CS2020
Data Structures and Algorithms

Welcome!

Roadmap

Last time: Graph Basics

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

About 4-coloring for Planar Graph

- Given a planar Graph
- Can you color each vertex with four colors only provided that each neighbor has a different color?

Not CS2020 Syllabus

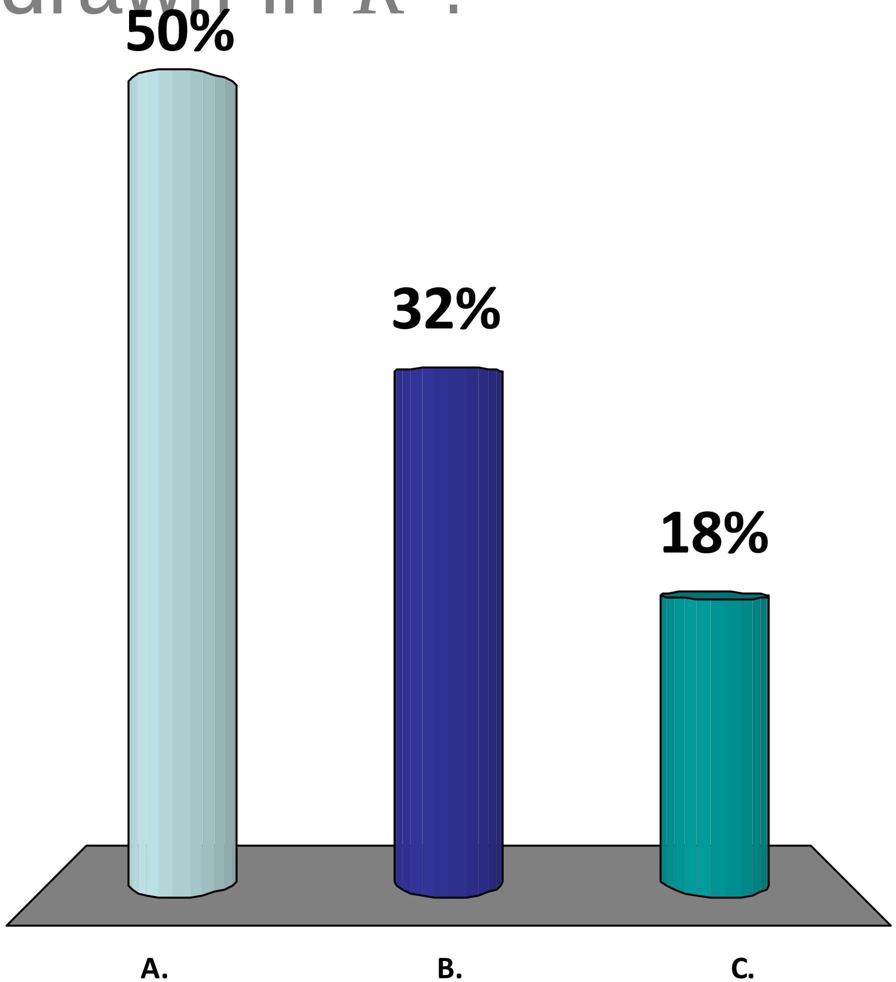
History

- 1852 when Francis Guthrie, while trying to color the map of counties of England . Conjecture appeared in a letter from Augustus De Morgan
- `Proof' by Kempe in 1879, Tait in 1880
  - Incorrectness was pointed out by Heawood in 1890
  - Petersen in 1891
- Confirmed by Appel and Haken in 1976 (*1476*)
- Again by Robertson, Sanders, Seymour and Thomas (*633*)

Not CS2020 Syllabus

Some graphs can be drawn on a plane
And some cannot
But can all graphs be drawn in R^3?

A. Yes
B. No
C. Wait… my head hurts…



Not CS2020 Syllabus

What is a hypergraph?

Graph consists of two types of elements:

- Nodes (or vertices)
  - At least one.

- Edges (or arcs)
  - Each edge connects >= 2 nodes in the graph
  - Each edge is unique.

(Not in CS2020)

About Embedding

- K-dim hypergraph

Thm. 4.1 Every k-dimensional abstract simplicial complex A has a geometric realization K in $\mathbb{R}^{2k+1}$.

PROOF. K satisfies the first condition for being a simplicial complex automatically. To prove the second condition holds, the idea is to map every vertex of A to a point on the *moment curve:* $M_d = \{(t, t^2, ..., t^d) \mid t \in \mathbb{R}\}$ with $d = 2k+1$ in this case. Because a hyperplane in $\mathbb{R}^d$ intersects M_d in at most d points, therefore, any $d+1$ of M_d are a.i. For any two simplices, $\sigma, \sigma' \in K$, the total number of vertices is at most $2k+2$ because $\dim(A) = k$ and all the vertices form a d-simplex. Hence, σ and σ' are the faces of the d-simplex. It follows that $\sigma \cap \sigma'$ is a face of the d-simplex, thus, a face of both. $\square$

Not CS2020 Syllabus

Roadmap

Last time: Graph Basics

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

Graph searching illustrations

See:

http://www.comp.nus.edu.sg/~stevenha/visualization/dfsbfs.html

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.

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.

Graph Search

Depth-first search:

Same algorithm, implemented with a stack:

Add start-node to stack.

Repeat until stack is empty:

- Pop node v from the front of the stack.
- Visit v.
- Explore all outgoing edges of v.
- Push all unvisited neighbors of v on the front of the stack.

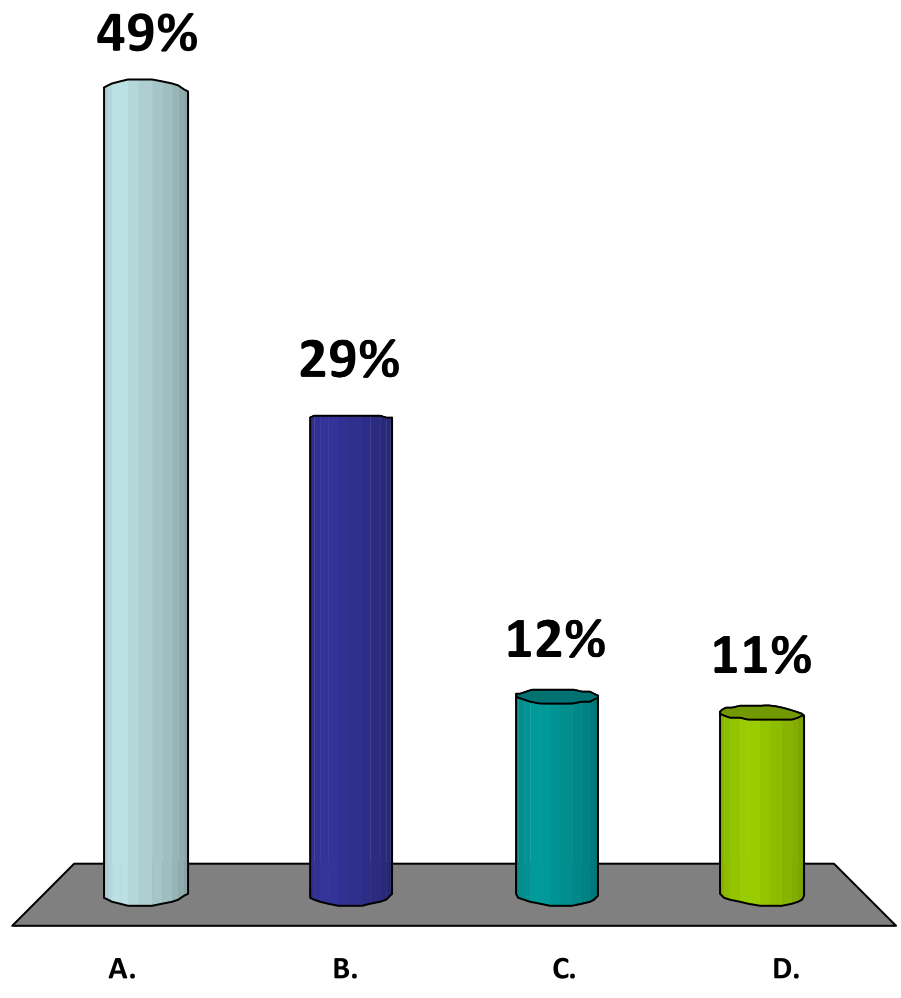
Review: Searching Graphs

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.

What do BFS and DFS solve? (Multiple answers)

A. They visit every node in the graph?
B. They visit every edge in the graph?
C. They visit every path in the graph?
D. They don't visit anything!



Common Mistake

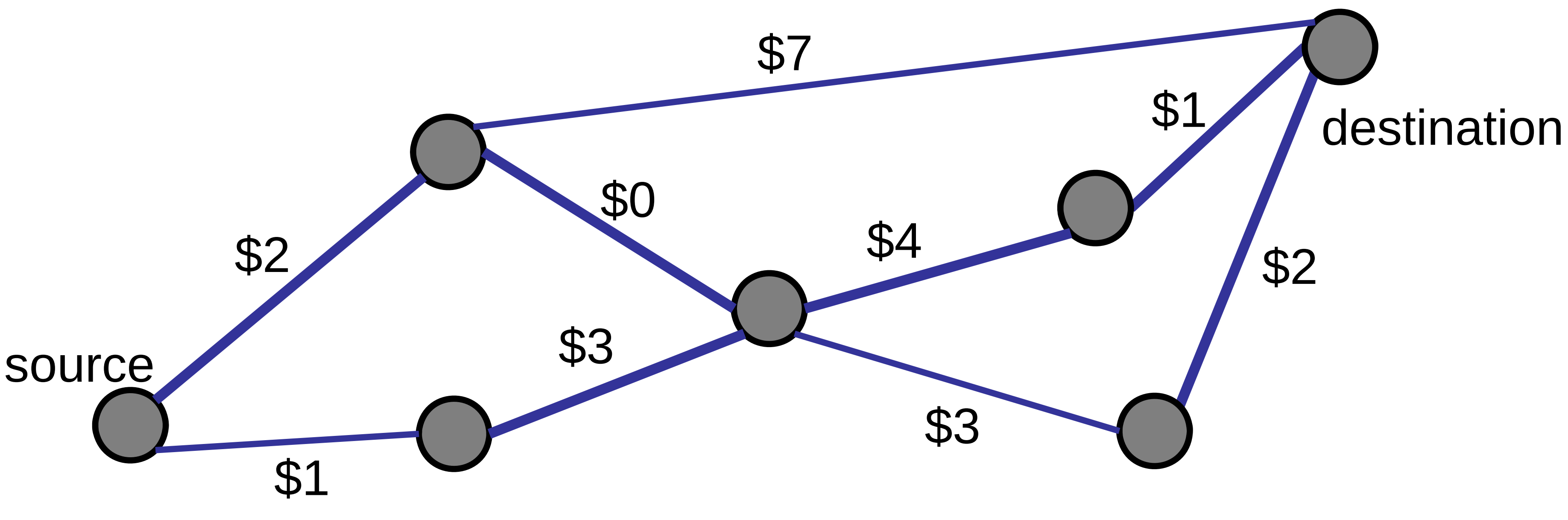
What do BFS and DFS solve?

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

Example: A Typical Graph Problem

Problem: Make Money

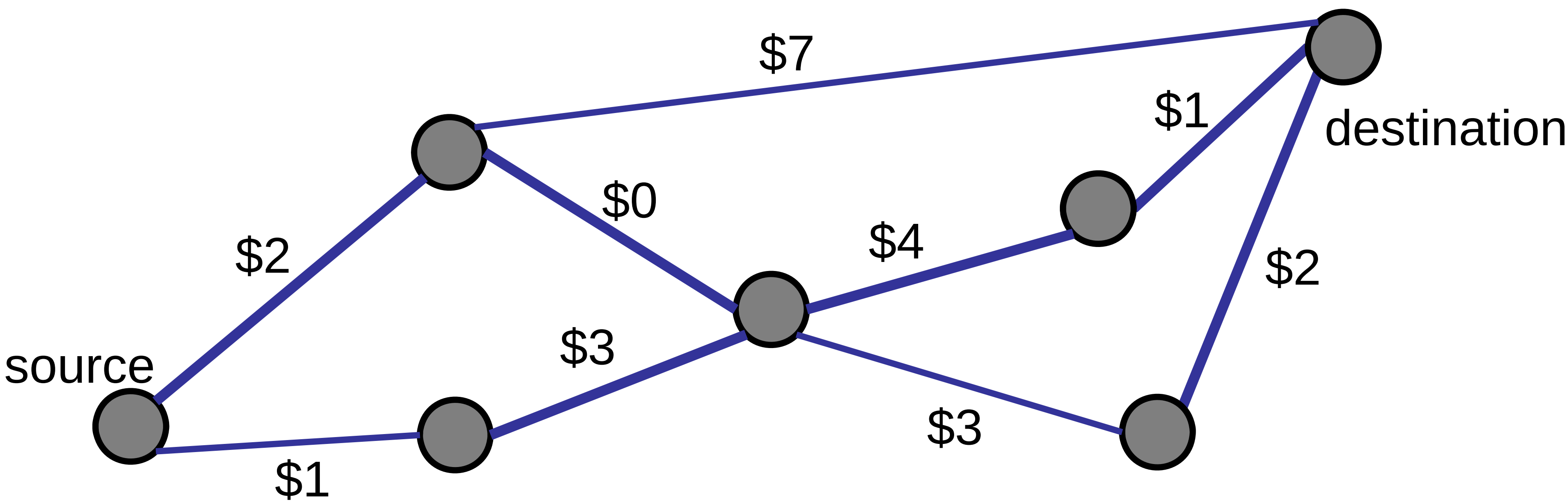
- Start at source s.
- Go to destination d.
- Each edge e earns money m(e).
- Find the path that makes the most money.



Example

NOT a solution:

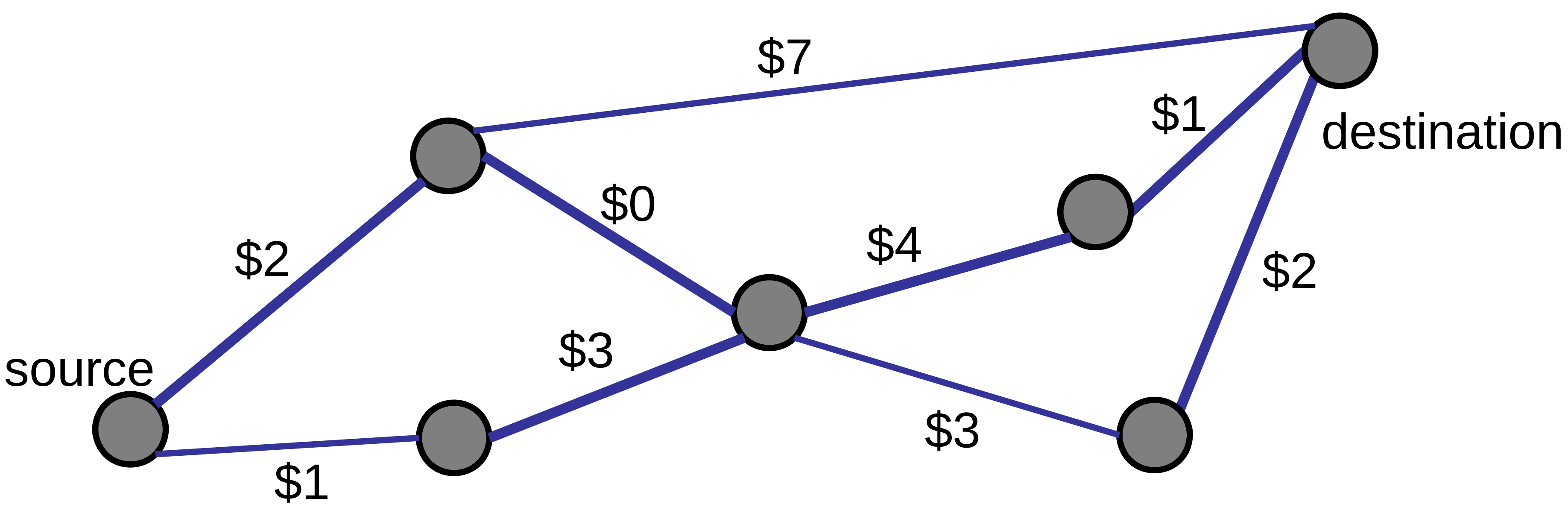
- Start at source s.
- Run BFS or DFS to explore every path.
- Keep track of the best path.



Example

Problem 1: Does not work.

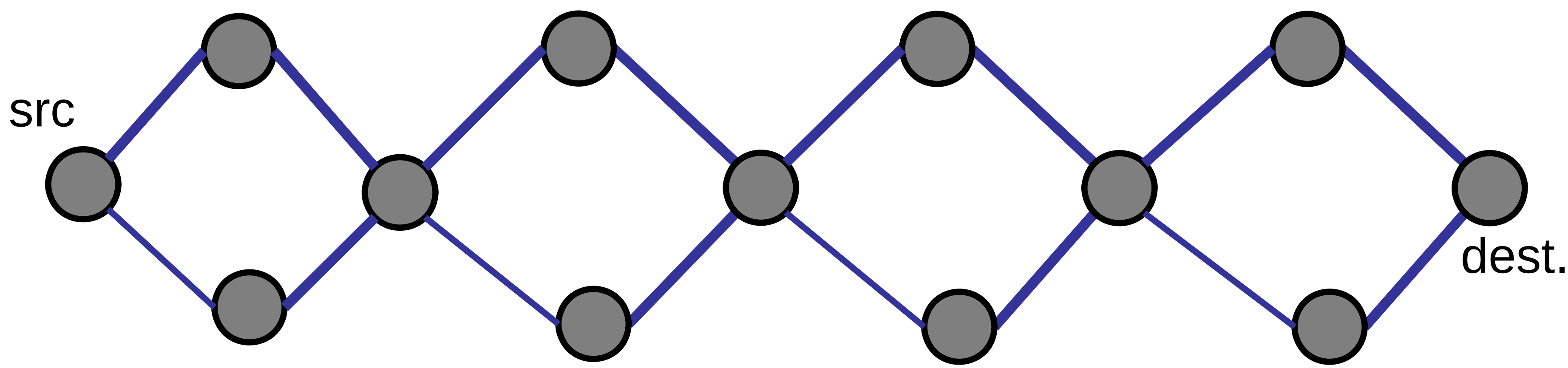
- DFS or BFS do NOT explore every path.
- Once a node is visited, it is never explored again.



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?~~

Roadmap

Part I: Directed Graphs

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

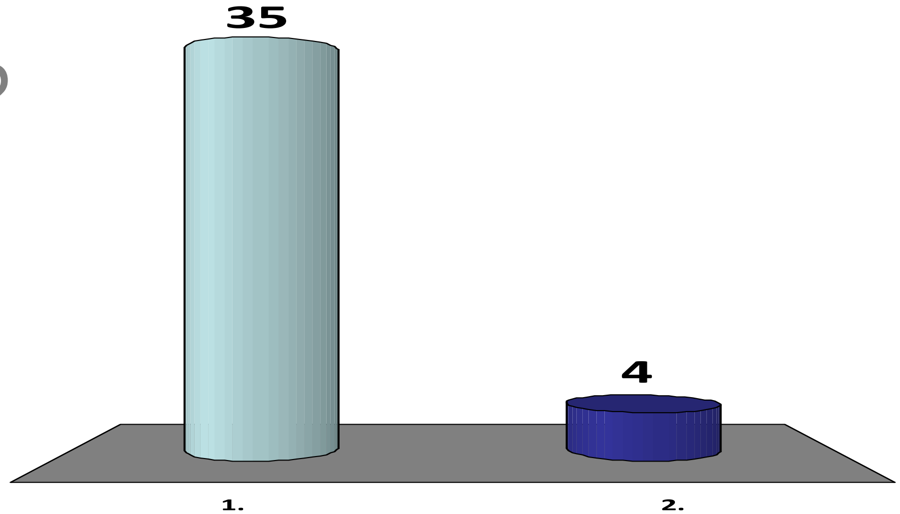
Part II: Shortest Paths

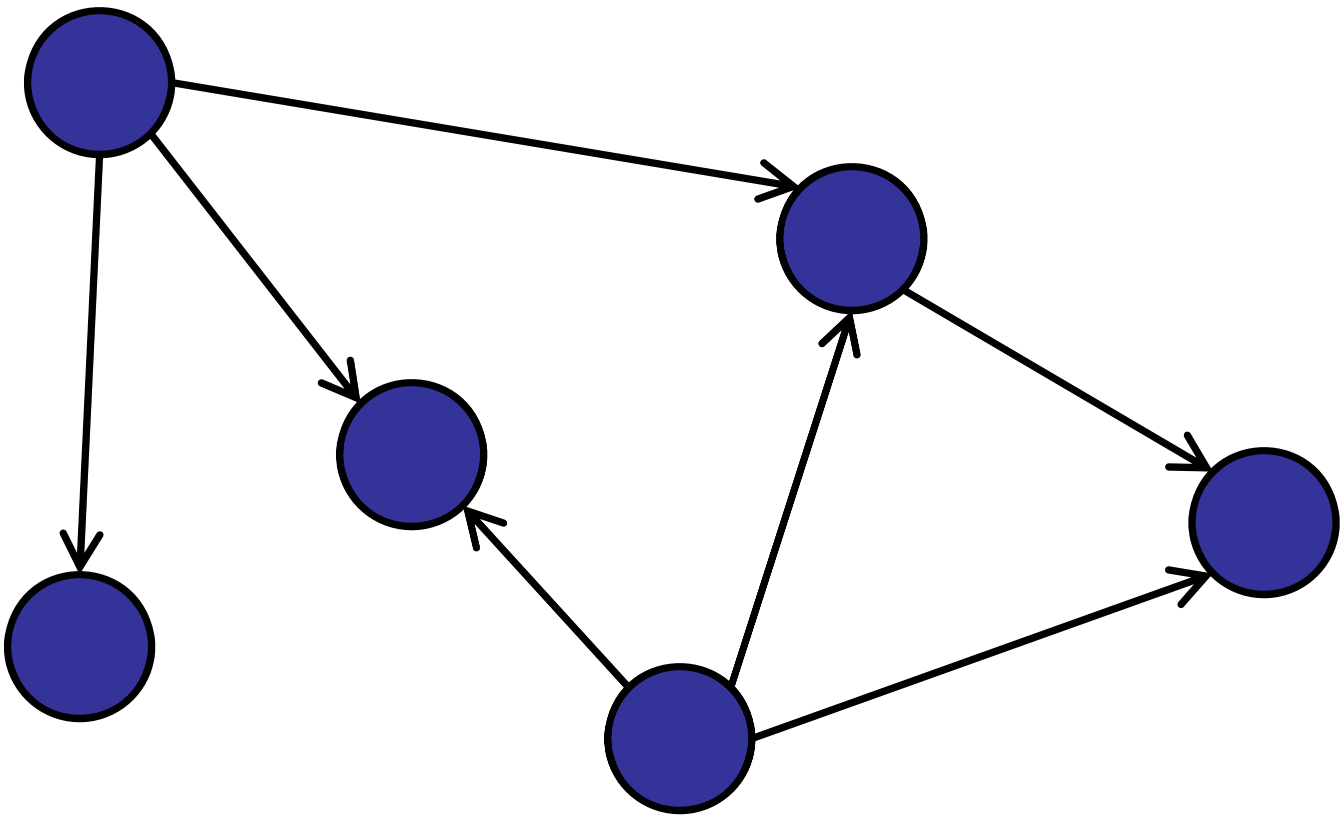
- The SSSP Problem
- Bellman-Ford

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

Is it a directed graph?

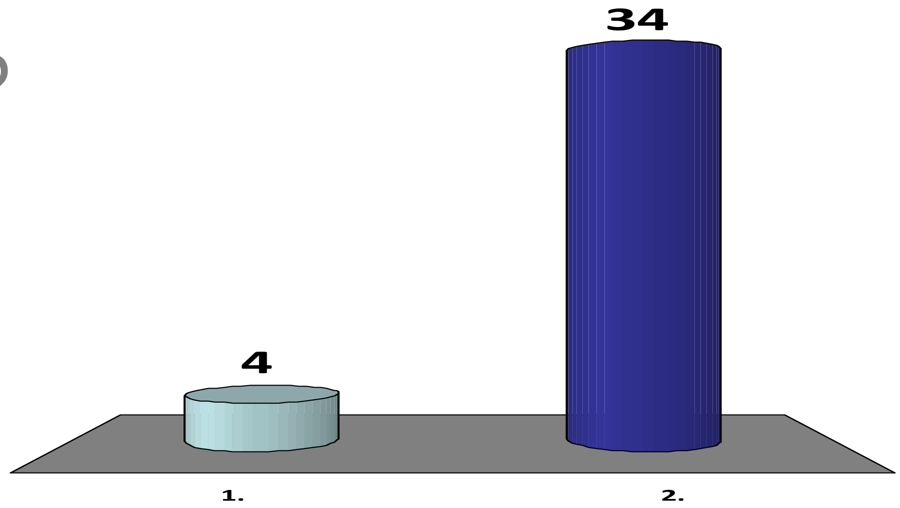
1. Yes
2. No.

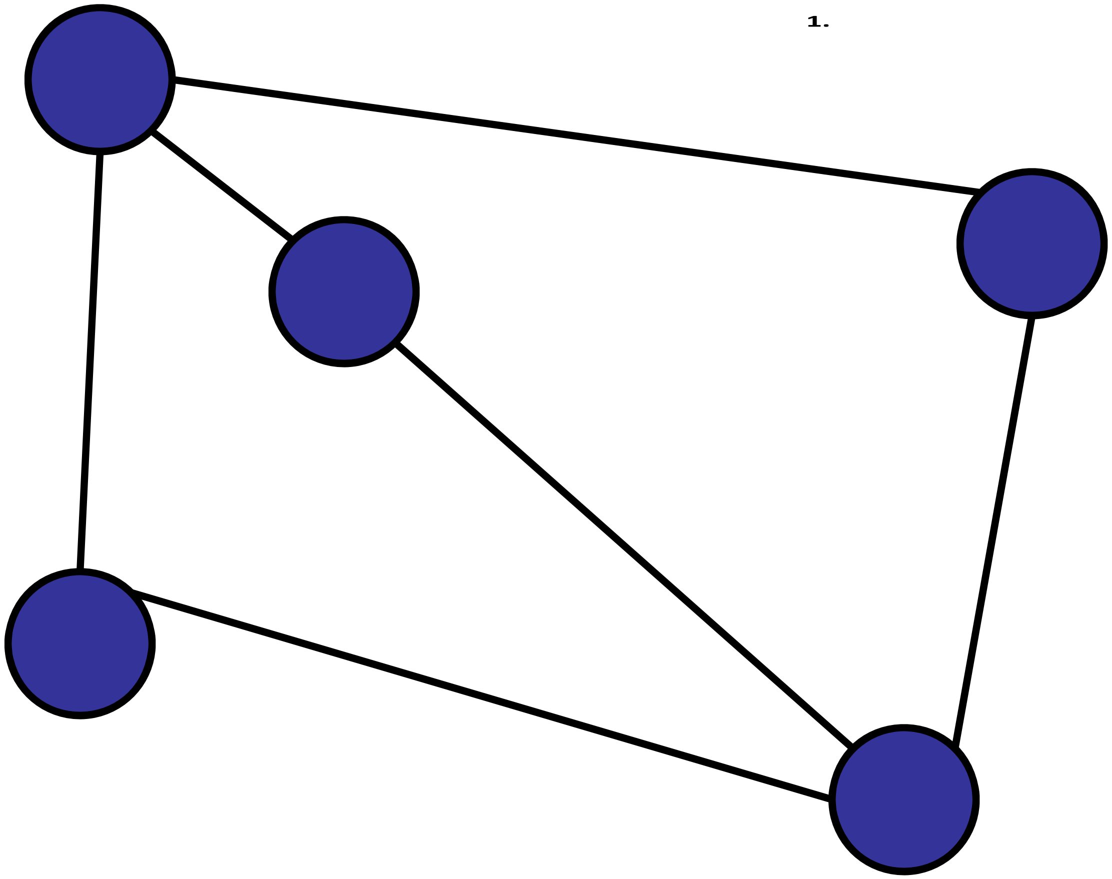




Is it a directed graph?

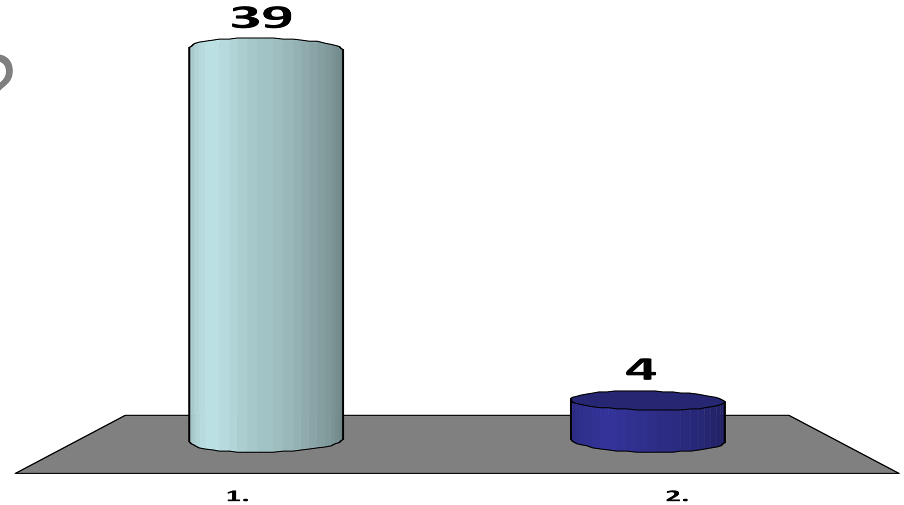
1. Yes
2. No.

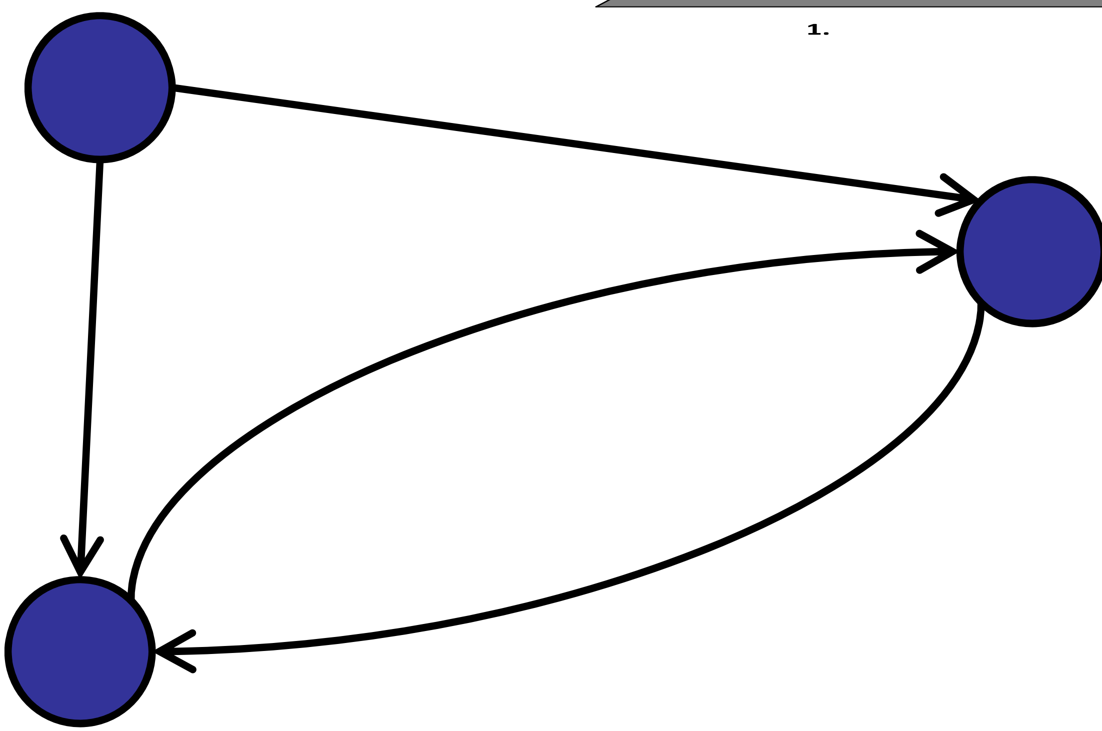




Is it a directed graph?

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>

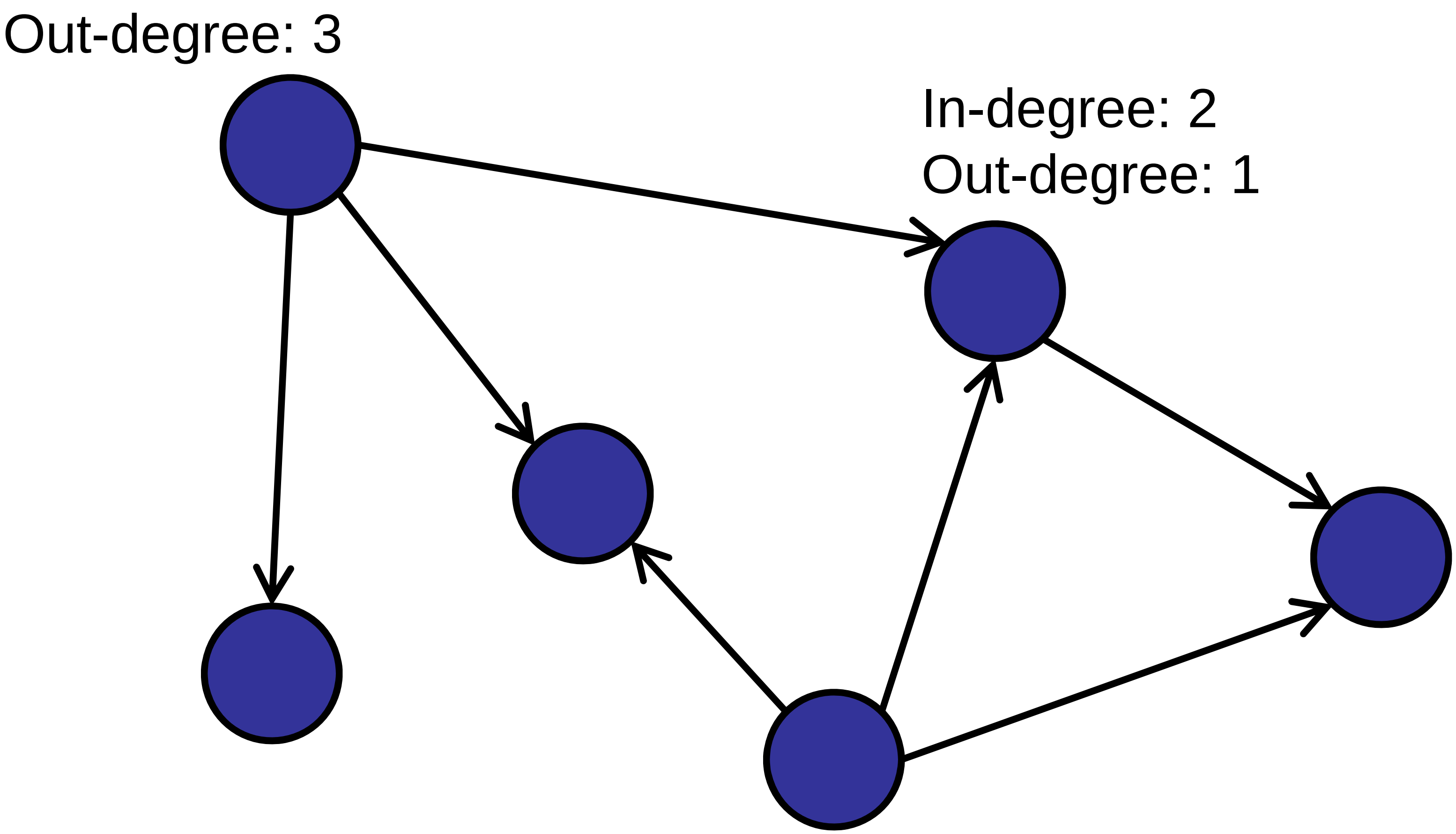
- V is a set of nodes
  - At least one: $|V| > 0$.
- E is a set of edges:
  - $E \subseteq \{ (v,w) : (v \in V), (w \in V) \}$
  - $e = (v,w)$
  - For all $e_1, e_2 \in E : e_1 \neq e_2$

Order matters!

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: $O(V + E)$

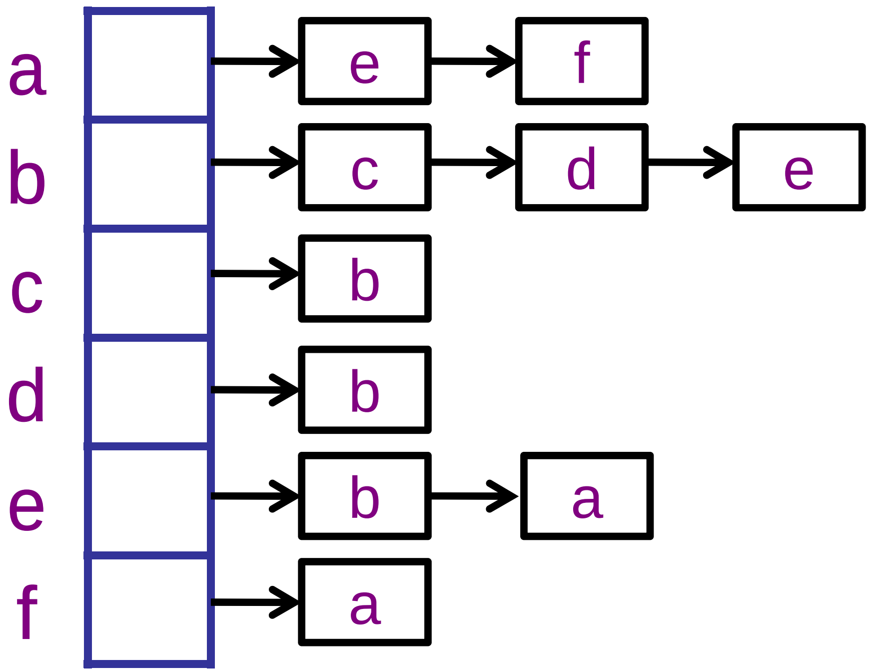
Adjacency Matrix:

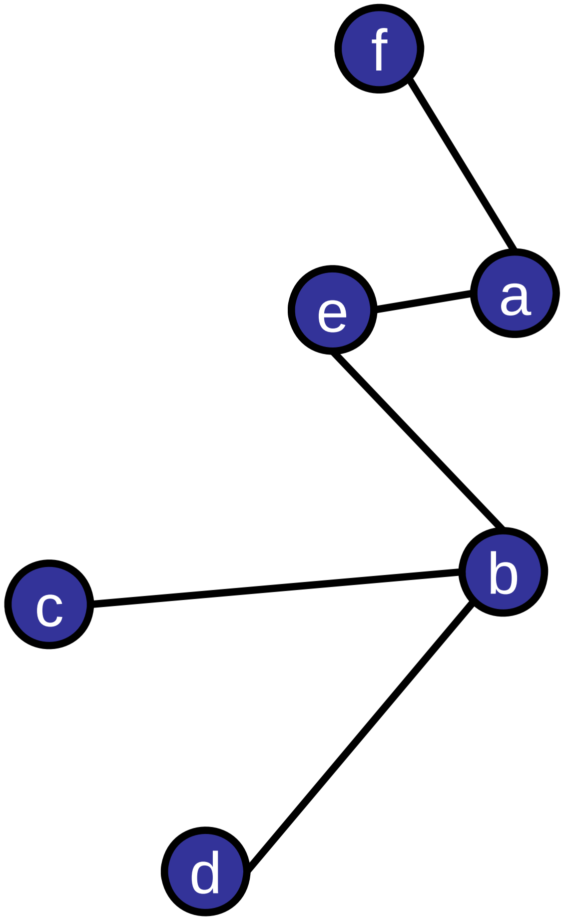
- Matrix $A[v,w]$ represents edge (v,w)
- Space: $O(V^2)$

Adjacency List

Graph consists of:

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

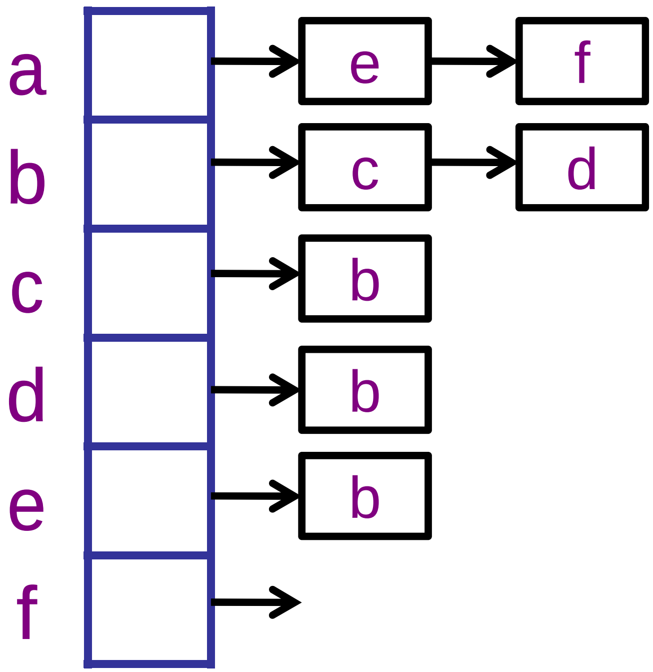


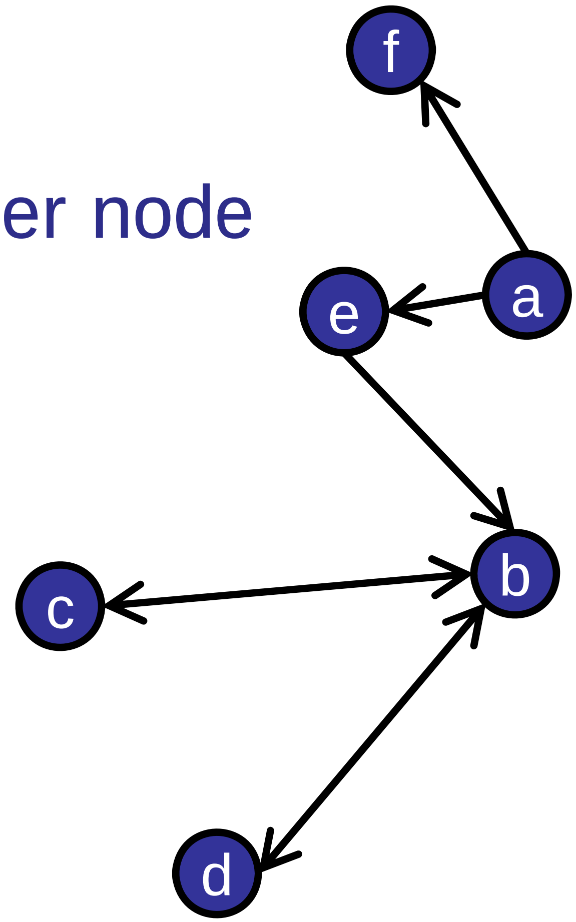


Adjacency List

Directed Graph consists of:

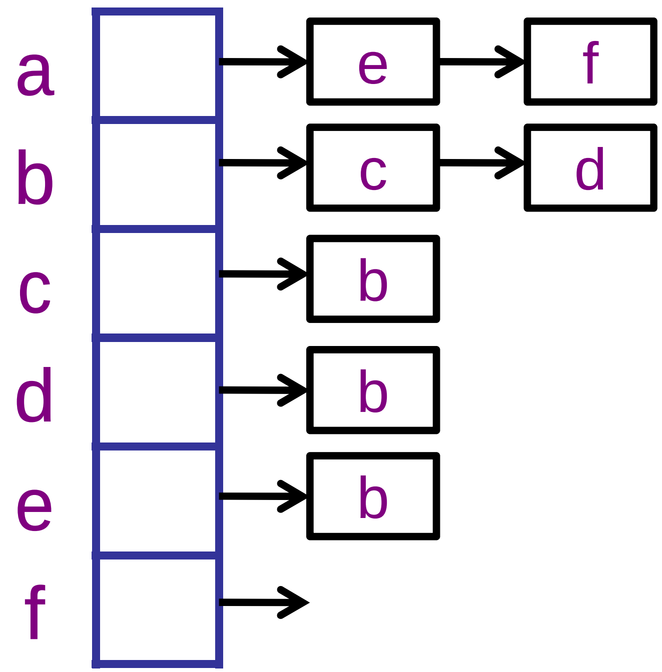
- Nodes: stored in an array
- **Outgoing** Edges: linked list per node





Adjacency List in Java

```
class NeighborList extends ArrayList<Integer> {
}

class Node {
  int key;
  NeighborList nbrs;
}

class Graph {
  Node[] nodeList;


}
```



Representing a (Directed) Graph

Adjacency List:

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

Adjacency Matrix:

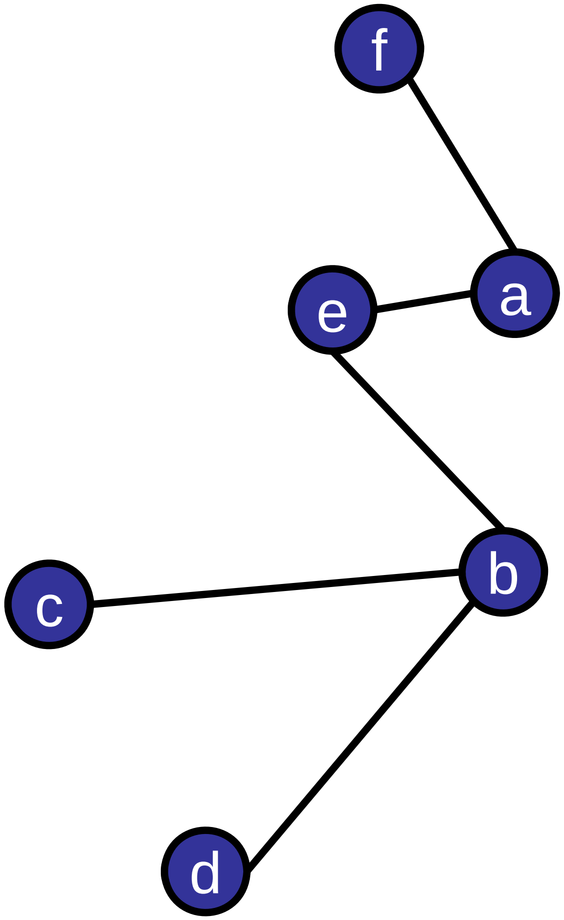
- Matrix A[v,w] represents edge (v,w)
- Space: $O(V^2)$

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

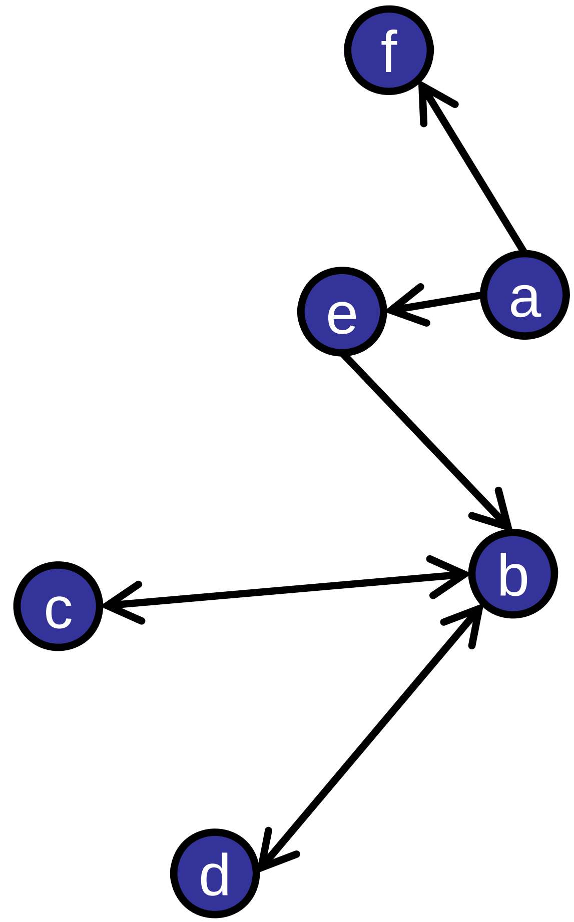


Adjacency Matrix

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:

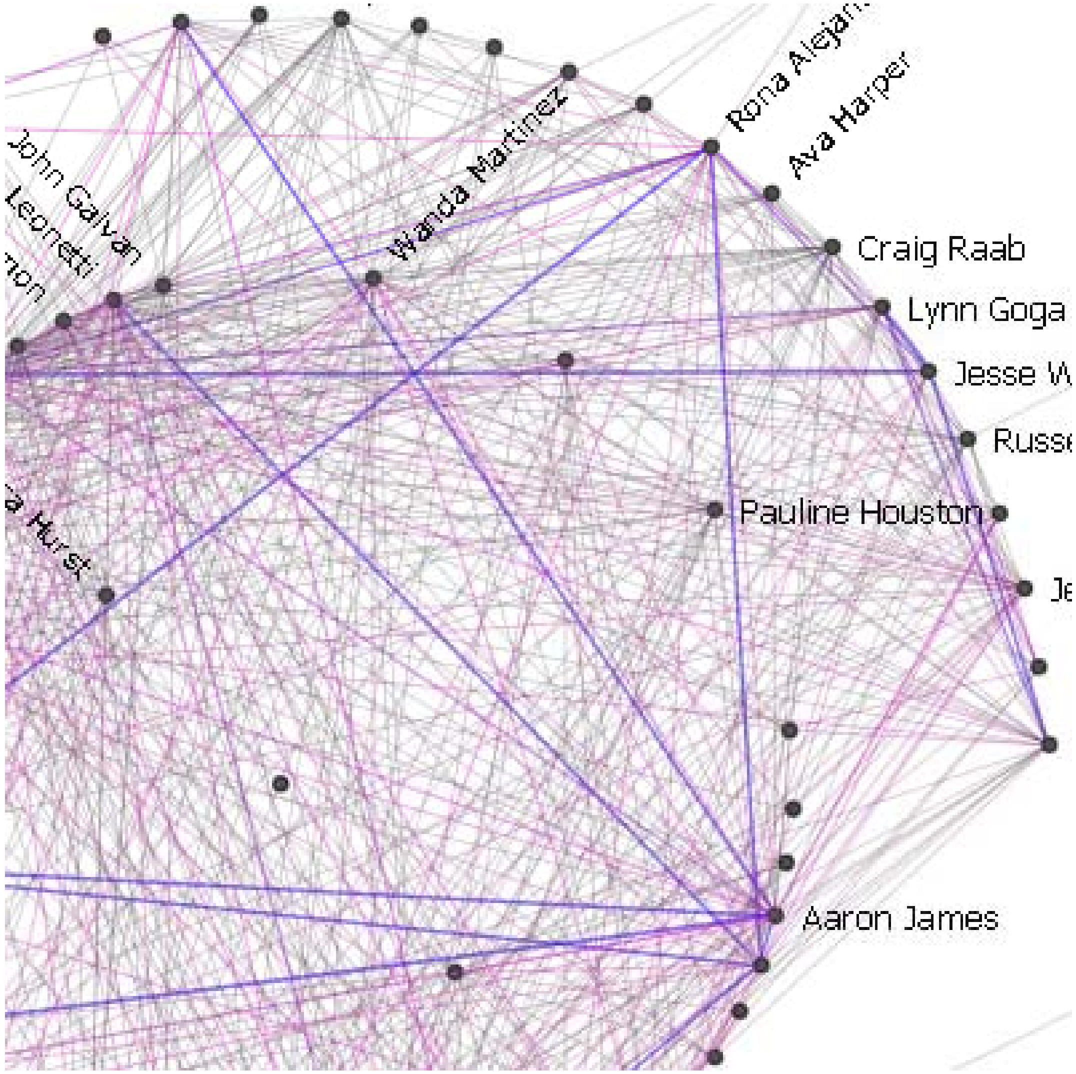
- Search recursively
- Follow outgoing edges
- Backtrack (through incoming edges)

Example of directed graphs

Directed Graphs

Is friendship always bidirectional?:

- Nodes are people
- Edge = friendship



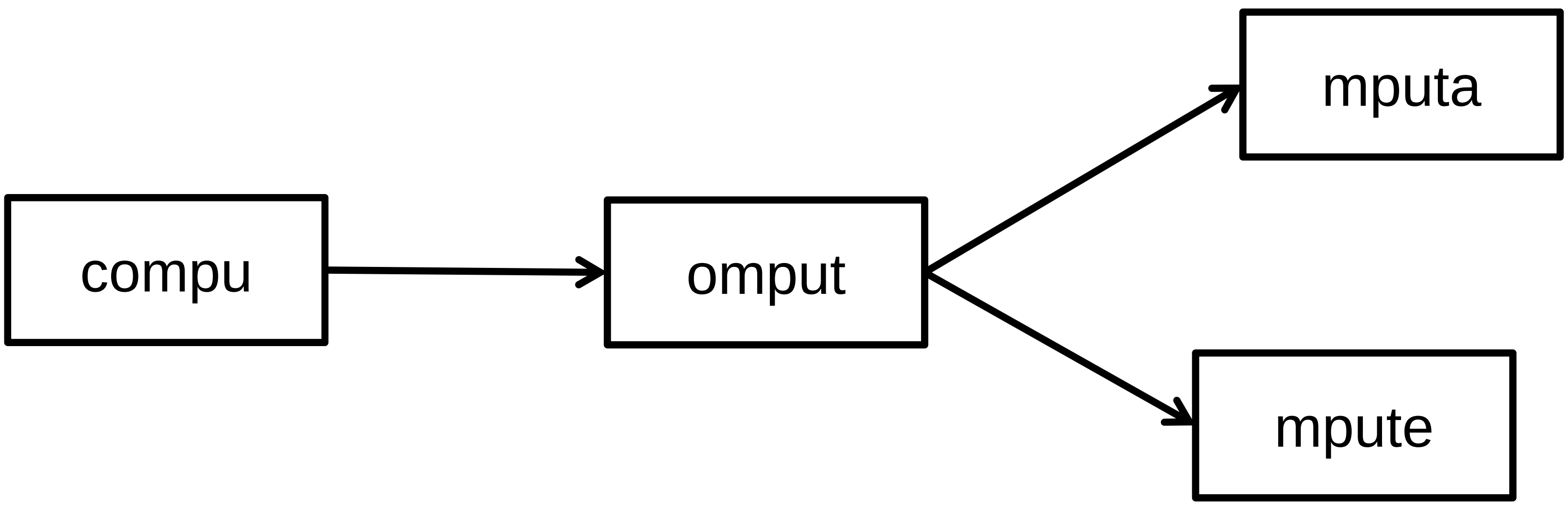
Facebook: yes

Google+: no

Directed Graphs

Markov text generation:

- Nodes are kgrams
  - A k-gram is a contiguous sequence of k items e.g. syllables, letters, words, etc.
- Edge = one kgram follows another



SCIgen - An Automatic CS Paper Generator

About Generate Examples Talks Code Donations Related People Blog

About

SCIgen is a program that generates random Computer Science research papers, including graphs, figures, and citations. It uses a hand-written **context-free grammar** to form all elements of the papers. Our aim here is to maximize amusement, rather than coherence.

One useful purpose for such a program is to auto-generate submissions to conferences that you suspect might have very low submission standards. A prime example, which you may recognize from spam in your inbox, is SCI/IIIS and its dozens of co-located conferences (check out the very broad conference description on the **WMSCI 2005** website). There's also a list of **known bogus conferences**. Using SCIgen to generate submissions for conferences like this gives us pleasure to no end. In fact, one of our papers was accepted to SCI 2005! See **Examples** for more details.

We went to WMSCI 2005. Check out the **talks and video**. You can find more details in our **blog**.

Also, check out our 10th anniversary celebration project: **SCIpher**!

https://pdos.csail.mit.edu/archive/scigen/

A conference accepted it!

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

Abstract

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. Introduction

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-tauted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in $\Omega((n + \log n))$ time [22]. In the end, we conclude.

II. Architecture

Our research is principled. Consider the early methodology by Martin and Smith: our model is similar, but will actually

Scheduling

Set of tasks for baking cookies:

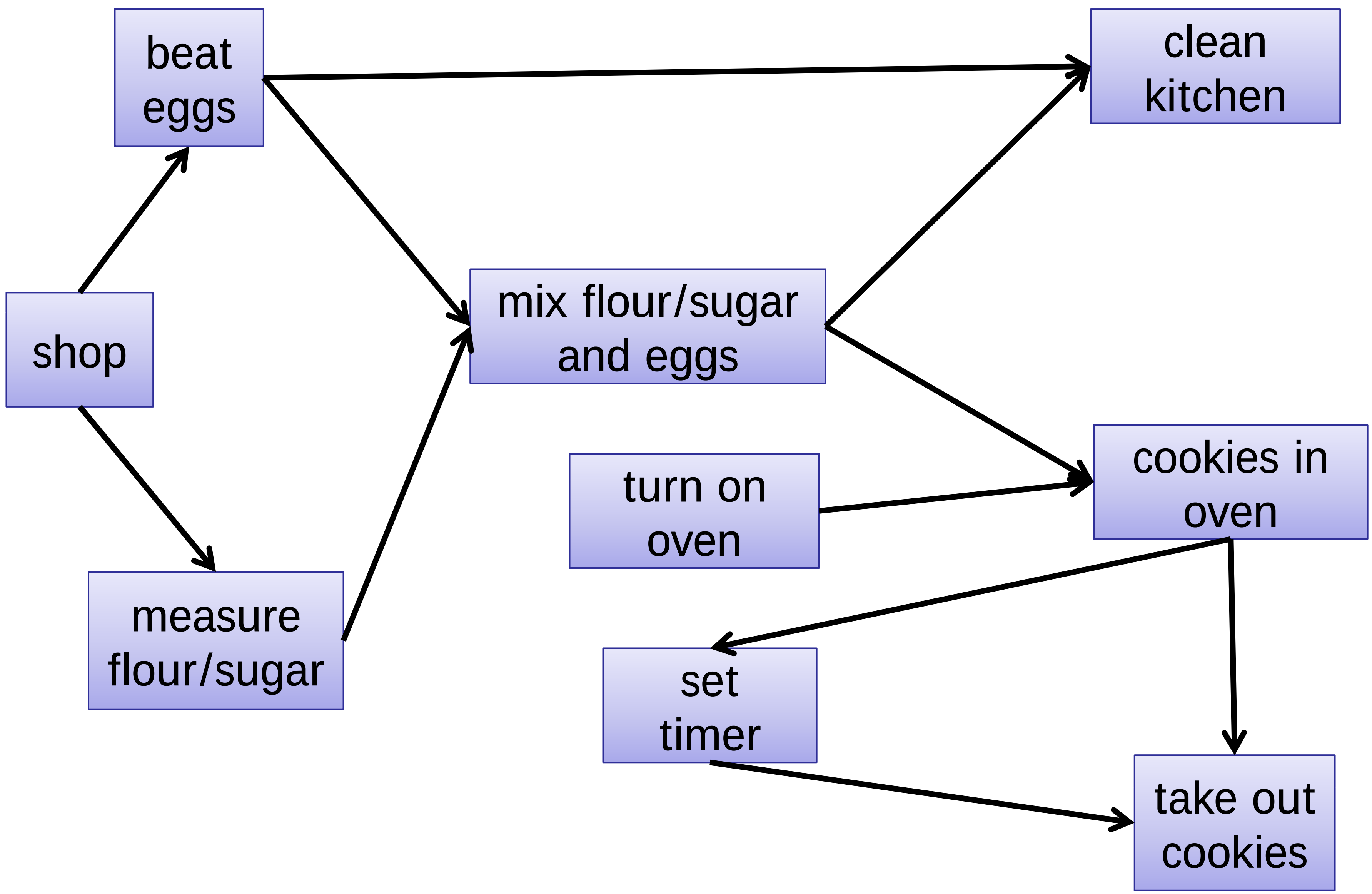
- Shop for groceries
- Put the cookies in the oven
- Clean the kitchen
- Beat the eggs in a bowl
- Measure the flour and sugar in a bowl
- Mix the eggs with the flour and sugar
- Turn on the oven
- Set the timer
- Take out the cookies

Scheduling

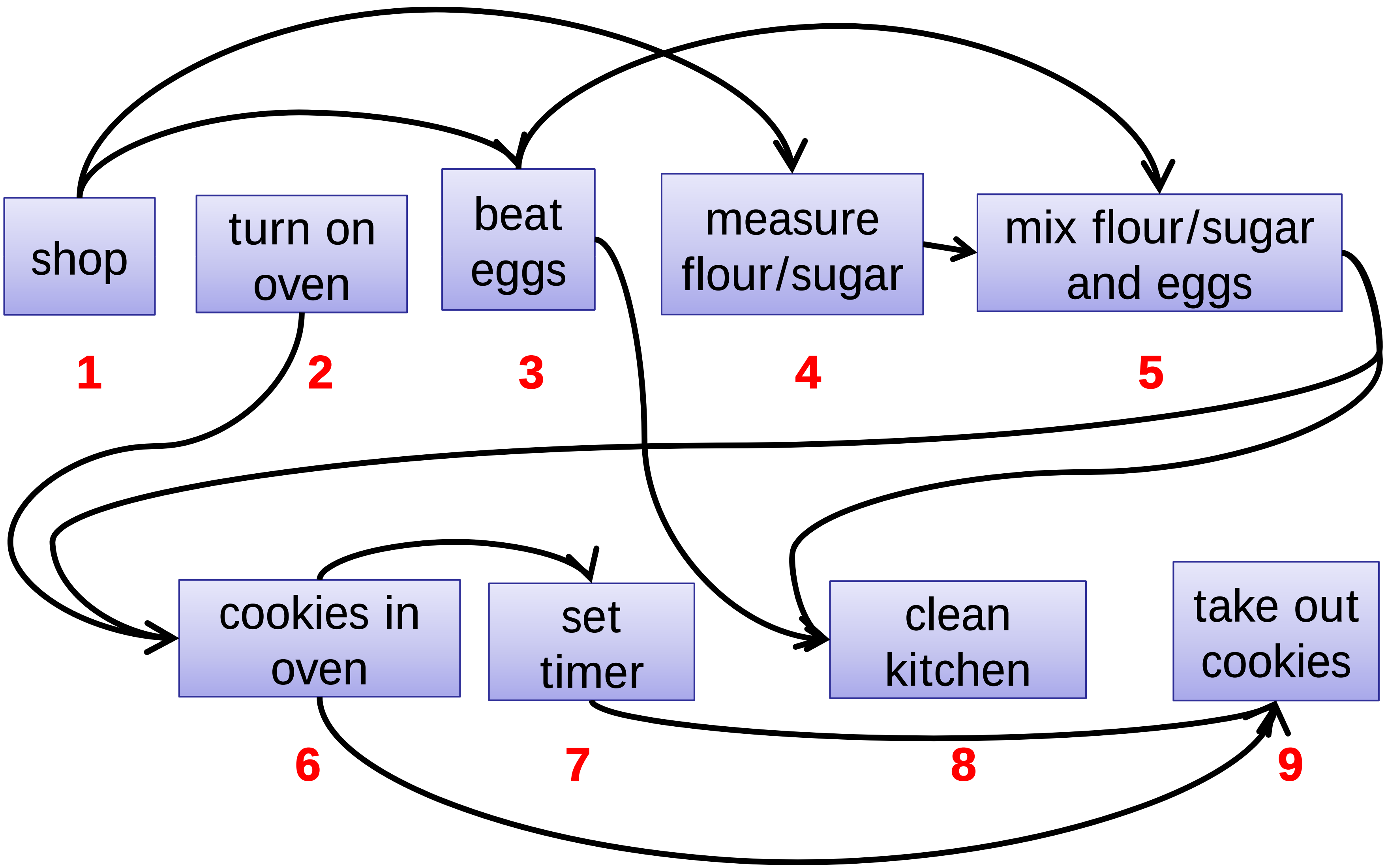
Ordering:

- 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:

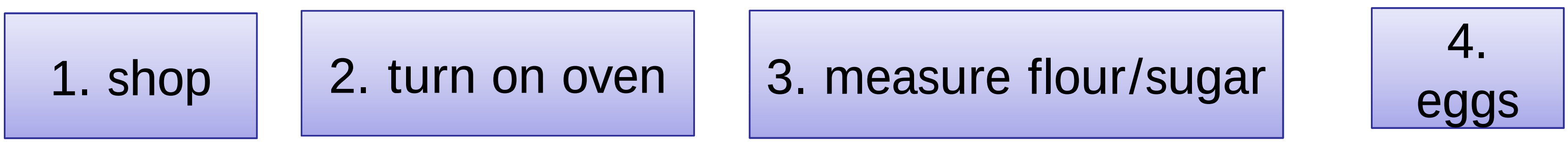
1. Sequential total ordering of all nodes

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

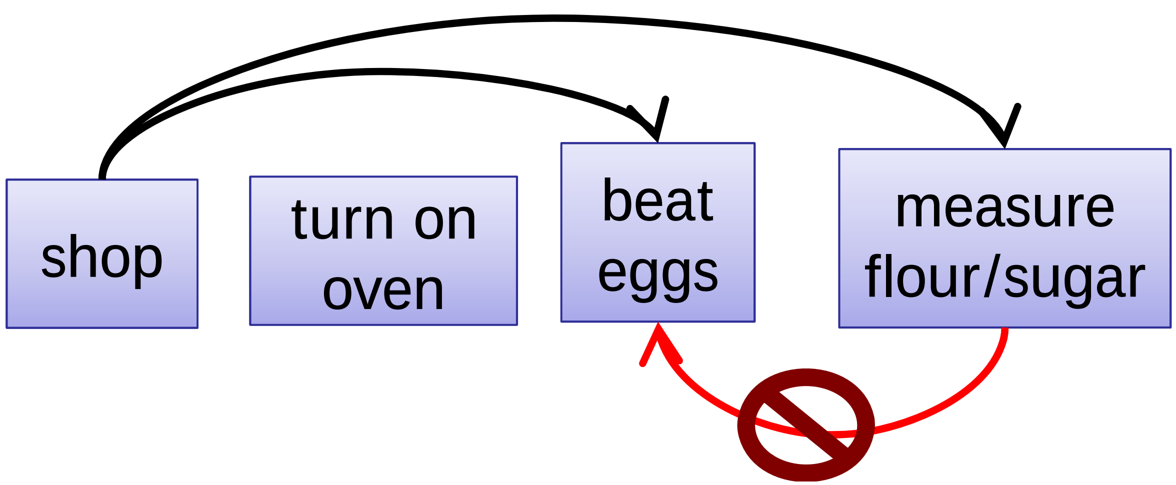
Topological Order

Properties:

1. Sequential total ordering of all nodes

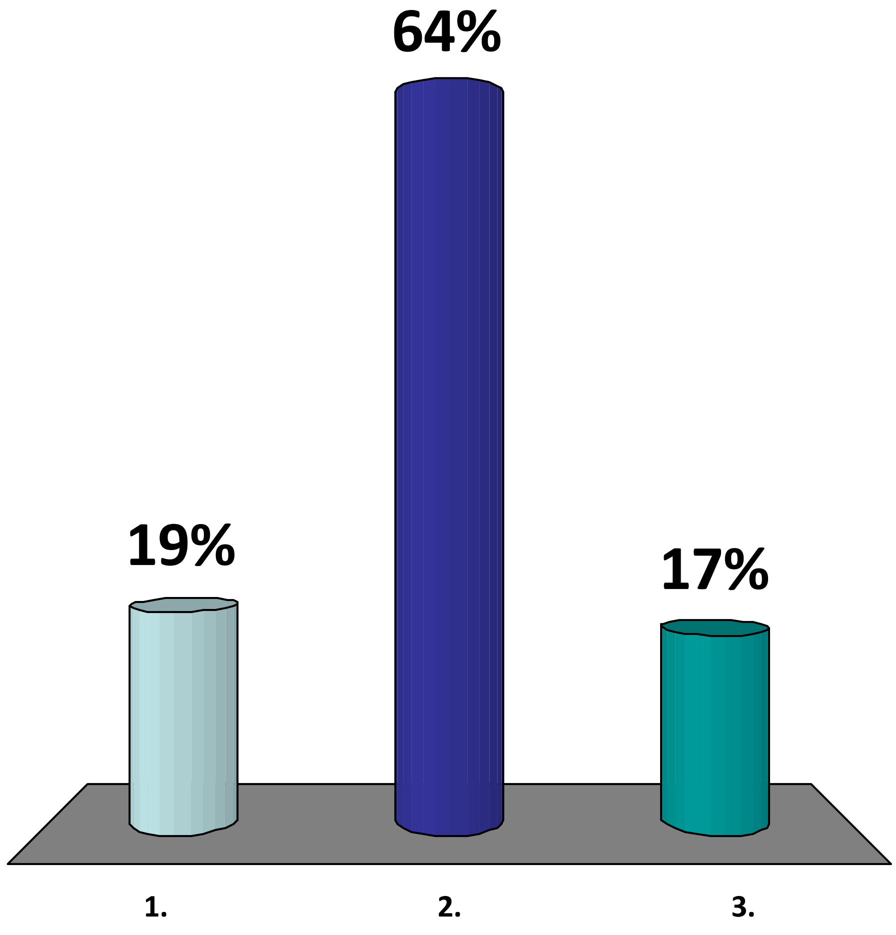


2. Edges only point forward



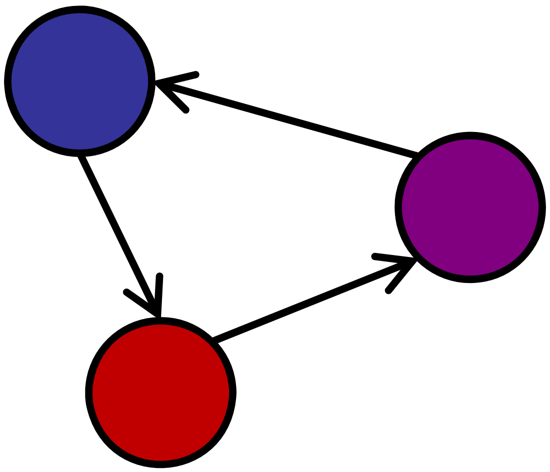
Does every directed graph have a topological ordering?

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

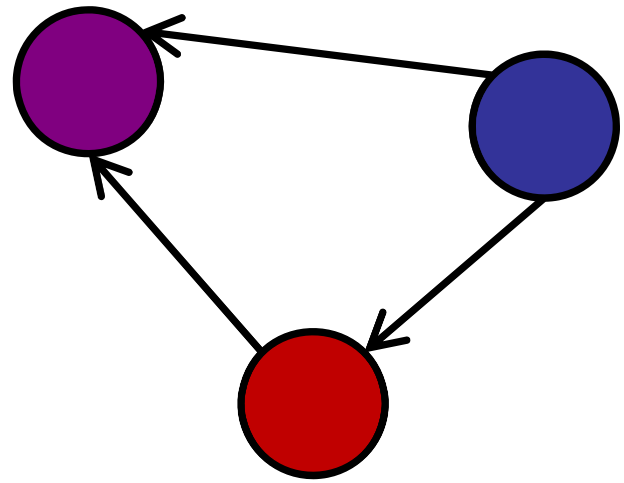


Directed Acyclic Graphs

Cyclic

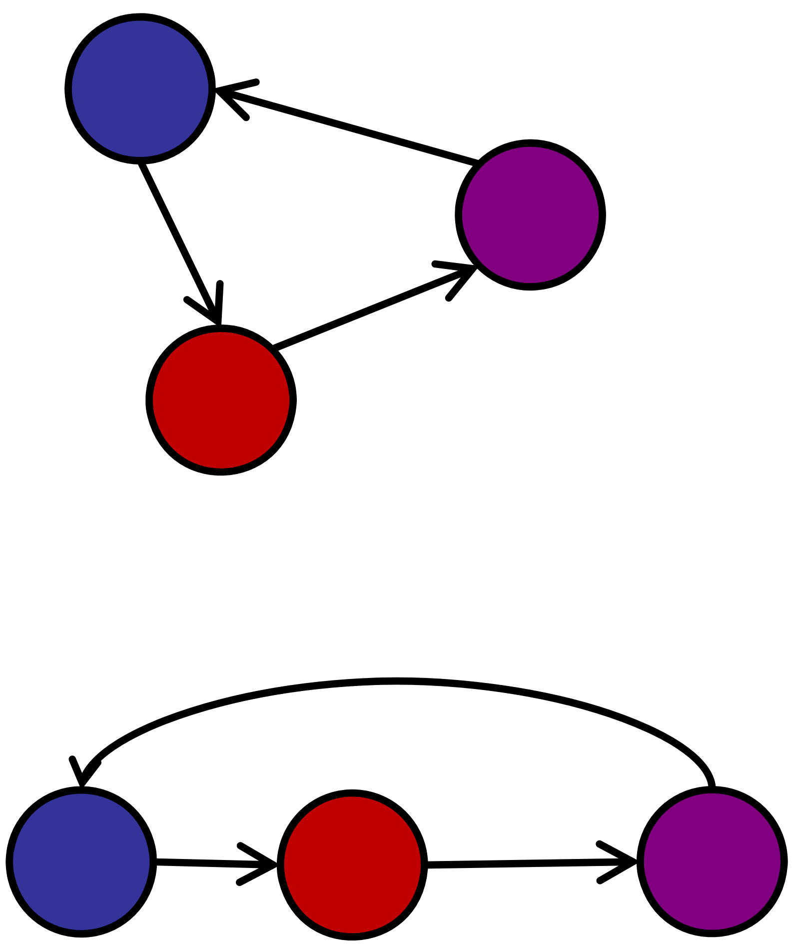


Acyclic

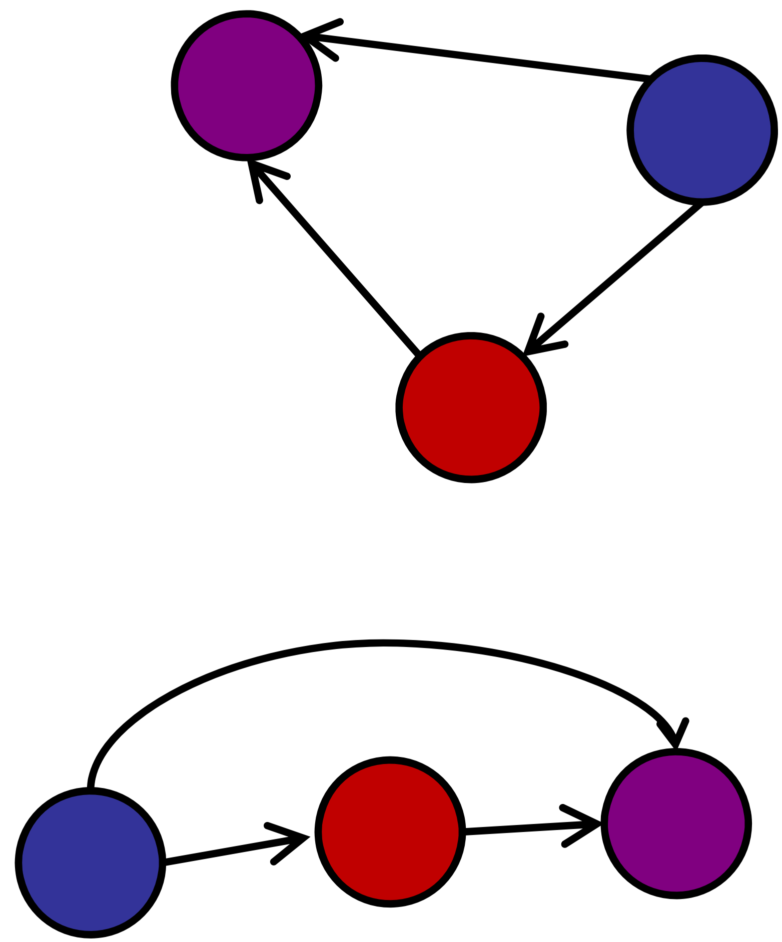


Directed Acyclic Graphs

Cyclic

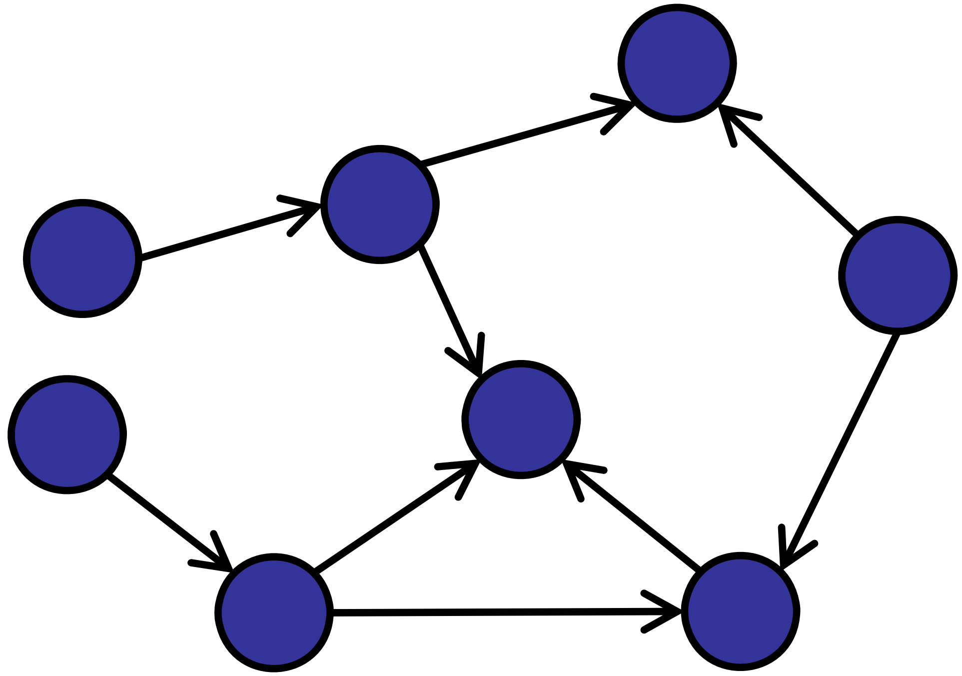


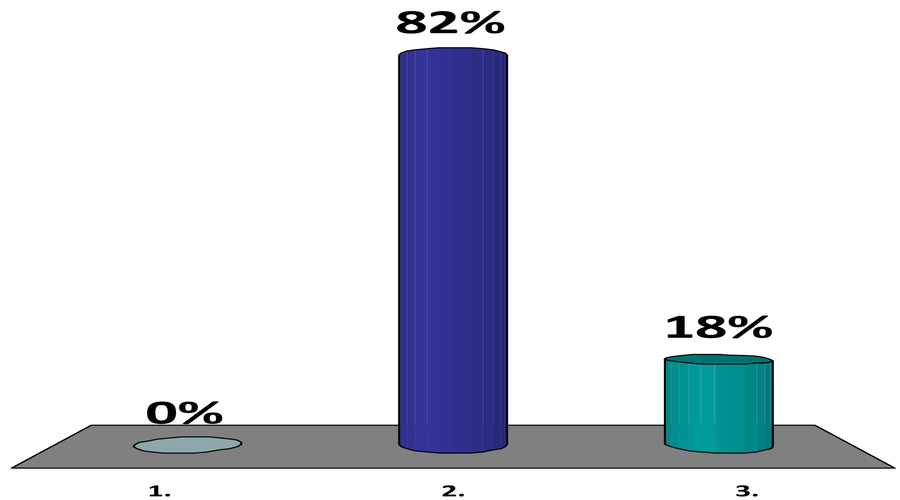
Acyclic



Is this graph:

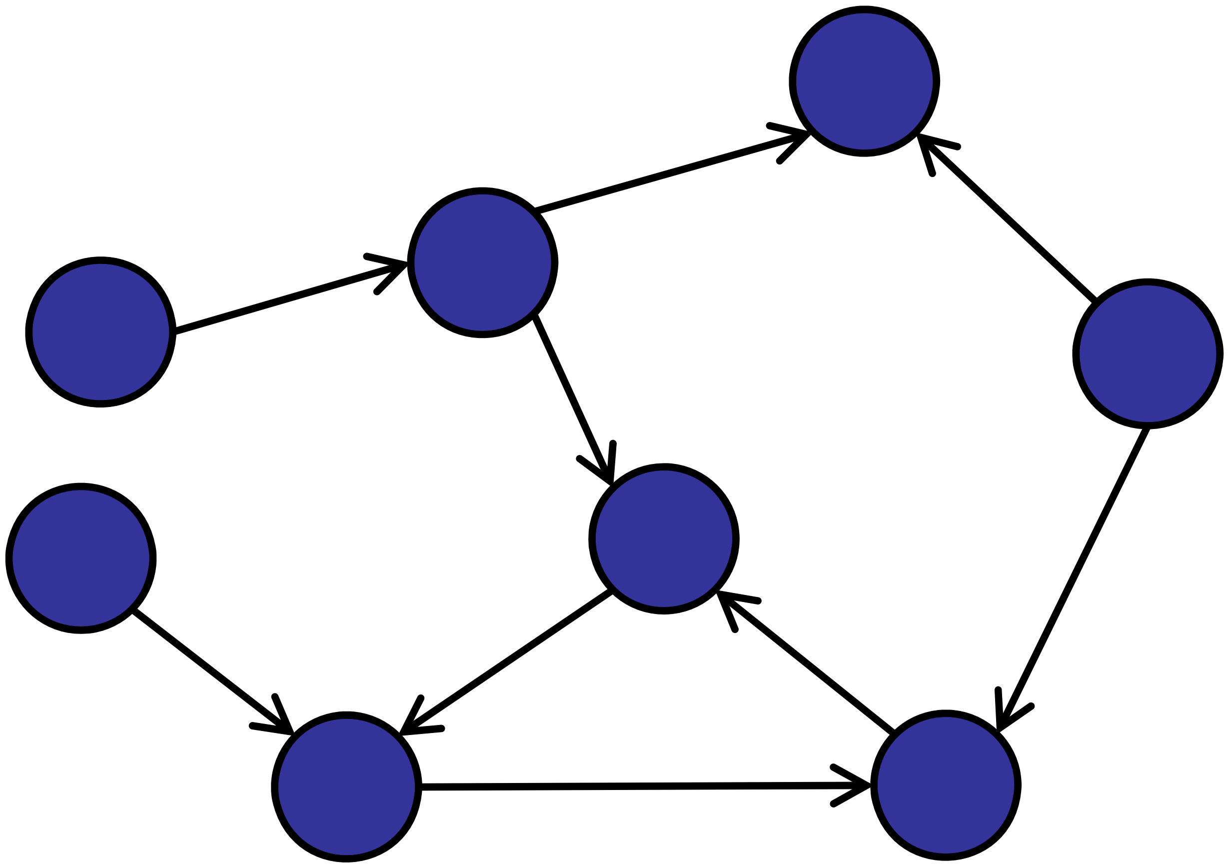
1. Cyclic
2. Acyclic
3. Transcendental



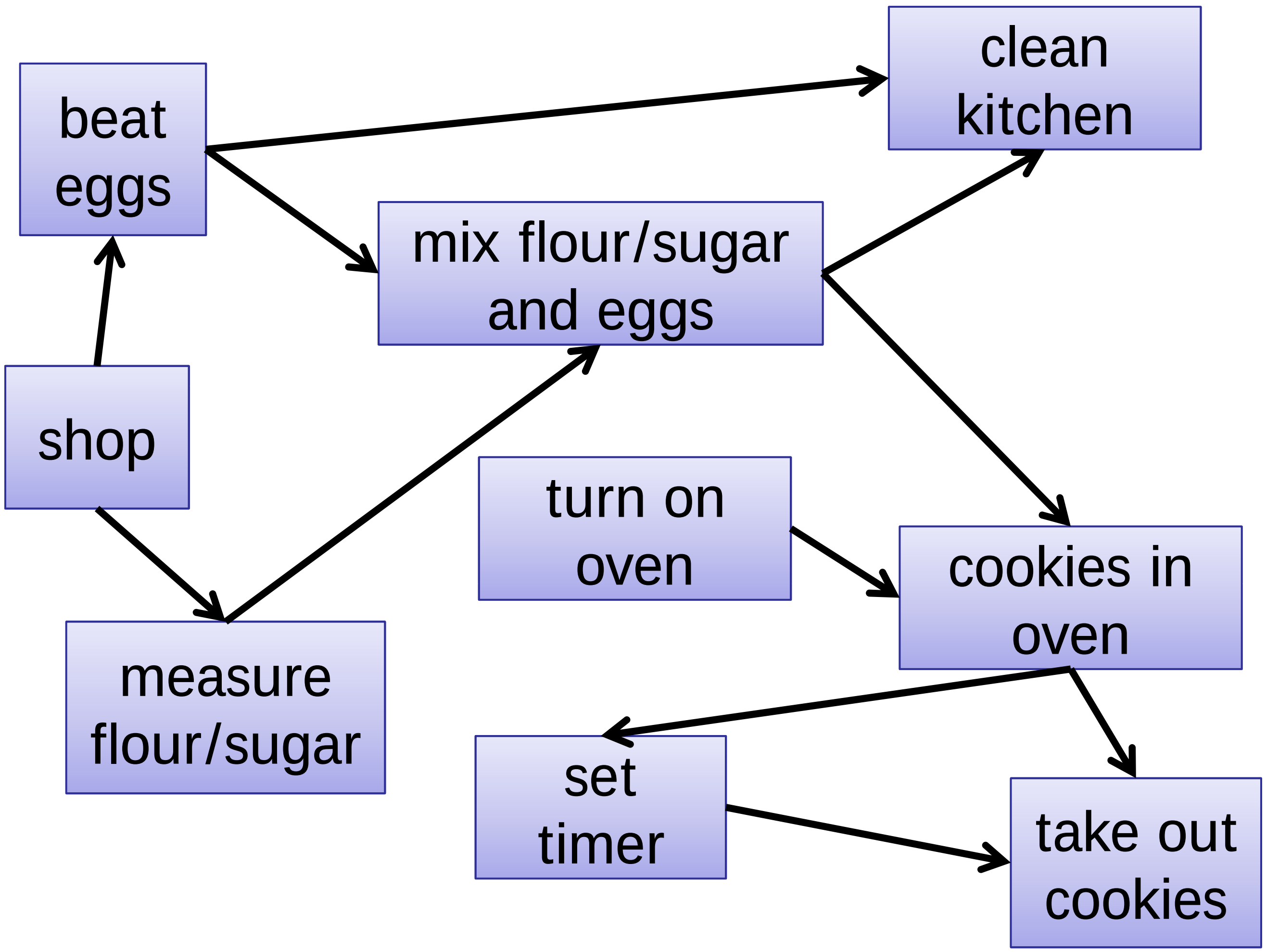


Directed Acyclic Graphs

Cyclic or Acyclic?



Directed Acyclic Graph (DAG)



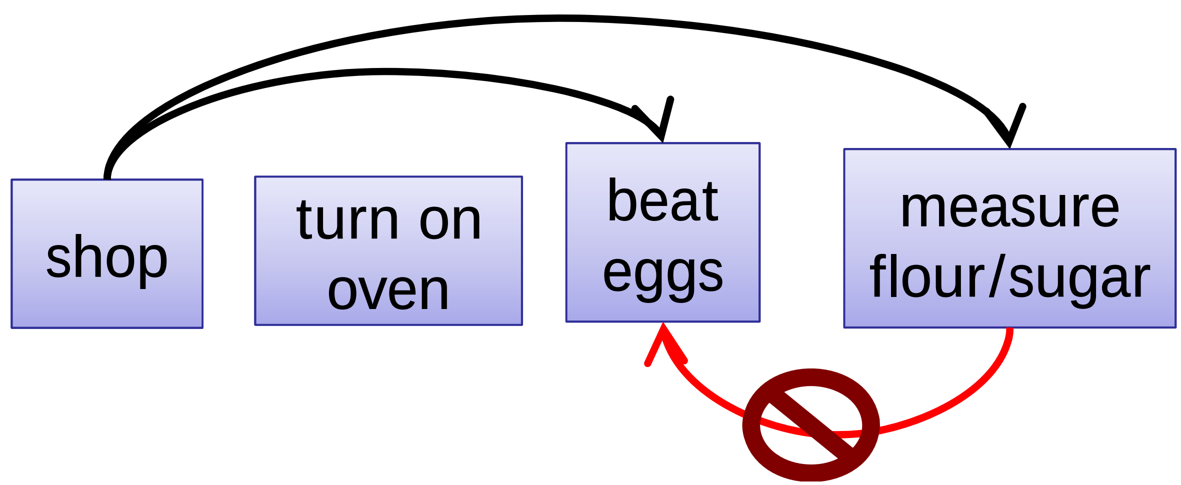
Topological Order

Properties:

1. Sequential total ordering of all nodes

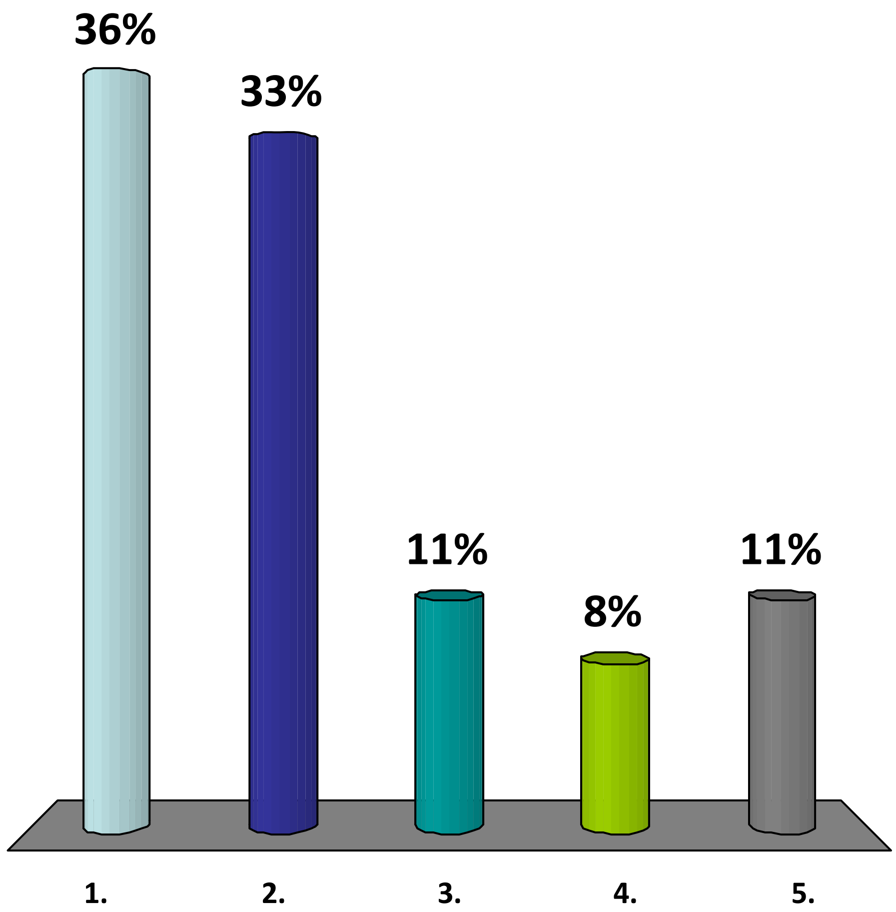
1. shop | 2. turn on oven | 3. measure flour/sugar | 4. eggs

2. Edges only point forward

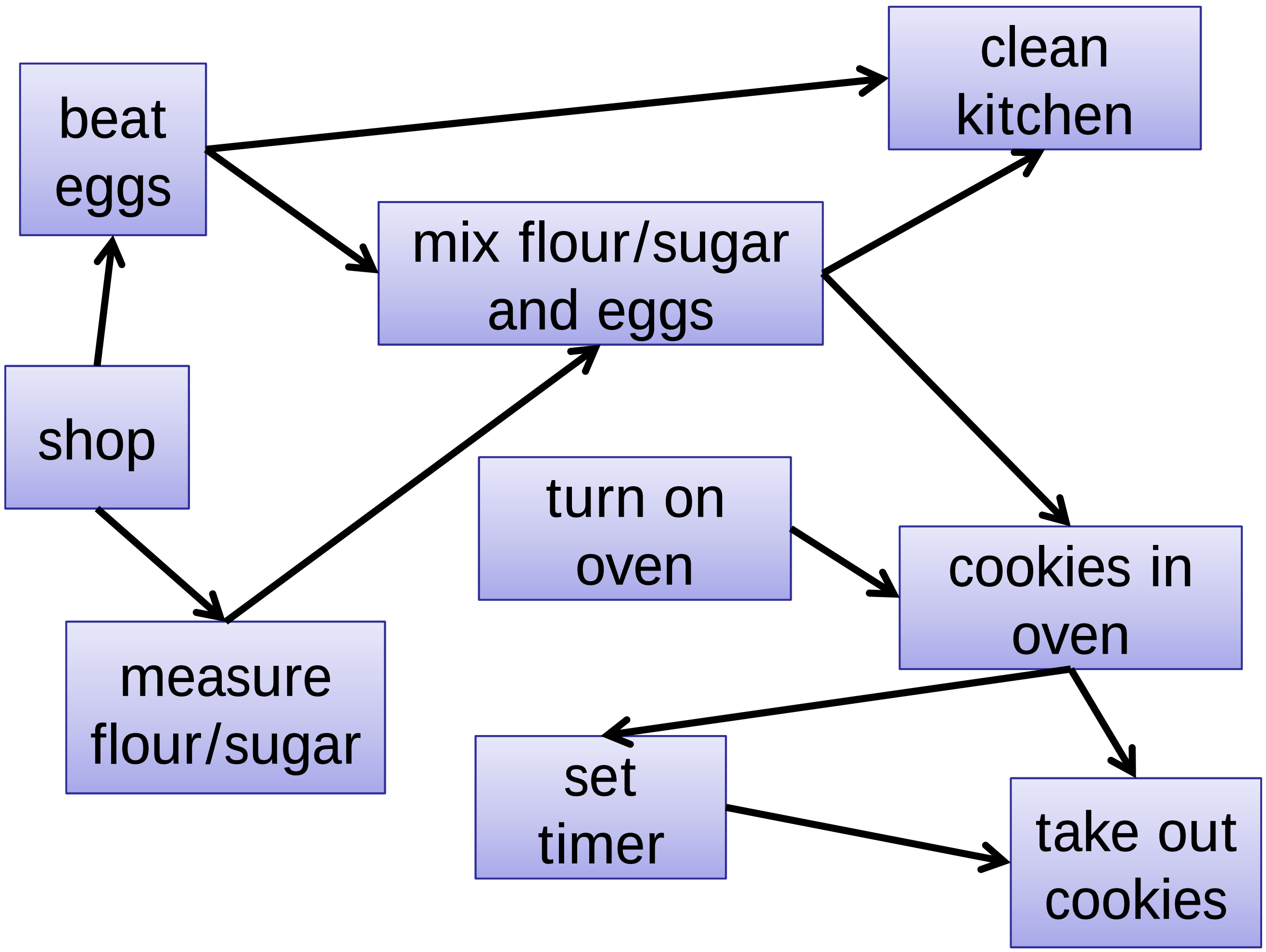


Which algorithm is best for finding a Topological Ordering in a DAG?

1. Breadth-first search
2. Depth-first search
3. Bellman-Ford
4. Prim's
5. Something else

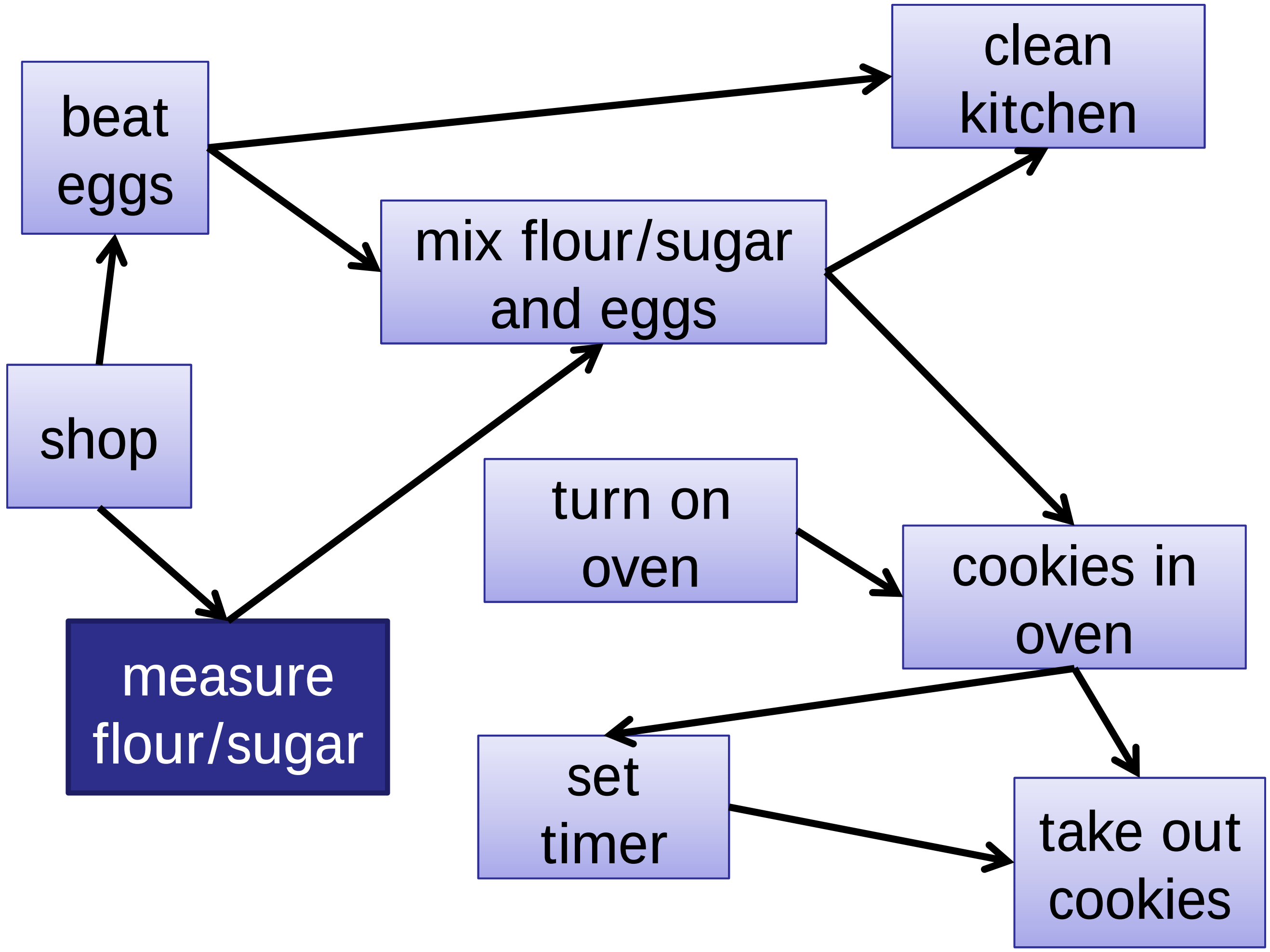


Depth-First Search



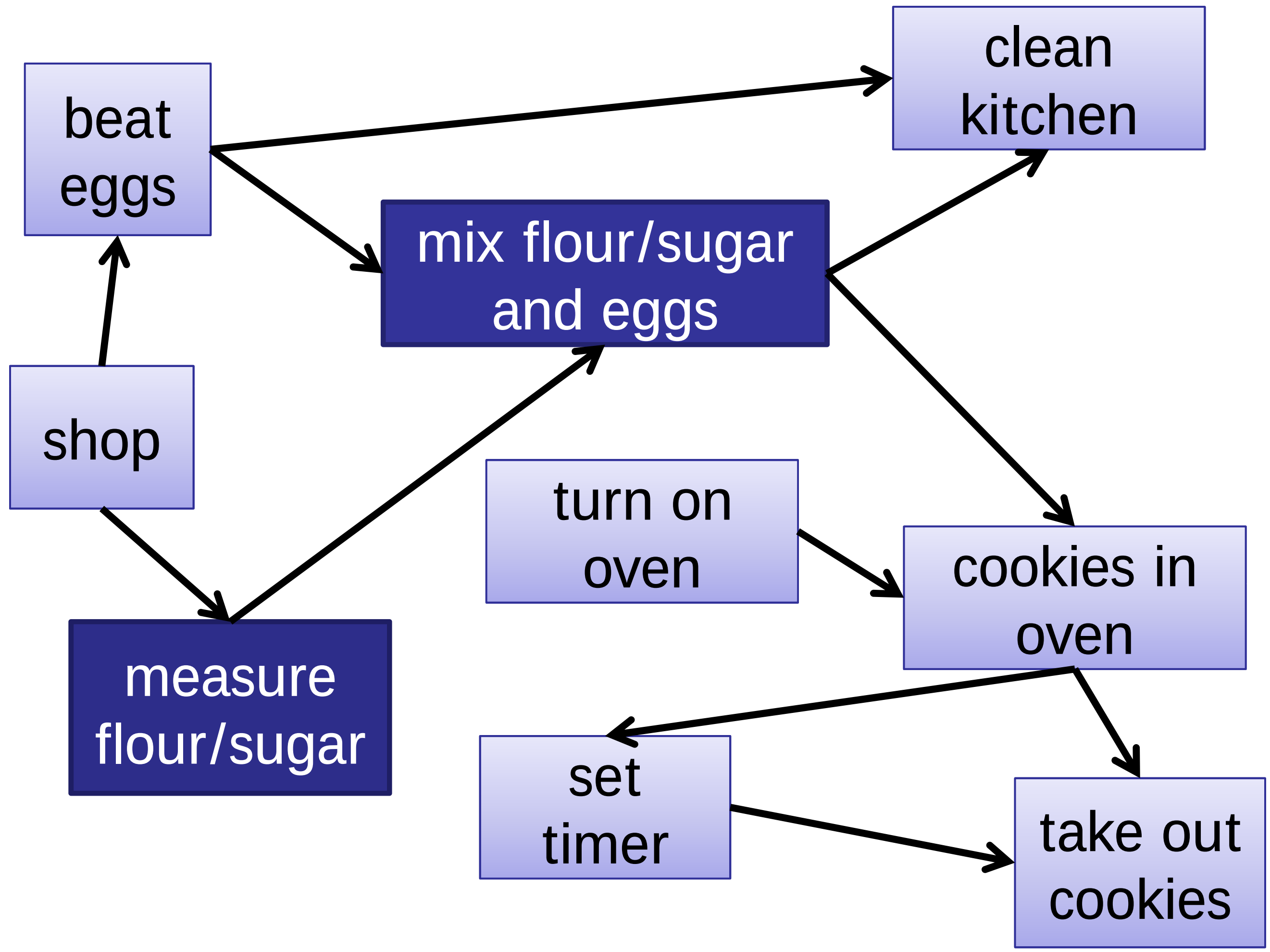
Depth-First Search

1. measure



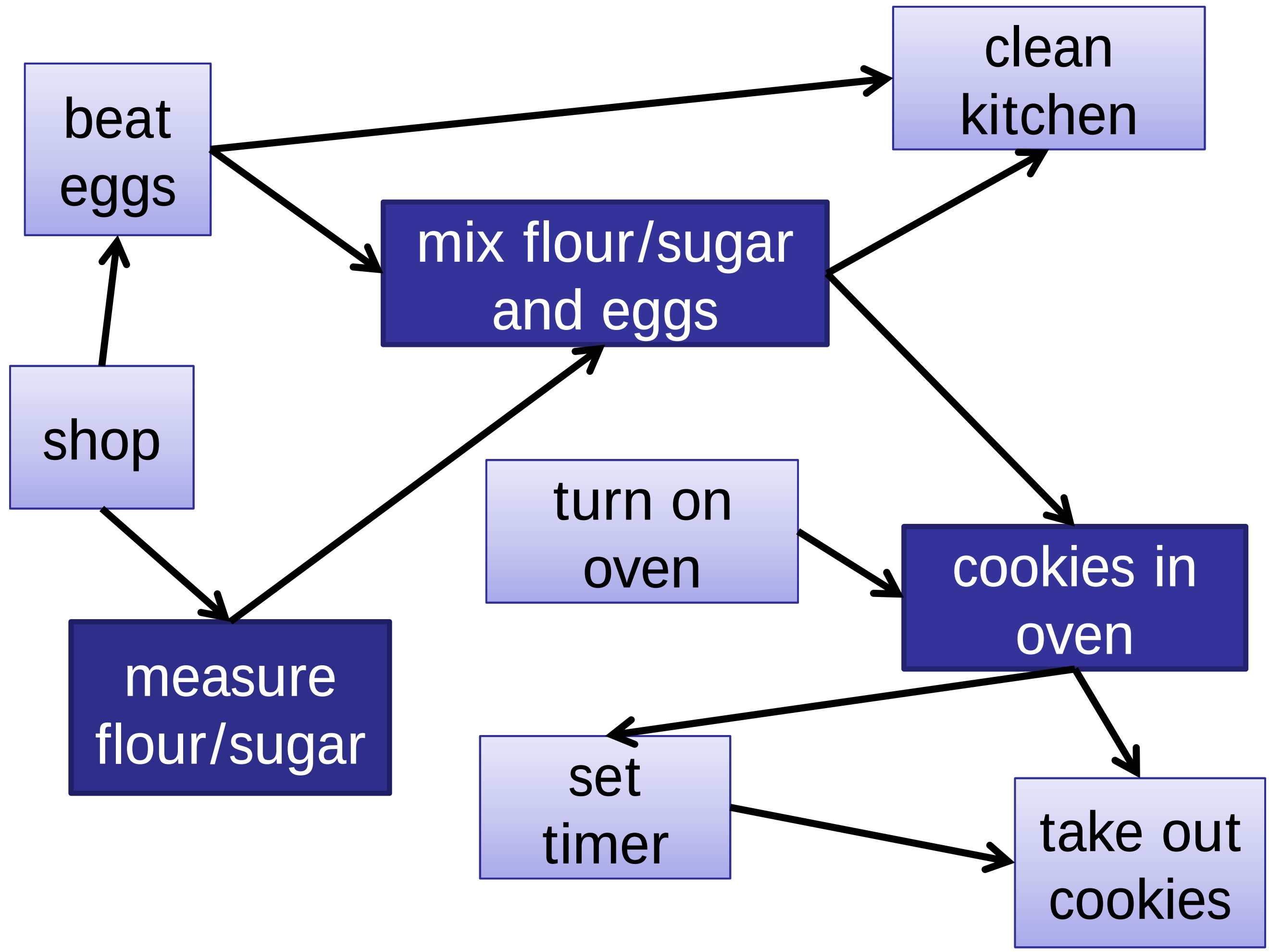
Depth-First Search

1. measure
2. mix



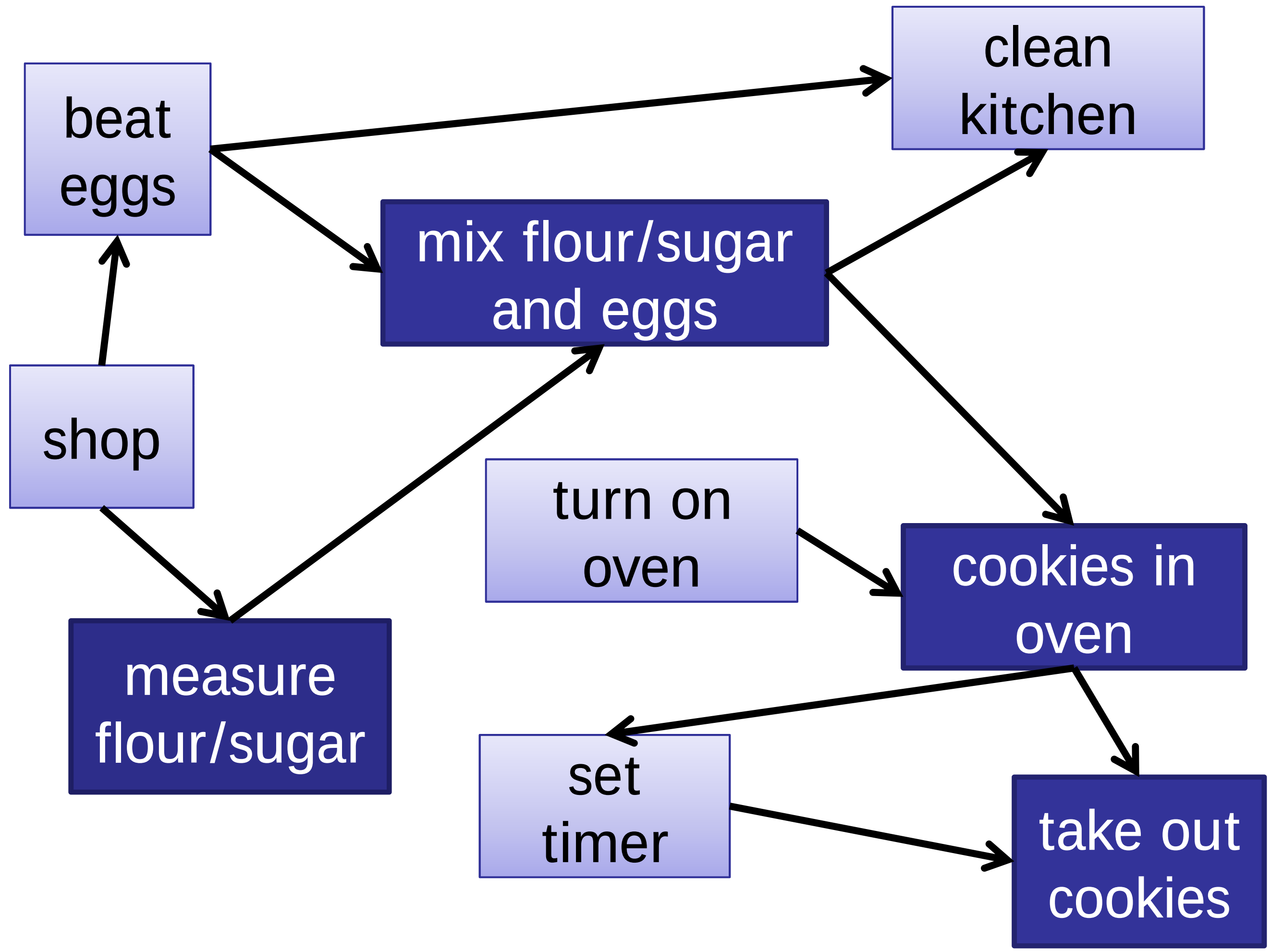
Depth-First Search

1. measure
2. mix
3. in oven



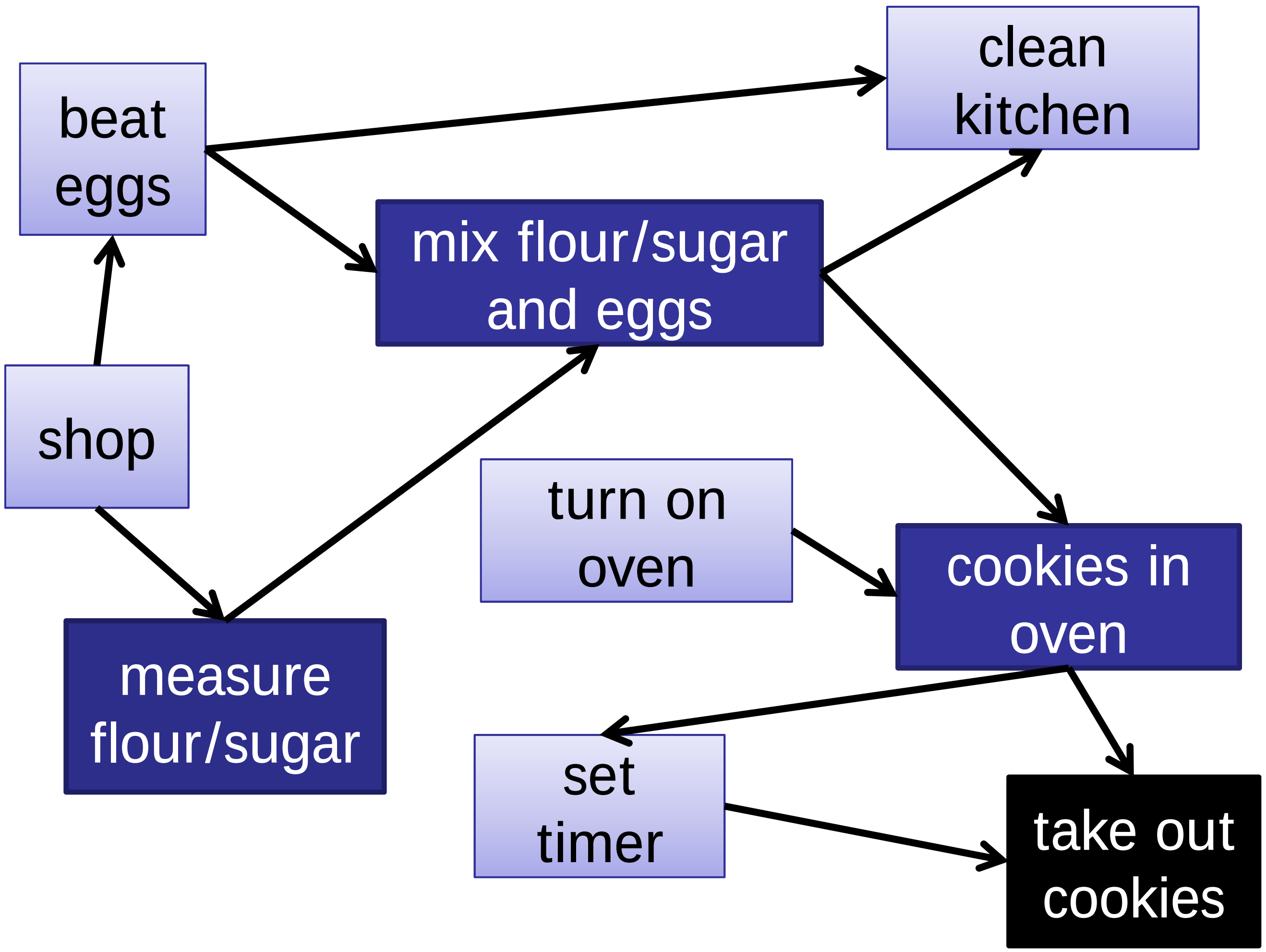
Depth-First Search

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



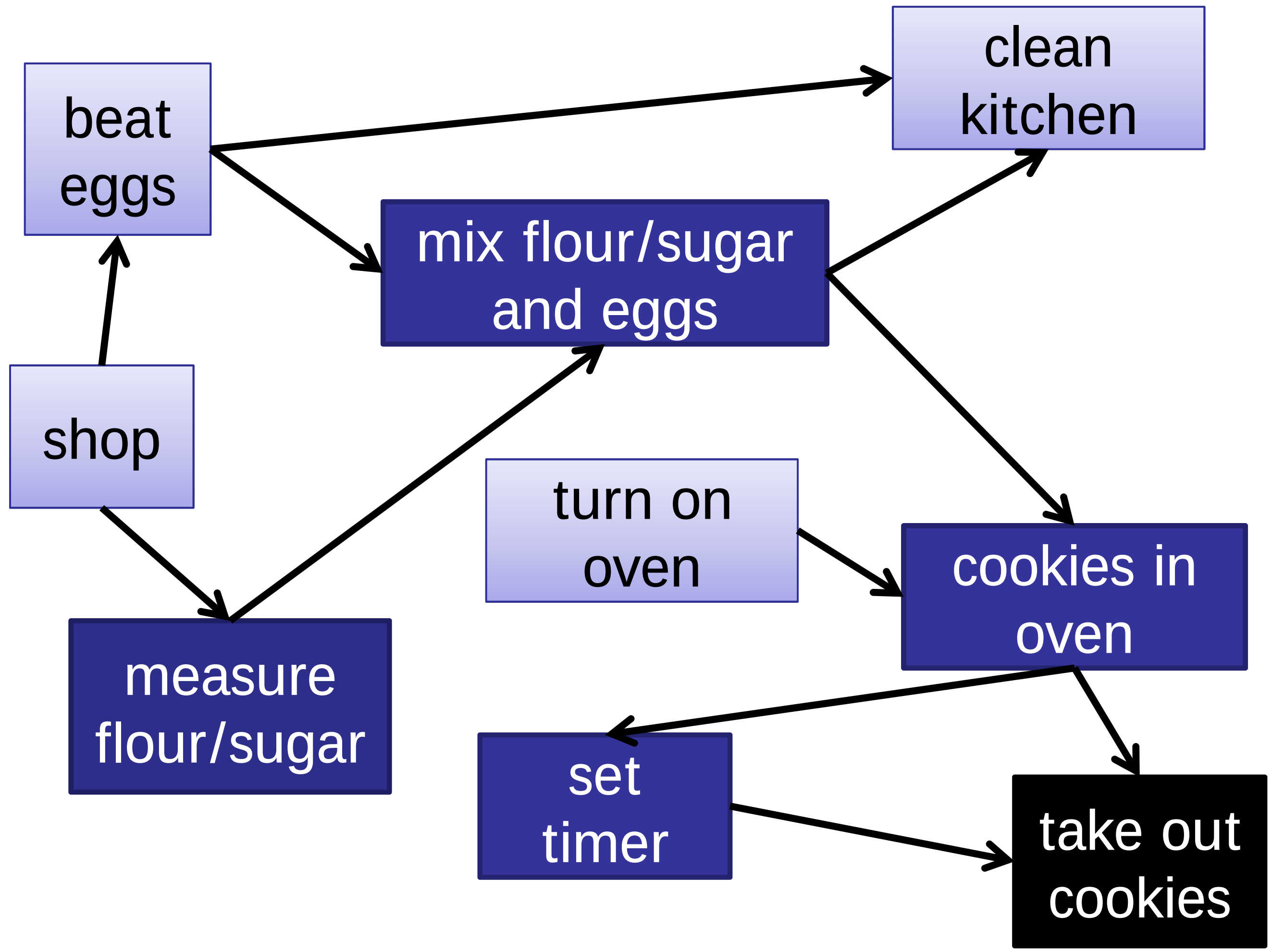
Depth-First Search

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



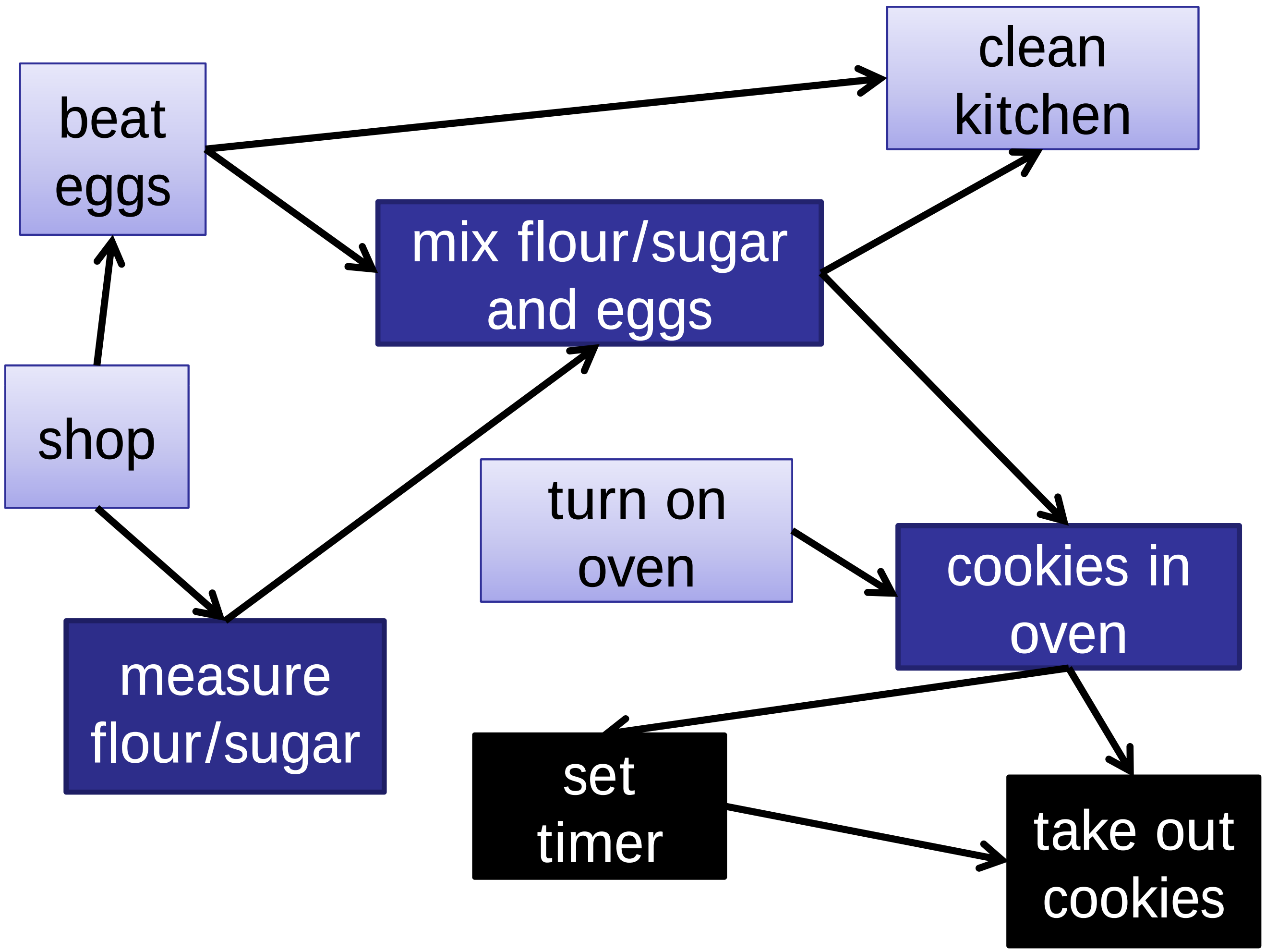
Depth-First Search

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



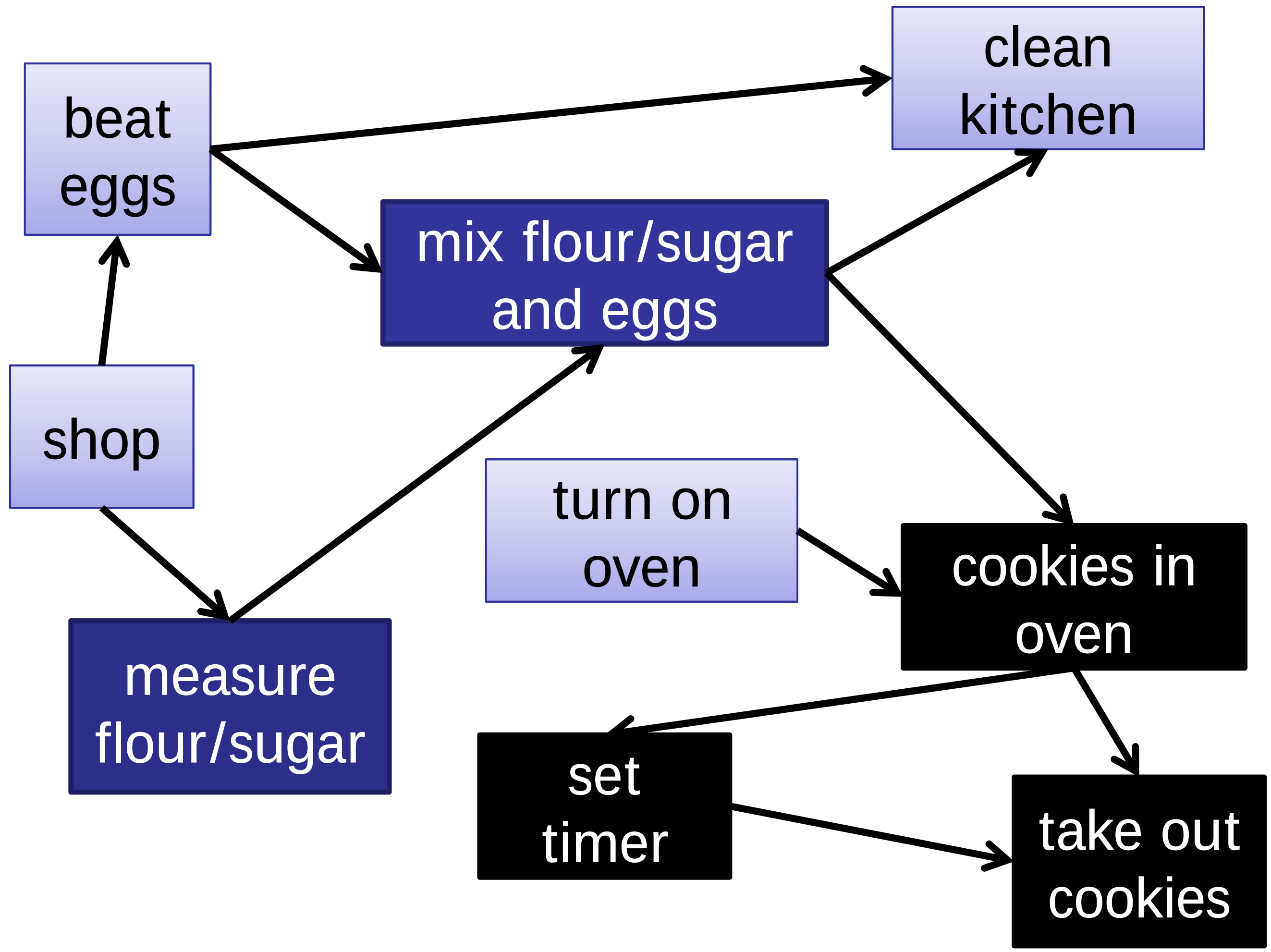
Depth-First Search

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



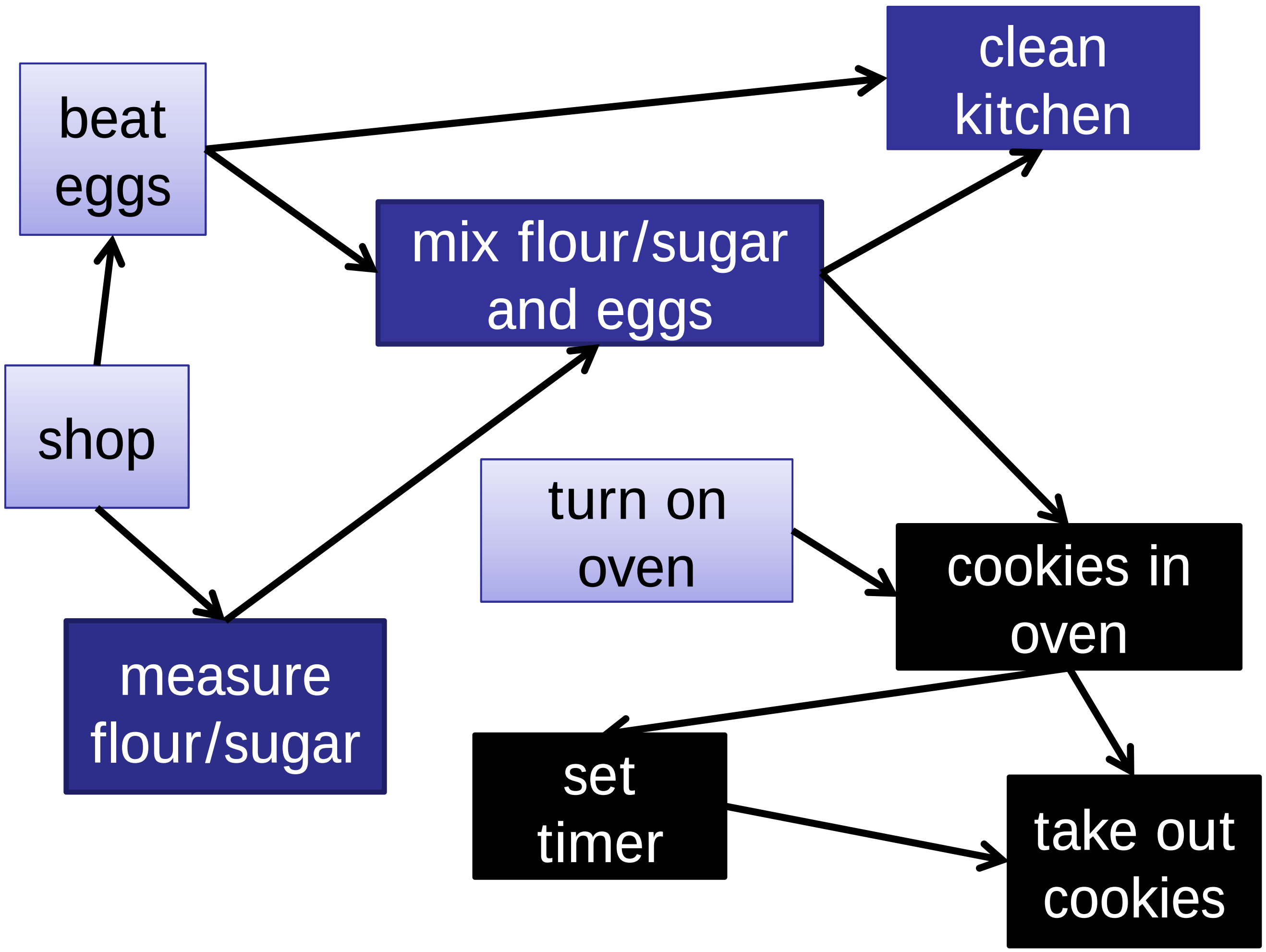
Depth-First Search

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



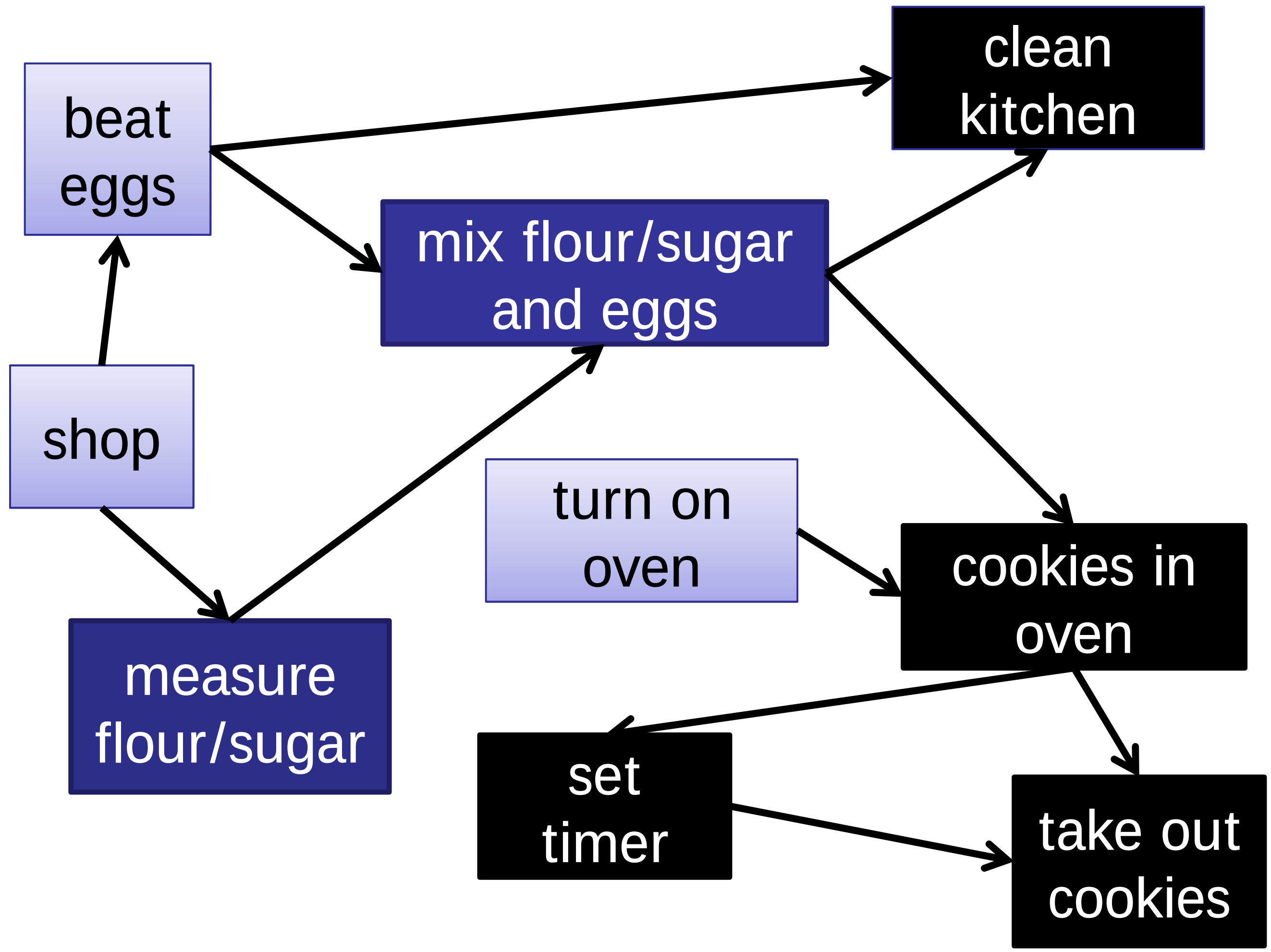
Depth-First Search

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



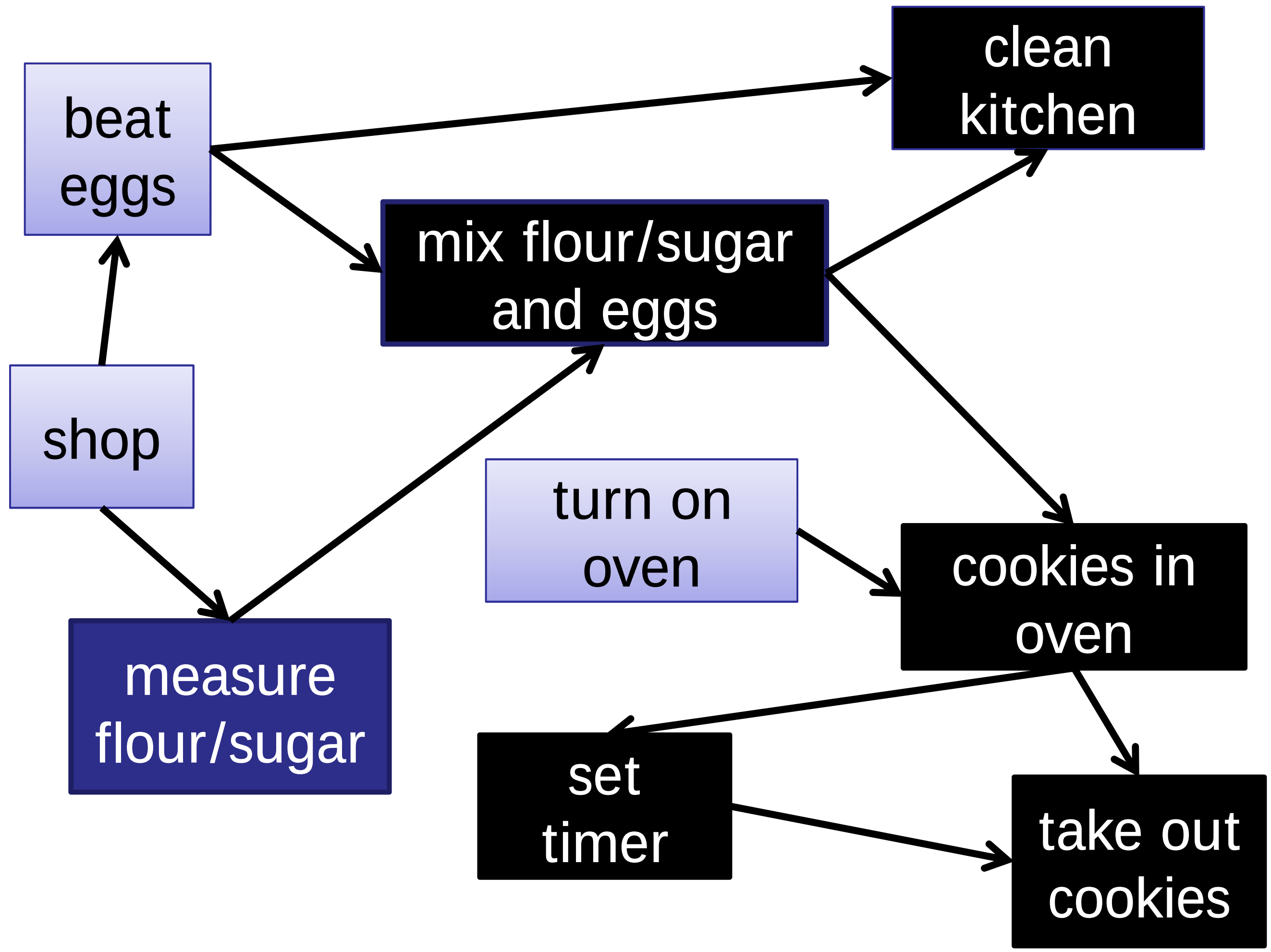
Depth-First Search

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



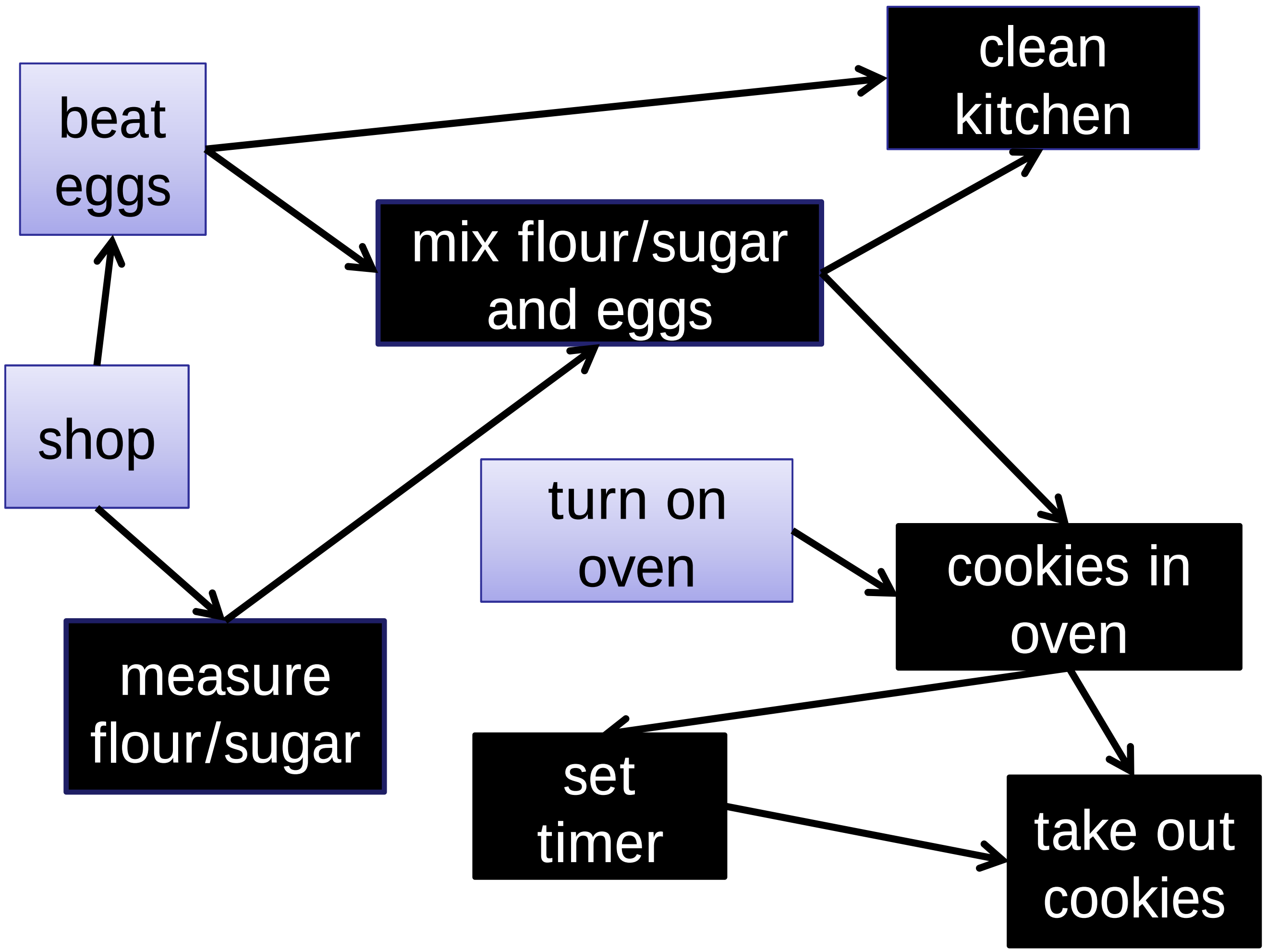
Depth-First Search

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.

Depth-First Search

```
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);
     }
  }
}
```

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);
           ProcessNode(v);
     }
  }
}
```

Searching a (Directed) Graph

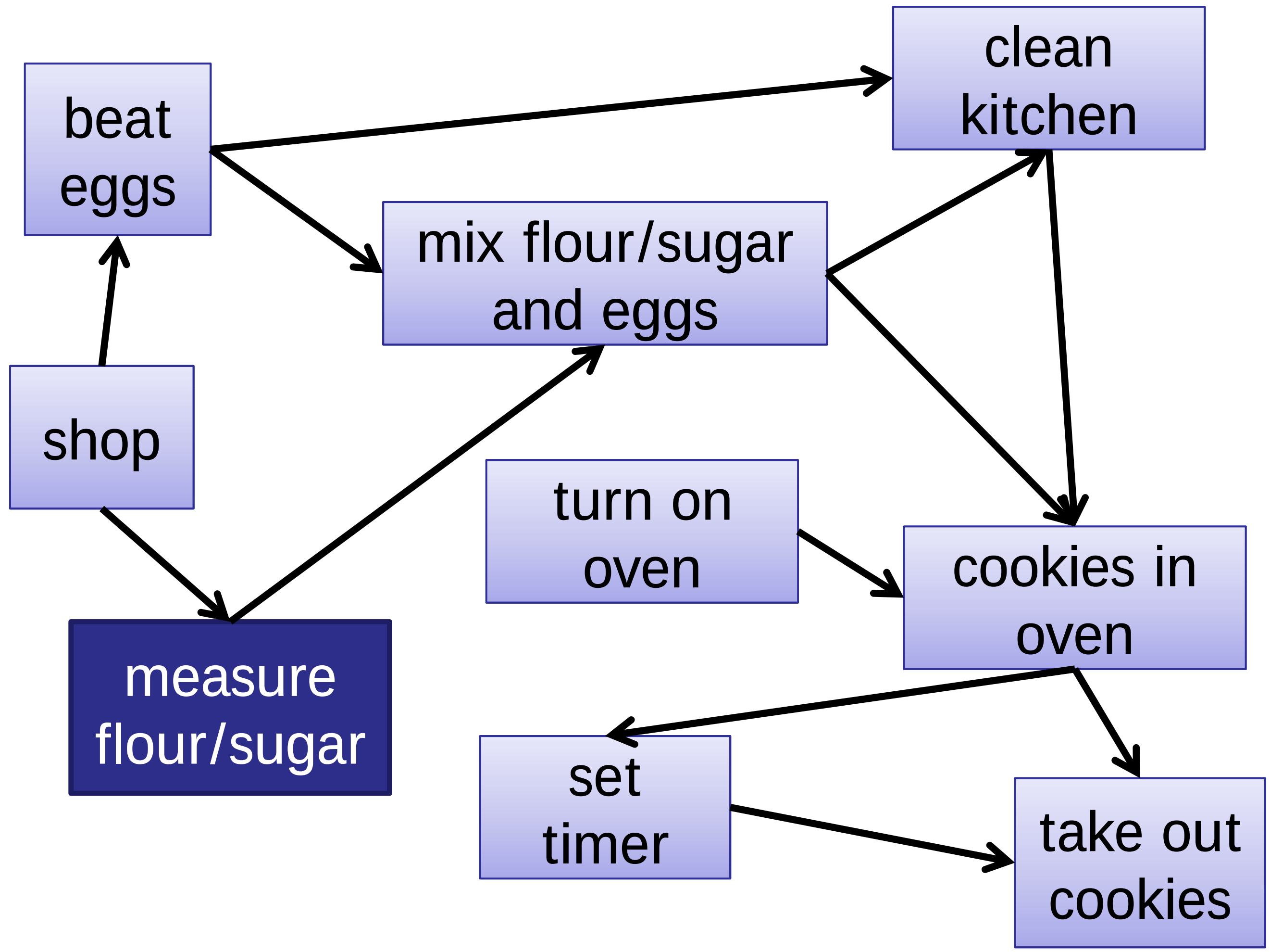
Pre-Order Depth-First Search:

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

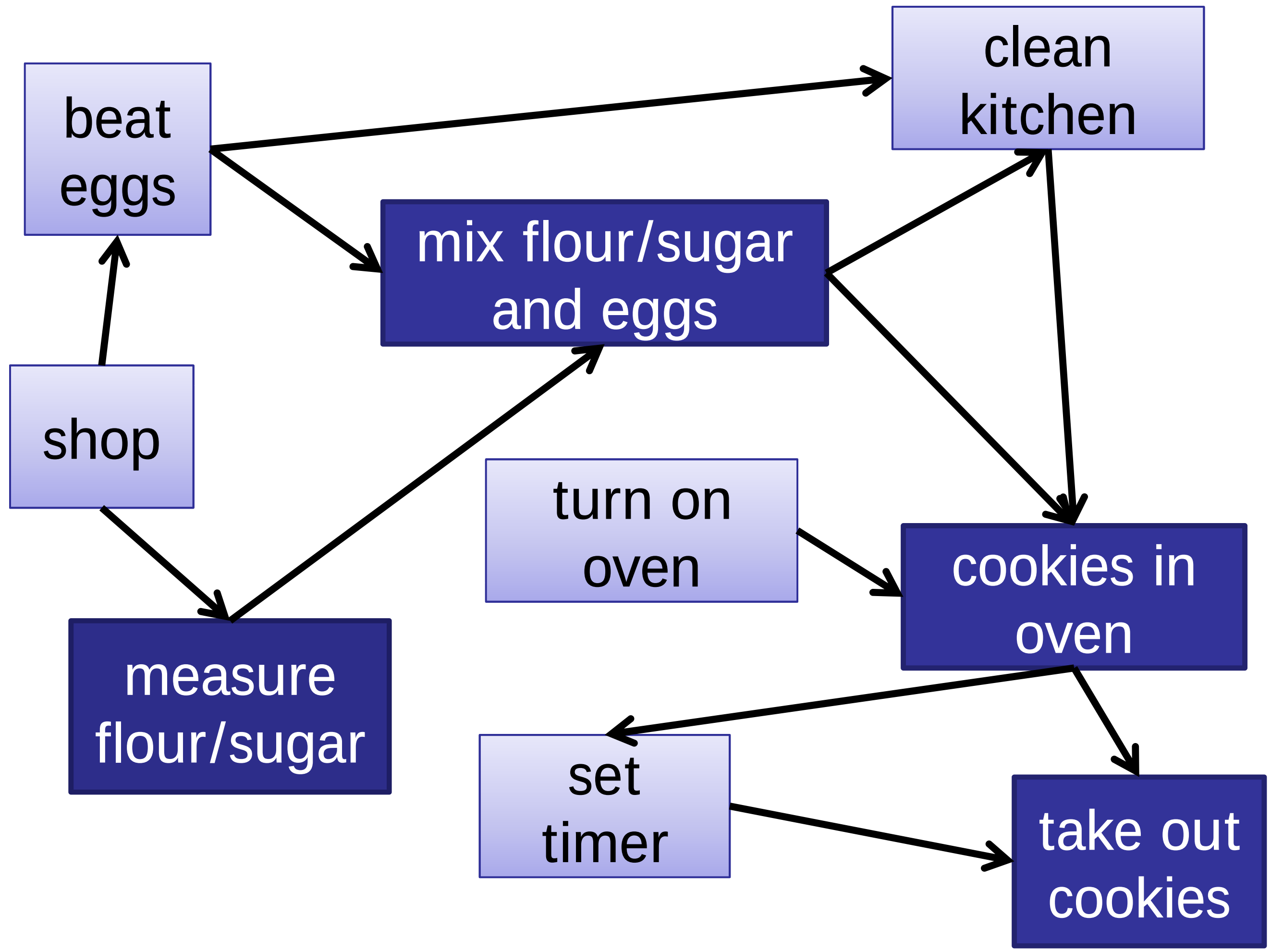
Post-Order Depth-First Search:

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

Post-Order Depth-First Search

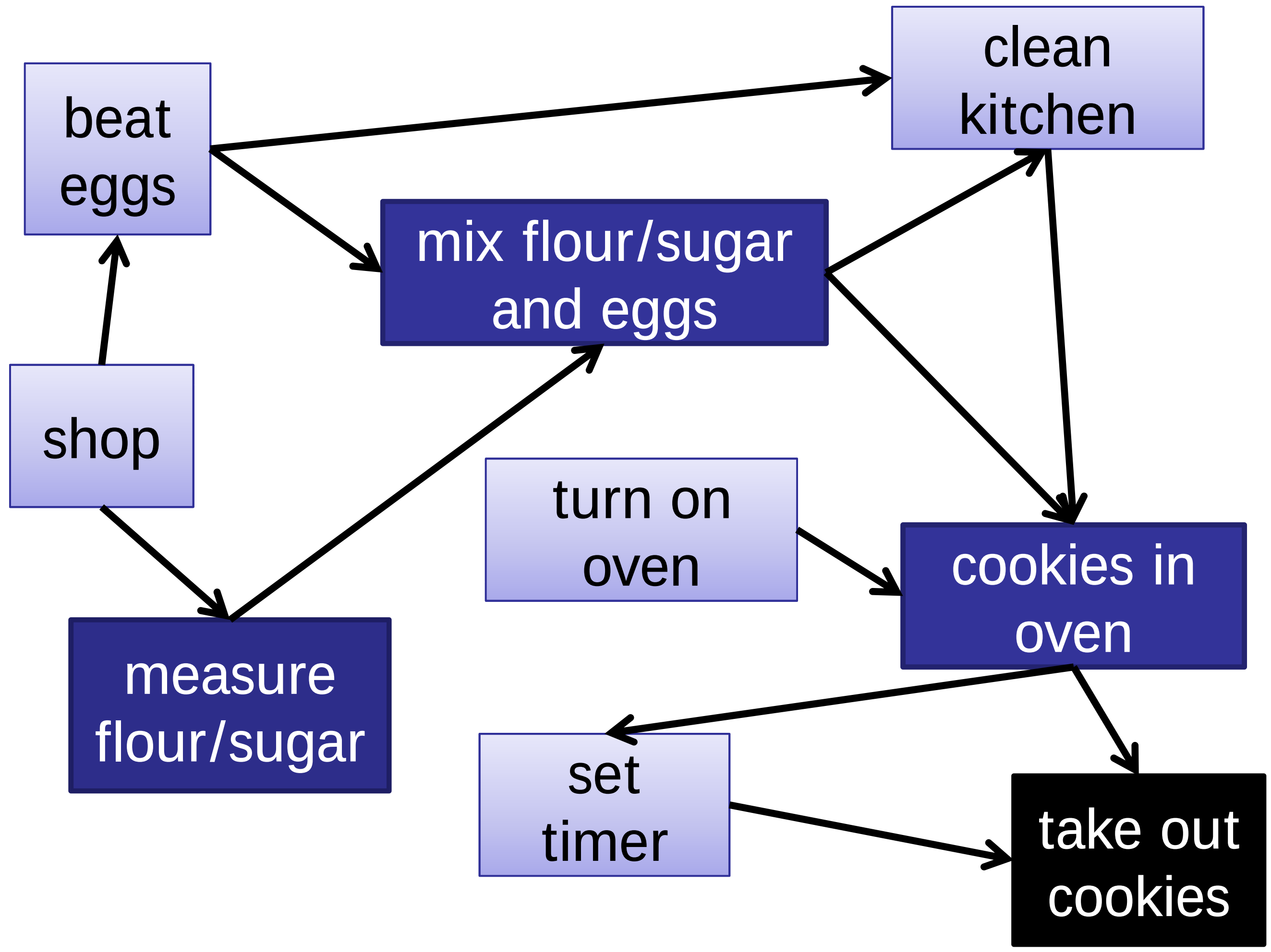


Post-Order Depth-First Search



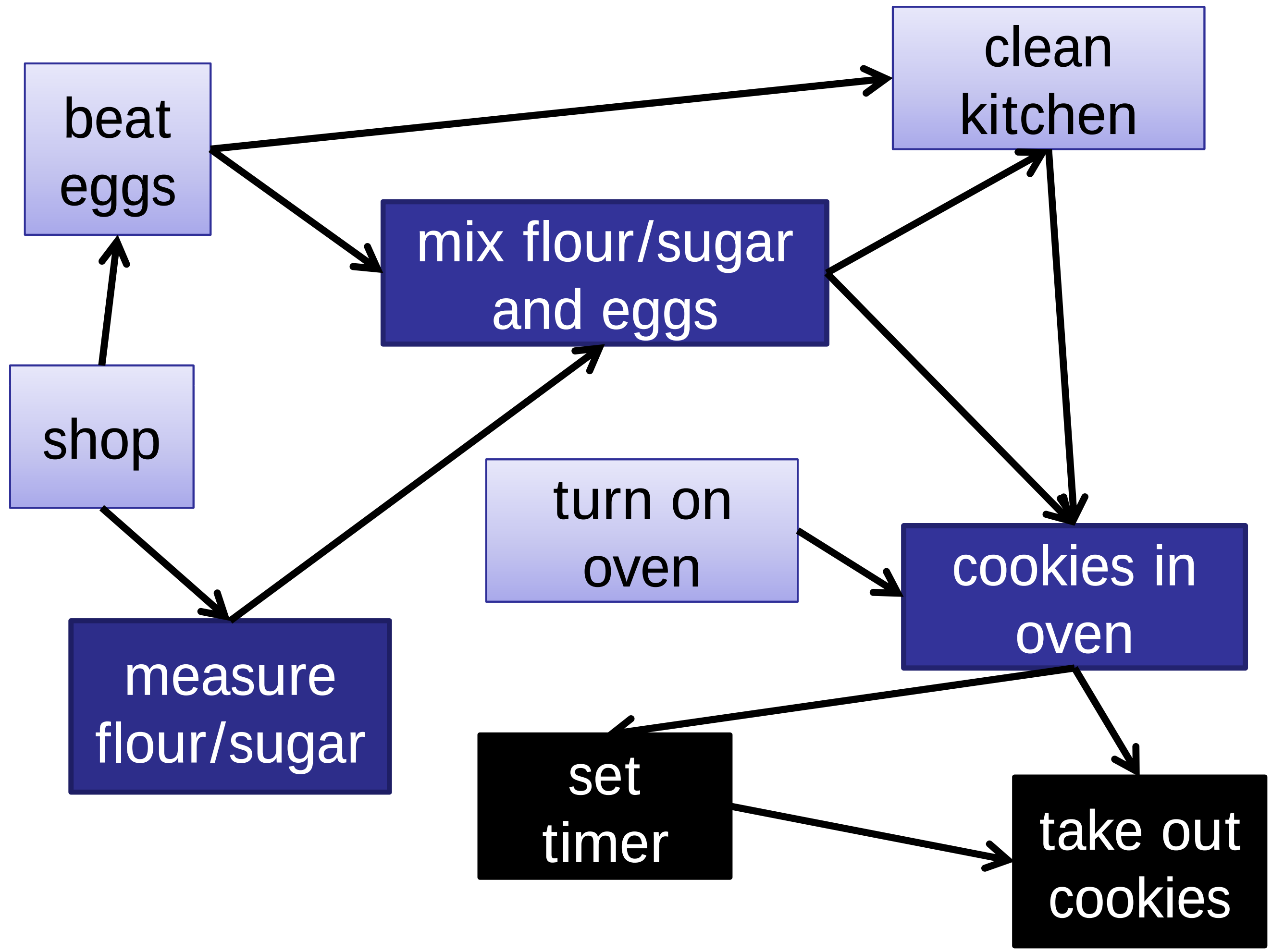
Post-Order Depth-First Search

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



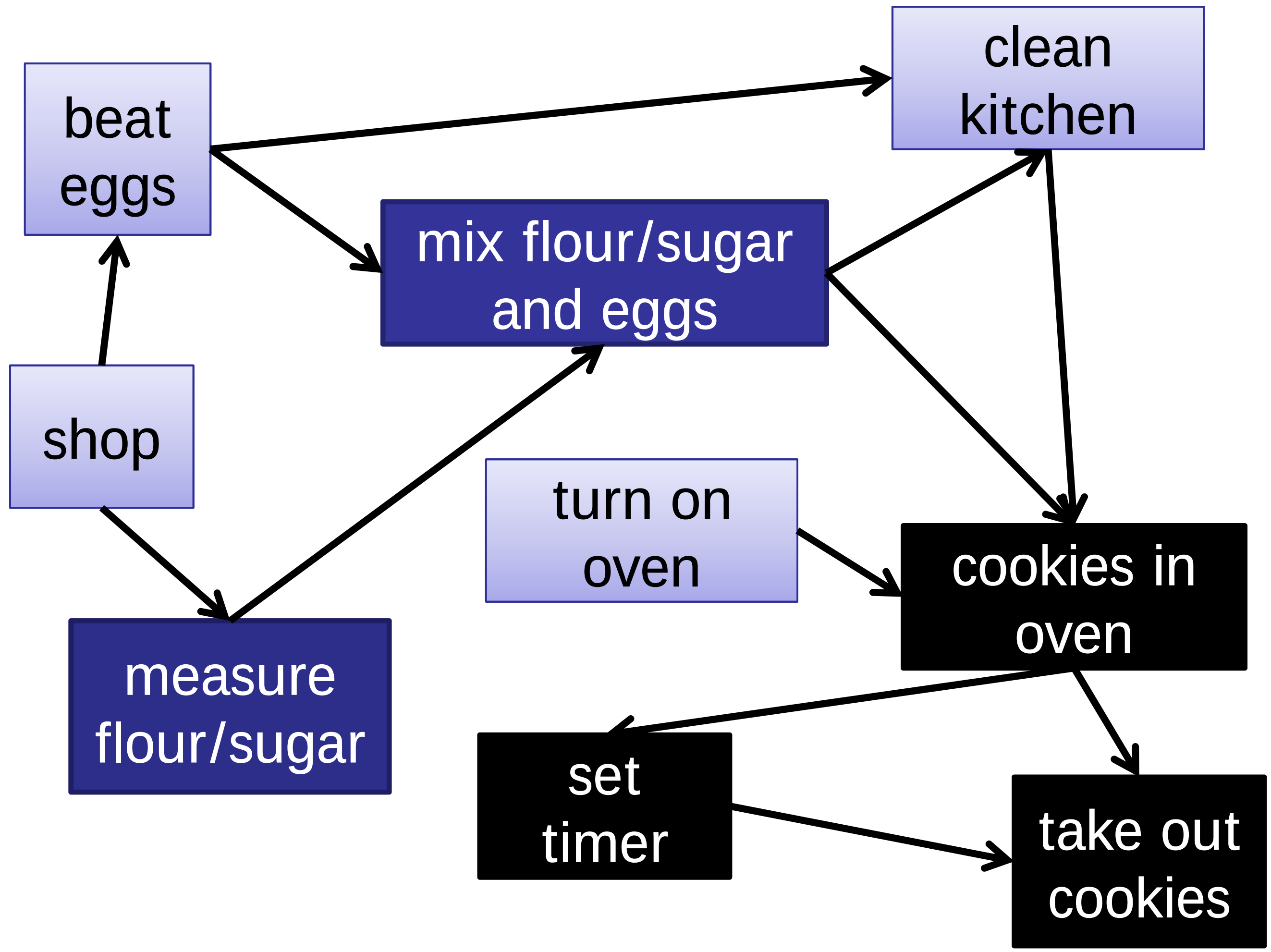
Post-Order Depth-First Search

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



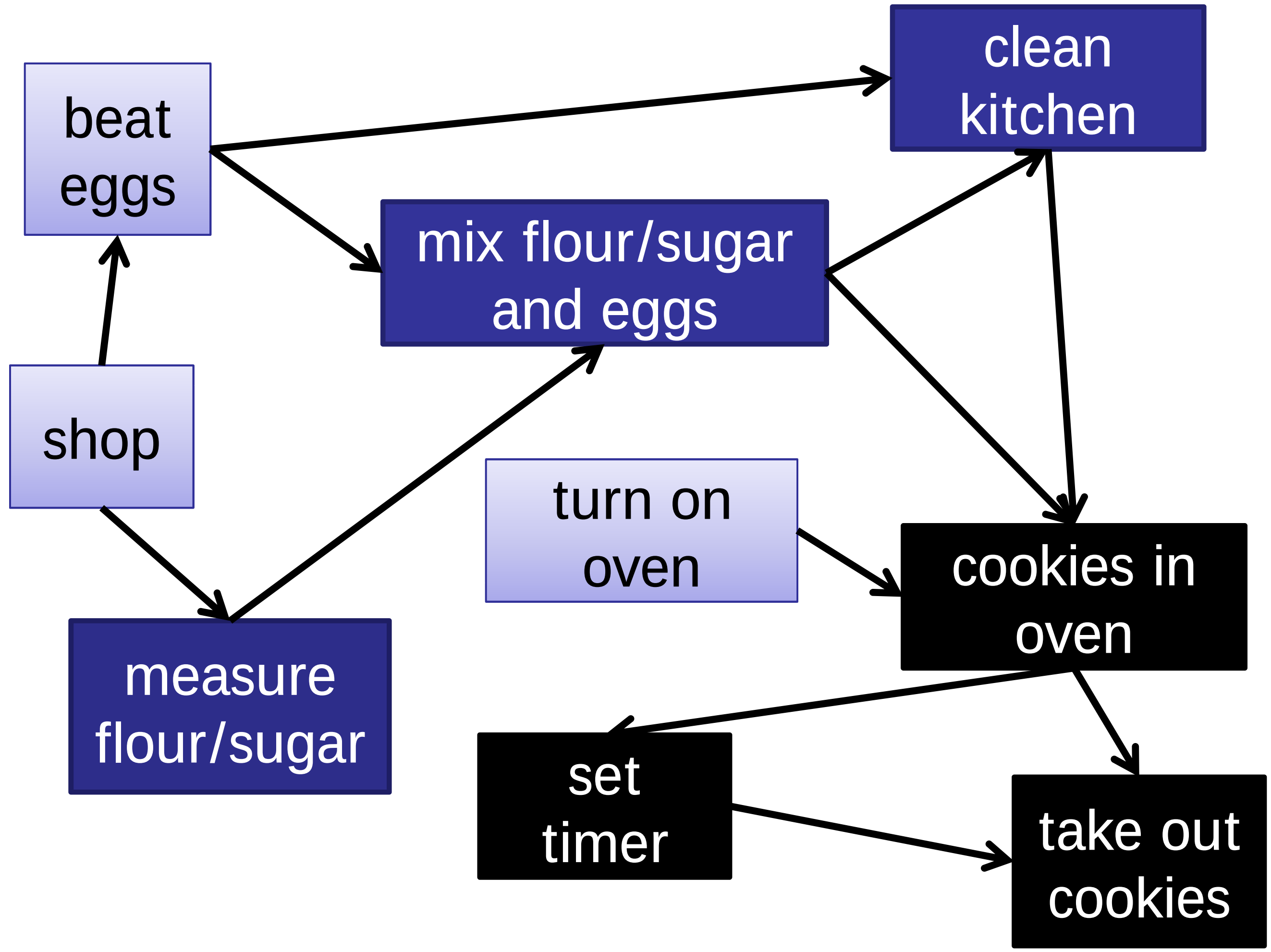
Post-Order Depth-First Search

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



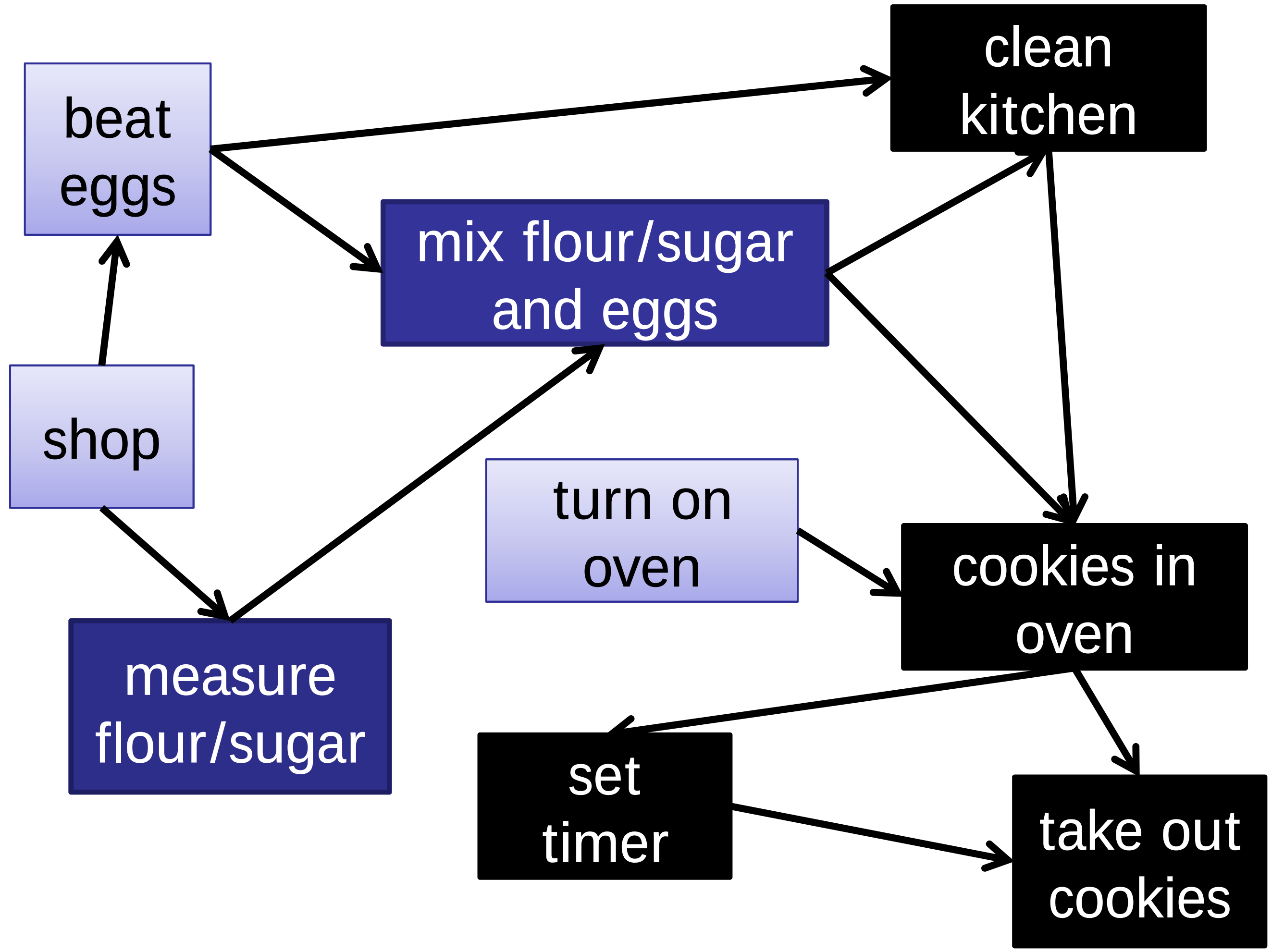
Post-Order Depth-First Search

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



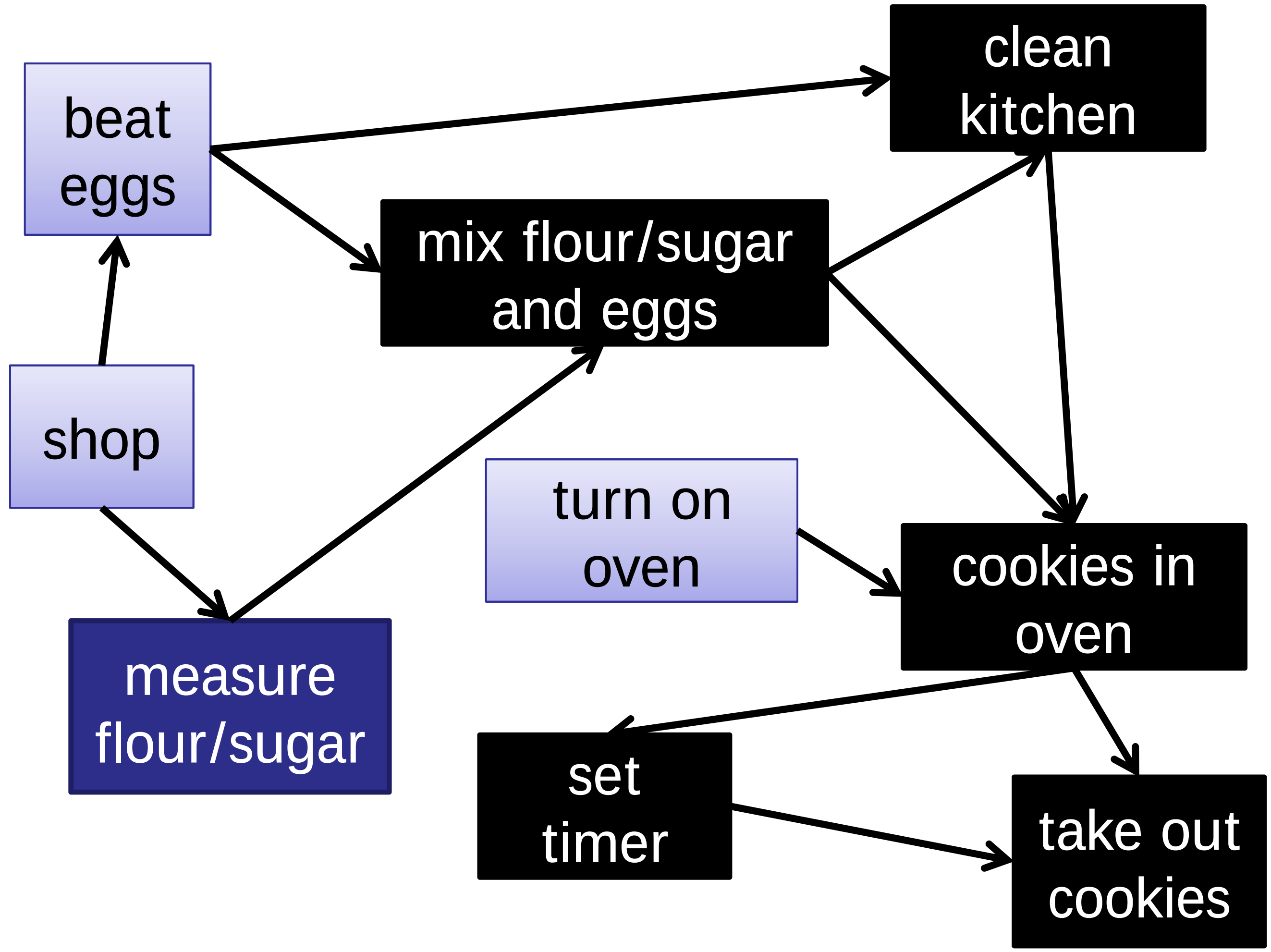
Post-Order Depth-First Search

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



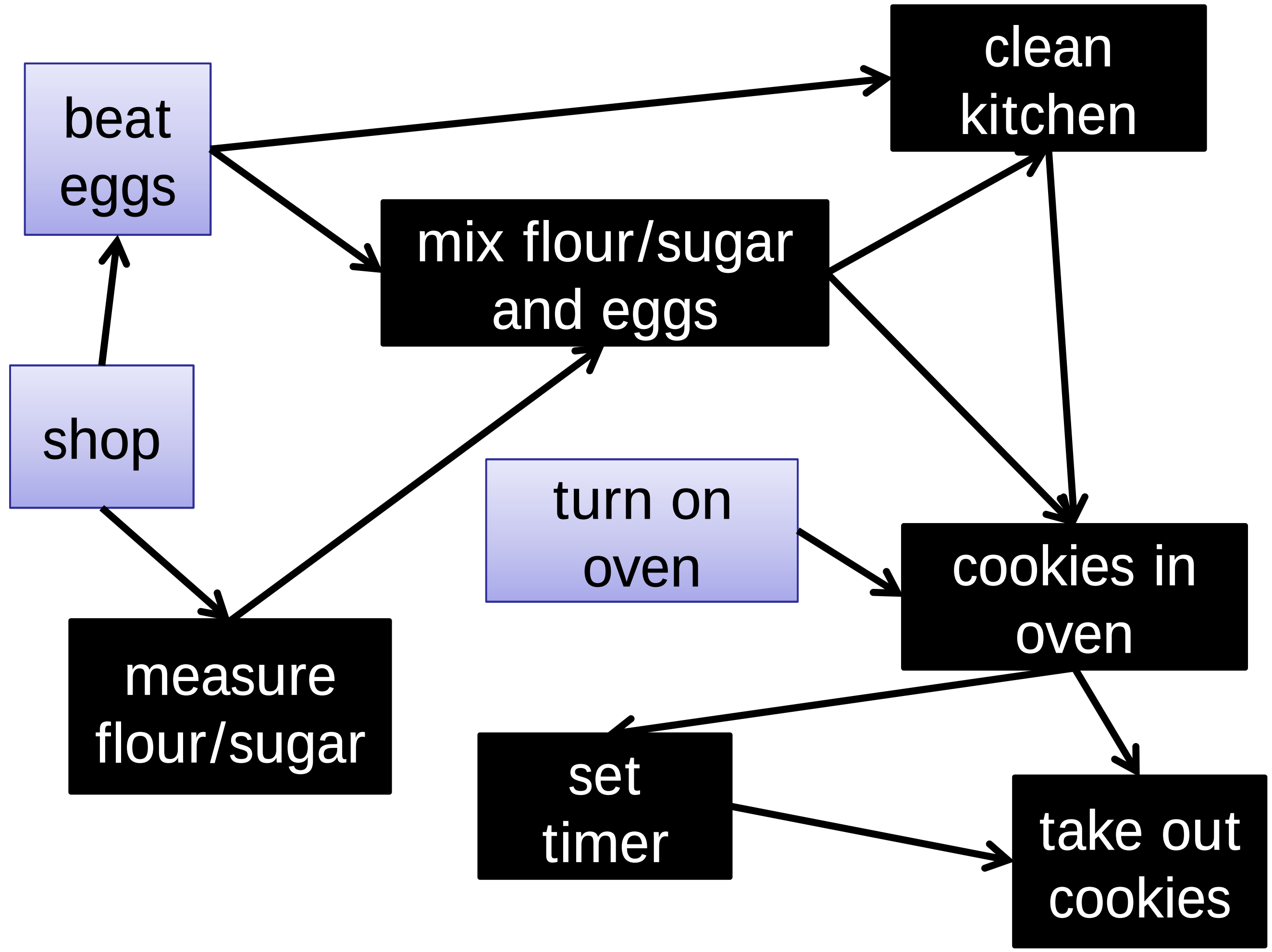
Post-Order Depth-First Search

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



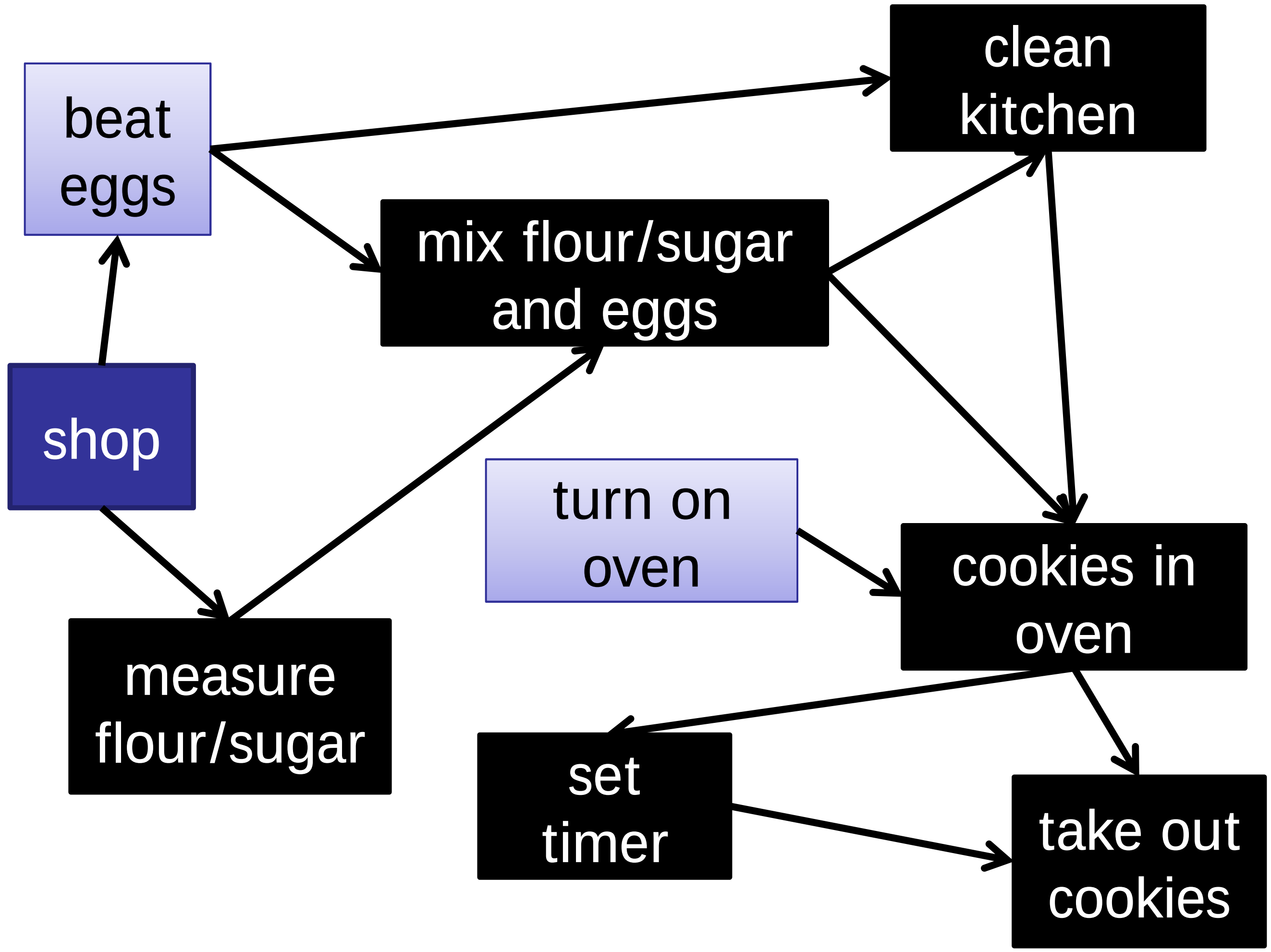
Post-Order Depth-First Search

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



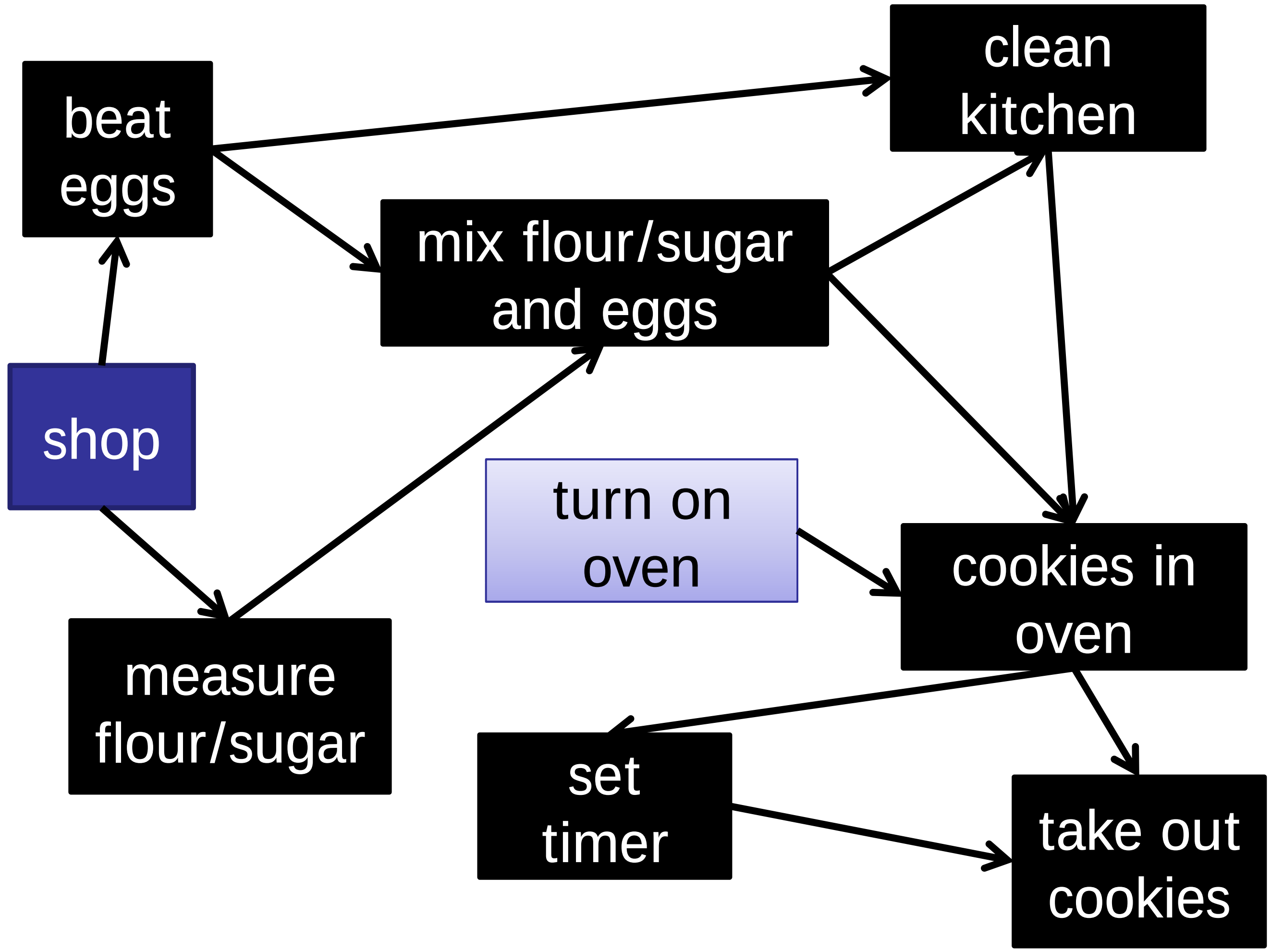
Post-Order Depth-First Search

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



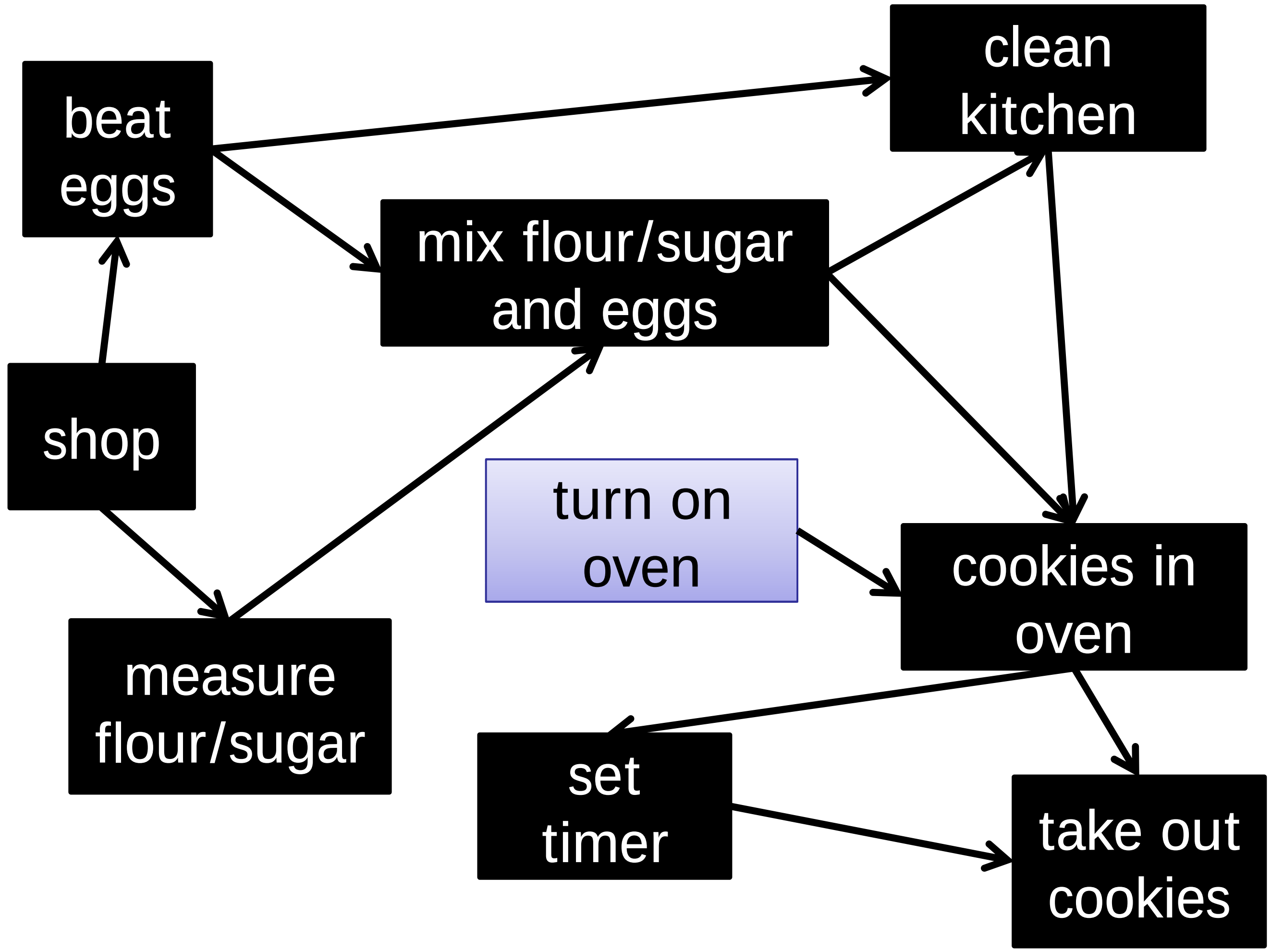
Post-Order Depth-First Search

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



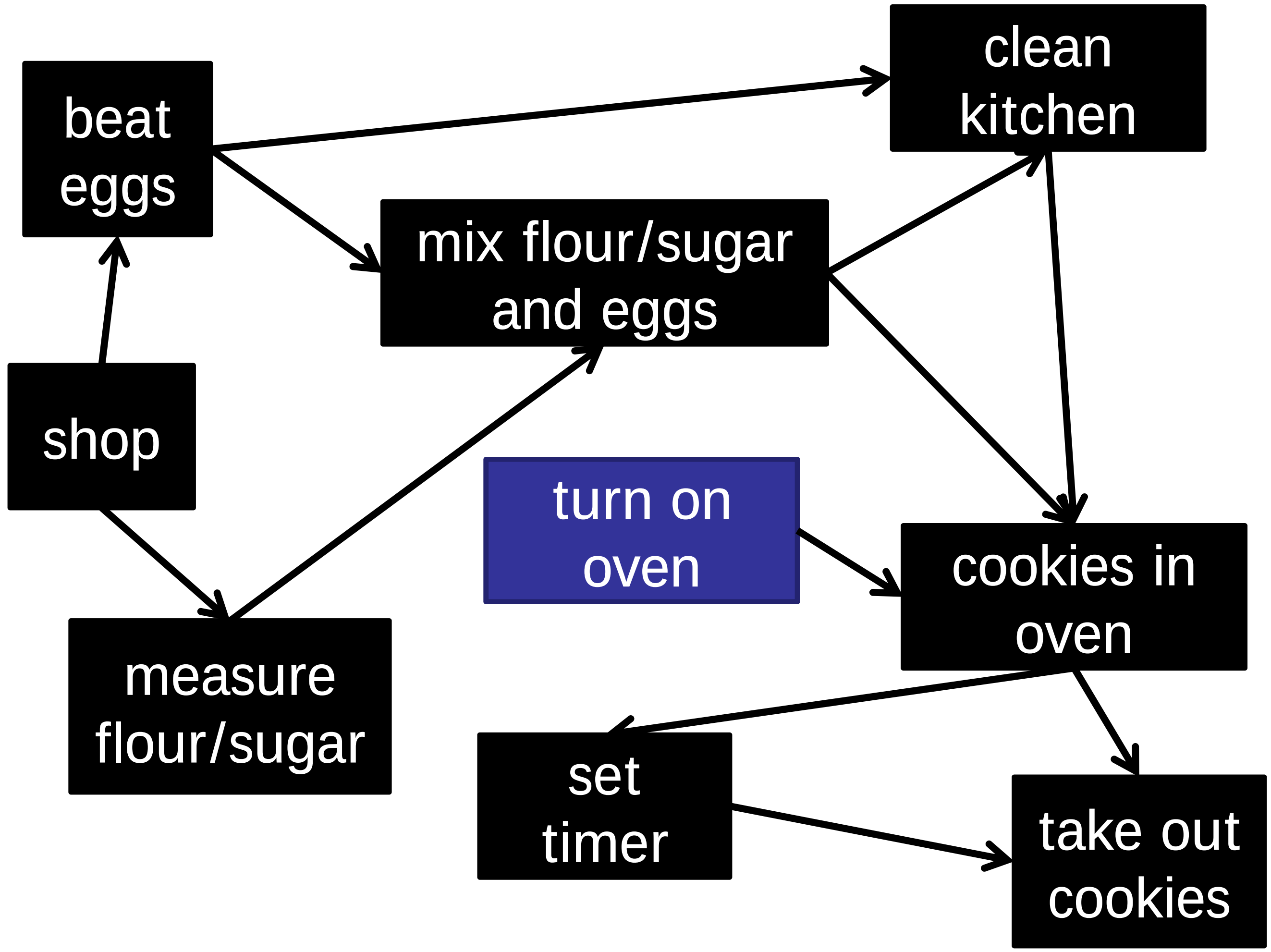
Post-Order Depth-First Search

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



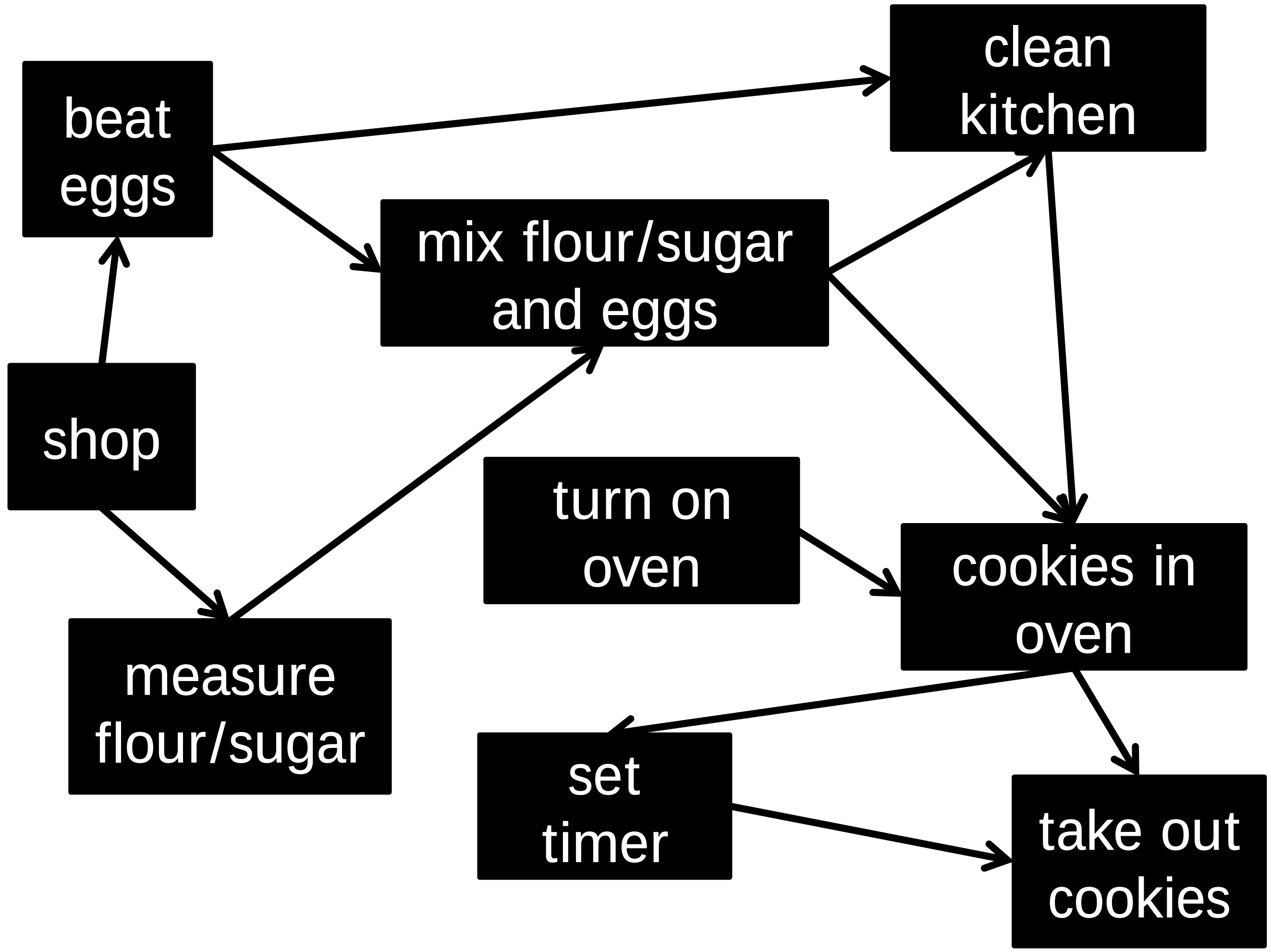
Post-Order Depth-First Search

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

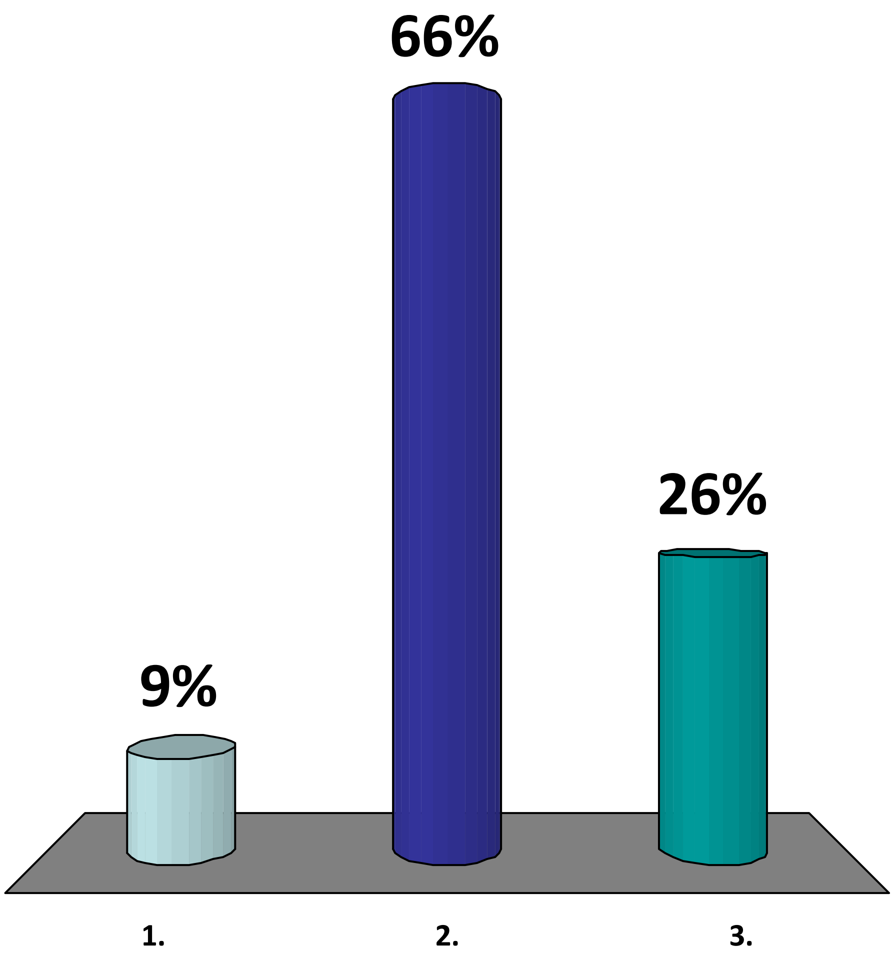
```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (Integer v : nodeList[startId].nbrList) {
     if (!visited[v]){
           visited[v] = true;
           DFS-visit(nodeList, visited, v);
           schedule.prepend(v);
     }
  }
}
```

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);
     }
  }
```

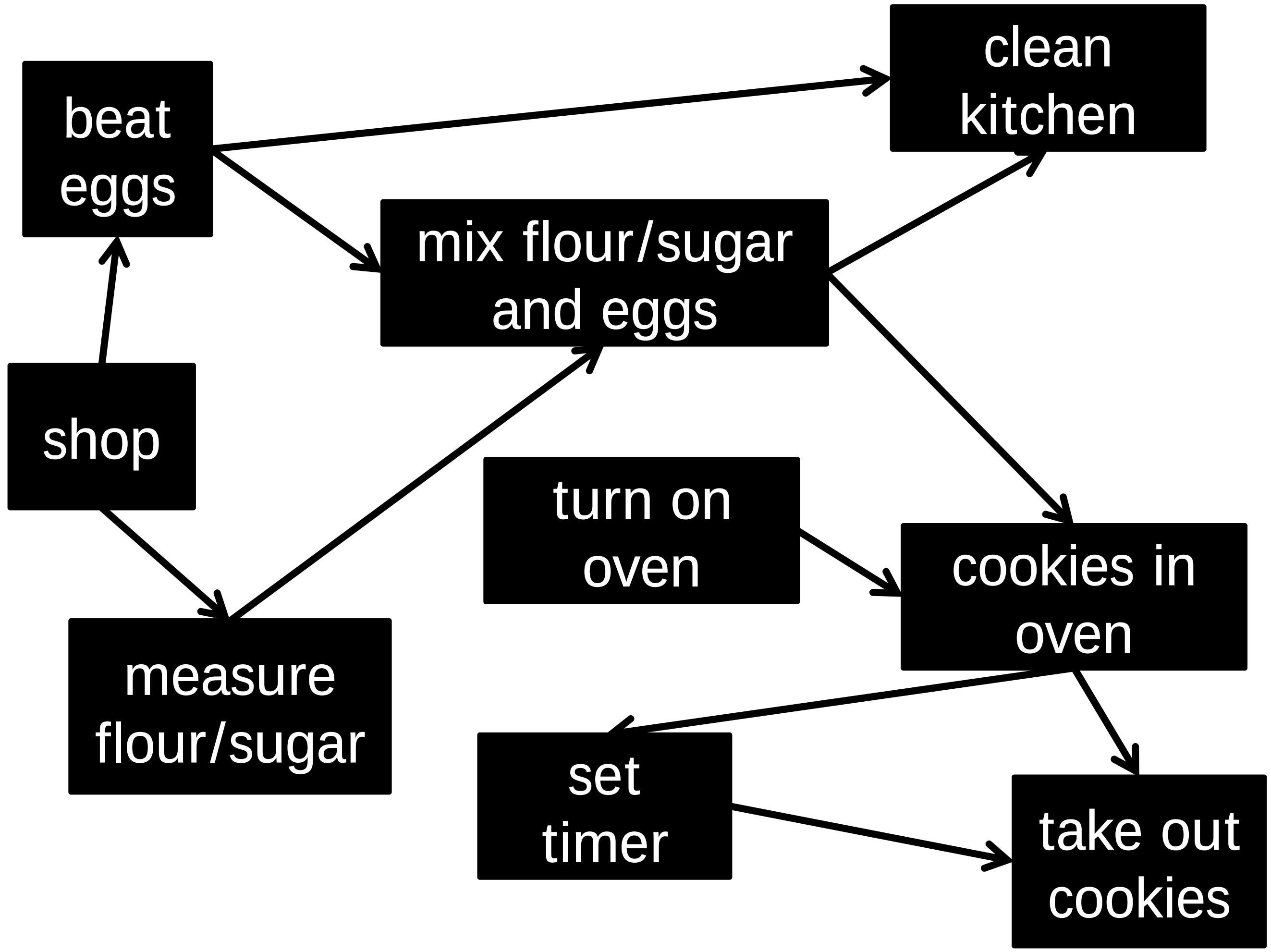
Is a topological ordering unique?

1. Yes
2. No
3. On Fridays.



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

Roadmap

Part I: Directed Graphs

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

Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford

Roadmap

Part I: Directed Graphs

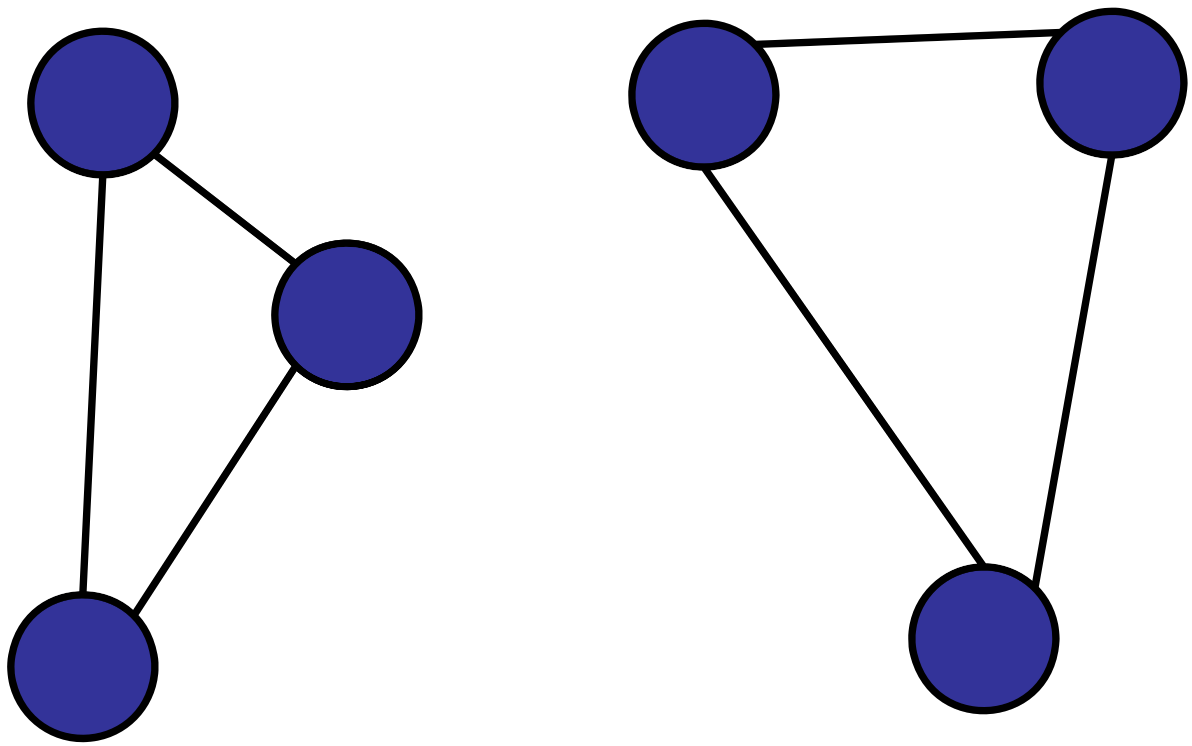
- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford

Connected Components

Undirected graphs

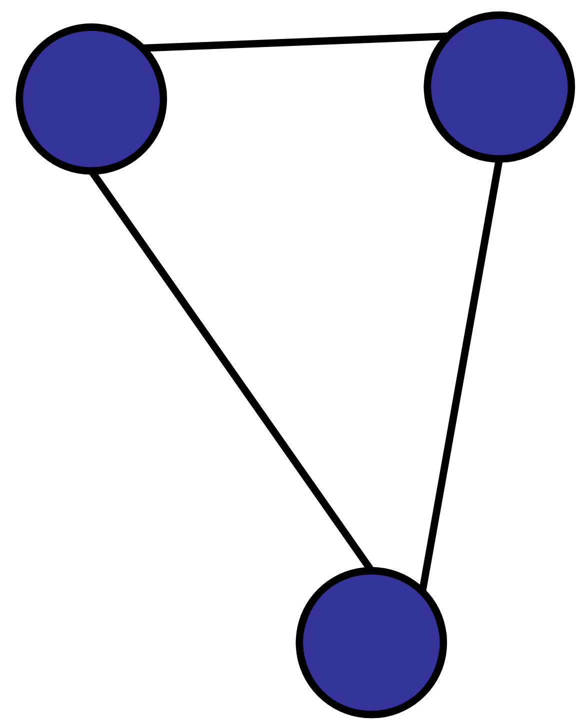


Two connected components

Connected Components

Undirected graphs

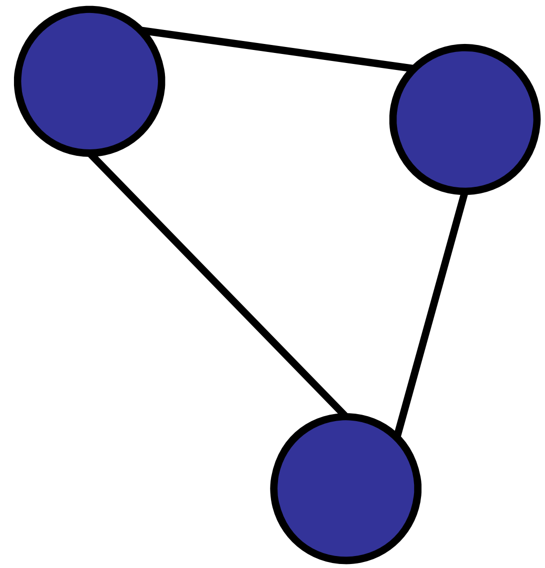
Vertex v and w are in the same connected component if and only if there is a path from v to w.



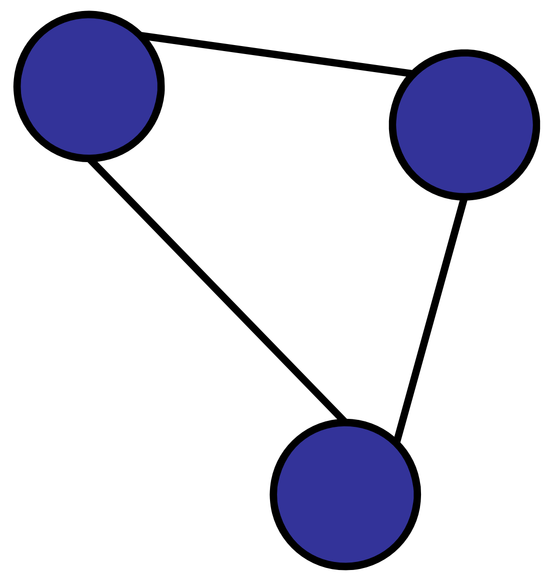
Connected Components

Undirected graphs

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

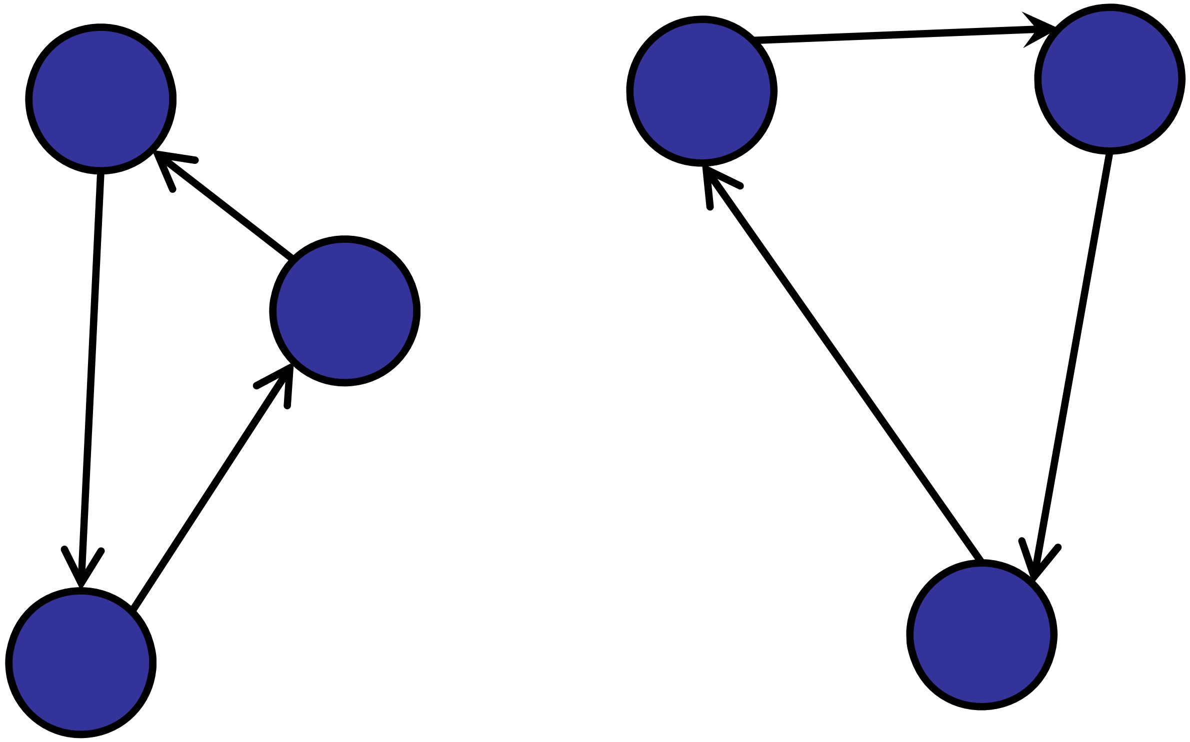


There is a set $\{v_1, v_2, \ldots, v_k\}$ where there is no path from any v_i to v_j if and only if there are k connected components.



Connected Components

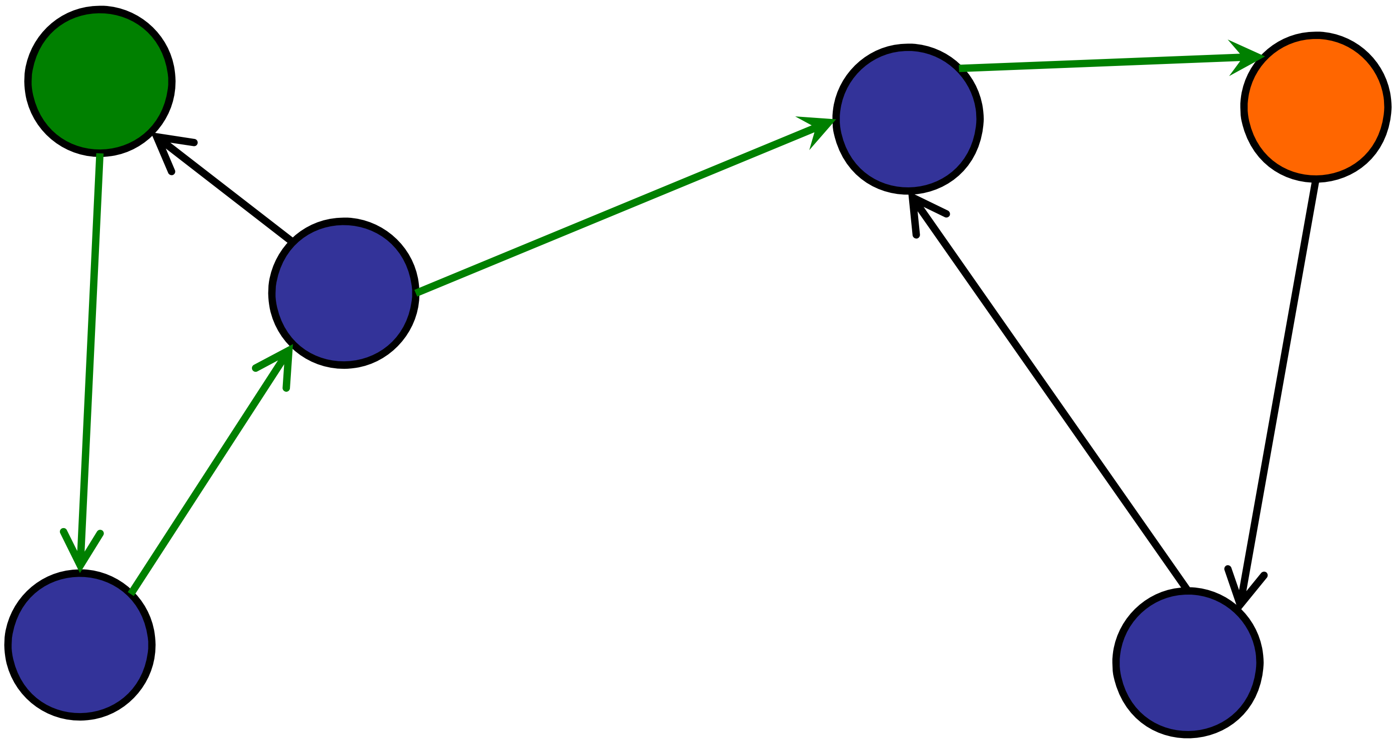
Directed graphs



Two connected components

Connected Components

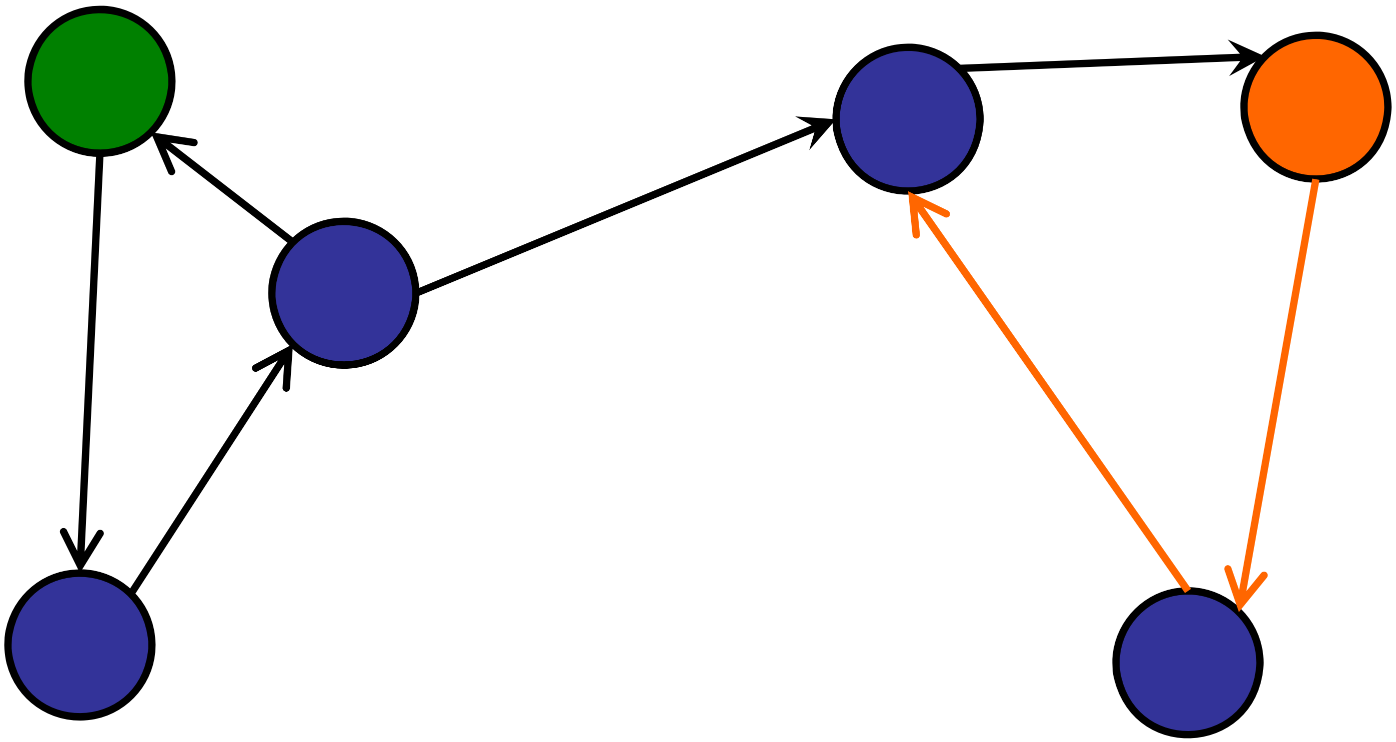
Directed graphs



Two connected components??

Connected Components

Directed graphs



Two connected components??

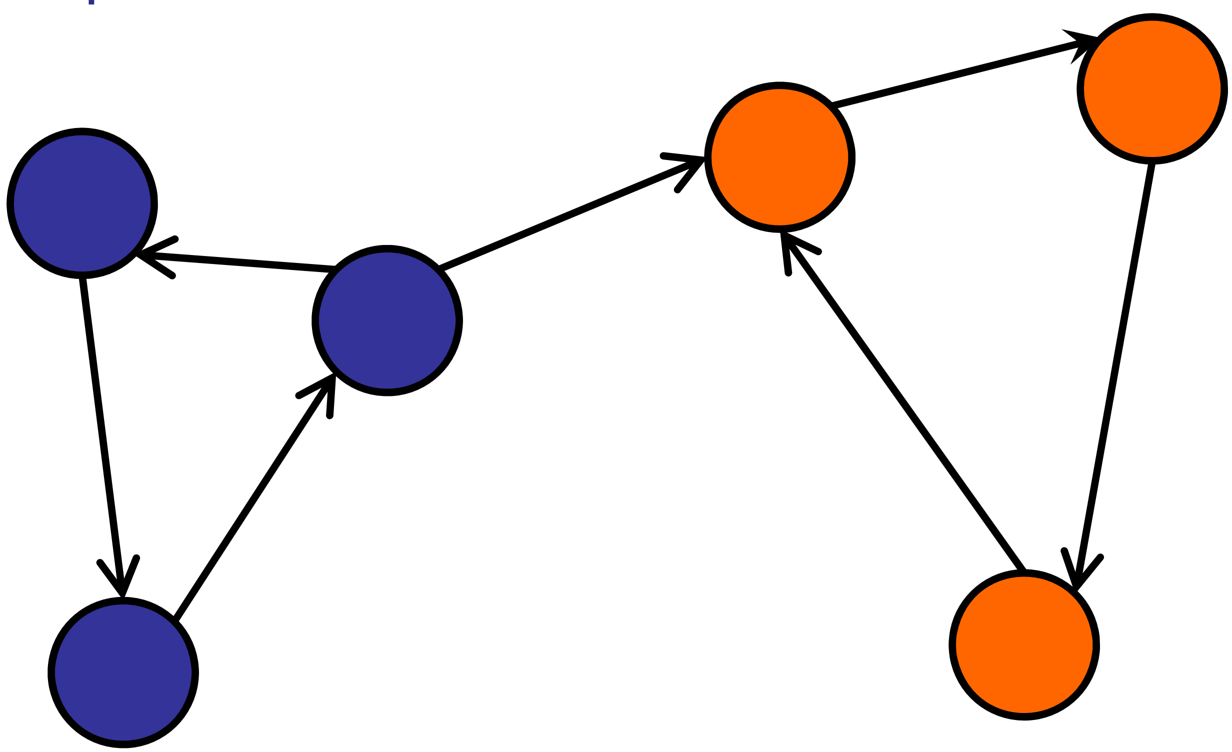
Connected Components

Strongly connected component

For every vertex v and w:

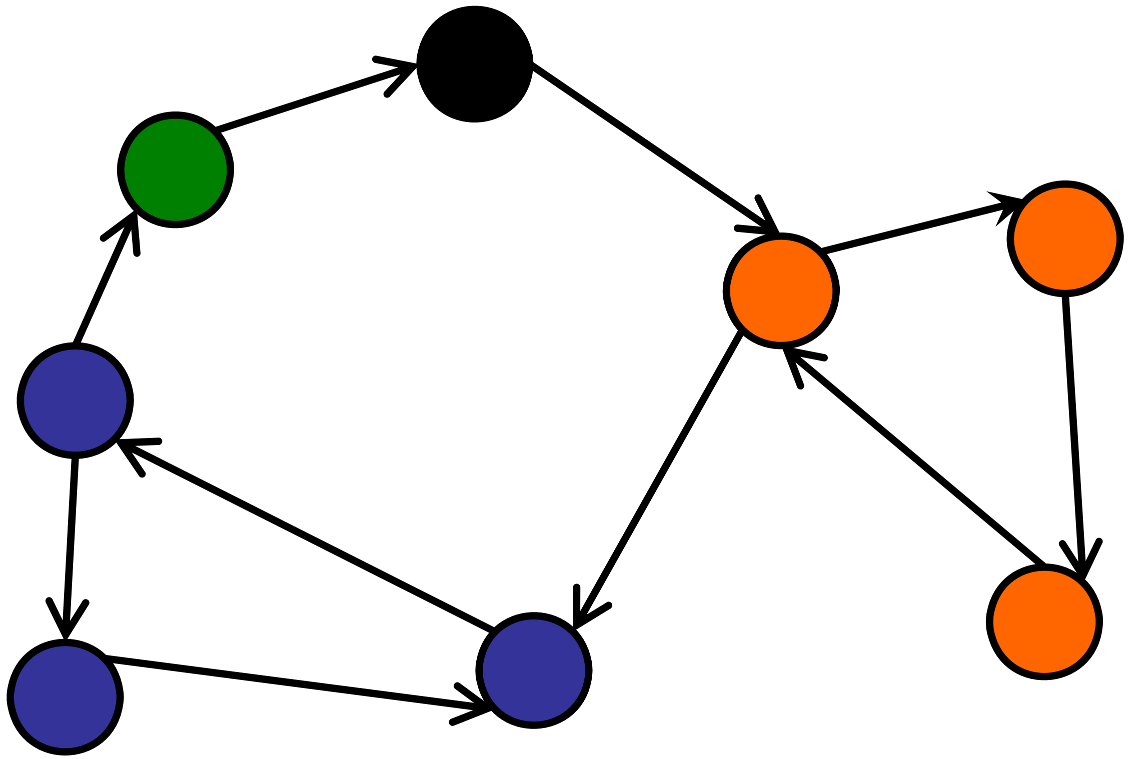
–There is a path from v to w.

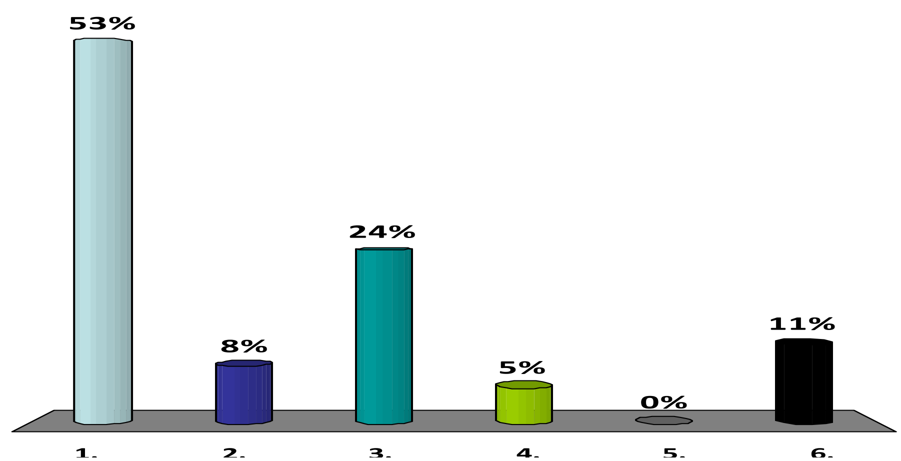
– There is a path from w to v.



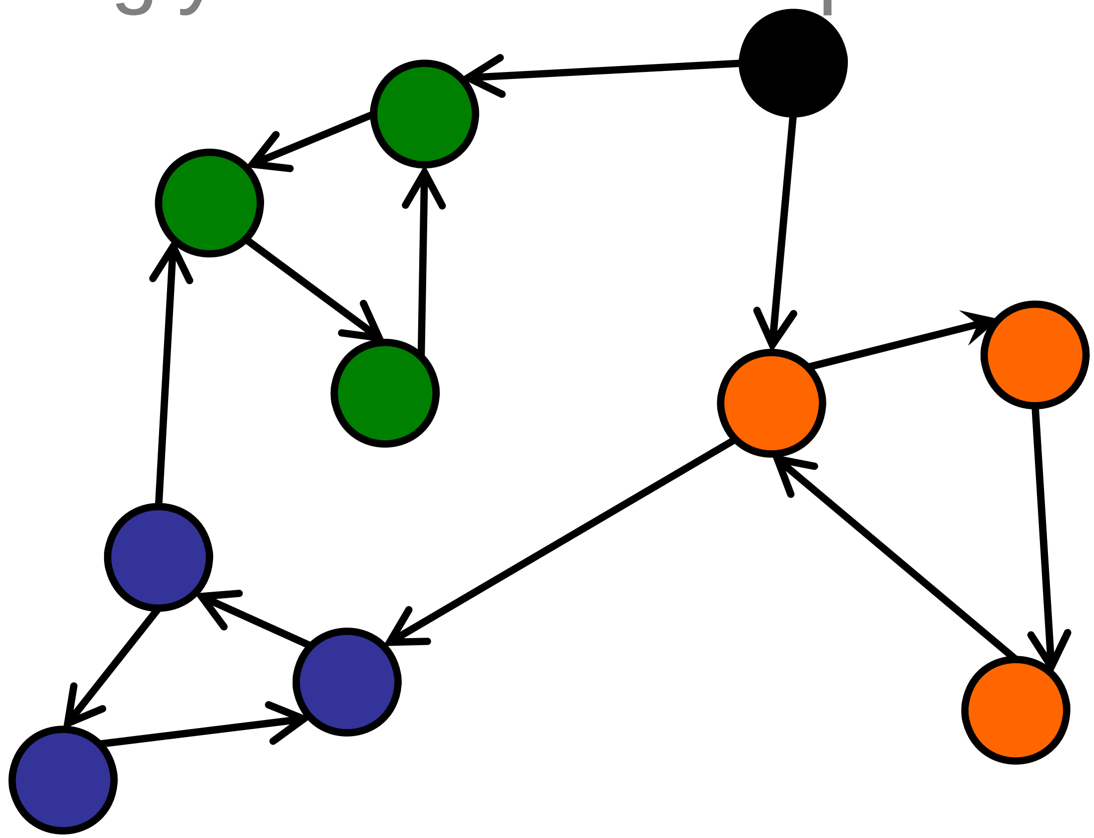
How many strongly connected components?

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

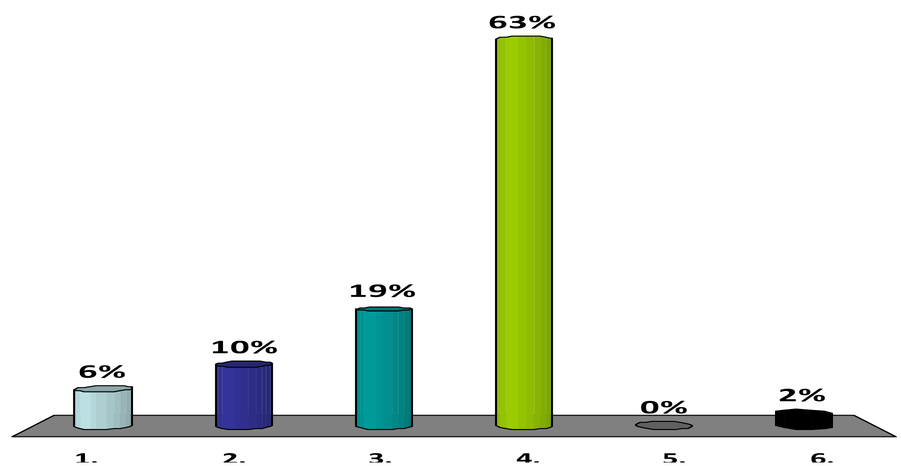




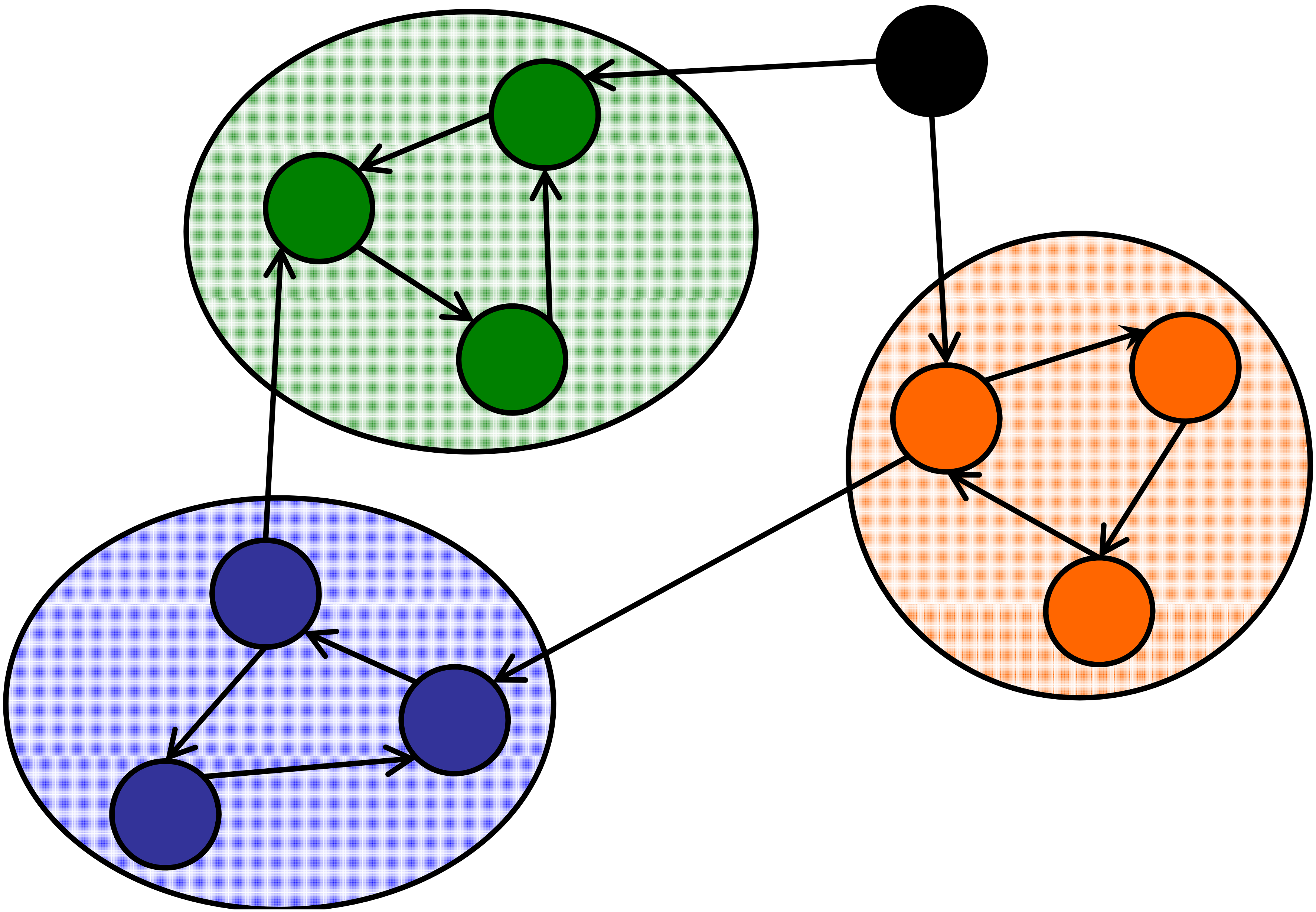
How many strongly connected components?



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

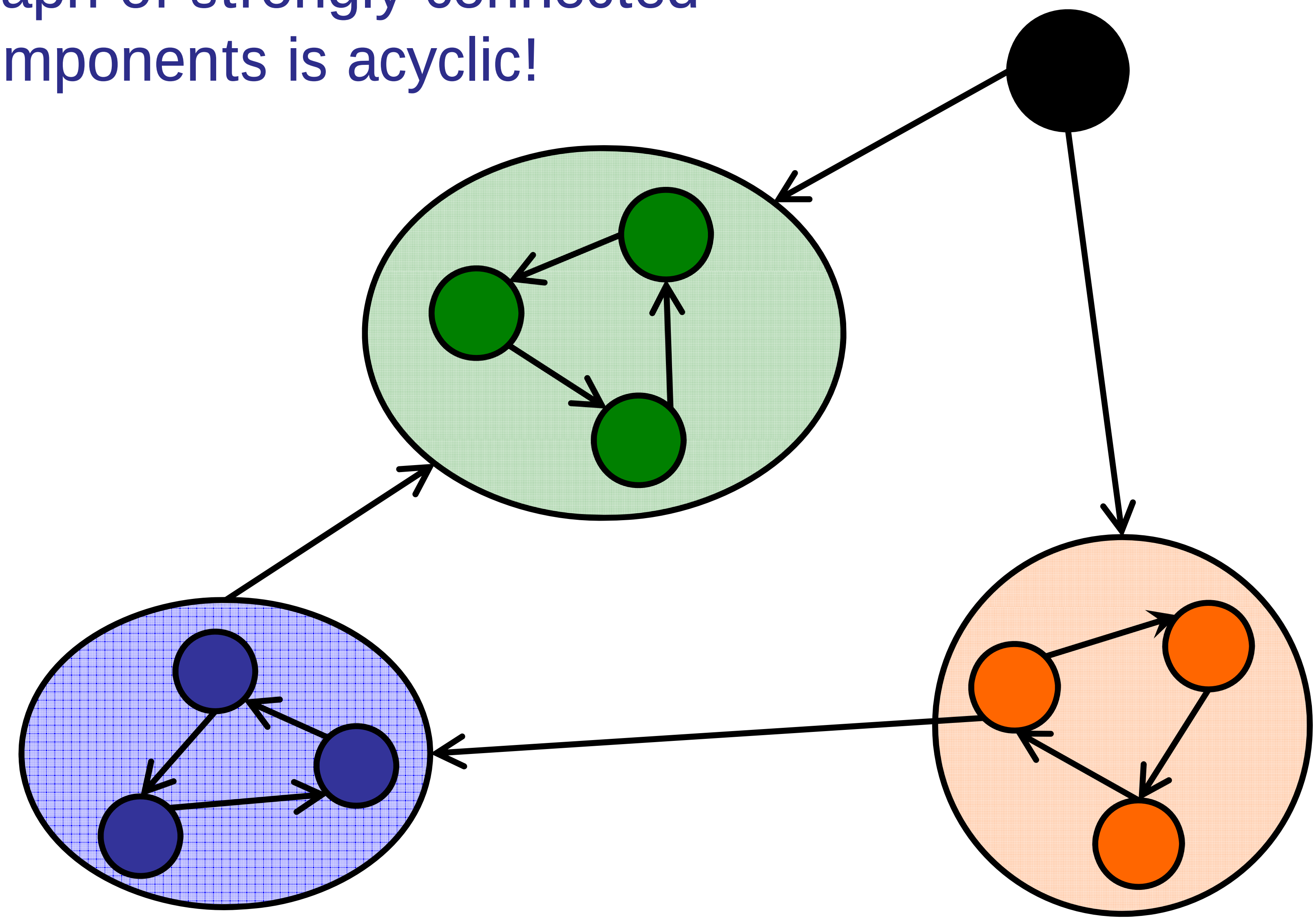


Connected Components



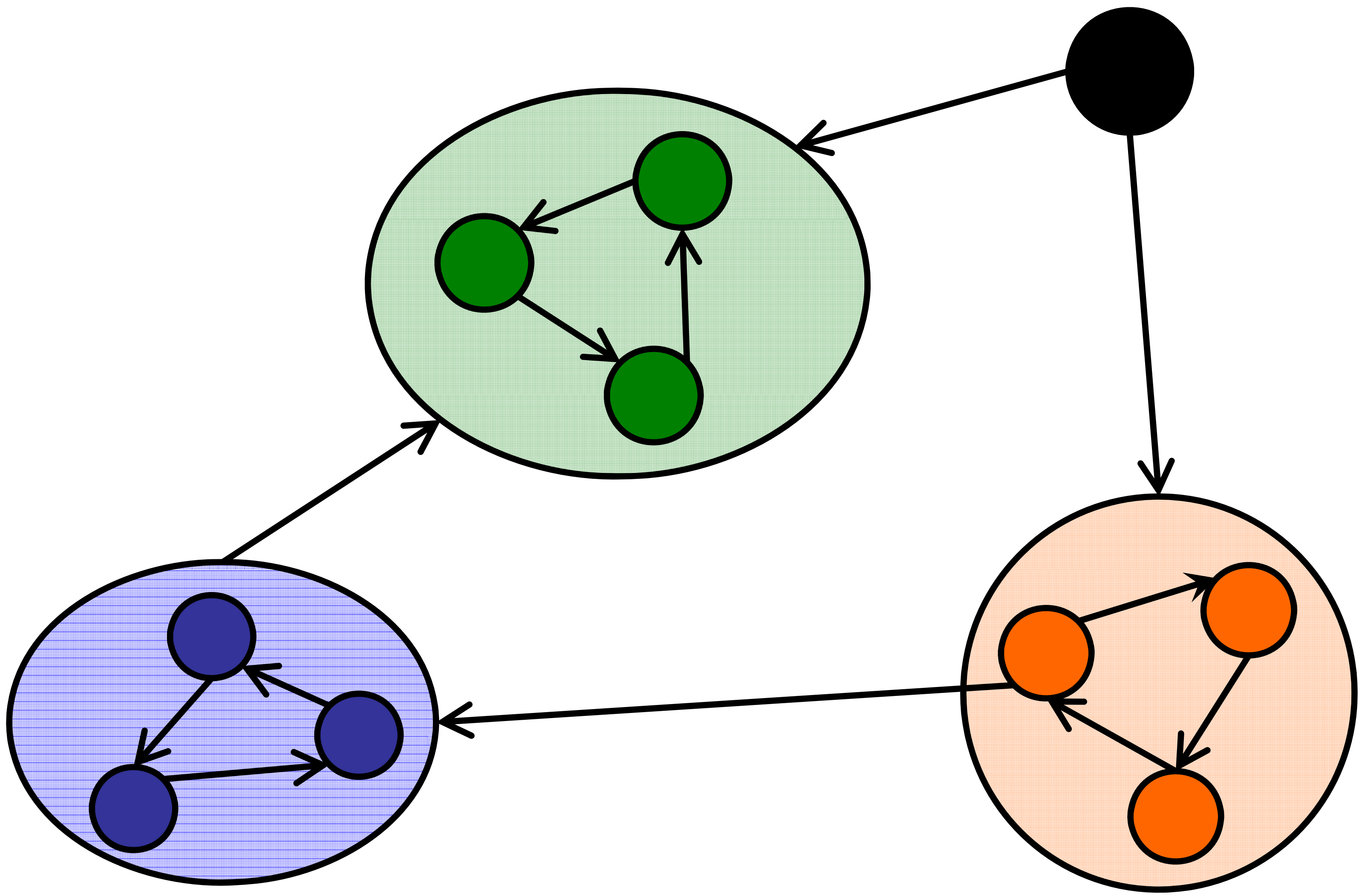
Connected Components

Graph of strongly connected components is acyclic!



Connected Components

Challenge: find all strongly connected components.



Roadmap

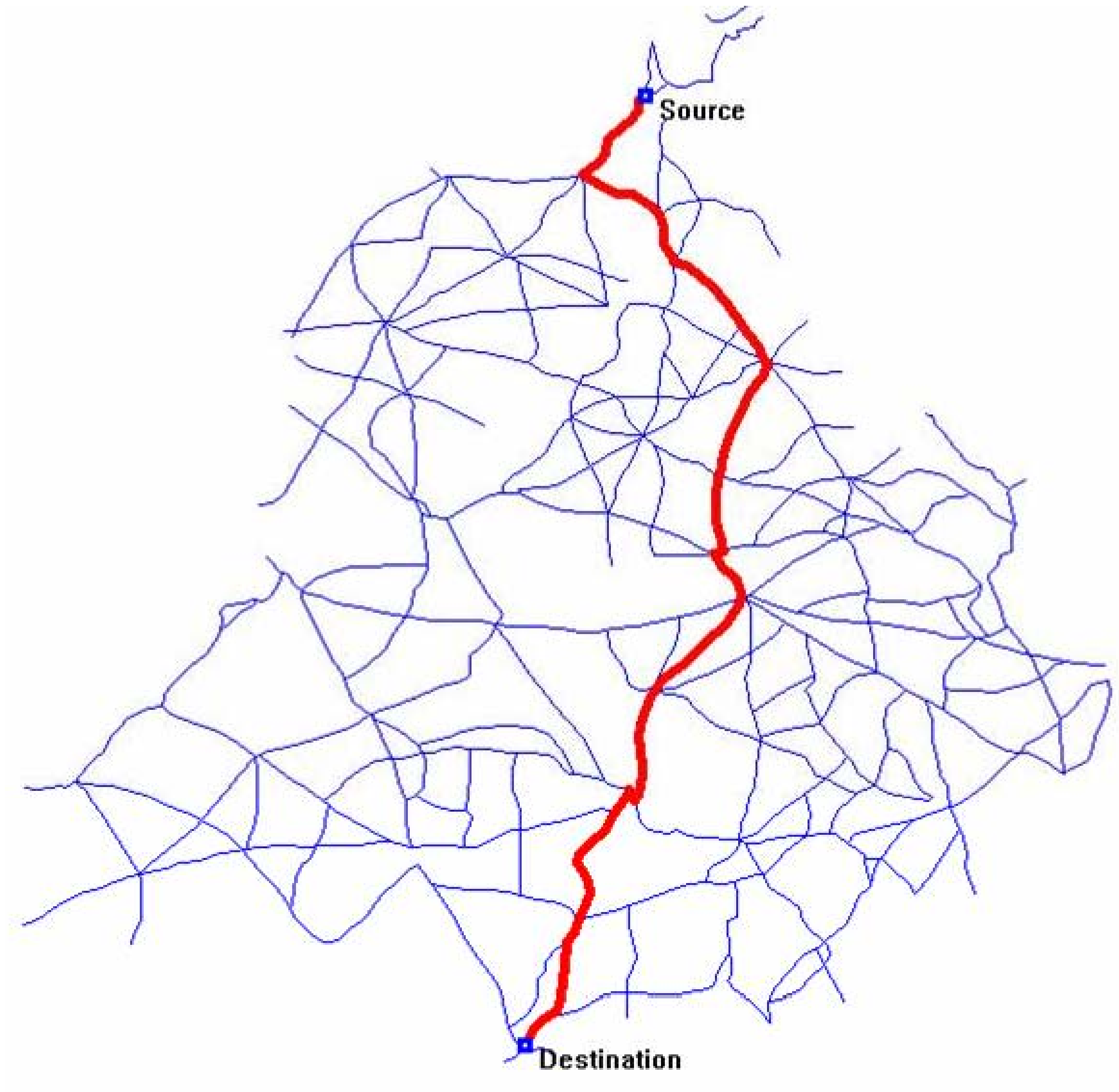
Part I: Directed Graphs

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

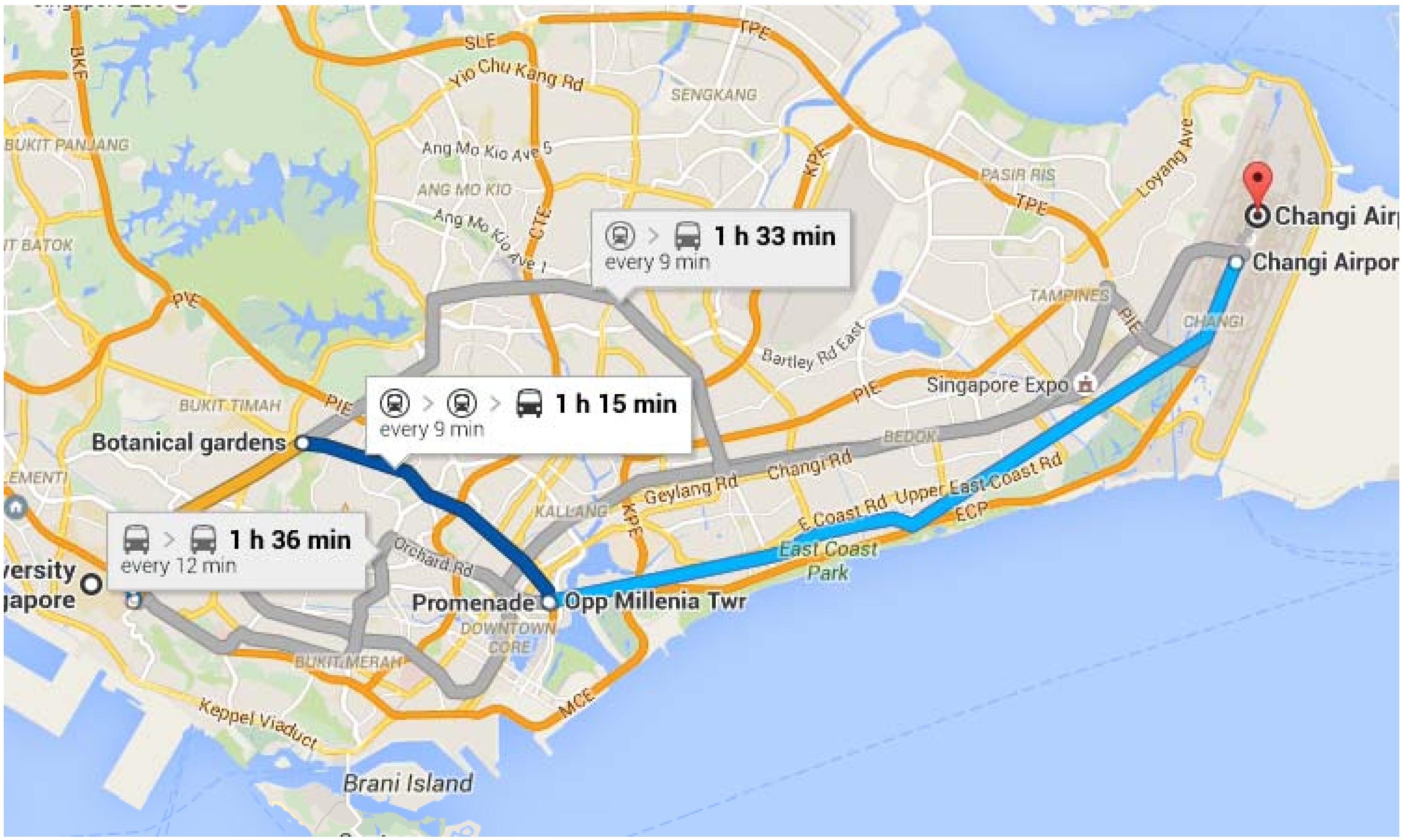
Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford

SHORTEST PATHS

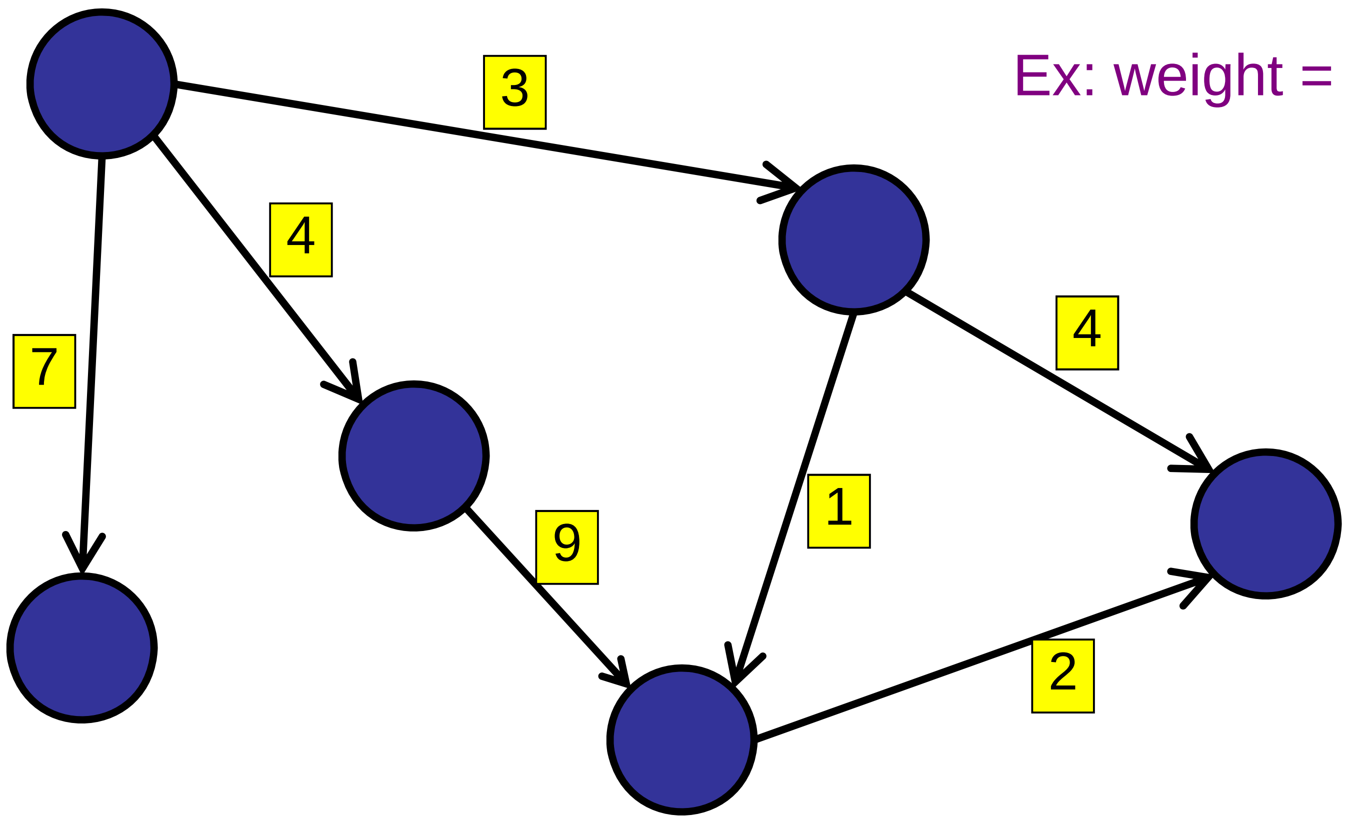


SHORTEST PATHS



Weighted Graphs

Edge weights: w(e) : E ➔ R

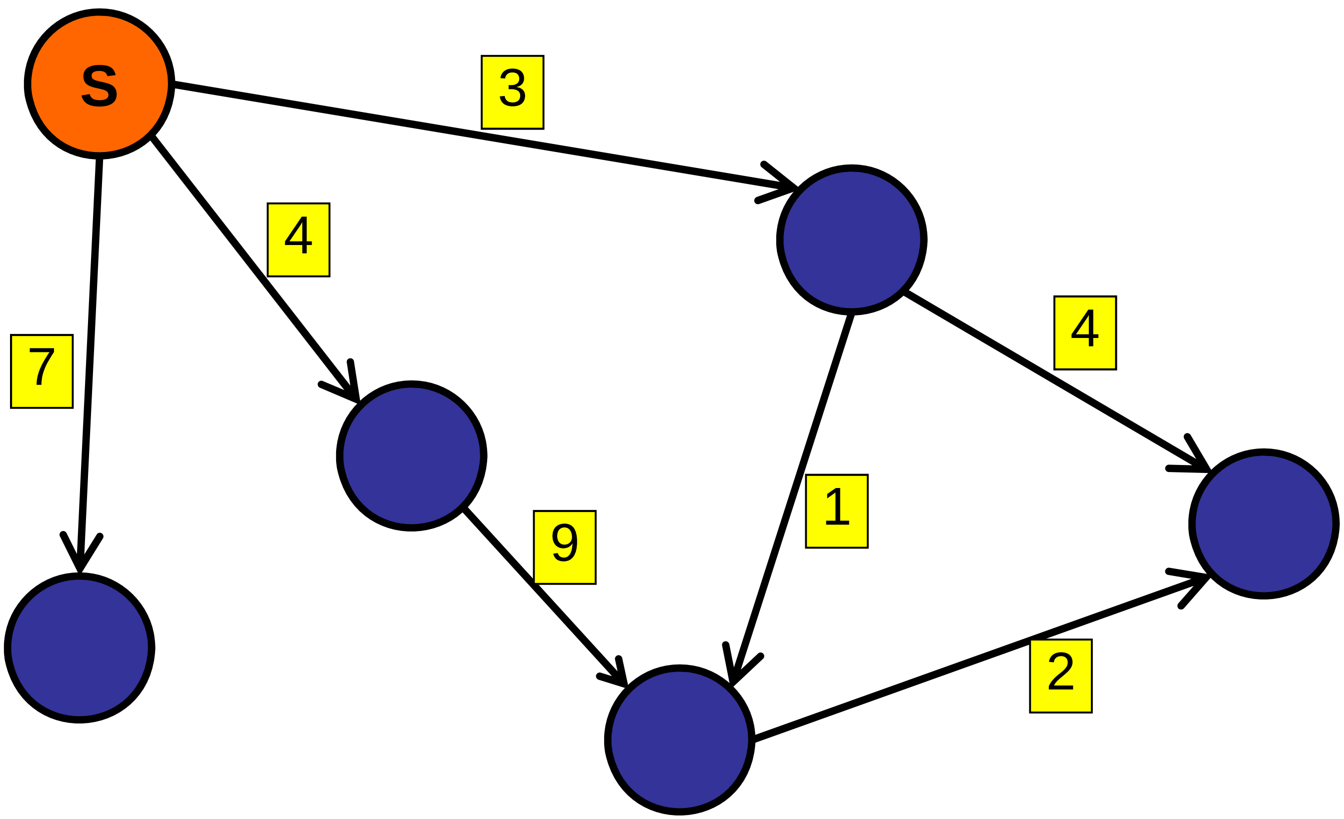


Ex: weight = distance

Adjacency list: stores weights with edge in NbrList

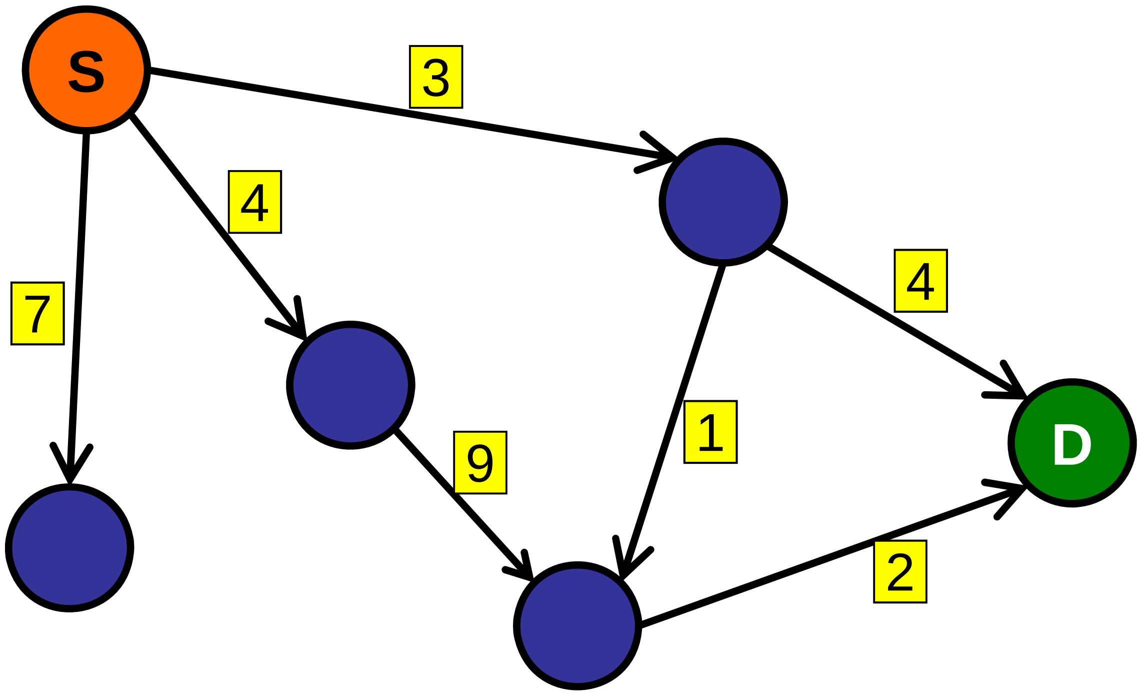
Shortest Paths

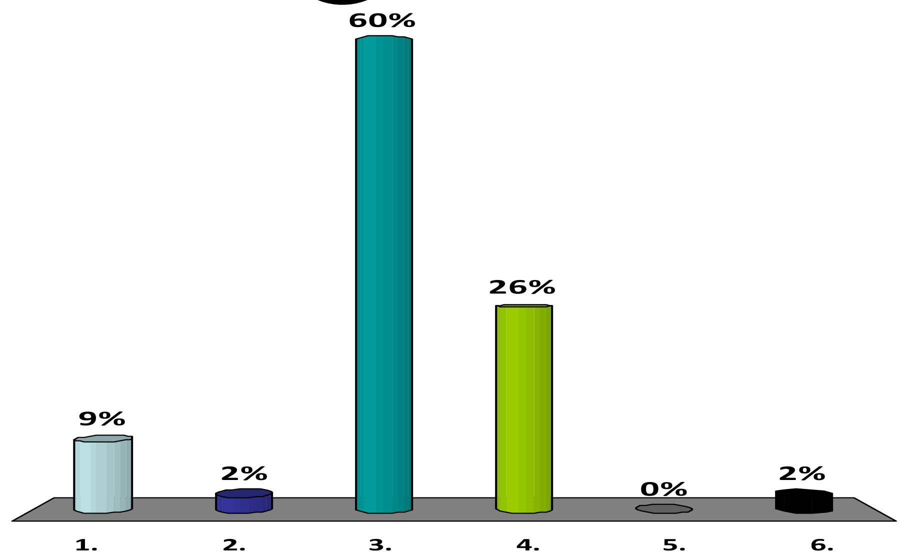
Distance from source?



What is the distance from S to D?

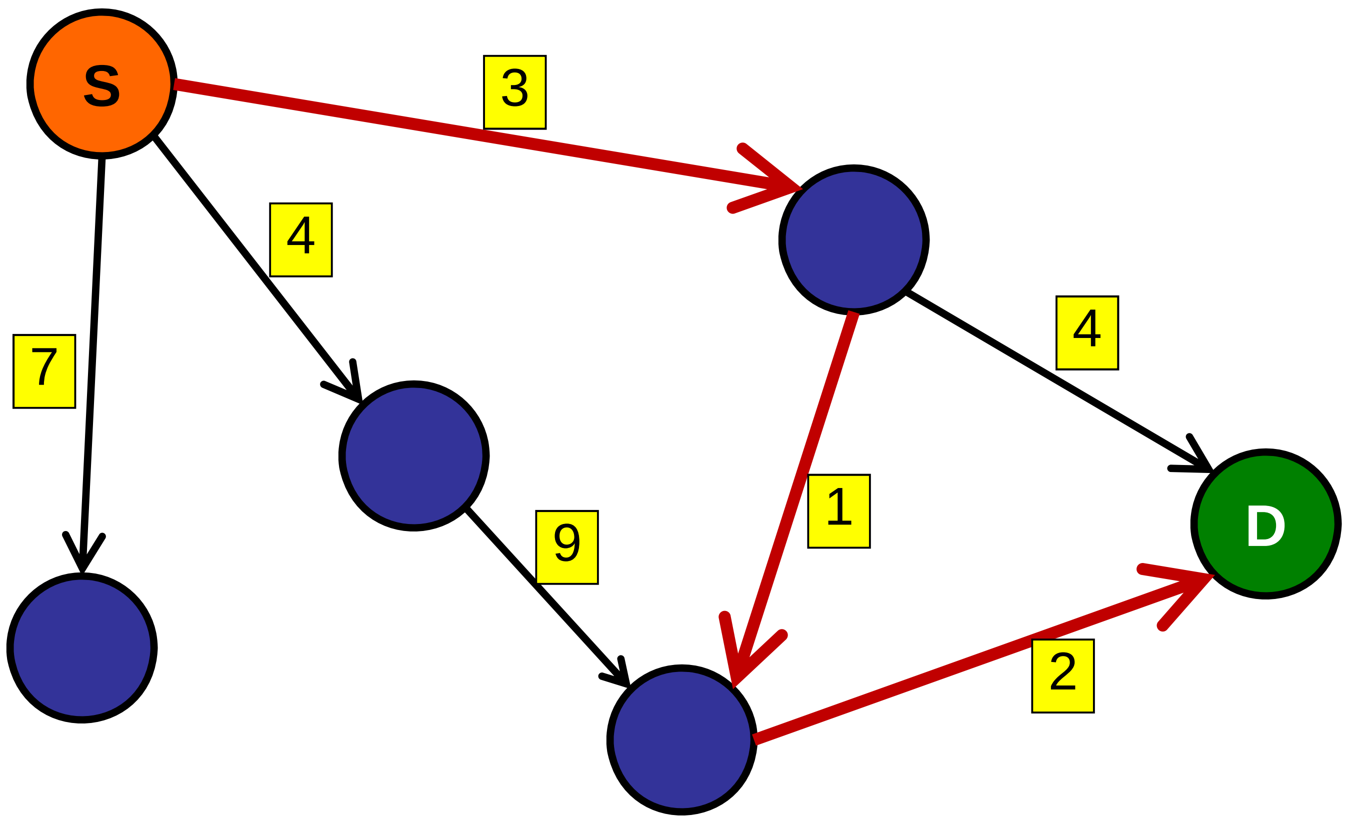
1. 2
2. 4
3. 6
4. 7
5. 9
6. Infinite





Shortest Paths

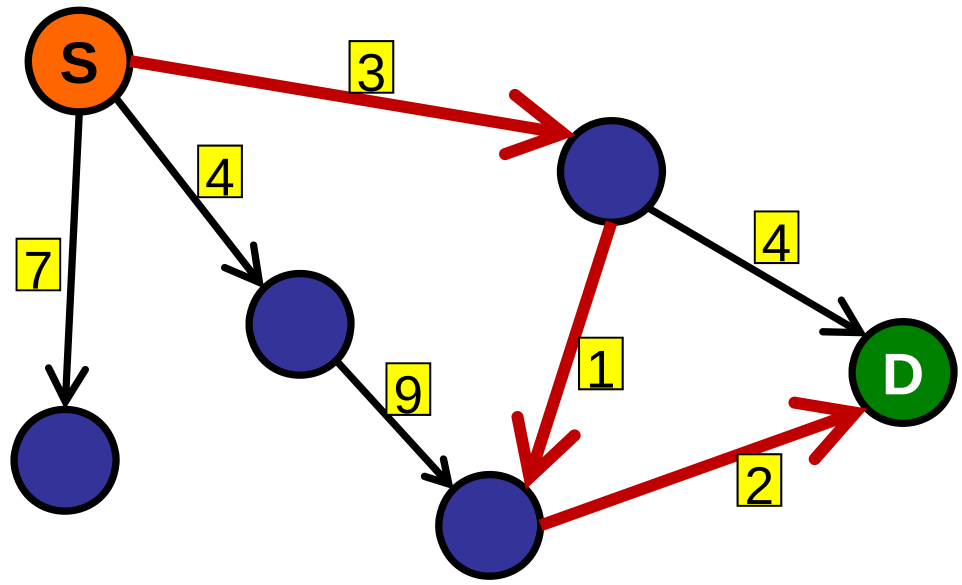
Distance from source?



Shortest Paths

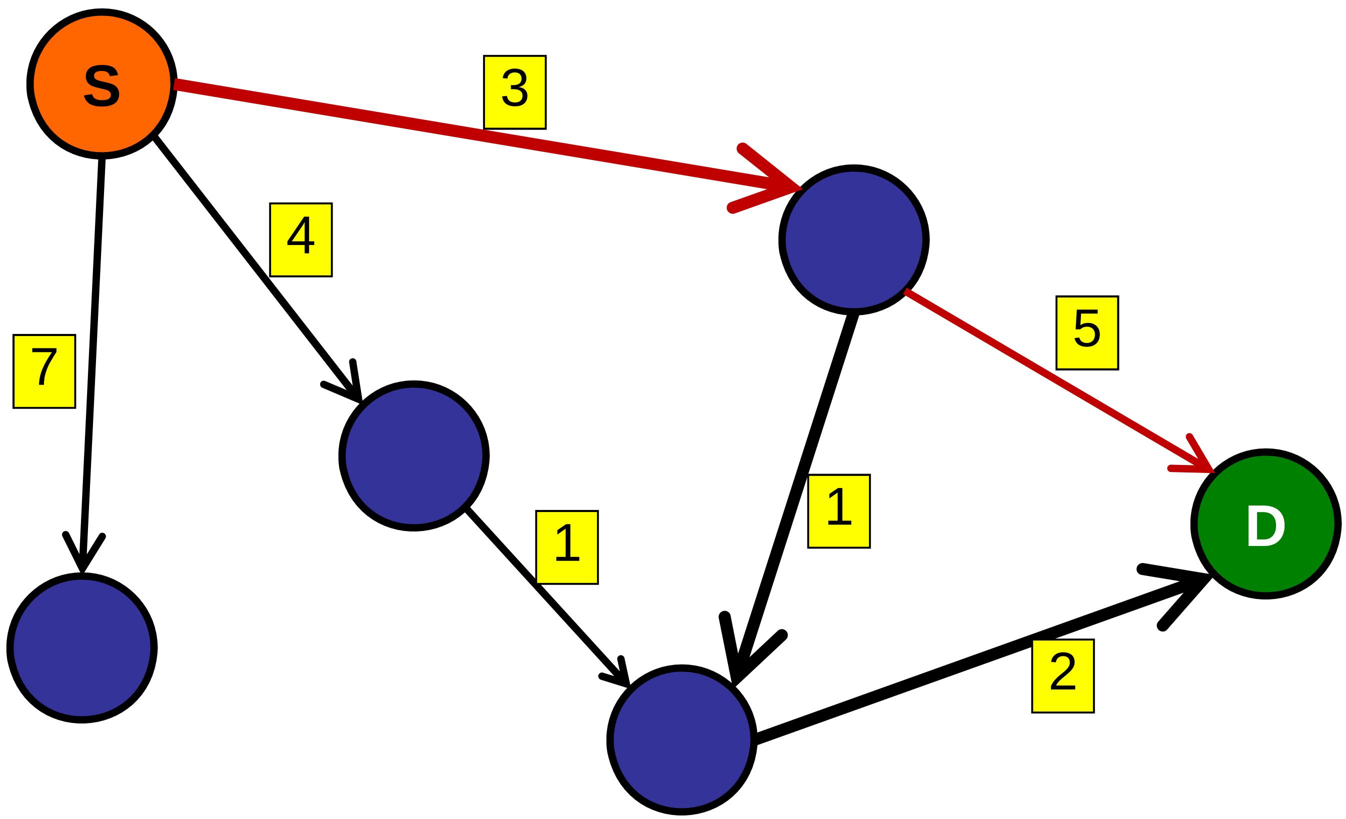
Questions:

- How far is it from S to D?
- What is the shortest path from S to D?
- Find the shortest path from S to every node.
- Find the shortest path between every pair of nodes.



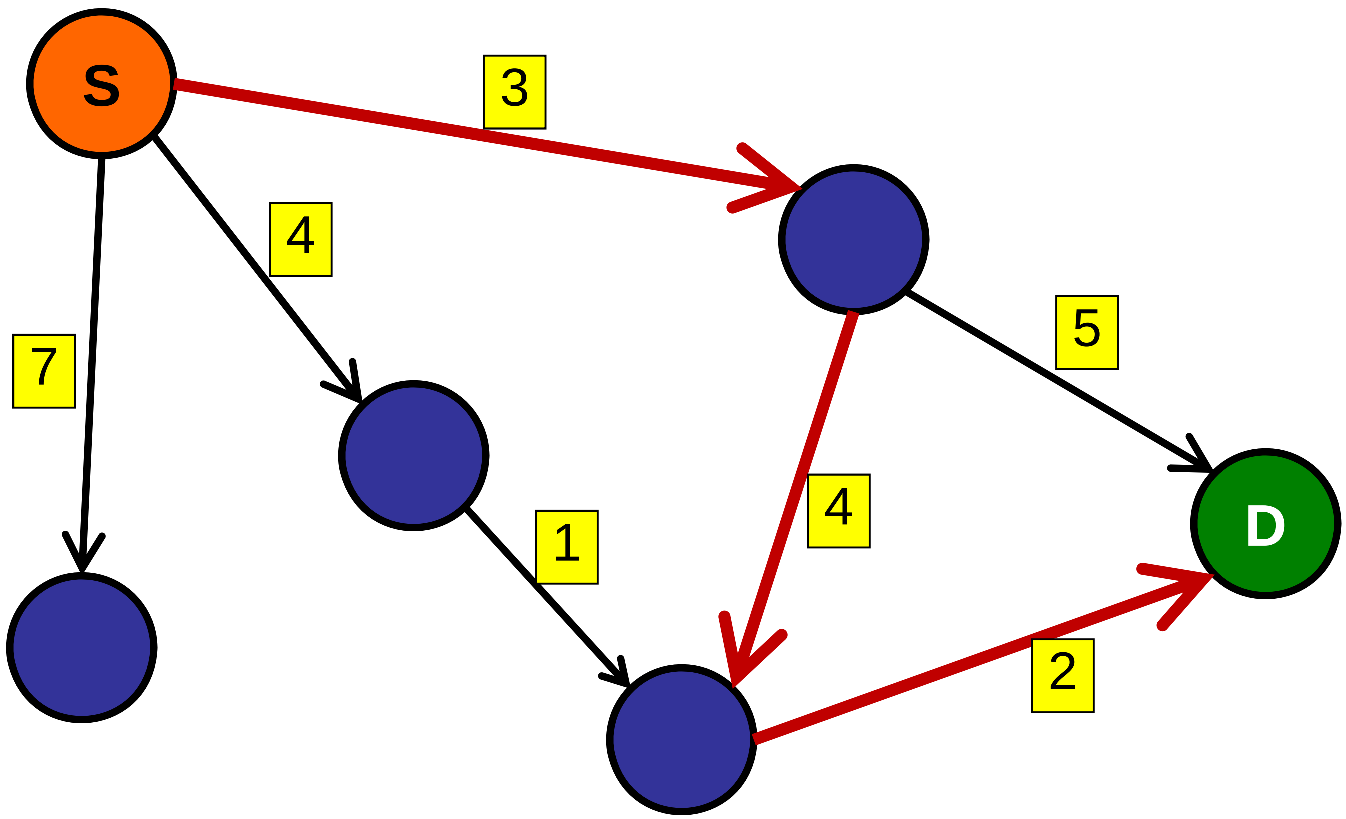
Shortest Paths

Common mistake: “Why can’t I use BFS?”



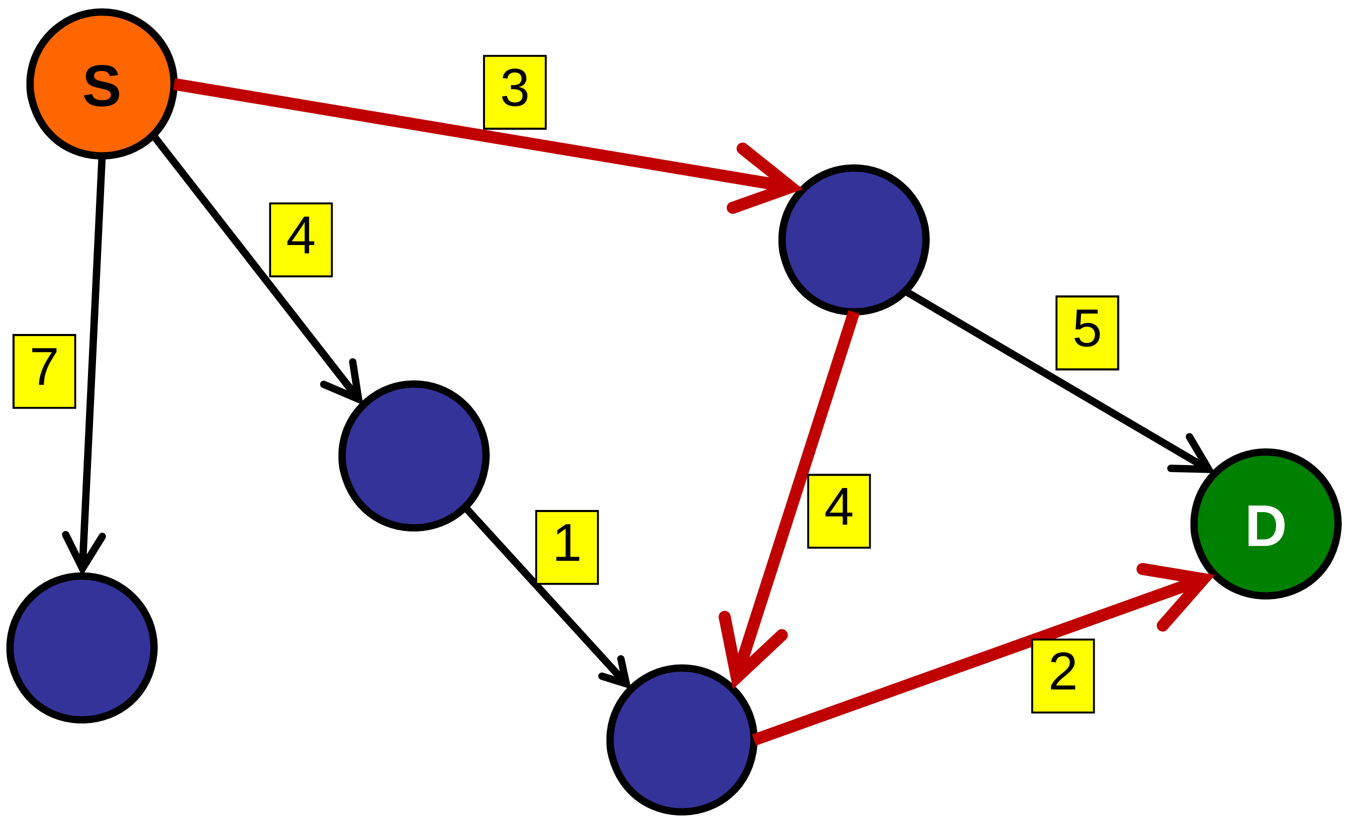
Shortest Paths

Common mistake: “Why can’t I use BFS?”



Shortest Paths

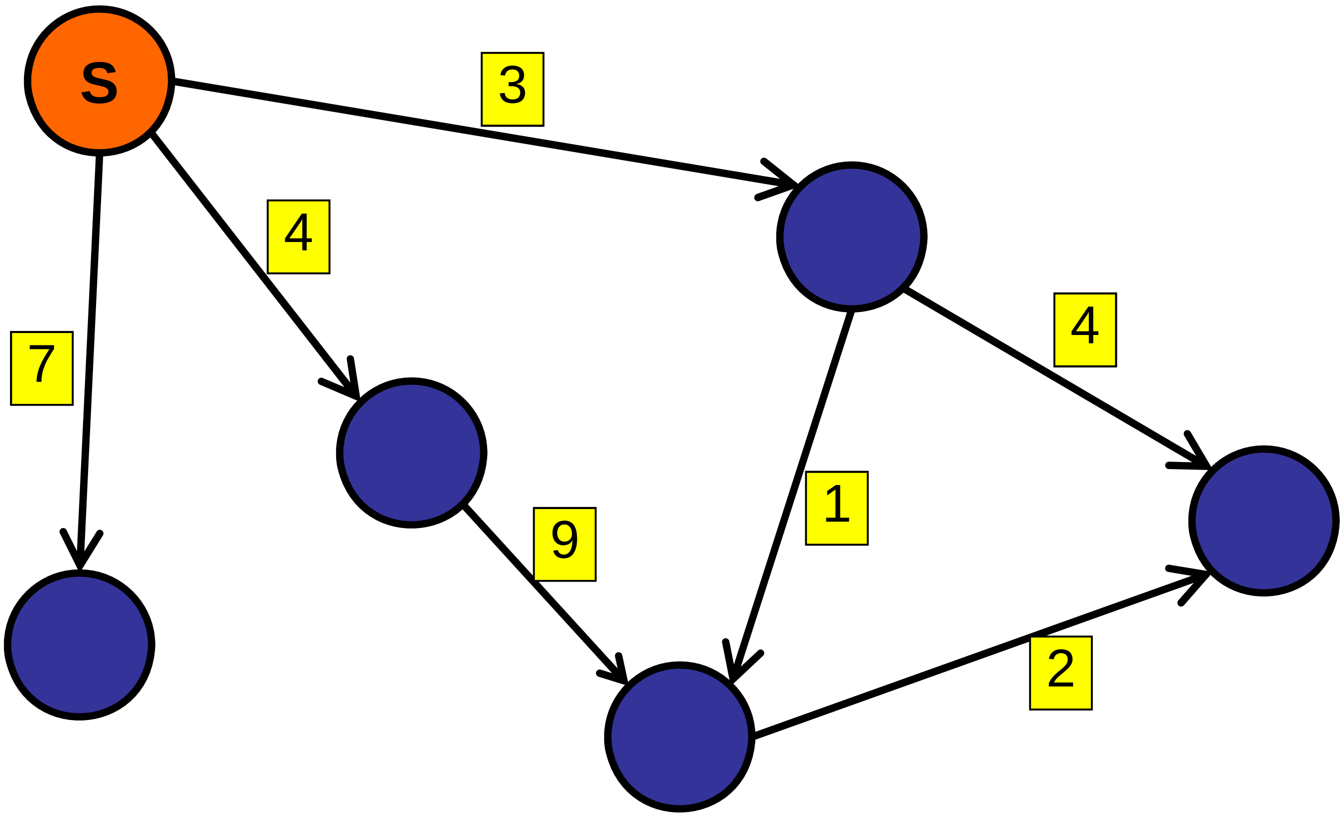
Common mistake: "Why can't I use BFS?"



BFS finds minimum number of HOPS not minimum DISTANCE.

Shortest Paths

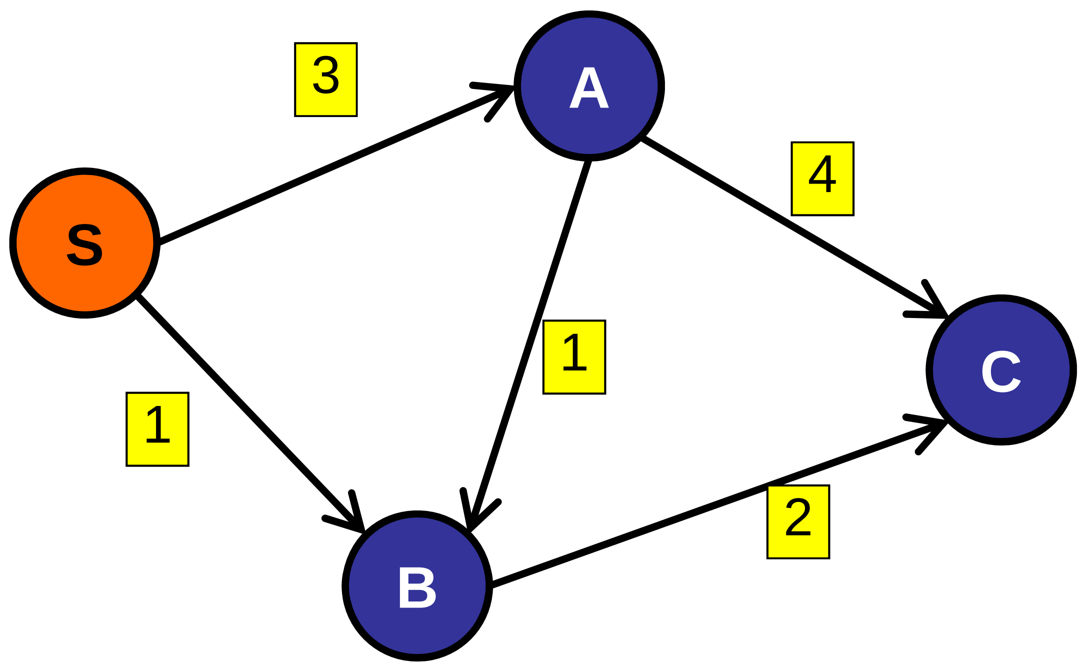
Notation: $\delta(u,v)$ = distance from u to v



Shortest Paths

Key idea: triangle inequality

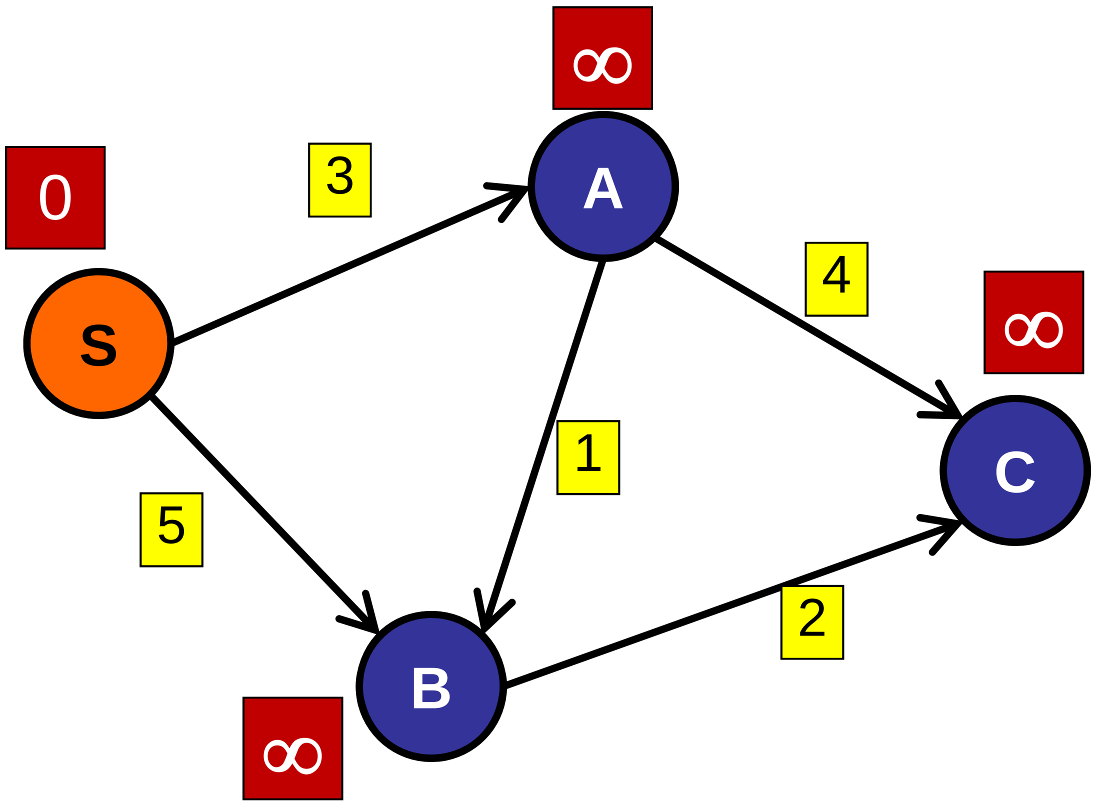
$\delta(S, C) \leq \delta(S, A) + \delta(A, C)$



Shortest Paths

Maintain estimate for each distance:

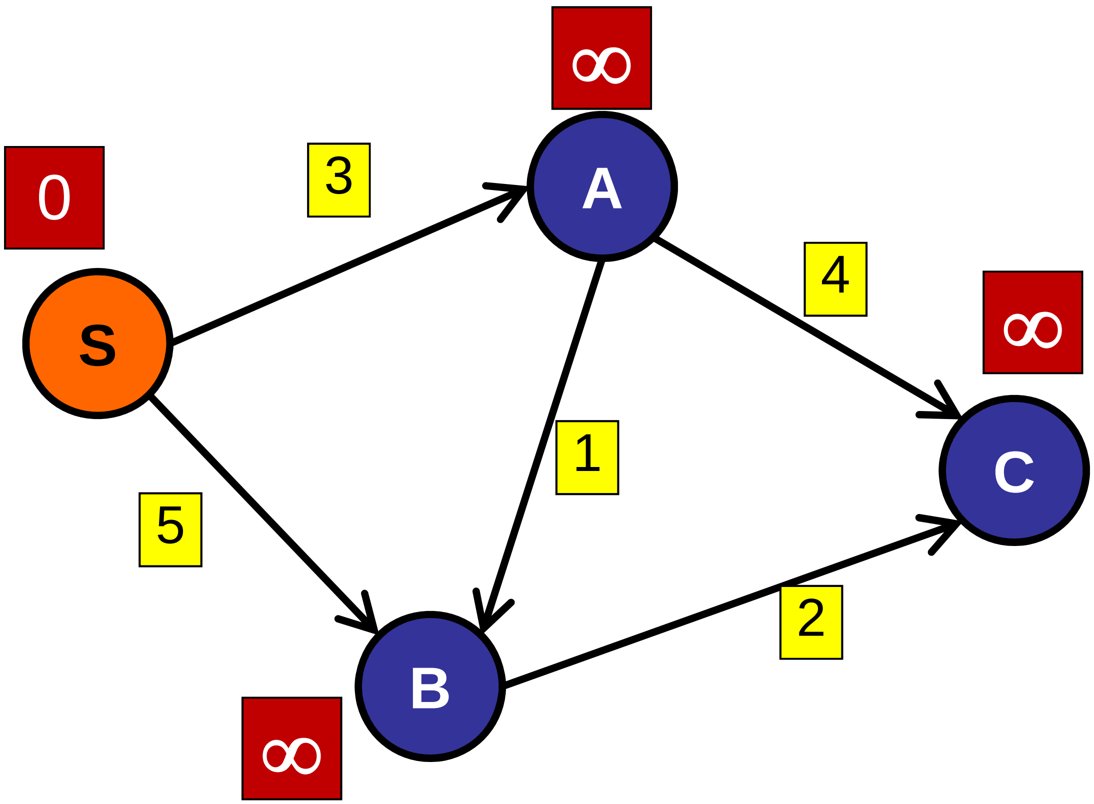
```
int[] dist = new int[V.length];

Arrays.fill(dist, INFTY);

dist[start] = 0;
```



Shortest Paths

Maintain estimate for each distance:

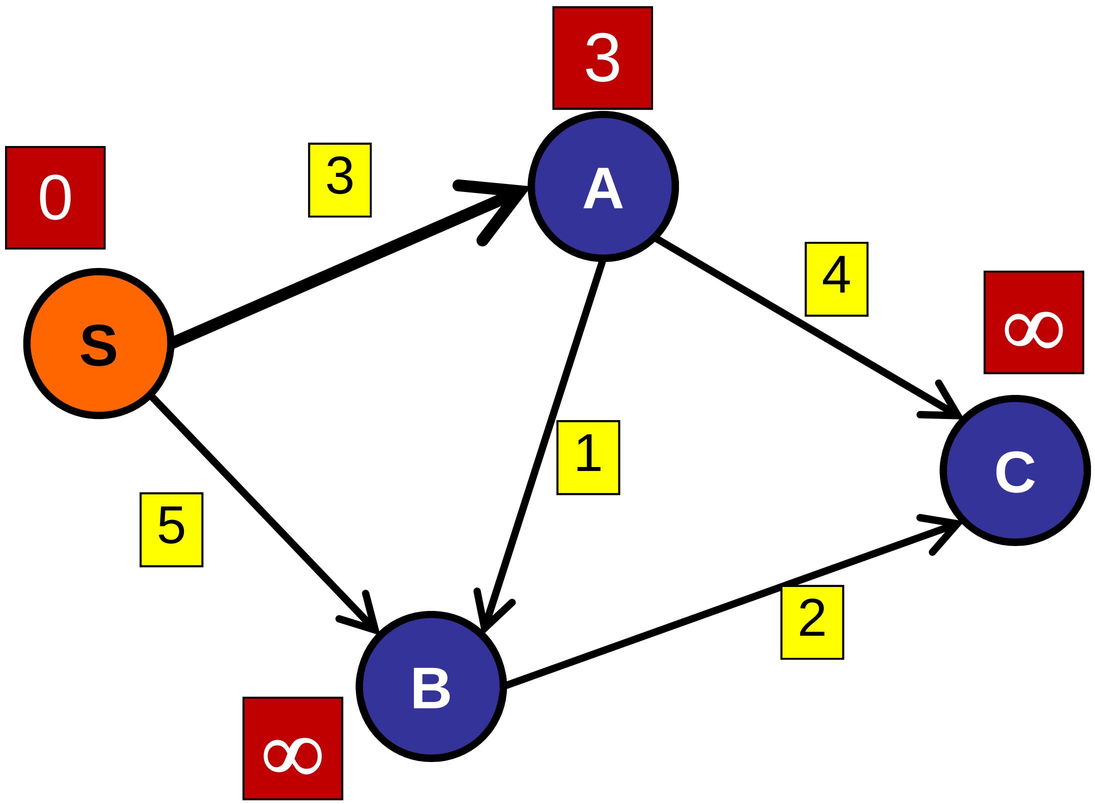
- Reduce estimate
- Invariant: estimate $\geq$ distance



Shortest Paths

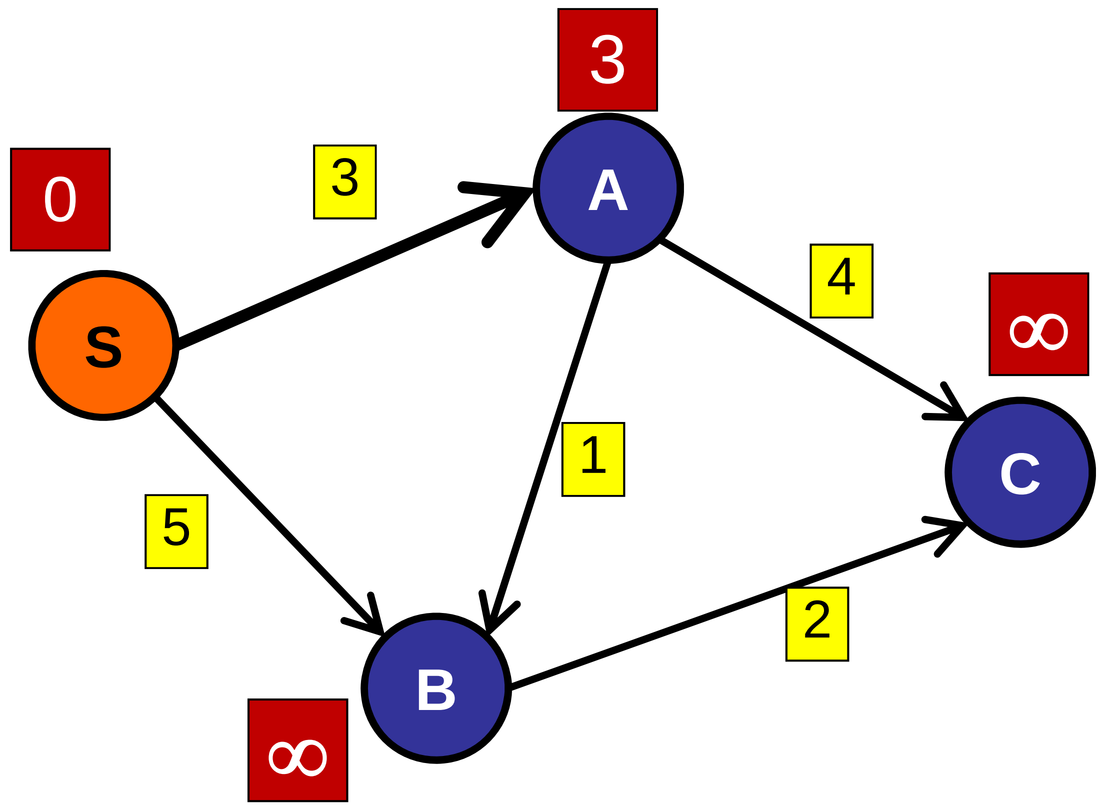
Maintain estimate for each distance:

relax(S, A)



Shortest Paths

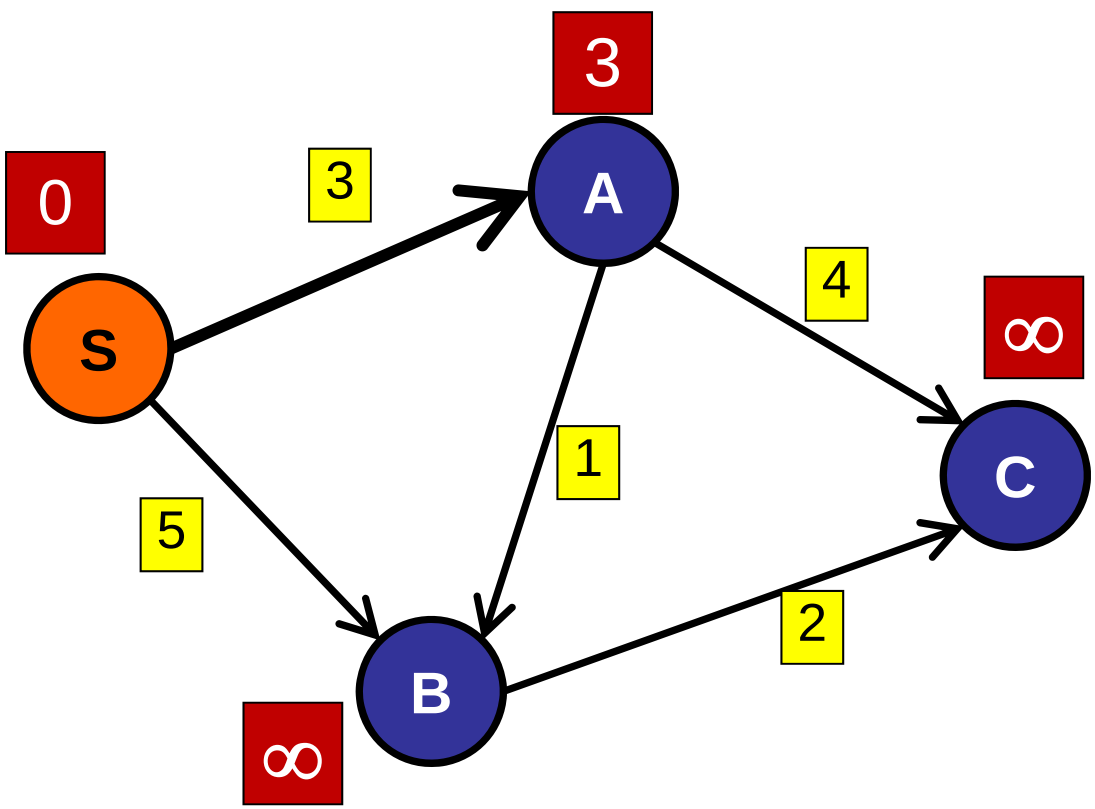
```
relax(int u, int v){
    if (dist[v] > dist[u] + weight(u,v))
        dist[v] = dist[u] + weight(u,v);
}
```



Shortest Paths

Maintain estimate for each distance:

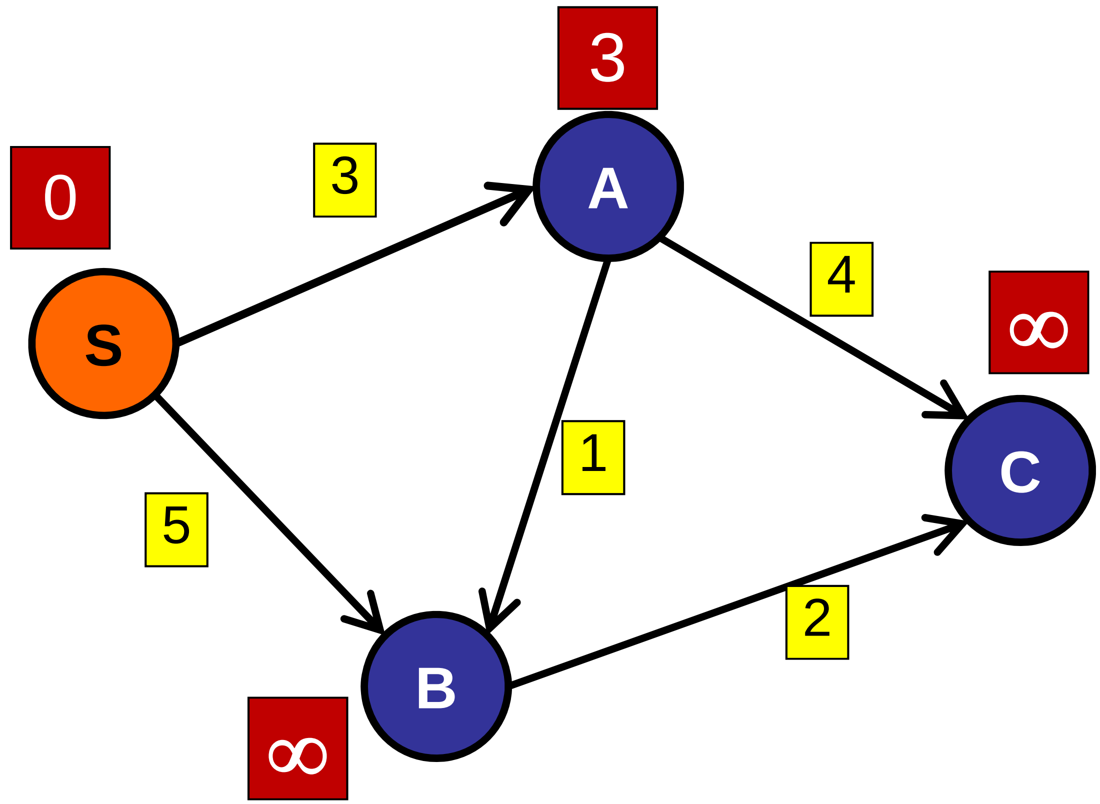
relax(S, A)



Shortest Paths

Maintain estimate for each distance:

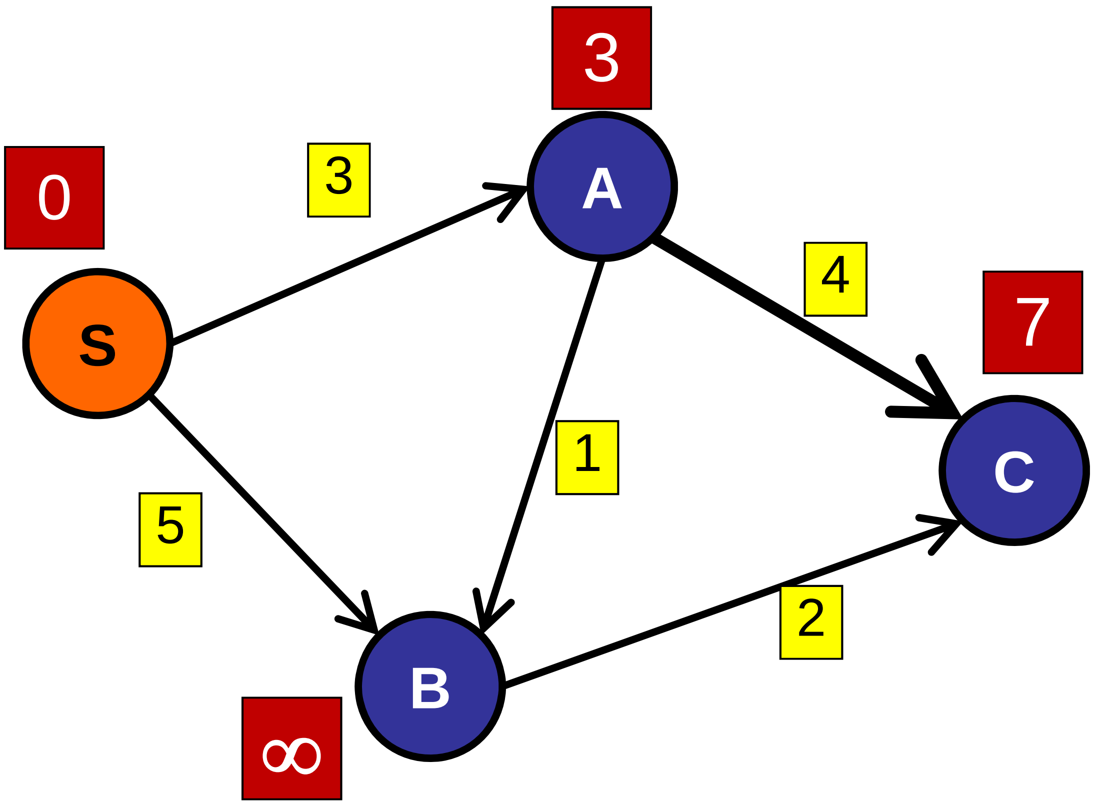
relax(A, C)



Shortest Paths

Maintain estimate for each distance:

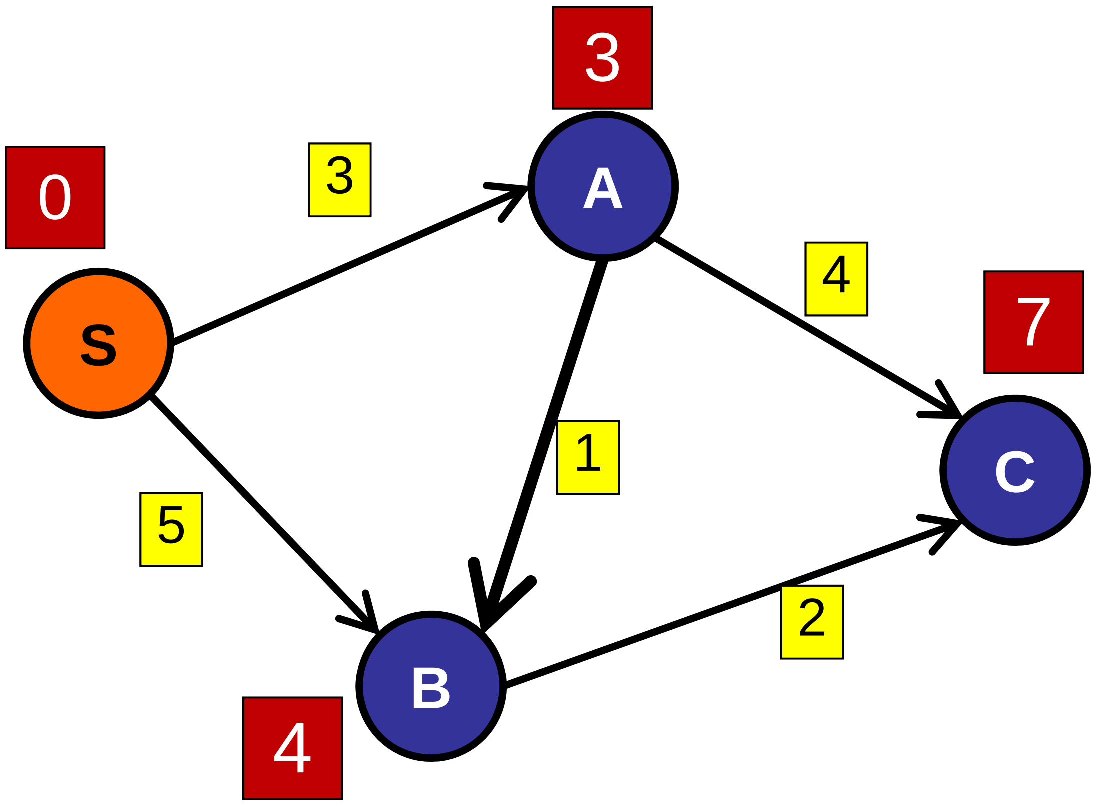
relax(A, C)



Shortest Paths

Maintain estimate for each distance:

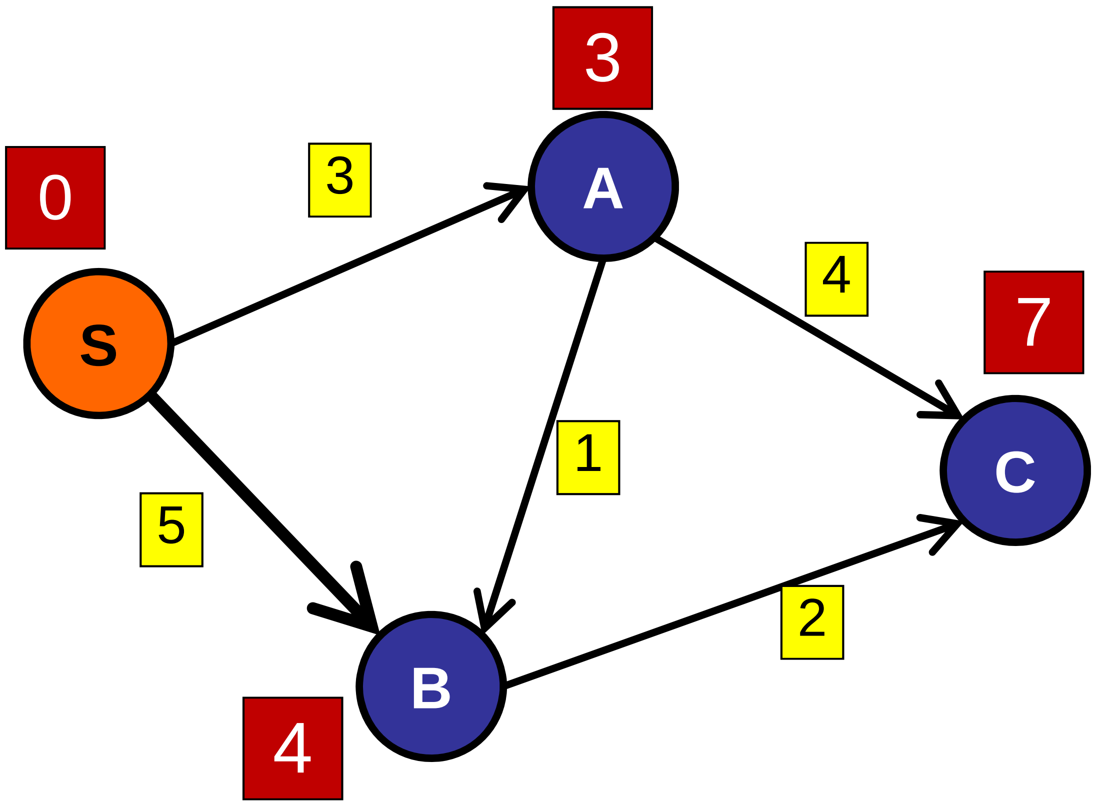
relax(A, B)



Shortest Paths

Maintain estimate for each distance:

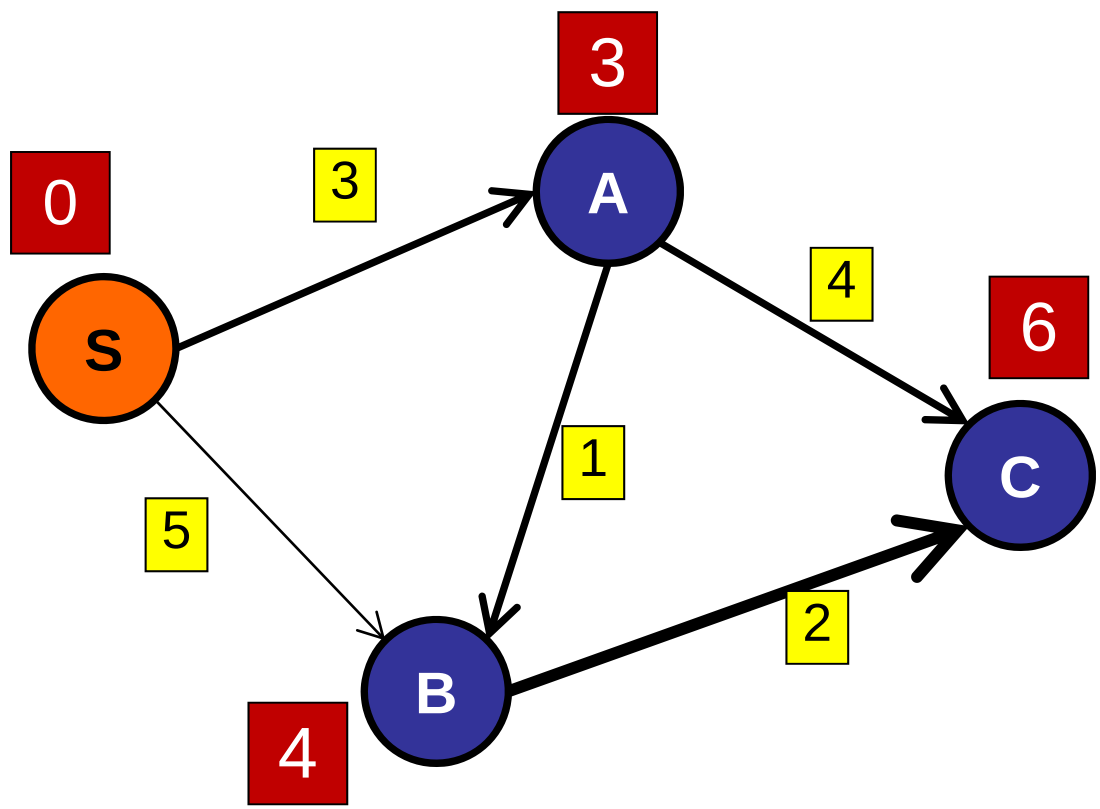
relax(S, B)



Shortest Paths

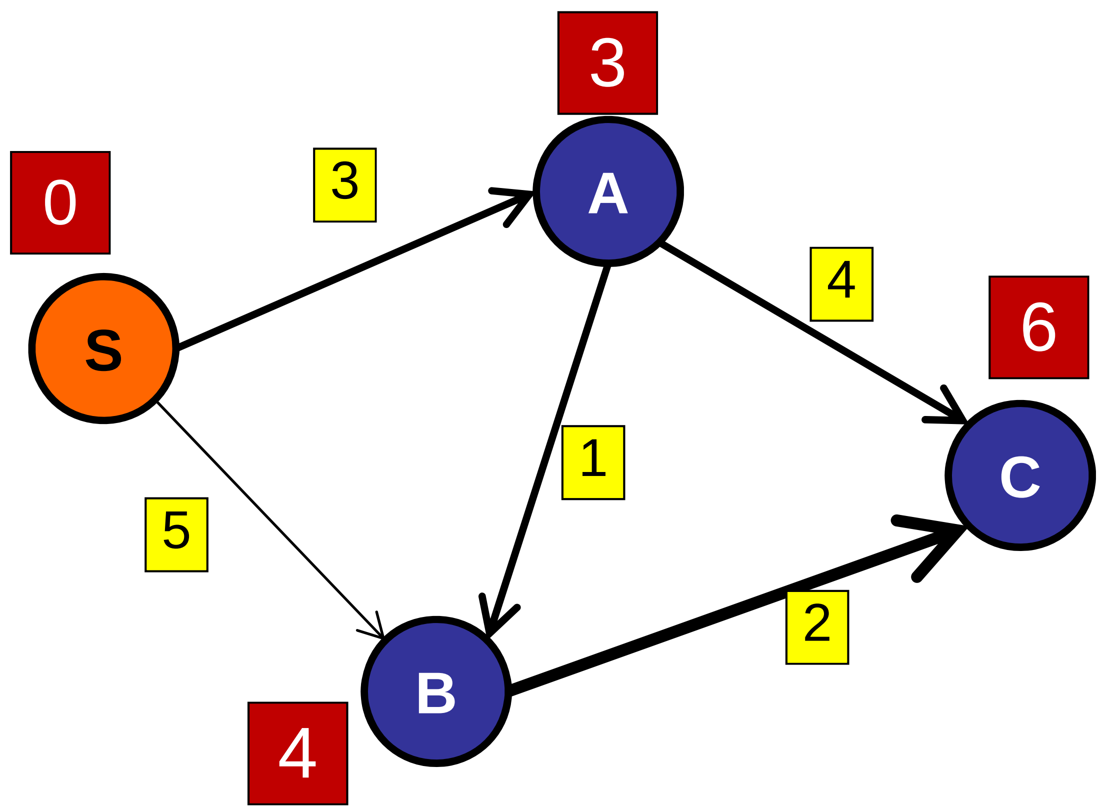
Maintain estimate for each distance:

relax(B, C)



Shortest Paths

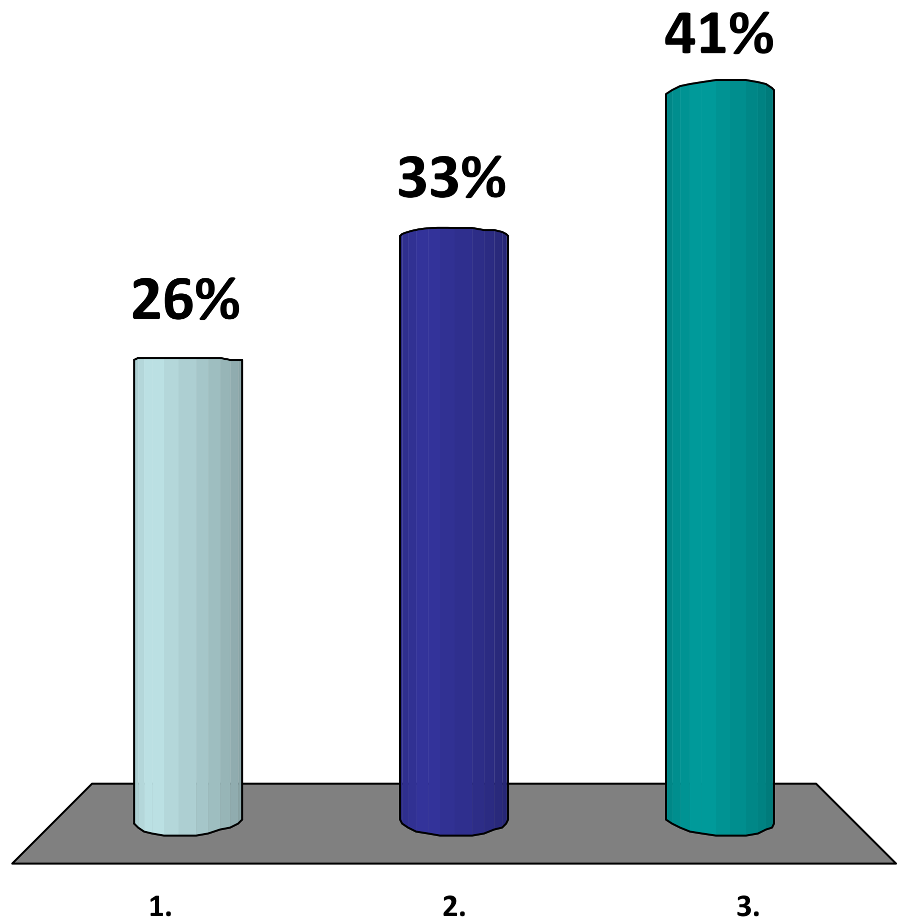
```
for (Edge e : graph)
    relax(e)
```



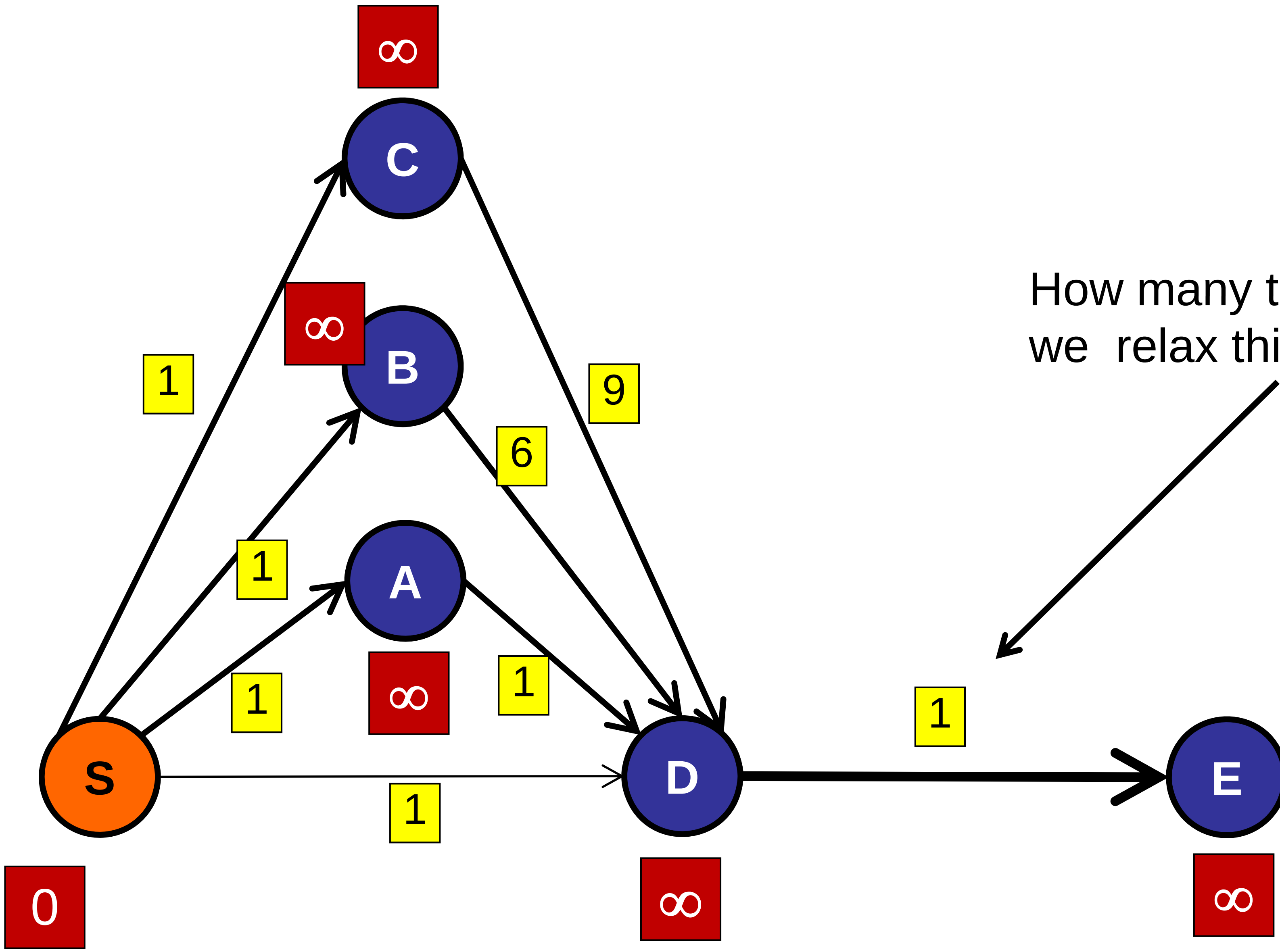
Does this algorithm work:

for every edge e: relax(e)

1. Yes
2. Sometimes
3. No

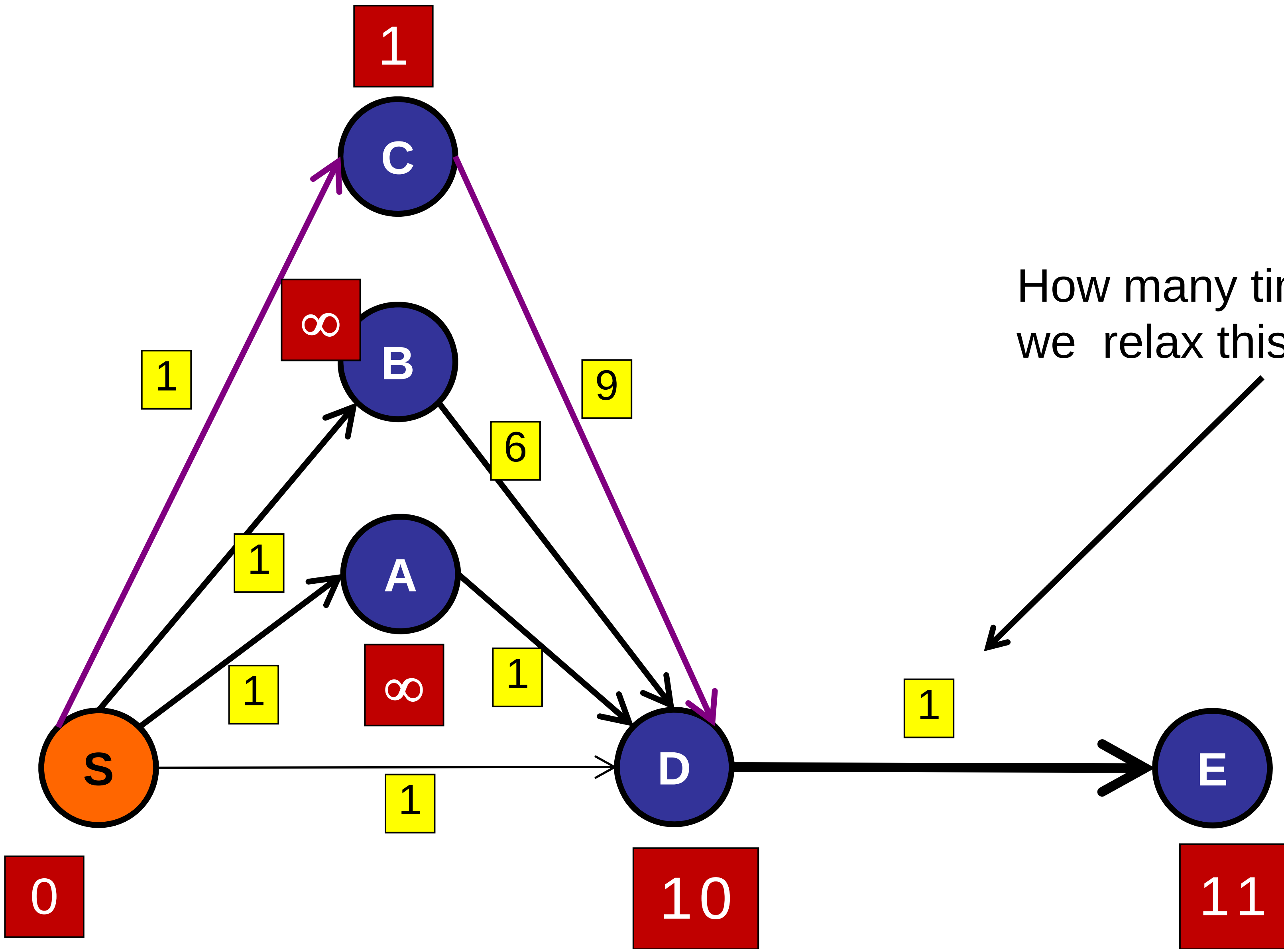


Shortest Paths



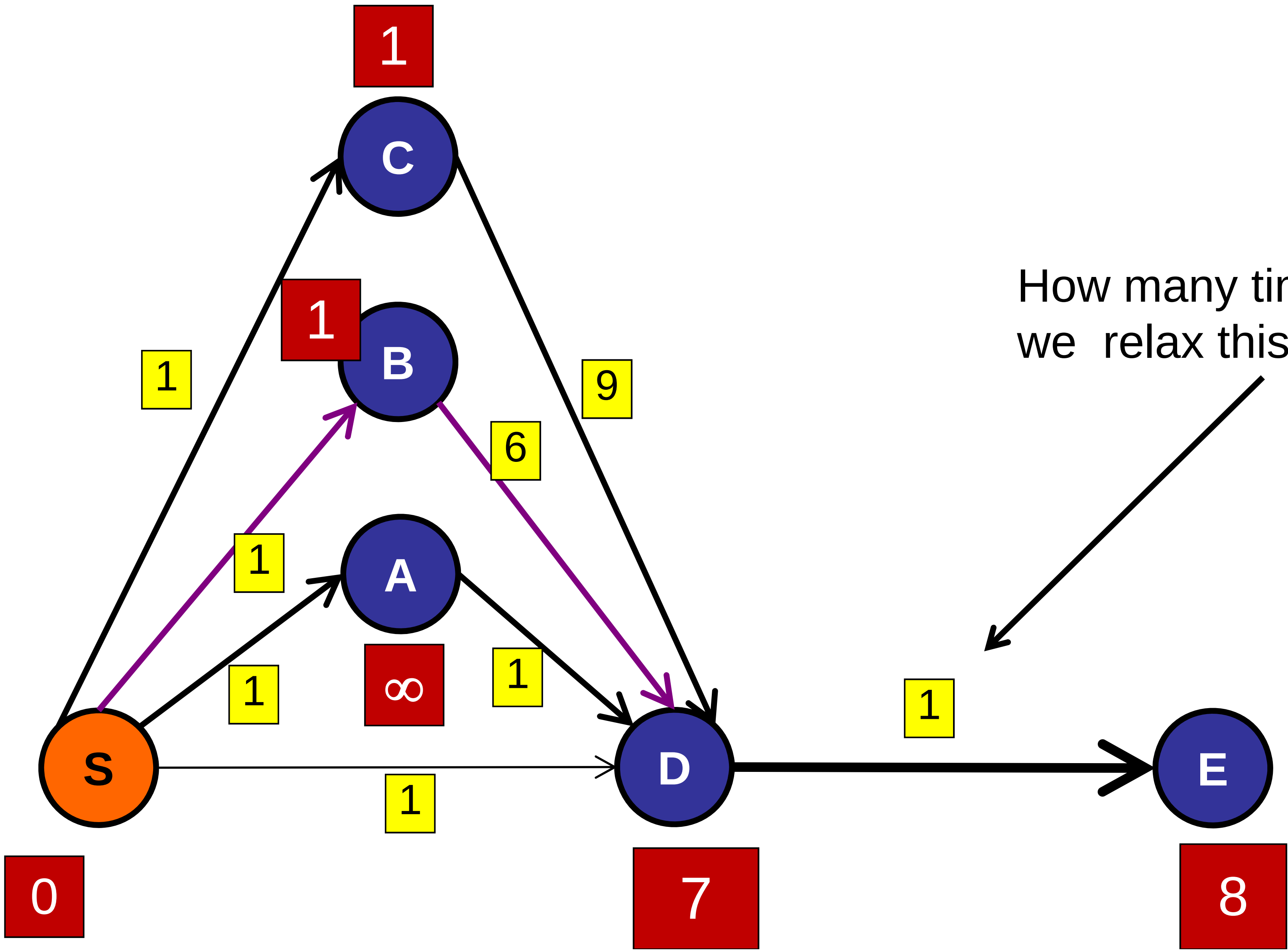
How many times might we relax this edge?

Shortest Paths



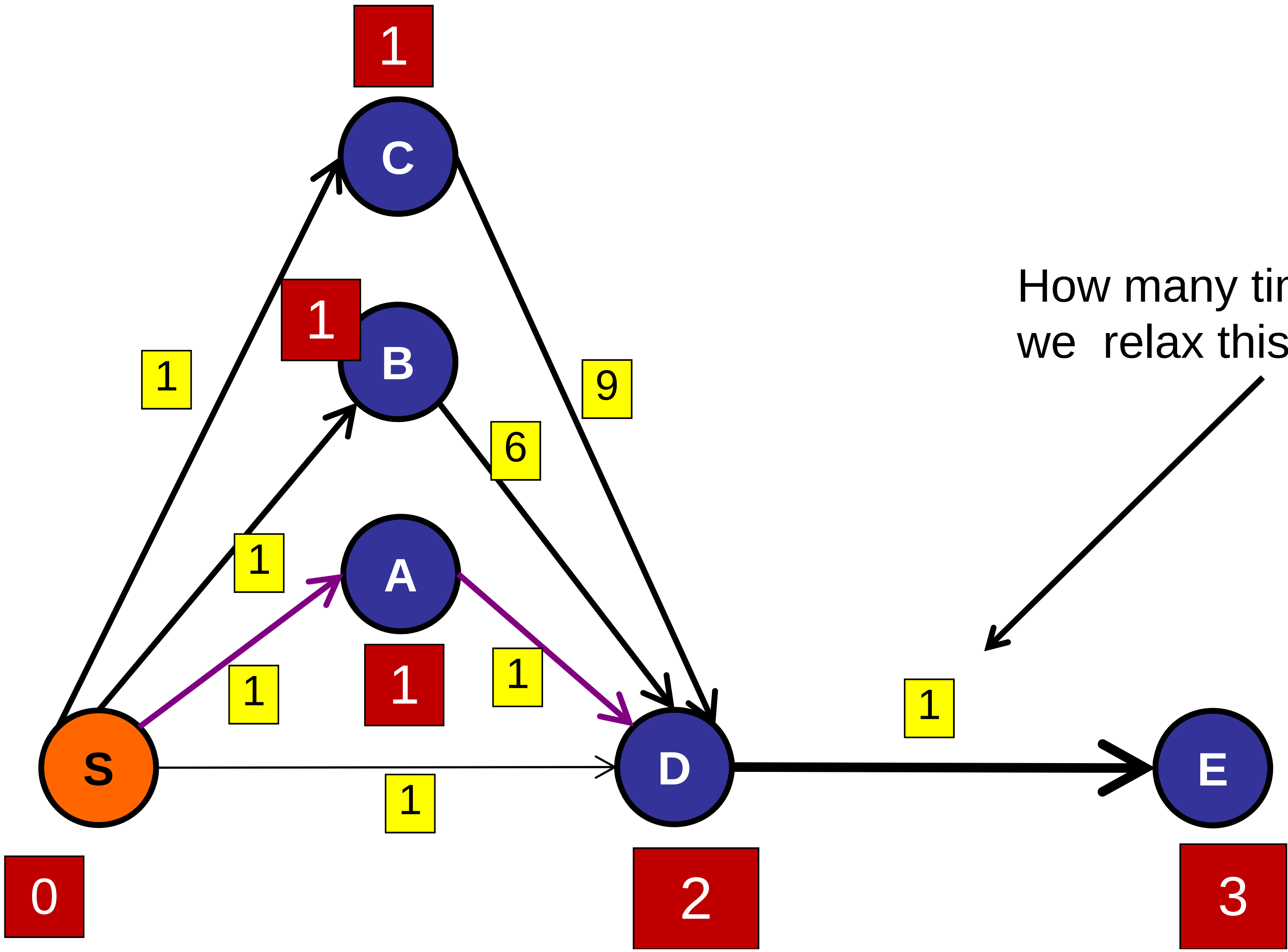
How many times might we relax this edge?

Shortest Paths



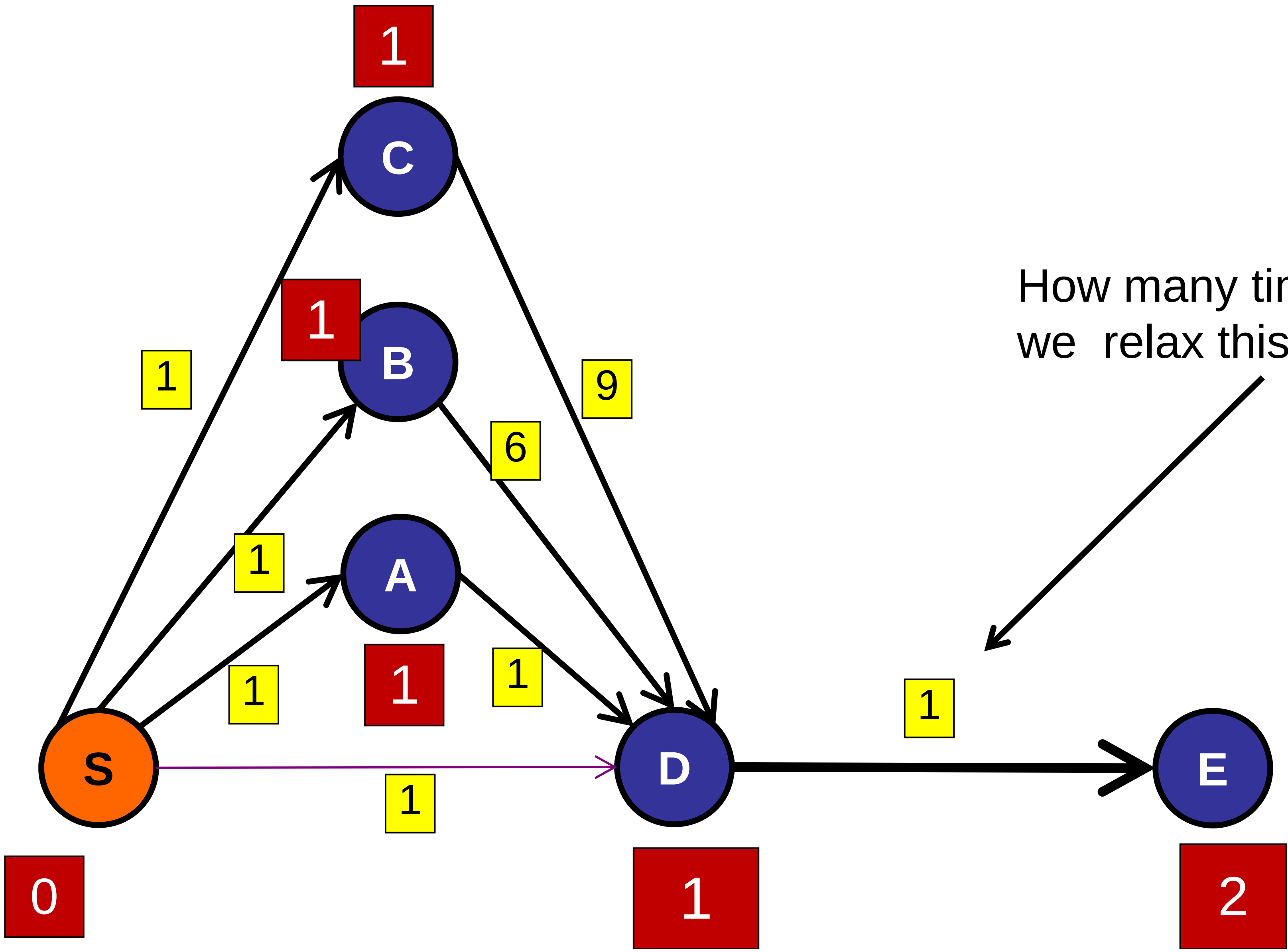
How many times might we relax this edge?

Shortest Paths



How many times might we relax this edge?

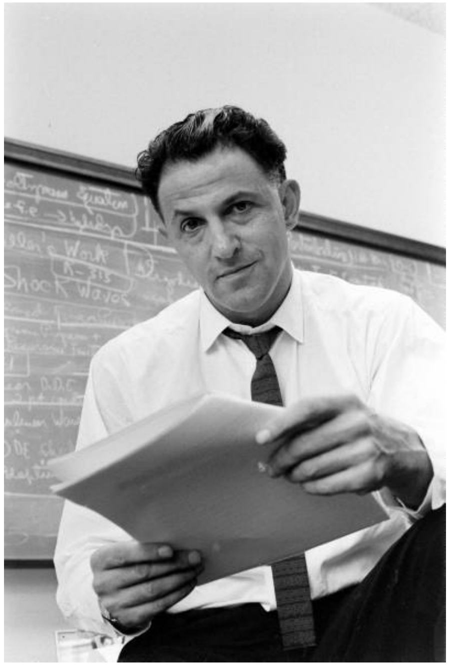
Shortest Paths



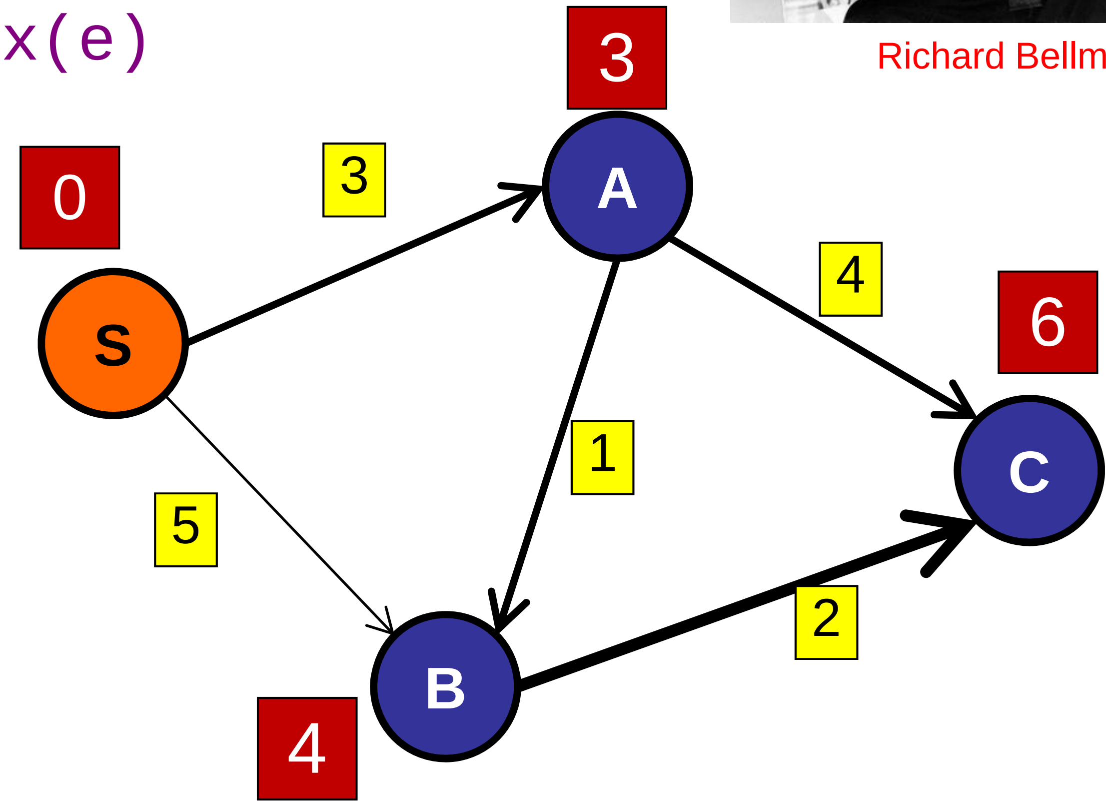
How many times might we relax this edge?

Bellman-Ford

```
n = V.length;
for (i=0; i<n; i++)
    for (Edge e : graph)
            relax(e)
```

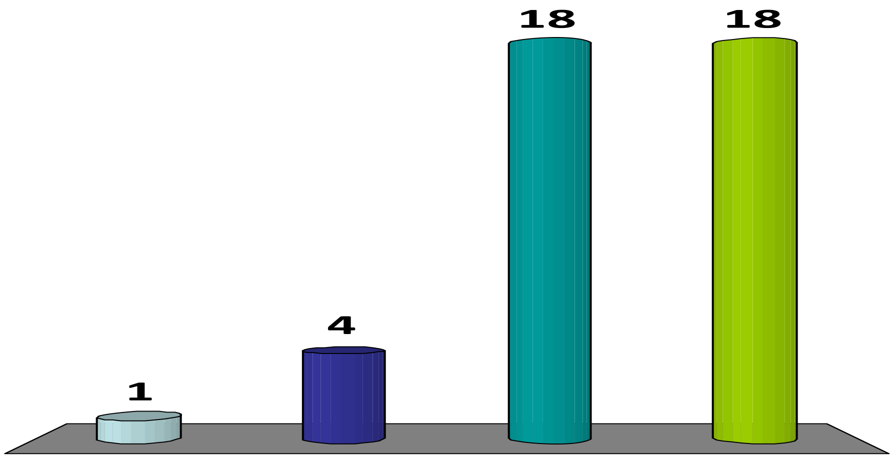


Richard Bellman



When can you terminate early?

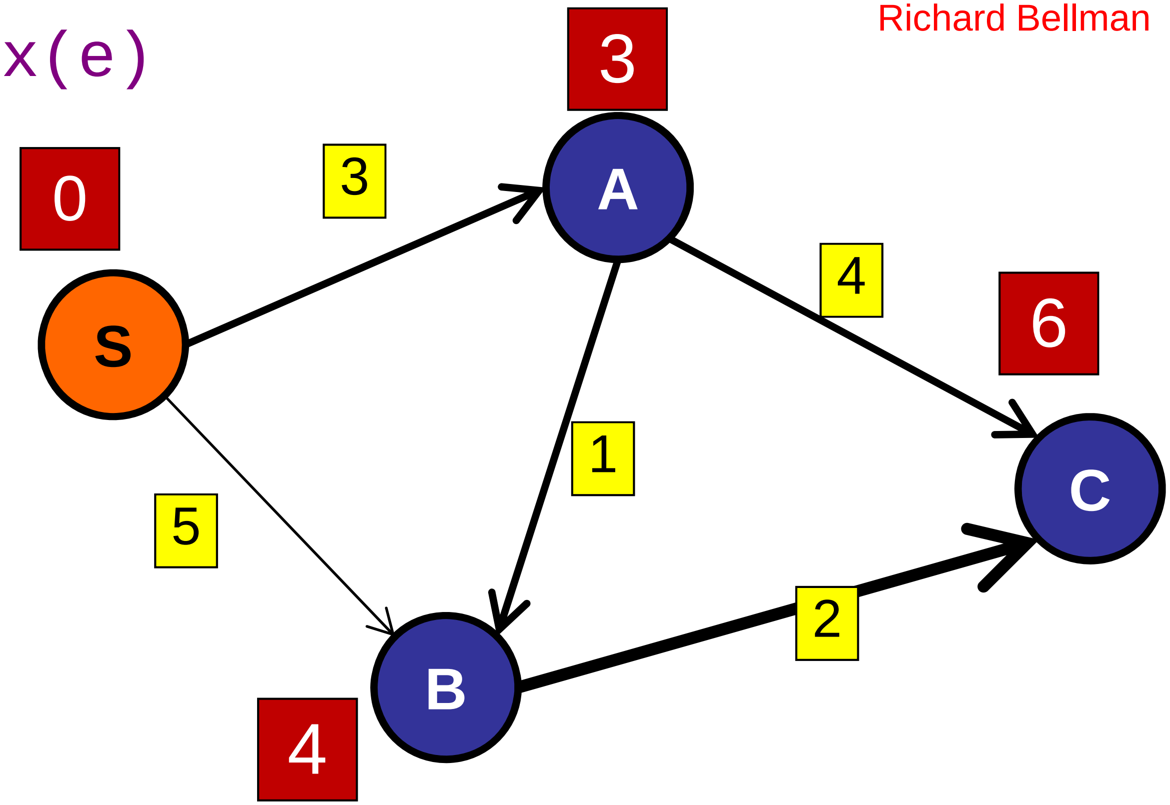
1. When a relax operation has no effect.
2. When two consecutive relax operations have no effect.
3. When an entire sequence of |E| relax operations have no effect.
4. Never. Only after |V| complete iterations.



Bellman-Ford

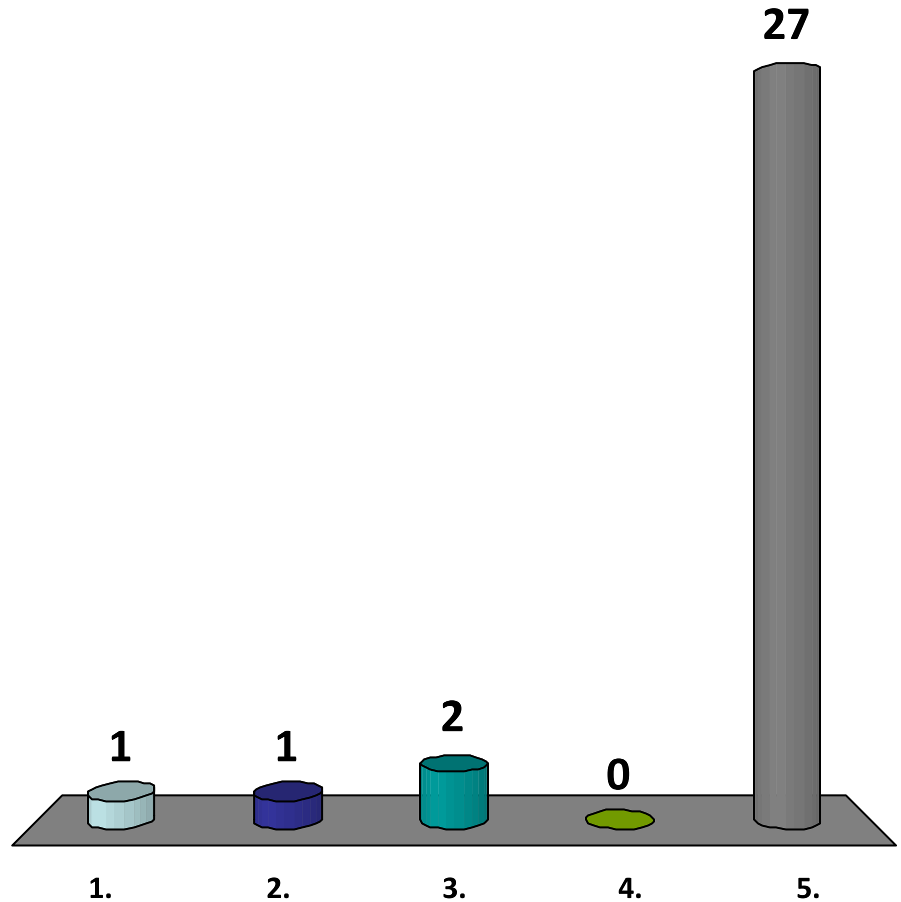
```
n = V.length;
for (i=0; i<n; i++)
    for (Edge e : graph)
            relax(e)
```

Richard Bellman



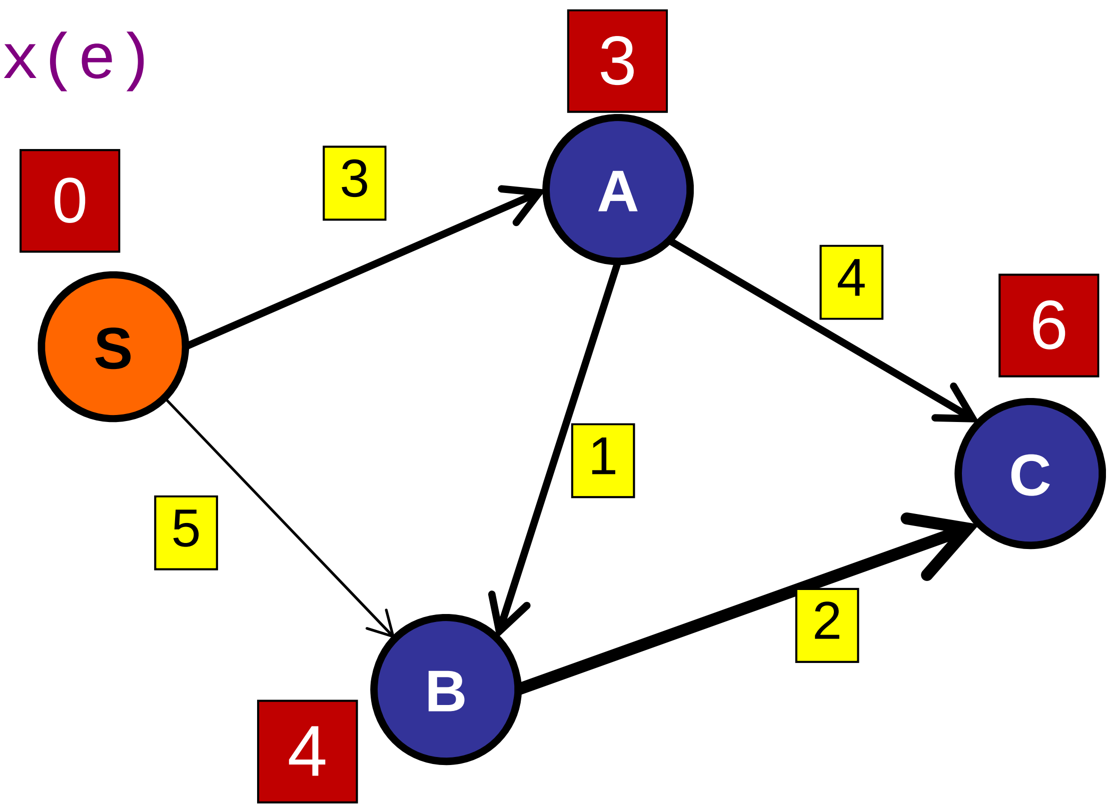
What is the running time of Bellman-Ford?

1. O(V)
2. O(E)
3. O(V+E)
4. O(E log V)
5. O(EV)



Bellman-Ford

```
n = V.length;
for (i=0; i<n; i++)
    for (Edge e : graph)
        relax(e)
```

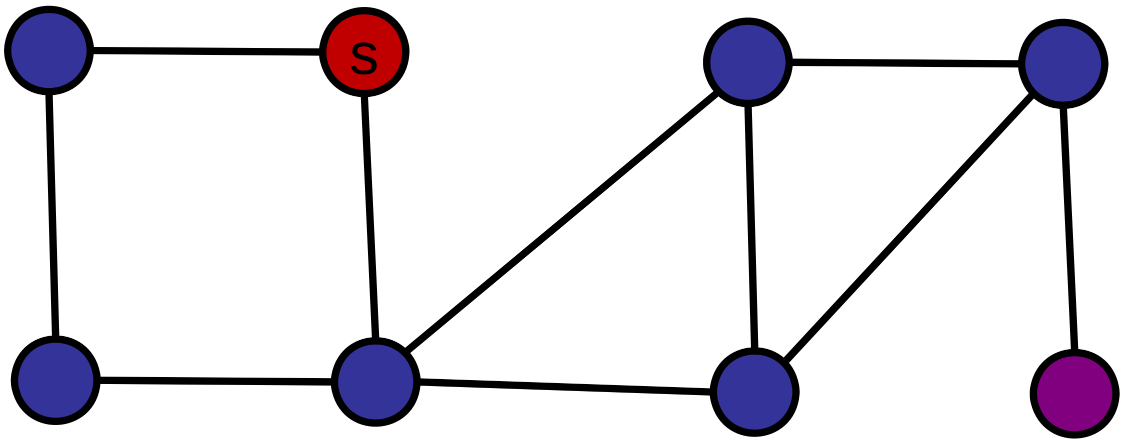


Bellman-Ford

Why does this work?

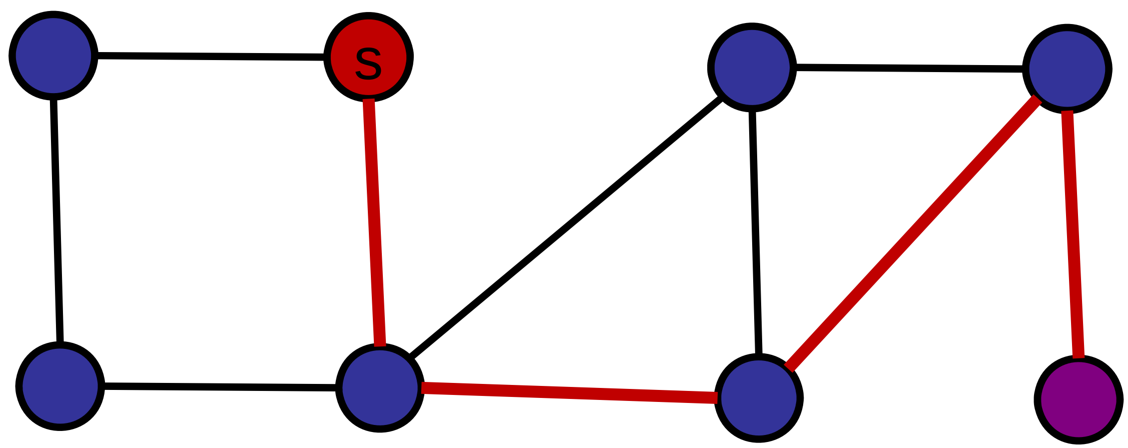
Bellman-Ford

Why does this work?



Bellman-Ford

Why does this work?

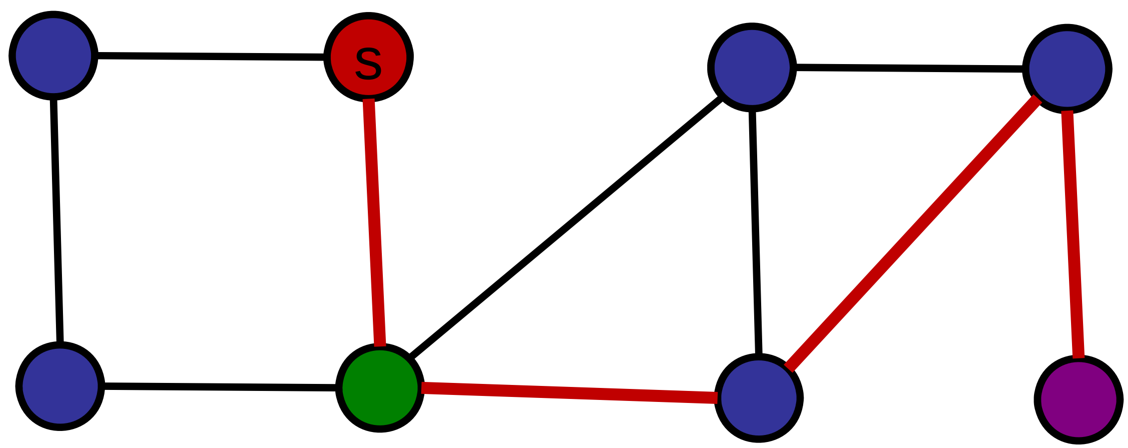


Look at minimum weight path from S to D.

(Path is simple: no loops.)

Bellman-Ford

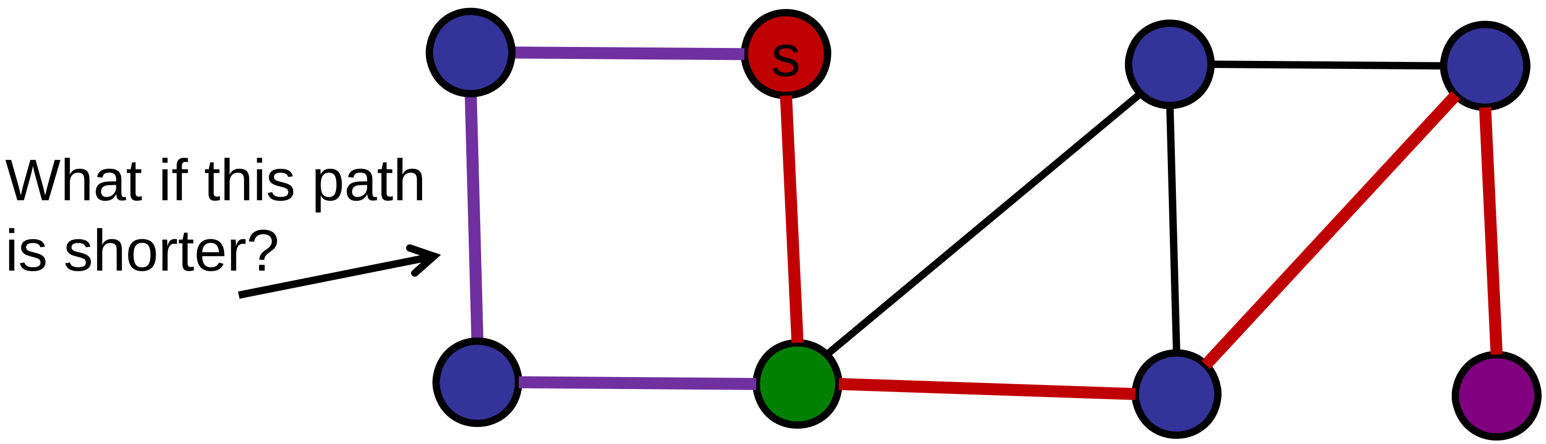
Why does this work?



After 1 iteration, 1 hop estimate is correct.

Bellman-Ford

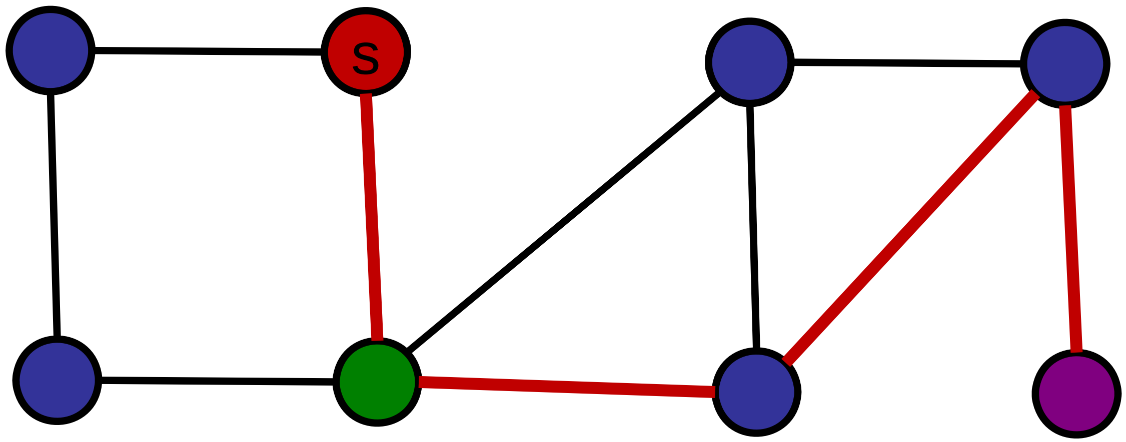
Why does this work?



After 1 iteration, 1 hop estimate is correct.

Bellman-Ford

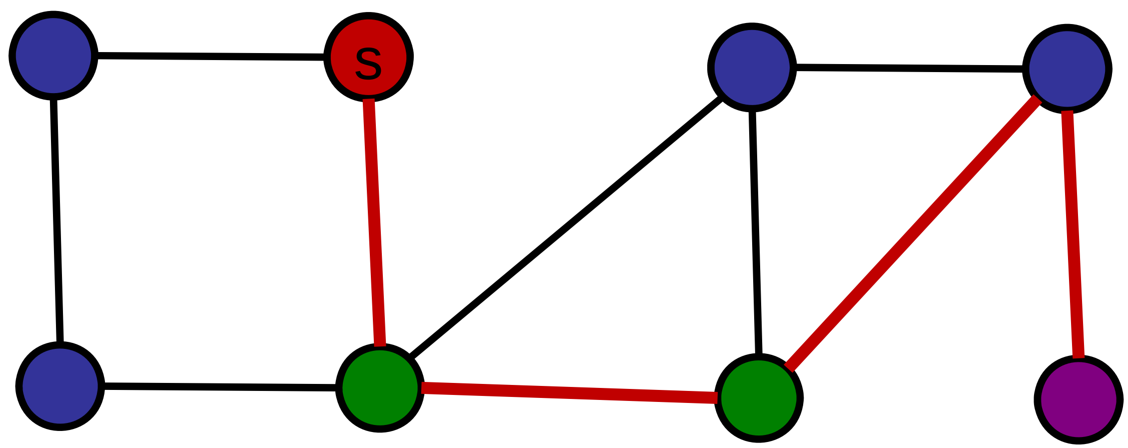
Why does this work?



After 1 iteration, 1 hop estimate is correct.

Bellman-Ford

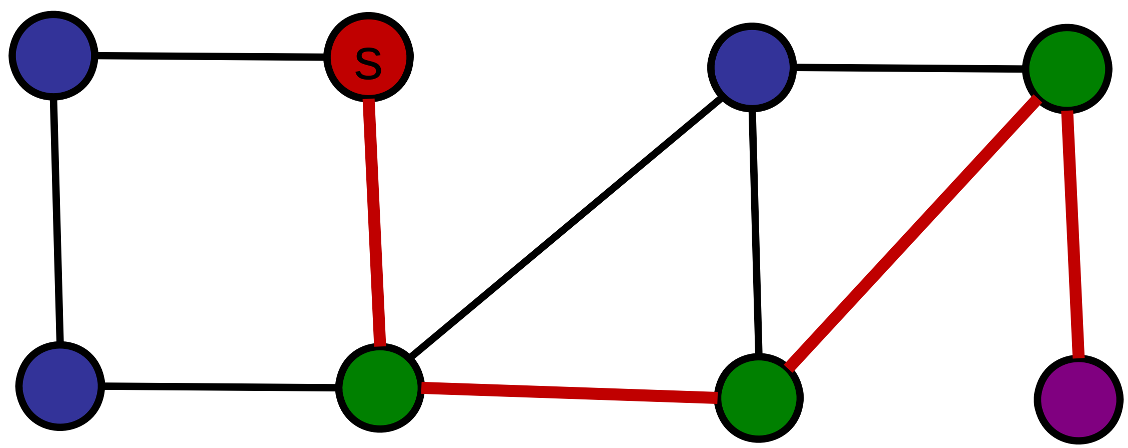
Why does this work?



After 2 iterations, 2 hop estimate is correct.

Bellman-Ford

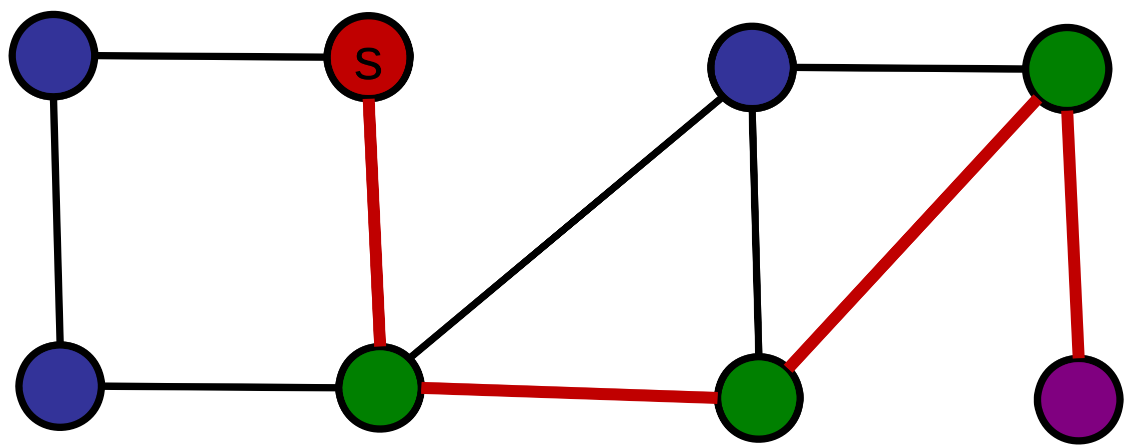
Why does this work?



After 3 iterations, 3 hop estimate is correct.

Bellman-Ford

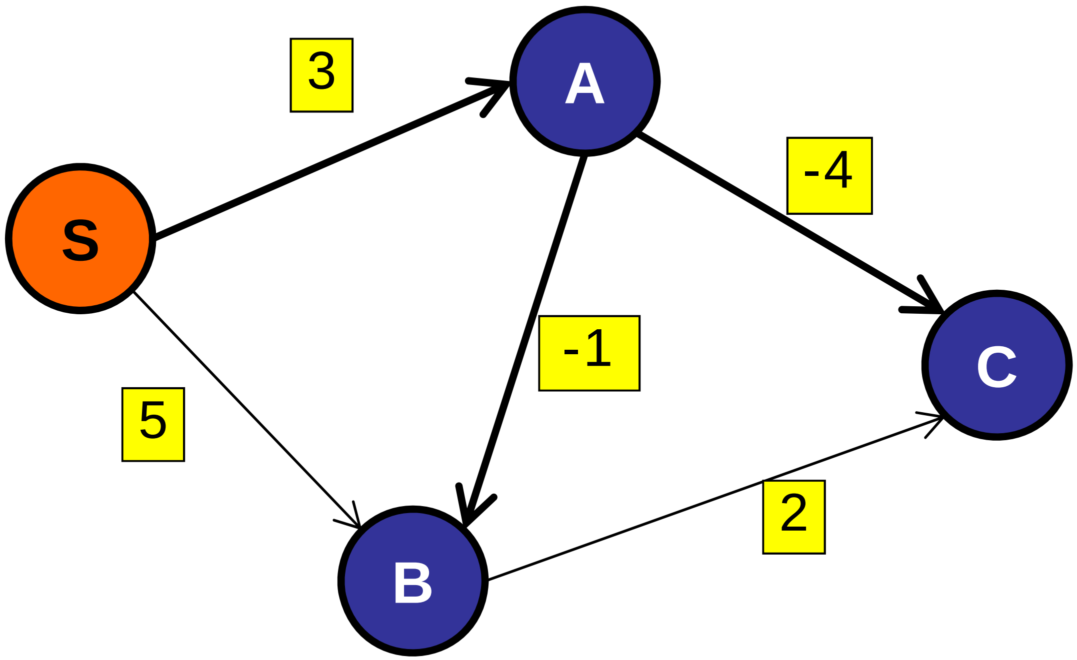
Why does this work?



After 4 iterations, D estimate is correct.

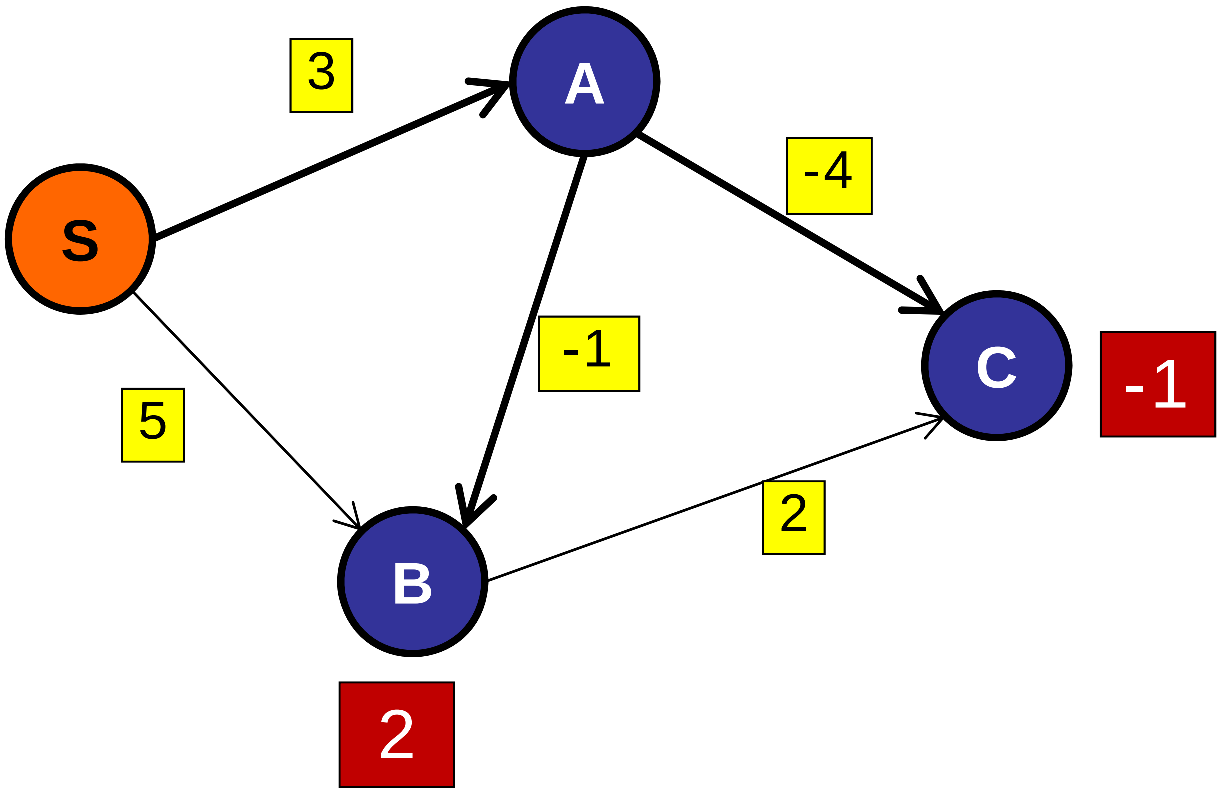
Bellman-Ford

What if edges have negative weight?



Bellman-Ford

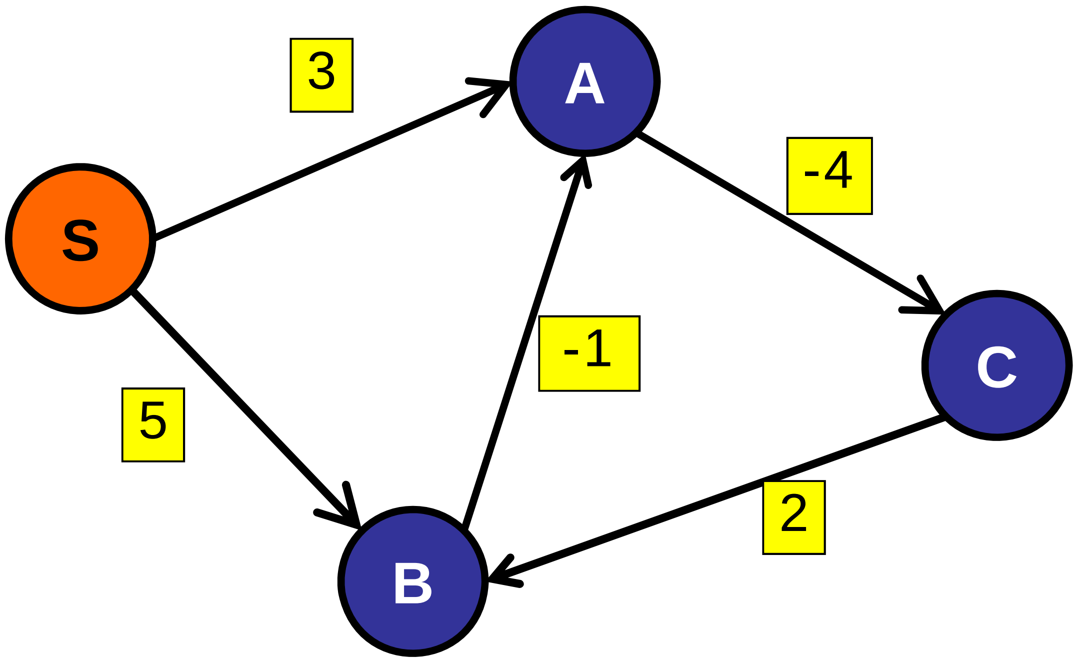
What if edges have negative weight?



No problem!

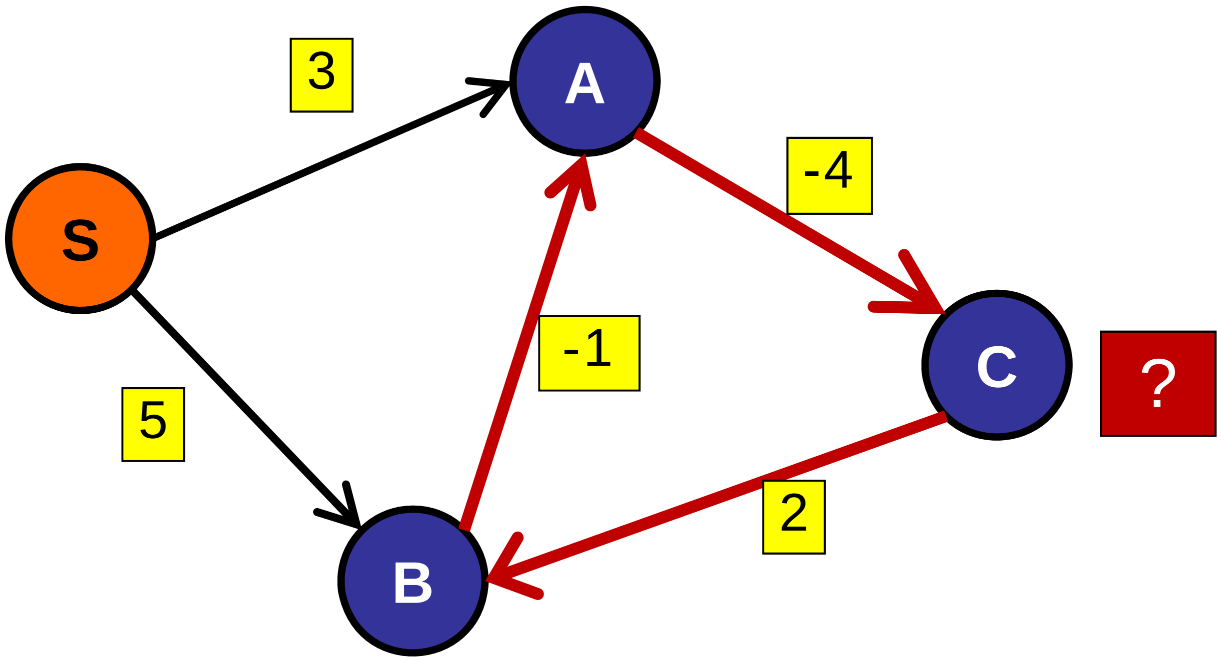
Bellman-Ford

What if edges have negative weight?



Bellman-Ford

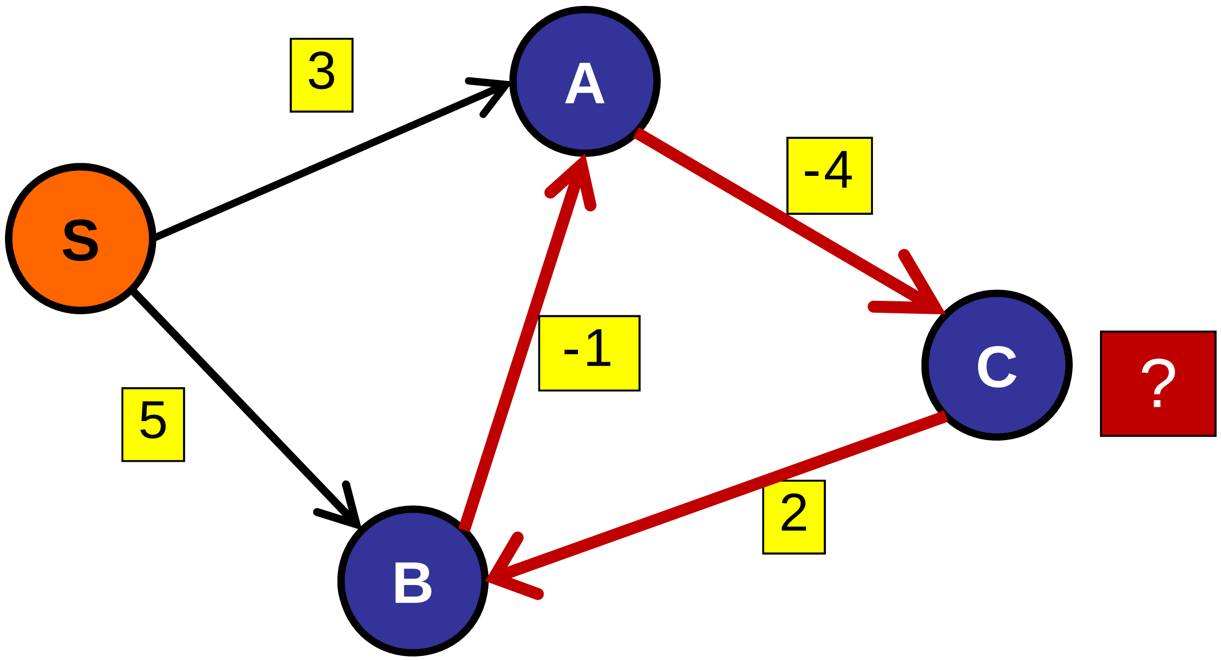
What if edges have negative weight?



d(S,C) is infinitely negative!

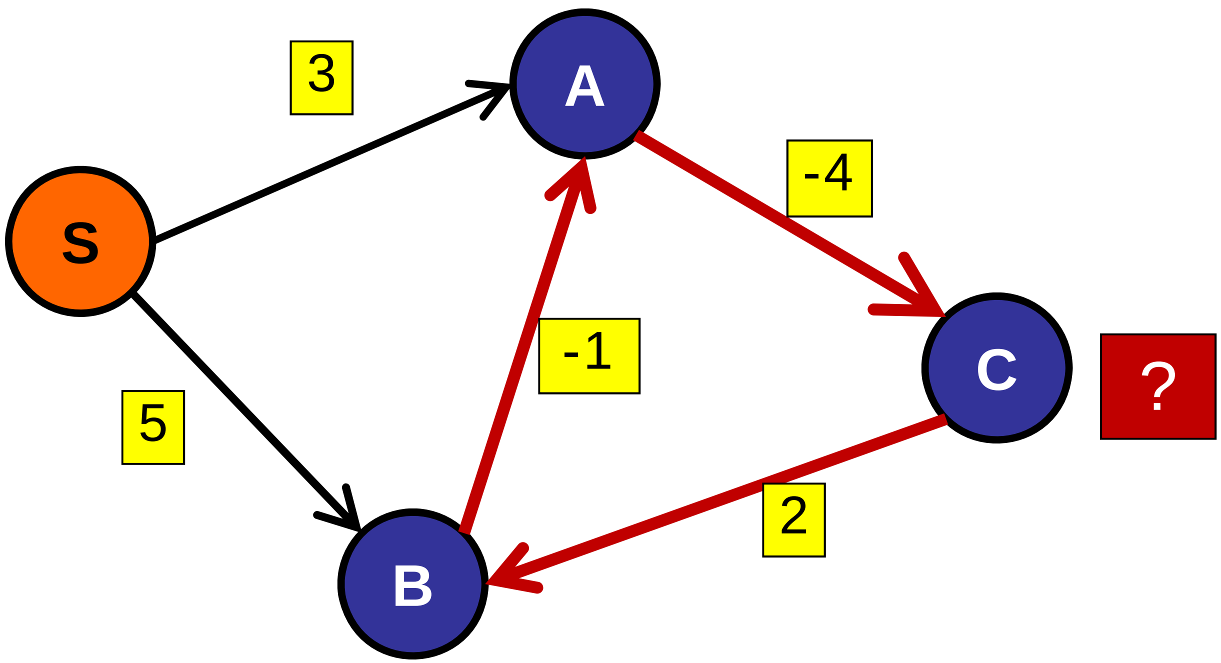
Negative weight cycles

How to detect negative weight cycles?



Negative weight cycles

How to detect negative weight cycles?

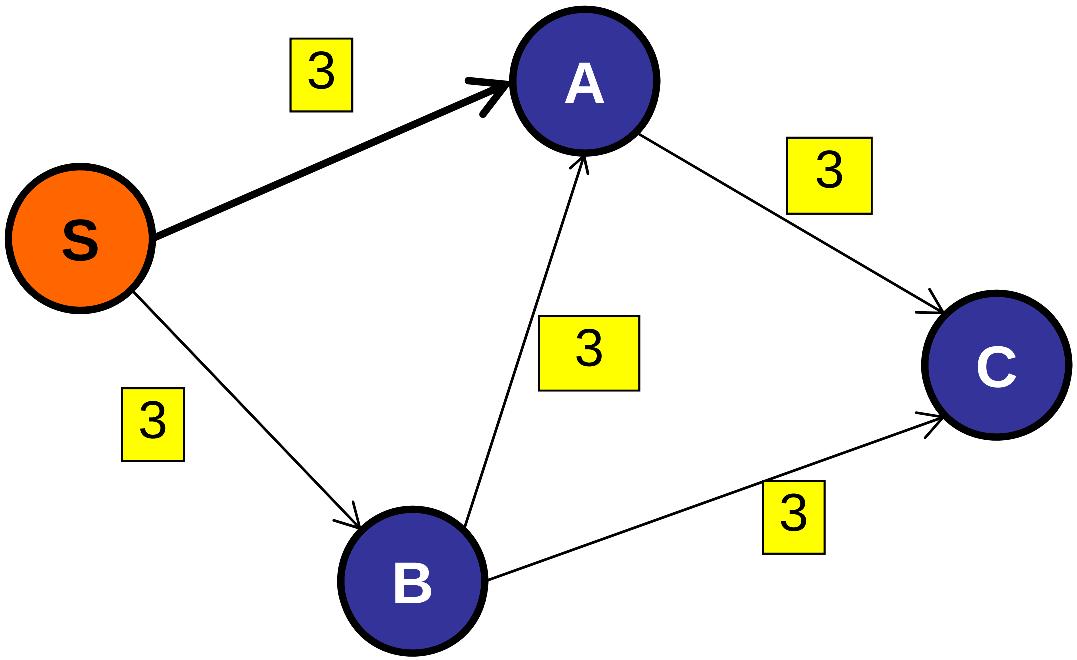


Run Bellman-Ford for |V|+1 iterations.

If an estimate changes in the last iteration...
then negative weight cycle.

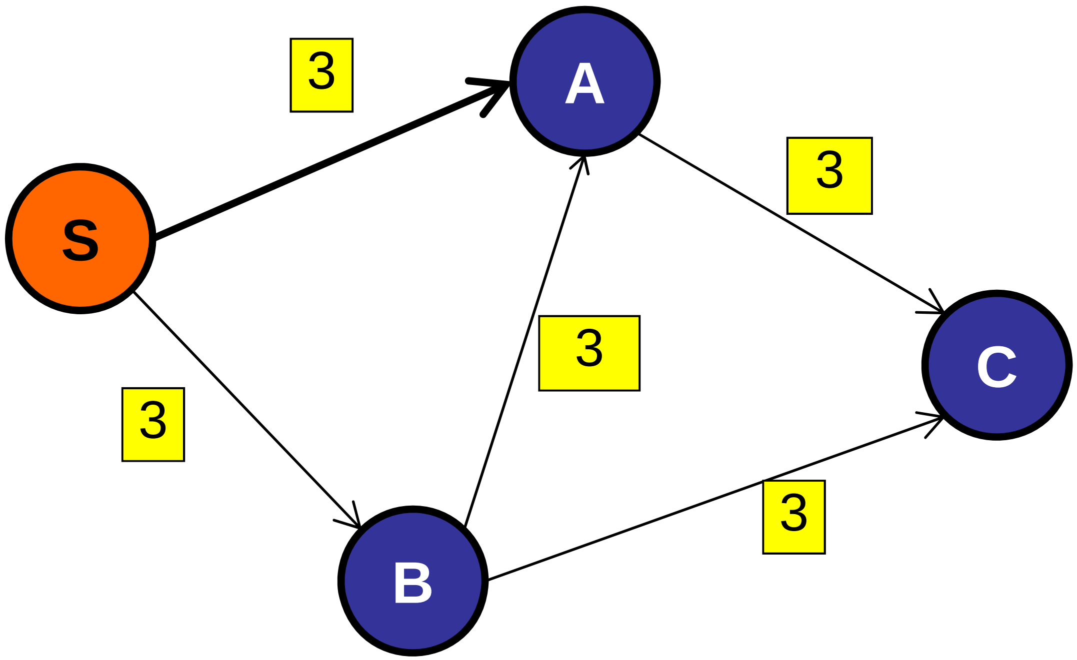
Bellman-Ford

Special case: all edges have the same weight



Bellman-Ford

Special case: all edges have the same weight.



Use regular Breadth-First Search.

Bellman-Ford Summary

Basic idea:

- Repeat |V| times: relax every edge
- Stop when “converges”.
- O(VE) time.

Special issues:

- If negative weight-cycle: impossible.
- Use Bellman-Ford to detect negative weight cycle.
- If all weights are the same, use BFS.

Roadmap

Part I: Directed Graphs

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

Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford