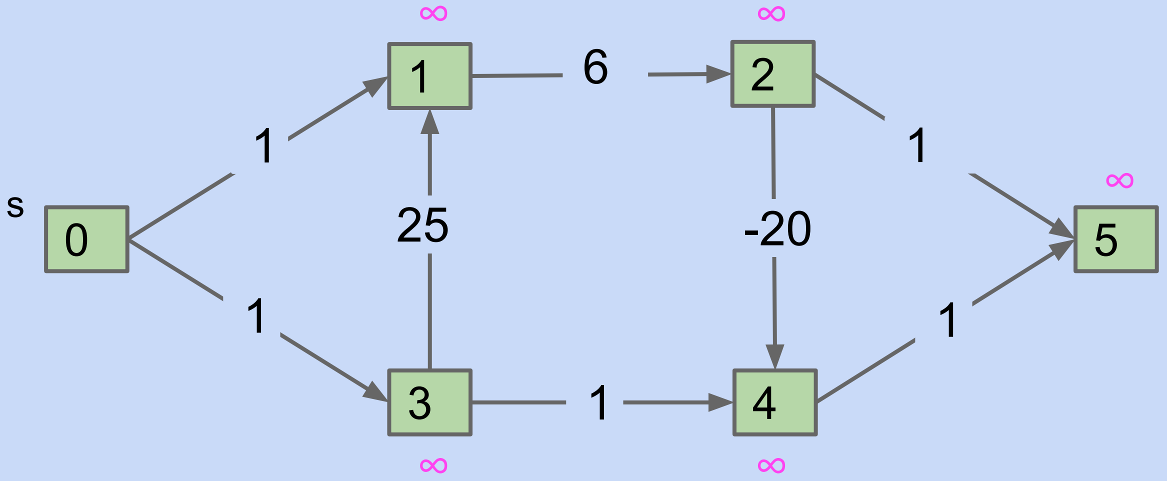
- For example, below we see Dijkstra’s just before vertex 2 is visited.
- Relaxation on 4 succeeds, but distance to 5 will never be updated.



Challenge

Try to come up with an algorithm for shortest paths on a DAG that works even if there are negative edges.

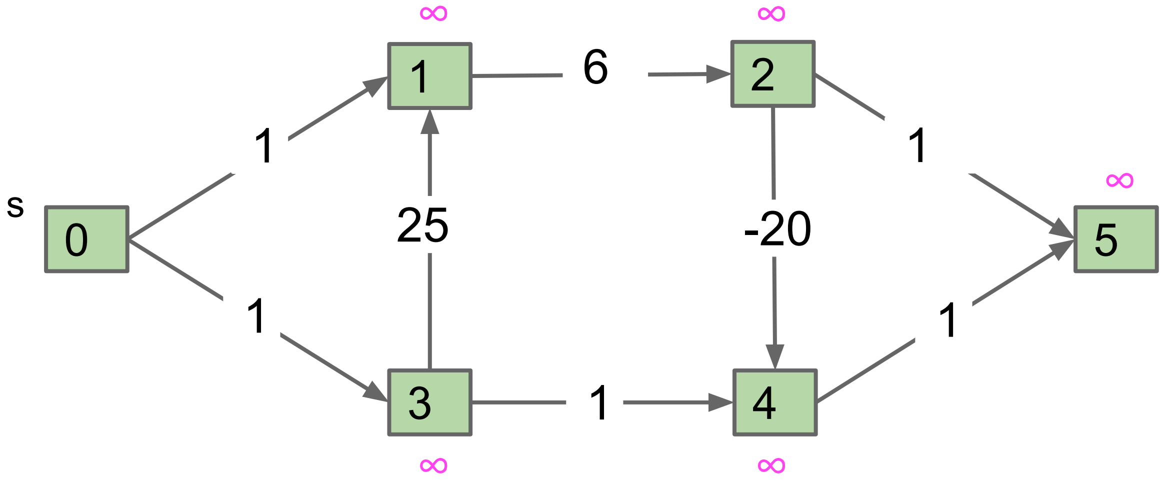
- Hint: You should still use the “relax” operation as a basic building block.



Challenge

Try to come up with an algorithm for shortest paths on a DAG that works even if there are negative edges.

- Hint: You should still use the “relax” operation as a basic building block.

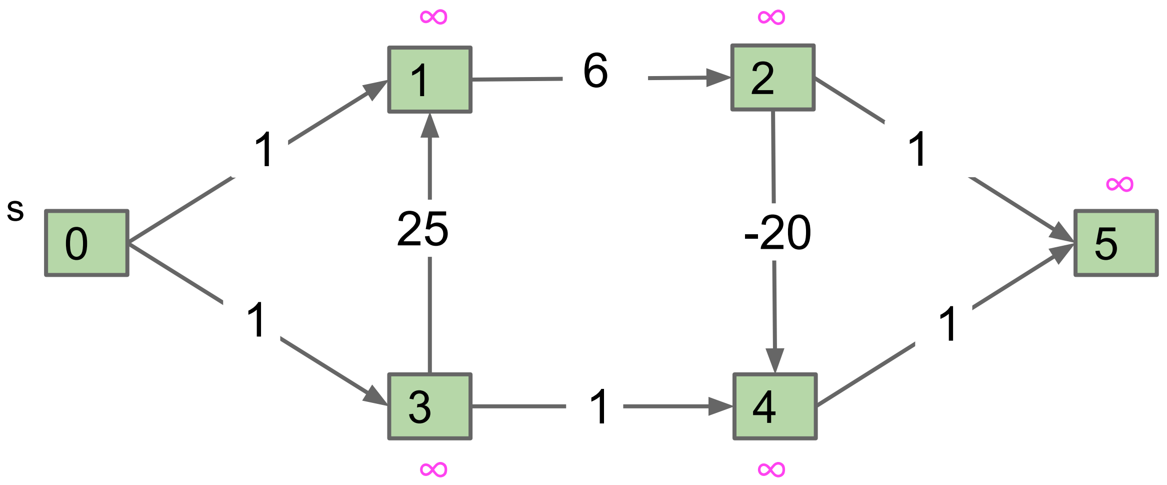


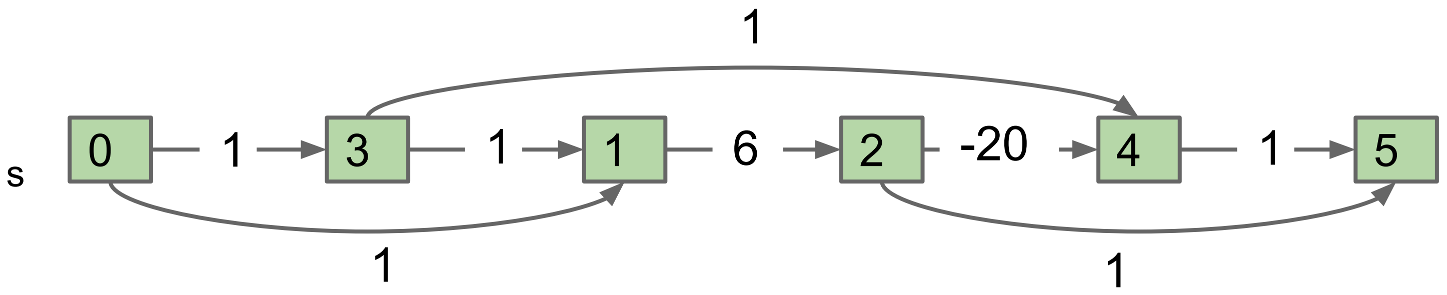
One simple idea: Visit vertices in topological order.

- On each visit, relax all outgoing edges.
- Each vertex is visited only when all possible info about it has been used!

The DAG SPT Algorithm: Relax in Topological Order

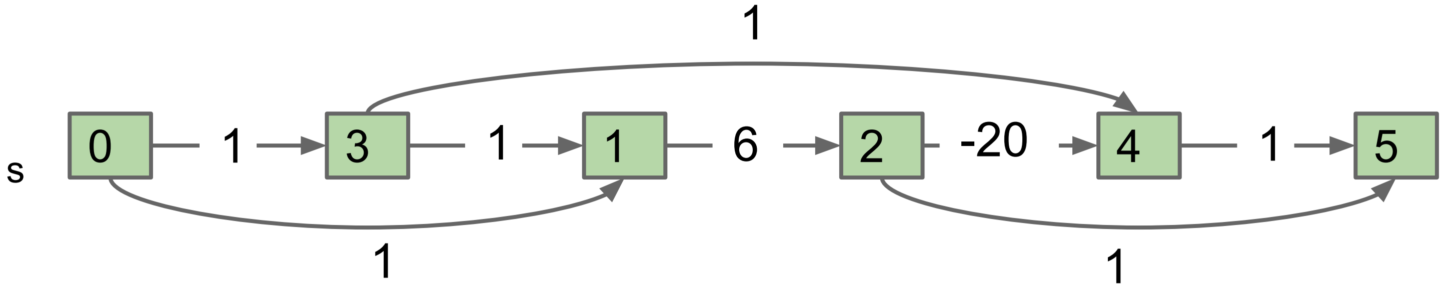
First: We have to find a topological order, e.g. 031245. Runtime is O(V + E).





The DAG SPT Algorithm: Relax in Topological Order

Second: We have to visit all the vertices in topological order, relaxing all edges as we go. Let's see a demo [Link].



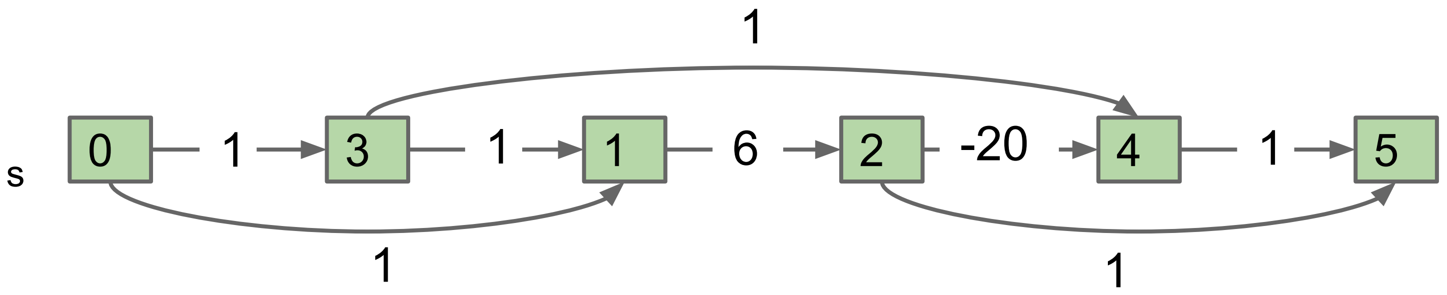
The DAG SPT Algorithm: Relax in Topological Order

Second: We have to visit all the vertices in topological order, relaxing all edges as we go. Let's see a demo [Link].

- Runtime for step 2 is also O(V + E).

Quick note: In office hours, someone asked, why isn't it O(V*E), since we're relaxing all edges from each vertex.

- Keep in mind that E is the TOTAL number of edges in the entire graph, not the number of edges per vertex, e.g. for graph below E = 8.



Graph Problems

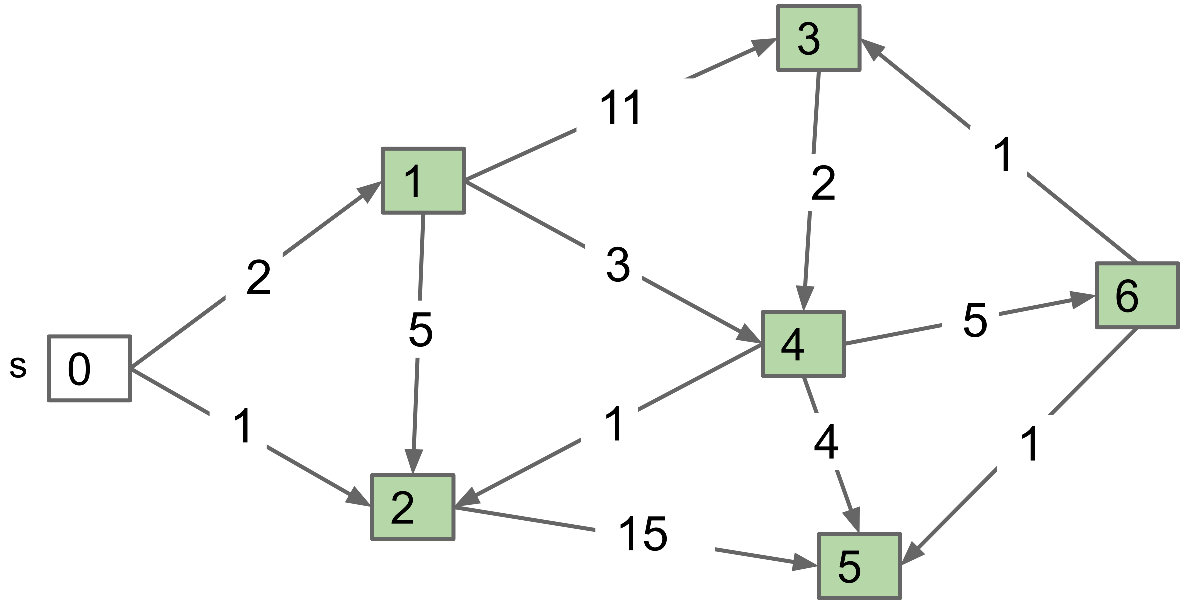
Problem	Problem Description	Solution	Efficiency
topological sort	Find an ordering of vertices that respects edges of our DAG.	Demo Code: Topological.java	O(V+E) time Θ(V) space
DAG shortest paths	Find a shortest paths tree on a DAG.	Demo Code: AcyclicSP.java	O(V+E) time Θ(V) space

Note: The DAG shortest paths solution uses the topological sort solution as a subroutine.

Longest Paths

The Longest Paths Problem

Consider the problem of finding the longest path tree (LPT) from s to every other vertex. The path must be simple (no cycles!).

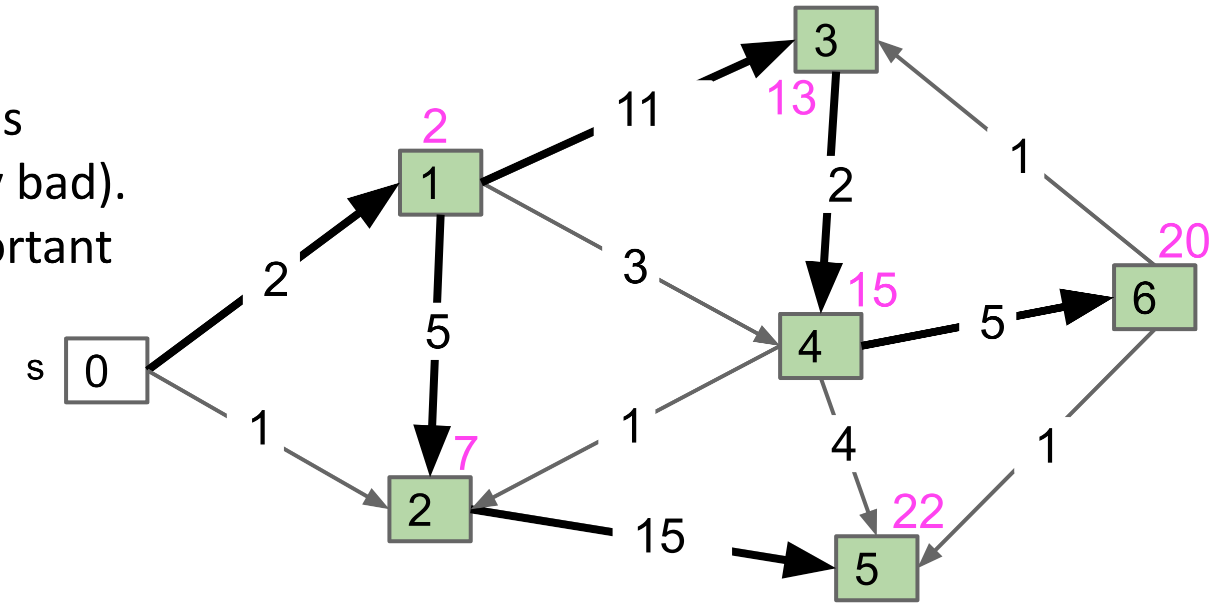


The Longest Paths Problem

Consider the problem of finding the longest path tree (LPT) from s to every other vertex. The path must be simple (no cycles!).

Some surprising facts:

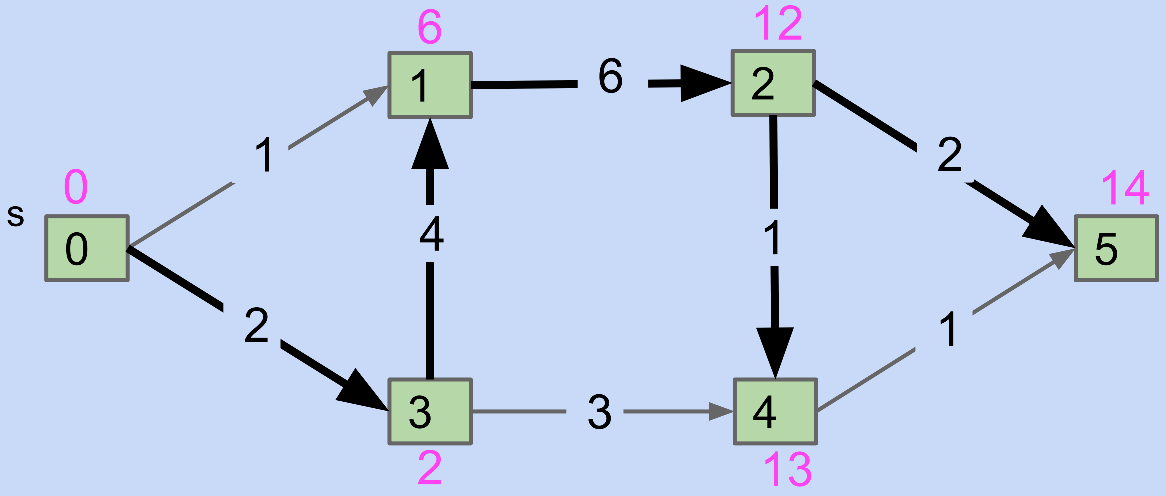
- Best known algorithm is exponential (extremely bad).
- Perhaps the most important unsolved problem in mathematics.



The Longest Paths Problem on DAGs

Difficult challenge for you.

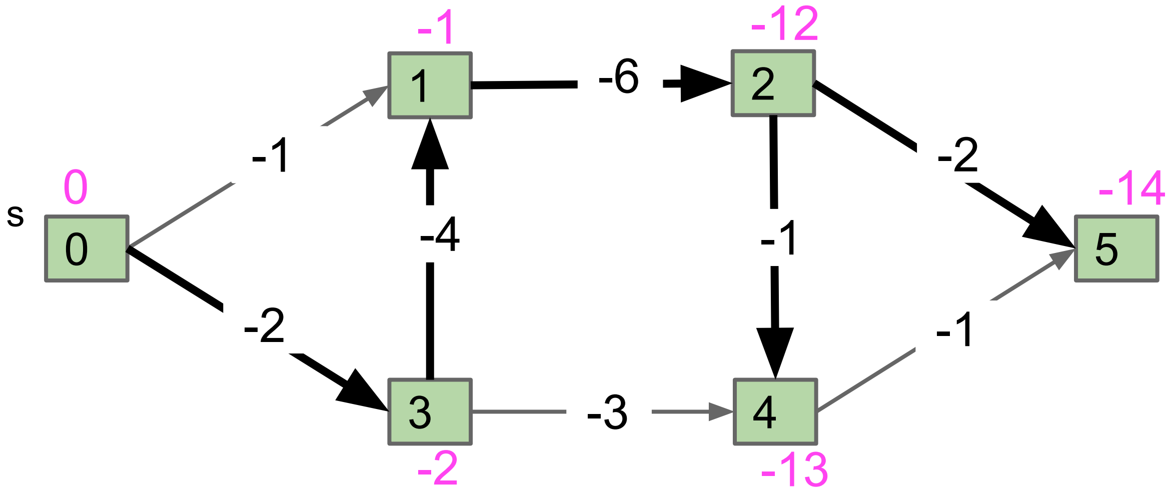
- Solve the LPT problem on a directed acyclic graph.
- Algorithm must be O(E + V) runtime.



The Longest Paths Problem on DAGs

DAG LPT solution for graph G:

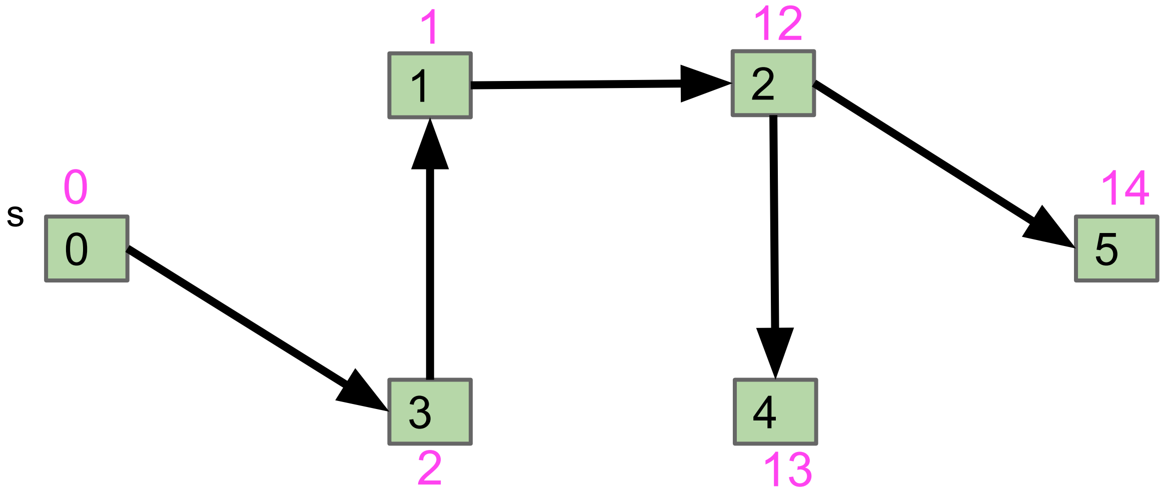
- Form a new copy of the graph G’ with signs of all edge weights flipped.
- Run DAGSPT on G’ yielding result X.
- Flip signs of all values in X.distTo. X.edgeTo is already correct.



The Longest Paths Problem on DAGs

DAG LPT solution for graph G:

- Form a new copy of the graph G' with signs of all edge weights flipped.
- Run DAGSPT on G' yielding result X.
- Flip signs of all values in X.distTo. X.edgeTo is already correct.



A Note on “Mathematical Maturity”

If you have a very high degree of so-called “mathematical maturity”, this algorithm should seem plainly correct.

There’s no real need to prove anything or show demos.

- We know DAG SPT works on graphs with negative edge weights.
- We also know that -(-a + -b + -c + -d) = a + b + c + d.

Part of what you’re learning in your intense technical education here at Berkeley is mathematical maturity. Hasn’t been a major focus in 61B, but will be in other courses like 16A, 16B, 70, 170, ...

Highly Recommended Exercise For Later

Play around with the longest paths problem and convince yourself that it is actually very hard.

- Try to develop an intuition for why it is hard. Even better if you try to put it into english.
- Try searching the internet for “why longest paths hard” or similar if you’re having trouble really pinning down what’s so hard about it.

Graph Problems

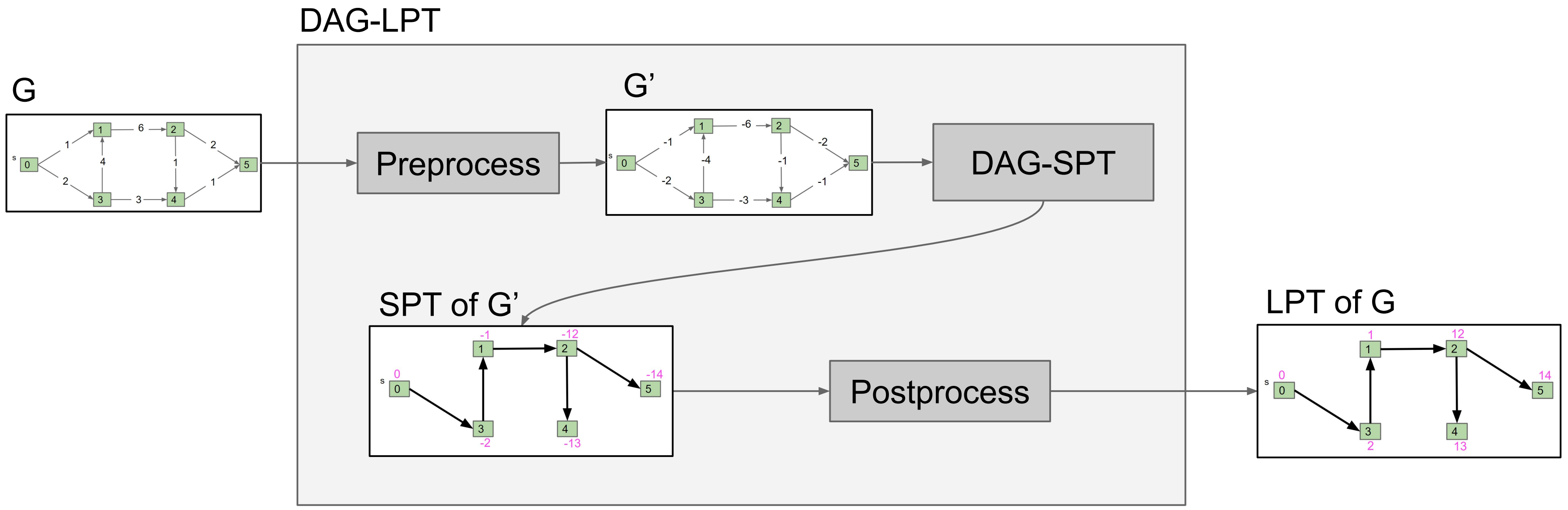
Problem	Problem Description	Solution	Efficiency
topological sort	Find an ordering of vertices that respects edges of our DAG.	Demo Code: Topological.java	O(V+E) time Θ(V) space
DAG shortest paths	Find a shortest paths tree on a DAG.	Demo Code: AcyclicSP.java	O(V+E) time Θ(V) space
longest paths	Find a longest paths tree on a graph.	No known efficient solution.	O(???) time O(???) space
DAG longest paths	Find a longest paths tree on a DAG.	Flip signs, run DAG SPT, flip signs again.	O(V+E) time Θ(V) space

Reduction (170 Preview)

DAG Longest Paths

The problem solving we just used probably felt a little different than usual:

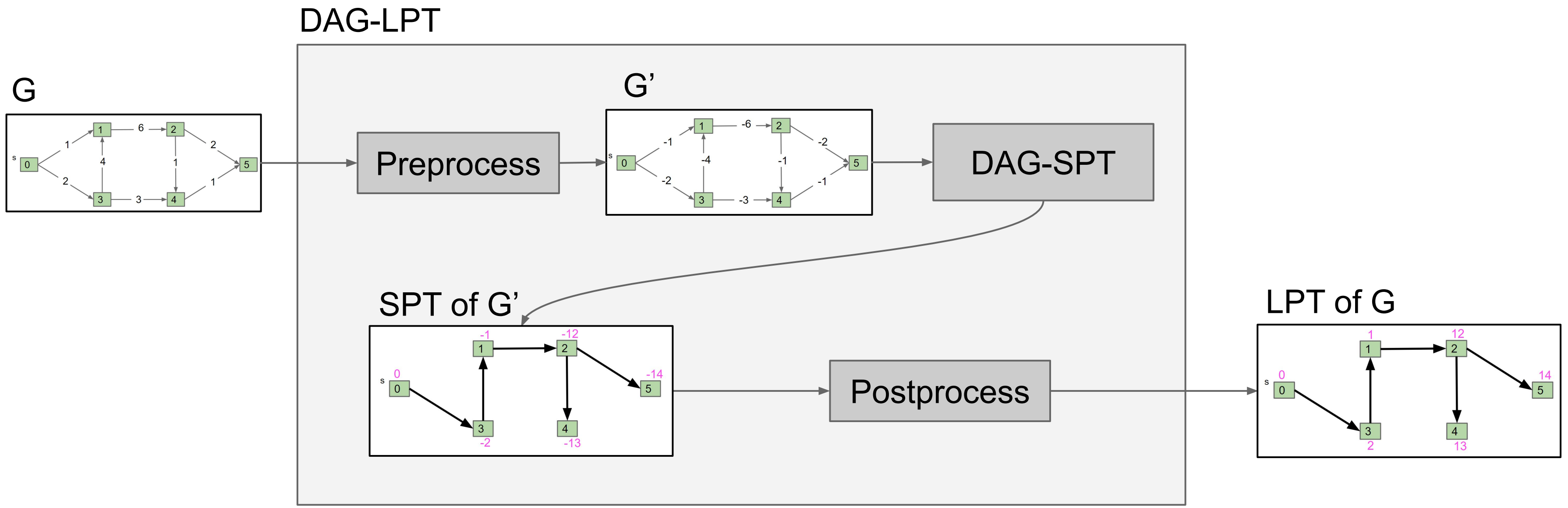
- Given a graph G, we created a new graph G' and fed it to a related (but different) algorithm, and then interpreted the result.



Reduction

This process is known as **reduction**.

- Since DAG-SPT can be used to solve DAG-LPT, we say that "DAG-LPT reduces to DAG-SPT".



Reduction Analogy

This process is known as reduction.

- Since DAG-SPT can be used to DAG-LPT, we say that “DAG-LPT reduces to DAG-SPT”.

As a real-world analog, suppose we want to climb a hill. There are many ways to do this:

- “Climbing a hill” reduces to “riding a ski lift.”
- “Climbing a hill” reduces to “being shot out of a cannon.”
- “Climbing a hill” reduces to “riding a bike up the hill.”

Reduction Definition (Informal)

This process is known as reduction.

- Since DAG-SPT can be used to DAG-LPT, we say that “DAG-LPT reduces to DAG-SPT”.

Algorithms by Dasgupta, Papadimitriou, and Vazirani defines a reduction informally as follows:

- “If any subroutine for task Q can be used to solve P, we say P reduces to Q.”

Can also define the idea formally, but **way** beyond the scope of our class.

- If you’re curious, you can read more about Karp and Cook reductions.

Reduction is More Than Sign Flipping

Let’s see a richer example of a reduction.

The Independent Set Problem

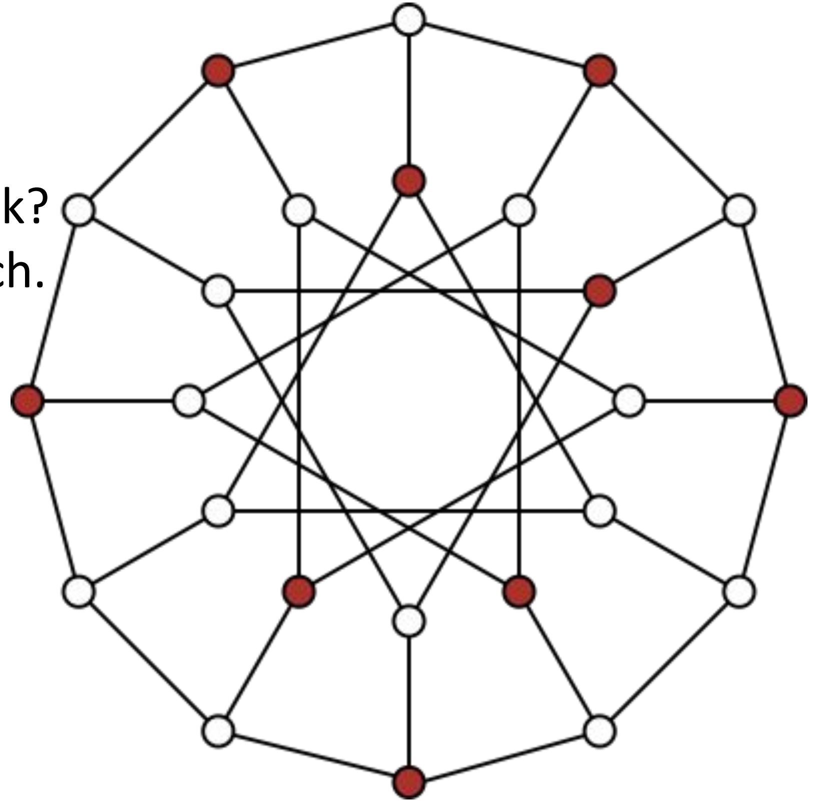
An independent set is a set of vertices in which no two vertices are adjacent.

The Independent-set Problem:

- Does there exist an independent set of size k?
- i.e. color k vertices red, such that none touch.

Example for the graph on the right and k = 9

- For this particular graph, N=24.



The 3SAT Problem

3SAT: Given a boolean formula, does there exist a truth value for boolean variables that obeys a set of 3-variable disjunctive constraints?

Example: $\overbrace{\text{(x1 || x2 || !x3)}}^{\text{3 variable disjunctive constraint}}$ && (x1 || !x1 || x1) && (x2 || x3 || x4)

- Solution: x1 = true, x2 = true, x3 = true, x4 = false

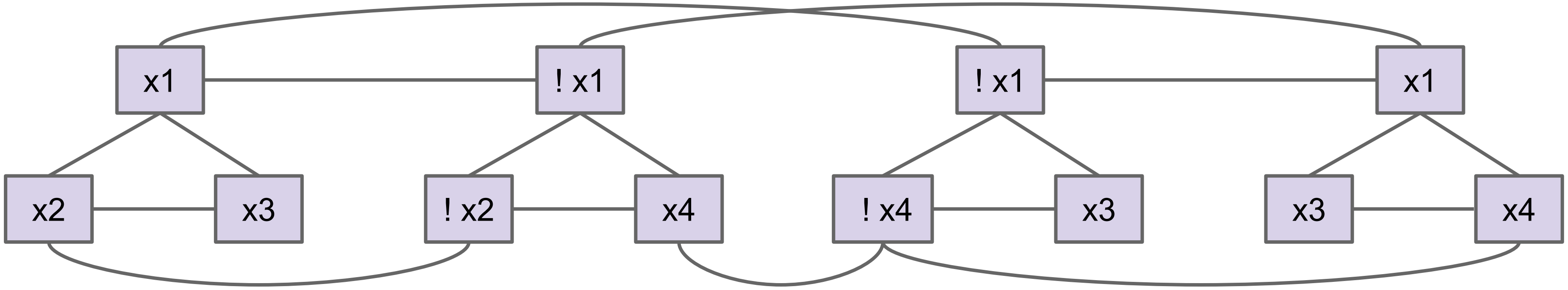
3SAT Reduces to Independent Set

Proposition: 3SAT reduces to Independent-set.

Proof. Given an instance ϕ of 3-SAT, create an instance G of Independent-set:

- For each clause in ϕ, create 3 vertices in a triangle.
- Add an edge between each literal and its negation (can't both be true in 3SAT means can't be in same set in Independent-set)

k = number of variables = 4



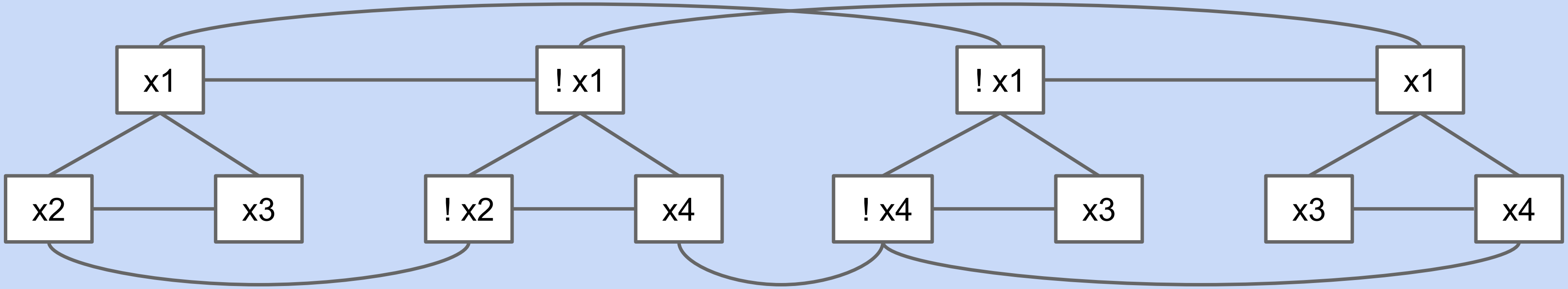
Φ = (x1 or x2 or x3) and (! x1 or ! x2 or x4) and (! x1 or x3 or ! x4) and (x1 or x3 or x4)

3SAT Reduces to Independent Set

Find an independent set of size k = 4. Use this set to generate a solution to the 3SAT problem.

- Reminder: An independent set of size 4 is a set of 4 (red) vertices that do not touch.

k = number of variables = 4



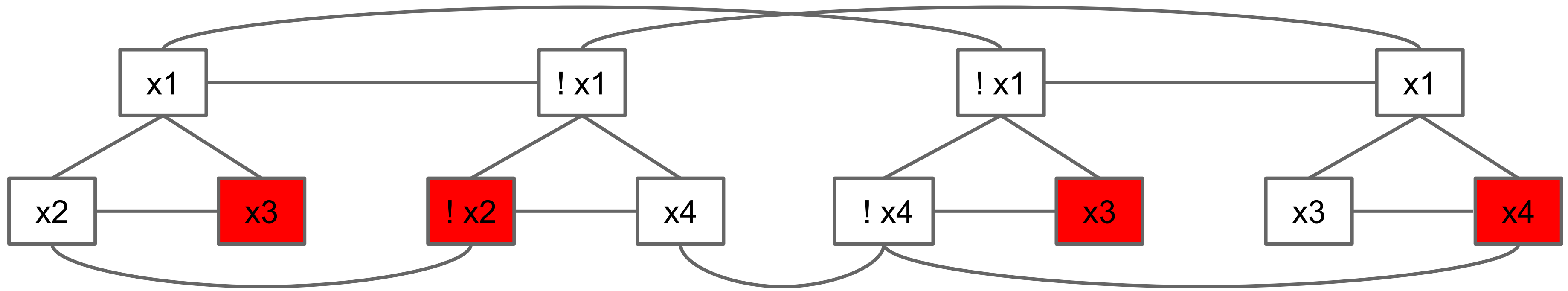
Φ = (x1 or x2 or x3) and (! x1 or ! x2 or x4) and (! x1 or x3 or ! x4) and (x1 or x3 or x4)

3SAT Reduces to Independent Set (Your Answer)

Find an independent set of size k = 4. Use this set to generate a solution to the 3SAT problem.

- Reminder: An independent set of size 4 is a set of 4 (red) vertices that do not touch.

k = number of variables = 4



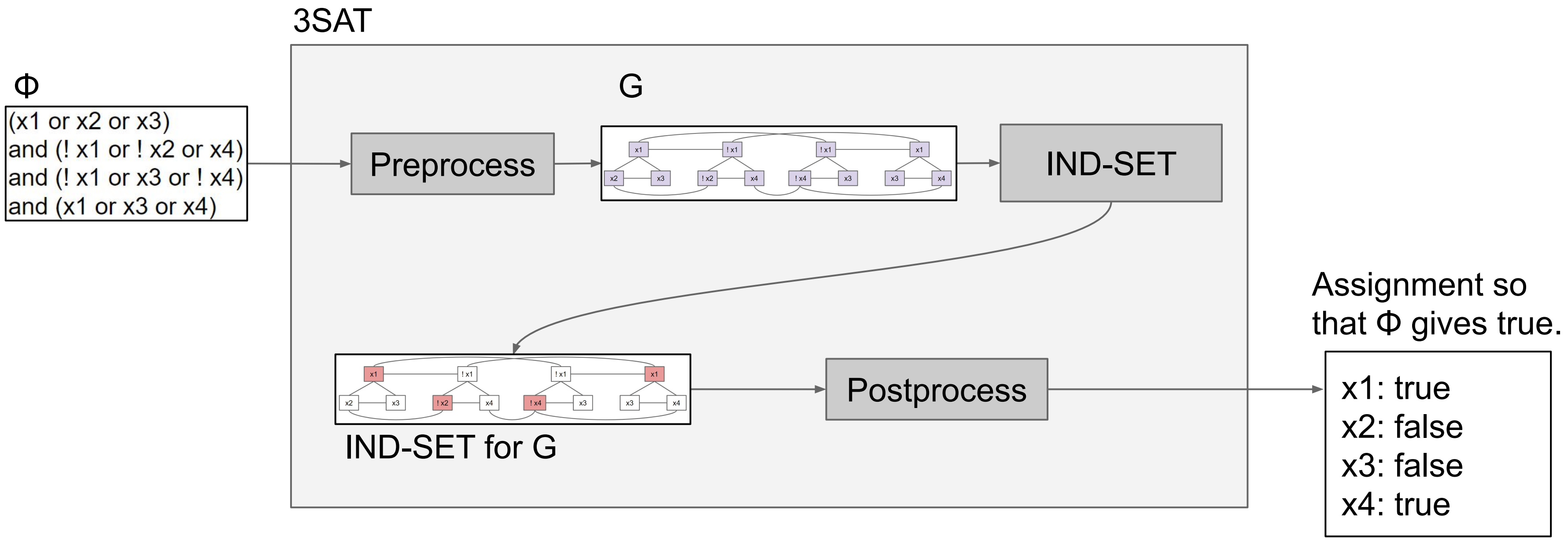
Φ = (x1 or x2 or x3) and (! x1 or ! x2 or x4) and (! x1 or x3 or ! x4) and (x1 or x3 or x4)

X3: True
X2: False
X4: True
X1: Doesn't matter

Reduction

Since IND-SET can be used to solve 3SAT, we say that "3SAT reduces to IND-SET".

- Note: 3SAT is not a graph problem!
- Note: Reductions don't always involve creating graphs.



Reductions and Decomposition

Arguably, we've been doing something like a reduction all throughout the course.

- Abstract lists reduce to arrays (or linked lists).
- Synthesizing guitar string sound reduces to ArrayRingBuffer.
- The percolation problem reduces to DisjointSets.
- ExtrinsicMinPQ reduces to (operations on whatever instance variables you used).

These examples aren't reductions exactly.

- We aren't just calling a subroutine.
- A better term would be decomposition: Taking a complex task and breaking it into smaller parts. This is the heart of computer science.
  - Using appropriate abstractions makes problem solving vastly easier.

Generational Changes in the Human Mind