

CS2040S

Data Structures and Algorithms

Graphs! (Part 2)

Puzzle of the Week:

Can you interchange blocks 14 and 15, leaving everything else the same?



Loyd writes of how he “drove the entire world crazy,” and that “A prize of \$1,000, offered for the first correct solution to the problem, has never been claimed, although there are thousands of persons who say they performed the required feat.” He continues,

People became infatuated with the puzzle and ludicrous tales are told of shopkeepers who neglected to open their stores; of a distinguished clergyman who stood under a street lamp all through a wintry night trying to recall the way he had performed the feat.... Pilots are said to have wrecked their ships, and engineers rush their trains past stations. A famous Baltimore editor tells how he went for his noon lunch and was discovered by his frantic staff long past midnight pushing little pieces of pie around on a plate! [9]

Roadmap

Next: Searching Graphs

- Searching graphs
- Shortest path problem
- Bellman-Ford Algorithm
- Dijkstra's Algorithm

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

BFS & DFS

Simple, but very, very important:

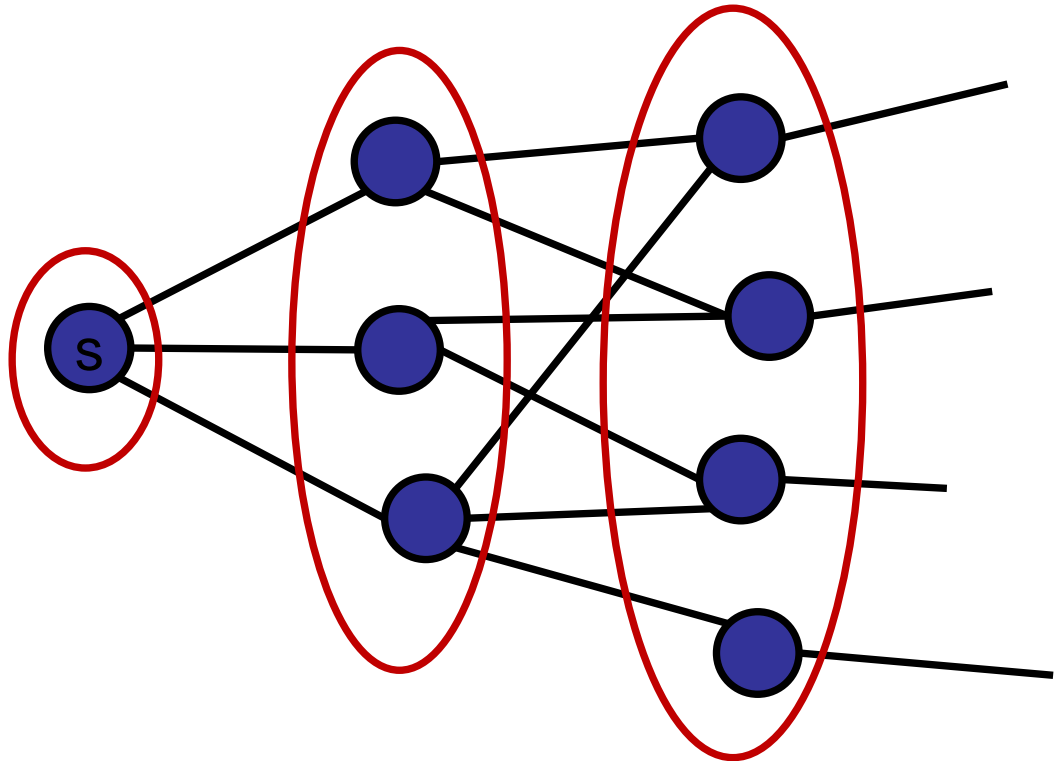
Recent conversation with CS2109S instructor
(Introduction to AI and Machine Learning):

“It is really, really important that every student who take CS2109S can easily code up a simple BFS or DFS. Critical foundation before we can do anything more interesting.”

Searching a graph

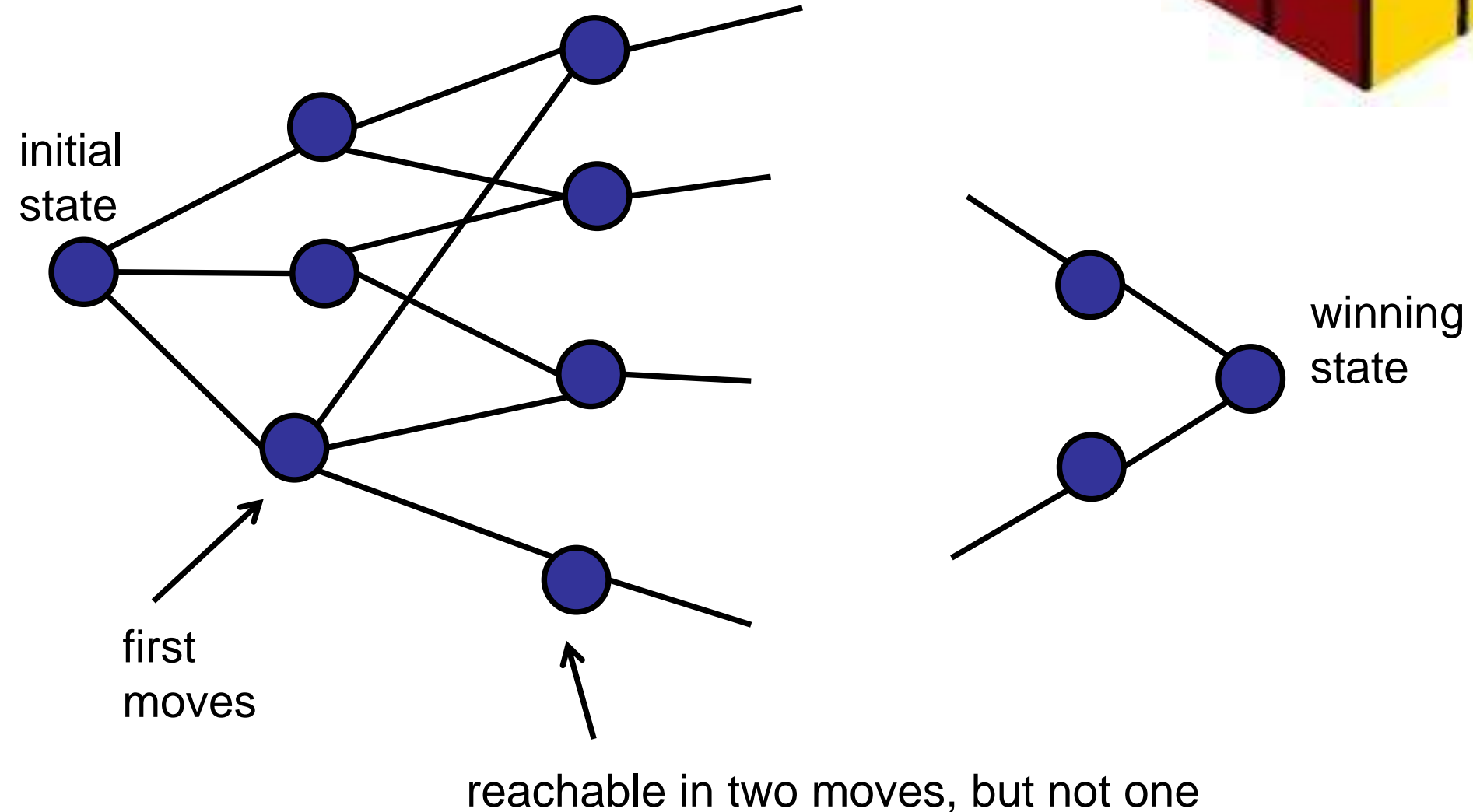
Breadth-First Search:

Explore level by level



2 x 2 x 2 Rubik's Cube

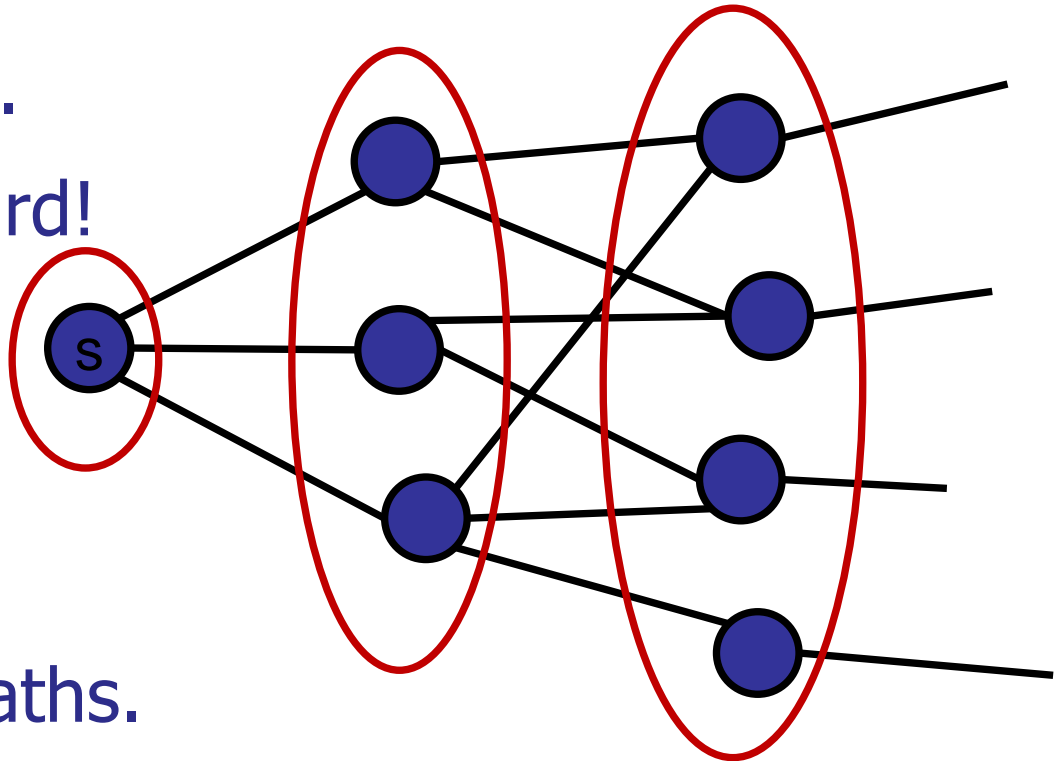
Geography of Rubik's configurations:



Searching a graph

Breadth-First Search:

- Explore level by level
- Frontier: current level
- Initially: $\{s\}$
- Advance frontier.
- Don't go backward!

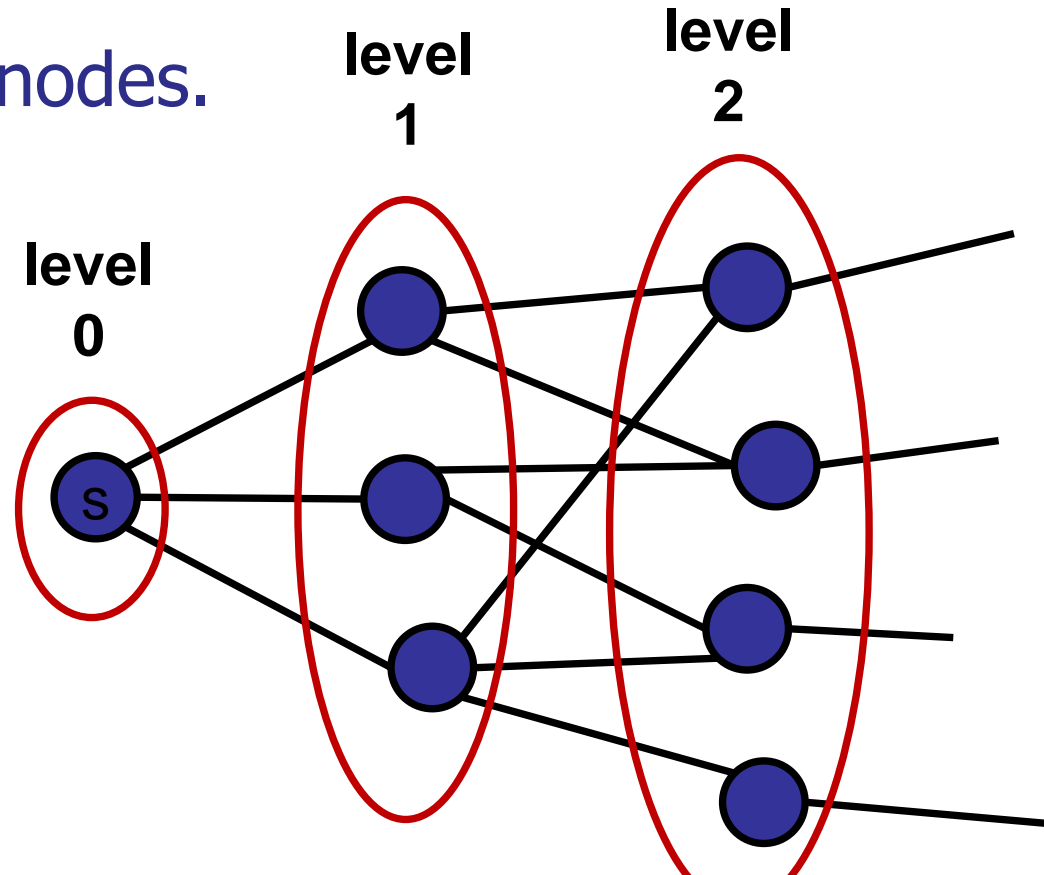


- Finds shortest paths.

Searching a graph

Breadth-First Search:

- Build levels.
- Calculate $\text{level}[i]$ from $\text{level}[i-1]$
- Skip already visited nodes.



Breadth-First Search

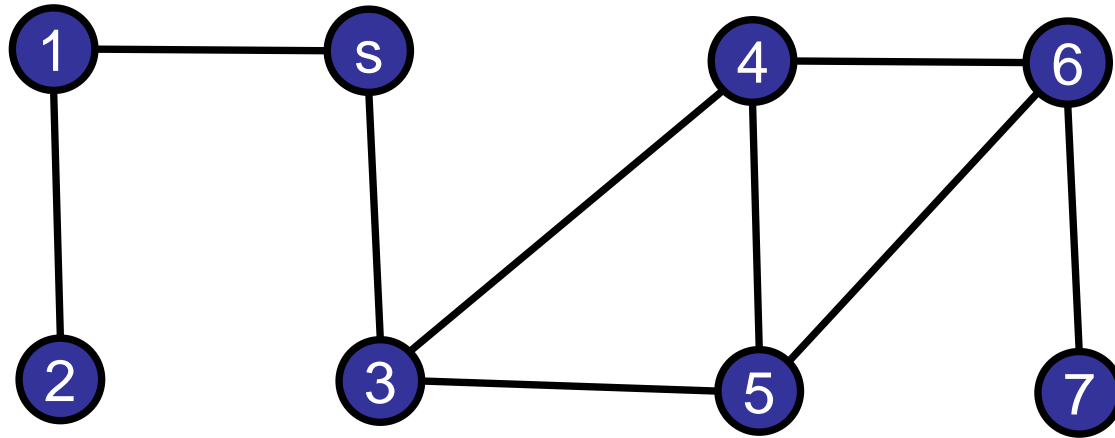
```
BFS(Node[] nodeList, int startId) {  
    boolean[] visited = new boolean[nodeList.length];  
    Arrays.fill(visited, false);  
  
    int[] parent = new int[nodelist.length];  
    Arrays.fill(parent, -1);  
  
    Collection<Integer> frontier = new Collection<Integer>;  
    frontier.add(startId);  
  
    // Main code goes here!  
  
}
```

Breadth-First Search

```
visited[startId] = true;

while (!frontier.isEmpty()) {
    Collection<Integer> nextFrontier = new Collection<Integer>;
    for (Integer v : frontier) {
        for (Integer w : nodeList[v].nbrList) {
            if (!visited[w]) {
                visited[w] = true;
                parent[w] = v;
                nextFrontier.add(w);
            }
        }
    }
    frontier = nextFrontier;
}
```

Breadth-First Search Example



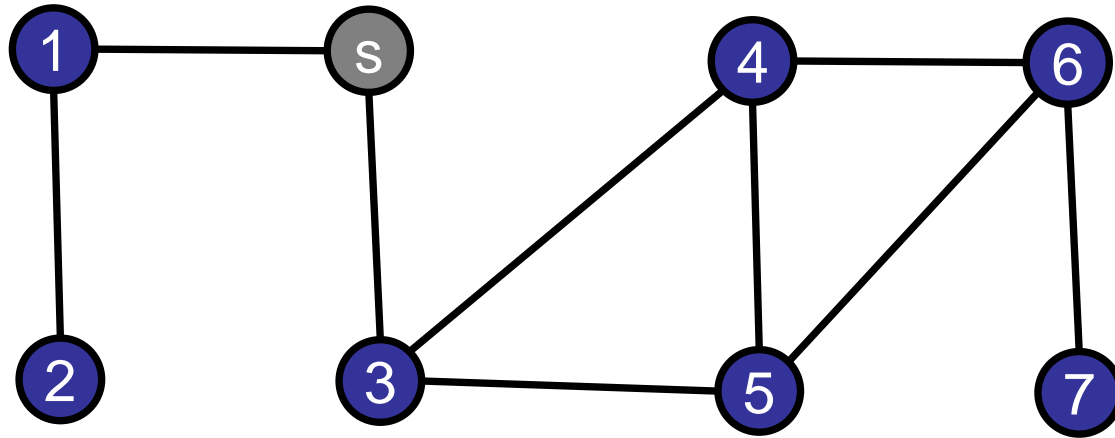
Red = active

Gray = visited

Blue = unvisited

Frontier: s

Breadth-First Search Example



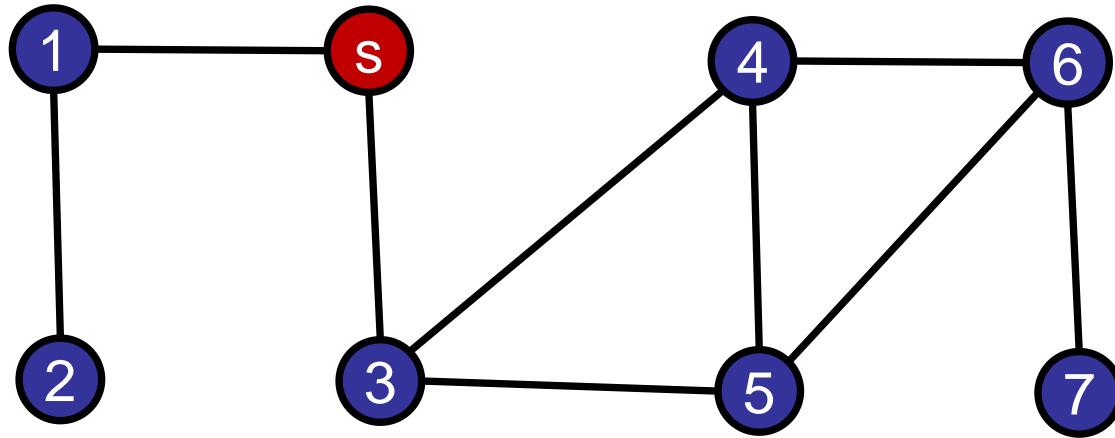
Red = active

Gray = visited

Blue = unvisited

Frontier: s

Breadth-First Search Example



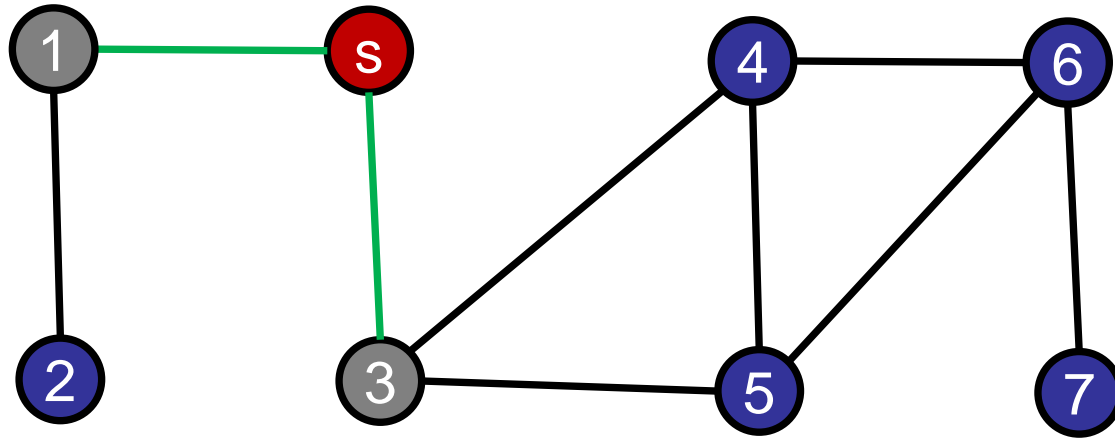
Red = active

Gray = visited

Blue = unvisited

Frontier: s

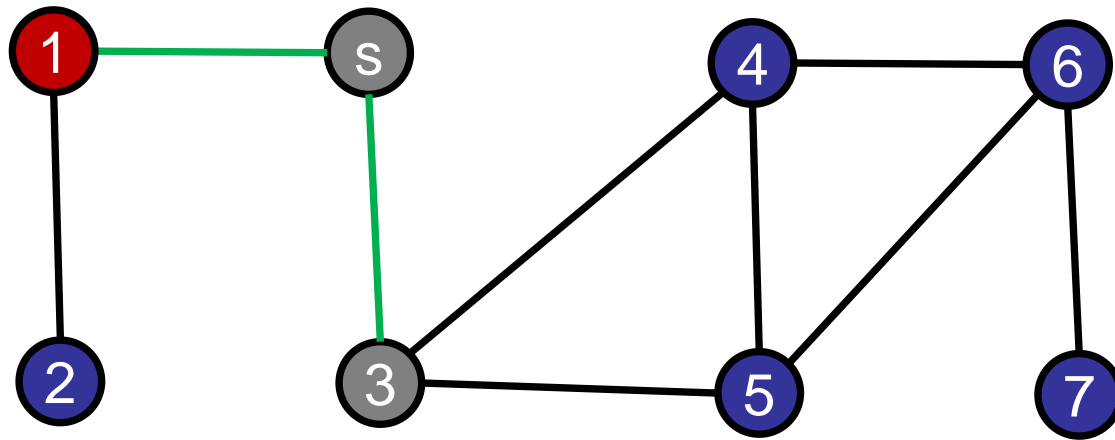
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: s
Next Frontier: 1, 3

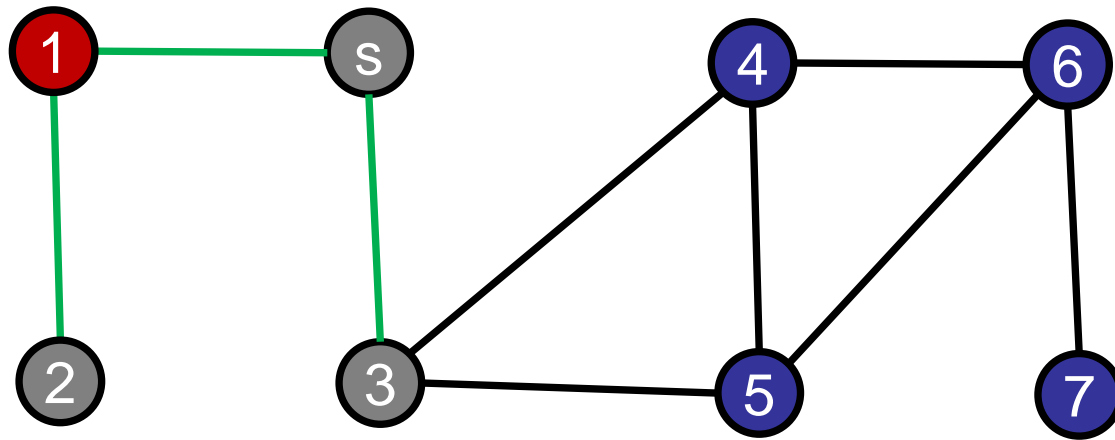
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 1, 3
Next Frontier:

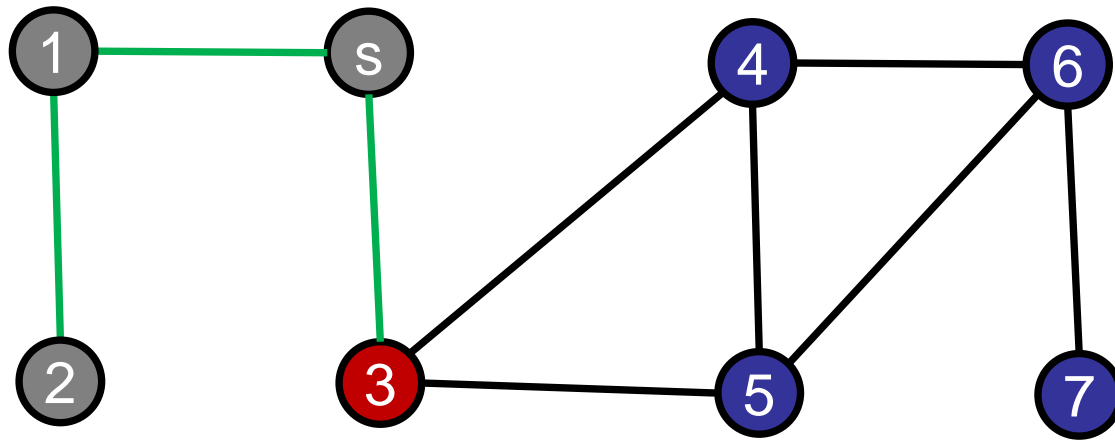
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 1, 3
Next Frontier: 2

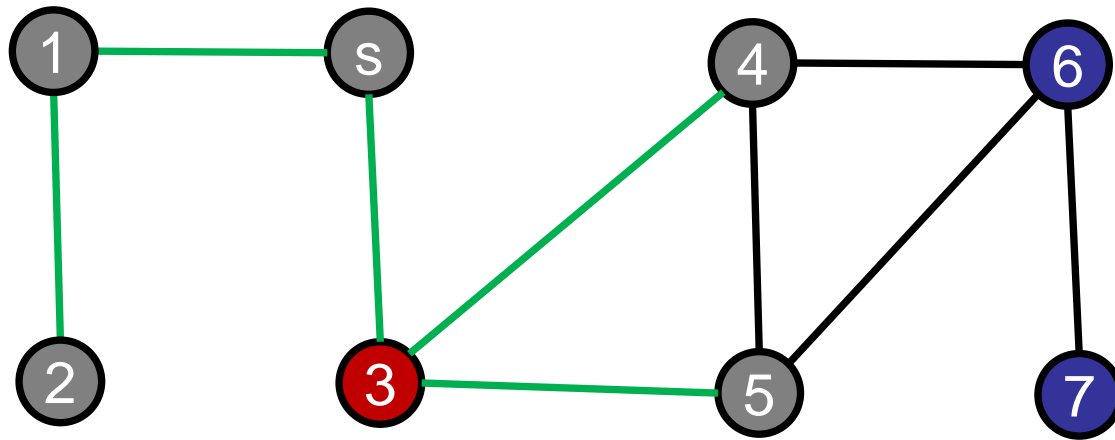
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 1, 3
Next Frontier: 2

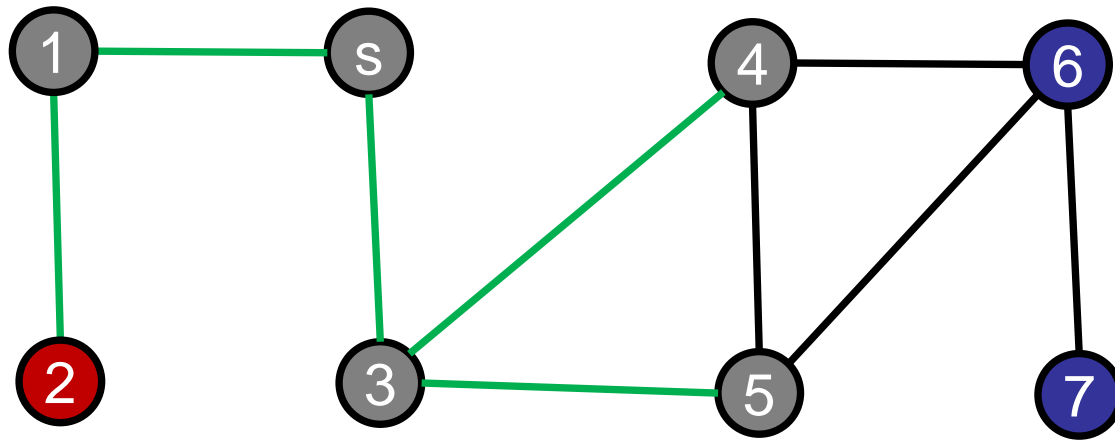
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 1, 3
Next Frontier: 2, 4, 5

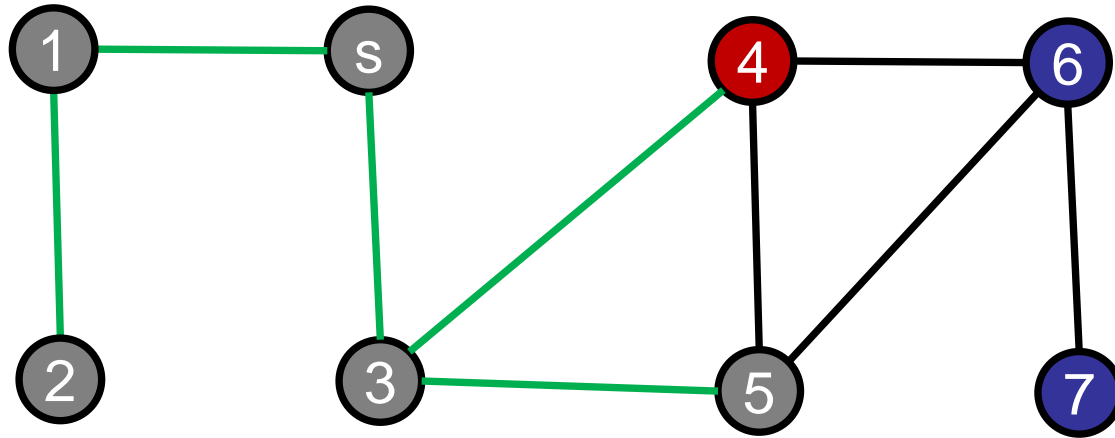
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 2, 4, 5
Next Frontier:

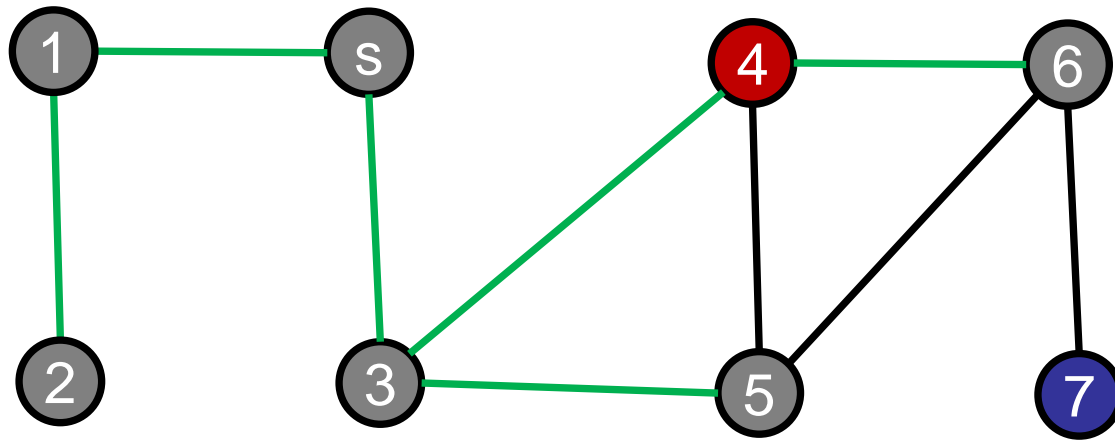
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 2, 4, 5
Next Frontier:

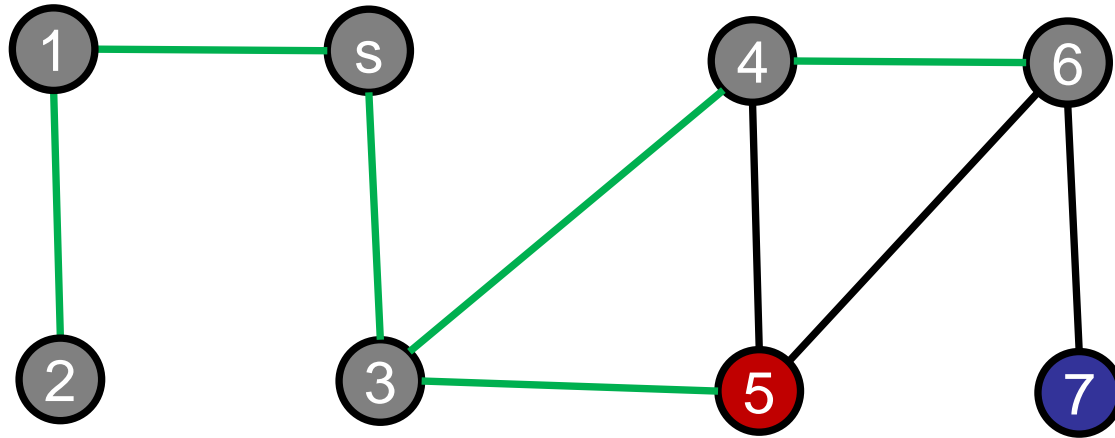
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 2, 4, 5
Next Frontier: 6

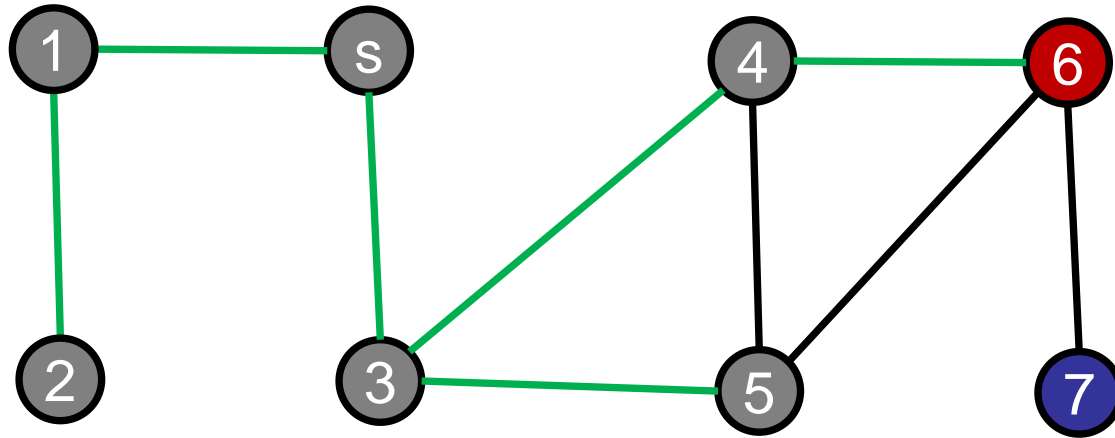
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 2, 4, 5
Next Frontier: 6

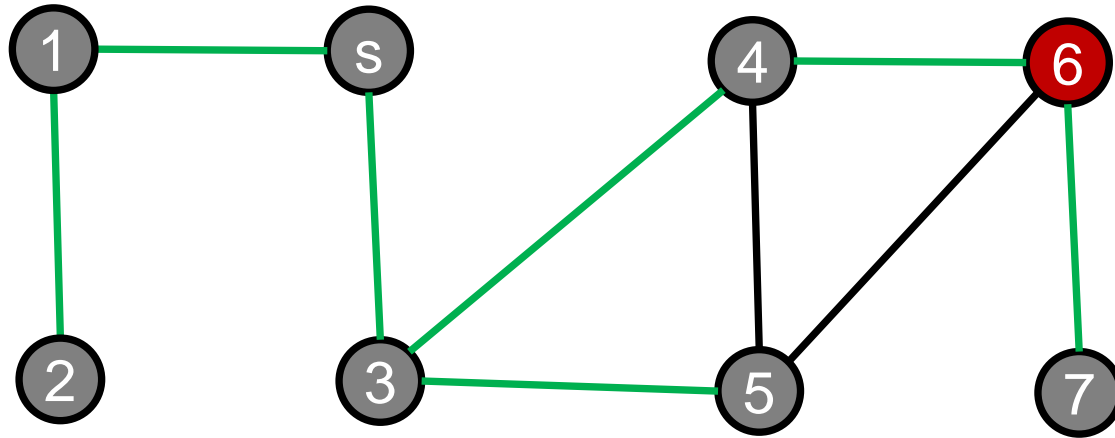
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 6
Next Frontier:

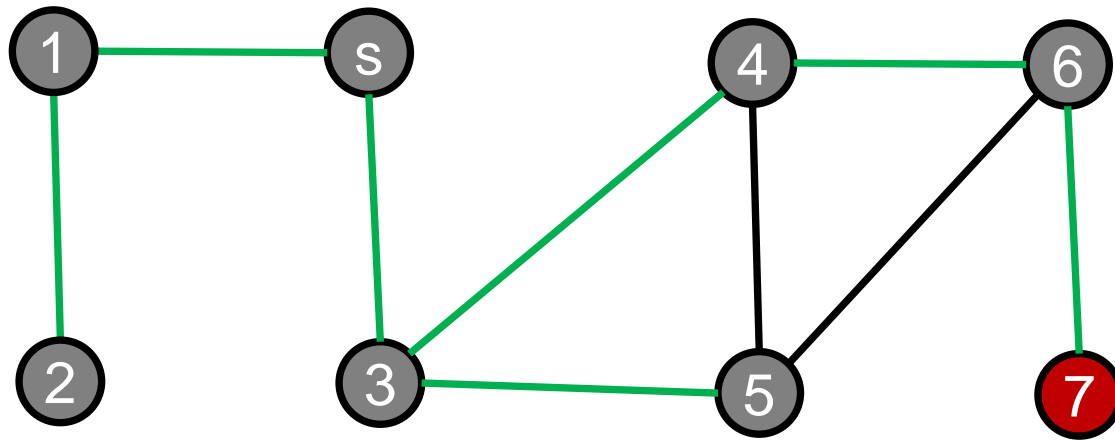
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 6
Next Frontier: 7

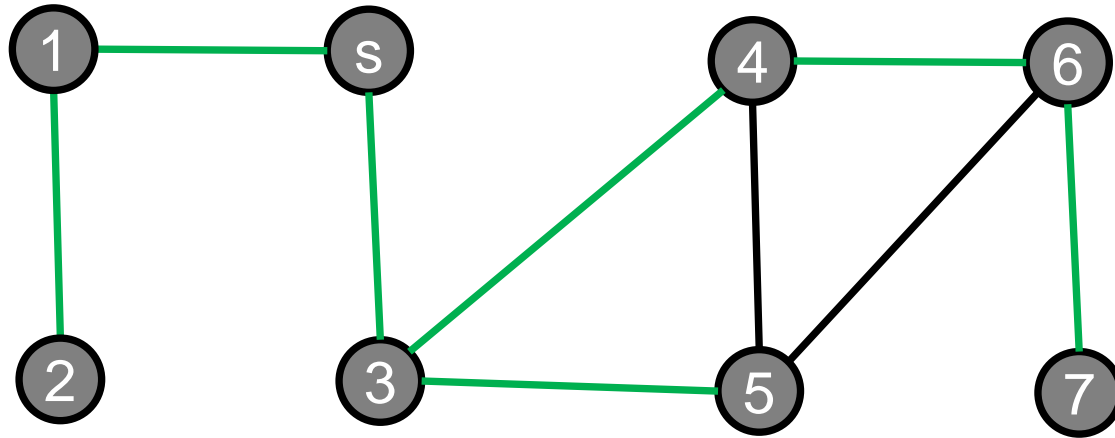
Breadth-First Search Example



Red = active
Gray = visited
Blue = unvisited

Frontier: 7
Next Frontier:

Breadth-First Search Example

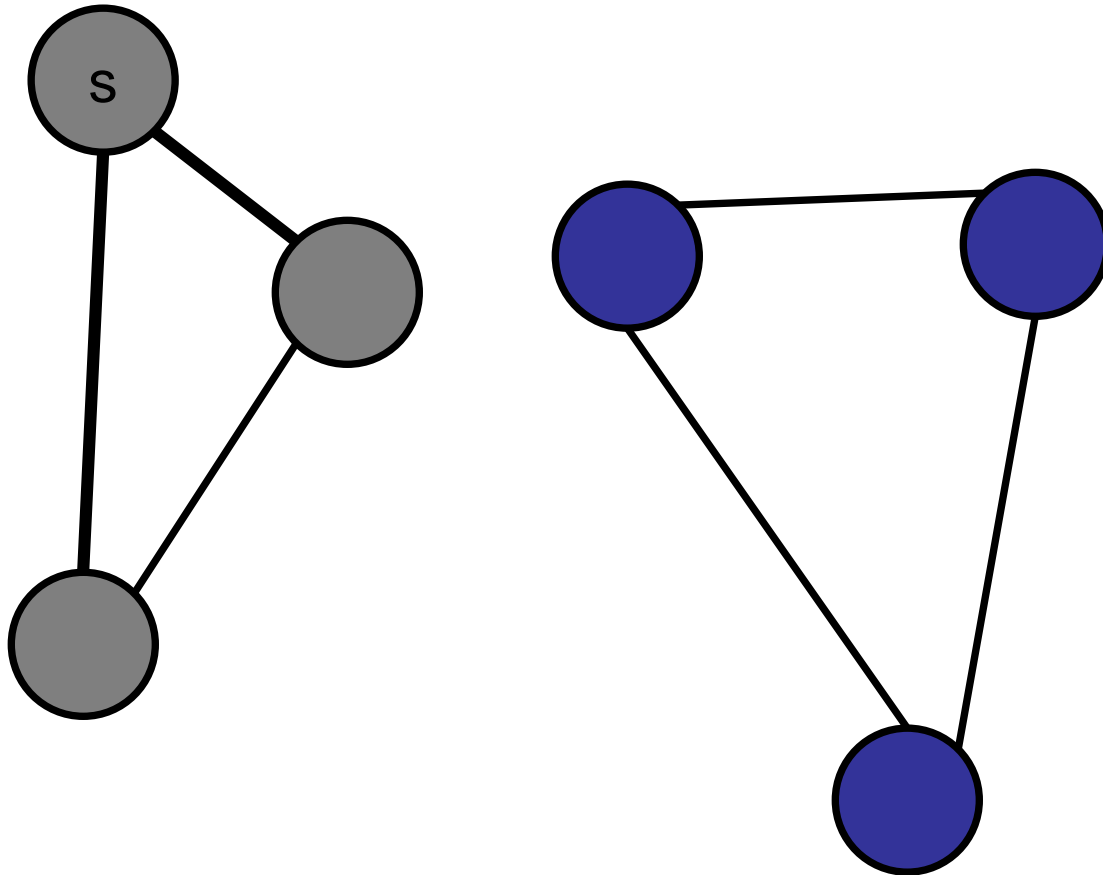


Red = active
Gray = visited
Blue = unvisited

Frontier:
Next Frontier:

BFS on Disconnected Graph

Example:



Breadth-First Search

```
BFS(Node[] nodeList) {  
    boolean[] visited = new boolean[nodeList.length];  
    Arrays.fill(visited, false);  
  
    int[] parent = new int[nodeList.length];  
    Arrays.fill(parent, -1);  
  
    for (int start = 0; start < nodeList.length; start++) {  
        if (!visited[start]){  
            Collection<Integer> frontier = new Collection<Integer>;  
            frontier.add(startId);  
  
            // Main code goes here!  
        }  
    }  
}
```

The running time of BFS (using adjacency list) is:

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

ARCHIPELAGO

is open

The running time of BFS (using adjacency list) is:

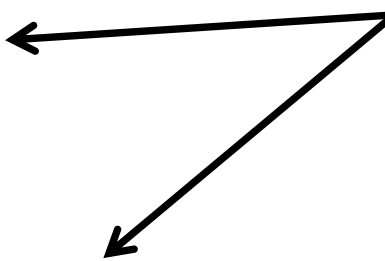
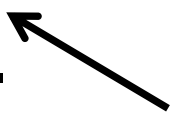
1. $O(V)$
2. $O(E)$
- ✓ 3. $O(V+E)$
4. $O(VE)$
5. (V^2)
6. I have no idea.

Depends on adjacency list vs. adjacency matrix.

Here: assume adjacency list.

Breadth-First Search

Analysis:

- Vertex v = “start” once. 
- Vertex v added to nextFrontier (and frontier) once.
 - After visited, never re-added.
- Each v .nbrlist is enumerated once.
 - When v becomes active in frontier. 

$O(V)$

$O(E)$

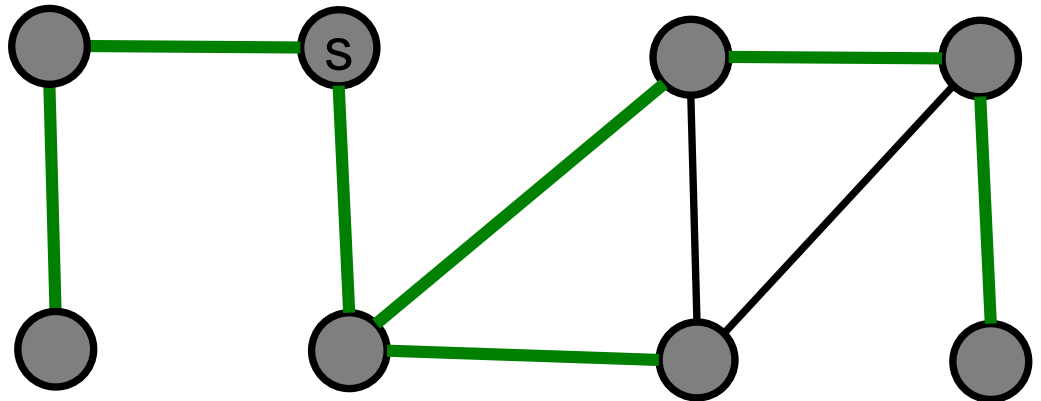
Breadth-First Search

```
while (!frontier.isEmpty()) {
    Collection<Integer> nextFrontier = new Collection<Integer>;
    for (Integer v : frontier) {
        for (Integer w : nodeList[v].nbrList) {
            if (!visited[w]) {
                visited[w] = true;
                parent[w] = v;
                nextFrontier.add(w);
            }
        }
    }
    frontier = nextFrontier;
}
```

Breadth-First Search

Shortest paths:

- Parent pointers store shortest path.



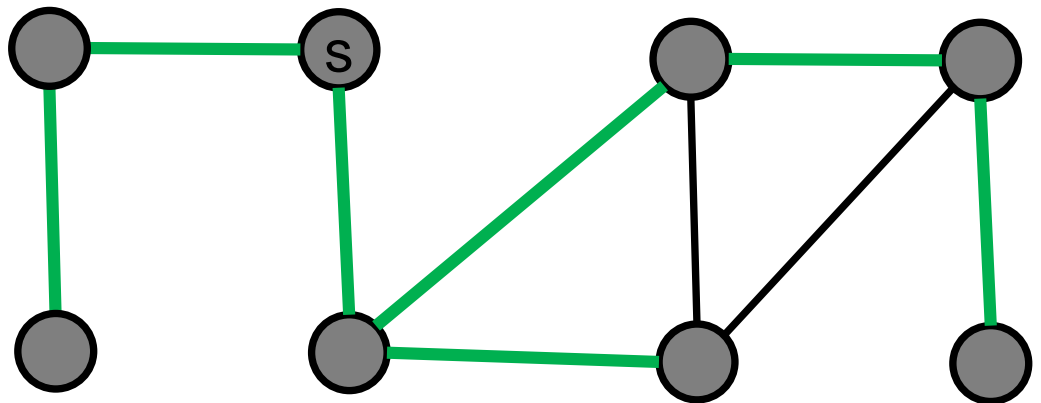
Which is true? (More than one may apply.)

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

Breadth-First Search

Shortest paths:

- Parent pointers store shortest path.
- Shortest path is a tree.
- (Possibly high degree; possibly high diameter.)



What if there are
two components?

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)

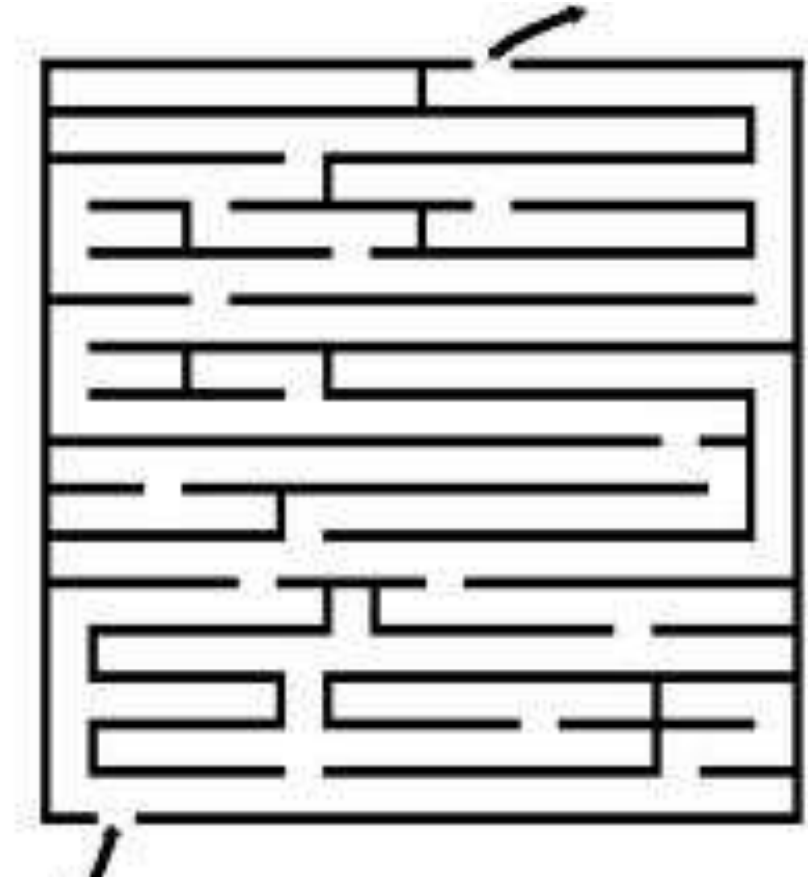
Graph representation:

- Adjacency list

Depth-First Search

Exploring a maze:

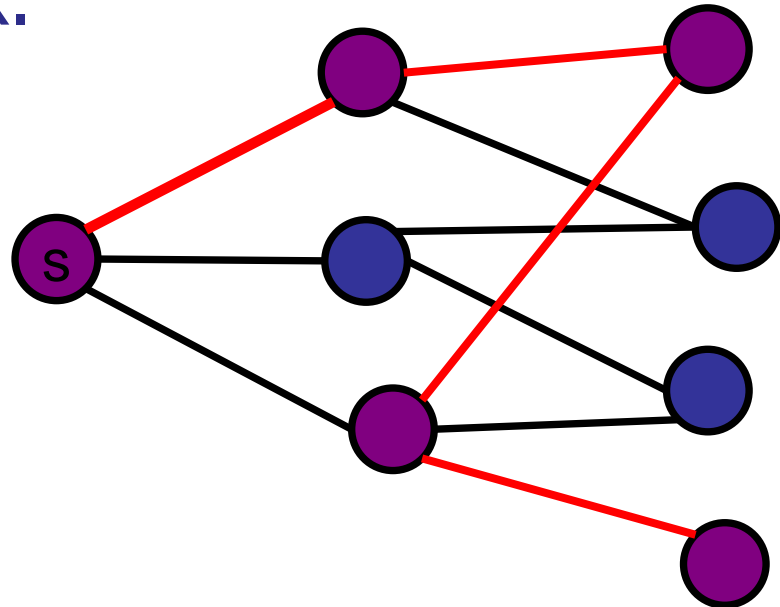
- Follow path until stuck.
- Backtrack along breadcrumbs until reach unexplored neighbor.
- Recursively explore.



Searching a graph

Depth-First Search:

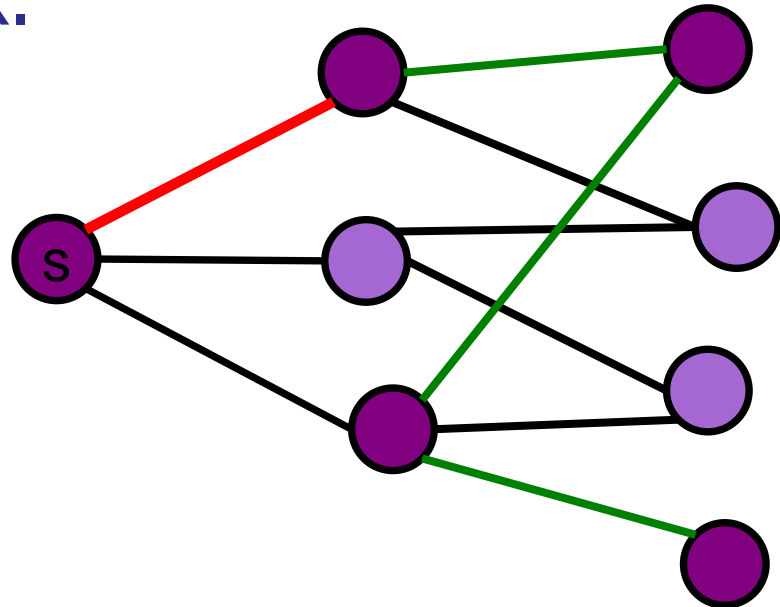
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don't repeat a vertex.



Searching a graph

Depth-First Search:

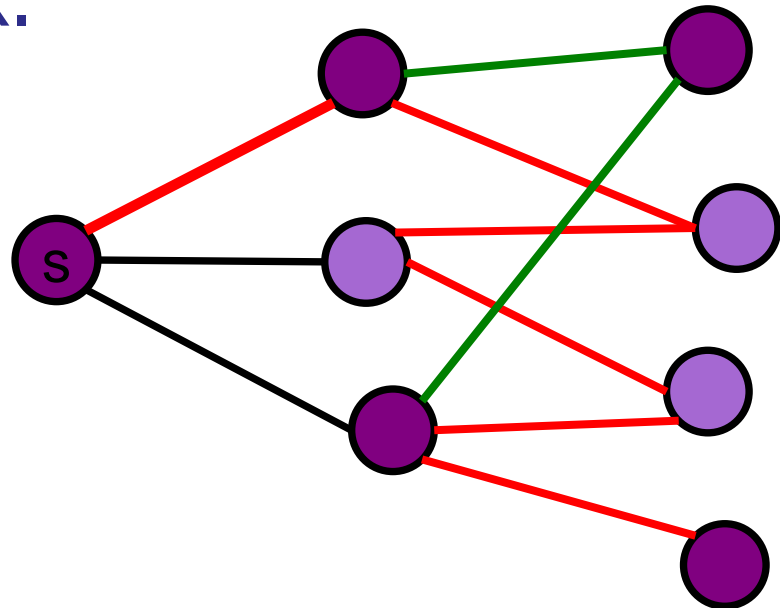
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don't repeat a vertex.



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.



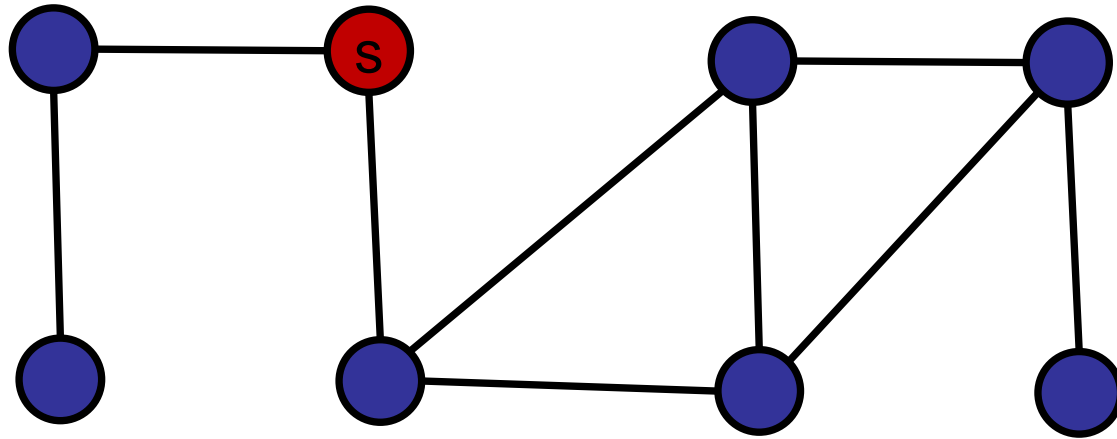
Depth-First Search

```
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 = 0; start < nodeList.length; start++) {  
        if (!visited[start]) {  
            visited[start] = true;  
            DFS-visit(nodeList, visited, start);  
        }  
    }  
}
```

Depth-First Search Example



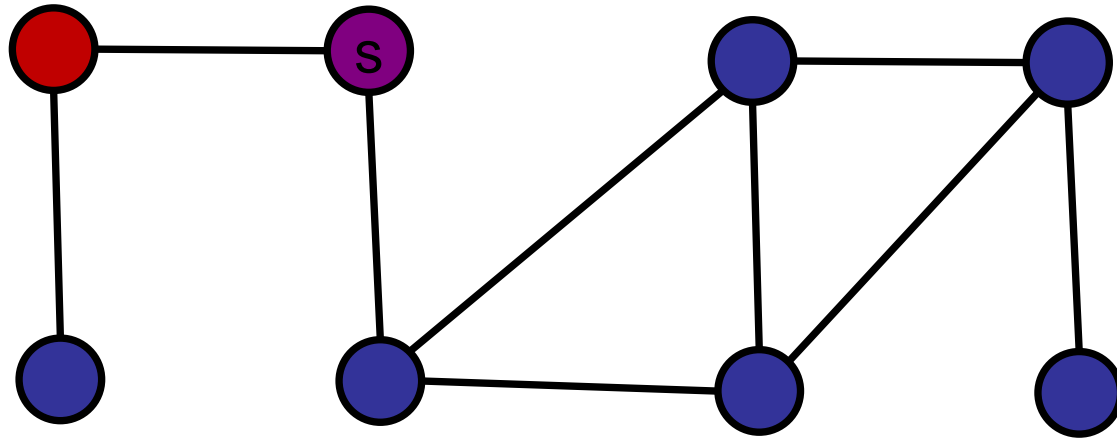
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



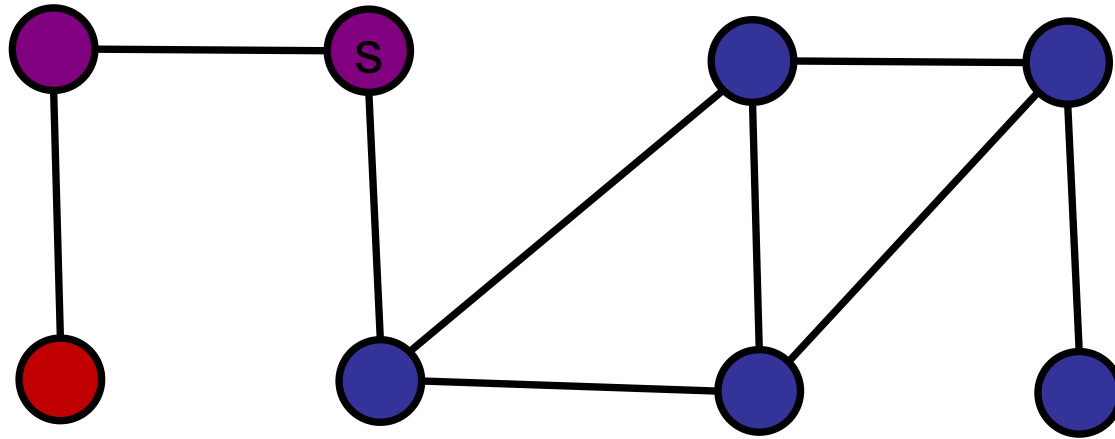
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



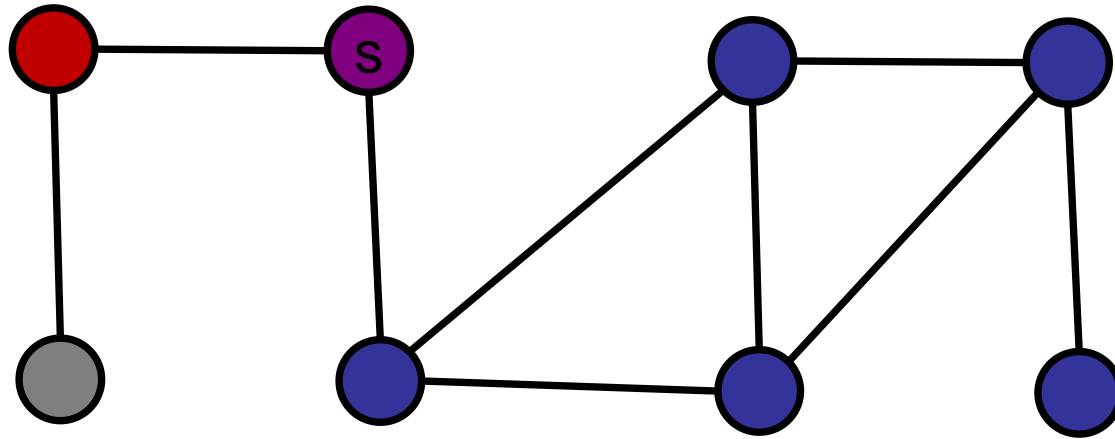
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



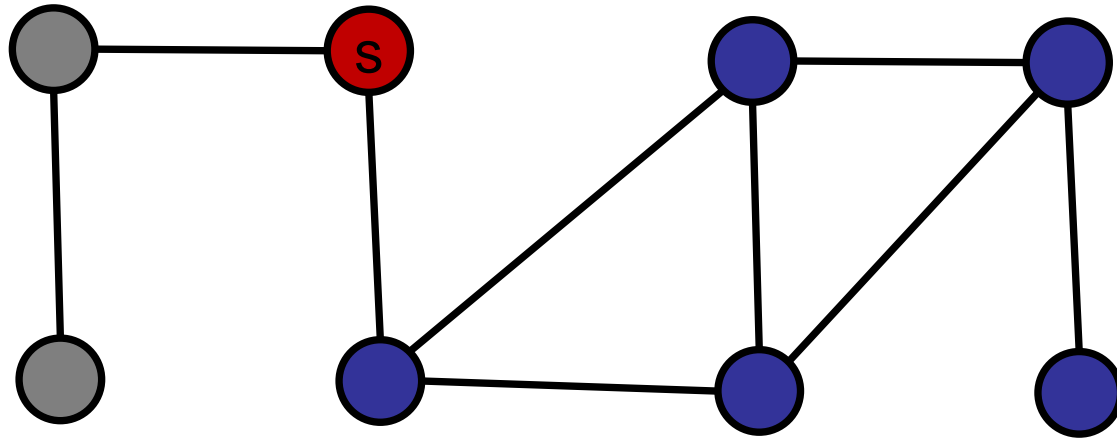
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



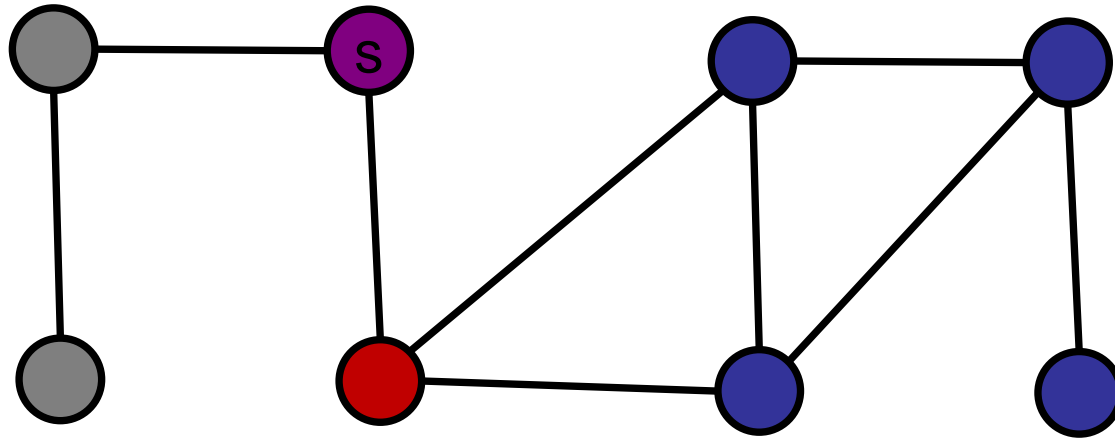
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



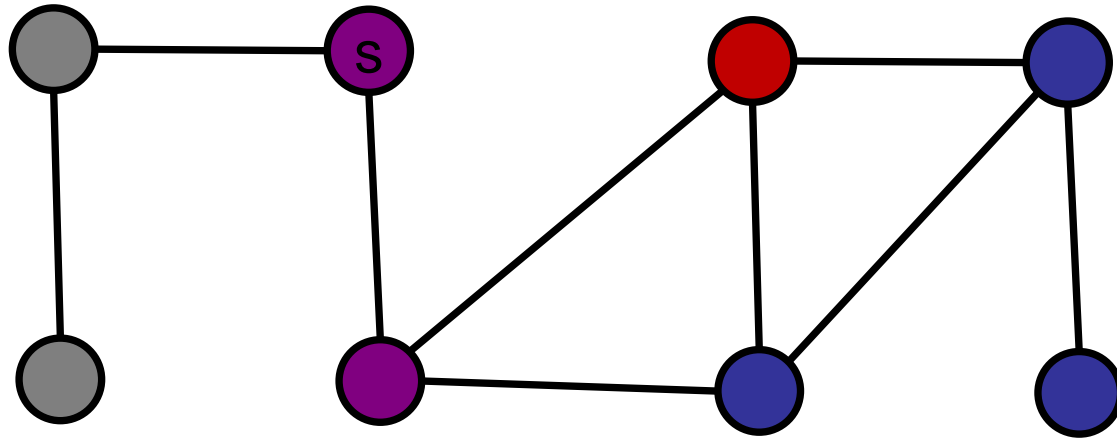
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



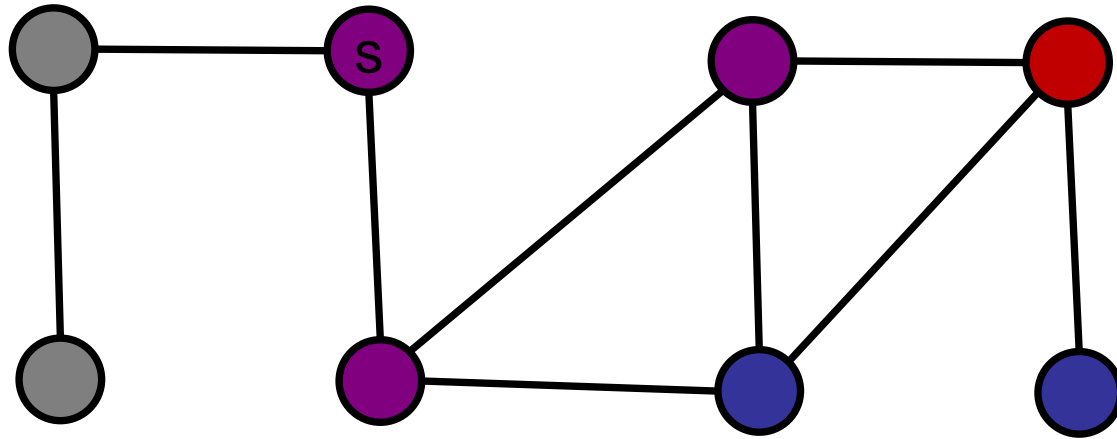
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



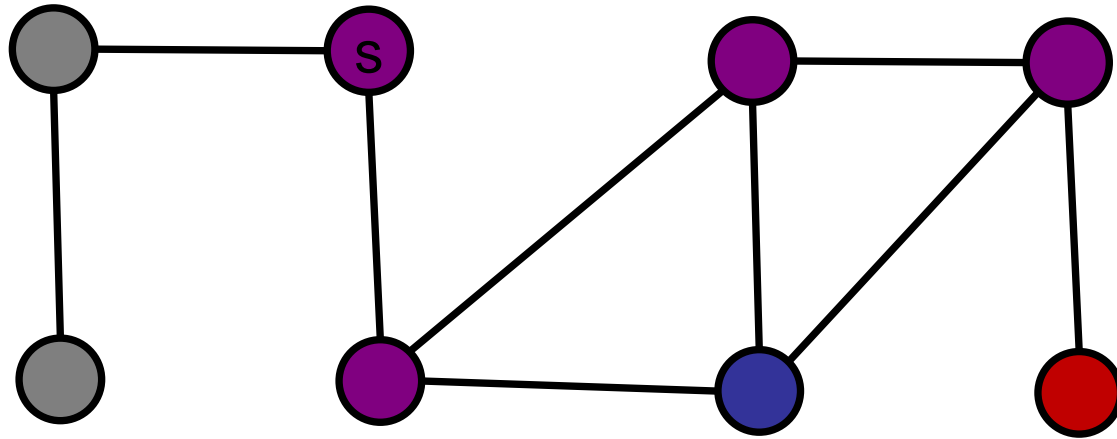
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



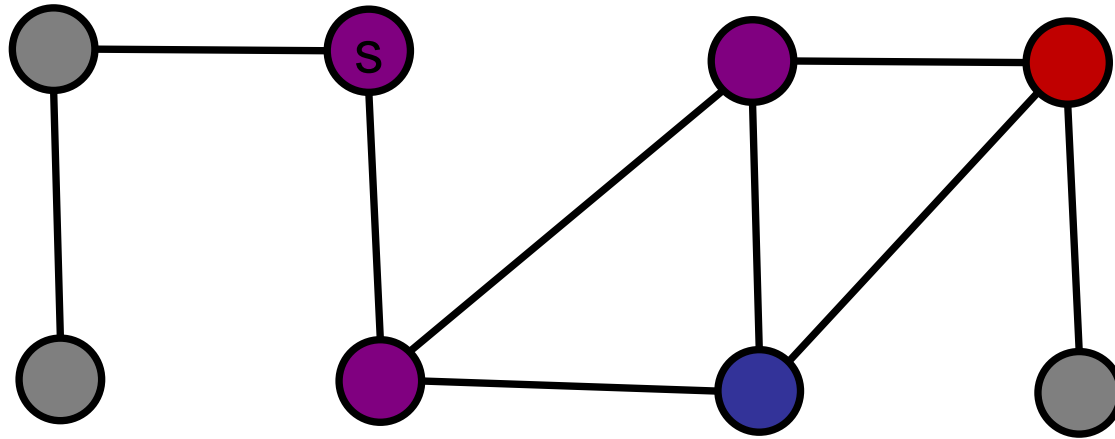
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



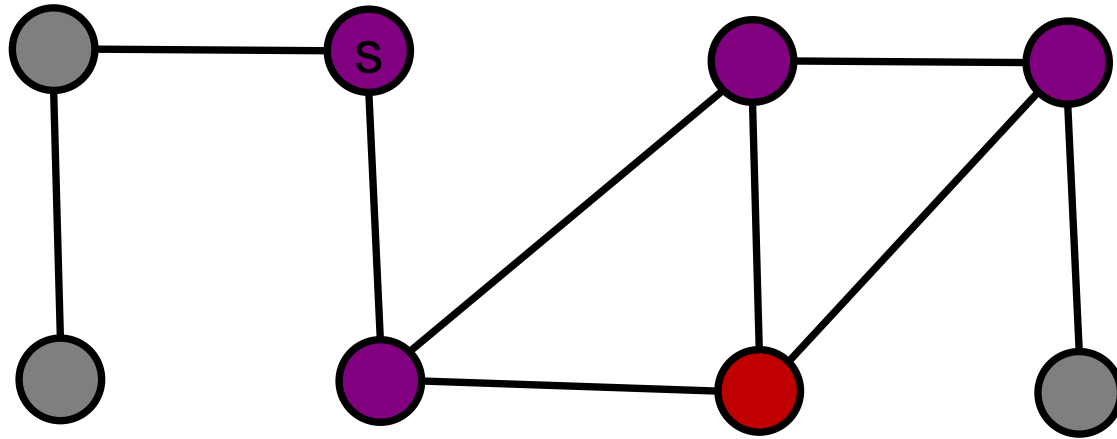
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



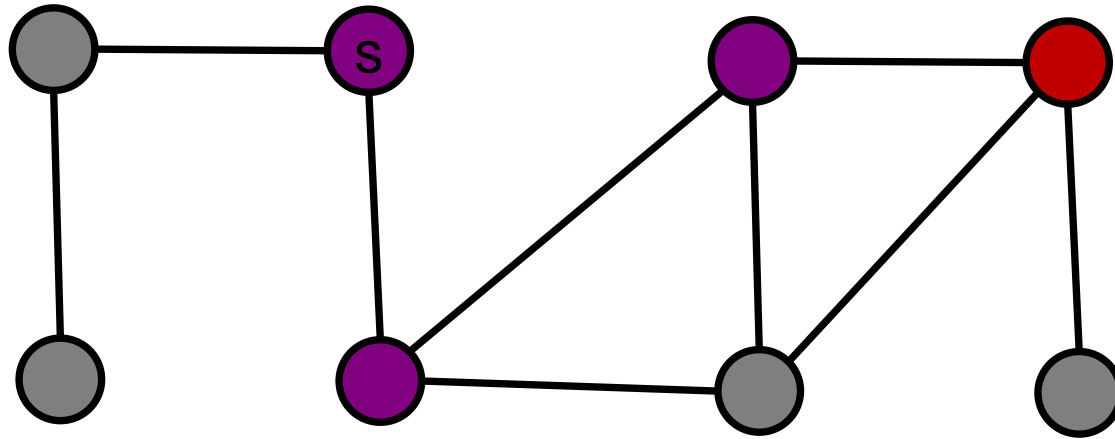
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



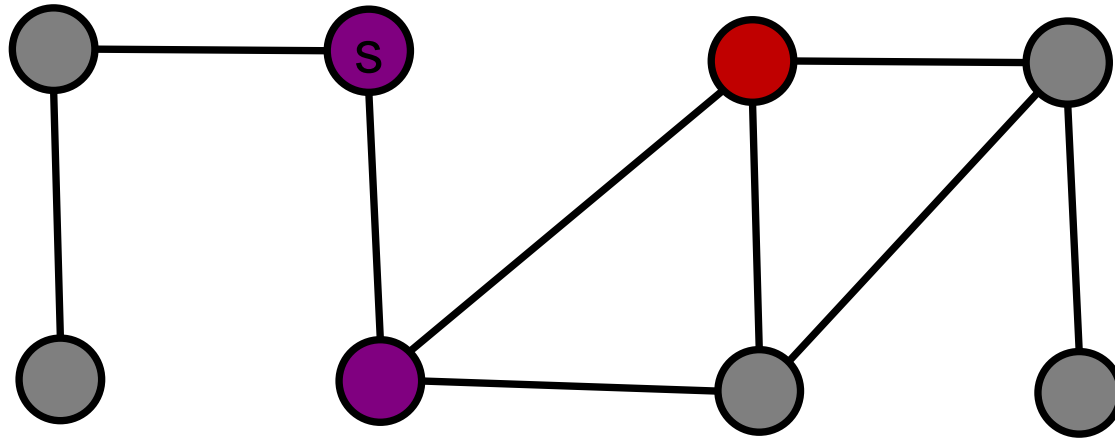
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



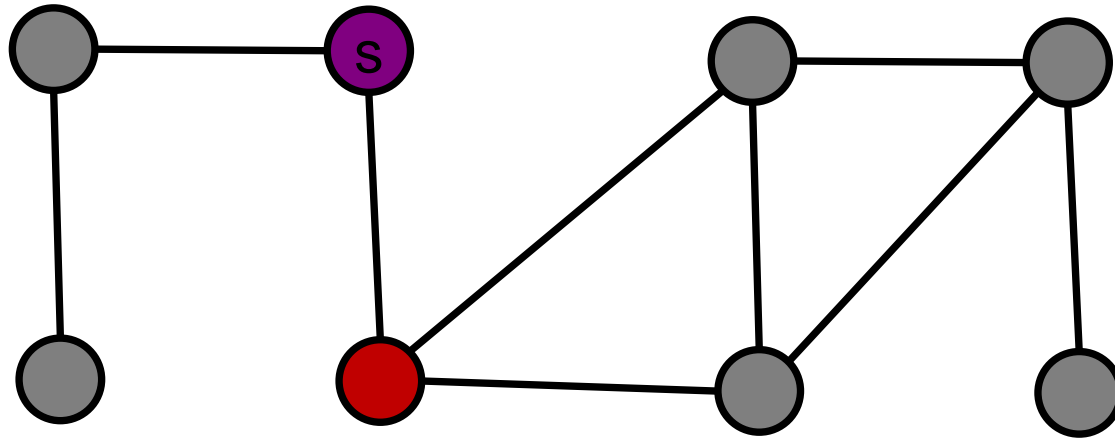
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



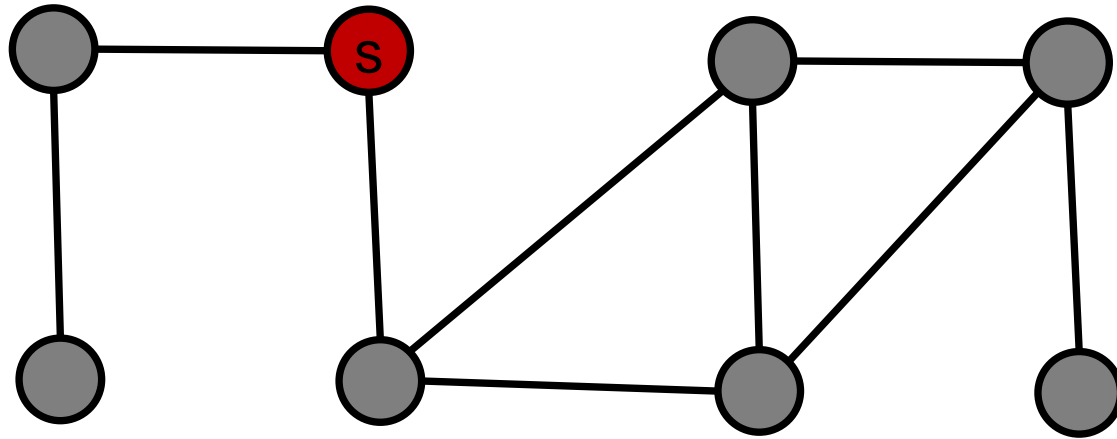
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



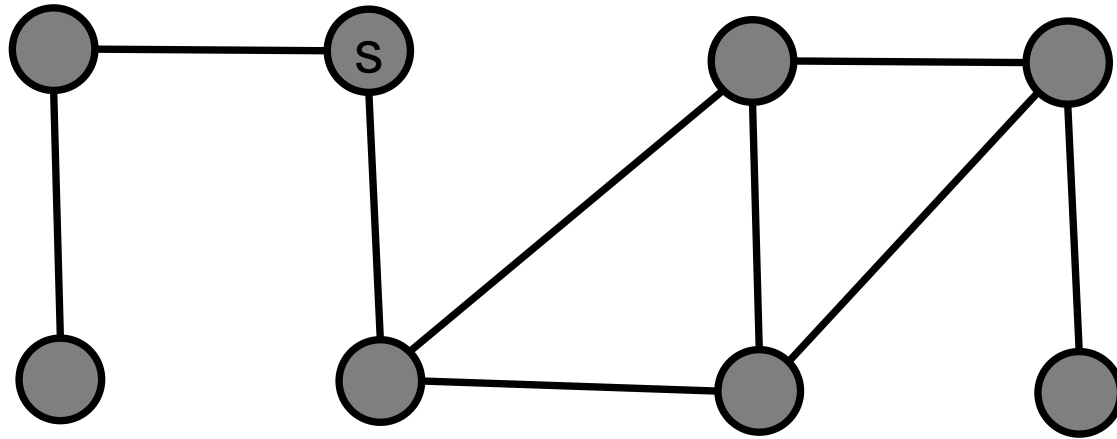
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example

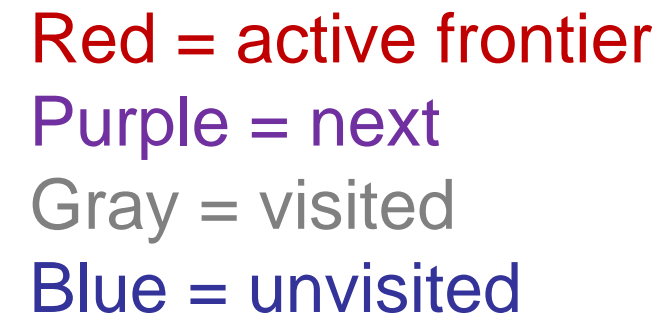


Red = active frontier

Purple = next

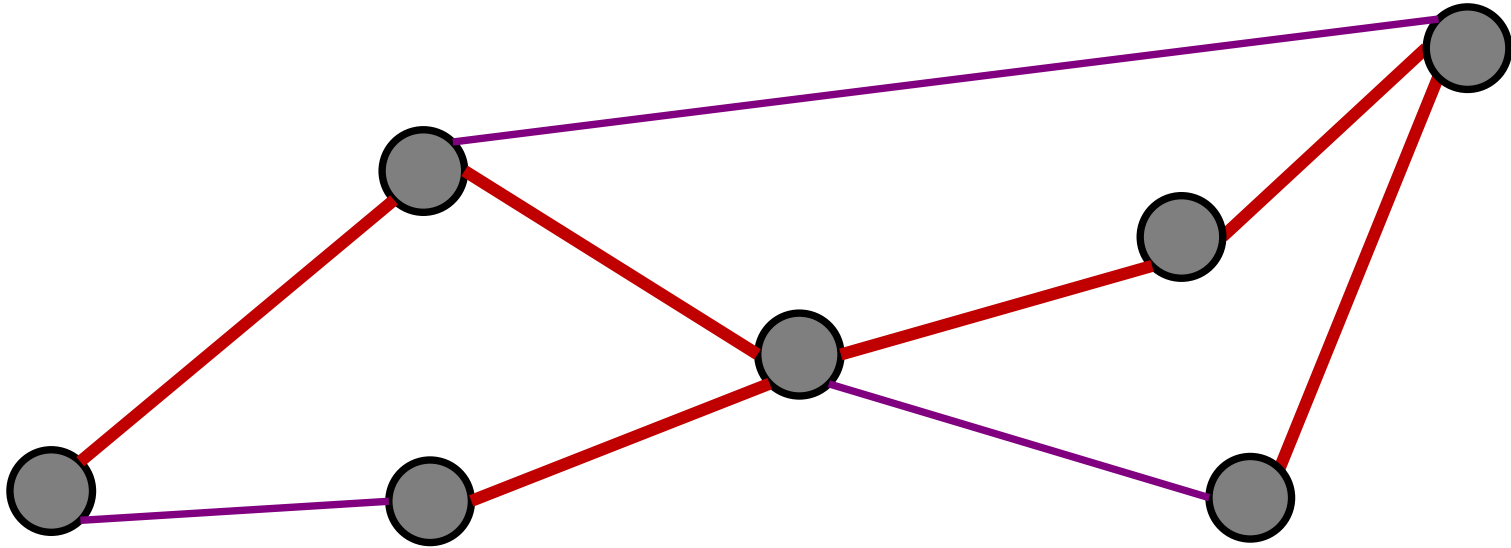
Gray = visited

Blue = unvisited



Blue = unvisited

DFS parent edges



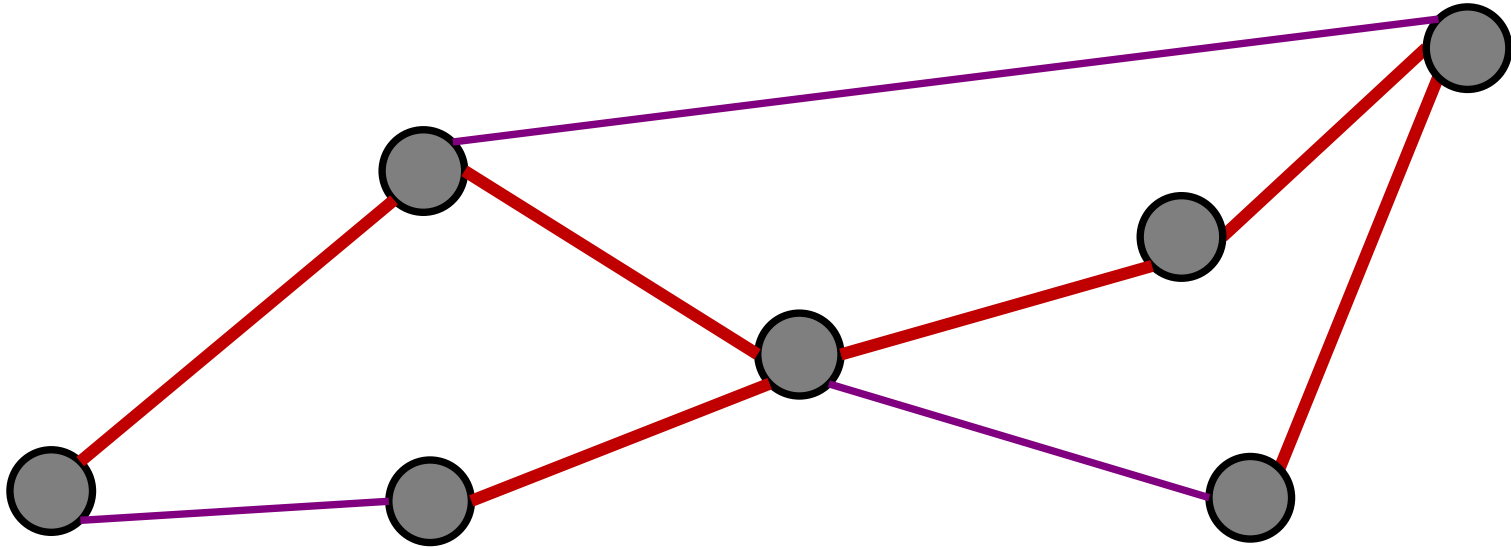
Red = Parent Edges

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.

DFS parent edges = tree



Red = Parent Edges

Purple = Non-parent edges

Note: not shortest paths!

The running time of DFS (using an adjacency list) is:


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

Depth-First Search

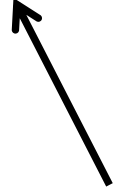
Analysis:

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

$O(V)$



$O(E)$



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. $O(E^2)$

ARCHIPELAGO


is open

Depth-First Search

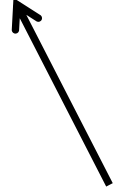
Analysis:

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

$O(V)$



$O(V)$
per
node



To implement an iterative version of DFS:

1. Use a queue.
- ✓ 2. Use a stack.
3. Use a set.
4. Don't.

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.

Visits

What do BFS and DFS visit?

- A) Every node
- B) Every edge
- C) Every path
- D) Every node and edge
- E) Every node, edge, and path

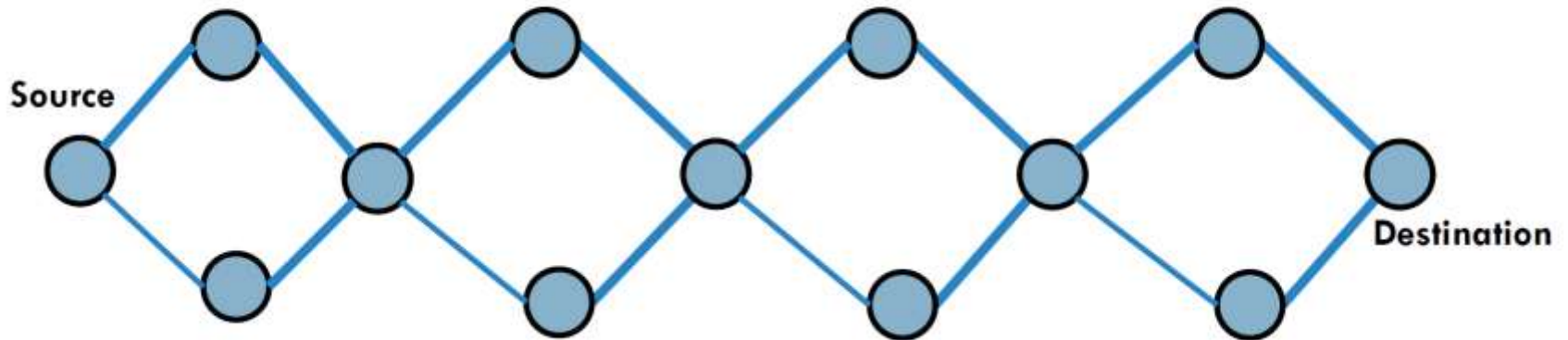


Visits

- BFS and DFS do **not** explore every path
- Once a node is visited, it's never explored again
- Too expensive: some graphs have an exponential number of paths!

Visits

- Too expensive: some graphs have an exponential number of paths!



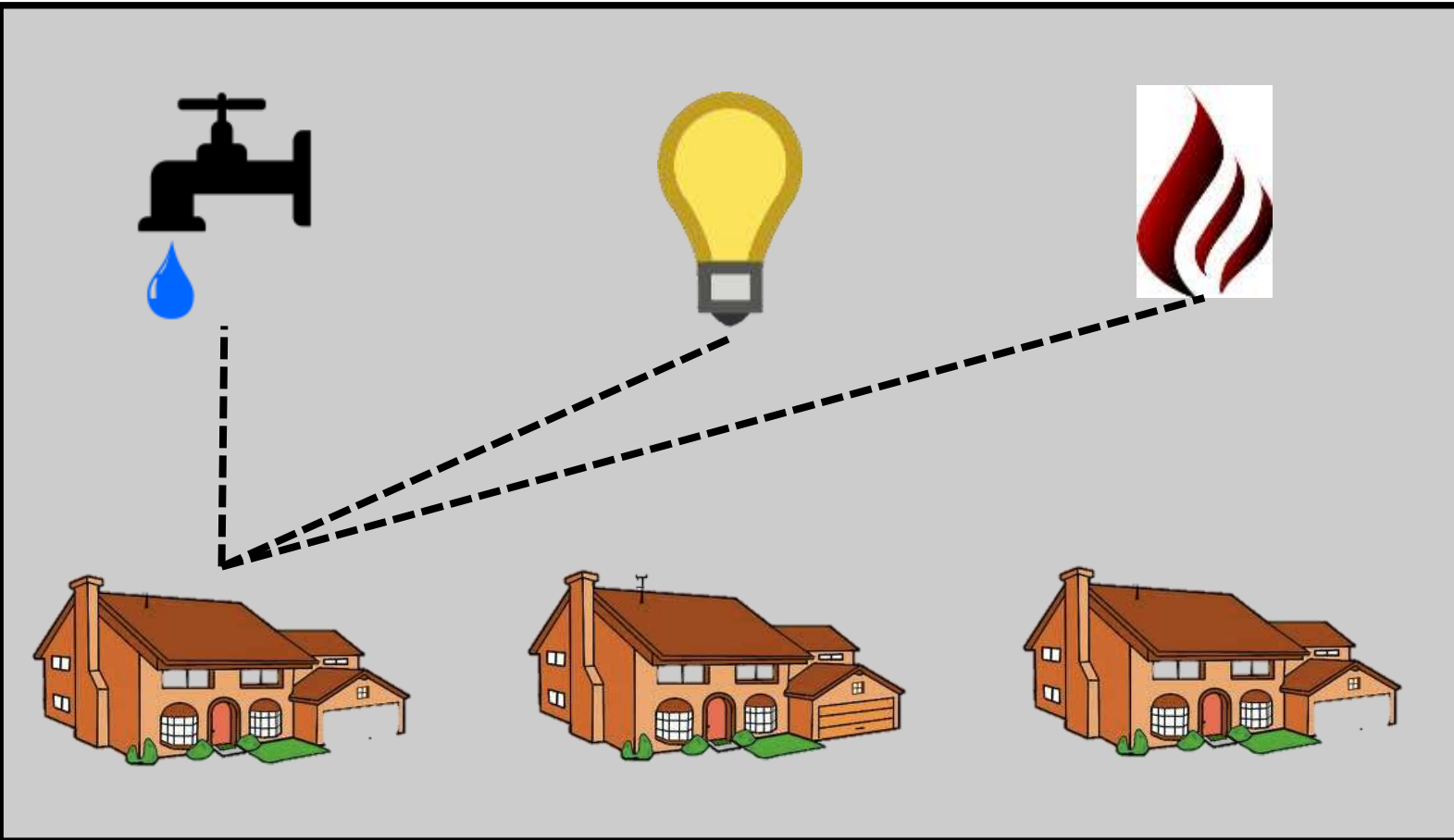
$2^{(n-1)/3}$ distinct paths from source to destination!

Roadmap

Today: Graph Basics

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

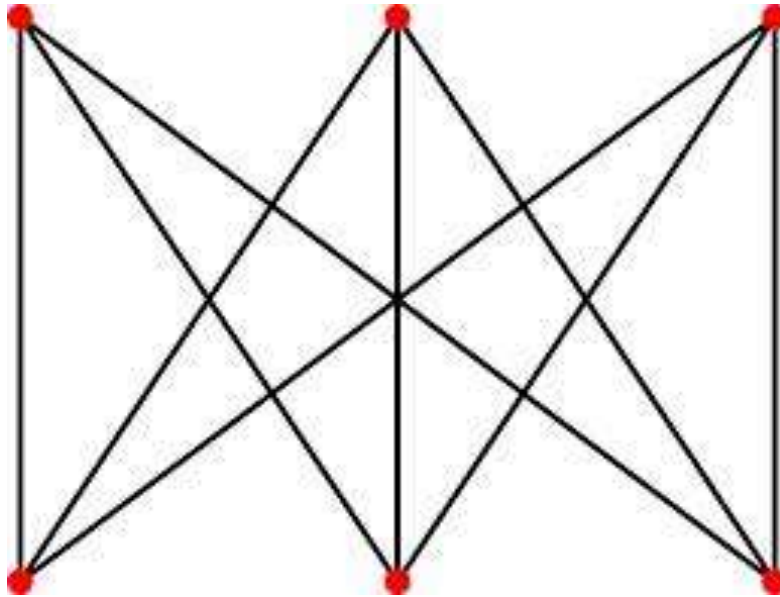
Puzzle



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?

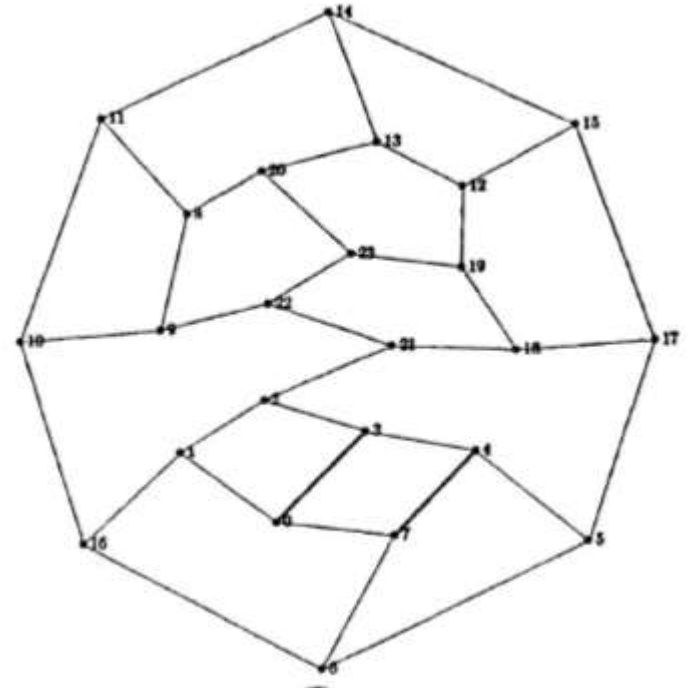
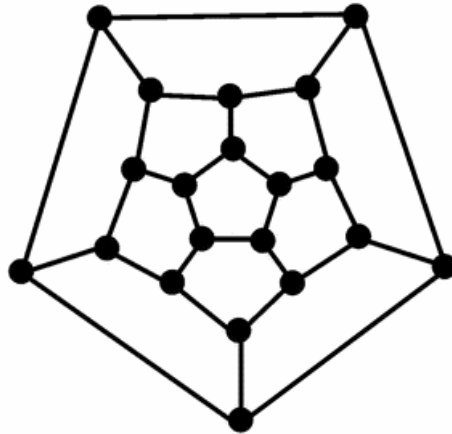
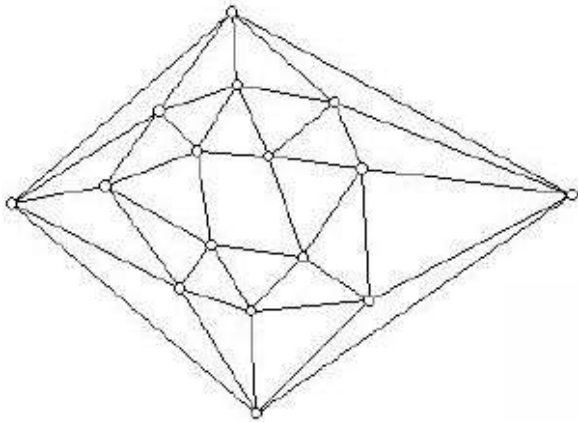


Bipartite Clique

Puzzle Explanation

Planar Graph:

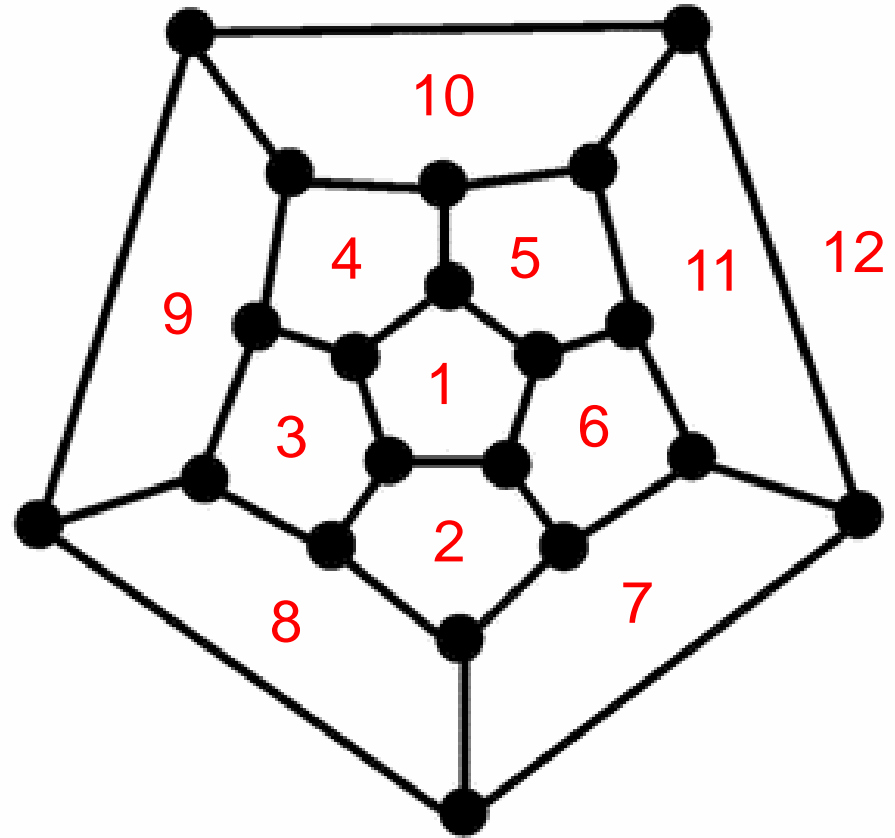
Any graph that can be drawn on a flat 2d piece of paper with no crossing lines.



Puzzle Explanation

Terms:

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



Puzzle Explanation

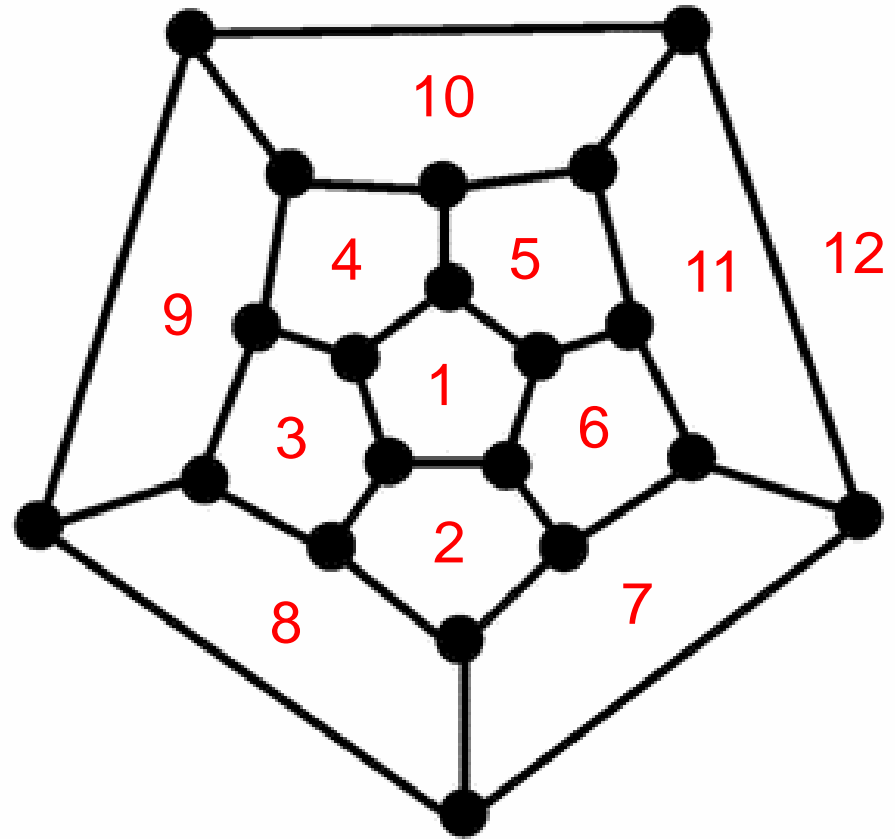
Euler's Formula:
(planar graphs)

$$V - E + F = 2$$

V = # vertices

E = # edges

F = # faces



Prove by induction.

Puzzle Explanation

Euler's Formula:
(planar graphs)

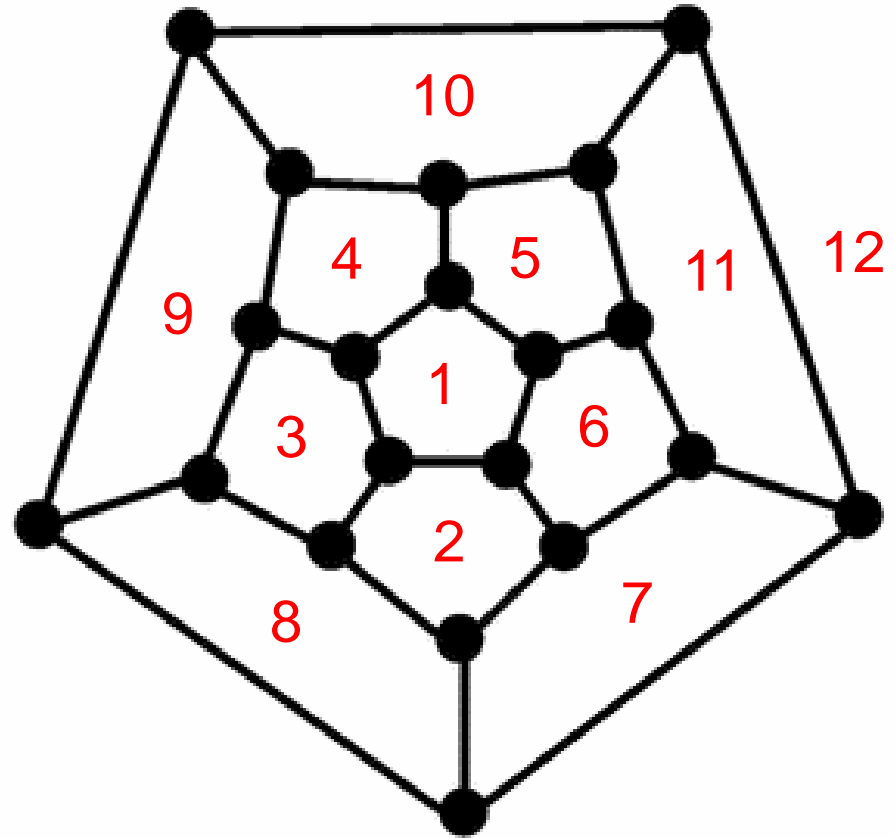
$$V - E + F = 2$$

$$V = 20$$

$$E = 30$$

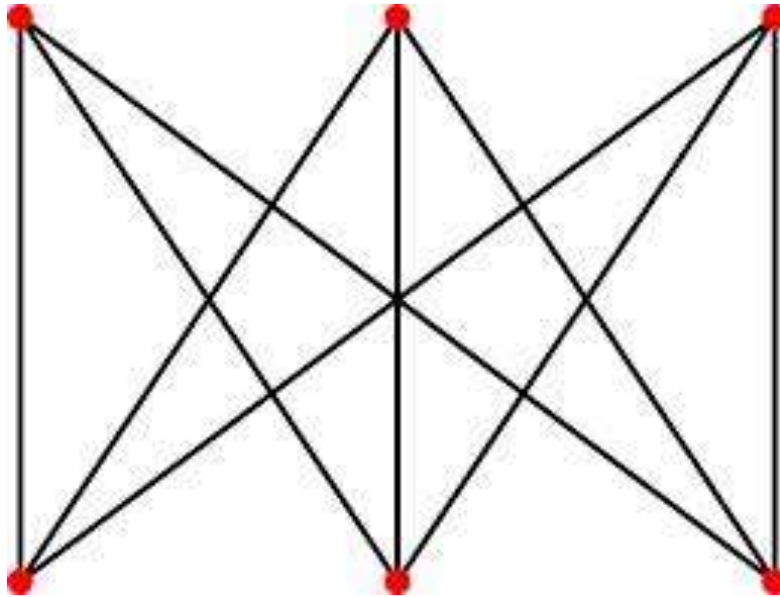
$$F = 12$$

$$20 - 30 + 12 = 2$$



Puzzle Explanation

Can you draw this graph with no crossing lines?



Bipartite Clique

Puzzle Explanation

Euler's Formula:
(planar graphs)

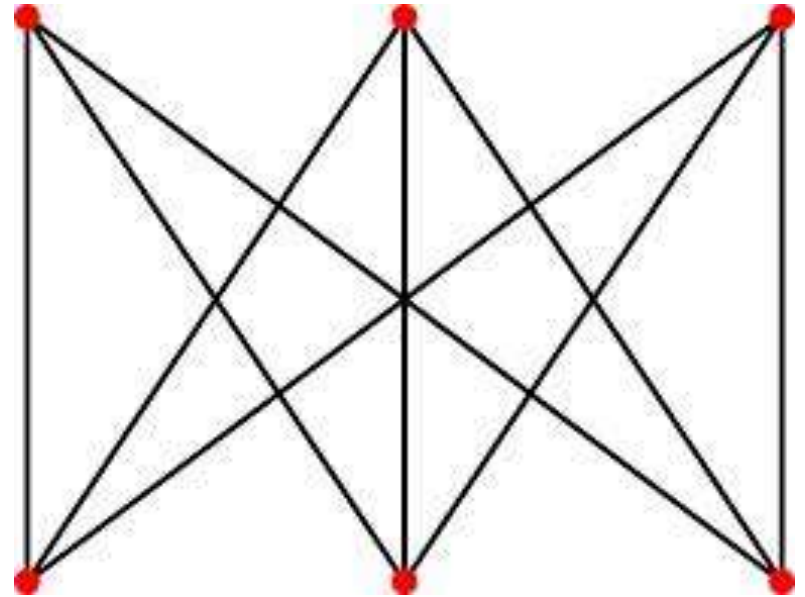
$$V - E + F = 2$$

$$V = 6$$

$$E = 9$$

$$F = ??$$

$$6 - 9 + F = 2$$



Puzzle Explanation

Euler's Formula:
(planar graphs)

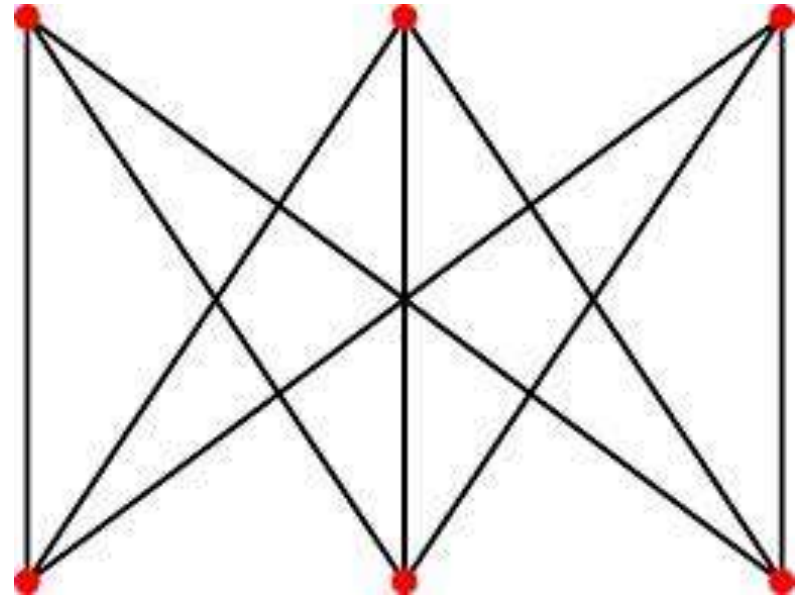
$$V - E + F = 2$$

$$V = 6$$

$$E = 9$$

$$F = 5$$

$$6 - 9 + F = 2$$



Puzzle Explanation

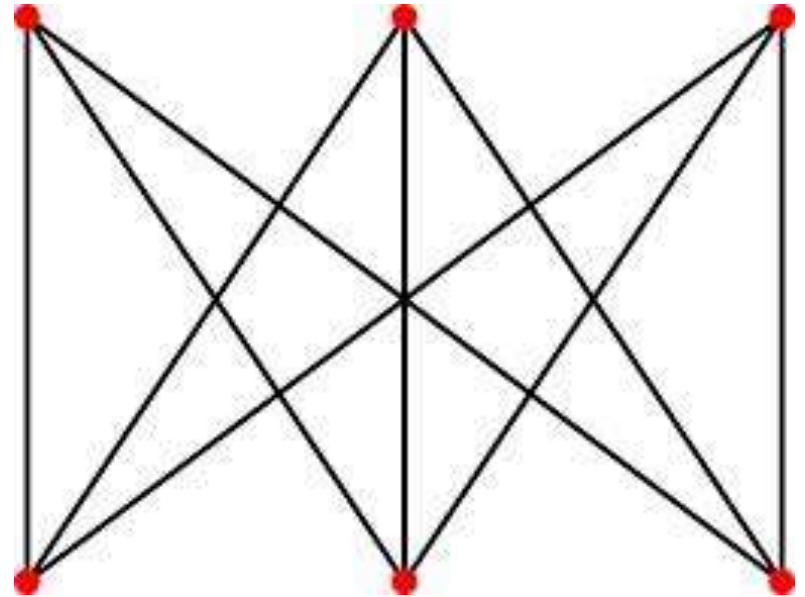
For bipartite clique:

Every face has at least 4 edges.

Every edge is used in at most 2 faces.

$$\rightarrow 2E \geq 4F \rightarrow$$

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



Puzzle Explanation

Impossible!

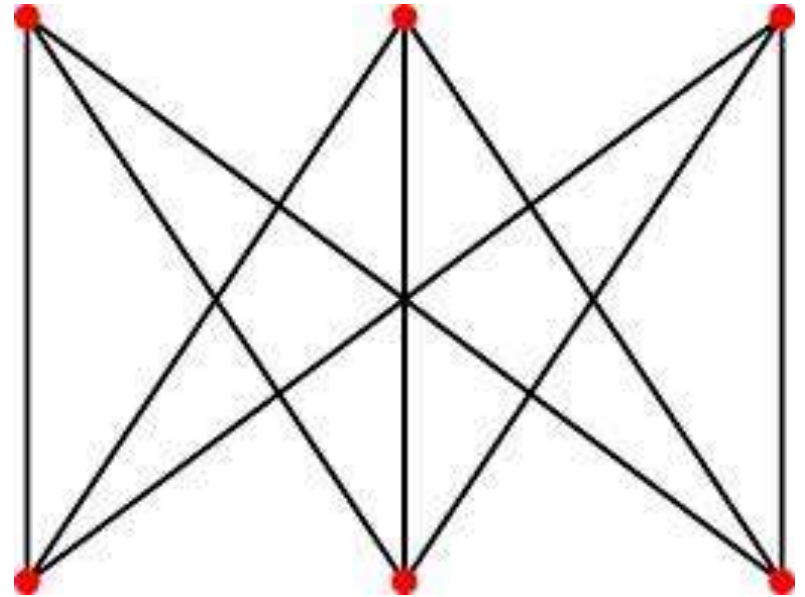
$$F \leq E/2$$

$$V = 6$$

$$E = 9$$

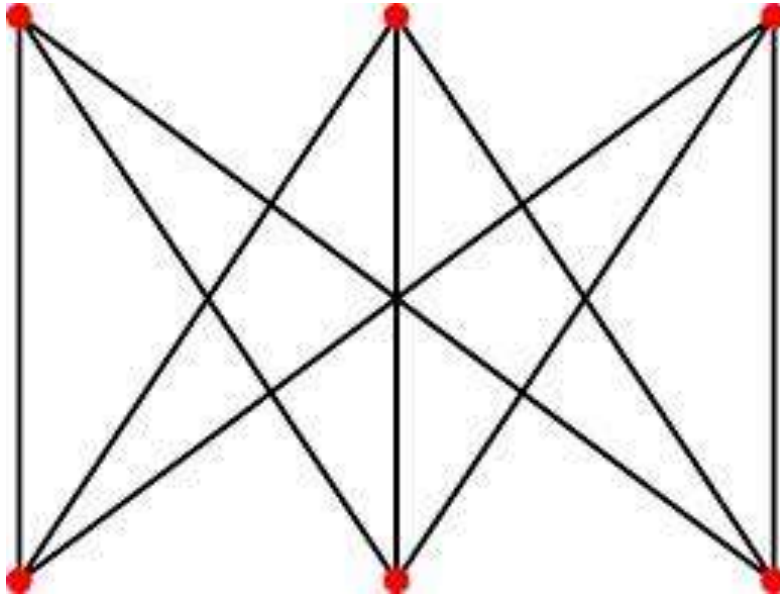
$$F = 5$$

$$\text{BUT: } 5 > 9/2$$



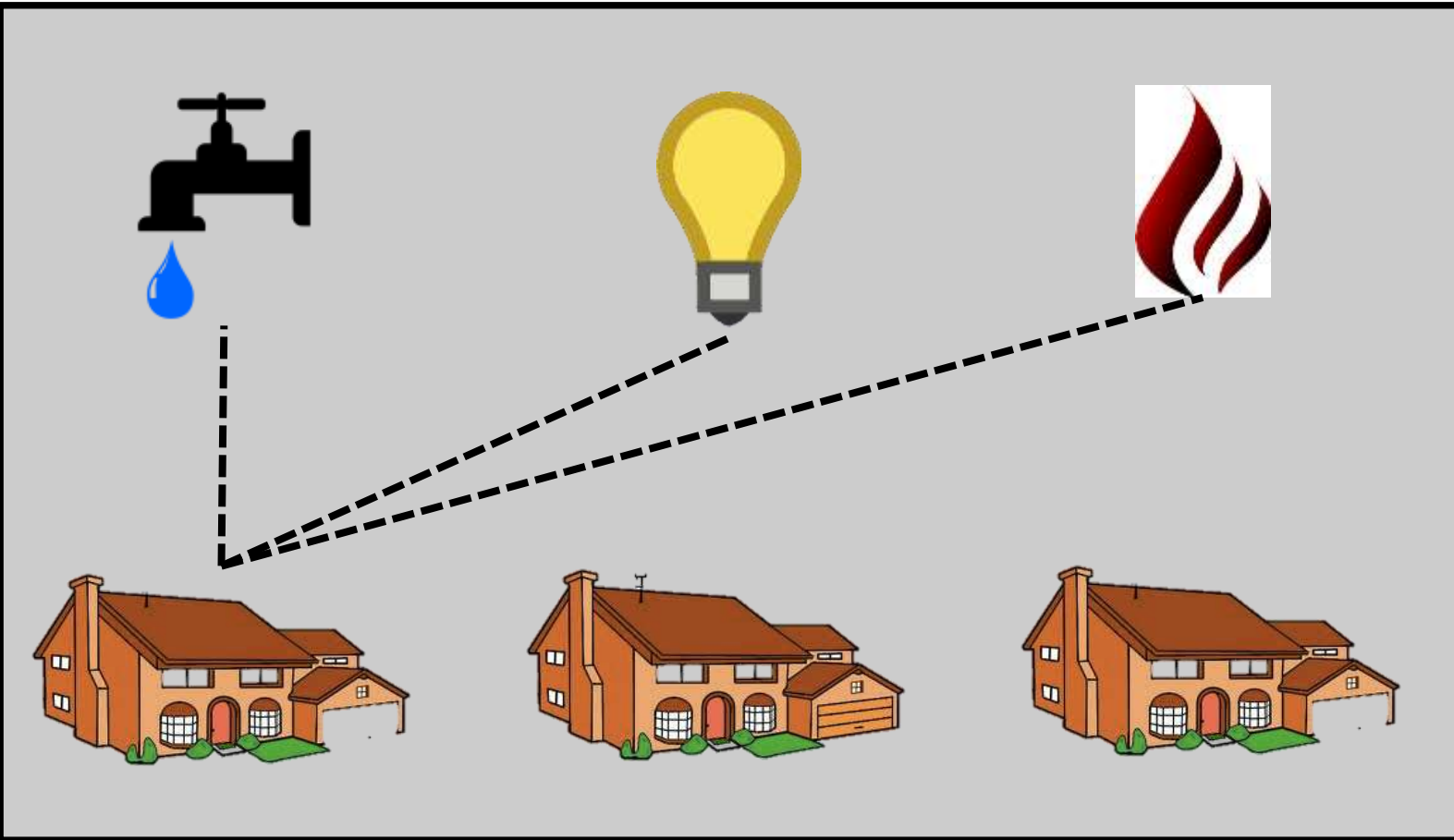
Puzzle Explanation

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.)