NPTEL MOOC,JAN-FEB 2015
Week 3, Module 4

DESIGN AND ANALYSIS OF ALGORITHMS

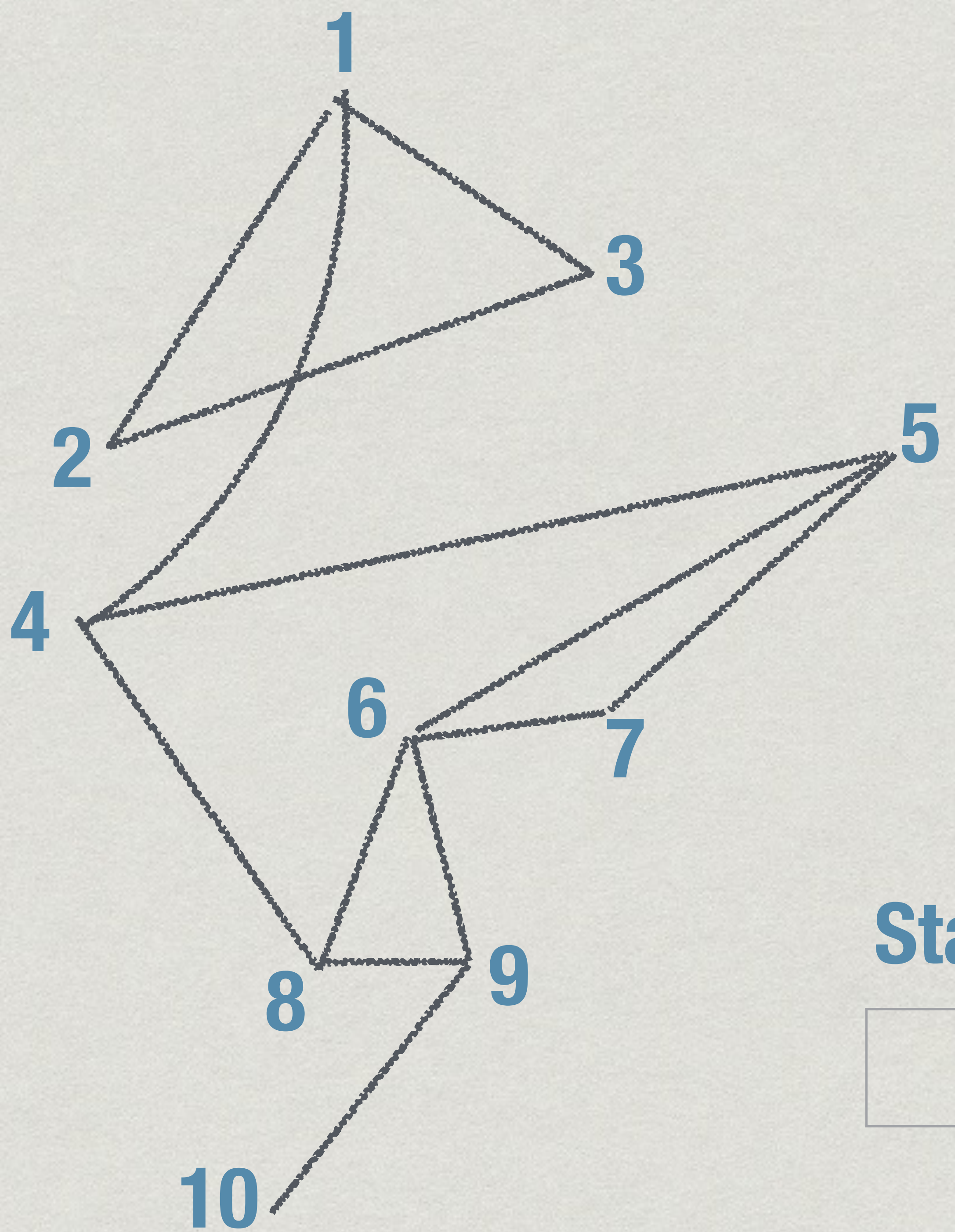
Depth first search (DFS)

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Depth first search

- Start from i, visit a neighbour j
- Suspend the exploration of i and explore j instead
- Continue till you reach a vertex with no unexplored neighbours
- Backtrack to nearest suspended vertex that still has an unexplored neighbour
- Suspended vertices are stored in a **stack**
  - Last in, first out: most recently suspended is checked first

Depth first search

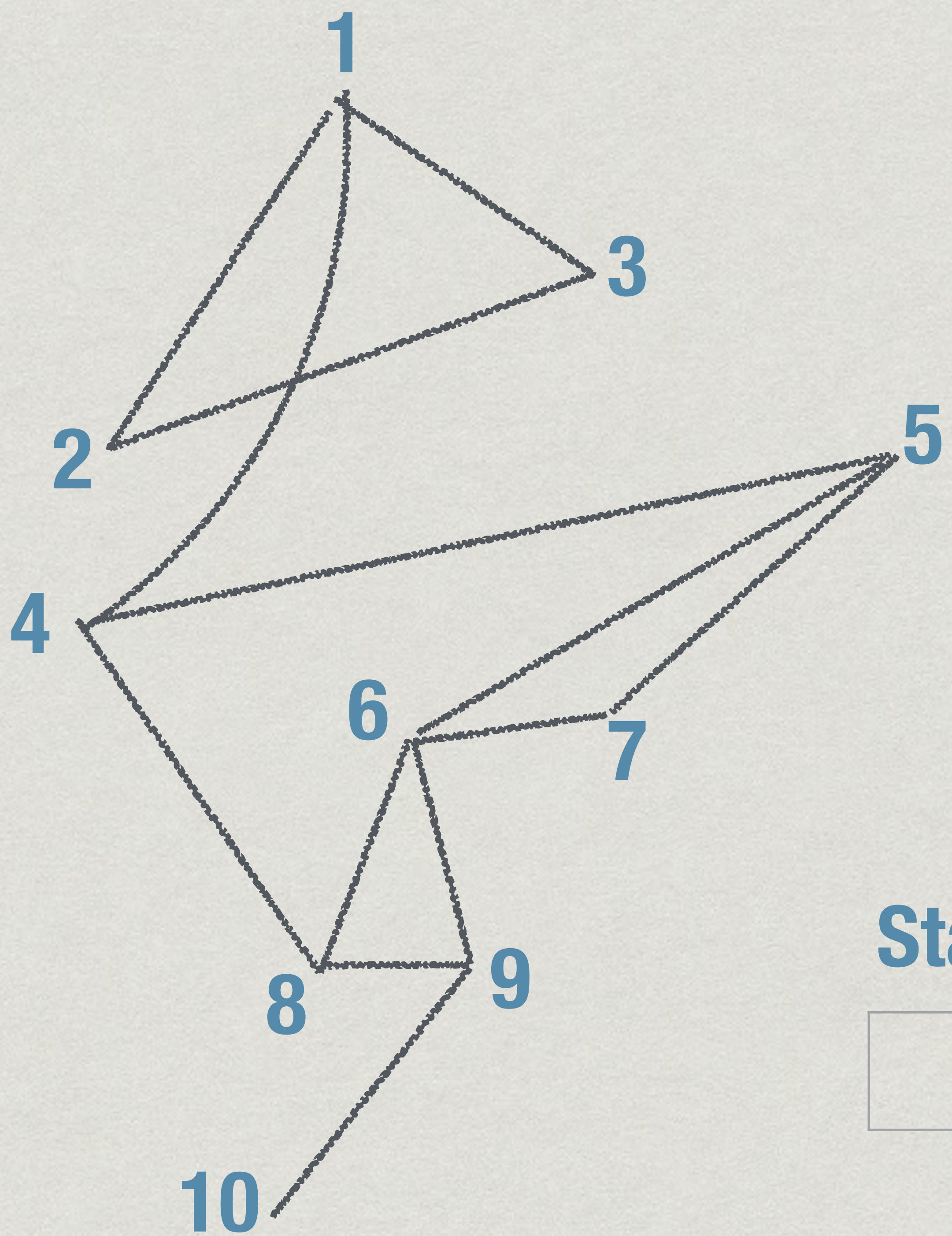


Visited

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

Depth first search



Start at 4

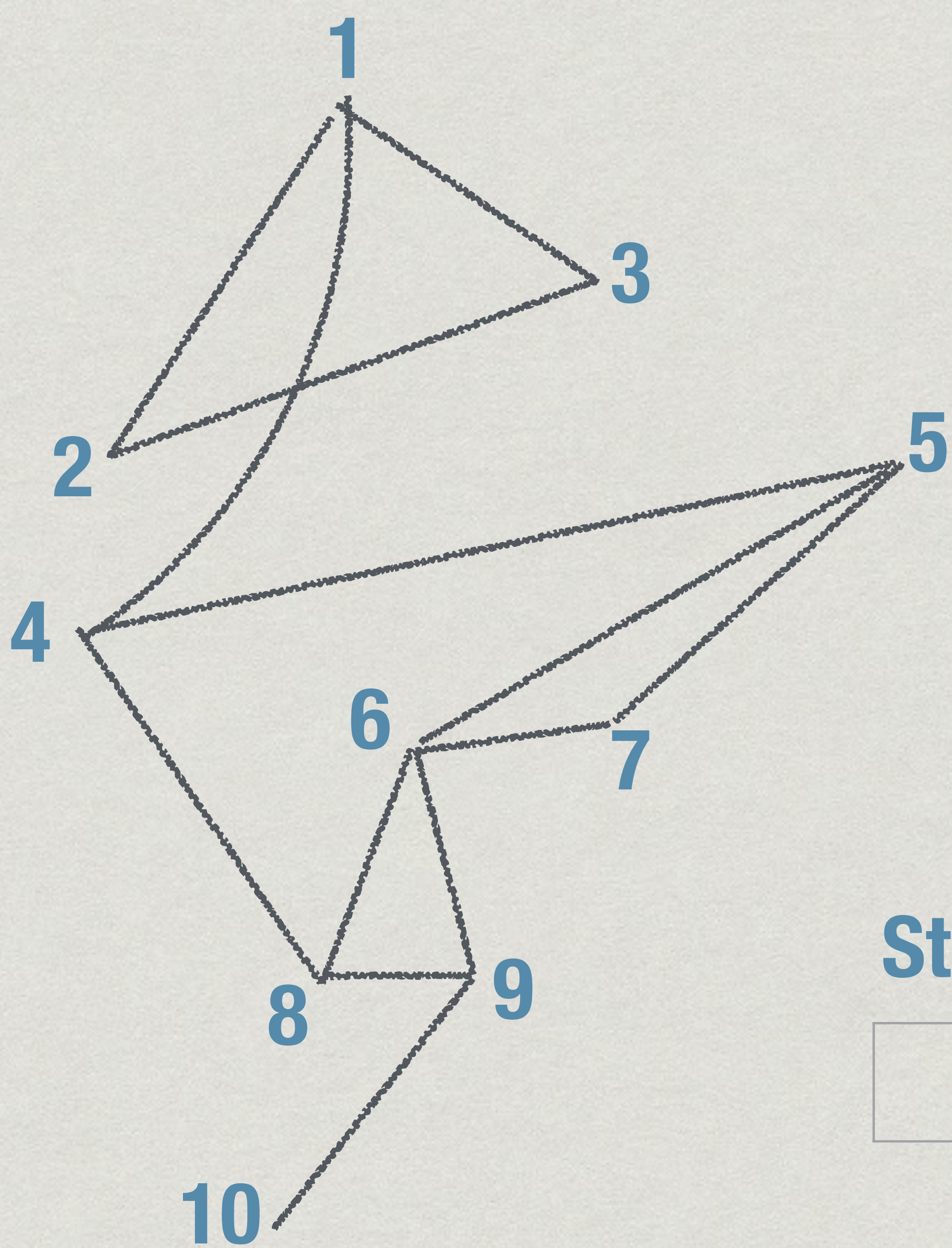
Visited

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

Depth first search

Start at 4



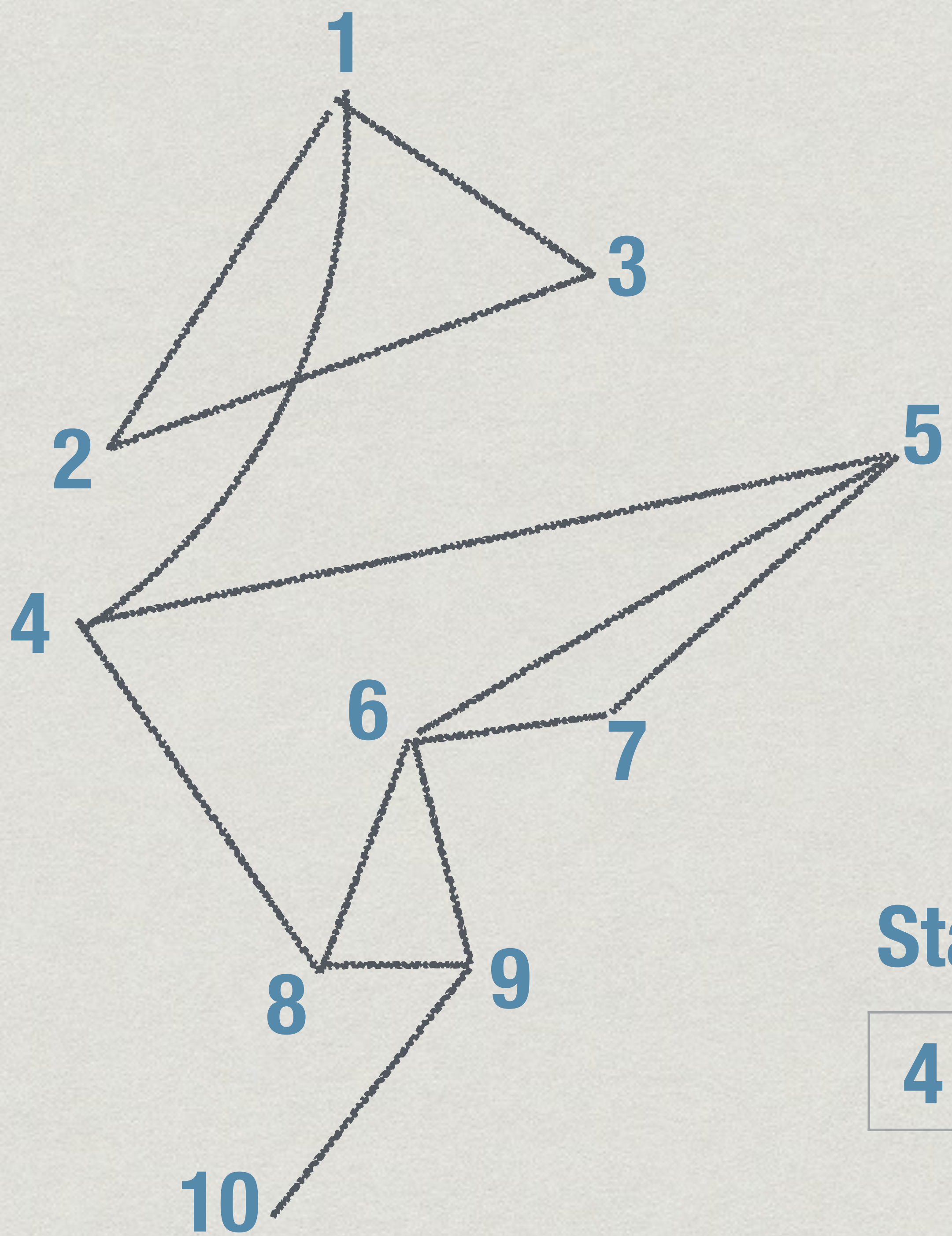
Visited

1	
2	
3	
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

Depth first search

Start at 4



Visited

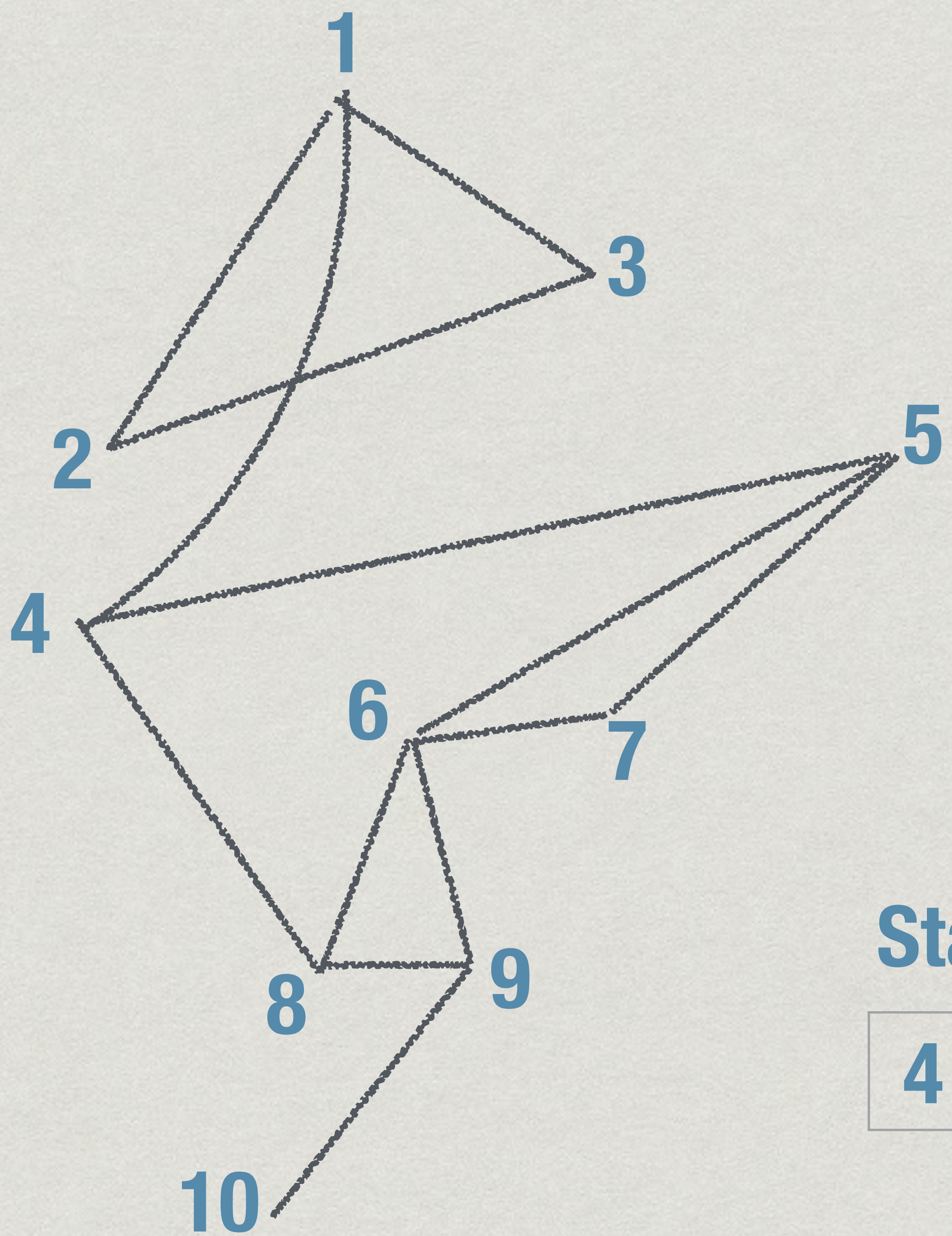
1	1
2	
3	
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

4									

Depth first search

Start at 4



Visited

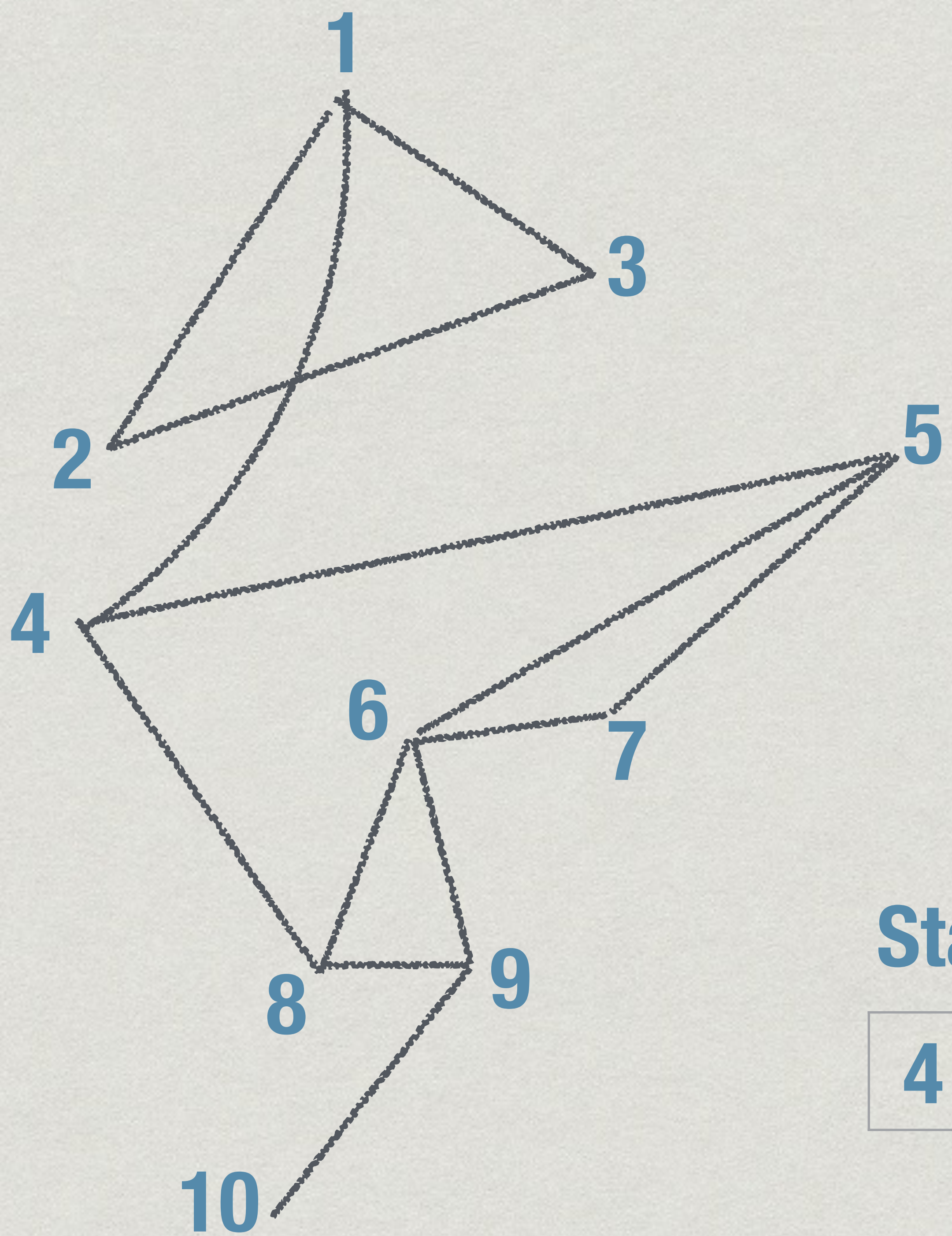
1	1
2	1
3	
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

4	1								

Depth first search

Start at 4



Visited

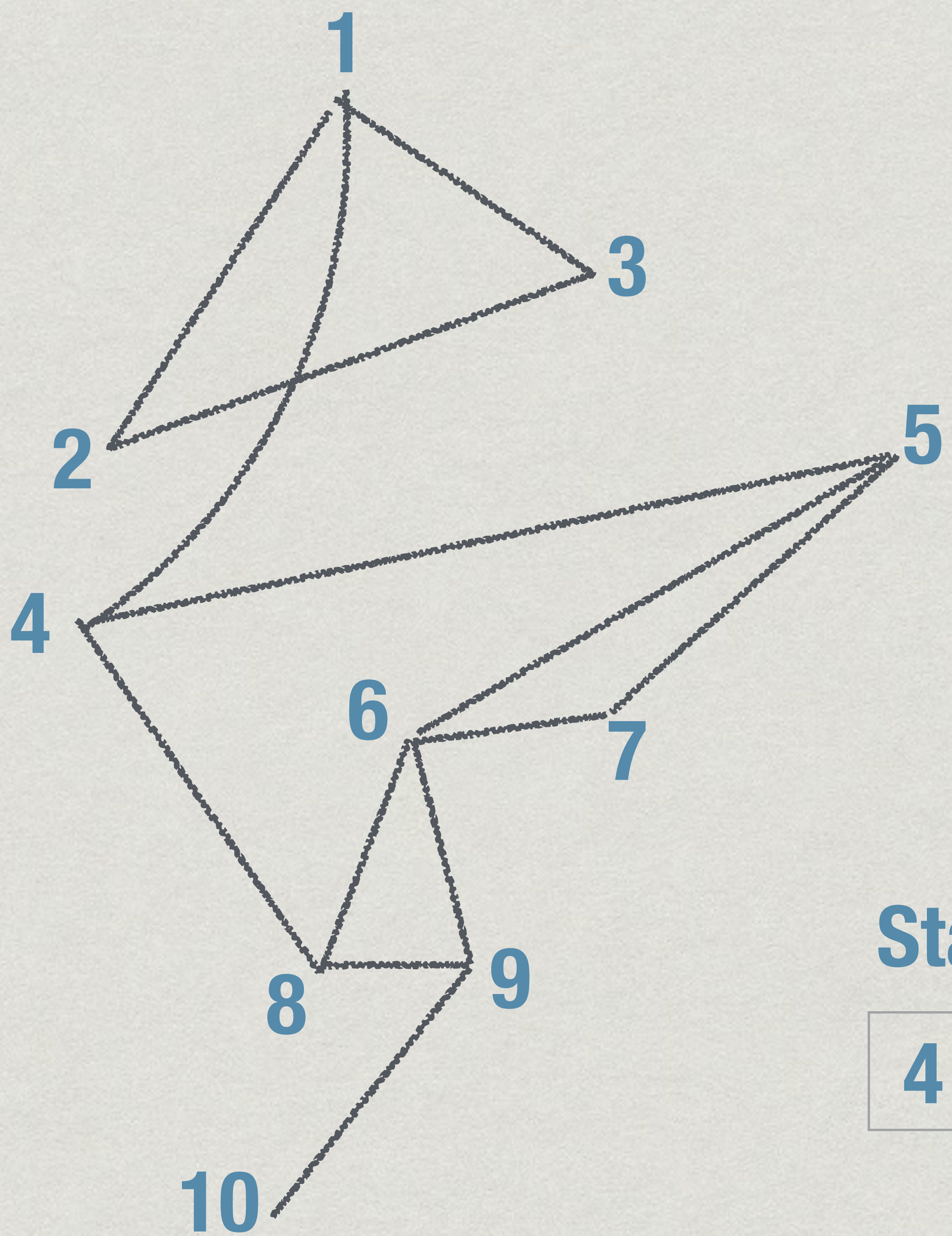
1	1
2	1
3	1
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

4	1	2							

Depth first search

Start at 4



Visited

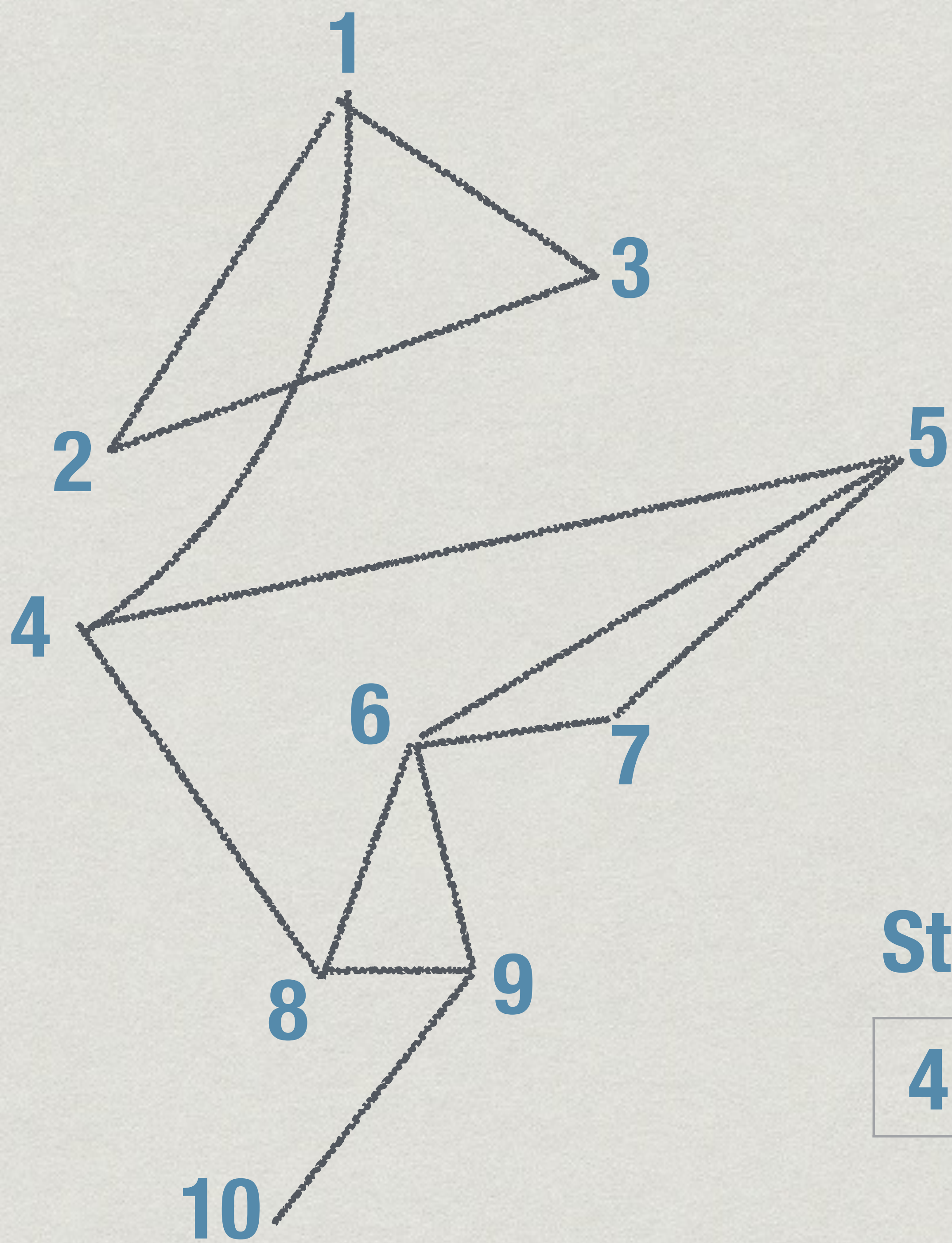
1	1
2	1
3	1
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

4	1								

Depth first search

Start at 4



Visited

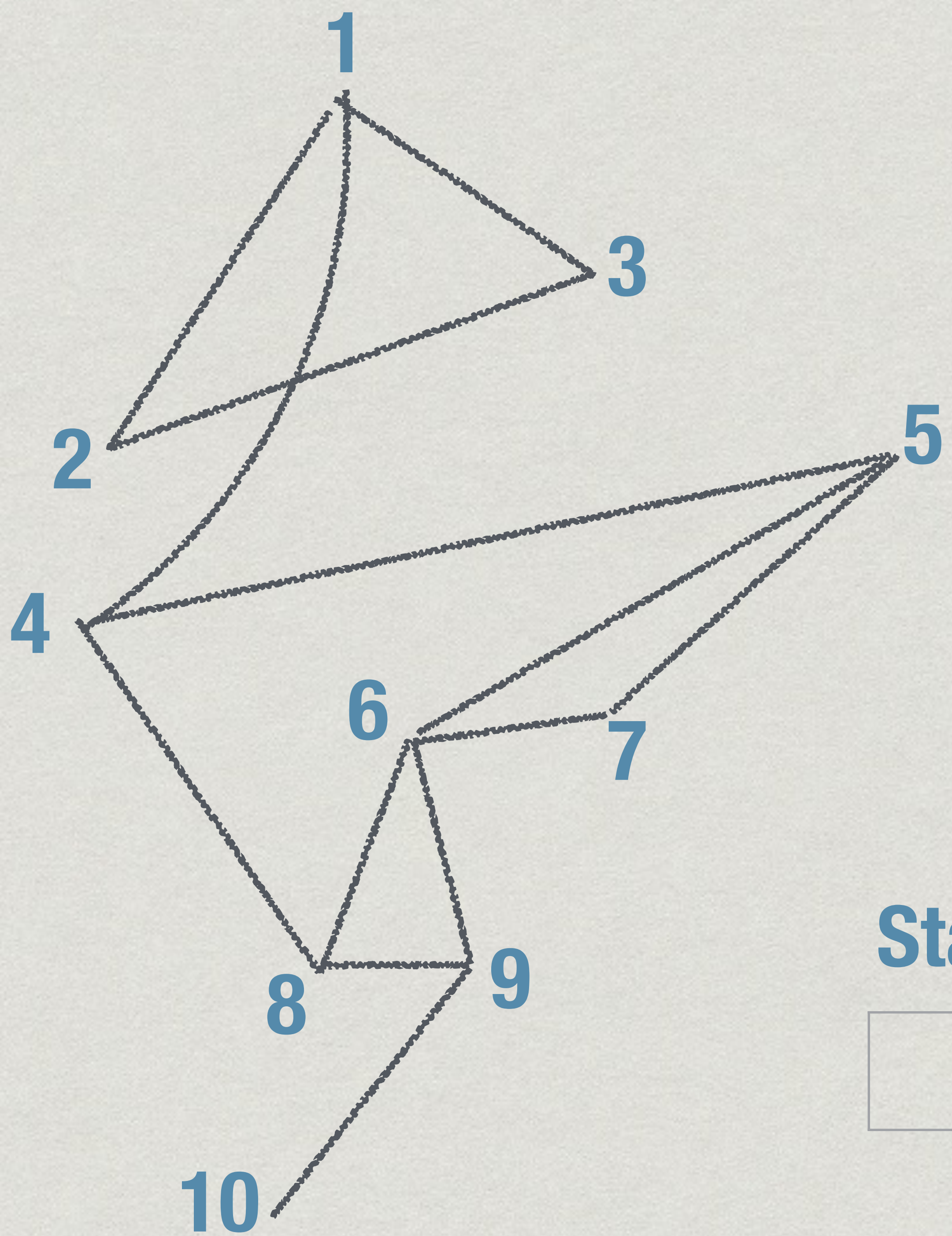
1	1
2	1
3	1
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

4									

Depth first search

Start at 4



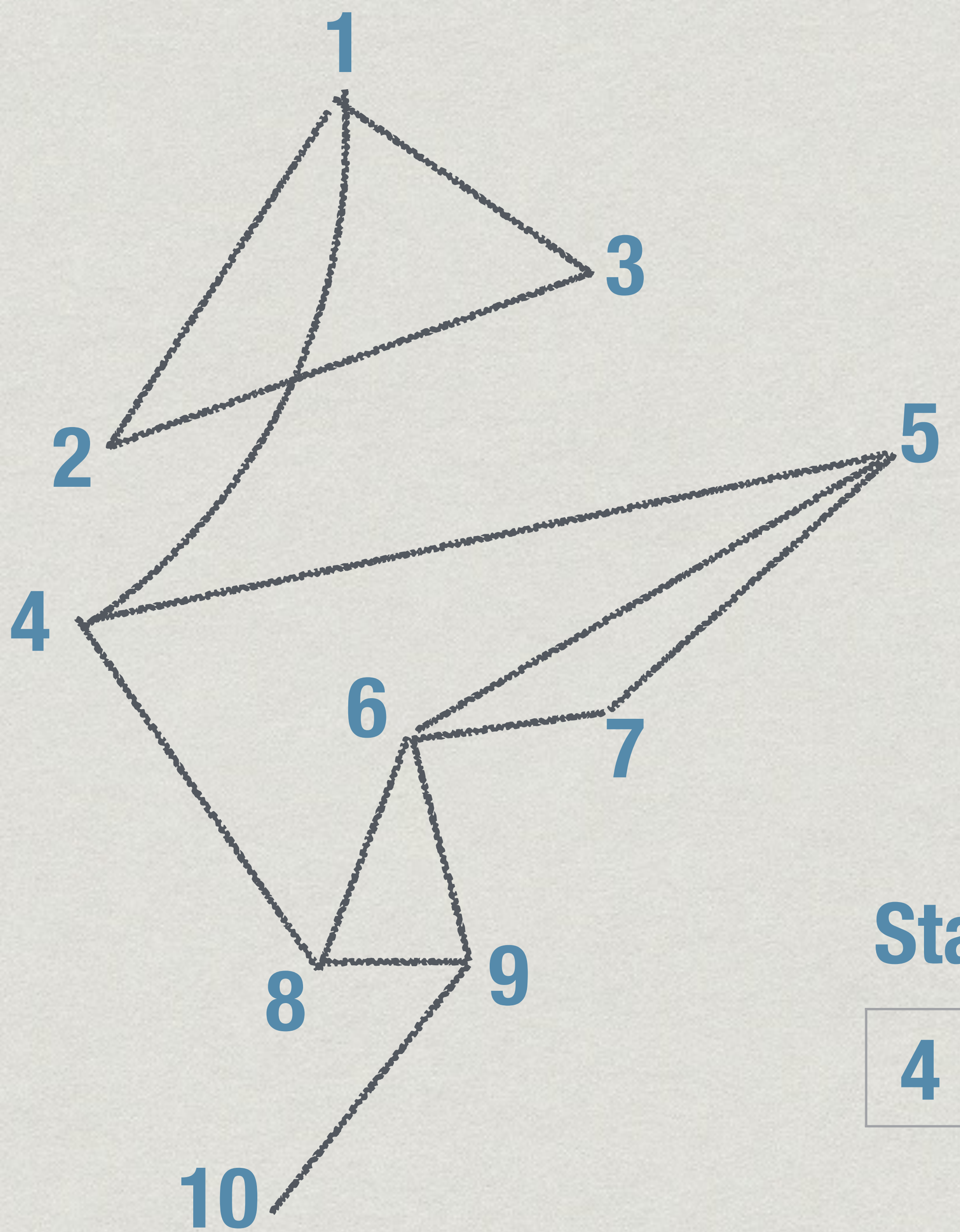
Visited

1	1
2	1
3	1
4	1
5	
6	
7	
8	
9	
10	

Stack of suspended vertices

Depth first search

Start at 4



Visited

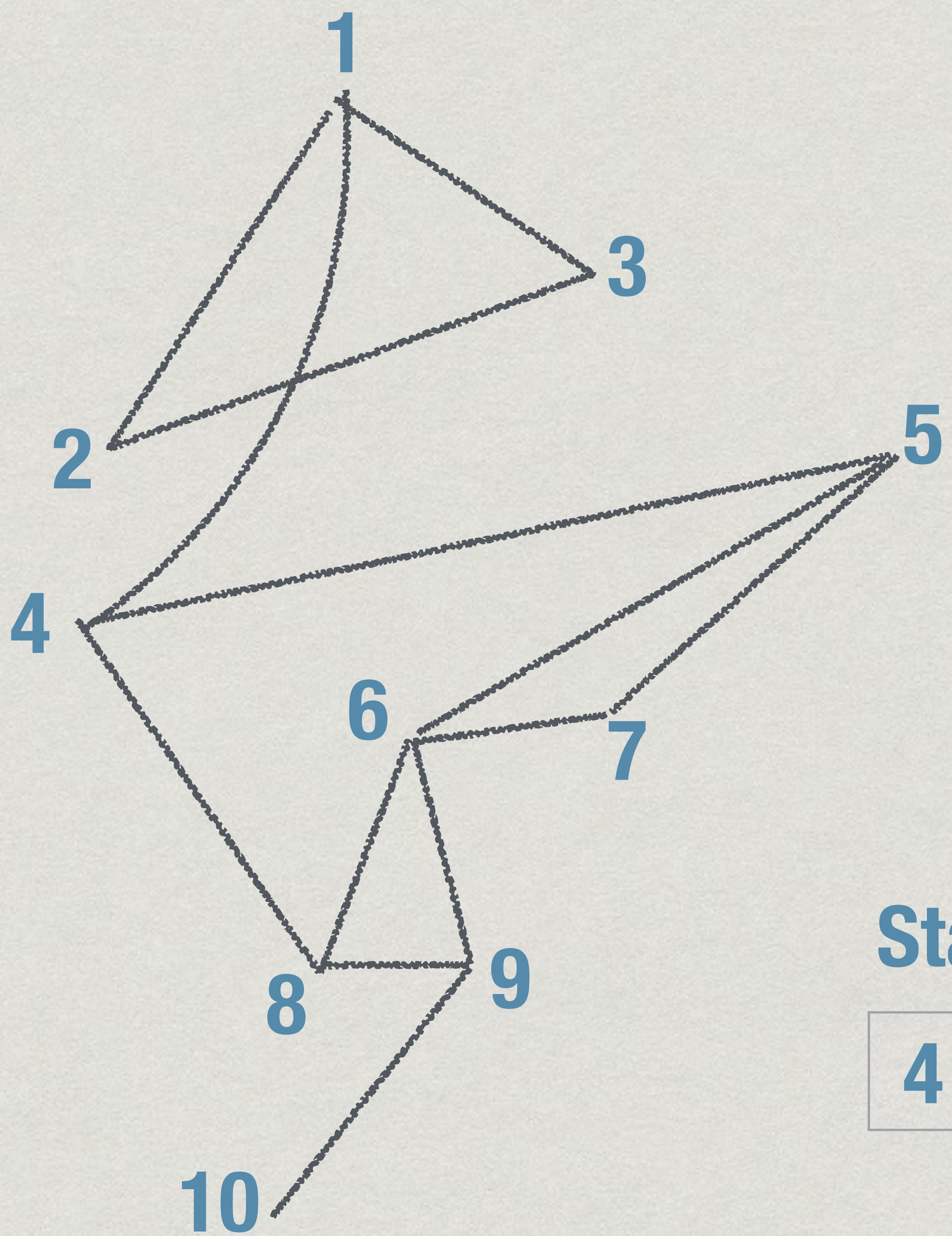
1	1
2	1
3	1
4	1
5	1
6	
7	
8	
9	
10	

Stack of suspended vertices

4									

Depth first search

Start at 4



Visited

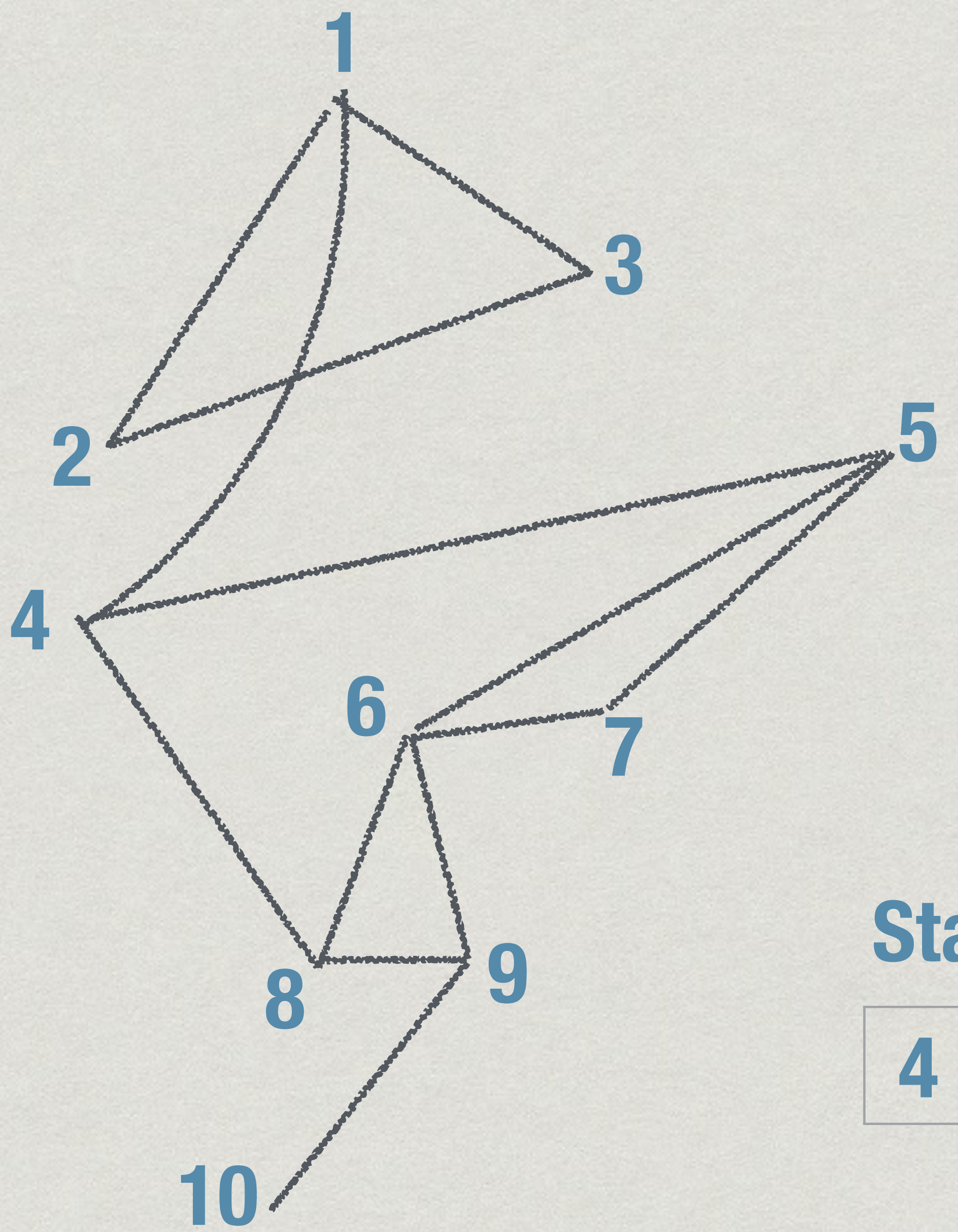
1	1
2	1
3	1
4	1
5	1
6	1
7	
8	
9	
10	

Stack of suspended vertices

4	5								

Depth first search

Start at 4



Visited

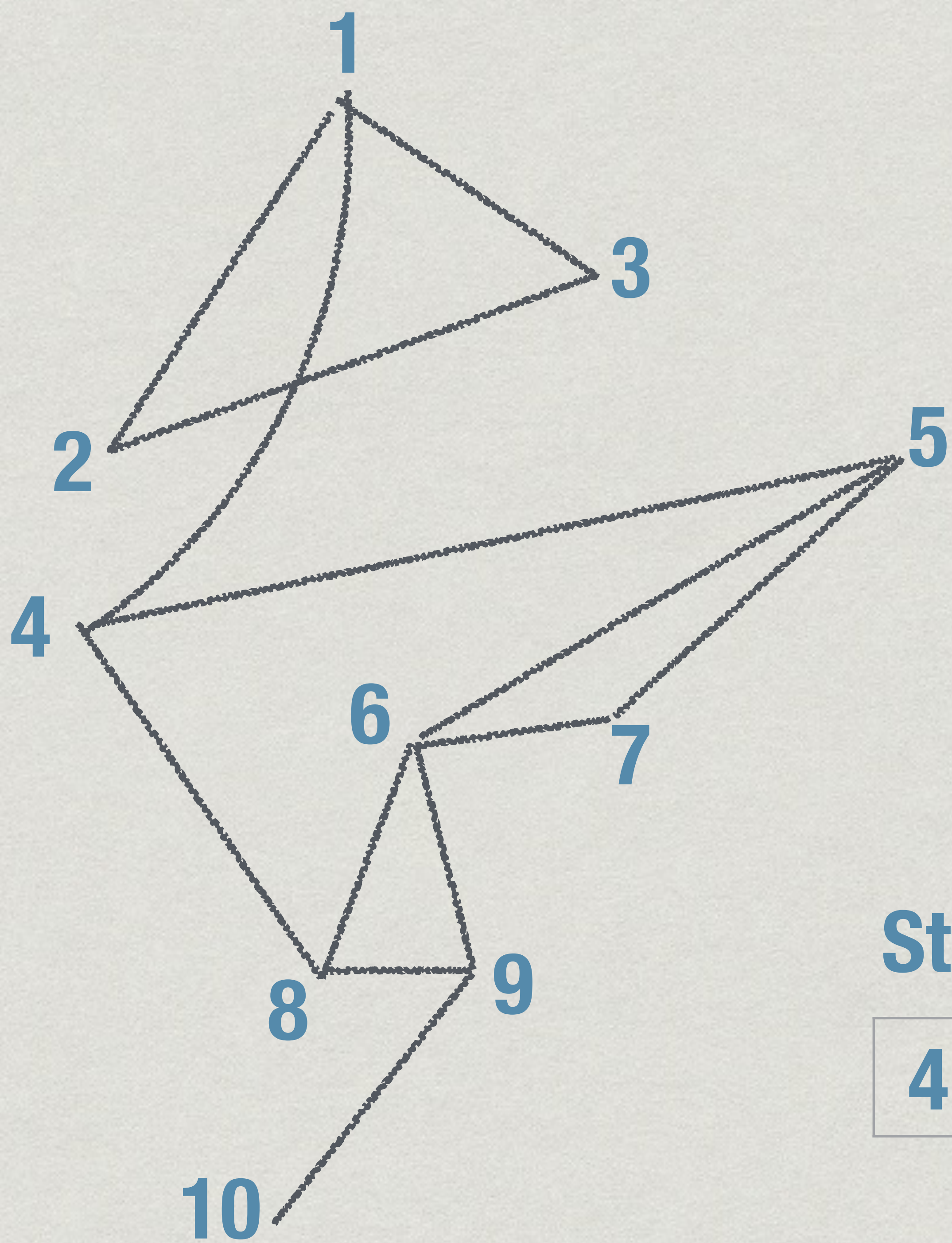
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	
9	
10	

Stack of suspended vertices

4	5	6							

Depth first search

Start at 4



Visited

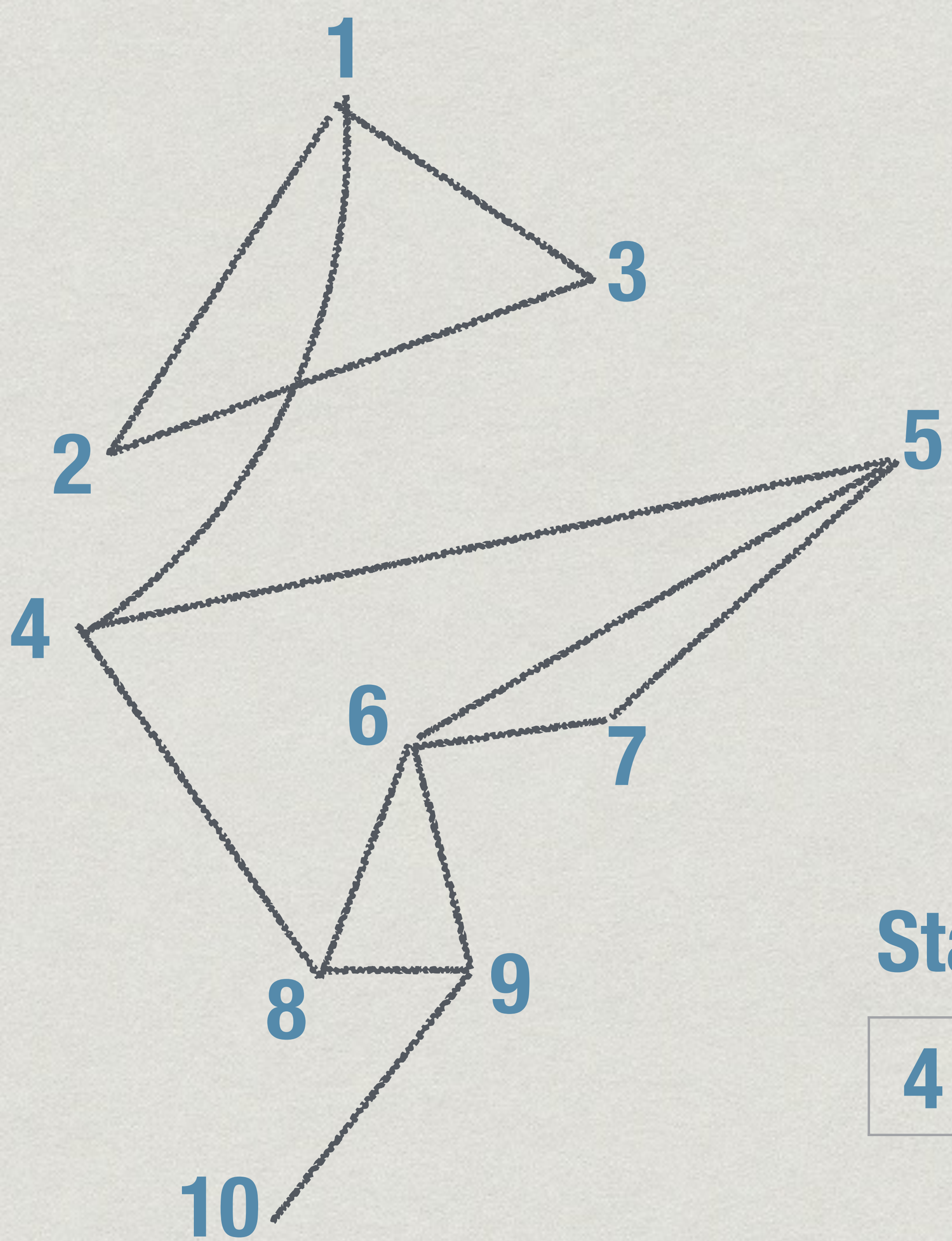
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	
9	
10	

Stack of suspended vertices

4	5								

Depth first search

Start at 4



Visited

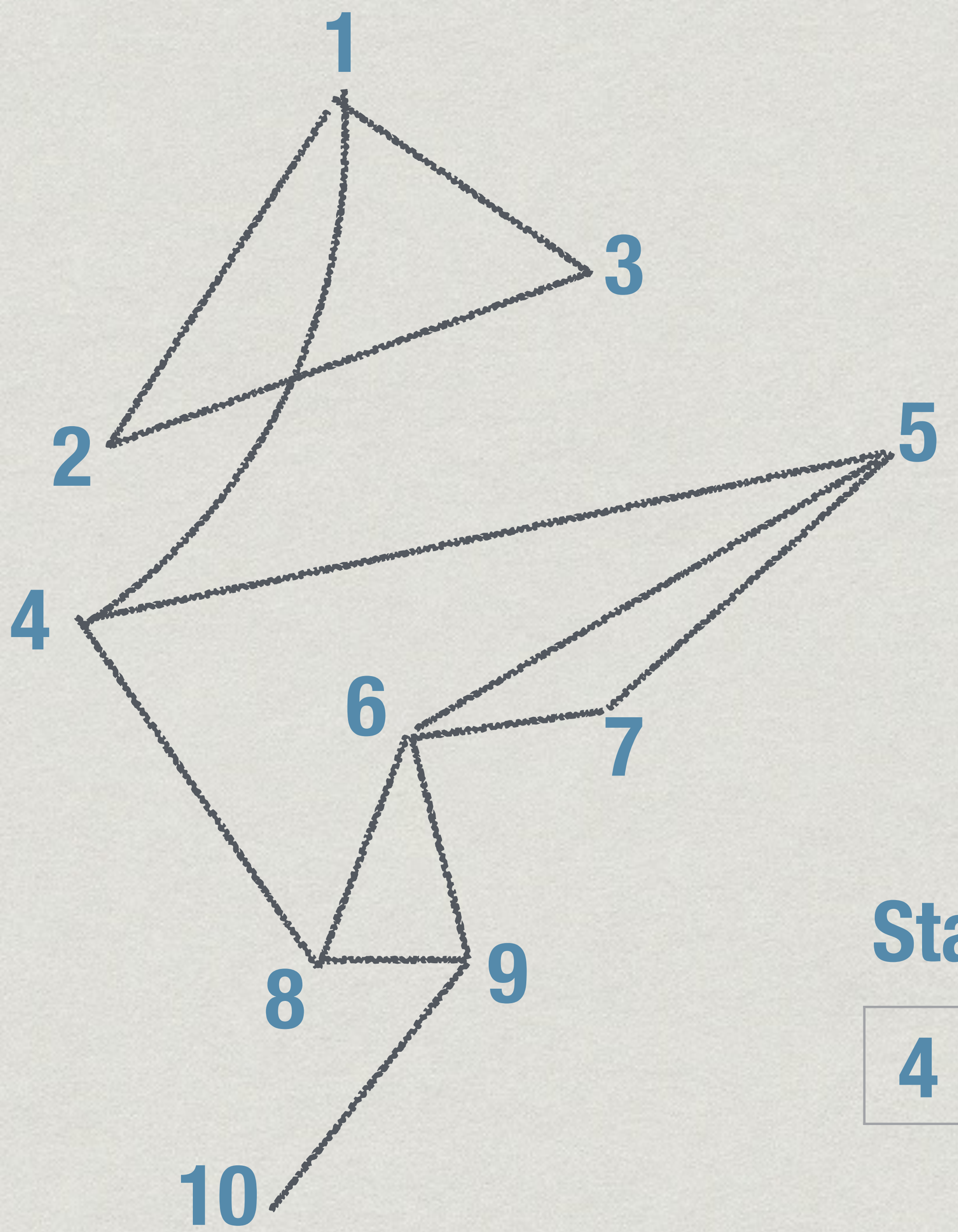
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	
10	

Stack of suspended vertices

4	5	6							

Depth first search

Start at 4



Visited

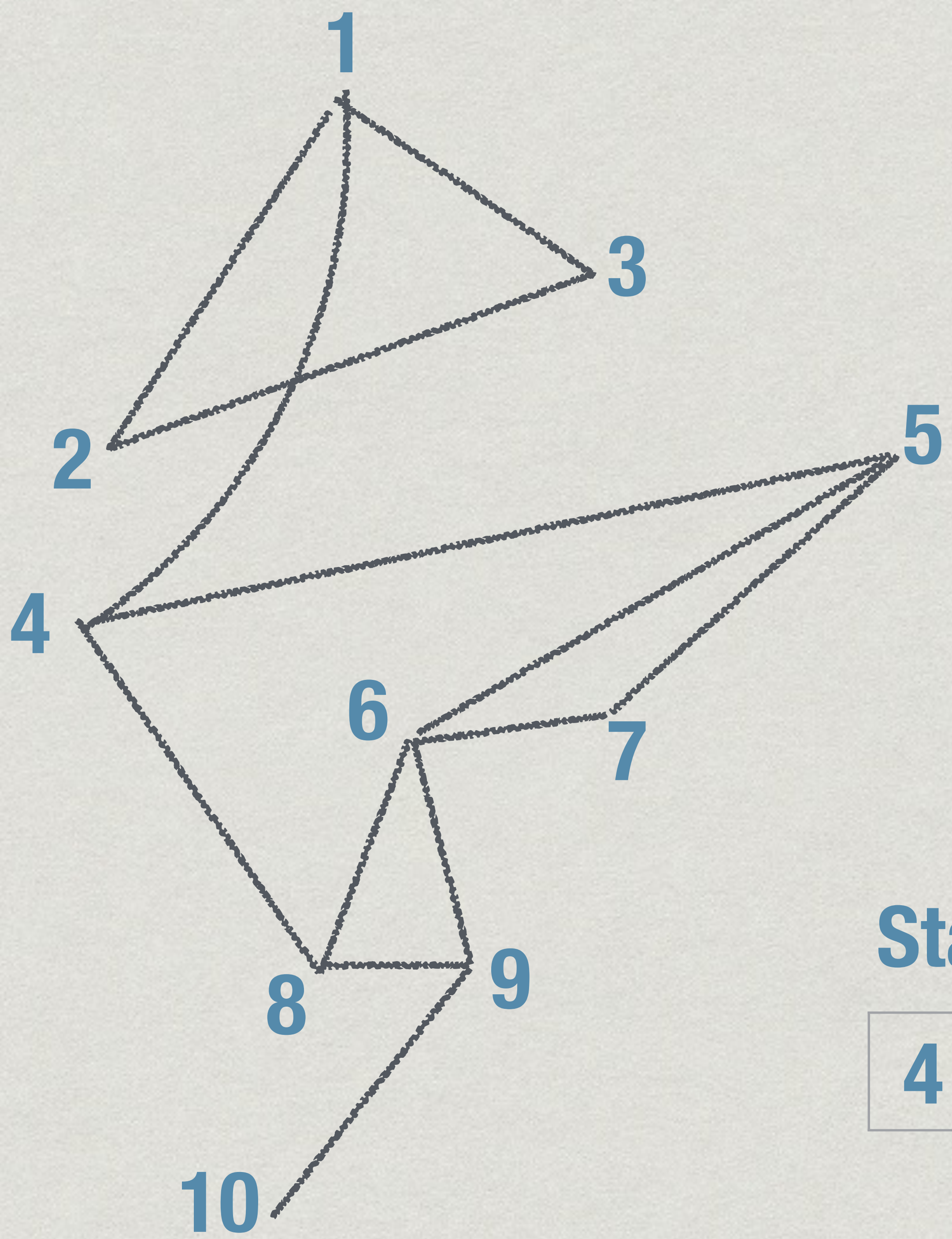
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	

Stack of suspended vertices

4	5	6	8						

Depth first search

Start at 4



Visited

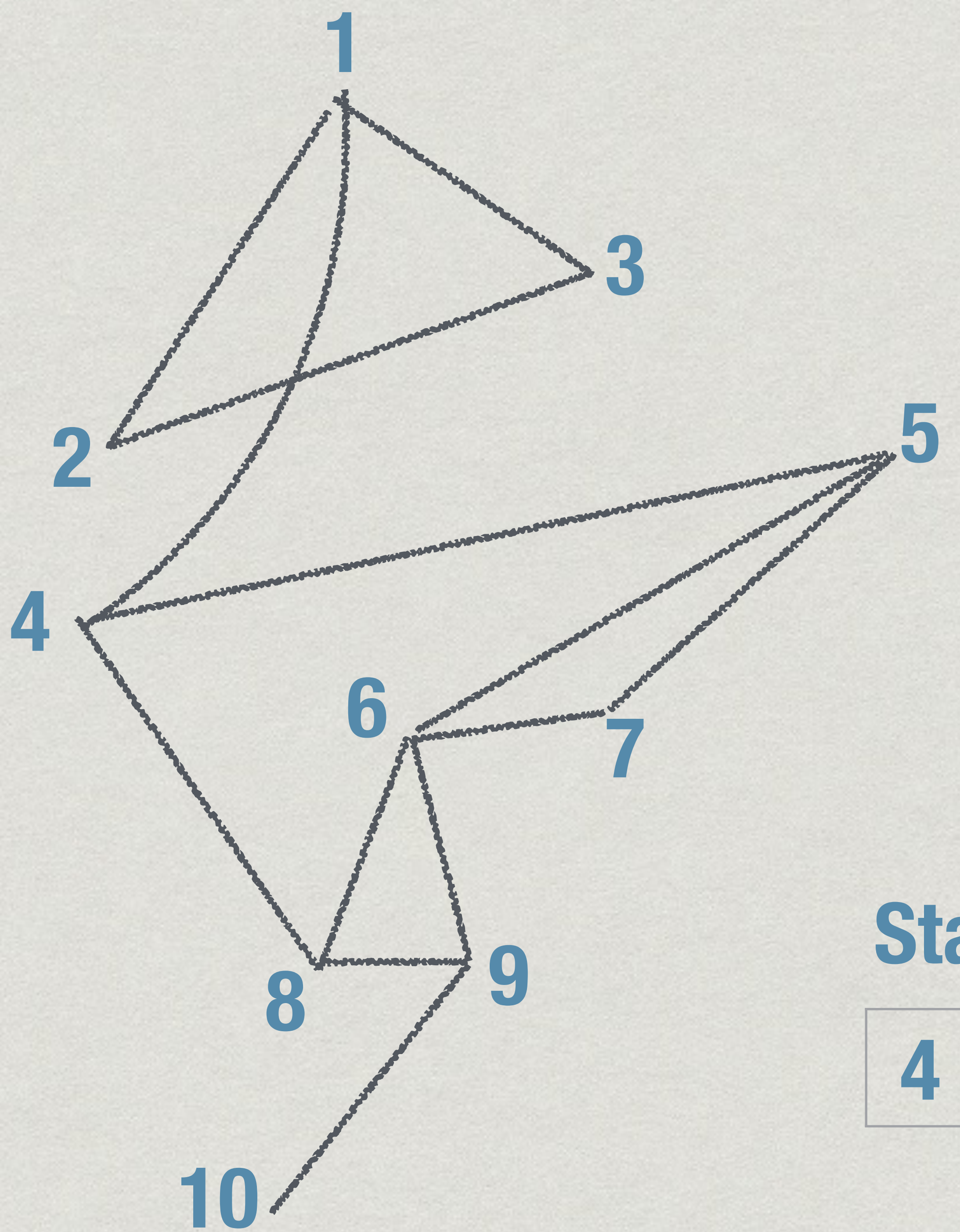
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Stack of suspended vertices

4	5	6	8	9					

Depth first search

Start at 4



Visited

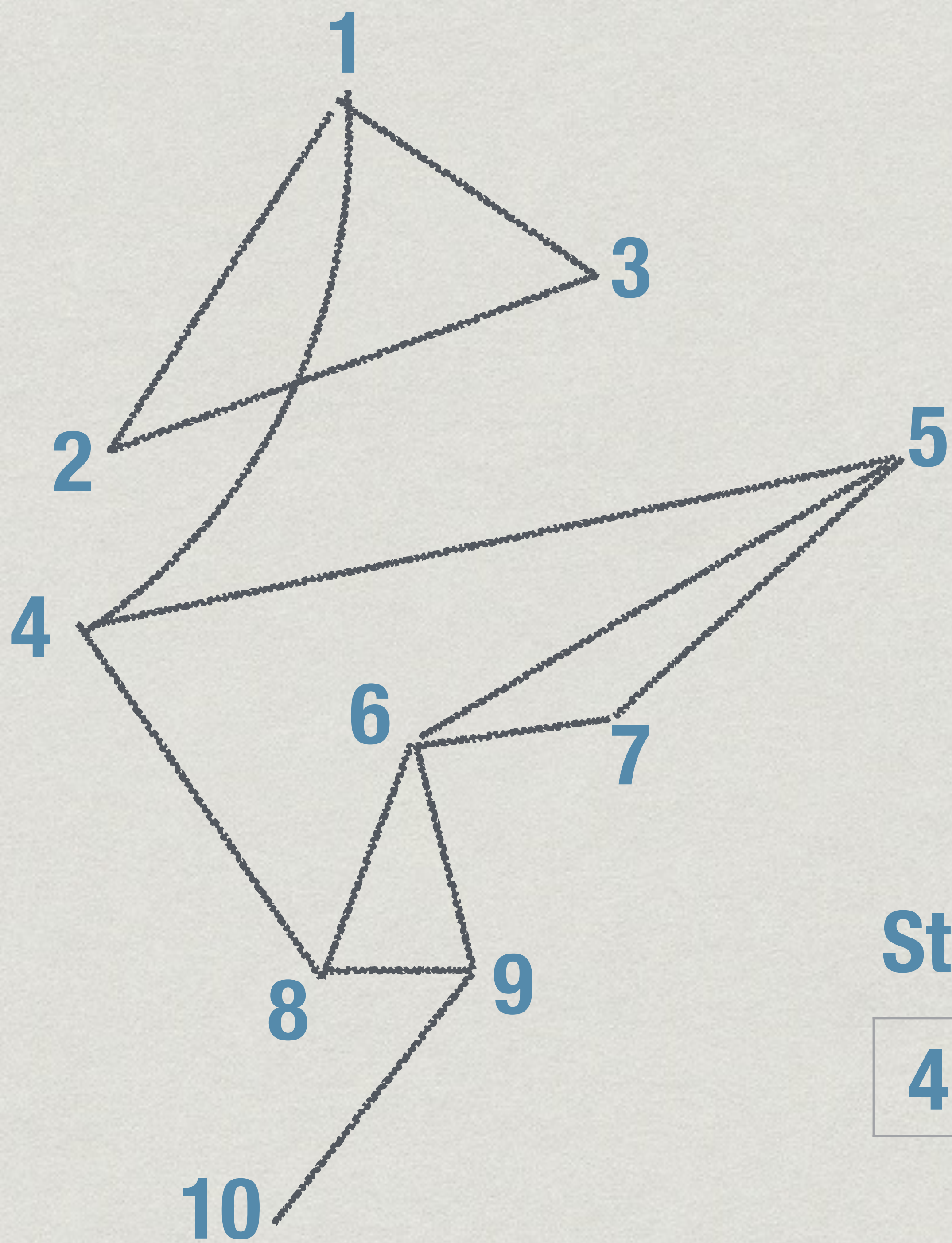
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Stack of suspended vertices

4	5	6	8						

Depth first search

Start at 4



Visited

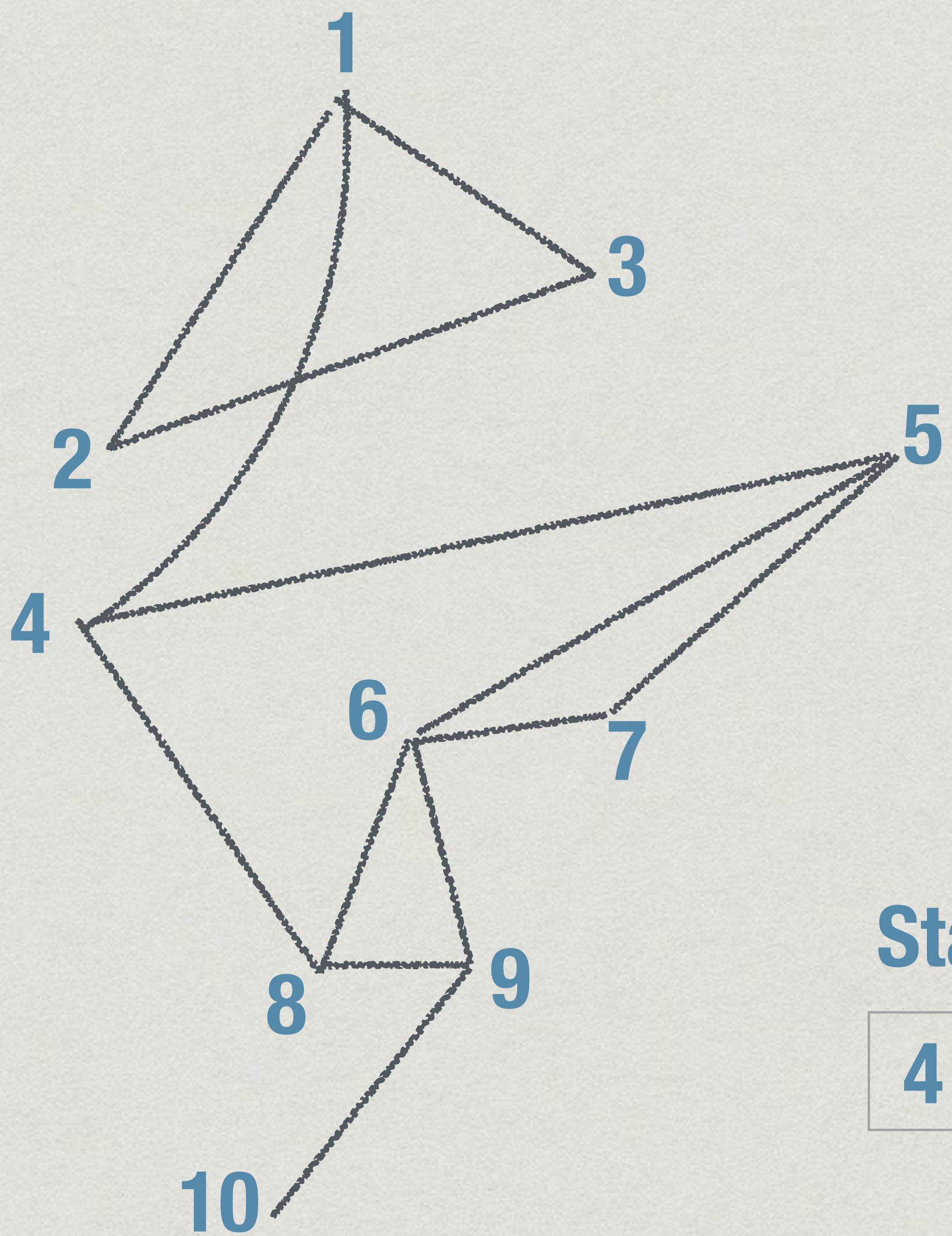
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Stack of suspended vertices

4	5	6							

Depth first search

Start at 4



Visited

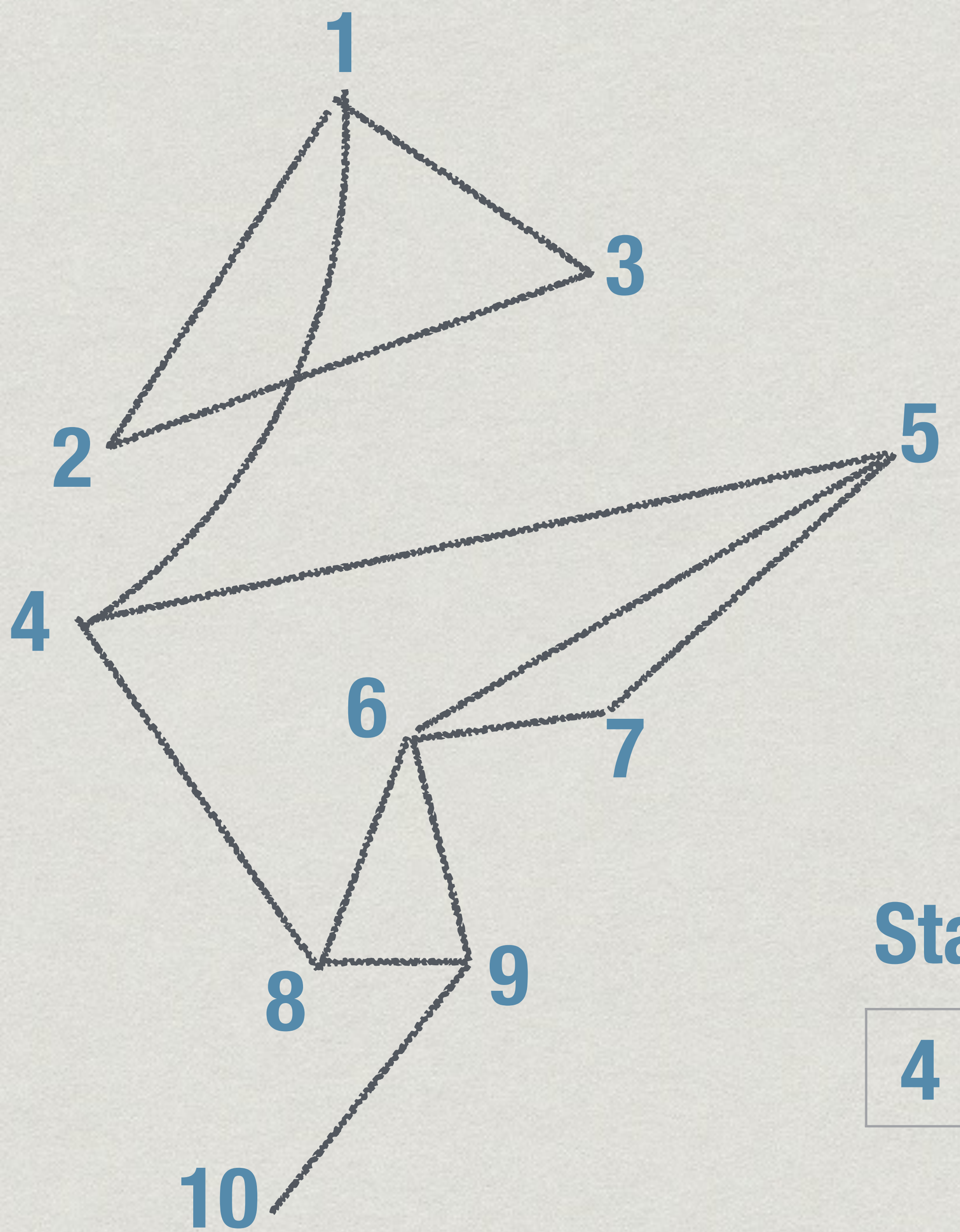
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Stack of suspended vertices

4	5								

Depth first search

Start at 4



Visited

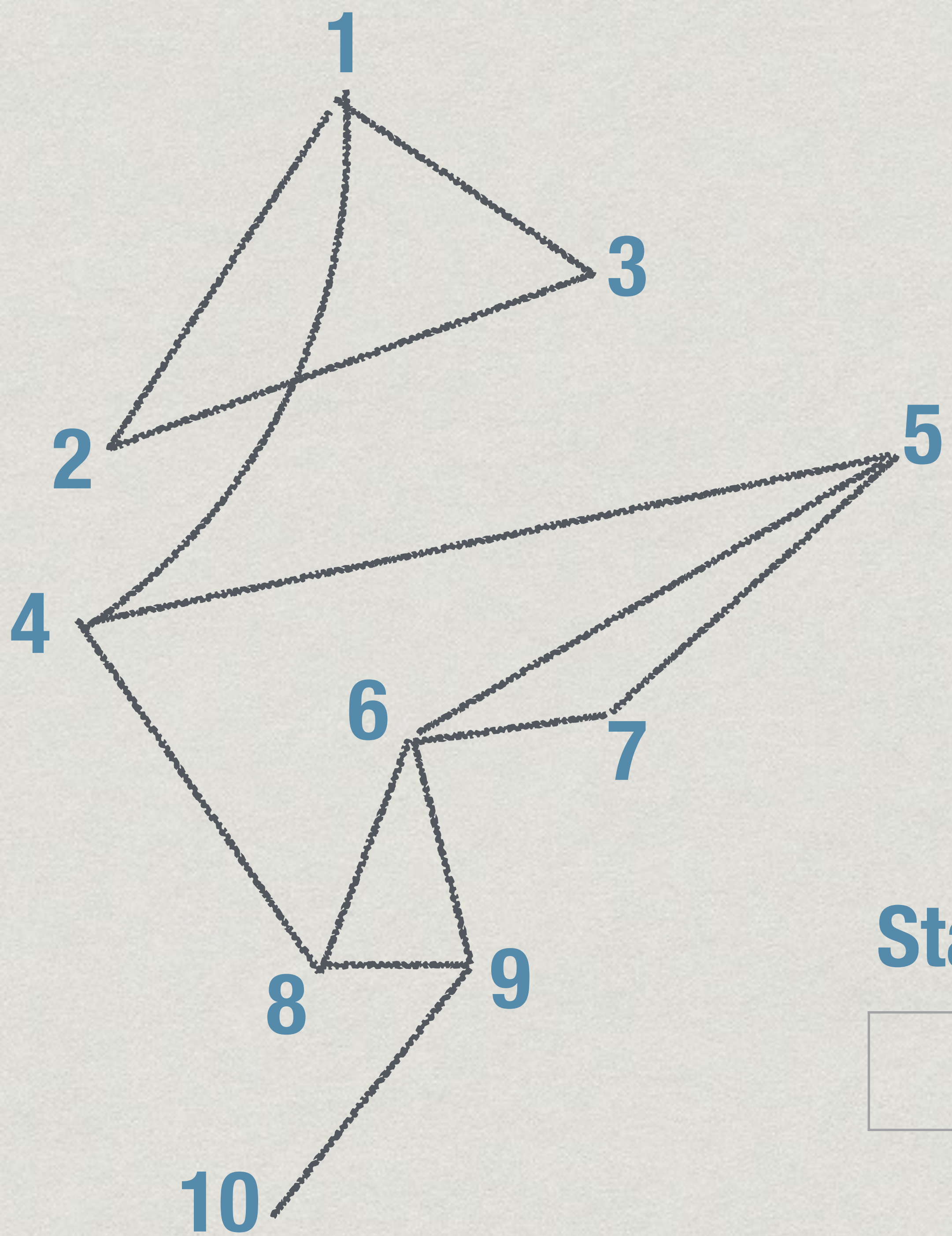
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Stack of suspended vertices

4									

Depth first search

Start at 4



Visited

1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1

Stack of suspended vertices

Depth first search

Depth first search

- DFS is most natural to implement recursively
    - For each unvisited neighbour j of i, call DFS(j)

Depth first search

- DFS is most natural to implement recursively
    - For each unvisited neighbour j of i, call DFS(j)
- No need to explicitly maintain a stack
    - Stack is maintained implicitly by recursive calls

Depth first search

```
//Initialization
   for j = 1..n {visited[j] = 0; parent[j] = -1}

function DFS(i) // DFS starting from vertex i

   //Mark i as visited
   visited[i] = 1

   //Explore each neighbour of i recursively
   for each (i,j) in E
      if visited[j] == 0
         parent[j] = i
         DFS(j)
```

Complexity of DFS

Complexity of DFS

* Each vertex marked and explored exactly once

Complexity of DFS

- Each vertex marked and explored exactly once
- DFS(j) need to examine all neighbours of j

Complexity of DFS

- Each vertex marked and explored exactly once
- DFS(j) need to examine all neighbours of j
- In adjacency matrix, scan row j: n entries
  - Overall $O(n^2)$

Complexity of DFS

- Each vertex marked and explored exactly once
- DFS(j) need to examine all neighbours of j
- In adjacency matrix, scan row j: n entries
  - Overall $O(n^2)$
- With adjacency list, scanning takes O(m) time across all vertices
  - Total time is O(m+n), like BFS

Properties of DFS

Properties of DFS

- Paths discovered by DFS are not shortest paths, unlike BFS

Properties of DFS

* Paths discovered by DFS are not shortest paths, unlike BFS
* Why use DFS at all?

Properties of DFS

- Paths discovered by DFS are not shortest paths, unlike BFS
- Why use DFS at all?
- Many useful features can be extracted from recording the order in which DFS visited vertices

Properties of DFS

- Paths discovered by DFS are not shortest paths, unlike BFS
- Why use DFS at all?
- Many useful features can be extracted from recording the order in which DFS visited vertices
  - **DFS numbering**

Properties of DFS

- Paths discovered by DFS are not shortest paths, unlike BFS
- Why use DFS at all?
- Many useful features can be extracted from recording the order in which DFS visited vertices
  - **DFS numbering**
  - Maintain a counter

Properties of DFS

- Paths discovered by DFS are not shortest paths, unlike BFS
- Why use DFS at all?
- Many useful features can be extracted from recording the order in which DFS visited vertices
  - **DFS numbering**
  - Maintain a counter
  - Increment and record counter value when entering and leaving a vertex.

Depth first search

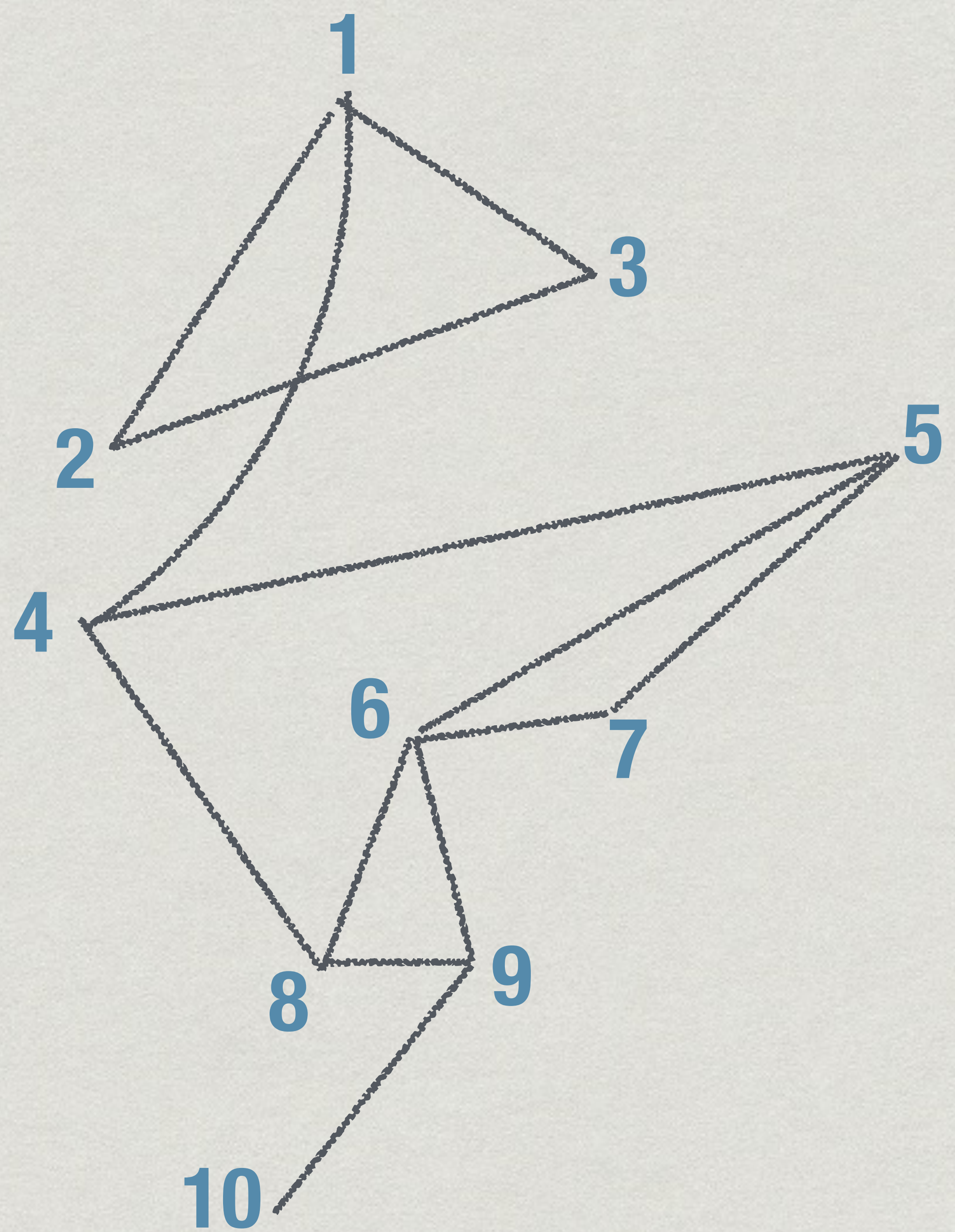
```
//Initialization
    for j = 1..n {visited[j] = 0; parent[j] = -1}
    count = 0

function DFS(i) // DFS starting from vertex i

    //Mark i as visited
    visited[i] = 1; pre[i] = count; count++

    //Explore each neighbours of i recursively
    for each (i,j) in E
        if visited[j] == 0
            parent[j] = i
            DFS(j)
            post[i] = count; count++
```

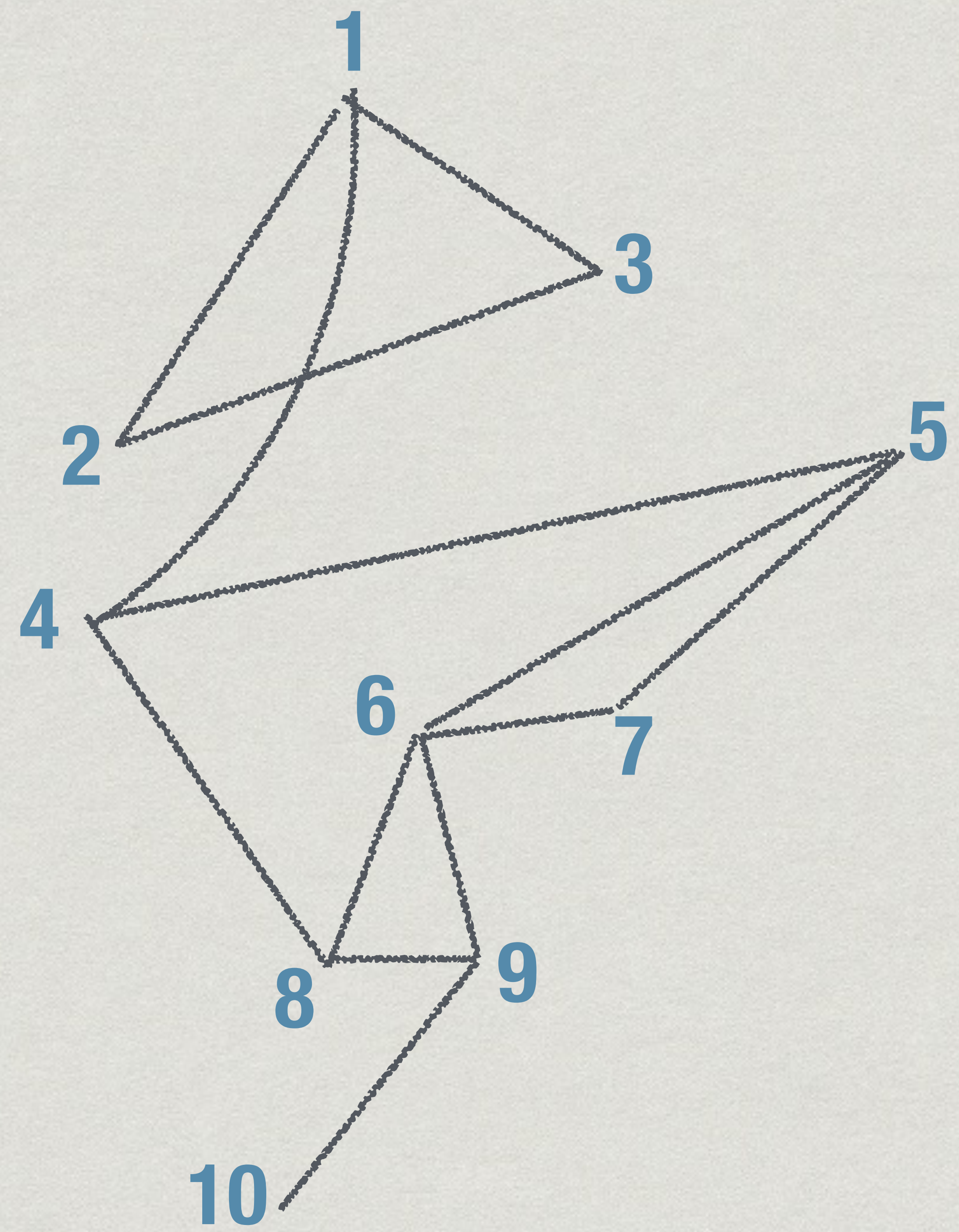
DFS numbering



DFS numbering

pre[i] and post[i] can be used to find

- if the graph has a **cycle** — i.e., a loop
- **cut vertex** — removal disconnects the graph
- ...



Summary

- BFS and DFS are two systematic ways to explore a graph
    - Both take time linear in the size of the graph with adjacency lists
- Recover paths by keeping parent information
- BFS can compute shortest paths, in terms of number of edges
- DFS numbering can reveal many interesting features