Announcements

- Homework 0 online, due F
- Course Discord linked on course webpage
- Remember to fill out FinAid survey

Today

- Levels of Algorithm Design
- Introduction to graph algorithms
  - Graph basics and representation
  - Explore/DFS

Levels of Algorithm Design

Levels of Algorithm Design

Naive Algorithms: Turn definition into algorithm

easy to write, good first pass, often very slow

Levels of Algorithm Design

Naive Algorithms: Turn definition into algorithm

easy to write, good first pass, often very slow

Toolkit: Algorithm designed using standard tools

the main focus of this course

Levels of Algorithm Design

Naive Algorithms: Turn definition into algorithm

easy to write, good first pass, often very slow

Toolkit: Algorithm designed using standard tools

the main focus of this course

Optimized: Use data structures or other ideas to make algorithm especially efficient

Levels of Algorithm Design

Naive Algorithms: Turn definition into algorithm

easy to write, good first pass, often very slow

Toolkit: Algorithm designed using standard tools

the main focus of this course

Optimized: Use data structures or other ideas to make algorithm especially efficient

Magic: Sometimes, an algorithm requires a surprising new insight

Graph and Connectivity (Ch 3)

- Graph basics and representation
- Depth First Search
- Connected components
- Pre- and Post- orderings
- DAGs / Topological Sort
- General directed graphs & strongly connected components

Graph Definition

Definition: A *graph* G = (V,E) consists of two things:

Graph Definition

Definition: A *graph* G = (V,E) consists of two things:

- A collection V of *vertices*, or objects to be connected.

Graph Definition

Definition: A *graph* G = (V,E) consists of two things:

- A collection V of *vertices*, or objects to be connected.
- A collection E of *edges*, each of which connects a pair of vertices.

Question: Which are graphs?

Which of the following can't be modeled by a graph? (multiple correct answers)

A) The internet, V = {websites}, E = {links}
B) The internet, V = {computers}, E = {physical connections}
C) UCSD, V = {students}, E = {classes}
D) Highway System, V = {intersections}, E = {roads}
E) A book, V = {words}

Question: Which are graphs?

Which of the following can't be modeled by a graph? (multiple correct answers)

A) The internet, V = {websites}, E = {links}
B) The internet, V = {computers}, E = {physical connections}
C) UCSD, V = {students}, E = {classes}
D) Highway System, V = {intersections}, E = {roads}
E) A book, V = {words}

Examples of Graphs in CS

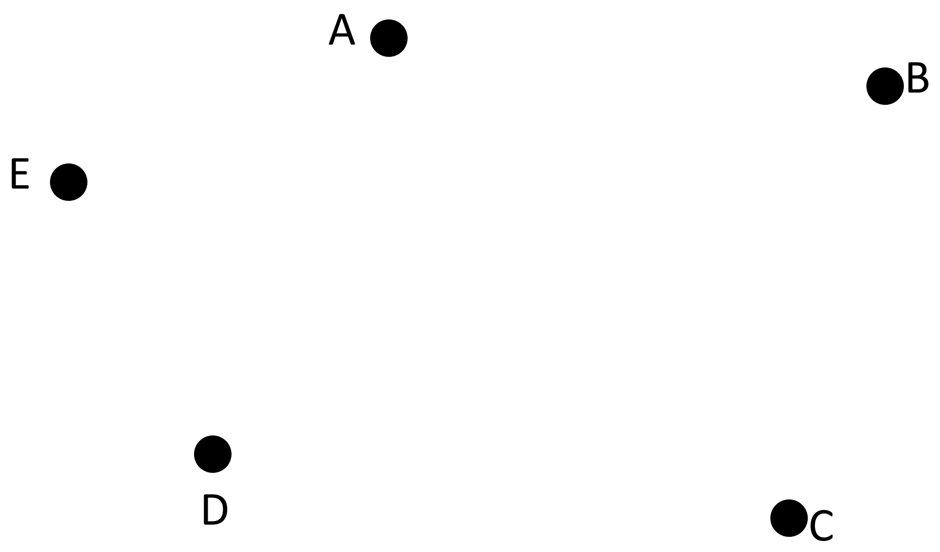
- The Internet (either webpages, or physical connections)
- Social Networks
- Transitions between states of a program
- Road maps

Drawing Graphs

- Draw vertices as points

Drawing Graphs

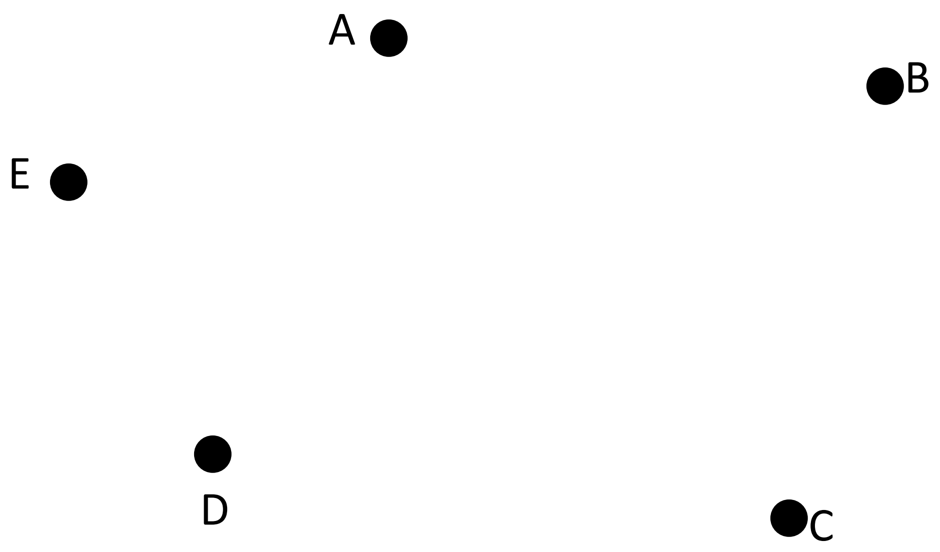
- Draw vertices as points



V = {A,B,C,D,E}

Drawing Graphs

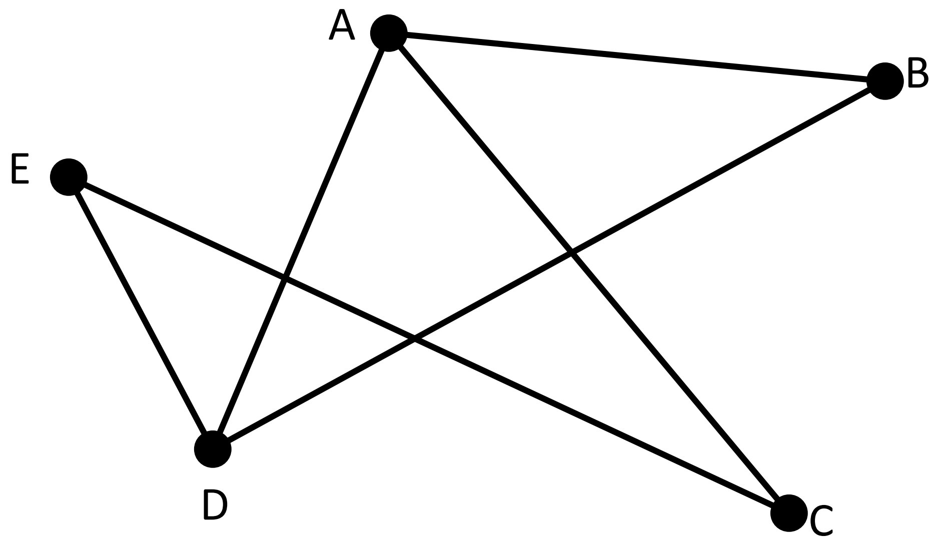
- Draw vertices as points
- Draw edges as line segments or curves connecting those points



$V = \{A,B,C,D,E\}$

Drawing Graphs

- Draw vertices as points
- Draw edges as line segments or curves connecting those points



V = {A,B,C,D,E}
E = {AB,AC,AD,
BD,CE,DE}

Exploring Graphs

You're playing a video game and want to make sure that you've found all the areas in this level before moving on to the next one.

Exploring Graphs

You're playing a video game and want to make sure that you've found all the areas in this level before moving on to the next one.

How do you ensure that you have found everything?

Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```

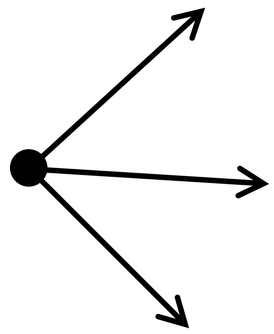
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```

•

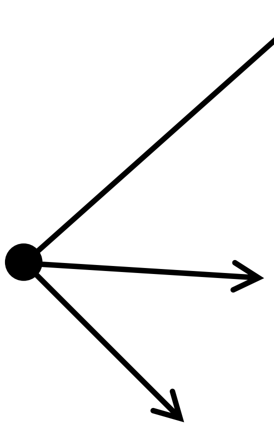
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



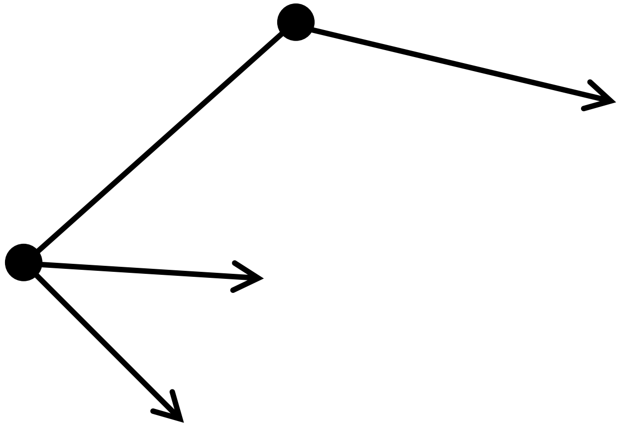
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



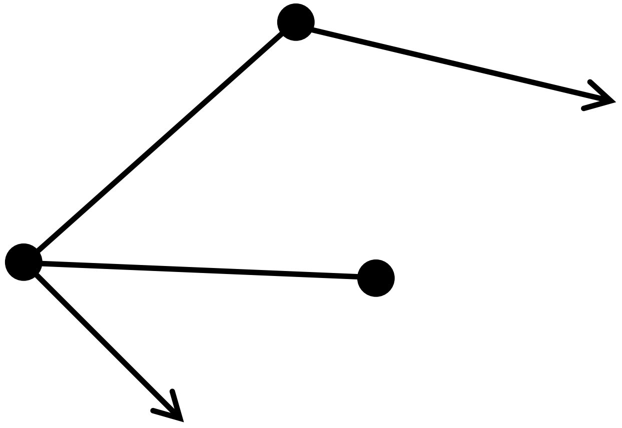
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



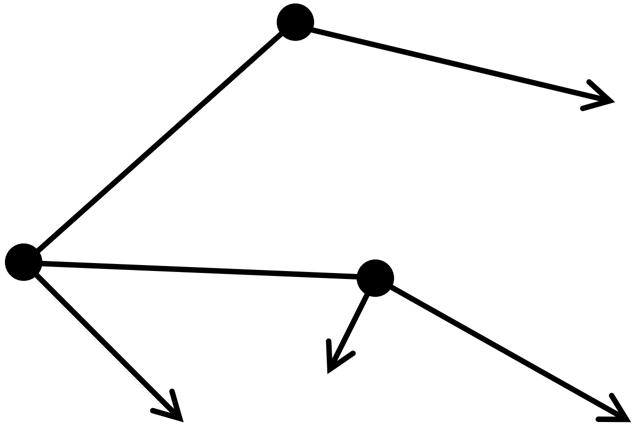
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



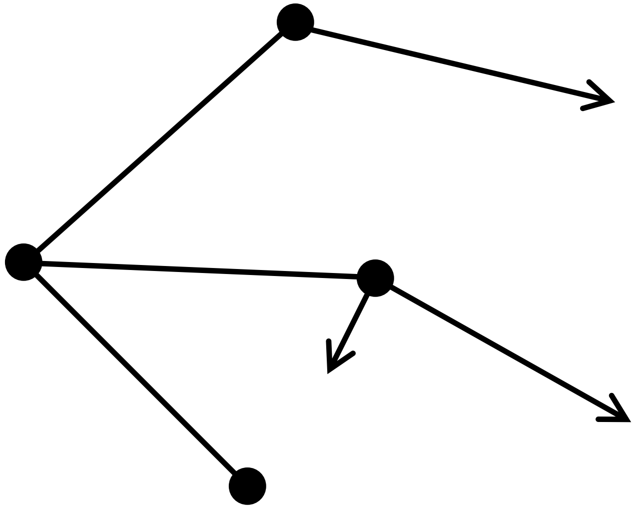
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



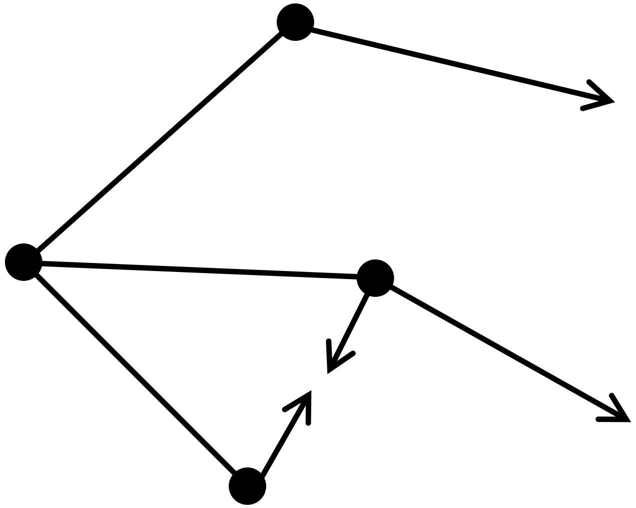
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



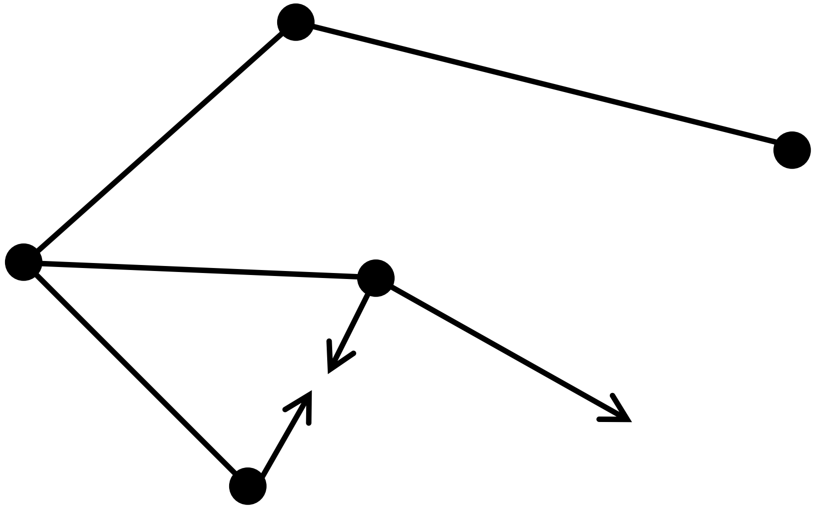
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



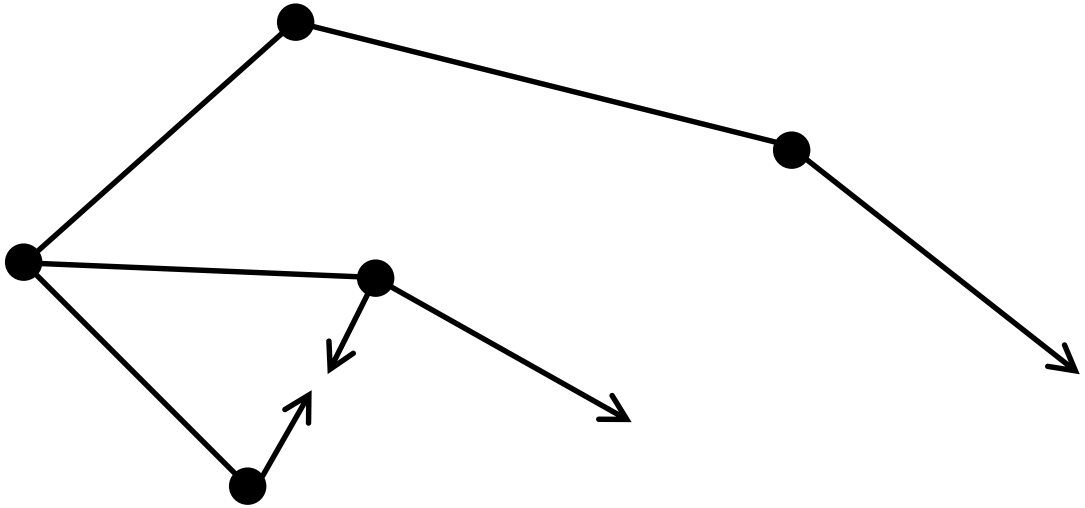
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



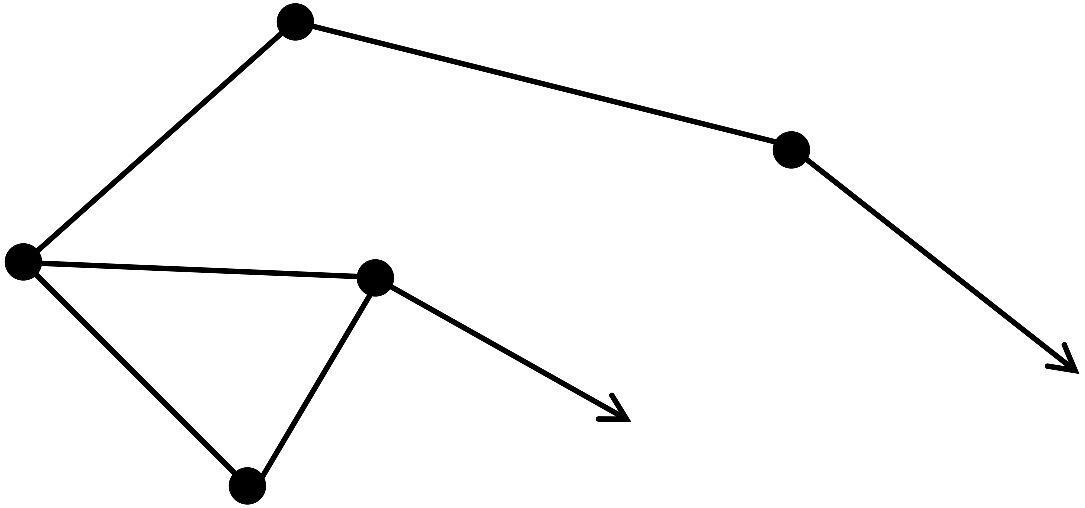
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



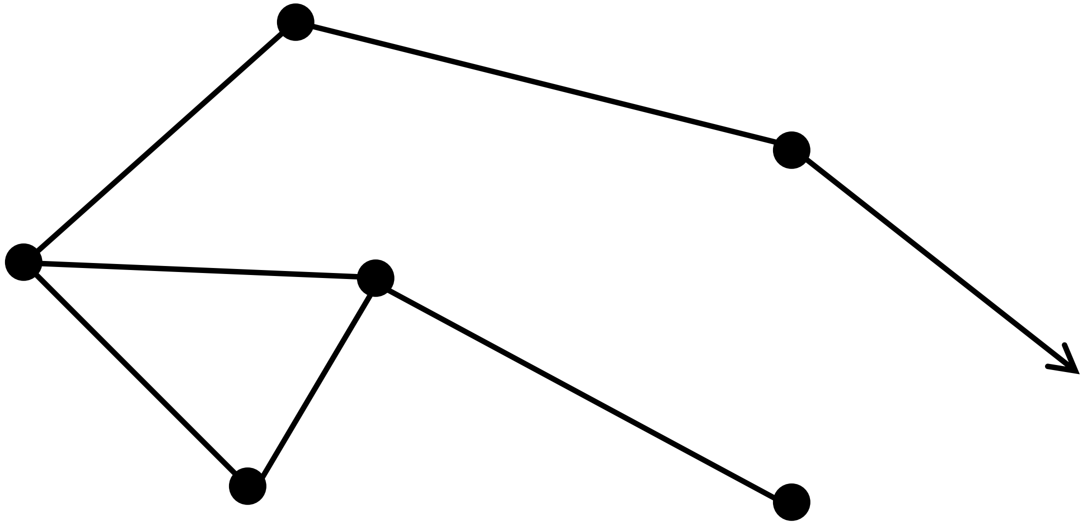
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



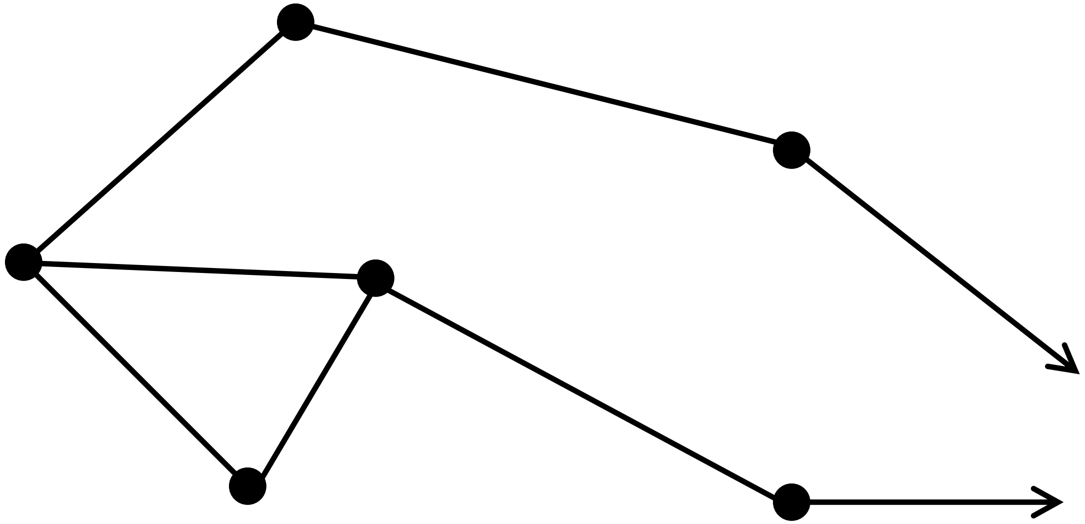
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



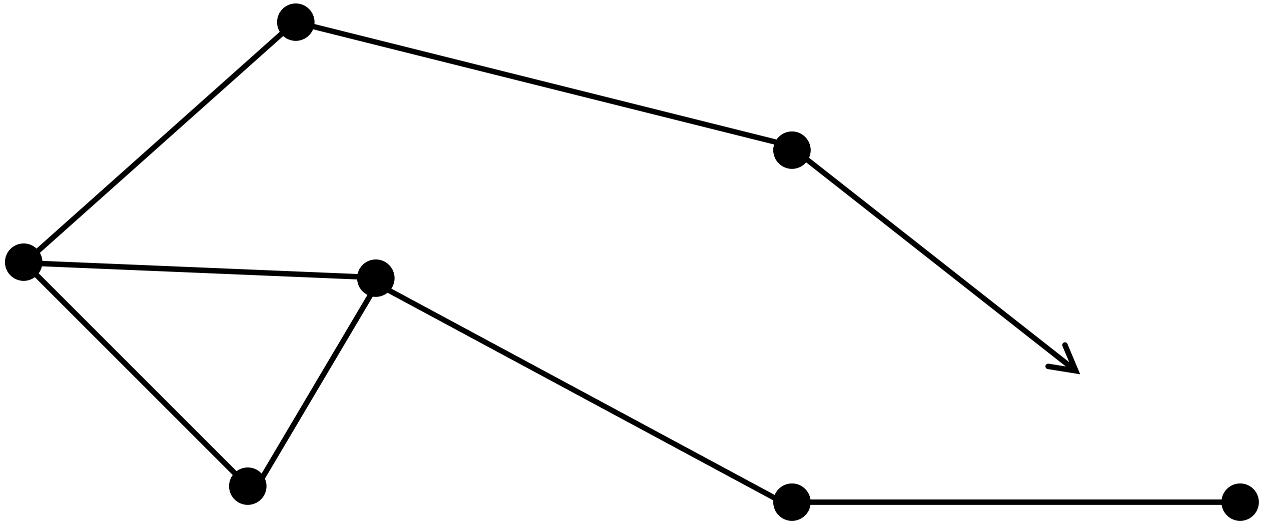
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



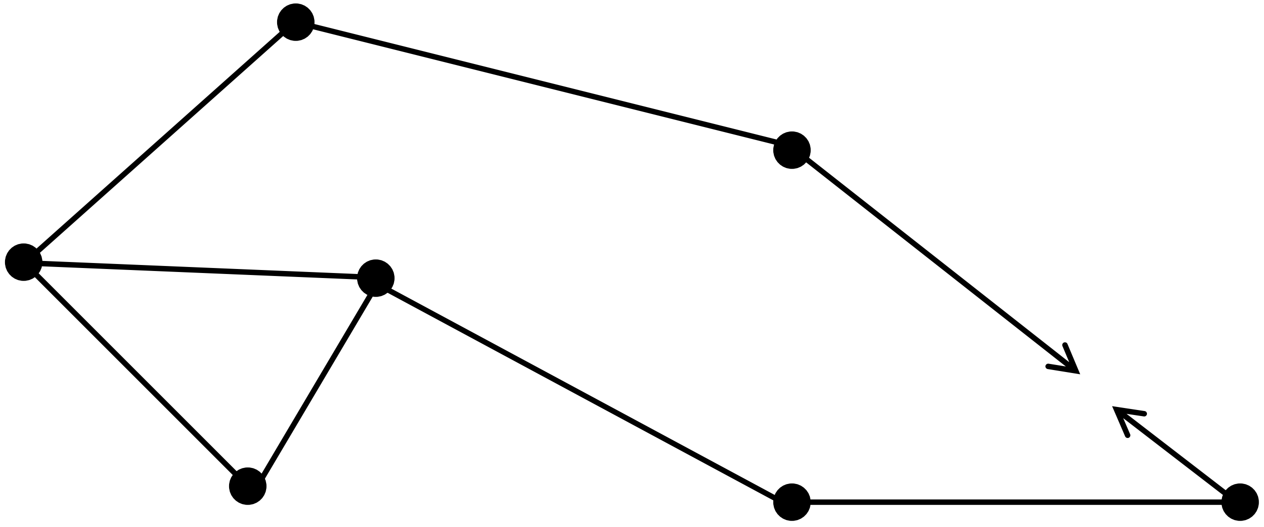
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



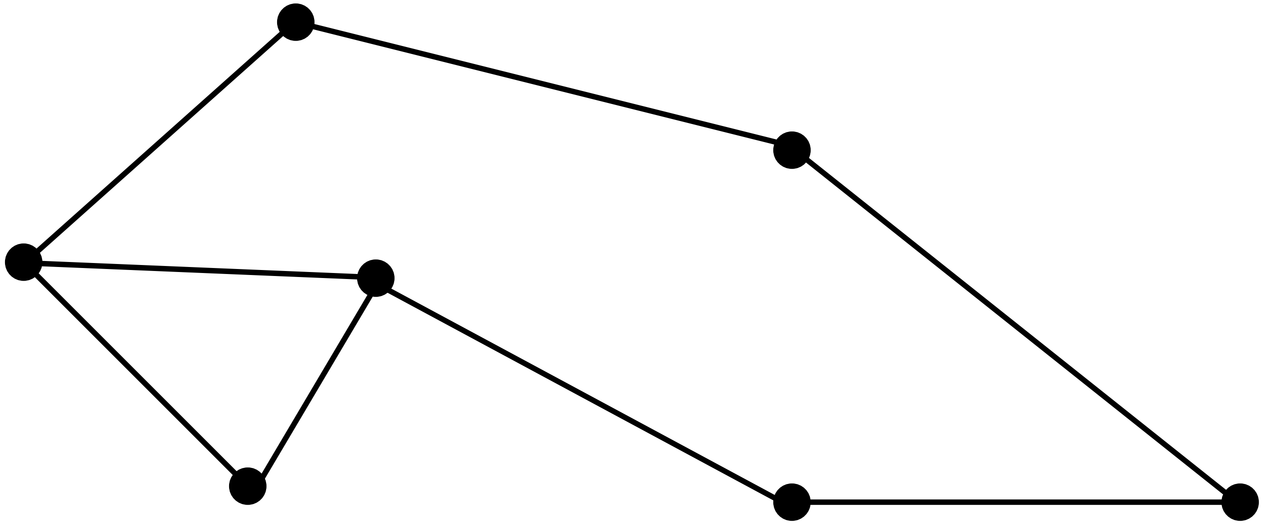
Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



Basic Algorithm

```
Keep track of all areas discovered
While there is an unexplored path,
  follow path
```



Systematize

Need to keep track of:

- Which vertices discovered
- Which edges have yet to be explored

Systematize

Need to keep track of:

- Which vertices discovered
- Which edges have yet to be explored

`explore` Algorithm will:

- Use a field `v.visited` to let us know which vertices we have seen.
- Store edges to be explored implicitly in the program stack.

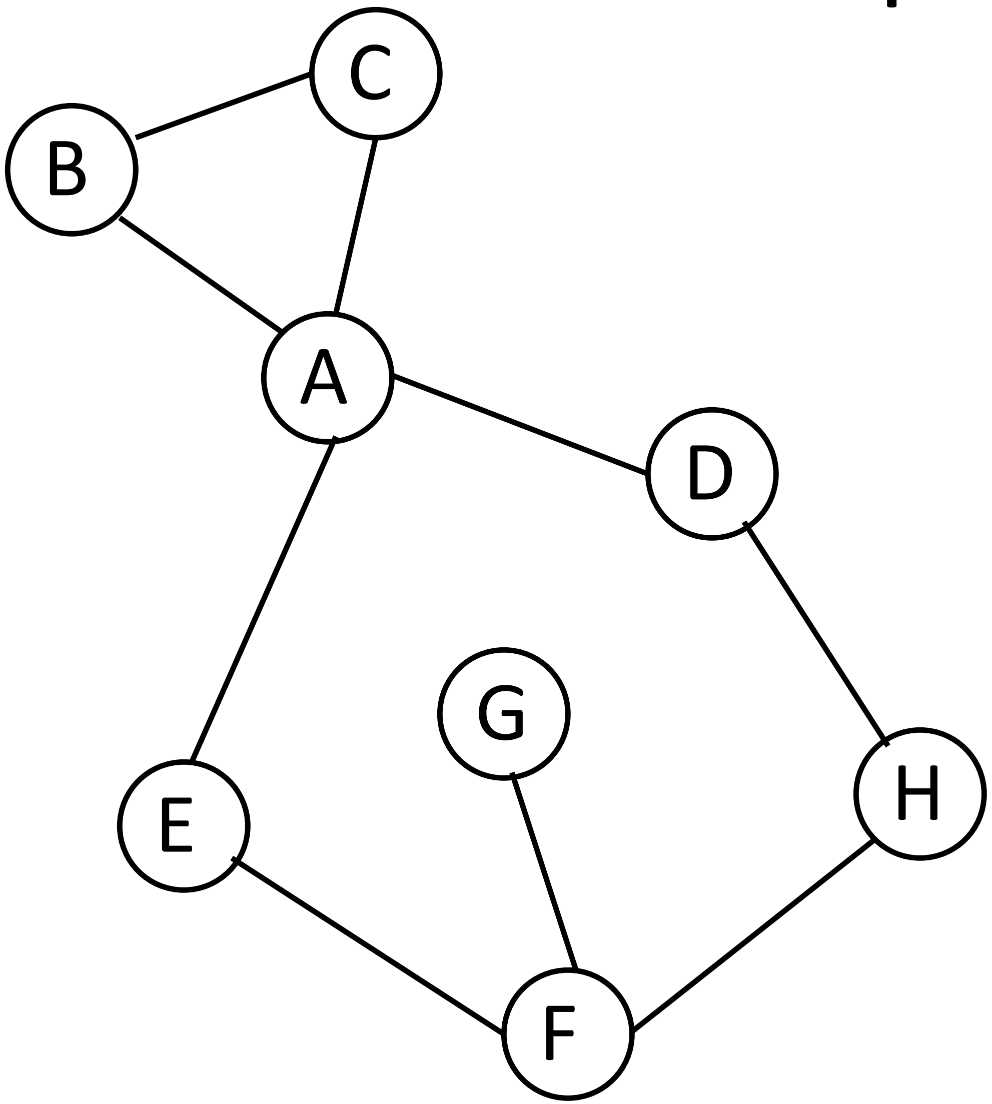
Explore

```
explore(v)
  v.visited ← true
  For each edge (v,w)
    If not w.visited
      explore(w)
```

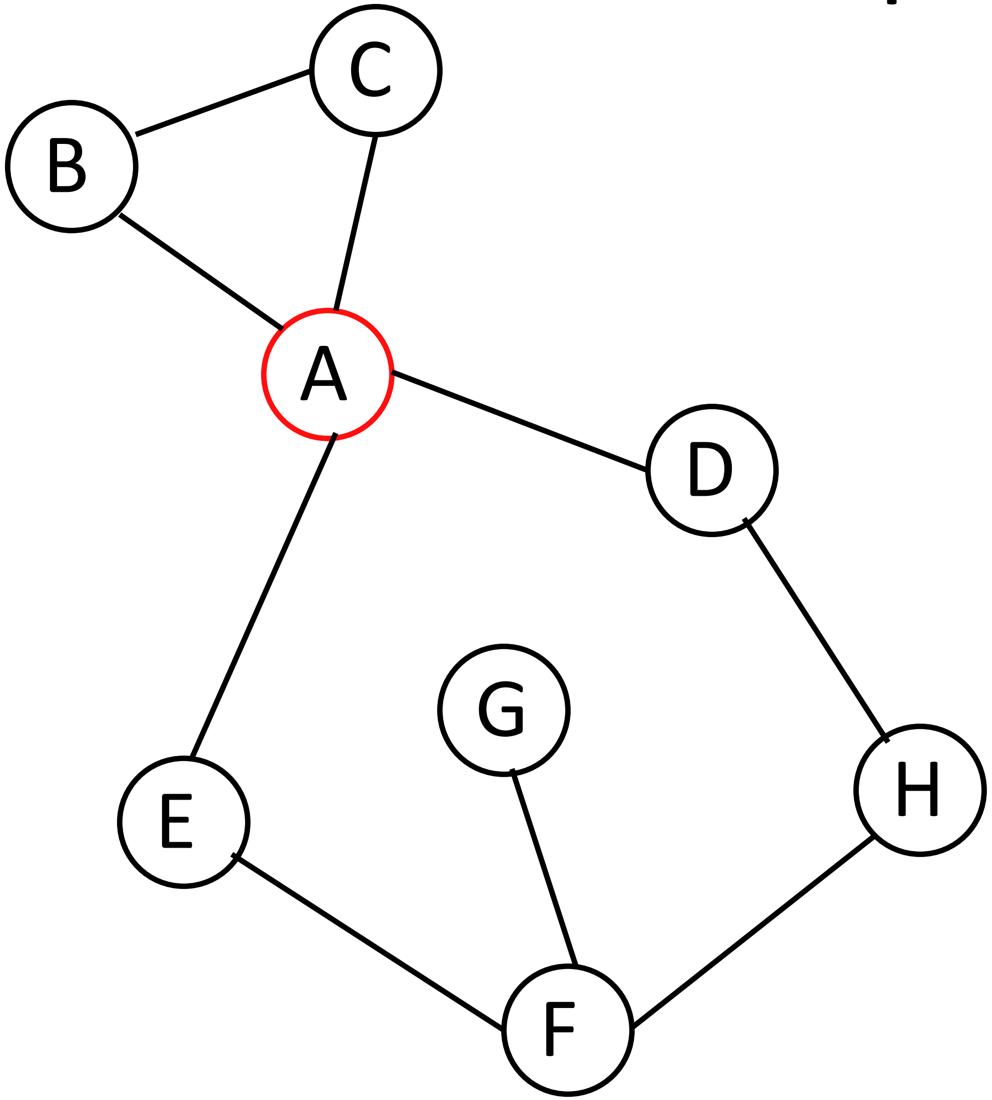
Explore

```
explore(v)
  v.visited ← true
  For each edge (v,w)
    If not w.visited
      explore(w)
      w.prev ← v
```

Example

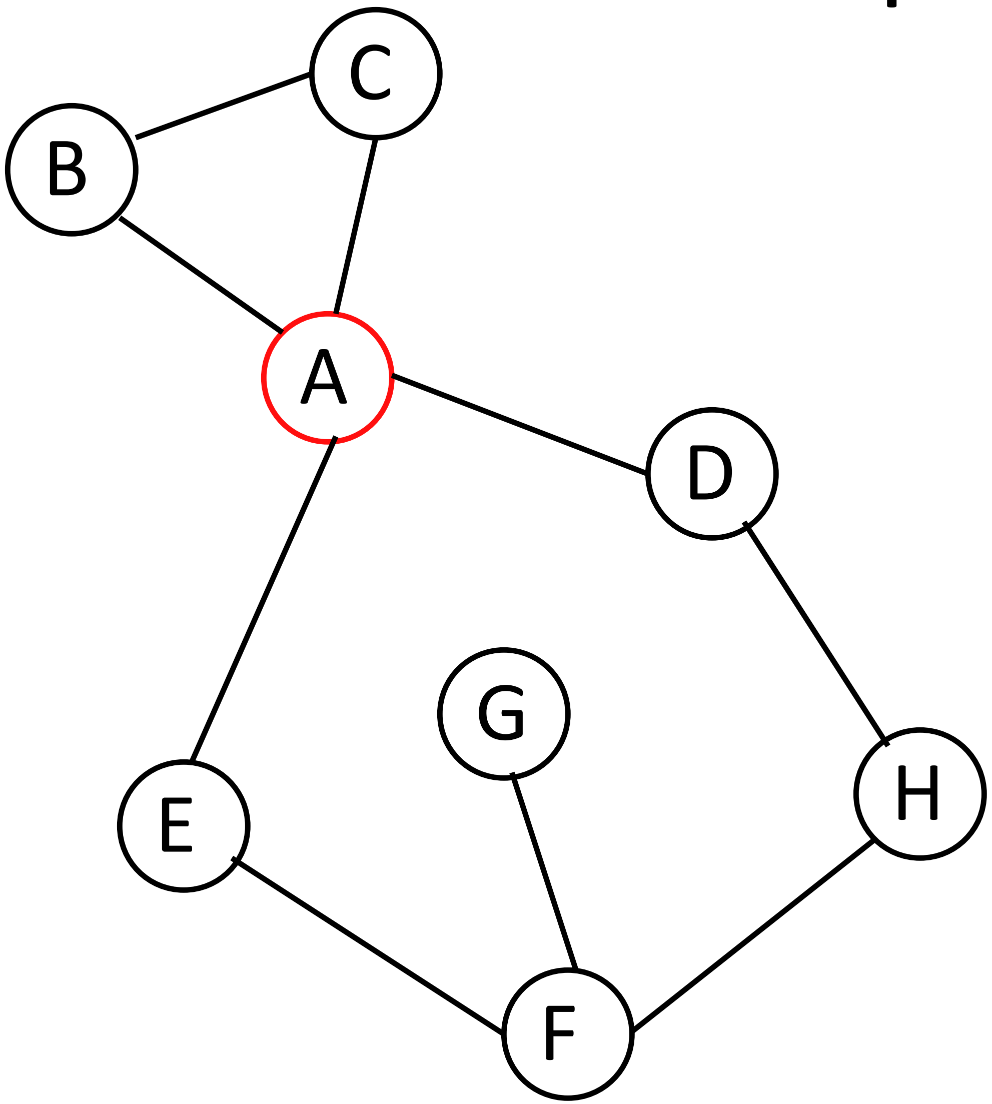


Example



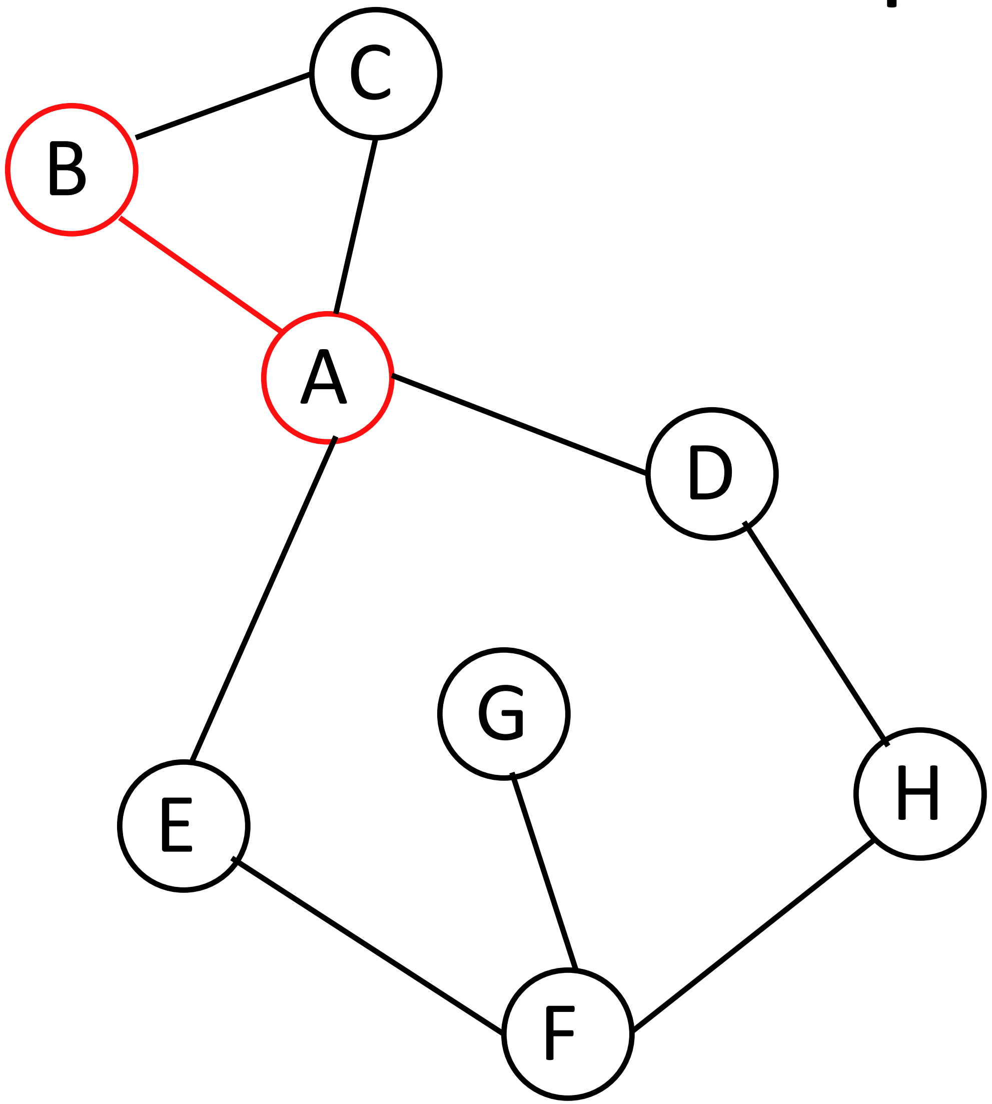
explore(A)

Example



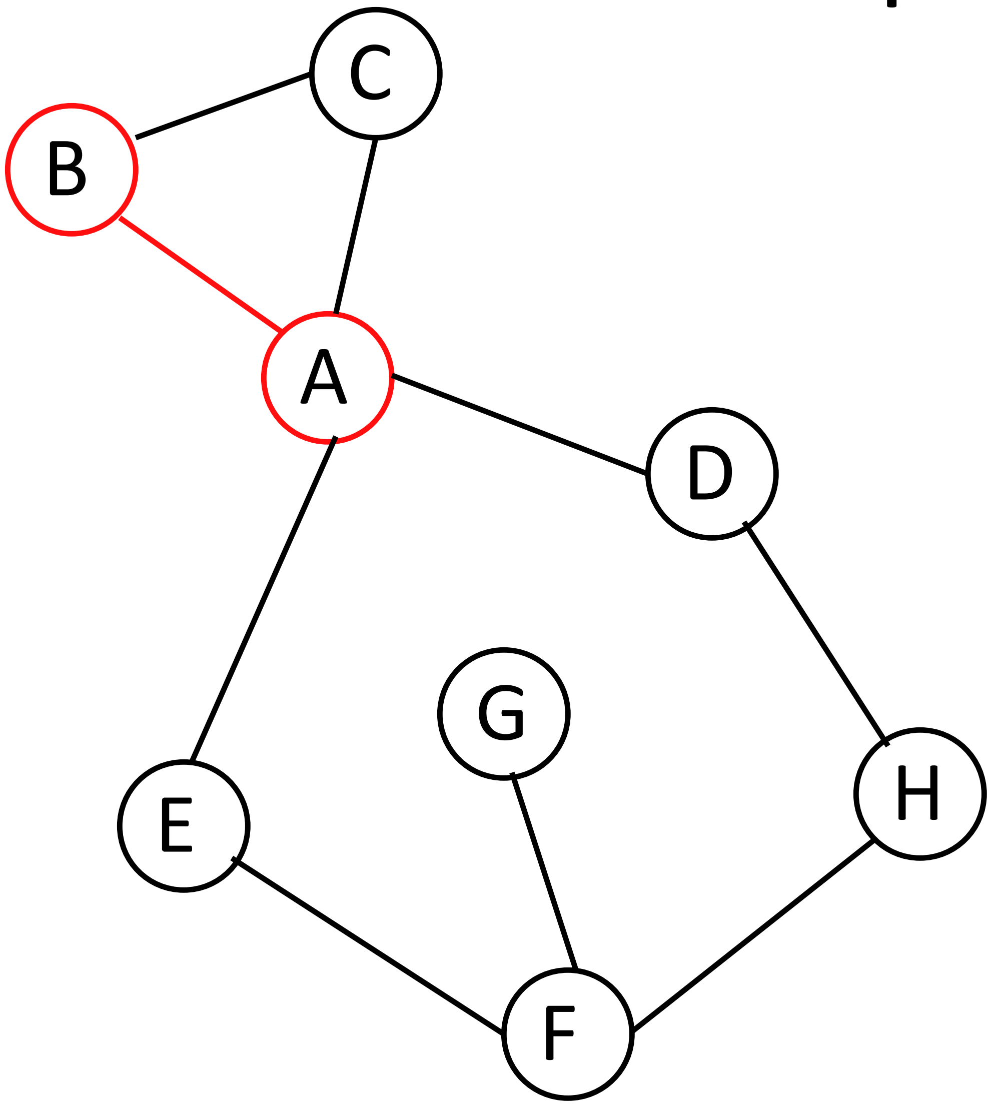

```
explore(A)
   explore(B)



   explore(C)
   explore(D)




   explore(E)
```

Example



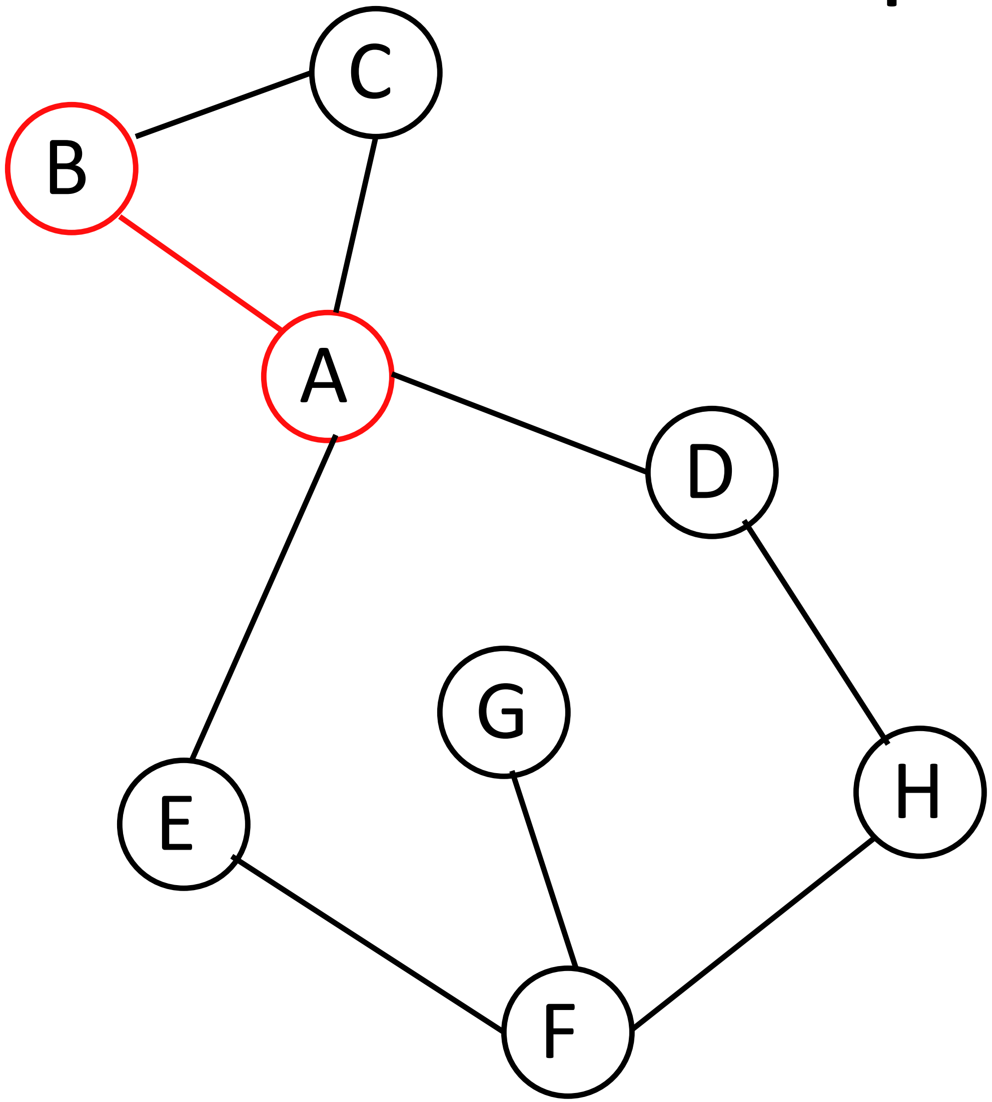

```
explore(A)
  explore(B)



  explore(C)
  explore(D)





  explore(E)
```

Example



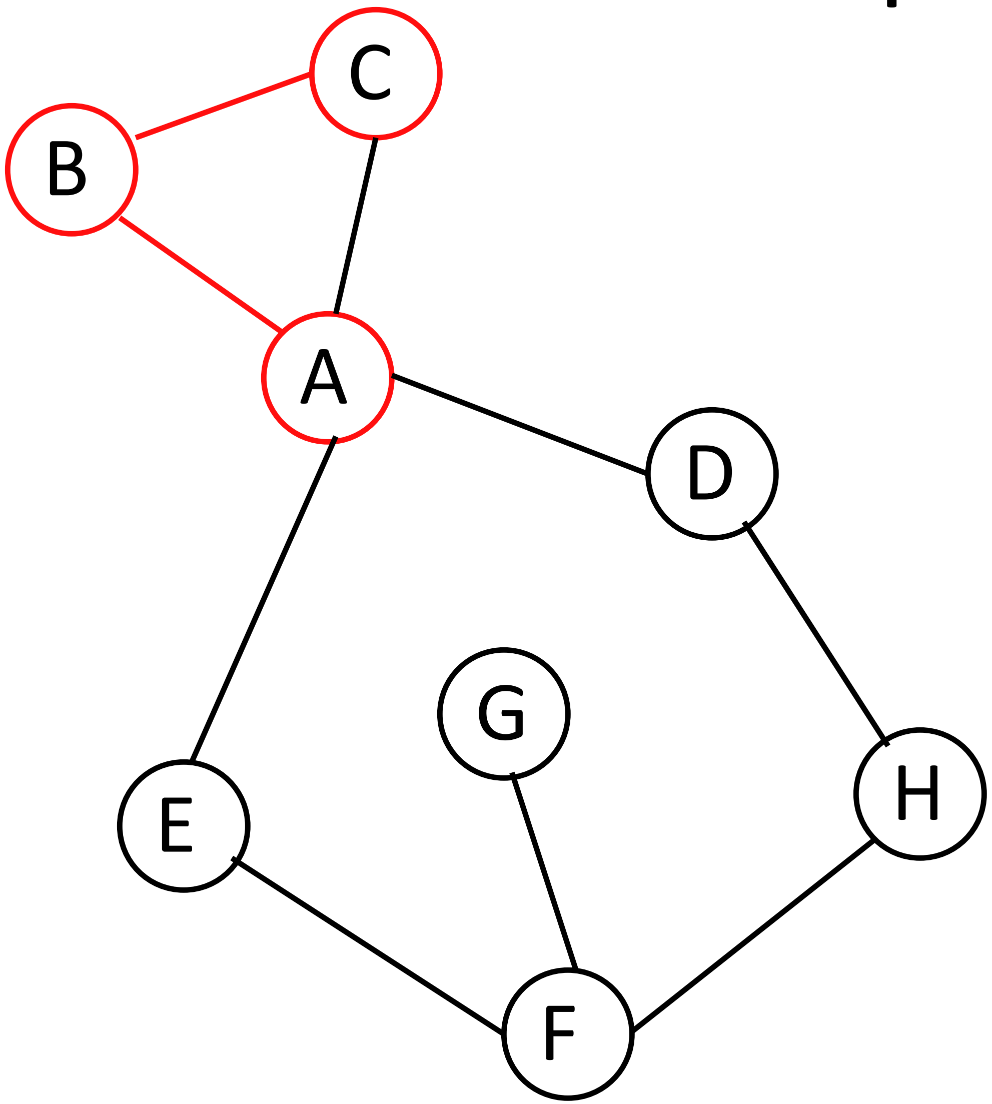

```
explore(A)
  explore(B)
    explore(A)
    explore(C)


  explore(C)
  explore(D)







  explore(E)
```

Example



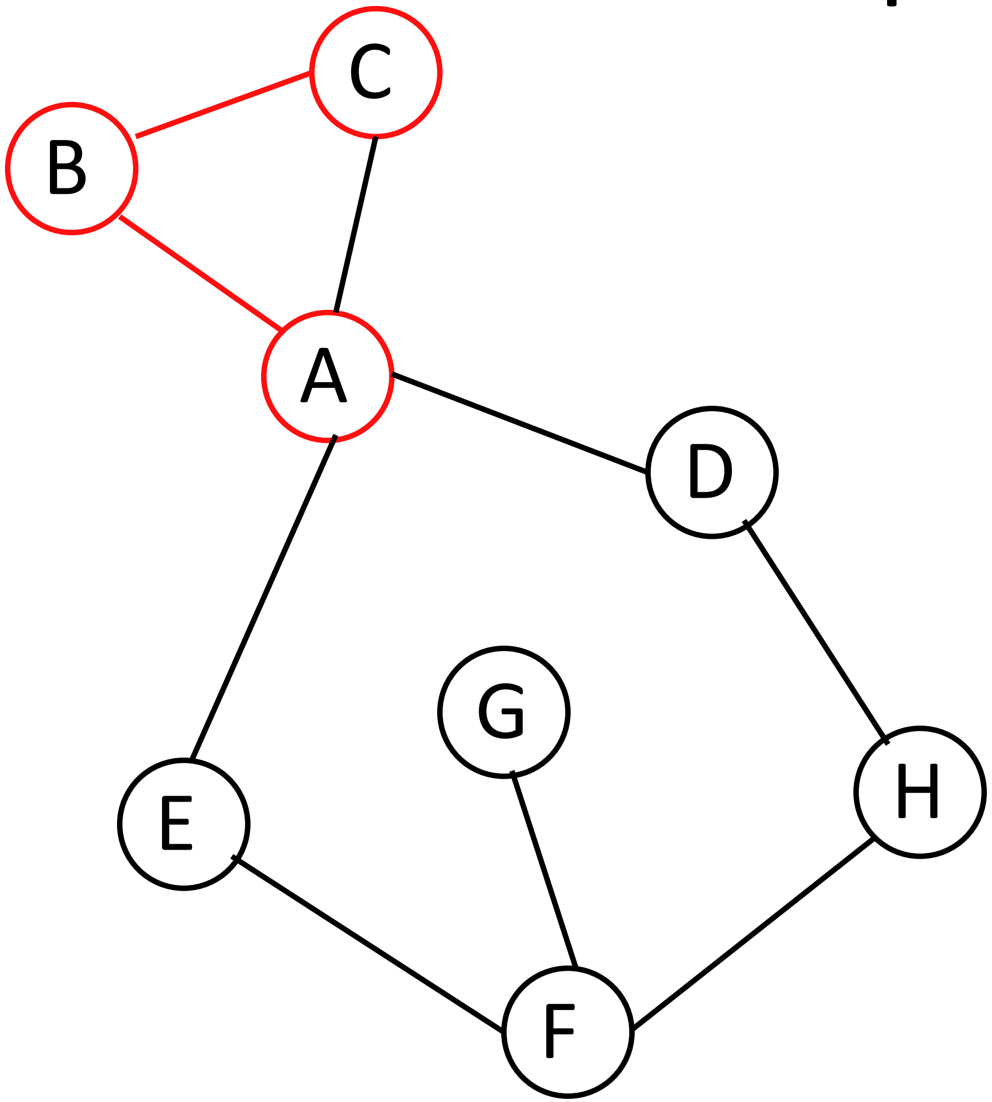

```
explore(A)
   explore(B)
      explore(A)
      explore(C)



   explore(C)
   explore(D)











   explore(E)
```

Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)


  explore(C)
  explore(D)





  explore(E)
```

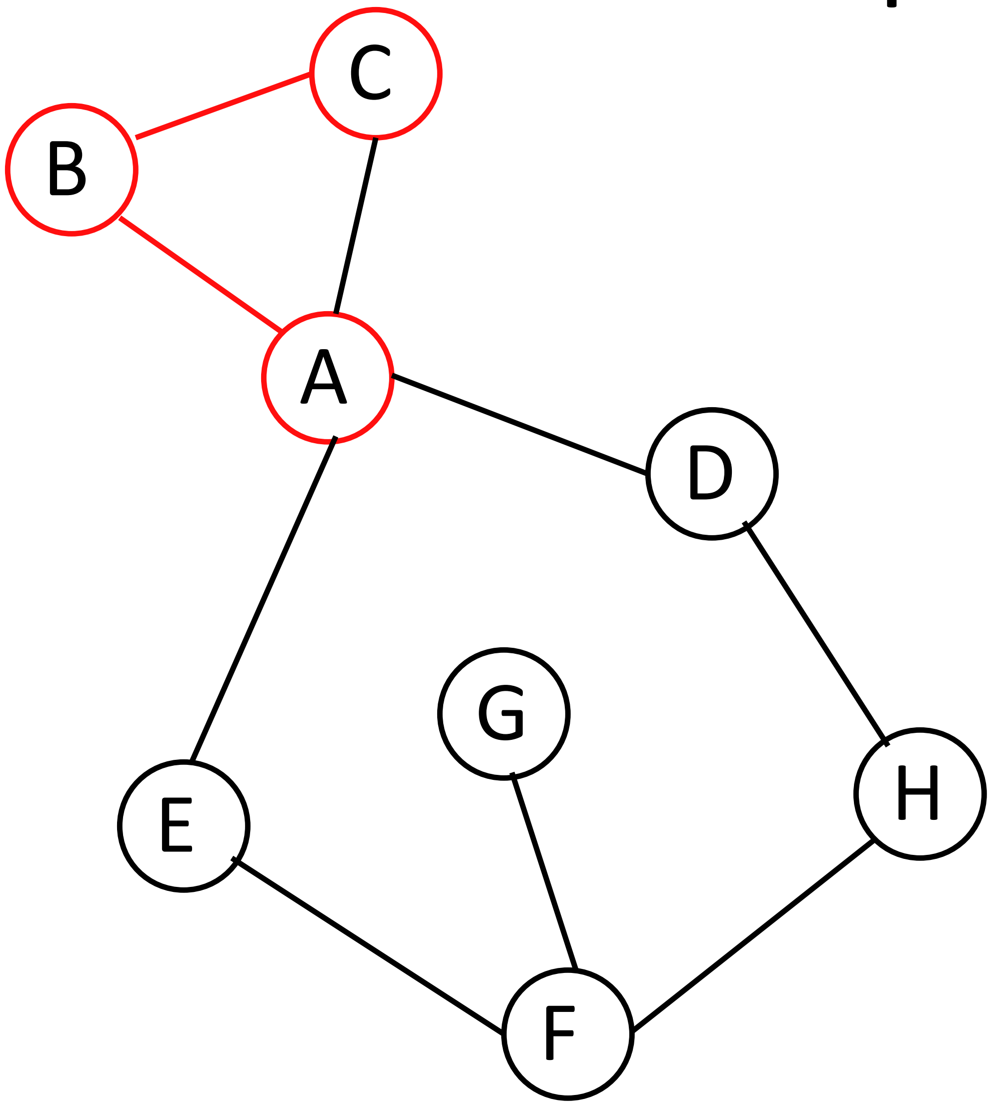
Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
```

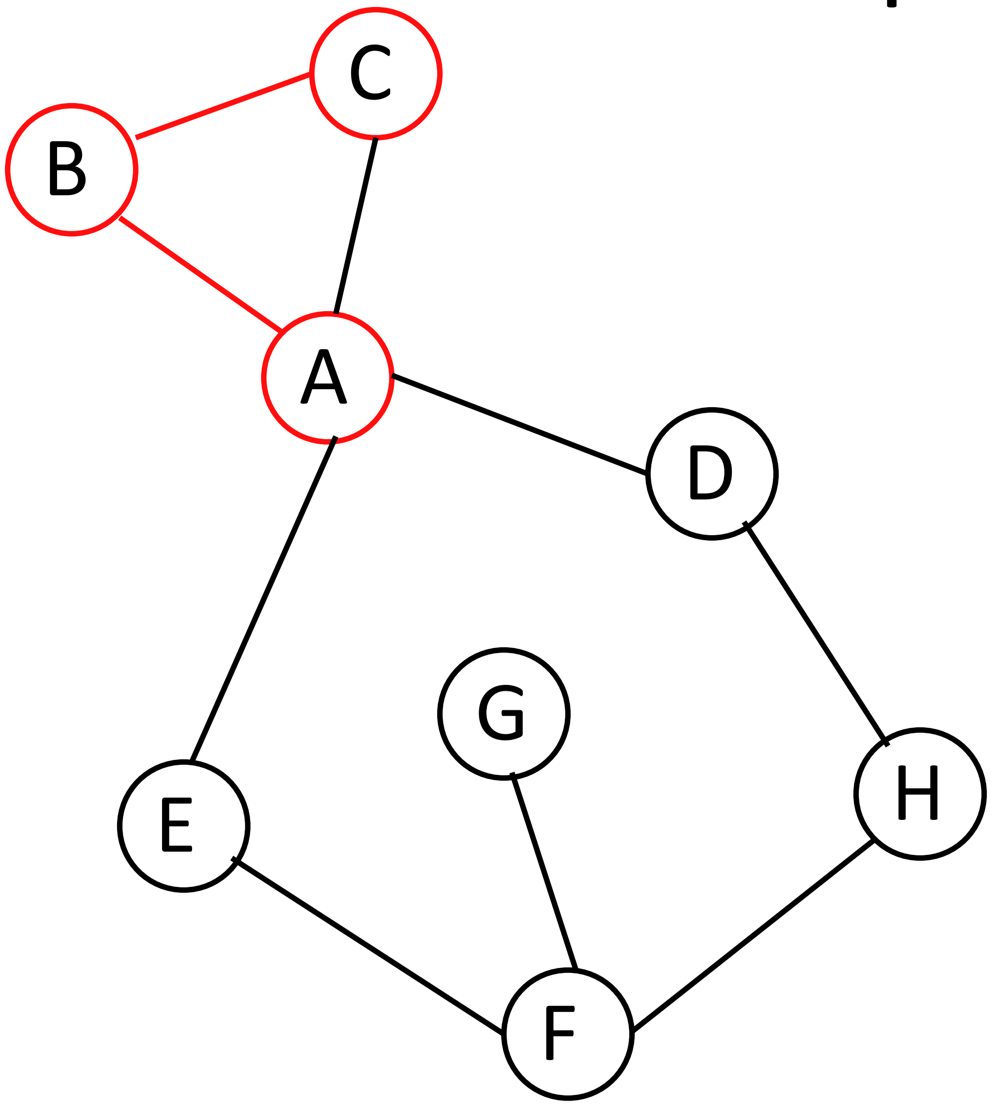
```
  explore(E)
```

Example



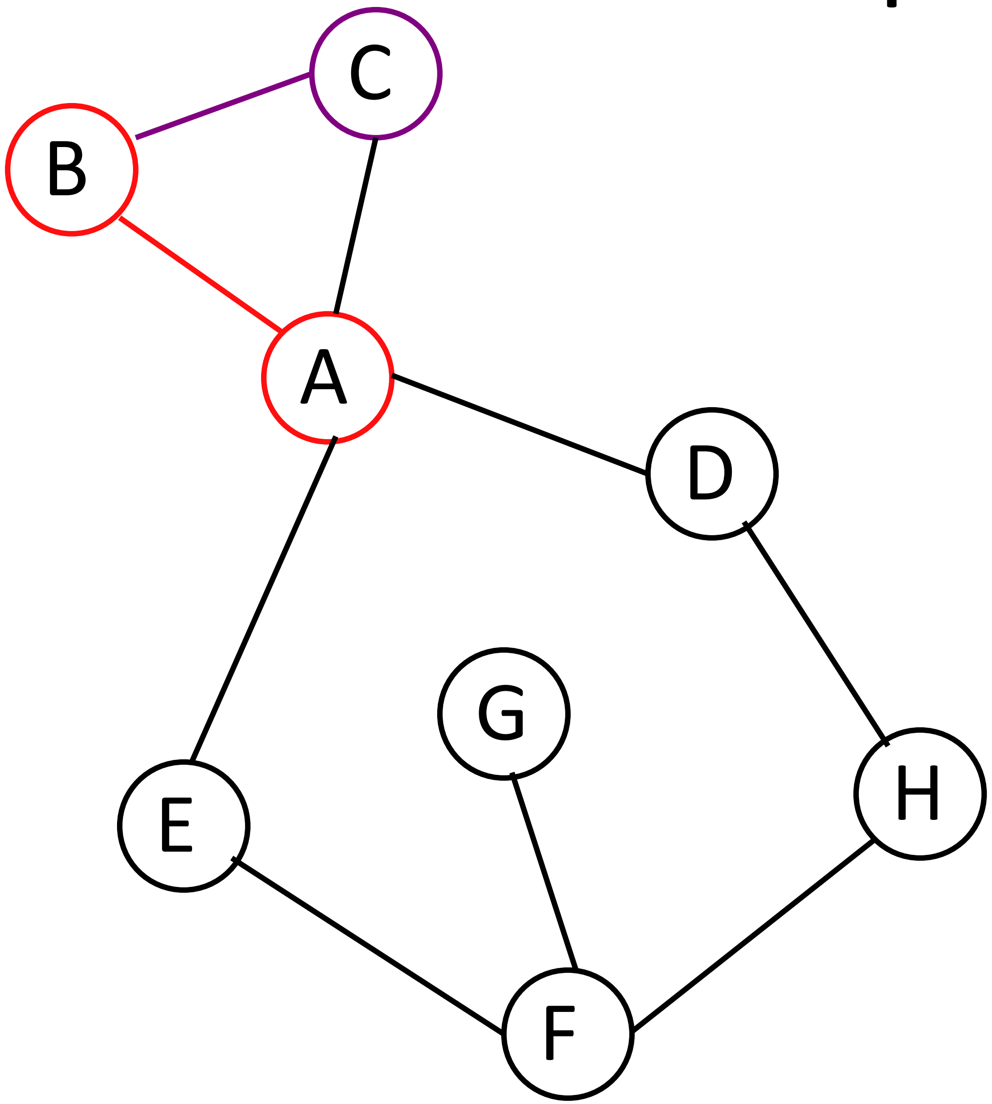

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)

  explore(E)
```

Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)





  explore(E)
```

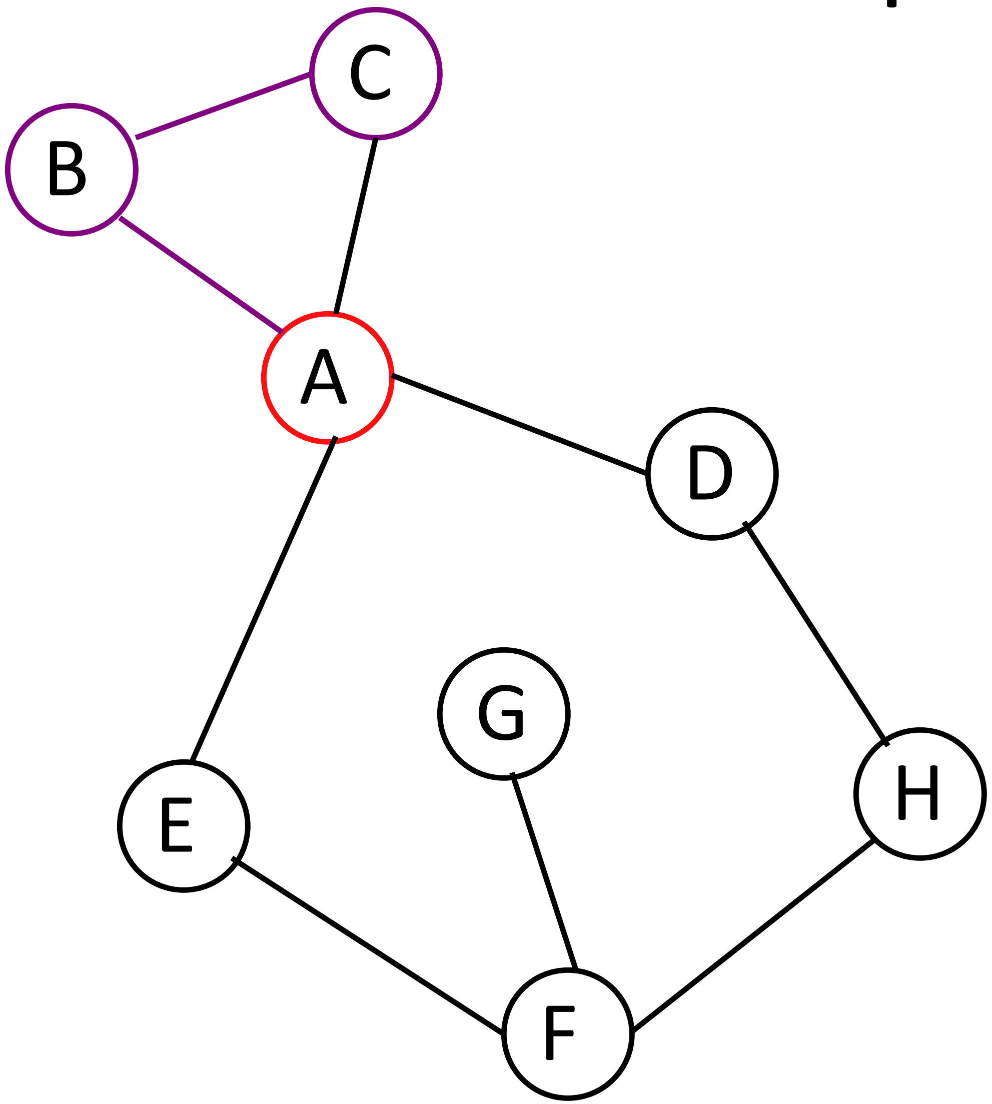
Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
```

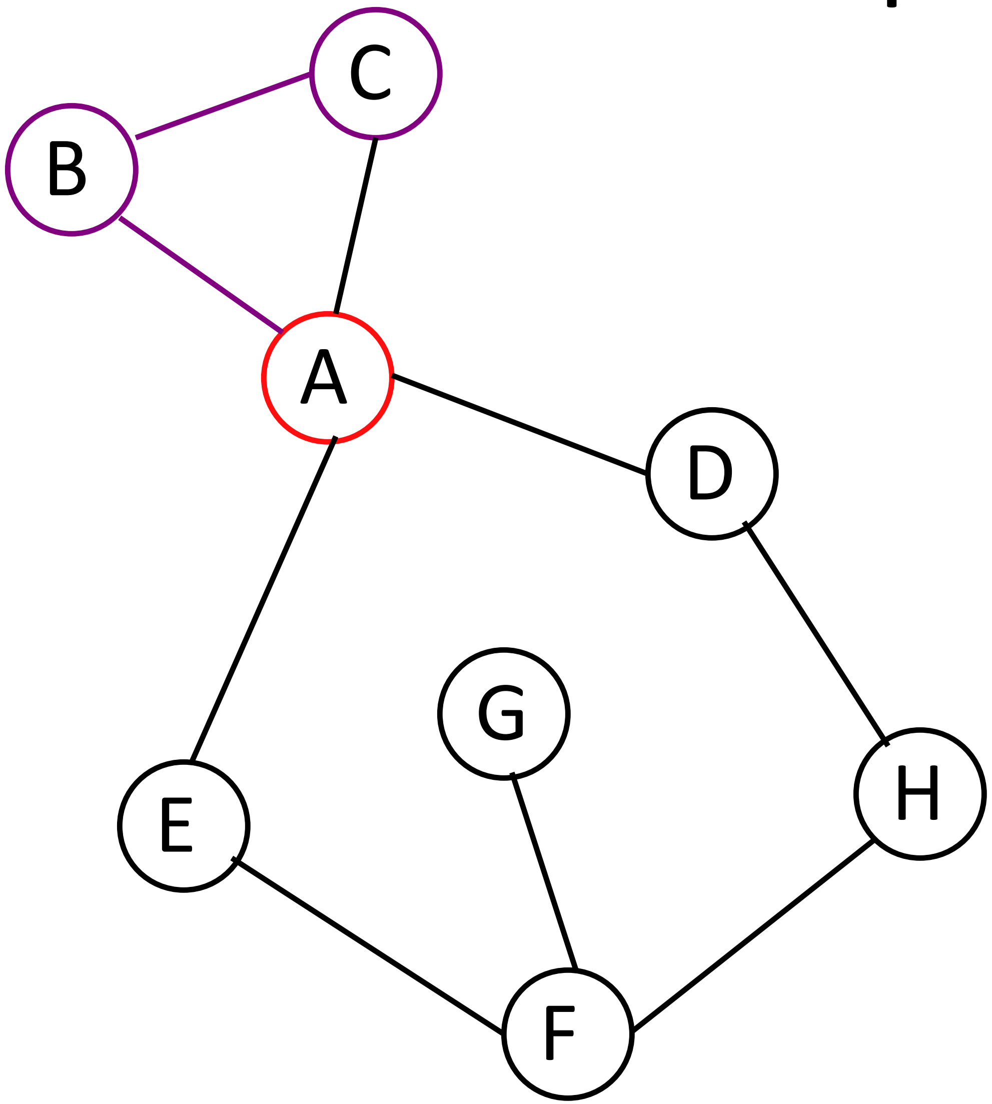
```
  explore(E)
```

Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)

  explore(E)
```

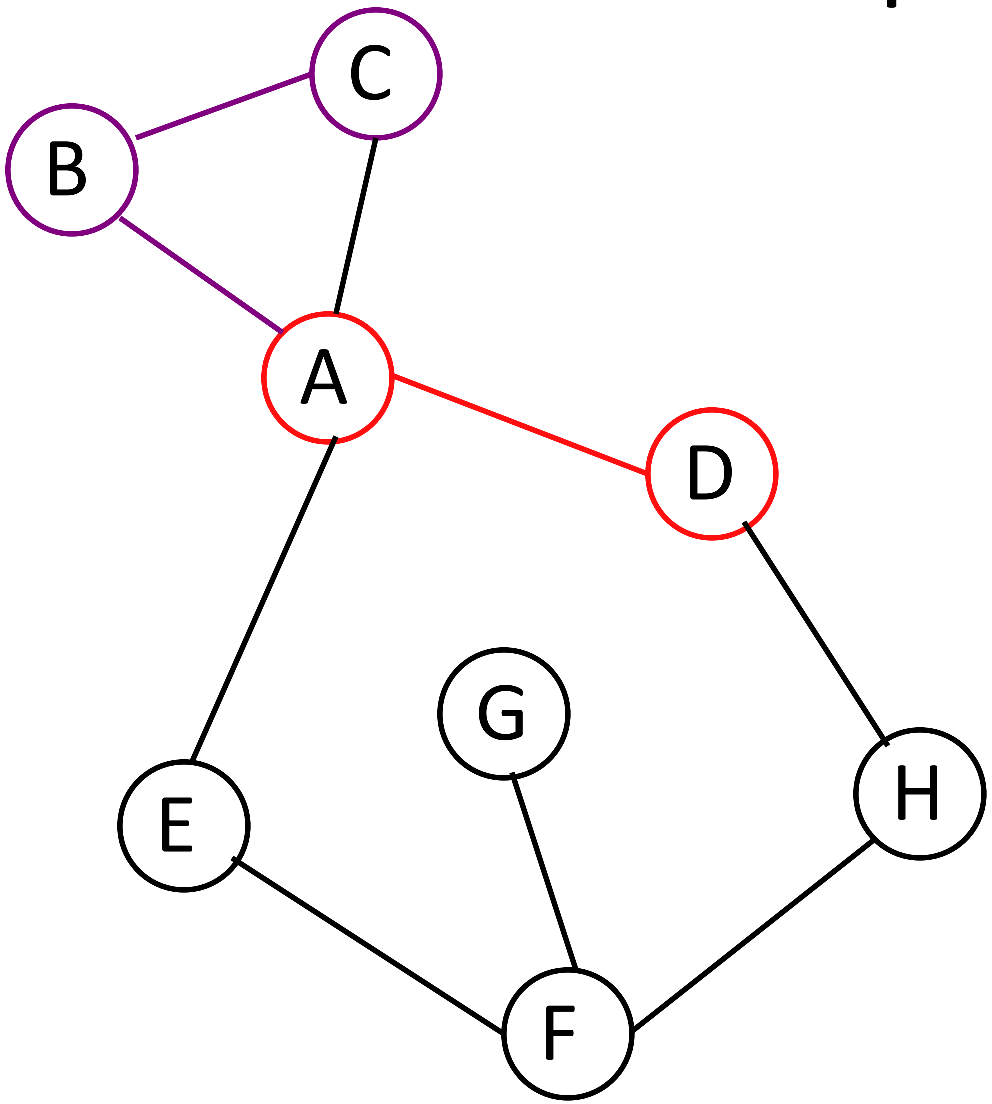
Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
```

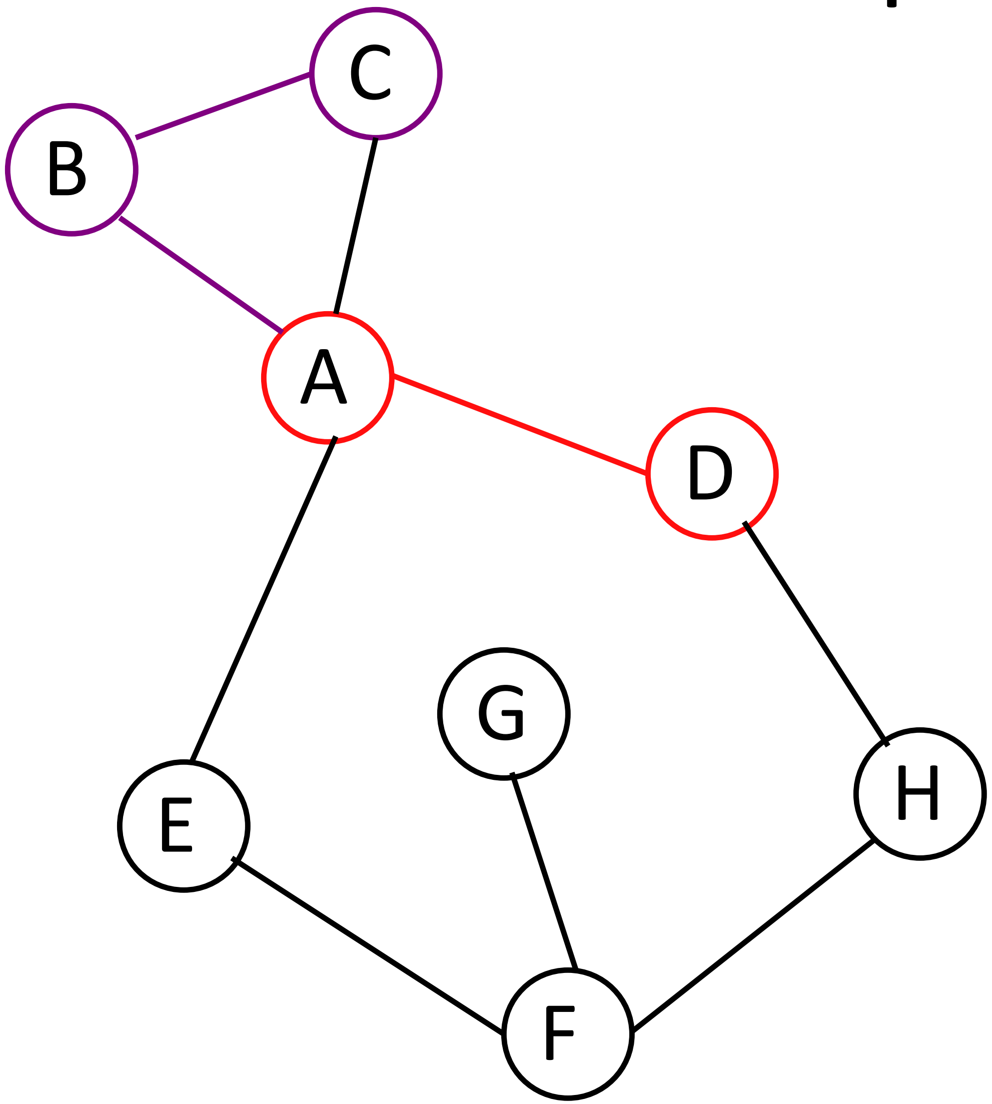
```
  explore(E)
```

Example



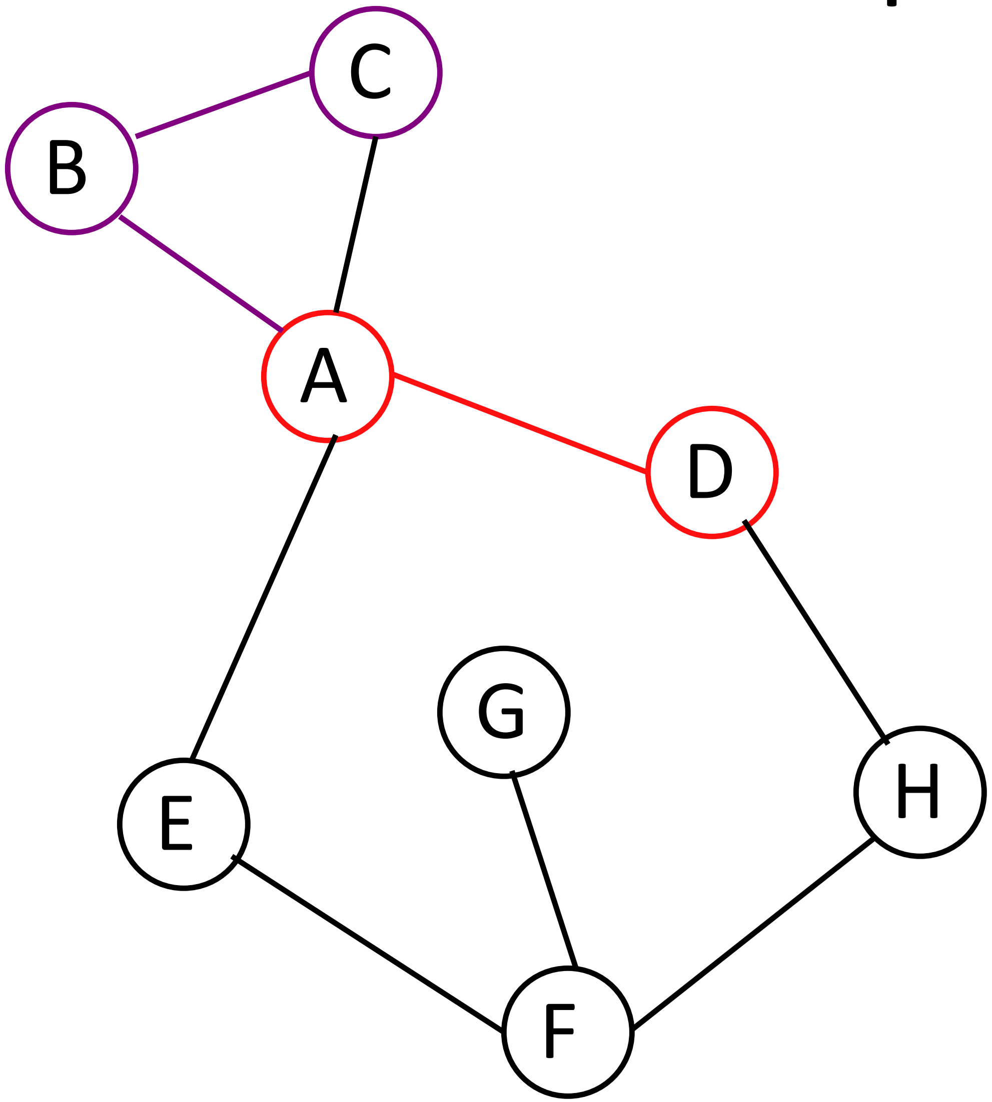

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)




  explore(E)
```

Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)



  explore(E)
```

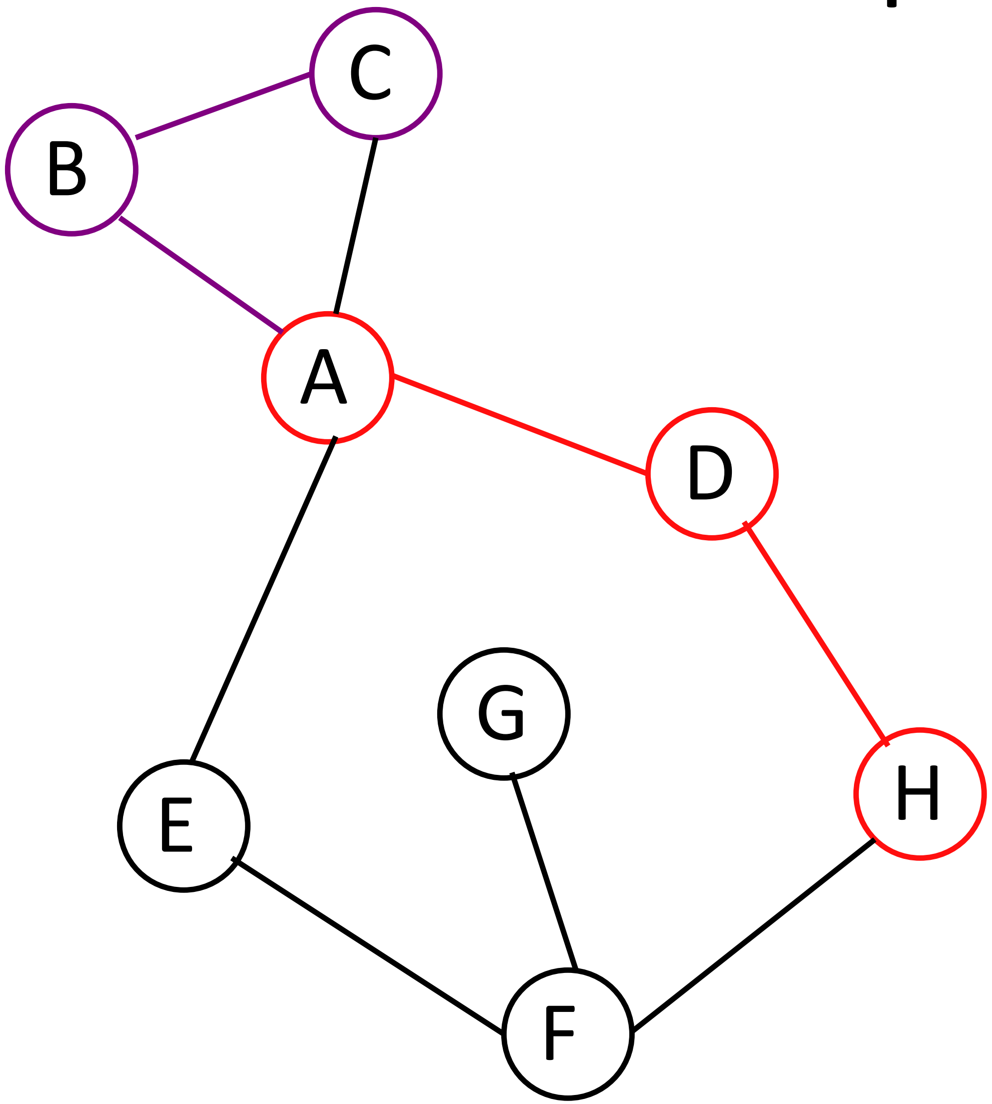
Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
```

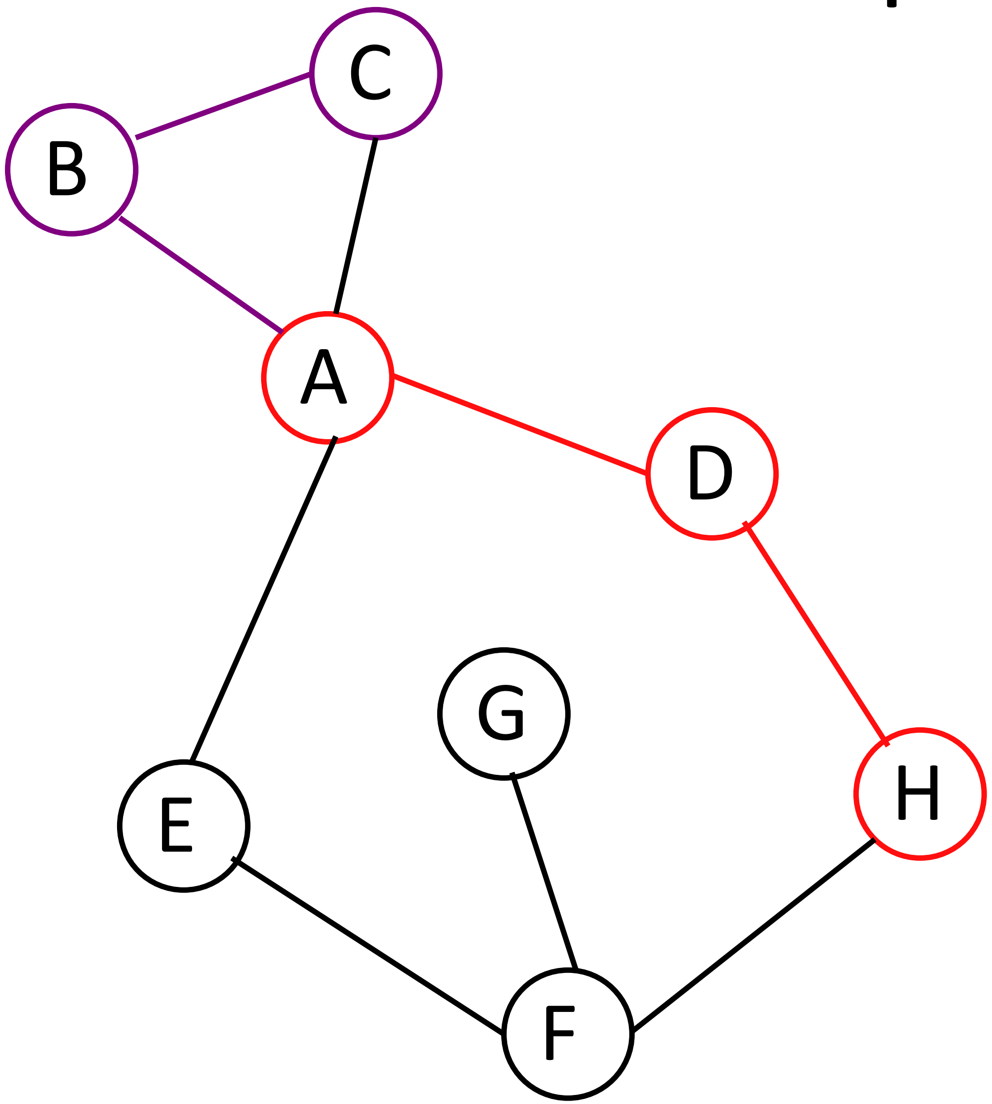
```
  explore(E)
```

Example



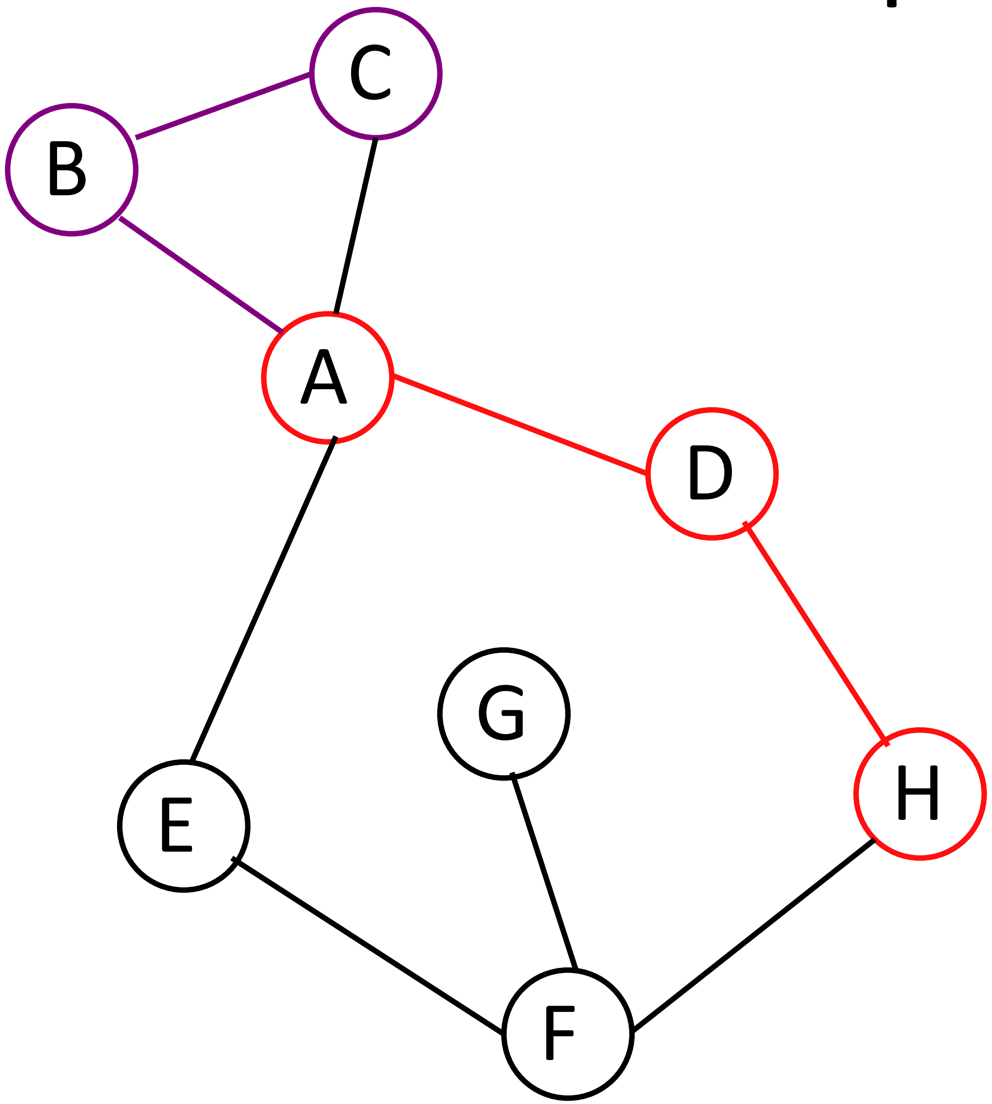

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)








  explore(E)
```

Example



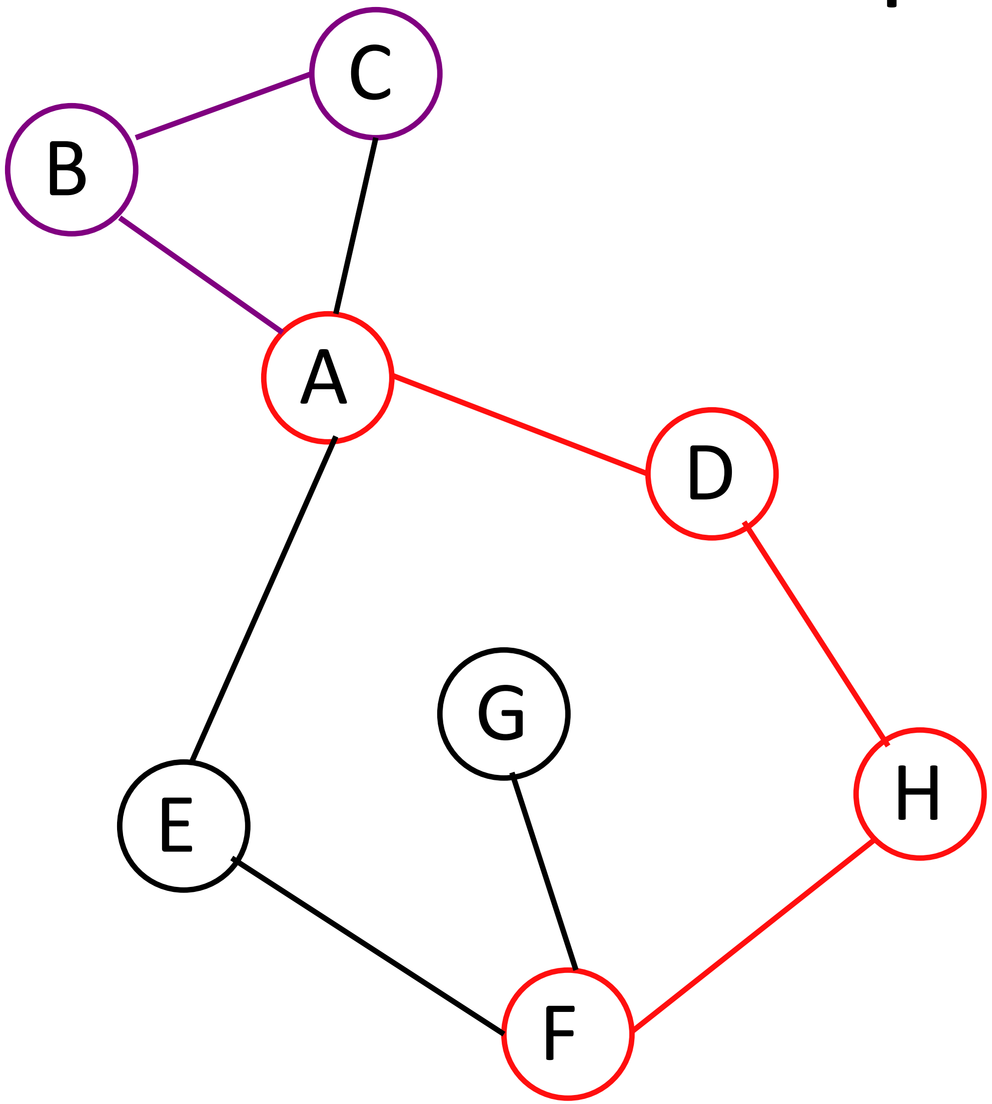

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)




  explore(E)
```

Example



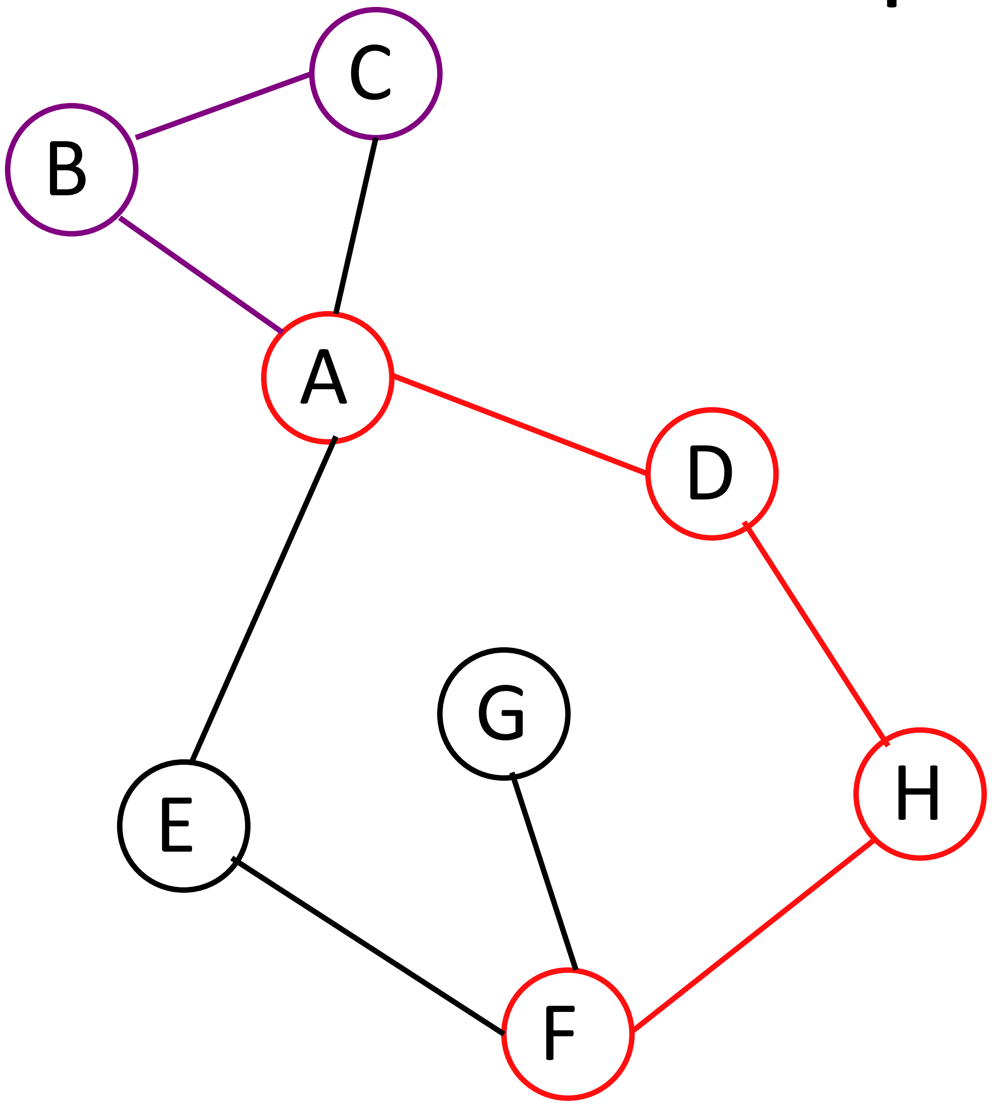

```
explore(A)
   explore(B)
      explore(A)
      explore(C)
         explore(A)
         explore(B)
   explore(C)
   explore(D)
      explore(A)
      explore(H)
         explore(D)
         explore(F)




   explore(E)
```

Example



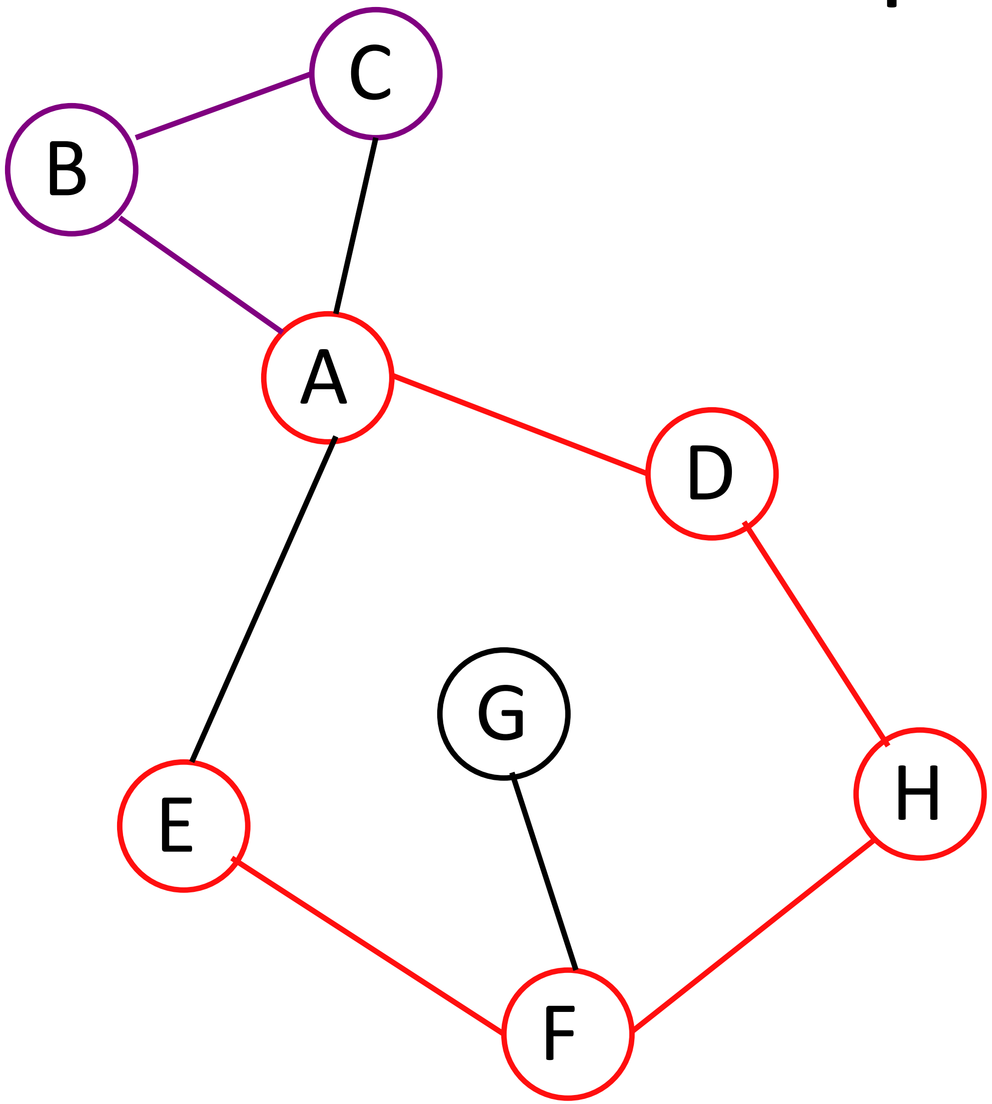

```
explore(A)
   explore(B)
      explore(A)
      explore(C)
         explore(A)
         explore(B)
   explore(C)
   explore(D)
      explore(A)
      explore(H)
         explore(D)
         explore(F)


   explore(E)
```

Example



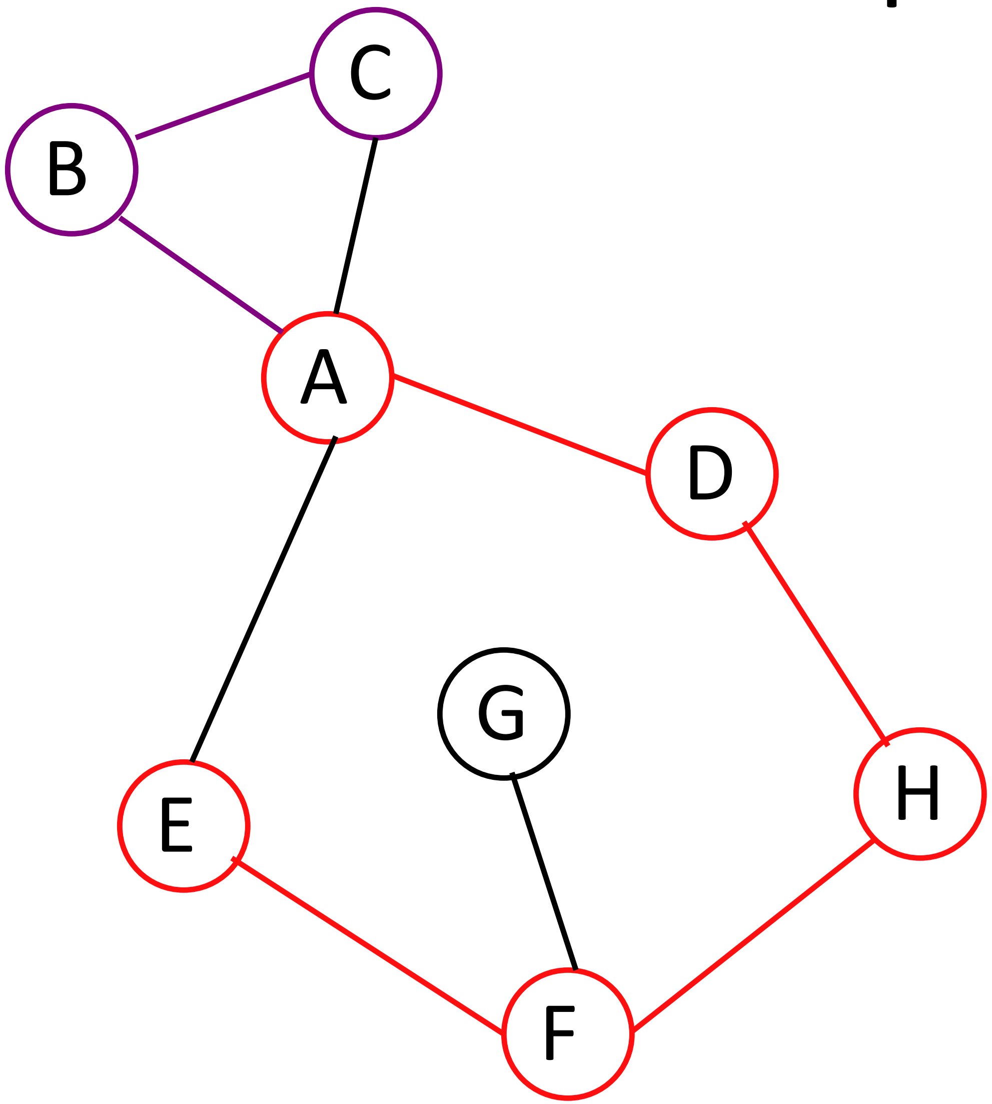

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)


        explore(G)

        explore(H)
  explore(E)
```

Example



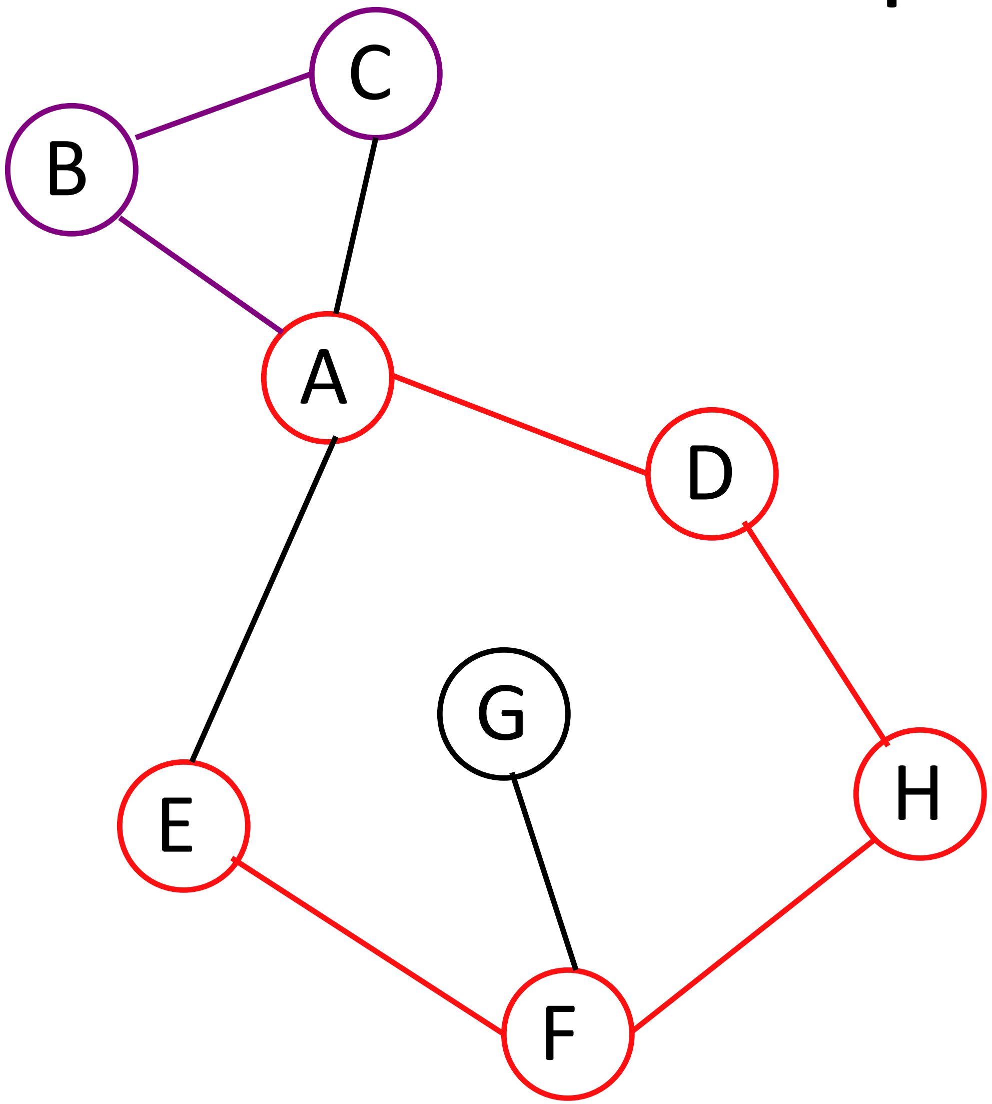

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)



        explore(G)

        explore(H)
  explore(E)
```

Example



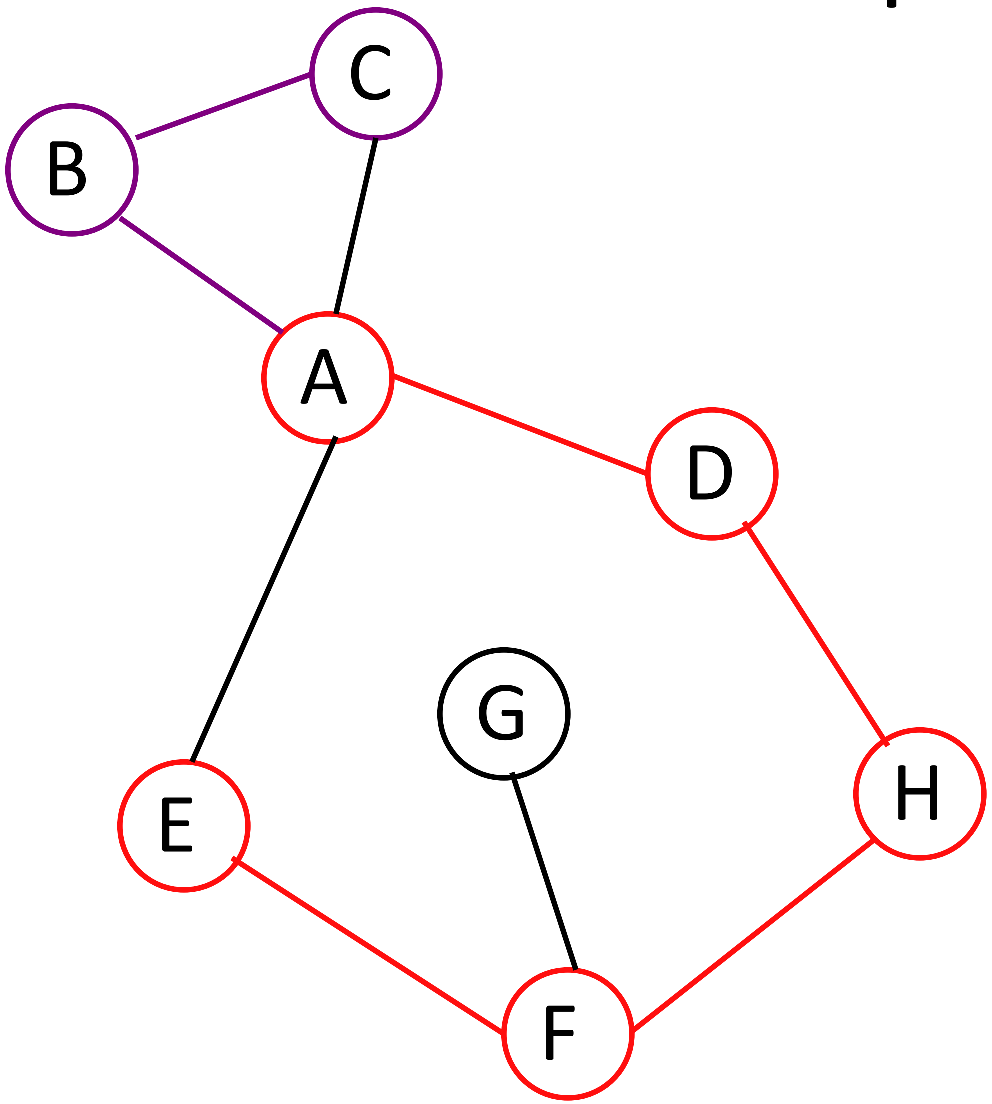

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)

        explore(H)
  explore(E)
```

Example



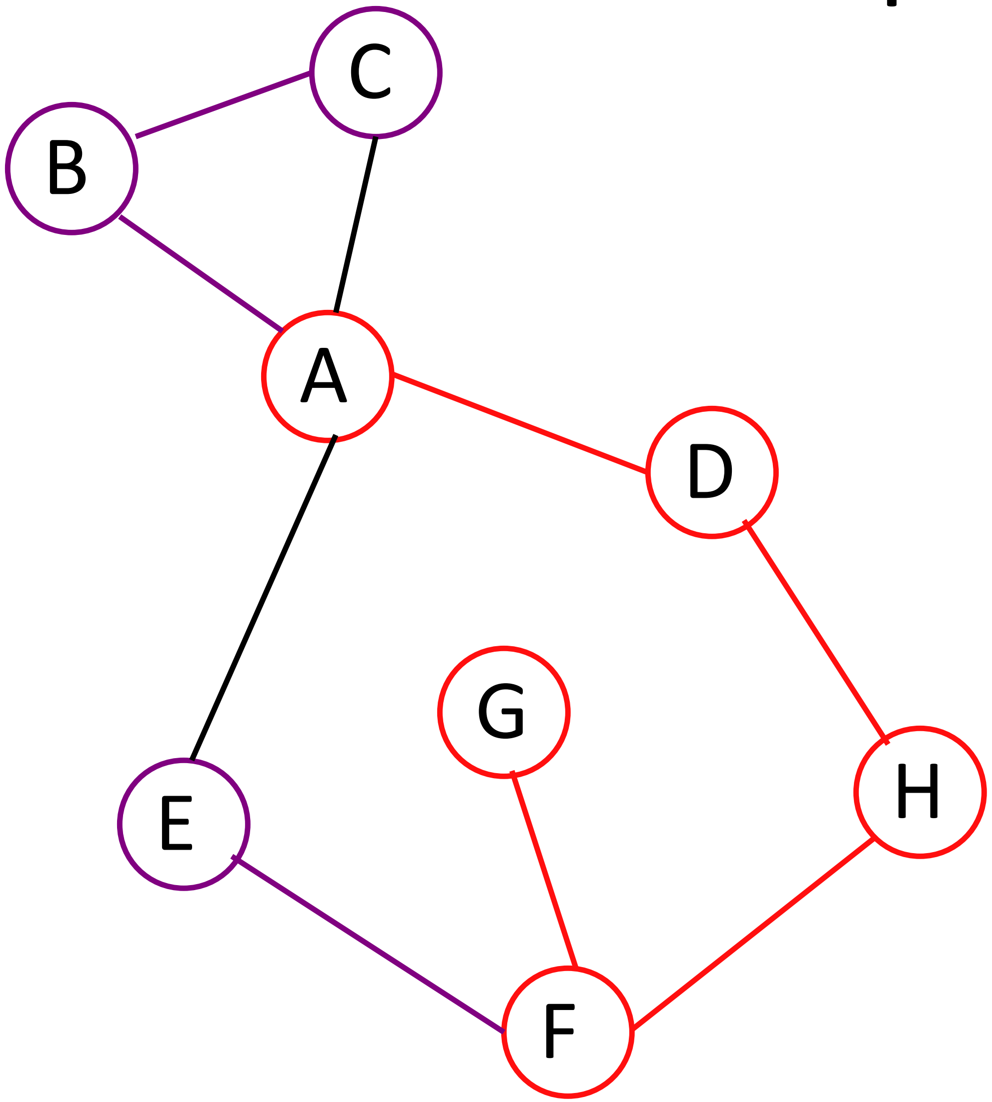

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)

        explore(H)
  explore(E)
```

Example



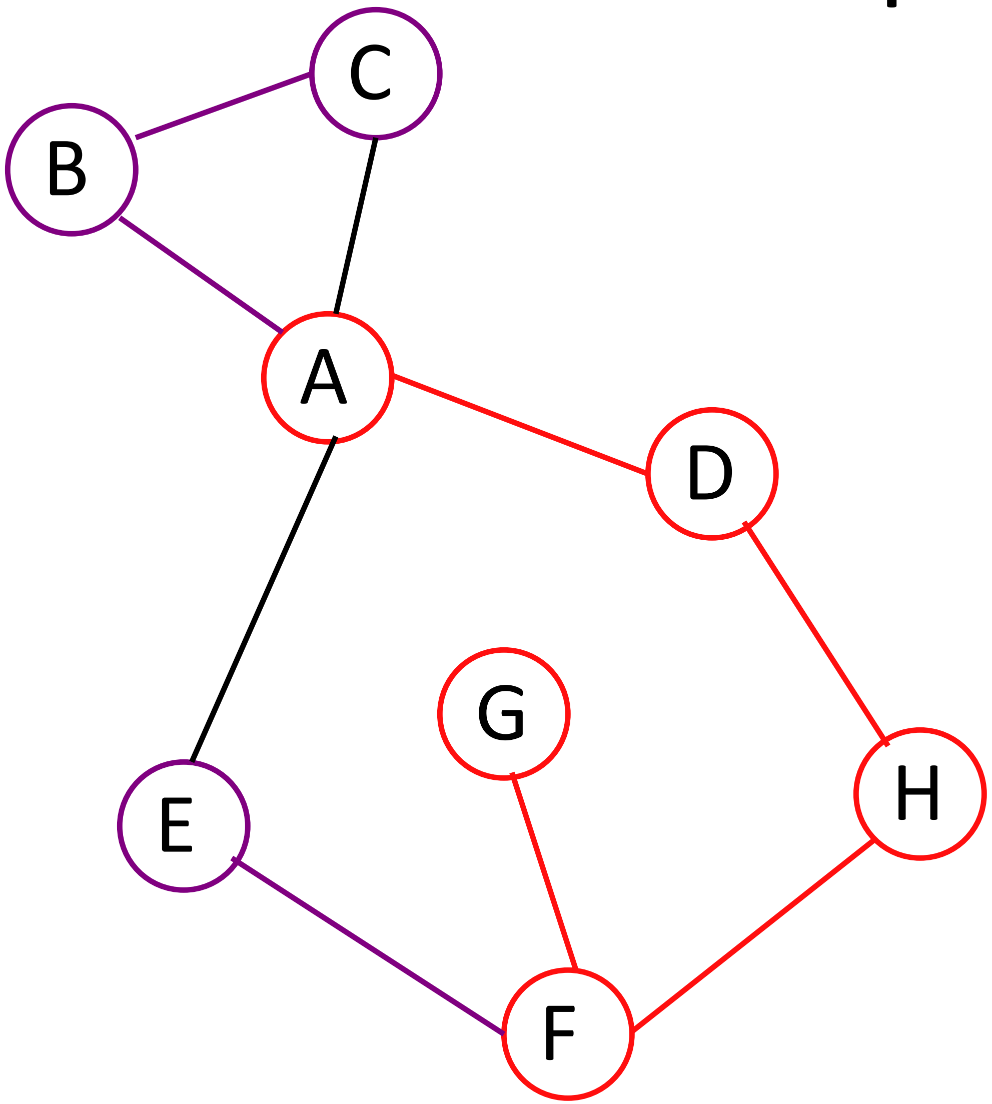

```
explore(A)
   explore(B)
      explore(A)
      explore(C)
         explore(A)
         explore(B)
   explore(C)
   explore(D)
      explore(A)
      explore(H)
         explore(D)
         explore(F)
            explore(E)
               explore(A)
               explore(F)
            explore(G)

            explore(H)
   explore(E)
```

Example



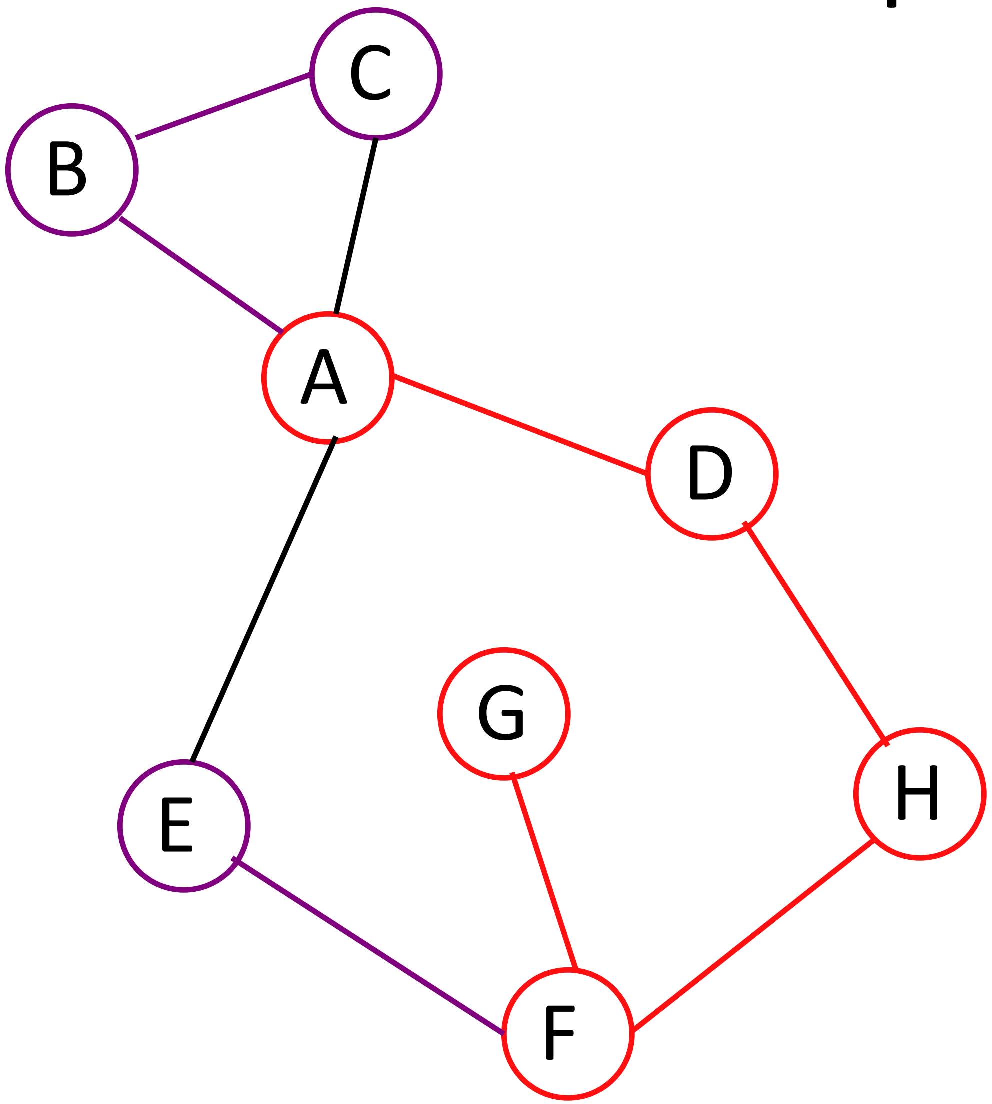

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)

        explore(H)
  explore(E)
```

Example



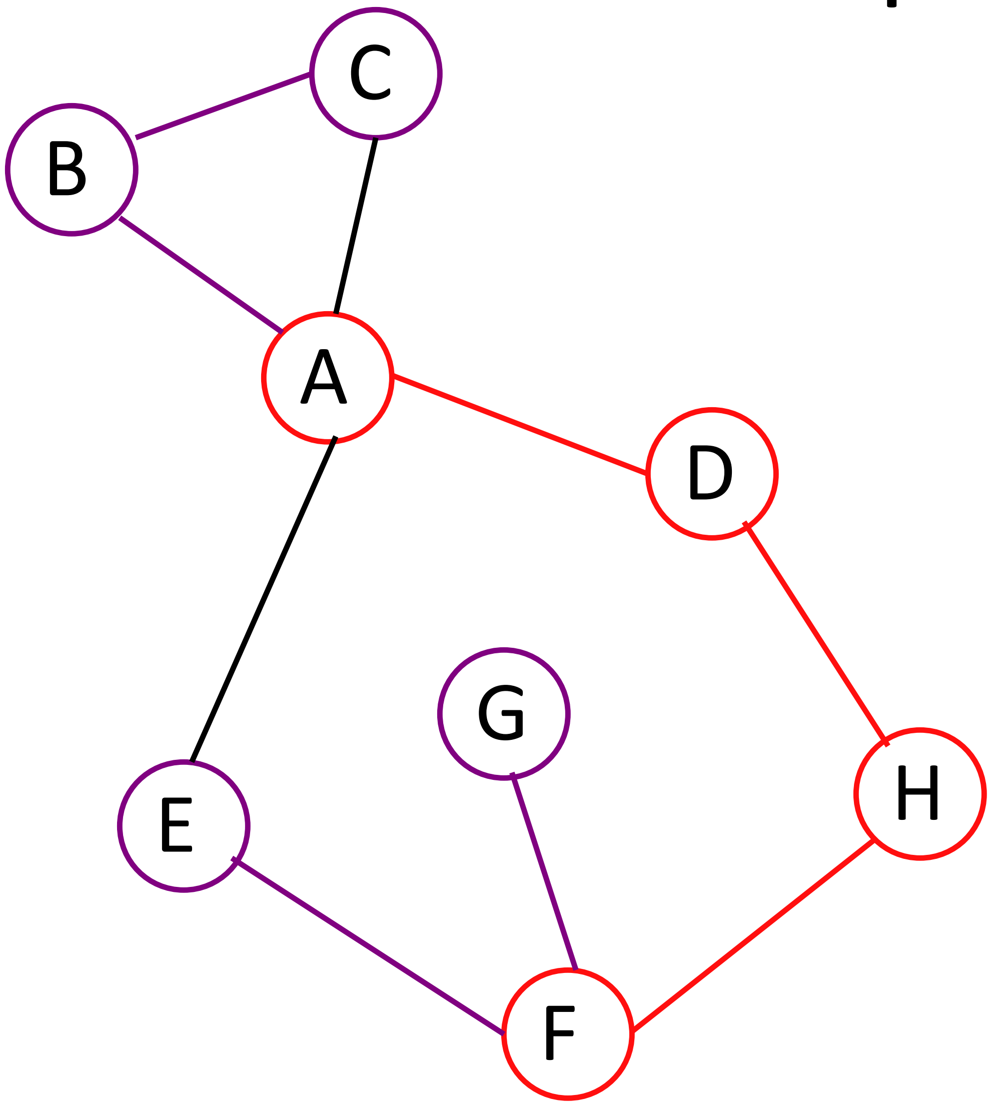

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example



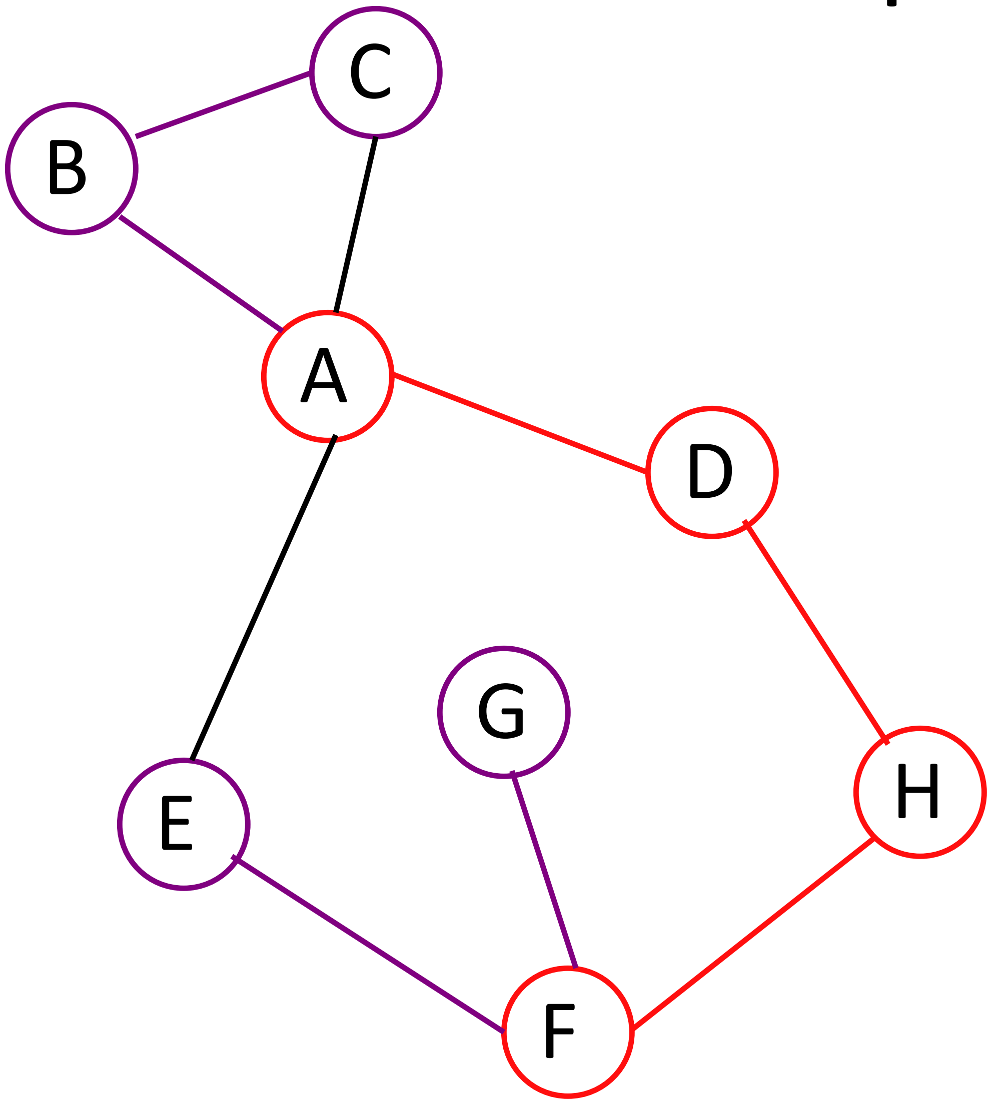

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example



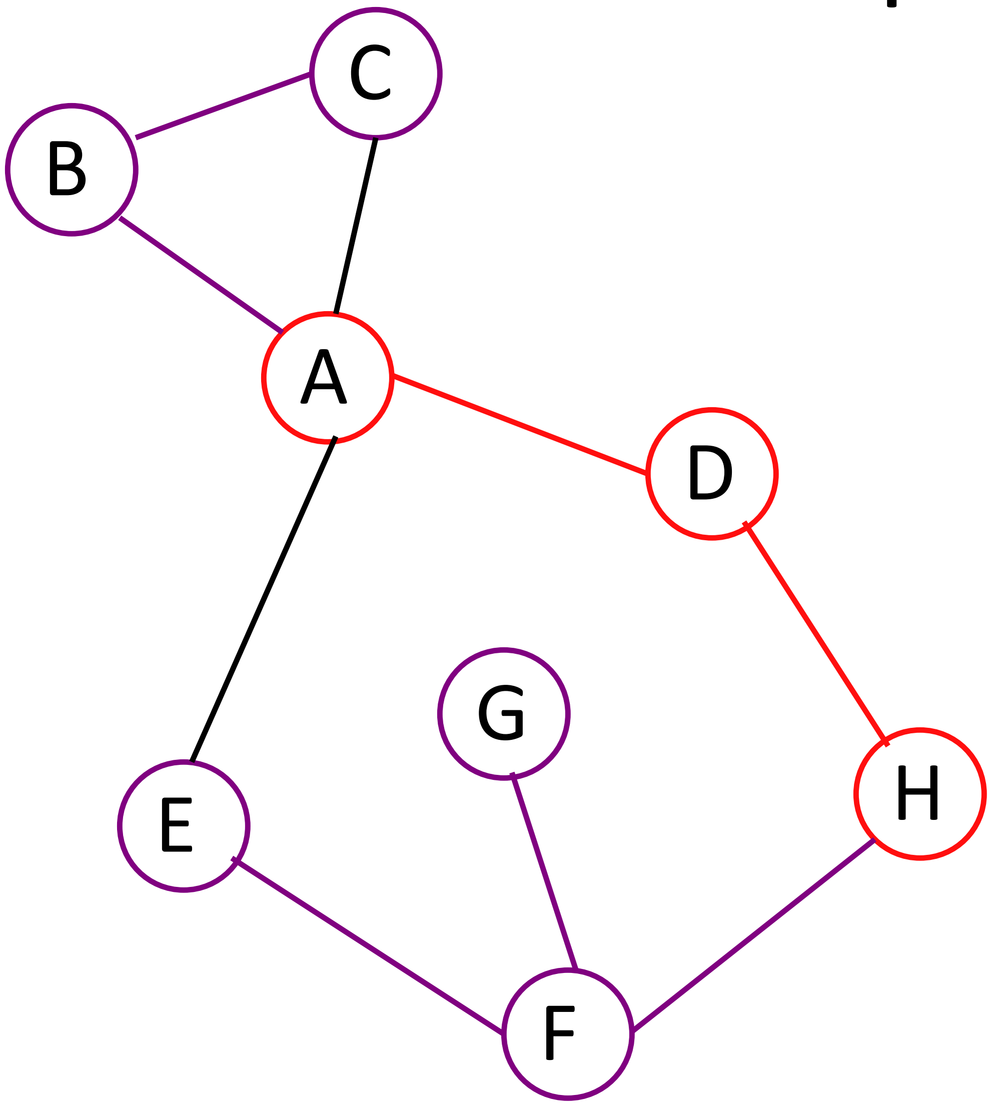

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example



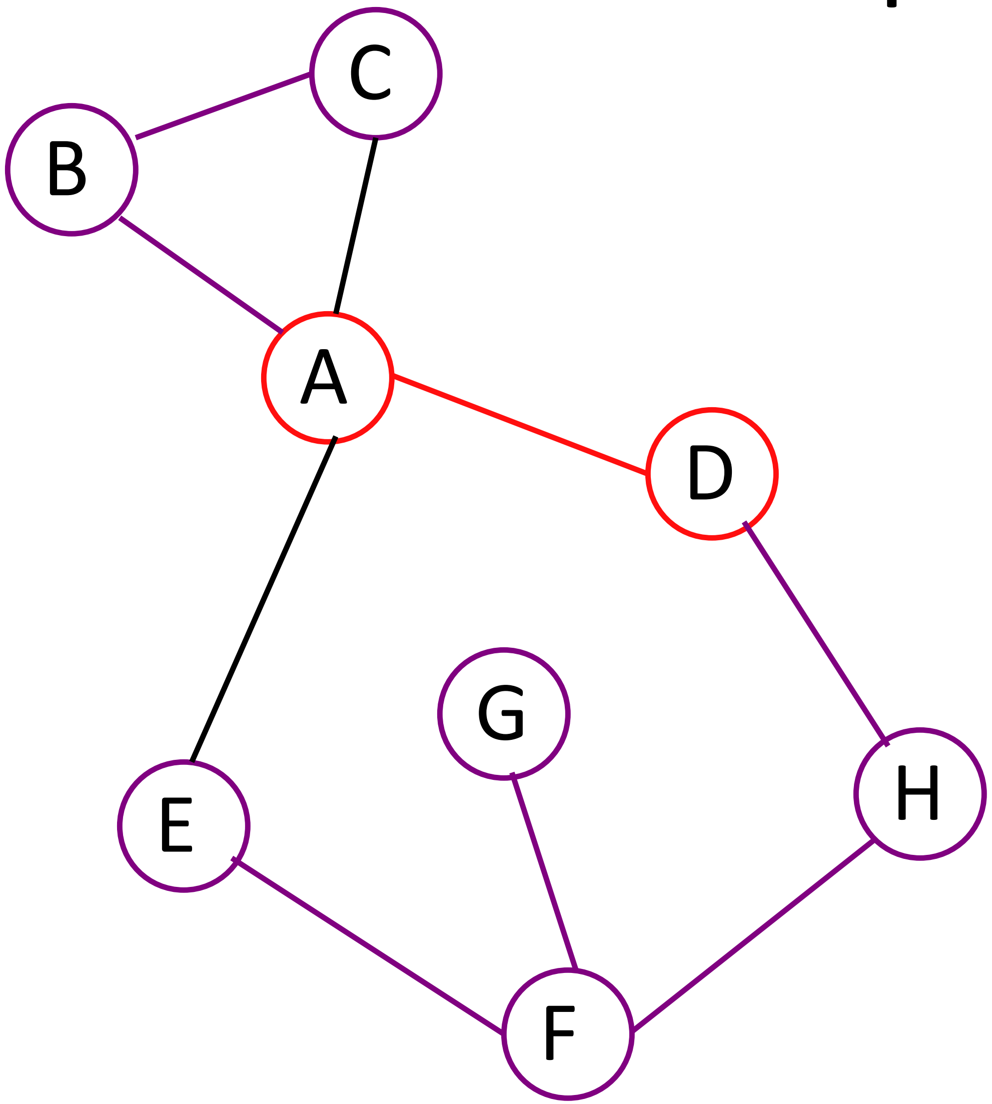

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example



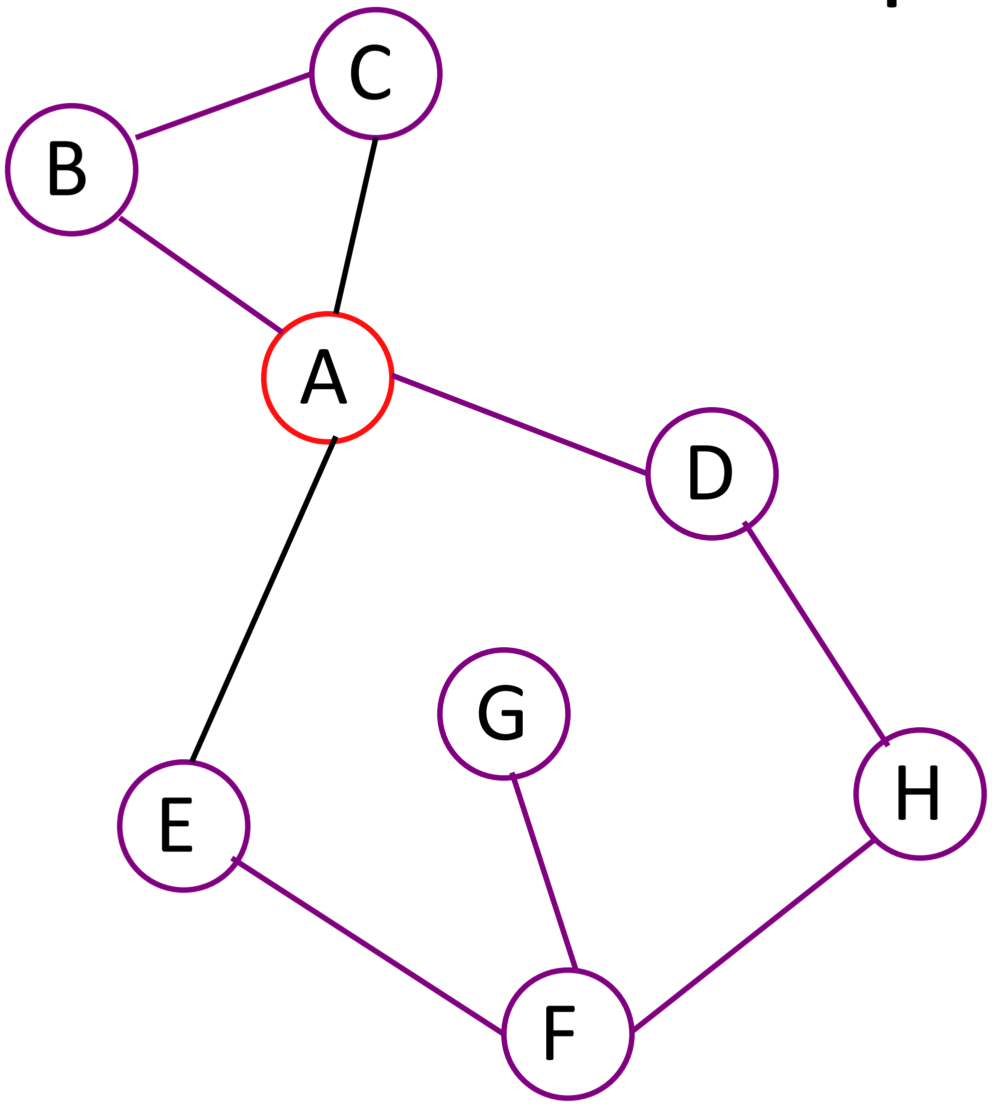

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example



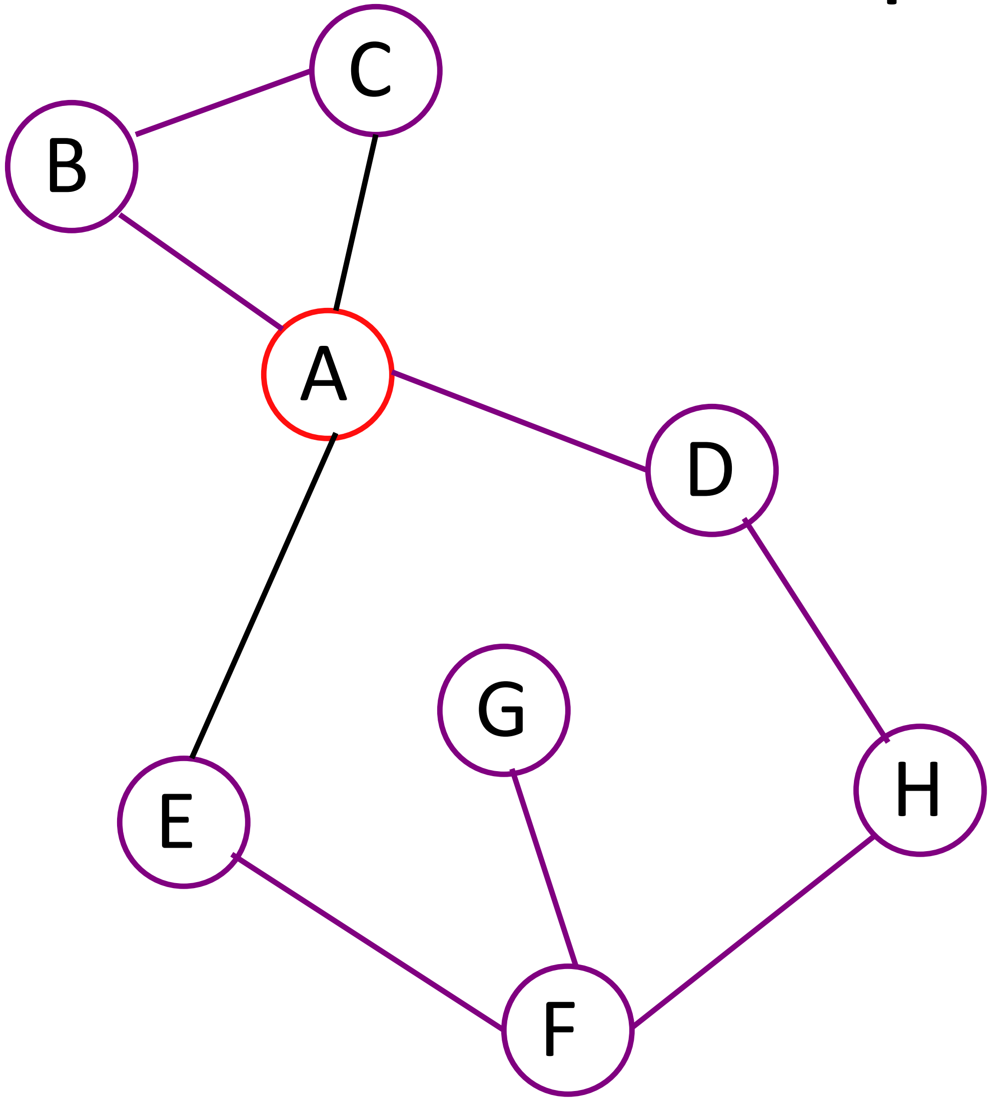

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example



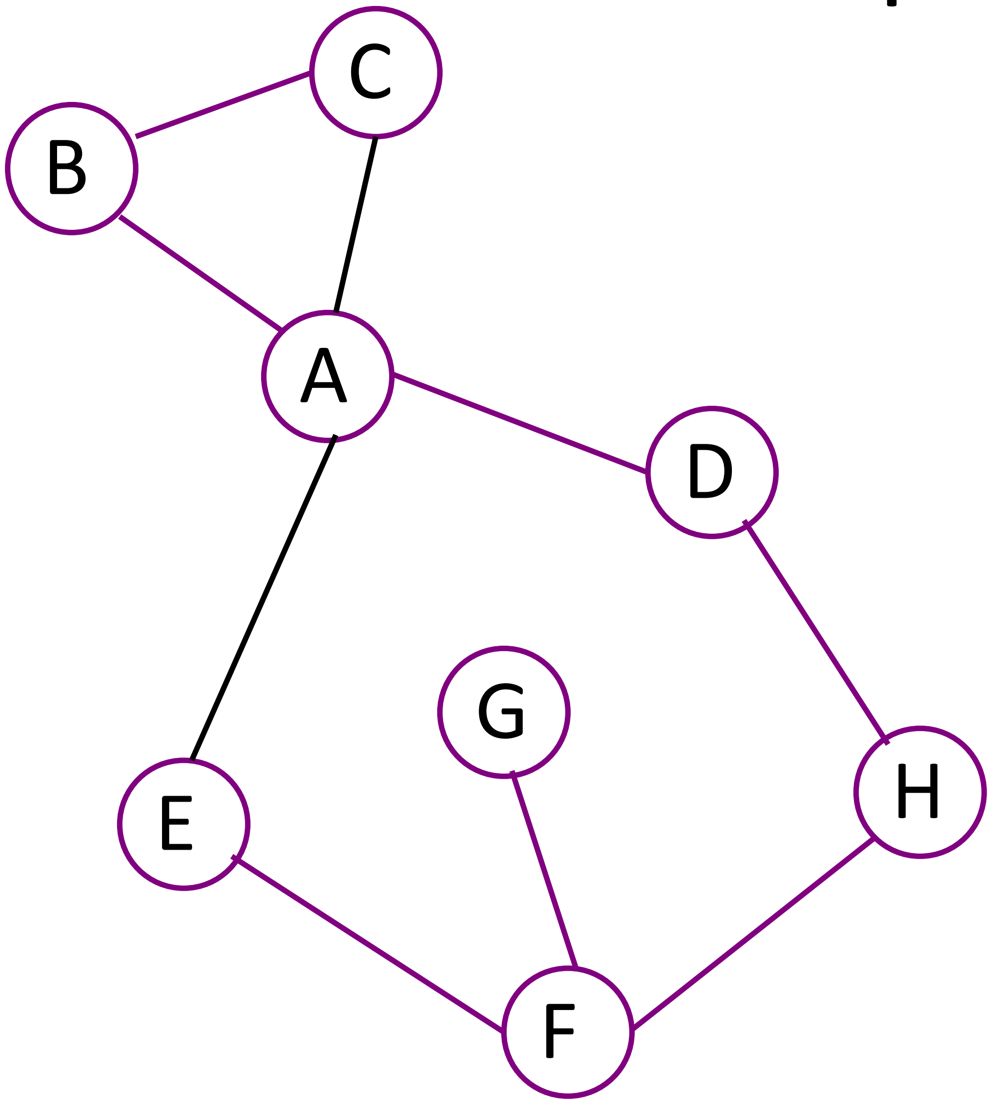

```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example




```
explore(A)
    explore(B)
        explore(A)
        explore(C)
            explore(A)
            explore(B)
    explore(C)
    explore(D)
        explore(A)
        explore(H)
            explore(D)
            explore(F)
                explore(E)
                    explore(A)
                    explore(F)
                explore(G)
                    explore(F)
                explore(H)
    explore(E)
```

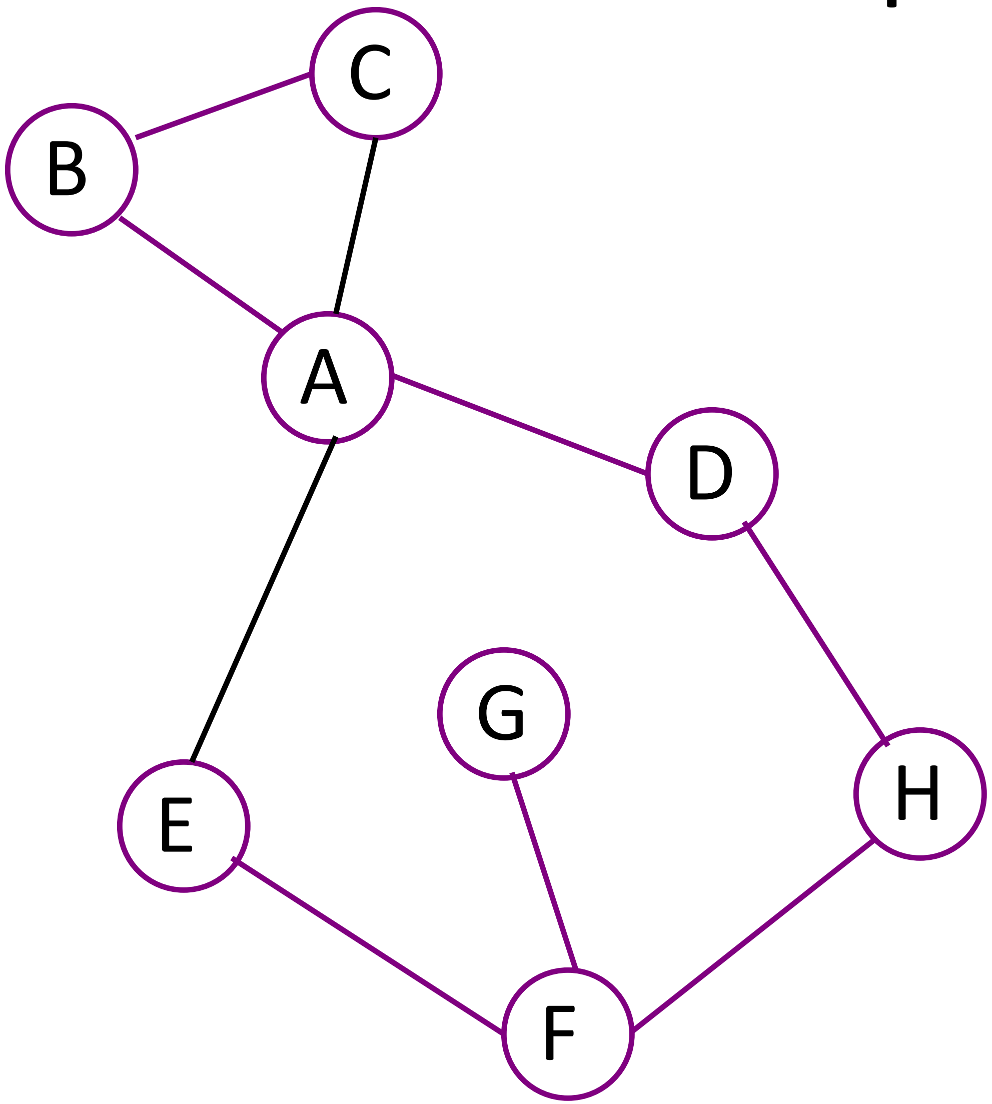
Example




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Example

Note: edges used leave behind "DFS tree".




```
explore(A)
  explore(B)
    explore(A)
    explore(C)
      explore(A)
      explore(B)
  explore(C)
  explore(D)
    explore(A)
    explore(H)
      explore(D)
      explore(F)
        explore(E)
          explore(A)
          explore(F)
        explore(G)
          explore(F)
        explore(H)
  explore(E)
```

Result

Theorem: If all vertices start unvisited, `explore(v)` marks as visited exactly the vertices reachable from v.

Result

Theorem: If all vertices start unvisited, `explore(v)` marks as visited exactly the vertices reachable from v.

Proof:

- Only visits vertices reachable from v.

Result

Theorem: If all vertices start unvisited, `explore(v)` marks as visited exactly the vertices reachable from v.

Proof:

- Only visits vertices reachable from v.
- If u is visited, eventually visit every adjacent w.

Result

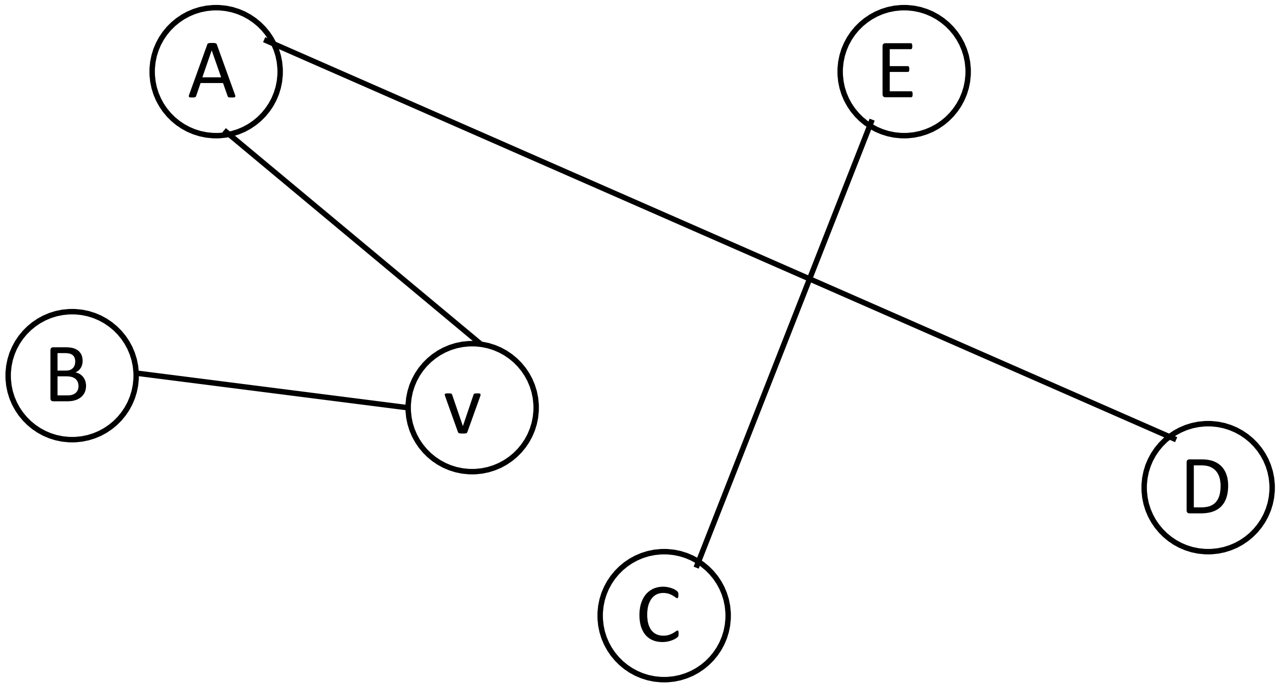
Theorem: If all vertices start unvisited, `explore(v)` marks as visited exactly the vertices reachable from v.

Proof:

- Only visits vertices reachable from v.
- If u is visited, eventually visit every adjacent w.
  - If there is a chain of vertices
    $v \rightarrow u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow w$
    eventually visit all of them

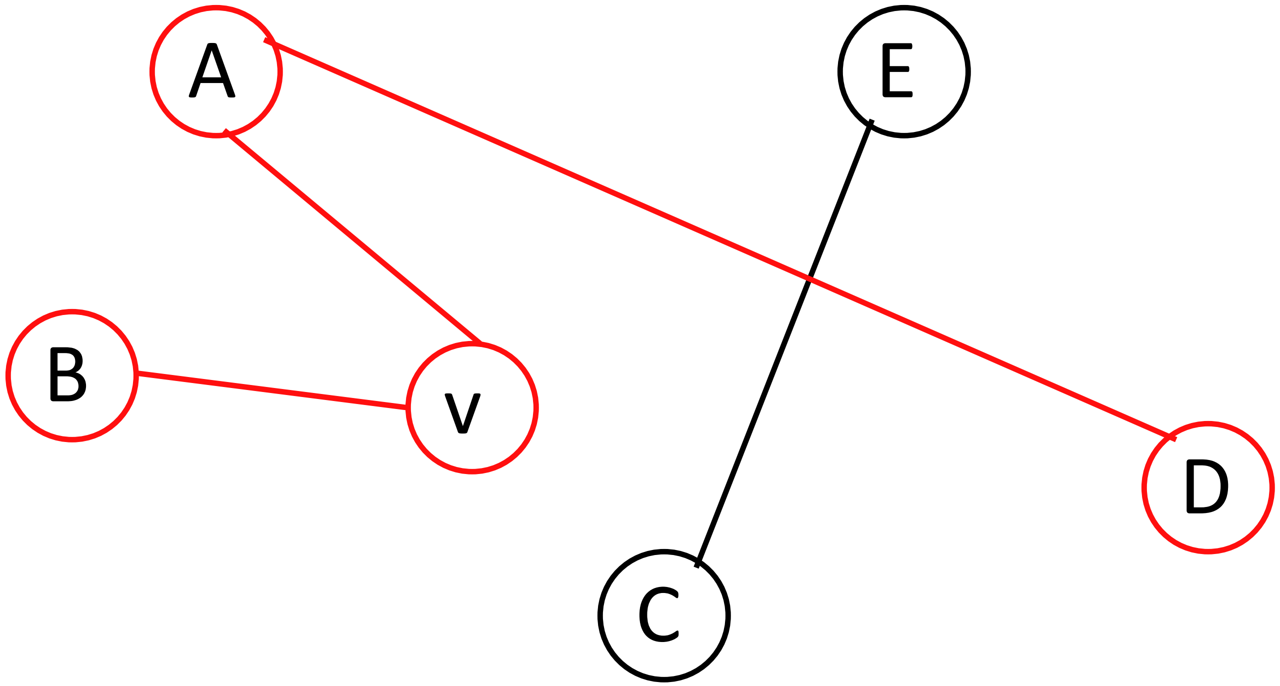
Question: explore

Which vertices does `explore(v)` mark as visited?



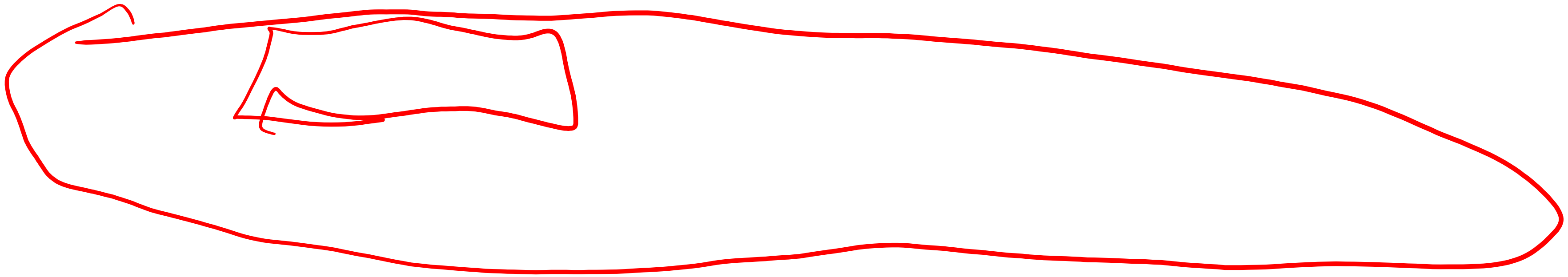
Question: explore

Which vertices does `explore(v)` mark as visited?



Depth First Search

`explore` only finds the part of the graph reachable from a single vertex. If you want to discover the entire graph, you may need to run it multiple times.

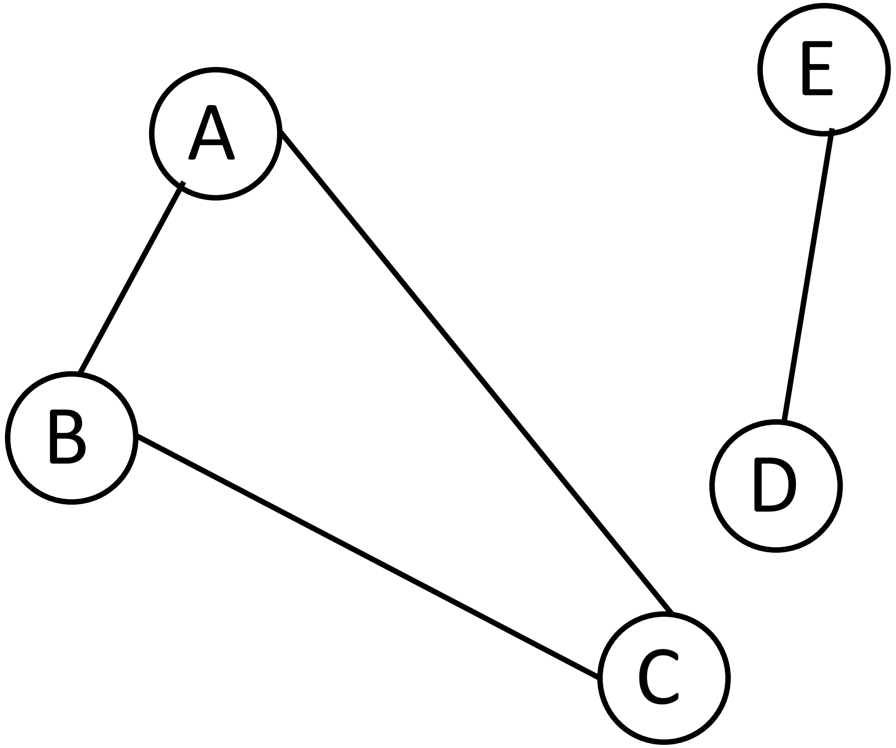


Depth First Search

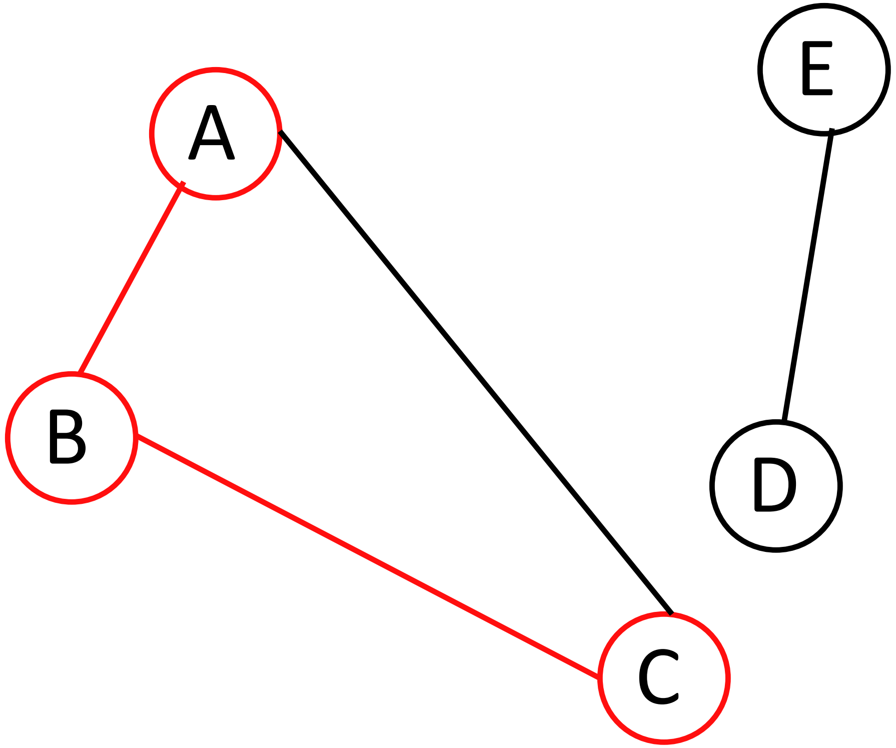
`explore` only finds the part of the graph reachable from a single vertex. If you want to discover the entire graph, you may need to run it multiple times.

```
DepthFirstSearh(G)
  Mark all v ∈ G as unvisited
  For v ∈ G
    If not v.visited, explore(v)
```

Example

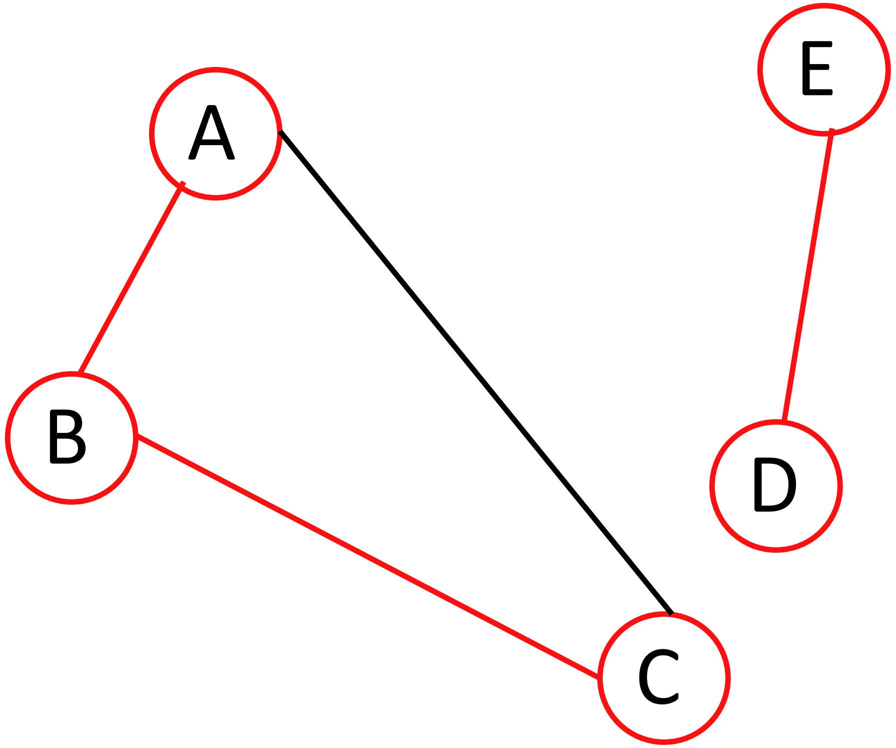


Example



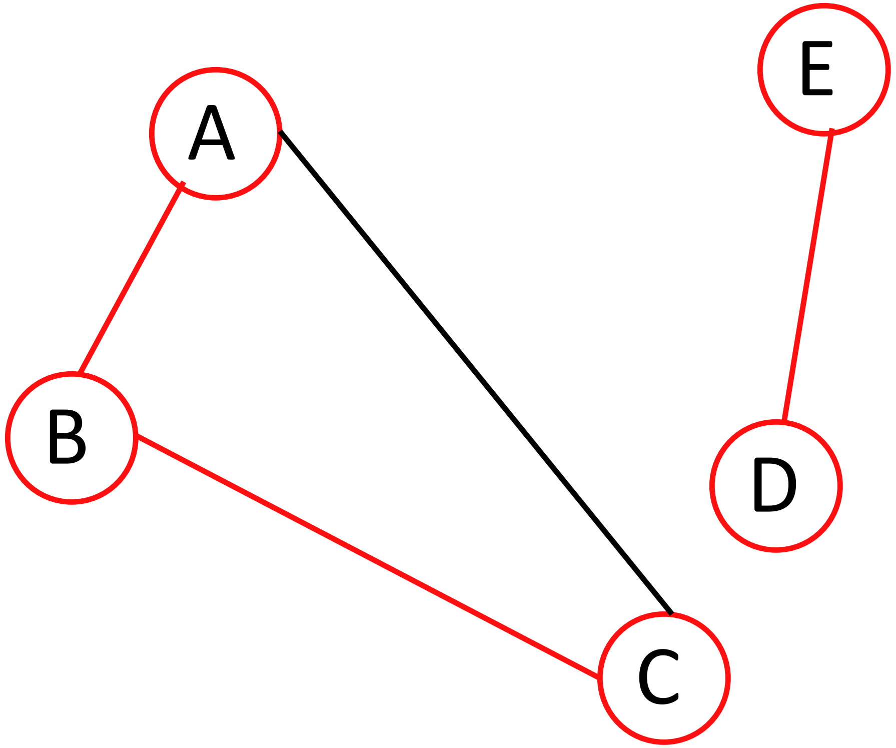
`explore(A)`

Example




```
explore(A)
explore(D)
```

Example




```
explore(A)
explore(D)
```

DFS(G) eventually discovers all vertices in G.

Runtime of DFS

```
explore(v)
  v.visited ← true
  For each edge (v,w)
     If not w.visited
          explore(w)
```

```
DFS(G)
  Mark all v ∈ G as unvisited
  For v ∈ G
    If not v.visited, explore(v)
```

Runtime of DFS

```
explore(v)
  v.visited ← true
  For each edge (v,w)
     If not w.visited
          explore(w)
```

Run once per vertex

O(|V|) total

```
DFS(G)
  Mark all v ∈ G as unvisited
  For v ∈ G
    If not v.visited, explore(v)
```

Runtime of DFS

```
explore(v)
  v.visited ← true
  For each edge (v,w)
     If not w.visited
          explore(w)
```

Run once per vertex

O(|V|) total

Run once per neighboring vertex

O(|E|) total

```
DFS(G)
  Mark all v ∈ G as unvisited
  For v ∈ G
    If not v.visited, explore(v)
```

Runtime of DFS

```
explore(v)
  v.visited ← true
  For each edge (v,w)
     If not w.visited
          explore(w)
```

Run once per vertex — O(|V|) total

Run once per neighboring vertex — O(|E|) total

```
DFS(G)
  Mark all v ∈ G as unvisited
  For v ∈ G
    If not v.visited, explore(v)
```

O(|V|)

Runtime of DFS

```
explore(v)
  v.visited ← true
  For each edge (v,w)
     If not w.visited
          explore(w)
```

Run once per vertex

O(|V|) total

Run once per neighboring vertex

O(|E|) total

```
DFS(G)
  Mark all v ∈ G as unvisited
  For v ∈ G
    If not v.visited, explore(v)
```

O(|V|)

Final runtime: O(|V|+|E|)

Note on Graph Algorithm Runtimes

Graph algorithm runtimes depend on both $|V|$ and $|E|$. (Note $O(|V|+|E|)$ is linear time)

Note on Graph Algorithm Runtimes

Graph algorithm runtimes depend on both $|V|$ and $|E|$. (Note $\boxed{O(|V|+|E|)}$ is linear time)

What algorithm is better may depend on relative sizes of these parameters.

Note on Graph Algorithm Runtimes

Graph algorithm runtimes depend on both $|V|$ and $|E|$. (Note $\boxed{O(|V|+|E|)}$ is linear time)

What algorithm is better may depend on relative sizes of these parameters.

<u>Sparse Graphs:</u>

$|E|$ small ($\approx V$)

Examples:

- Internet
- Road maps

<u>Dense Graphs:</u>

$|E|$ large ($\approx V^2$)

Examples:

- Flight maps
- Wireless networks

Question: Graph Runtimes

Suppose that you have two graph algorithms for the same problem

Alg1 has runtime $O(|V|^{3/2})$

Alg2 has runtime $O(|E|)$

Which of the following is likely true?

A) Alg1 is faster on most graphs

B) Alg2 is faster on most graphs

C) Alg1 is faster on sparse graphs, slower on dense graphs

D) Alg2 is faster on sparse graphs, slower on dense graphs

Question: Graph Runtimes

Suppose that you have two graph algorithms for the same problem

Alg1 has runtime $O(|V|^{3/2})$

Alg2 has runtime $O(|E|)$

Which of the following is likely true?

A) Alg1 is faster on most graphs

B) Alg2 is faster on most graphs

C) Alg1 is faster on sparse graphs, slower on dense graphs

D) Alg2 is faster on sparse graphs, slower on dense graphs

Graph Representations

How do you store a graph in a computer?

Graph Representations

How do you store a graph in a computer?

- Adjacency matrix: Store list of vertices and an array A[i,j] = 1 if edge between v_i and v_j.
  - Small space for dense graphs.
  - Slow for most operations.

Graph Representations

How do you store a graph in a computer?

- Adjacency matrix: Store list of vertices and an array A[i,j] = 1 if edge between v_i and v_j.
  - Small space for dense graphs.
  - Slow for most operations.
- Edge list: List of all vertices, list of all edges
  - Hard to determine edges out of single vertex.

Graph Representations

How do you store a graph in a computer?

- Adjacency matrix: Store list of vertices and an array A[i,j] = 1 if edge between v_i and v_j.
  - Small space for dense graphs.
  - Slow for most operations.
- Edge list: List of all vertices, list of all edges
  - Hard to determine edges out of single vertex.
- Adjacency list: For each vertex store list of neighbors.
  - Needed for DFS to be efficient
  - We will usually assume this representation

Connected Components

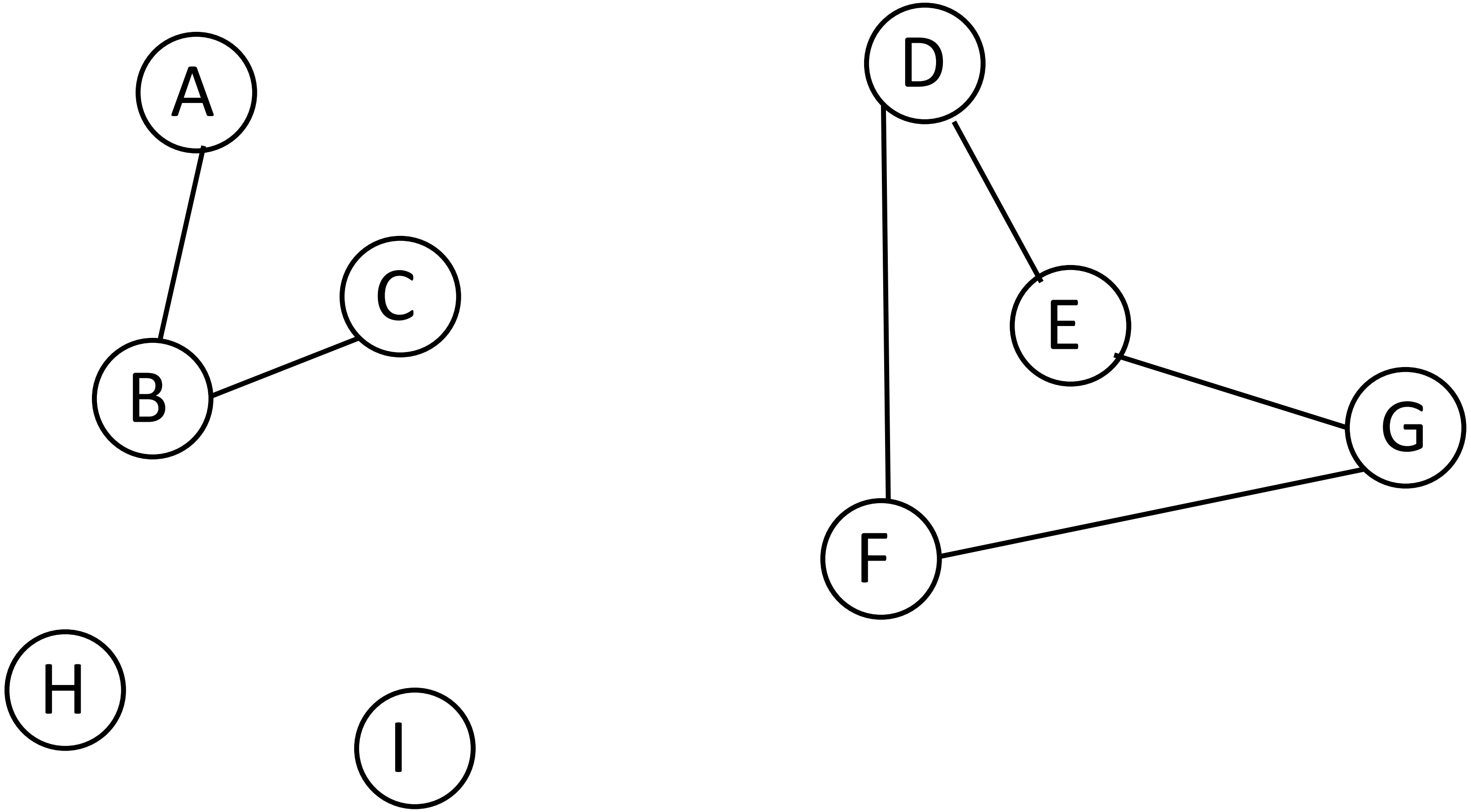
- Want to understand which vertices are reachable from which others in graph G.
- `explore(v)` finds which vertices are reachable from a given vertex.

Connected Components

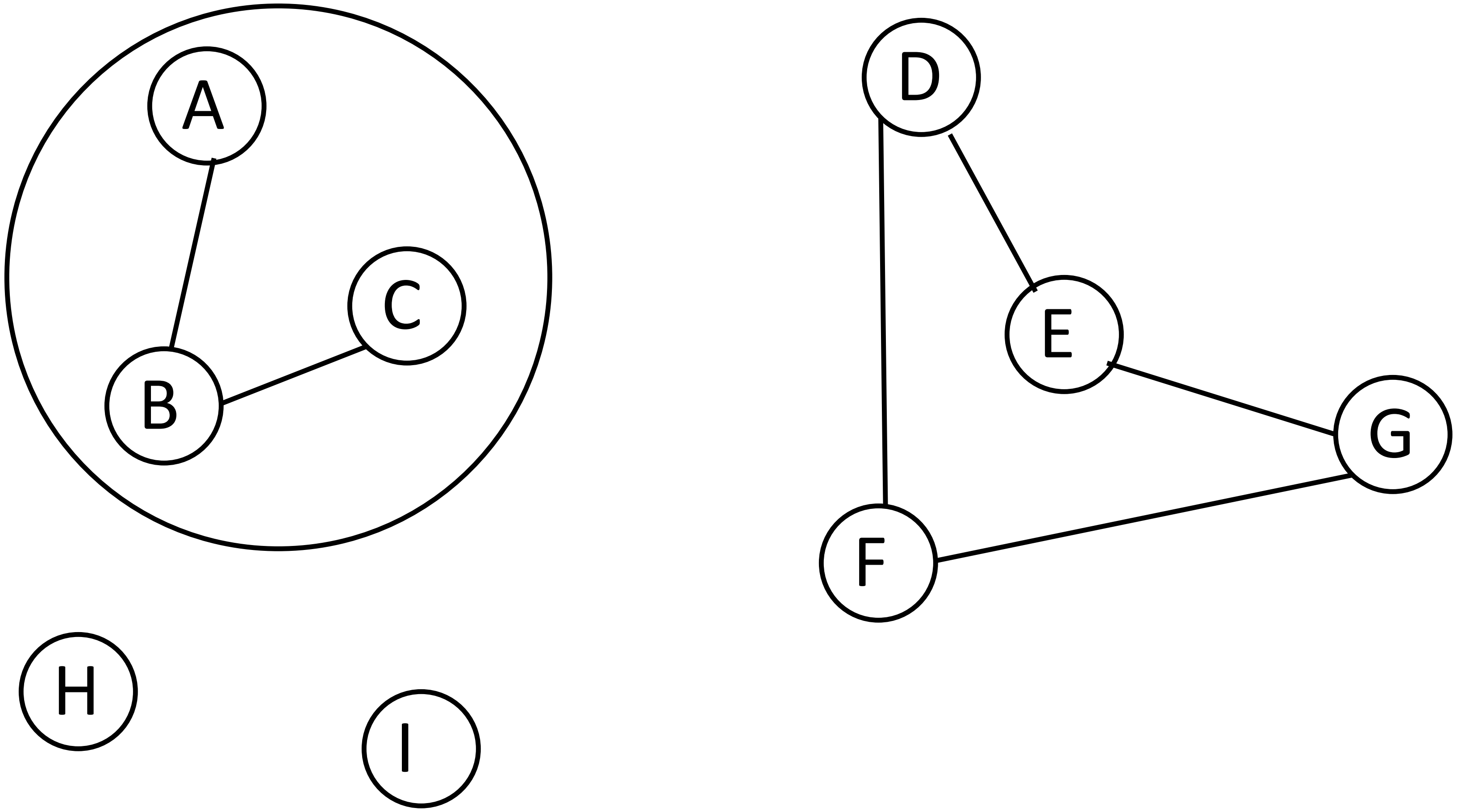
- Want to understand which vertices are reachable from which others in graph G.
- `explore(v)` finds which vertices are reachable from a given vertex.

<u>Theorem:</u> The vertices of a graph G can be partitioned into *connected components* so that v is reachable from w if and only if they are in the same connected component.

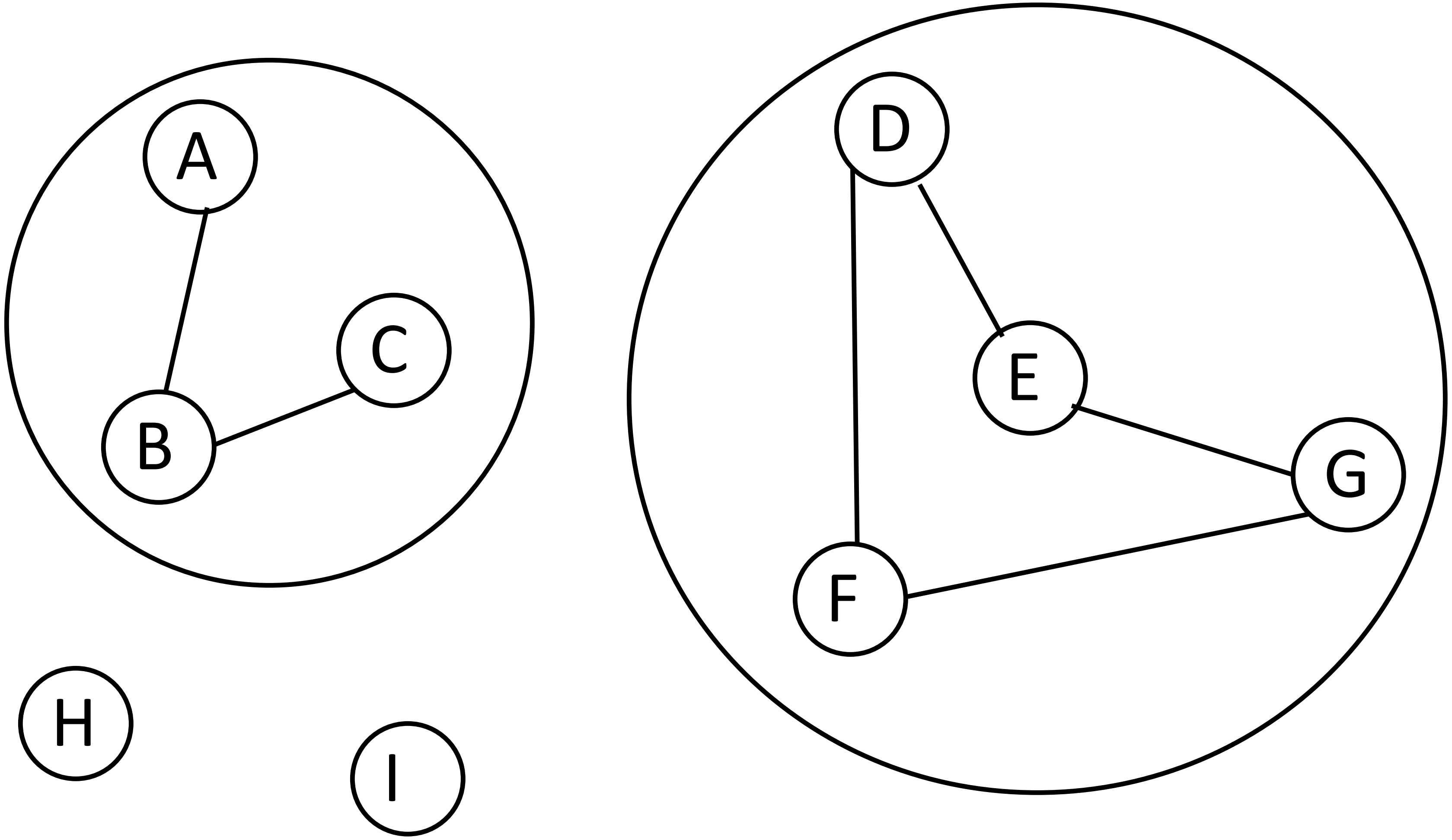
Example



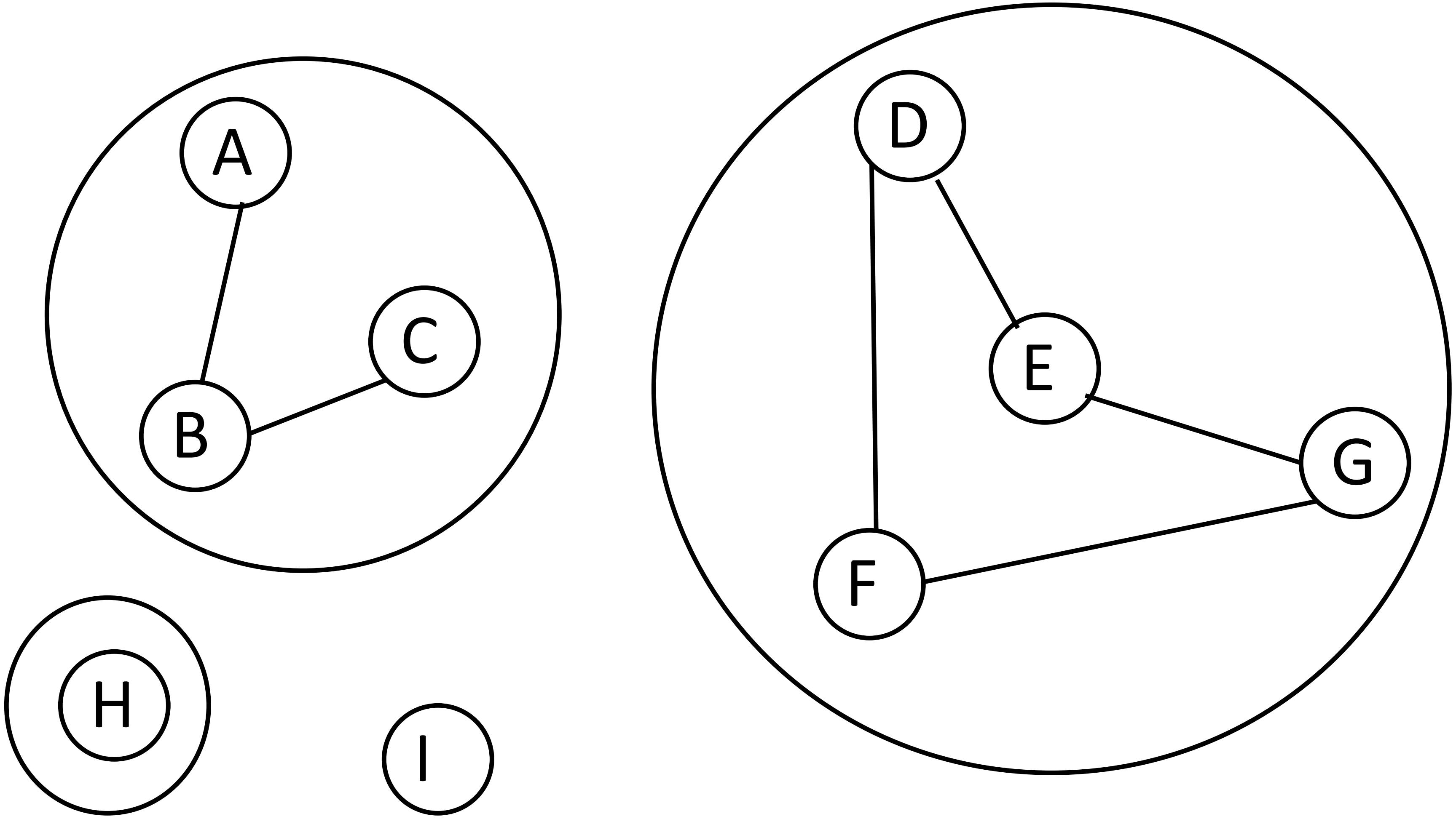
Example



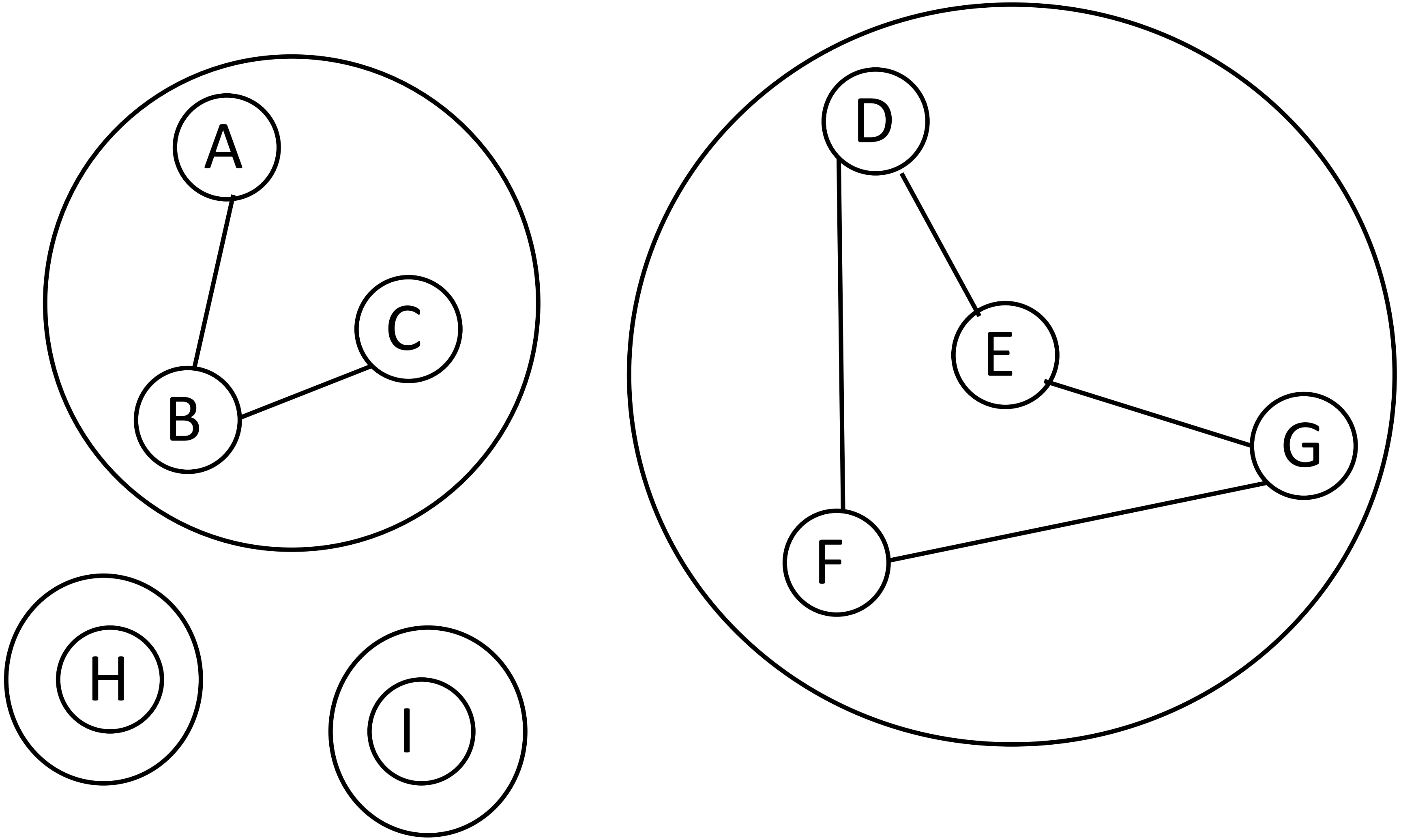
Example



Example



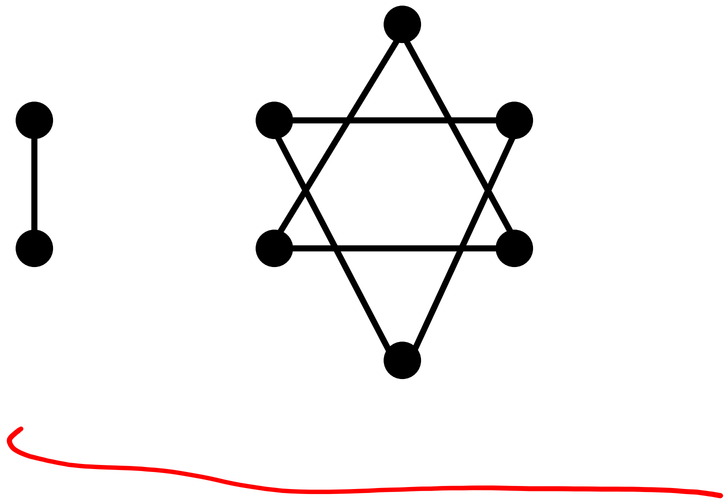
Example



Question: Connected Components

How many connected components does the graph below have?

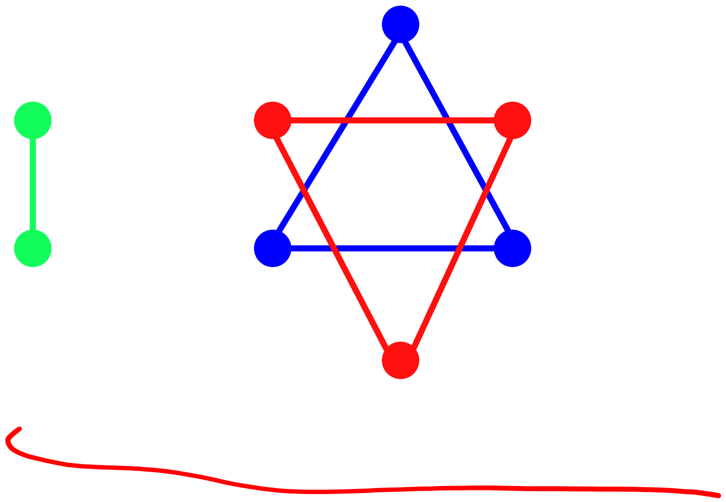
A) 1
B) 2
C) 3
D) 4
E) 5



Question: Connected Components

How many connected components does the graph below have?

A)1
B)2
C)3
D)4
E)5



Problem: Computing Connected Components

Problem: Given a graph G, compute its connected components.