CS2040S
Data Structures and Algorithms

(e-learning edition)

Graphs!

(Part 4)

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)

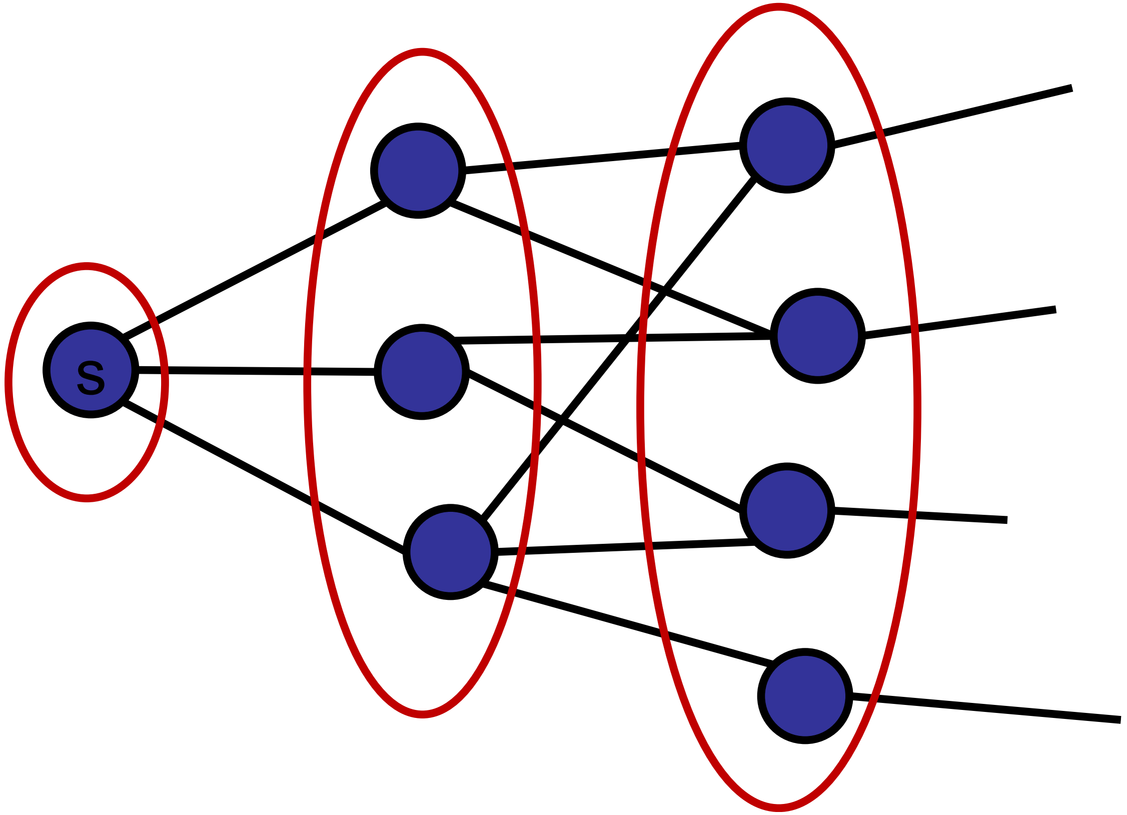
Graph representation:

- Adjacency list

Searching a graph

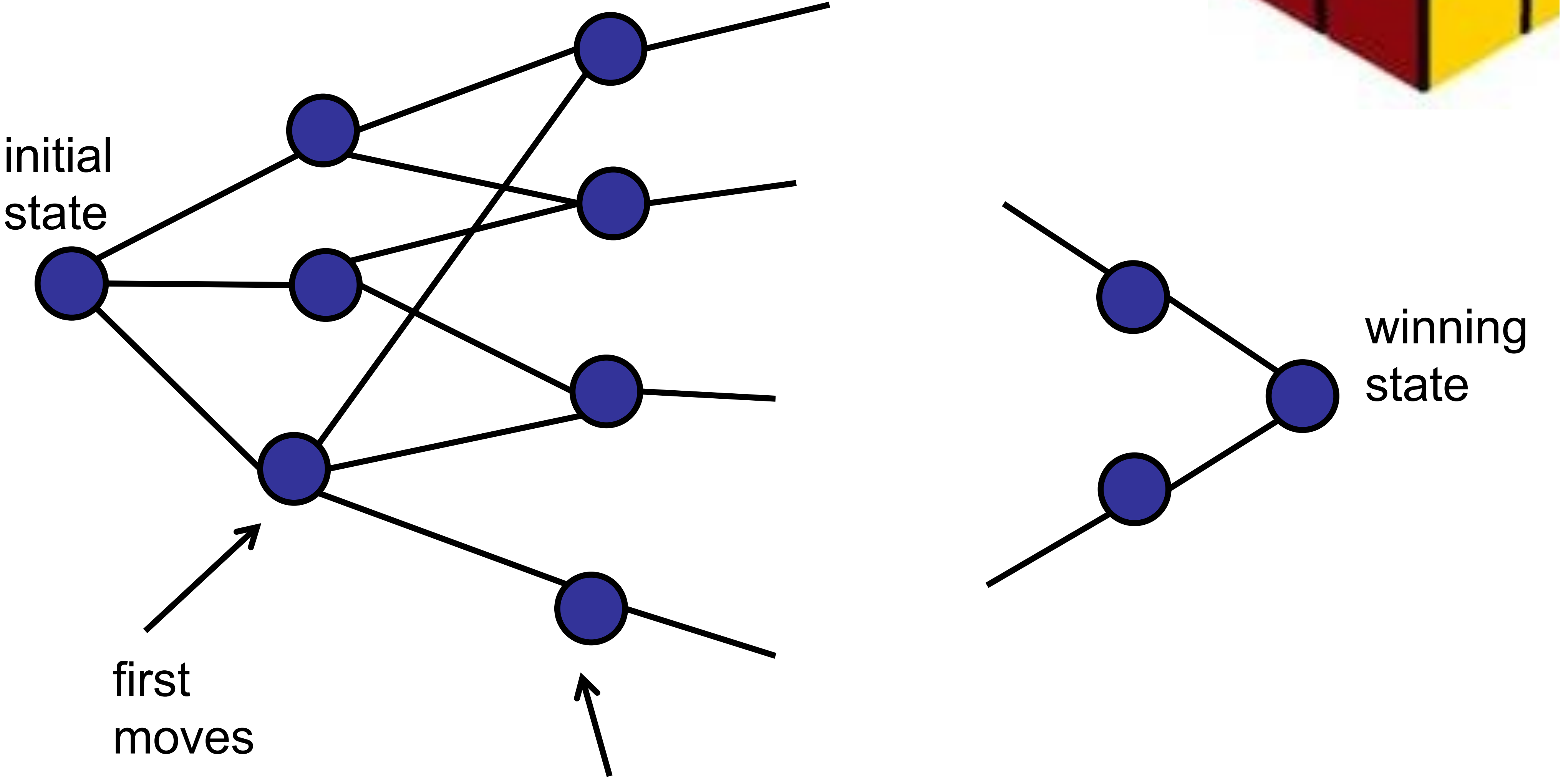
Breadth-First Search:

- Explore level by level



2 x 2 x 2 Rubik's Cube

Geography of Rubik's configurations:



initial state

first moves

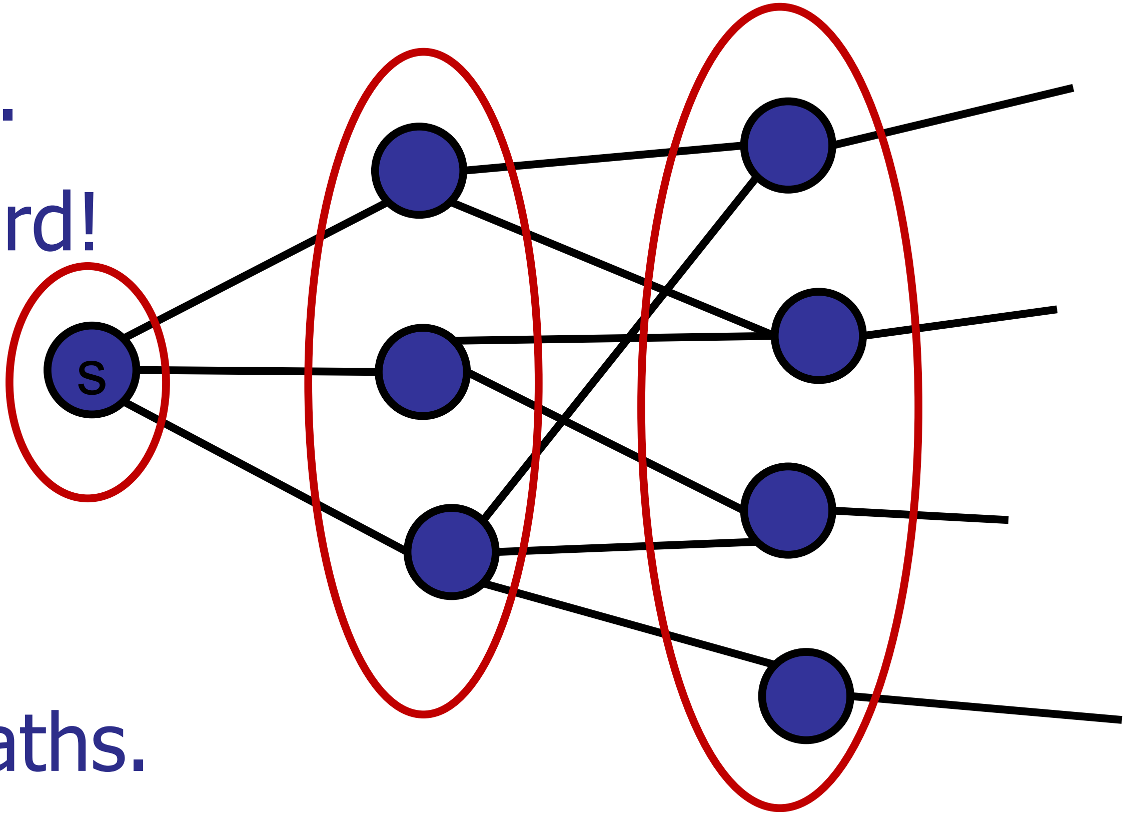
winning state

reachable in two moves, but not one

Searching a graph

Breadth-First Search:

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

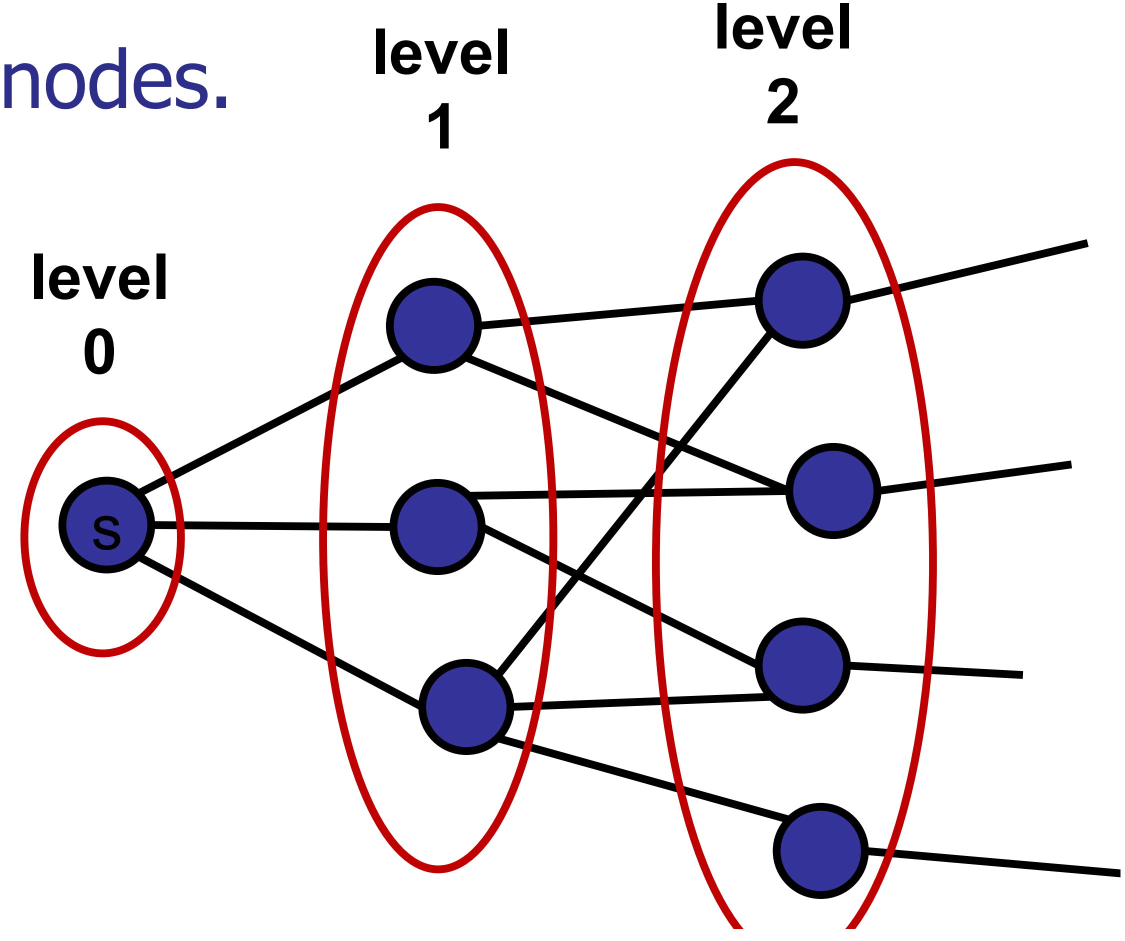


- Finds shortest paths.

Searching a graph

Breadth-First Search:

- Build levels.
- Calculate level[i] from level[i-1]
- Skip already visited nodes.



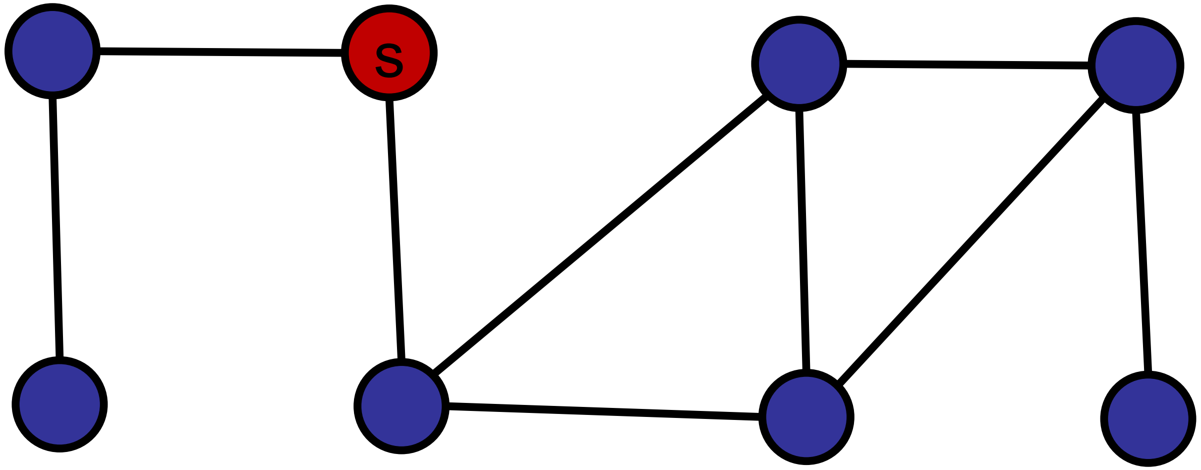
Breadth-First Search

```
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

```
while (!frontier.isEmpty()){
   Collection<Integer> nextFrontier = new … ;
   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



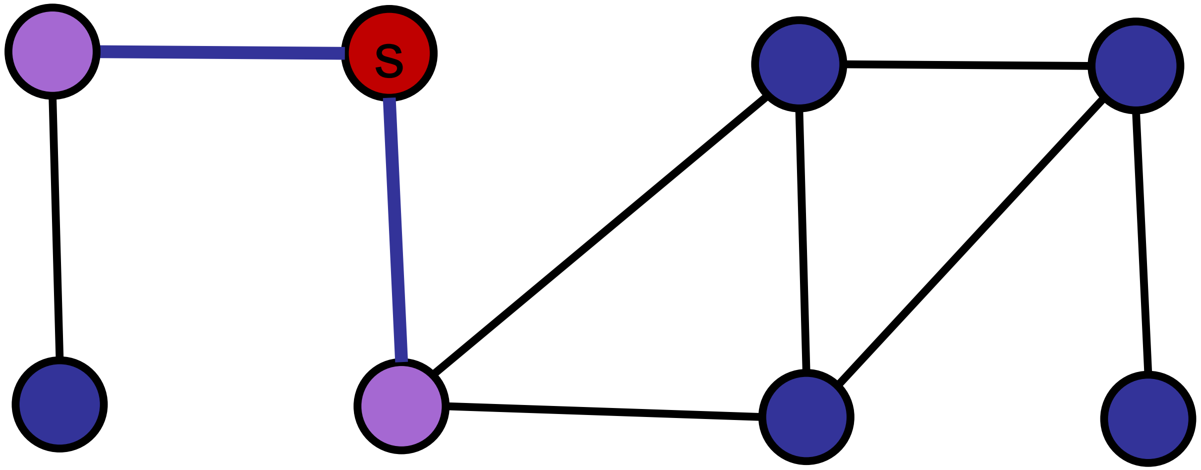
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



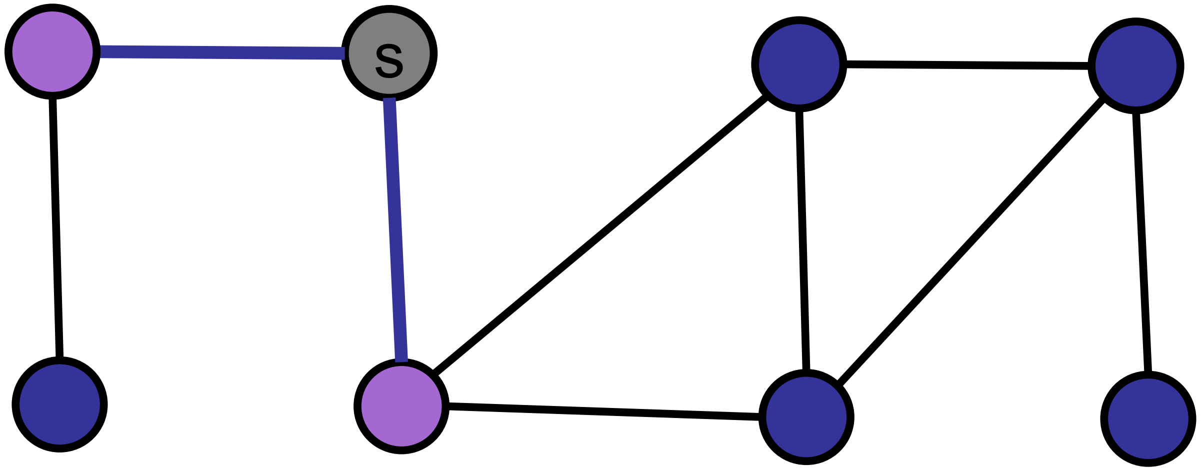
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



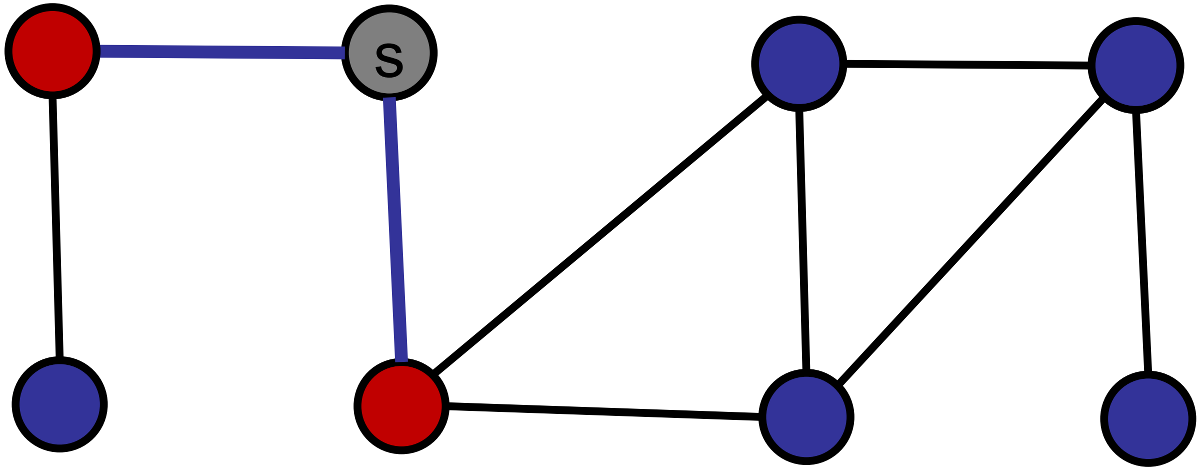
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



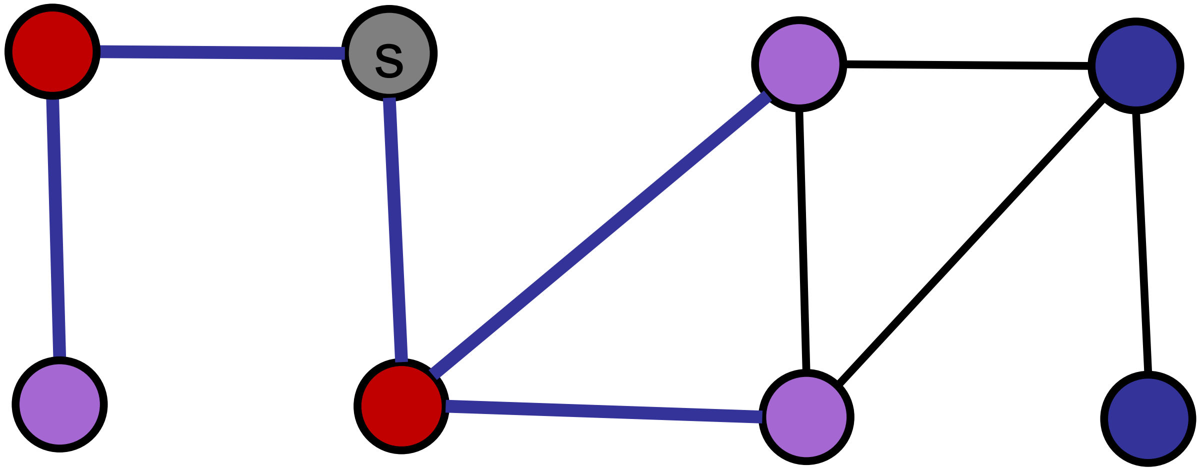
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



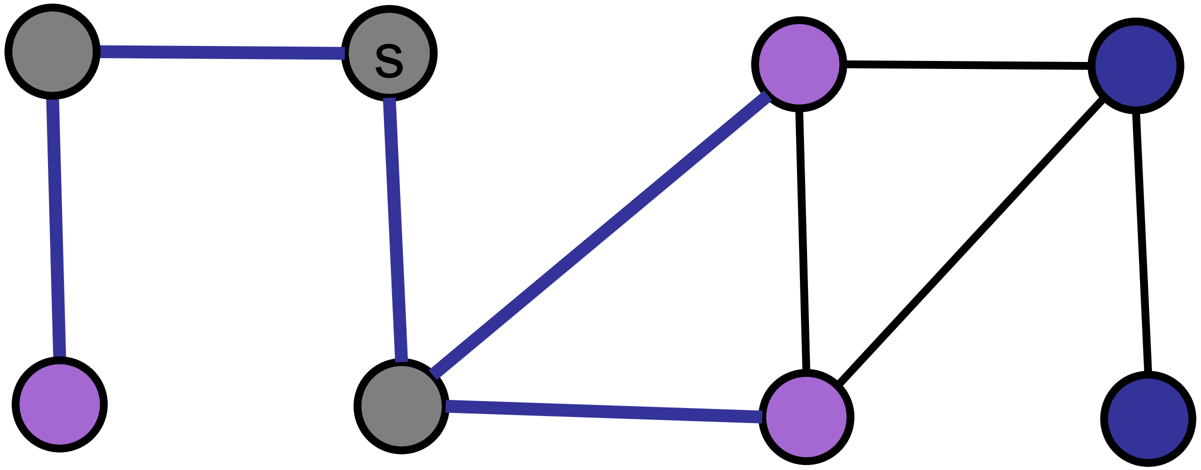
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



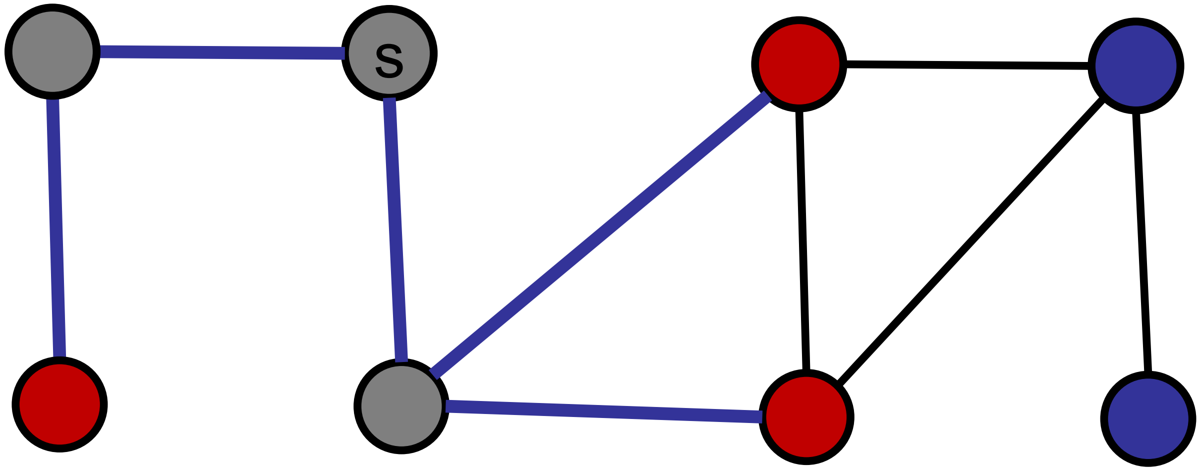
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



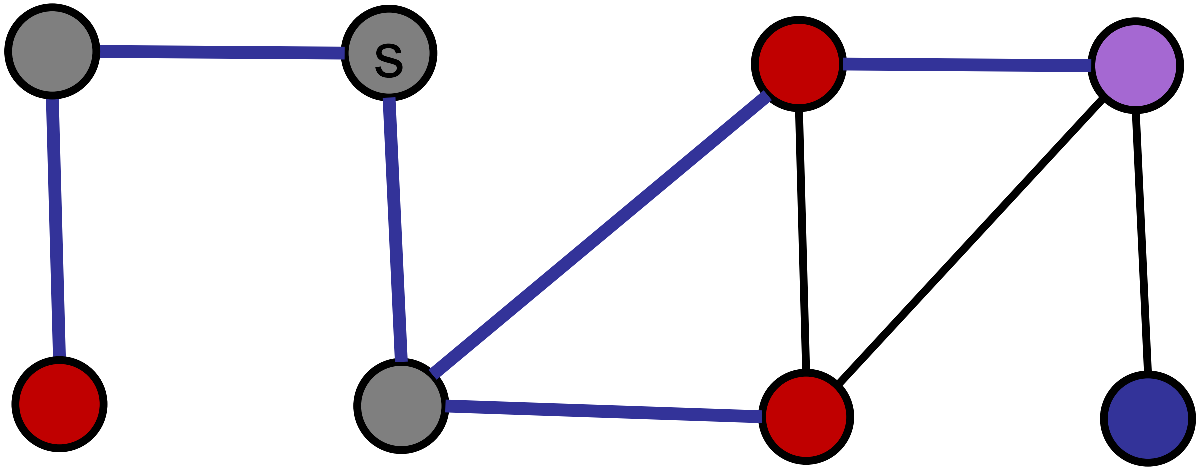
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



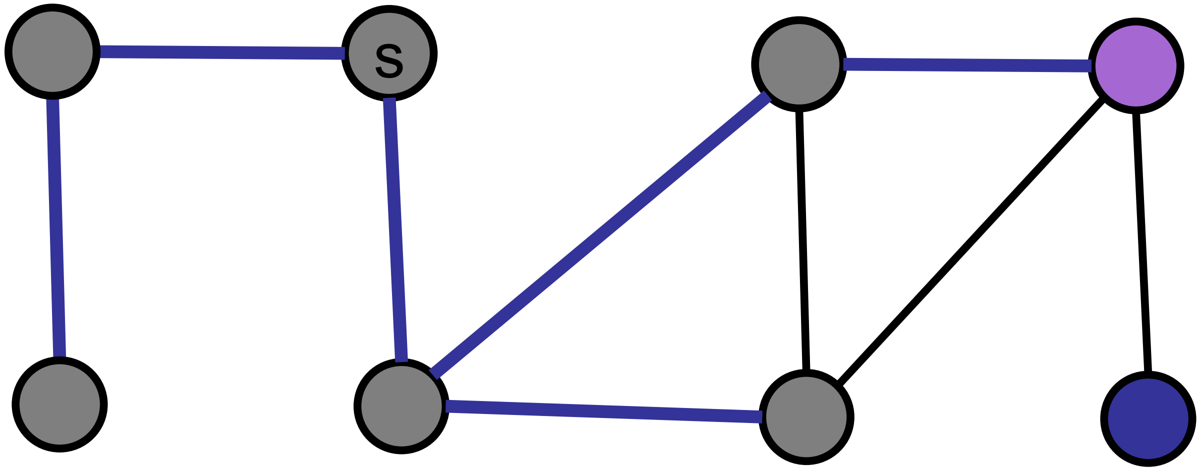
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



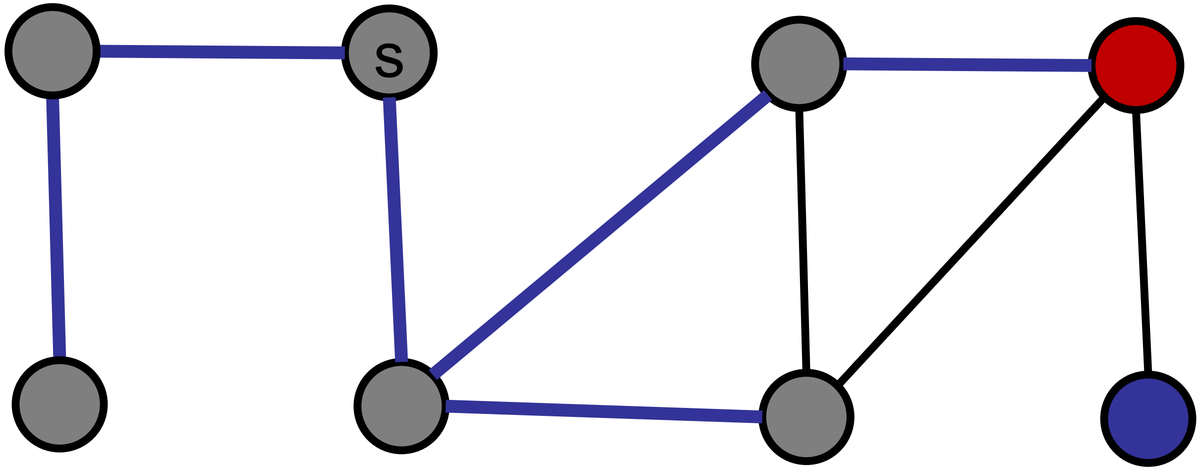
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



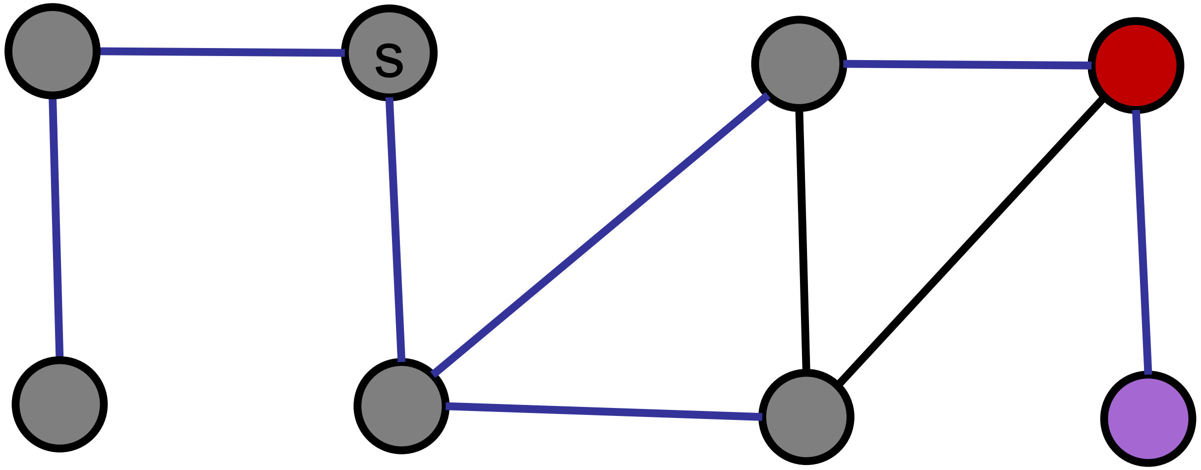
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



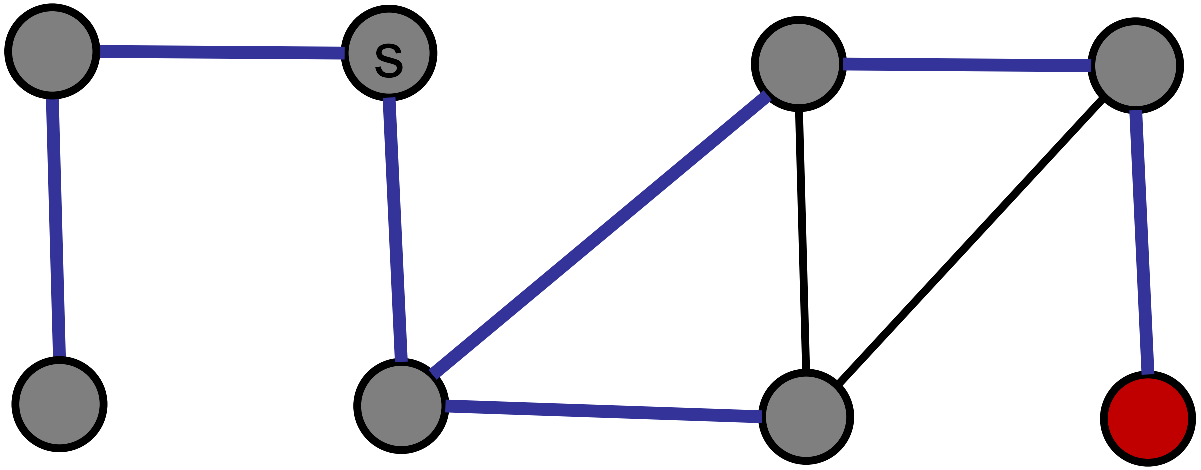
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



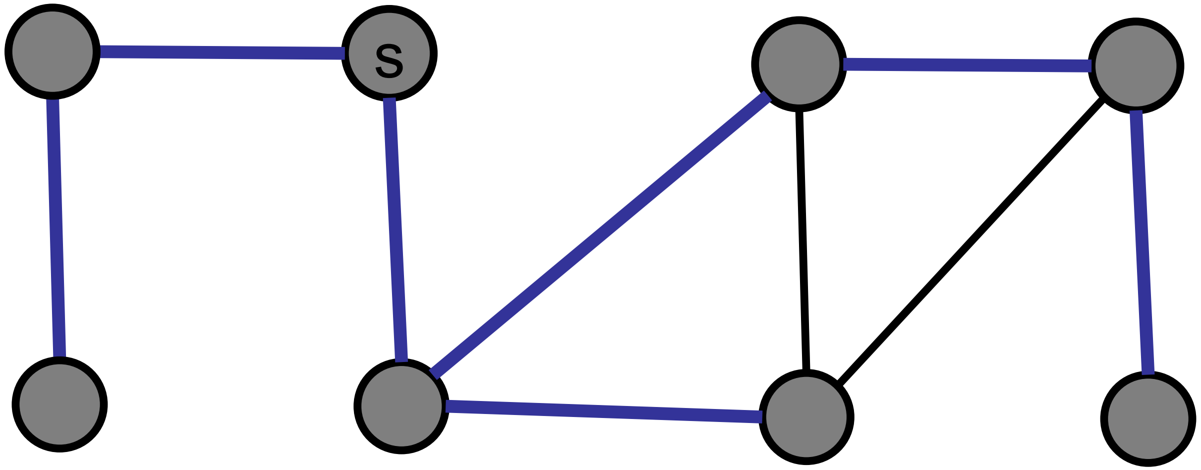
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



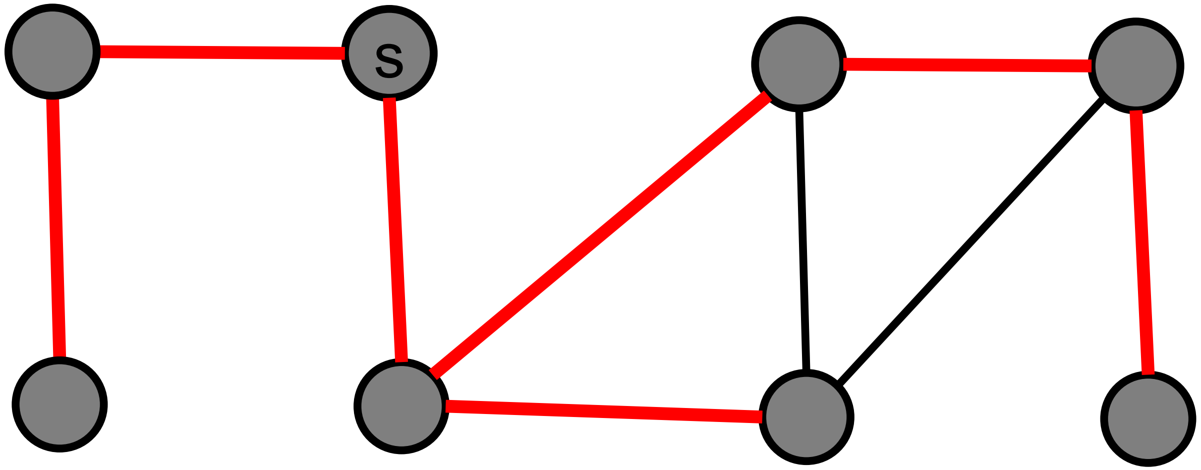
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Breadth-First Search Example



Red = active frontier

Purple = next

Gray = visited

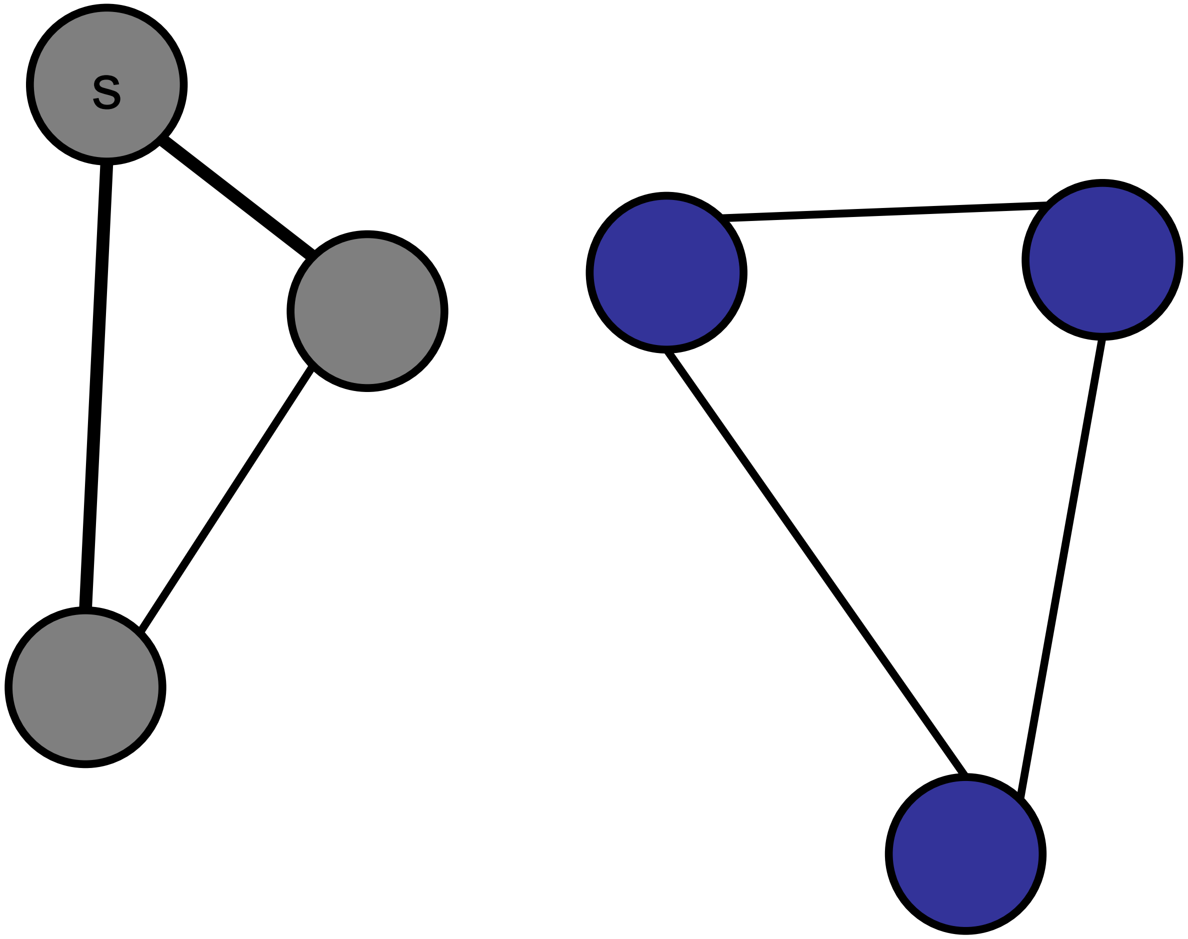
Blue = unvisited

When does BFS fail to visit every node?

1. In a clique.
2. In a cycle.
3. [✓ In a graph with two components.]
4. In a sparse graph.
5. In a dense graph.
6. Never.

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]){
           Bag<Integer> frontier = new Bag<Integer>;
           frontier.add(startId);

           // Main code goes here!
     }
  }
```

The running time of BFS is:

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

Breadth-First Search

Analysis:

- Vertex v = “start” once. ← O(V)
- Vertex v added to nextFrontier (and frontier) once.
  - After visited, never re-added.
- Each v.nbrlist is enumerated once.
  - When v is removed from frontier. ↖ O(E)

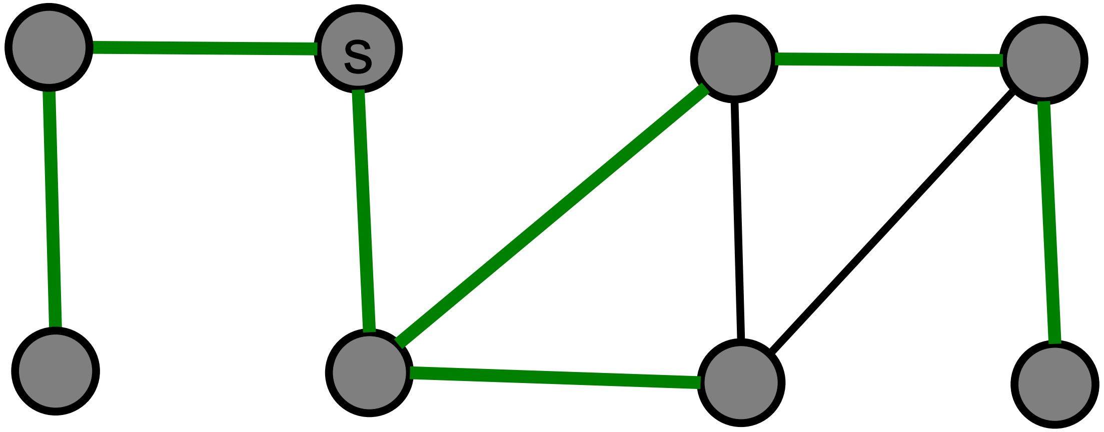
Breadth-First Search

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

Breadth-First Search

Shortest paths:

- Parent pointers store shortest path.



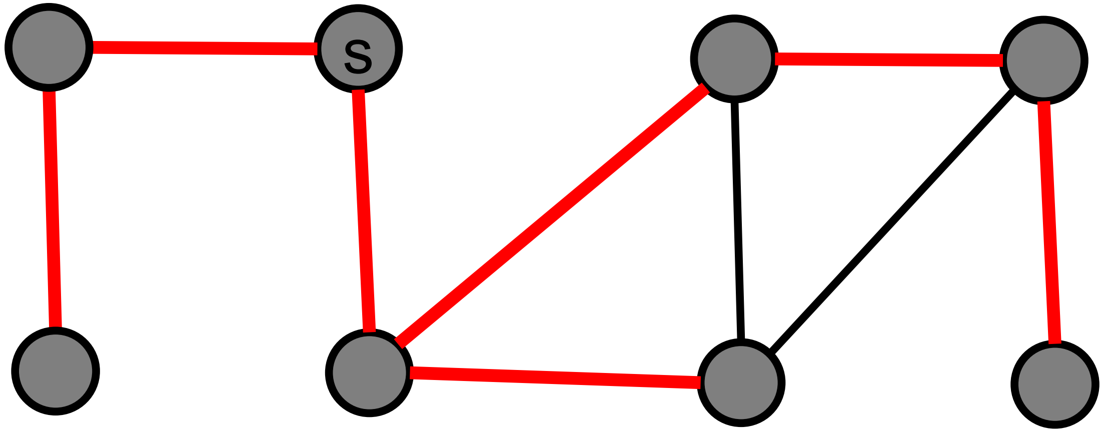
Which is true? (More than one may apply.)

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)

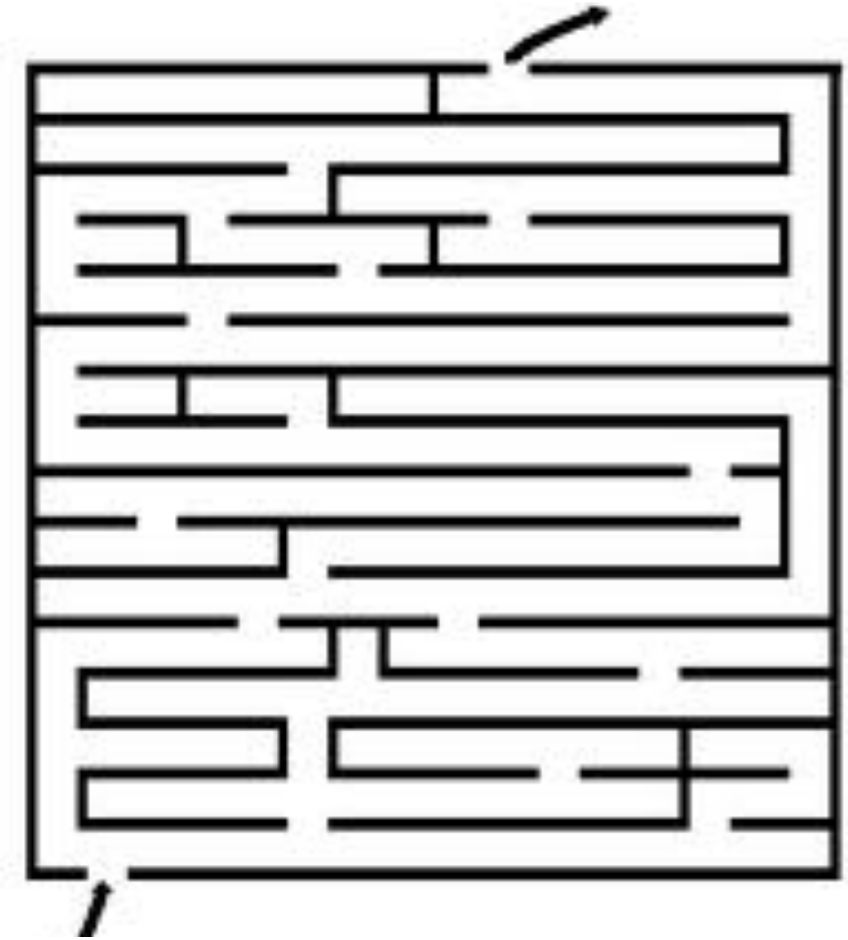
Graph representation:

- Adjacency list

Depth-First Search

Exploring a maze:

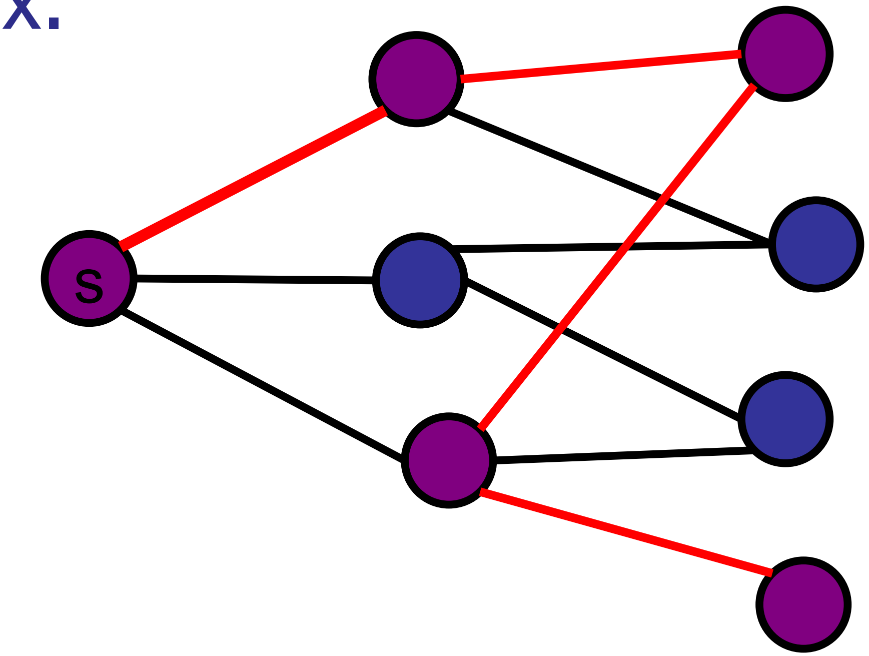
- Follow path until stuck.
- Backtrack along breadcrumbs until reach unexplored neighbor.
- Recursively explore.



Searching a graph

Depth-First Search:

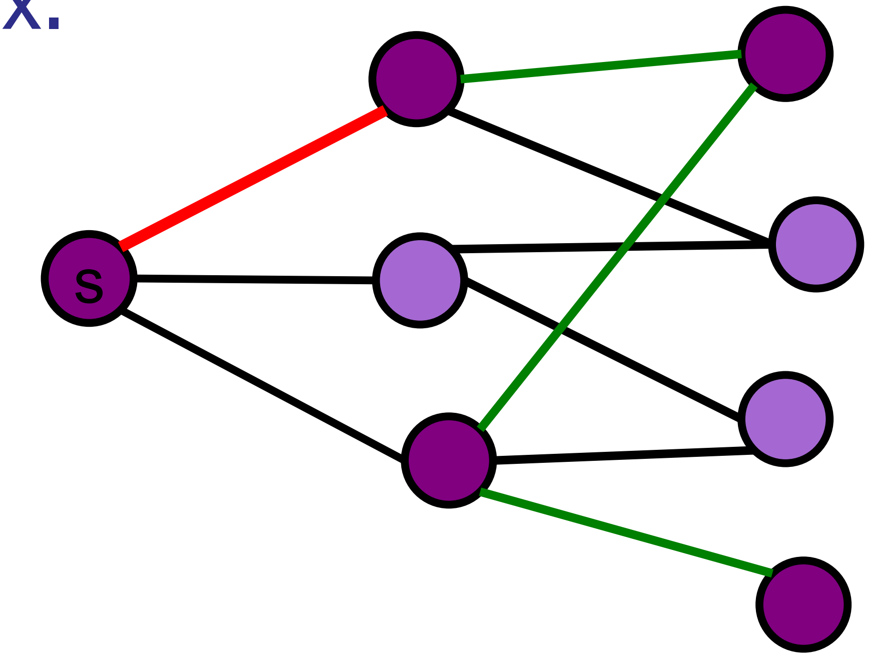
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don’t repeat a vertex.



Searching a graph

Depth-First Search:

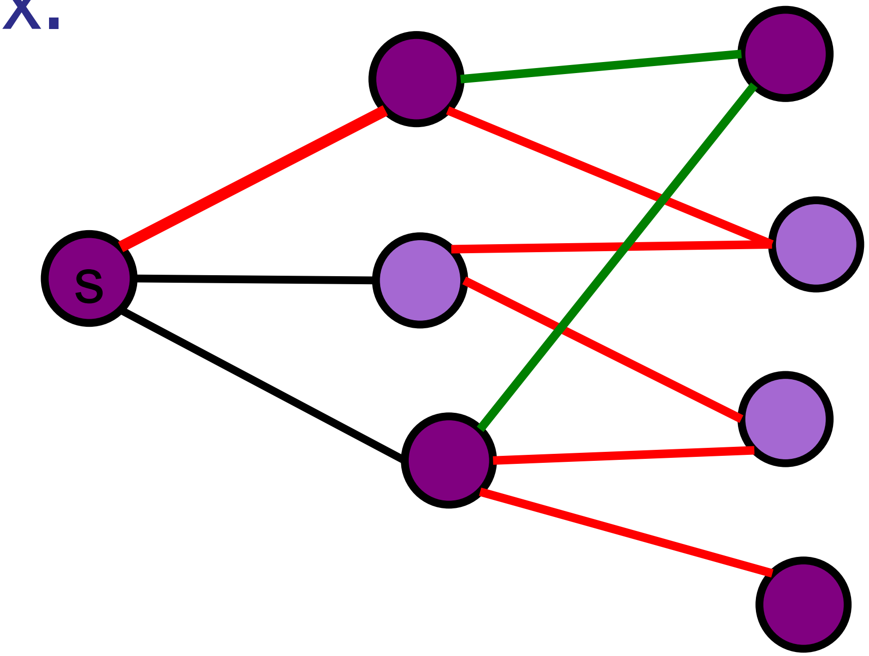
- Follow path until you get stuck
- Backtrack until you find a new edge
- Recursively explore it
- Don’t repeat a vertex.



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.



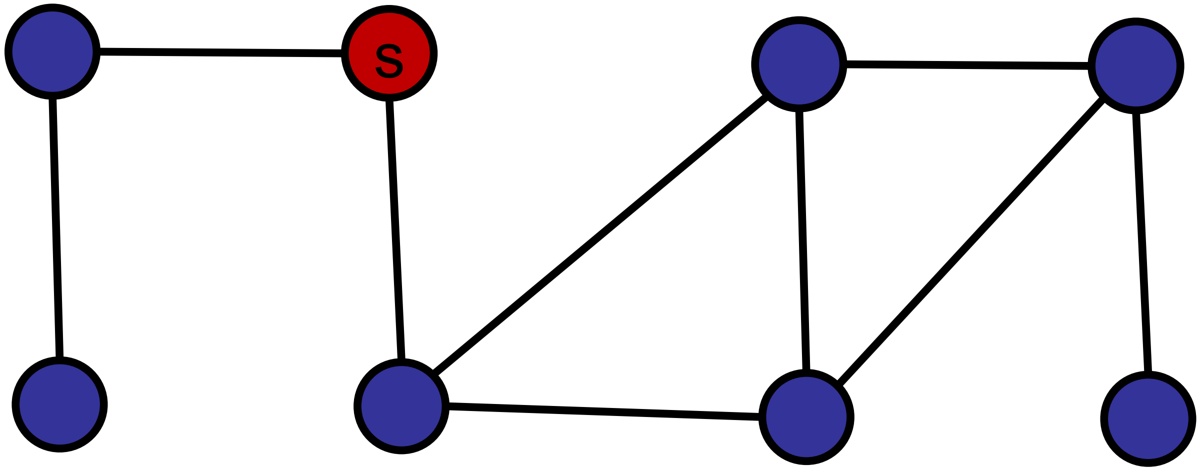
Depth-First Search

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

Depth-First Search Example



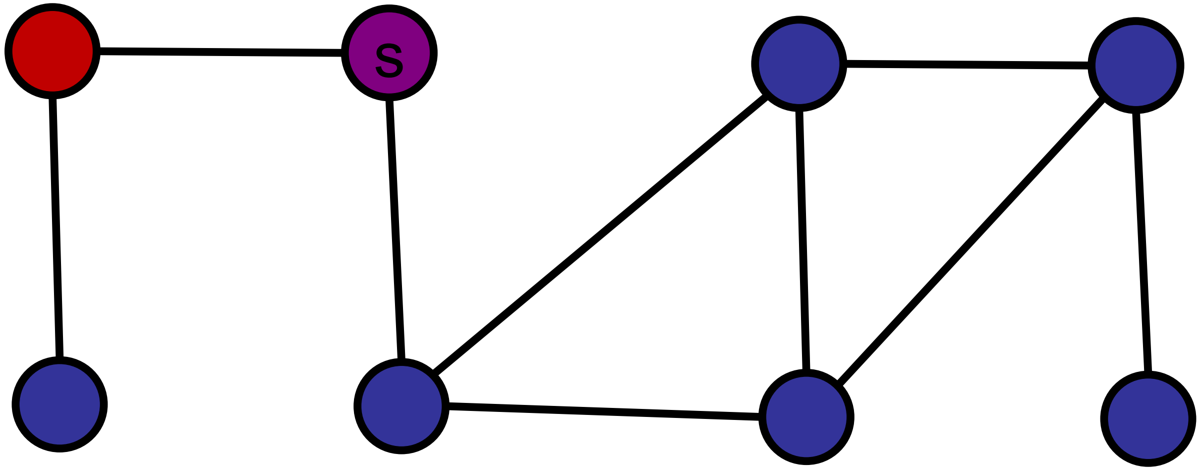
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



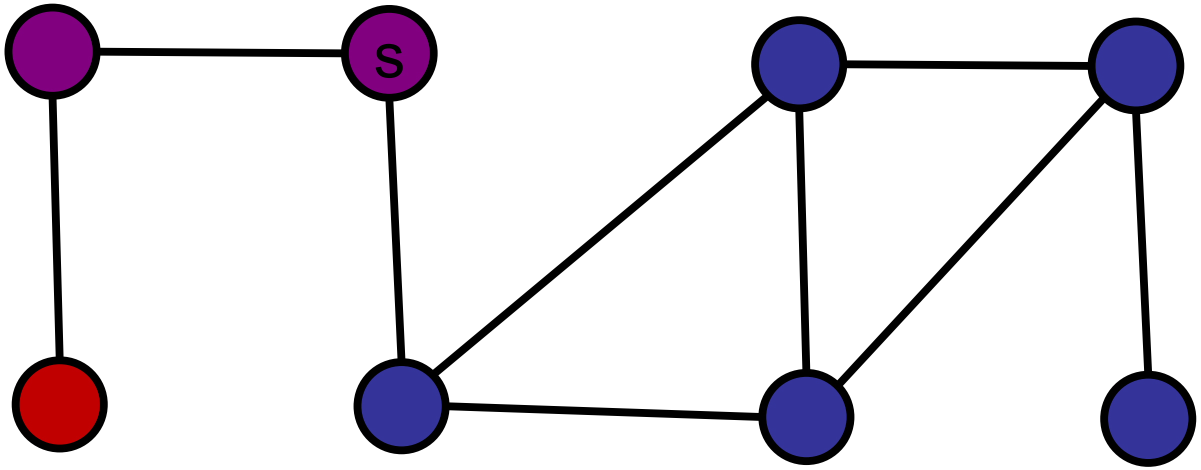
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



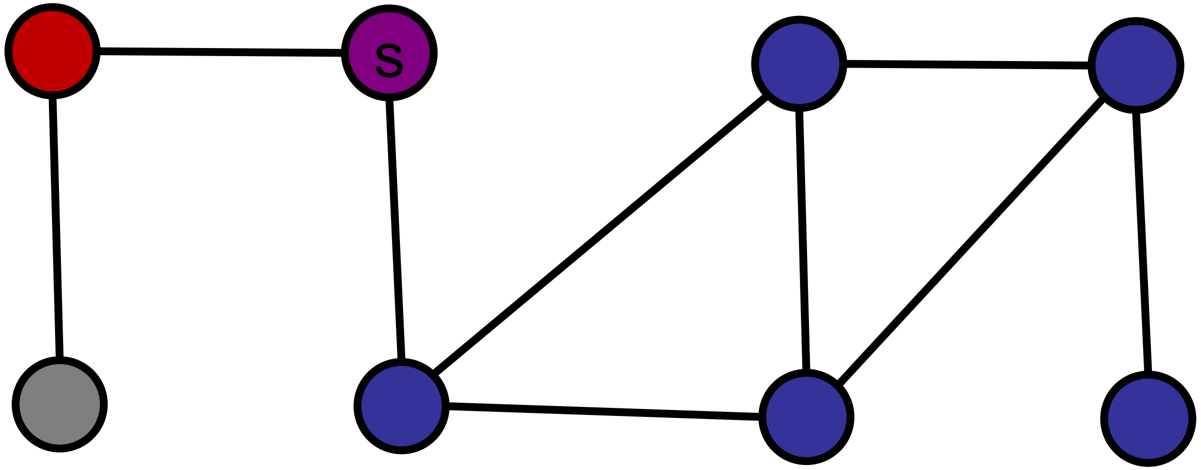
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



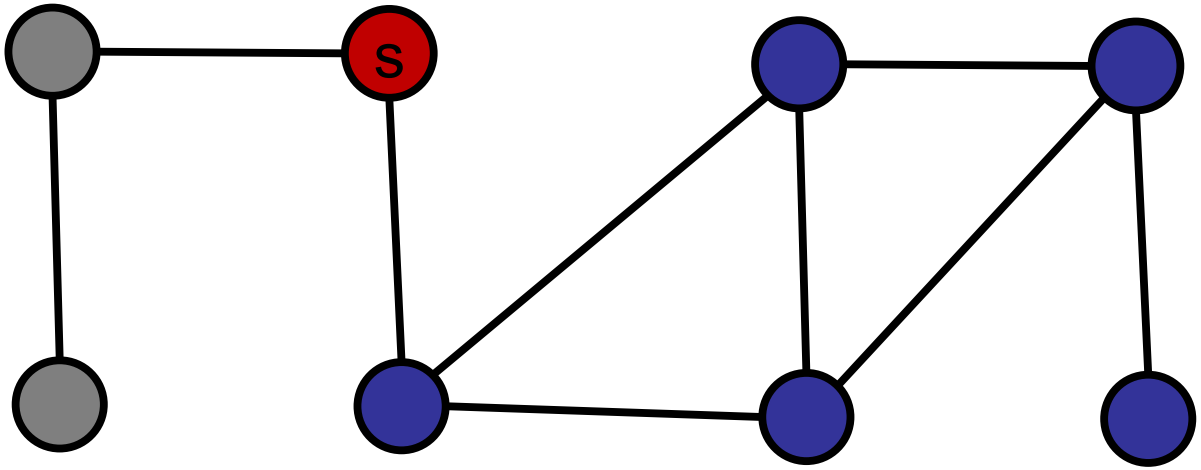
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



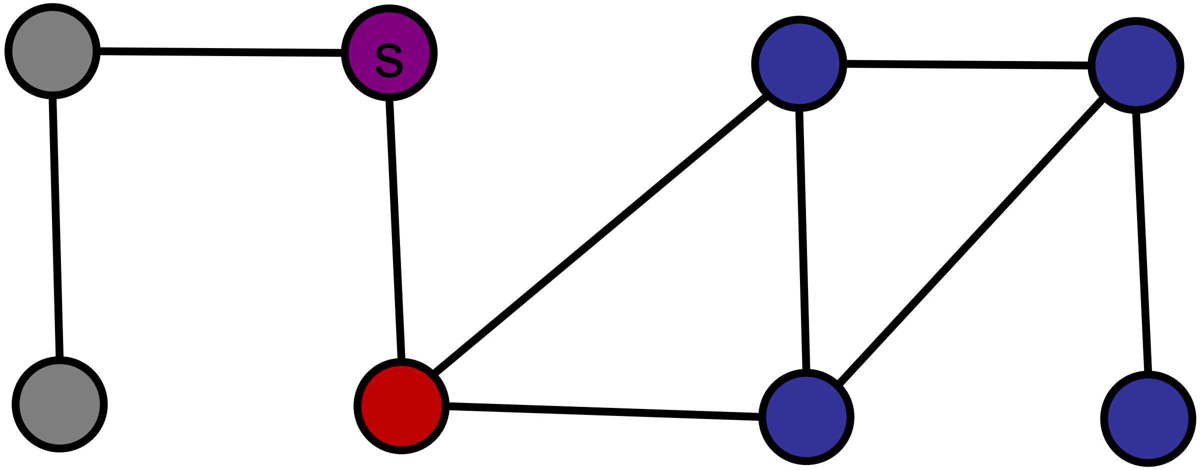
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



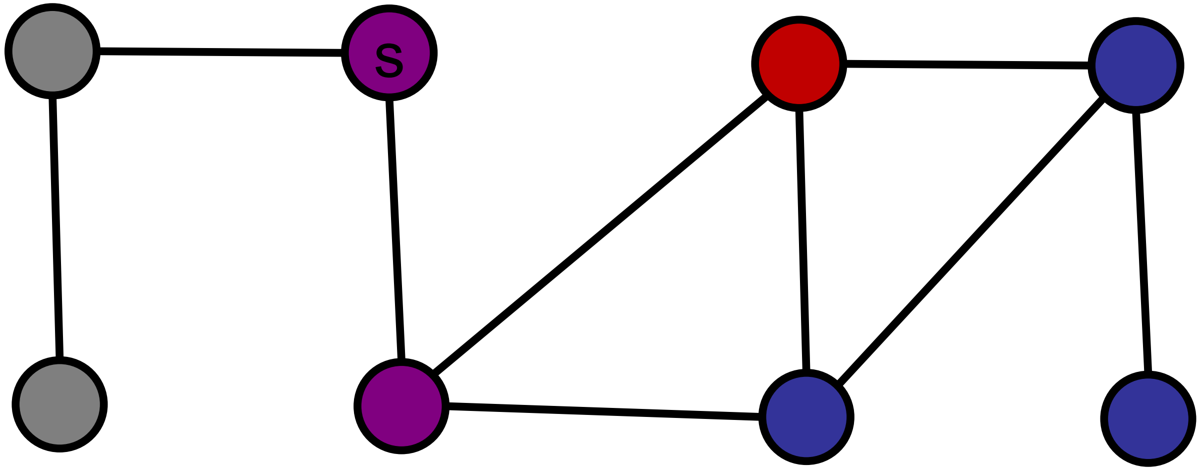
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



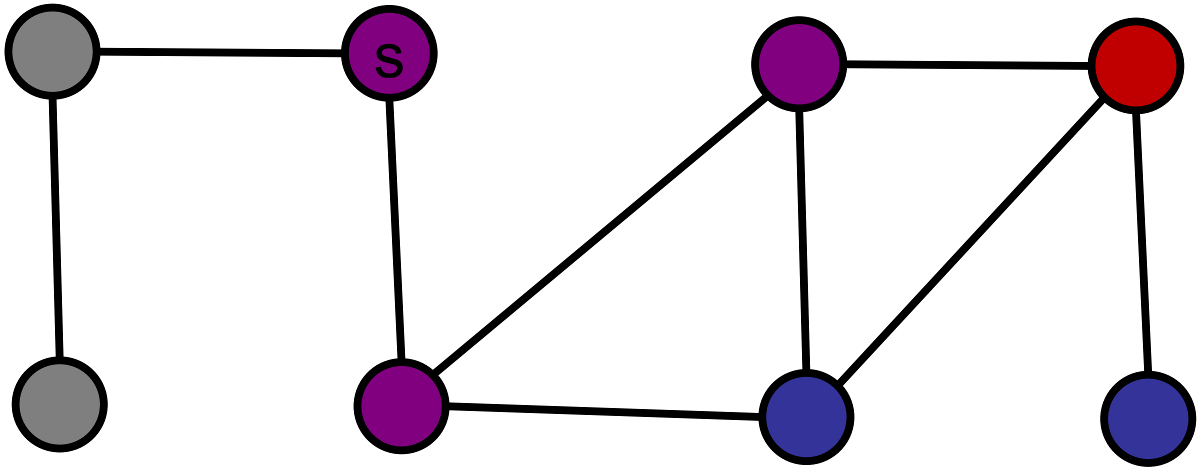
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



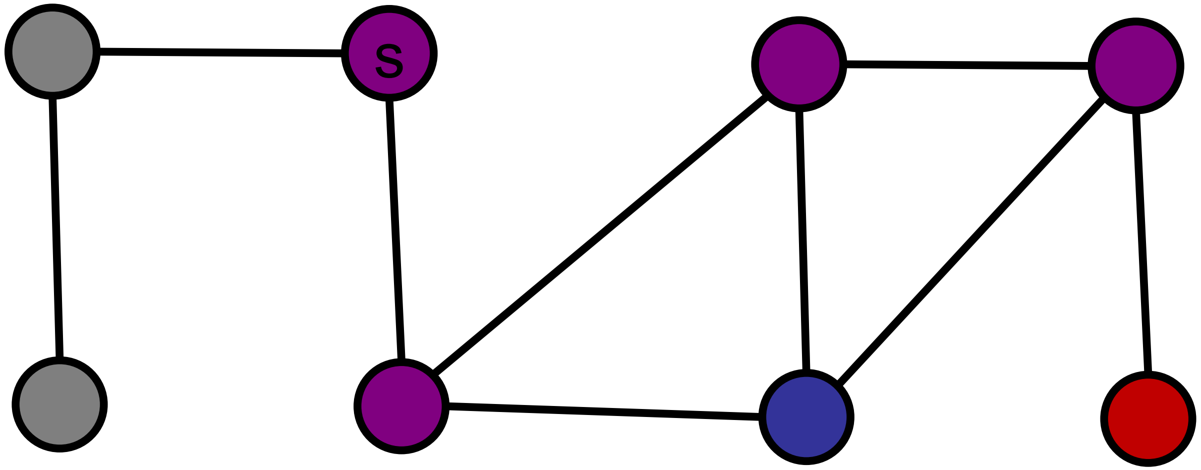
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



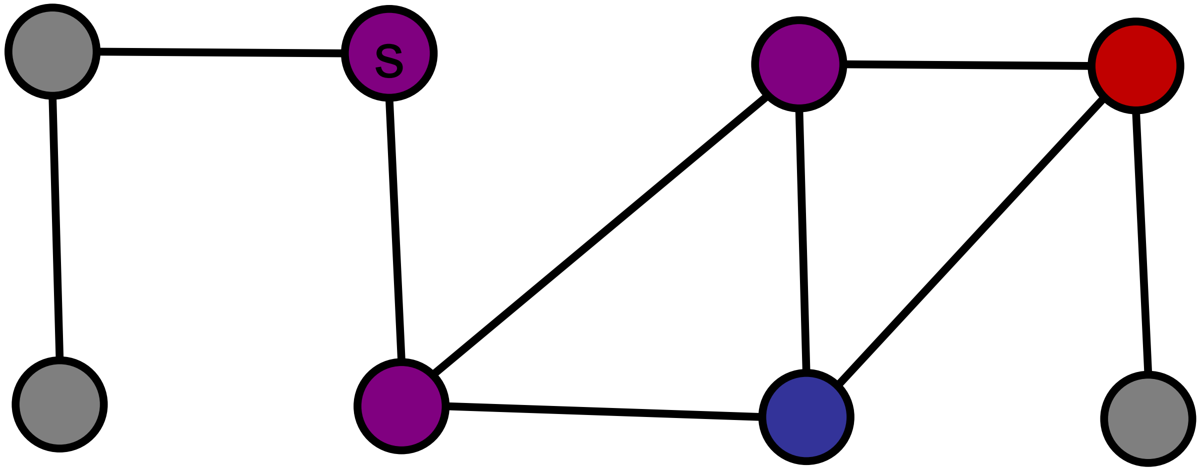
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



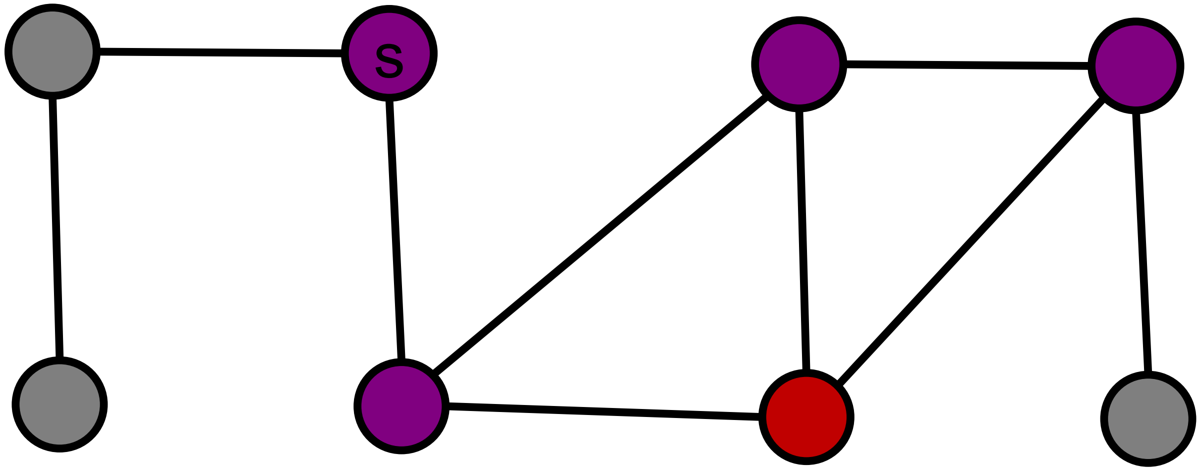
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



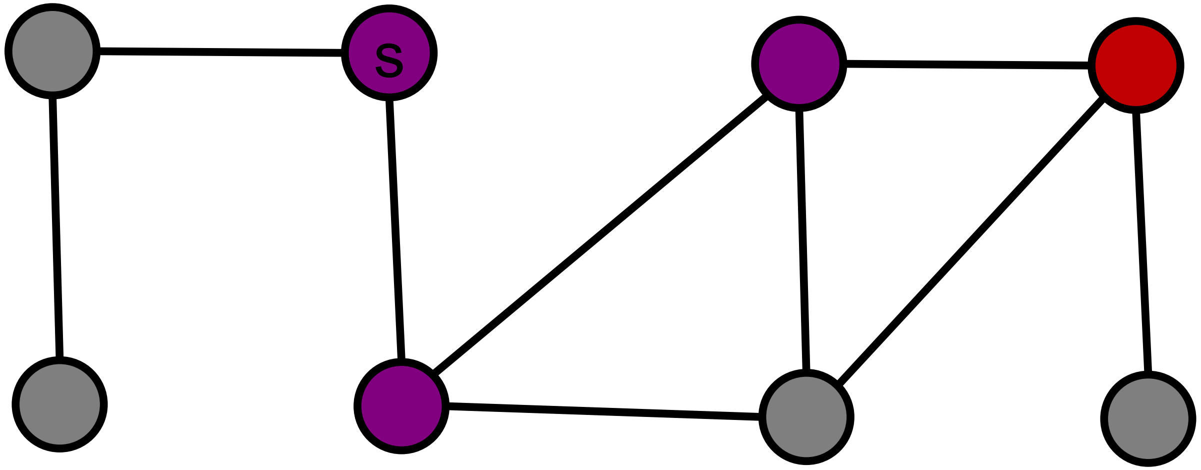
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



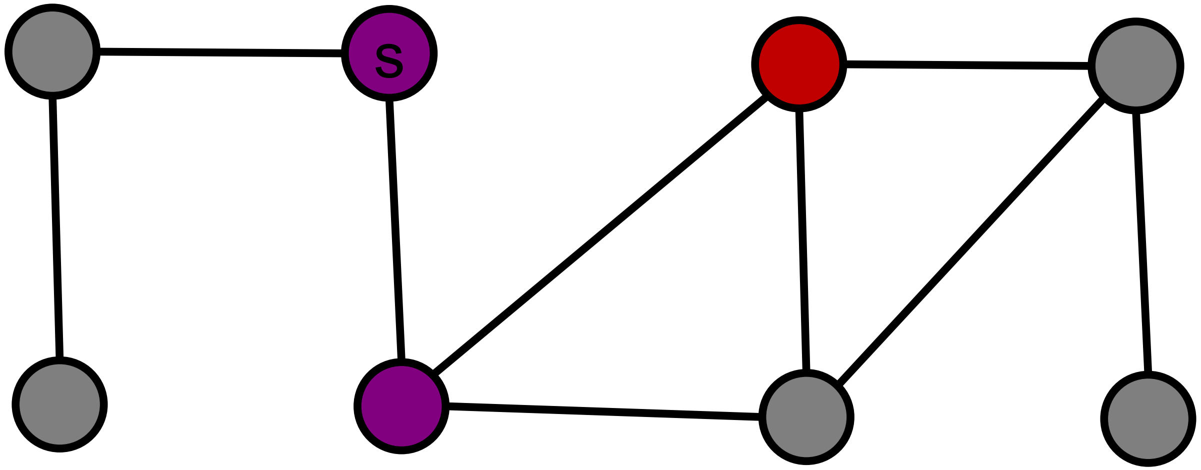
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



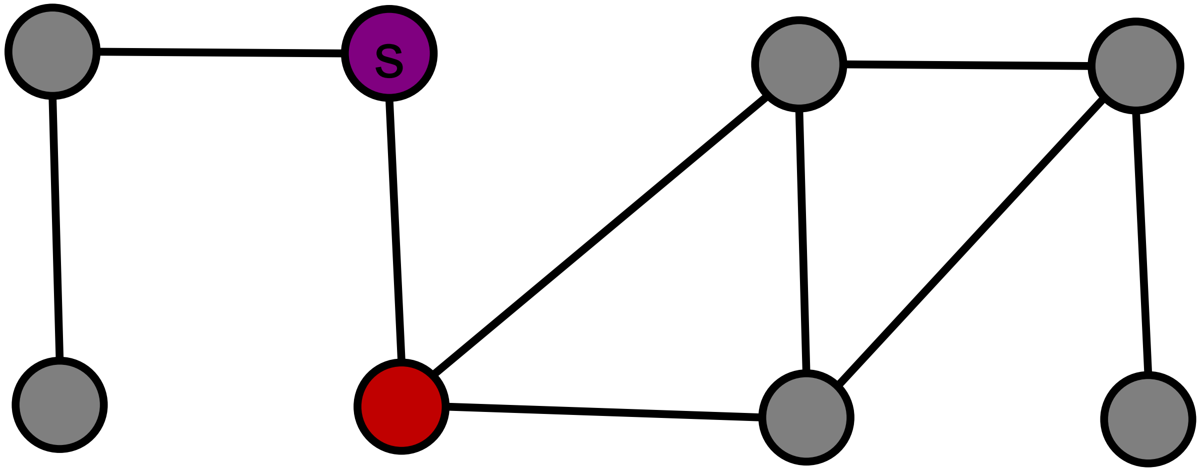
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



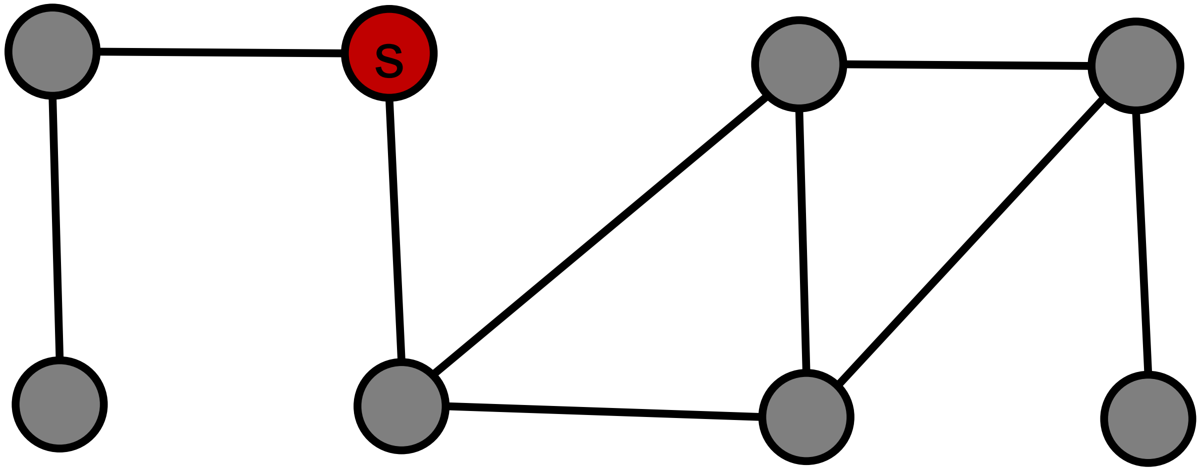
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



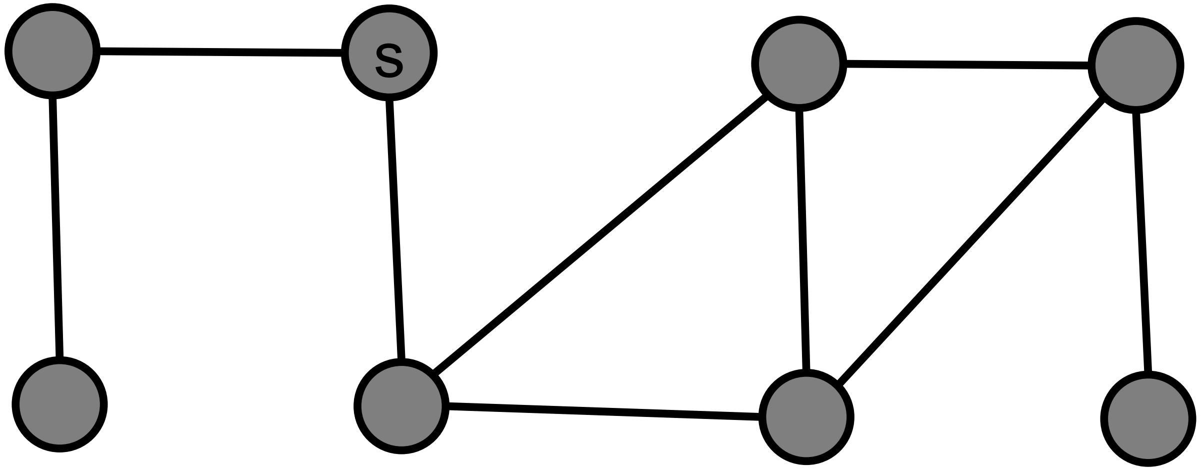
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



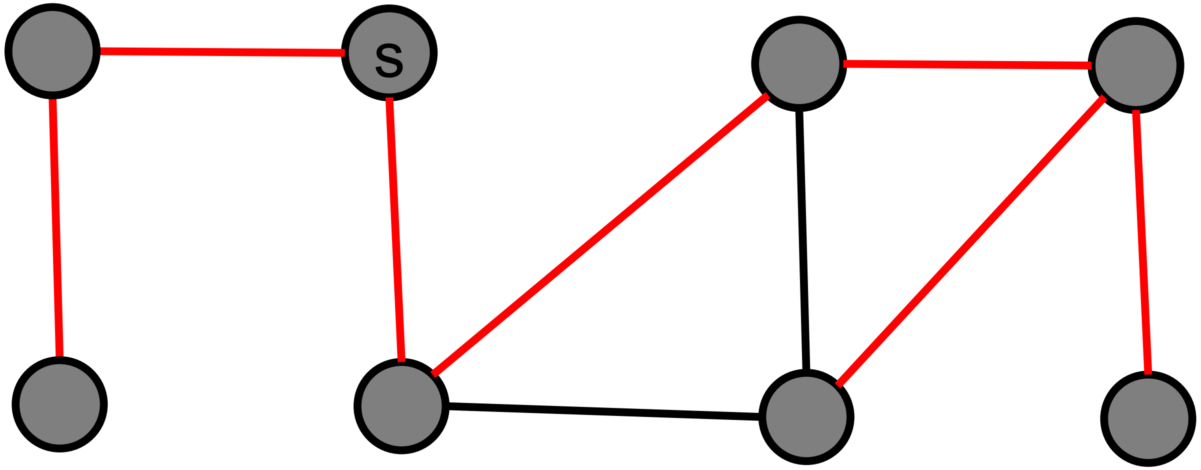
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

Depth-First Search Example



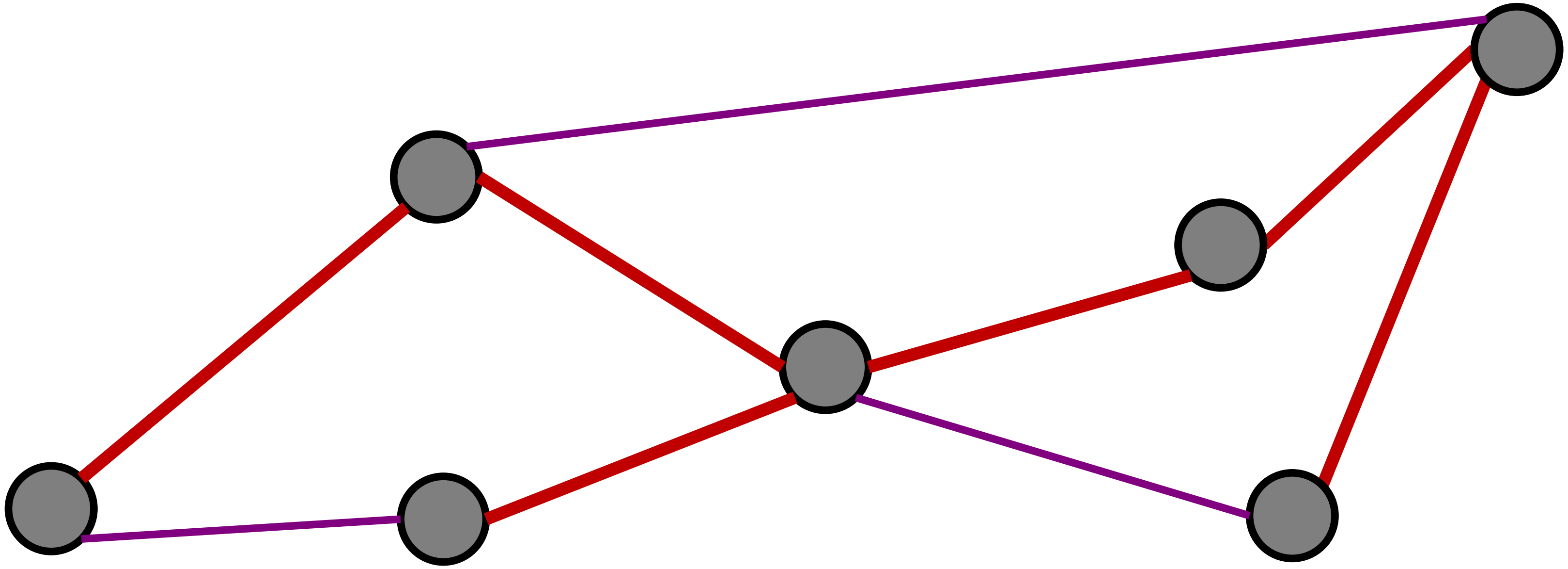
Red = active frontier

Purple = next

Gray = visited

Blue = unvisited

DFS parent edges



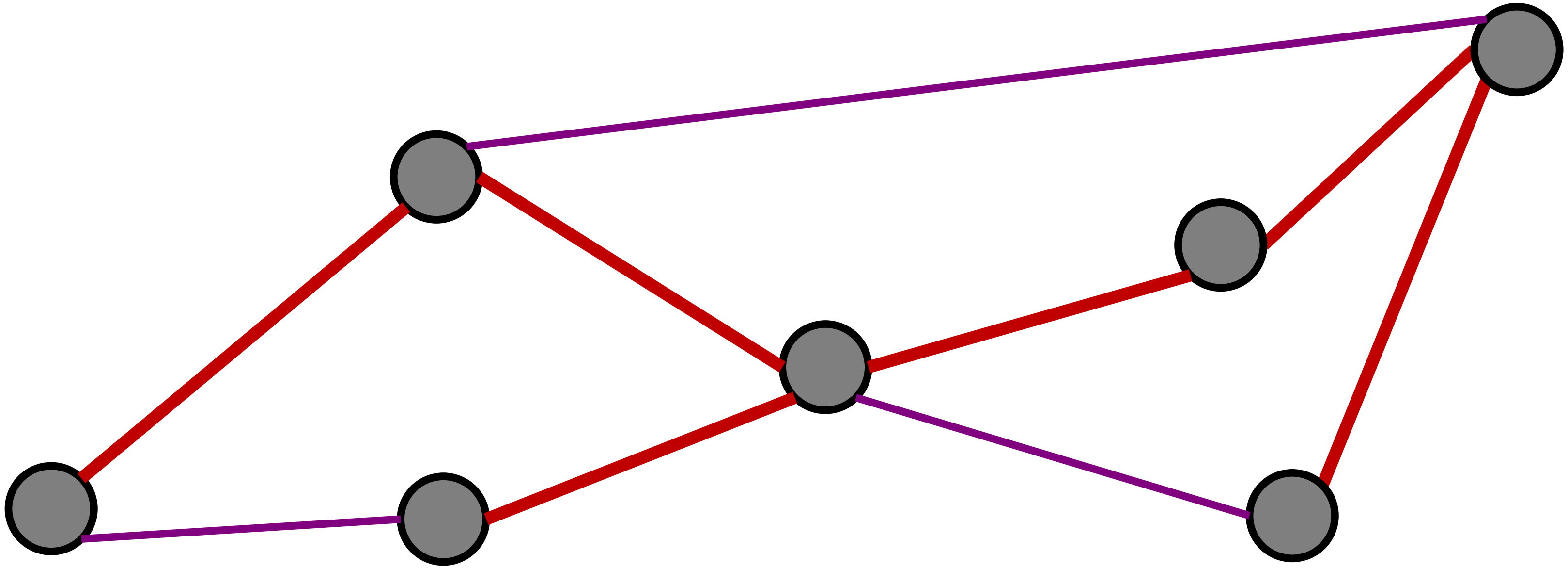
Red = Parent Edges

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.

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. (V^2)
6. I have no idea.

Depth-First Search

O(V)

Analysis:

- DFS-visit called only once per node.
    - 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

O(V)

Analysis:

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

- 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 bag.
4. Use a set.
5. 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.

Roadmap

Today: Graph Basics

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