CS2040S

Data Structures and Algorithms

BFS, DFS, and Directed Graphs!

Puzzle of the Week

- What color is square (0,0)?
- How does this puzzle relate to the Fibonacci Sequence?

	0	1	2	3	4	5	6	7	8	9
0	?	gray	yellow	gray	yellow	red	yellow	gray	yellow	gray
1	gray	gray	gray	green	gray	gray	gray	gray	green	gray
2	yellow	green	yellow	gray	yellow	gray	blue	gray	yellow	gray
3	gray	gray	gray	gray	green	gray	gray	gray	gray	green
4	yellow	gray	blue	gray	yellow	gray	yellow	green	yellow	gray
5	red	gray	gray	gray	gray	red	gray	gray	gray	gray
6	yellow	gray	yellow	green	yellow	gray	yellow	gray	blue	gray
7	gray	green	gray	gray	gray	gray	green	gray	gray	gray
8	yellow	gray	yellow	gray	blue	gray	yellow	gray	yellow	green
9	gray	gray	green	gray	gray	gray	gray	green	gray	gray

Roadmap

Last time: Graph Basics

- What is a graph?
- Modeling problems as graphs.
- Graph representations (list vs. matrix)
- Searching graphs: BFS

Contest: Speed Demon

Make it fast! Due: Wednesday

- Process your data faster than a speeding bullet.
- Can you perceive the world in less than an attosecond?

(Upload entries to the competition server as well as Coursemology.)

Problem Set 7 (3 parts)

Find your way out of the maze, if you can!

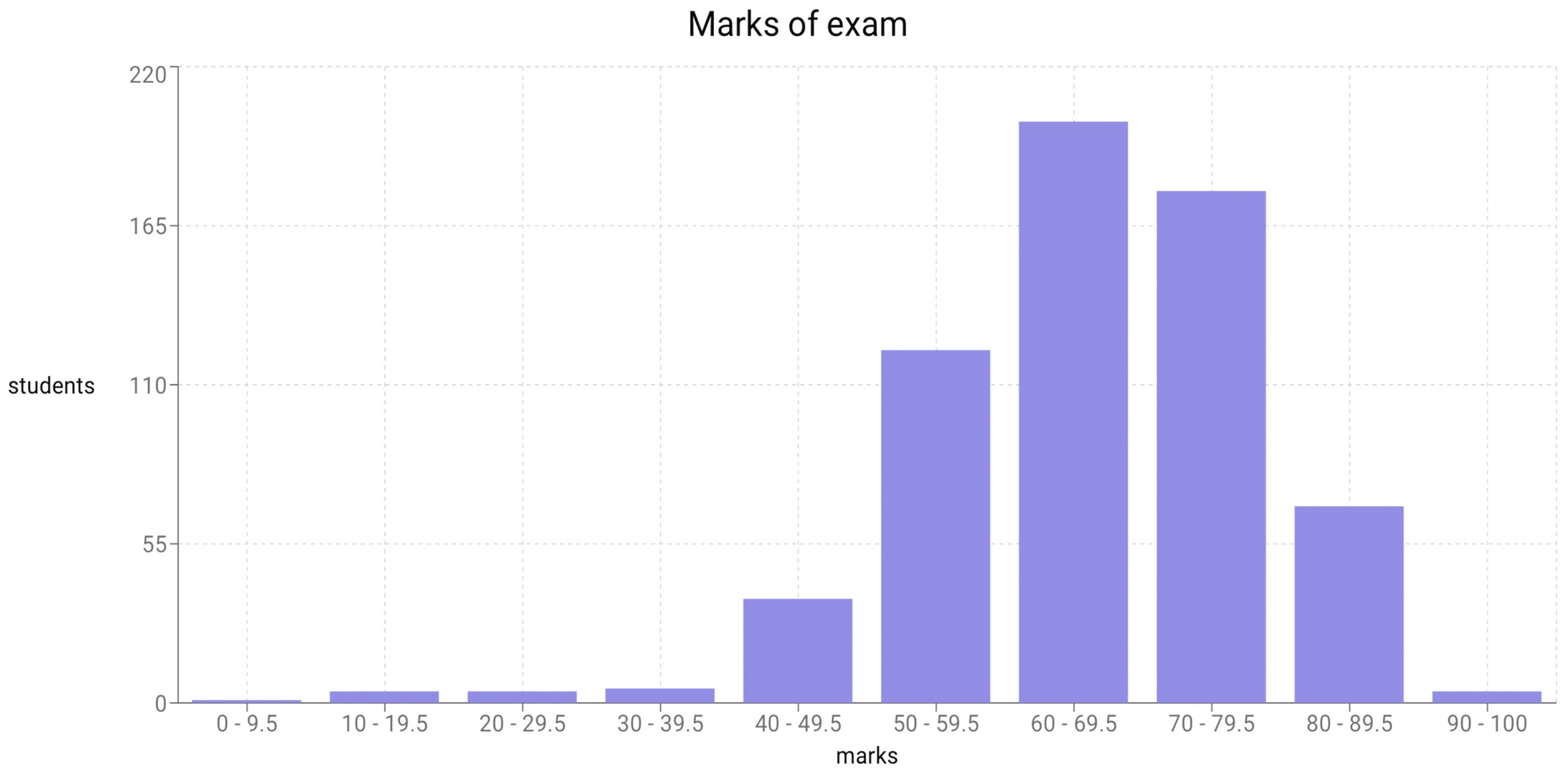
Which superhero are you?

Use your powers to escape!

But don't let fear of the dark

And scary spirits overwhelm you!

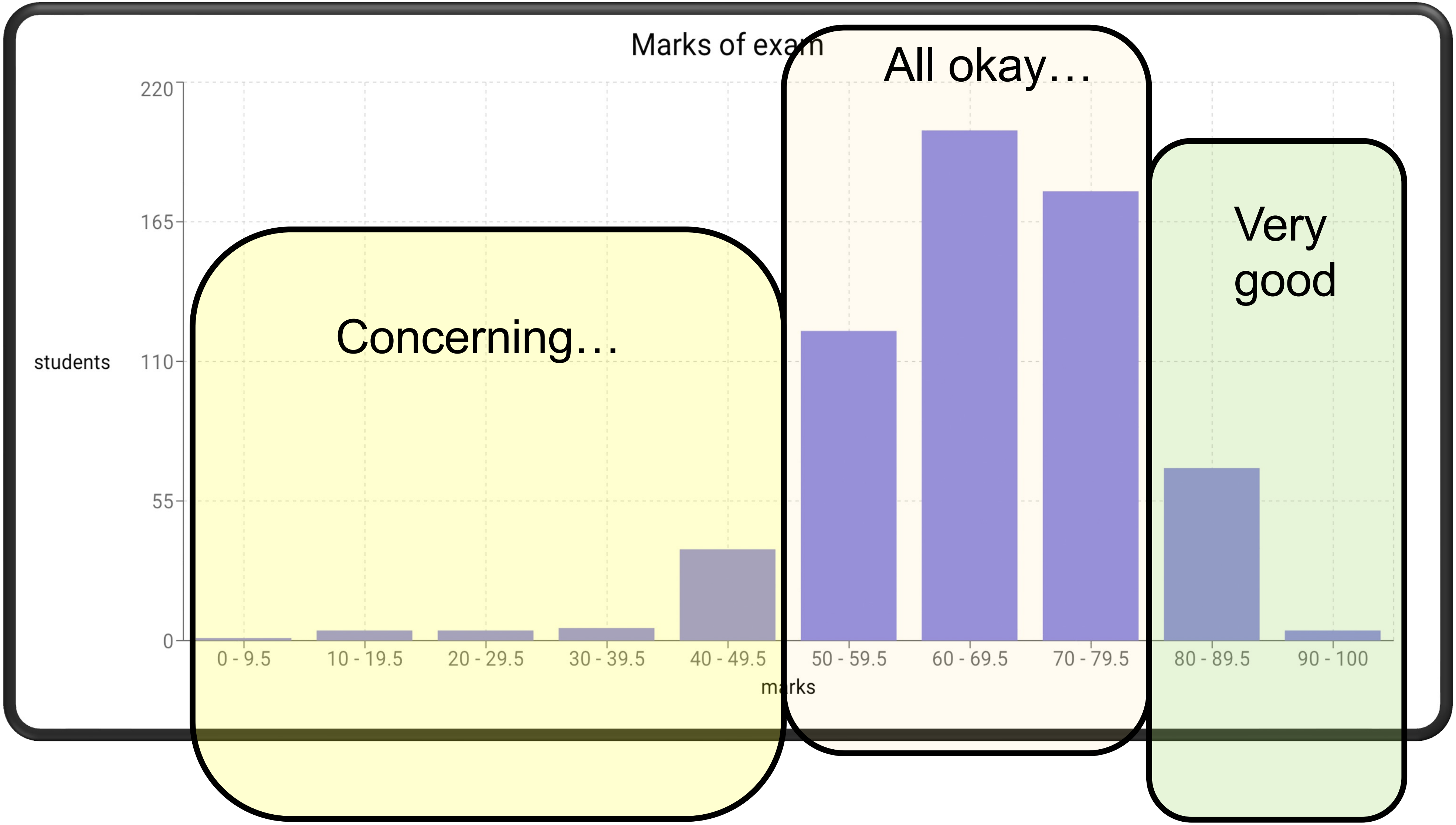
Midterm Exam



Average: 66
Median: 67

You will receive an e-mail with a link to your graded script by tomorrow night.

Midterm Exam



How to use the midterm?

What don’t you understand?

- AVL trees seem to confuse many.
- Invariants seem difficult.
- Rank/Select augmented trees seems very hard.

How can you prepare better for the final?

- Think about how algorithms work.
- Think about invariants and properties of algos.
- Think about how to use algorithms.

Grading issues

What to do if answer was mis-scanned?

- Fill out form on Coursemology.
- If you *clearly* indicated one answer, and the script is marked for a different answer, fill out the form.
- I will notify you when your request is processed.

(I did a fair amount of manual work to check that the scripts were correctly scanned, but I'm sure there are at least a few I missed.)

Grading issues

What to do if you disagree with an answer?

- Fill out form on Coursemology.
- Note that I am unlikely to change an answer, unless there is a mistake in the question.
- Only do this if you think there is an actual mistake in the question, not just because you don’t like it.
- I will not respond individually. I will announce any changes made.

Grading issues

What if you misunderstood a question (or mis-bubbled) but knew the real answer?

- Sorry!
- We try to make questions as clear as possible.
- But it is inevitable that there are some misunderstandings.
- Also, some questions *do* require a judgement call (e.g., "*Is algorithm x efficient*?"). That's intentional.
- Luckily, many questions. Even if you misunderstood one or two, it won't matter much!

Grading issues

What if you don’t understand why your solution is wrong (or why the real solution is right)?

- See my notes on the solutions (to be published).
- Discuss on the forum.
- Ask your tutor.
- Discuss in tutorial.

Do not e-mail me until you have tried those options. If you cannot resolve the issue on the forum or with tutor, then talk to me or recitation instructor.

Searching a Graph

Goal:

- Start at some vertex **s** = start.
- Find some other vertex **f** = finish.

  Or: visit **all** the nodes in the graph;

Two basic techniques:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)

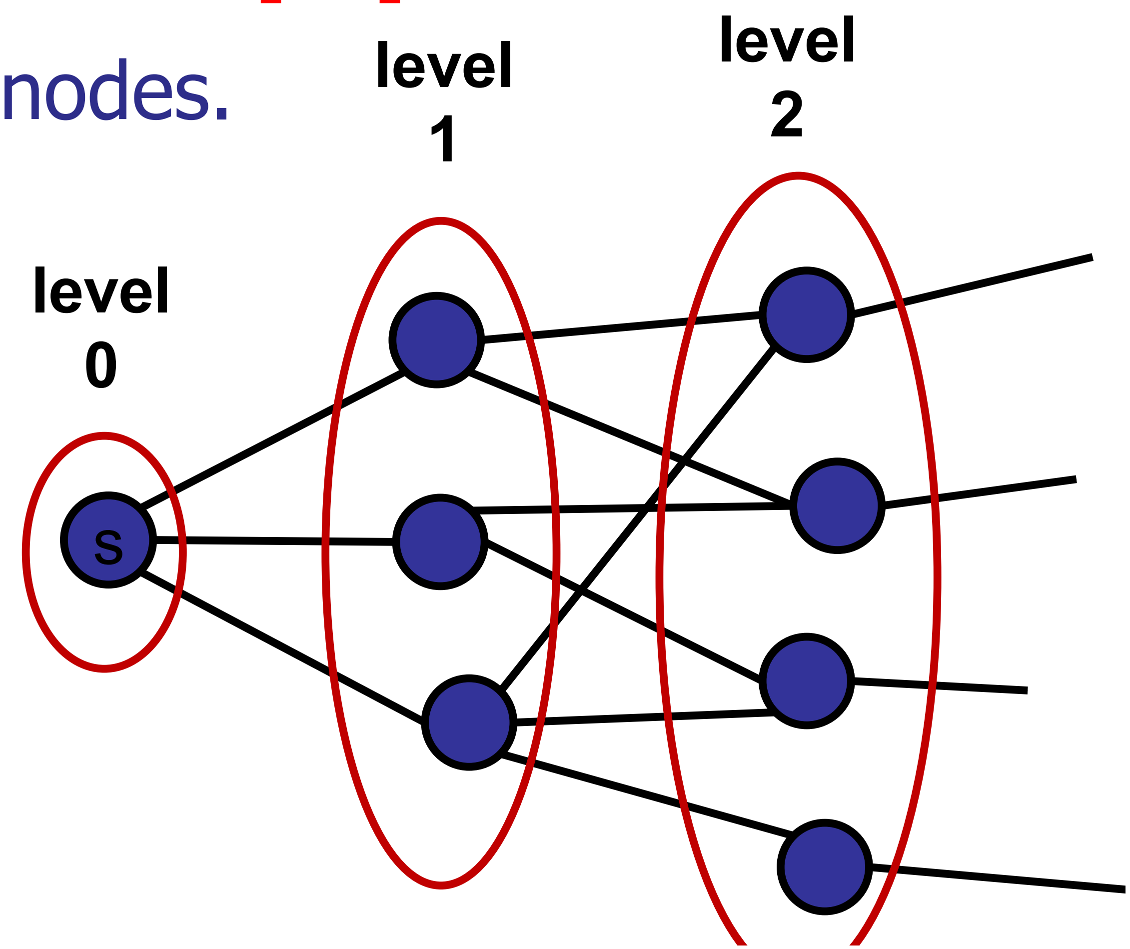
Graph representation:

- Adjacency list

Searching a graph

Breadth-First Search:

- Explore graph level by level.
- Calculate level[i] from level[i-1]
- Skip already visited nodes.



Searching a graph

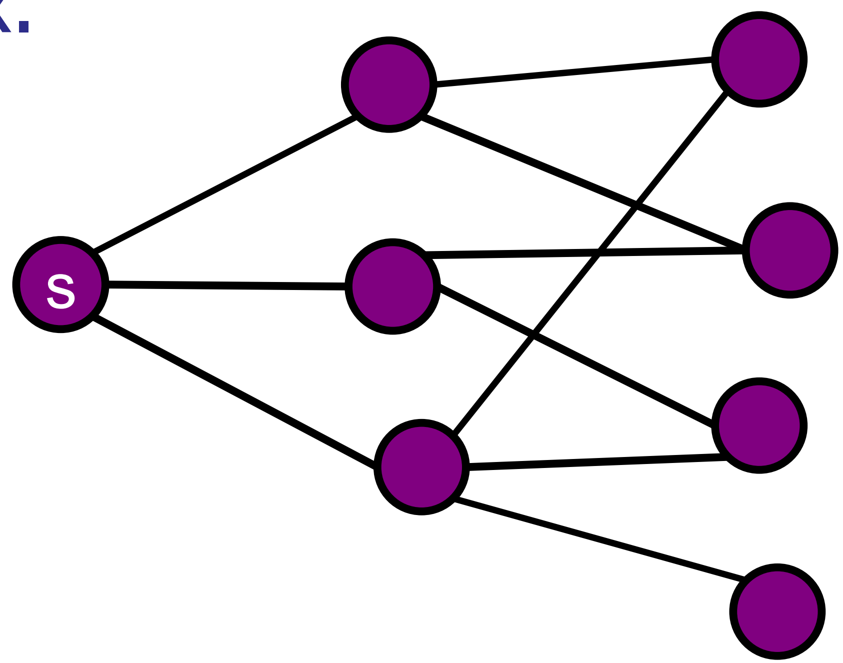
Breadth-First Search:

```
frontier = {s}
while frontier is not empty:
   next-frontier = {}
   for each node u in the frontier:
      for each edge (u,v) in the graph:
          if v has not yet been visited, add v to next-frontier
   frontier = next-frontier
```

Searching a graph

Depth-First Search:

- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don’t repeat a vertex.



Searching a graph

Breadth-First Search:

```
dfs-visit(u)
   for each neighbor v of u:
        if v is not marked visited:
             mark v as visited
             dfs-visit(v)
```

Running Time

BFS: $O(V+E)$

- Each node is in only one frontier.
- Each node is only visited once.
- Each neighbor is only enumerated once.

DFS: $O(V+E)$

- DFS-visit called only once per node.
- Each neighbor is enumerated once.

$O(V)$

$O(E)$

Graph Search

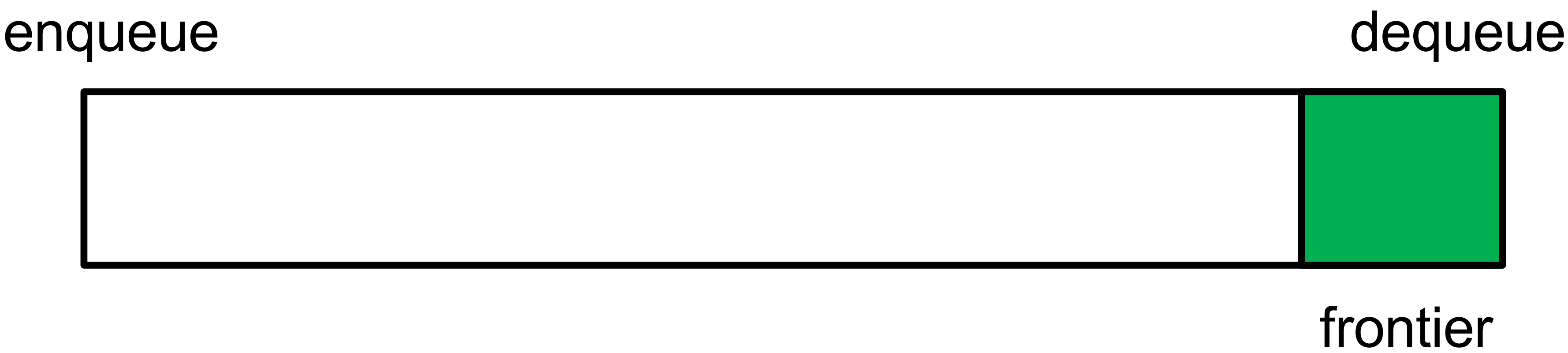
BFS and DFS are the same algorithm:

- BFS: use a queue
  - Every time you visit a node, add all unvisited neighbors to the queue.
- DFS: use a stack
  - Every time you visit a node, add all unvisited neighbors to the stack.

Searching a graph

Breadth-First Search:

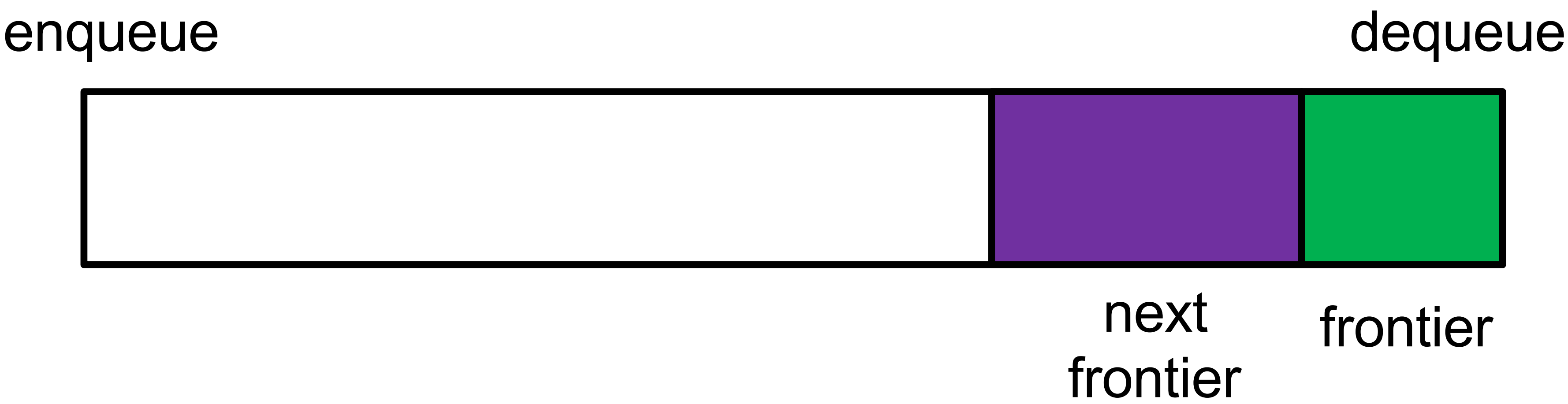
```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
            if v has not yet been visited, add Queue.enqueue(v)
```



Searching a graph

Breadth-First Search:

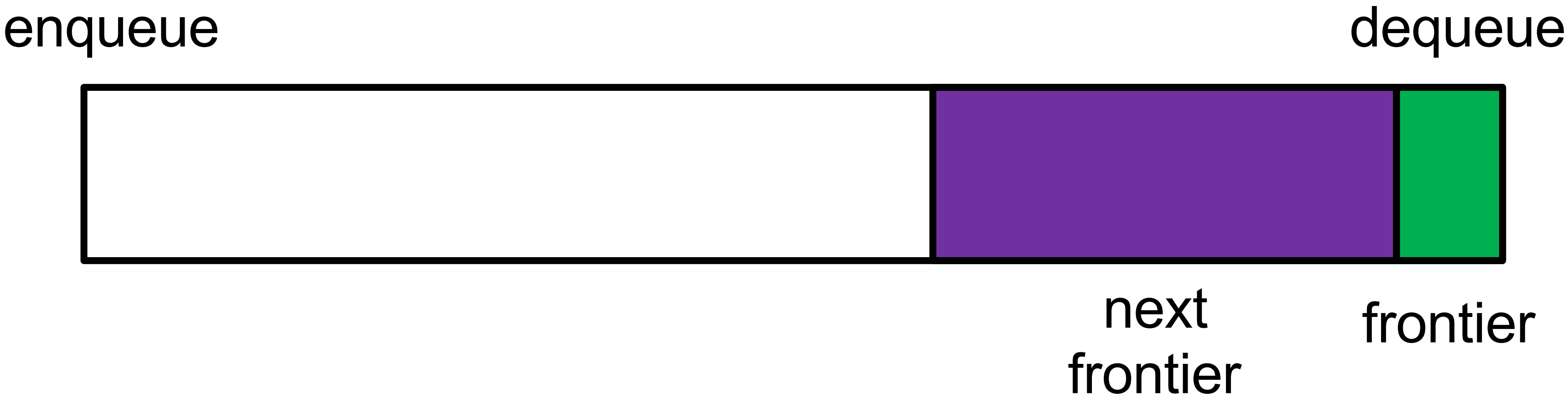
```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
          if v has not yet been visited, add Queue.enqueue(v)
```



Searching a graph

Breadth-First Search:

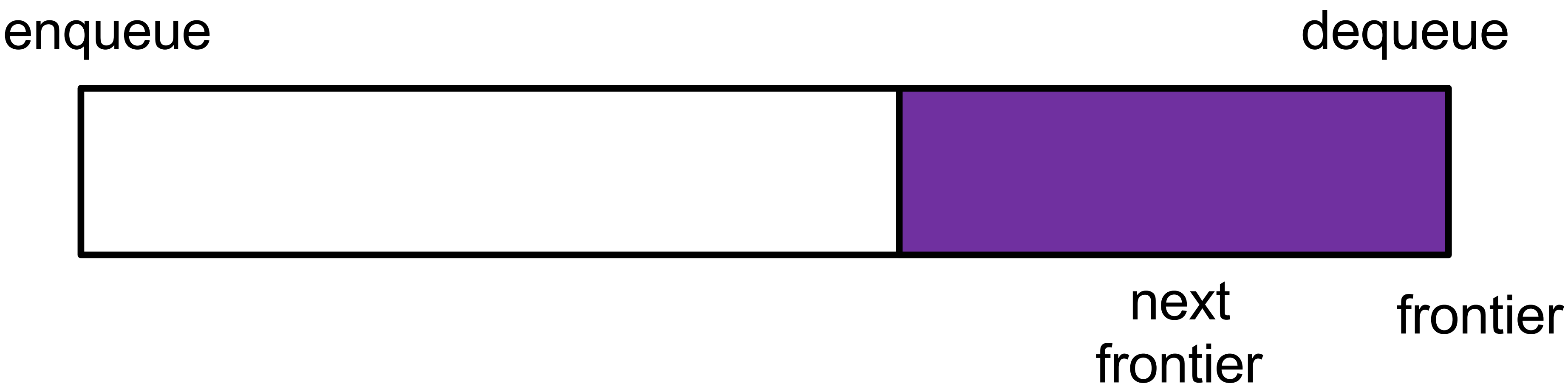
```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
          if v has not yet been visited, add Queue.enqueue(v)
```



Searching a graph

Breadth-First Search:

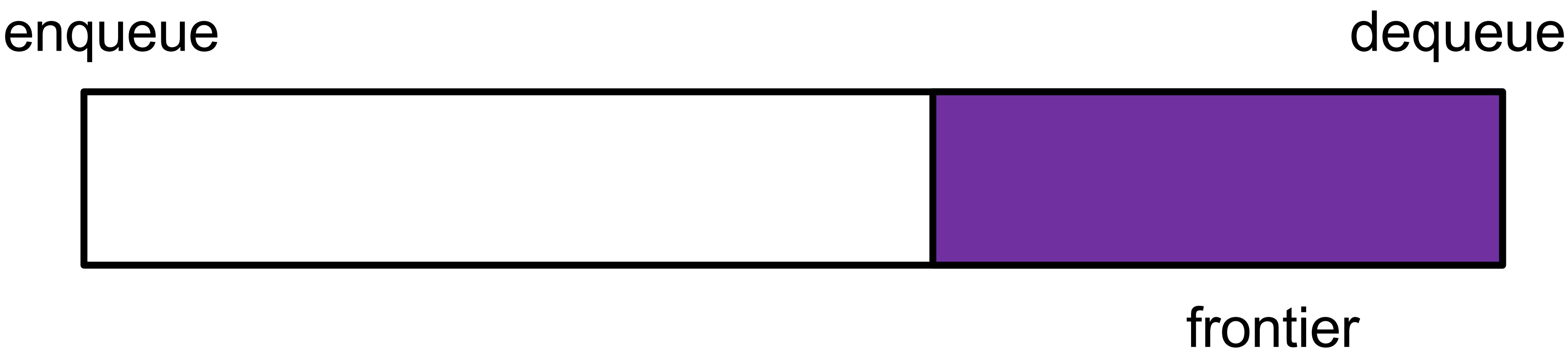
```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
          if v has not yet been visited, add Queue.enqueue(v)
```



Searching a graph

Breadth-First Search:

```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
           if v has not yet been visited, add Queue.enqueue(v)
```



Graph Search

Breadth-first search:

Same algorithm, implemented with a queue:

Add start-node to queue.

Repeat until queue is empty:

- Remove node v from the front of the queue.
- Visit v.
- Explore all outgoing edges of v.
- Add all unvisited neighbors of v to the queue.

Searching a graph

Breadth-First Search:

```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
           if v has not yet been visited, add Queue.enqueue(v)
```

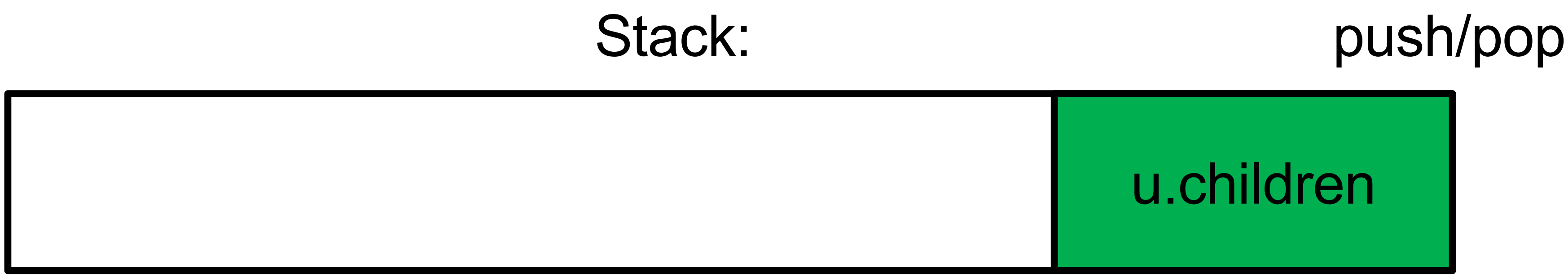
Depth-First Search:

```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
           if v has not yet been visited, add Stack.push(v)
```

Searching a graph

Depth-First Search:

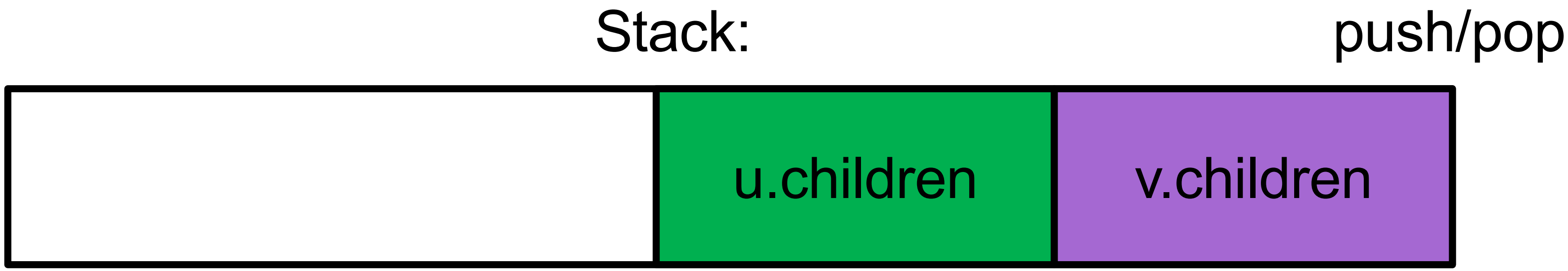
```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
         if v has not yet been visited, add Stack.push(v)
```



Searching a graph

Depth-First Search:

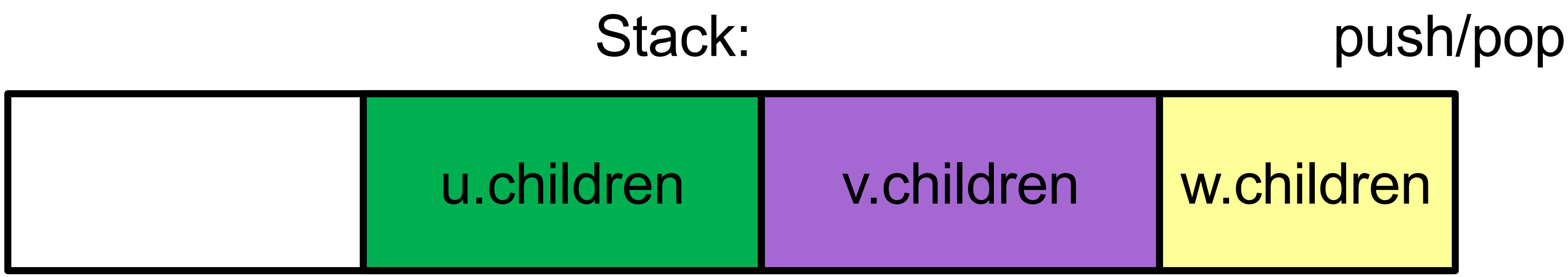
```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
           if v has not yet been visited, add Stack.push(v)
```



Searching a graph

Depth-First Search:

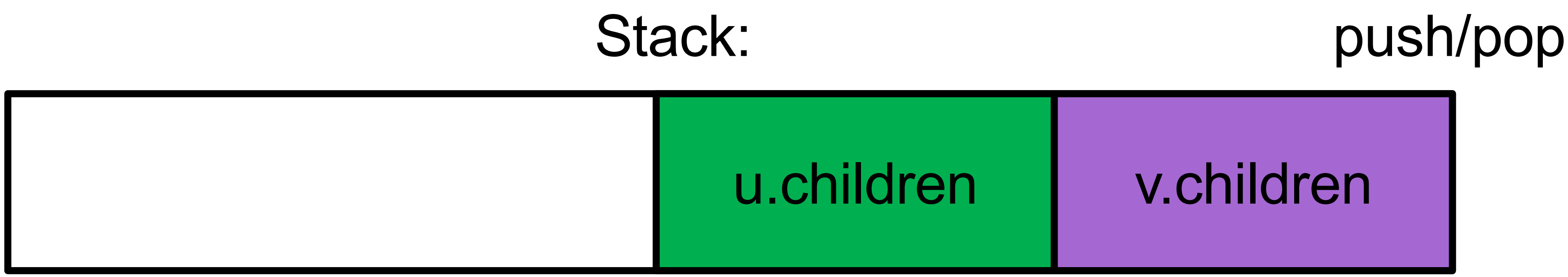
```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
          if v has not yet been visited, add Stack.push(v)
```



Searching a graph

Depth-First Search:

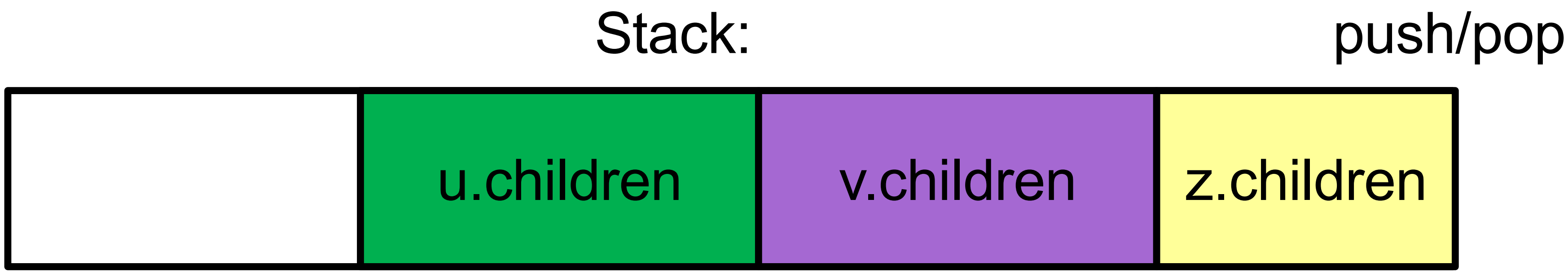
```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
          if v has not yet been visited, add Stack.push(v)
```



Searching a graph

Depth-First Search:

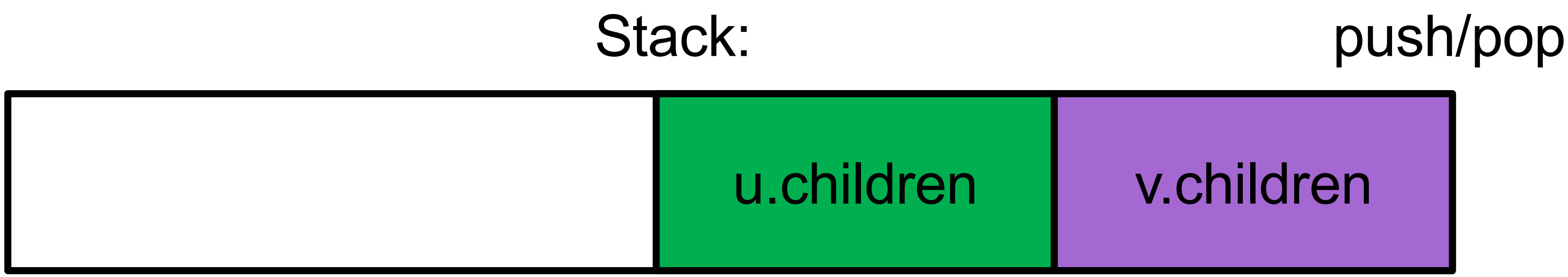
```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
          if v has not yet been visited, add Stack.push(v)
```



Searching a graph

Depth-First Search:

```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
            if v has not yet been visited, add Stack.push(v)
```



Searching a graph

Breadth-First Search:

```
Queue.enqueue(s)
while Queue is not empty:
    u = Queue.dequeue()
     for each edge (u,v) in the graph:
           if v has not yet been visited, add Queue.enqueue(v)
```

Depth-First Search:

```
Stack.push(s)
while Stack is not empty:
    u = Stack.pop()
     for each edge (u,v) in the graph:
           if v has not yet been visited, add Stack.push(v)
```

Common Mistake

What do BFS and DFS solve?

- They visit every node in the graph?
- They visit every edge in the graph?
- They visit every path in the graph?

ARCHIPELAGO is open

Common Mistake

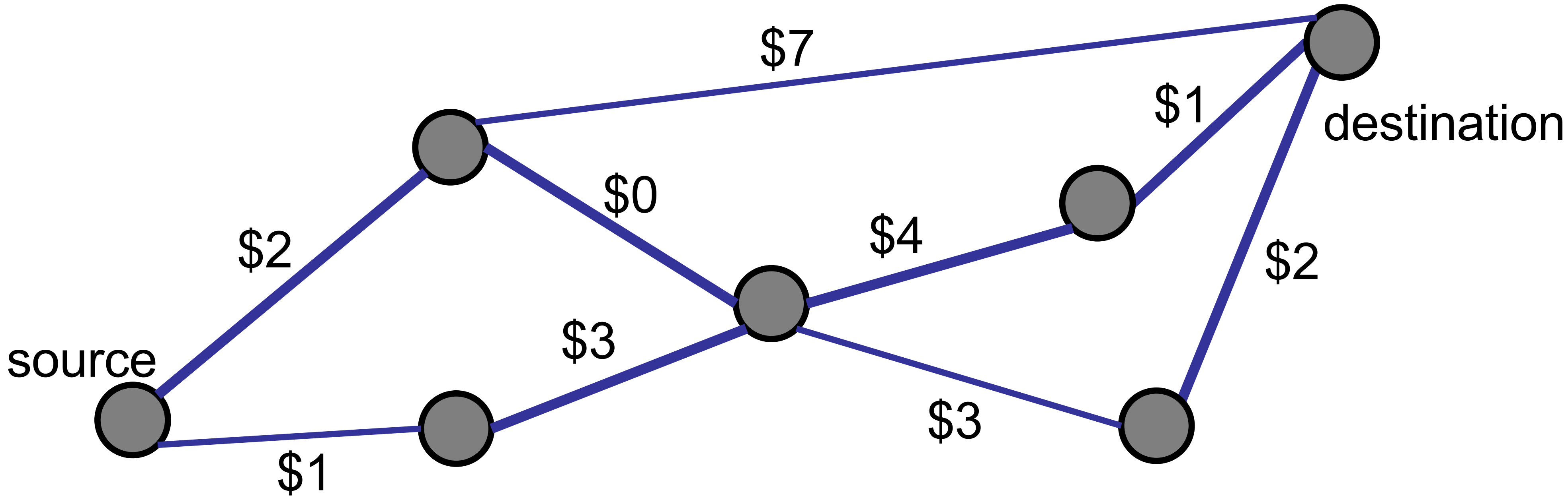
What do BFS and DFS solve?

- They visit every node in the graph? Yes.
- They visit every edge in the graph? Yes.
- ~~They visit every path in the graph?~~

Example

Problem: Make Money

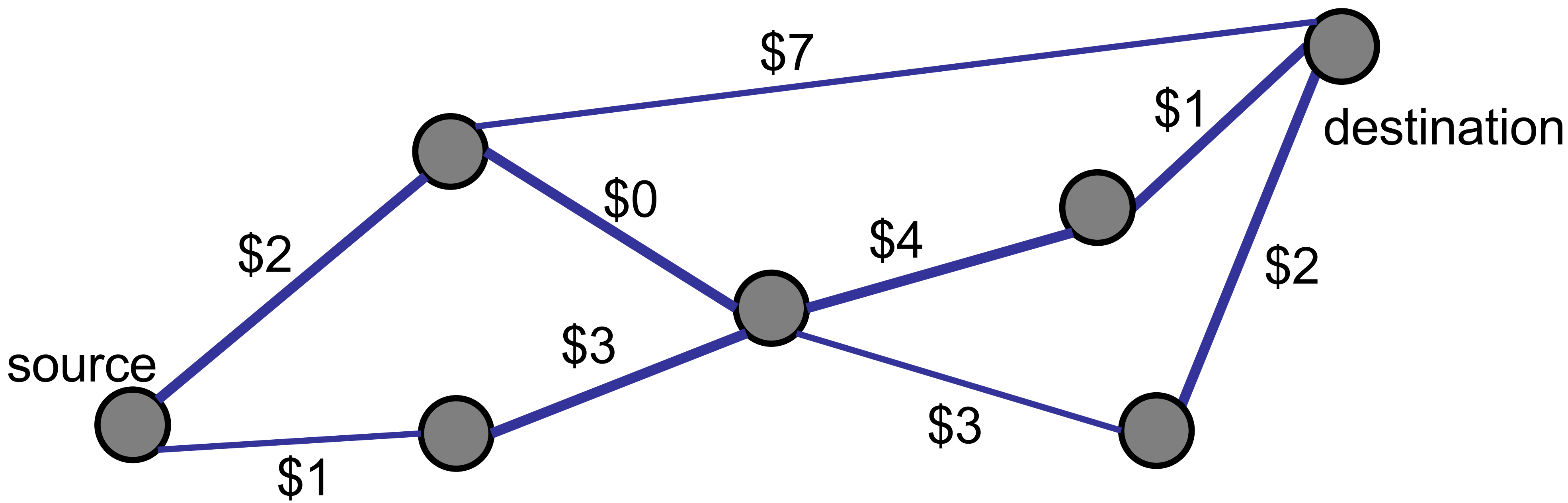
- Start at source s.
- Go to destination d.
- Each edge e earns money m(e).
- Find the path that makes the most money.



Example

NOT a solution:

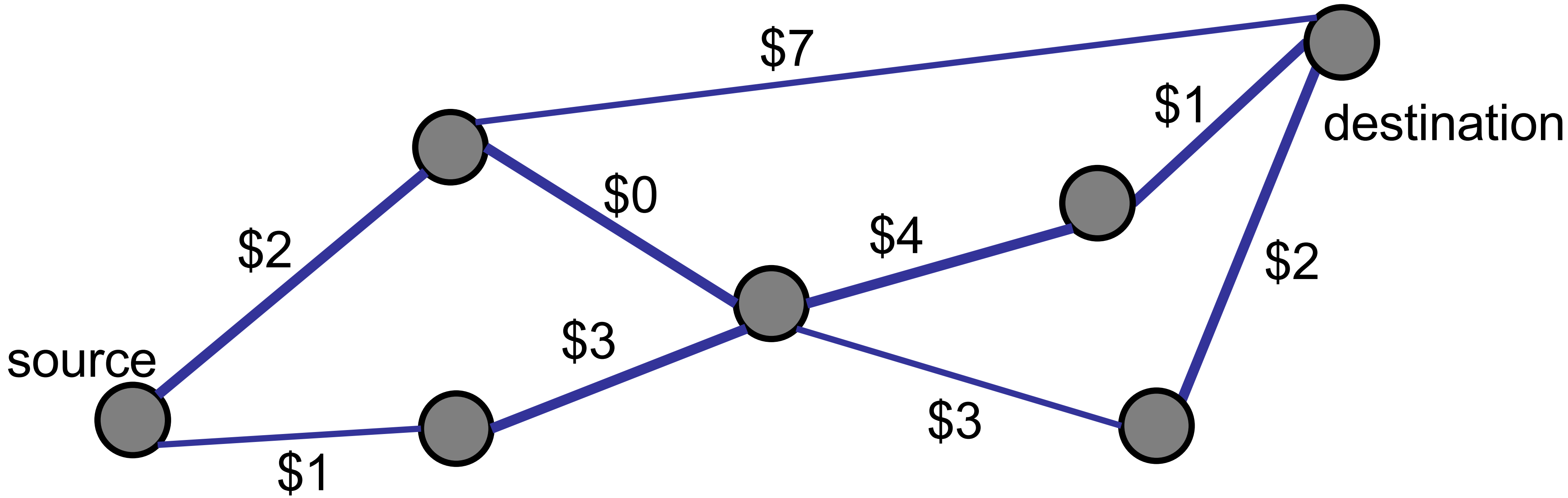
- Start at source s.
- Run BFS (or DFS) to explore every path.
- Keep track of the best path.



Example

Problem 1: Does not work.

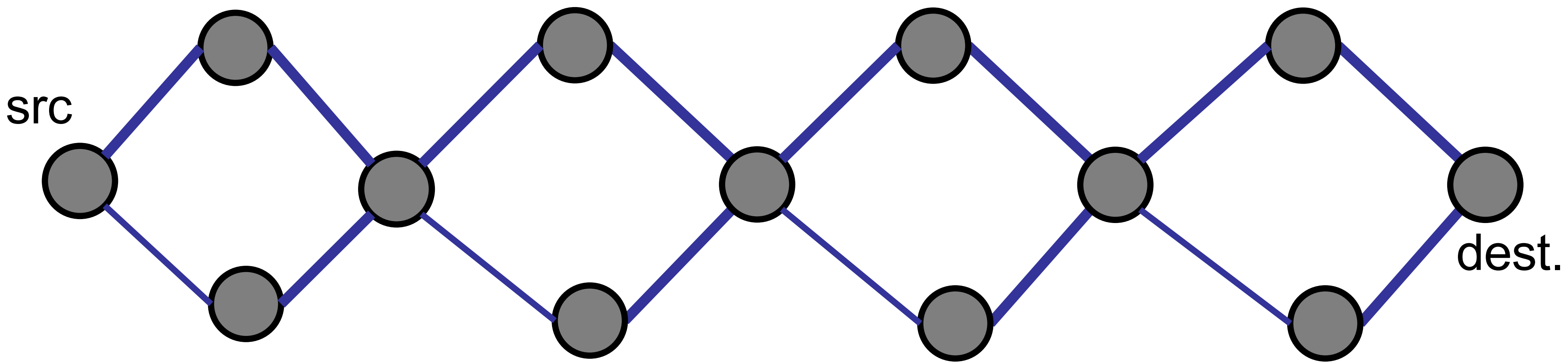
- DFS or BFS do NOT explore every path.
- Once a node is visited, it is never explored again.



Example

Problem 2: Too expensive.

- Some graphs have an exponential number of paths.
- It takes exponential time to explore all paths.



Example: $2^4 > 2^{n/4}$ different s->d paths.

Common Mistake

What do BFS and DFS solve?

- They visit every node in the graph? Yes.
- They visit every edge in the graph? Yes.
- ~~They visit every path in the graph?~~

** If you modify BFS/DFS to backtrack / re-visit existing nodes, you *probably* have an exponential time algorithm (or an algorithm that never terminates because of loops).

Roadmap

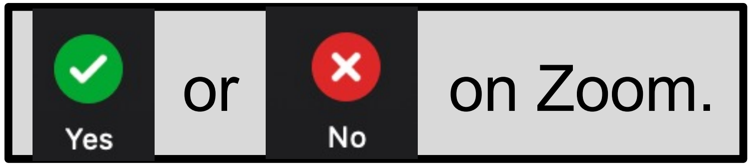
Directed Graphs

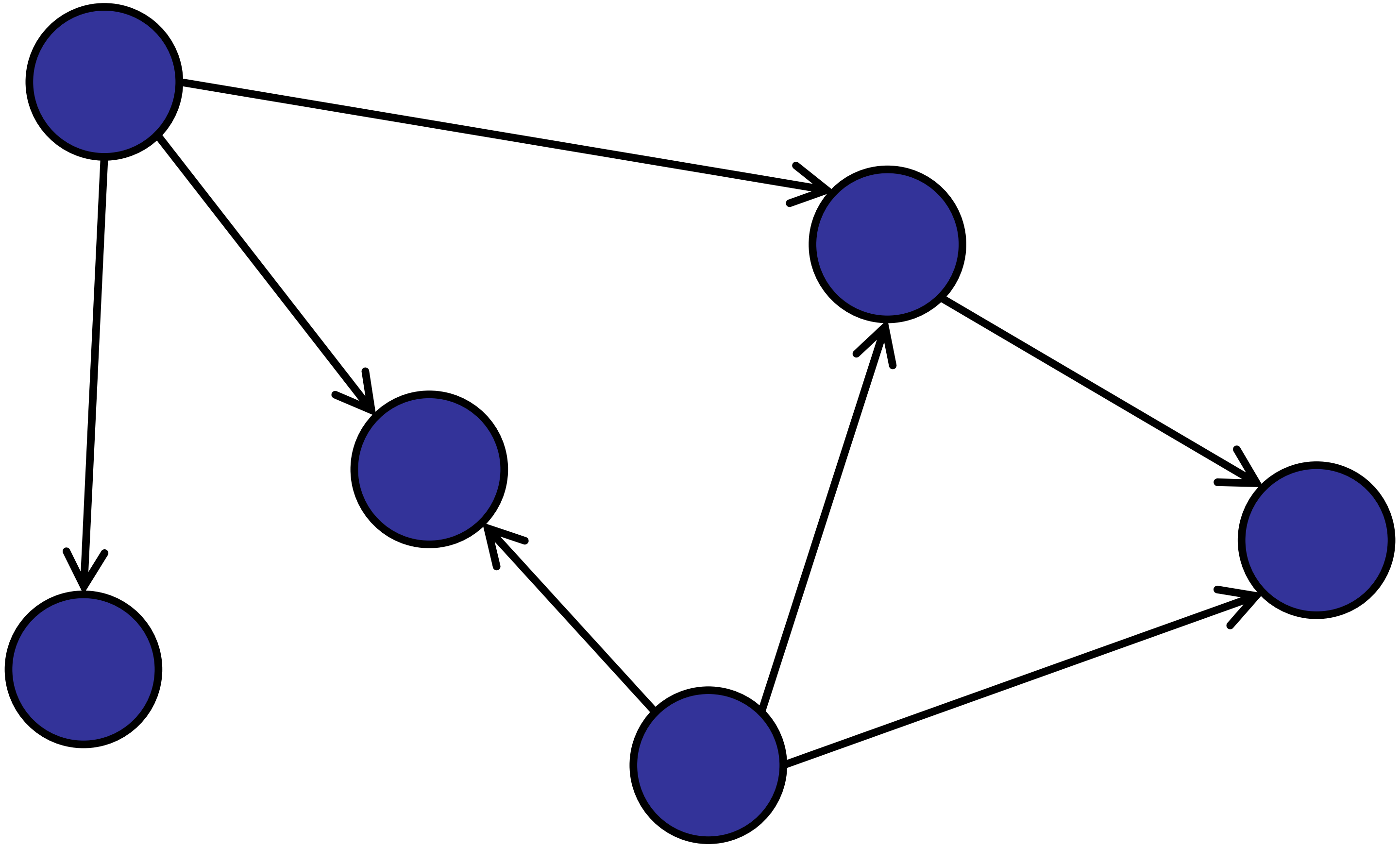
- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

What is a **directed** graph? (Digraph)

Is it a directed graph?

1. Yes
2. No.

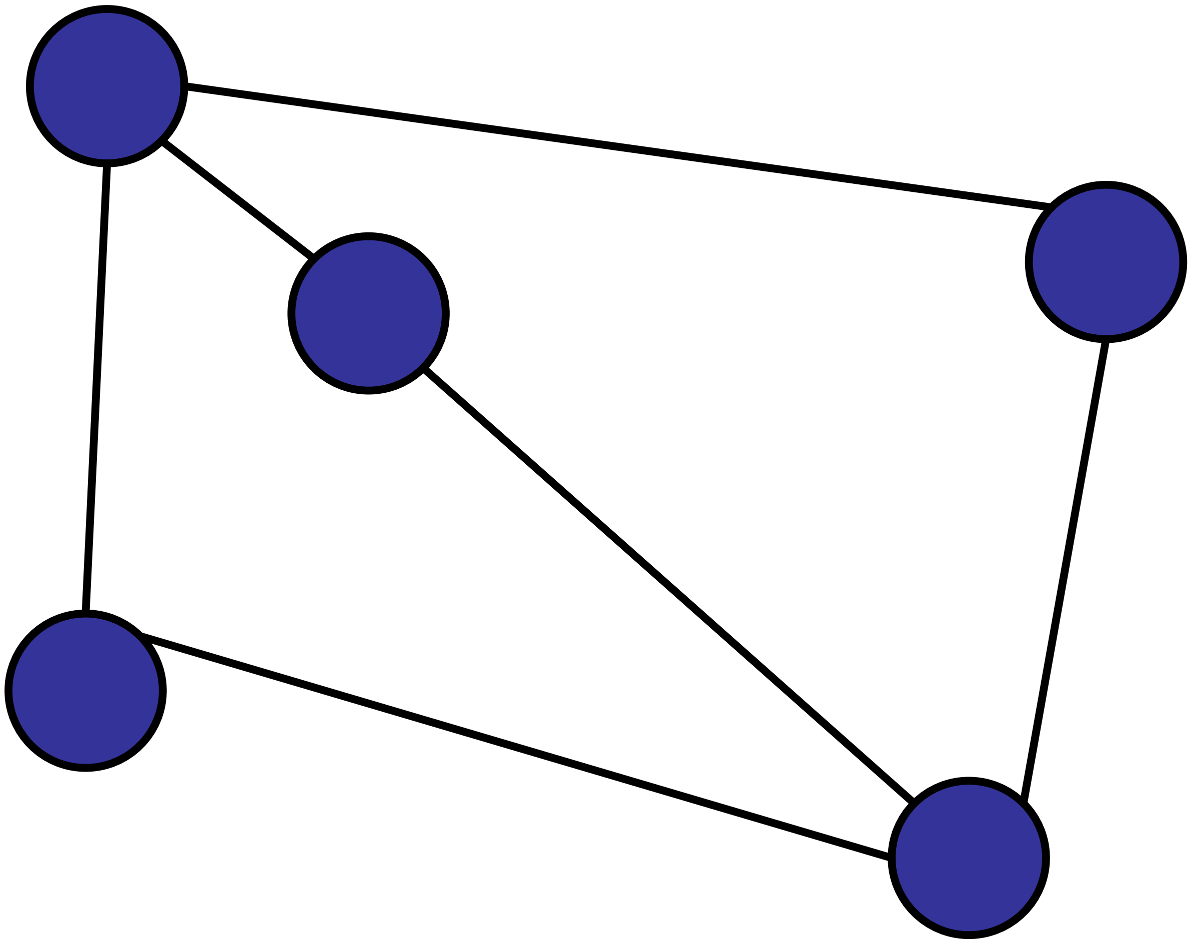




Is it a directed graph?

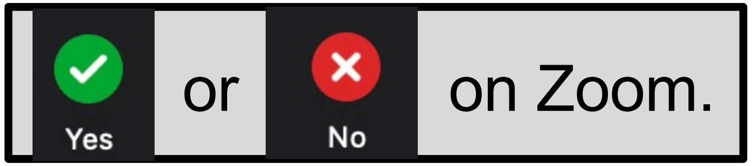
1. Yes
2. No.

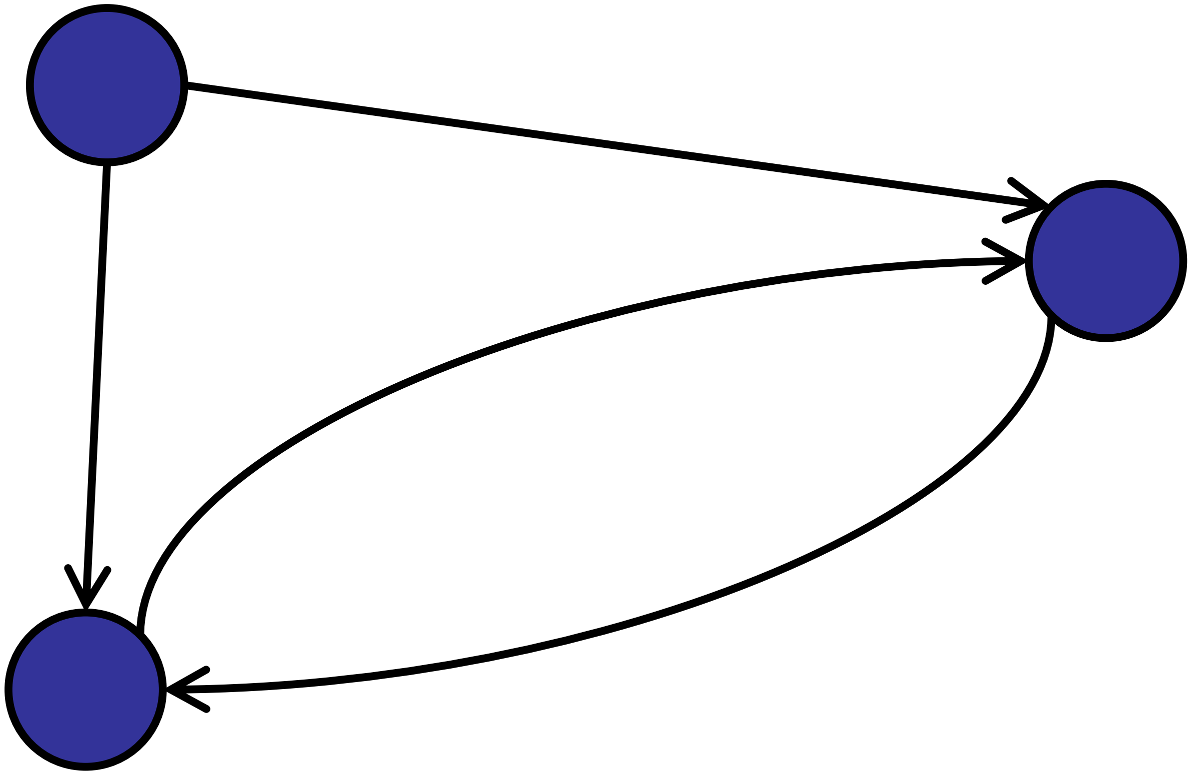
Yes or No on Zoom.



Is it a directed graph?

1. Yes
2. No.





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**.

What is a directed graph?

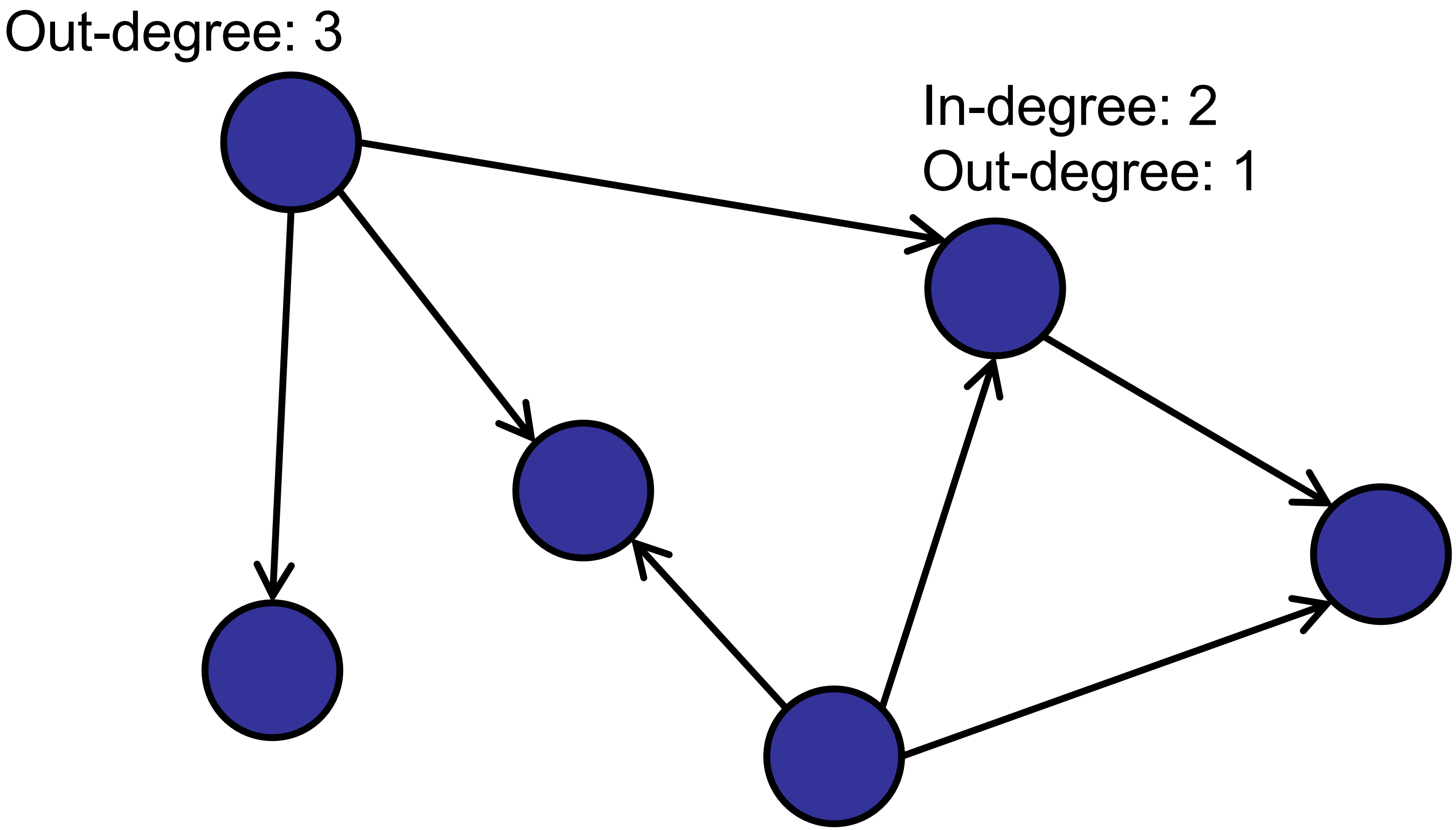
Graph G = <V, E>

- V is a set of nodes
  - At least one: $|V| > 0$.
- E is a set of edges:
  - $E \subseteq \{ (v,w) : (v \in V), (w \in V) \}$
  - $e = (v,w)$ Order matters!
  - For all $e_1, e_2 \in E : e_1 \neq e_2$

What is a directed graph?

In-degree: number of incoming edges

Out-degree: number of outgoing edges



Representing a (Directed) Graph

Adjacency List:

- Array of nodes
- Each node maintains a list of neighbors
- Space: $O(V + E)$

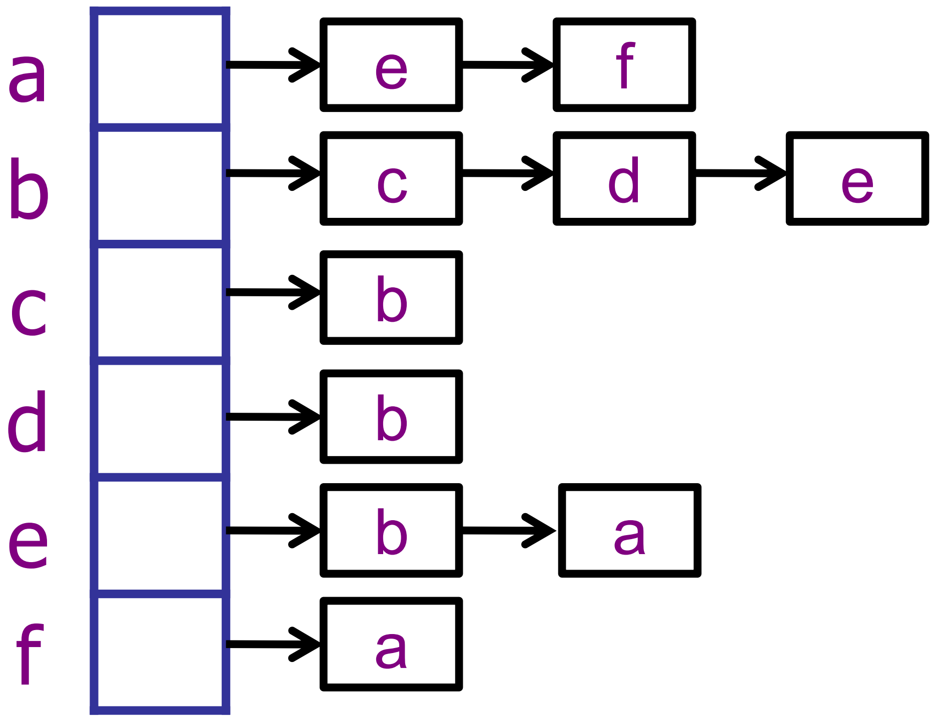
Adjacency Matrix:

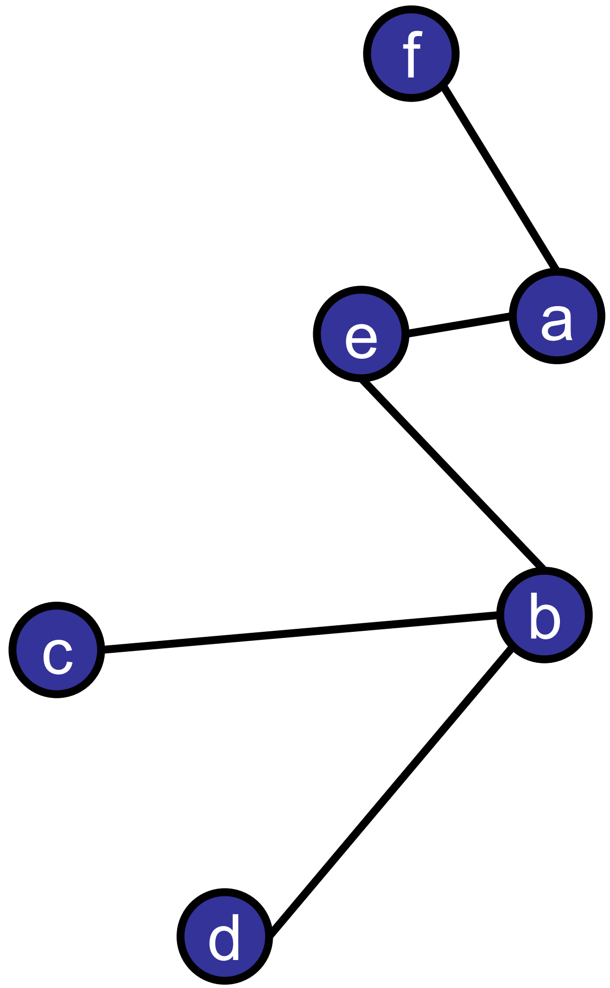
- Matrix $A[v,w]$ represents edge (v,w)
- Space: $O(V^2)$

Adjacency List

Graph consists of:

- Nodes: stored in an array
- Edges: linked list per node

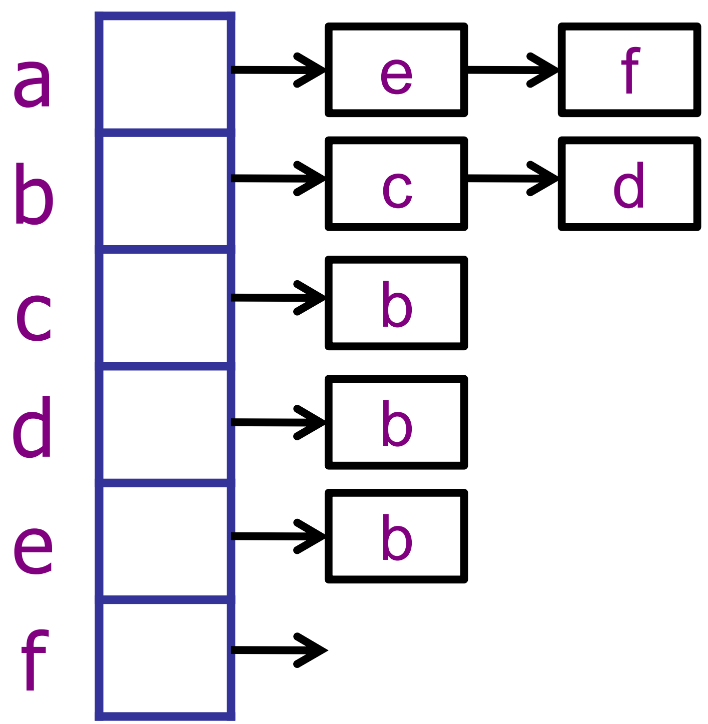


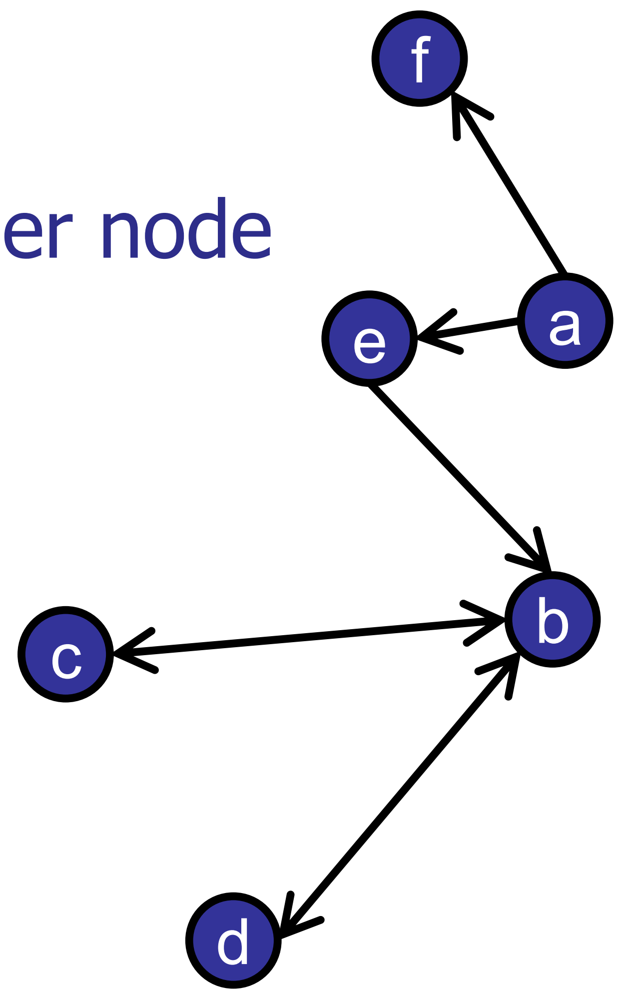


Adjacency List

Directed Graph consists of:

- Nodes: stored in an array
- **Outgoing** Edges: linked list per node





Representing a (Directed) Graph

Adjacency List:

- Array of nodes
- Each node maintains a list of neighbors
- Space: $O(V + E)$

Adjacency Matrix:

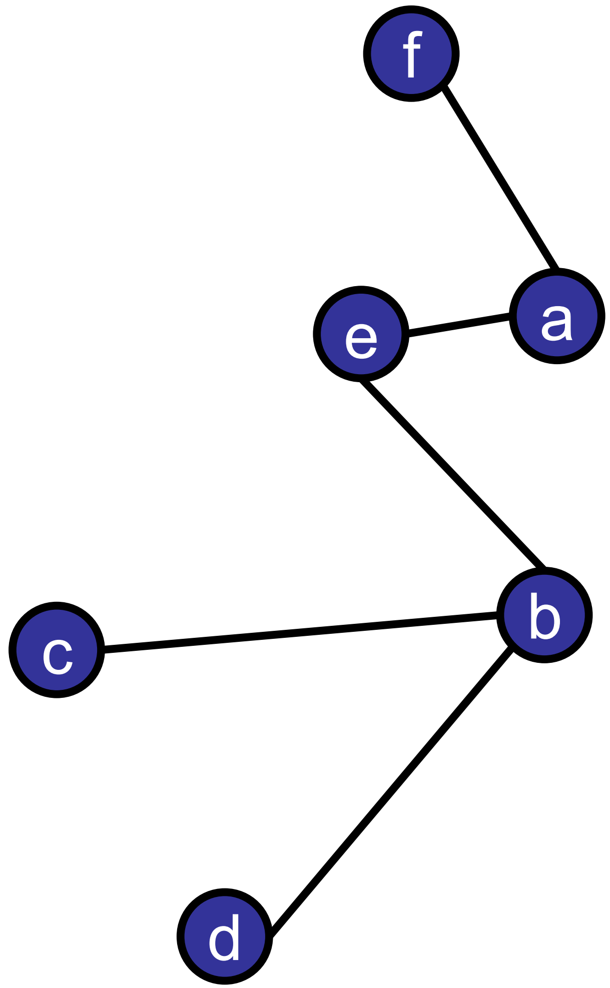
- Matrix $A[v,w]$ represents edge (v,w)
- Space: $O(V^2)$

Adjacency Matrix

Graph consists of:

- Nodes
- Edges = pairs of nodes

	a	b	c	d	e	f
a	0	0	0	0	**1**	**1**
b	0	0	**1**	**1**	**1**	0
c	0	**1**	0	0	0	0
d	0	**1**	0	0	0	0
e	**1**	**1**	0	0	0	0
f	**1**	0	0	0	0	0

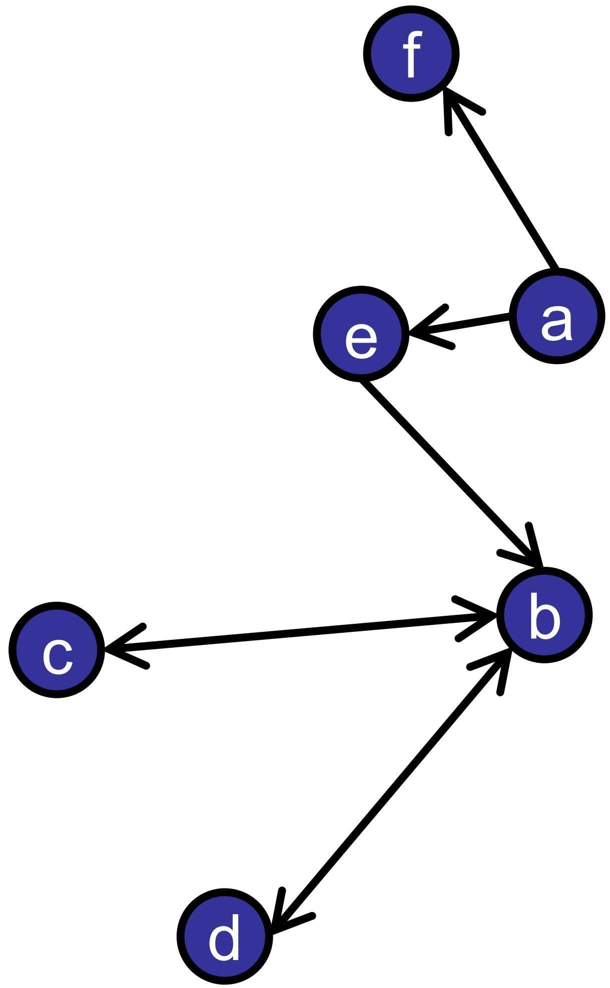


Adjacency Matrix

Directed Graph consists of:

- Nodes
- Edges = pairs of nodes

	a	b	c	d	e	f
a	0	0	0	0	**1**	**1**
b	0	0	**1**	**1**	**0**	0
c	0	**1**	0	0	0	0
d	0	**1**	0	0	0	0
e	**0**	**1**	0	0	0	0
f	**0**	0	0	0	0	0



Adjacency Matrix

Graph represented as:

$A[v][w] = 1$ iff $(v,w) \in E$

	a	b	c	d	e	f
a	0	0	0	0	**1**	**1**
b	0	0	**1**	**1**	**0**	0
c	0	**1**	0	0	0	0
d	0	**1**	0	0	0	0
e	**0**	**1**	0	0	0	0
f	**0**	0	0	0	0	0

Searching a (Directed) Graph

Breadth-First Search:

- Search level-by-level
- Follow outgoing edges
- Ignore incoming edges

Depth-First Search:

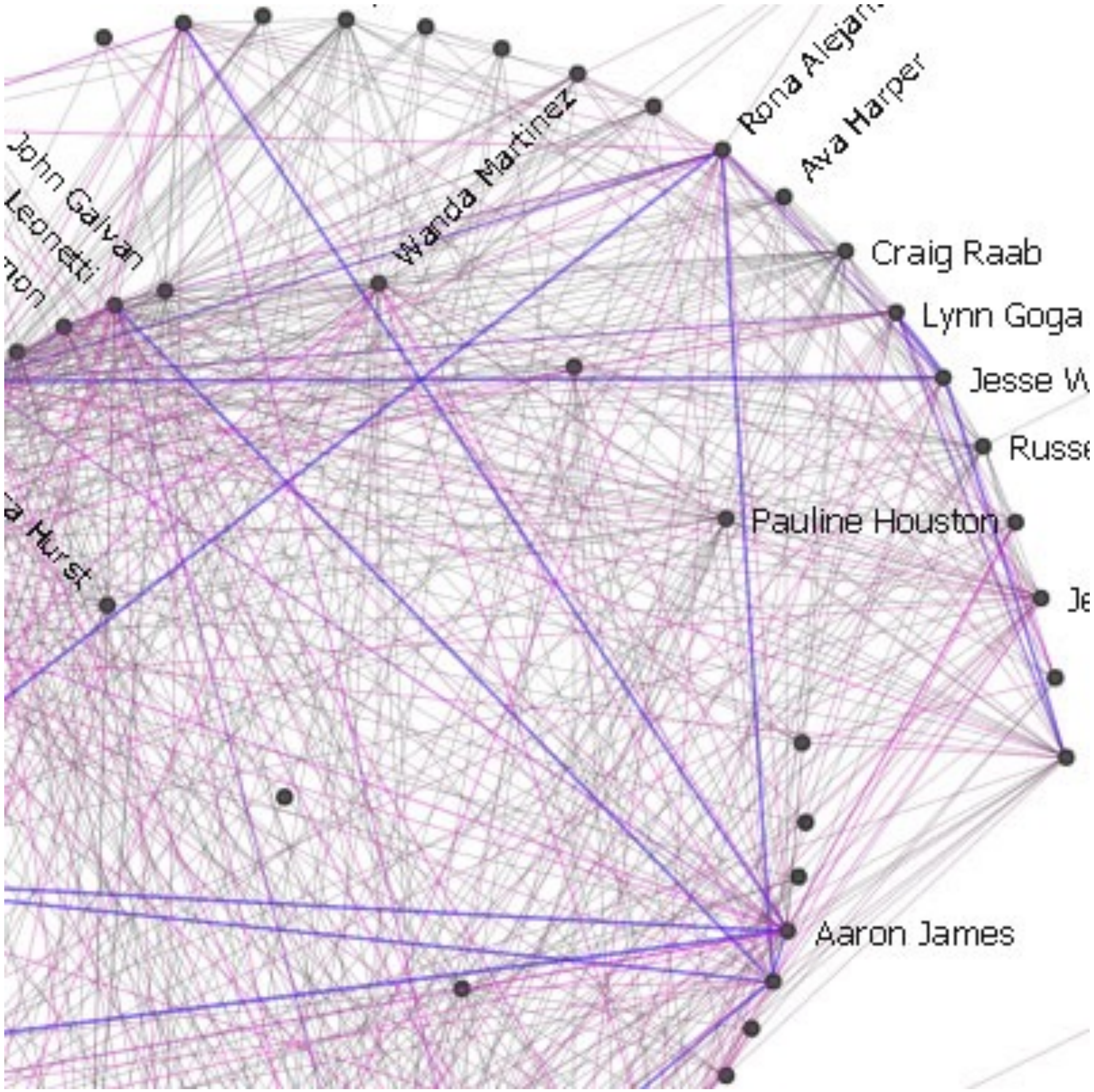
- Search recursively
- Follow outgoing edges
- Backtrack (through incoming edges)

Example of directed graphs

Directed Graphs

Is friendship always bidirectional?:

- Nodes are people
- Edge = friendship



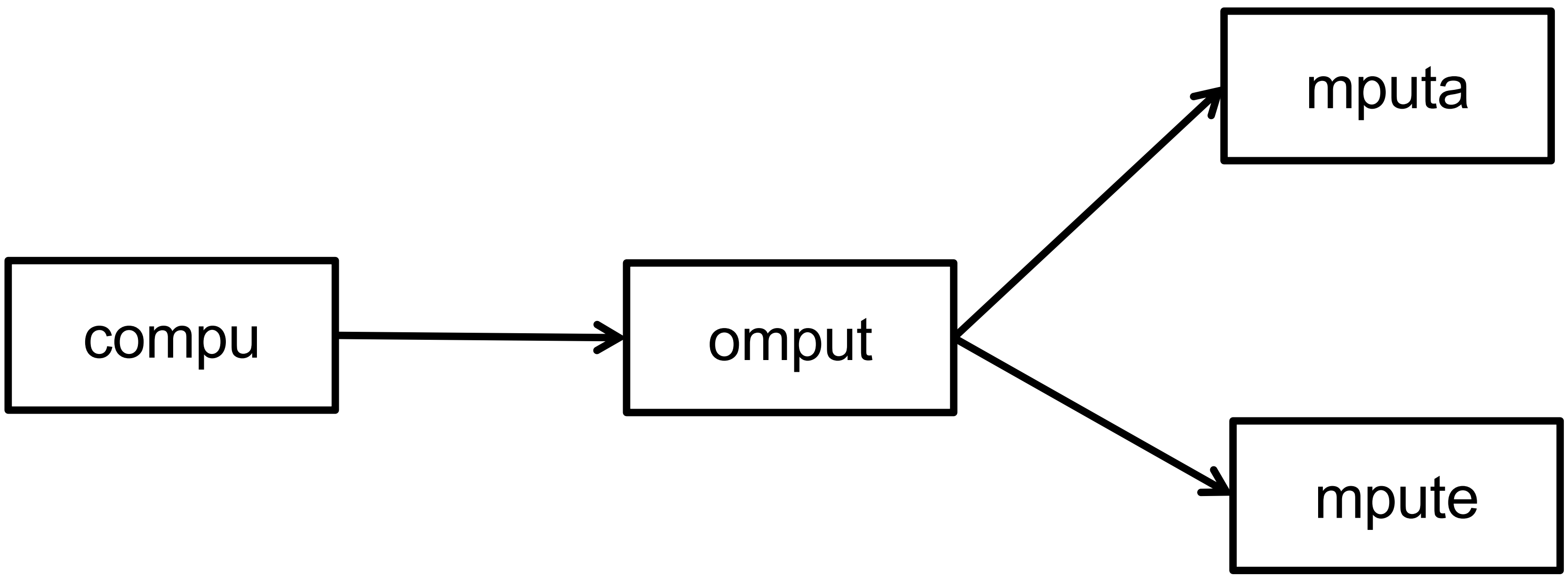
Facebook: yes

Twitter: no

Directed Graphs

Markov text generation:

- Nodes are kgrams
- Edge = one kgram follows another



Scheduling

Set of tasks for baking cookies:

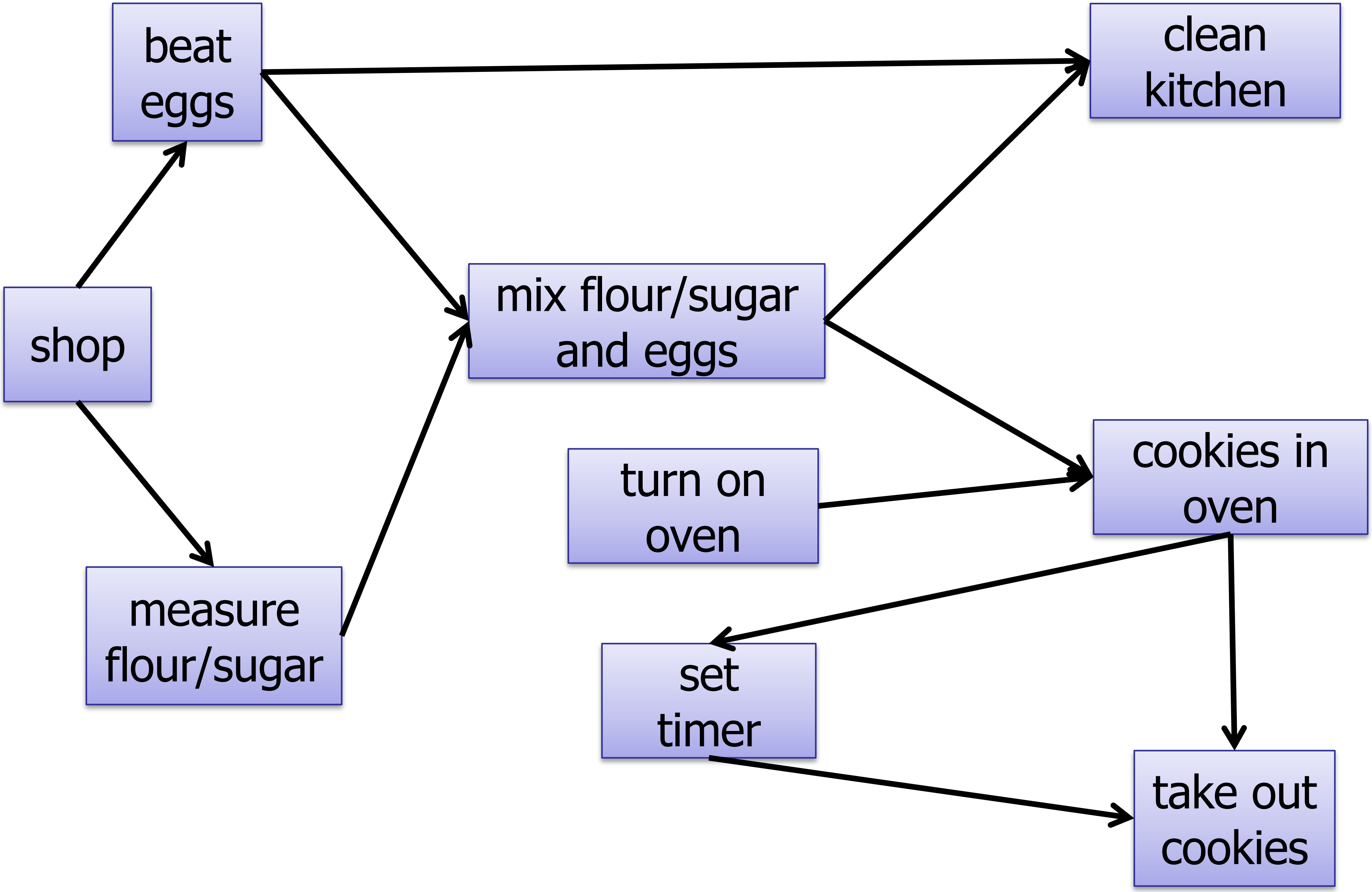
- Shop for groceries
- Put the cookies in the oven
- Clean the kitchen
- Beat the eggs in a bowl
- Measure the flour and sugar in a bowl
- Mix the eggs with the flour and sugar
- Turn on the oven
- Set the timer
- Take out the cookies

Scheduling

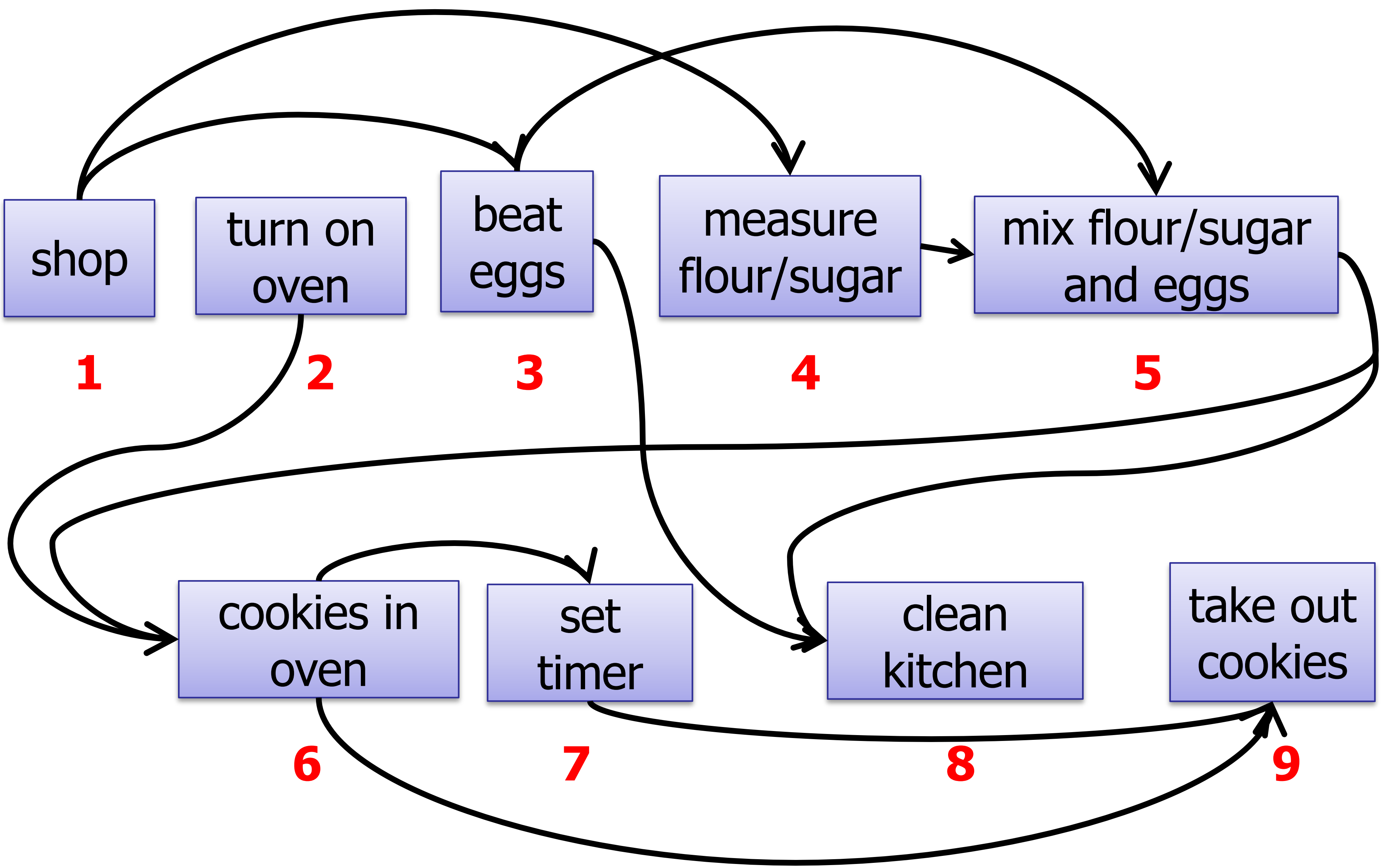
Ordering:

- 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 Order



Topological Order

Properties:

1. Sequential total ordering of all nodes

1. shop

2. turn on oven

3. measure flour/sugar

4. eggs

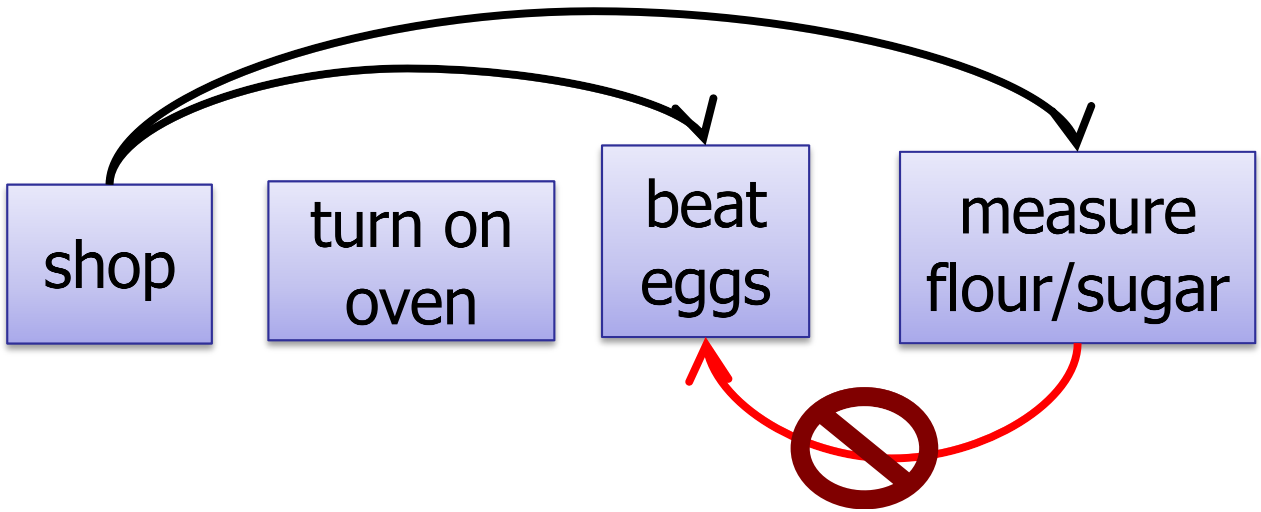
Topological Order

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

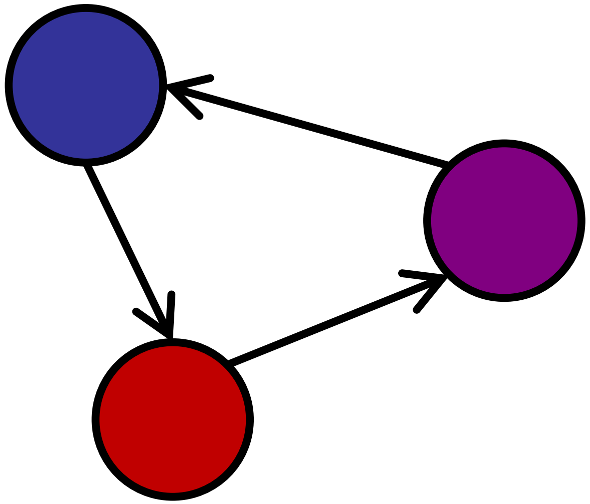


Does every directed graph have a topological ordering?

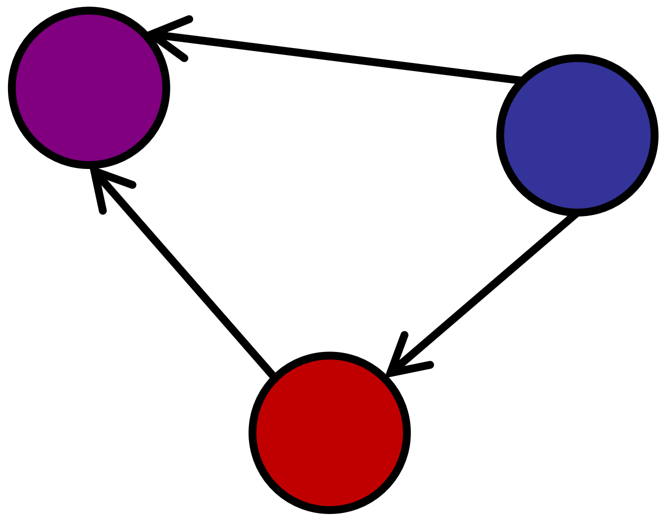
1. Yes
2. No
3. Only if the adjacency matrix has small second eigenvalue.

Directed Acyclic Graphs

Cyclic

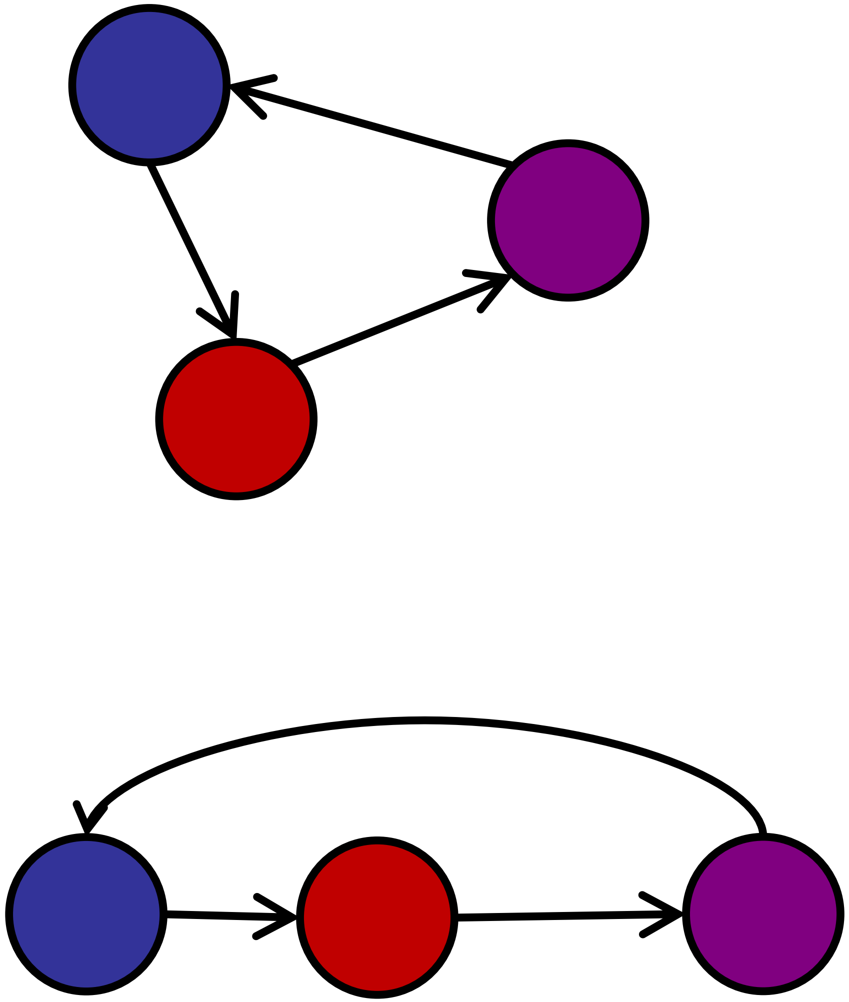


Acyclic

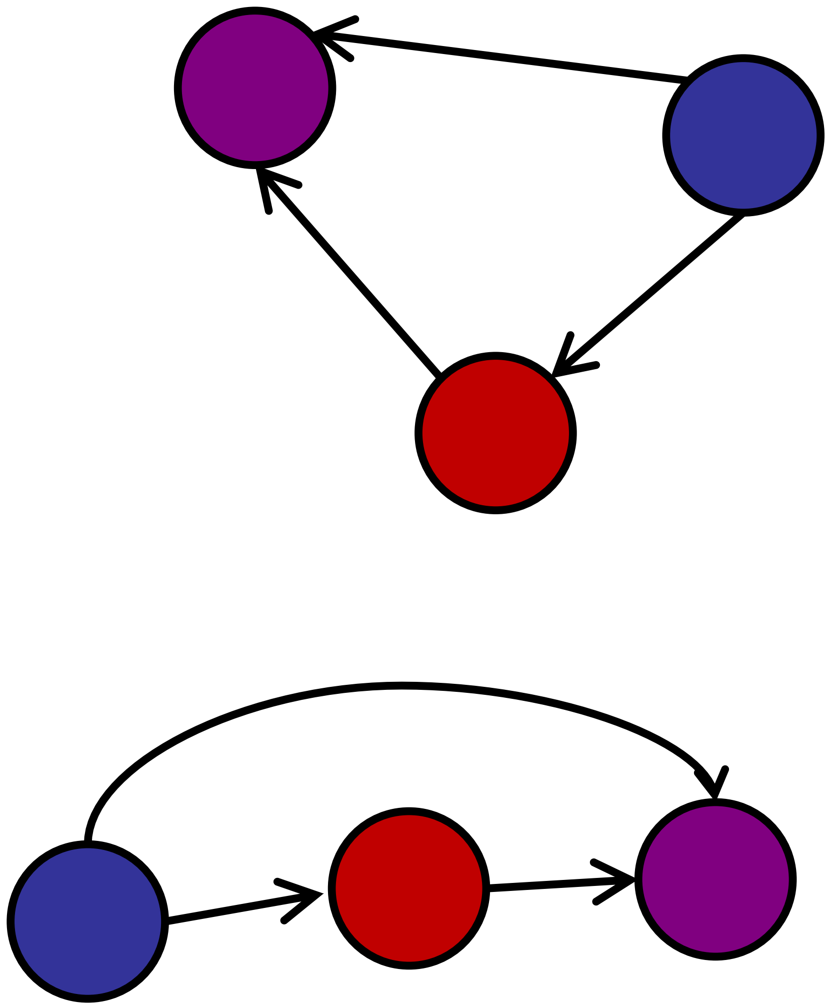


Directed Acyclic Graphs

Cyclic

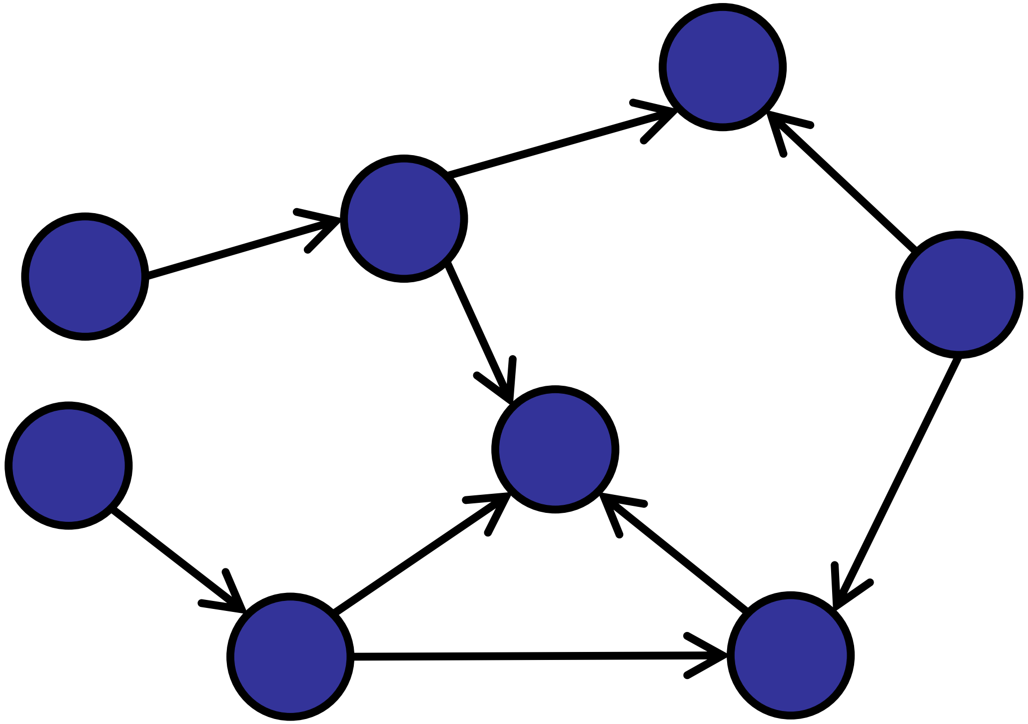


Acyclic



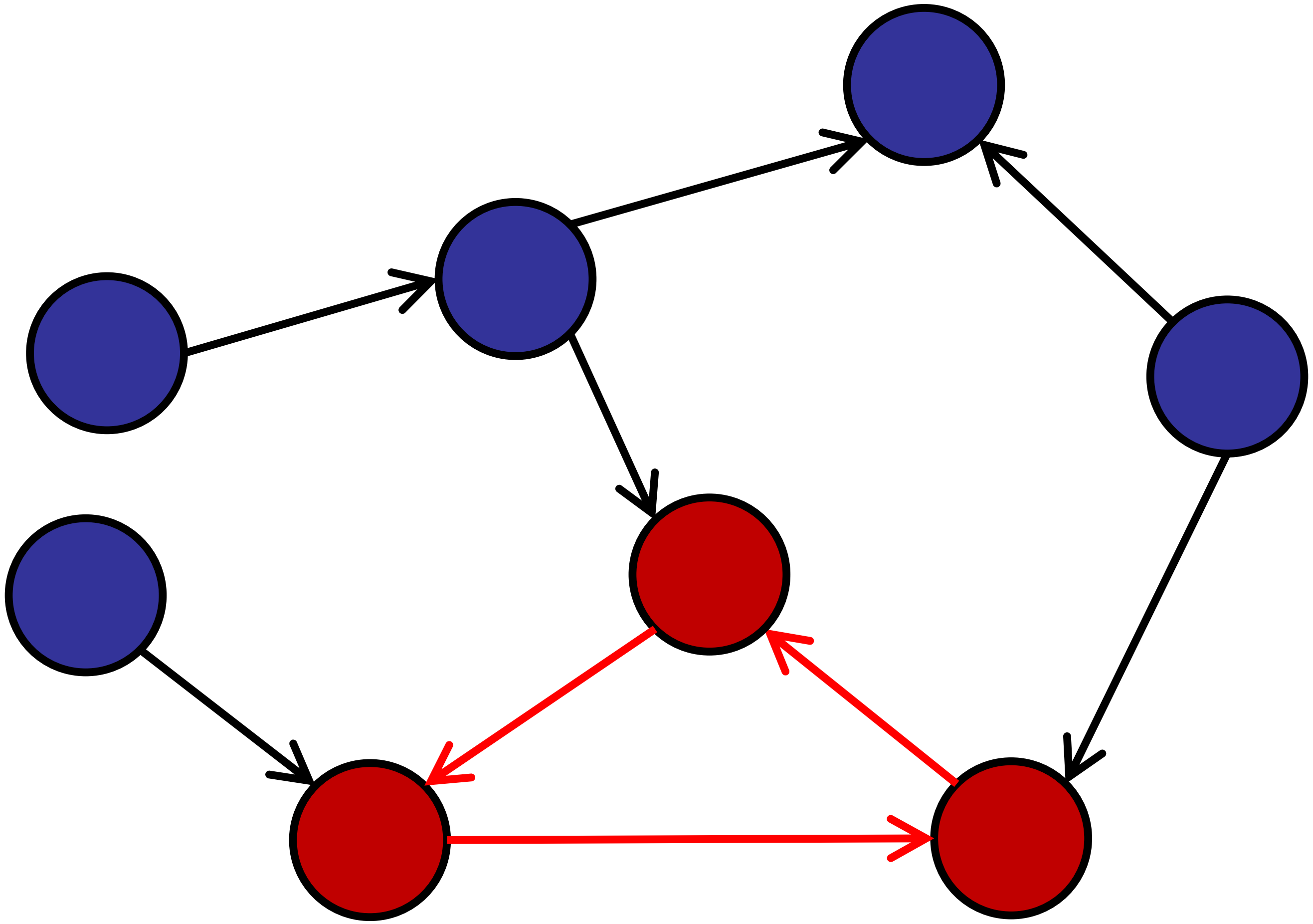
Is this graph:

1. Cyclic
2. Acyclic
3. Transcendental

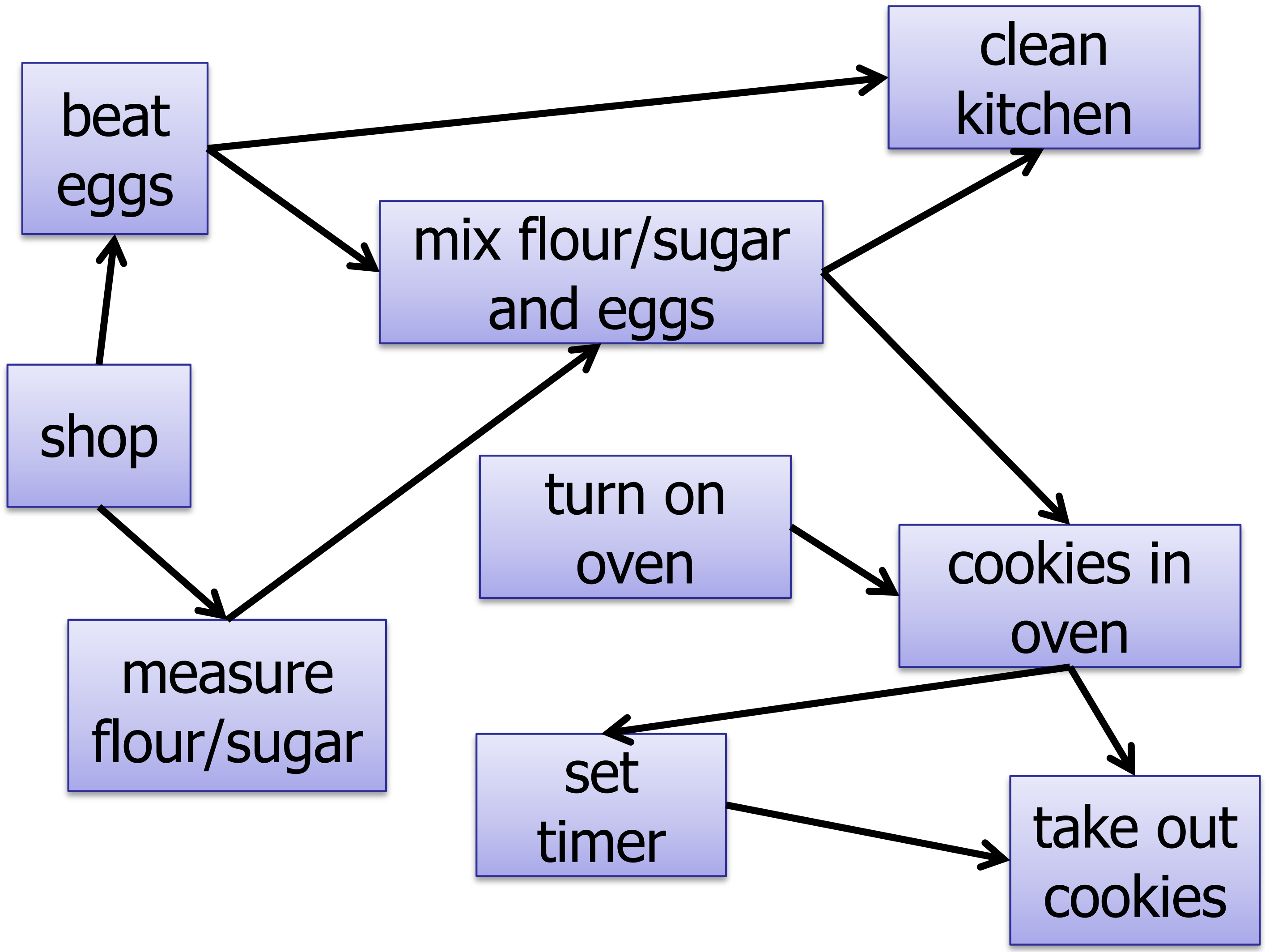


Directed Acyclic Graphs

Does it have a topological ordering?



Directed Acyclic Graph



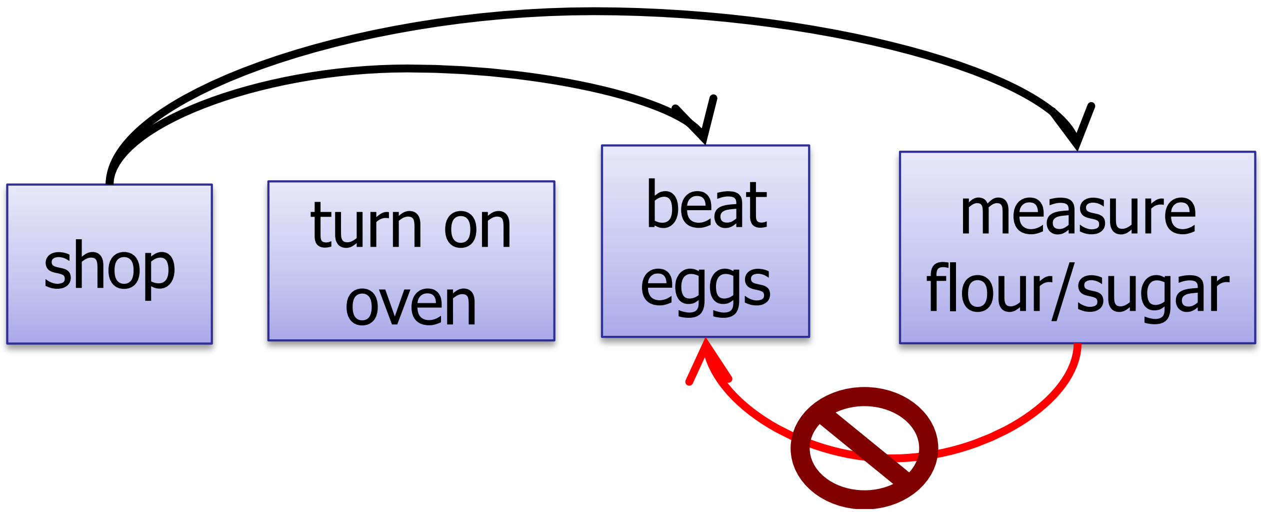
Topological Order

Properties:

1. Sequential total ordering of all nodes

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

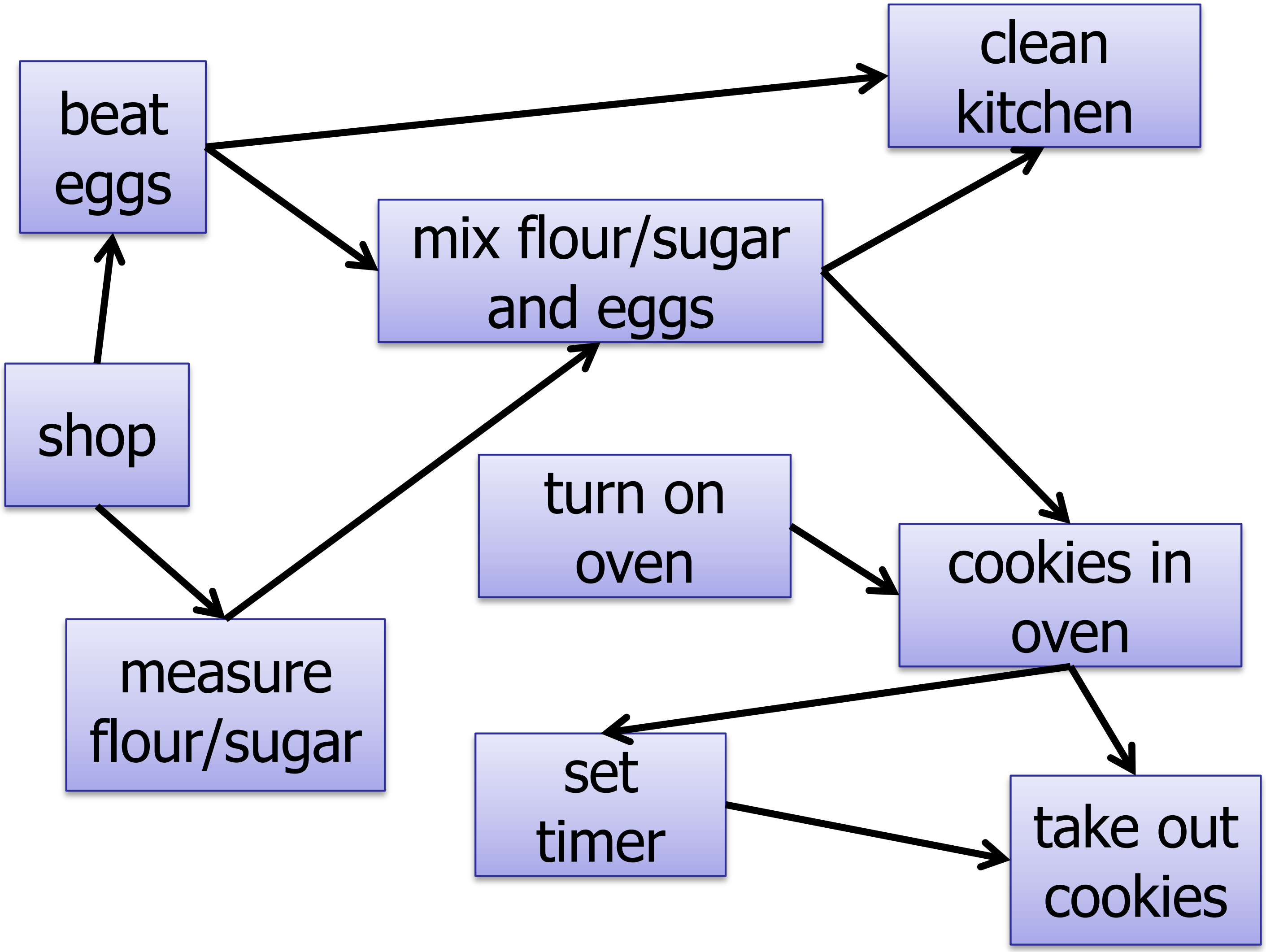
2. Edges only point forward



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

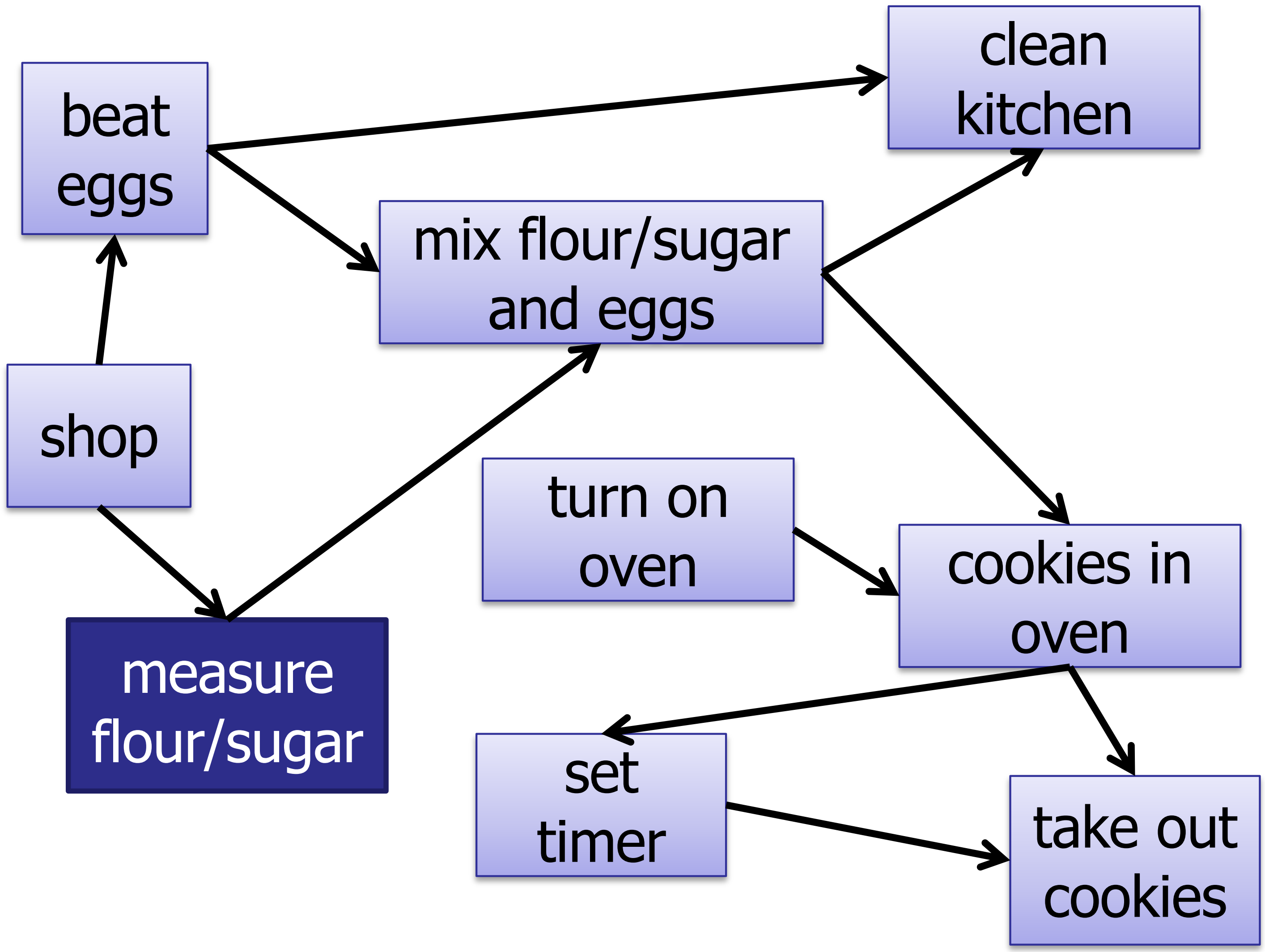
1. Breadth-first search
2. ✓ Depth-first search
3. Either BFS or DFS.
4. Karatsuba algorithm
5. Something else

Depth-First Search



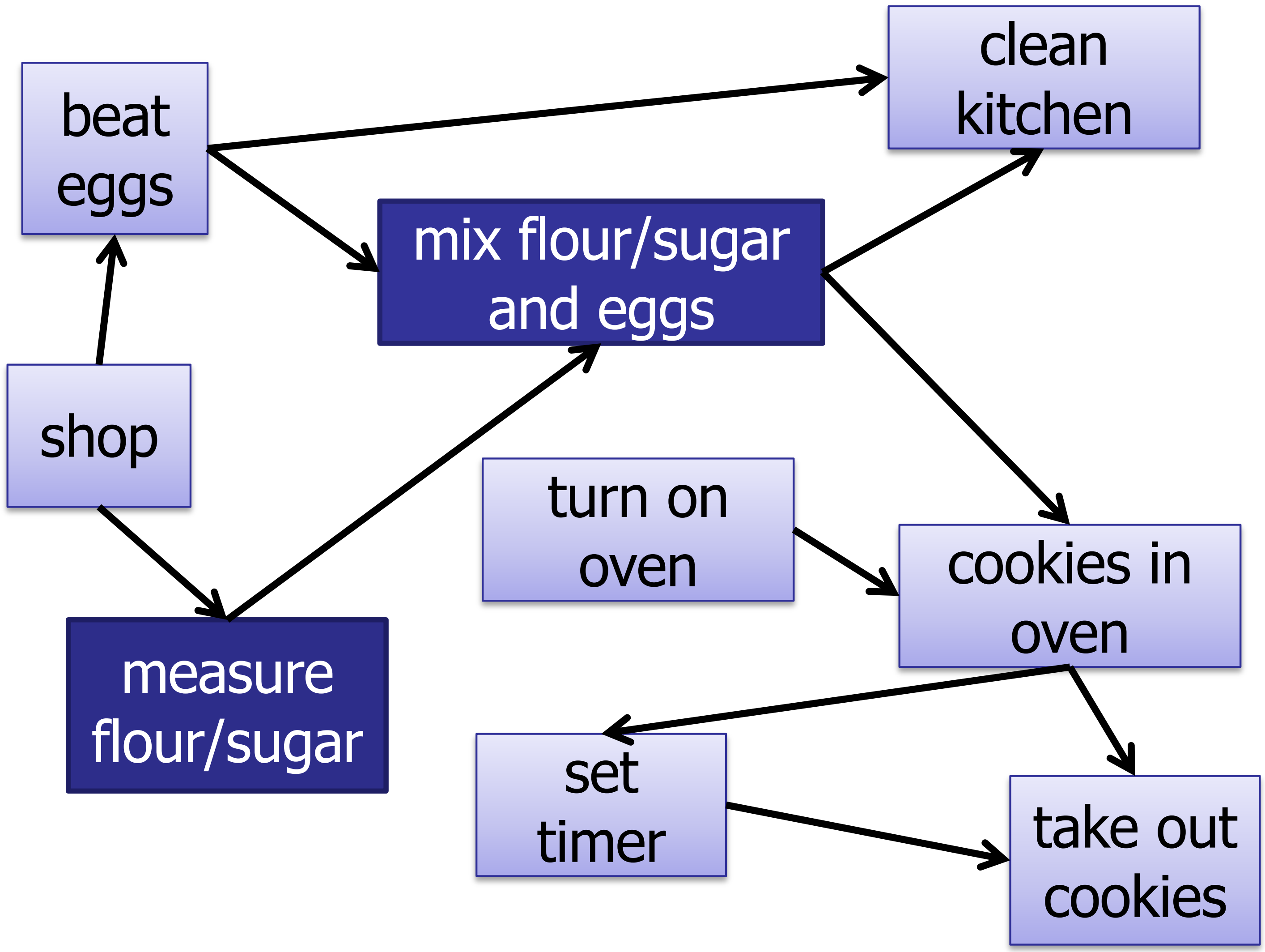
Depth-First Search

1. measure



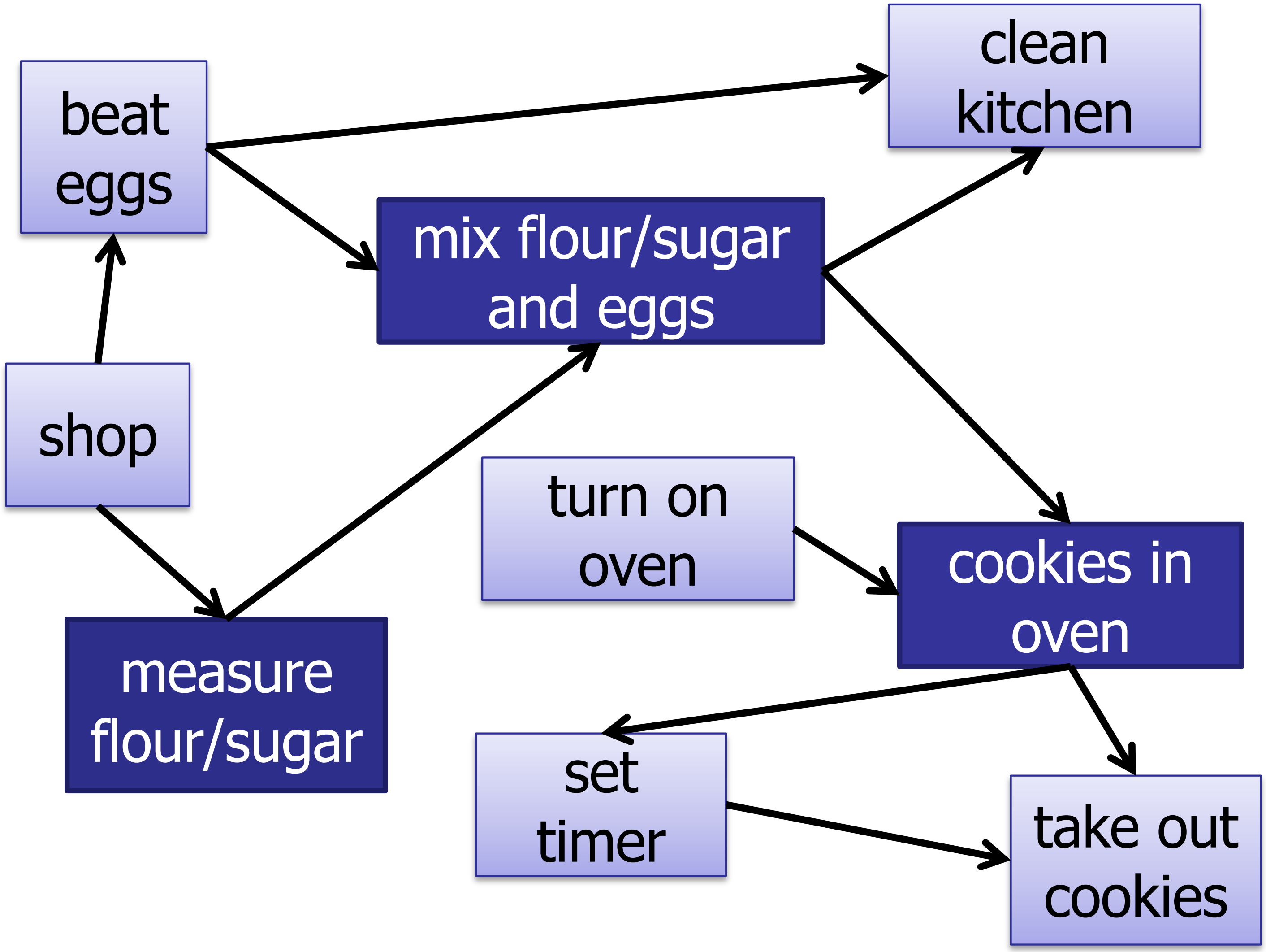
Depth-First Search

1. measure
2. mix



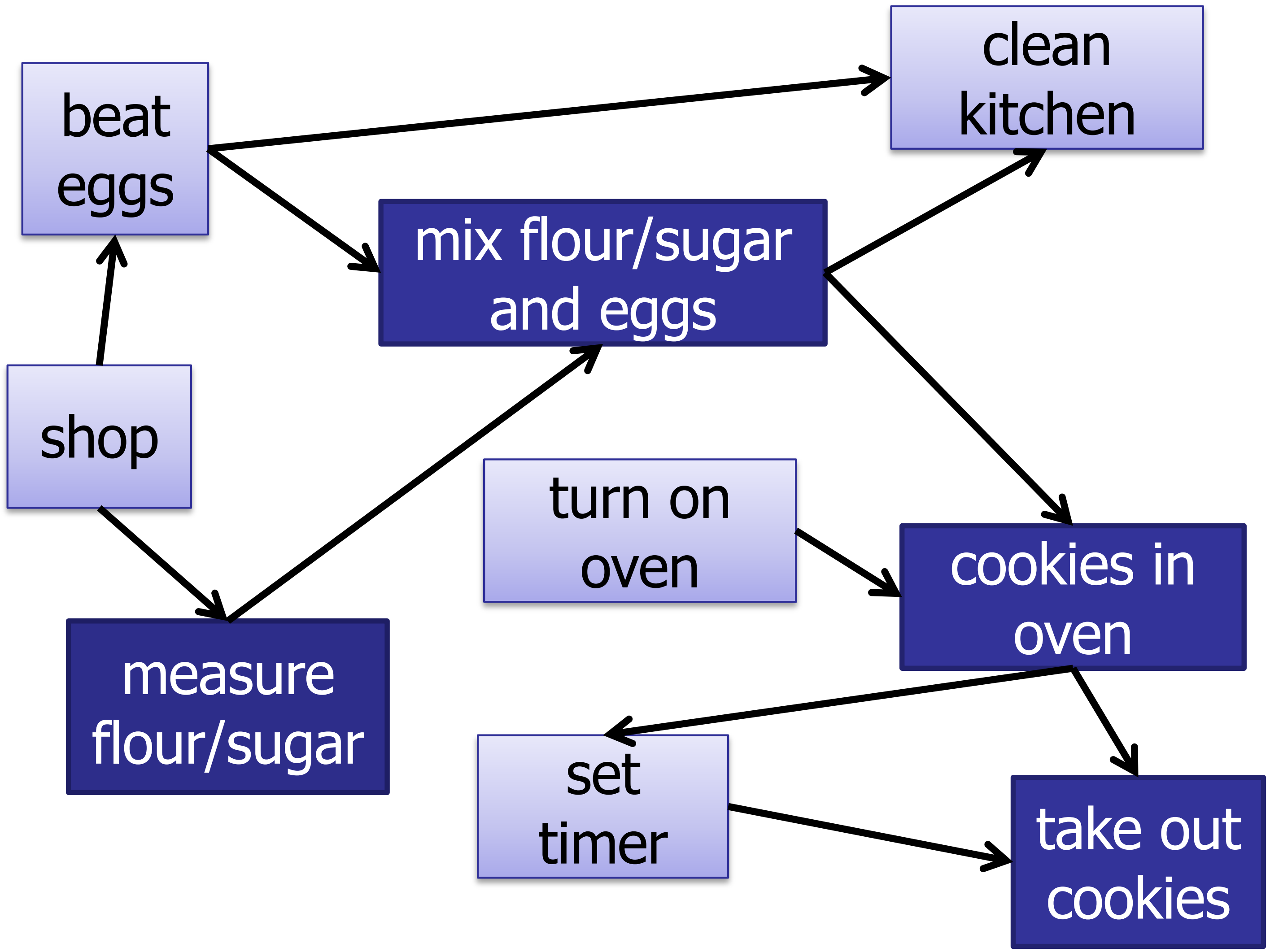
Depth-First Search

1. measure
2. mix
3. in oven



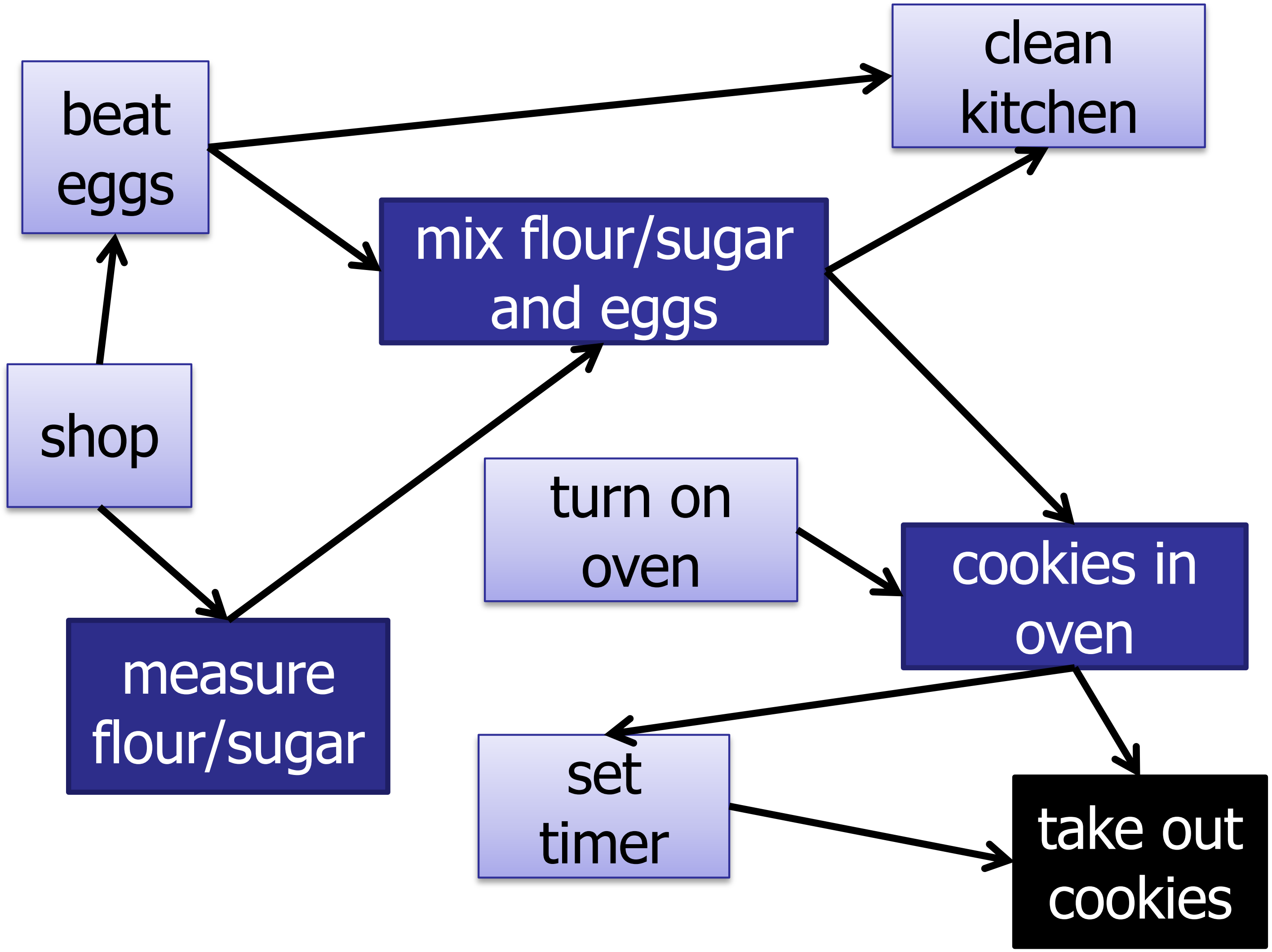
Depth-First Search

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



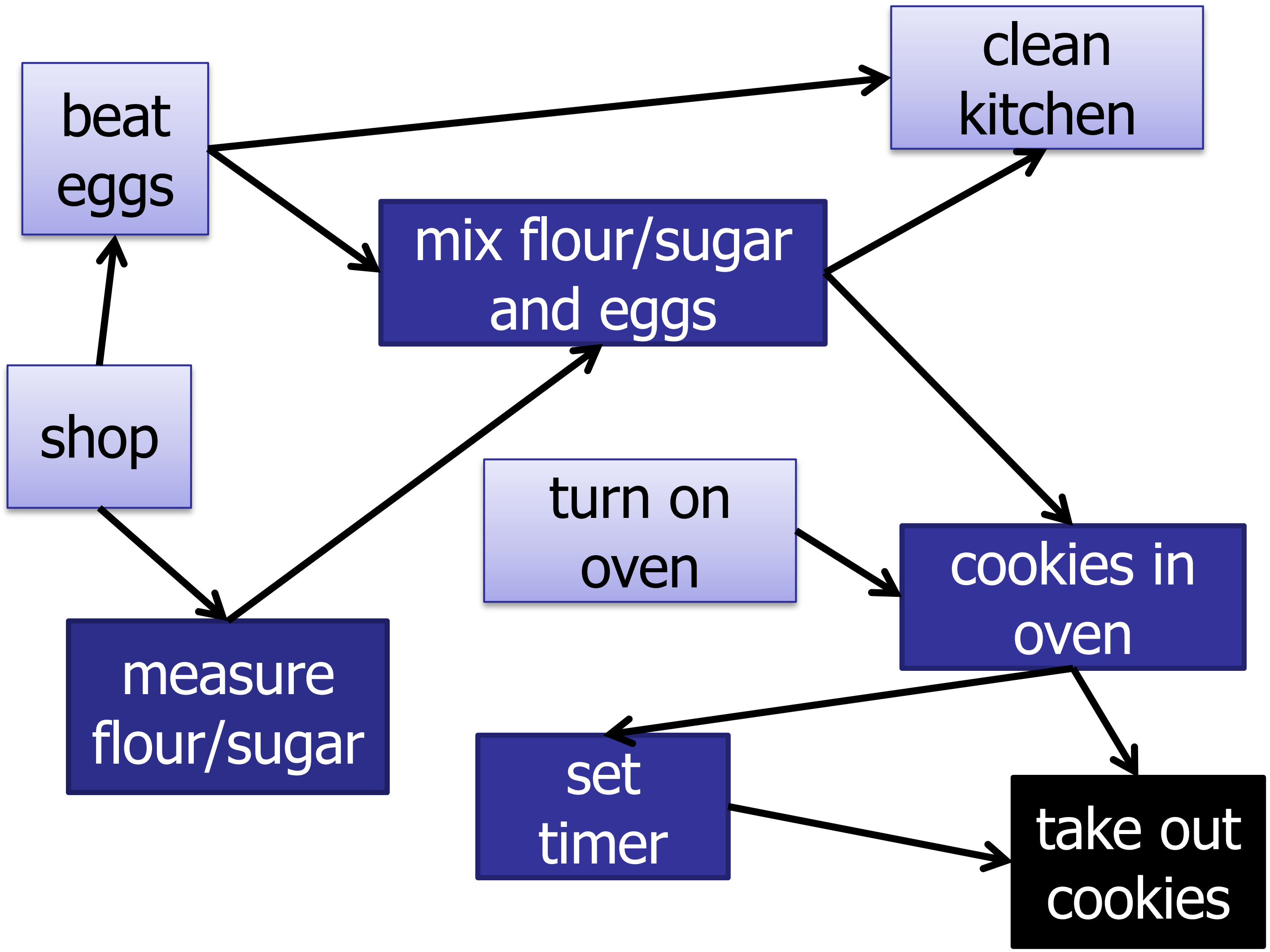
Depth-First Search

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



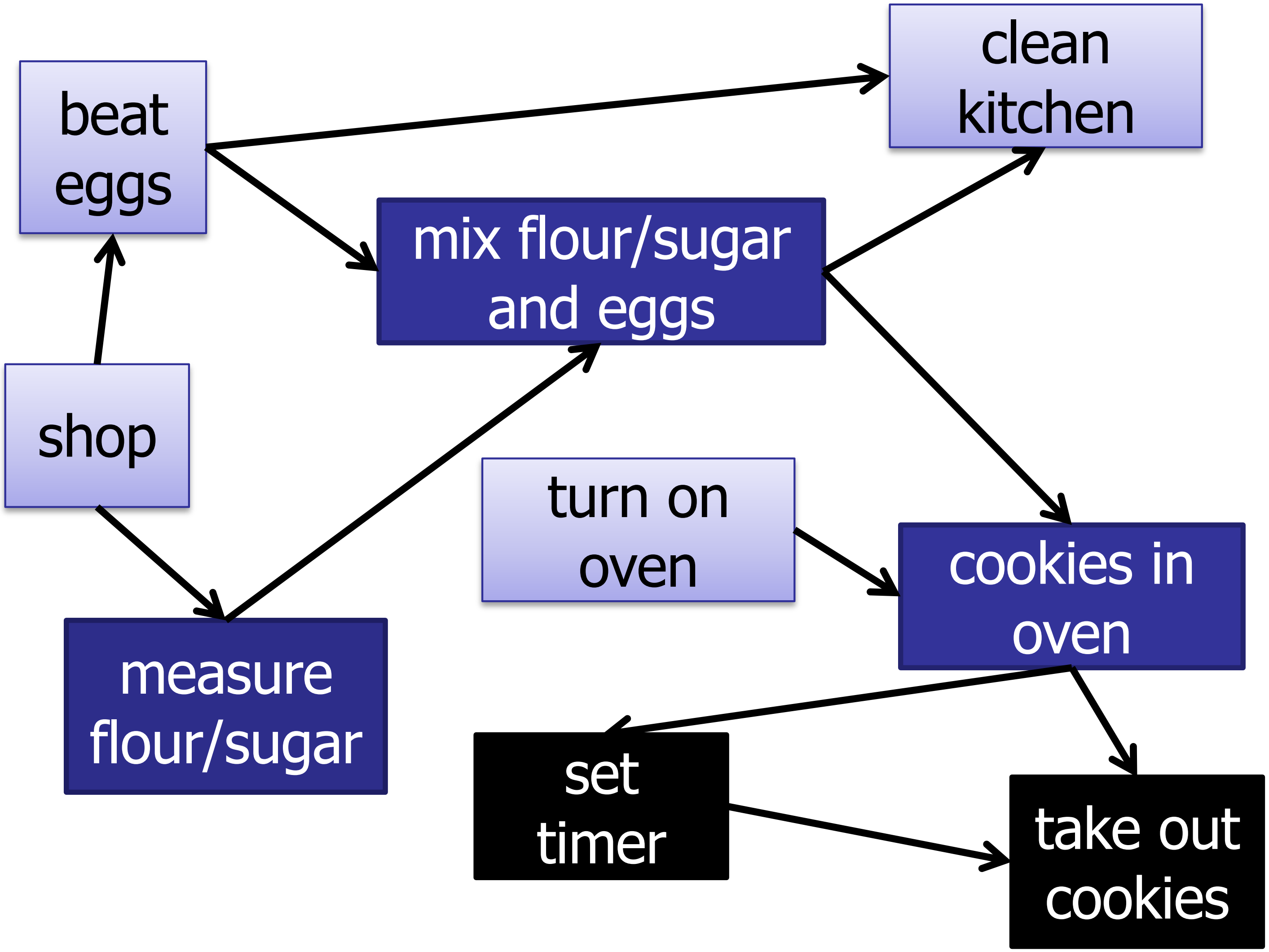
Depth-First Search

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



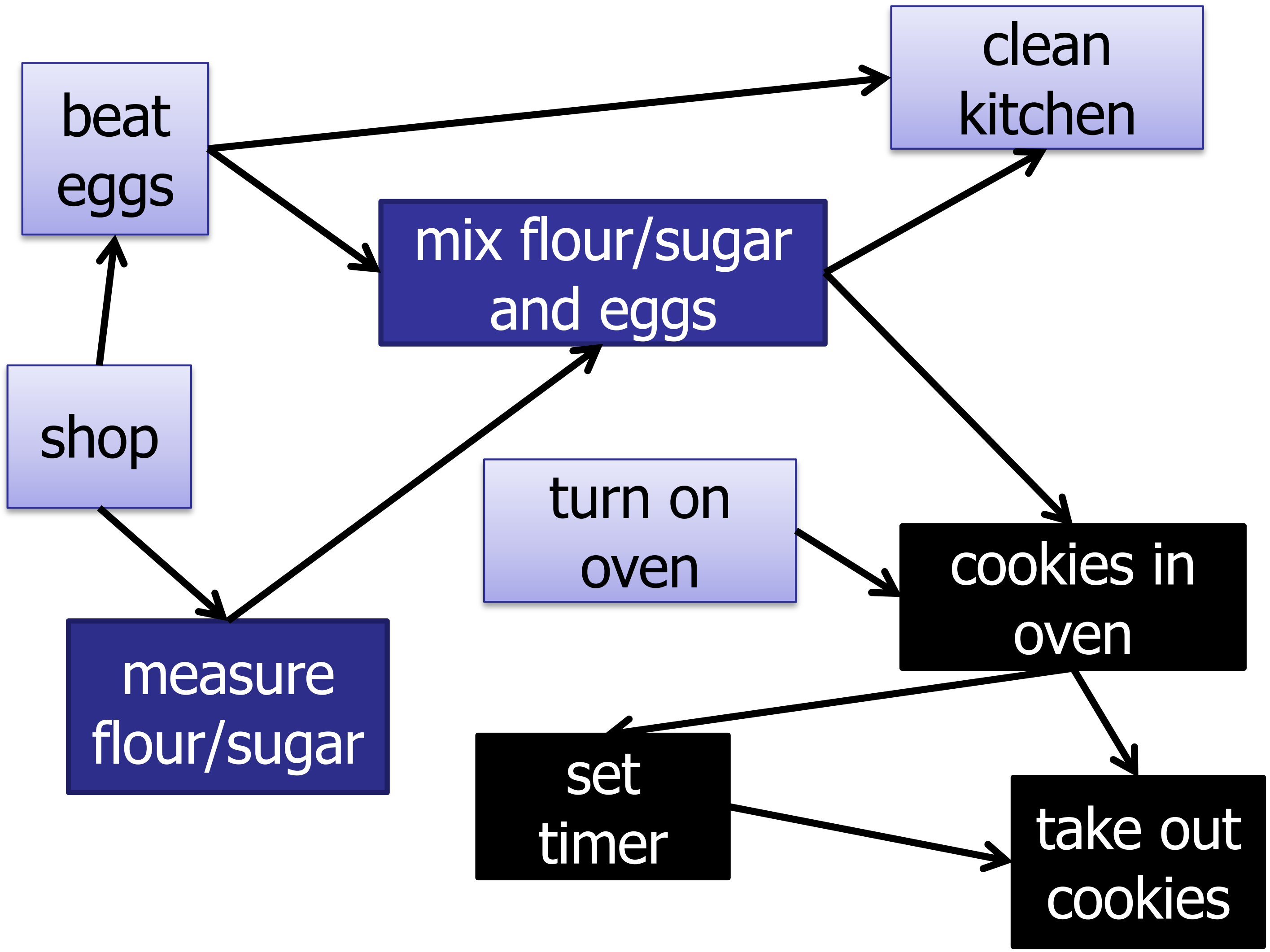
Depth-First Search

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



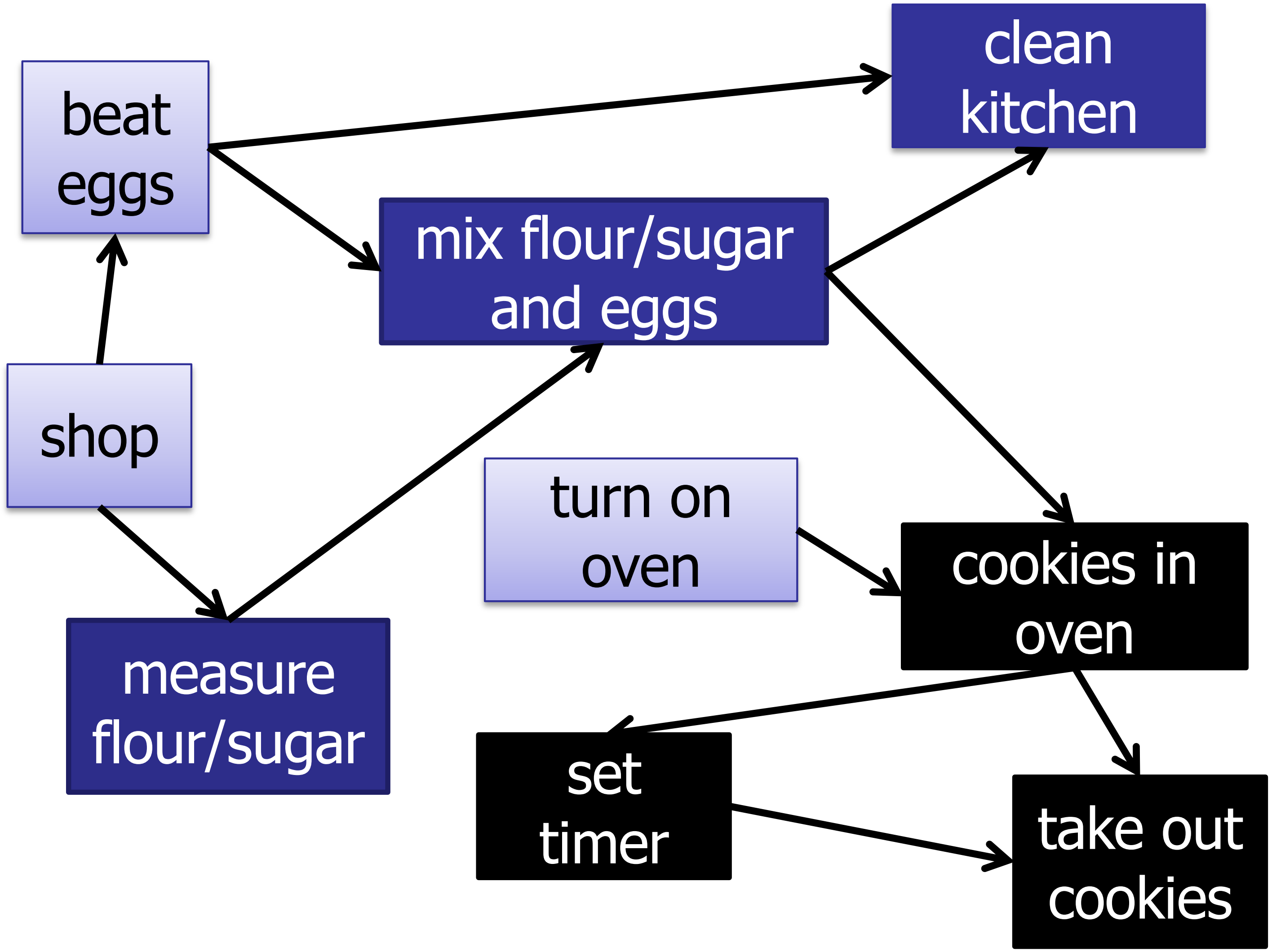
Depth-First Search

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



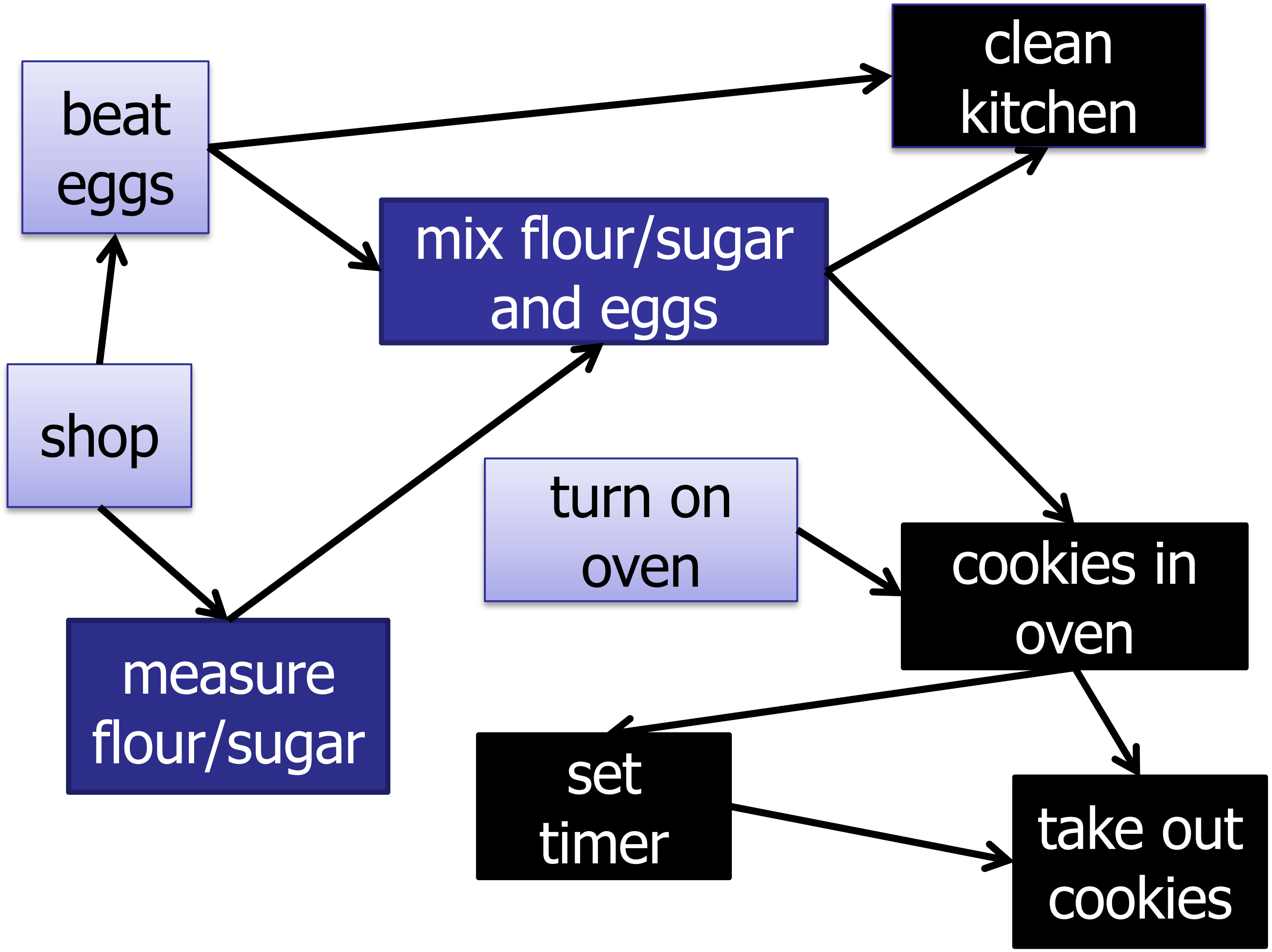
Depth-First Search

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



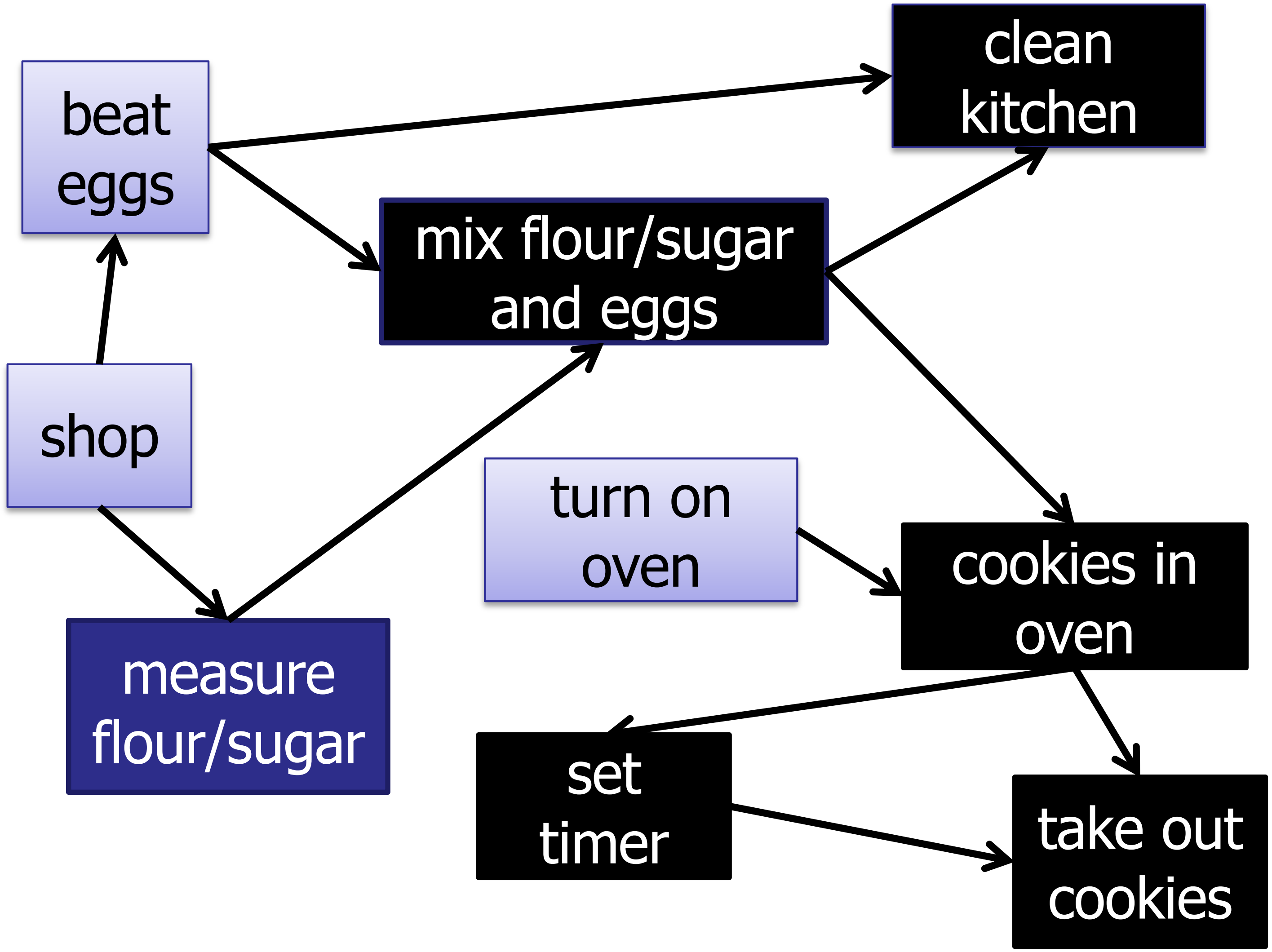
Depth-First Search

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



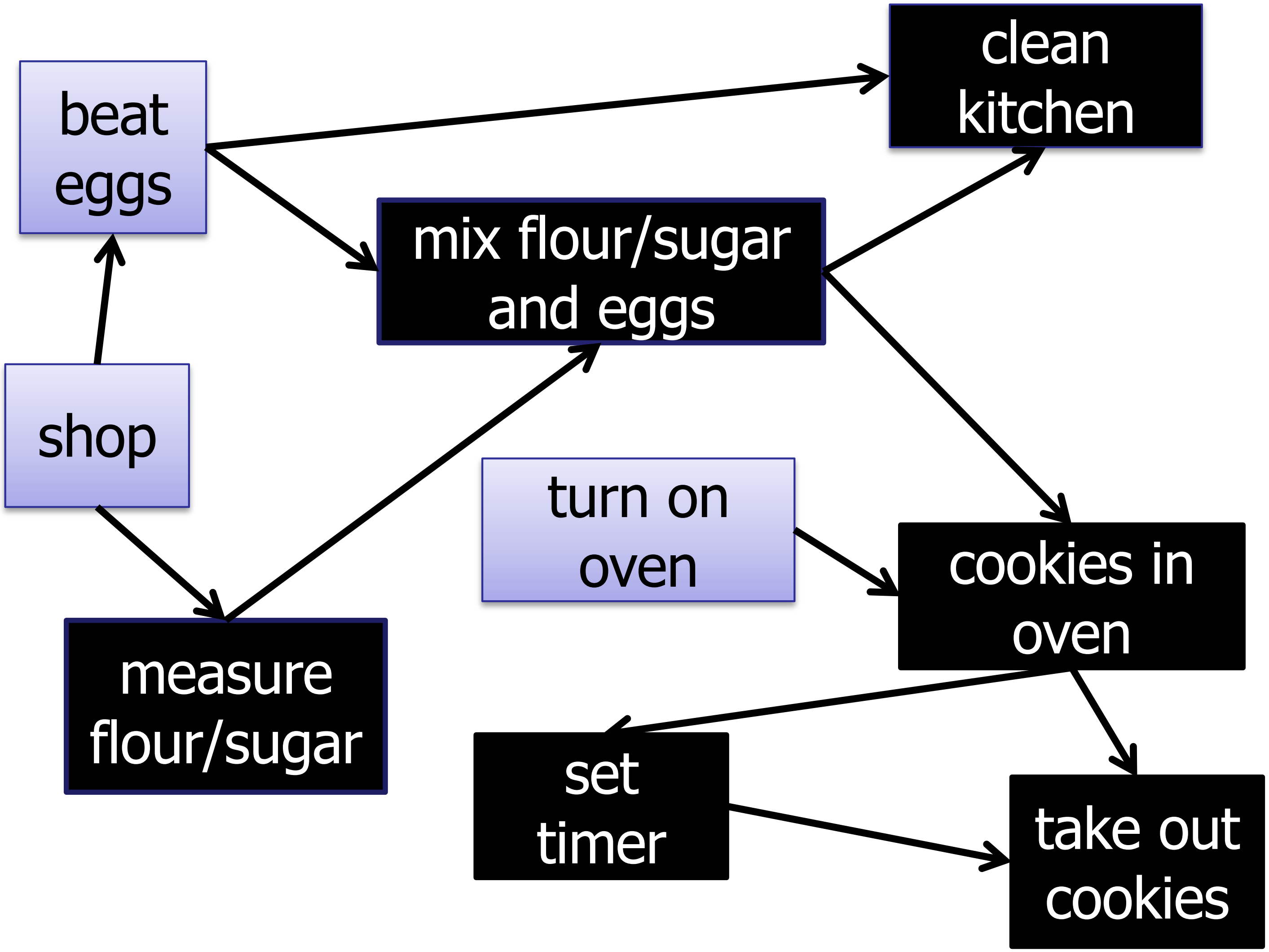
Depth-First Search

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

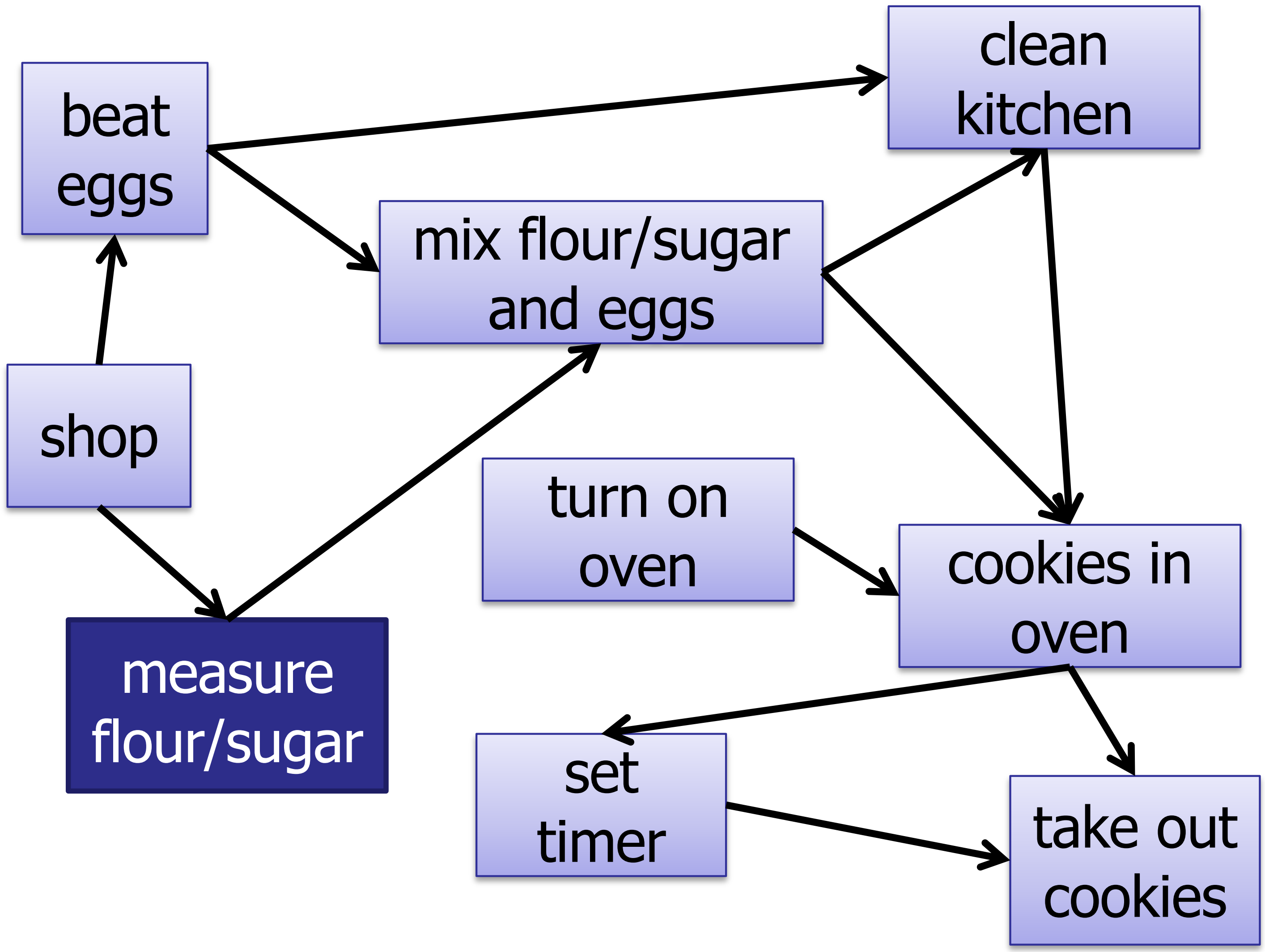
Pre-Order Depth-First Search:

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

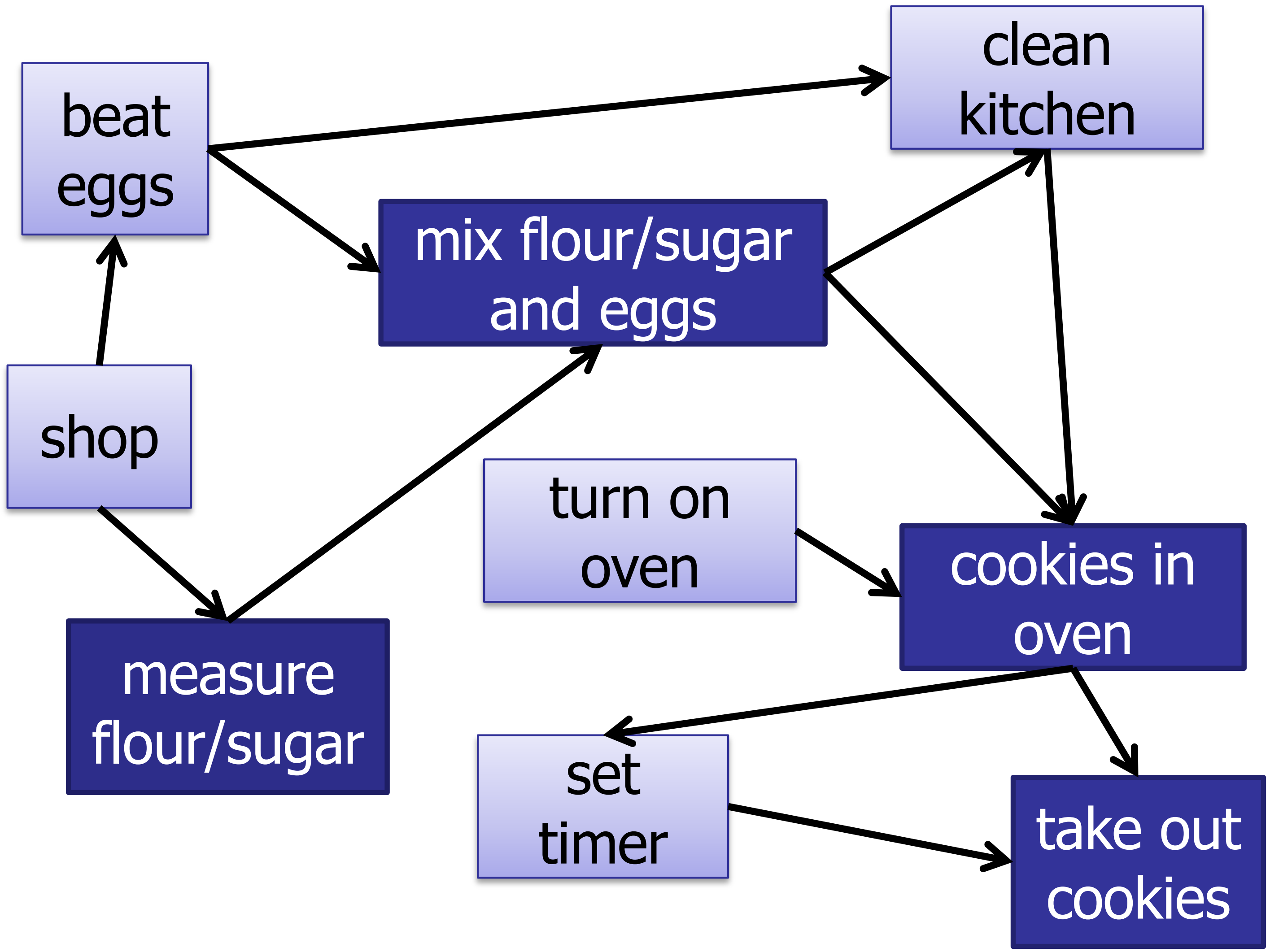
Post-Order Depth-First Search:

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

Post-Order Depth-First Search

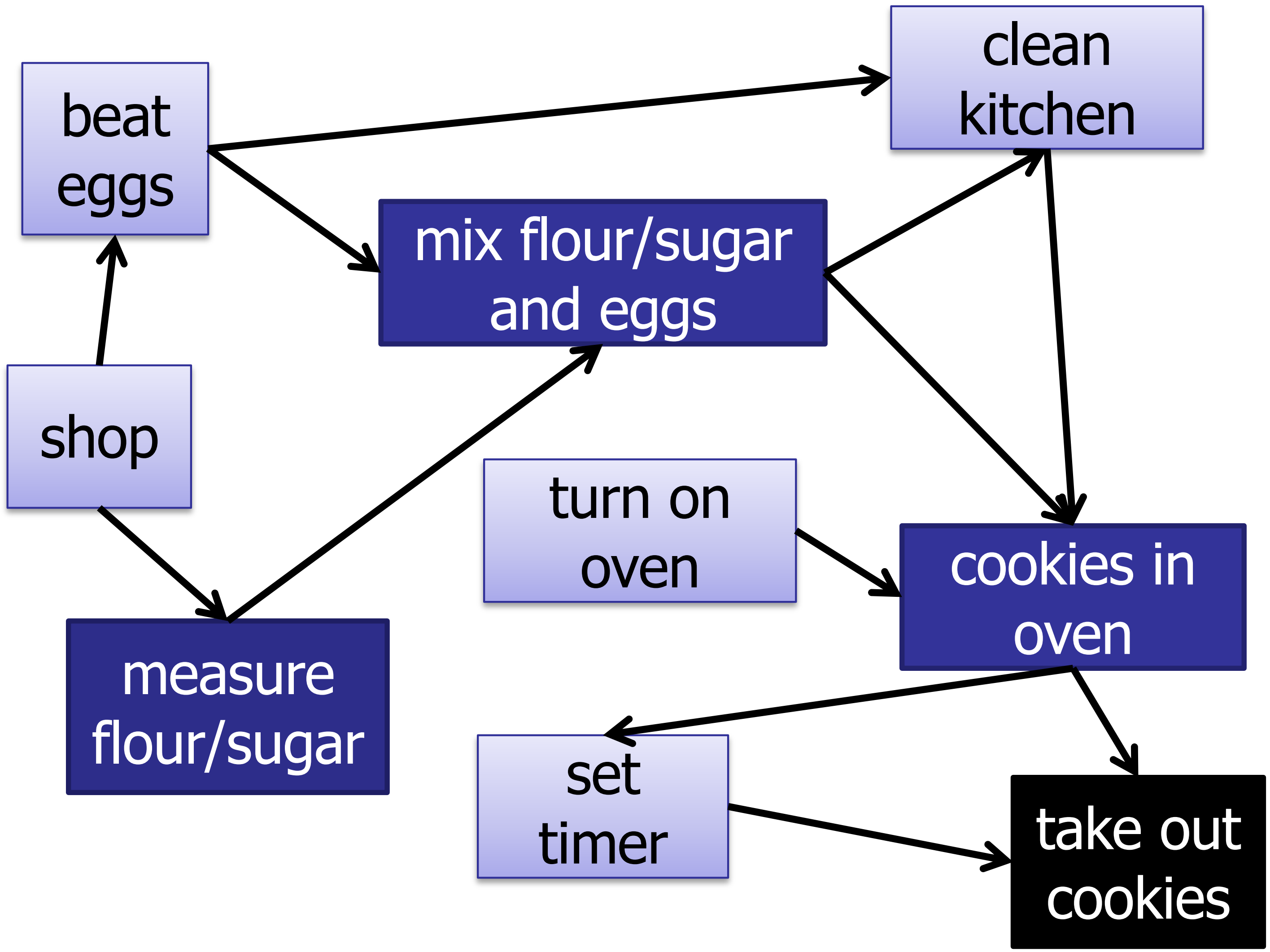


Post-Order Depth-First Search



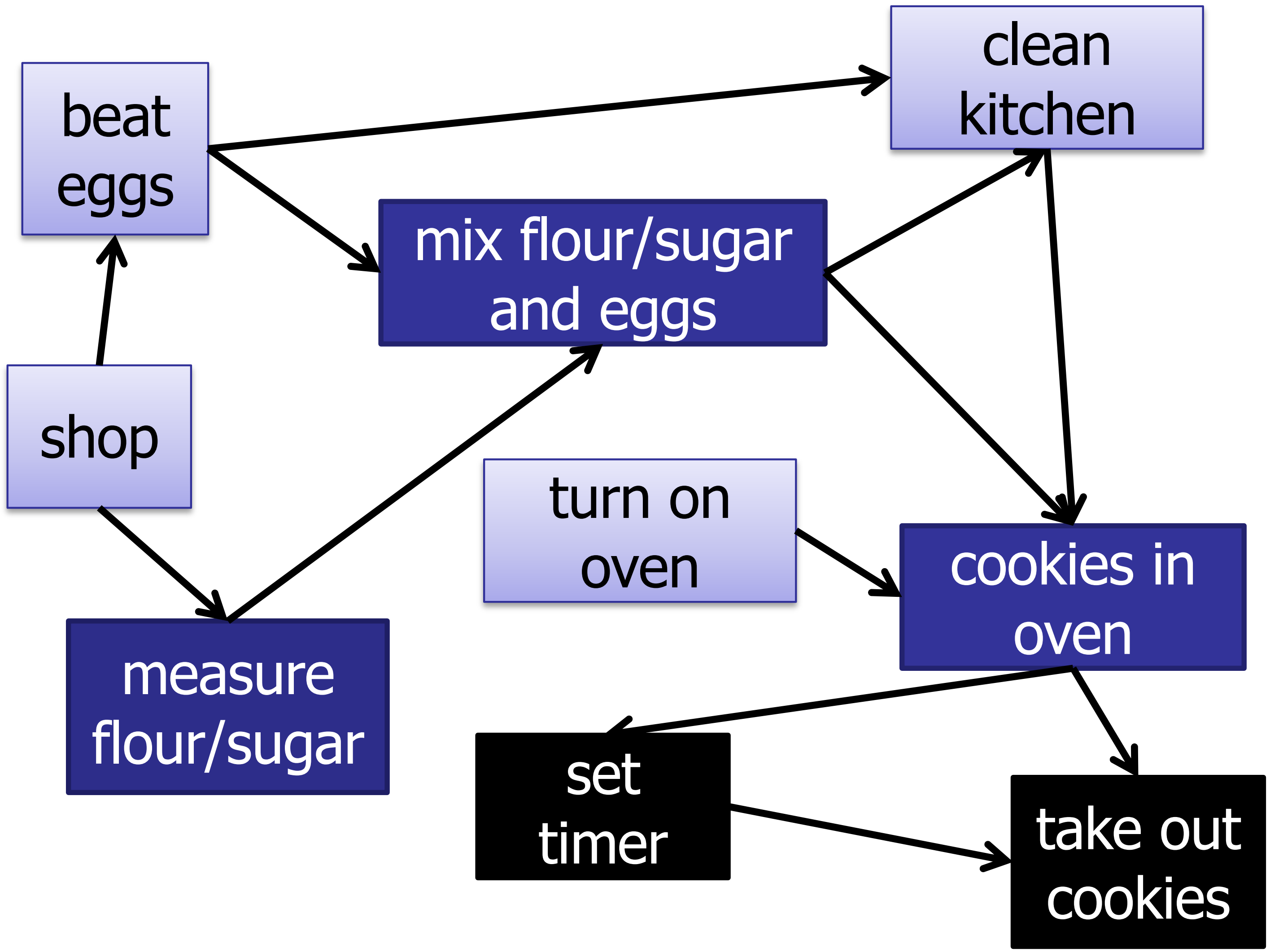
Post-Order Depth-First Search

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



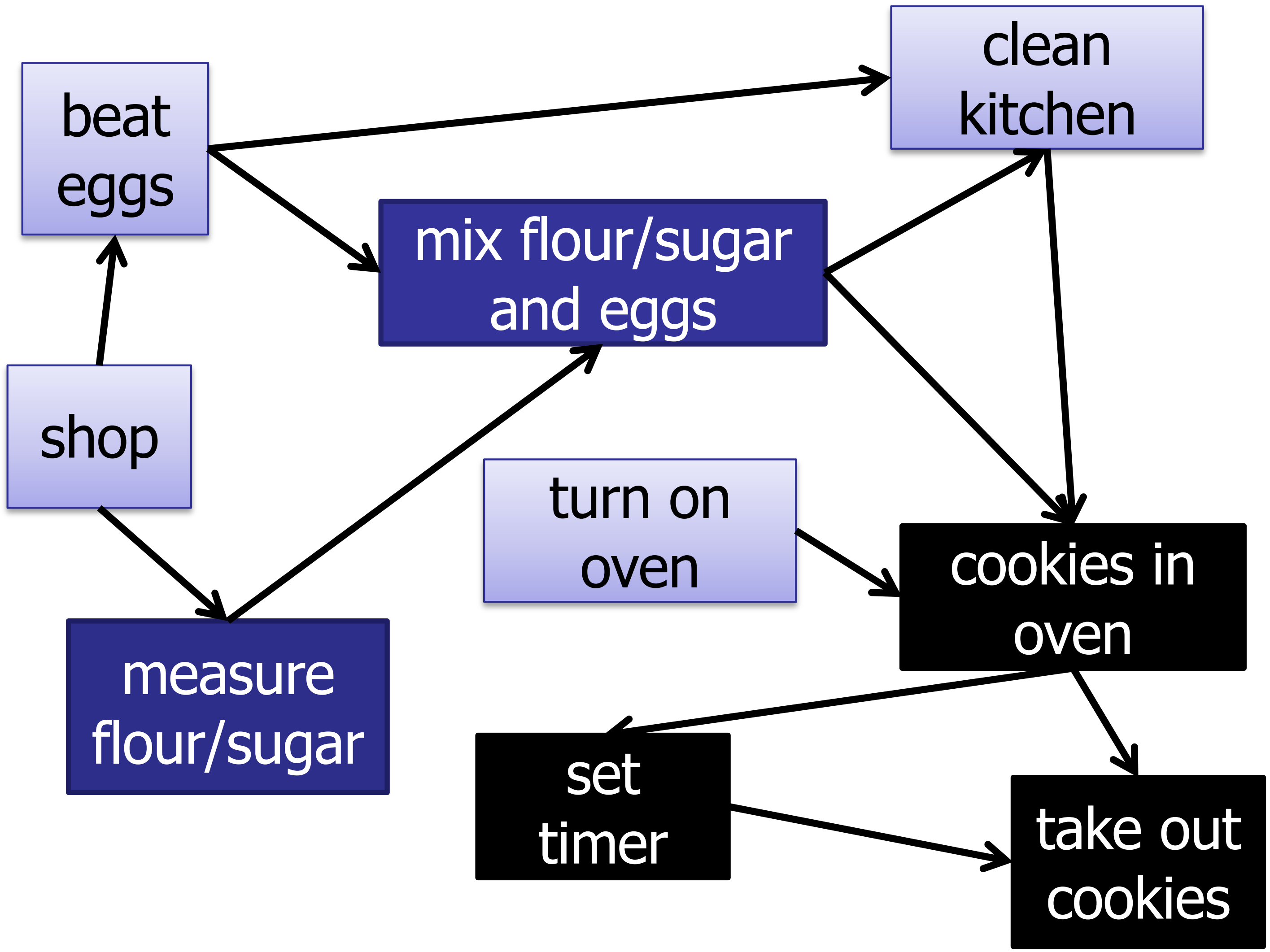
Post-Order Depth-First Search

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



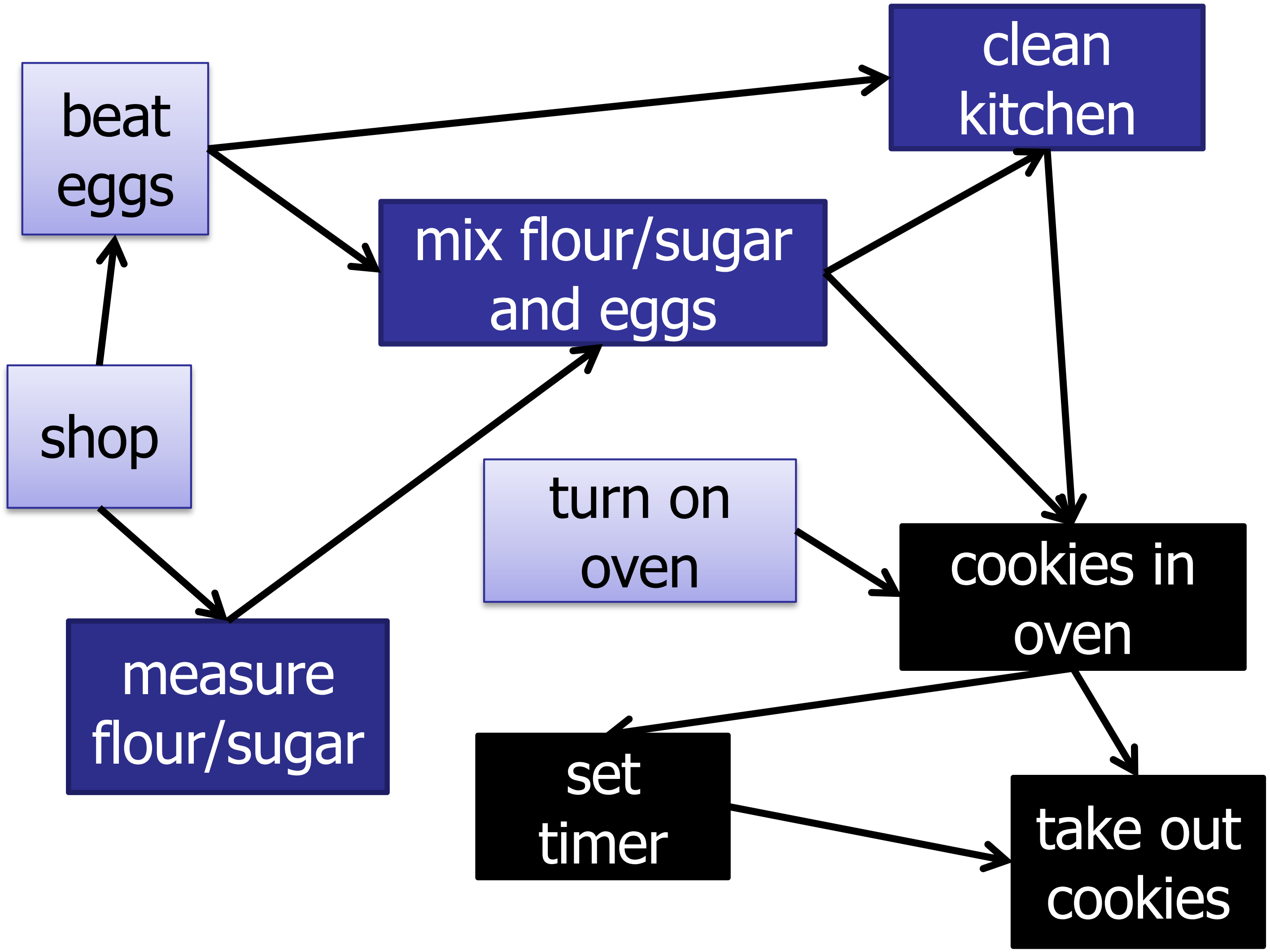
Post-Order Depth-First Search

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



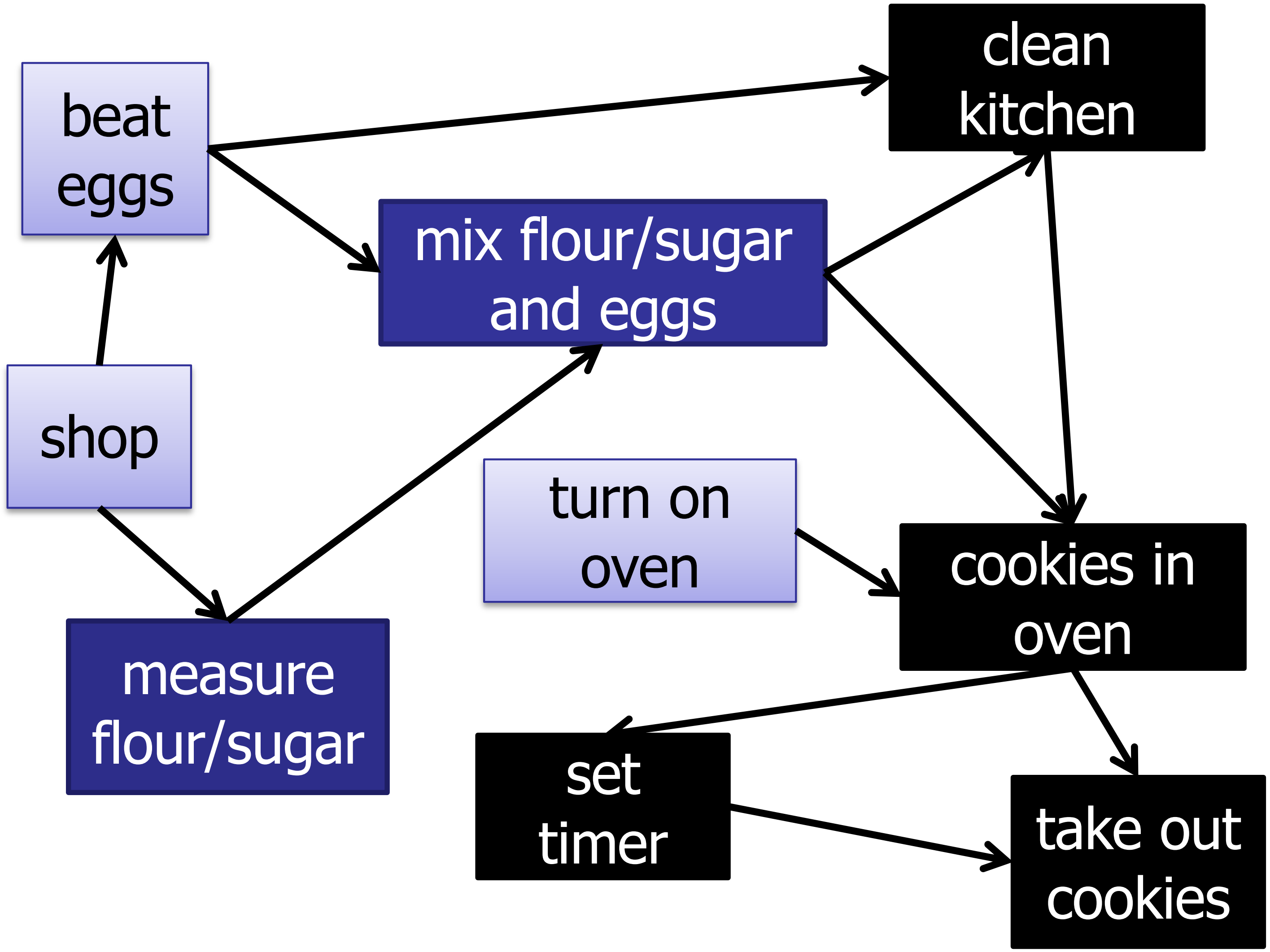
Post-Order Depth-First Search

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



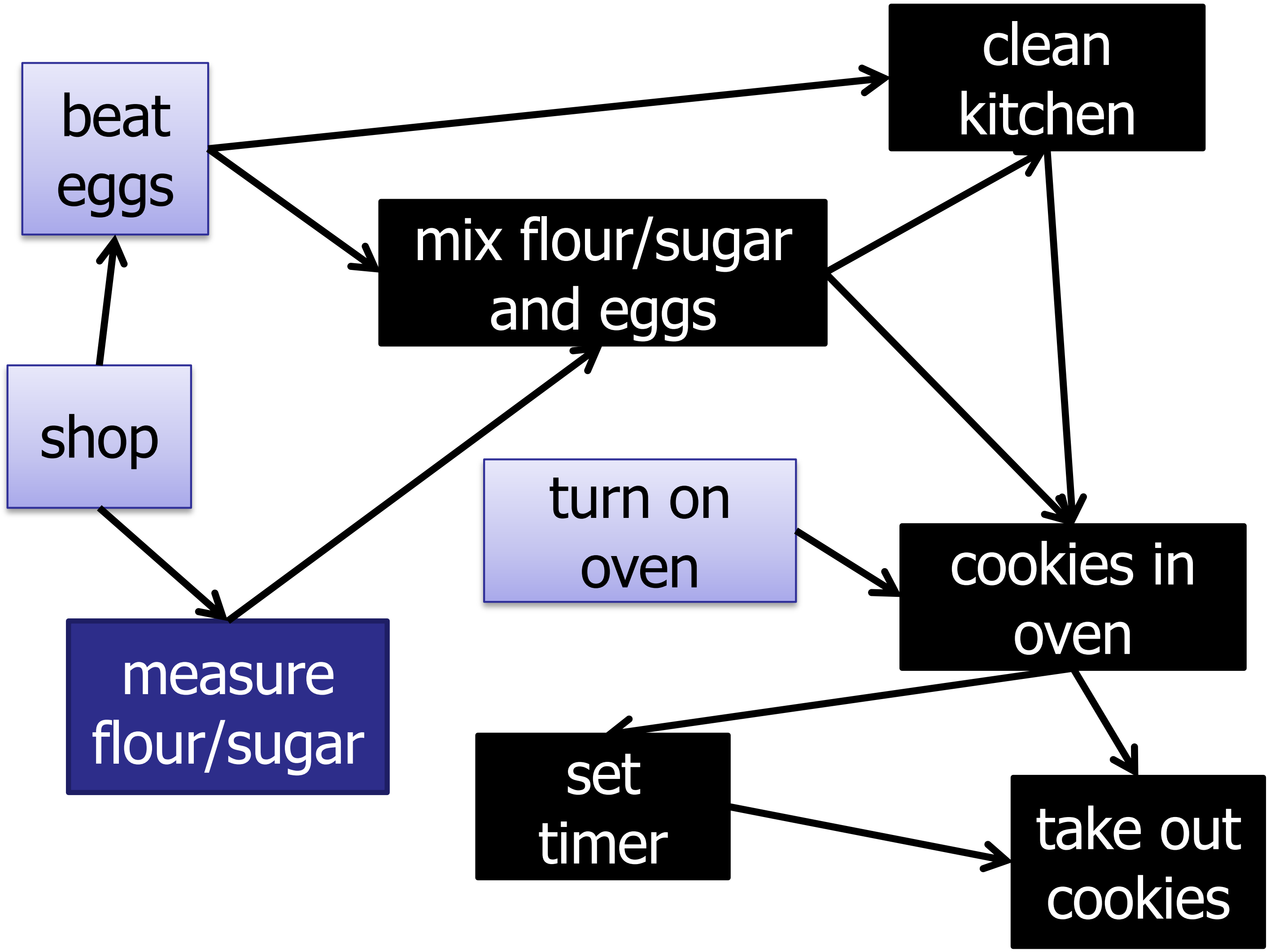
Post-Order Depth-First Search

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



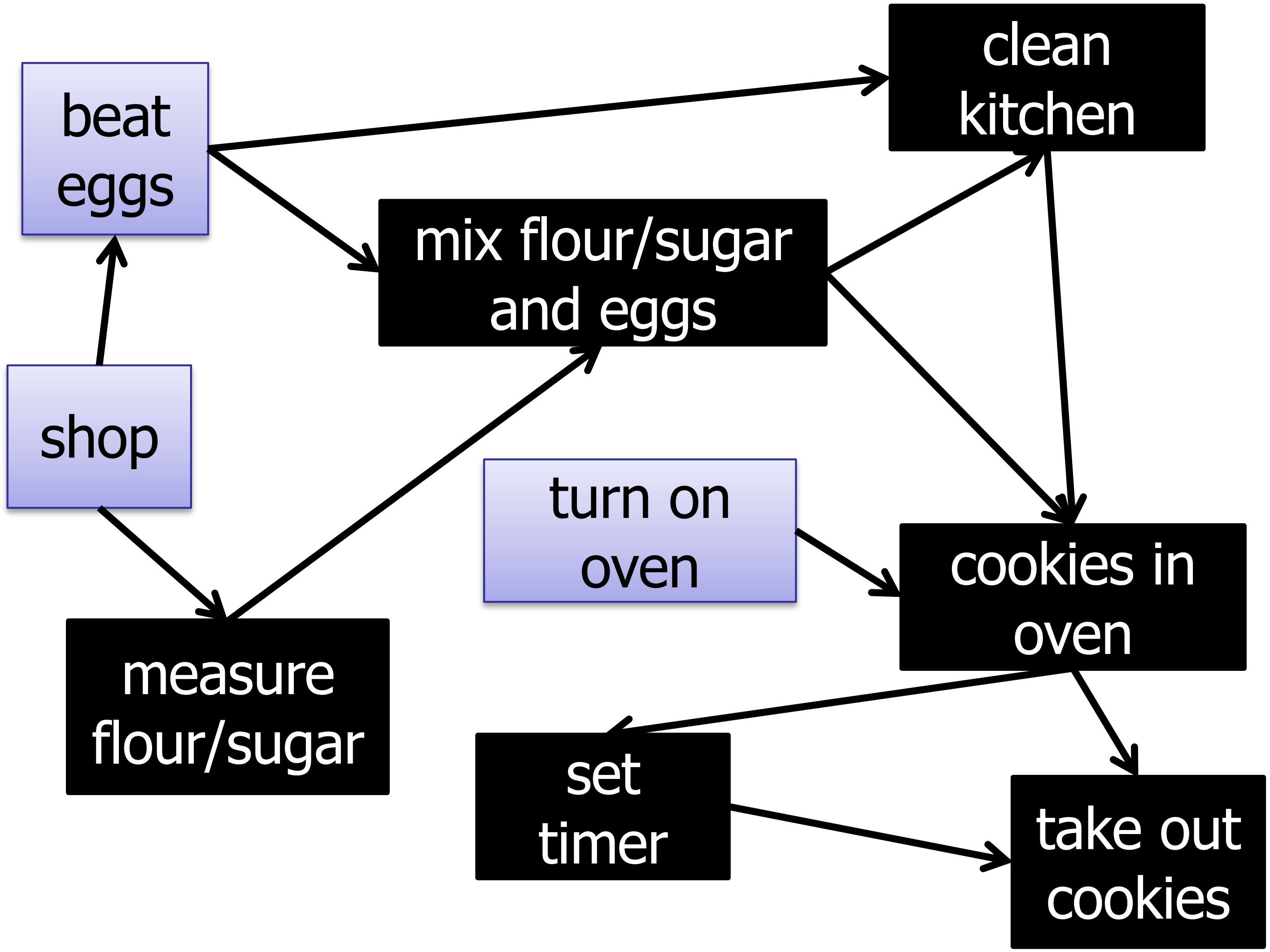
Post-Order Depth-First Search

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



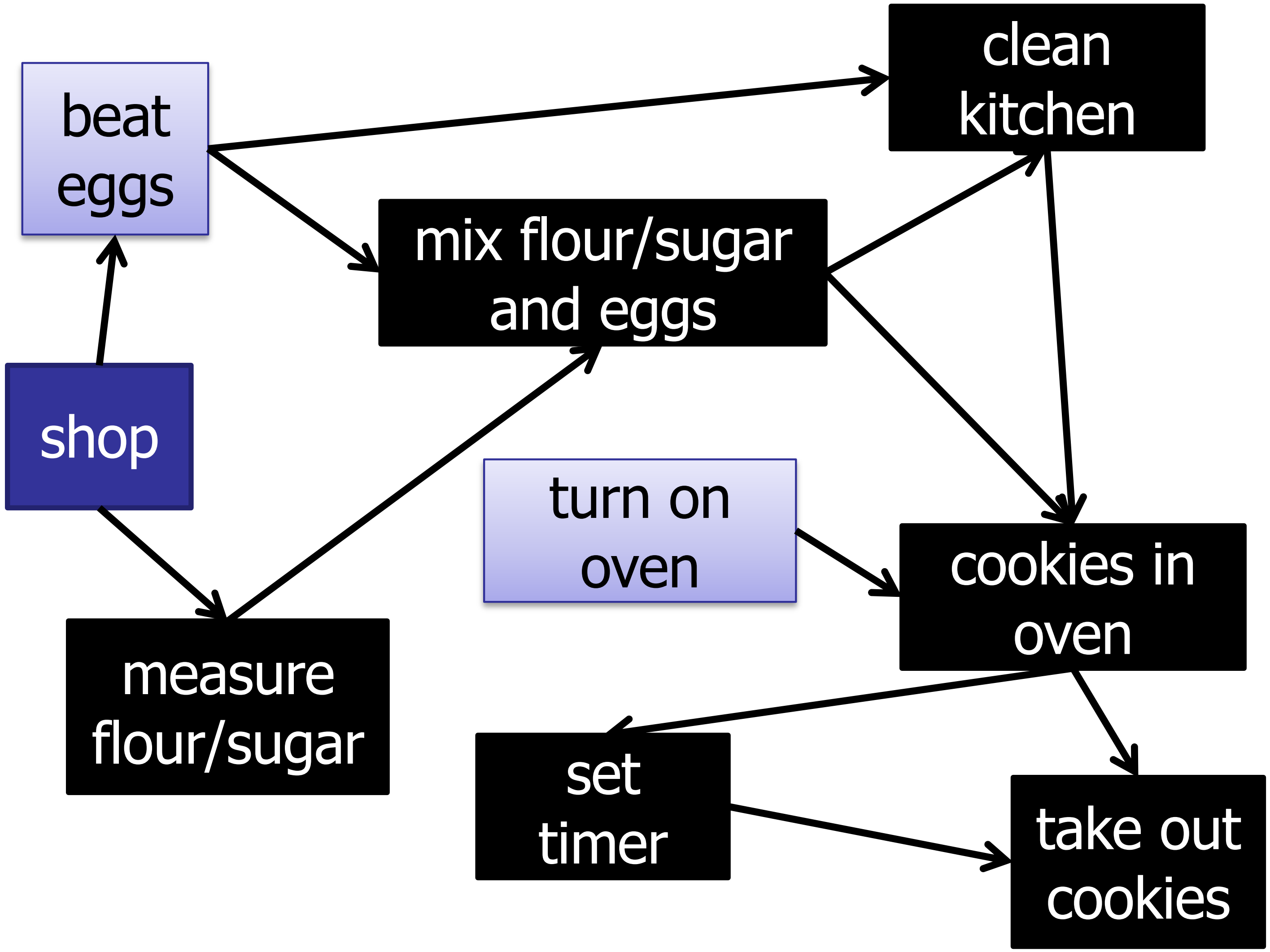
Post-Order Depth-First Search

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



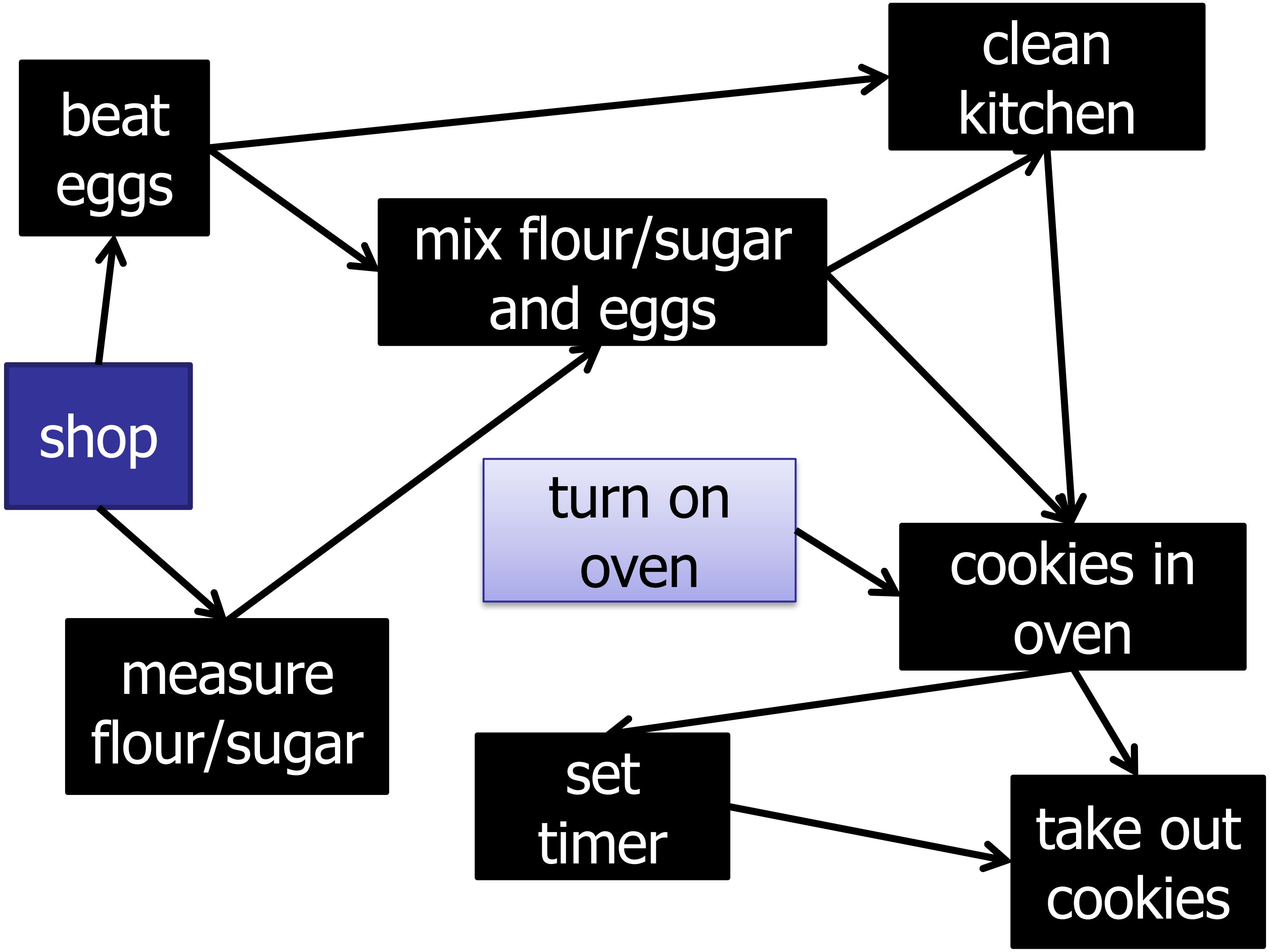
Post-Order Depth-First Search

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



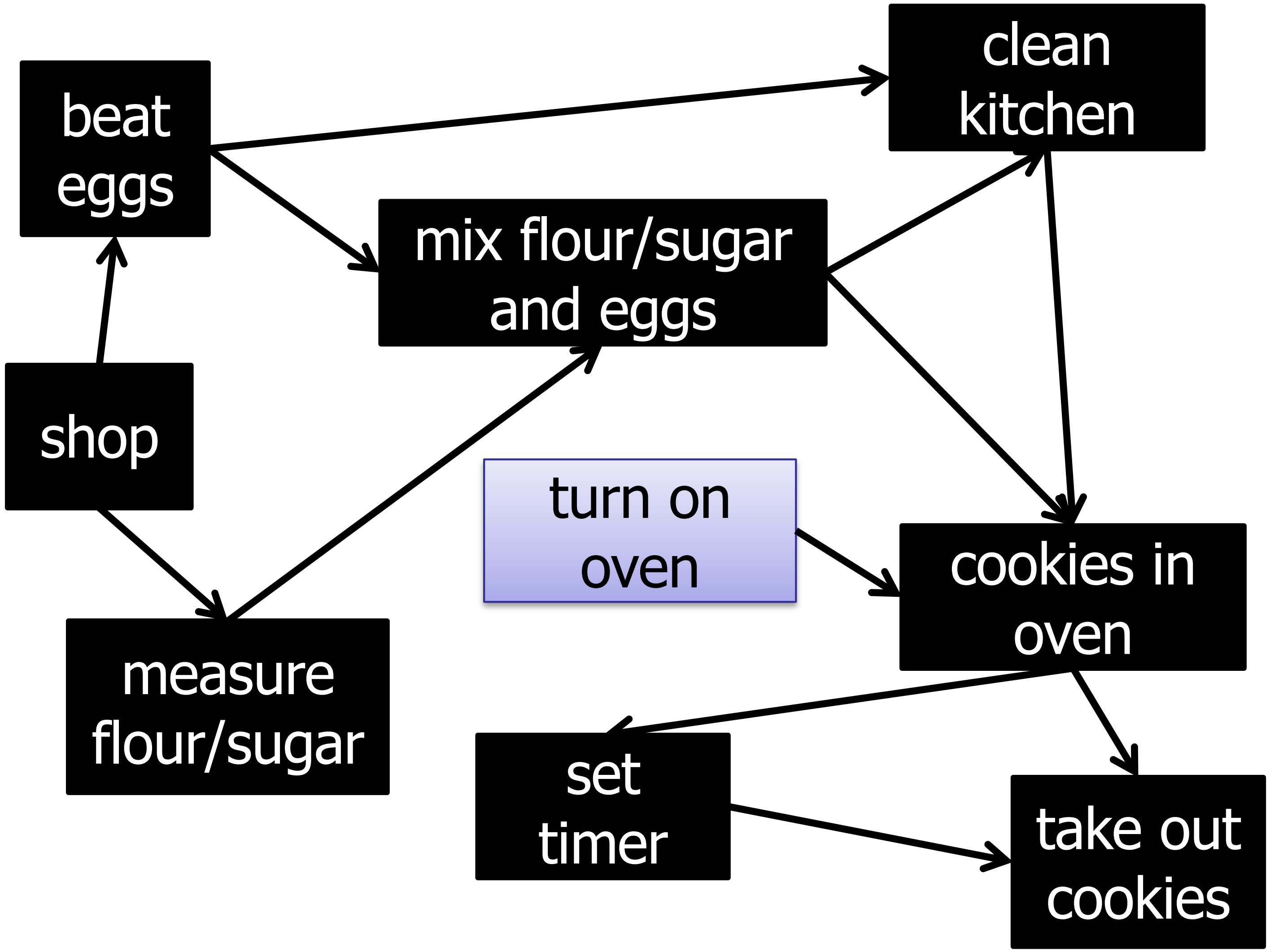
Post-Order Depth-First Search

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



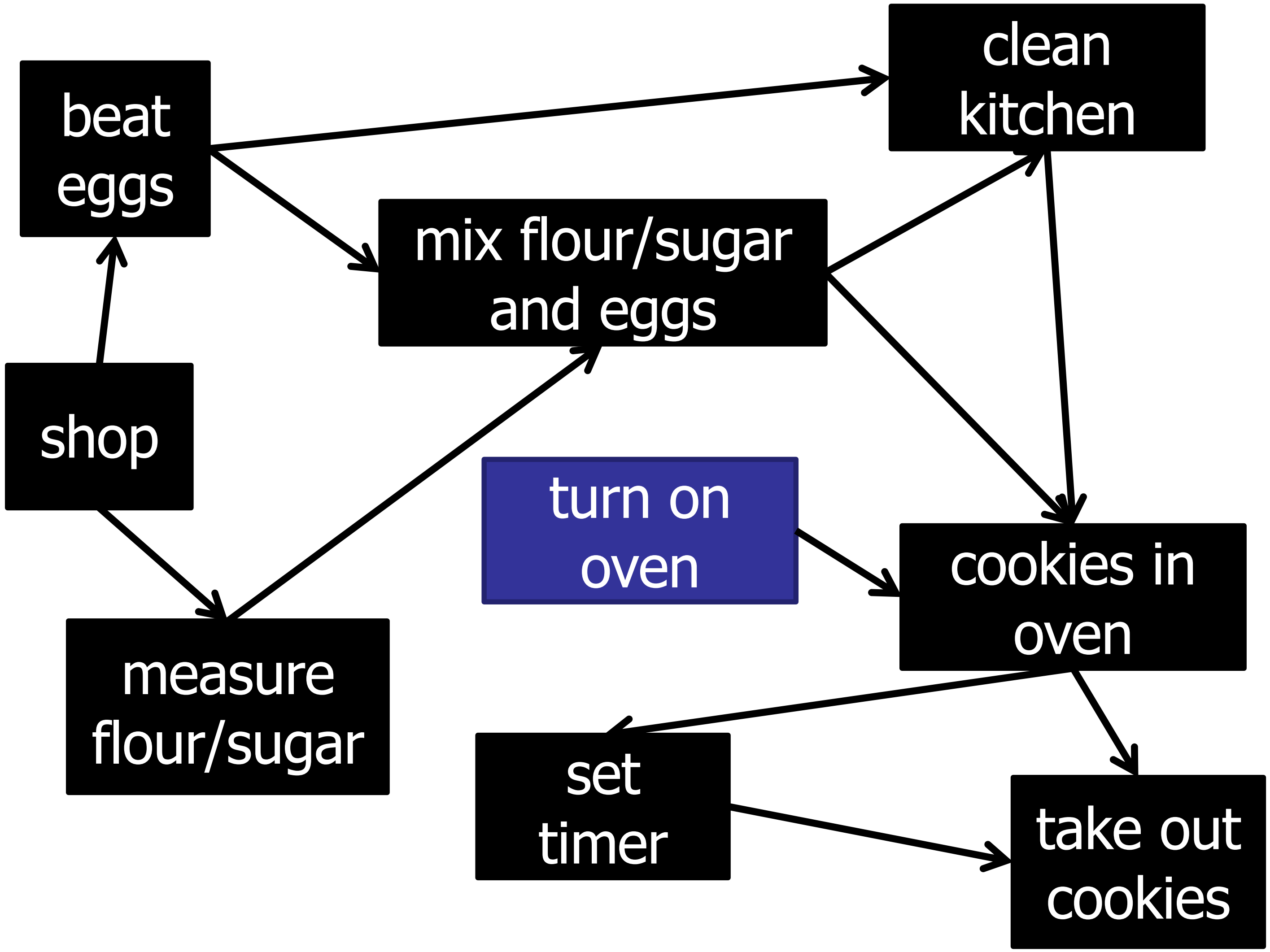
Post-Order Depth-First Search

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



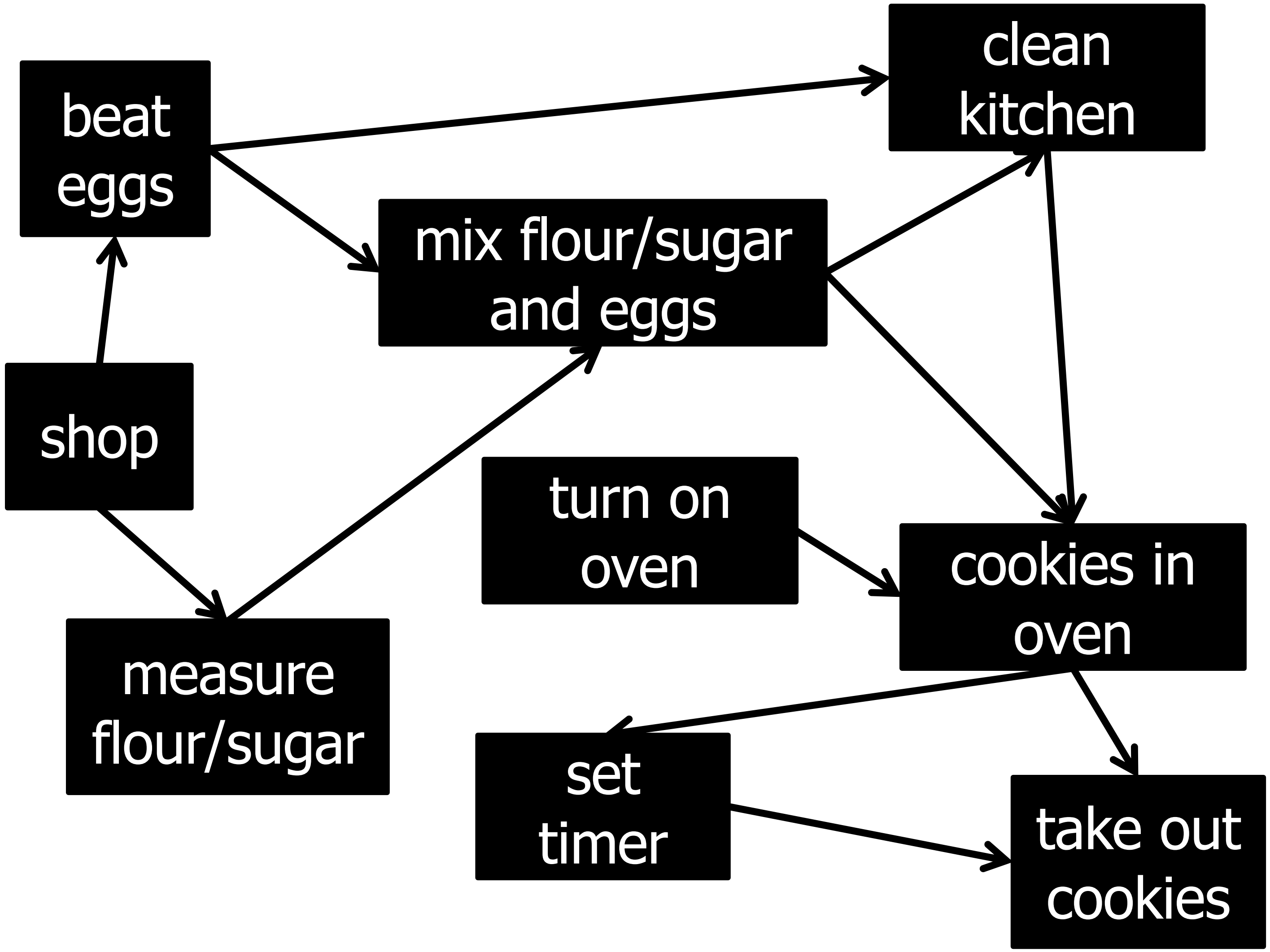
Post-Order Depth-First Search

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);
     }
  }
```

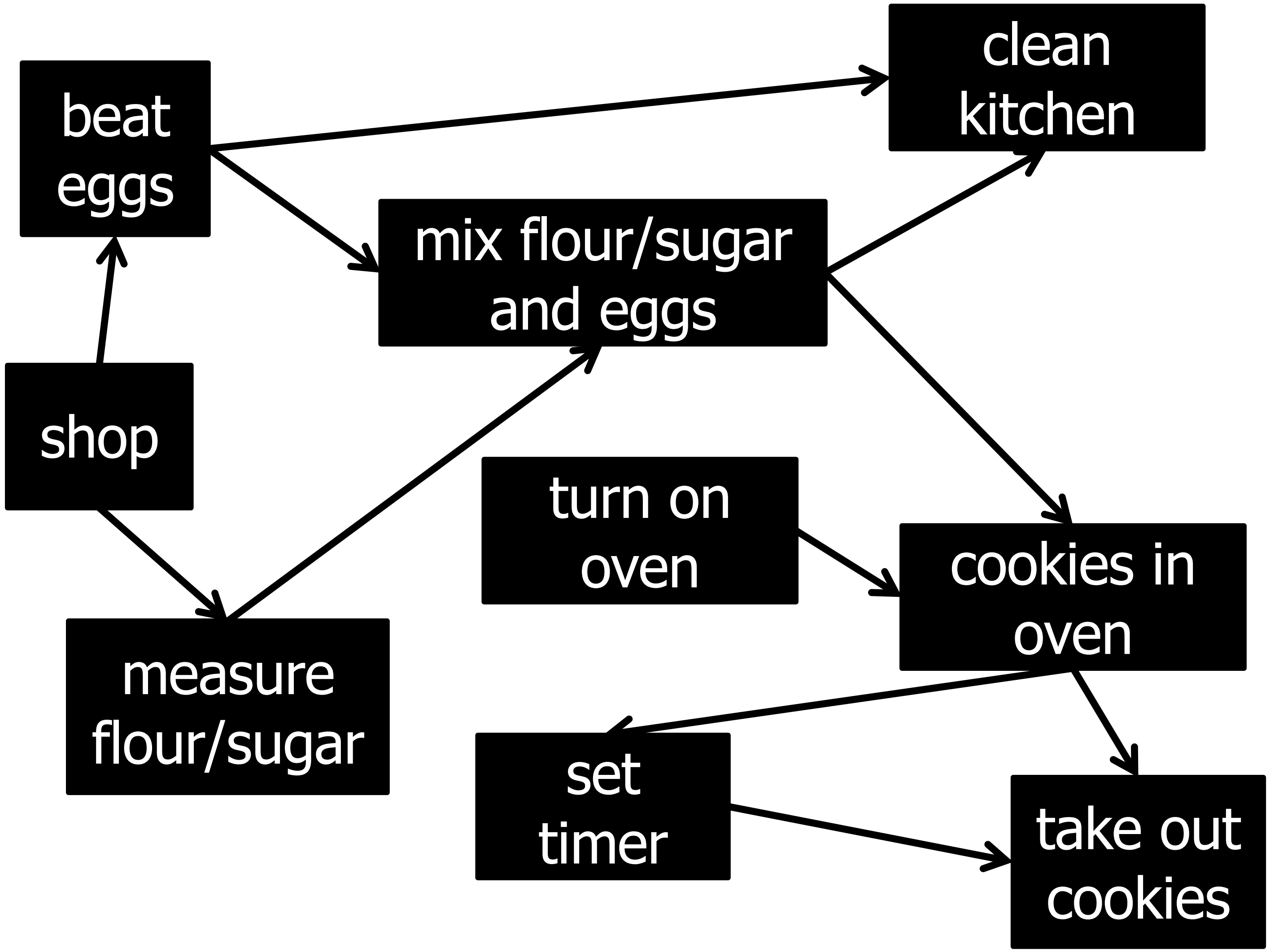
Is a topological ordering unique?

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

ARCHIPELAGO
is open

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 approach: Kahn's 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

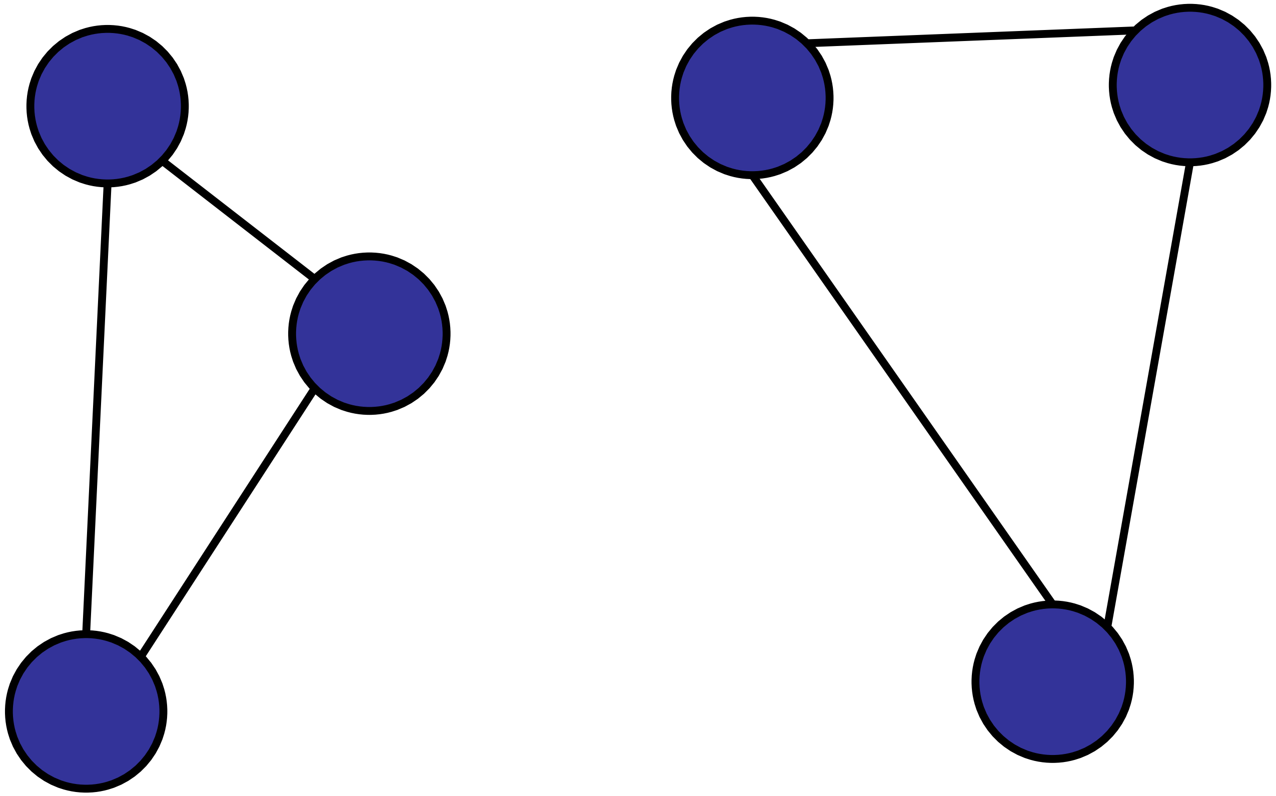
Roadmap

Directed Graphs

- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

Connected Components

Undirected graphs

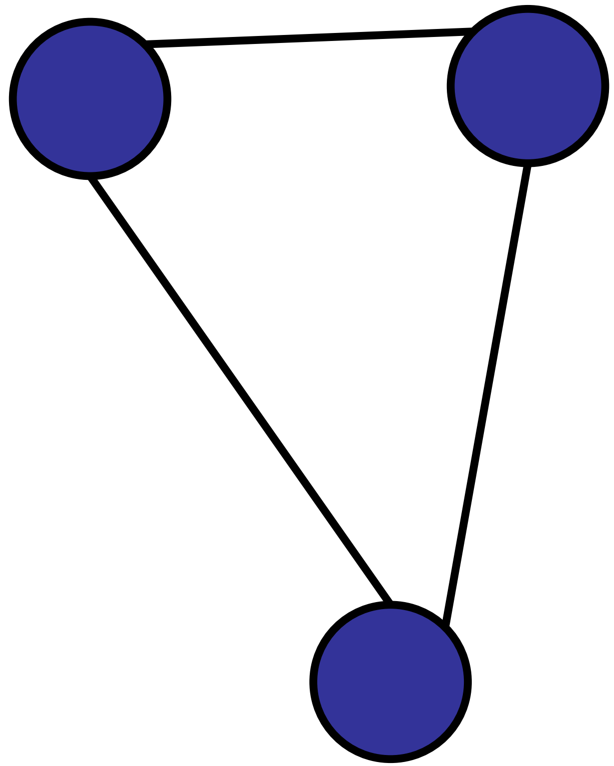


Two connected components

Connected Components

Undirected graphs

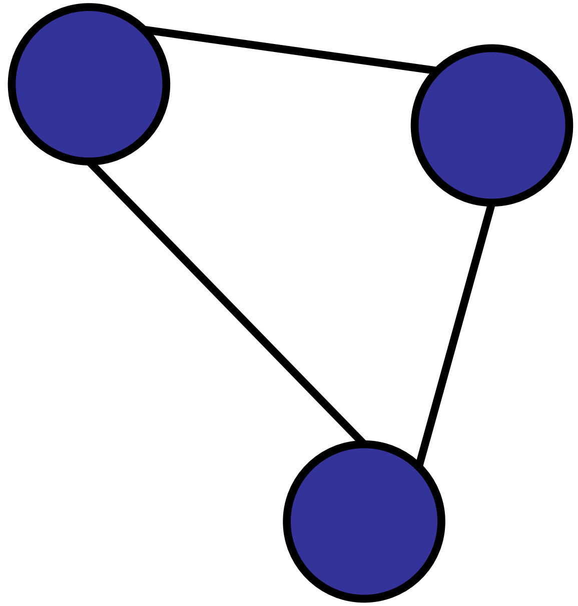
Vertex v and w are in the same <u>connected component</u> if and only if there is a path from v to w.



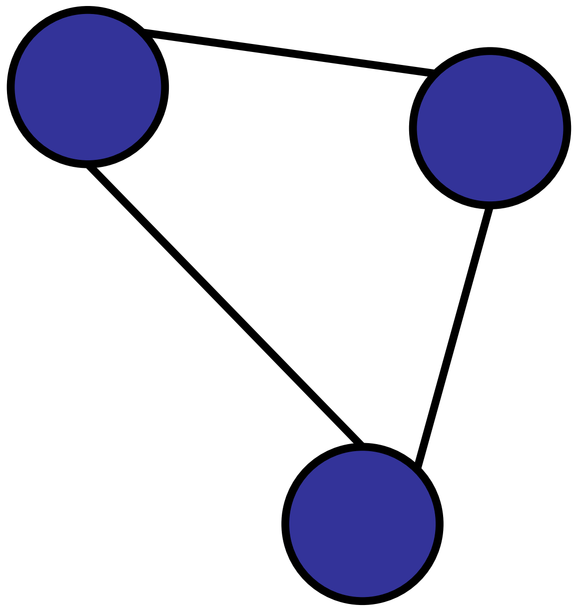
Connected Components

Undirected graphs

Vertex v and w are in the same <u>connected component</u> if and only if there is a path from v to w.

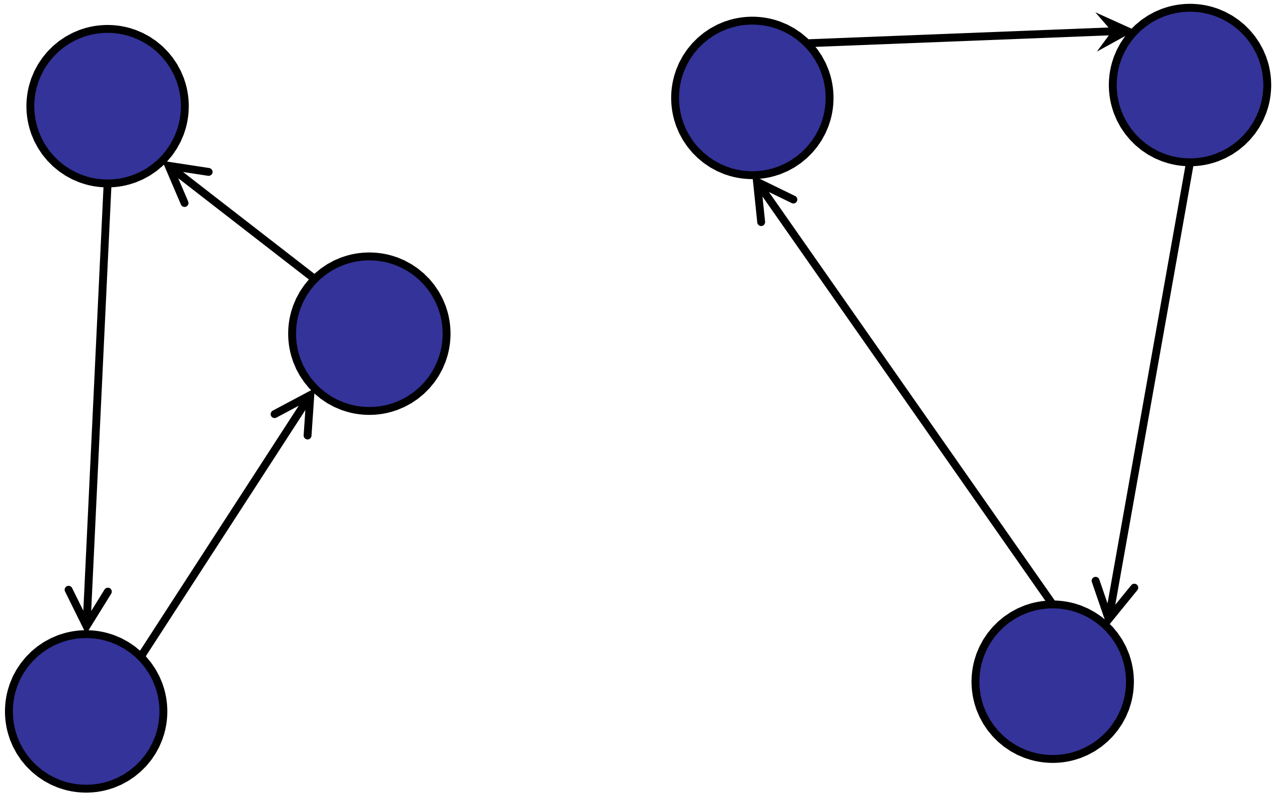


There is a set $\{v_1, v_2, \ldots, v_k\}$ where there is no path from any v_i to v_j if and only if there are k connected components.



Connected Components

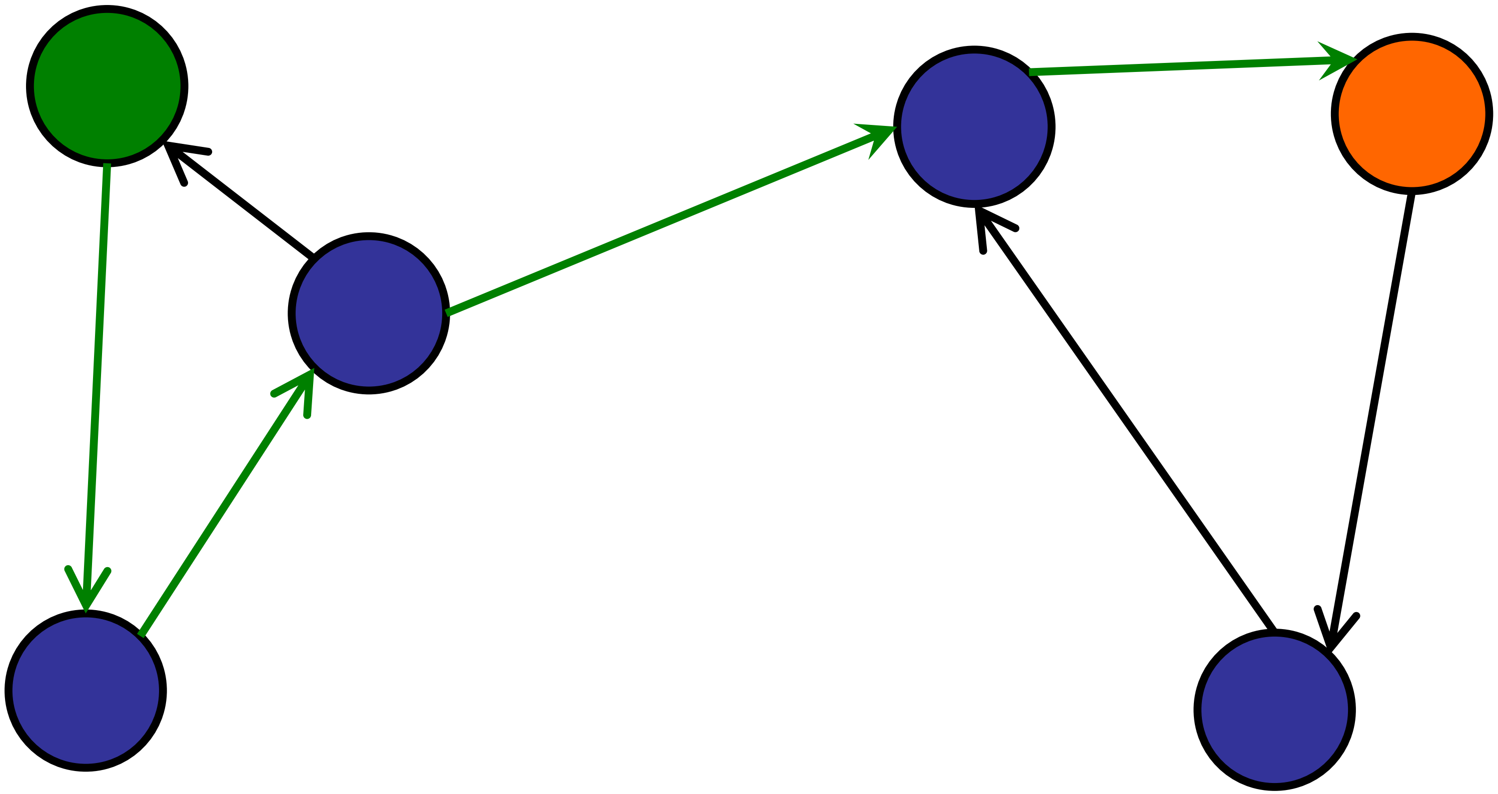
Directed graphs



Two connected components

Connected Components

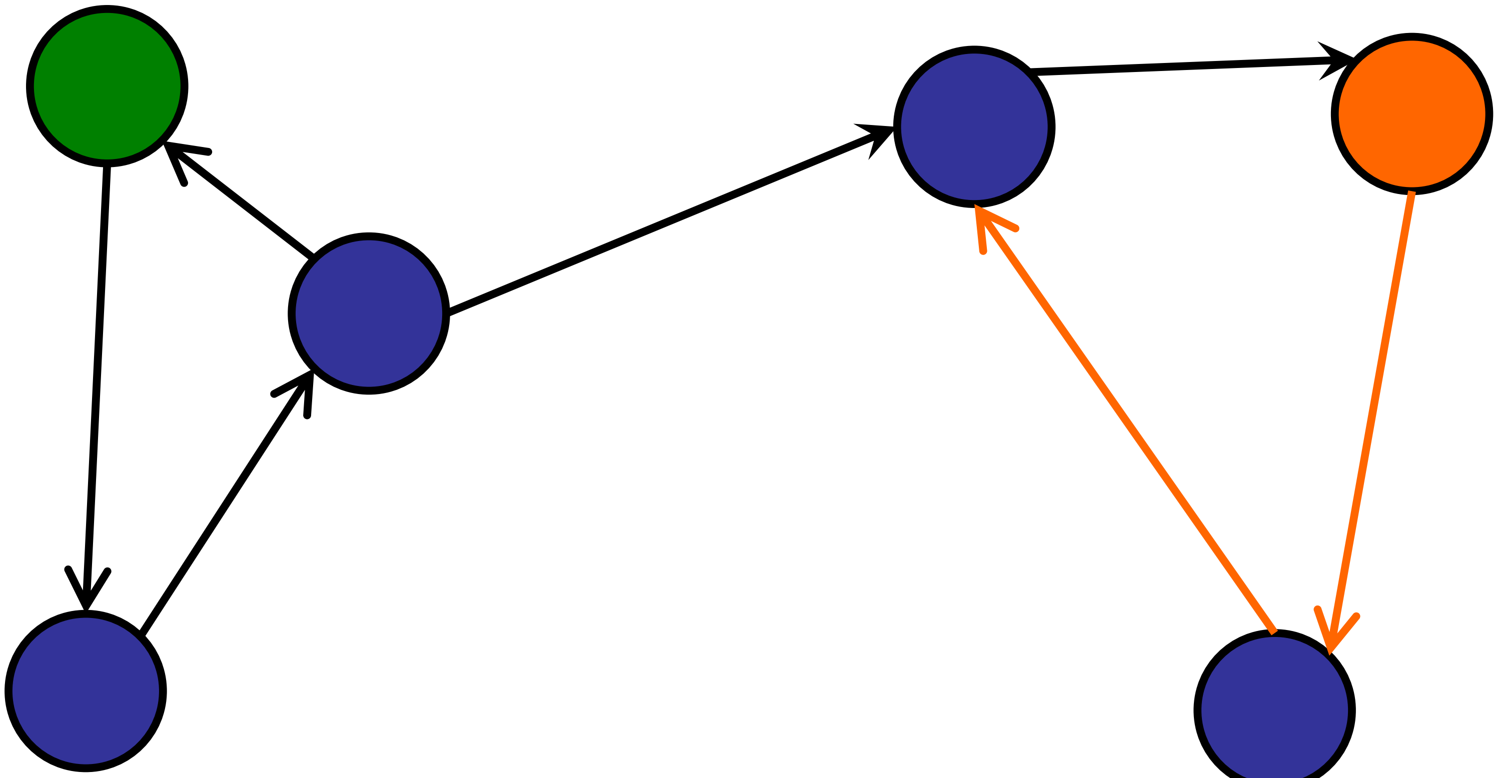
Directed graphs



Two connected components??

Connected Components

Directed graphs



Two connected components??

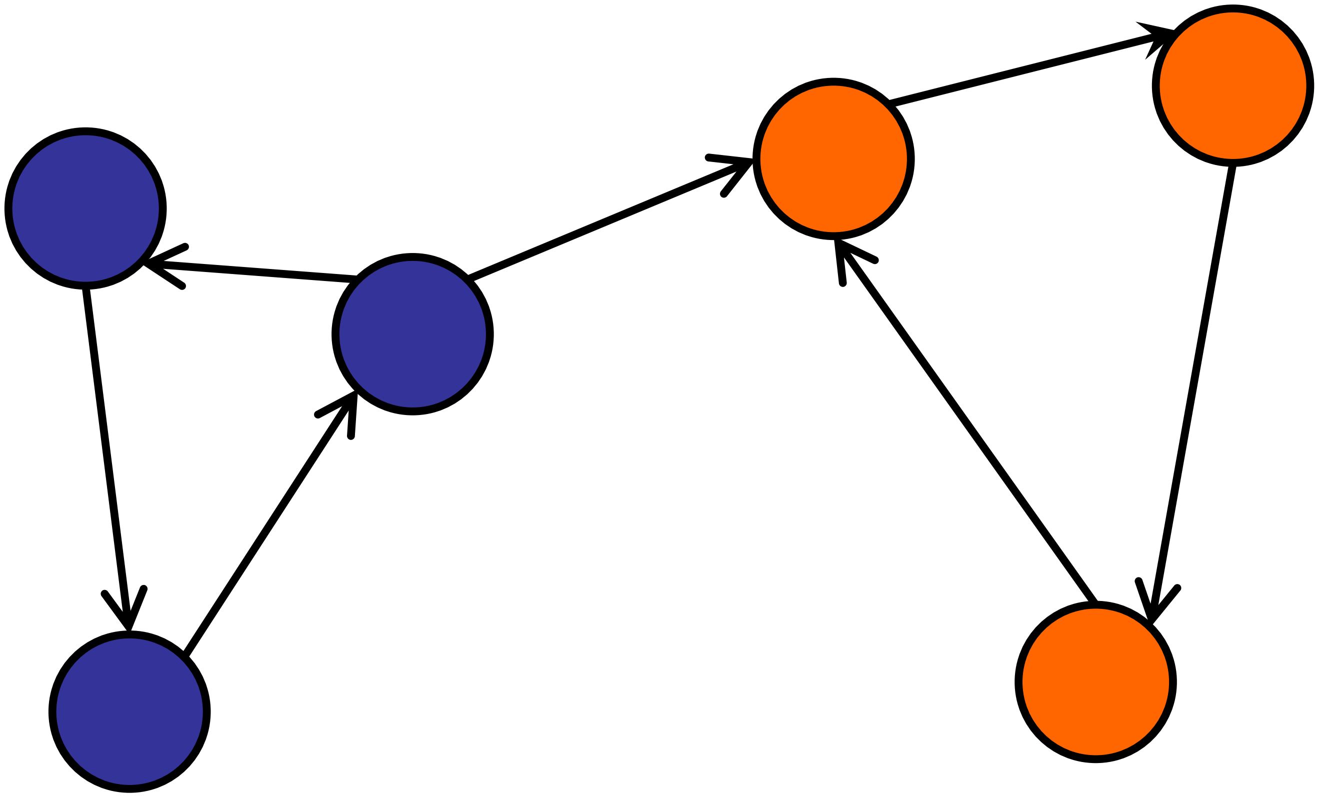
Connected Components

Strongly connected component

For every vertex v and w:

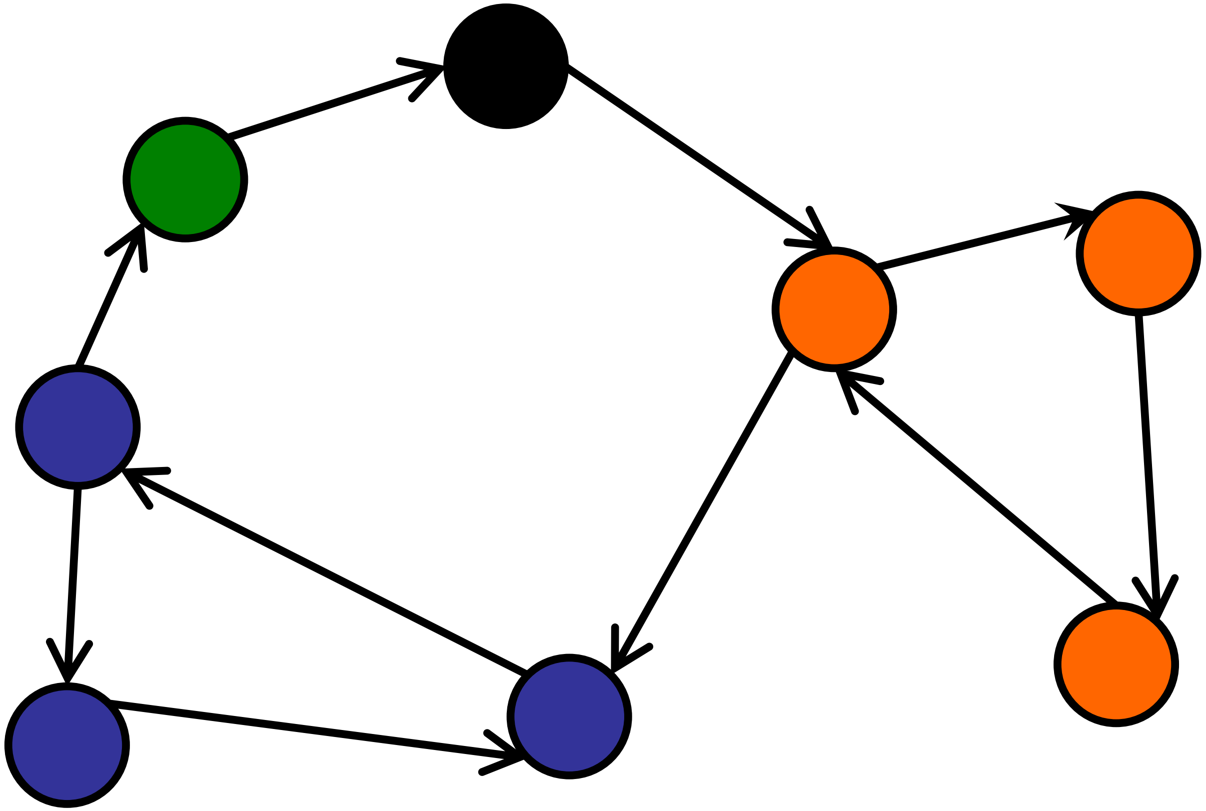
–There is a path from v to w.

– There is a path from w to v.



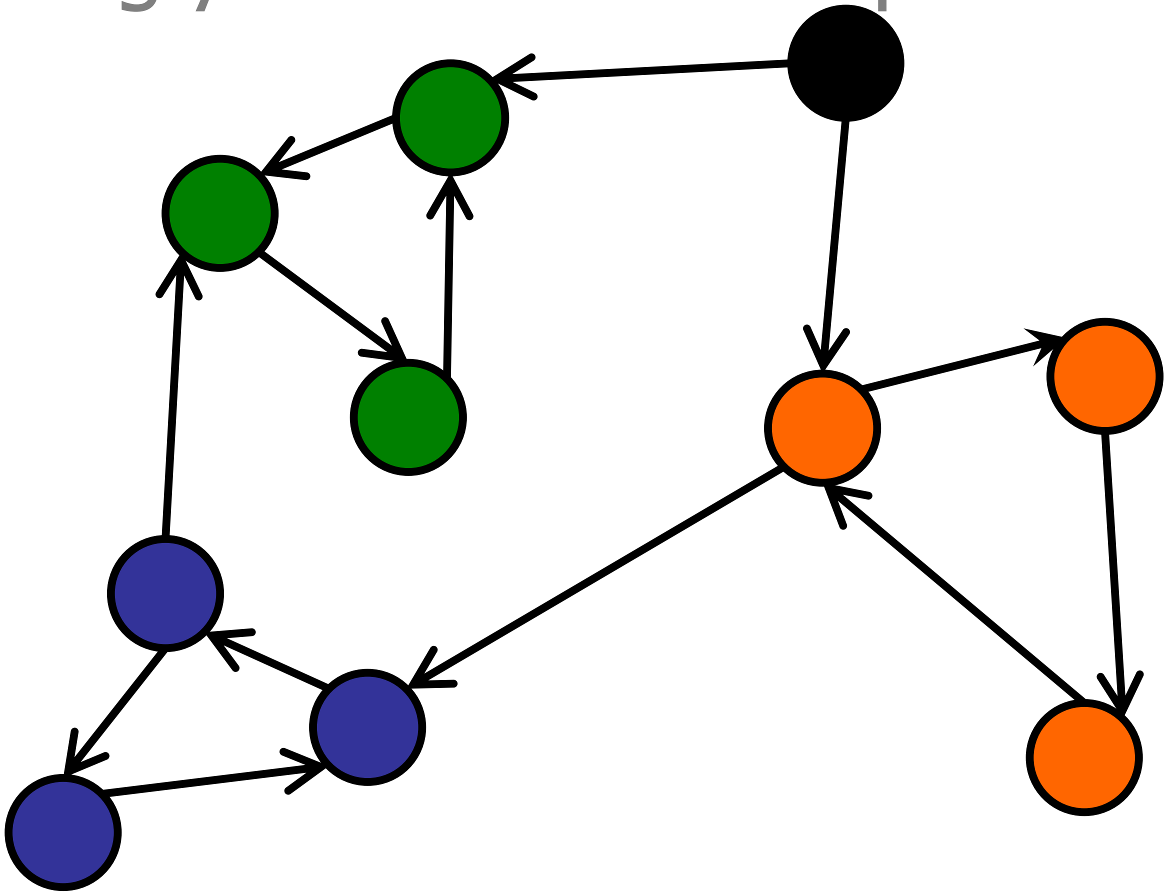
How many strongly connected components?

1. 1 ✓
2. 2
3. 3
4. 4
5. 5
6. Other

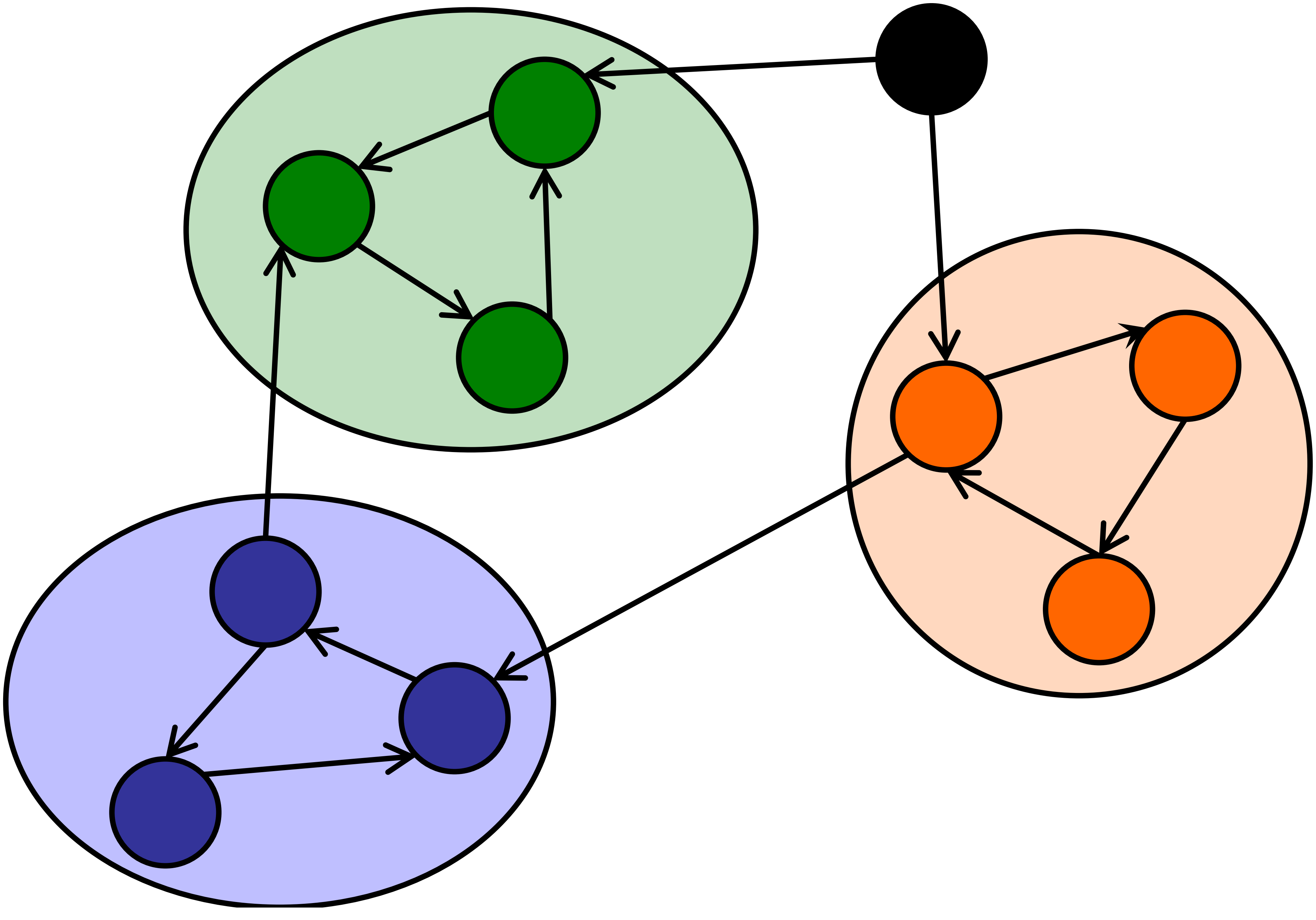


How many strongly connected components?

1. 1
2. 2
3. 3
4. 4
5. 5
6. Other

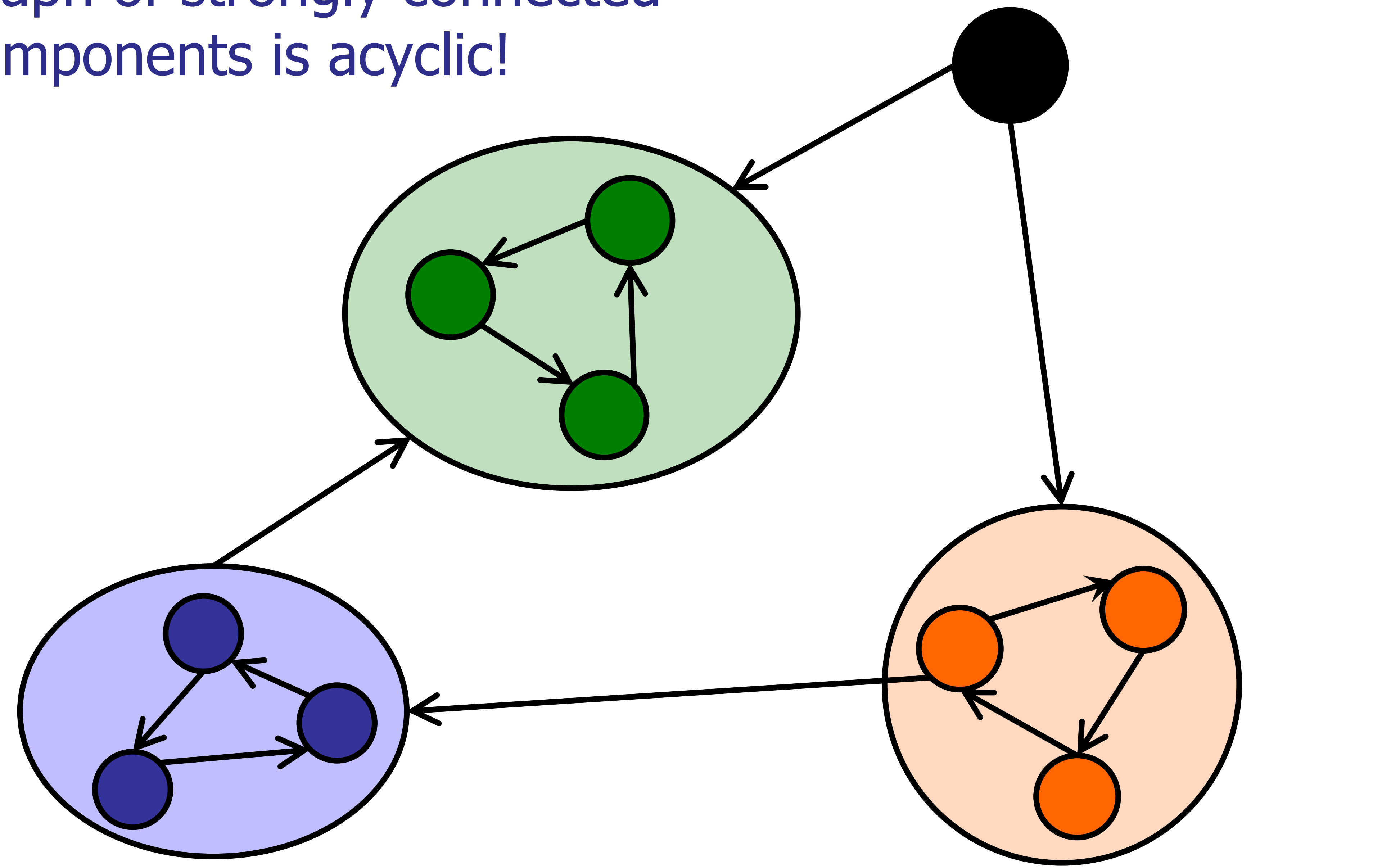


Connected Components



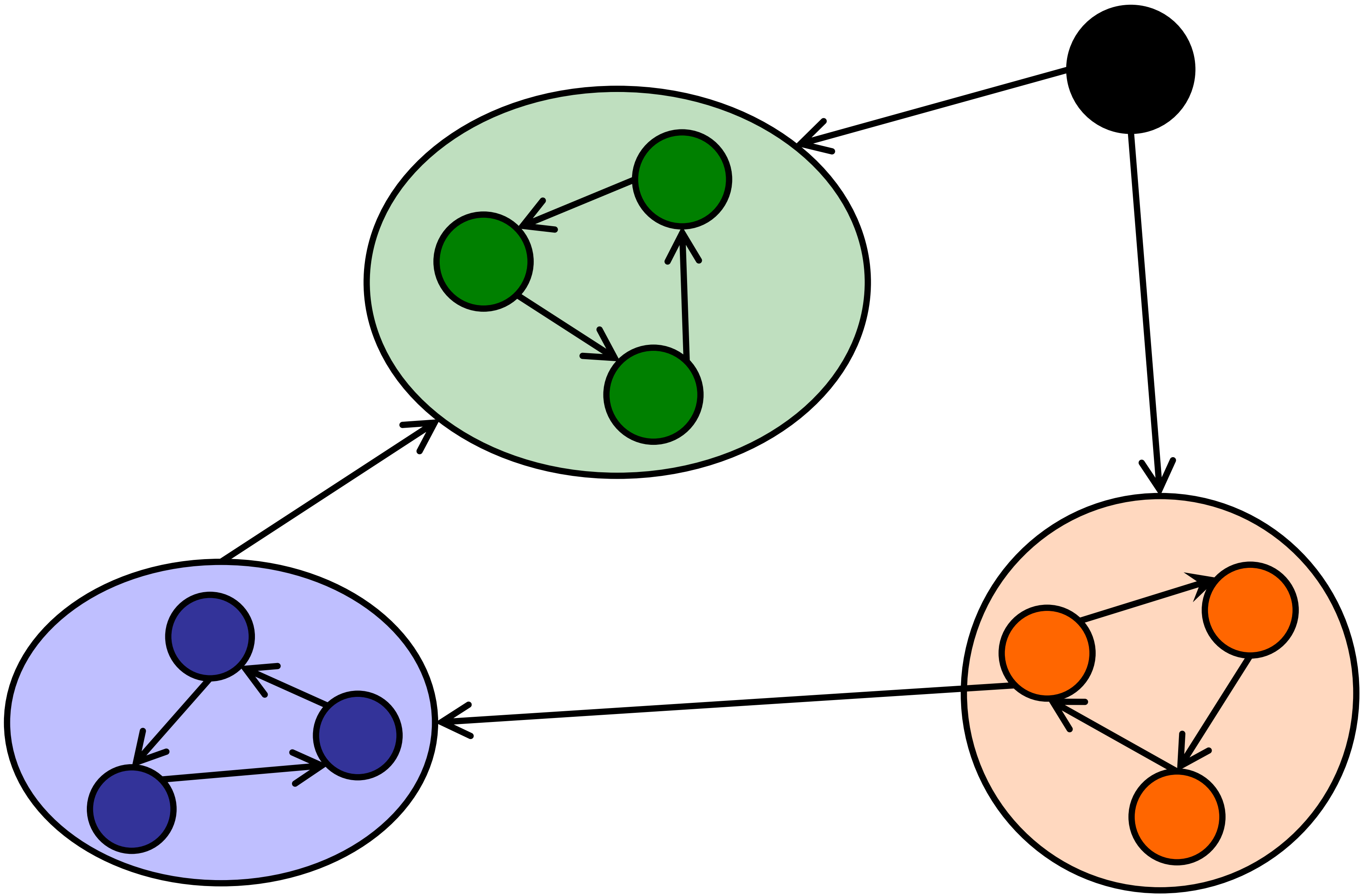
Connected Components

Graph of strongly connected components is acyclic!



Connected Components

Challenge: find all strongly connected components.

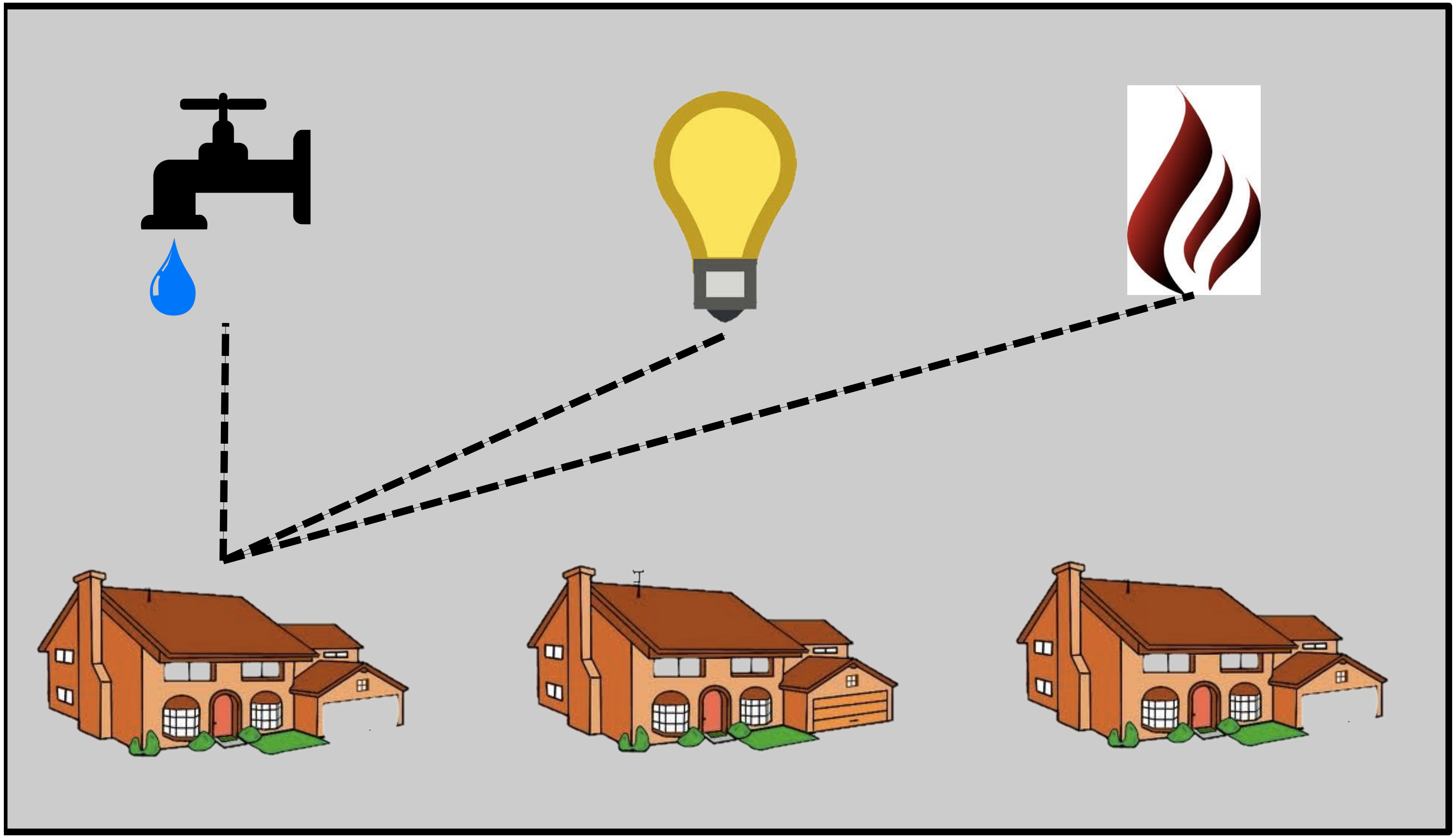


Roadmap

Directed Graphs

- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

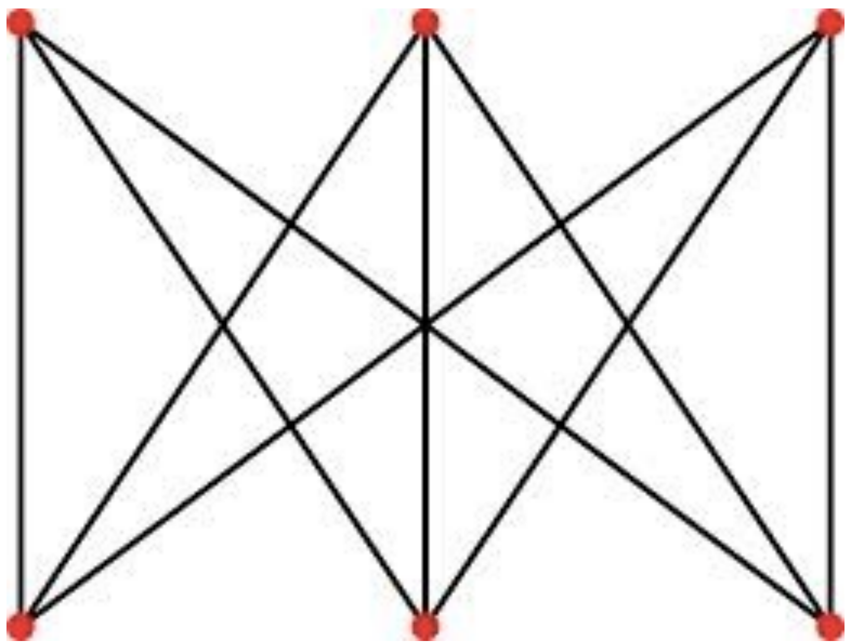
Puzzle



Connect each house to all three utilities (water, electricity, gas).
Do not let any of the cables or pipes cross.
(Or show that it is impossible.)

Puzzle Explanation

Can you draw this graph with no crossing lines?

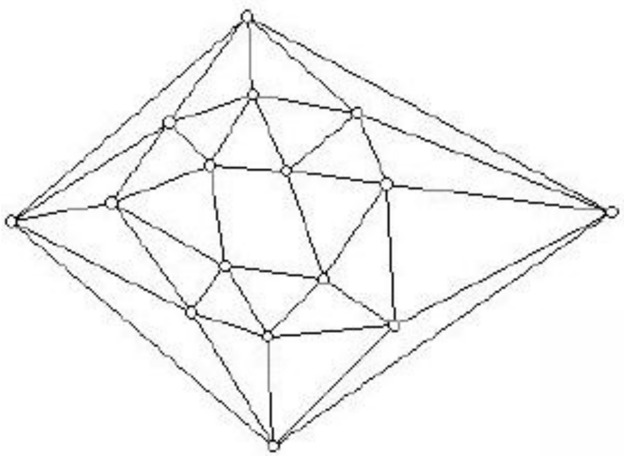


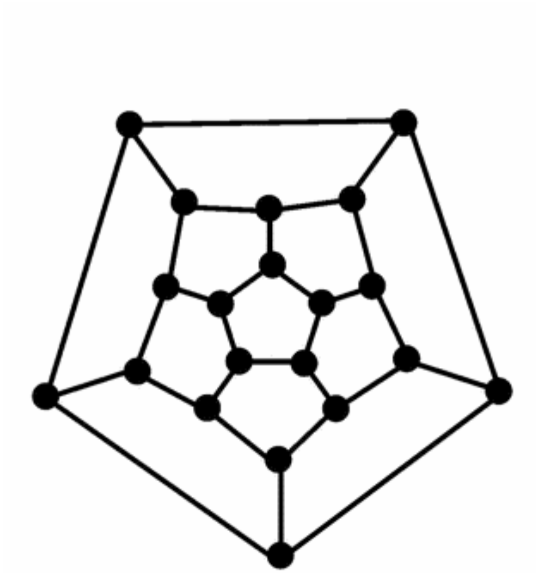
Bipartite Clique

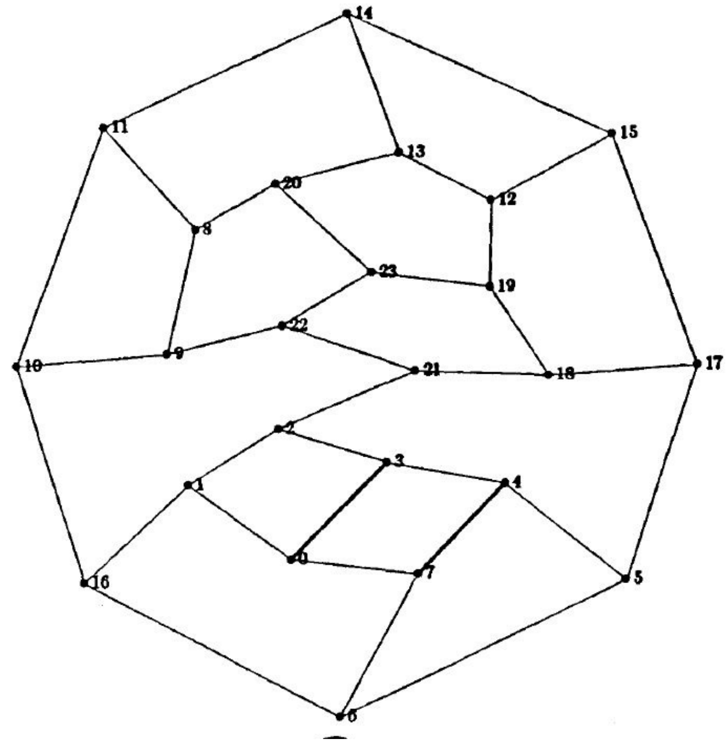
Puzzle Explanation

Planar Graph:

Any graph that can be drawn on a flat 2d piece of paper with no crossing lines.



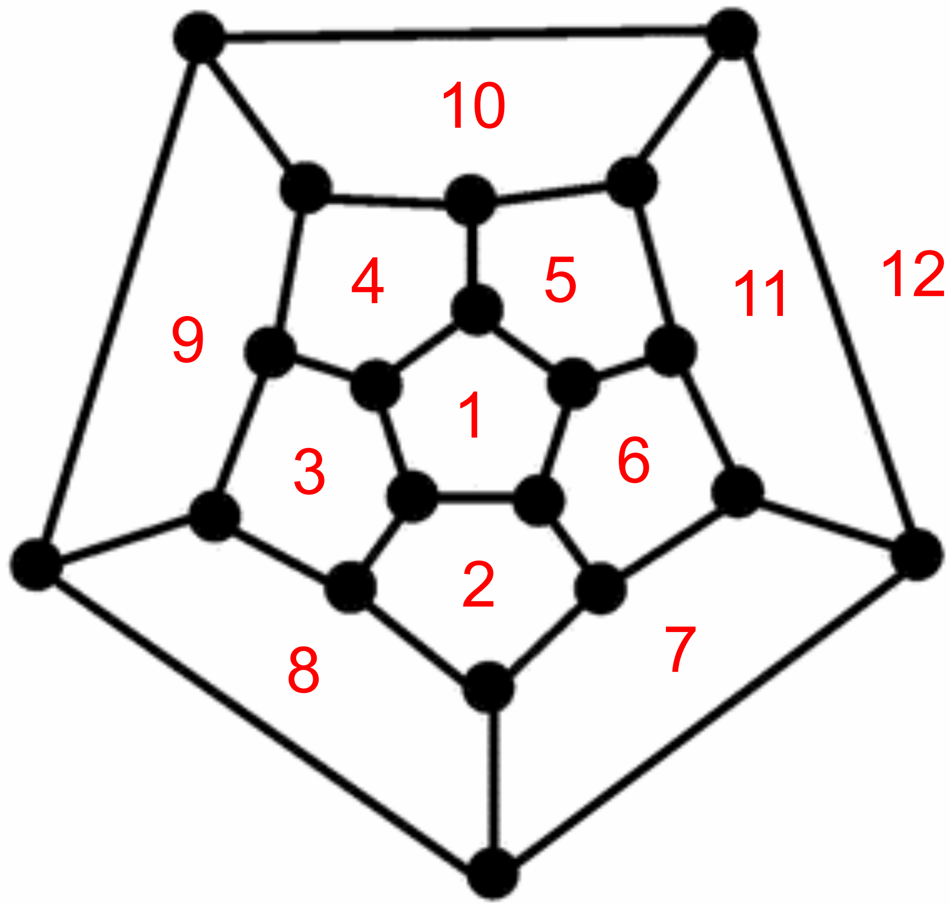




Puzzle Explanation

Terms:

- vertex
- edge
- face
  - area bounded by edges
  - outer (infinite) area

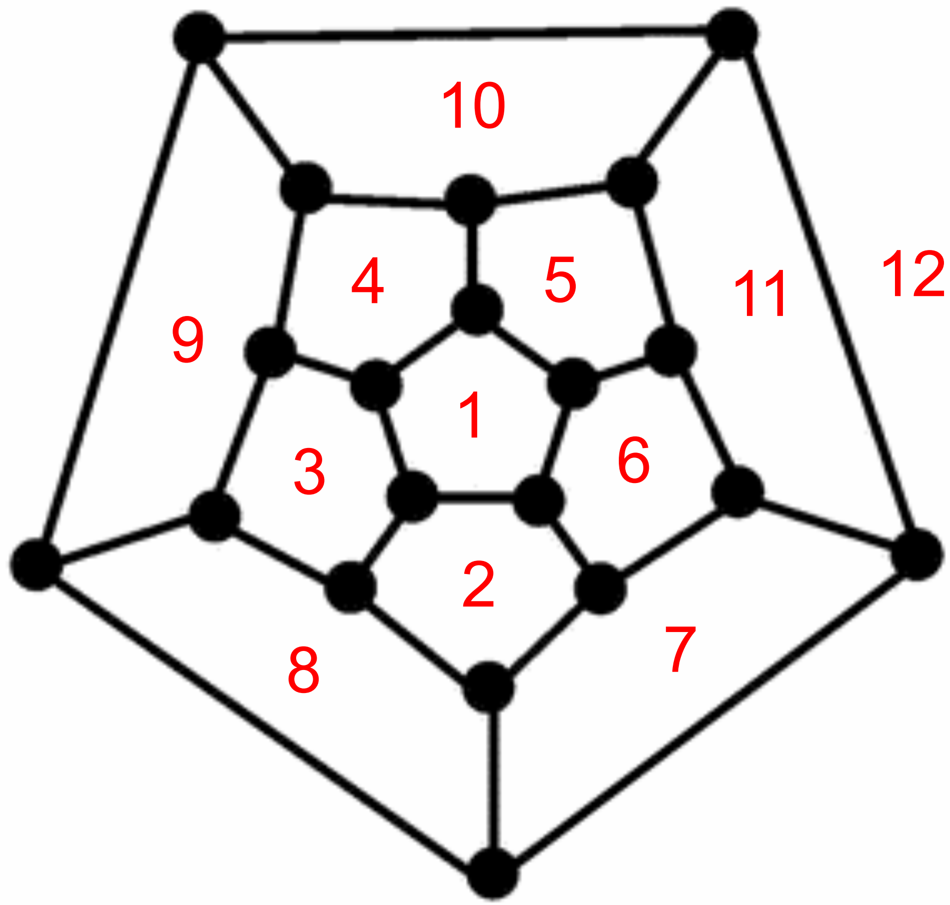


Puzzle Explanation

Euler's Formula:
(planar graphs)

$V - E + F = 2$

V = # vertices
E = # edges
F = # faces



Prove by induction.

Puzzle Explanation

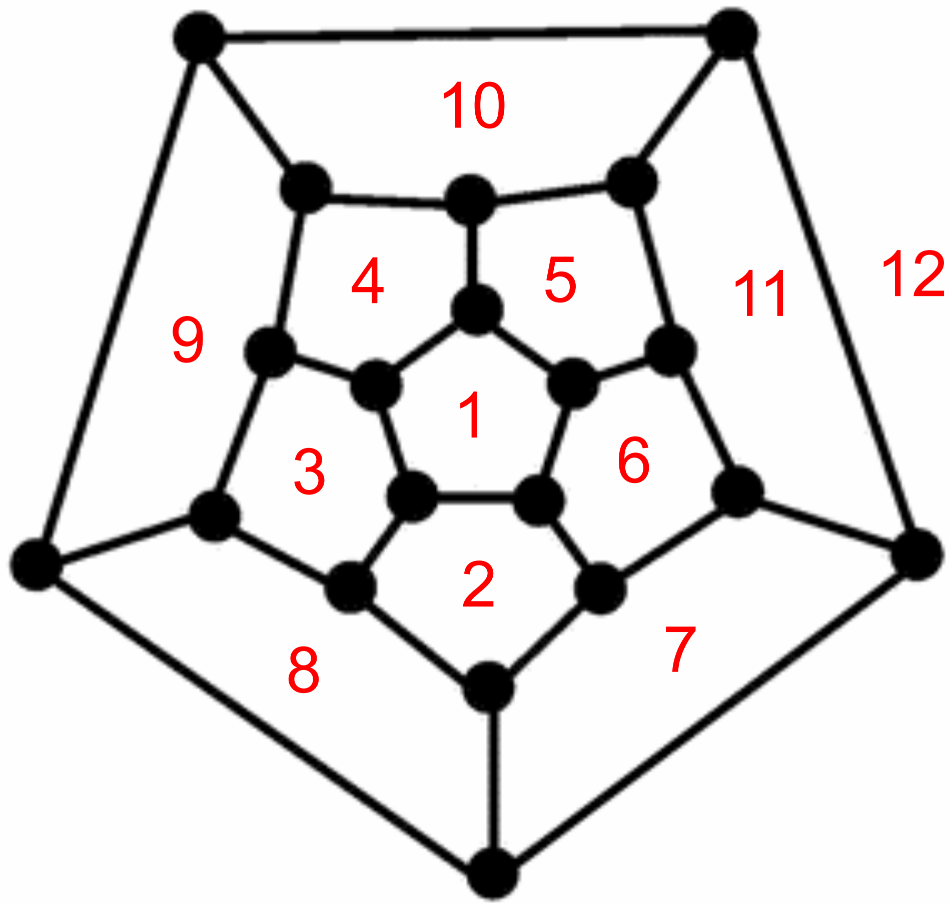
Euler's Formula:
(planar graphs)

$$V - E + F = 2$$

$V = 20$

$E = 30$

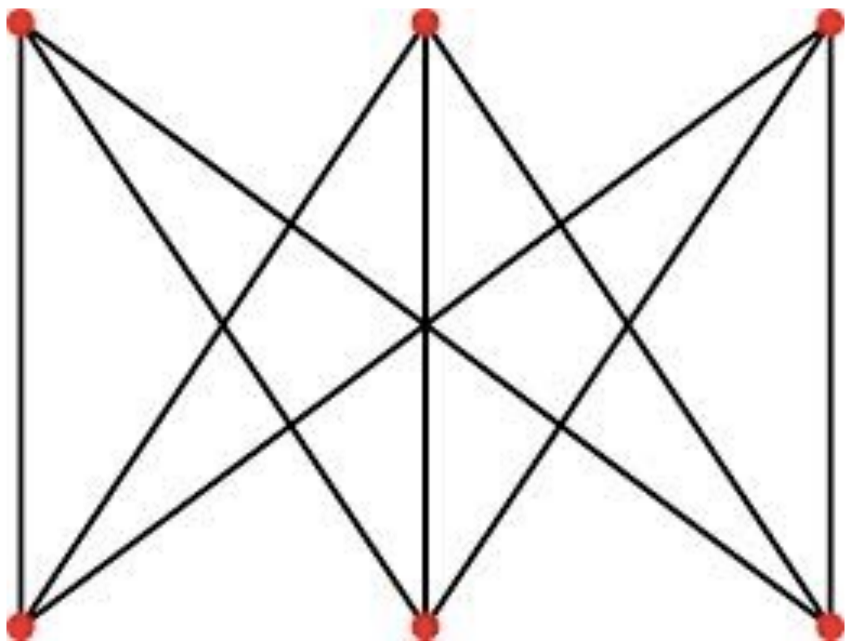
$F = 12$



$$20 - 30 + 12 = 2$$

Puzzle Explanation

Can you draw this graph with no crossing lines?

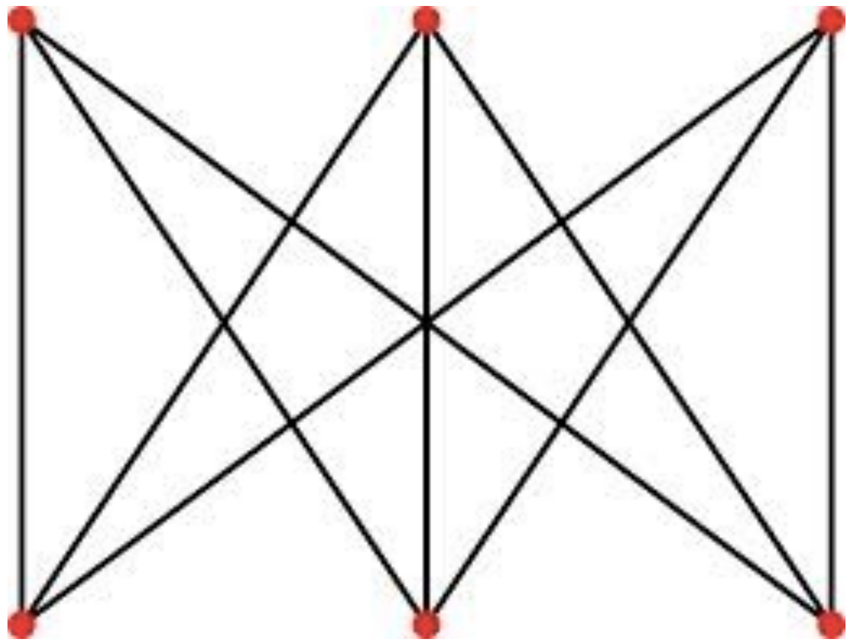


Bipartite Clique

Puzzle Explanation

Euler's Formula:
(planar graphs)

$V - E + F = 2$



$V = 6$

$E = 9$

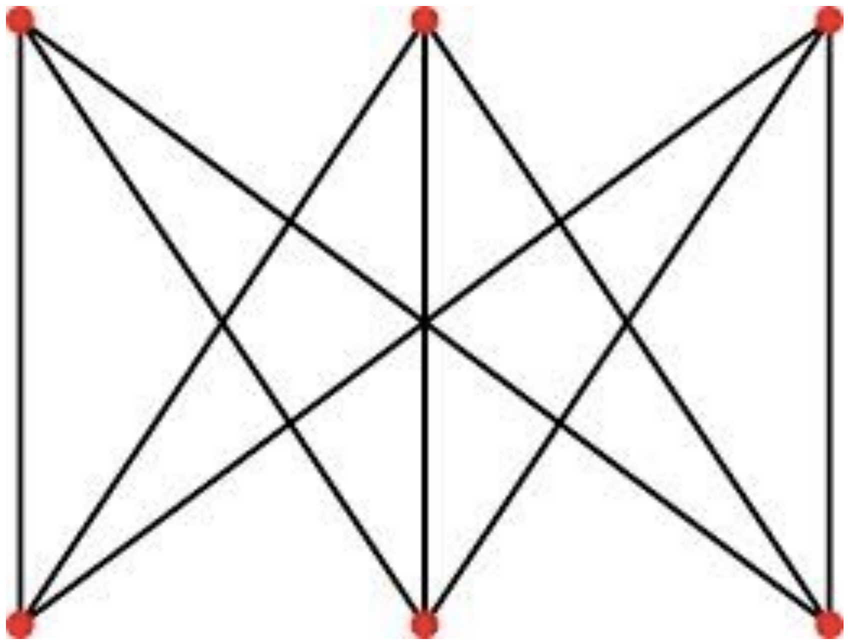
$F = ??$

$6 - 9 + F = 2$

Puzzle Explanation

Euler's Formula:
(planar graphs)

$V - E + F = 2$



$V = 6$

$E = 9$

$F = 5$

$6 - 9 + F = 2$

Puzzle Explanation

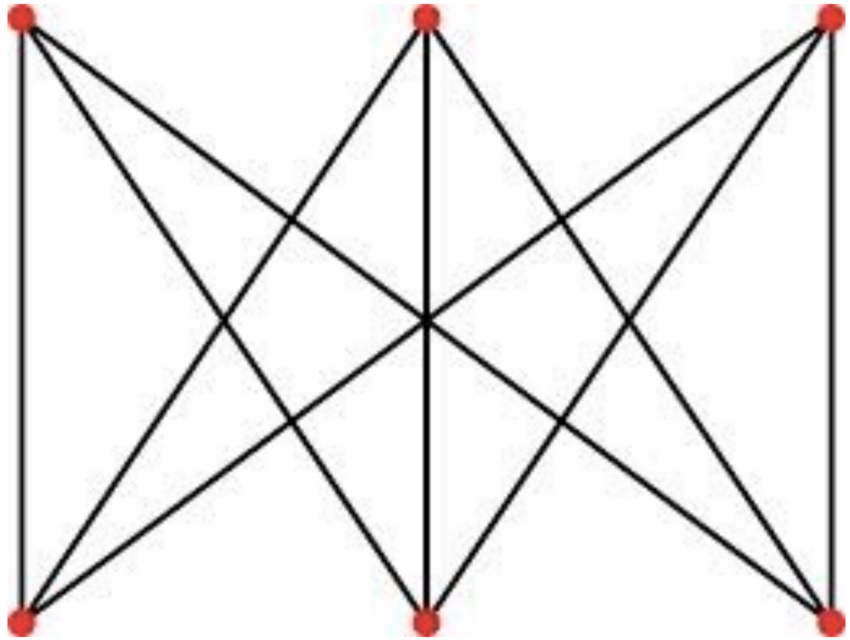
For bipartite clique:

Every face has at least 4 edges.

Every edge is used in at most 2 faces.

➔ E >= 4F/2 ➔

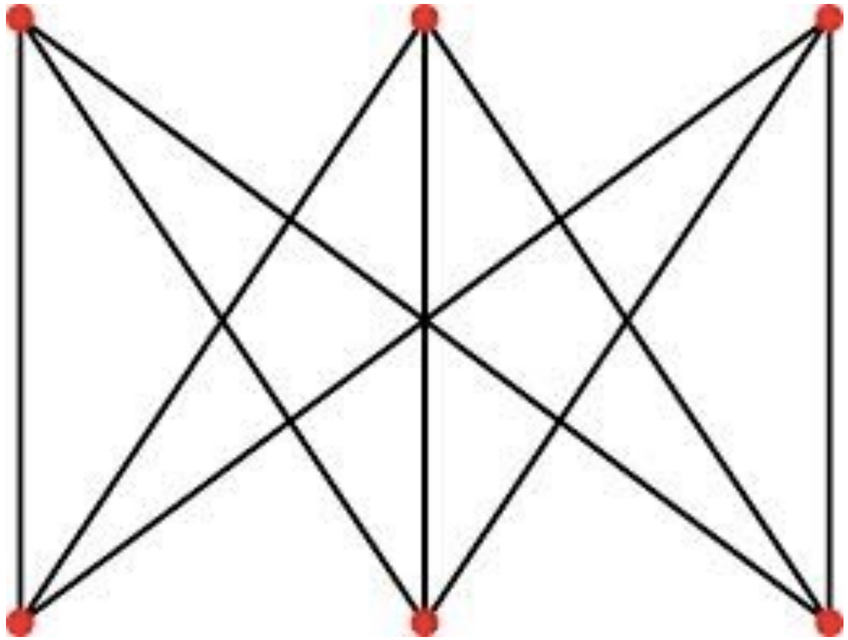
$F \leq (2E) / 4 \leq E/2$



Puzzle Explanation

Impossible!

$F \leq E/2$



$V = 6$

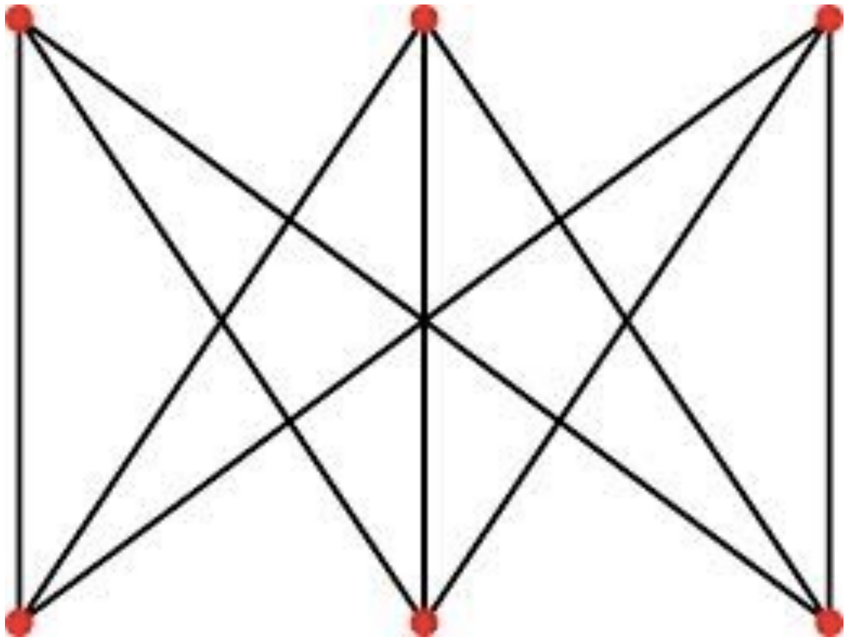
$E = 9$

$F = 5$

BUT: $5 > 9/2$

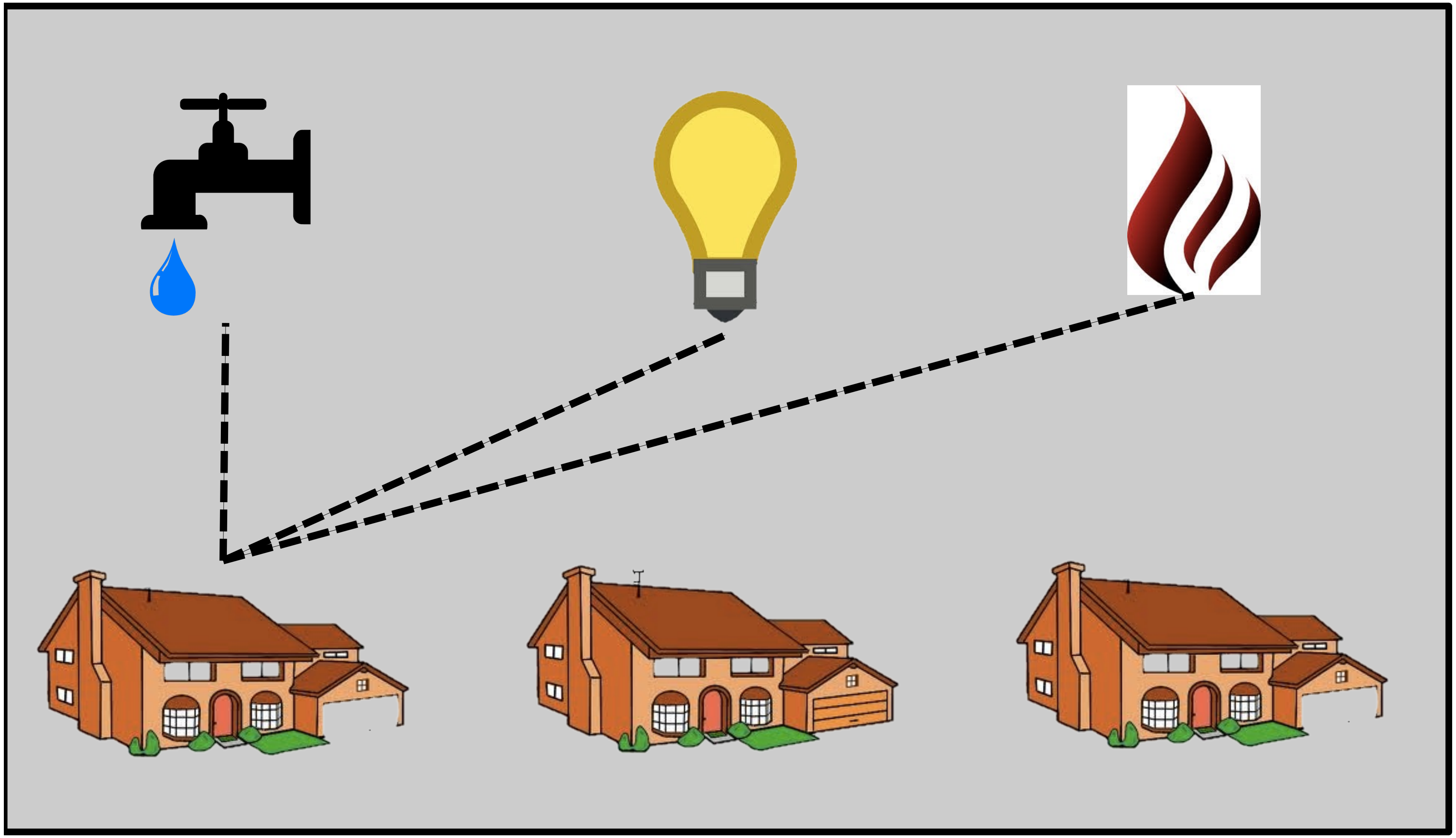
Puzzle Explanation

Impossible to draw bipartite clique without crossing lines.



Bipartite Clique

Puzzle



Connect each house to all three utilities (water, electricity, gas).
Do not let any of the cables or pipes cross.
(Or show that it is impossible.)