# CS2040S
# Data Structures and Algorithms

Directed Acyclic Graphs!

# Plan for today:

Directed Acyclic Graphs (DAG)

Topological Order

Topological Sort

Shortest Path in a DAG

Shortest Path in a tree

# What is a directed graph?

Graph consists of two types of elements:

Nodes (or vertices)

- At least one.

Edges (or arcs)

- Each edge connects two nodes in the graph
- Each edge is unique.
- Each edge is **directed**.

# Scheduling

Set of tasks for baking cookies:

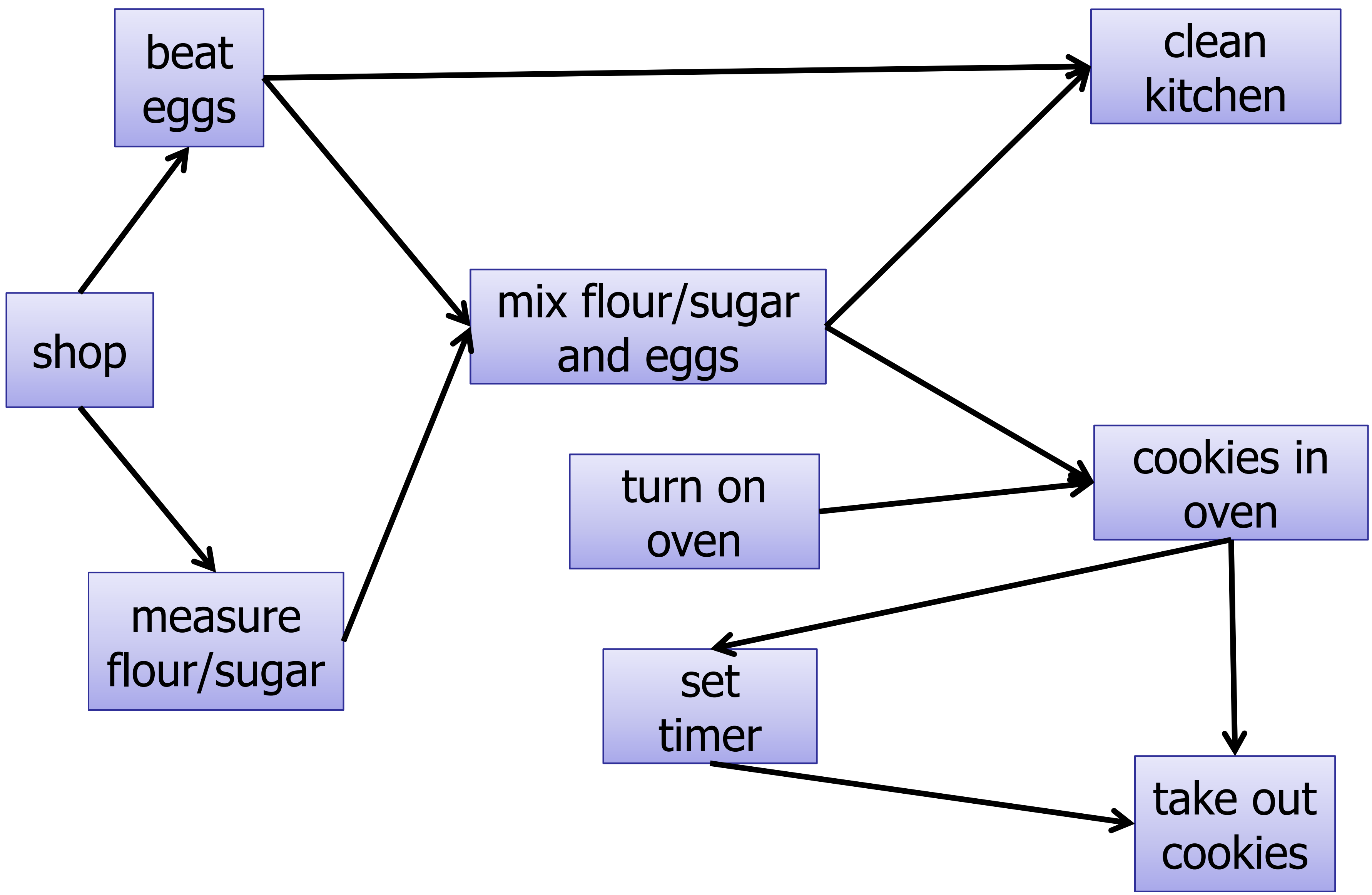
- Shop for groceries
- Put the cookies in the oven
- Clean the kitchen
- Beat the eggs in a bowl
- Measure the flour and sugar in a bowl
- Mix the eggs with the flour and sugar
- Turn on the oven
- Set the timer
- Take out the cookies

# Scheduling

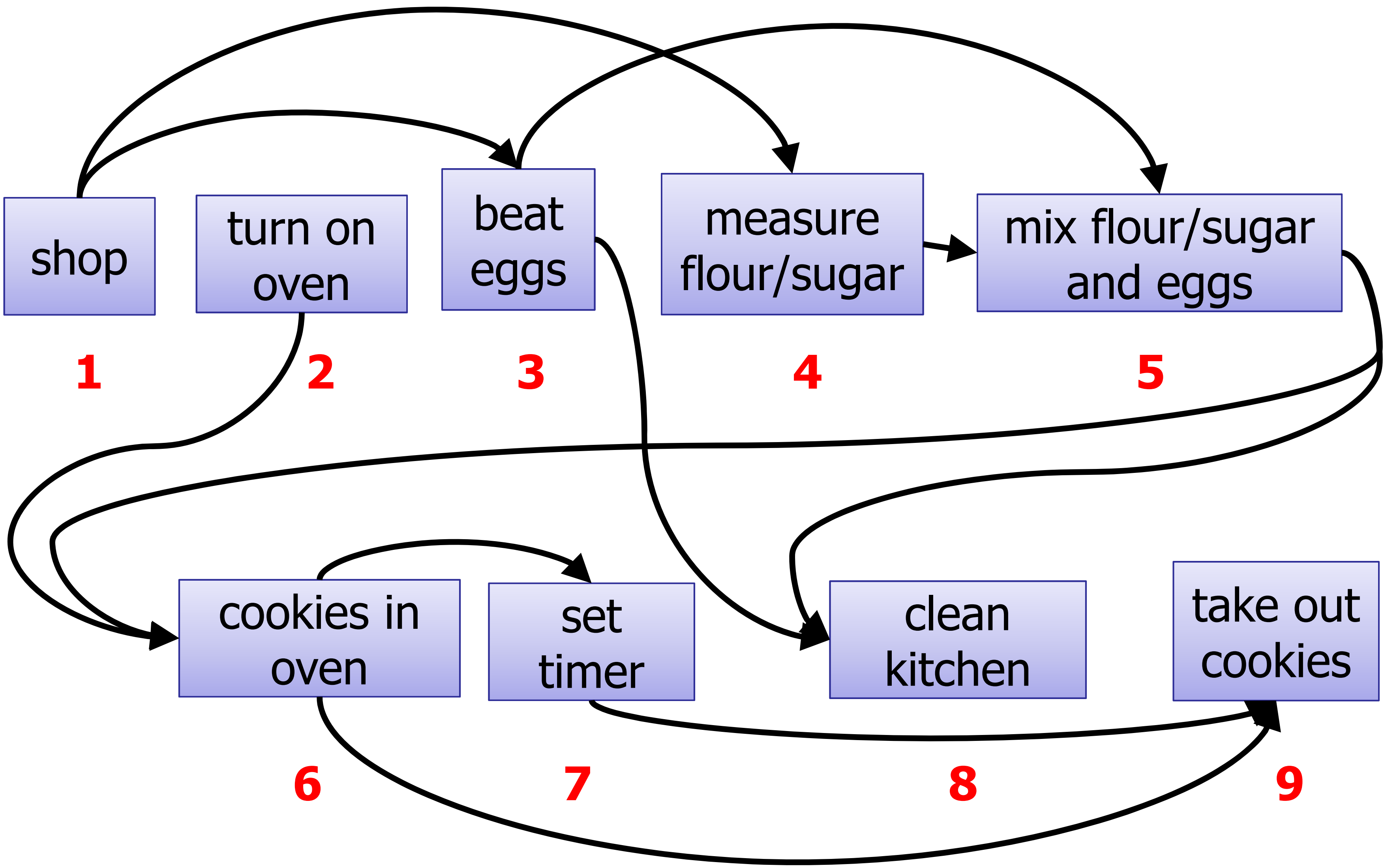
Ordering:

- Shop for groceries before beat the eggs
- Shop for groceries before measure the flour
- Turn on the oven before put the cookies in the oven
- Beat the eggs before mix the eggs with the flour
- Measure the flour before mix the eggs with the flour
- Put the cookies in the oven before set the timer
- Measure the flour before clean the kitchen
- Beat the eggs before clean the kitchen
- Mix the flour and the eggs before clean the kitchen

# Scheduling


beat eggs
clean kitchen
shop
mix flour/sugar and eggs
turn on oven
cookies in oven
measure flour/sugar
set timer
take out cookies

# Topological Ordering



# Topological Order

Properties:

1. Sequential total ordering of all nodes

1. shop | 2. turn on oven | 3. measure flour/sugar | 4. eggs

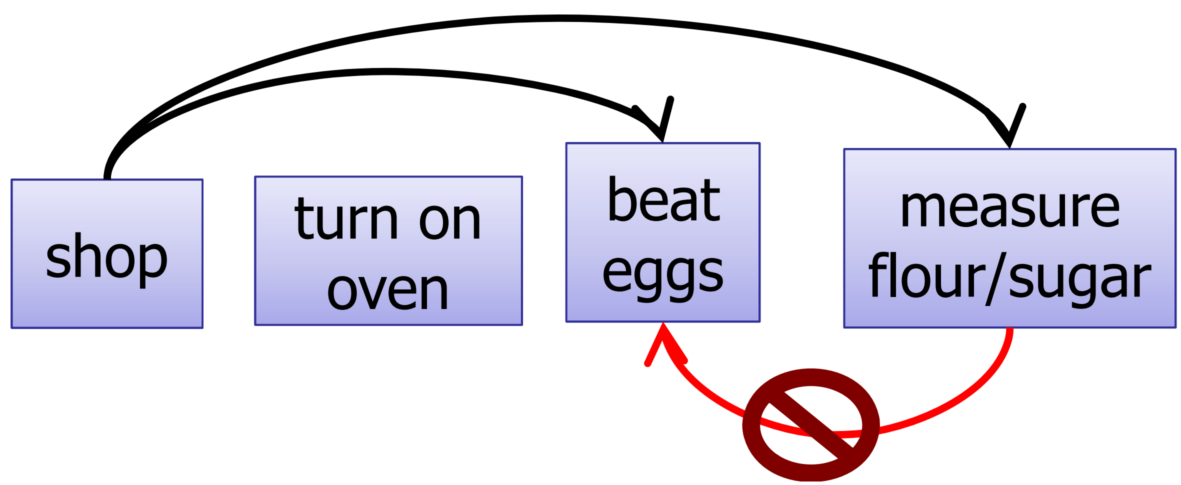
# Topological Order

Properties:

1. Sequential total ordering of all nodes

1. shop | 2. turn on oven | 3. measure flour/sugar | 4. eggs

2. Edges only point forward

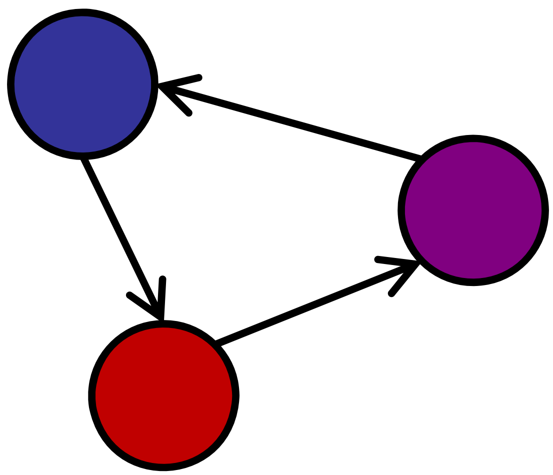


Does every directed graph have a topological ordering?

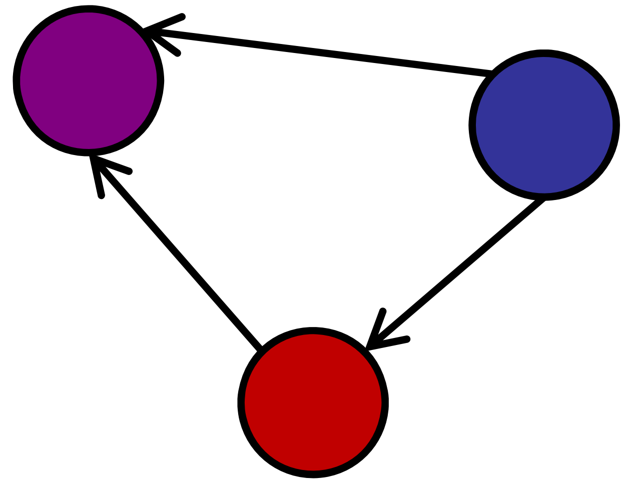
1. Yes
2. No
3. Only if the adjacency matrix has small second eigenvalue.

# Directed Acyclic Graphs

Cyclic

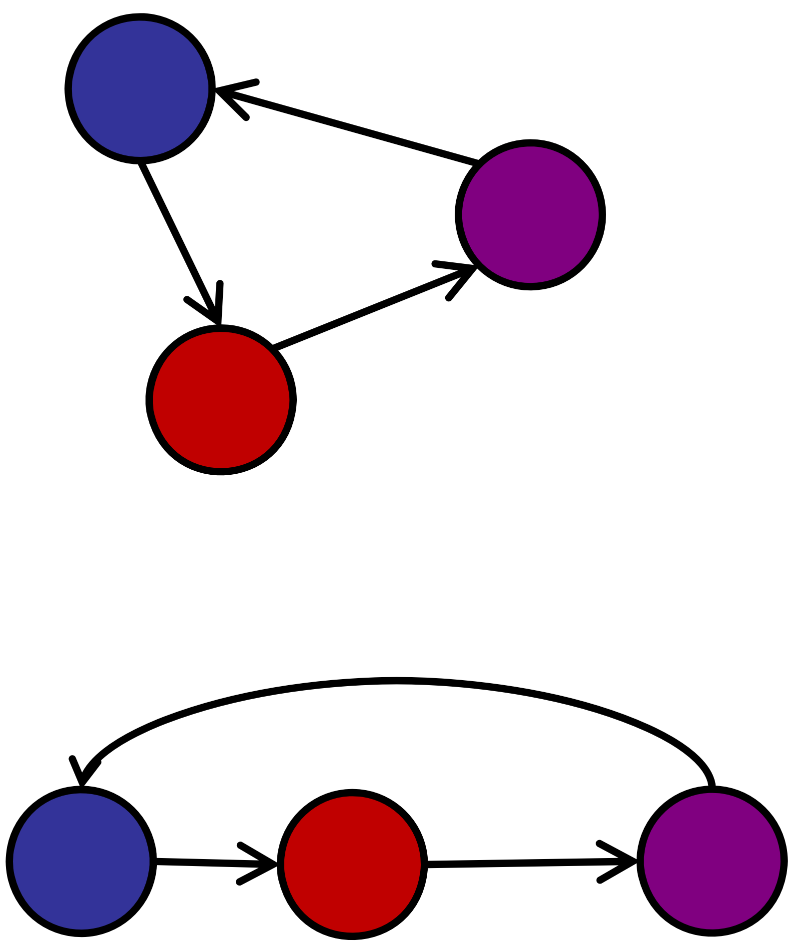


Acyclic

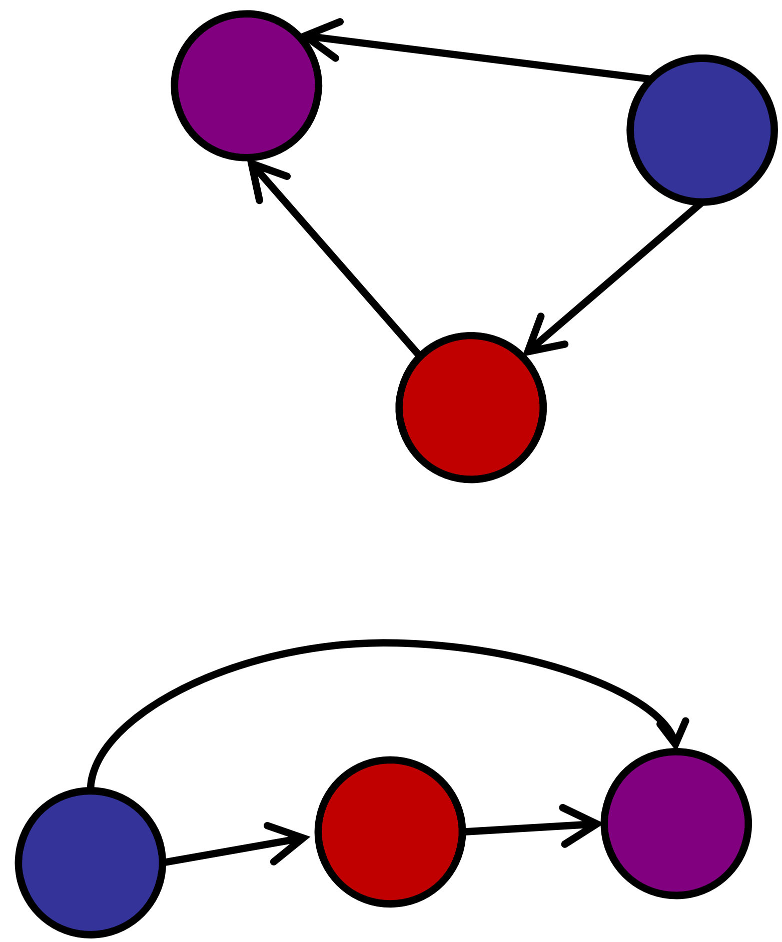


# Directed Acyclic Graphs

## Cyclic

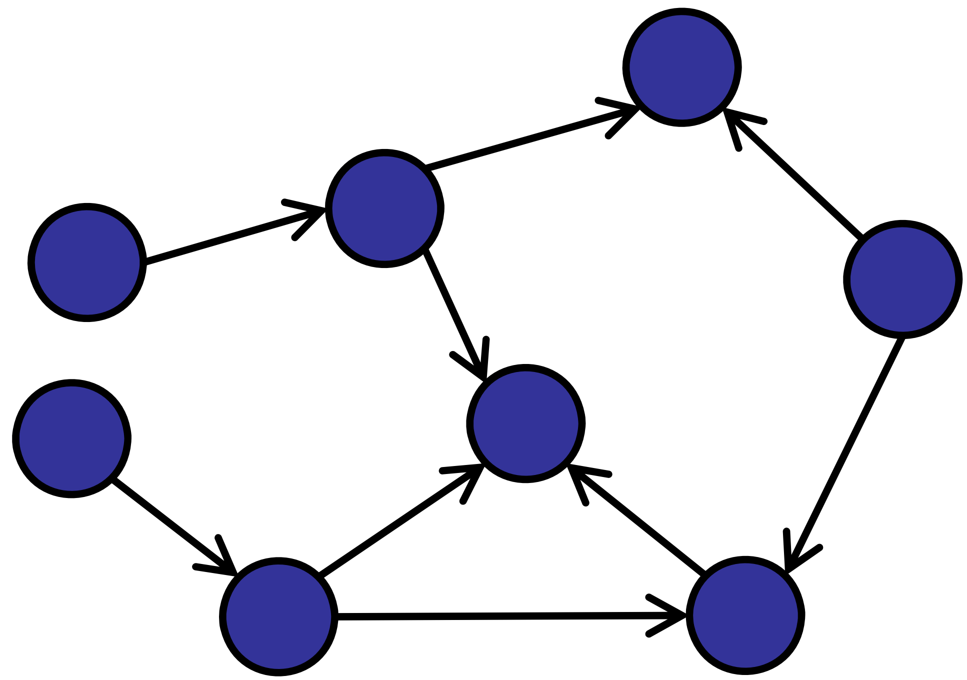


## Acyclic



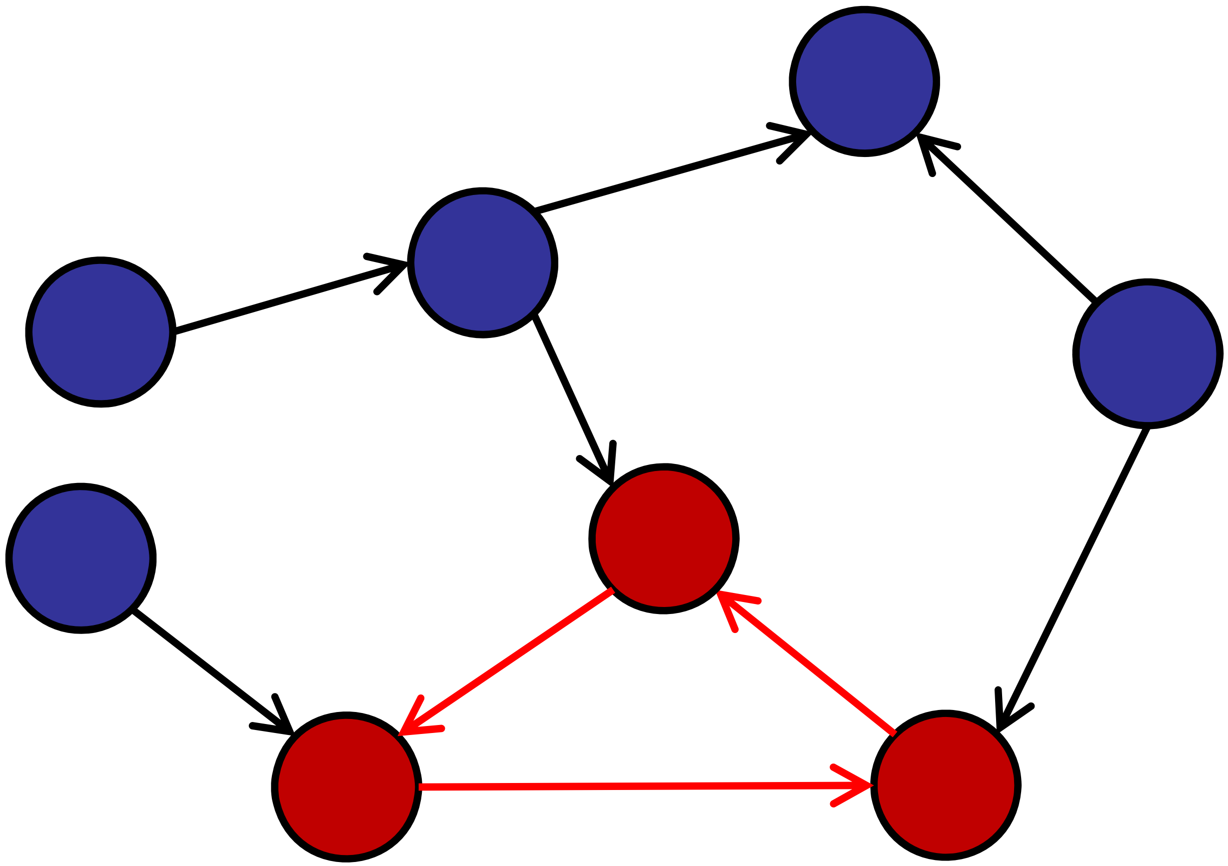
Is this graph:

1. Cyclic
2. Acyclic
3. Transcendental

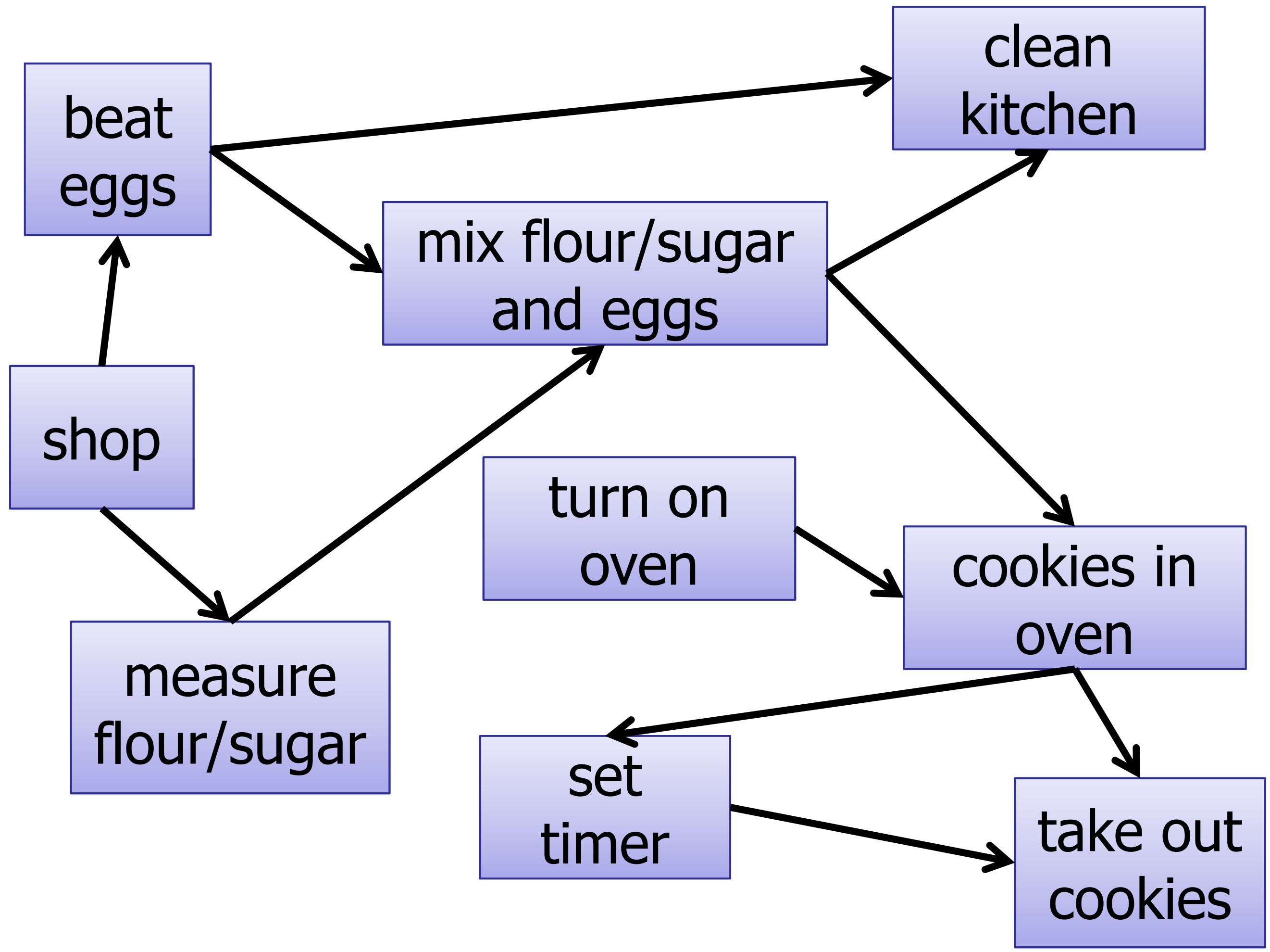


# Directed Acyclic Graphs

Does it have a topological ordering?



# Directed Acyclic Graph



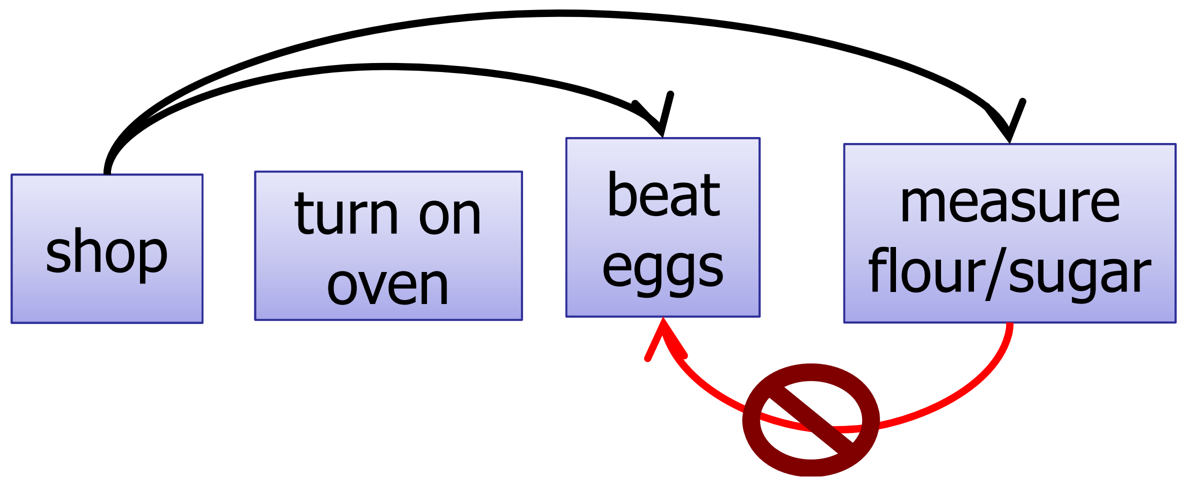
# Topological Order

Properties:

1. Sequential total ordering of all nodes

1. shop | 2. turn on oven | 3. measure flour/sugar | 4. eggs

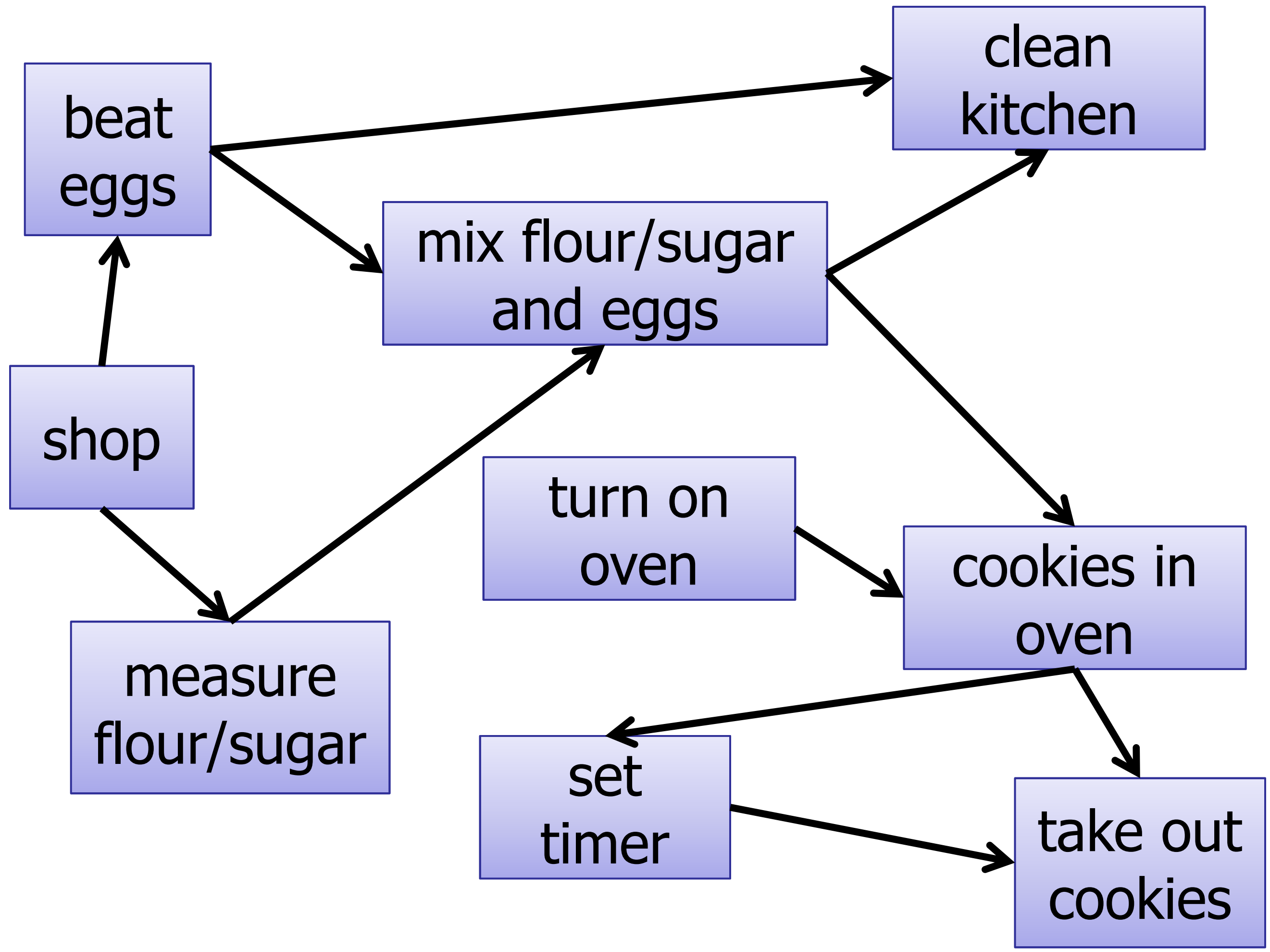
2. Edges only point forward



## Which algorithm is best for finding a Topological Ordering in a DAG?

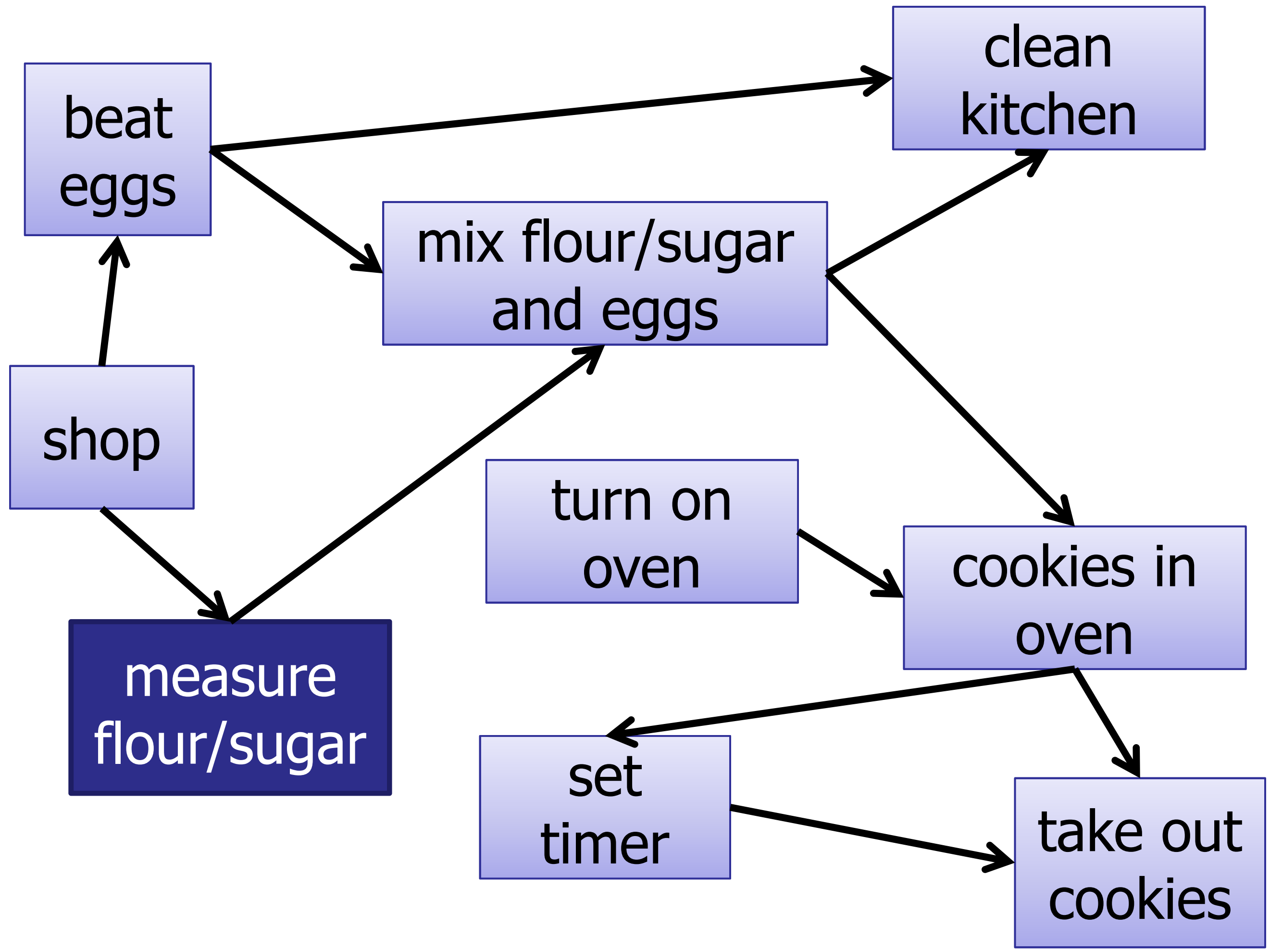
1. Breadth-first search
2. Depth-first search
3. Bloom Filter
4. Karatsuba algorithm
5. Something else

# Depth-First Search



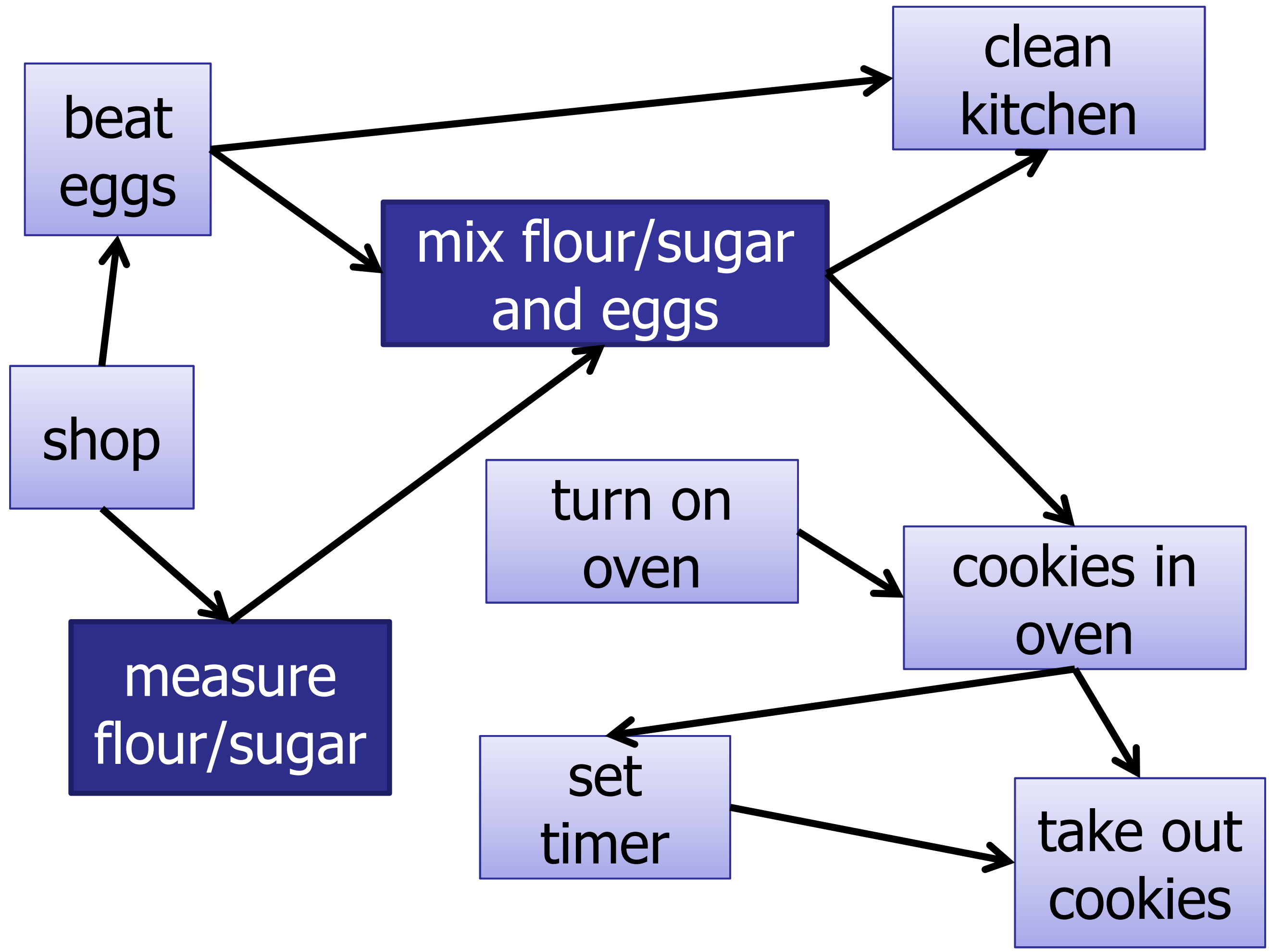
# Depth-First Search

1. measure



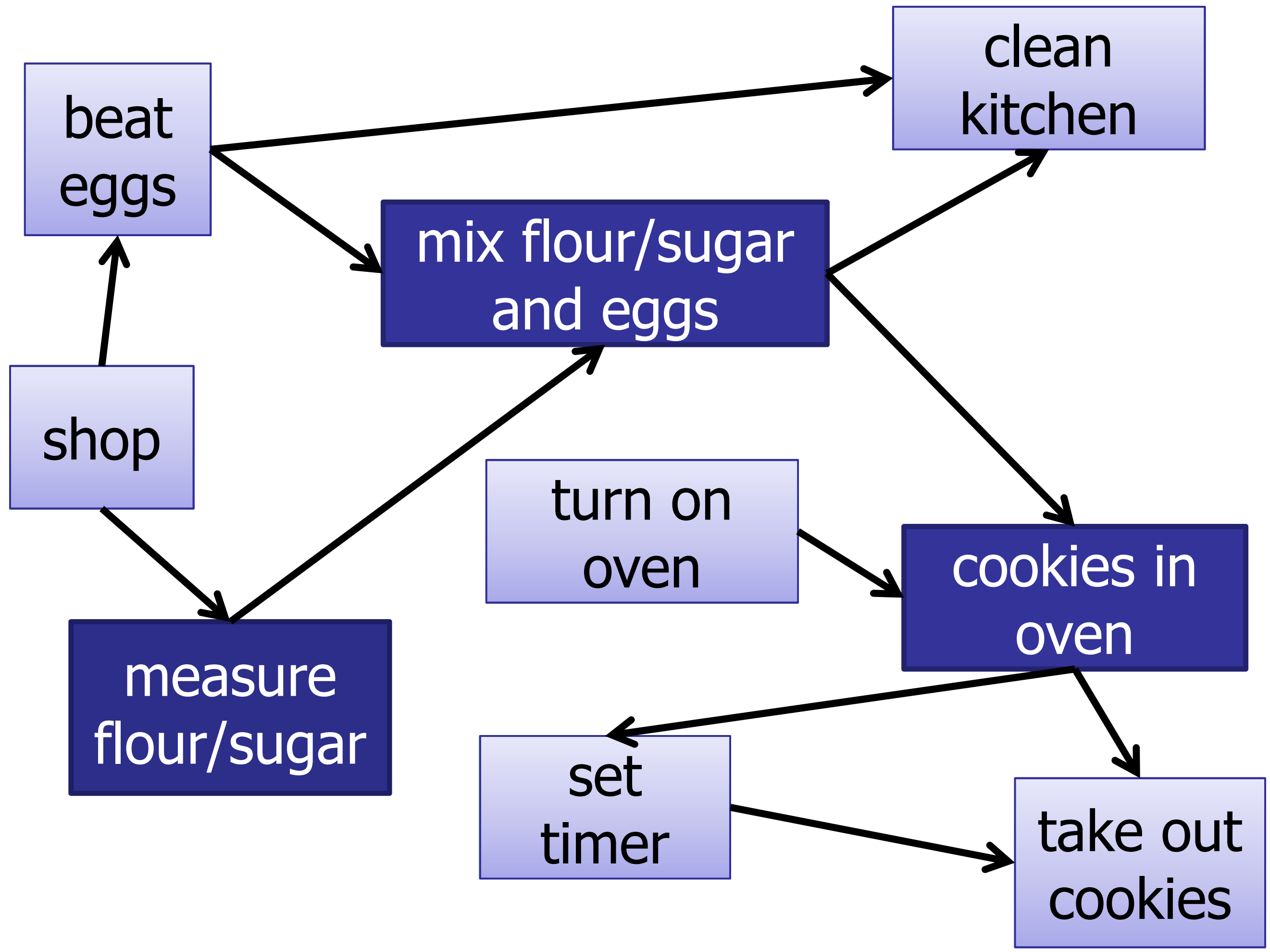
# Depth-First Search

1. measure
2. mix



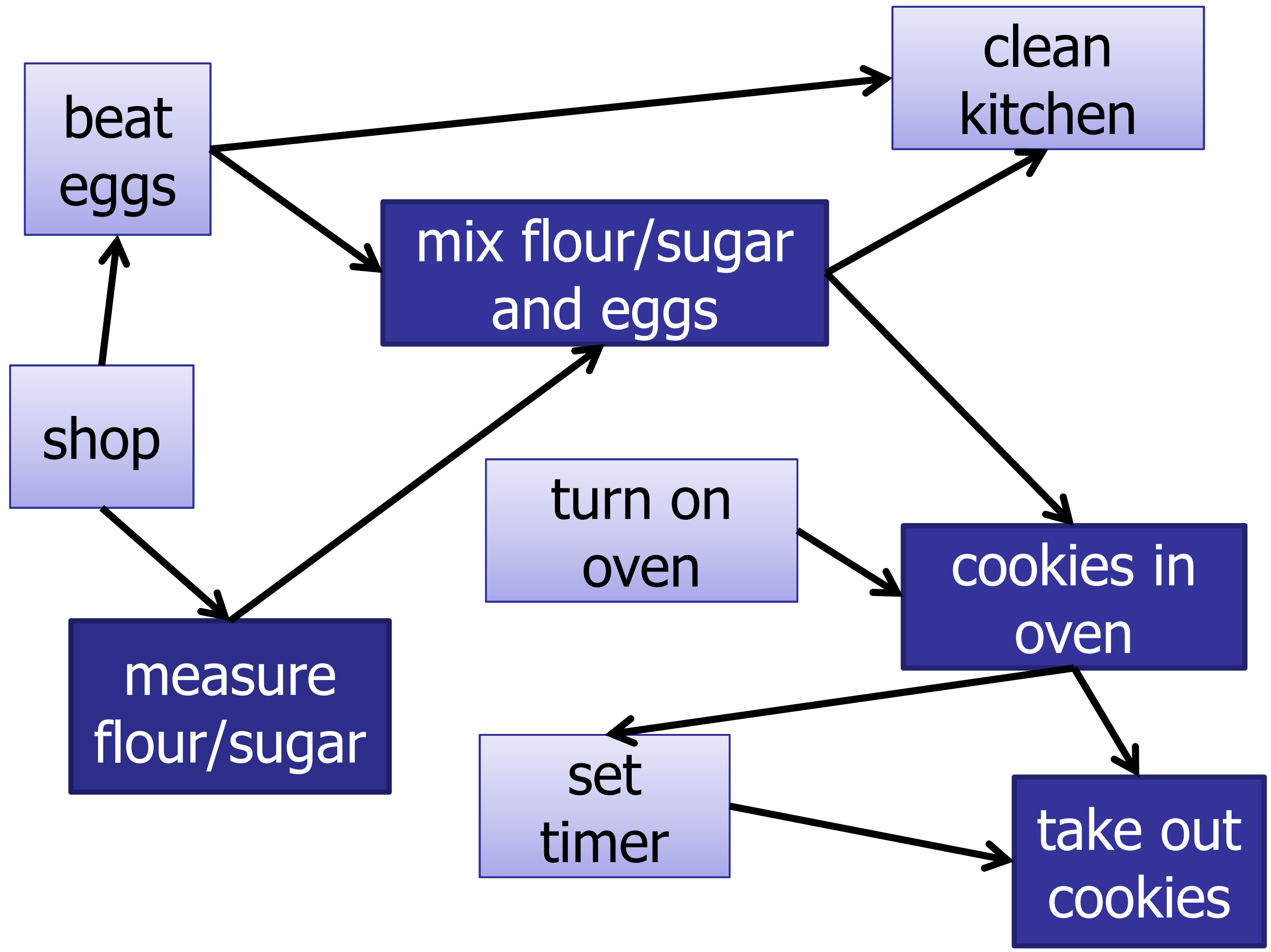
# Depth-First Search

1. measure
2. mix
3. in oven



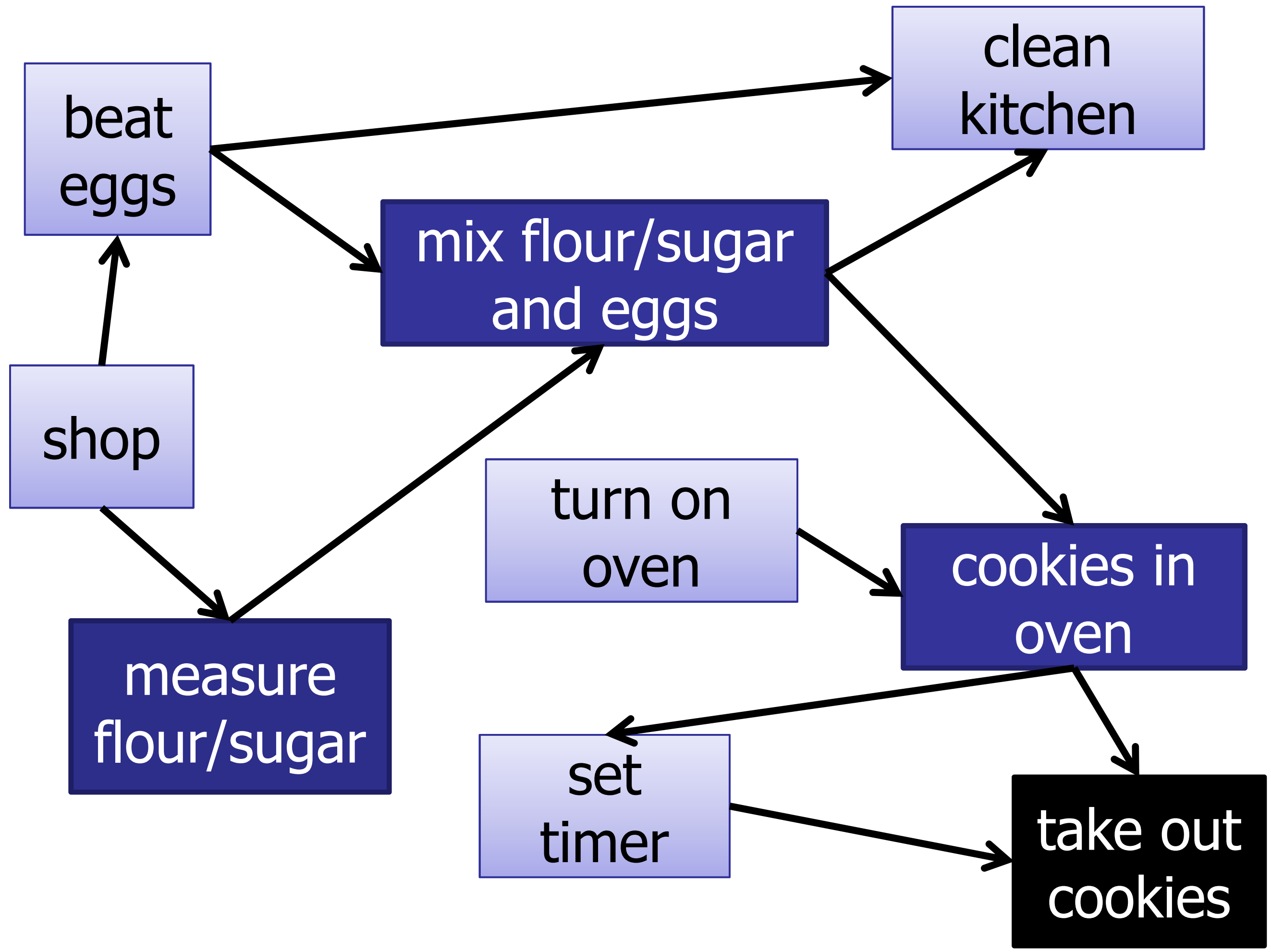
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out



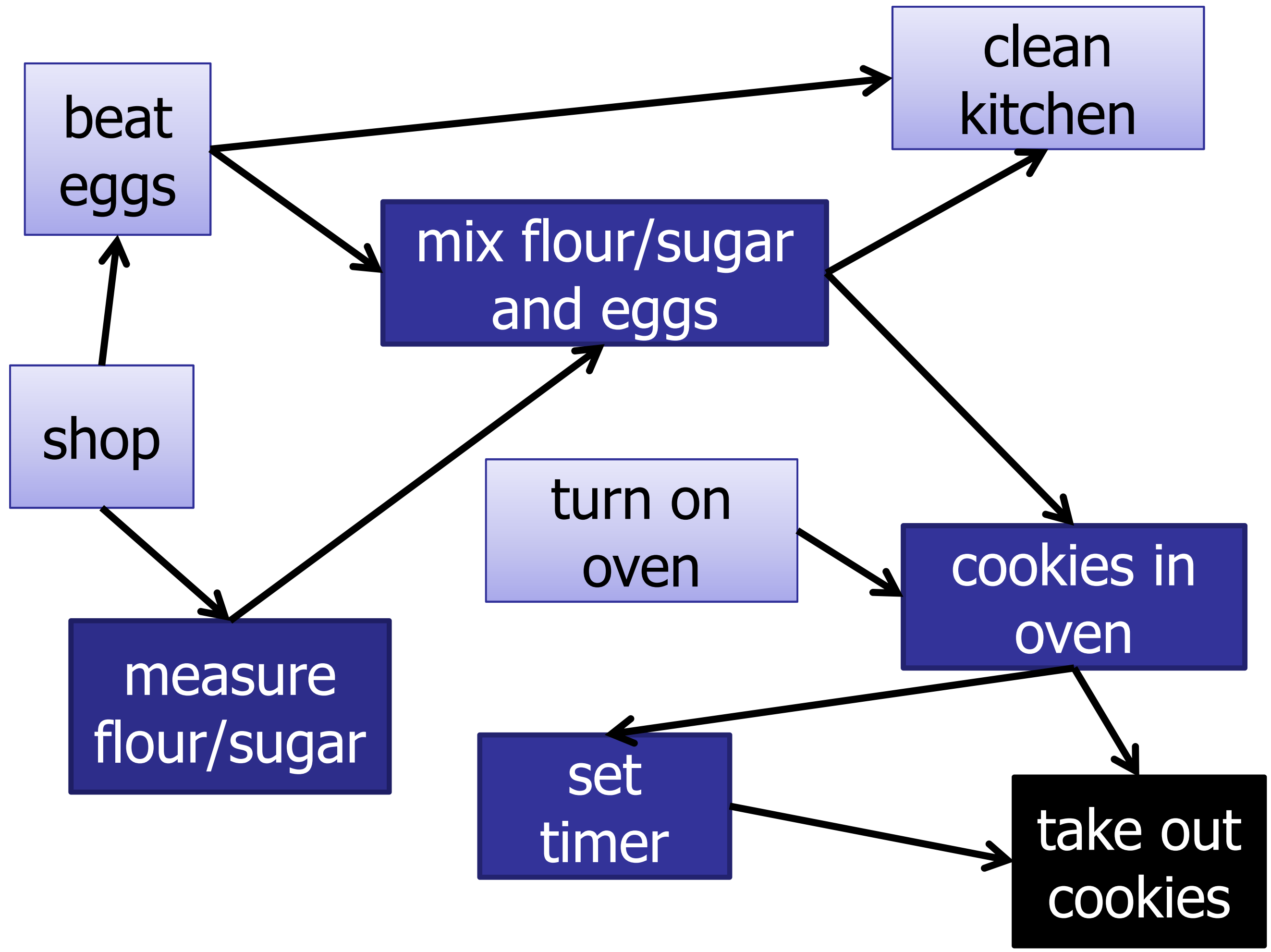
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out



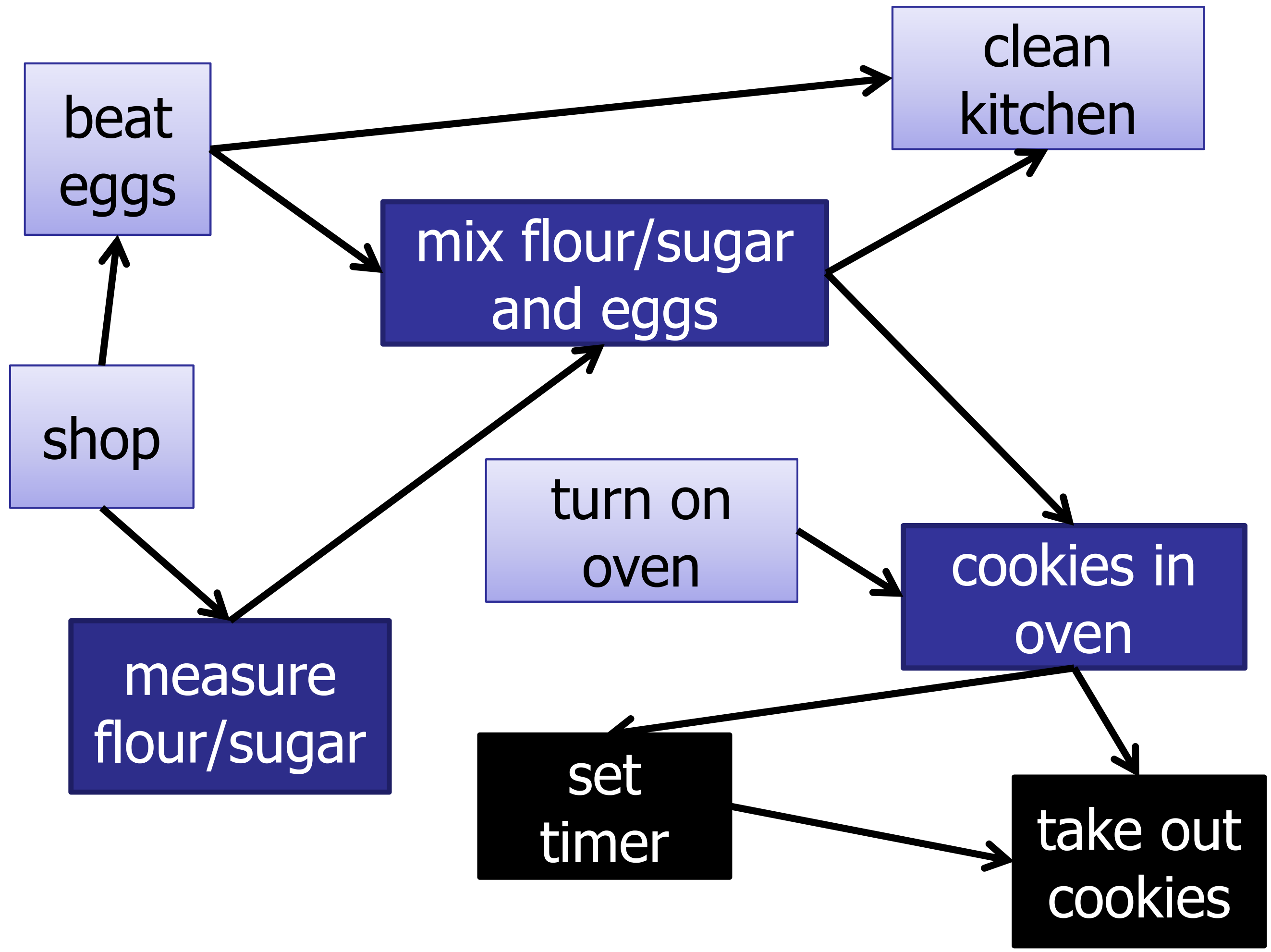
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer



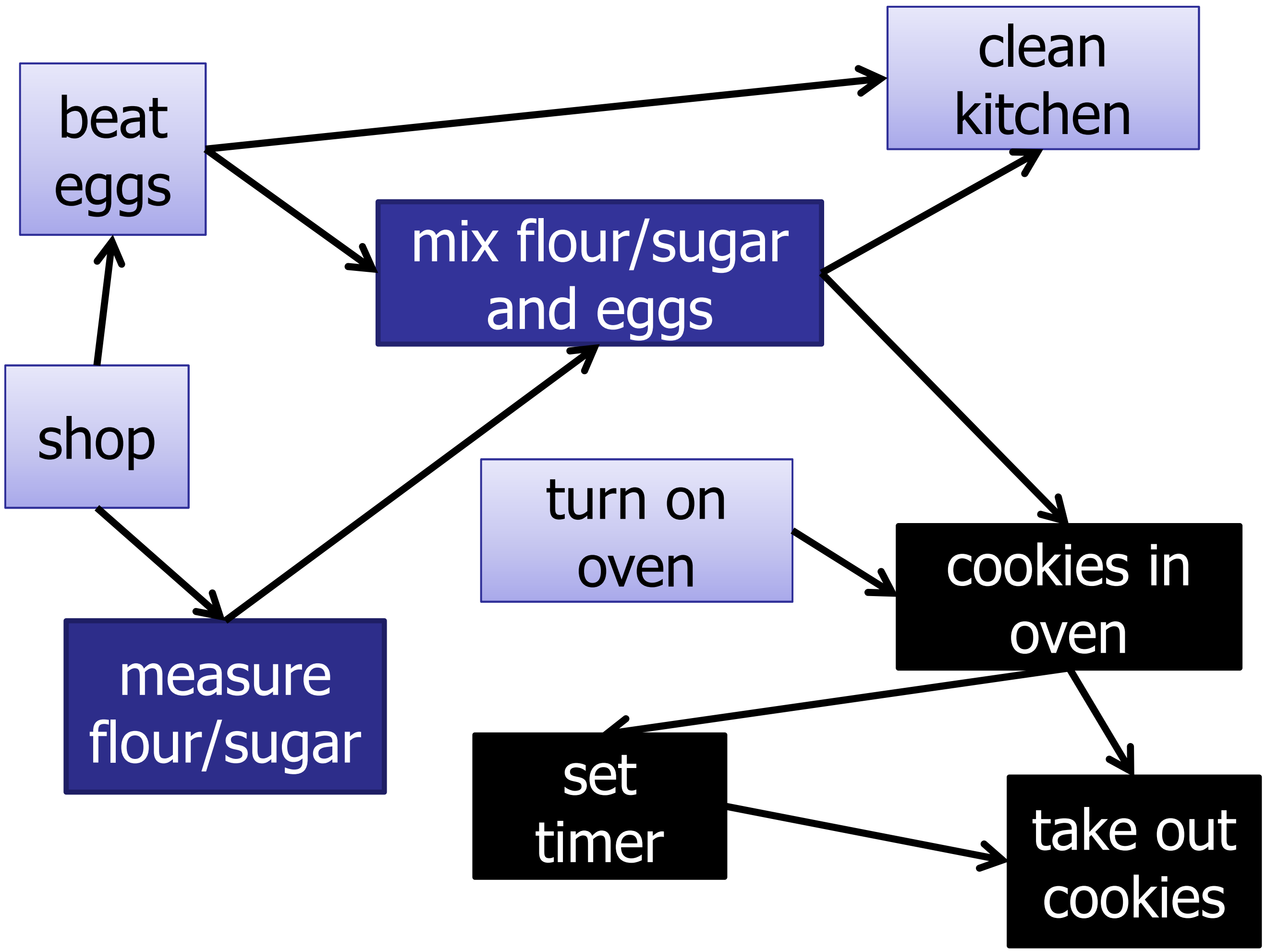
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer



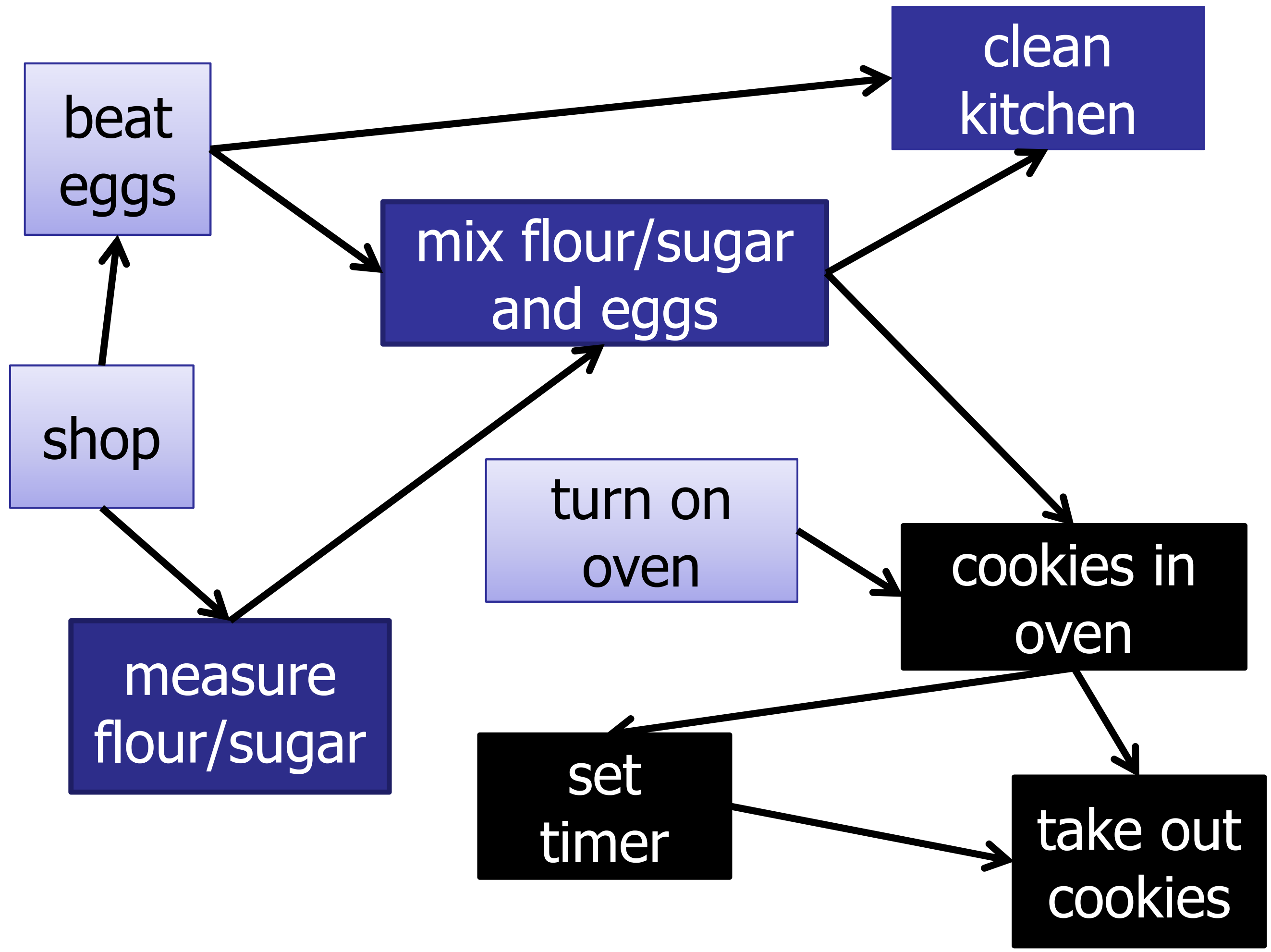
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer



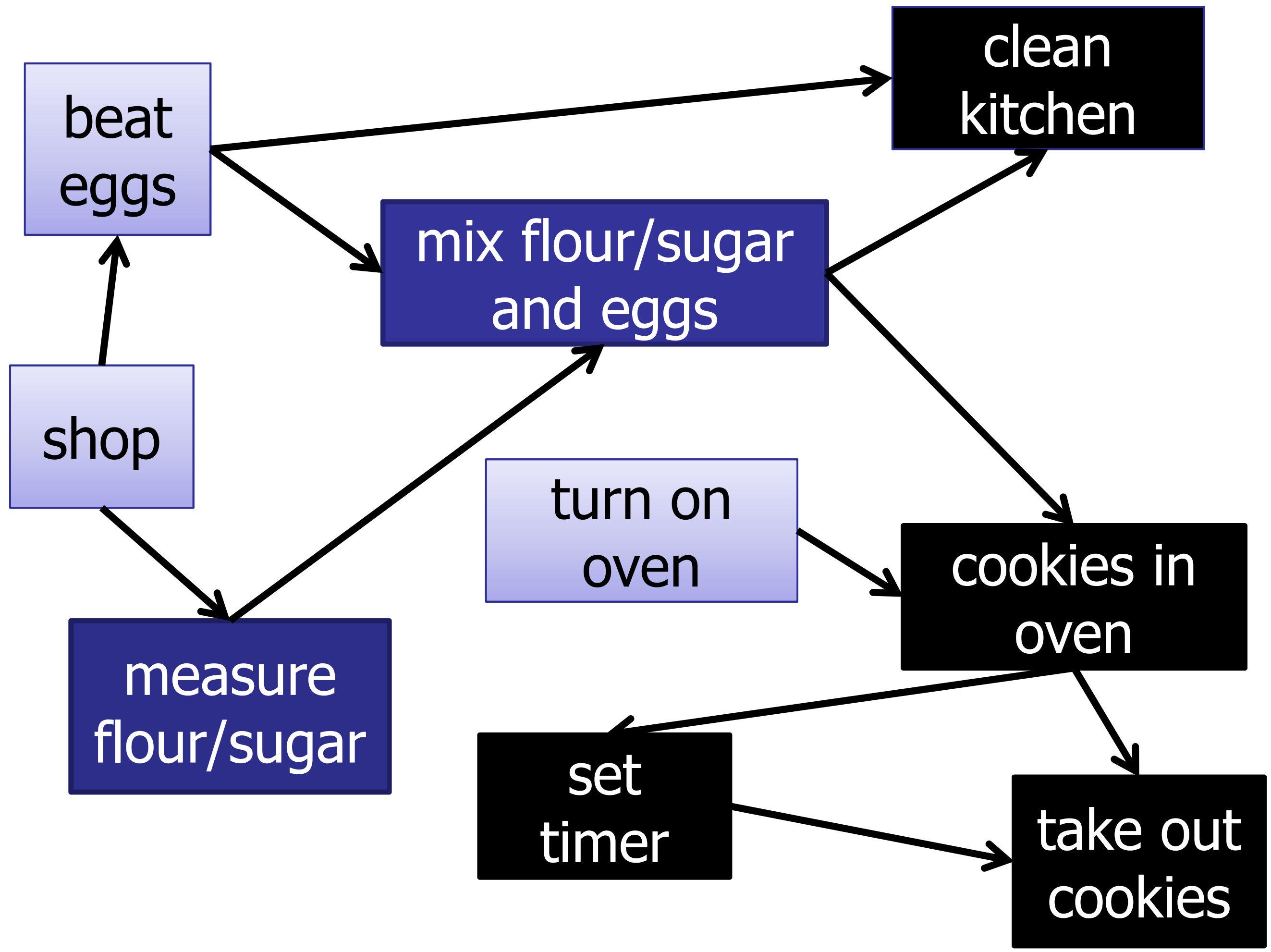
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



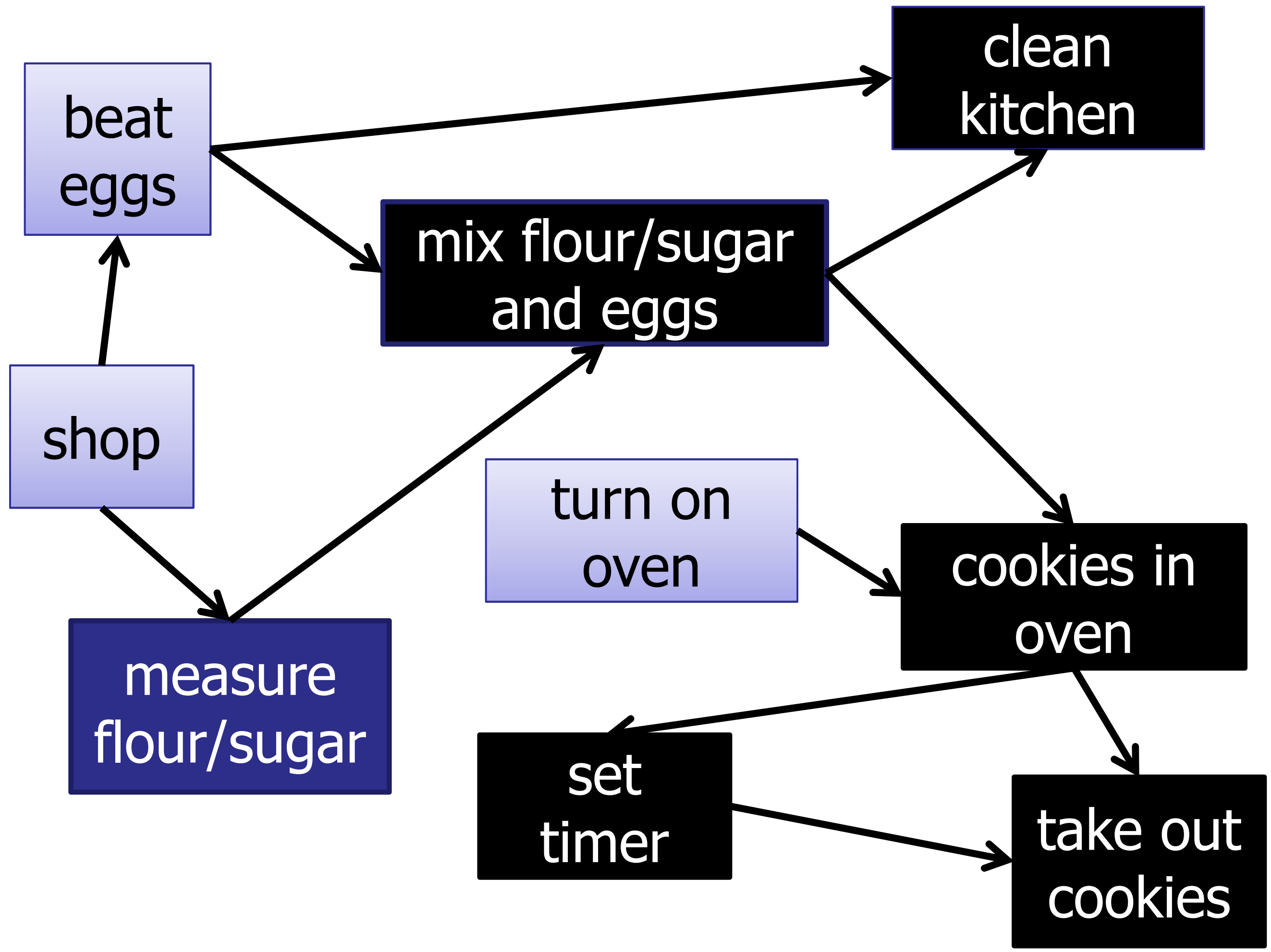
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



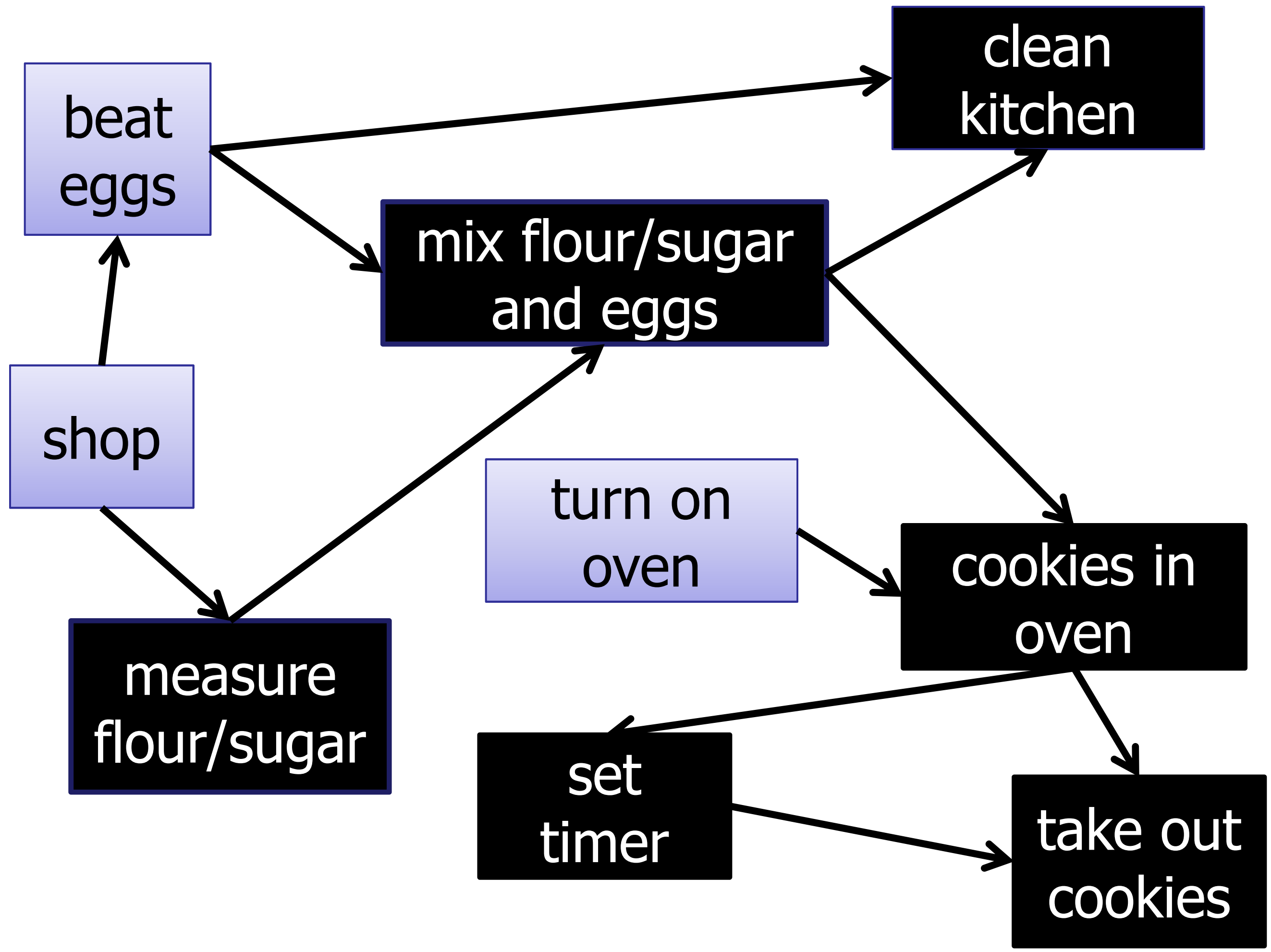
# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



# Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



# Searching a (Directed) Graph

**Pre-Order** Depth-First Search:

– Process each node when it is *first* visited.

# Searching a (Directed) Graph

**Pre-Order** Depth-First Search:

– Process each node when it is *first* visited.

**Post-Order** Depth-First Search:

– Process each node when it is *last* visited.

# DFS: Pre-Order

```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (Integer v : nodeList[startId].nbrList) {
     if (!visited[v]){
          visited[v] = true;
          ProcessNode(v);
          DFS-visit(nodeList, visited, v);
     }
  }
}
```

# DFS Post-Order

```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (Integer v : nodeList[startId].nbrList) {
     if (!visited[v]){
           visited[v] = true;
           DFS-visit(nodeList, visited, v);
           ProcessNode(v);
     }
  }
}
```

# Searching a (Directed) Graph

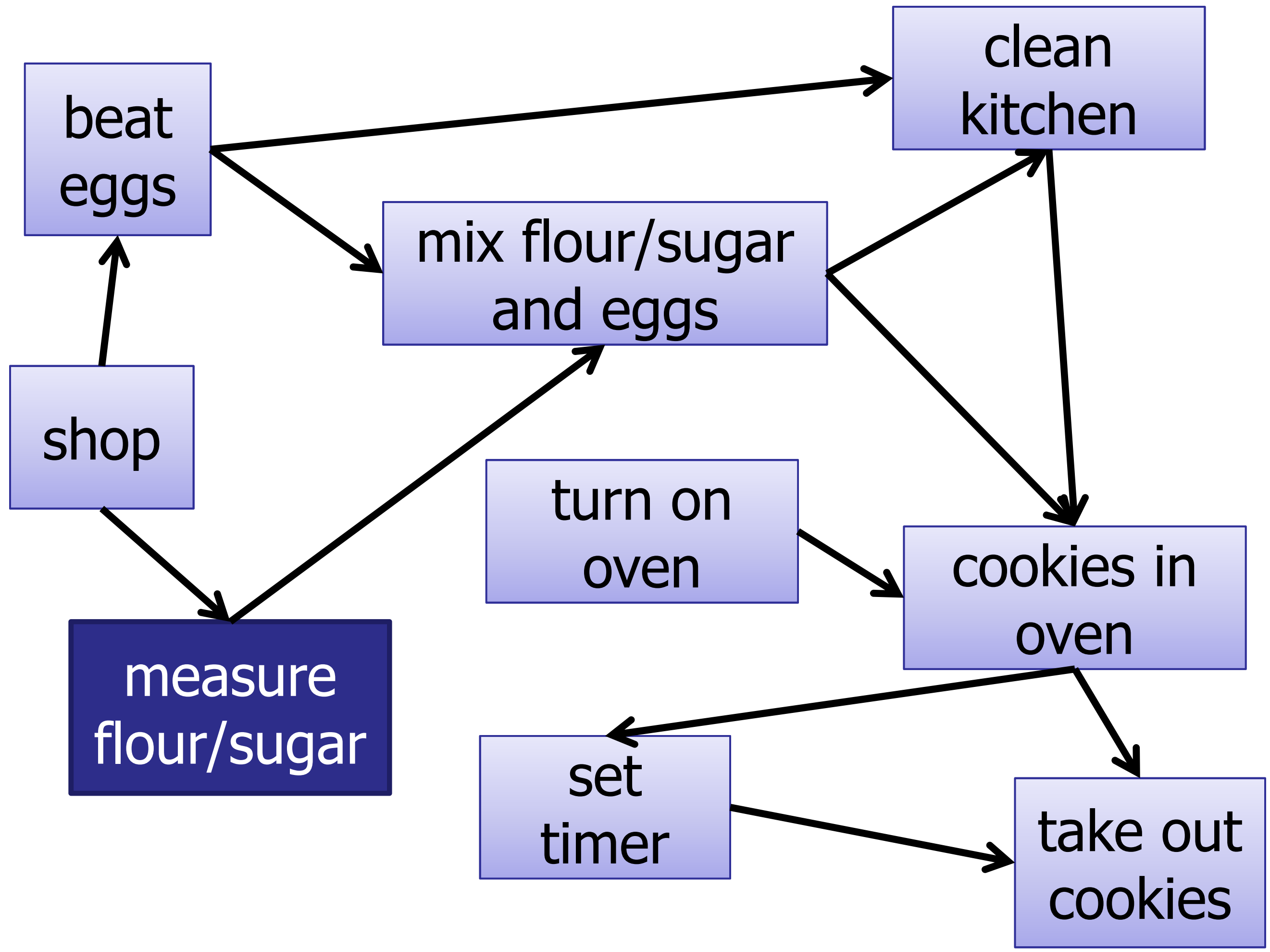
**Pre-Order** Depth-First Search:

- Process each node when it is *first* visited.

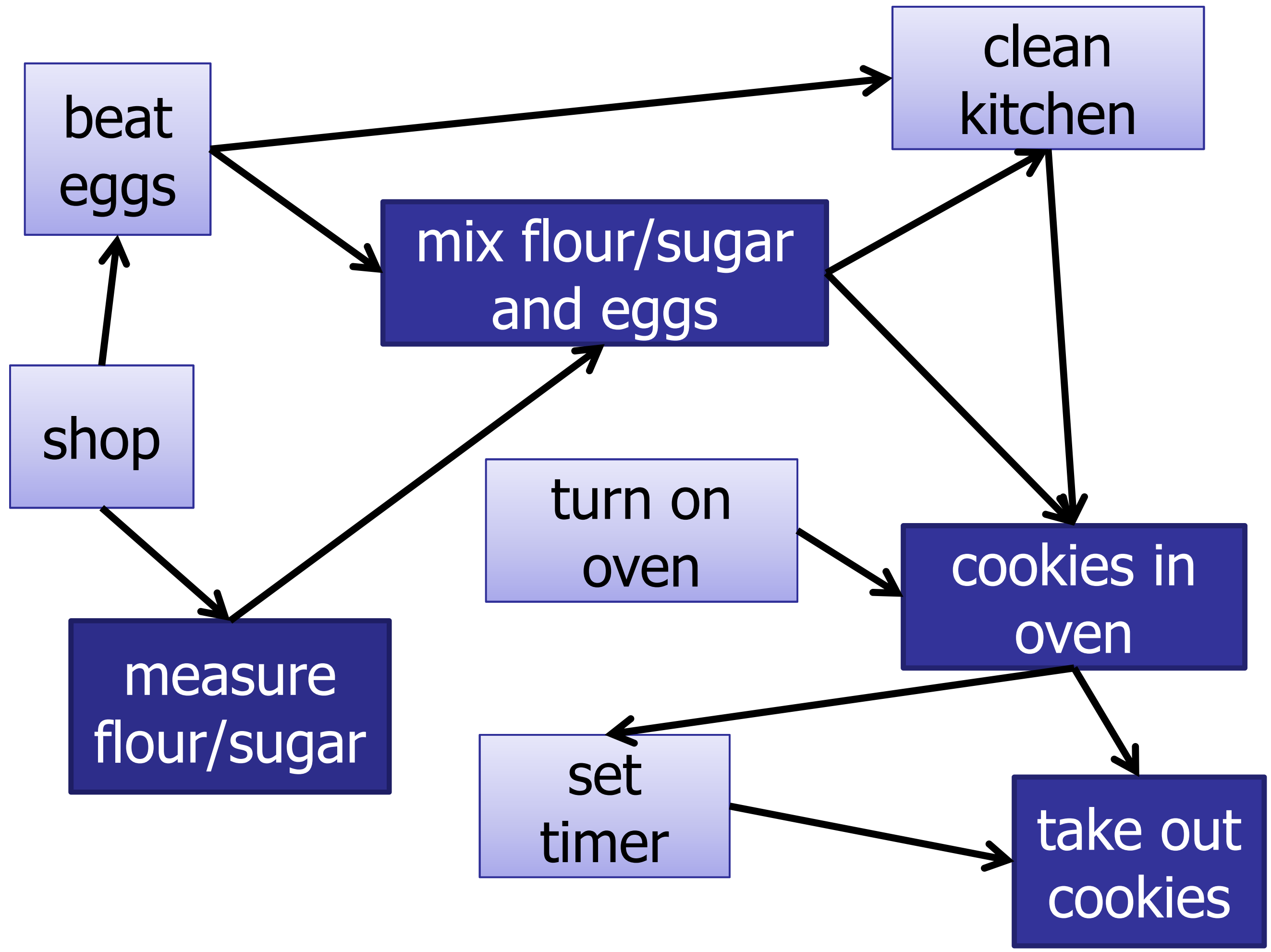
**Post-Order** Depth-First Search:

- Process each node when it is *last* visited.

# Post-Order Depth-First Search

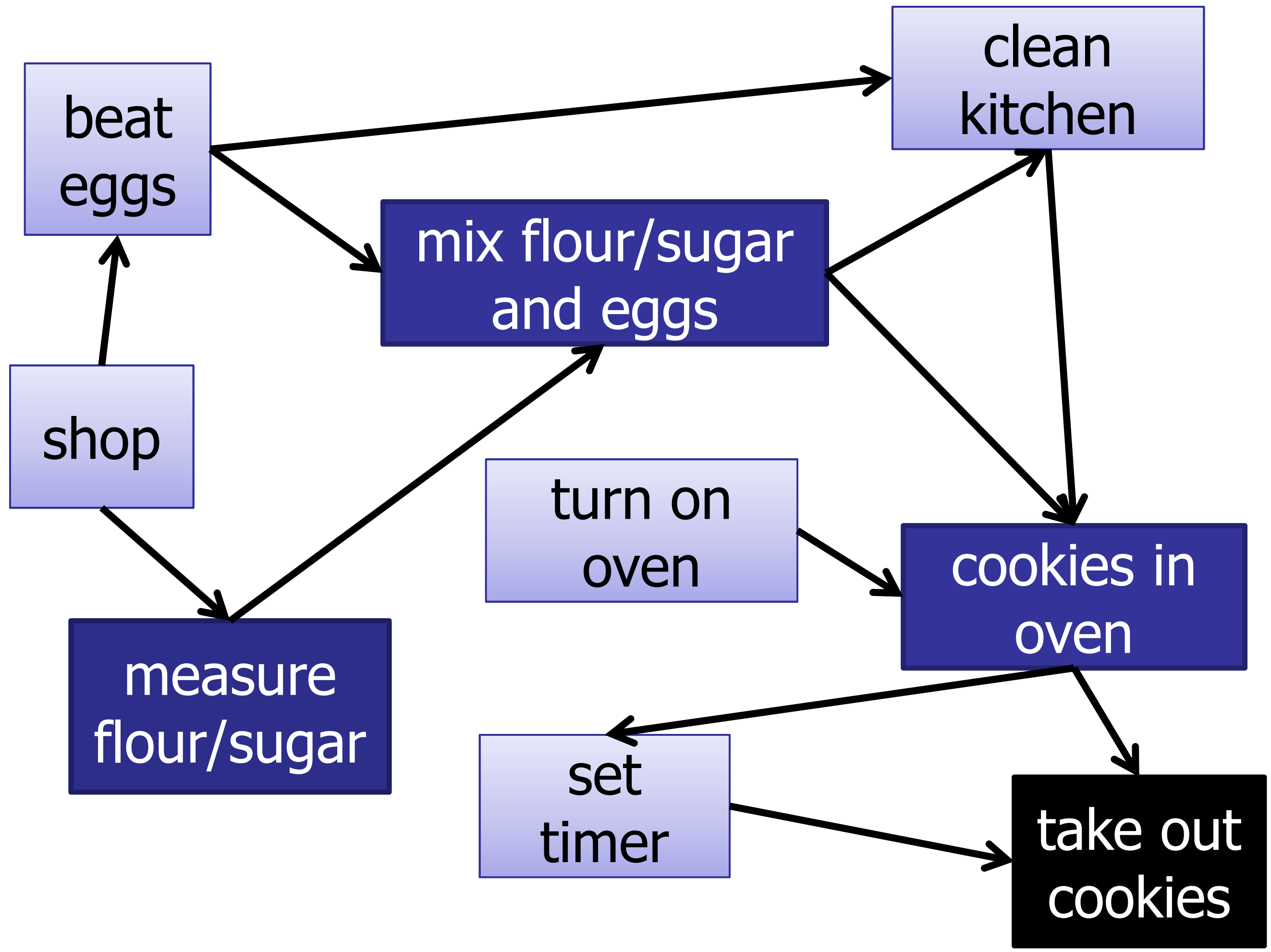


# Post-Order Depth-First Search



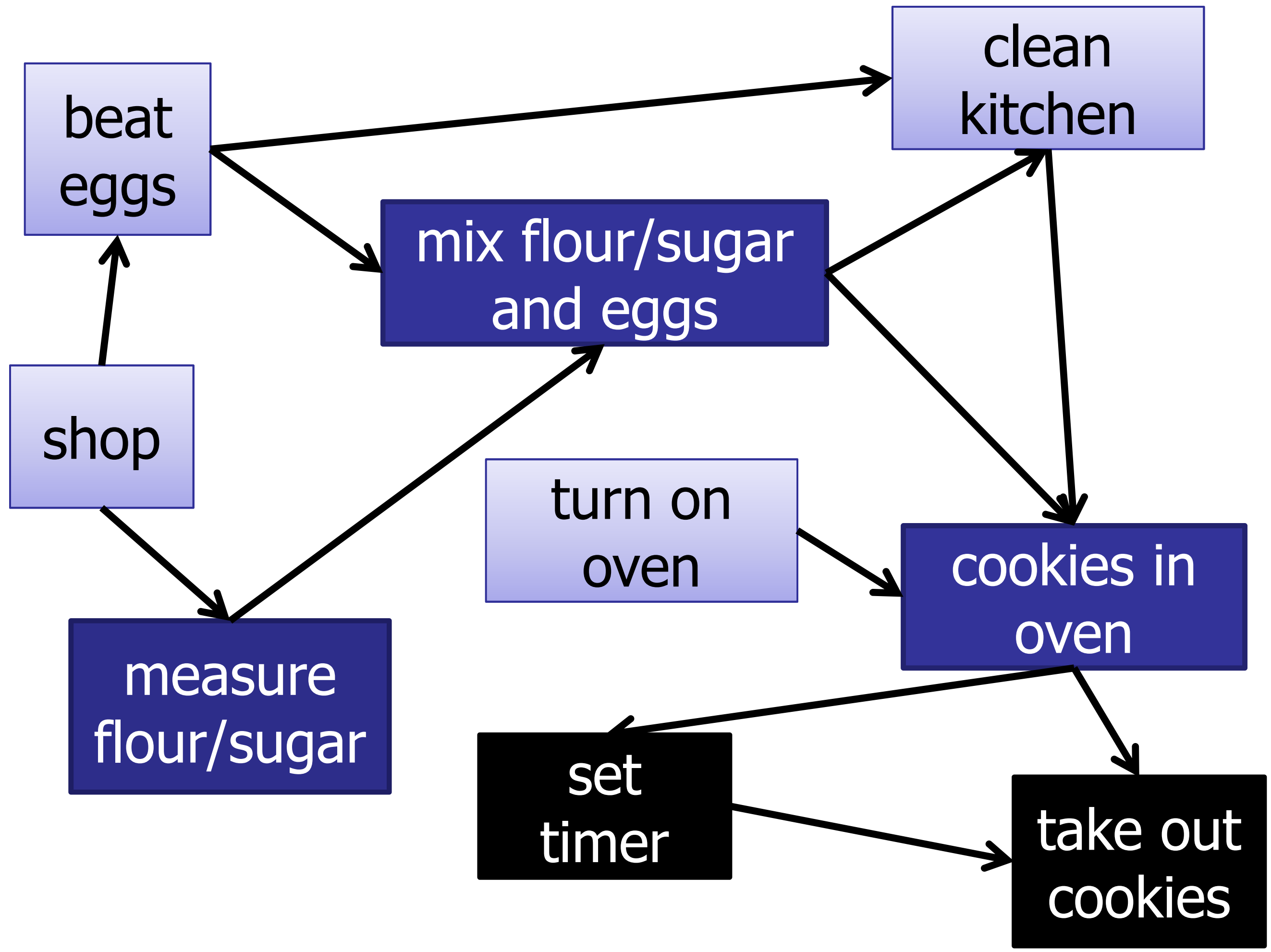
# Post-Order Depth-First Search

1.
2.
3.
4.
5.
6.
7.
8.
9. take out



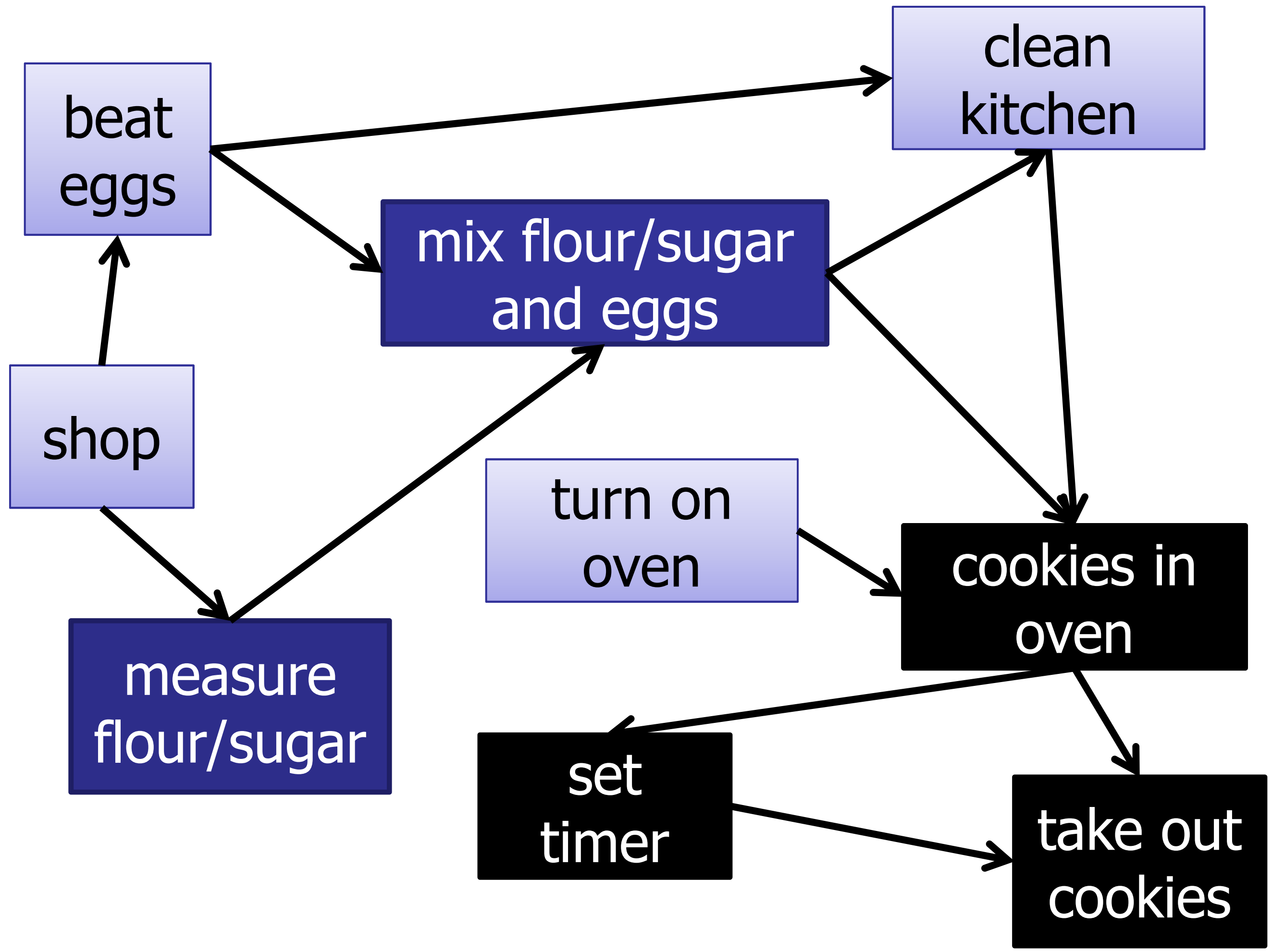
# Post-Order Depth-First Search

1.
2.
3.
4.
5.
6.
7.
8. set timer
9. take out



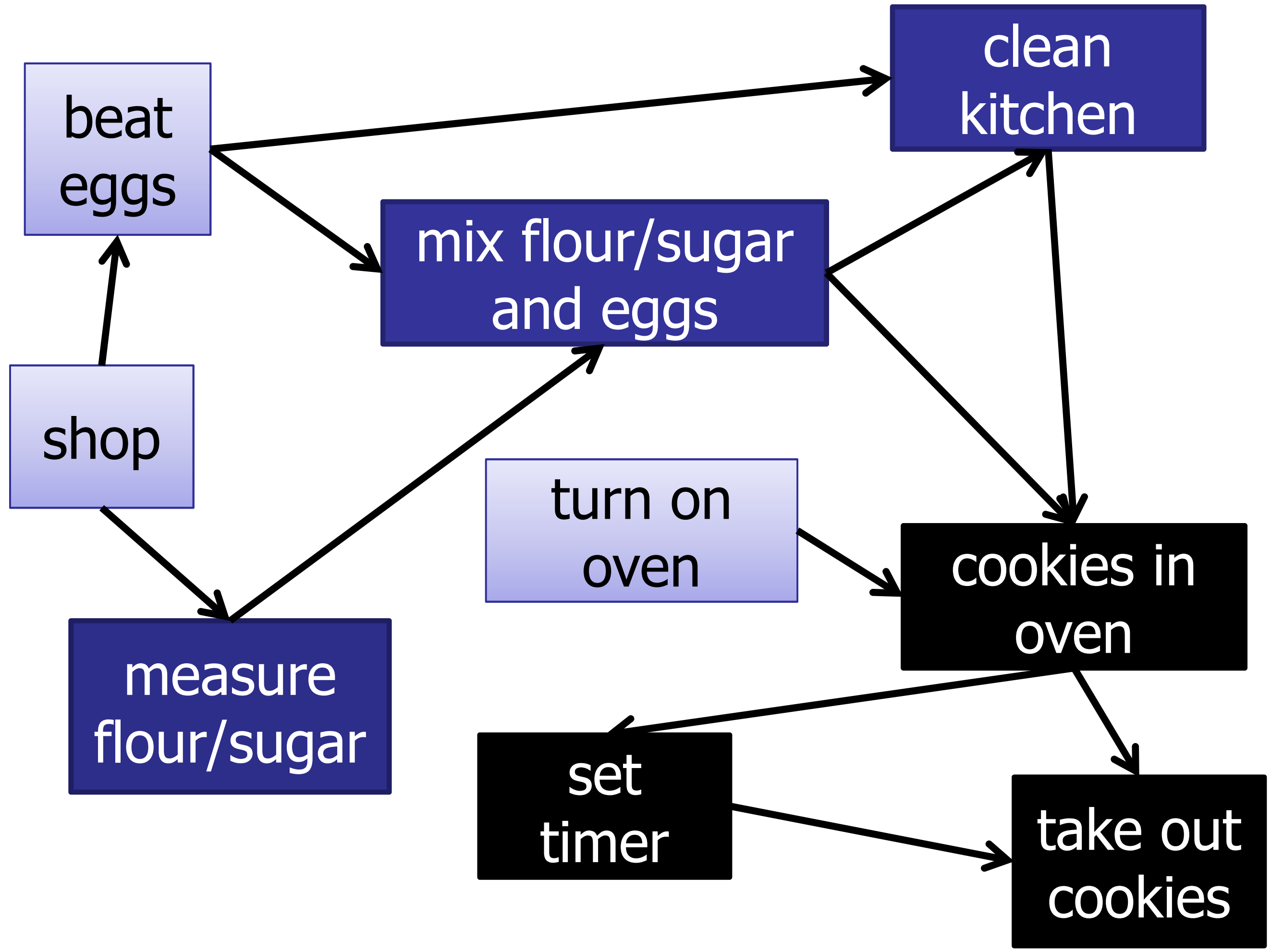
# Post-Order Depth-First Search

1.
2.
3.
4.
5.
6.
7. in oven
8. set timer
9. take out



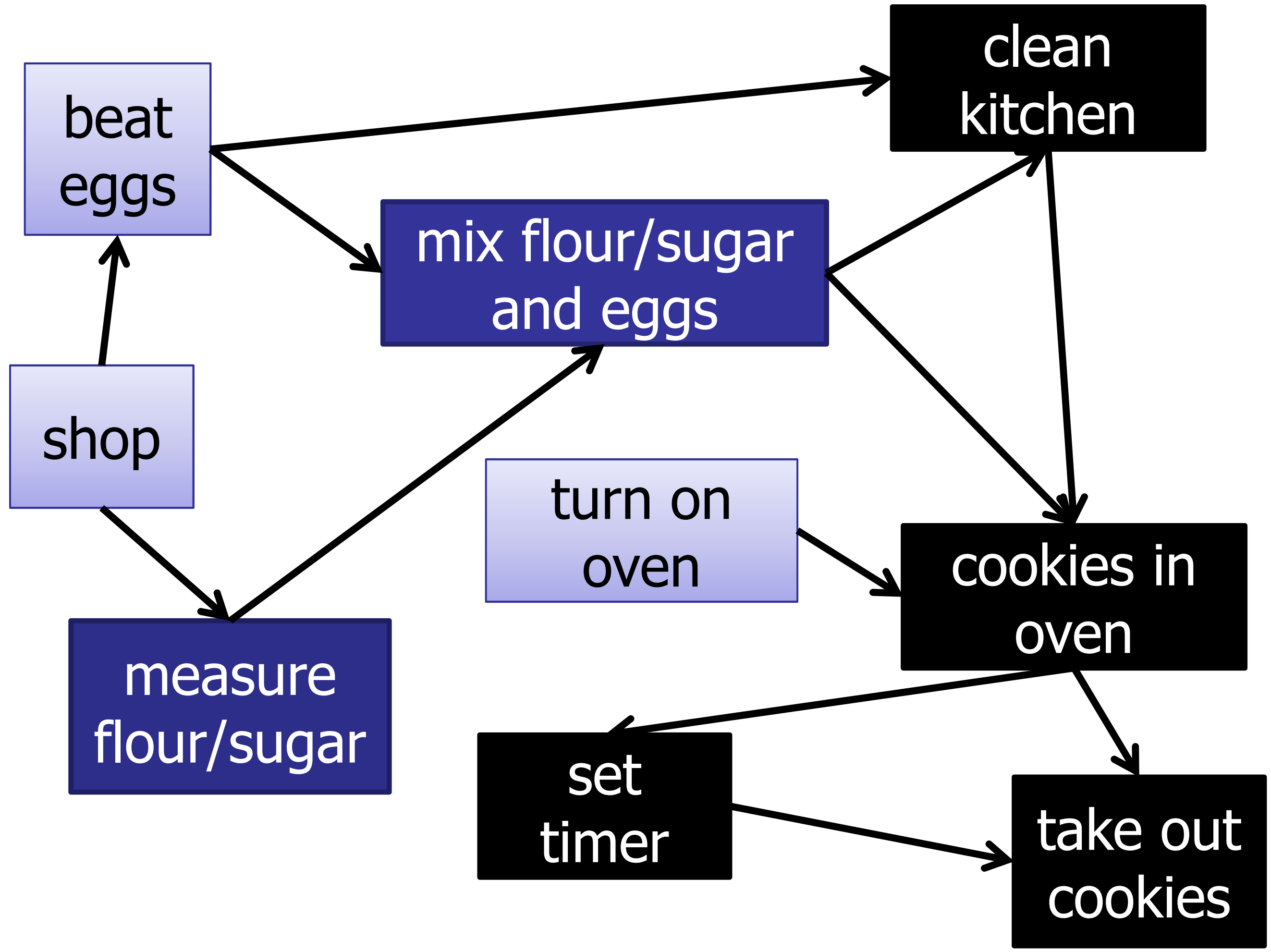
# Post-Order Depth-First Search

1.
2.
3.
4.
5.
6.
7. in oven
8. set timer
9. take out



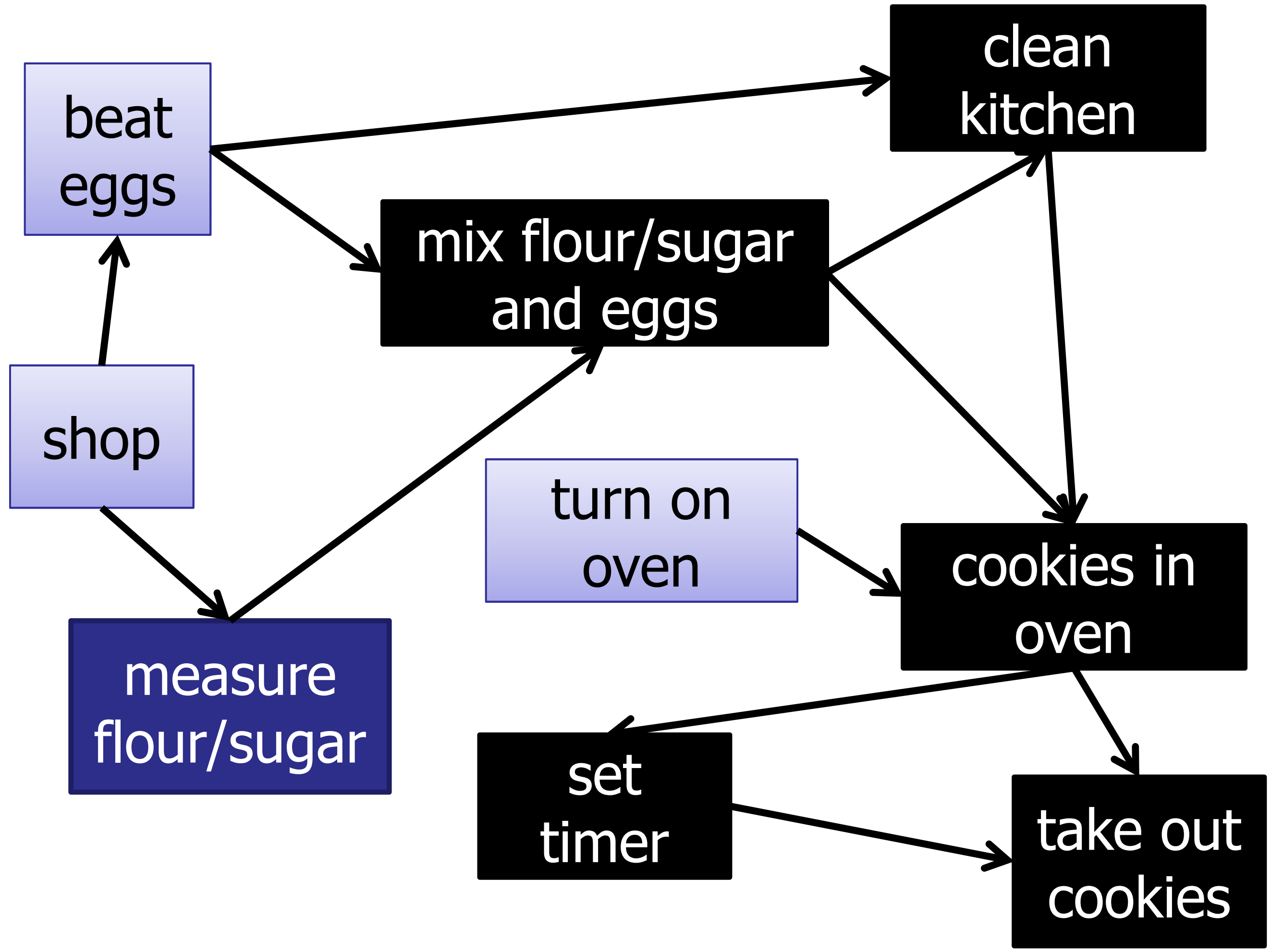
# Post-Order Depth-First Search

1.
2.
3.
4.
5.
6. clean
7. in oven
8. set timer
9. take out



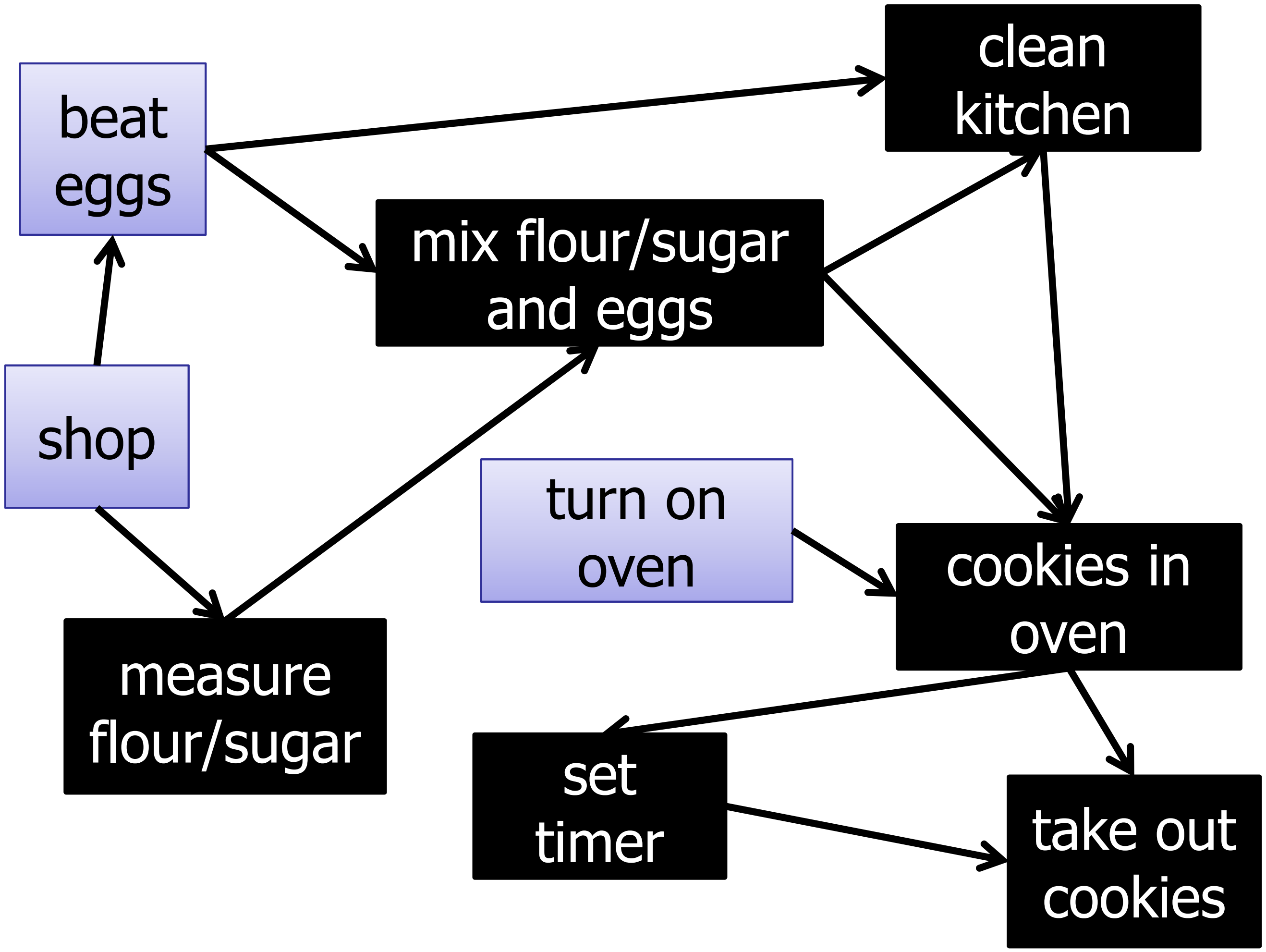
# Post-Order Depth-First Search

1.
2.
3.
4.
5. mix
6. clean
7. in oven
8. set timer
9. take out



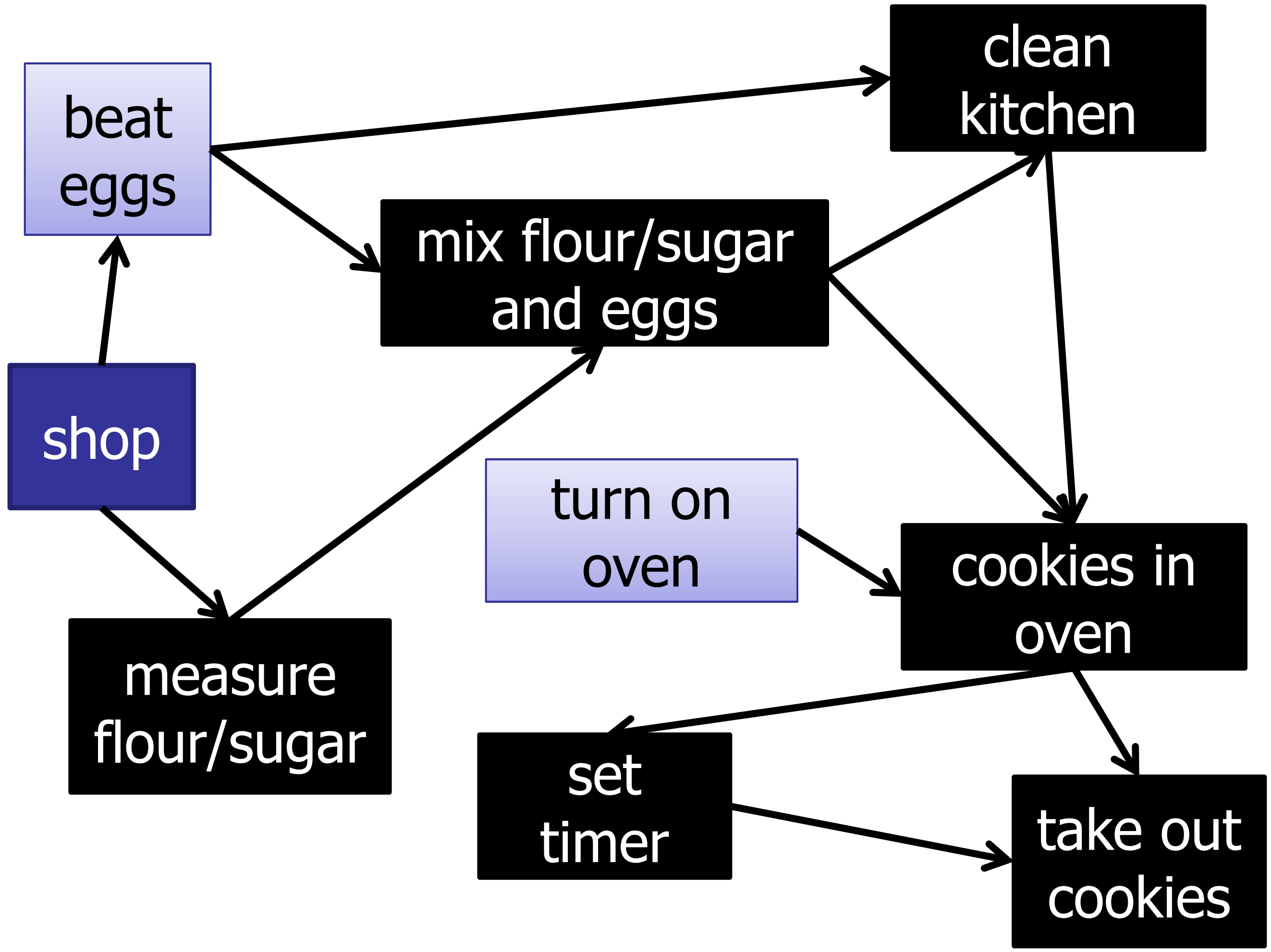
# Post-Order Depth-First Search

1.
2.
3.
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



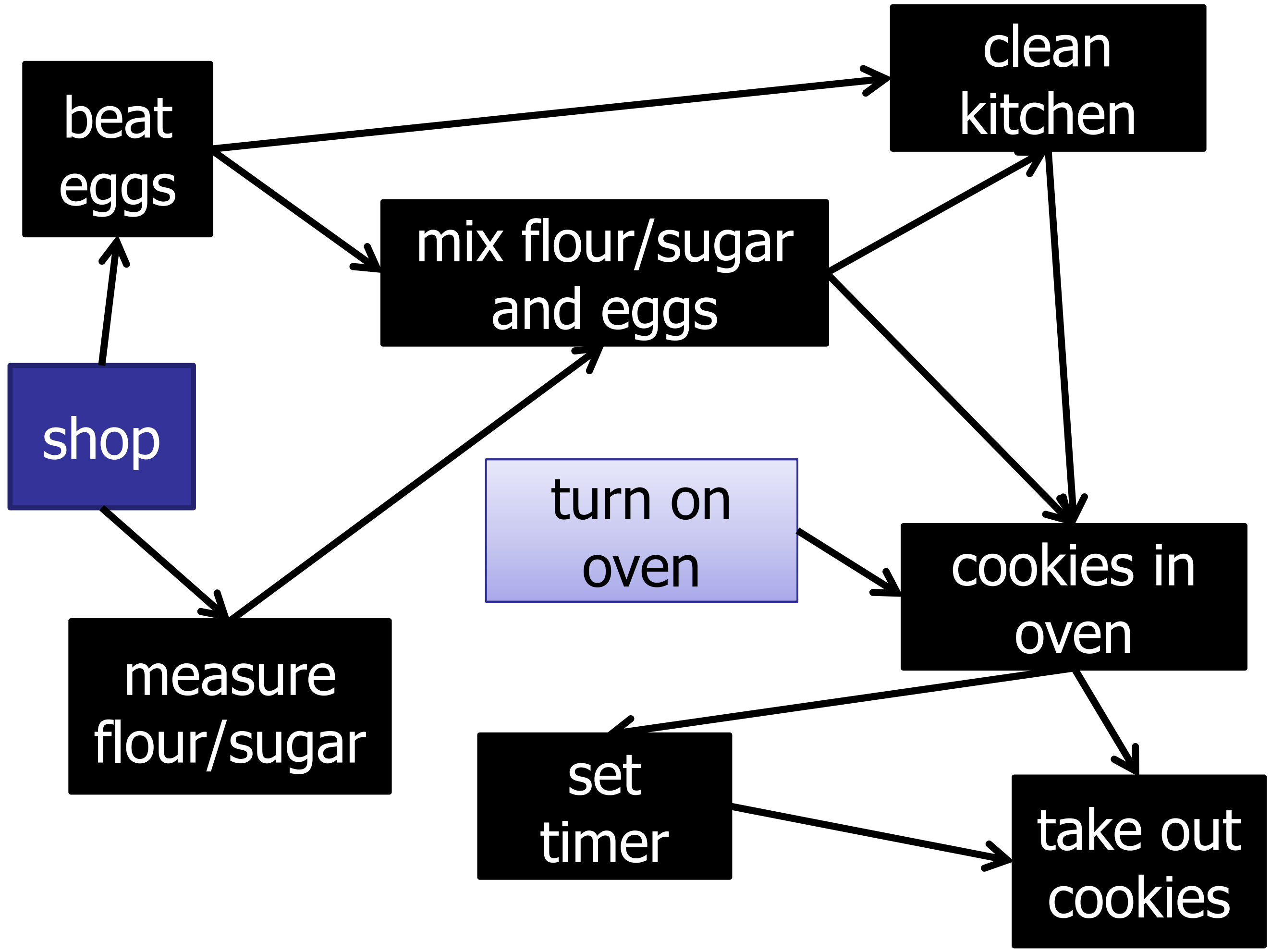
# Post-Order Depth-First Search

1.
2.
3.
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



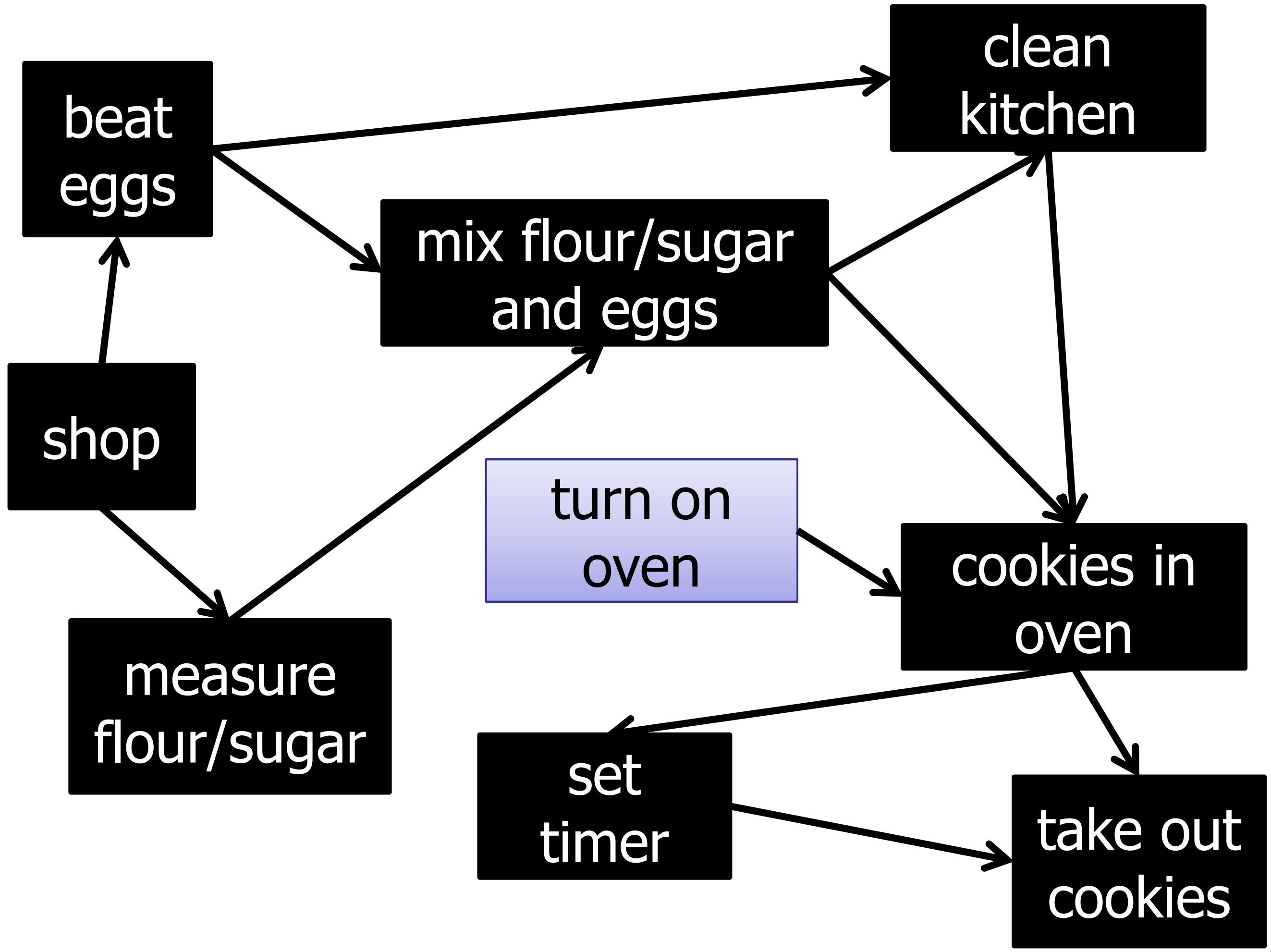
# Post-Order Depth-First Search

1.
2.
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



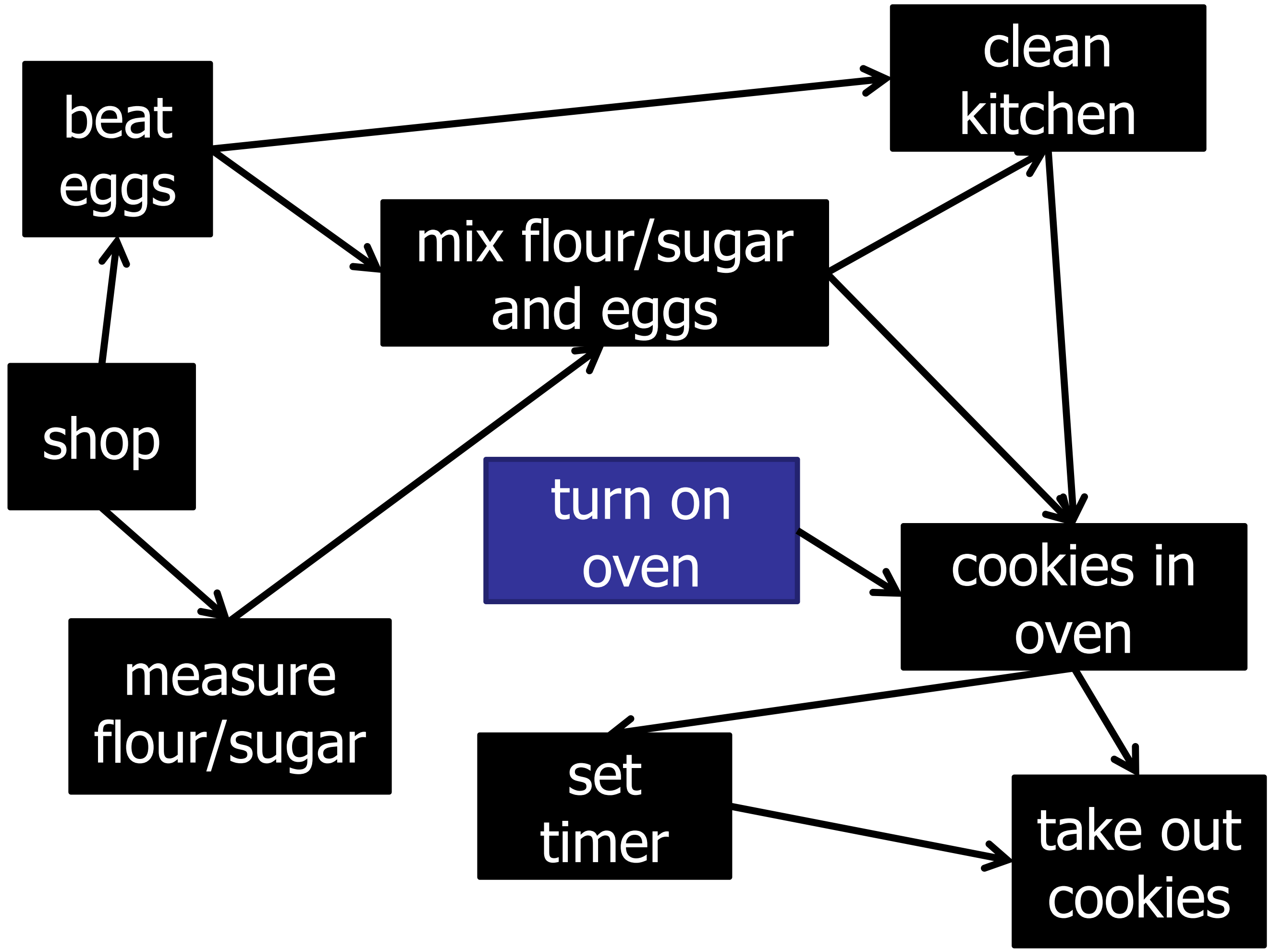
# Post-Order Depth-First Search

1.
2. shop
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



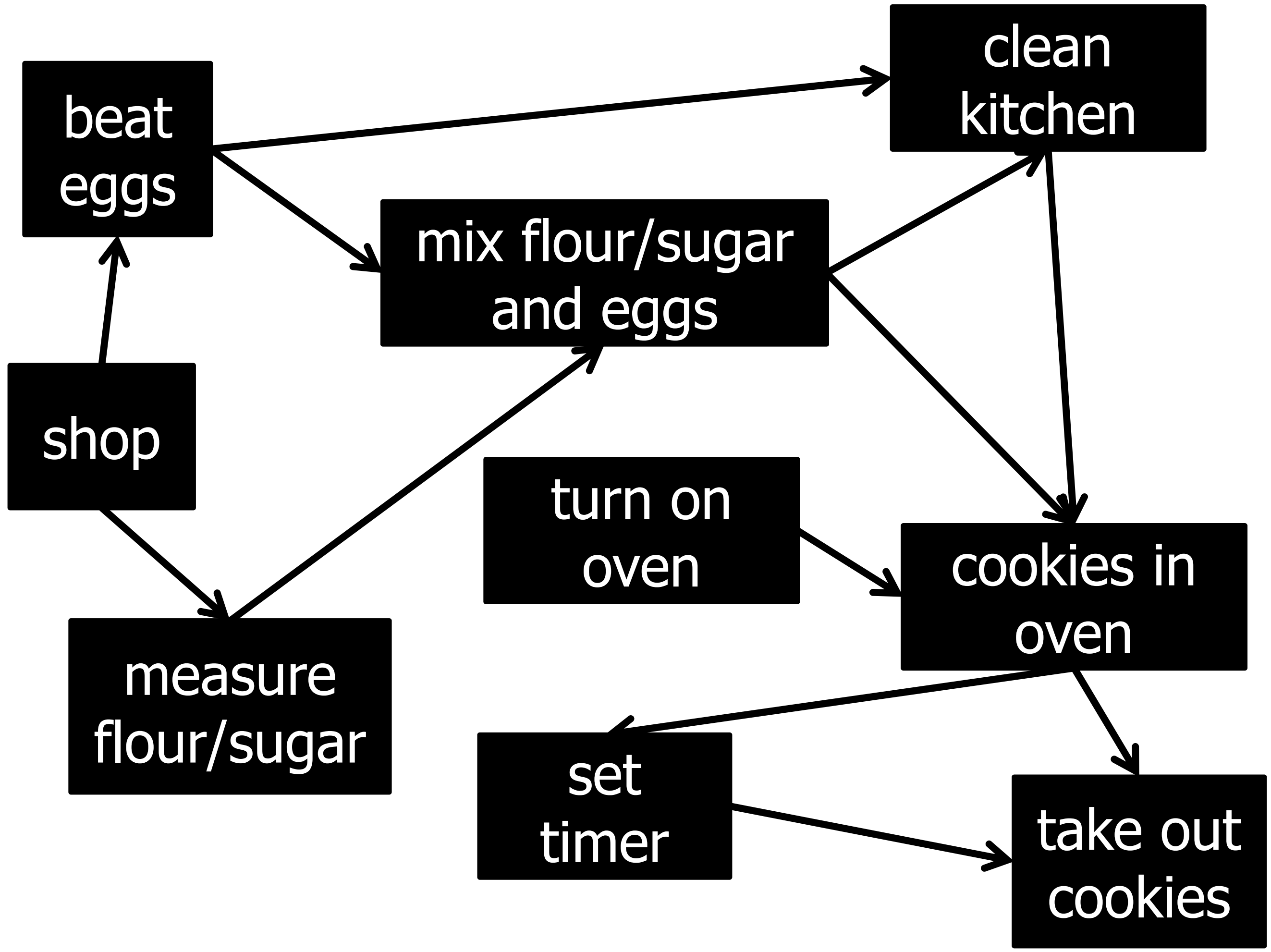
# Post-Order Depth-First Search

1.
2. shop
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



# Post-Order Depth-First Search

1. on oven
2. shop
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



# Topological Sort

What is the time complexity of topological sort?

DFS: O(V+E)

# Depth-First Search

```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (Integer v : nodeList[startId].nbrList) {
     if (!visited[v]){
           visited[v] = true;
           DFS-visit(nodeList, visited, v);
           schedule.prepend(v);
     }
  }
}
```

# Depth-First Search

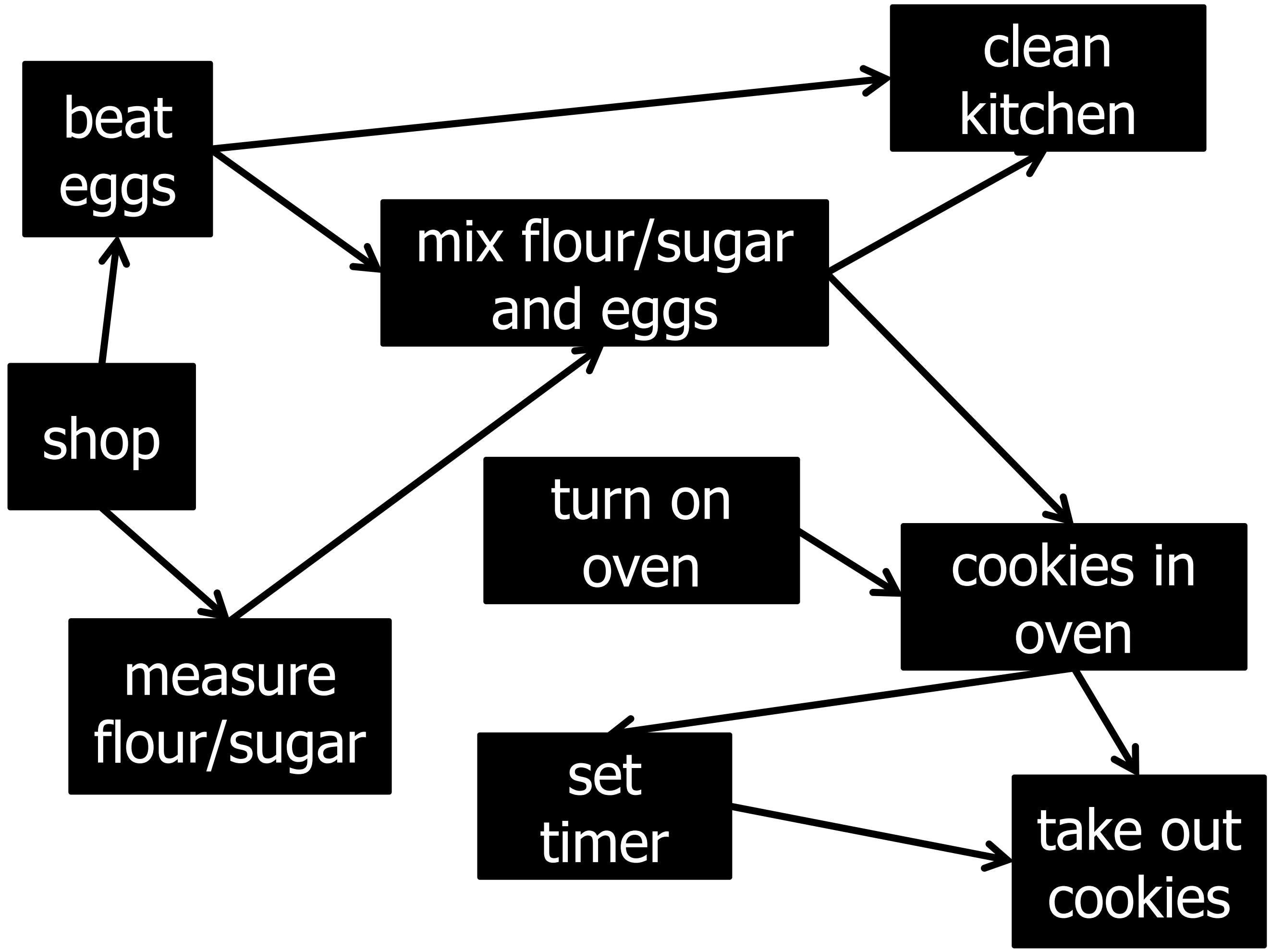
```
DFS(Node[] nodeList){
boolean[] visited = new boolean[nodeList.length];
Arrays.fill(visited, false);

  for (start = i; start<nodeList.length; start++) {
     if (!visited[start]){
           visited[start] = true;
           DFS-visit(nodeList, visited, start);
           schedule.prepend(v);
     }
  }
```

# Is a topological ordering unique?

1. Yes
2. No
3. Only on Thursdays.

# Post-Order Depth-First Search

1. **on oven**
2. **shop**
3. beat
4. measure
5. mix
6. **clean**
7. in oven
8. **set timer**
9. take out



# Topological Sort

Input:

- Directed Acyclic Graph (DAG)

Output:

- Total ordering of nodes, where all edges point forwards.

Algorithm:

- Post-order Depth-First Search
- $O(V + E)$ time complexity

# Topological Sort

Alternative algorithm:

Input: directed graph G

Repeat:

- S = all nodes in G that have *no* incoming edges.
- Add nodes in S to the topo-order
- Remove all edges adjacent to nodes in S
- Remove nodes in S from the graph

Time:

- O(V + E) time complexity

# Topological Sort

<u>Kahn's Algorithm:</u>
Repeat:
S = nodes in G that have *no* incoming edges.
Add nodes in S to the topo-order
Remove all edges adjacent to nodes in S
Remove nodes in S from the graph

## Implementation:

- Maintain all nodes in priority queue.
- Keys are incoming edges.
- Remove min-degree node u.
- For each outgoing edge (u,v):
  *decrease-key of v by 1.*

What is the running time of this implementation?

1. $O(V+E)$
2. $O(E \log V)$
3. $O(V \log E)$
4. $O(V^2)$
5. $O(VE)$

# Topological Sort

<u>Kahn's Algorithm:</u>

Repeat:

- S = nodes in G that have *no* incoming edges.
- Add nodes in S to the topo-order
- Remove all edges adjacent to nodes in S
- Remove nodes in S from the graph

## Challenge:

Implement Kahn's Algorithm in O(V+E).

# Plan for today:

Directed Acyclic Graphs (DAG)

Topological Order

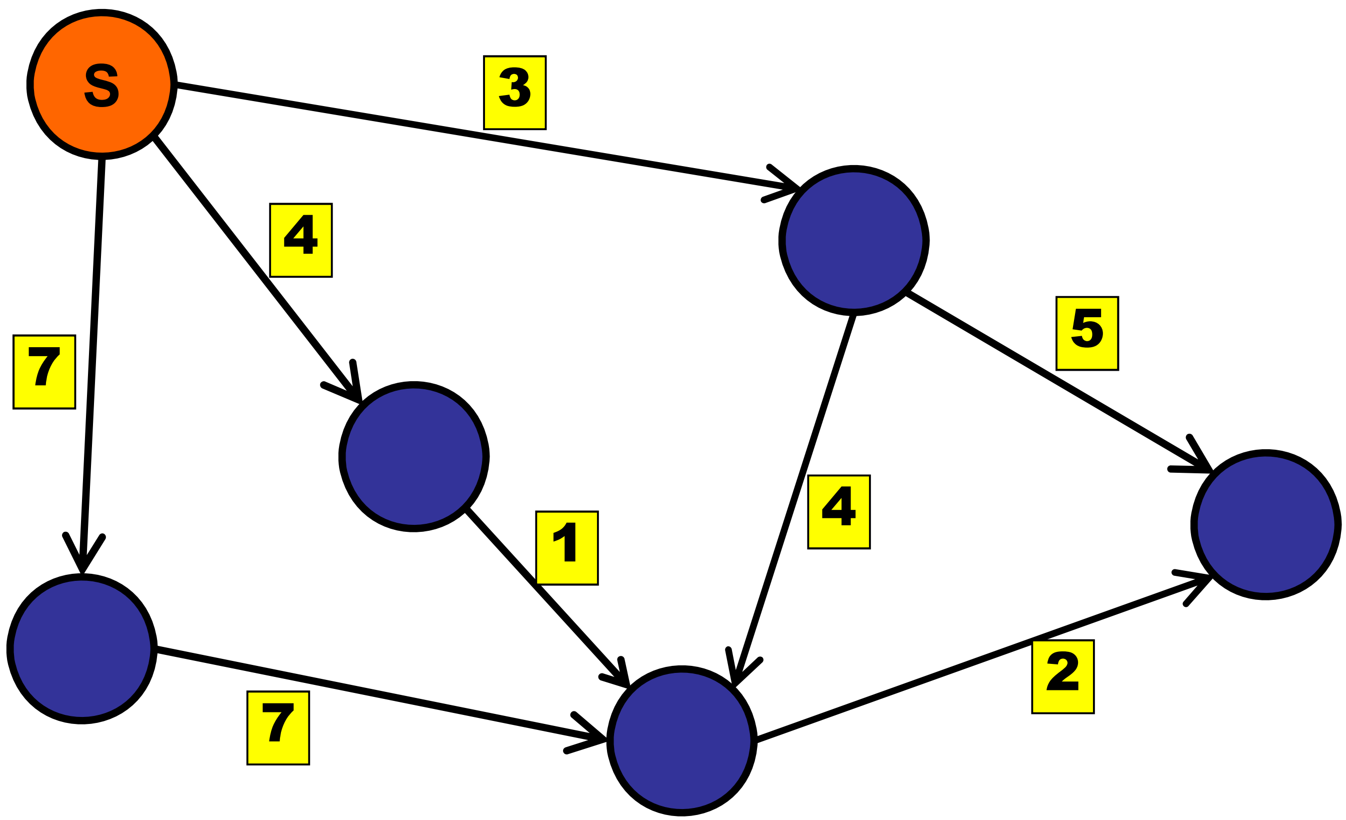
Topological Sort

Shortest Path in a DAG

Shortest Path in a tree

# Shortest Paths

Acyclic Graph: Suppose the graph has no cycles.



# Shortest Paths

Key idea:

Relax the edges in the "right" order.

Only relax each edge once:

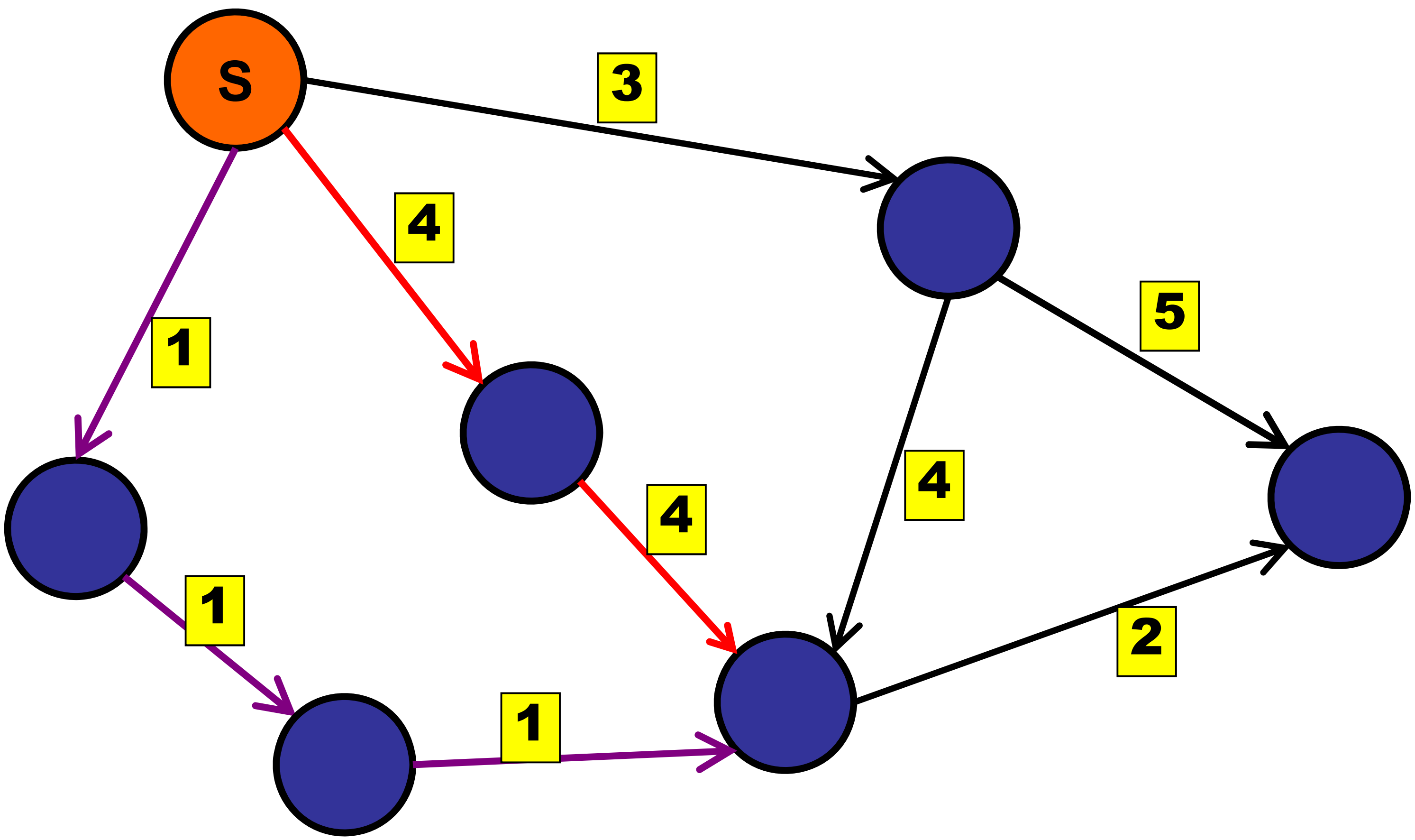
- O(E) cost (for relaxation step).

# What order should we relax the nodes?

1. BFS
2. DFS pre-order
3. DFS post-order
4. Shortest edge
5. Longest edge
6. Other

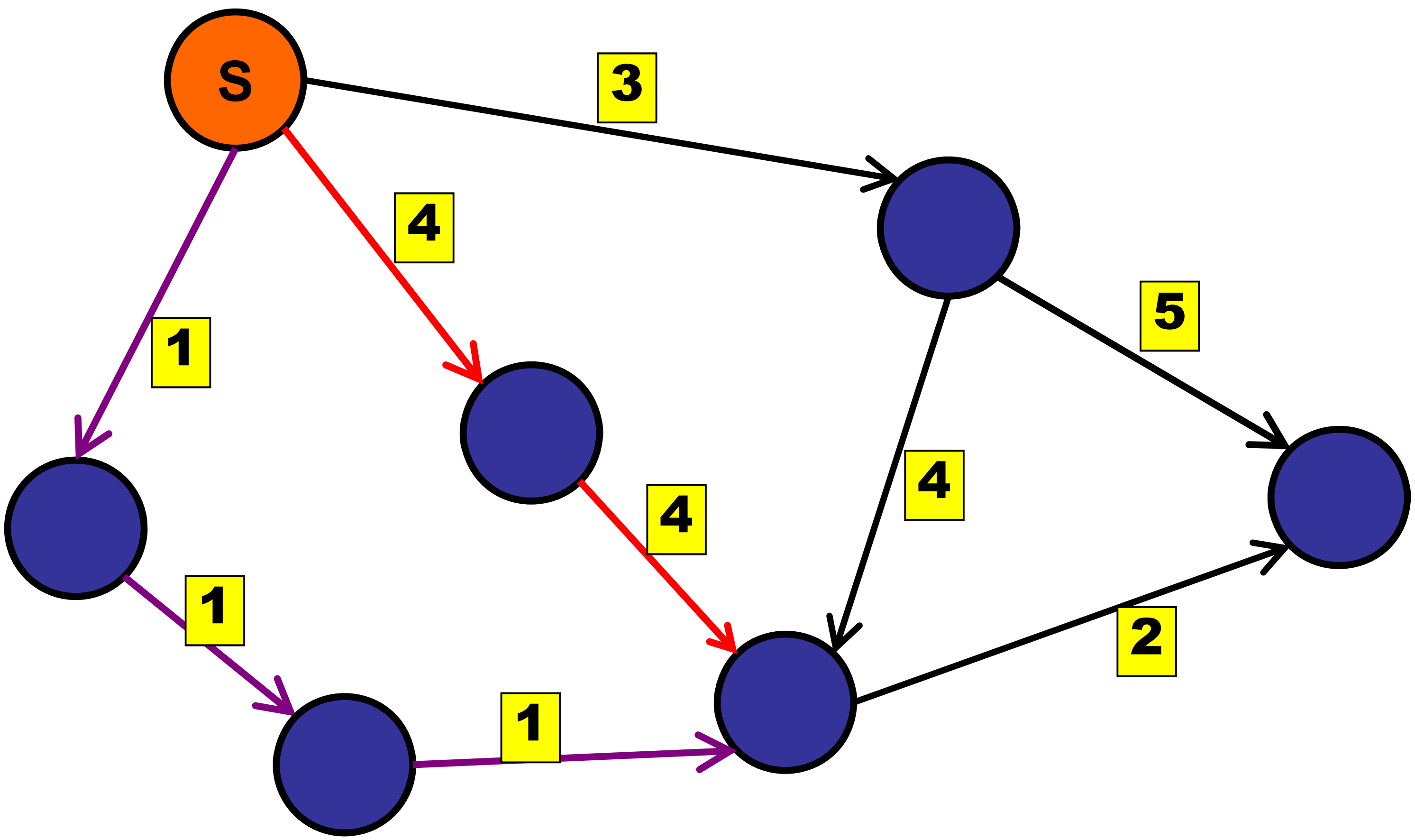
# Shortest Paths

Acyclic Graph: Not BFS.



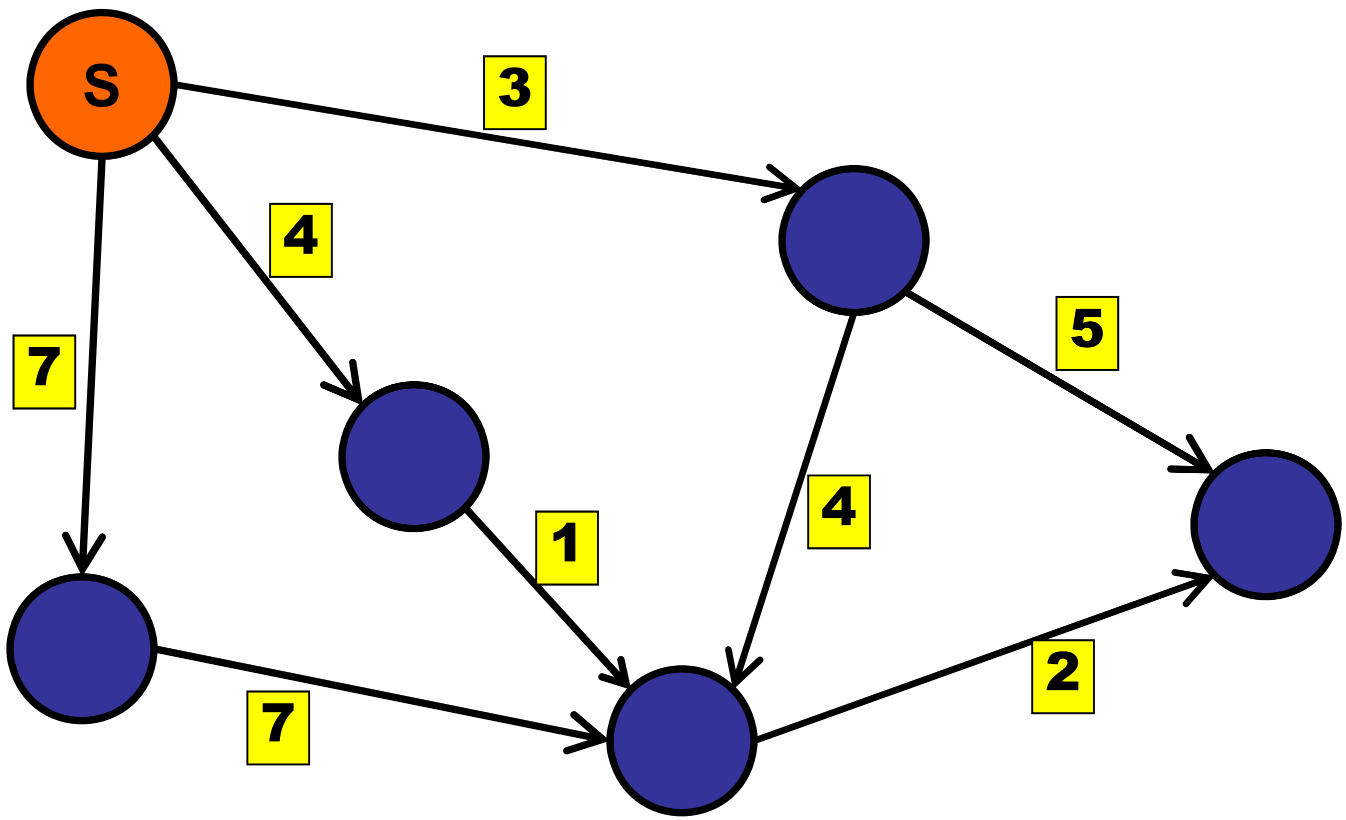
# Shortest Paths

Acyclic Graph: Not DFS-preorder.



# Shortest Paths

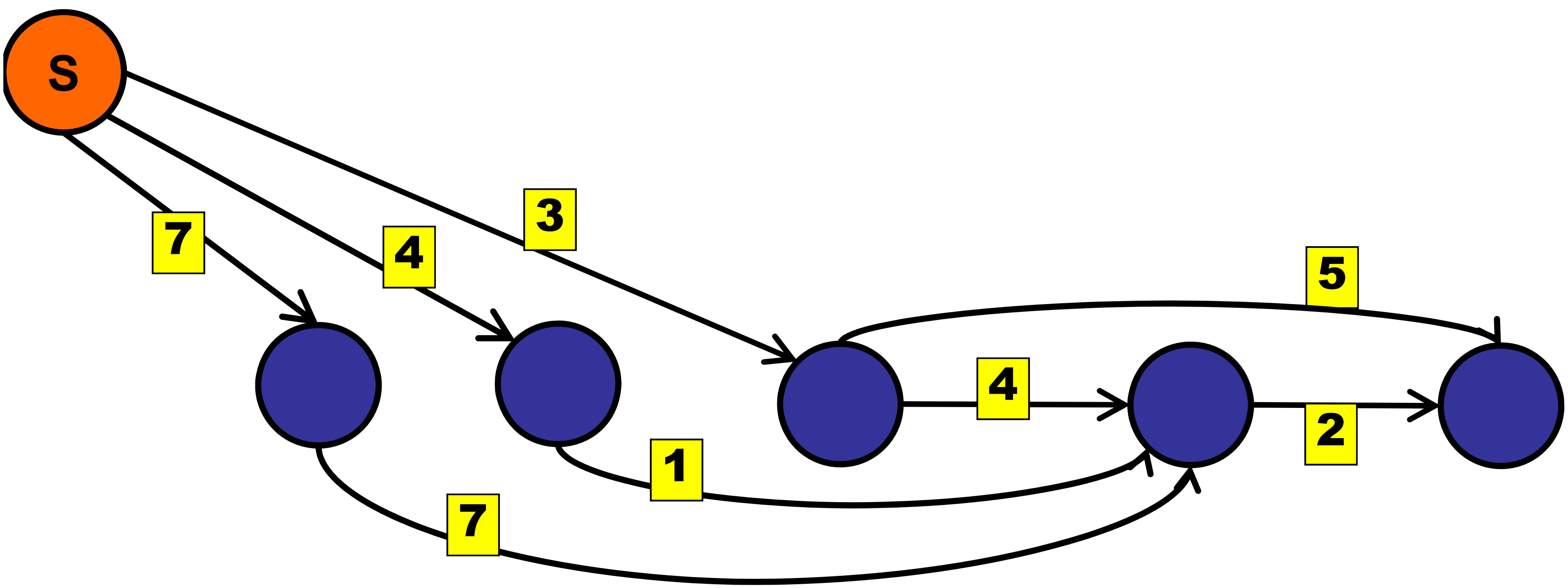
Acyclic Graph: DFS post-order ➔ topological order.



# Shortest Paths

Acyclic Graph: has no cycles.

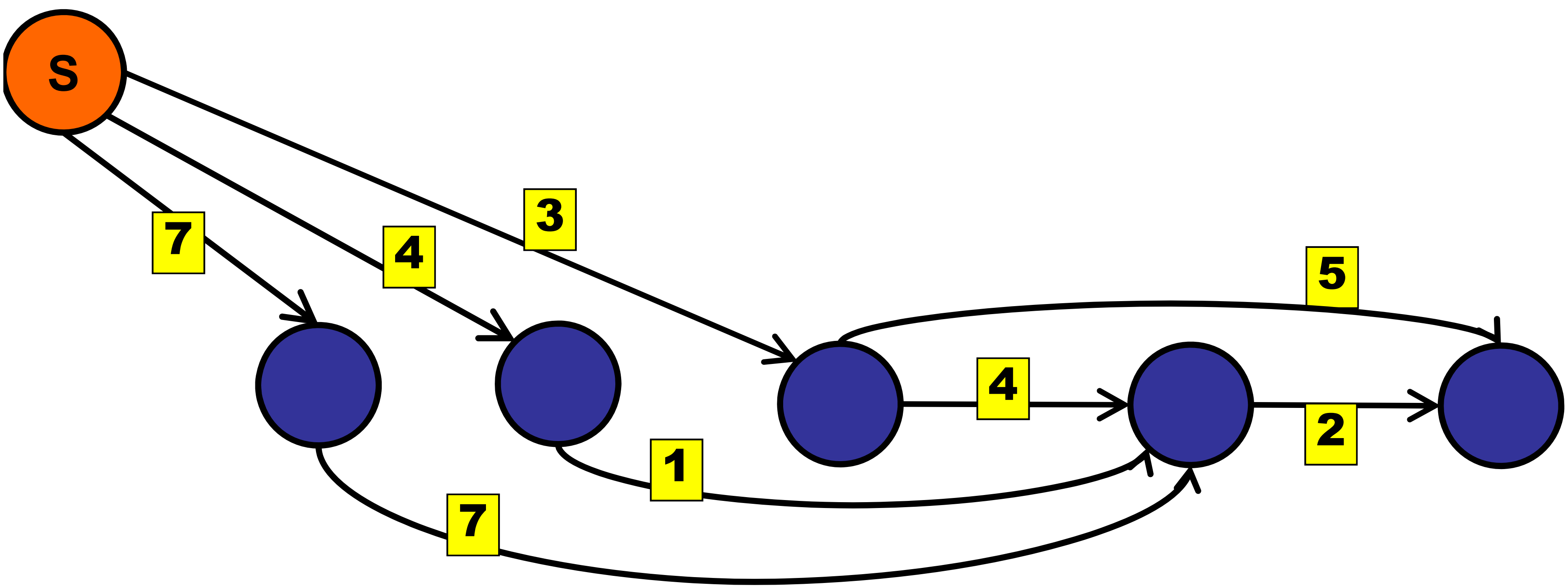
1. Topological sort



# Shortest Paths

Acyclic Graph: has no cycles.

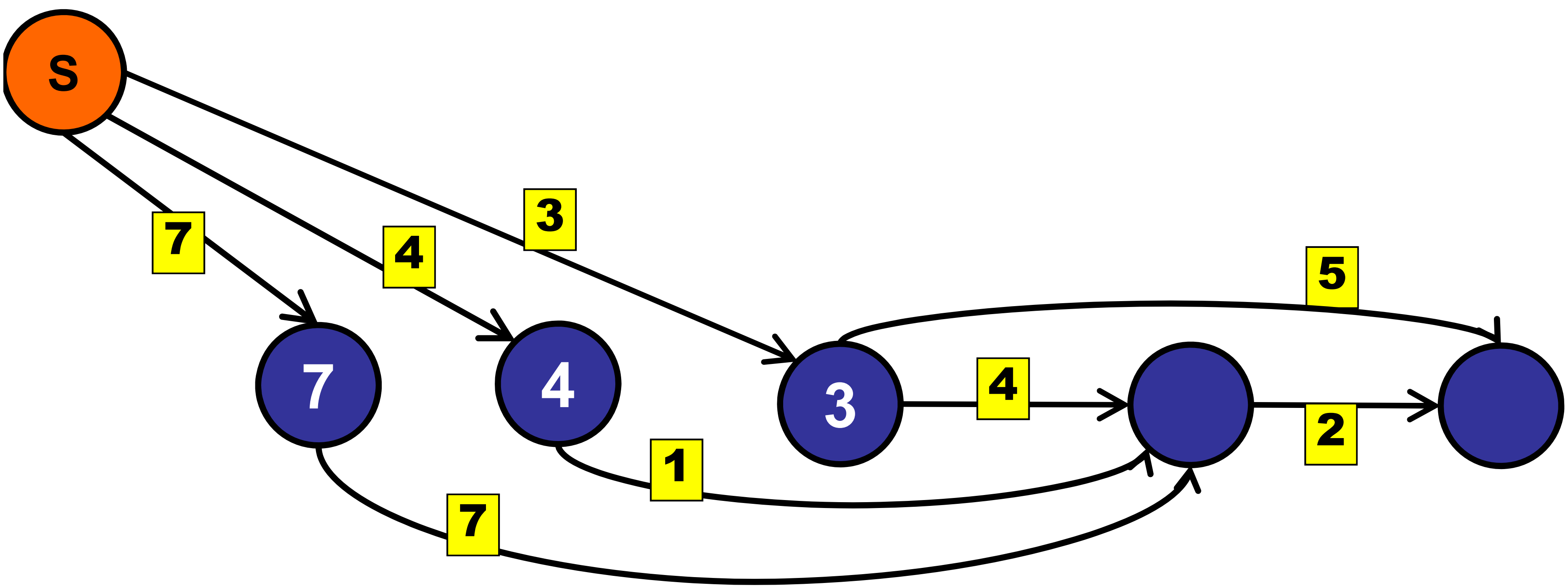
1. Topological sort
2. Relax in order.



# Shortest Paths

Acyclic Graph: has no cycles.

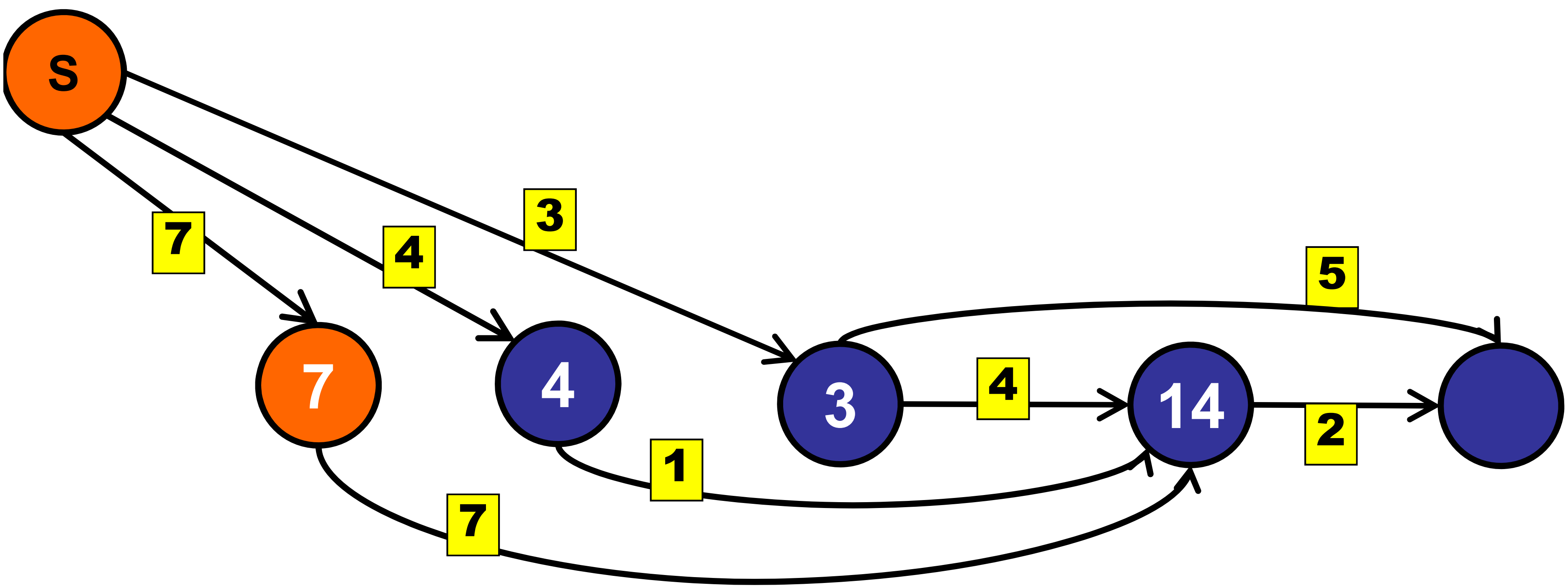
1. Topological sort
2. Relax in order.



# Shortest Paths

Acyclic Graph: has no cycles.

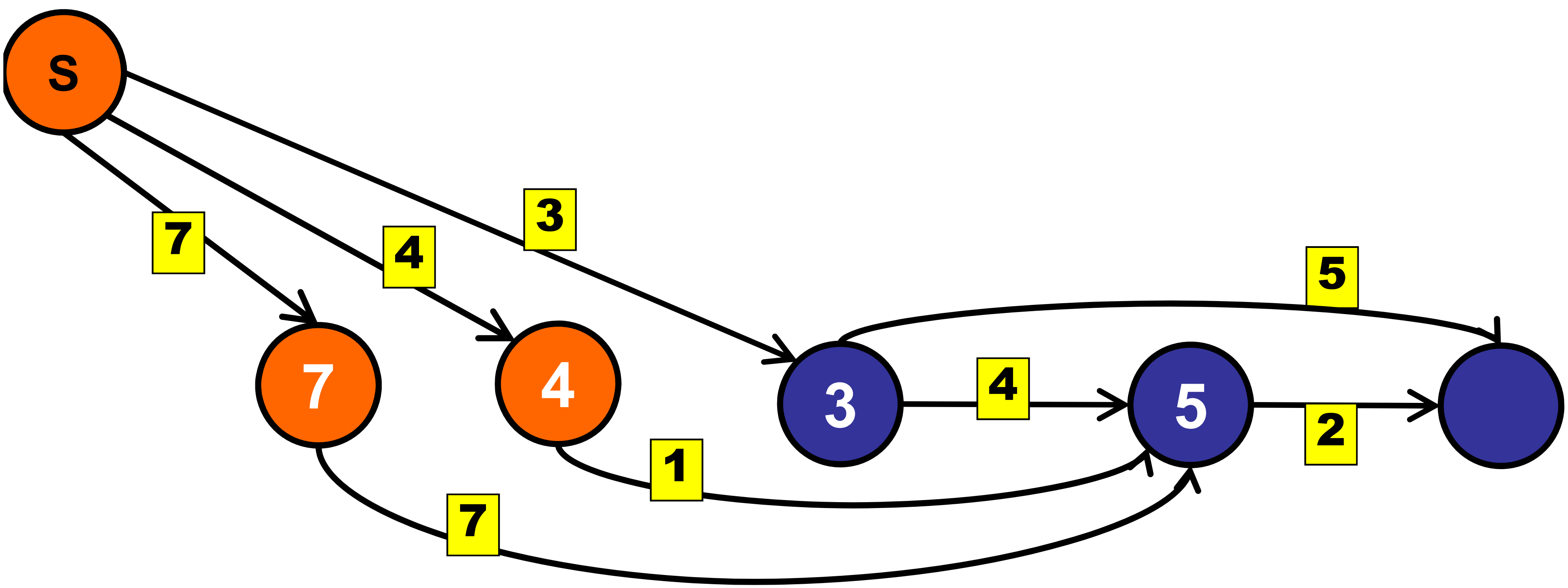
1. Topological sort
2. Relax in order.



# Shortest Paths

Acyclic Graph: has no cycles.

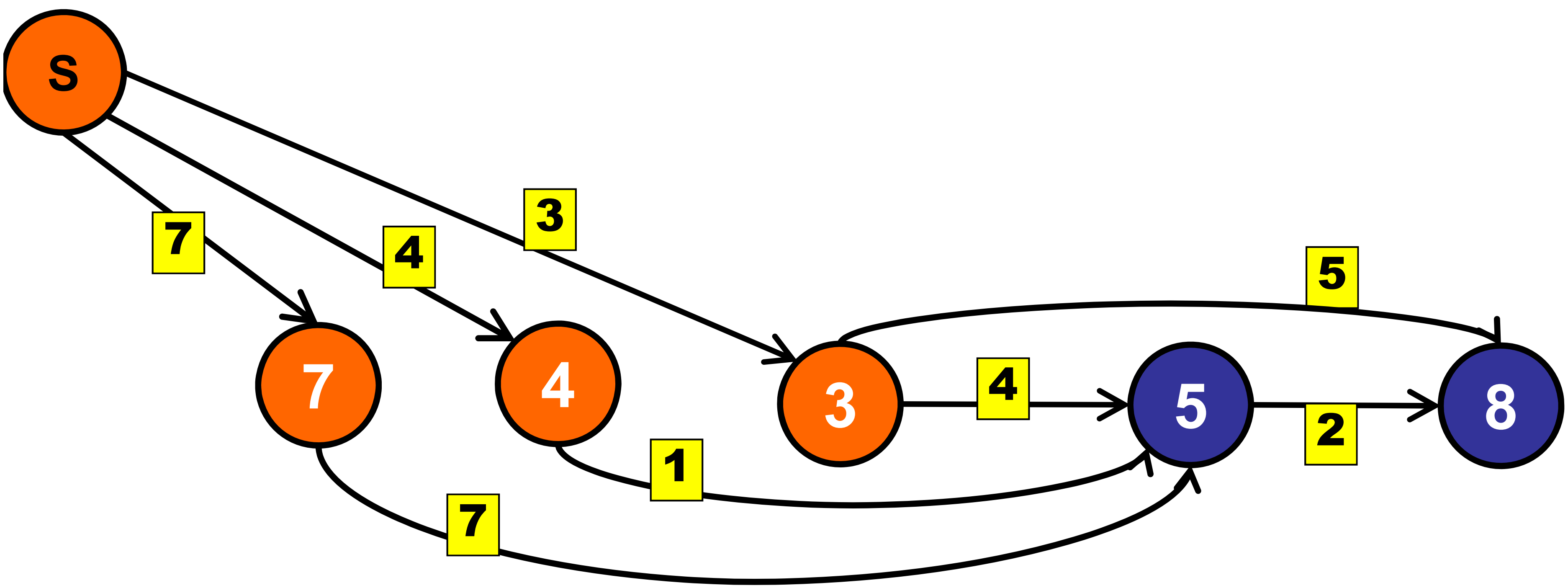
1. Topological sort
2. Relax in order.



# Shortest Paths

Acyclic Graph: has no cycles.

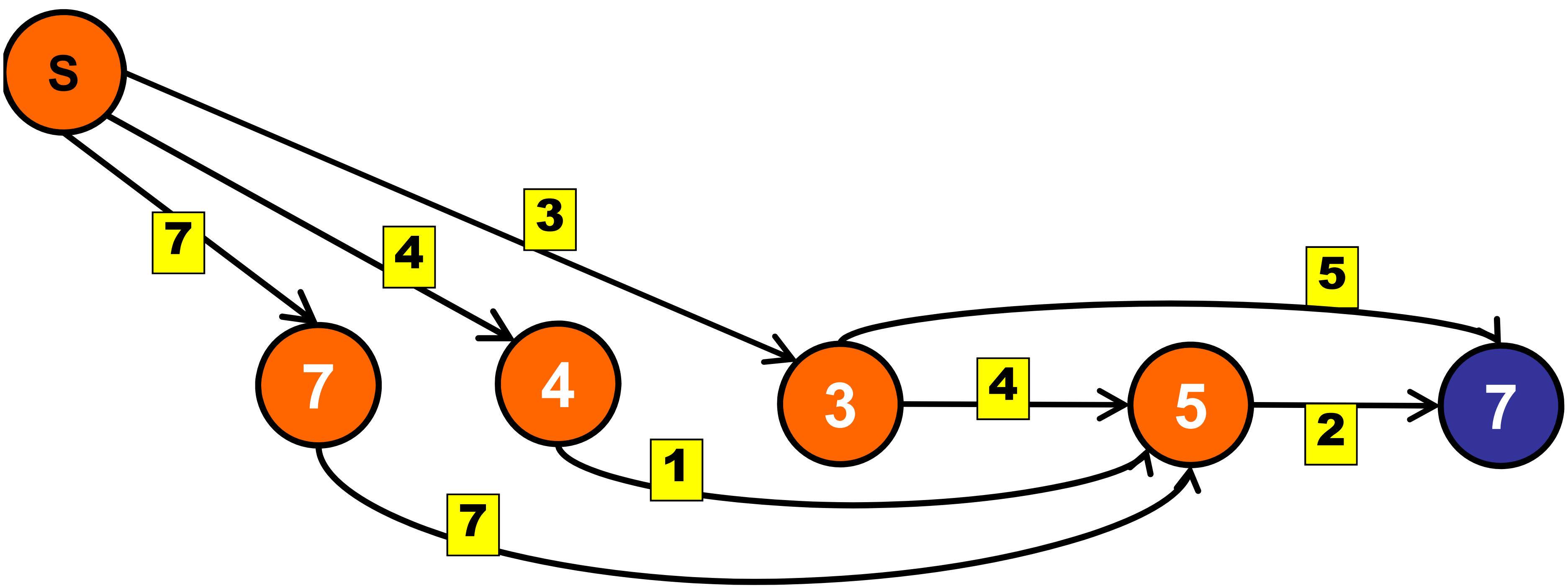
1. Topological sort
2. Relax in order.



# Shortest Paths

Acyclic Graph: has no cycles.

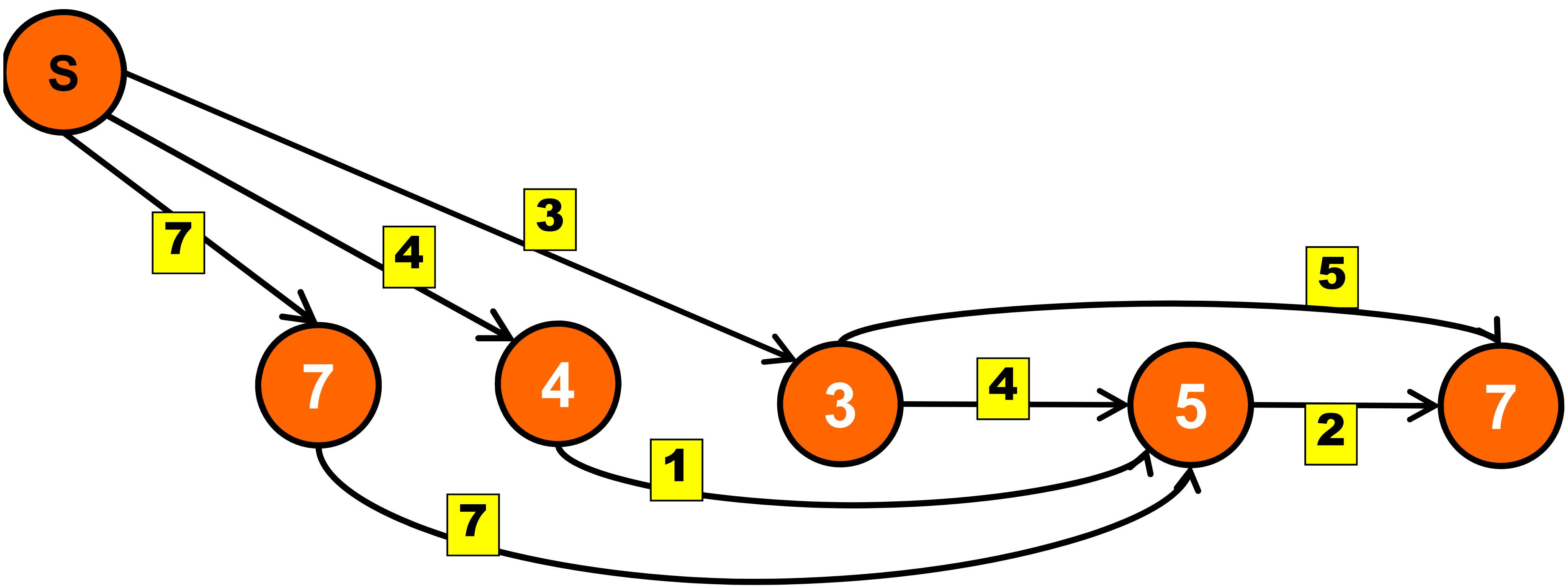
1. Topological sort
2. Relax in order.



# Shortest Paths

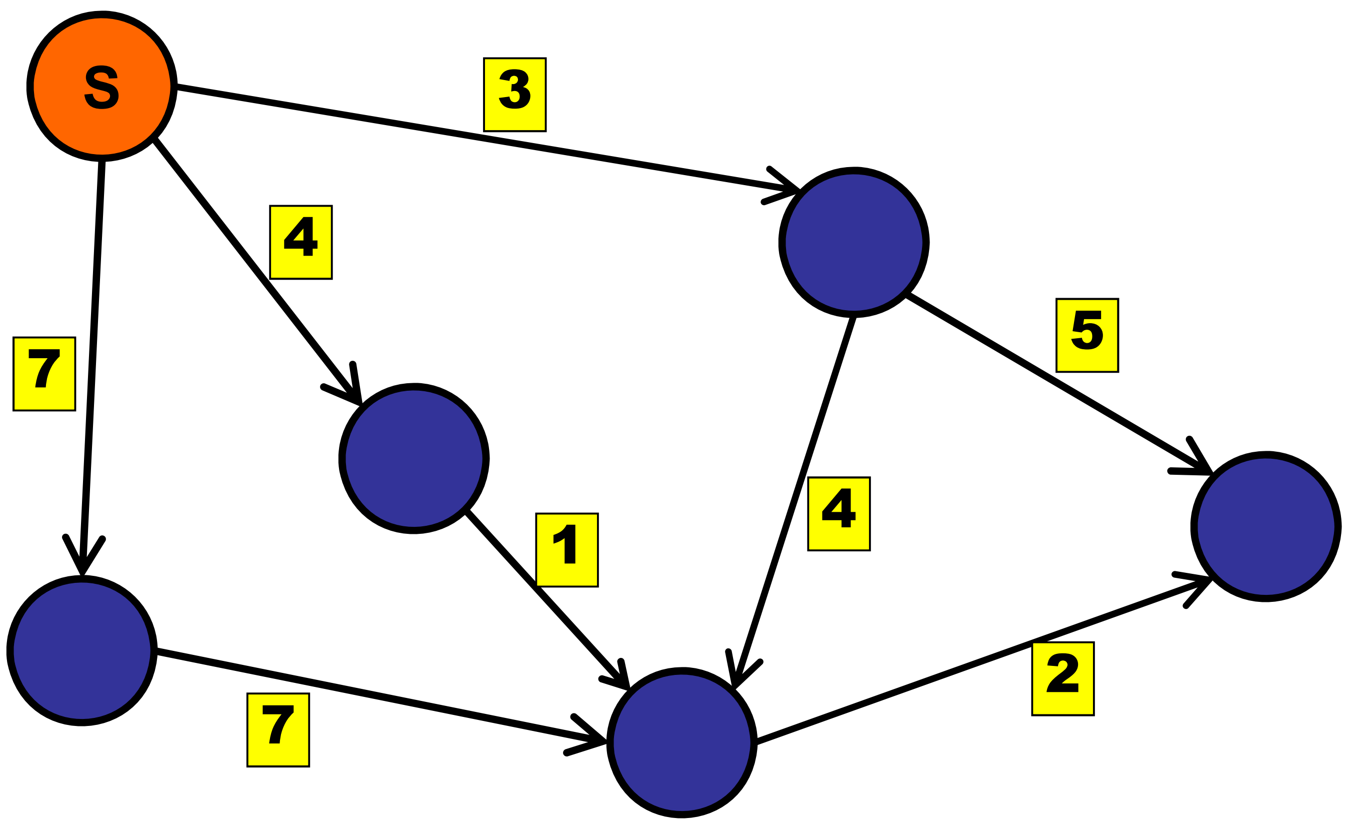
Acyclic Graph: has no cycles.

1. Topological sort
2. Relax in order.



# Shortest Paths

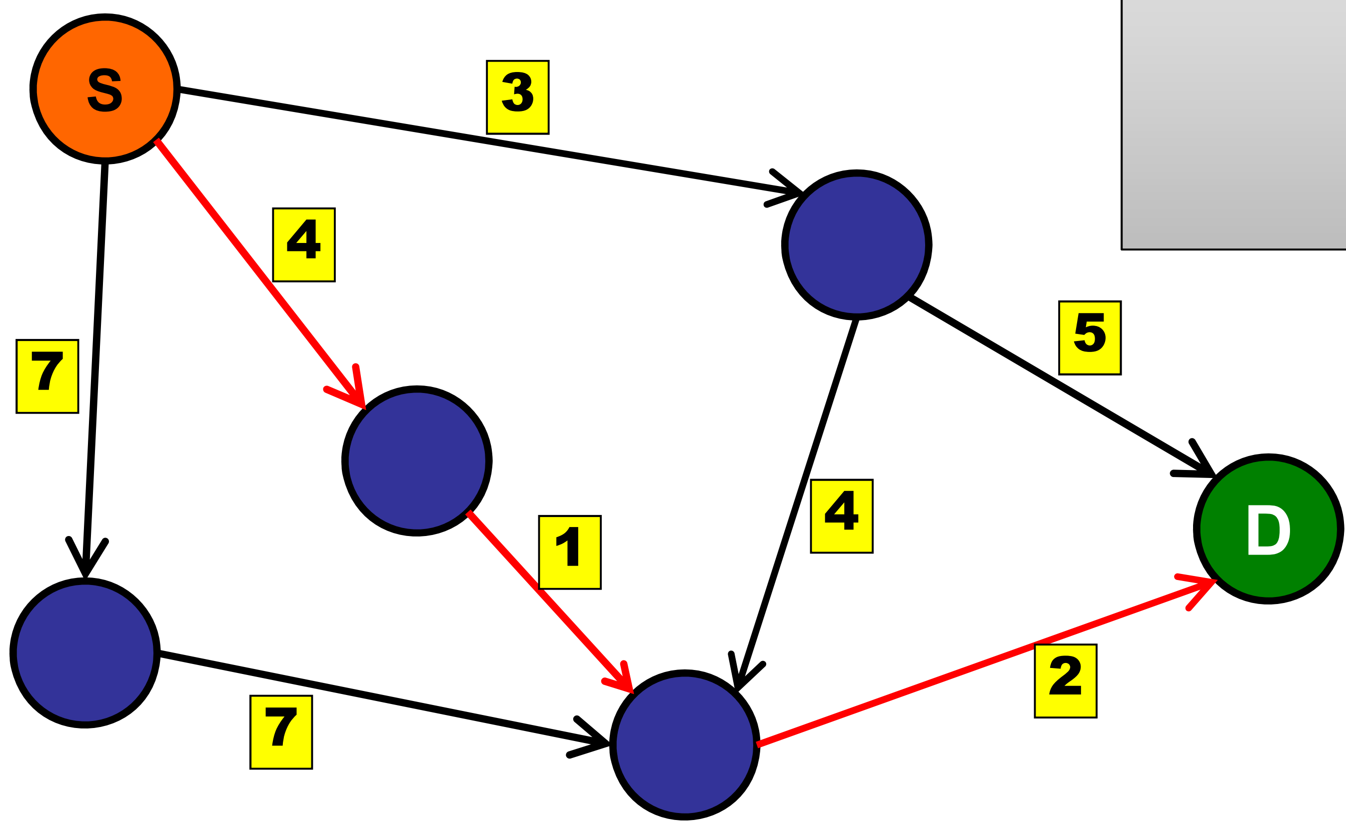
Acyclic Graph: Why topological order?



# Shortest Paths

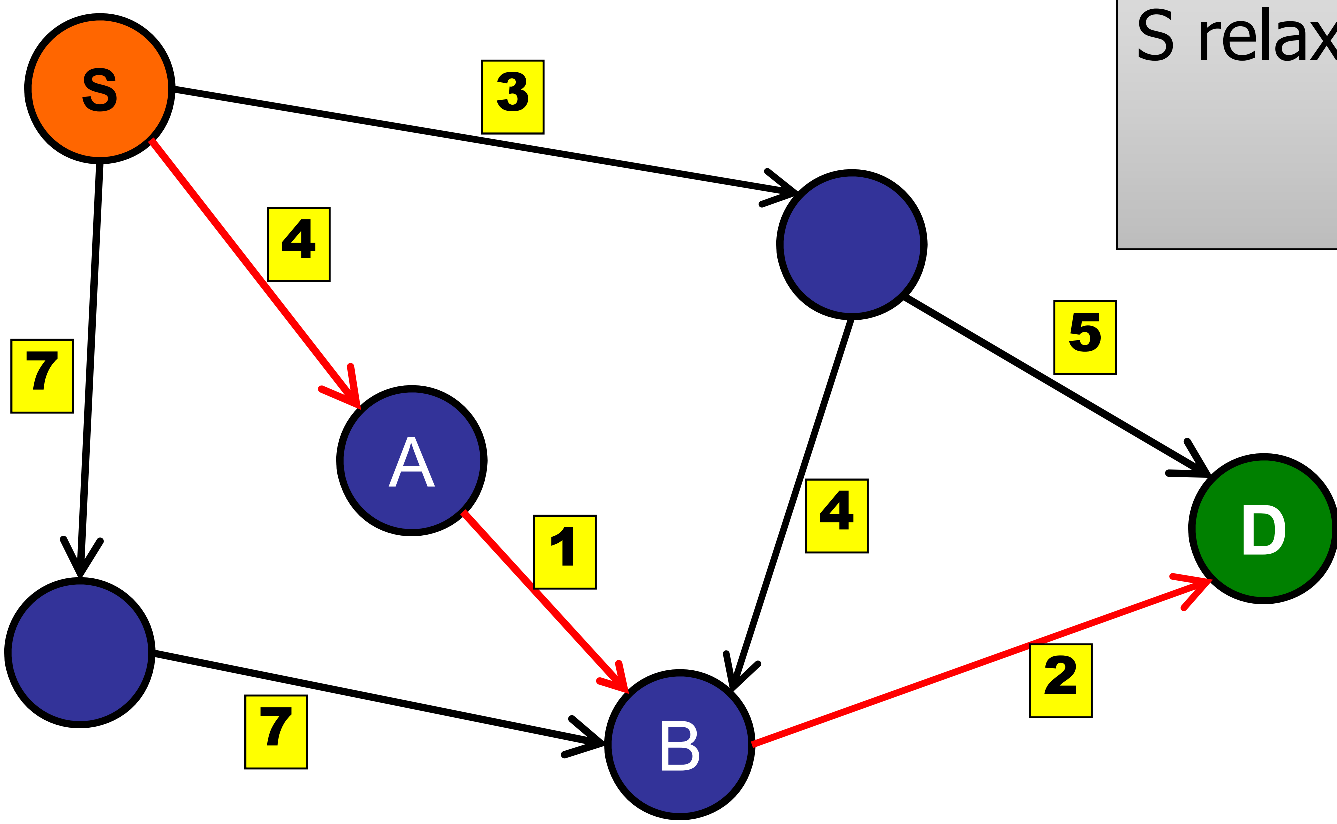
Acyclic Graph: Why topological order?

Fix S-D shortest path.



# Shortest Paths

Acyclic Graph: Why topological order?

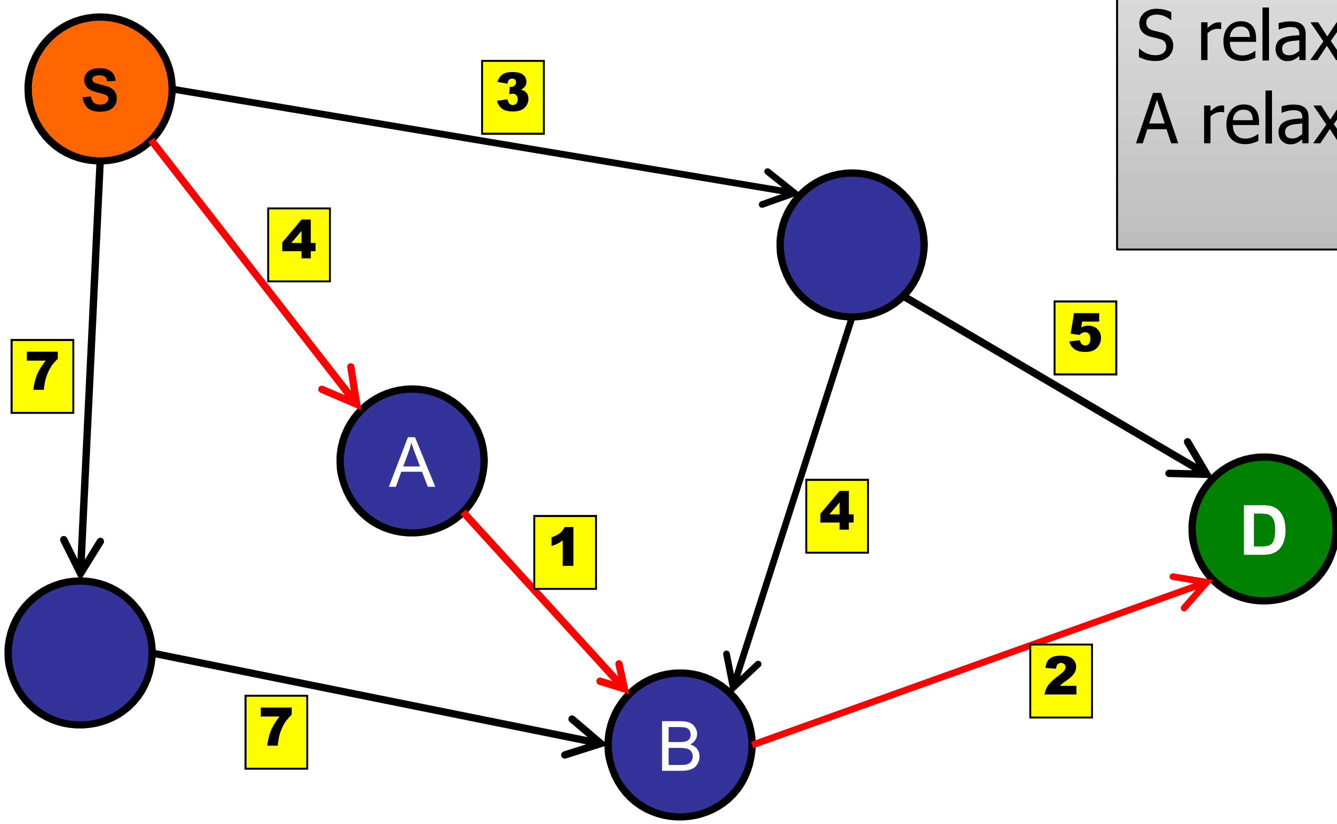



Fix S-D shortest path.

S relaxed before A.

# Shortest Paths

Acyclic Graph: Why topological order?

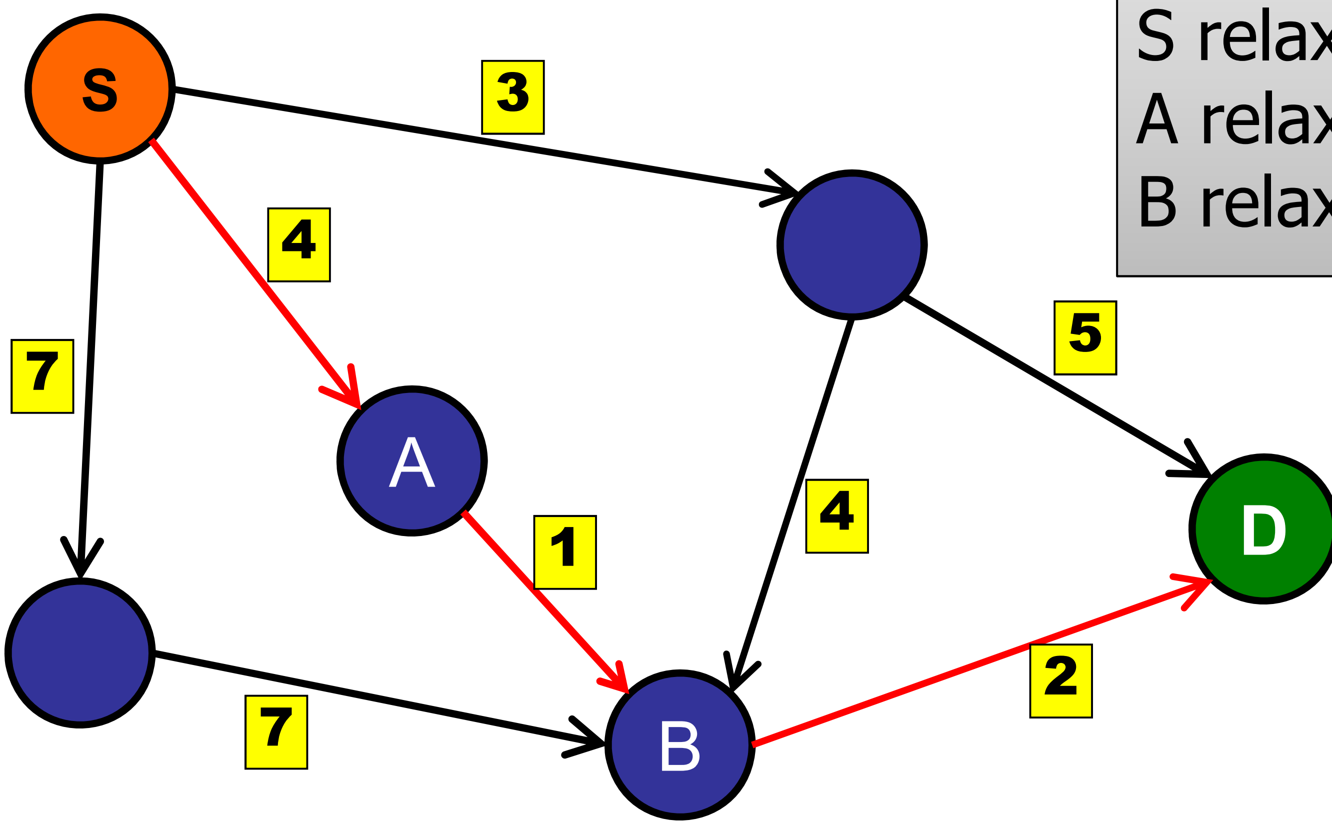



Fix S-D shortest path.

S relaxed before A.
A relaxed before B.

# Shortest Paths

Acyclic Graph: Why topological order?

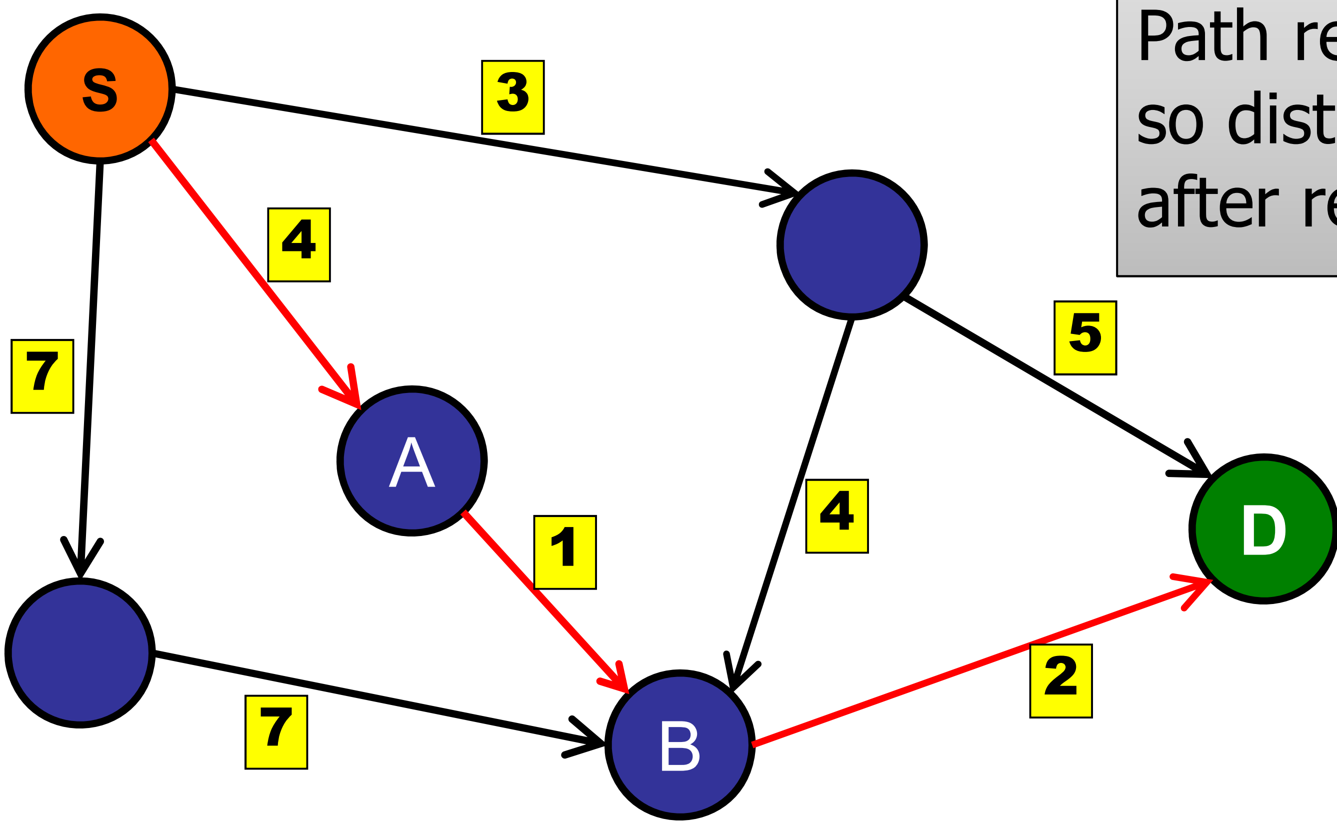



Fix S-D shortest path.

S relaxed before A.
A relaxed before B.
B relaxed before D.

# Shortest Paths

## Acyclic Graph: Why topological order?




Fix S-D shortest path.

Path relaxed in-order, so distance is correct after relaxation.

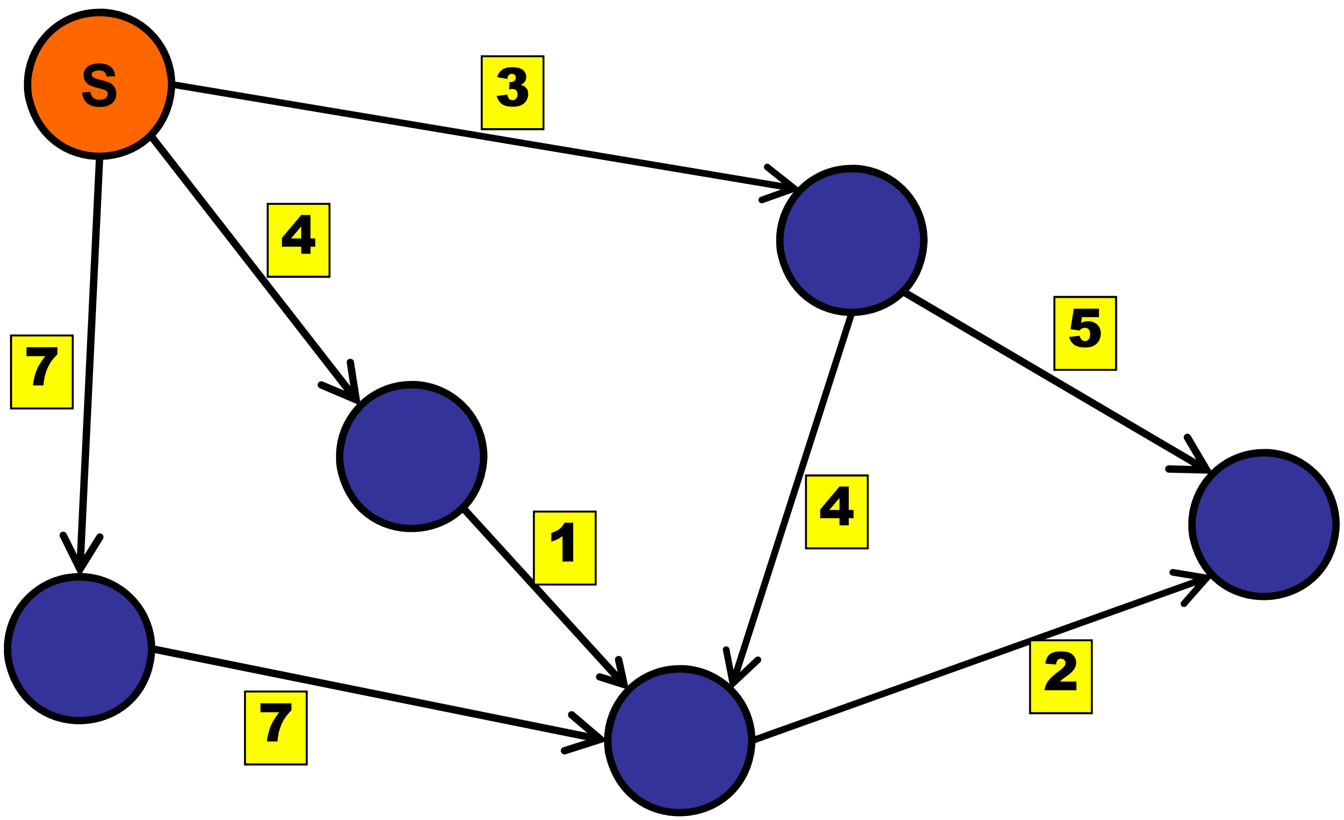
What is the running time of shortest paths on a DAG?

1. $O(V)$
2. $O(E)$
3. $O(V^2)$
4. $O(E \log V)$
5. $O(V \log E)$
6. $O(VE)$

# Longest Paths

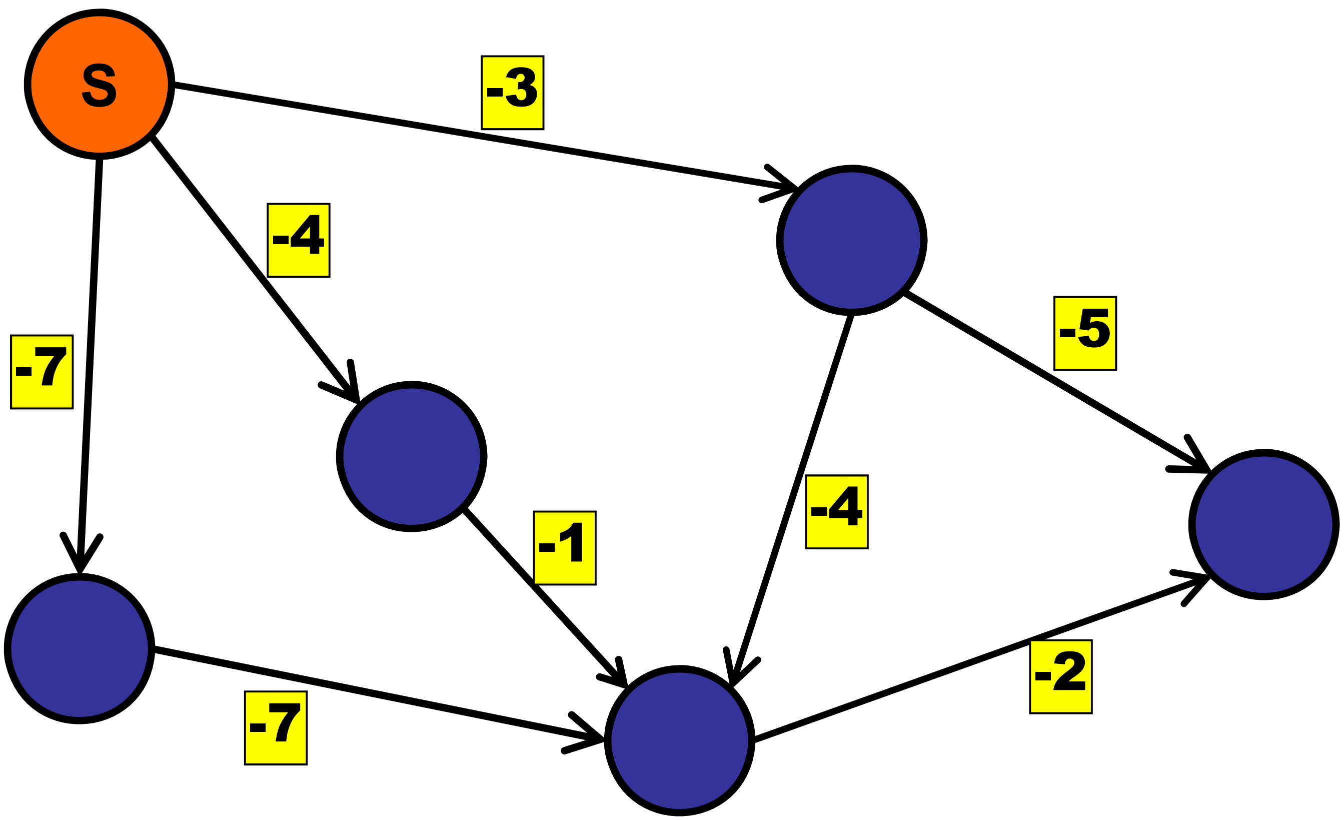
ARCHIPELAGO
is open

## Acyclic Graph: Any ideas?



# Longest Paths

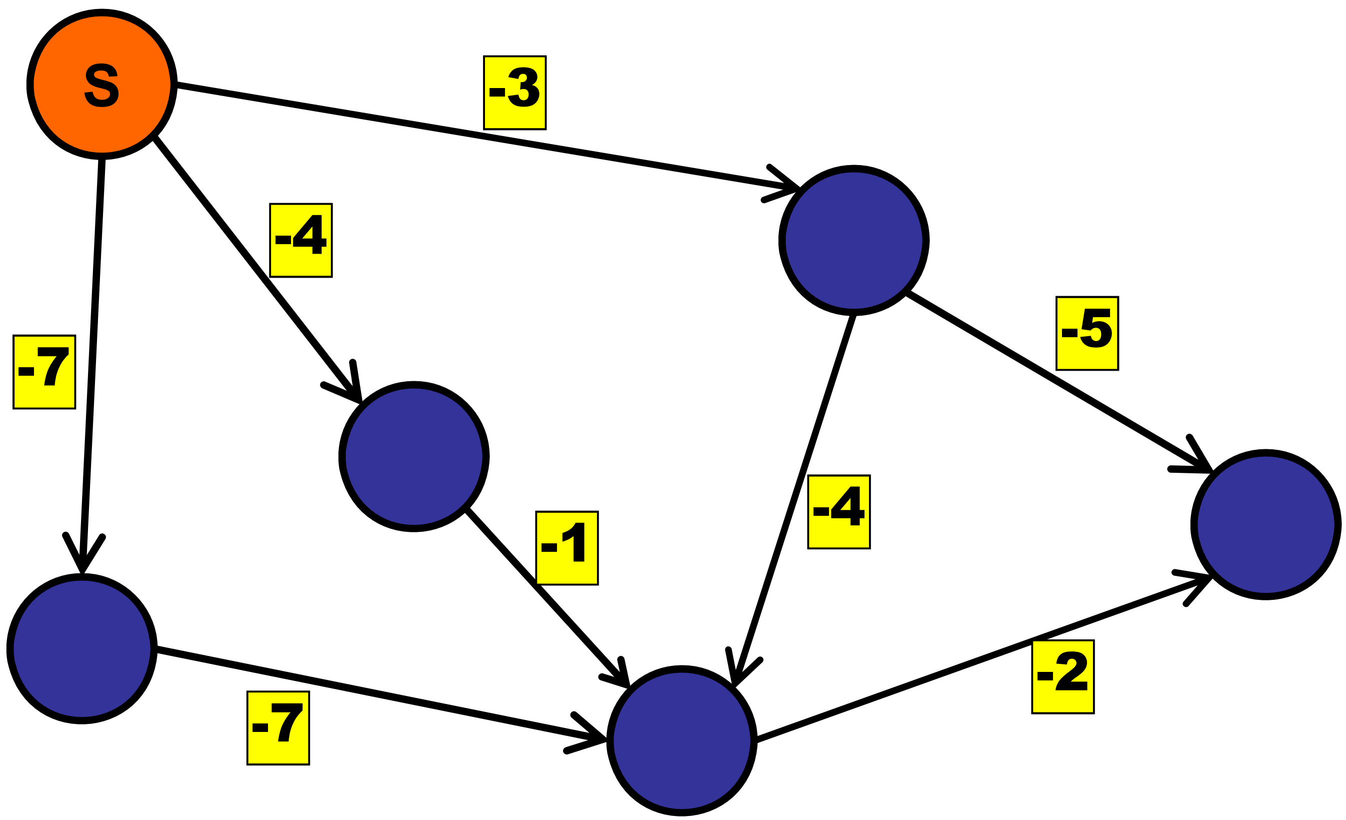
Acyclic Graph: Negate the edges!



# Longest Paths

Acyclic Graph:

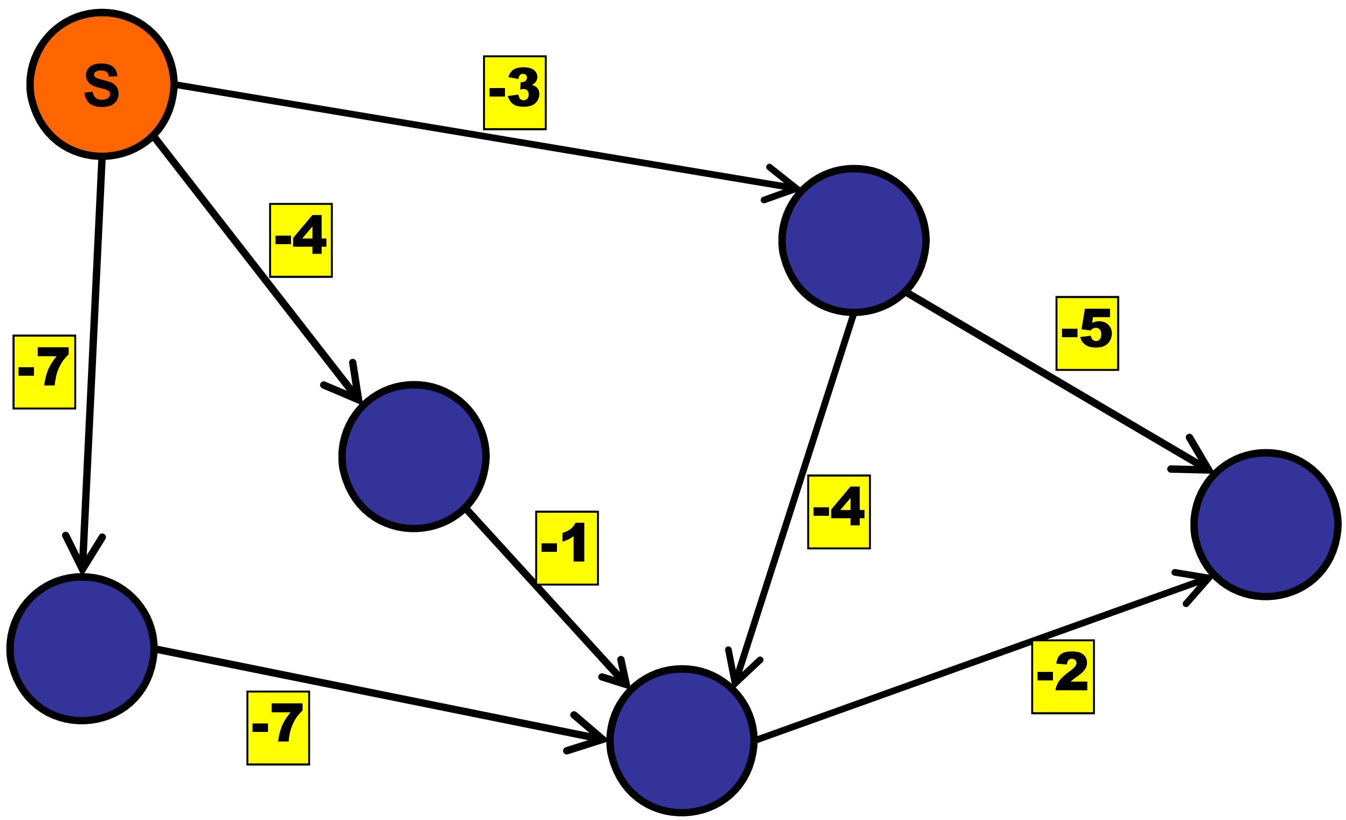
shortest path in negated=longest path in regular



# Longest Paths

Acyclic Graph:

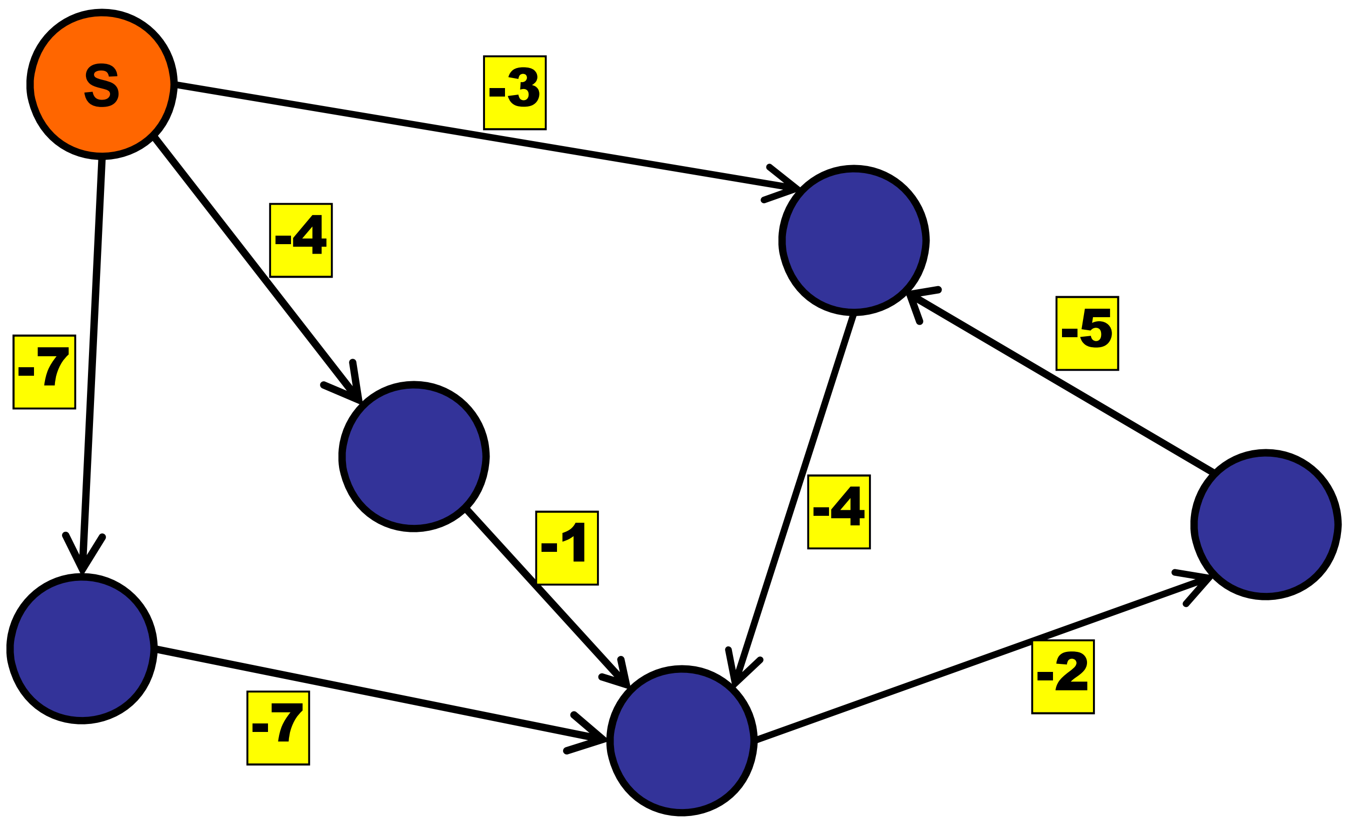
OR: modify relax function!



# Longest Paths

General (cyclic) Graph:  (positive weights)

Can we use the same trick?

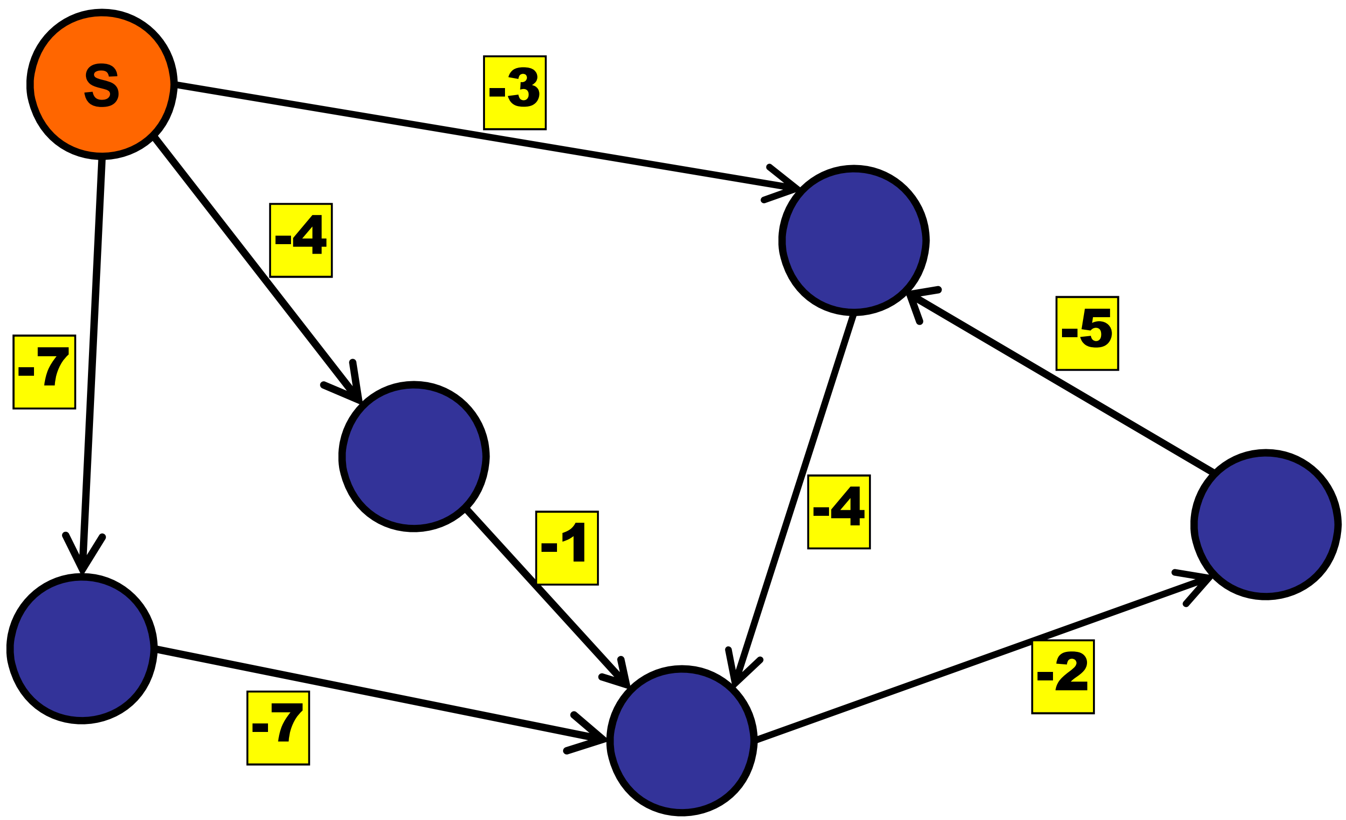


# Longest Paths

General (cyclic) Graph: (positive weights)

Can we use the same trick? NO

Negative weight cycles!



# Longest Path

Directed Acyclic Graph:

- Solvable efficiently using topological sort

General (cyclic) Graphs:

- NP-Hard
- Reduction from Hamiltonian Path:
  - If you could find the longest simple path, then you could decide if there is a path that visits every vertex.
  - Any polynomial time algorithm for longest path thus implies a polynomial time algorithm for HAMPATH.

# Plan for today:

Directed Acyclic Graphs (DAG)

Topological Order

Topological Sort

Shortest Path in a DAG

Shortest Path in a tree

# Shortest Paths

Key idea:

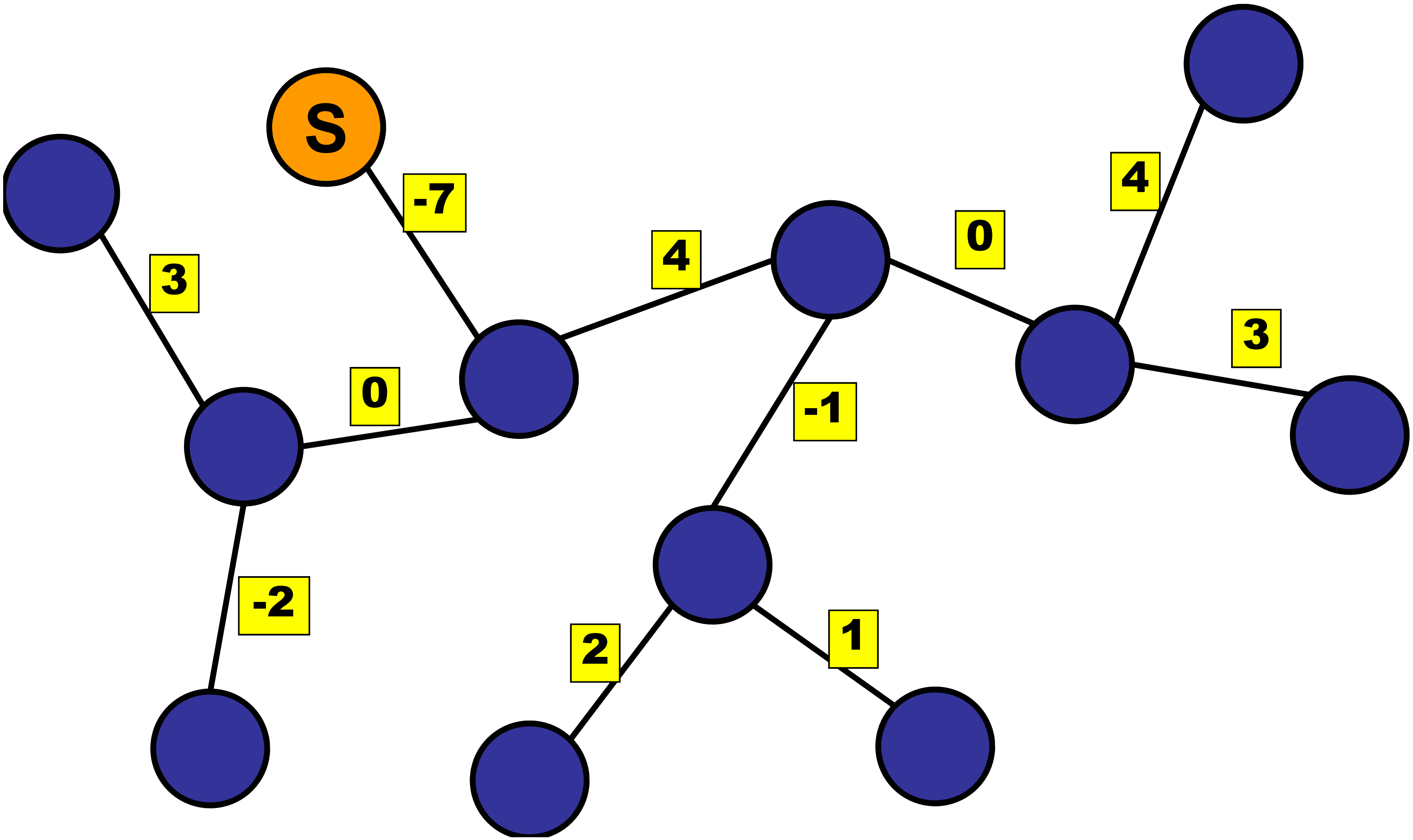
Relax the edges in the “right” order.

Only relax each edge once:

– O(E) cost (for relaxation step).

# Shortest Path: Tree

Undirected, weighted



# Aside: Trees, Redefined

## What is an (undirected) tree?

- A graph with no cycles is an (undirected) tree.

## What is a *rooted* tree?

- A tree with a special designated root note.

## Our previous (recursive) definition of a *tree*:

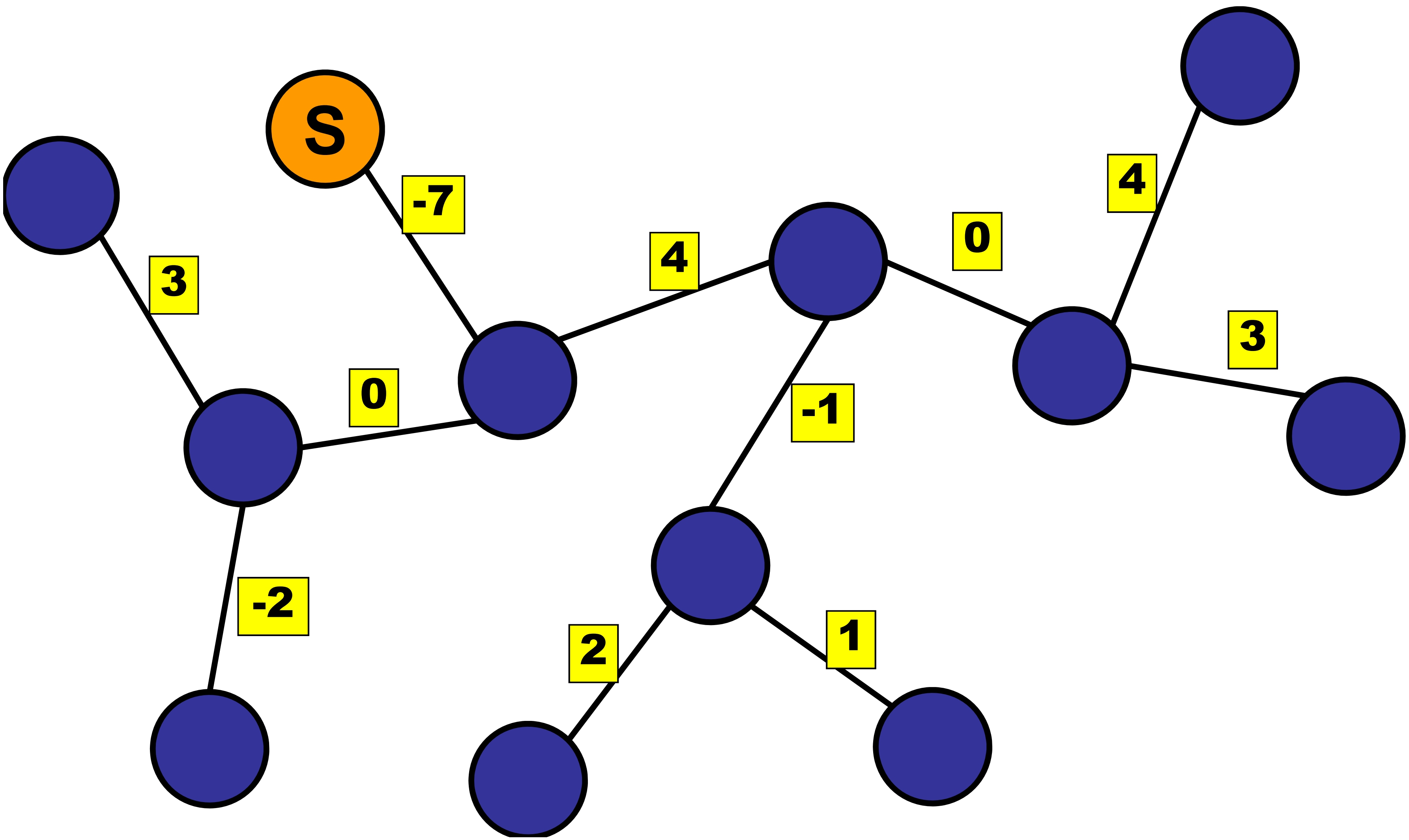
- A node with zero, one, or more sub-trees.
- I.e., a *rooted* tree.

# Shortest Path: Tree

ARCHIPELAGO

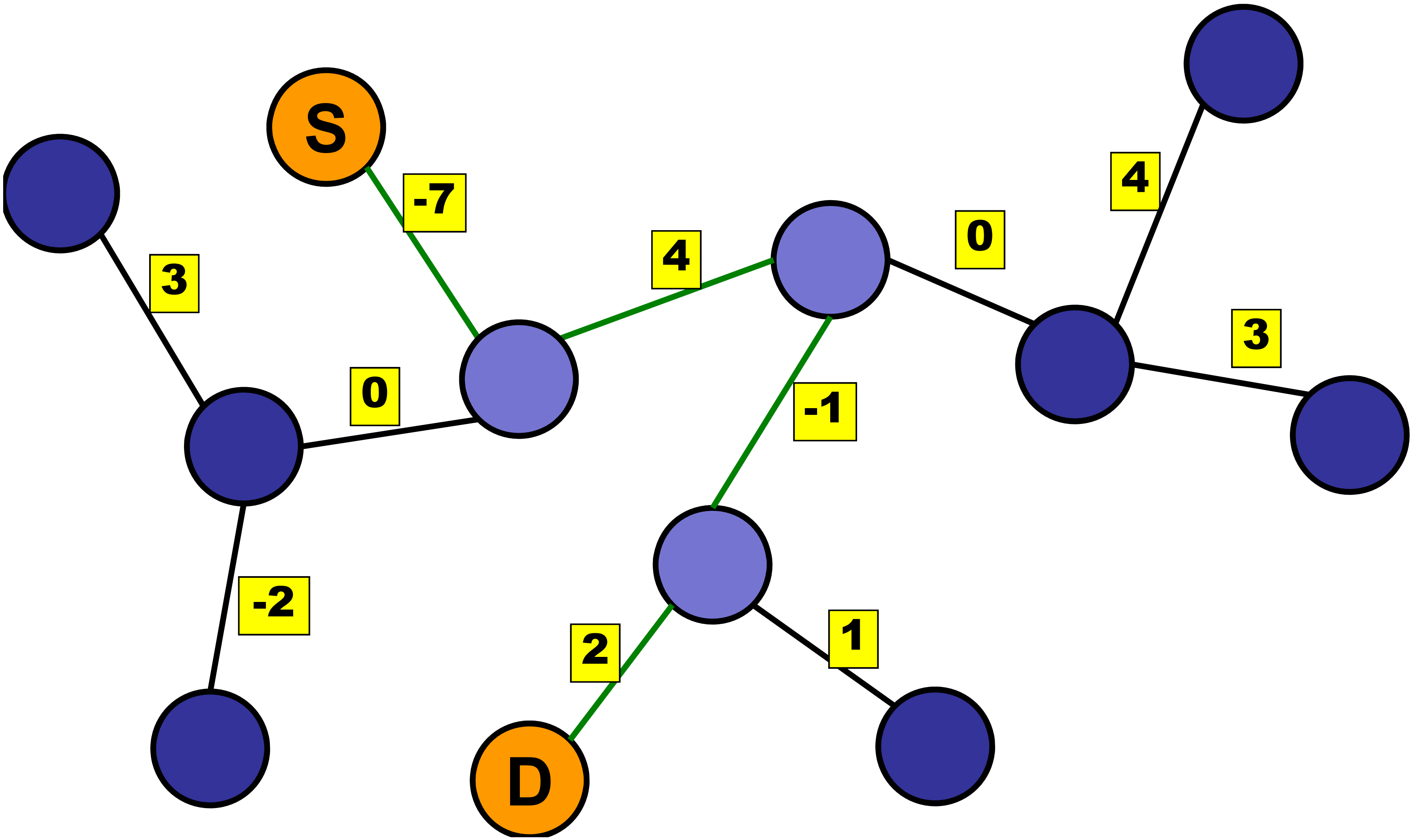
is open

What order to relax the edges?



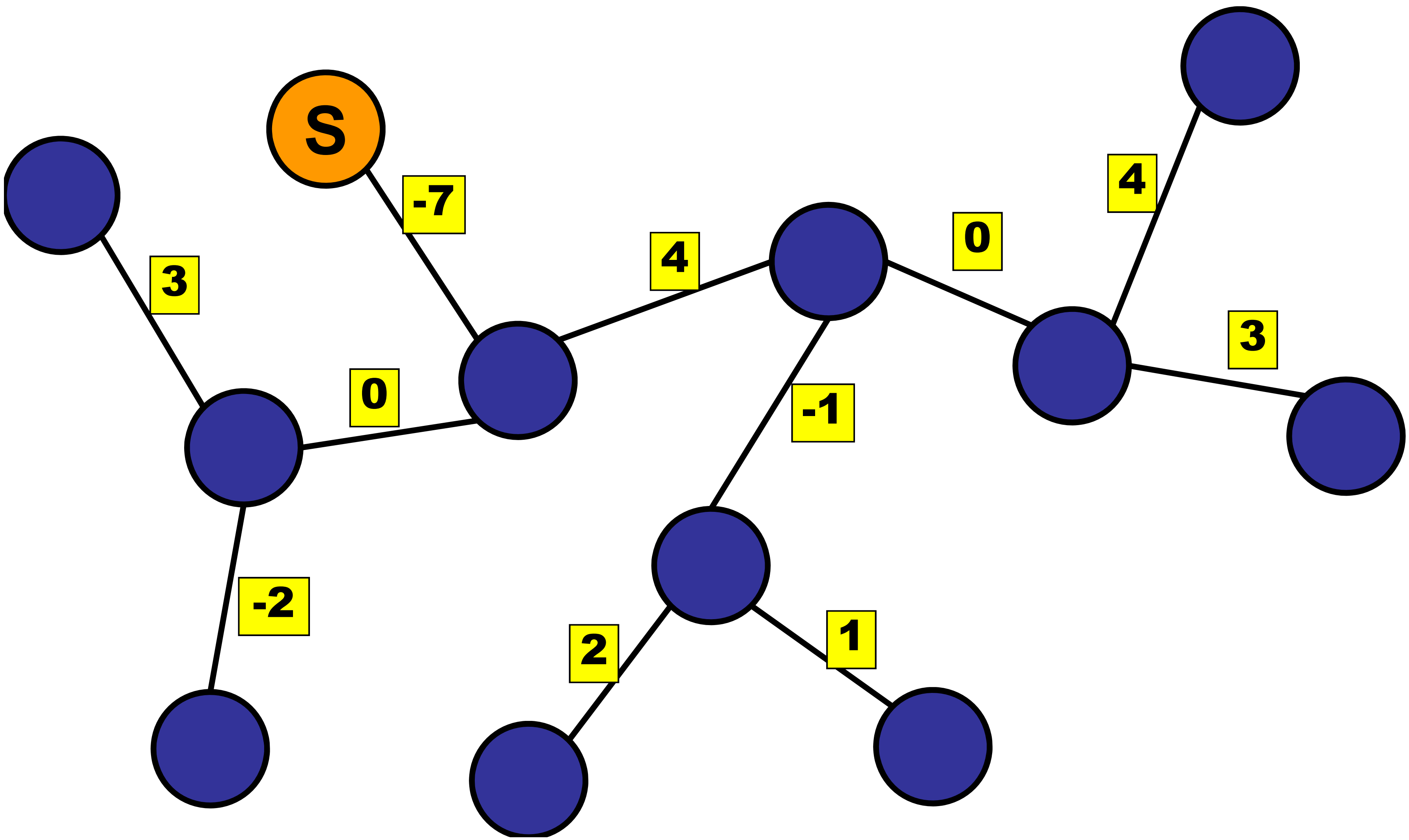
# Shortest Path: Tree

source-to-destination: only one possible path!



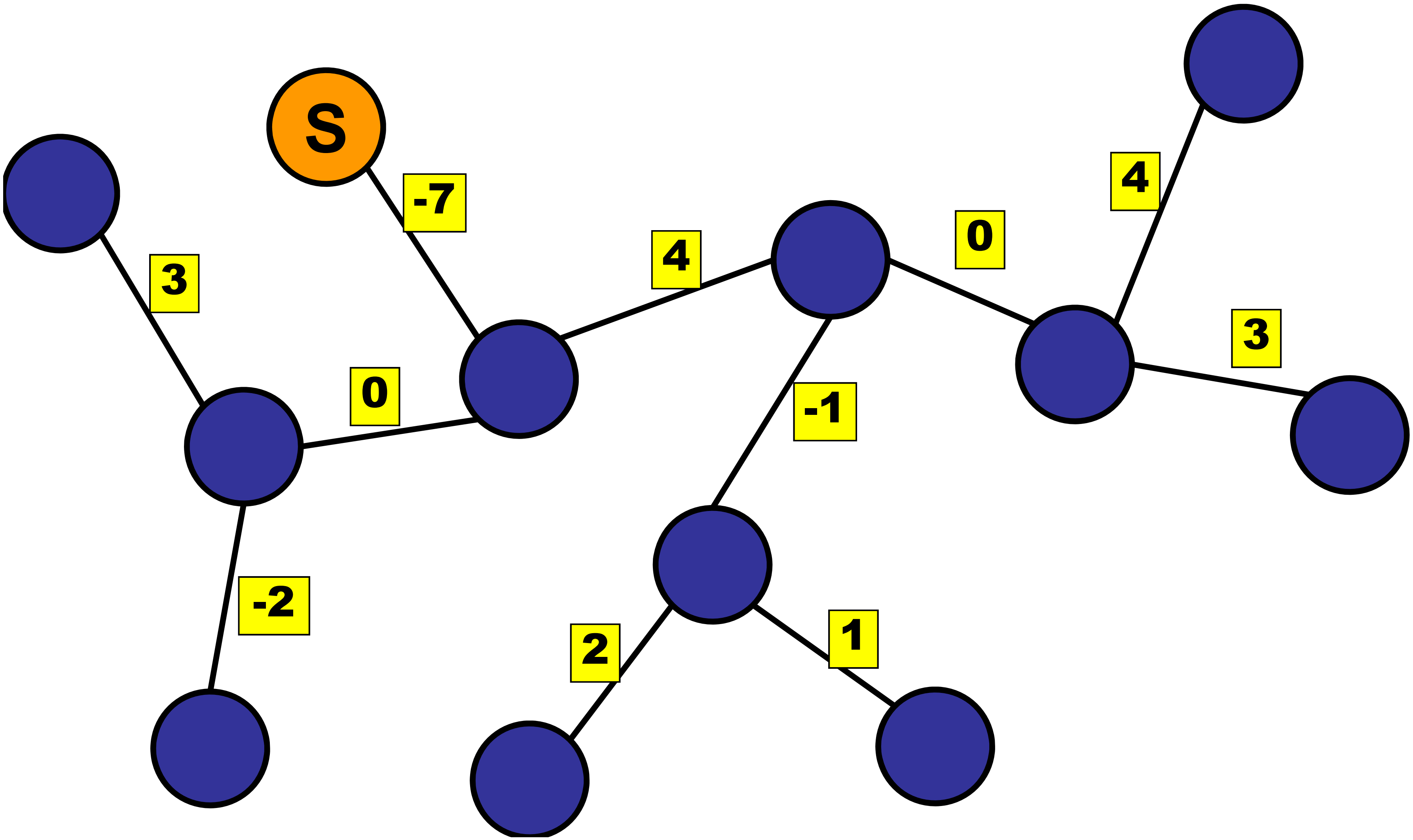
# Shortest Path: Tree

source-to-all: what order to relax?



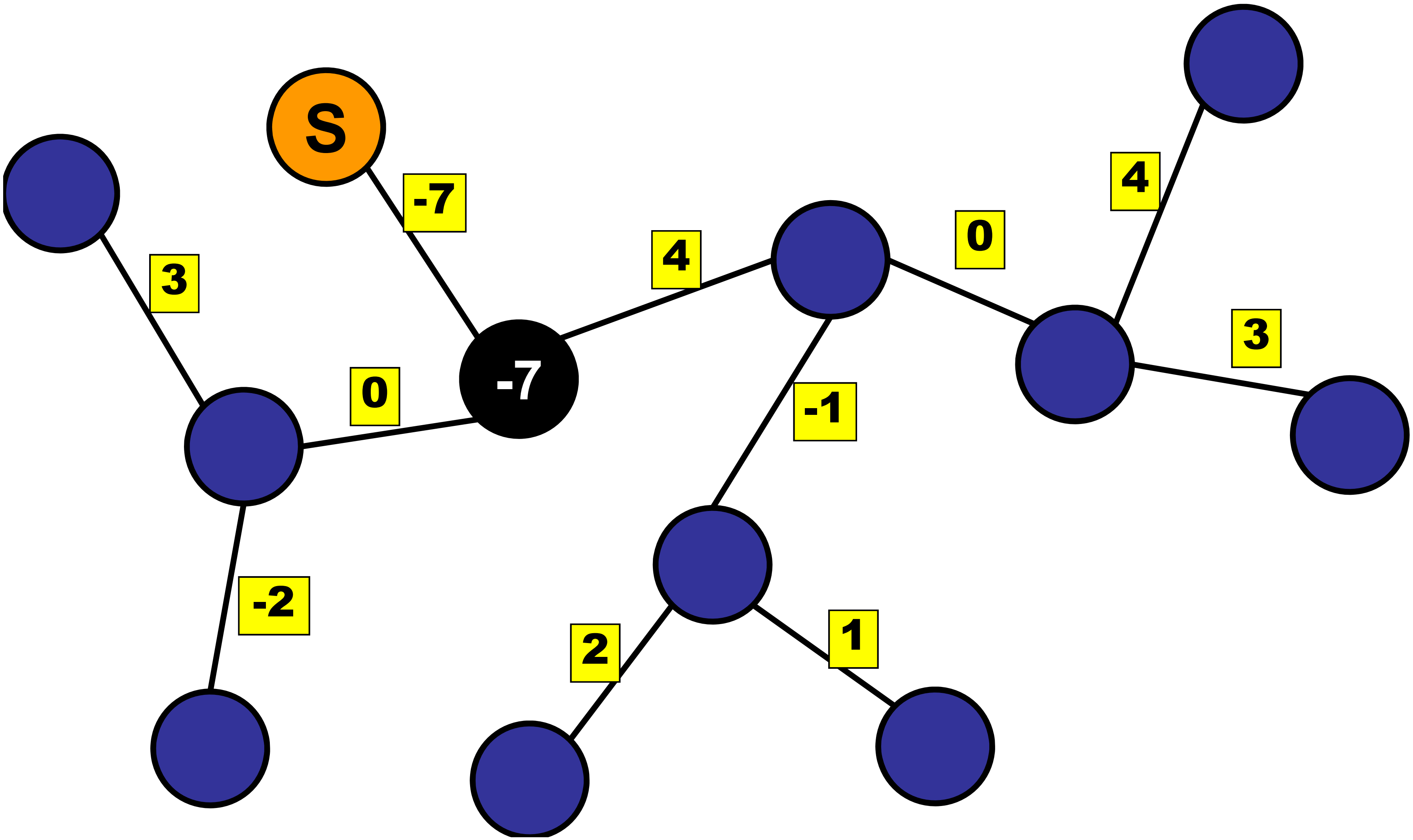
# Shortest Path: Tree

Relax edges in (BFS or DFS order).



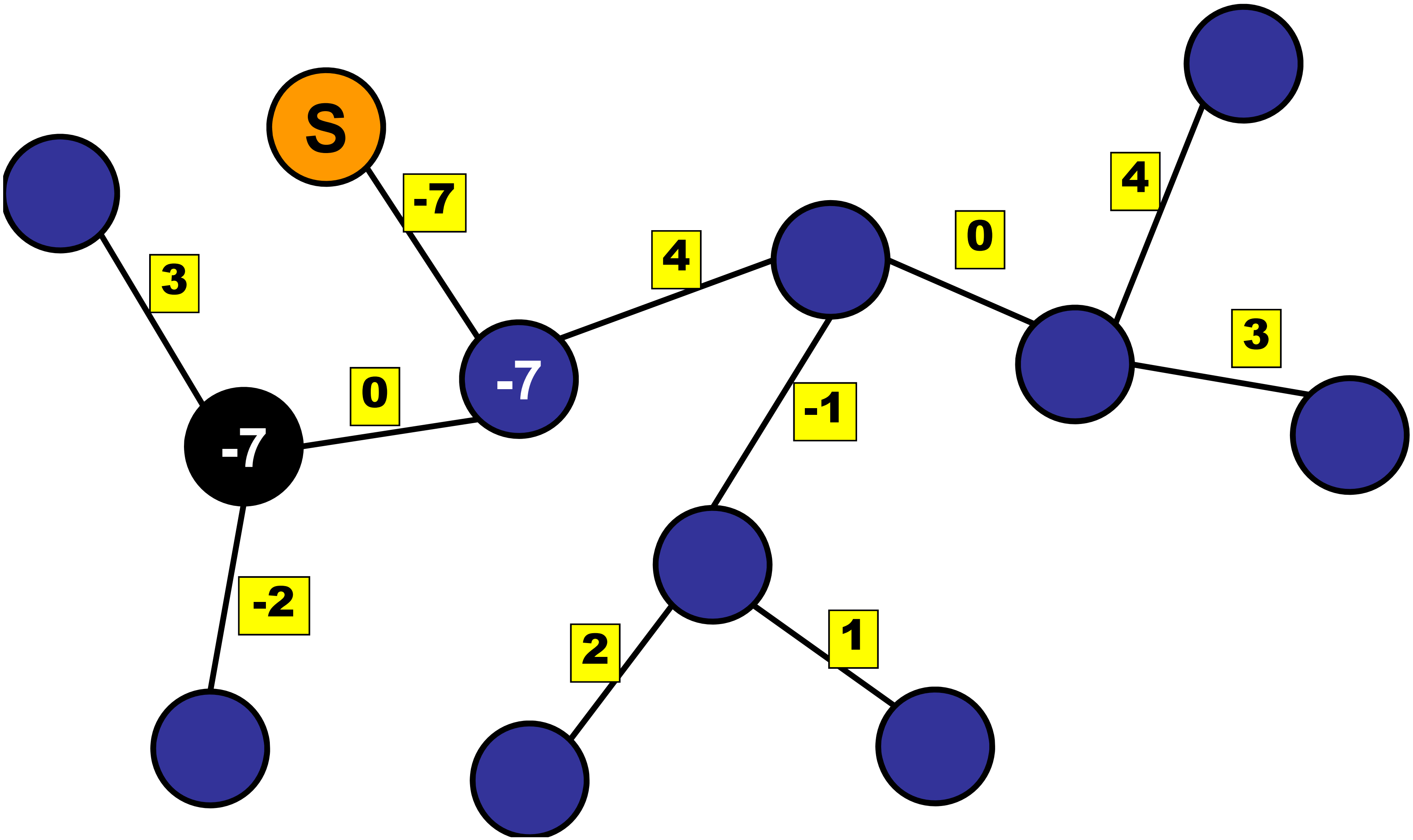
# Shortest Path: Tree

Relax edges in DFS order.



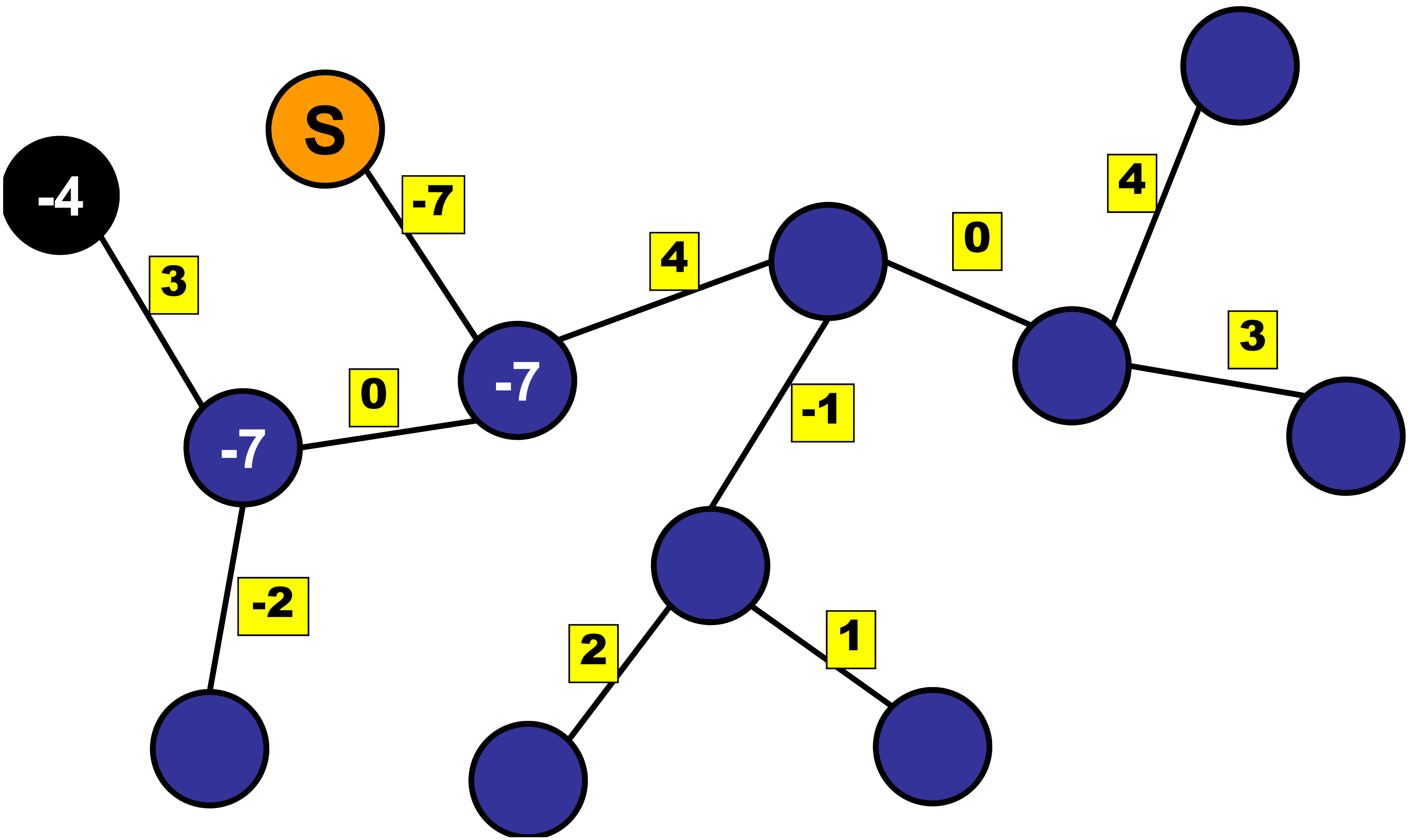
# Shortest Path: Tree

Relax edges in DFS order.



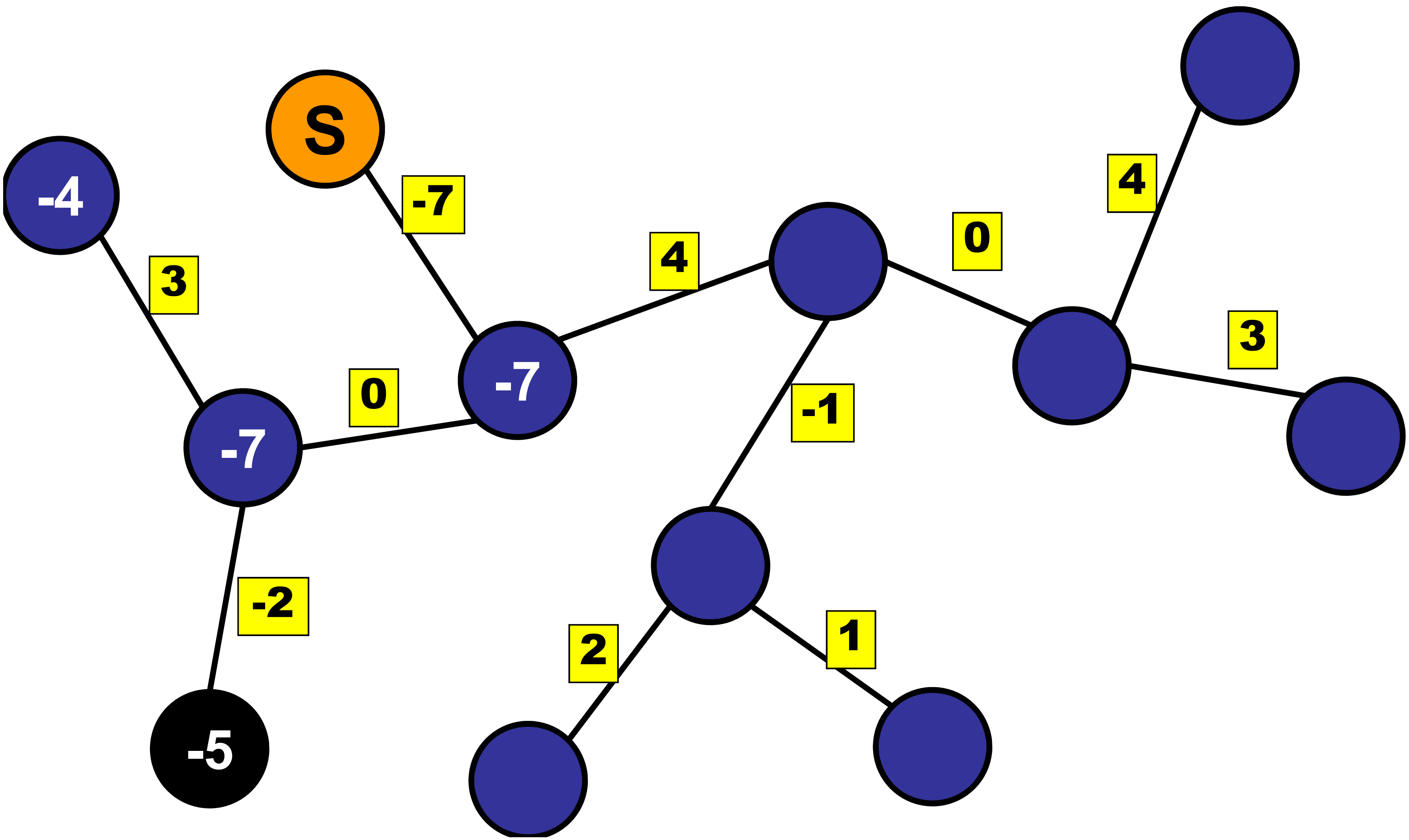
# Shortest Path: Tree

Relax edges in DFS order.



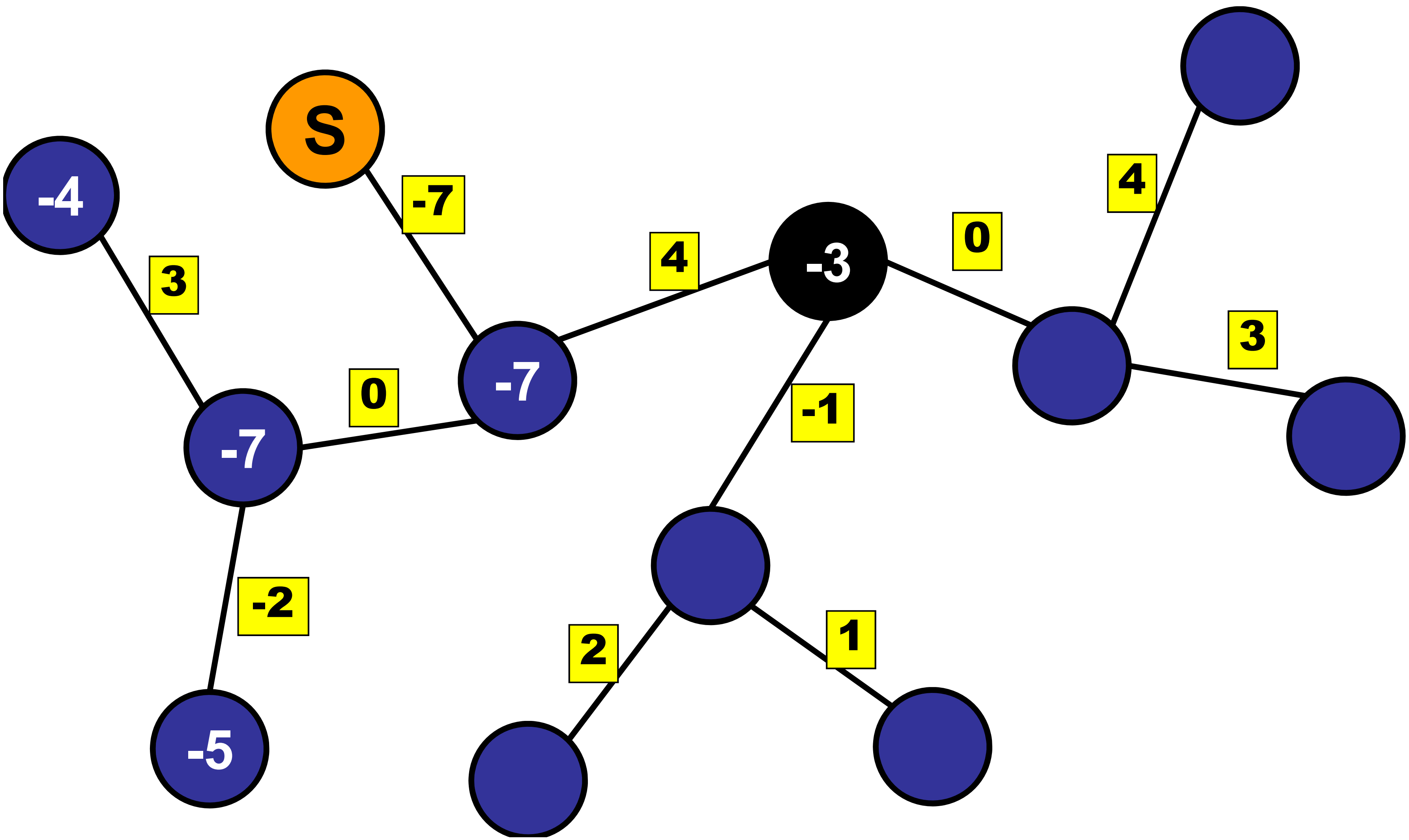
# Shortest Path: Tree

Relax edges in DFS order.



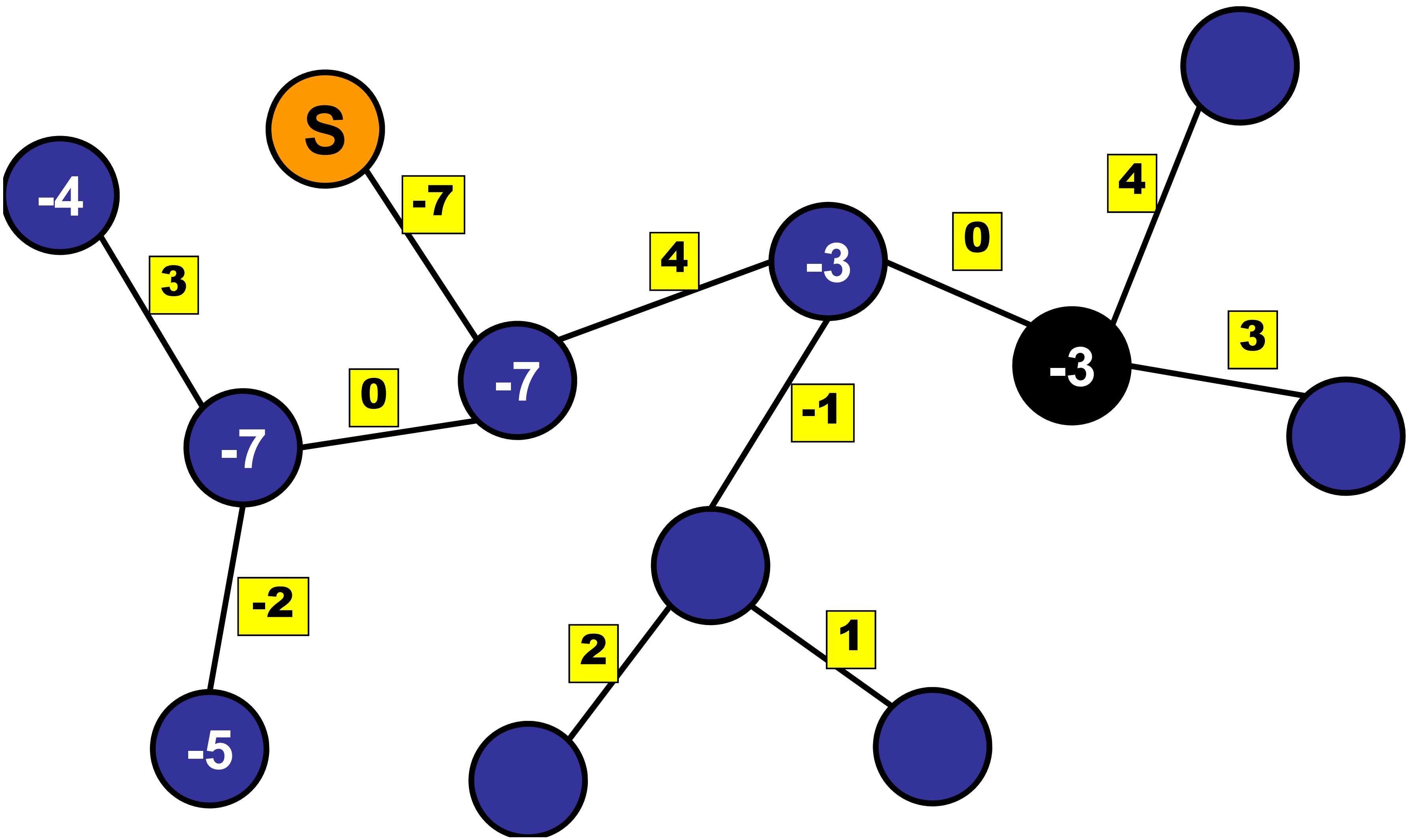
# Shortest Path: Tree

Relax edges in DFS order.



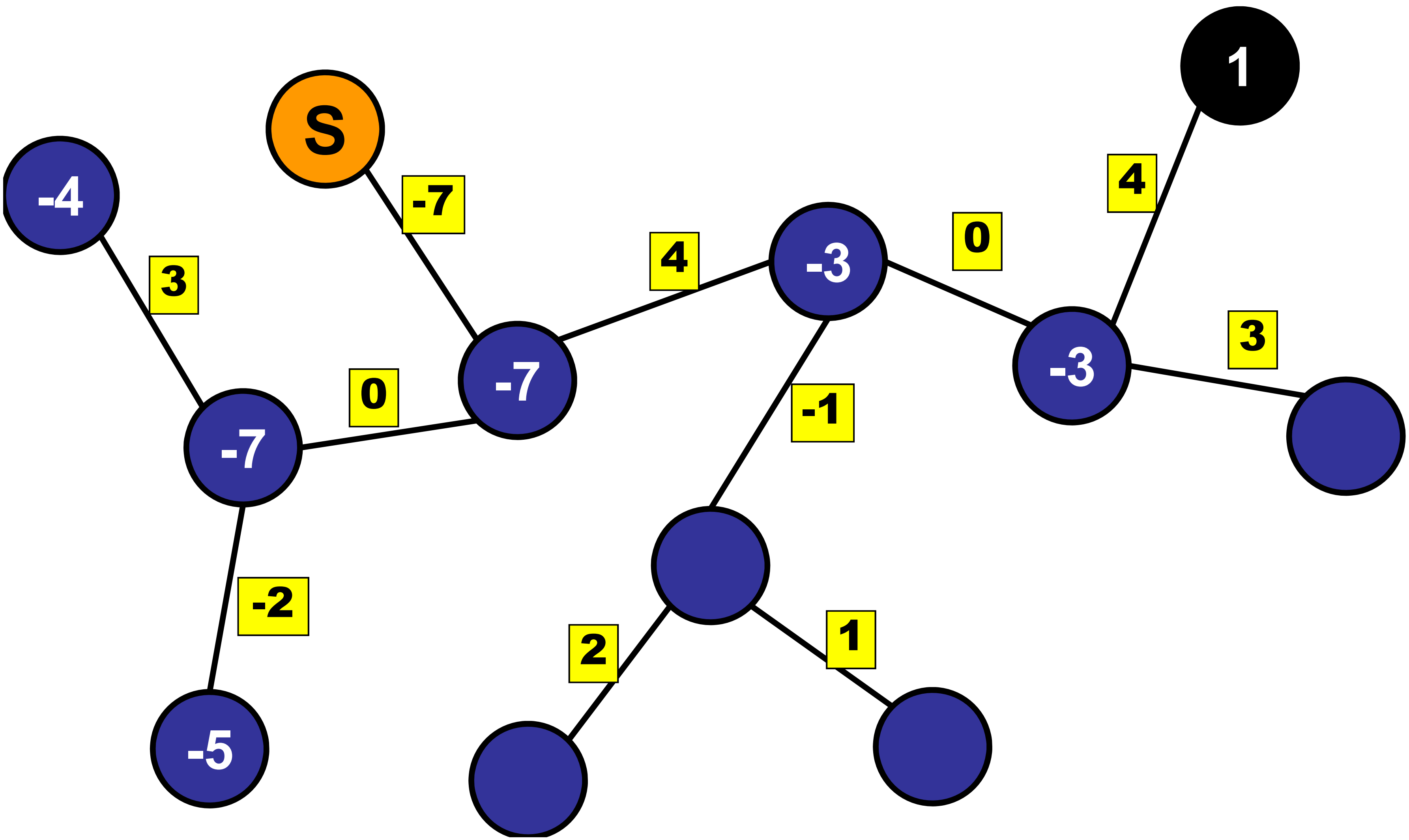
# Shortest Path: Tree

Relax edges in DFS order.



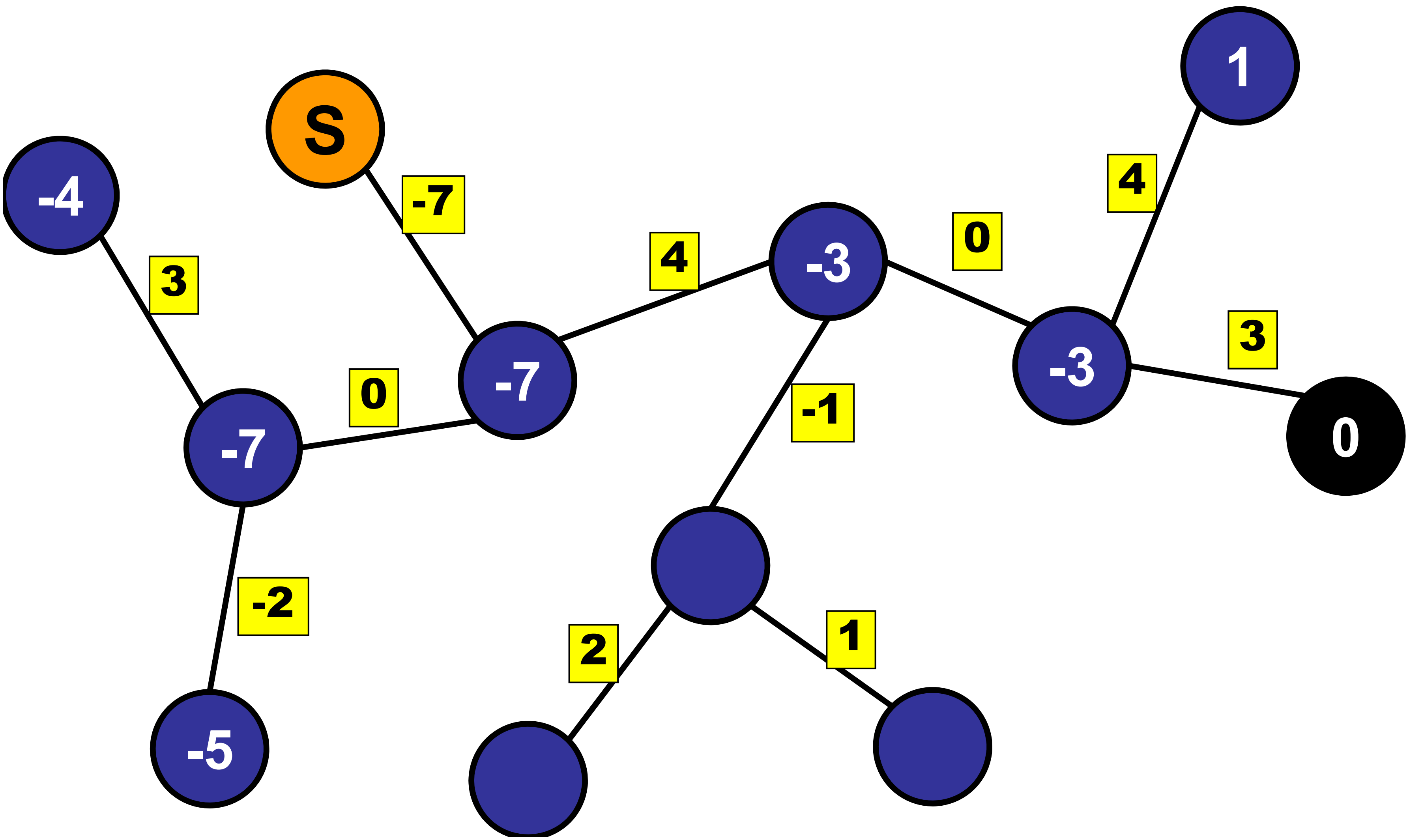
# Shortest Path: Tree

Relax edges in DFS order.



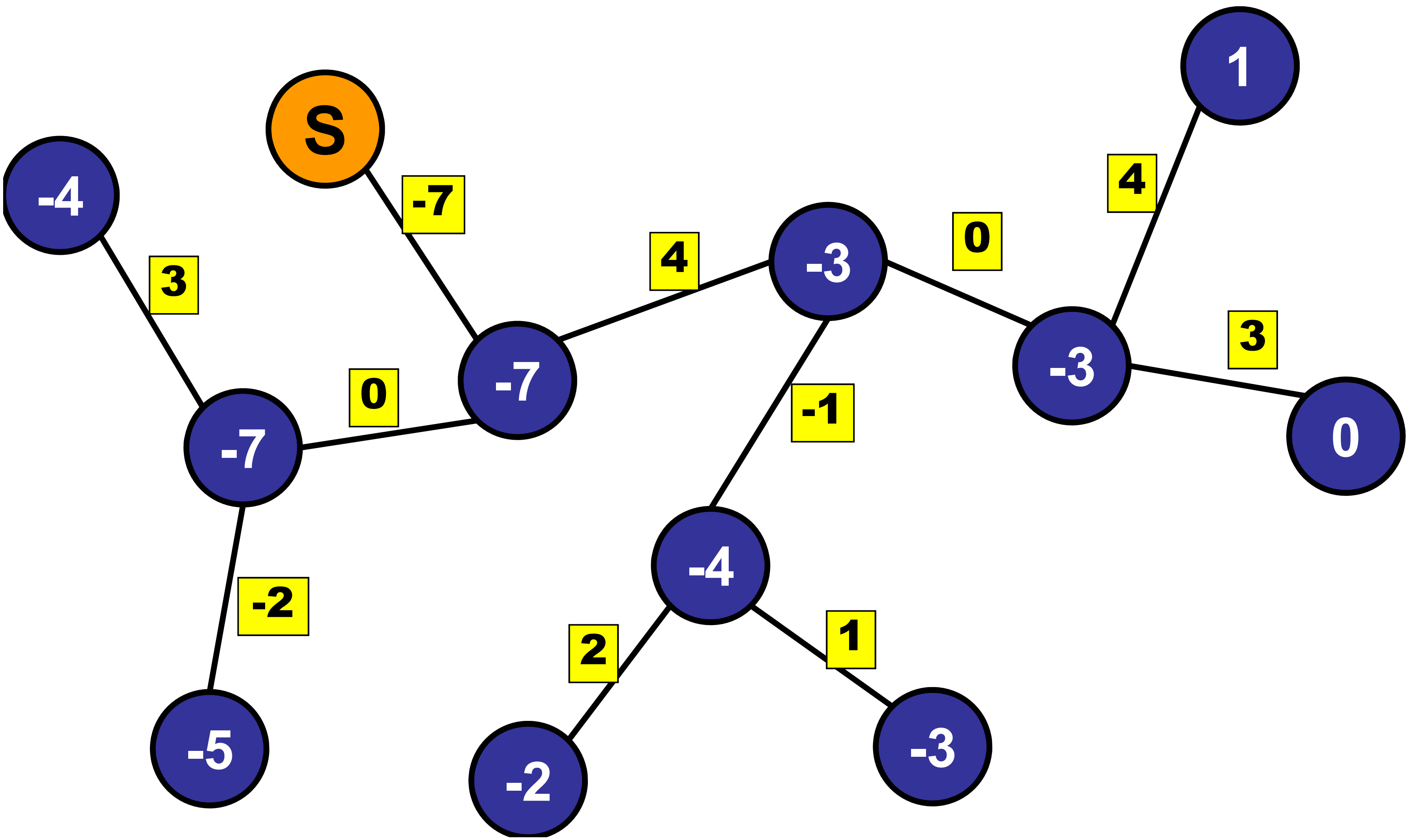
# Shortest Path: Tree

Relax edges in DFS order.



# Shortest Path: Tree

Relax edges in DFS order.



# Shortest Path: Tree

Basic idea:

- Perform DFS or BFS
- Relax each edge the first time you see it.
- $O(V)$ time.

Assumptions:

- Weighted edges
- Positive or negative weights
- Undirected tree

## Why is the running time O(V)?

1. You only need to explore 1 outgoing edge for each vertex.
2. DFS/BFS run in O(V) time on a graph.
3. ✓ There are only O(V) edges in a tree.
4. It is not O(V): you need to explore every edge!
5. I'm confused.

# Shortest Path: Tree

Basic idea:

- Perform DFS or BFS
- Relax each edge the first time you see it.
- $O(V)$ time.

Assumptions:

- Weighted edges
- Positive or negative weights
- Undirected tree

# Plan for today:

Directed Acyclic Graphs (DAG)

Topological Order

Topological Sort

Shortest Path in a DAG

Shortest Path in a tree

# Shortest Path Summary:

| Graph Type | Algorithm | Time |
|---|---|---|
| No negative weight cycles | Bellman-Ford | O(VE) |
| No negative edges | Dijkstra | O(E log V) |
| No directed cycles | TopoSort + Relax | O(E) |
| No cycles | DFS + Relax | O(V) |
| Planar… | | |
| Bounded arboricity… | | |
| Minor-Free Graphs… | | |

# Plan for today:

Directed Acyclic Graphs (DAG)

Topological Order

Topological Sort

Shortest Path in a DAG

Shortest Path in a tree