CS2040S
Data Structures and Algorithms

Graphs!

Roadmap

Today: Graph Basics

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

Roadmap

Next: Searching Graphs

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

Roadmap

Next next:

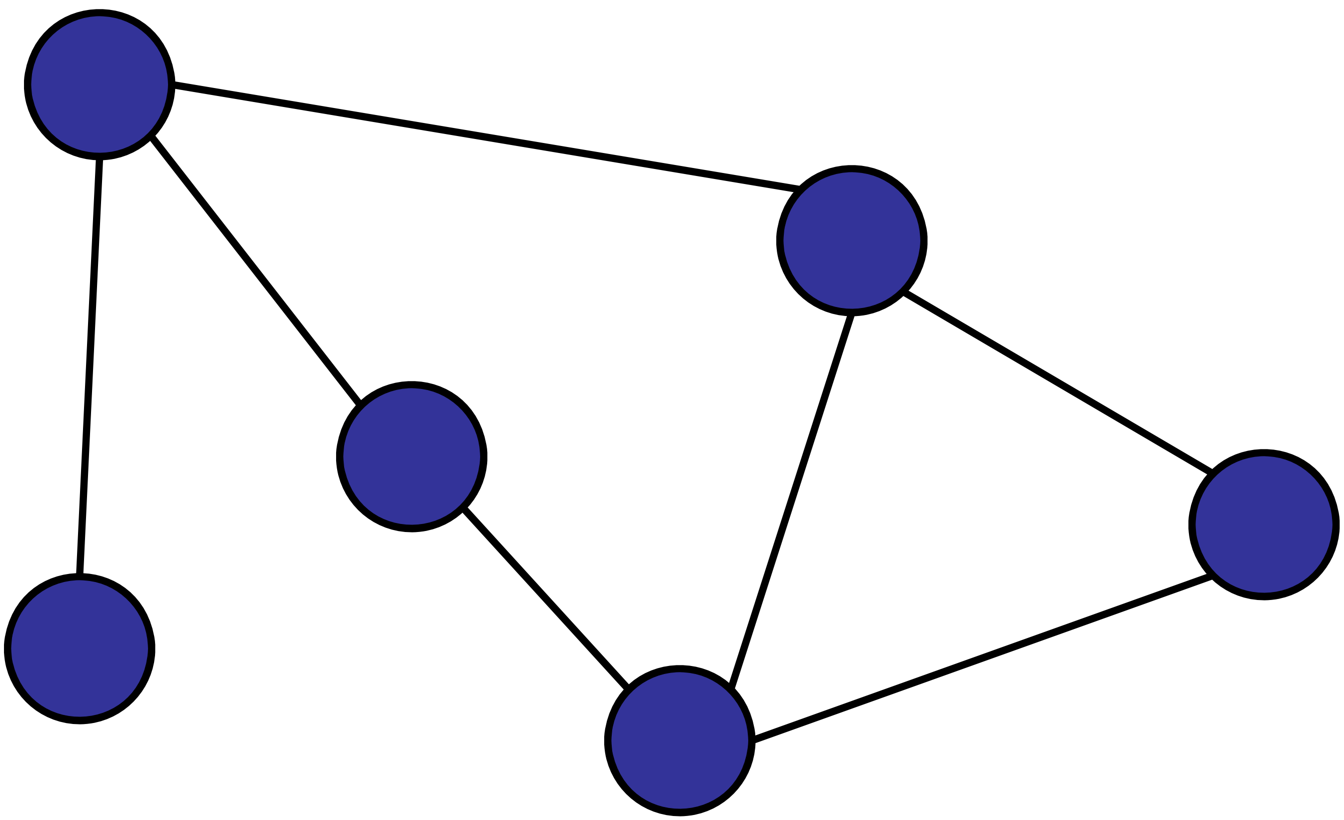
- Connected component problem
  - Union-Find data structure
- The Minimum Spanning Tree Problem
  - Kruskal’s Algorithm
  - Prim’s Algorithm

What is a graph?

Is it a graph?

1. Yes
2. No.

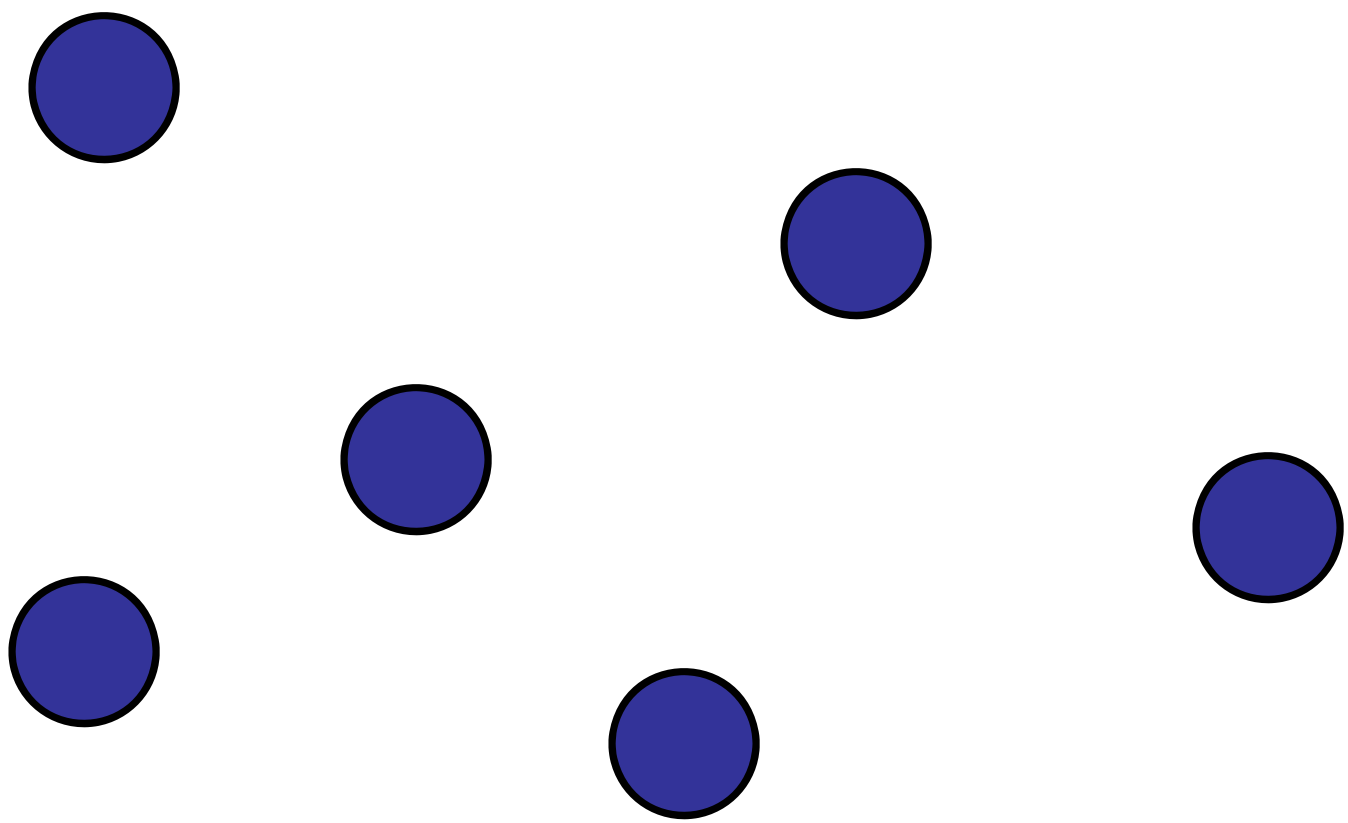
Yes or No on Zoom.



Is it a graph?

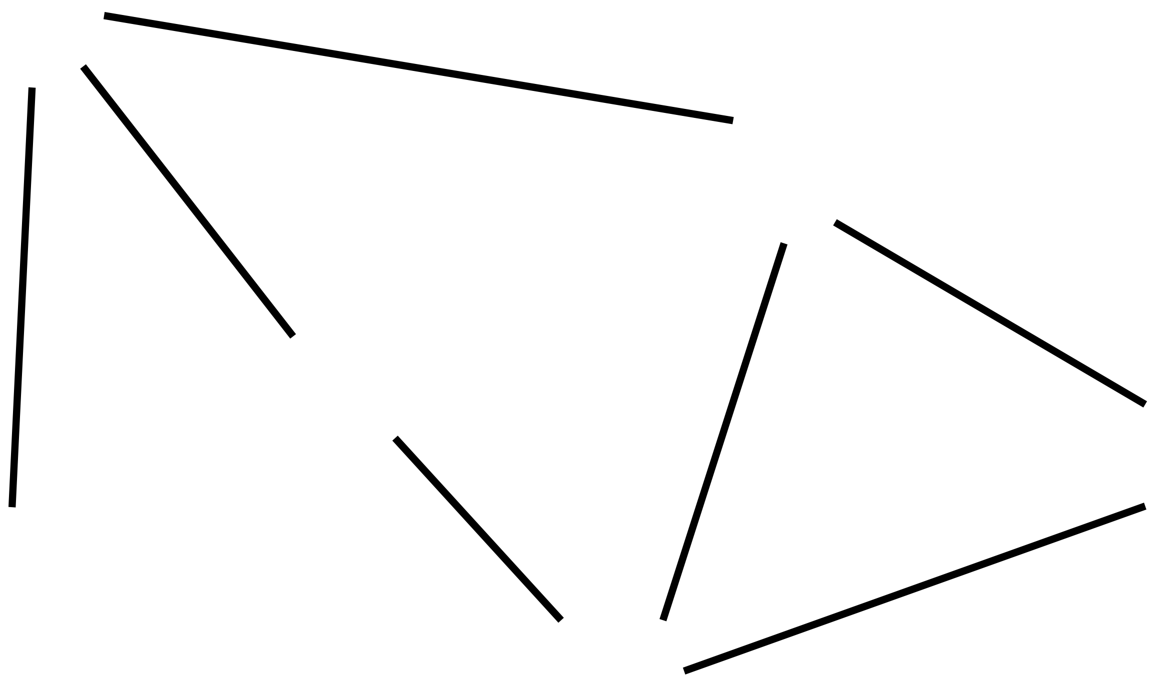
✓ 1. Yes
2. No.

Yes or No on Zoom.



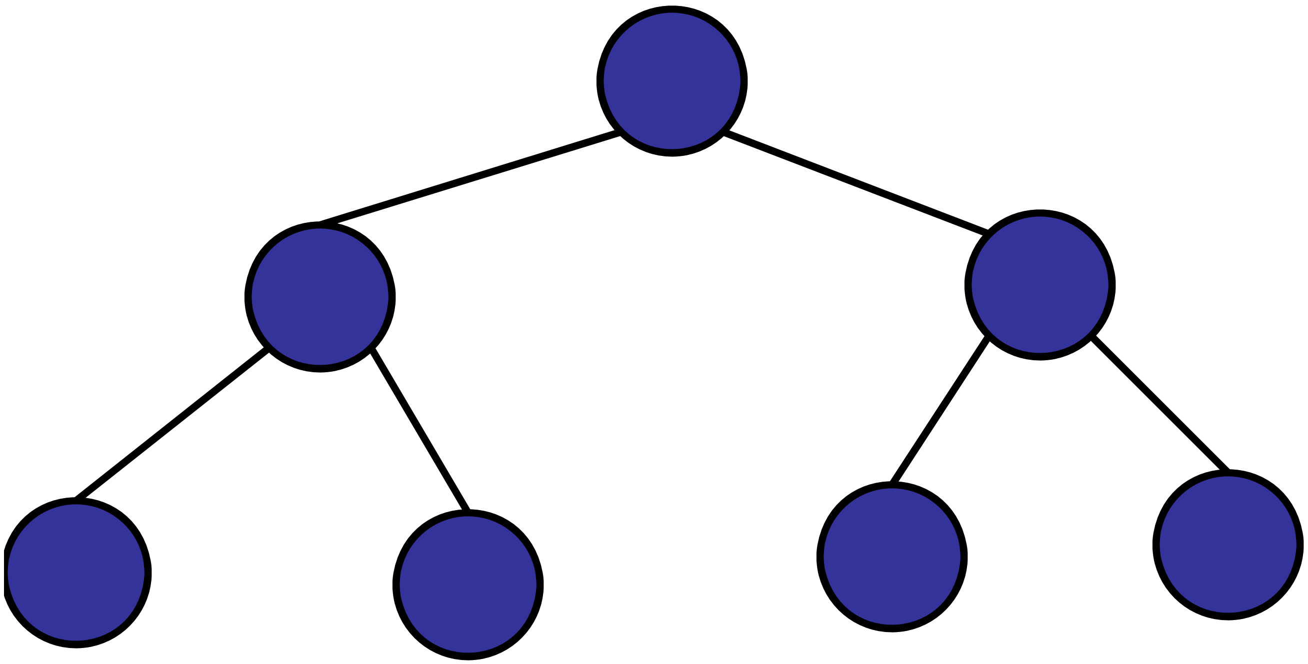
Is it a graph?

1. Yes
2. No.



Is it a graph?

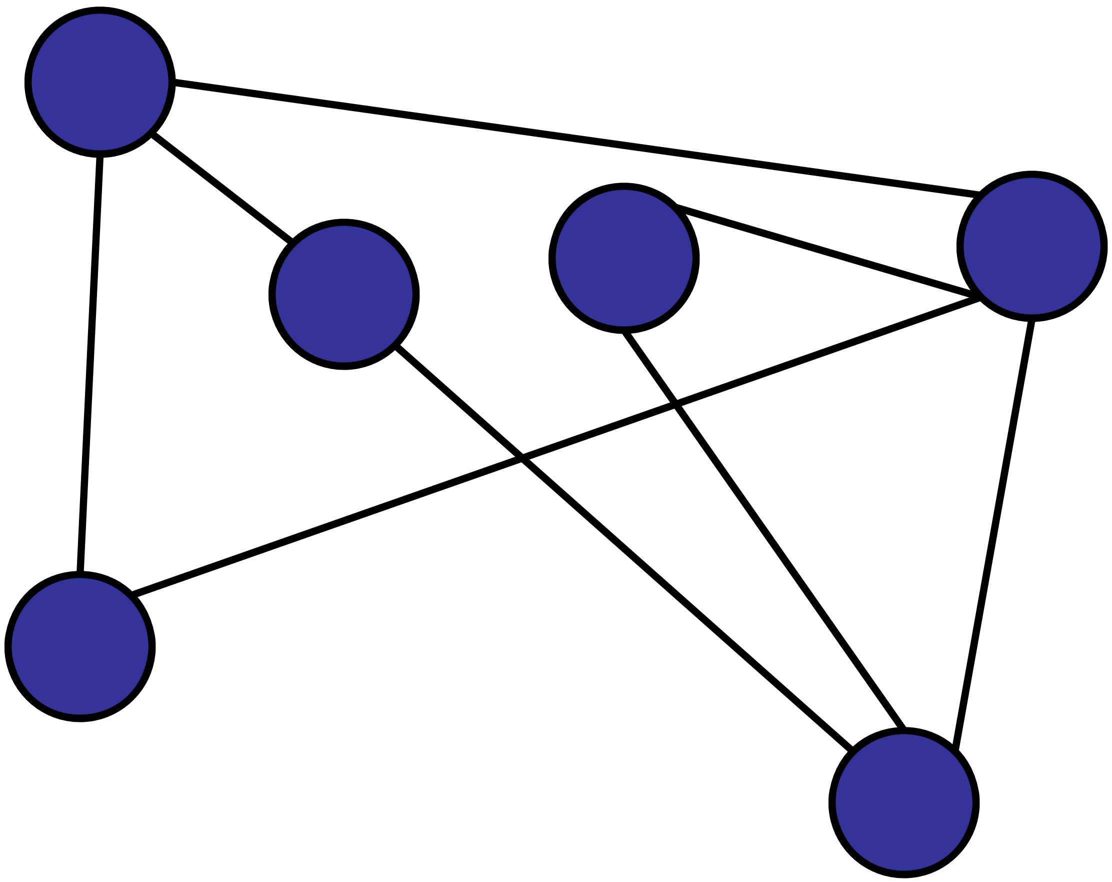
1. Yes
2. No.



Is it a graph?

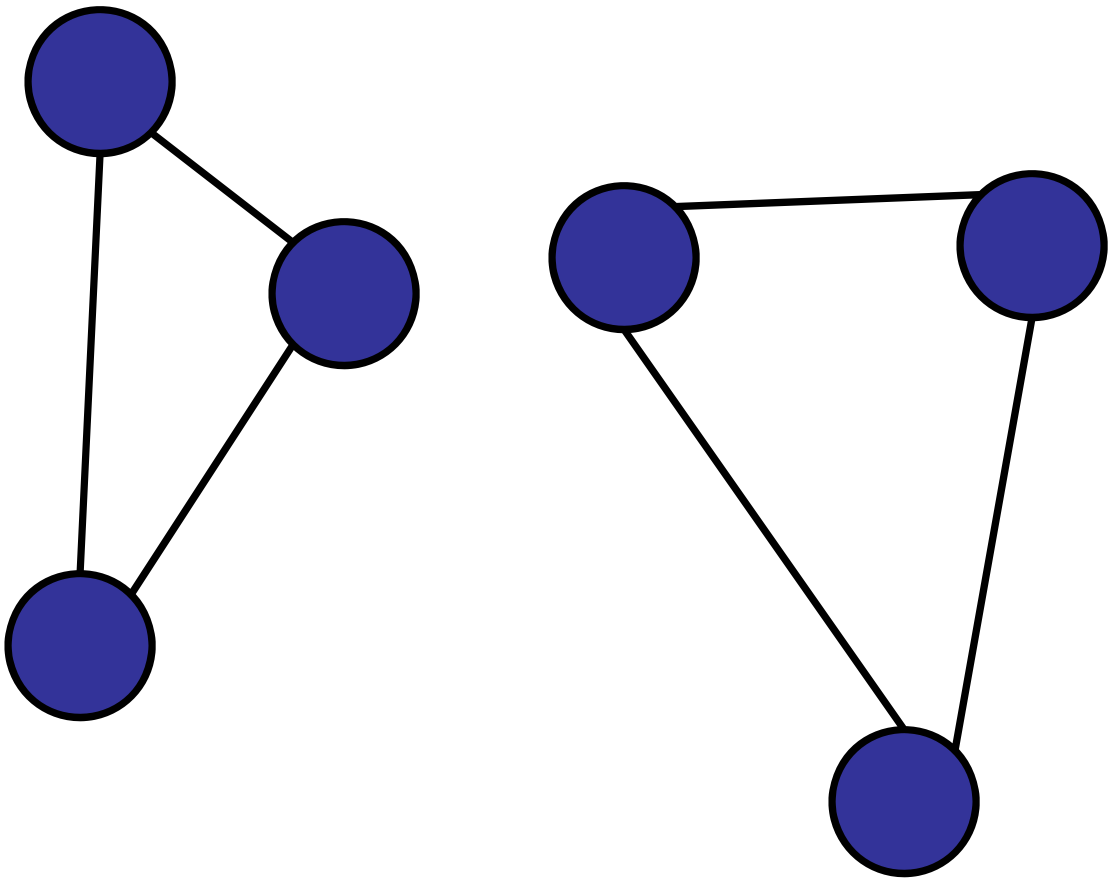
1. Yes
2. No.

Yes or No on Zoom.



Is it a graph?

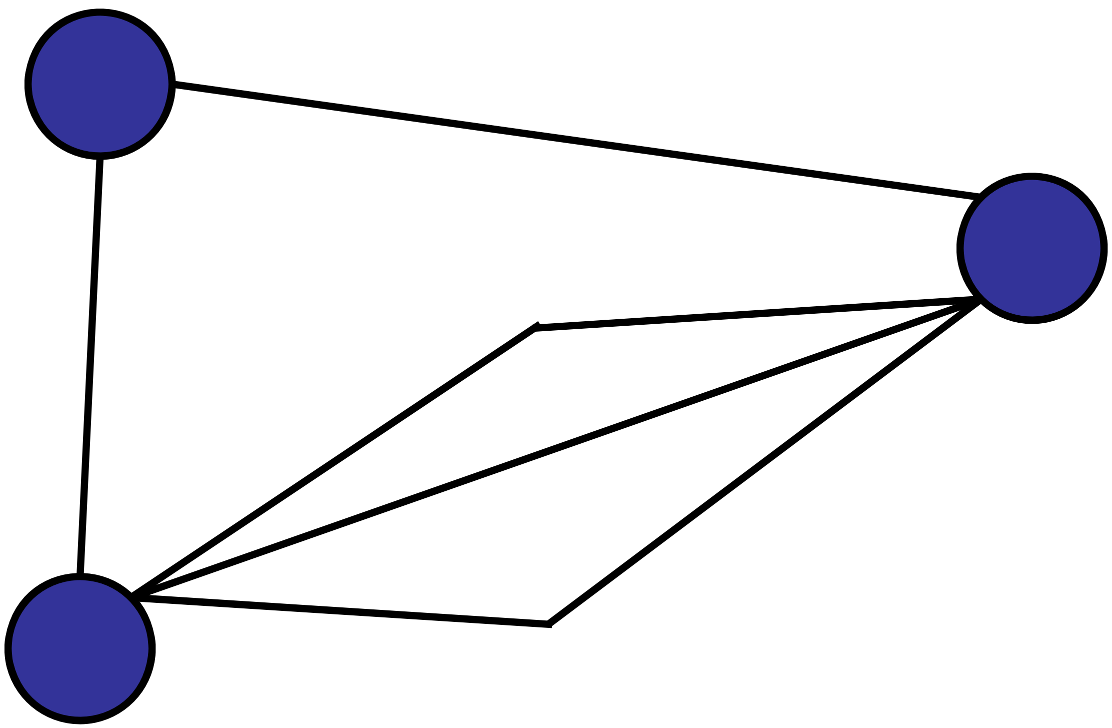
1. Yes
2. No.



Is it a graph?

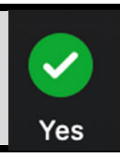
1. Yes
2. No.

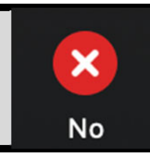
Yes or No on Zoom.



Is it a graph?

1. Yes
2. No.



or


on Zoom.

What is a 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.

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 common in CS2040S)

What is a multigraph?

Graph consists of two types of elements:

- Nodes (or vertices)
  - At least one.

- Edges (or arcs)
  - Each edge connects two nodes in the graph
  - Two nodes may be connected by more than one edge.

(Rare in CS2040S.)

What is a graph?

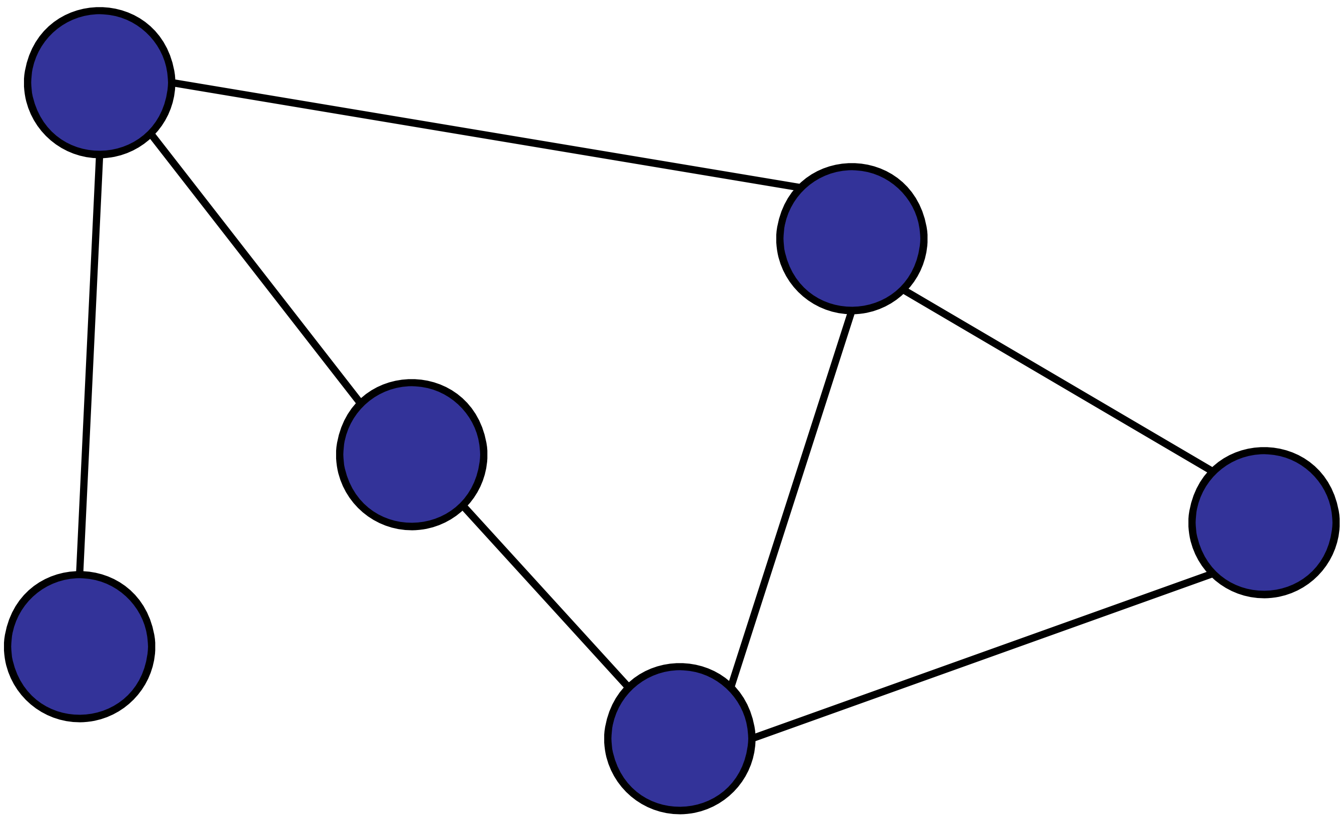
Graph $G = \langle V, E \rangle$

- 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 $e_1, e_2 \in E : e_1 \neq e_2$

Graph Terminology

Connected:

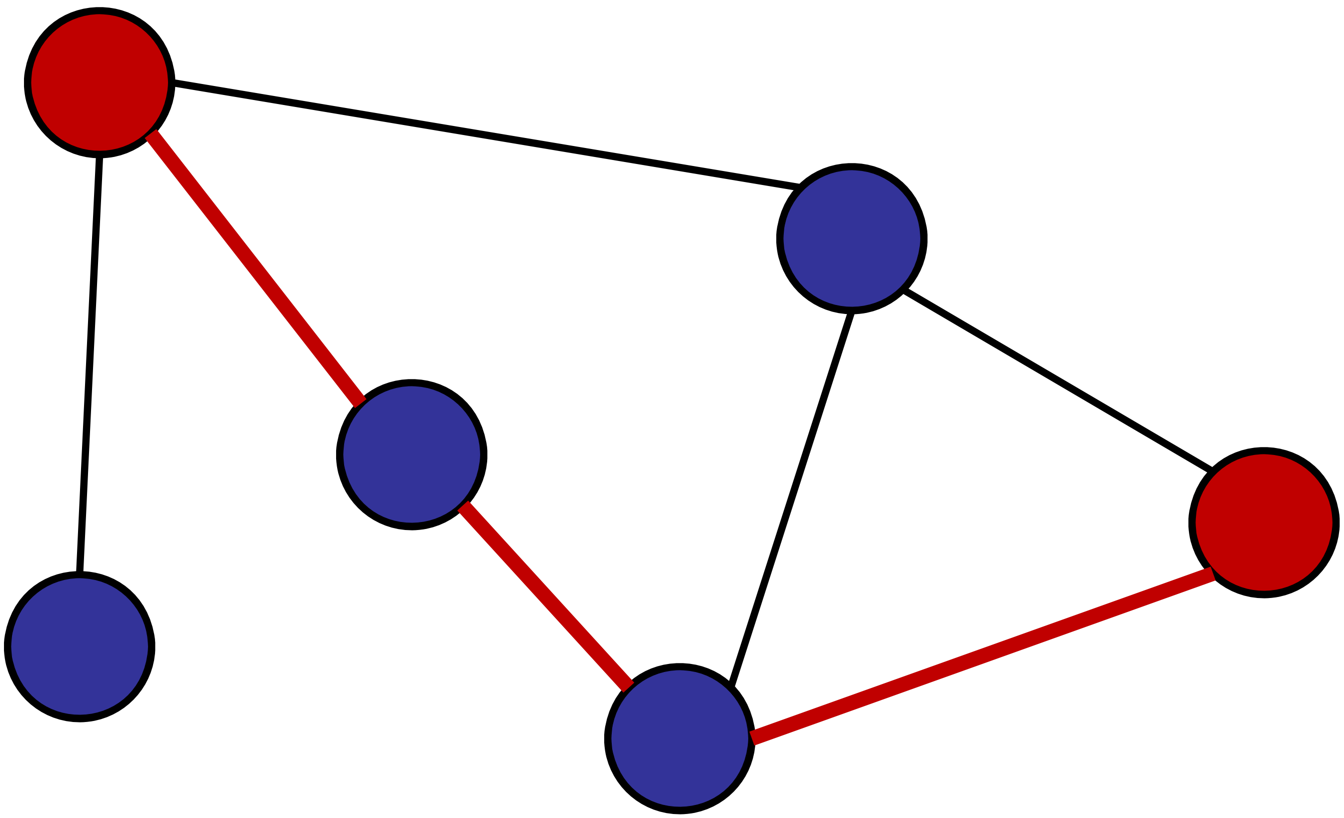
– Every pair of nodes is connected by a path.



Graph Terminology

Connected:

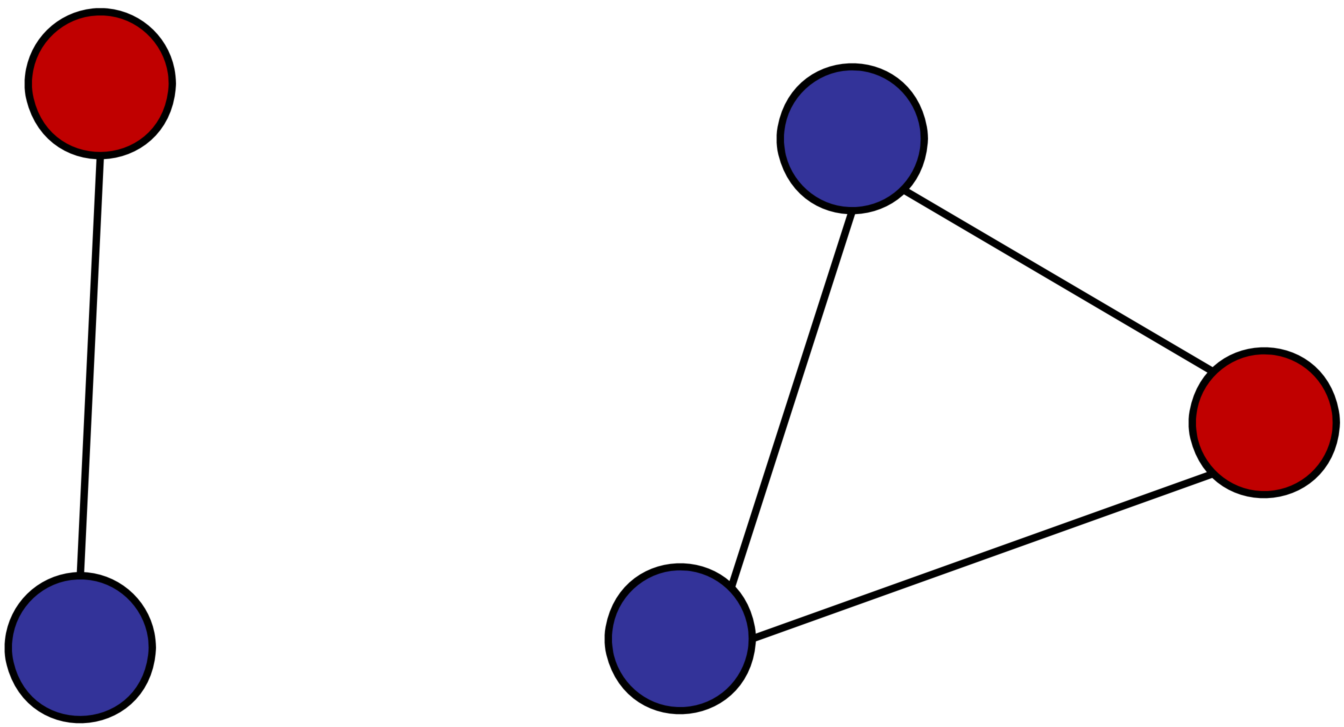
- Every pair of nodes is connected by a path.



Graph Terminology

Disconnected:

– Some pair of nodes is not connected by a path.

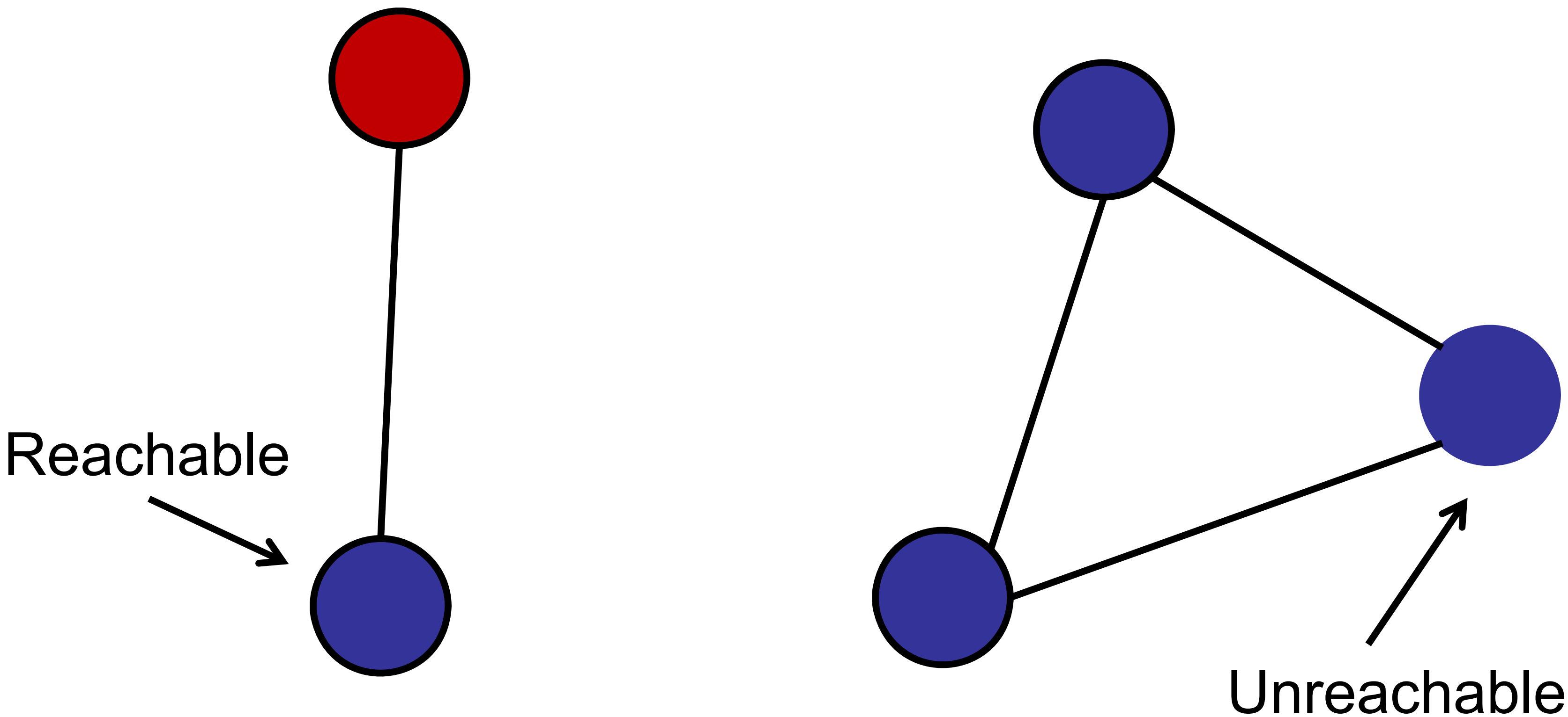


– Two **connected components**.

Graph Terminology

Disconnected:

- Some pair of nodes is not connected by a path.

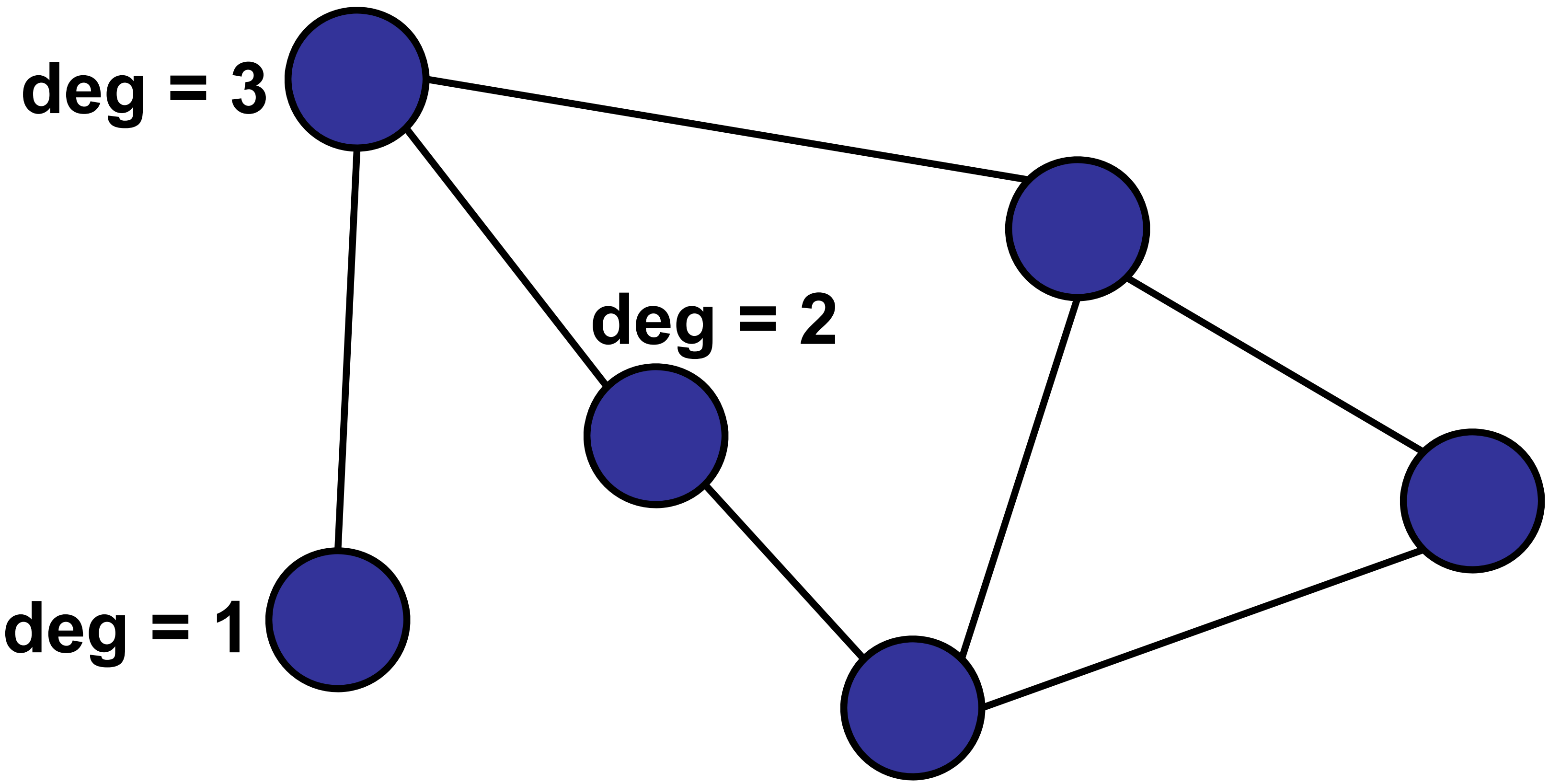


- Two **connected components**.

Graph Terminology

Degree of a node:

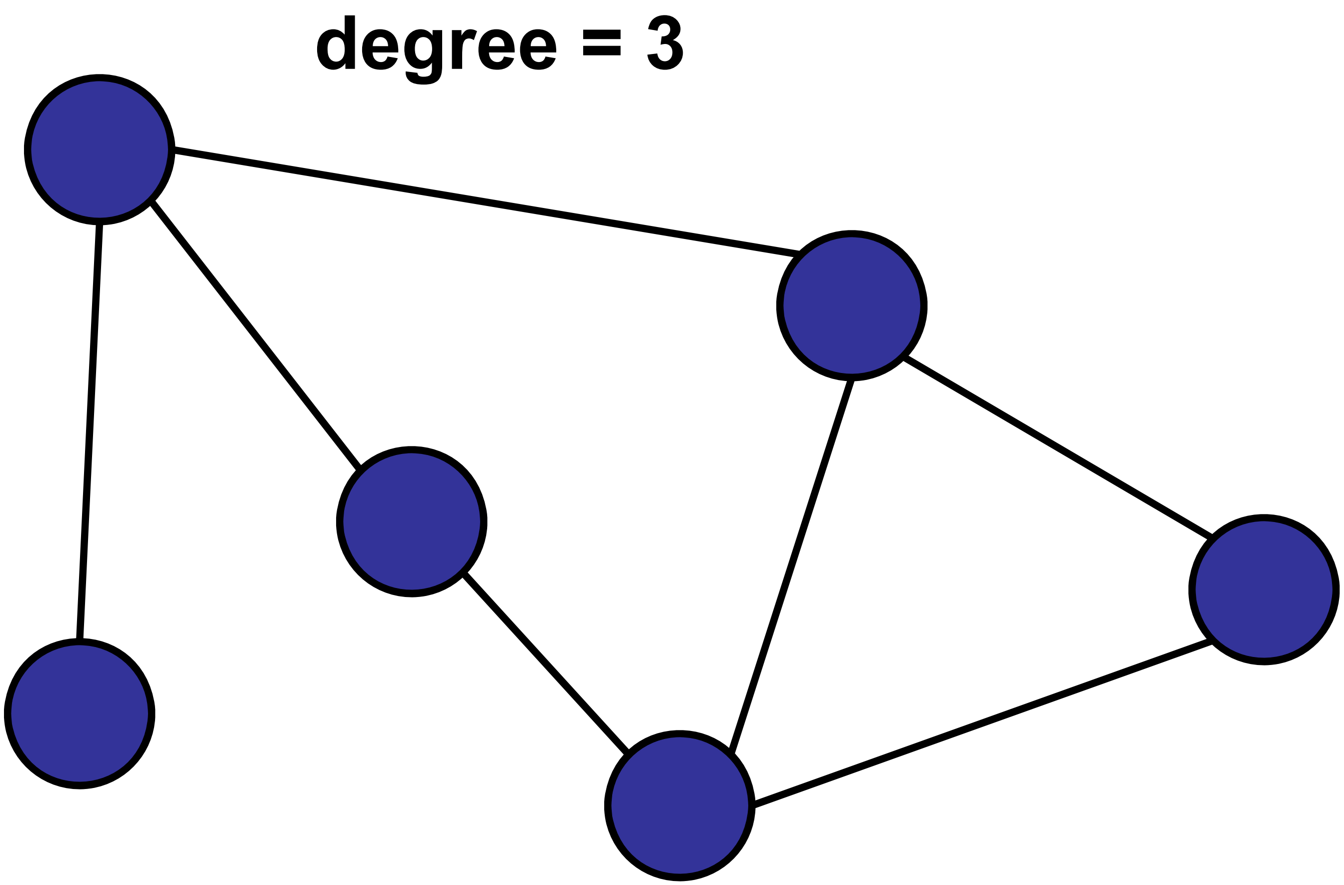
– Number of **adjacent** edges.



Graph Terminology

Degree of a graph:

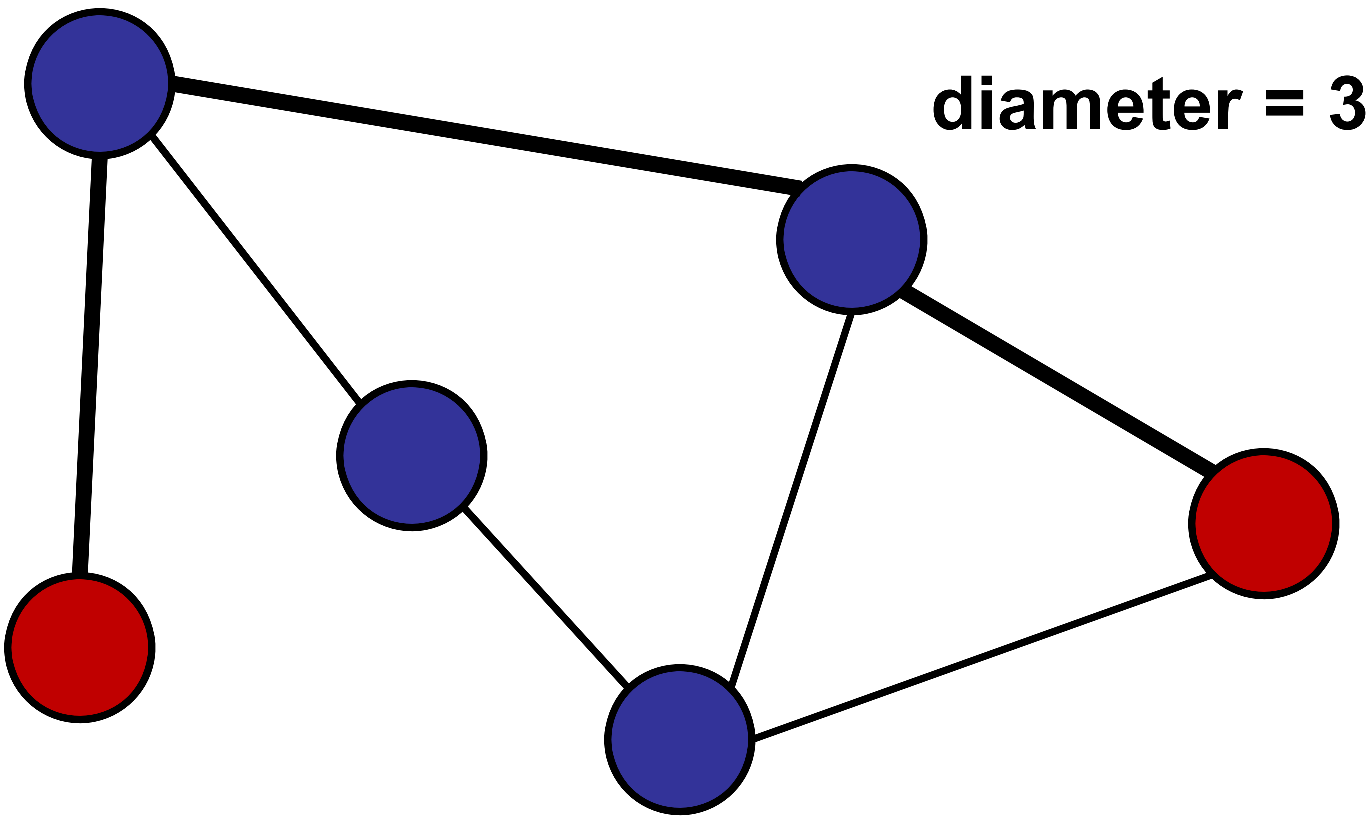
– Maximum number of **adjacent** edges.



Graph Terminology

Diameter:

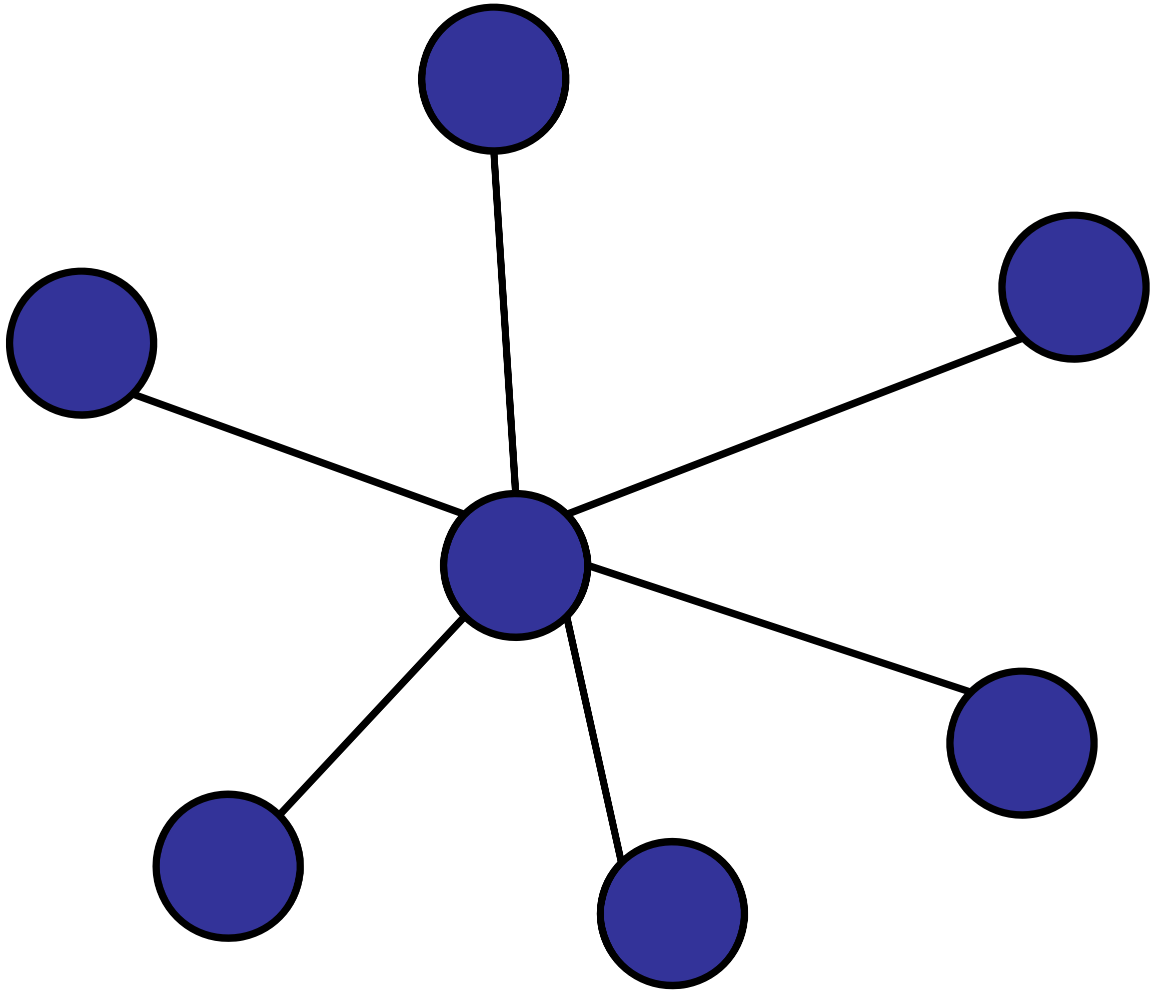
– Maximum distance between two nodes, following the shortest path.



Special Graphs

Special Graphs

Star



One central node, all edges connect center to edges.

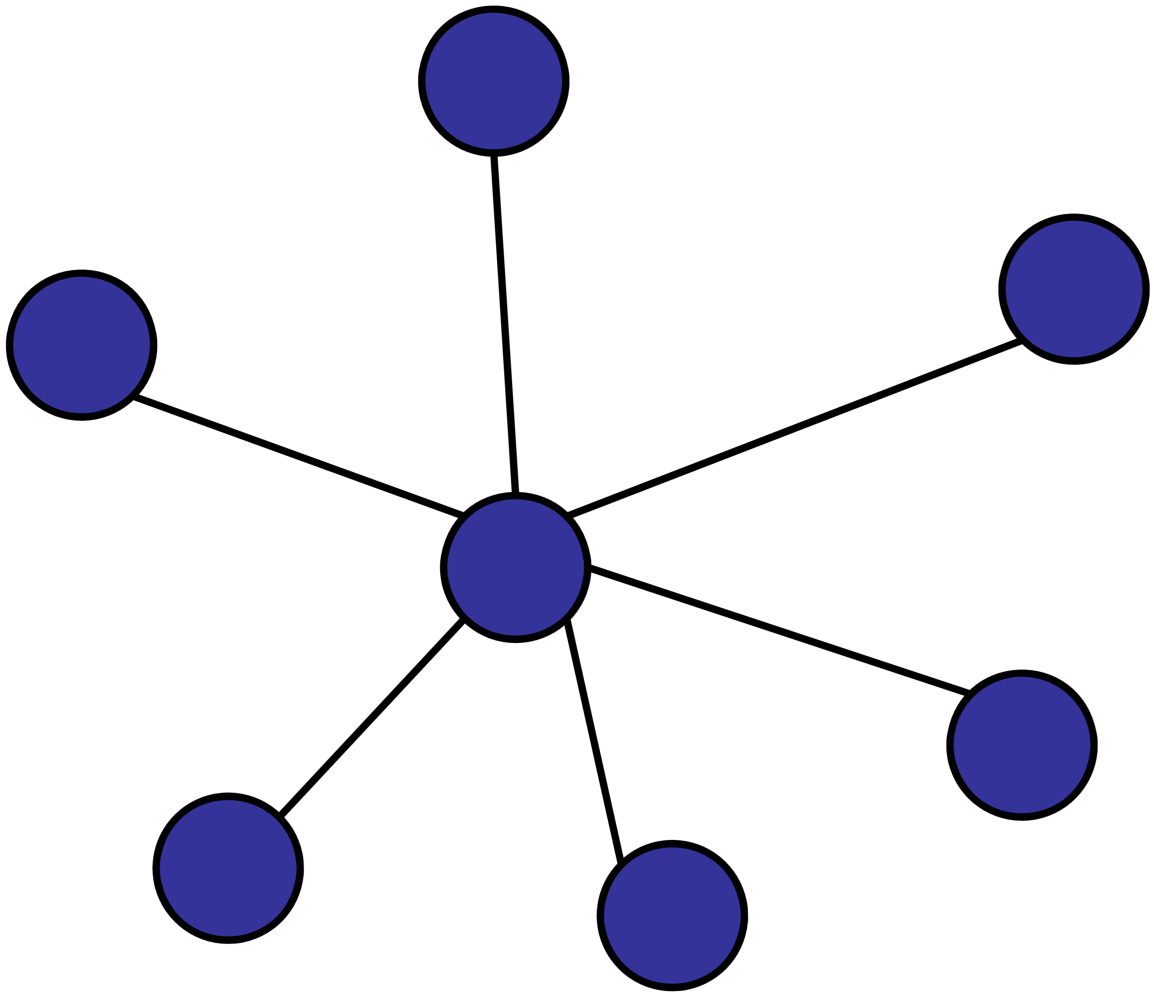
Degree of n-node star is:

1. 1
2. 2
3. n/2
4. n-2
5. n-1
6. n

ARCHIPELAGO is open

Special Graphs

Star



One central node, all edges connect center to edges.

Diameter of n-node star:

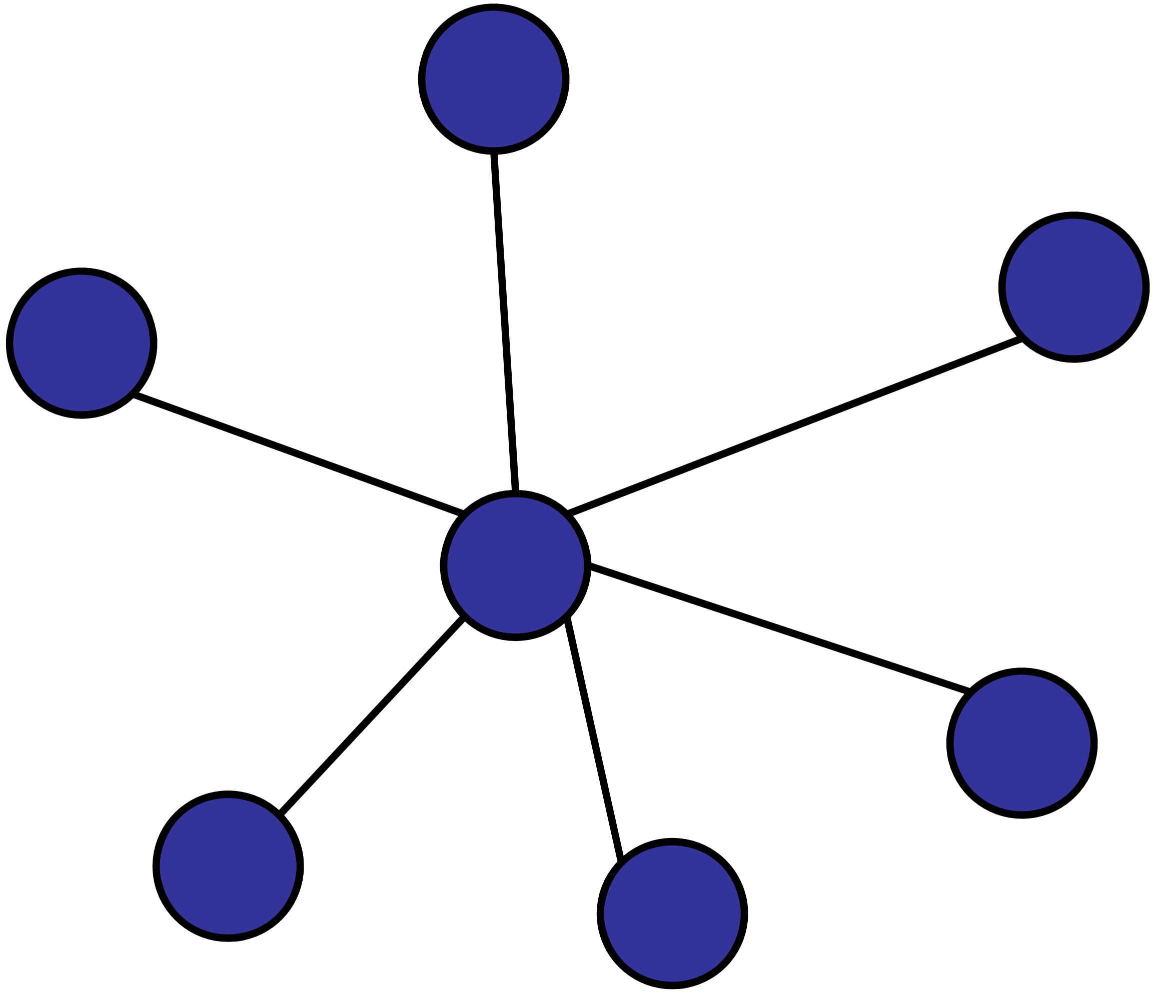
1. 1
2. 2
3. n/2
4. n-2
5. n-1
6. n

ARCHIPELAGO

is open

Special Graphs

Star

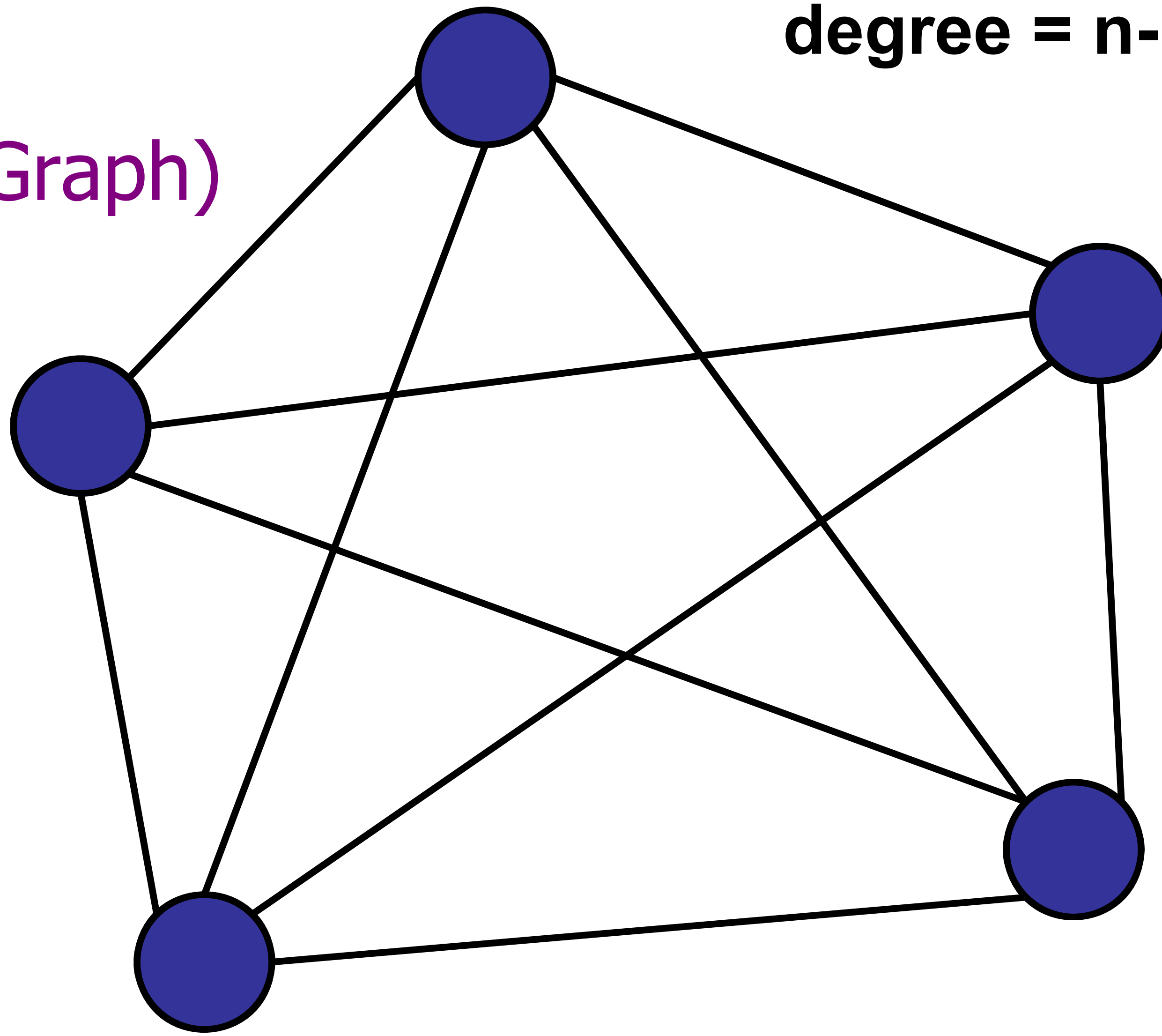


One central node, all edges connect center to edges.

Special Graphs

Clique

(Complete Graph)



diameter = 1

degree = n-1

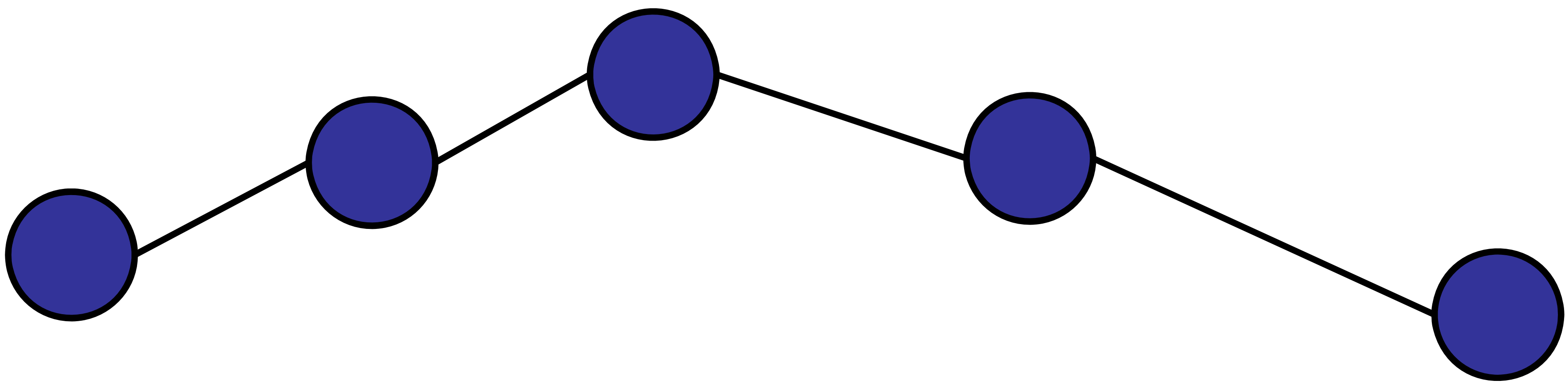
All pairs connected by edges.

Special Graphs

diameter = n-1

degree = 2

Line (or path)



Special Graphs

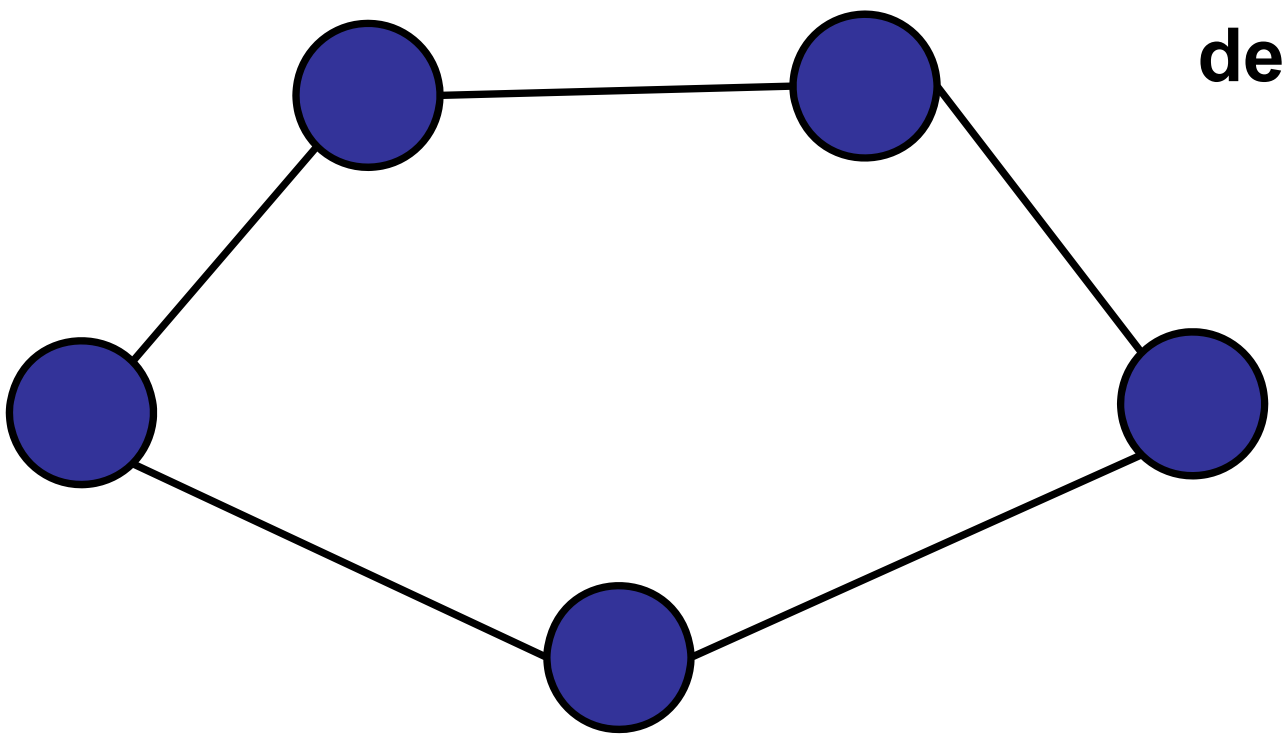
Cycle

diameter = n/2

or

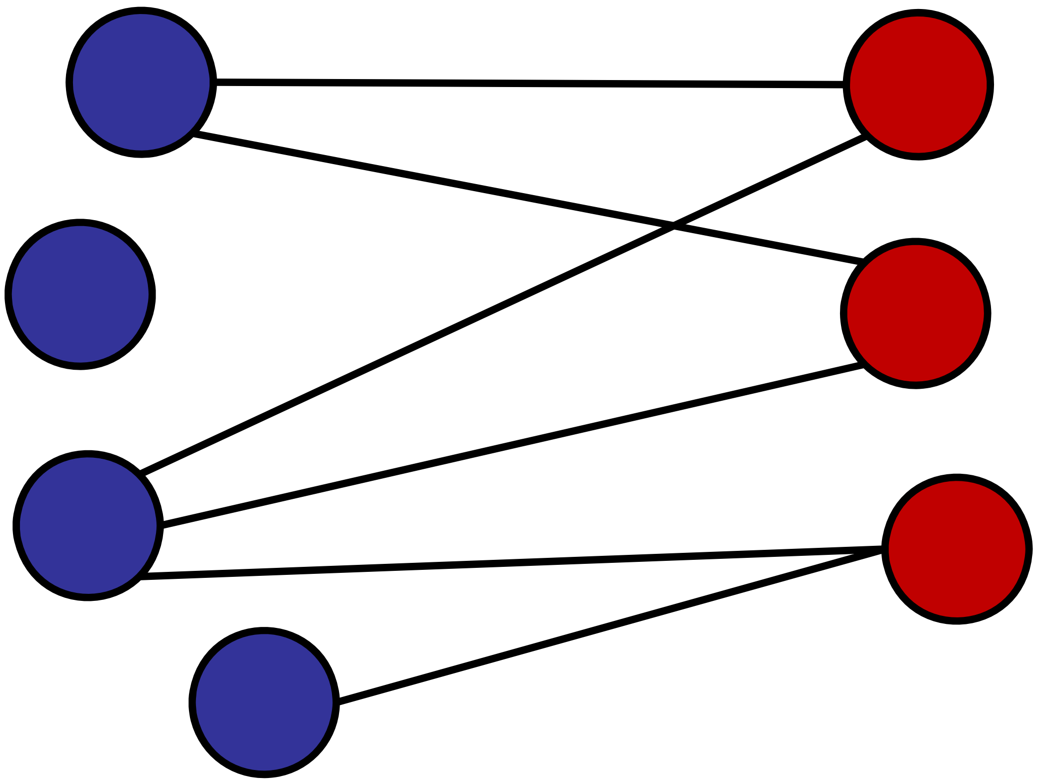
diameter = n/2-1

degree = 2



Special Graphs

Bipartite Graph



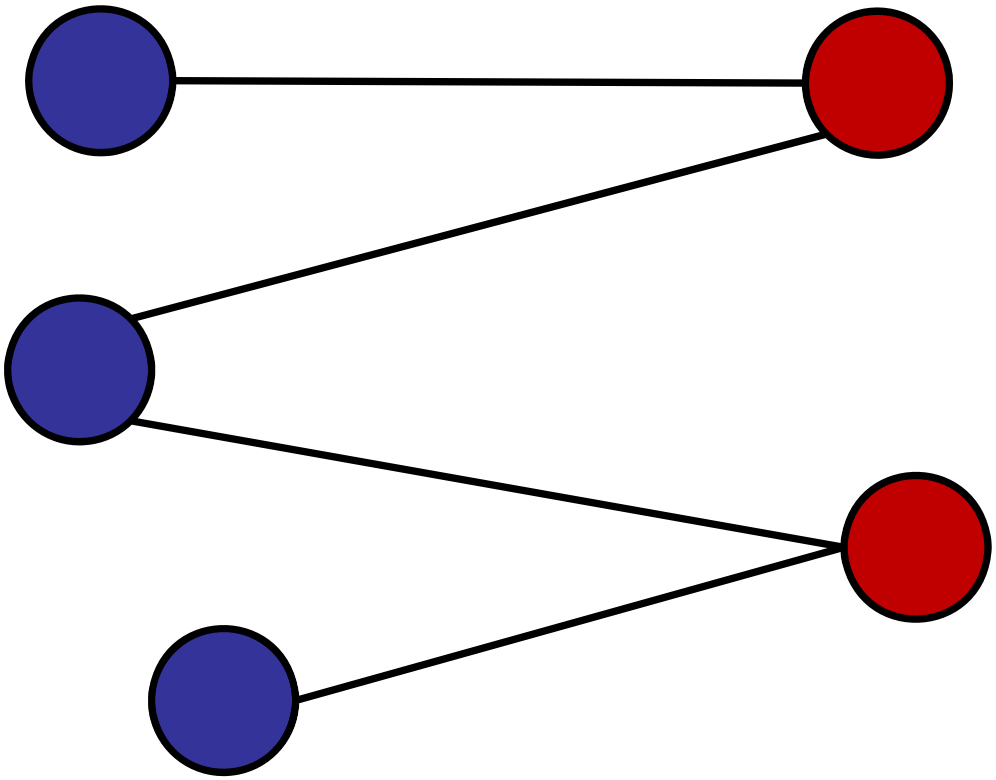
Nodes divided into two sets with no edges between nodes in the same set.

Max. diameter of n-node bipartite graph is:

1. 1
2. 2
3. n/2-1
4. n/2
5. n-1
6. n

Special Graphs

Bipartite Graph

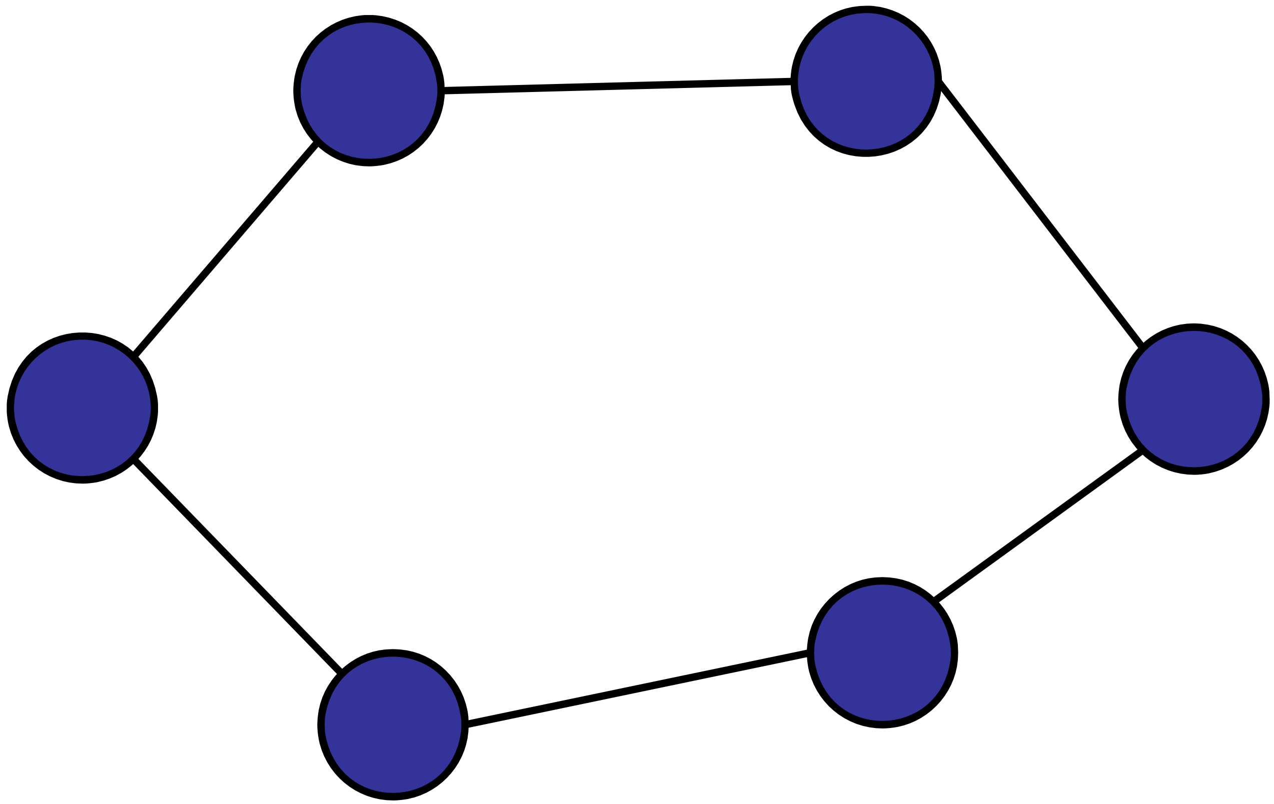


Nodes divided into two sets with no edges between nodes in the same set.

Is it bipartite?

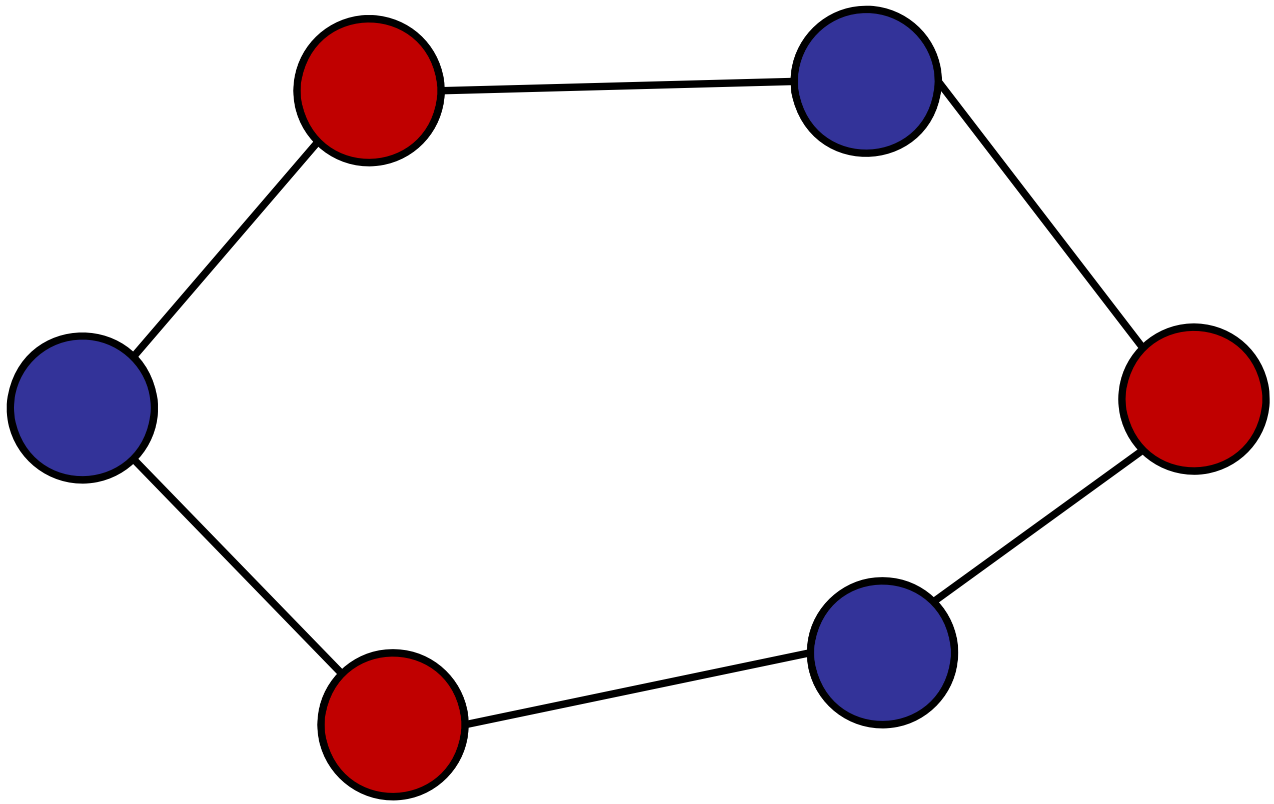
Yes or No on Zoom.

1. Yes
2. No



Is it bipartite?

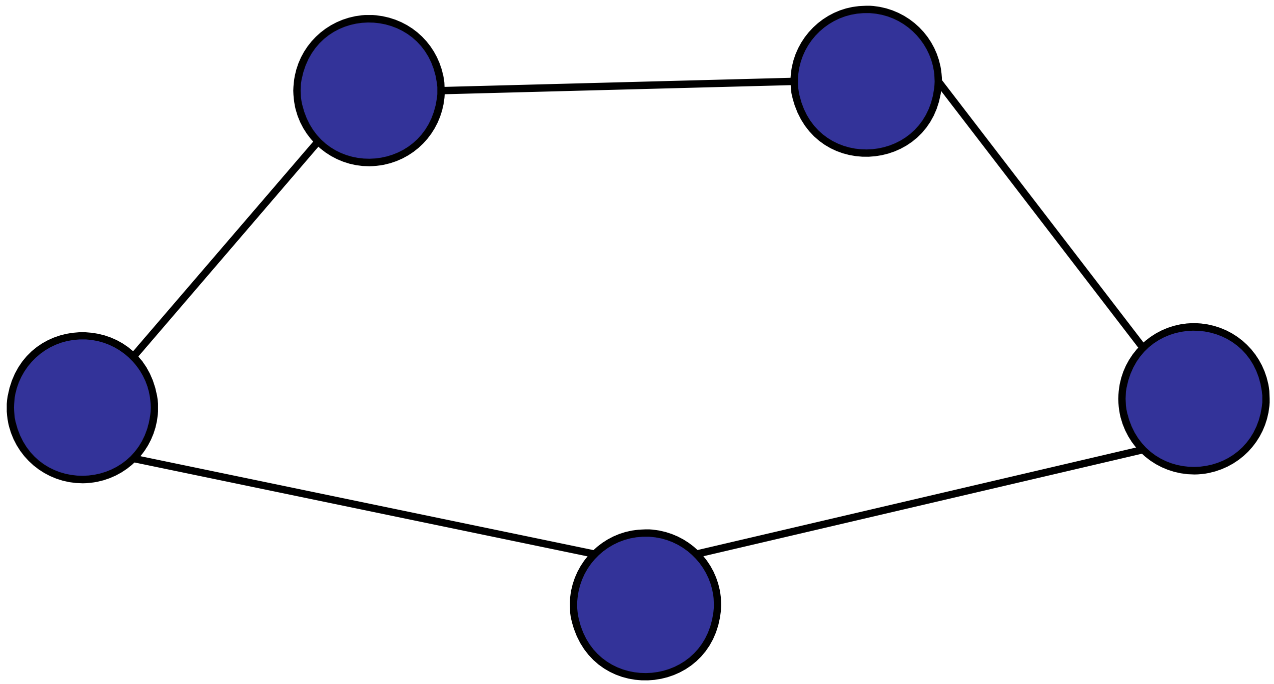
1. Yes
2. No



Is it bipartite?

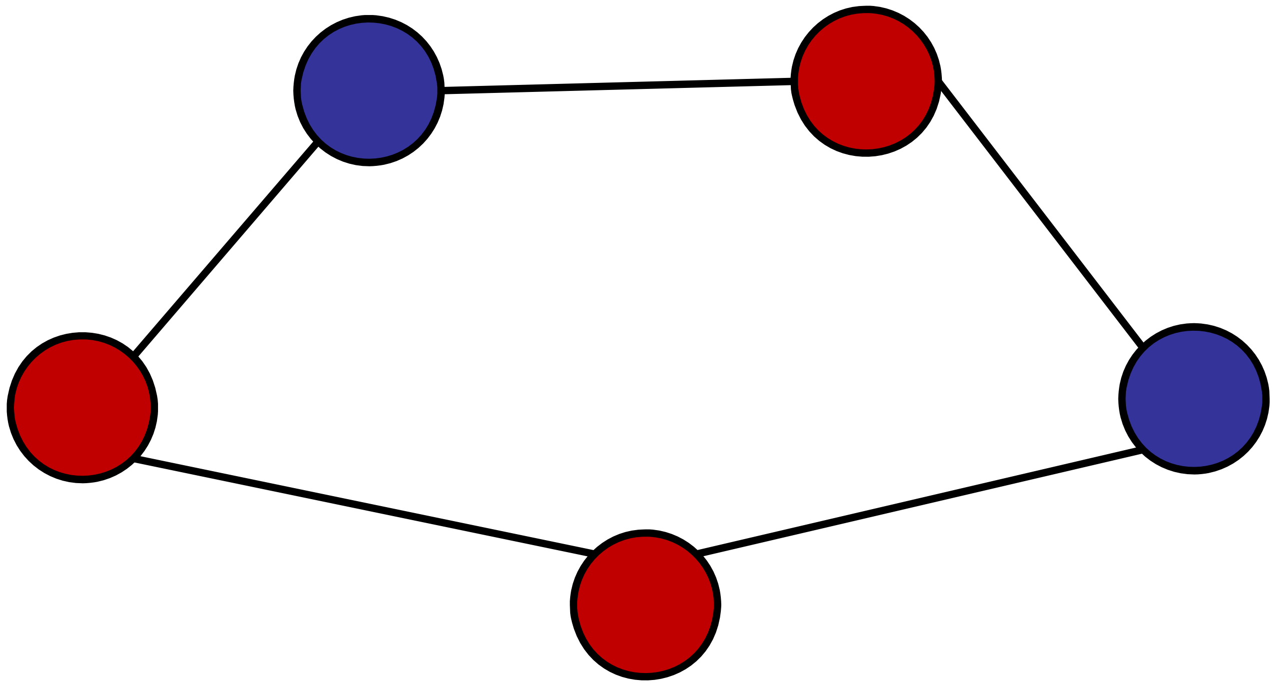
Yes or No on Zoom.

1. Yes
2. No



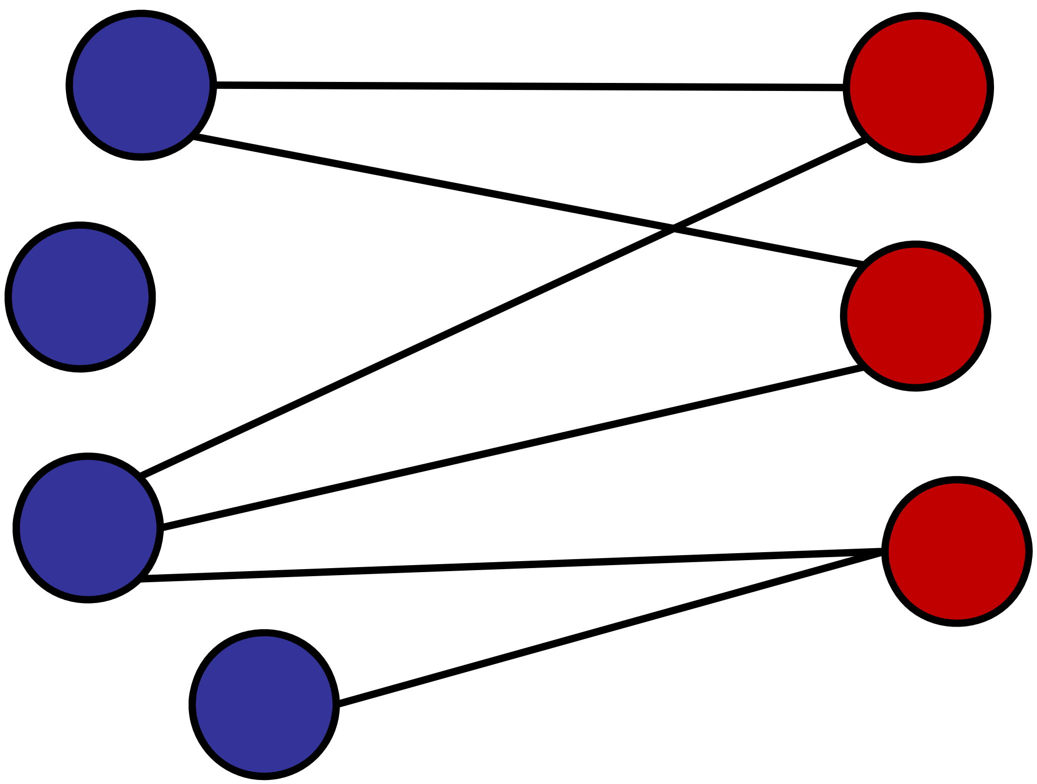
Is it bipartite?

1. Yes
2. No



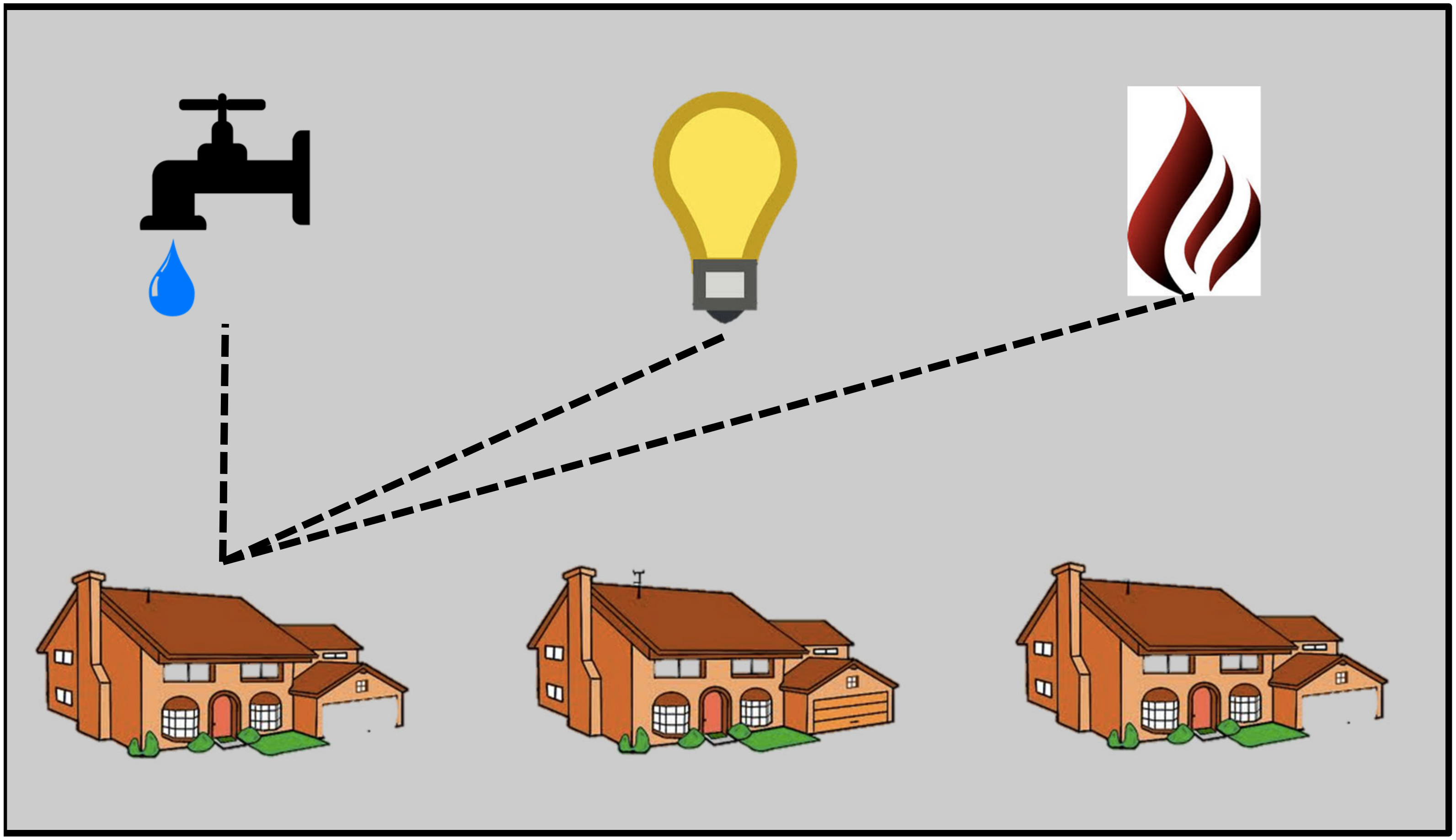
Special Graphs

Bipartite Graph



Nodes divided into two sets with no edges between nodes in the same set.

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.)

Roadmap

Today: Graph Basics

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

CS2040S
Data Structures and Algorithms

Graphs!
(Part 2)

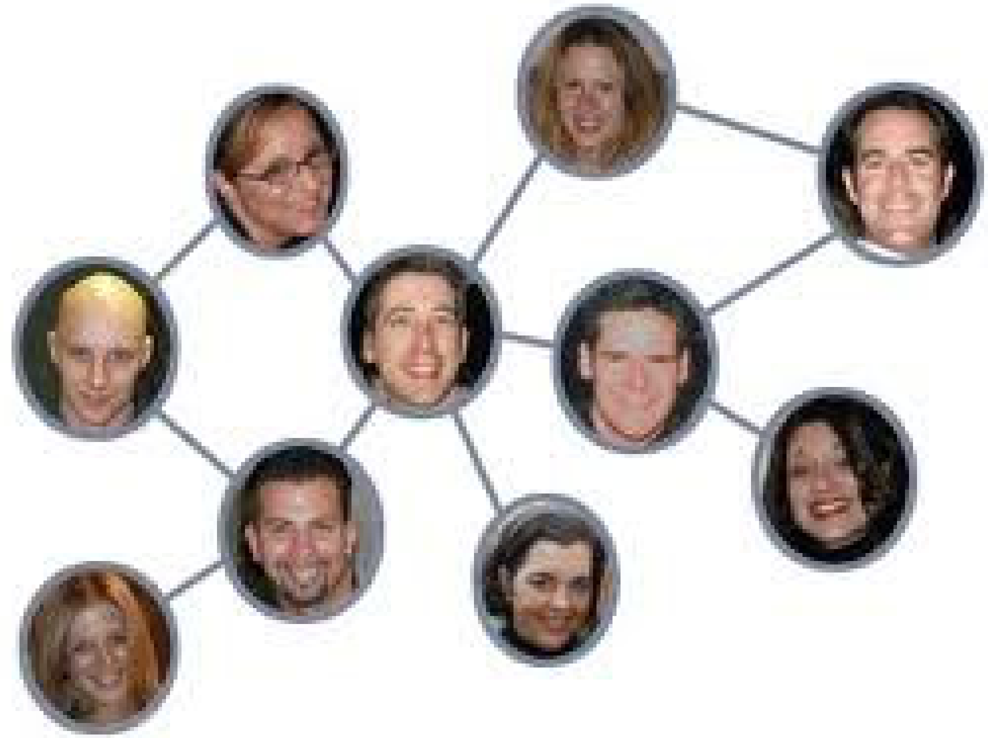
Where do we find graphs?

(How to model real problems as a graph!)

Where do we find graphs?

Social network:

- Nodes are people
- Edge = friendship



facebook

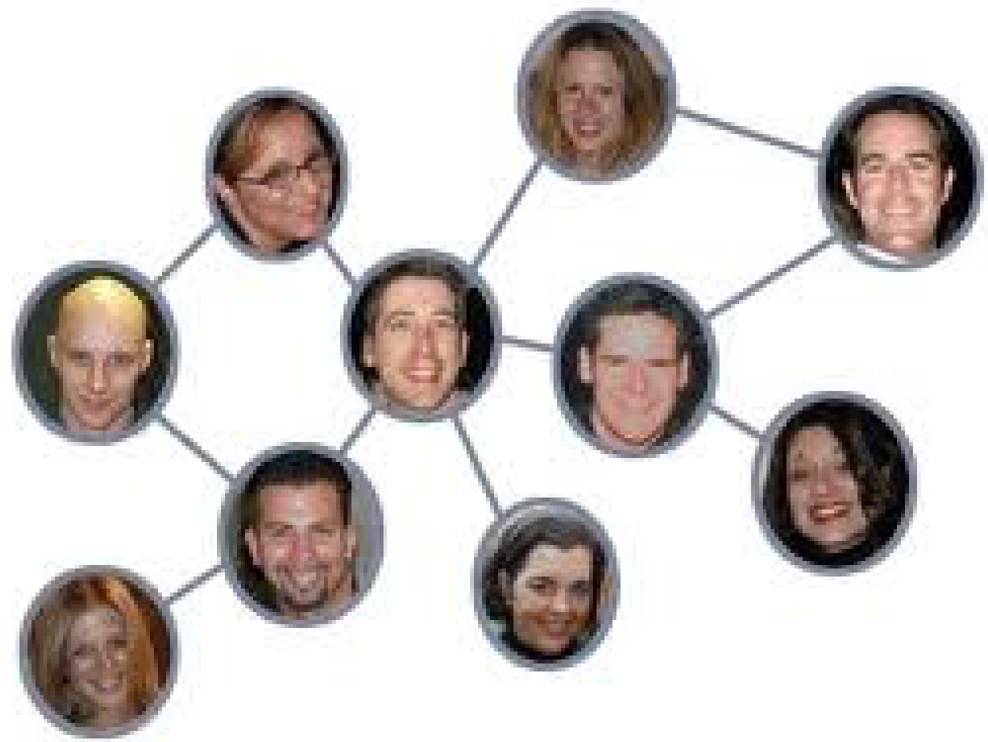
Where do we find graphs?

Social network:

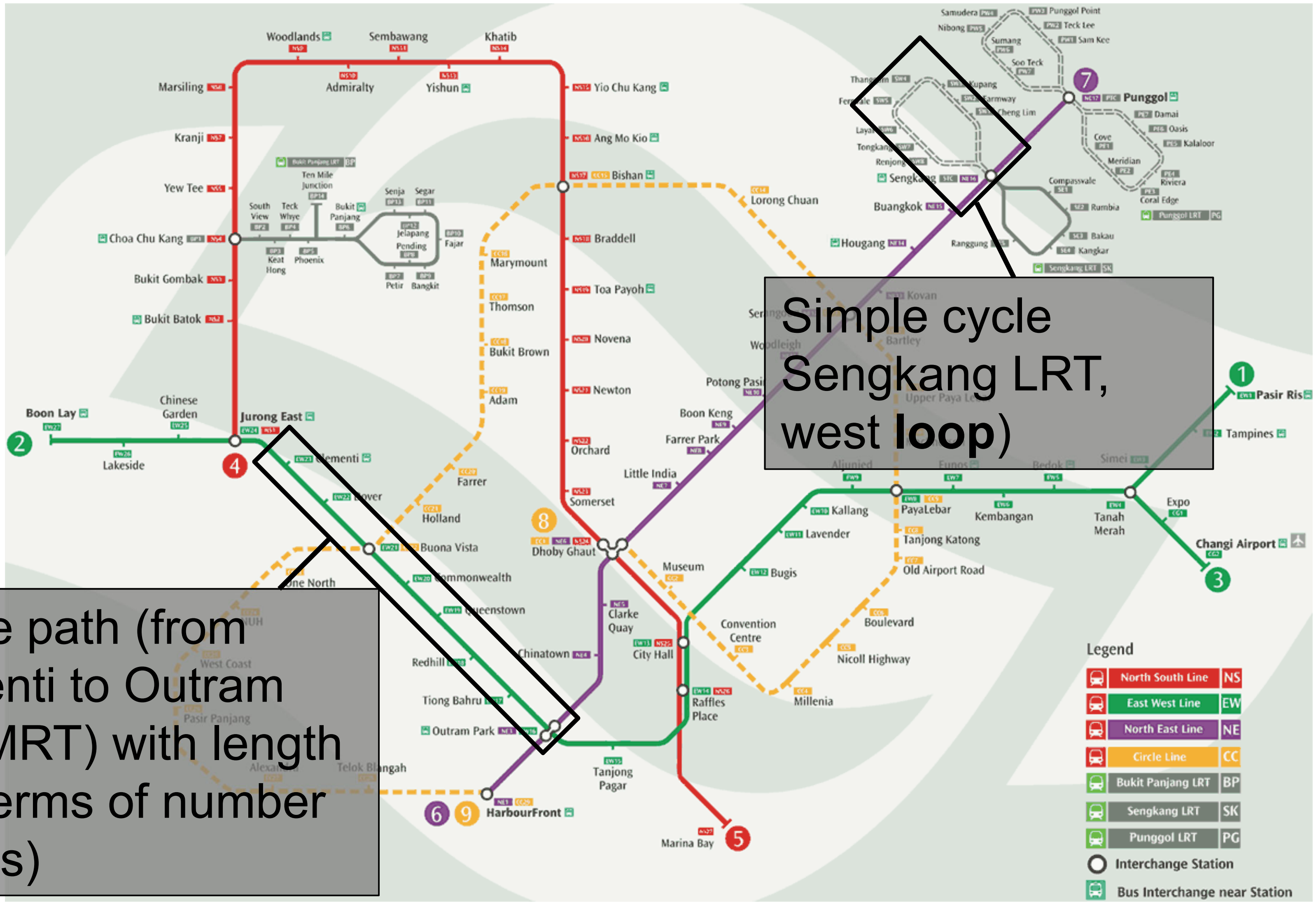
- Nodes are people
- Edge = friendship

Questions:

- Connected?
- Diameter?
- Degree?



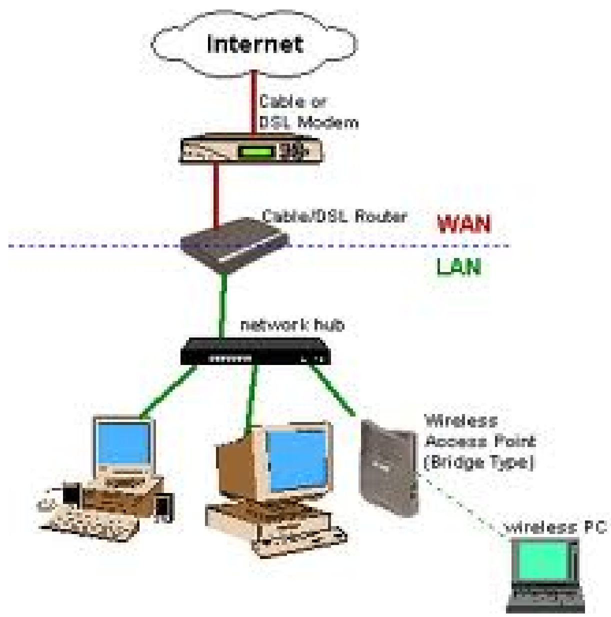
Transportation Network

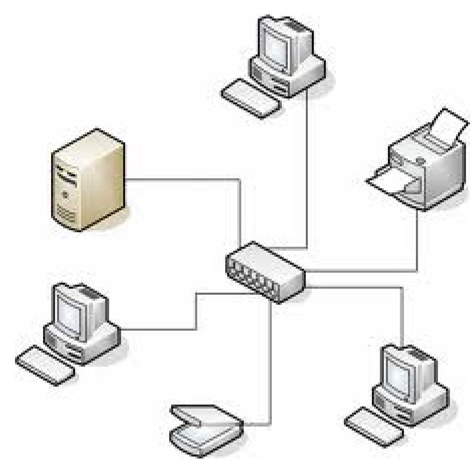


Simple cycle Sengkang LRT, west **loop**)

Simple path (from Clementi to Outram Park MRT) with length 7 (in terms of number of hops)

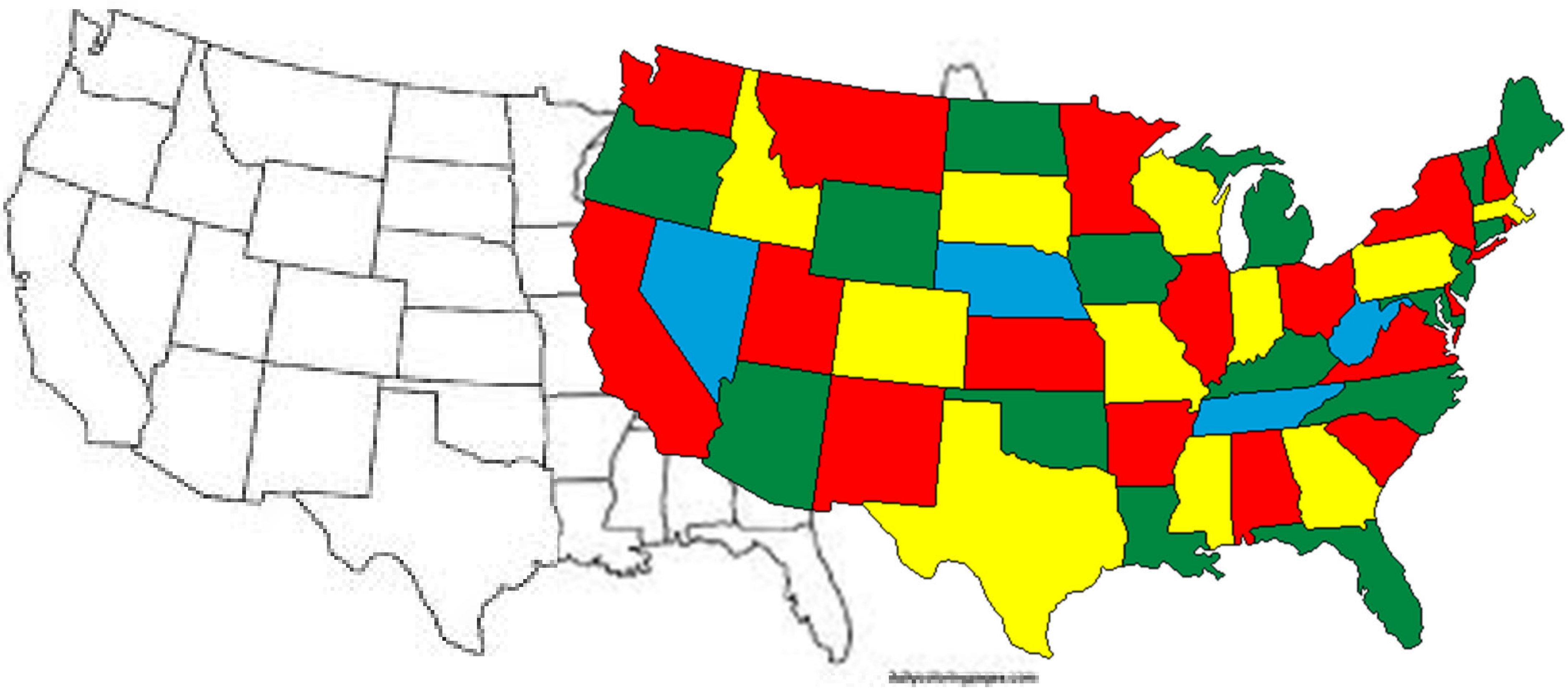
Internet / Computer Networks



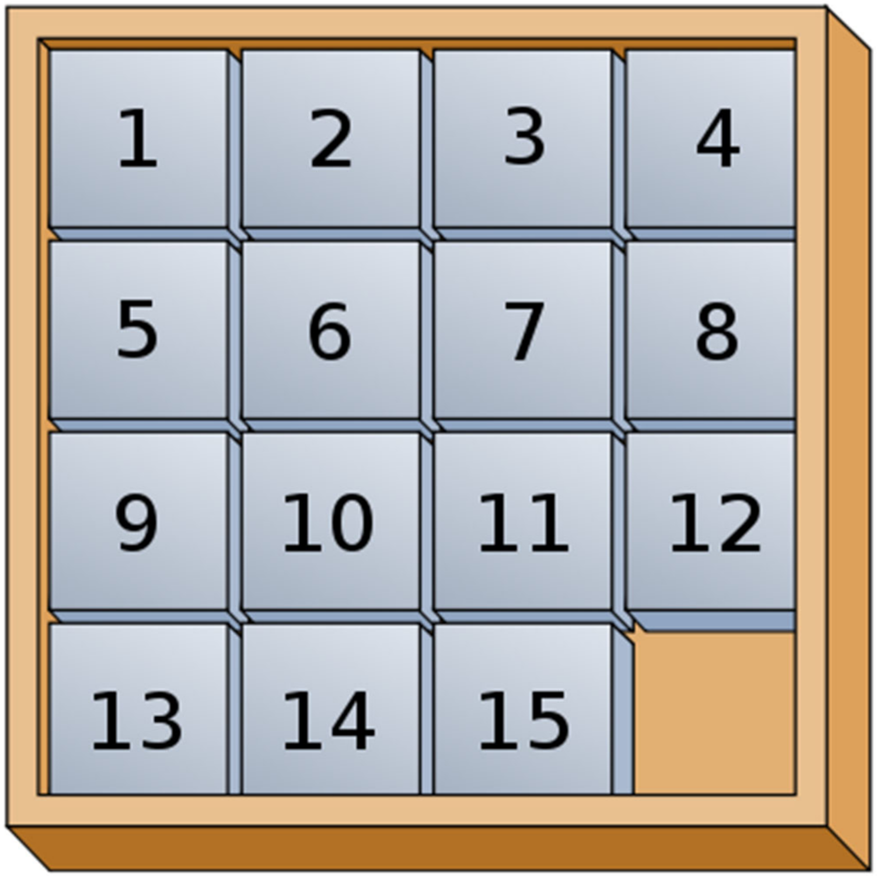


Communication Network

Optimization



Sliding Puzzle



Sliding Puzzle

4	5	7
3	1	6
8	2	

Sliding Puzzle

4	5	7
3	1	
8	2	6

Sliding Puzzle

4	5	
3	1	7
8	2	6

Sliding Puzzle

4		5
3	1	7
8	2	6

Sliding Puzzle

4	1	5
3		7
8	2	6

Sliding Puzzle

4	1	5
	3	7
8	2	6

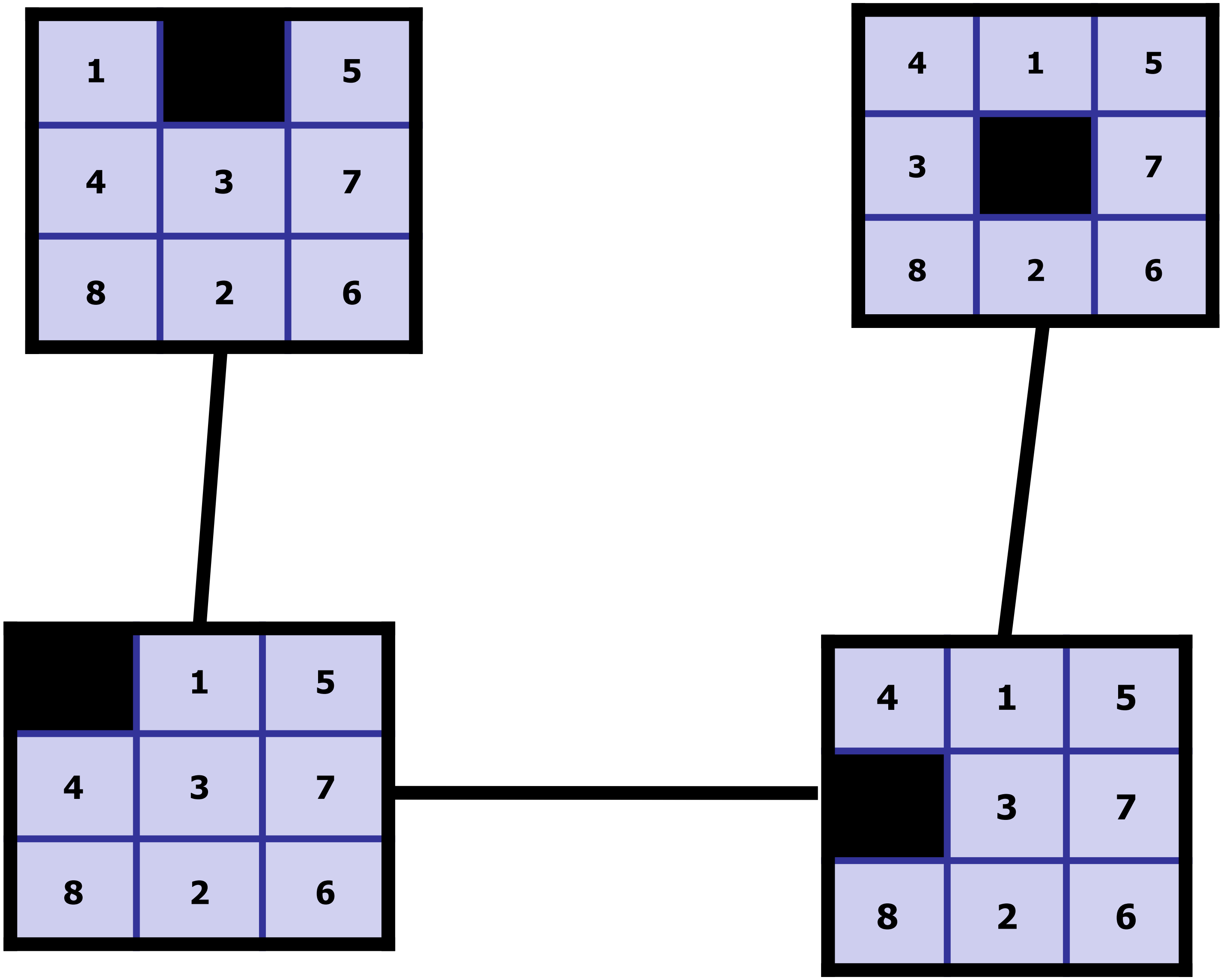
Sliding Puzzle

	1	5
4	3	7
8	2	6

Sliding Puzzle

1		5
4	3	7
8	2	6

Sliding Puzzle is a Graph



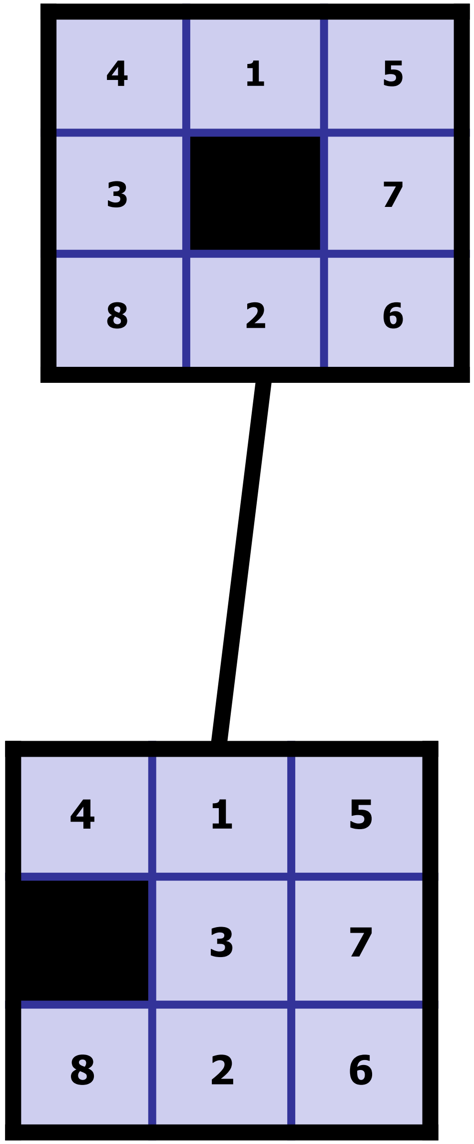
Sliding Puzzle

Nodes:

- State of the puzzle
- Permutation of nine tiles

Edges:

- Two states are edges if they differ by only one move.



What is the maximum degree of the Sliding Puzzle graph?

1. 1
2. 2
3. 3
4. 4
5. n/2
6. n
7. n!

Sliding Puzzle

Nodes:

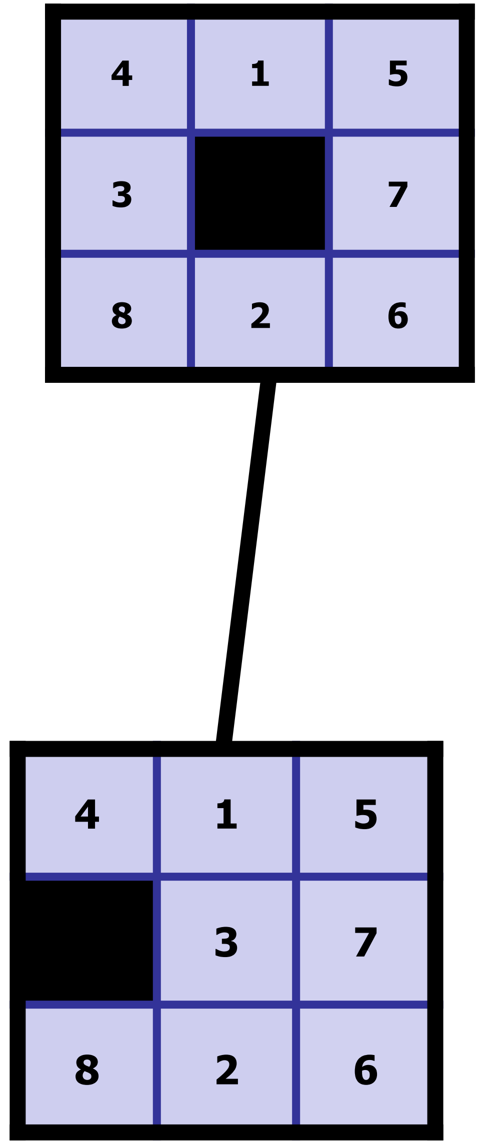
- State of the puzzle
- Permutation of nine tiles

Edges:

- Two states are edges if they differ by only one move.

Nodes = 9! = 362,880

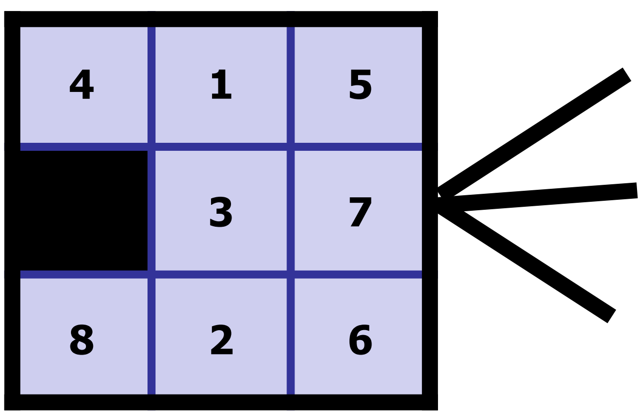
Edges < 4*9! < 1,451,520



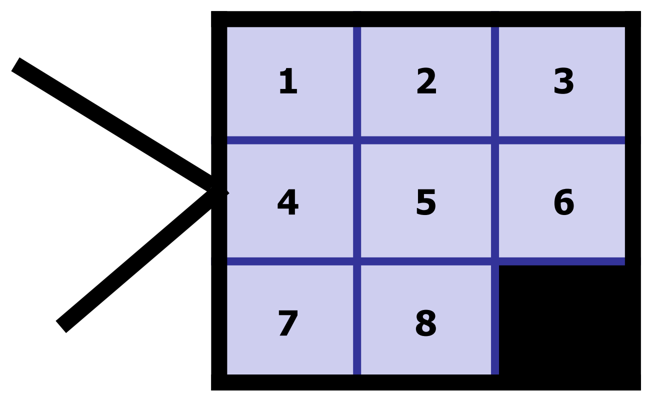
Sliding Puzzle

Number of moves to solve the puzzle?

Initial, scrambled state:



Final, unscrambled state:

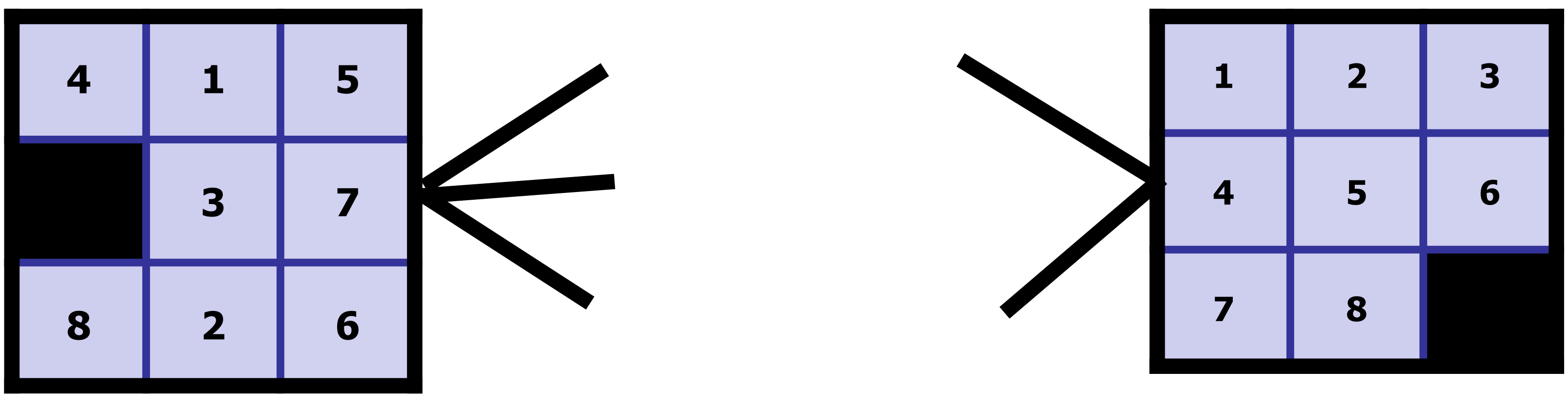


Sliding Puzzle

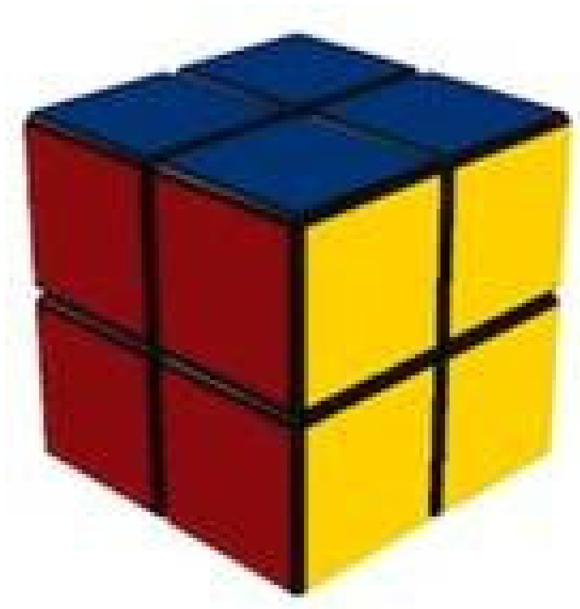
Number of moves <= Diameter

Initial, scrambled state:

Final, unscrambled state:

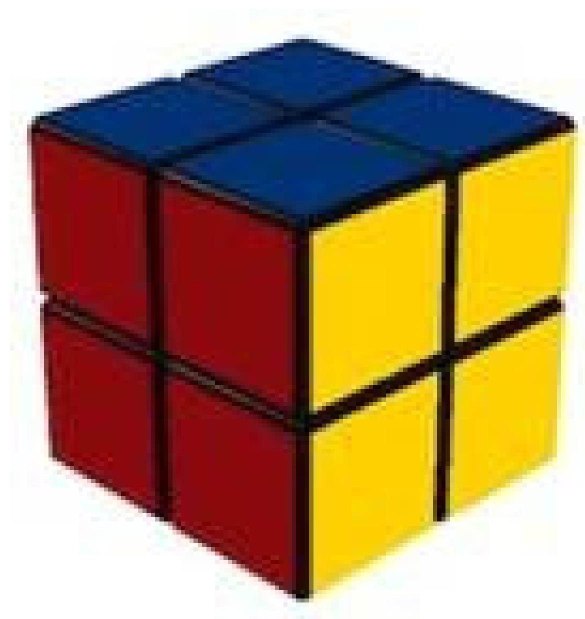


2 x 2 x 2 Rubik's Cube



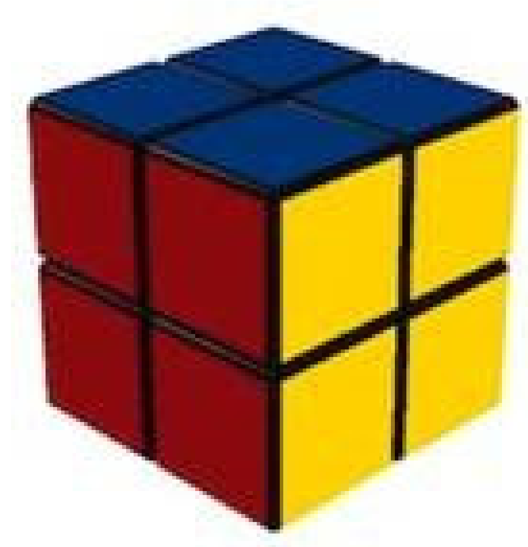
[redacted]

2 x 2 x 2 Rubik's Cube



Record solve time: 0.69 seconds

2 x 2 x 2 Rubik's Cube

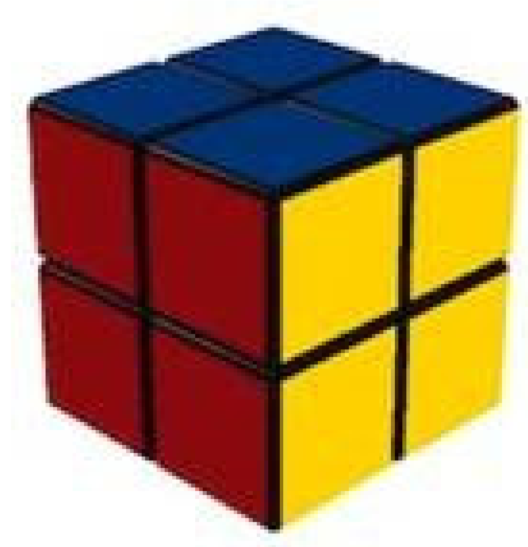


Configuration Graph

- 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.

2 x 2 x 2 Rubik's Cube



How many vertices?

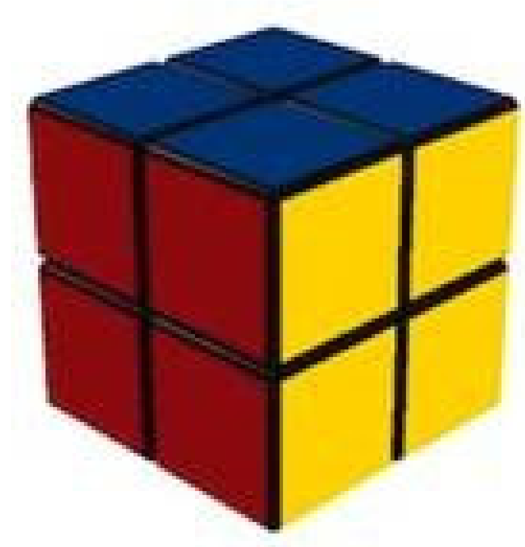
$$8! \cdot 3^8 = 264{,}539{,}520$$

cubelets

Each cubelet is
in one of 8 positions.

Each of the 8 cubelets
can be in one of three
orientations

2 x 2 x 2 Rubik's Cube



How many vertices?

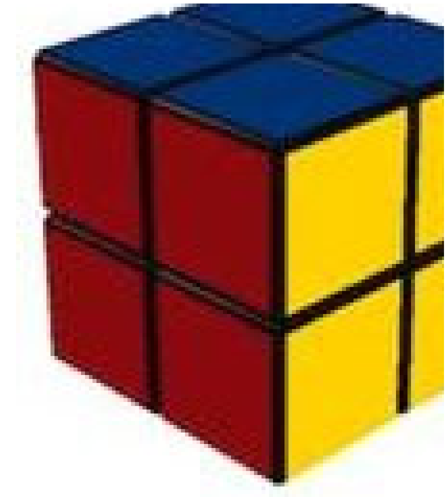
$$7! \cdot 3^7 = 11{,}022{,}480$$

Symmetry:

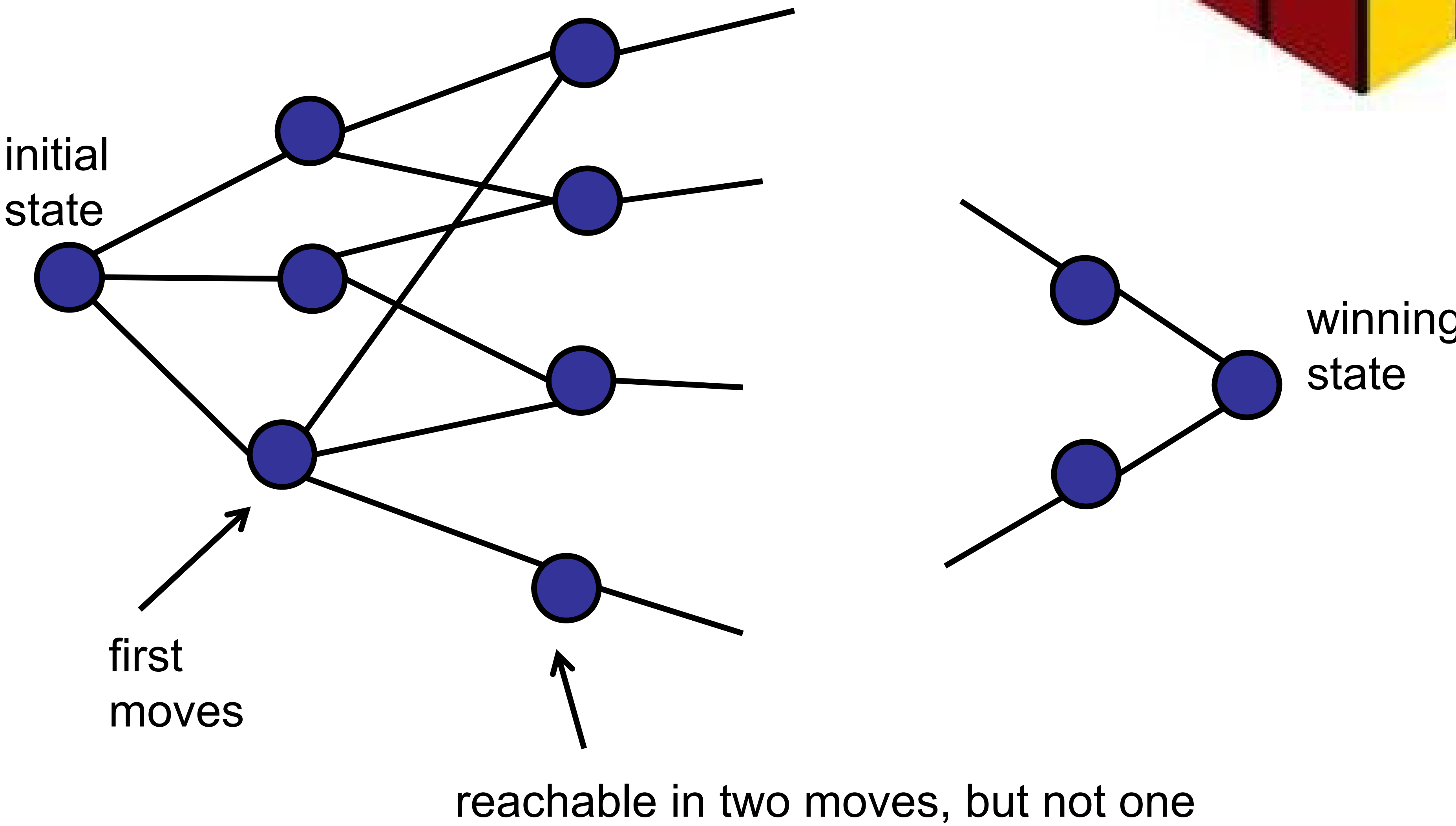
Fix one cubelet.

Each of the 8 cubelets can be in one of three orientations

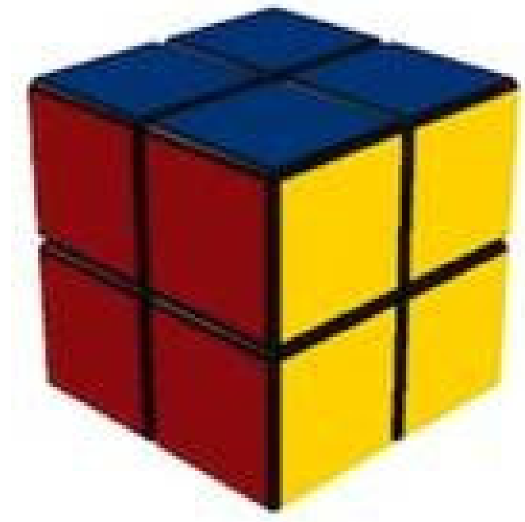
2 x 2 x 2 Rubik's Cube



Geography of Rubik's configurations:



Reachable configurations



Distance	90 deg. turns	90/180 deg. turns
0	1	1
1	6	9
2	27	54
3	120	321
4	534	1,847
5	2,256	9,992
6	8,969	50,136
7	33,058	227,536
8	114,149	870,072
9	360,508	1,887,748
0	930,588	623,800
11	1,350,852	2,644
12	782,536	
13	90,280	
14	276	

diameter

Reachable configurations

Distance	90 deg. turns	90/120 deg. turns
0	1	1
1	6	9
2	27	54
9	360,508	1,887,748
0	930,588	623,800
11	1,350,852	2,644
12	782,536	
13	90,280	
14	276	

diameter

Challenge:
How do you generate this table?

3 x 3 x 3 Rubik's Cube

Configuration Graph

- 43 quintillion vertices (approximately)
- Diameter: 20
  - 1995: require at least 20 moves.
  - 2008: 20 moves is enough from every position.
  - Using Google server farm.
  - 35 CPU-years of computation.
  - 20 seconds / set of 19.5 billion positions.
  - Lots of mathematical and programming tricks.

3 x 3 x 3 Rubik's Cube

What is the diameter of an (n x n x n) cube?

$\theta(n^2 / \log n)$

Roadmap

Today: Graph Basics

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

CS2040S
Data Structures and Algorithms

Graphs!
(Part 3)

Representing a Graph

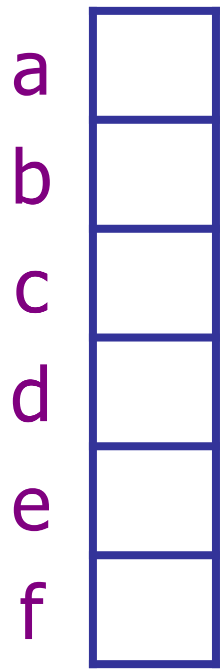
Graph consists of:

- Nodes
- Edges

Representing a Graph

Graph consists of:

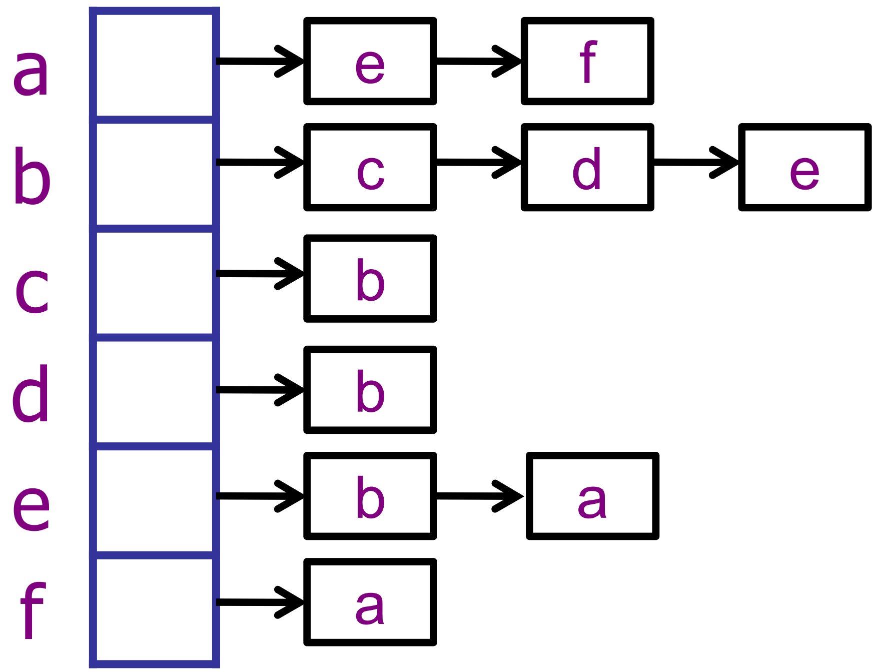
- Nodes: stored in an array
- Edges

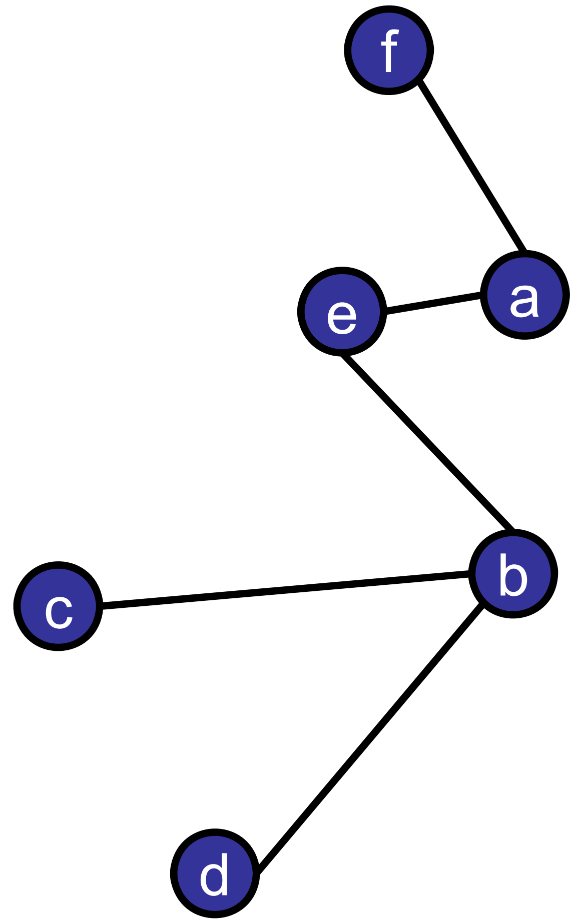


Adjacency List

Graph consists of:

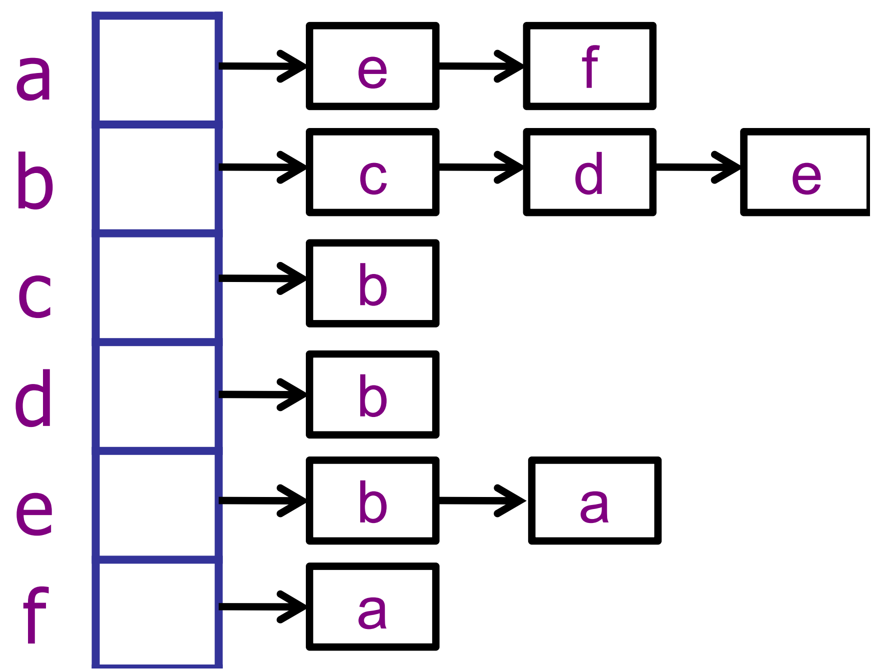
- Nodes: stored in an array
- Edges: linked list per node





Adjacency List in Java

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

class Node {
  int key;
  NeighborList nbrs;
}

class Graph {
  Node[] nodeList;



}
```

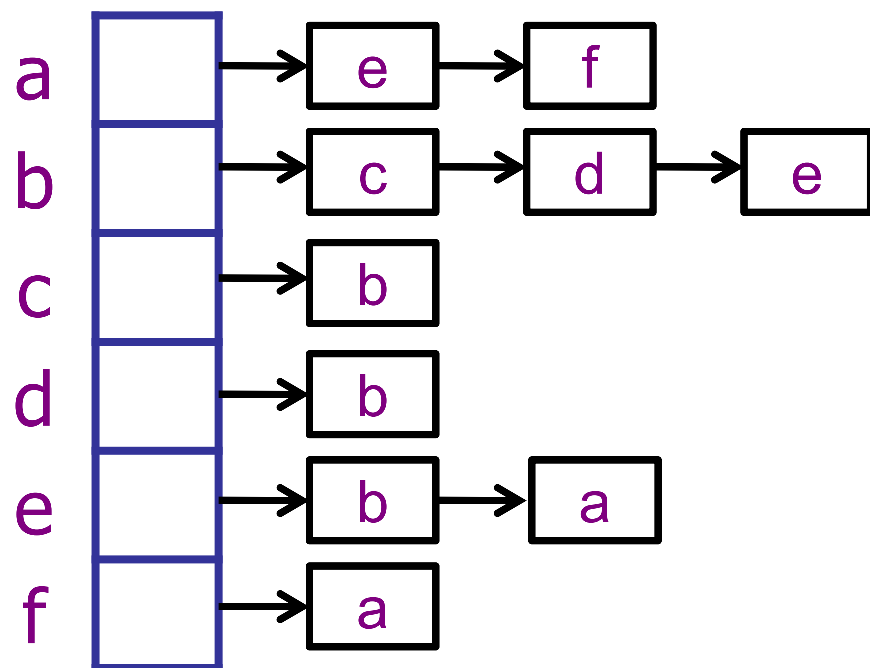


Adjacency List in Java

```
class Graph{
    List<List<Integer>> nodes;


}
```

More concise code is not *always* better…

- Harder to read
- Harder to debug
- Harder to extend



Representing a Graph

Graph consists of:

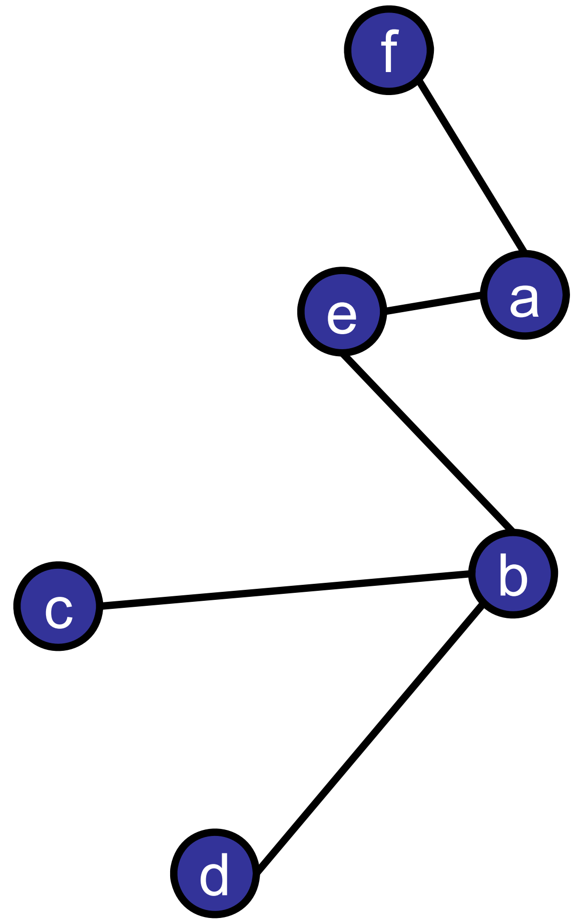
- Nodes
- Edges = pairs of nodes

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

Graph represented as:

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

Neat property:

- A^2 = length 2 paths

	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

To find out if c and d are 2-hop neighbors:

- Let $B = A^2$
- $B[c, d] = A[c, .] \bullet A[., d]$

- $B[c, d] >= 1$ iff

  $A[c, x] == A[x, d]$

  for some x.

	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

Graph represented as:

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

Neat properties:

- A^2 = length 2 paths
- A^4 = length 4 paths

	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

Neat way to figure out connectivity…
Neat way to figure out diameter…
Not always the most efficient…
Parallelizes well….

Adjacency Matrix

Graph represented as:

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

Neat properties:

- A^2 = length 2 paths
- A^4 = length 4 paths
- A^∞ = Google pagerank

	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 in Java

Graph represented as:

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

```
class Graph {
  boolean[][] adjMatrix;

}
```

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

Adjacency Matrix in Java

Graph represented as:

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

```
class Graph {

  Node[][] adjMatrix;


}
```

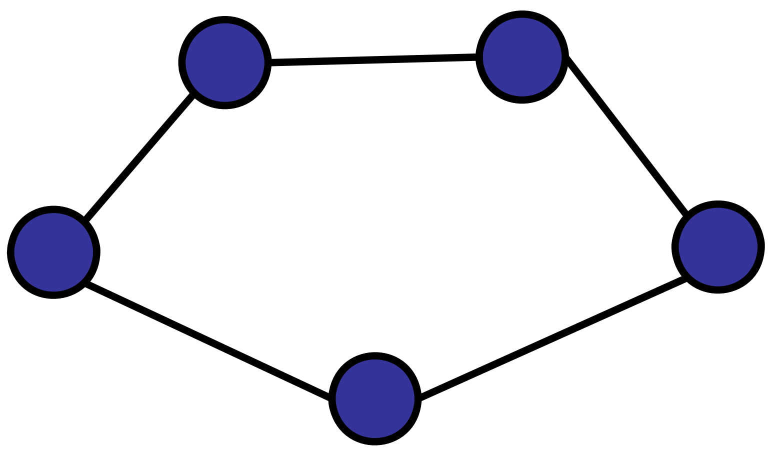
	a	b	c	d
a	0	0	0	0
b	0	0	**1**	**1**
c	0	**1**	0	0
d	0	**1**	0	0
e	**1**	**1**	0	0
f	**1**	0	0	0

Trade-offs

Adjacency Matrix vs. Array?

For a cycle, which representation is better?

✓ 1. Adjacency list
2. Adjacency matrix
3. Equivalent



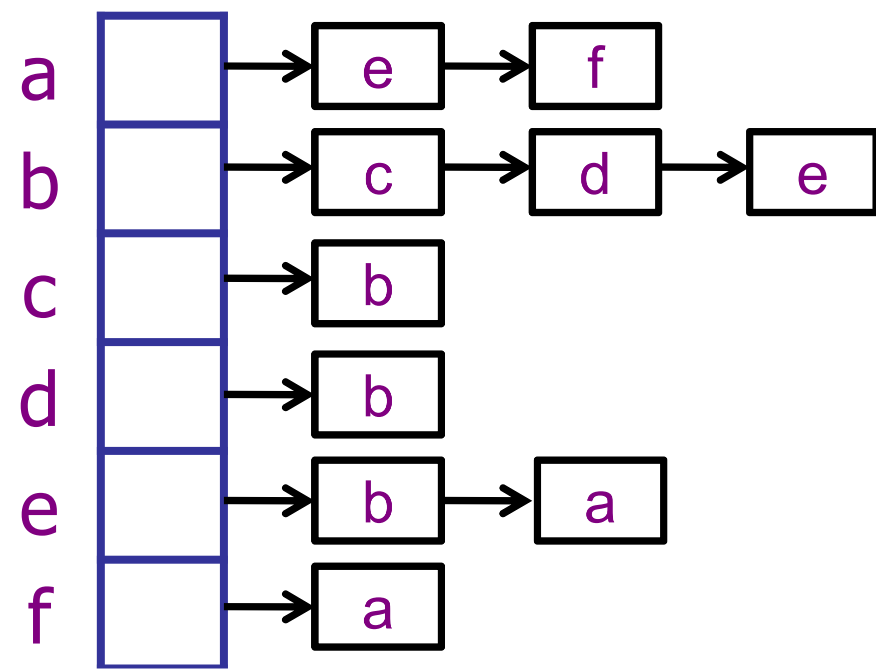
Adjacency List

Memory usage for graph G = (V, E):

- array of size |V|
- linked lists of size |E|

Total: O(V + E)

For a cycle: O(V)



Adjacency Matrix

Memory usage for graph G = (V, E):

- array of size |V|*|V|

Total: $O(V^2)$

For a cycle: $O(V^2)$

	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

For a clique, which representation is better?

1. Adjacency matrix
2. Adjacency list
3. Equivalent

Adjacency List vs. Matrix

Memory usage for graph $G = (V, E)$:

- Adjacency List: $O(V + E)$
- Adjacency Matrix: $O(V^2)$

For a cycle: $O(V)$ vs. $O(V^2)$

For a clique: $O(V + E) = O(V^2)$ vs. $O(V^2)$

Adjacency List vs. Matrix

Memory usage for graph $G = (V, E)$:

- Adjacency List: $O(V + E)$
- Adjacency Matrix: $O(V^2)$

For a cycle: $O(V)$ vs. $O(V^2)$

For a clique: $O(V + E) = O(V^2)$ vs. $O(V^2)$

Base rule: if graph is dense then use an adjacency matrix; else use an adjacency list.

dense: $|E| = \theta(V^2)$

Which representation for Facebook Graph?
Query: Are Bob and Joe friends?

1. Adjacency List
2. Adjacency Matrix
3. Equivalent

List: (much) better space.
Matrix: somewhat faster

Which representation for Facebook Graph?
Query: List all my friends?

✓ 1. Adjacency List
2. Adjacency Matrix
3. Equivalent

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?

Graph Representations

Key questions to ask:

- Space usage: is graph dense or sparse?
- Queries: what type of queries do I need?
    - Enumerate neighbors?
    - Query relationship?

Roadmap

Today: Graph Basics

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

CS2040S
Data Structures and Algorithms

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)

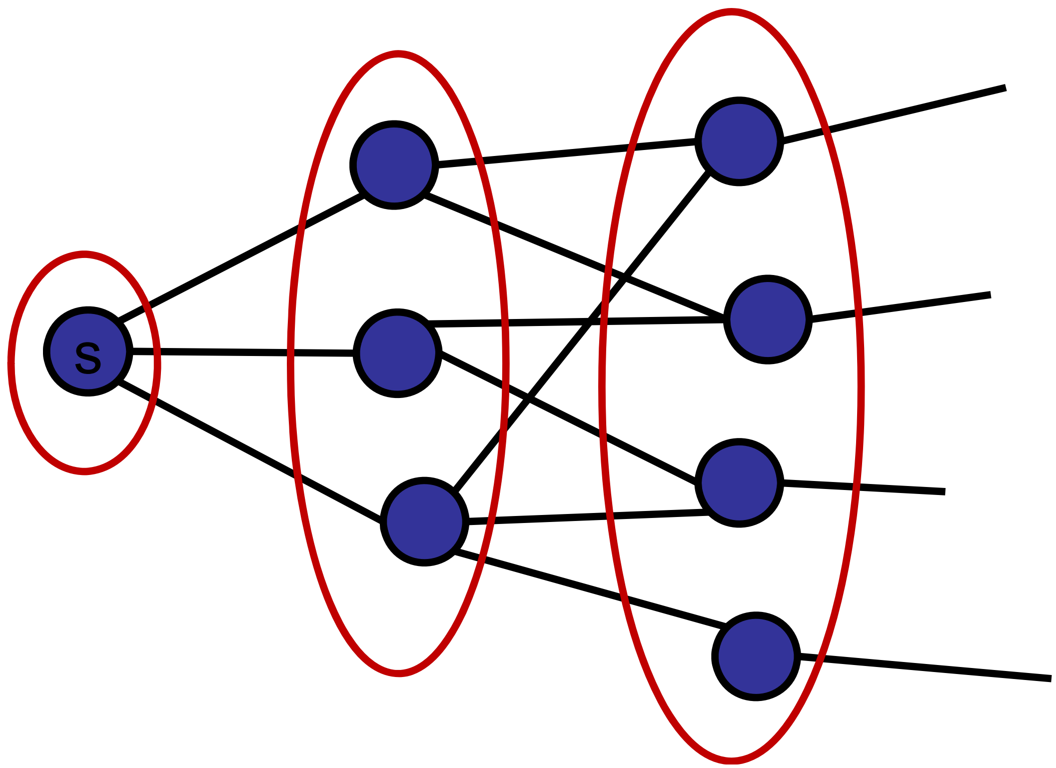
Graph representation:

- Adjacency list

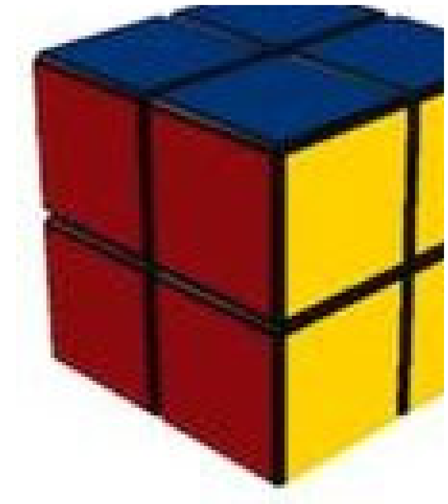
Searching a graph

Breadth-First Search:

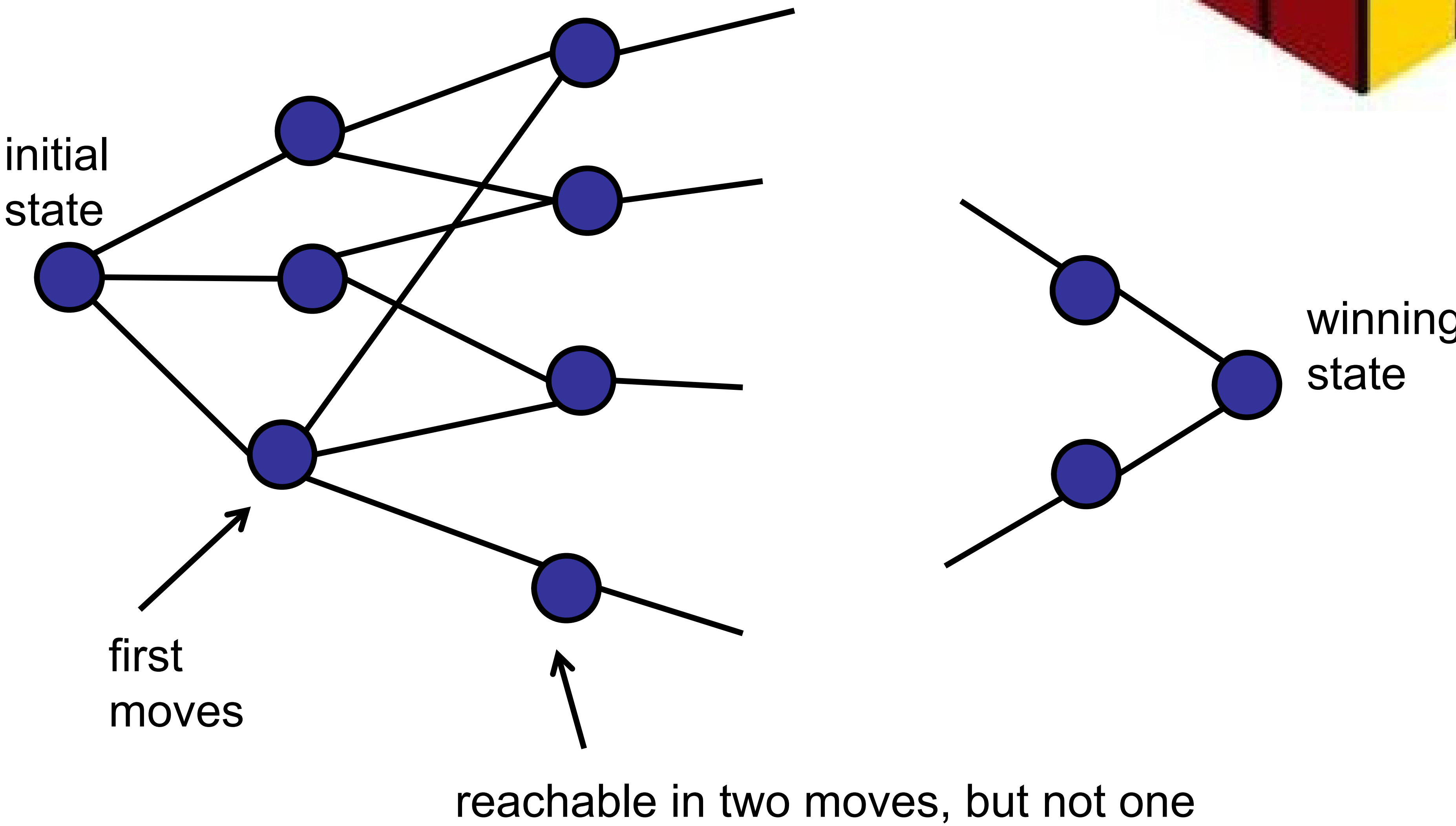
- Explore level by level



2 x 2 x 2 Rubik's Cube



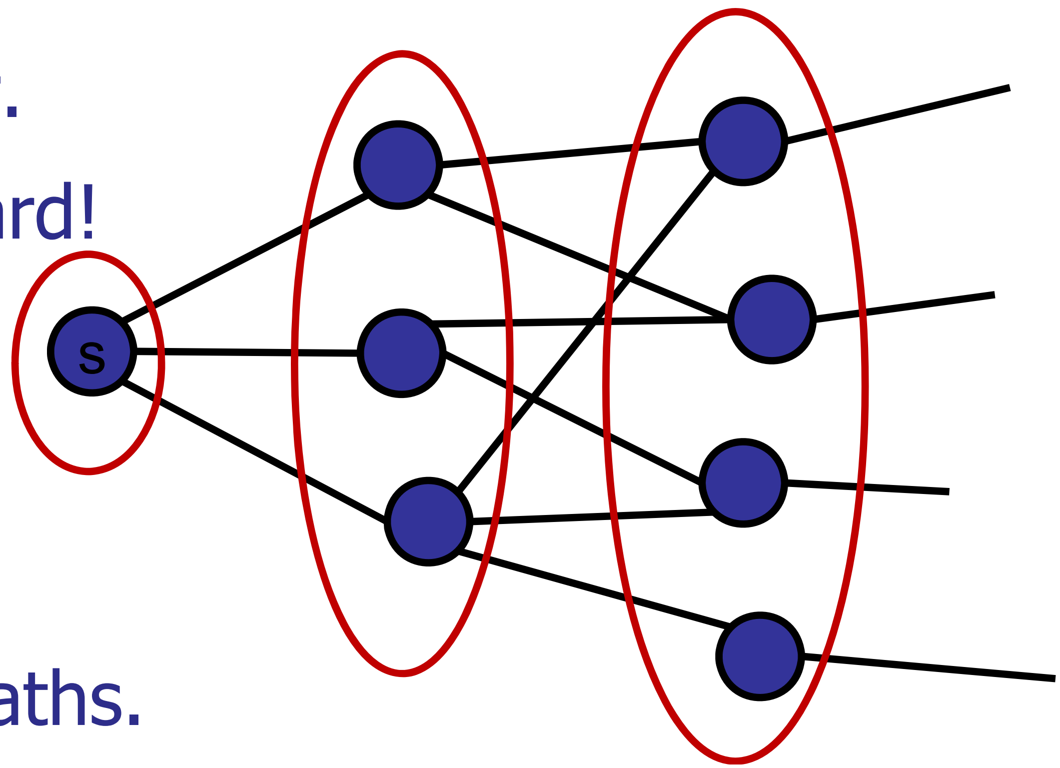
Geography of Rubik's configurations:



Searching a graph

Breadth-First Search:

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

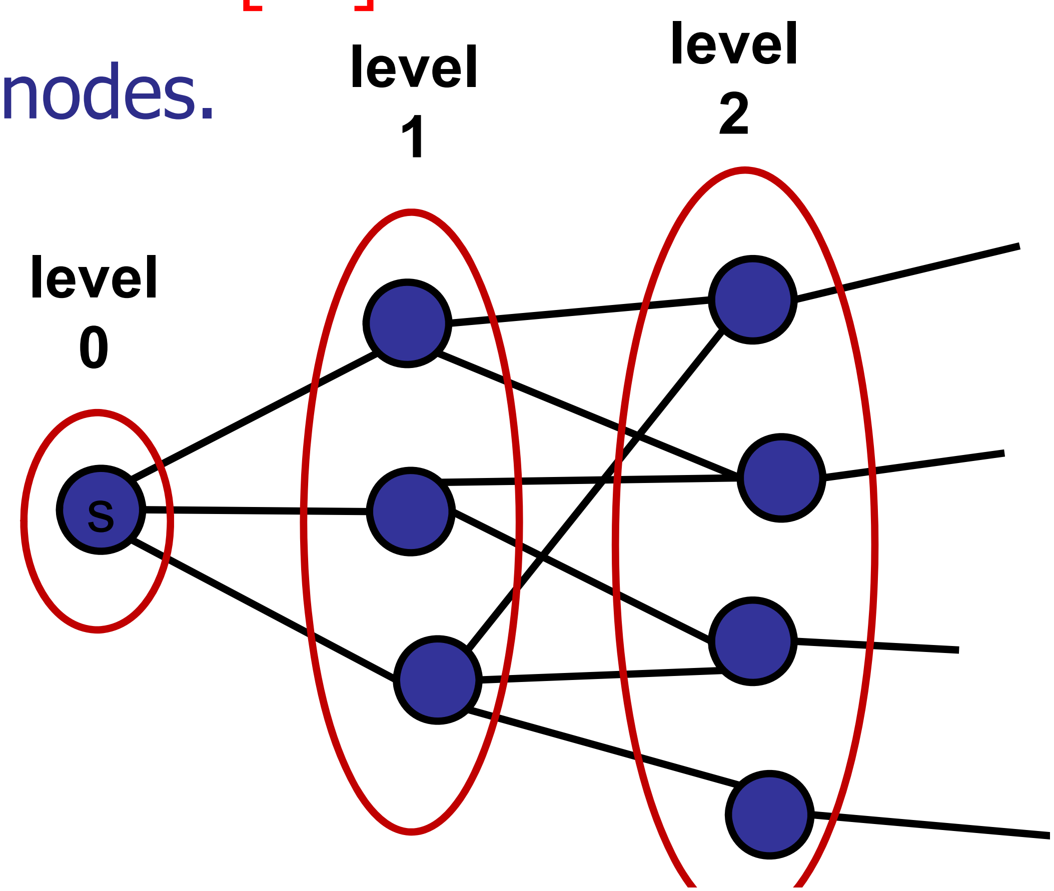


- Finds shortest paths.

Searching a graph

Breadth-First Search:

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



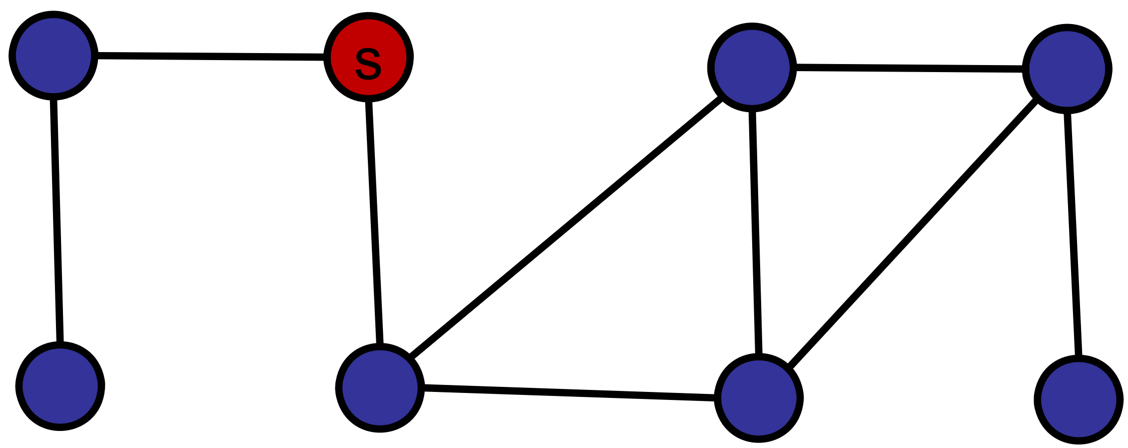
Breadth-First Search

```
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

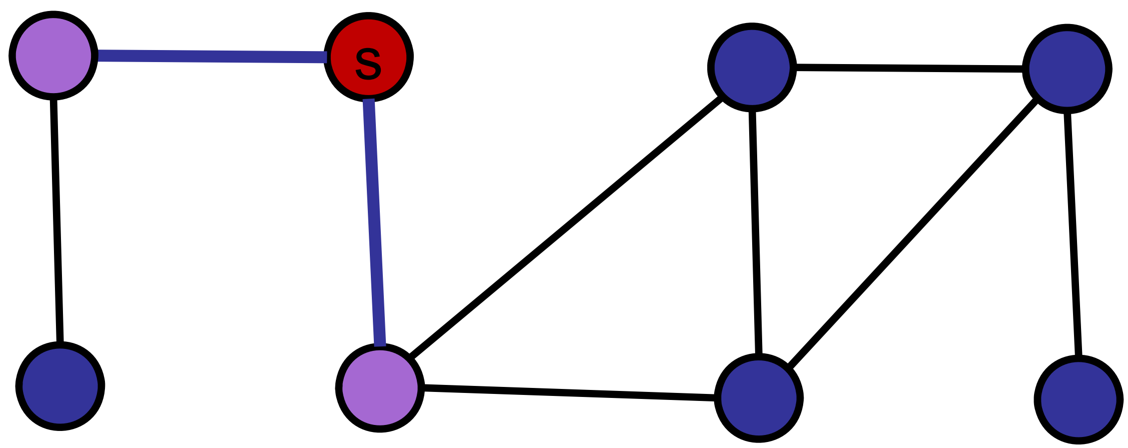
```
while (!frontier.isEmpty()){
   Collection<Integer> nextFrontier = new … ;
   for (Integer v : frontier) {
         for (Integer w : nodeList[v].nbrList) {
               if (!visited[w]) {
                     visited[w] = true;
                     parent[w] = v;
                     nextFrontier.add(w);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

Breadth-First Search Example



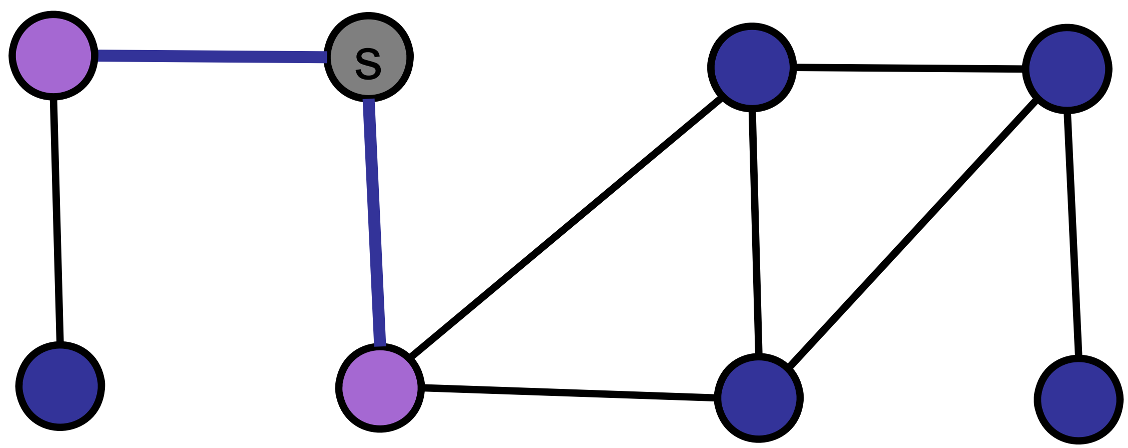
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



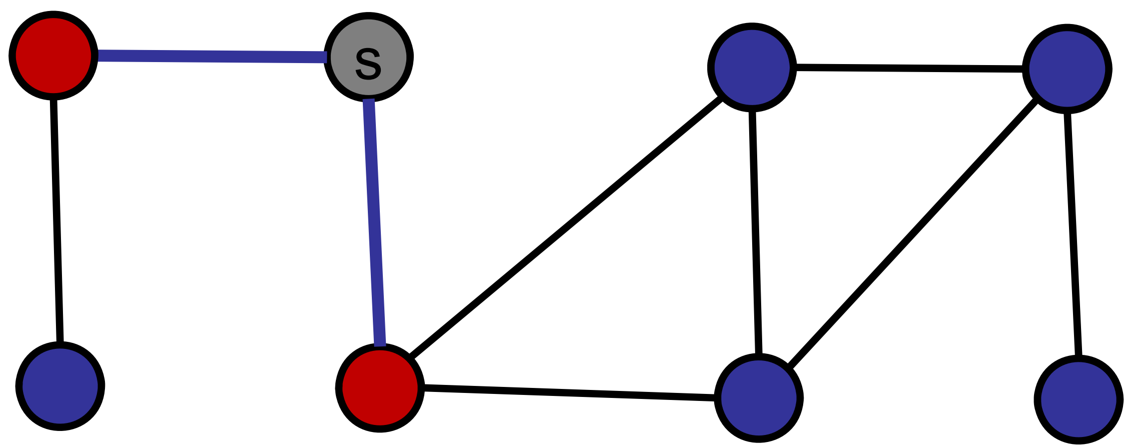
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



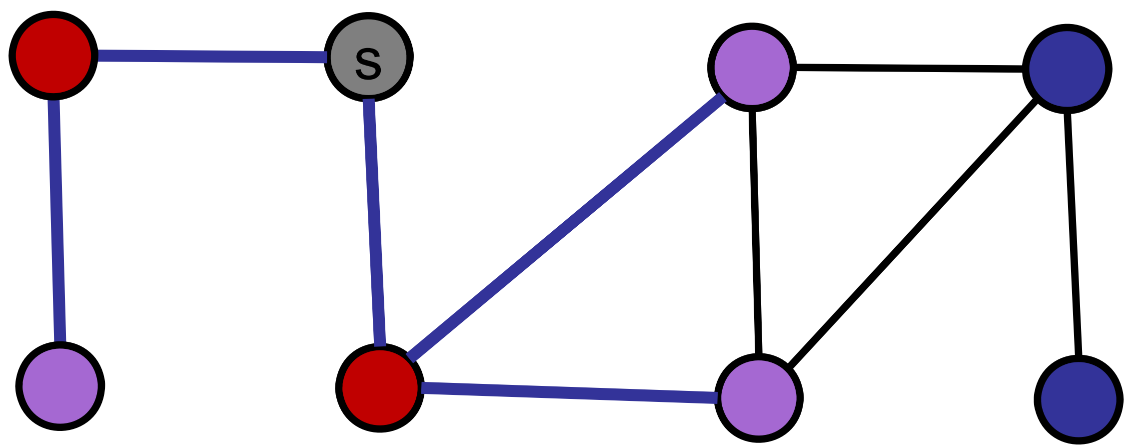
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



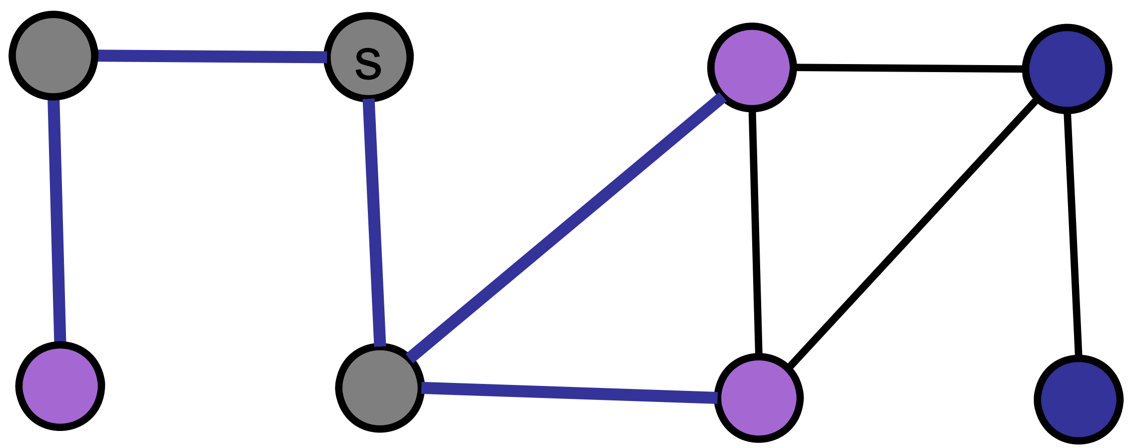
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



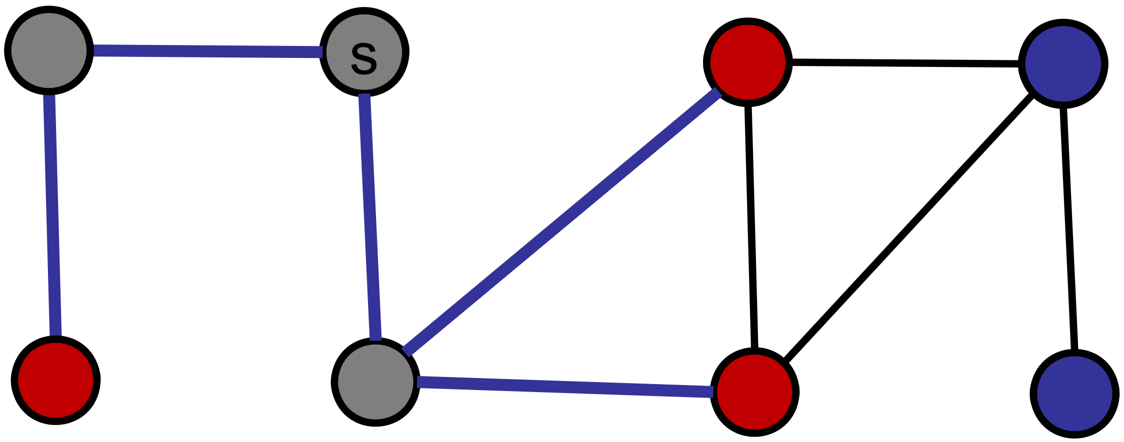
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



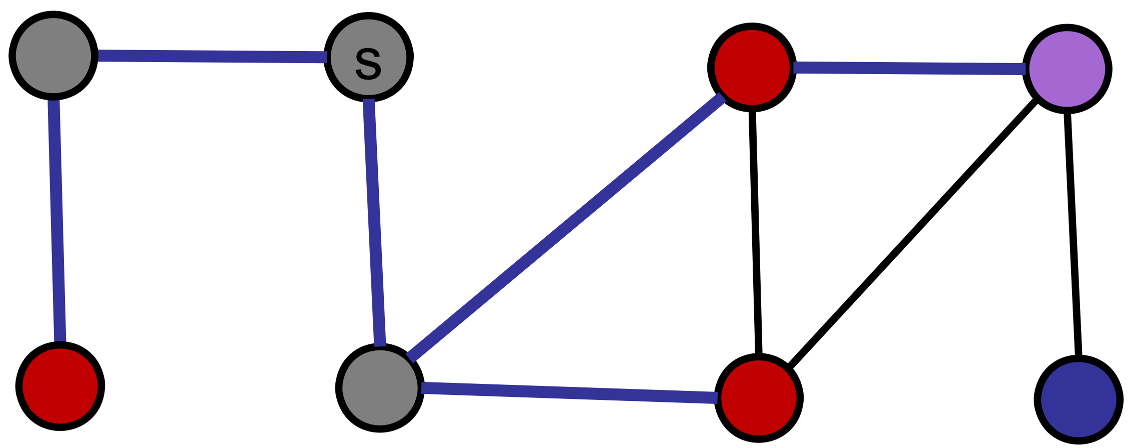
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



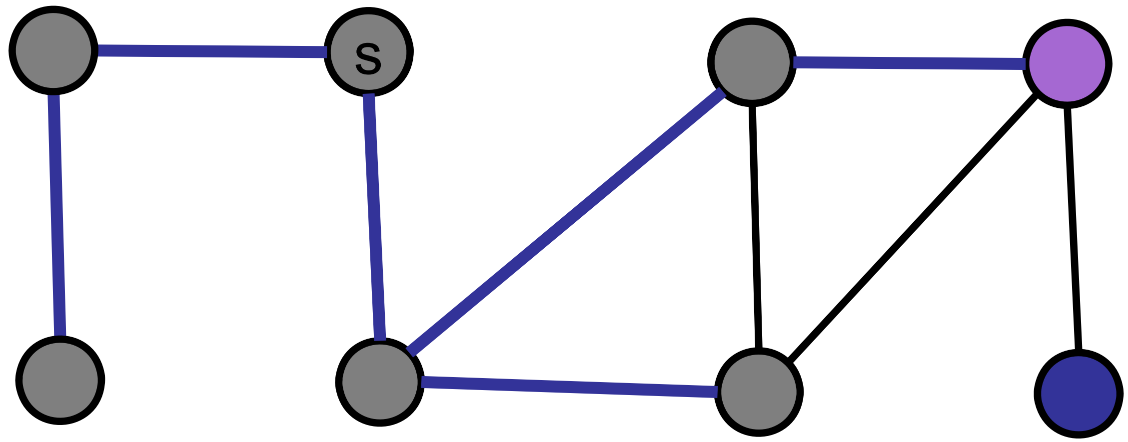
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



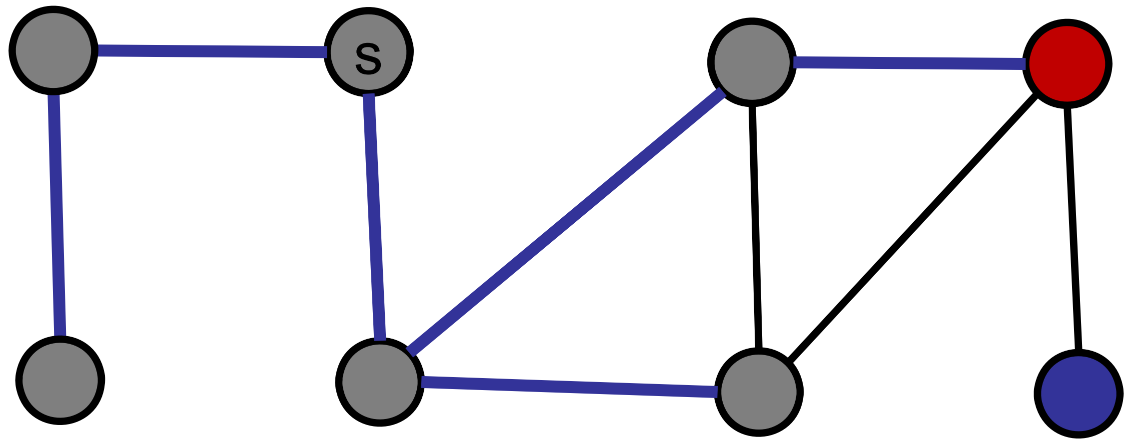
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



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

Breadth-First Search Example



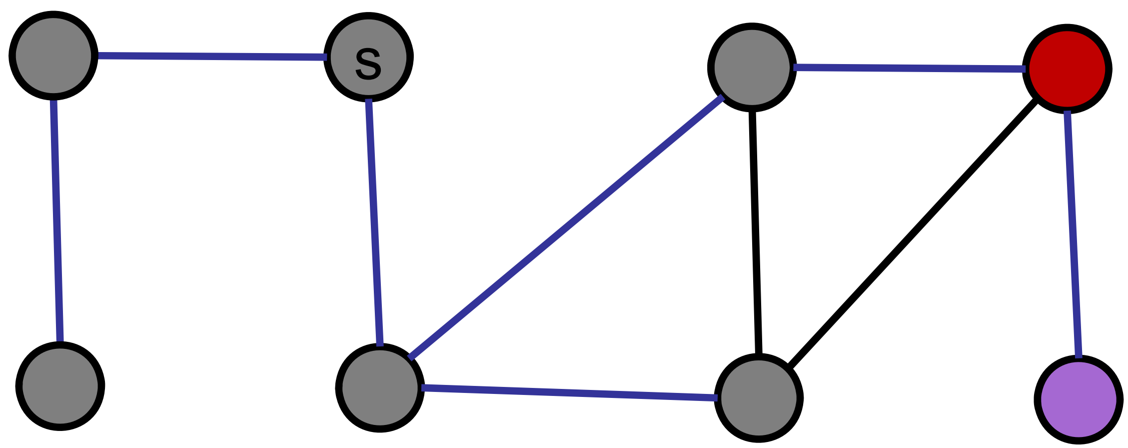
Red = active frontier

Purple = next

Gray = visited

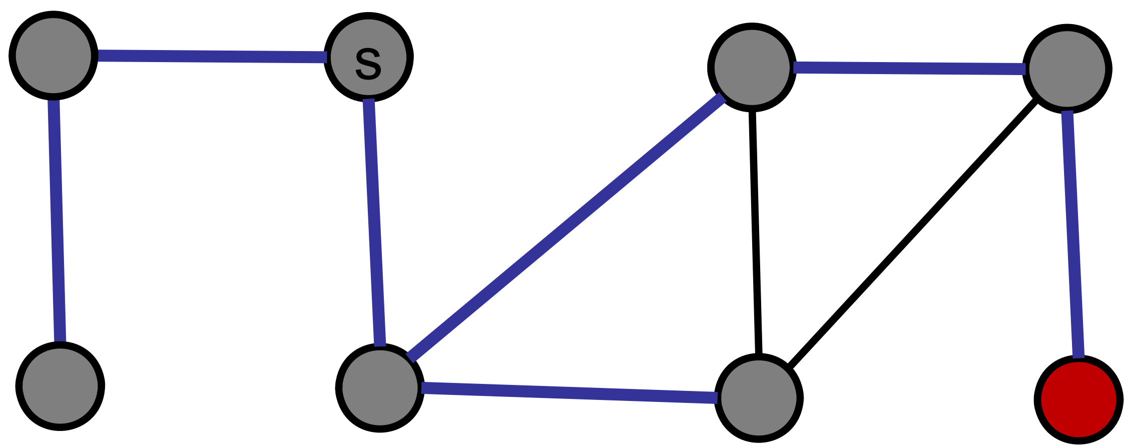
Blue = unvisited

Breadth-First Search Example



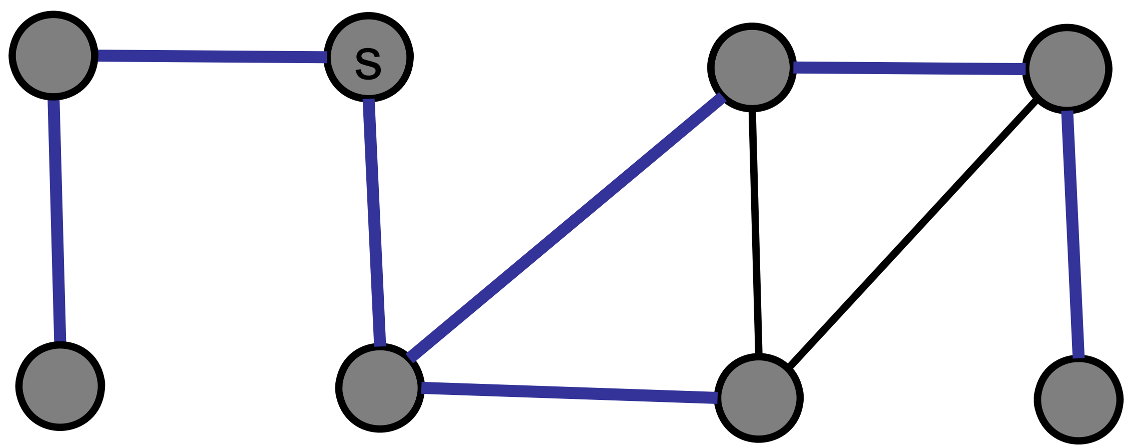
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



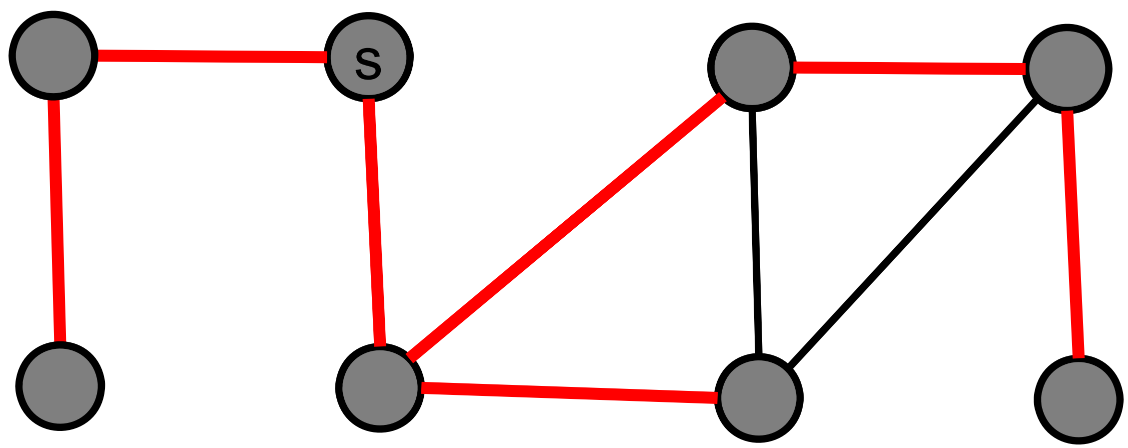
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

Breadth-First Search Example



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

Breadth-First Search Example



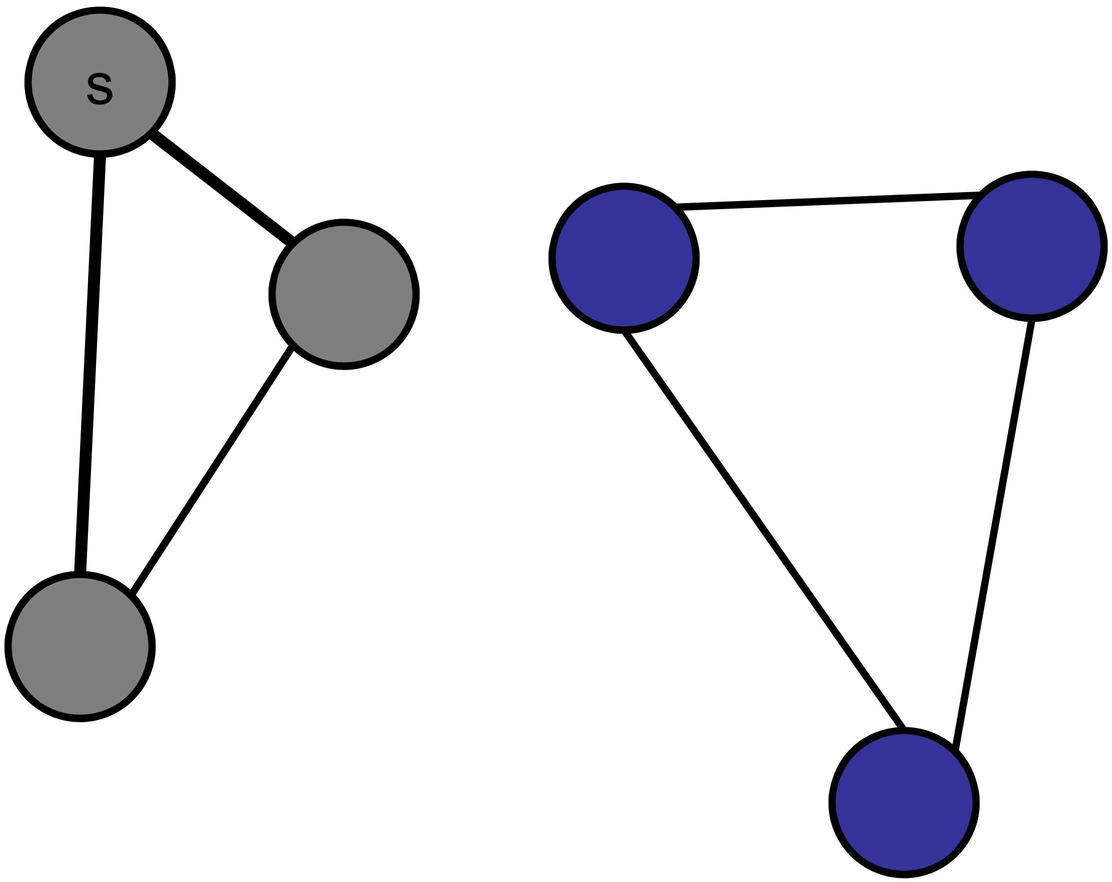
Red = active frontier
Purple = next
Gray = visited
Blue = unvisited

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.

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.

Depends on adjacency list vs. adjacency matrix.

Here: assume adjacency list.

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)

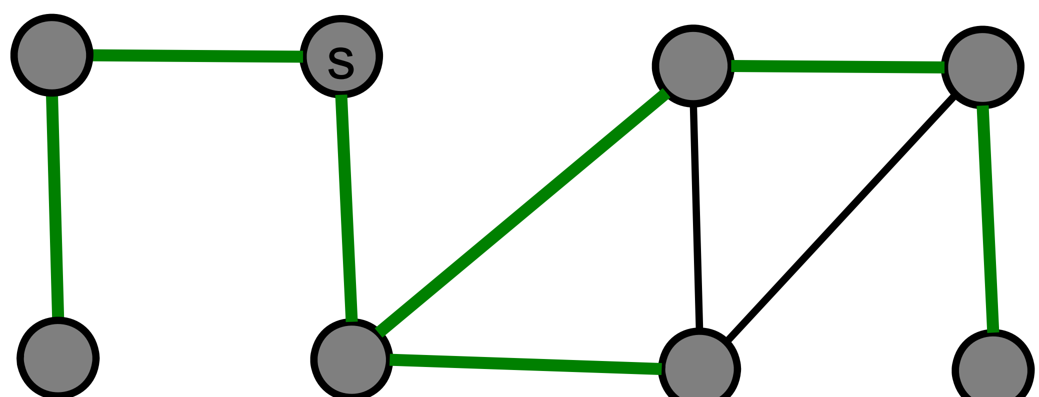
Breadth-First Search

```
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.



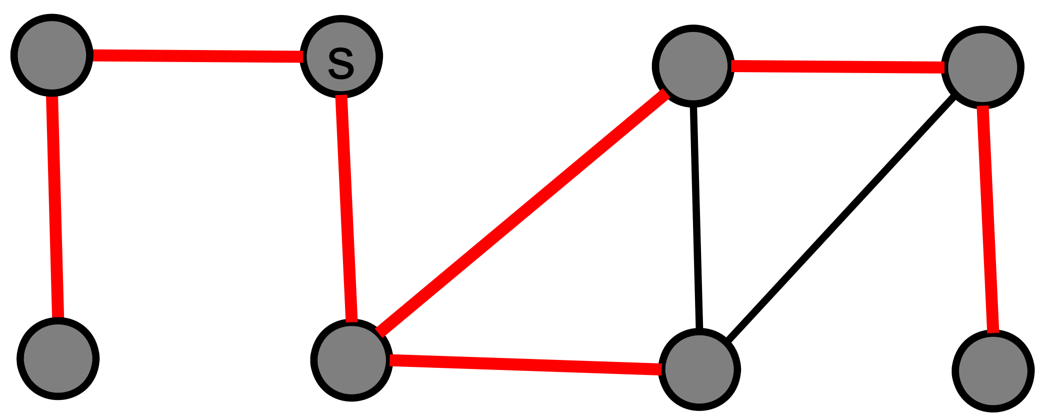
Which is true? (More than one may apply.)

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