CS2040S
Data Structures and Algorithms

BFS, DFS, and Directed Graphs!

Roadmap

Last time: Graph Basics

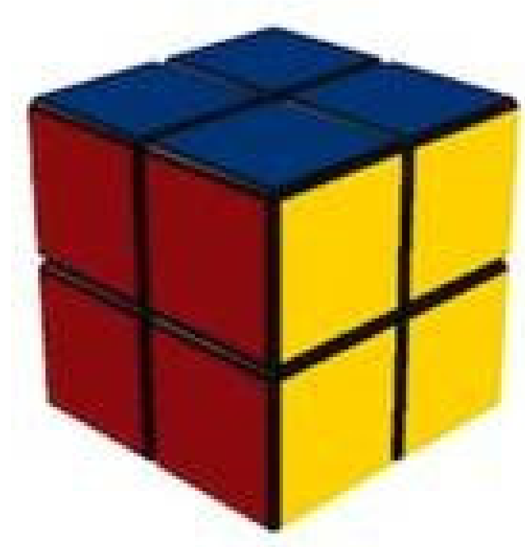
- What is a graph?
- Modeling problems as graphs.
- Graph representations (list vs. matrix)
- Searching graphs: BFS

What is a 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 $e_1, e_2 \in E : e_1 \neq e_2$

2 x 2 x 2 Rubik's Cube



Configuration Graph

- Vertex for each possible state
- Edge for each basic move
  - 90 degree turn
  - 180 degree turn

Puzzle: given initial state, find a path to the solved state.

Trade-offs

Adjacency Matrix:

- Fast query: are v and w neighbors?
- Slow query: find me any neighbor of v.
- Slow query: enumerate all neighbors.

Adjacency List:

- Fast query: find me any neighbor.
- Fast query: enumerate all neighbors.
- Slower query: are v and w neighbors?

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)

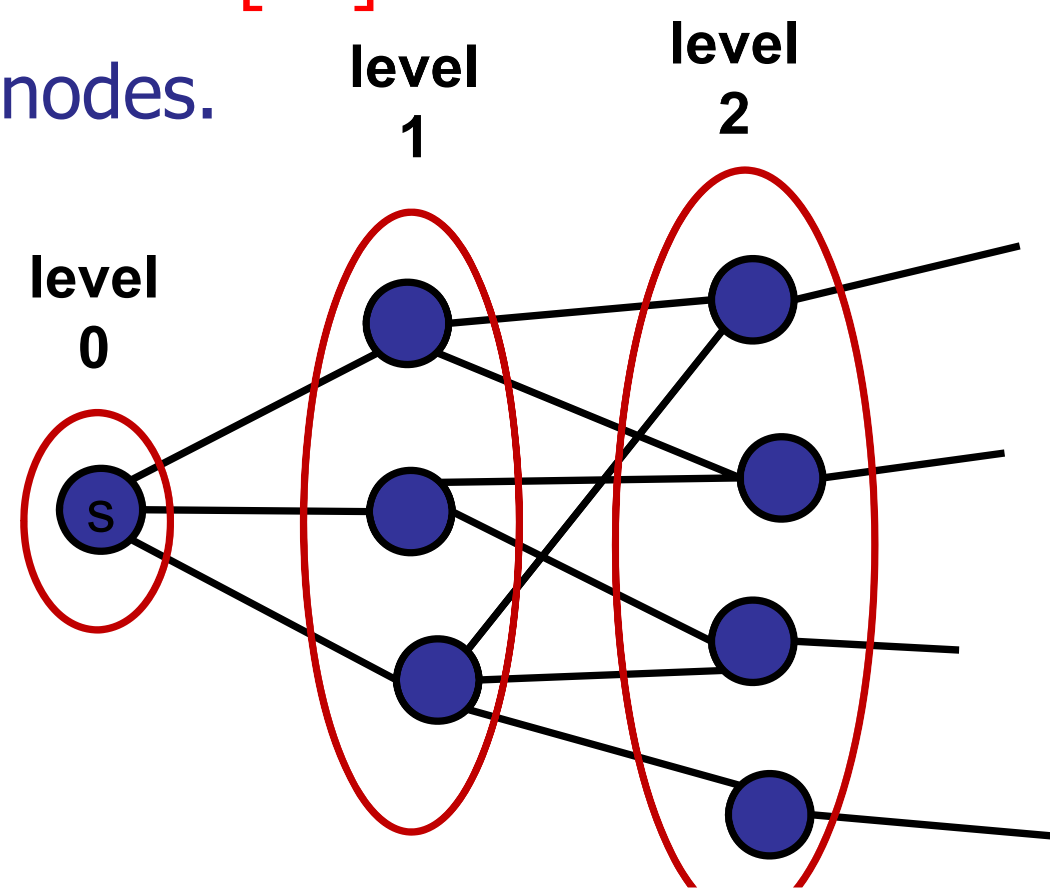
Graph representation:

- Adjacency list

Searching a graph

Breadth-First Search:

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

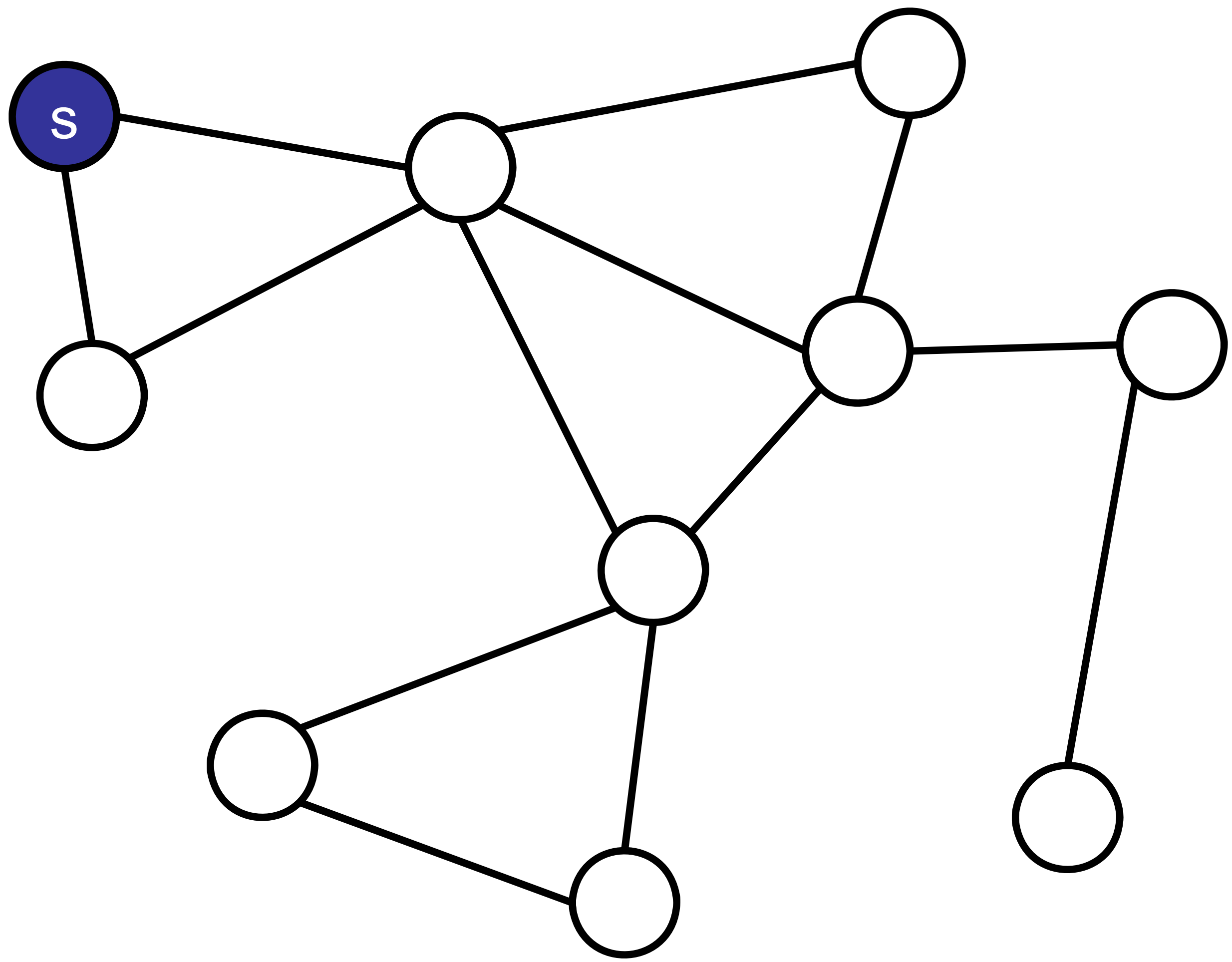


Searching a graph

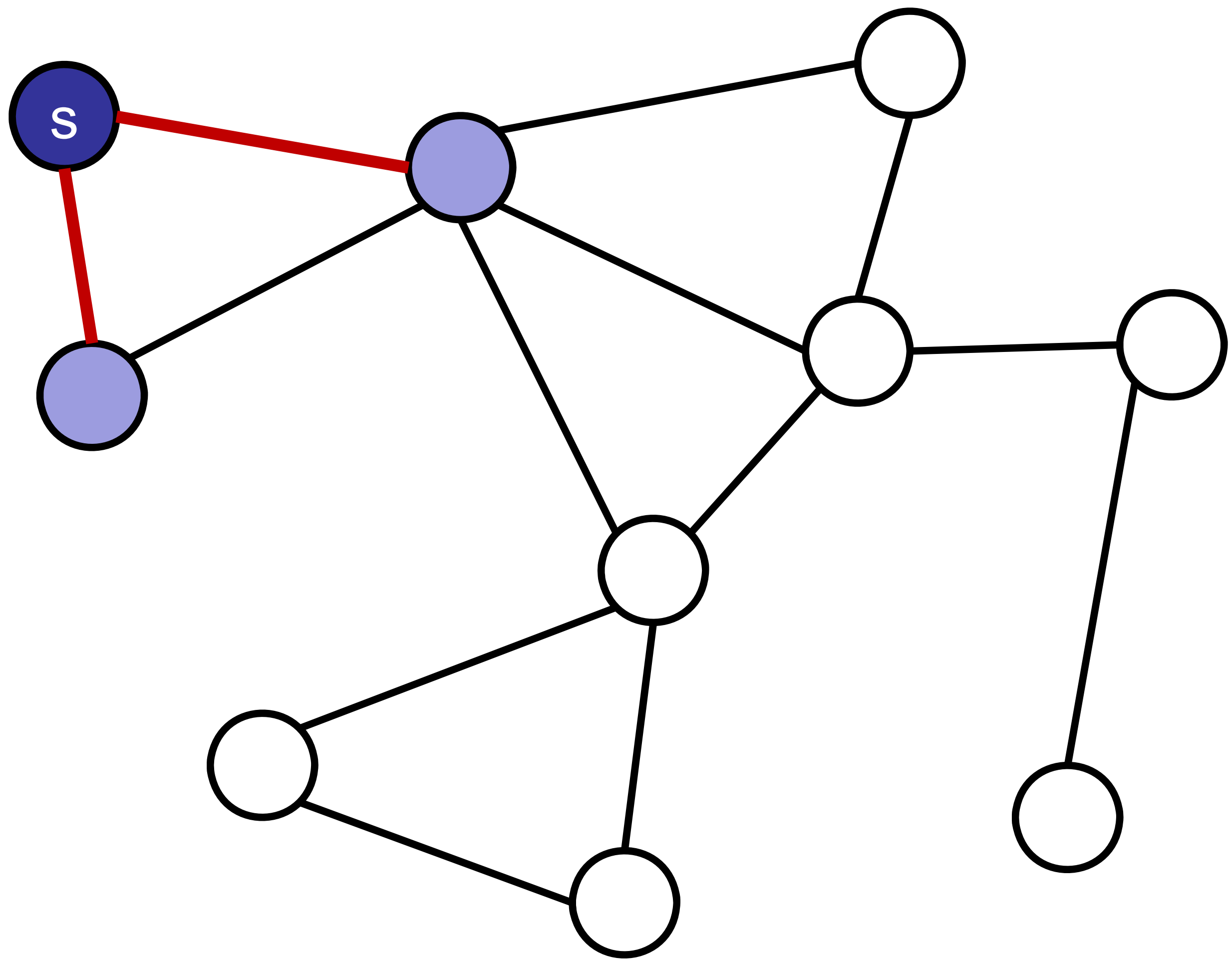
Breadth-First Search:

```
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
```

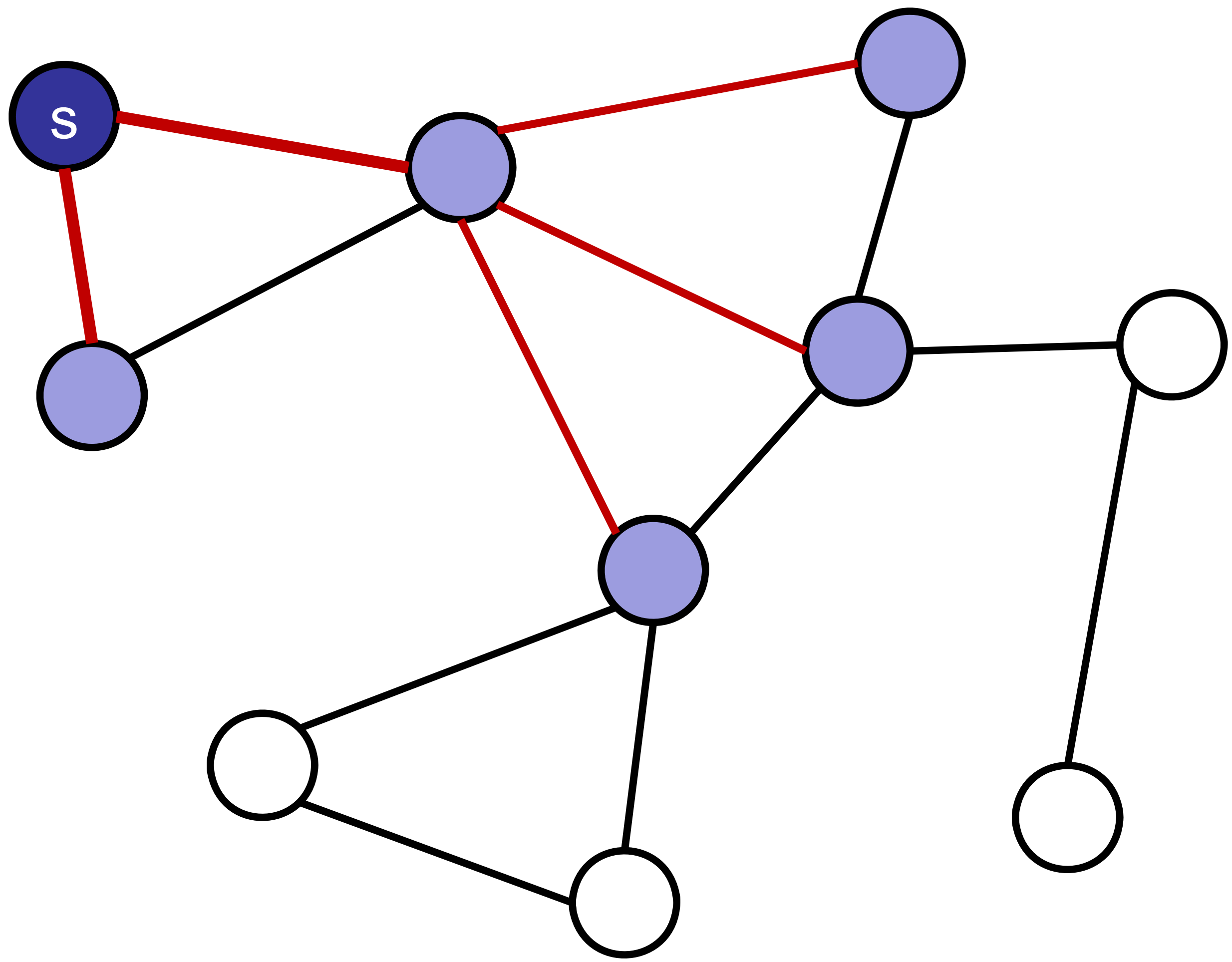
Breadth First Search



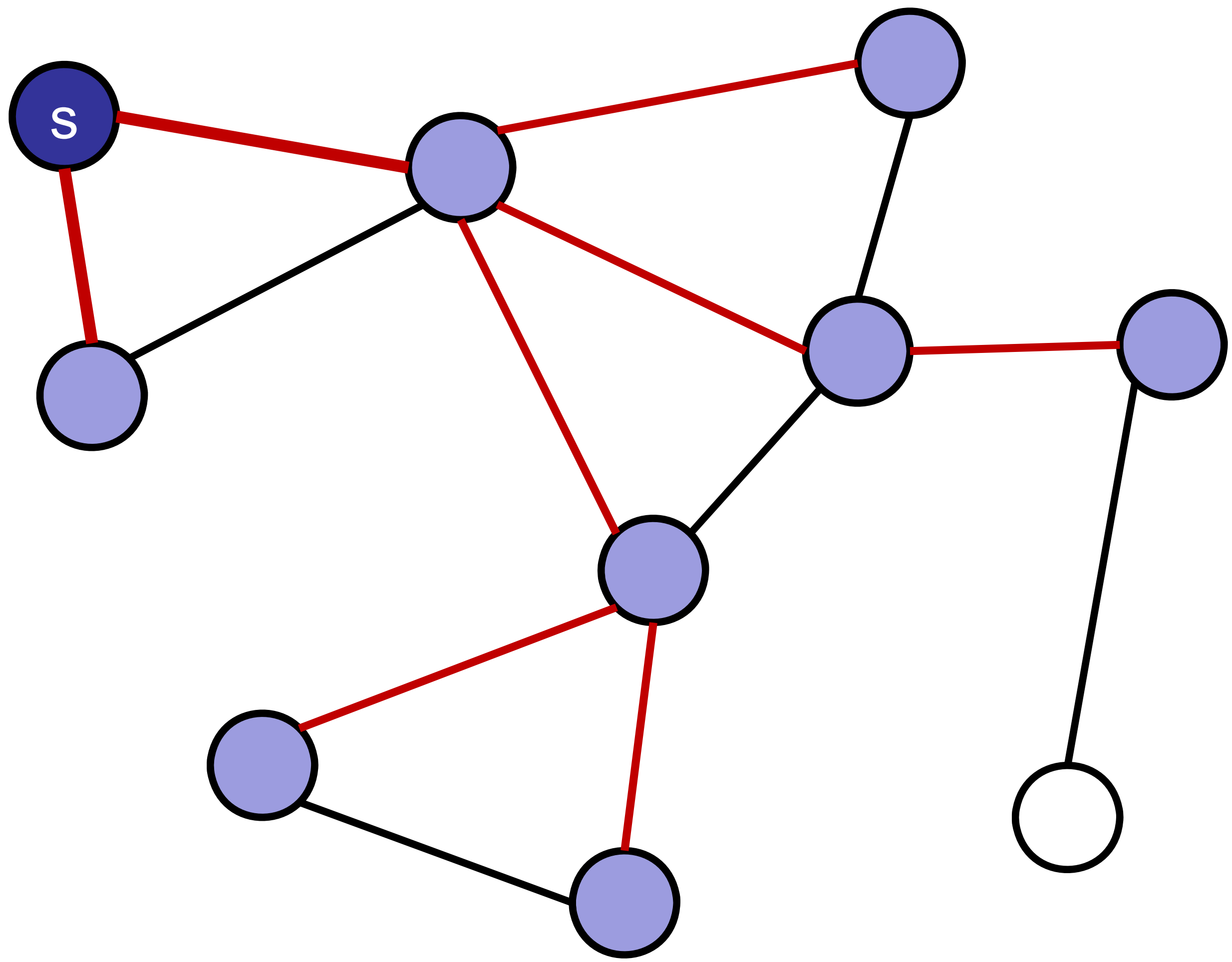
Breadth First Search



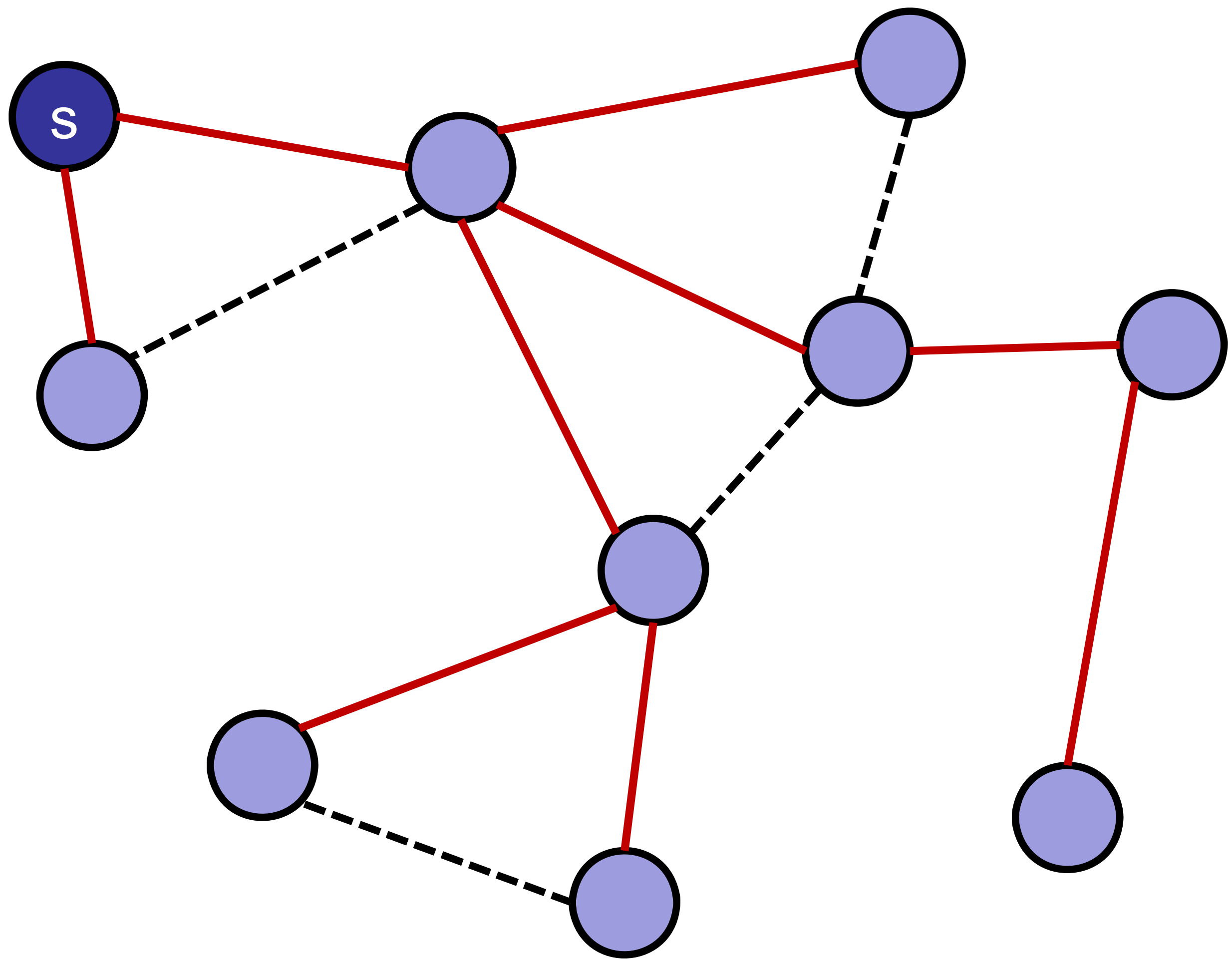
Breadth First Search



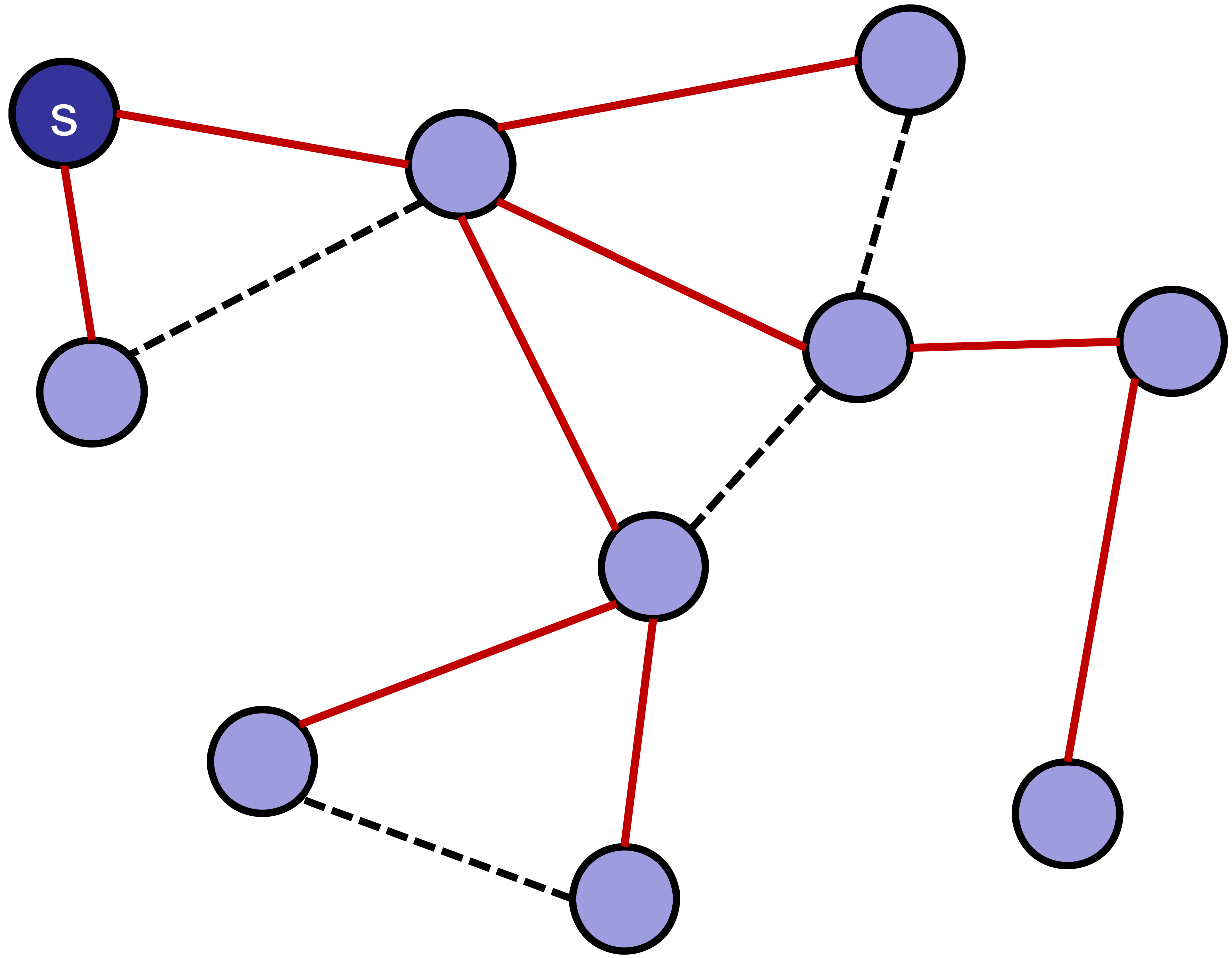
Breadth First Search



Breadth First Search



Breadth First Search

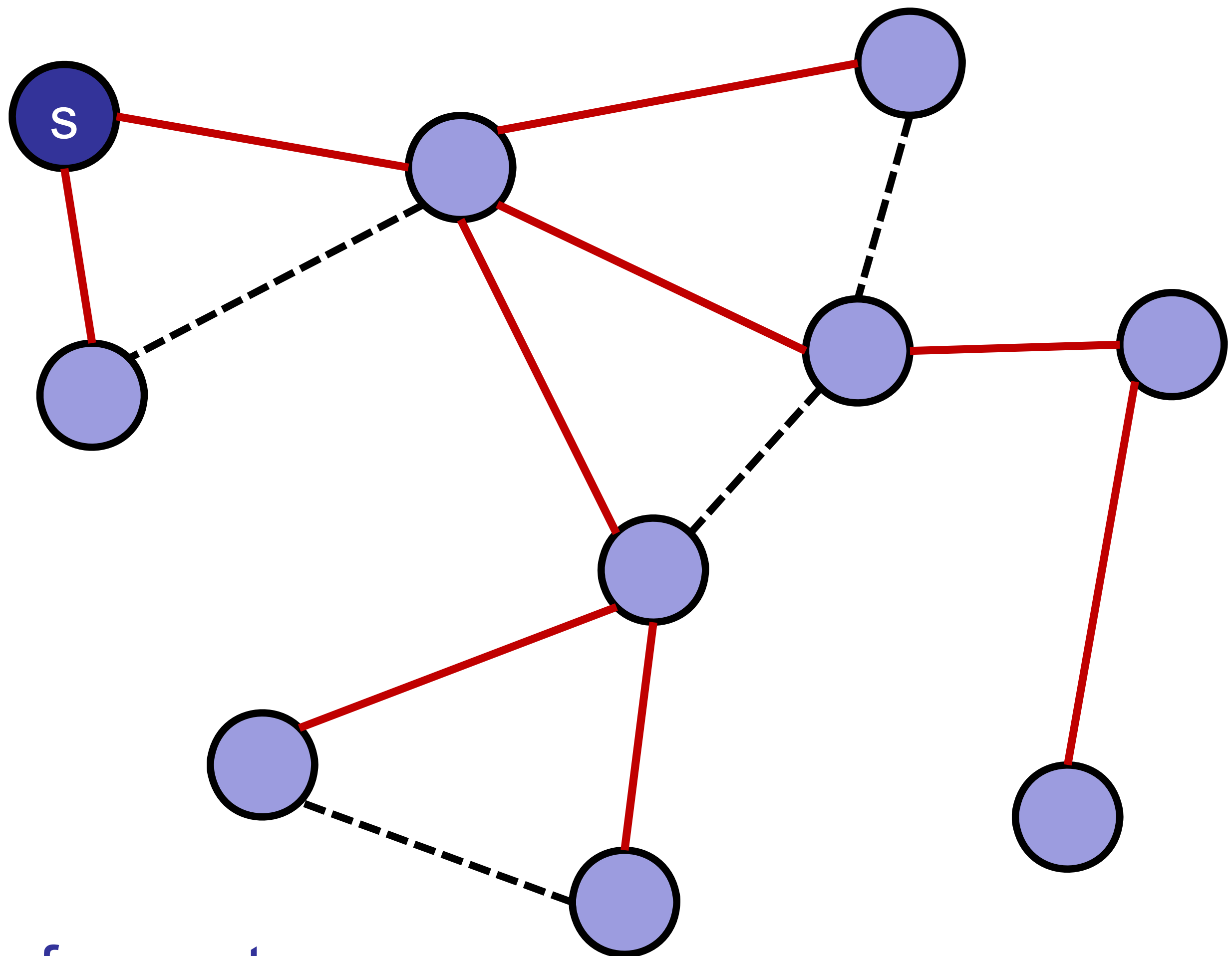


ARCHIPELAGO

is open

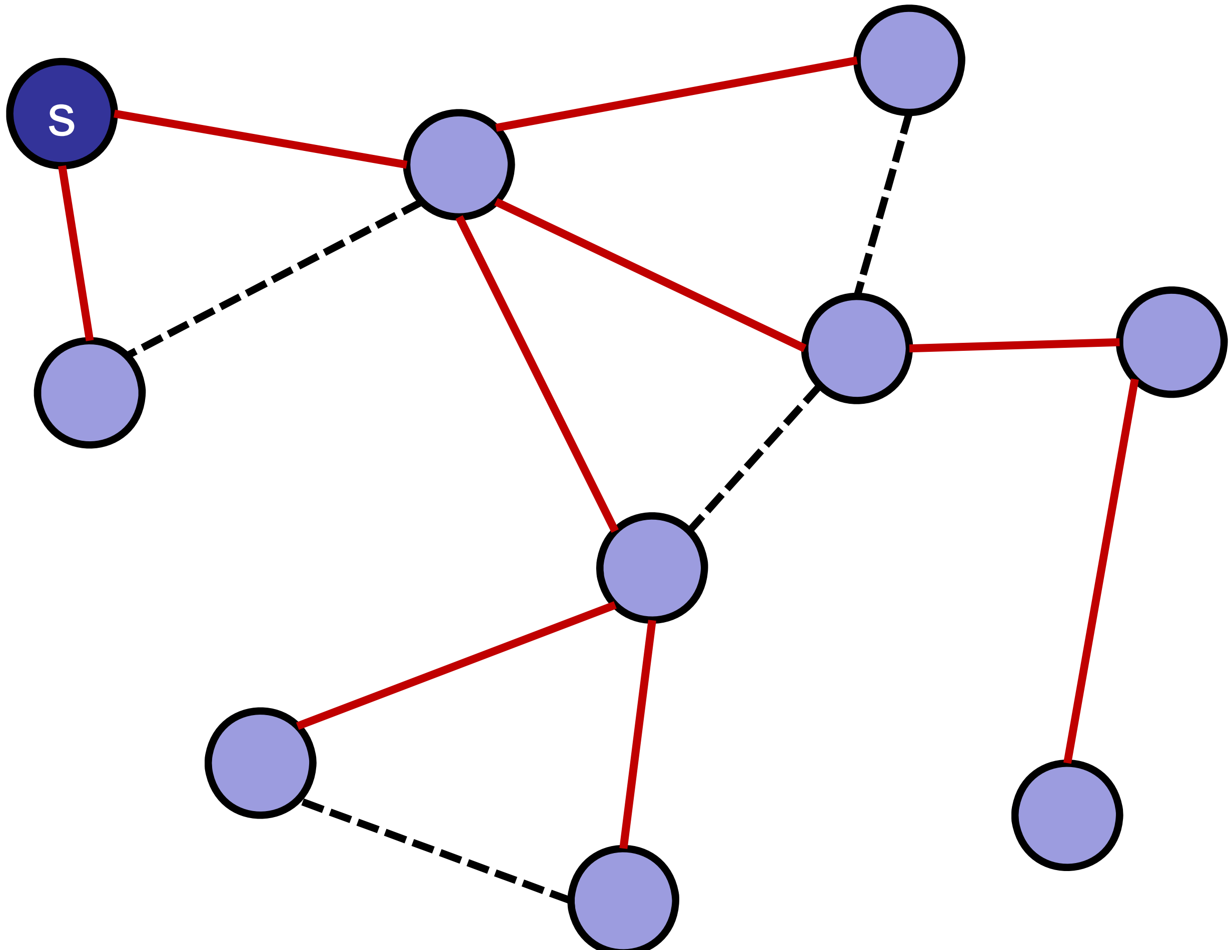
What are the properties of the parent edges?

Breadth First Search



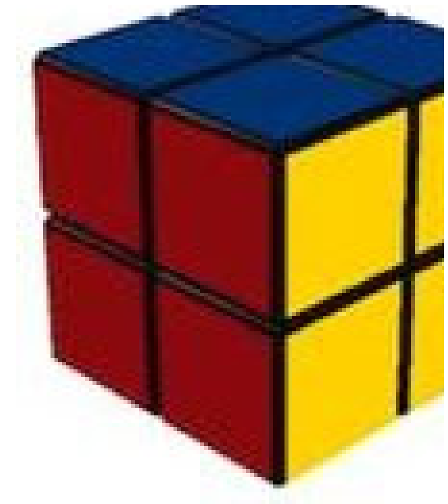
1. Parent edges form a tree
   (No cycles.)

Breadth First Search

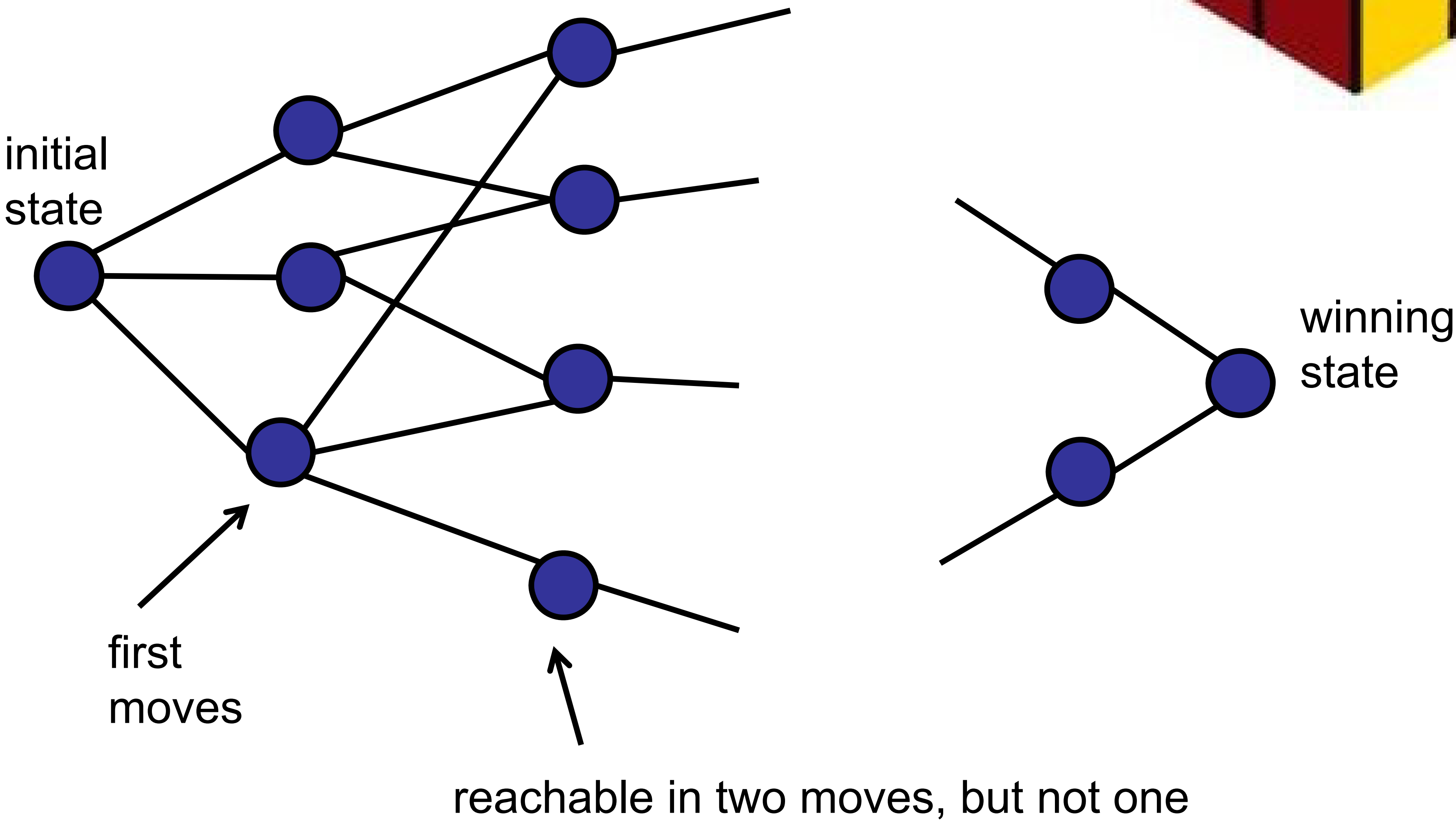


2. Parent edges are shortest paths from s.

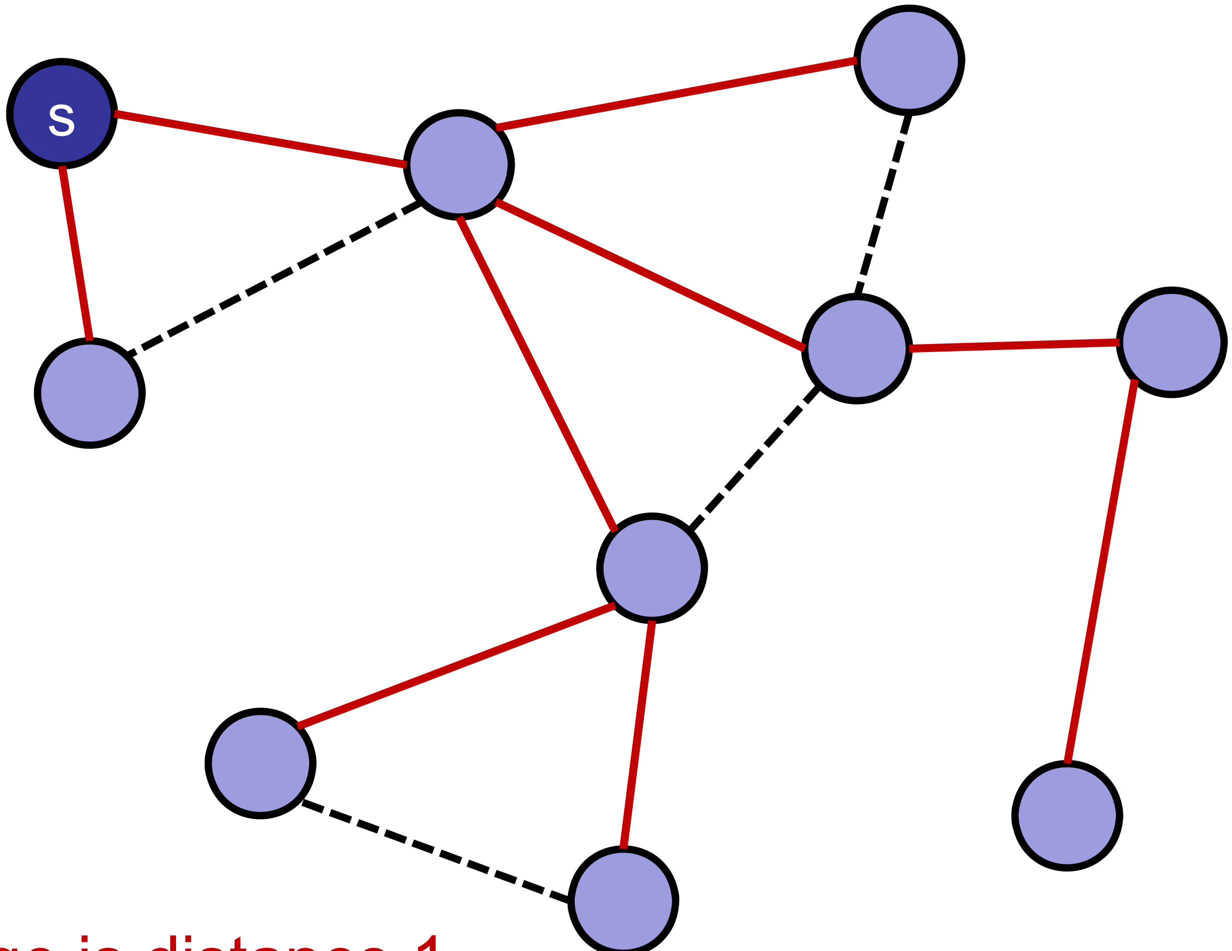
2 x 2 x 2 Rubik's Cube



Layers define shortest paths:



Breadth First Search



Beware: each edge is distance 1.

Next week: graphs with distances on the edges.

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)

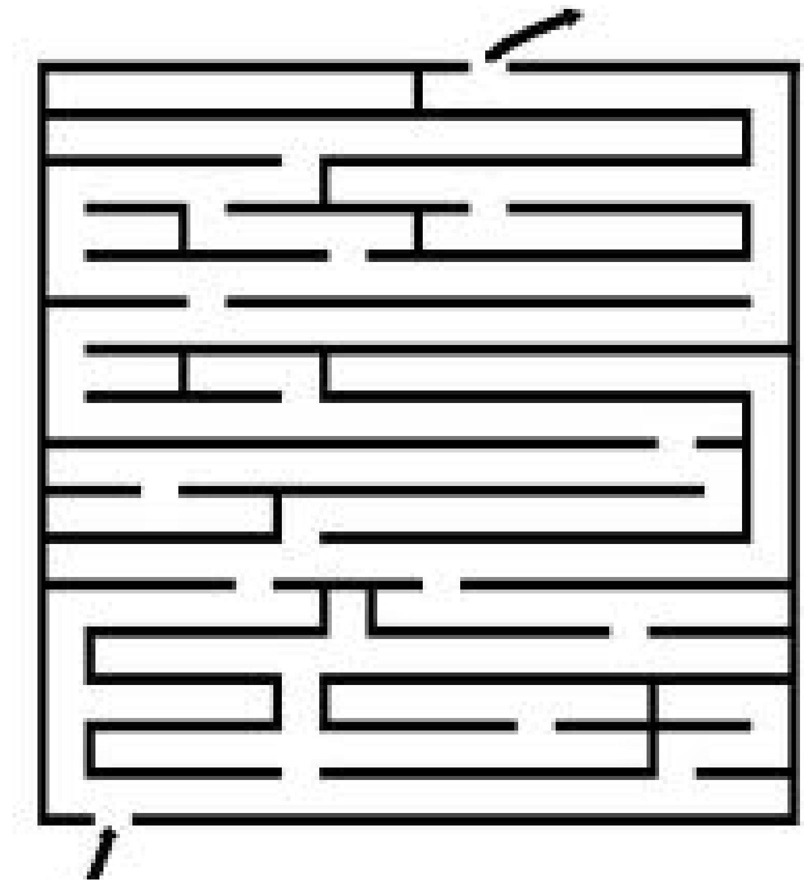
Graph representation:

- Adjacency list

Depth-First Search

Exploring a maze:

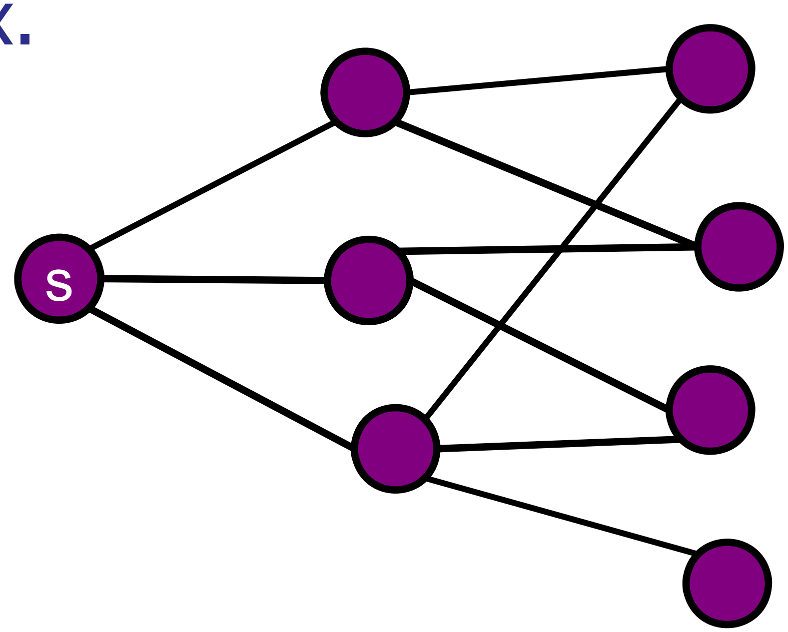
- Follow path until stuck.
- Backtrack along breadcrumbs until reach unexplored neighbor.
- Recursively explore.



Searching a graph

Depth-First Search:

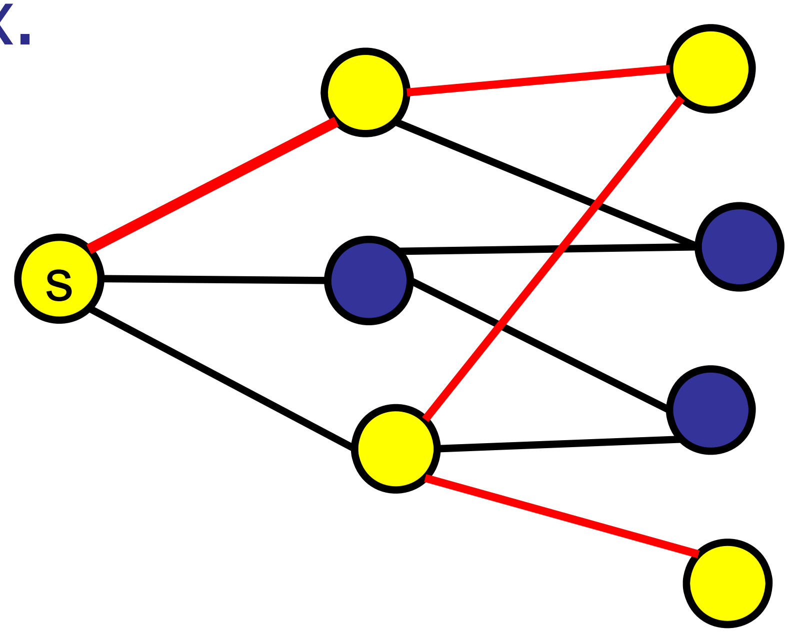
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don't repeat a vertex.



Searching a graph

Depth-First Search:

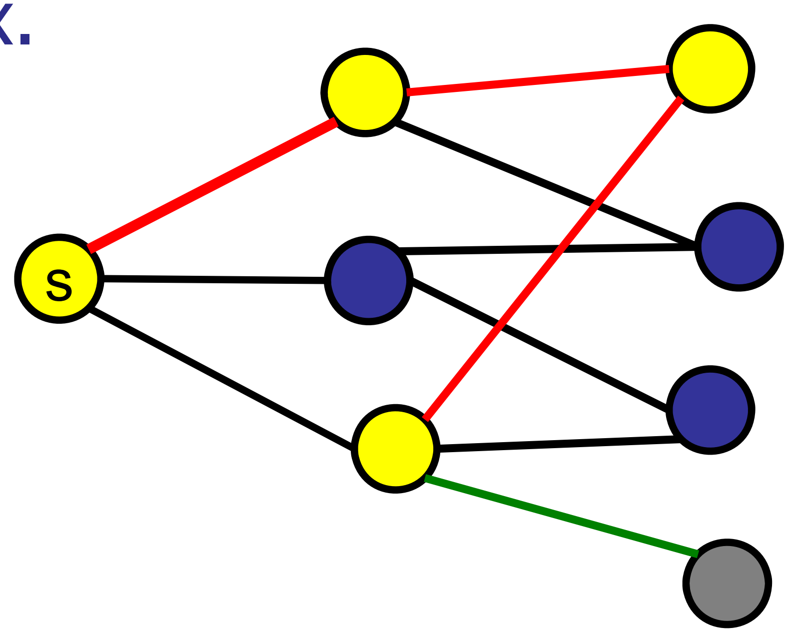
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don't repeat a vertex.



Searching a graph

Depth-First Search:

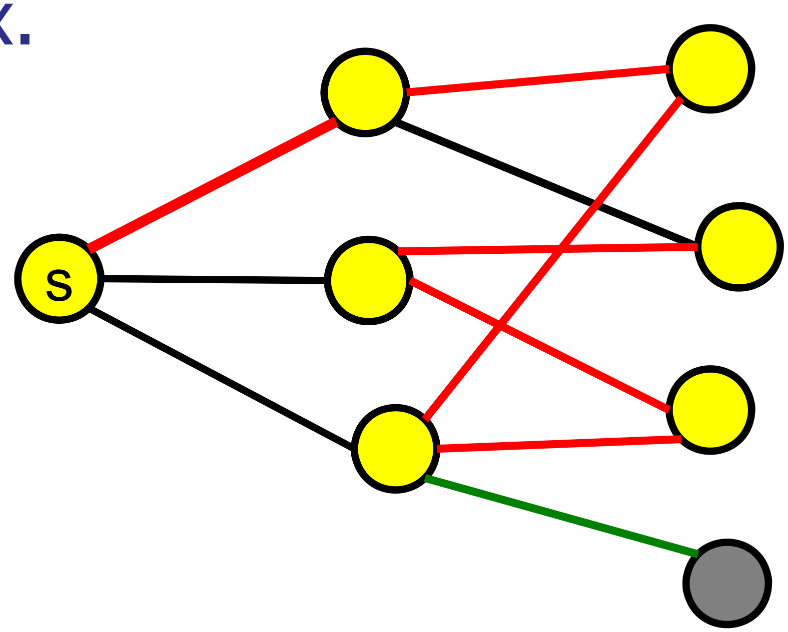
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don’t repeat a vertex.



Searching a graph

Depth-First Search:

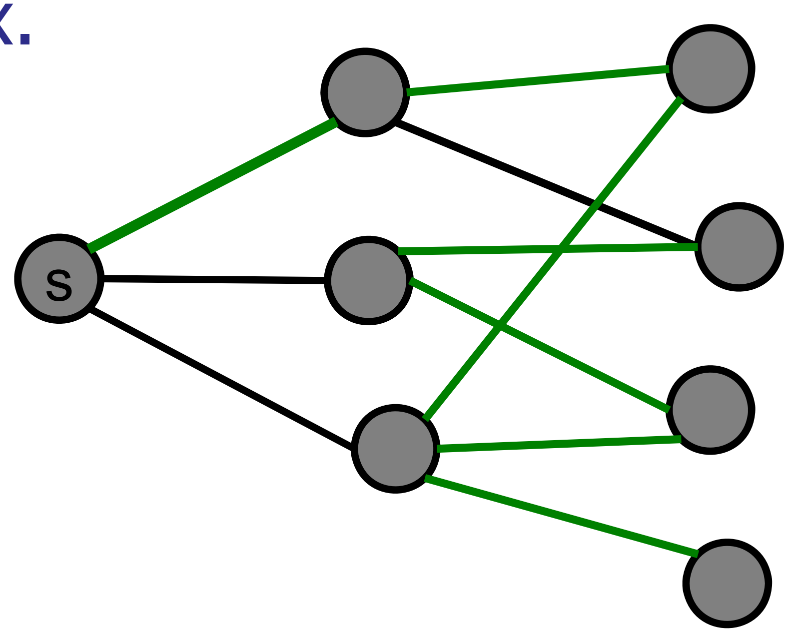
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don't repeat a vertex.



Searching a graph

Depth-First Search:

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



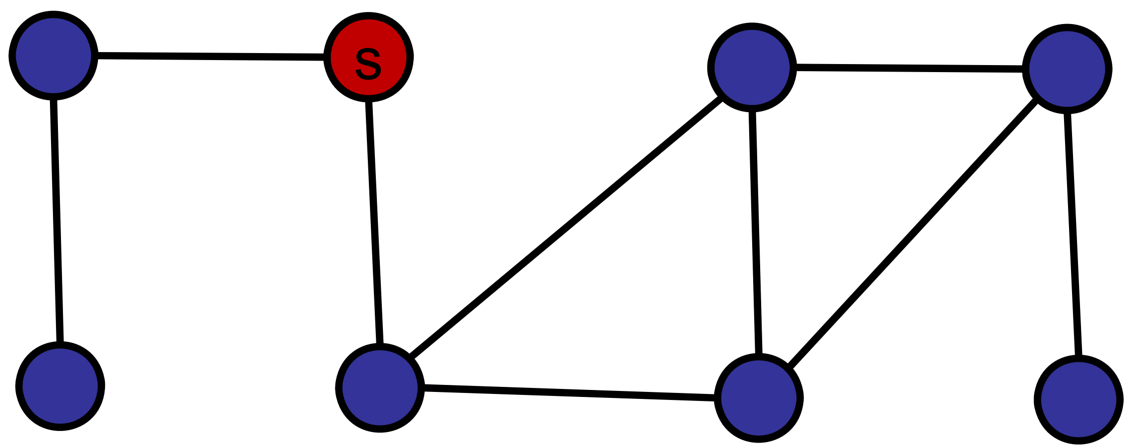
Depth-First Search

```
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);
     }
  }
}
```

Depth-First Search

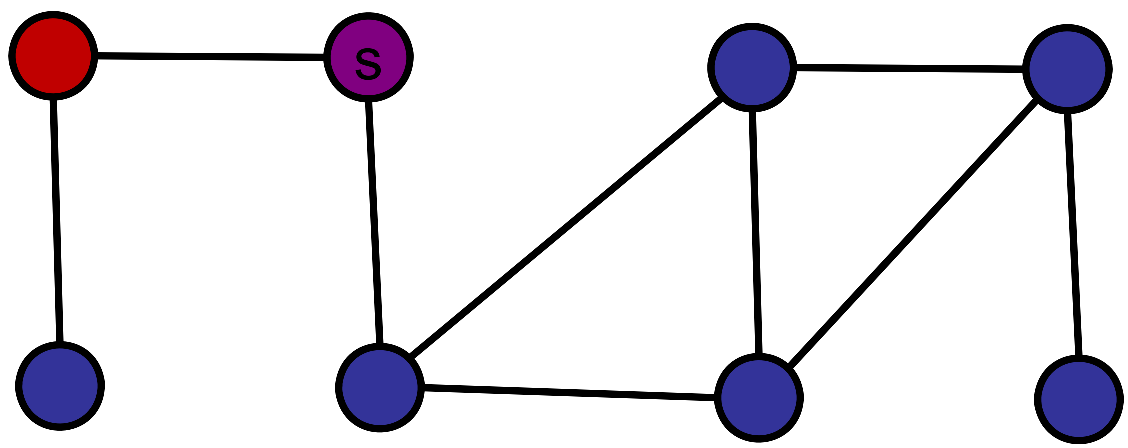
```
DFS(Node[] nodeList){
  boolean[] visited = new boolean[nodeList.length];
  Arrays.fill(visited, false);

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

Depth-First Search Example



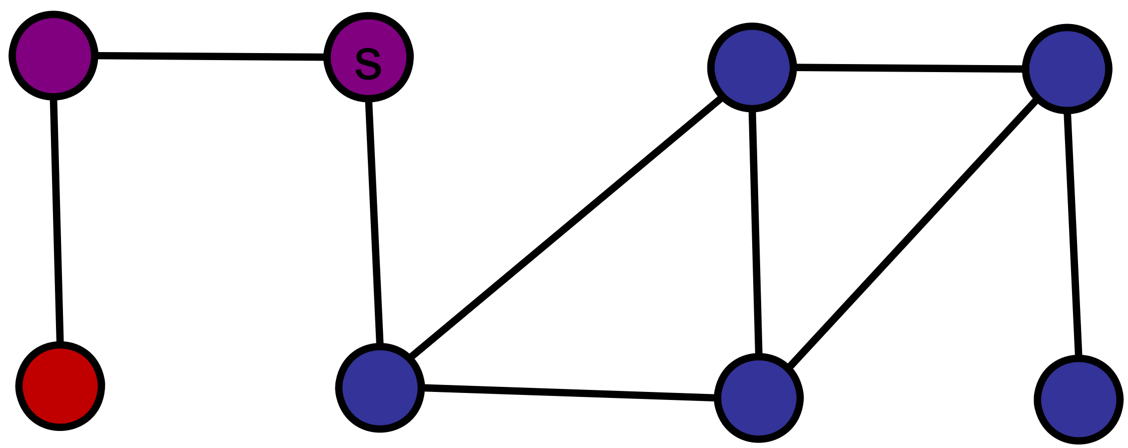
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



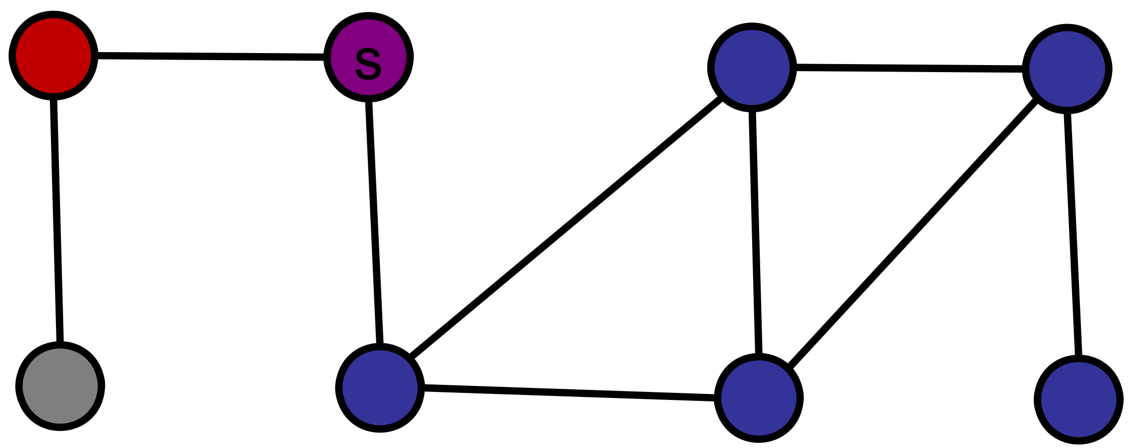
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



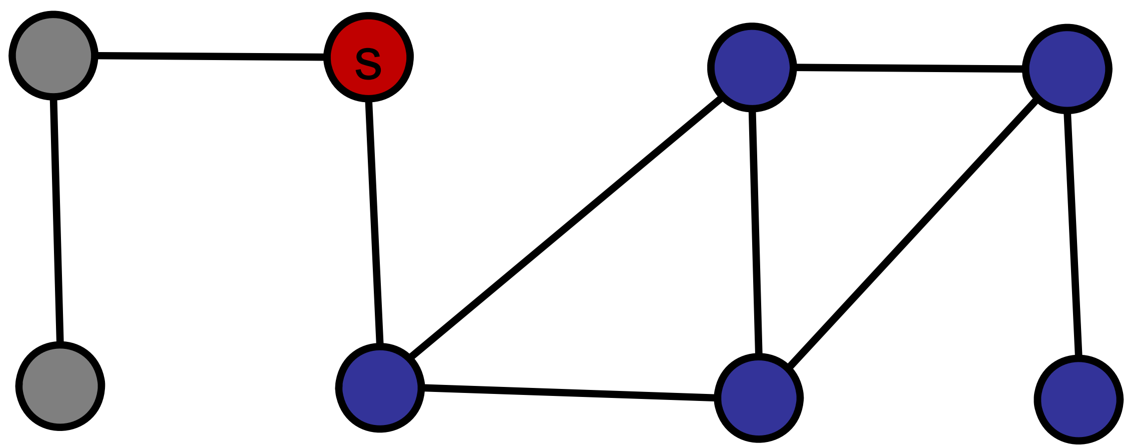
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



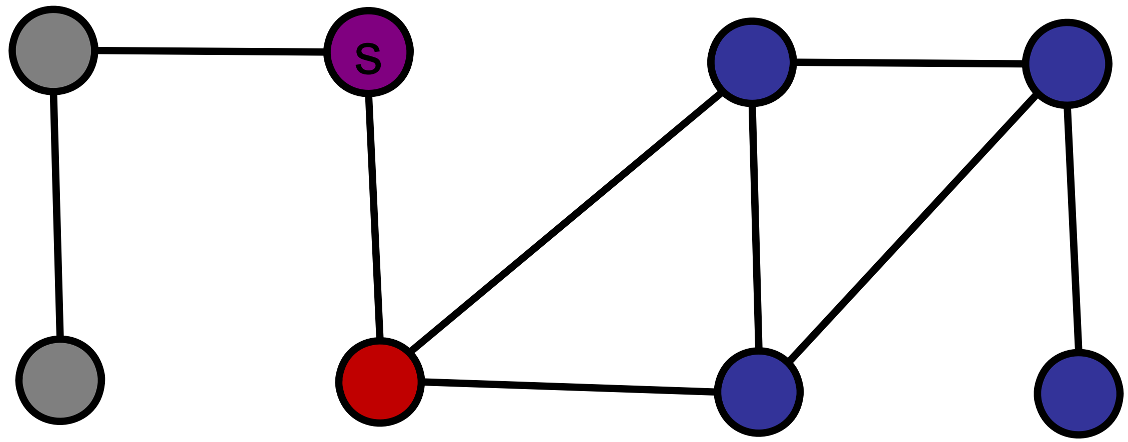
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



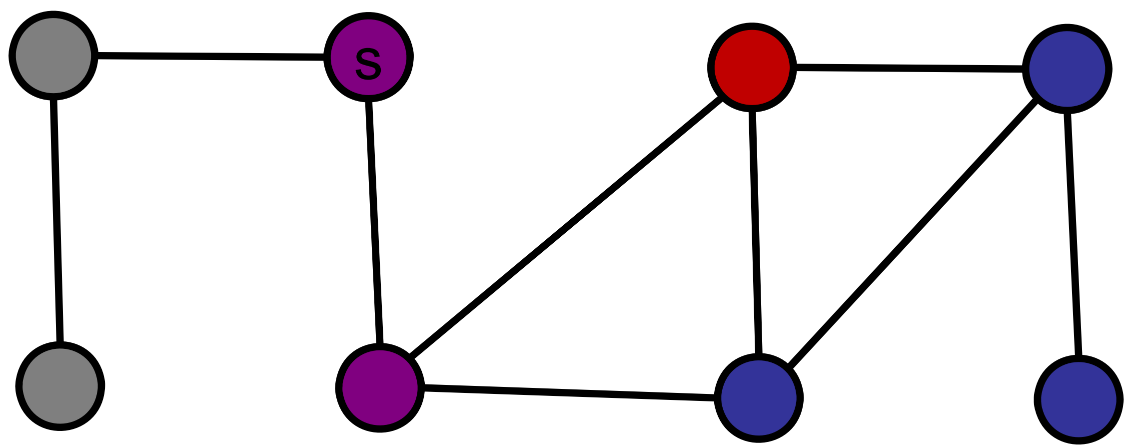
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



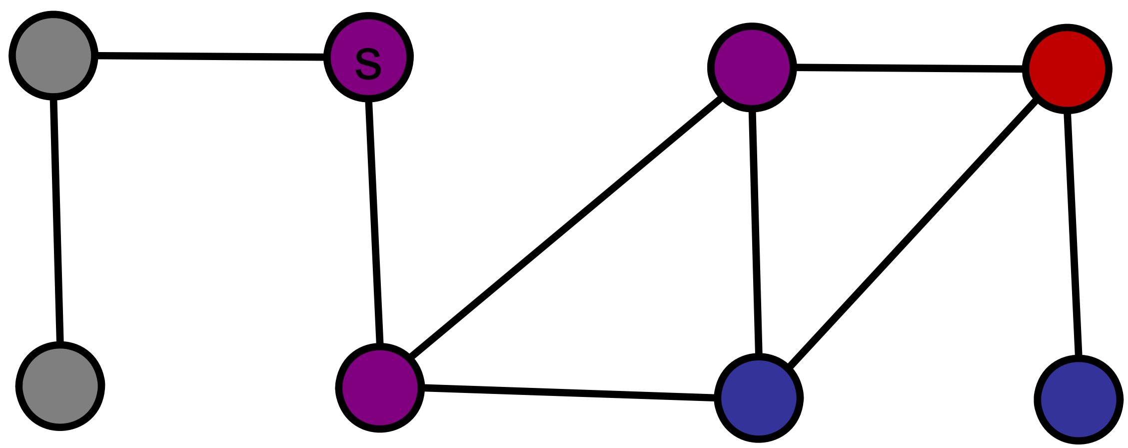
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



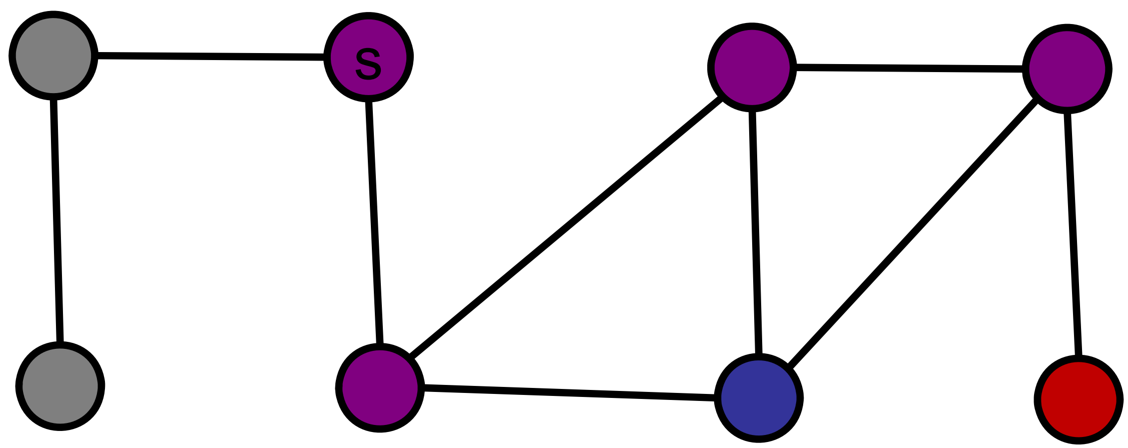
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



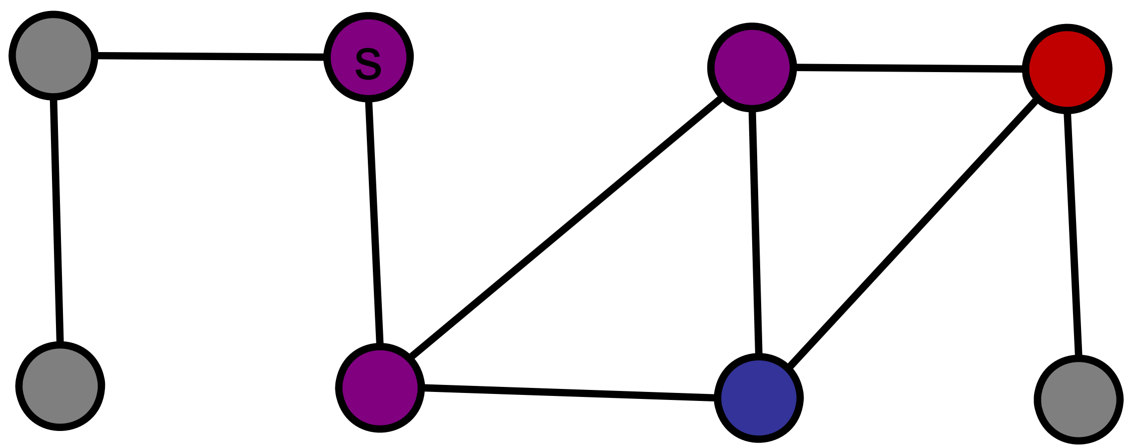
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



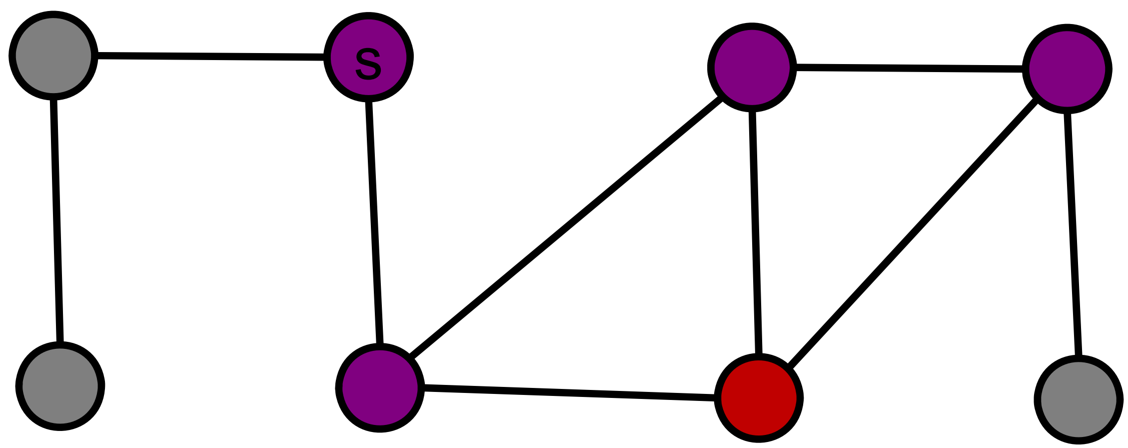
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



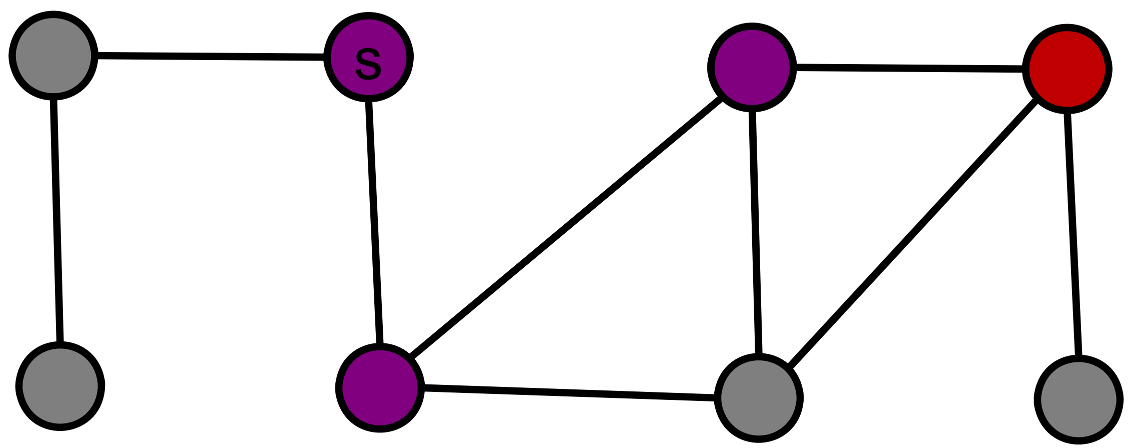
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



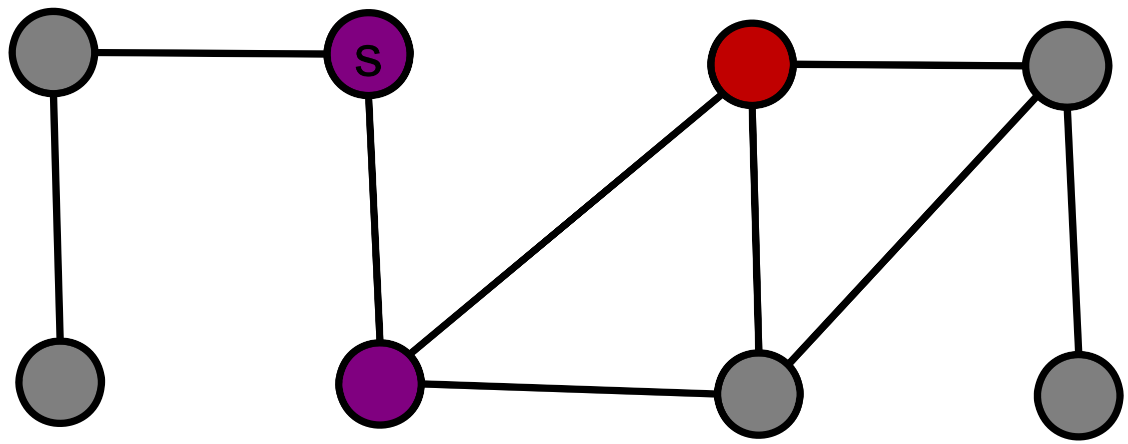
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



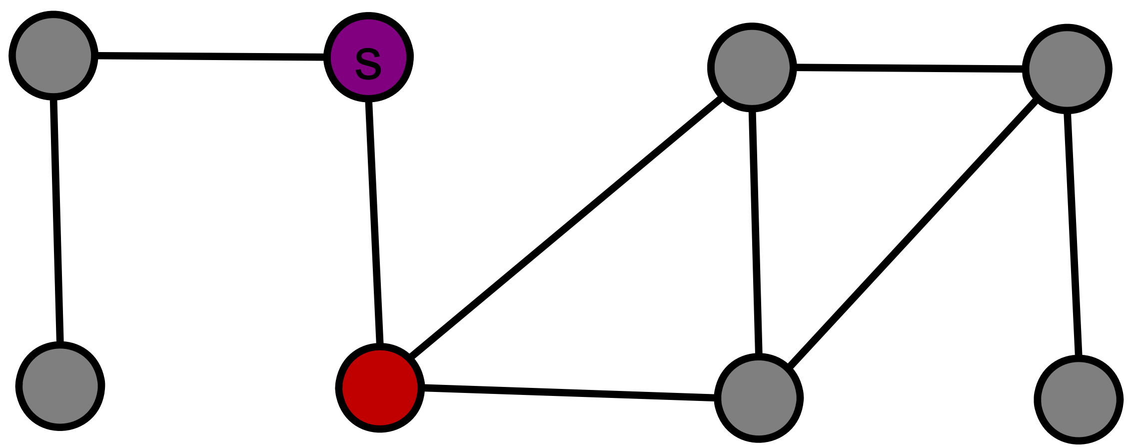
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



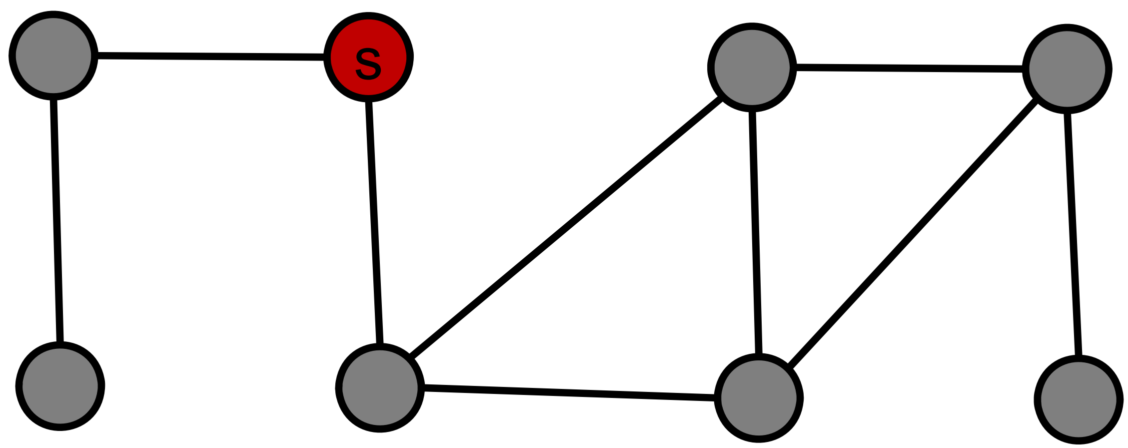
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



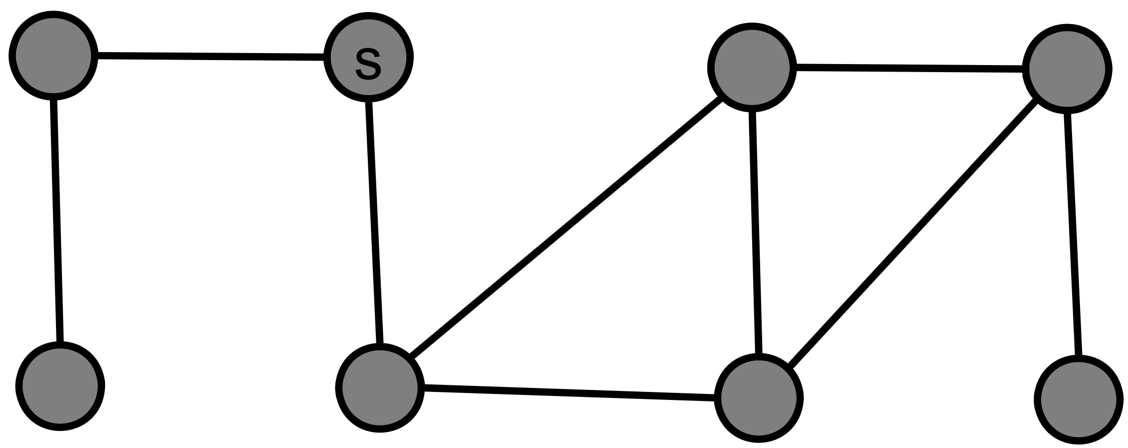
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Depth-First Search Example



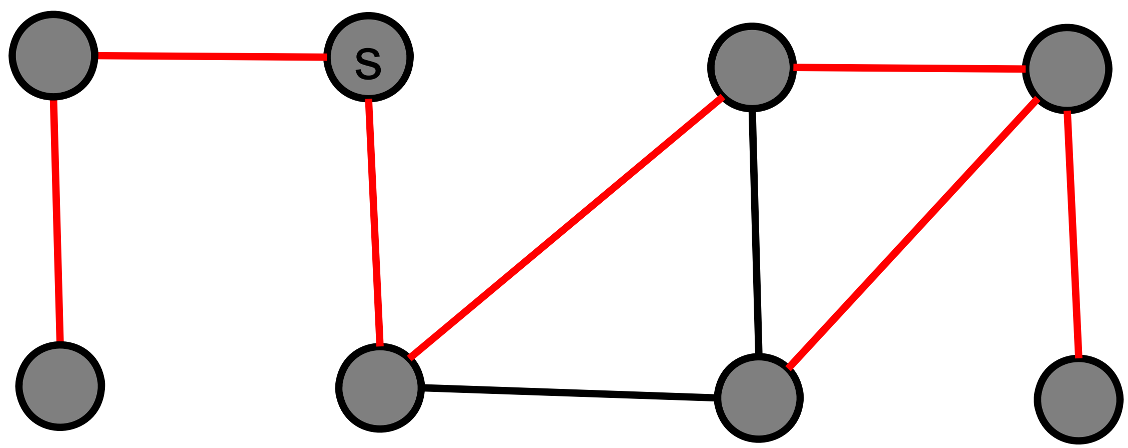
Red = active frontier

Purple = next

Gray = visited

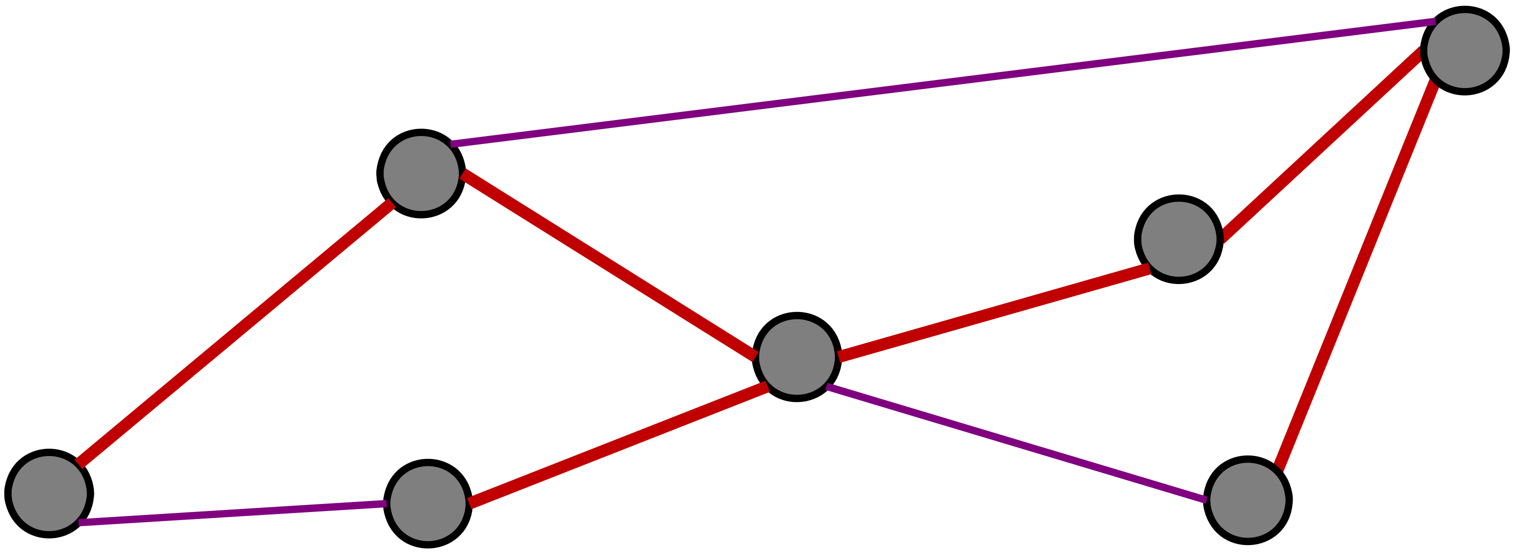
Blue = unvisited

Depth-First Search Example



Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

DFS parent edges



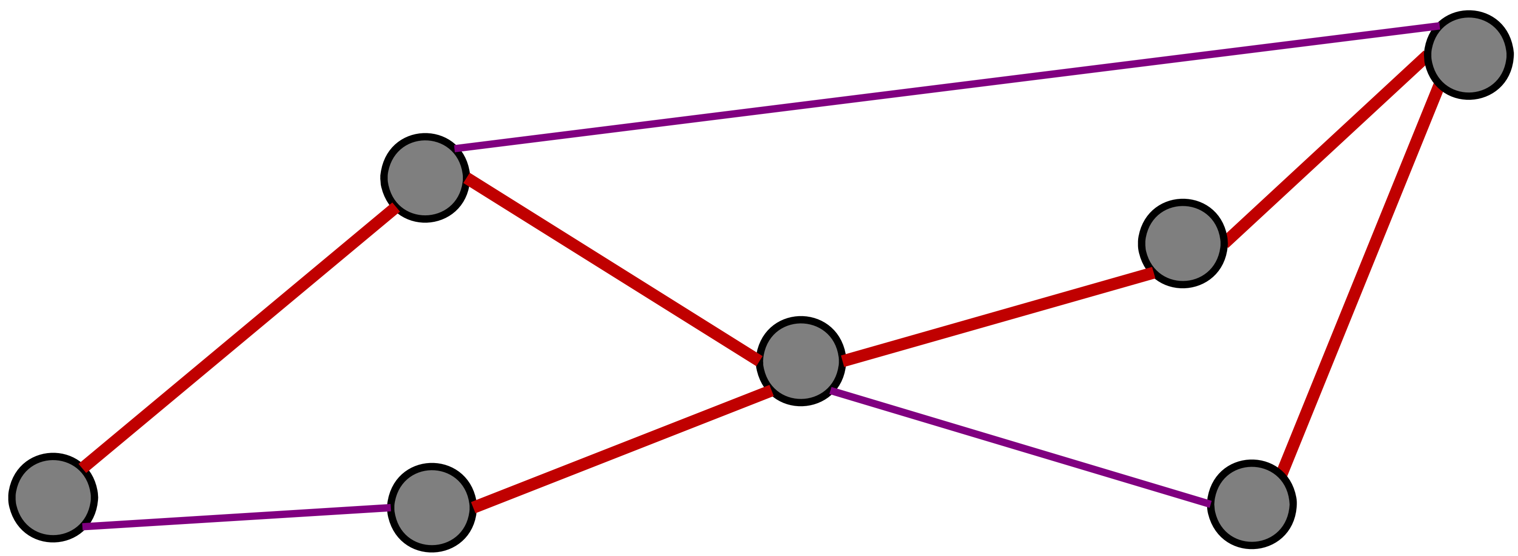
Red = Parent Edges
Purple = Non-parent edges

Which is true? (More than one may apply.)

1. DFS parent graph is a cycle.
2. DFS parent graph is a tree.
3. DFS parent graph has low-degree.
4. DFS parent graph has low diameter.
5. None of the above.

ARCHIPELAGO is open

DFS parent edges = tree



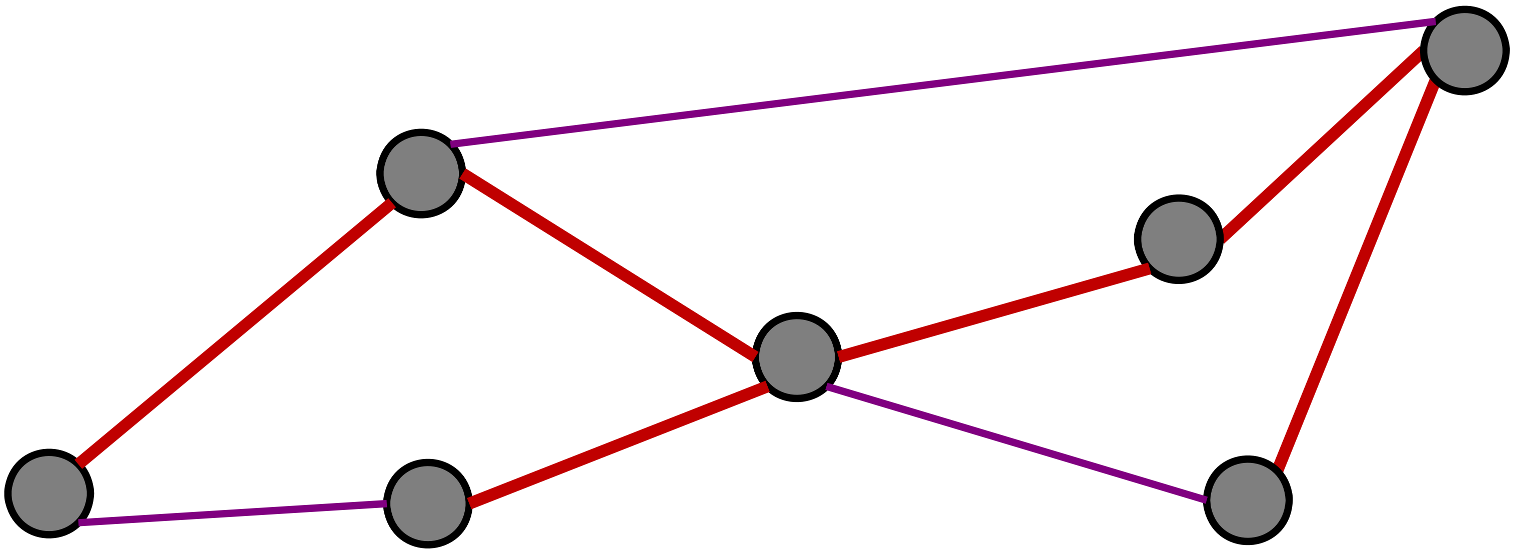
Red = Parent Edges
Purple = Non-parent edges

True or false:
DFS parent graph contains shortest paths.

1. True
2. False

ARCHIPELAGO
is open

DFS parent edges = tree



Red = Parent Edges
Purple = Non-parent edges

Note: not shortest paths!

The running time of DFS is:

1. O(V)
2. O(E)
3. O(V+E)
4. O(VE)
5. O(V^2)
6. I have no idea.

ARCHIPELAGO

is open

Depth-First Search

Analysis:

- DFS-visit called only once per node. O(V)
  - After visited, never call DFS-visit again.
- In DFS-visit, each neighbor is enumerated. O(E)

If the graph is stored as an adjacency matrix, what is the running time of DFS?

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

Depth-First Search

Analysis:

- DFS-visit called only once per node.
  - After visited, never call DFS-visit again.

O(V)

- In DFS-visit, each neighbor is enumerated.

O(V)
per
node

To implement an iterative version of DFS:

1. Use a queue.
2. Use a stack.
3. Use a set.
4. Use a tree.

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.

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.

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?

Common Mistake

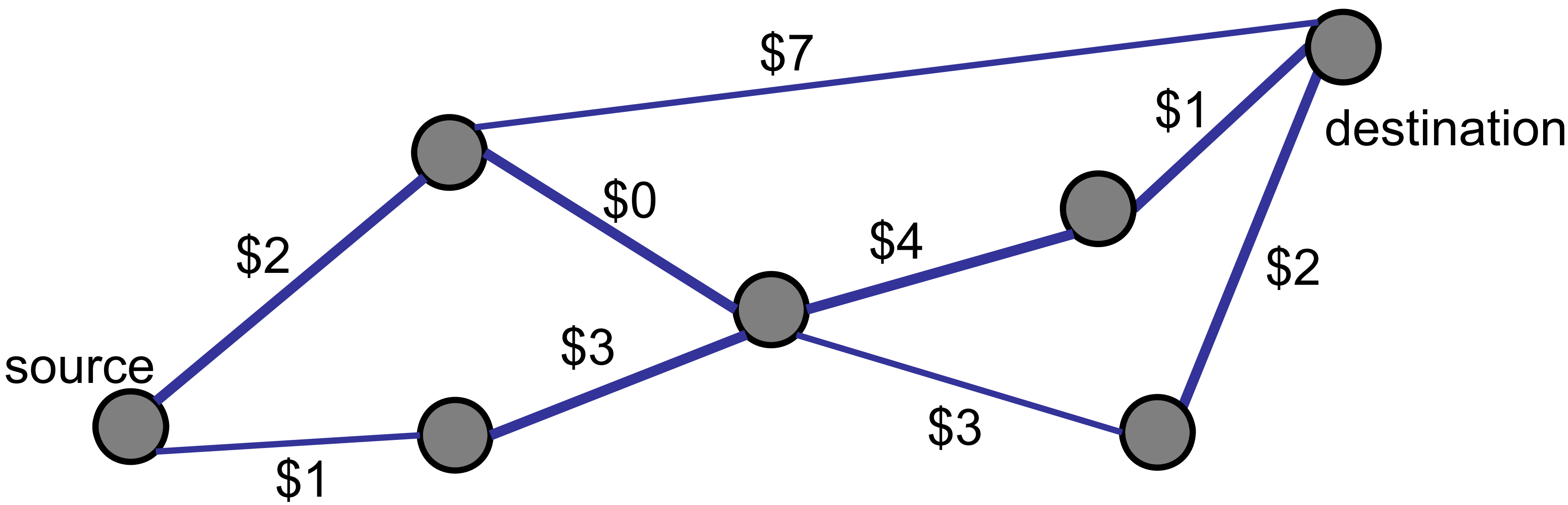
What do BFS and DFS solve?

- 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

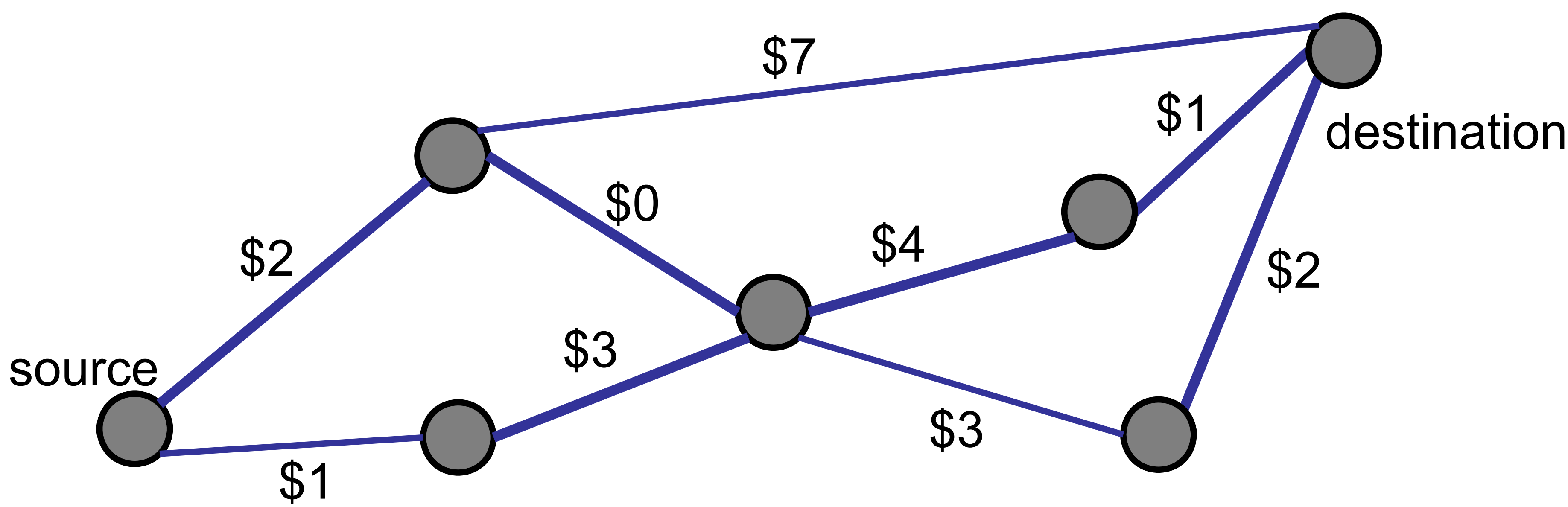
- Start at source s.
- Go to destination d.
- Each edge e earns money m(e).
- Find the path that makes the most money.



Example

NOT a solution:

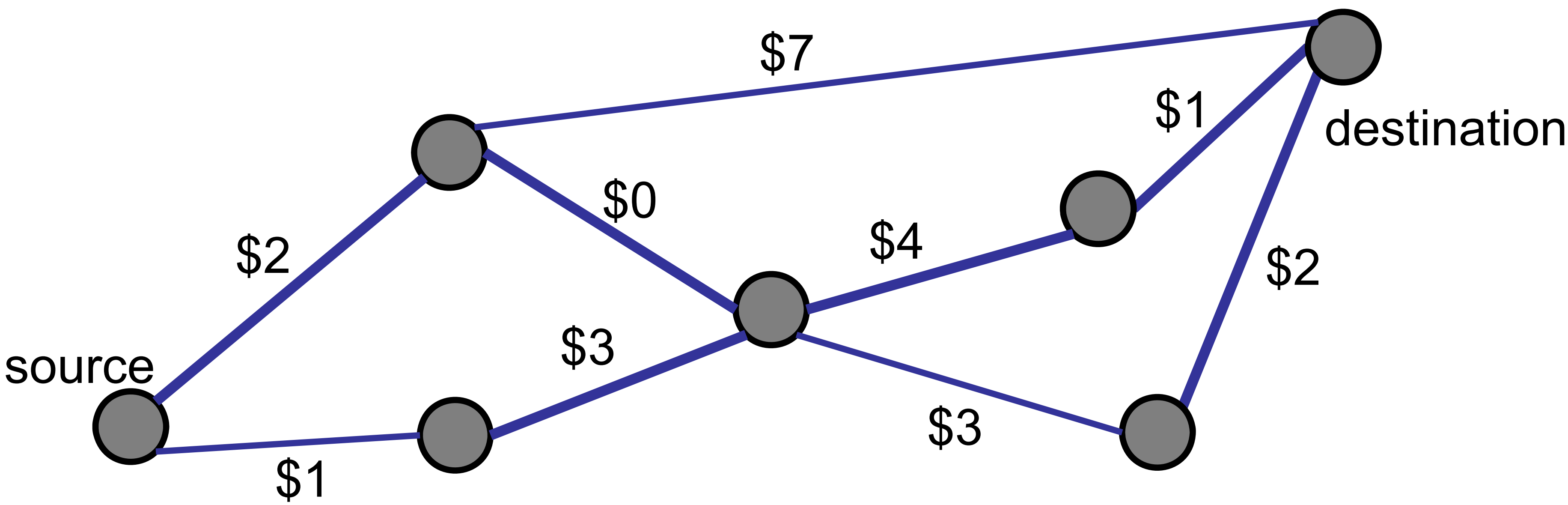
- Start at source s.
- Run BFS (or DFS) to explore every path.
- Keep track of the best path.



Example

Problem 1: Does not work.

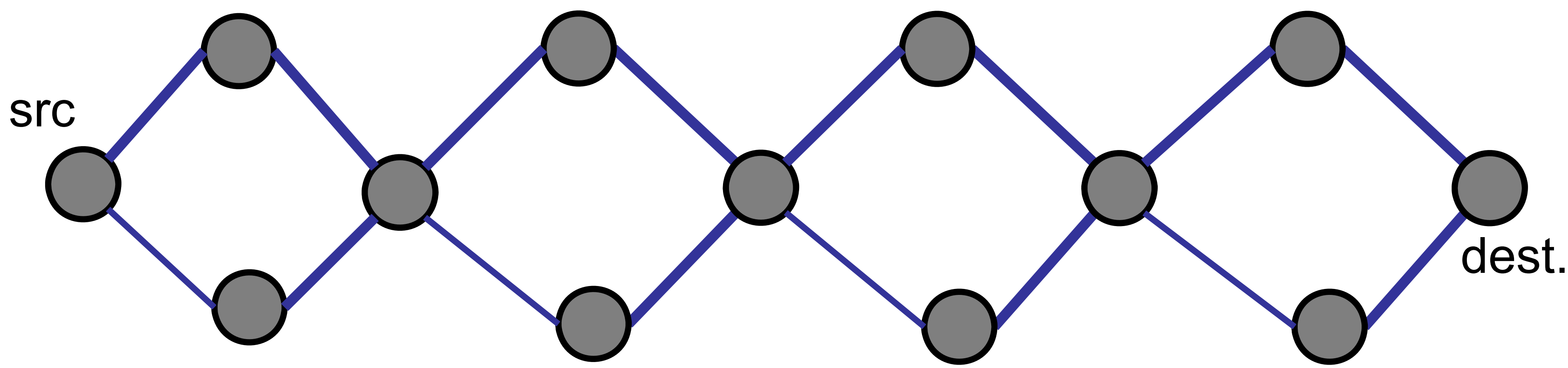
- DFS or BFS do NOT explore every path.
- Once a node is visited, it is never explored again.



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?~~

Roadmap

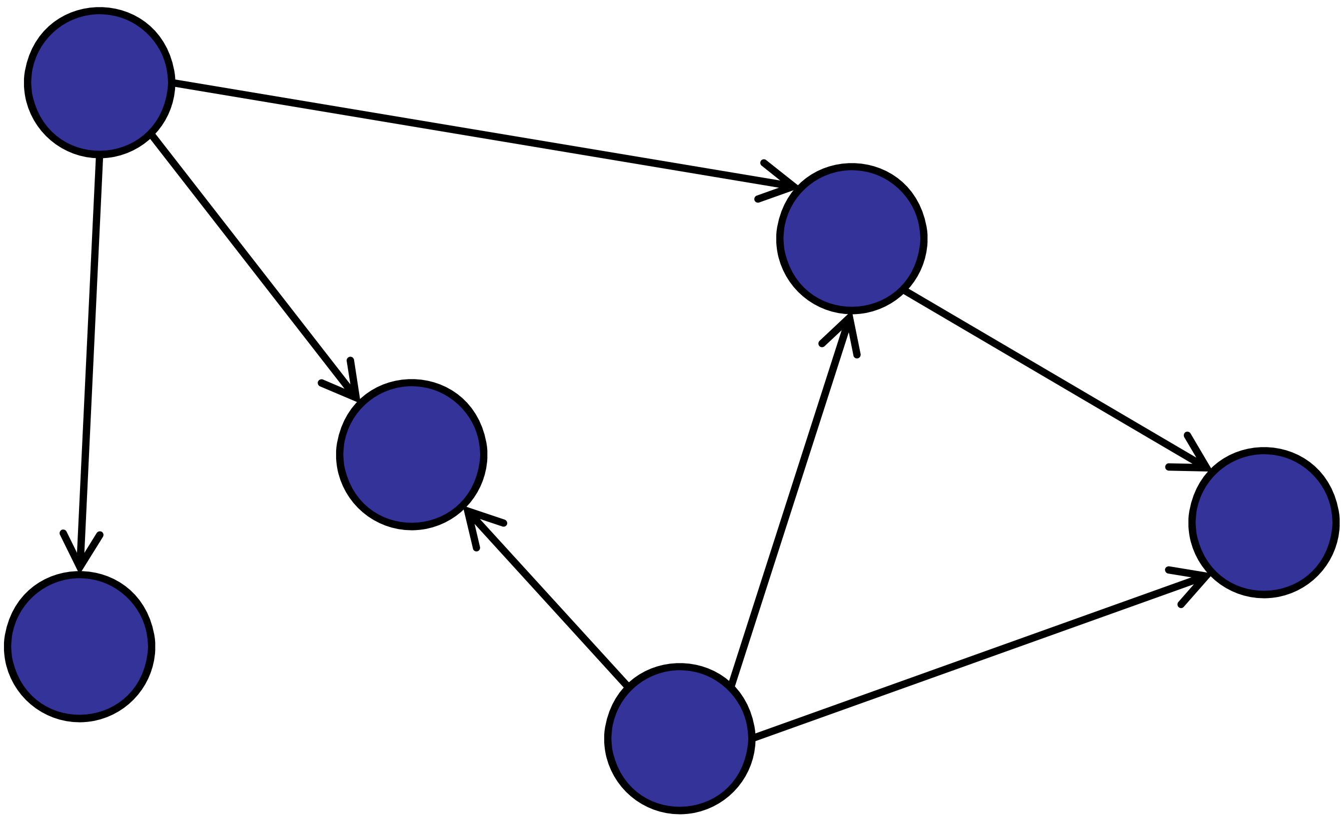
Directed Graphs

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

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

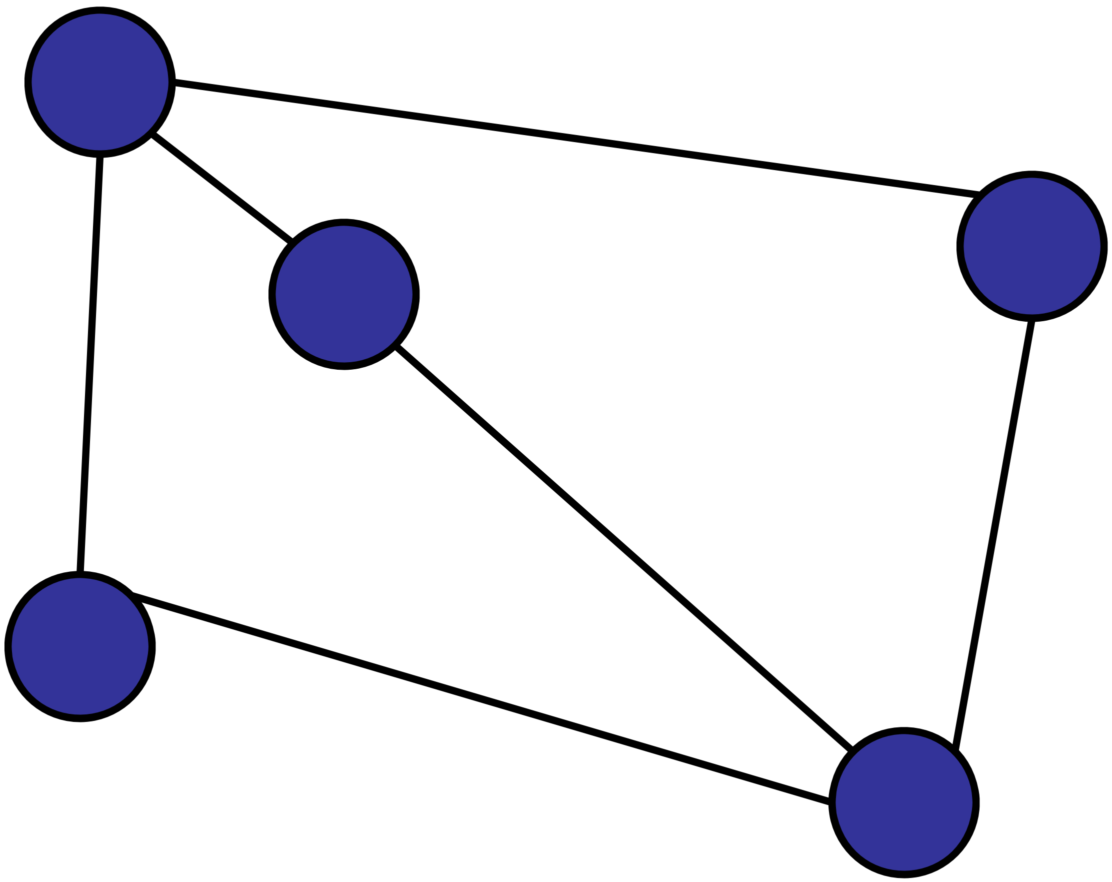
Is it a directed graph?

1. Yes
2. No.



Is it a directed graph?

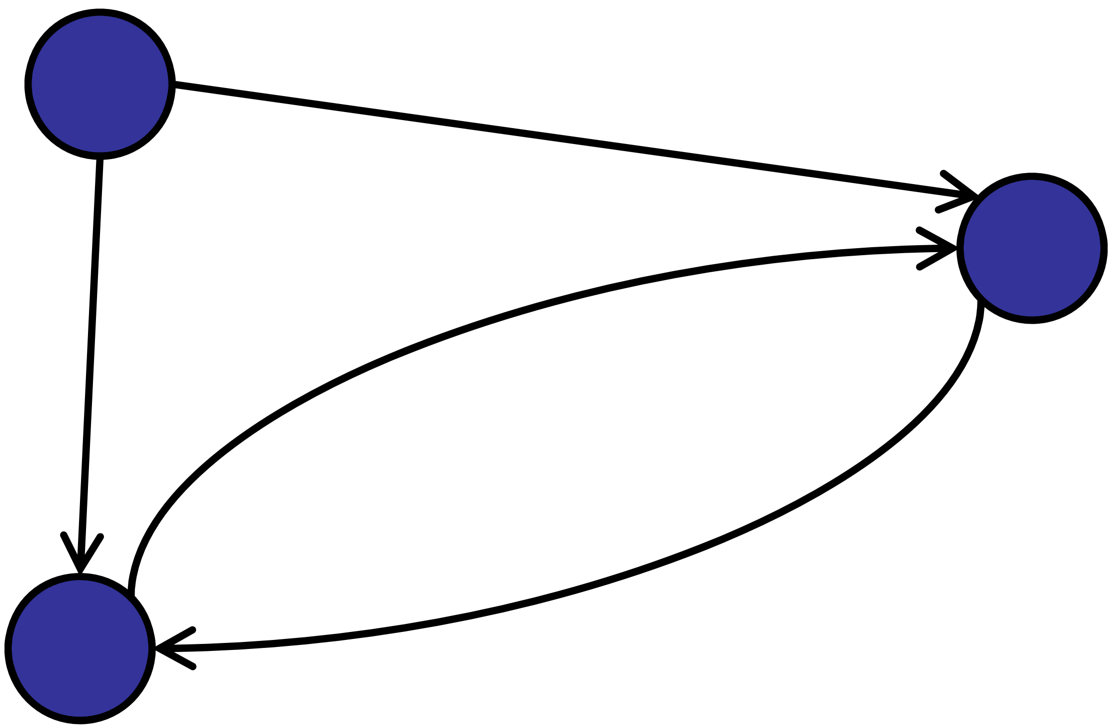
1. Yes
2. No.



Is it a directed graph?

1. Yes
2. No.

Yes or No on Zoom.



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?

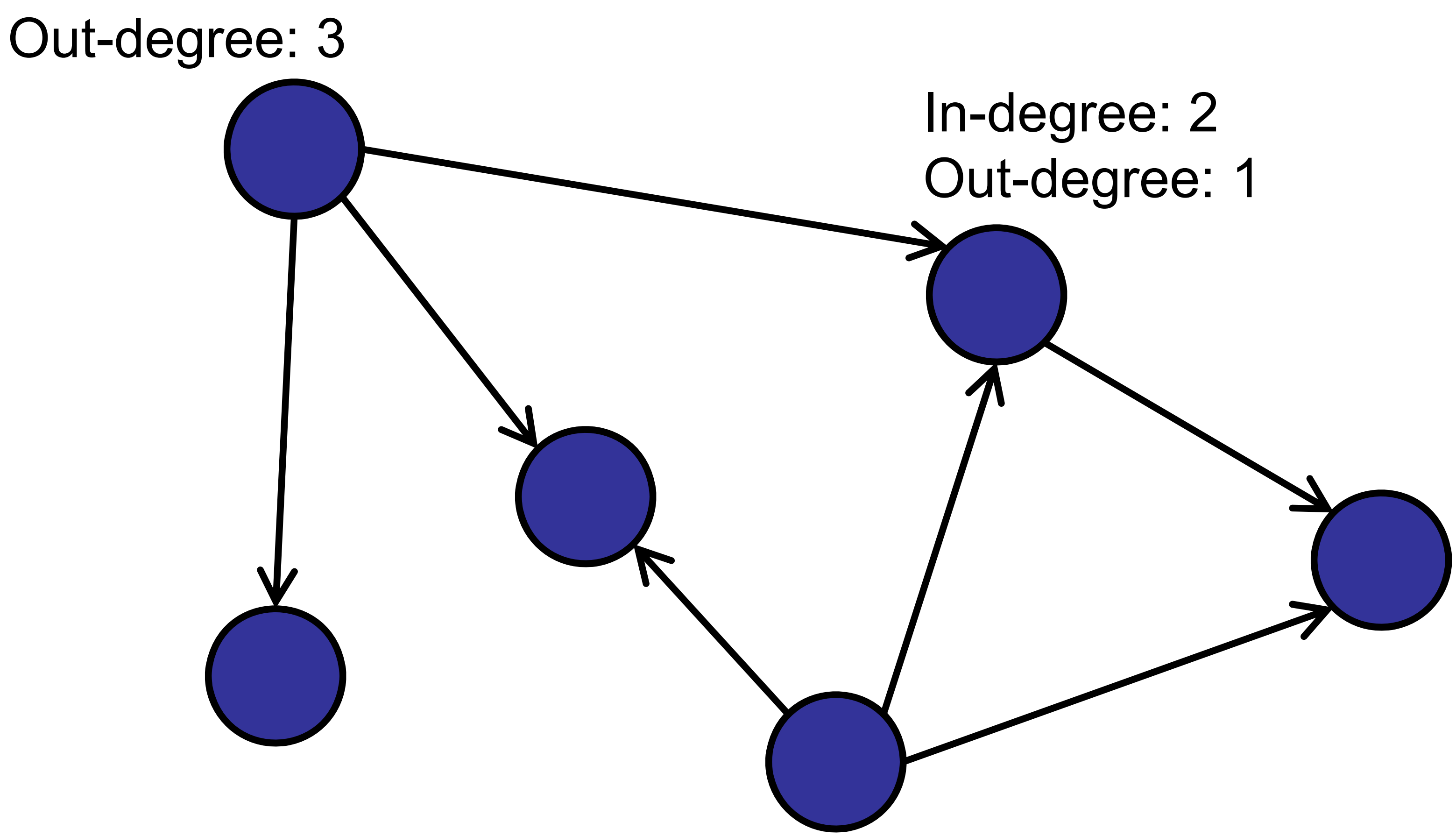
Graph G = <V, E>

- 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: $O(V + E)$

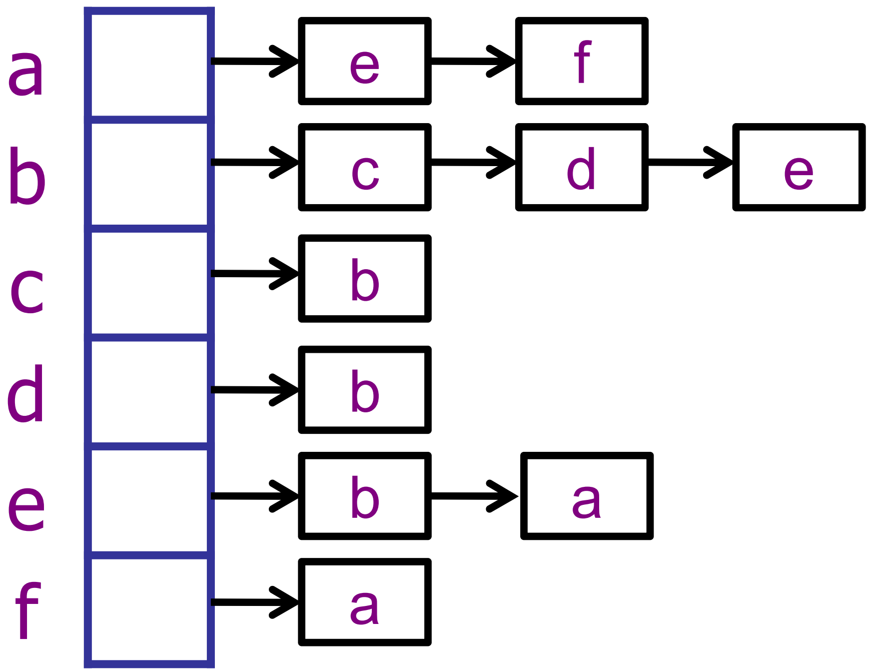
Adjacency Matrix:

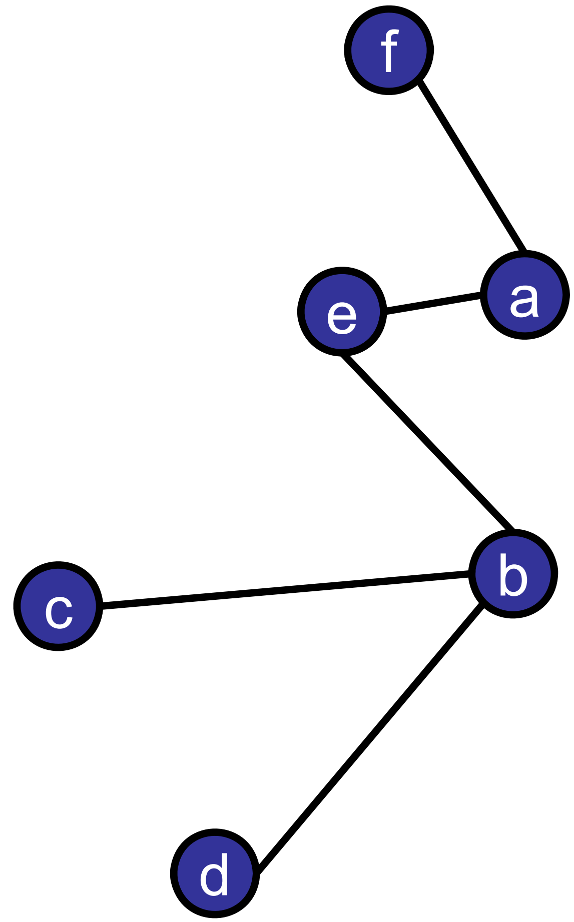
- Matrix $A[v,w]$ represents edge (v,w)
- Space: $O(V^2)$

Adjacency List

Graph consists of:

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

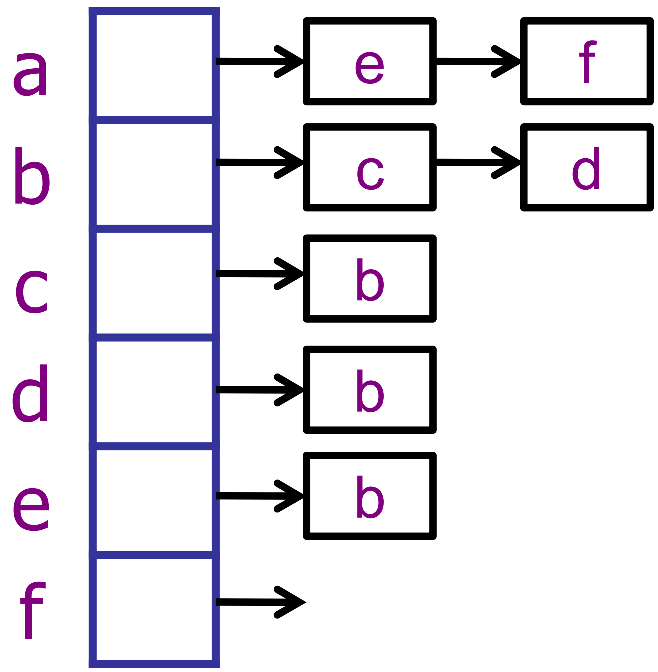


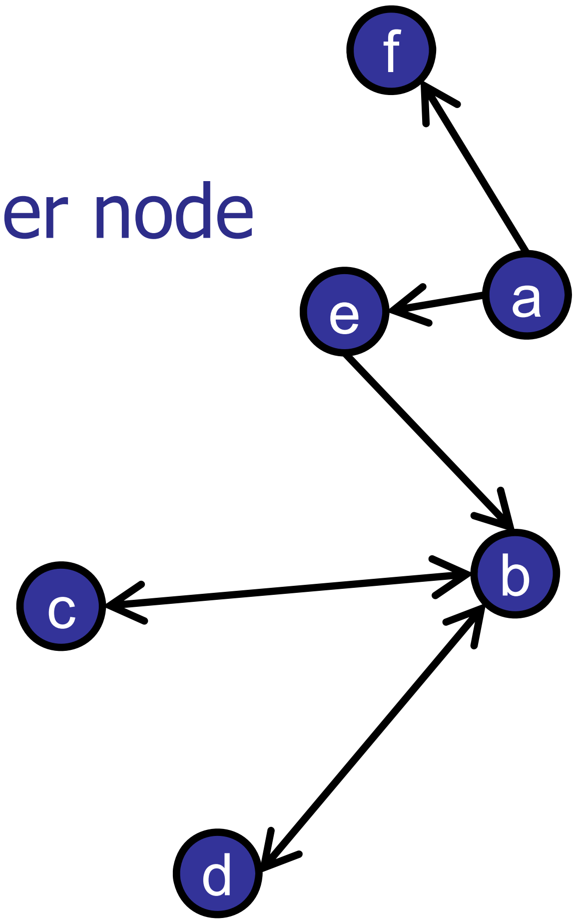


Adjacency List

Directed Graph consists of:

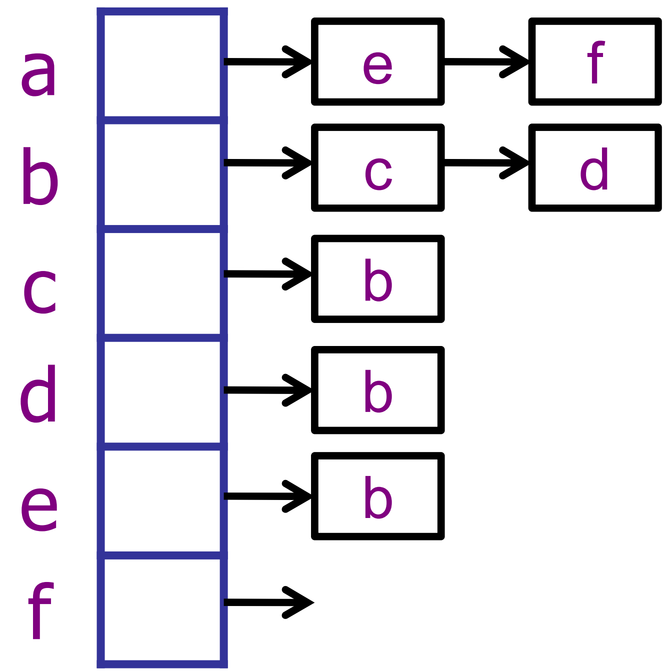
- Nodes: stored in an array
- **Outgoing** Edges: linked list per node





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:

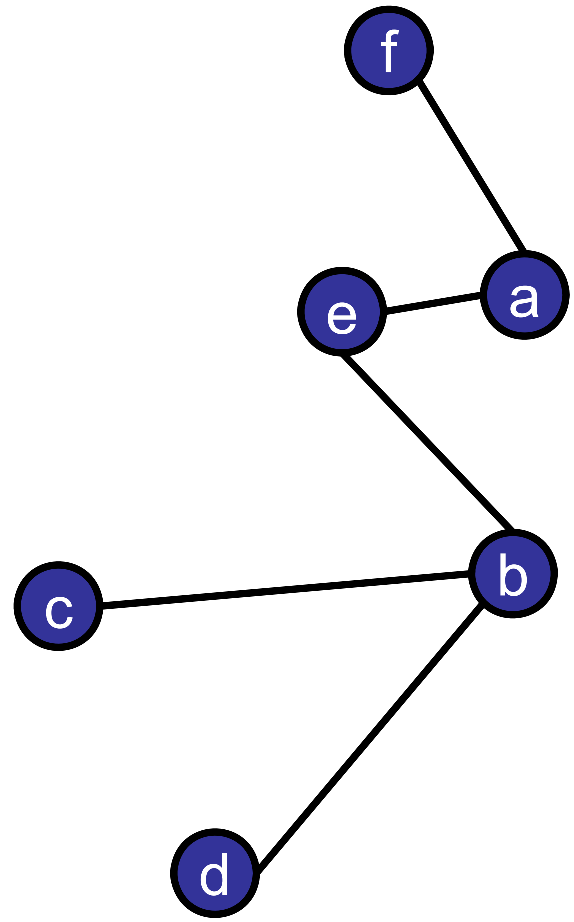
- Matrix $A[v,w]$ represents edge (v,w)
- Space: $O(V^2)$

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

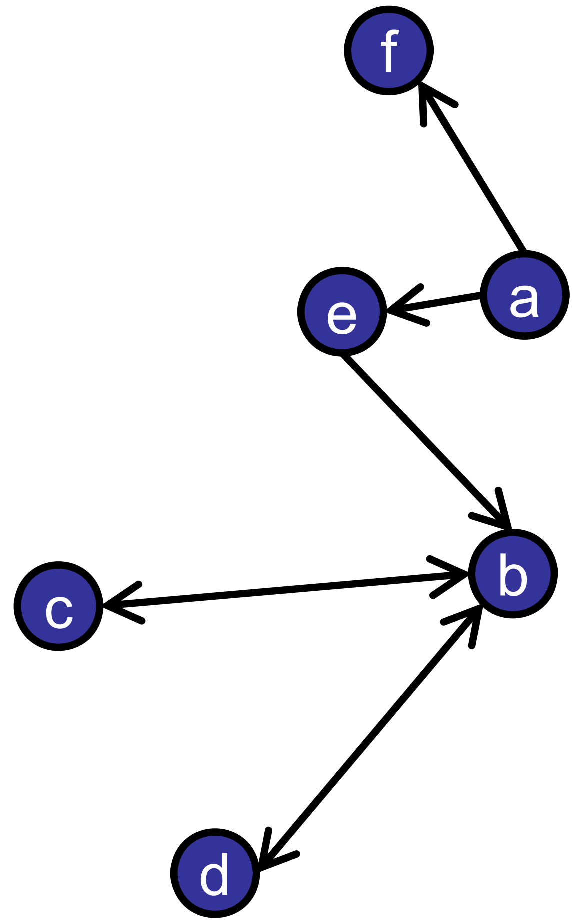


Adjacency Matrix

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:

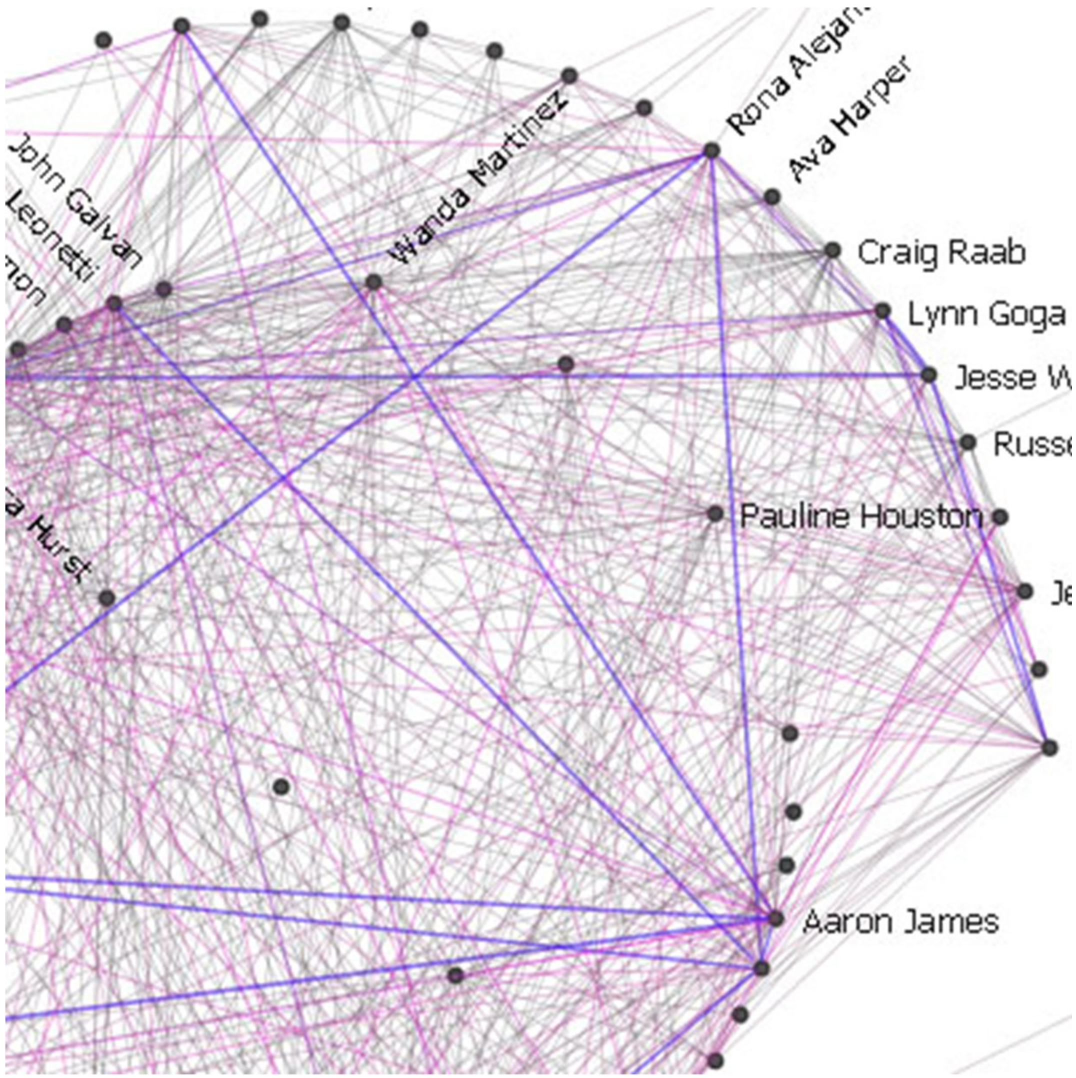
- Search recursively
- Follow outgoing edges
- Backtrack (through incoming edges)

Example of directed graphs

Directed Graphs

Is friendship always bidirectional?:

- Nodes are people
- Edge = friendship



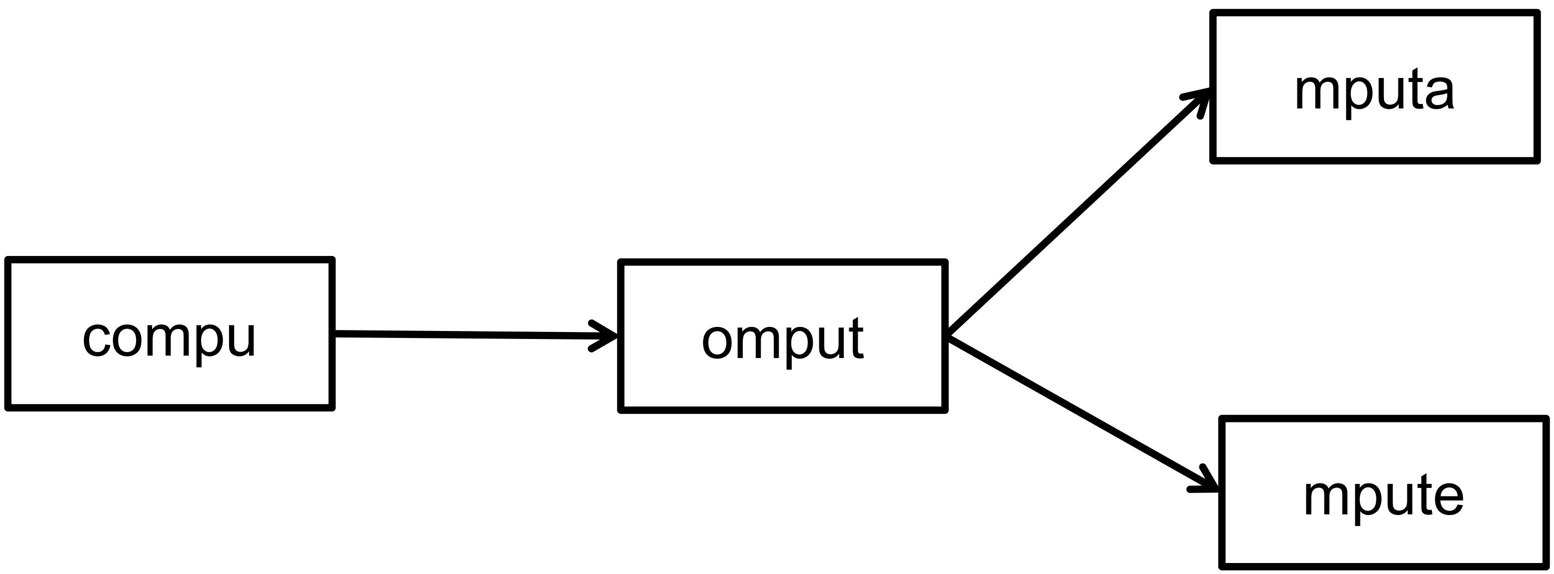
Facebook: yes

Twitter: no

Directed Graphs

Markov text generation:

- Nodes are kgrams
- Edge = one kgram follows another



Scheduling

Set of tasks for baking cookies:

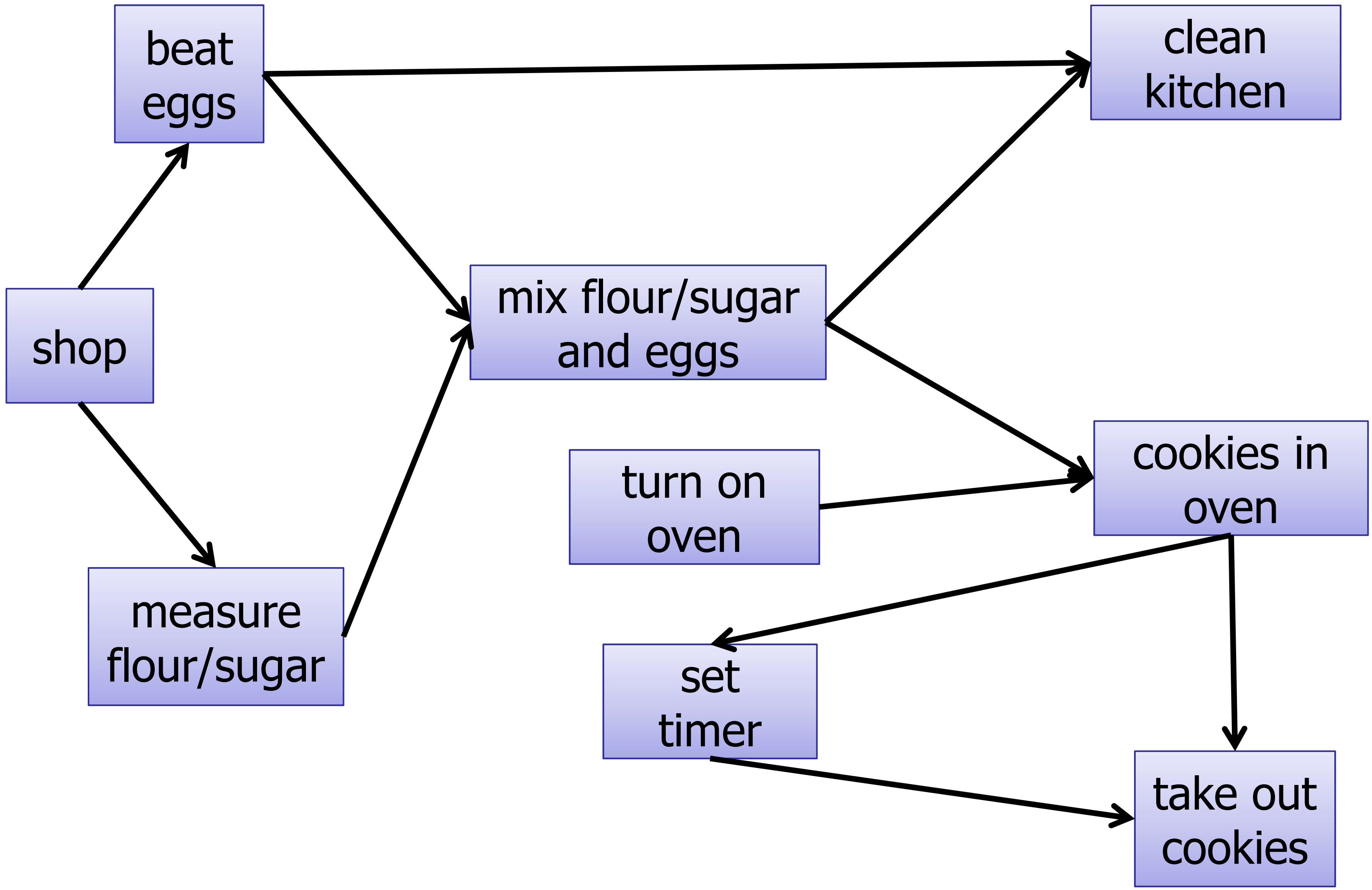
- Shop for groceries
- Put the cookies in the oven
- Clean the kitchen
- Beat the eggs in a bowl
- Measure the flour and sugar in a bowl
- Mix the eggs with the flour and sugar
- Turn on the oven
- Set the timer
- Take out the cookies

Scheduling

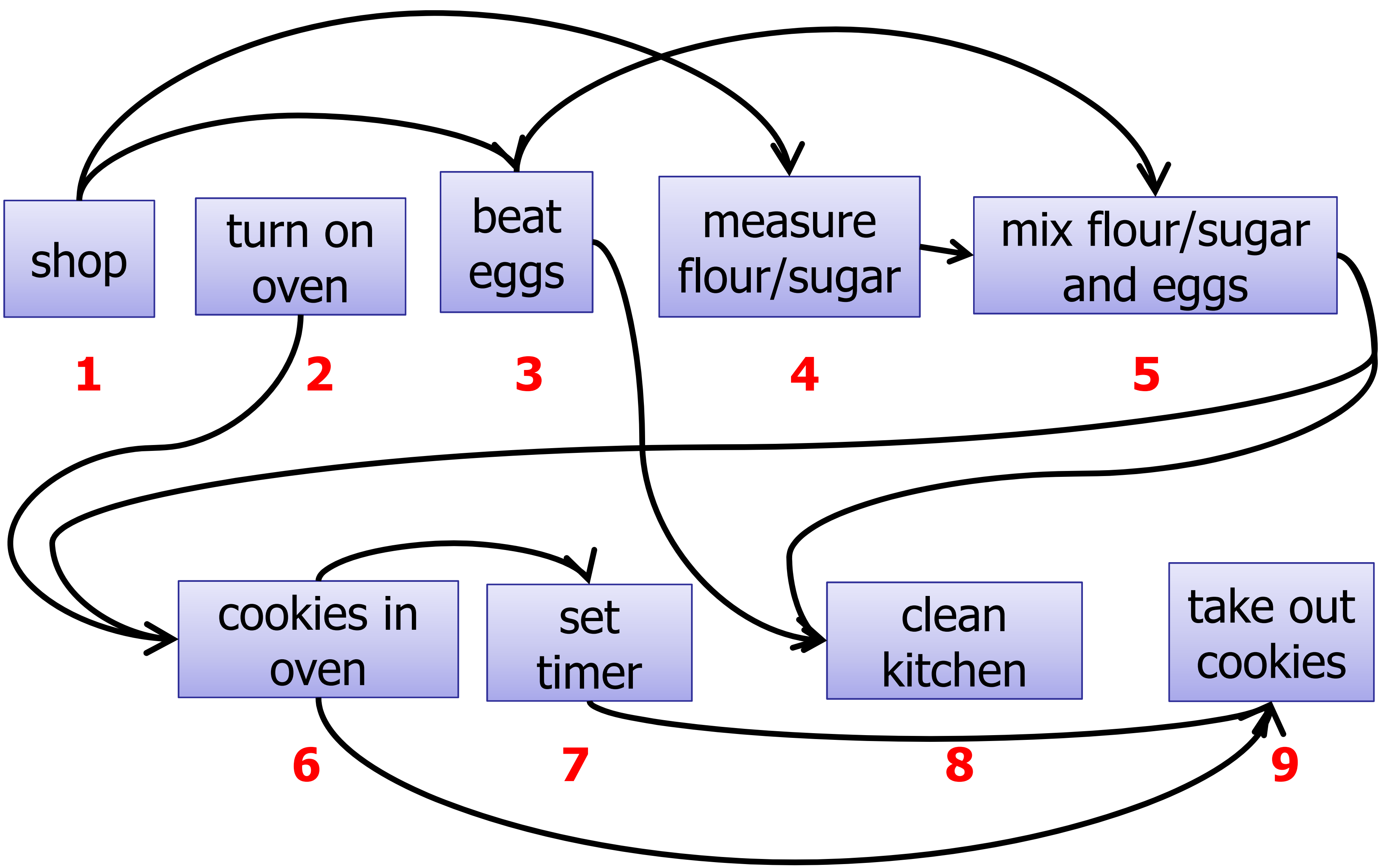
Ordering:

- 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 Ordering



Topological Order

Properties:

1. Sequential total ordering of all nodes

1. shop

2. turn on oven

3. measure flour/sugar

4. eggs

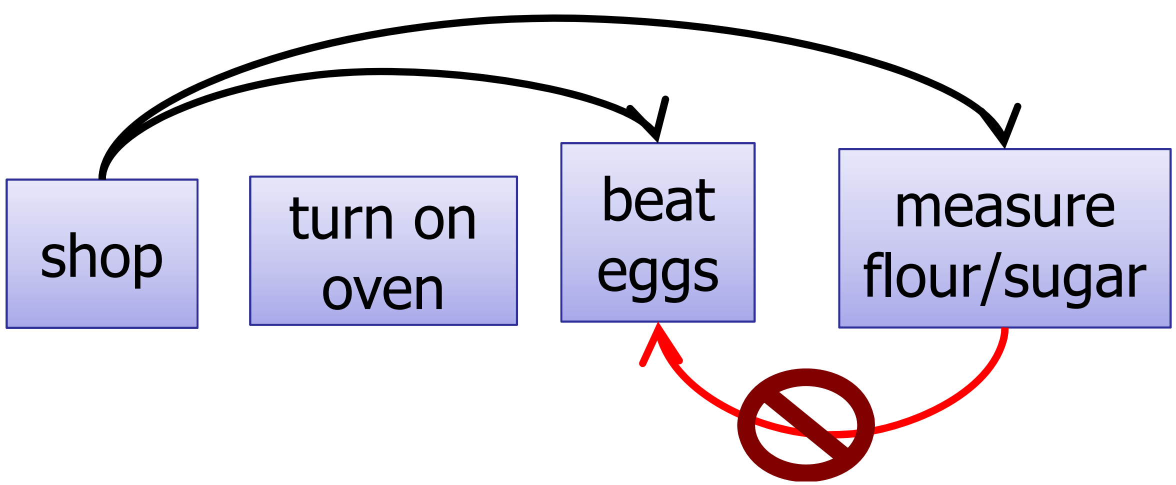
Topological Order

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.

Today

Searching a graph

- BFS (finding shortest paths)
- DFS

Directed Graphs

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