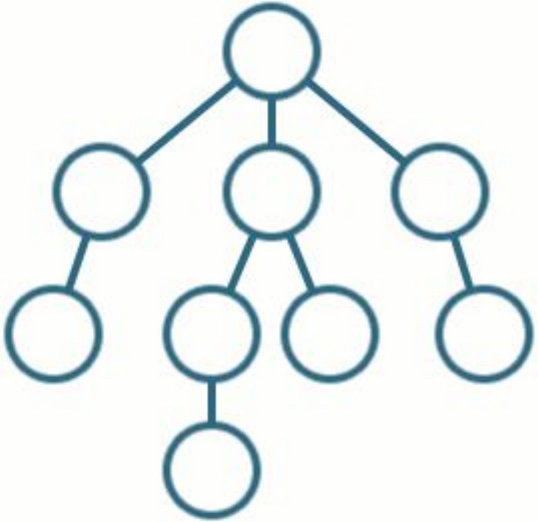
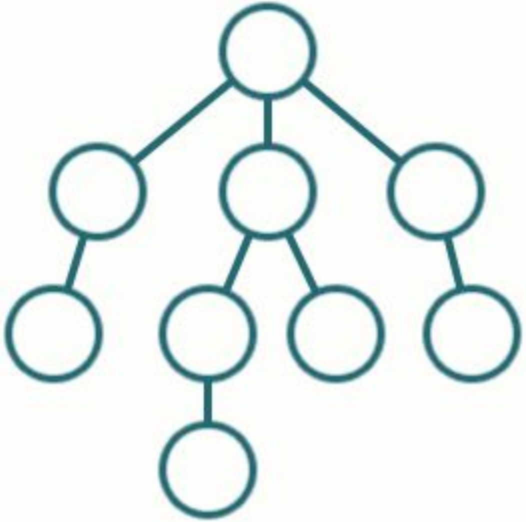
Breadth first search

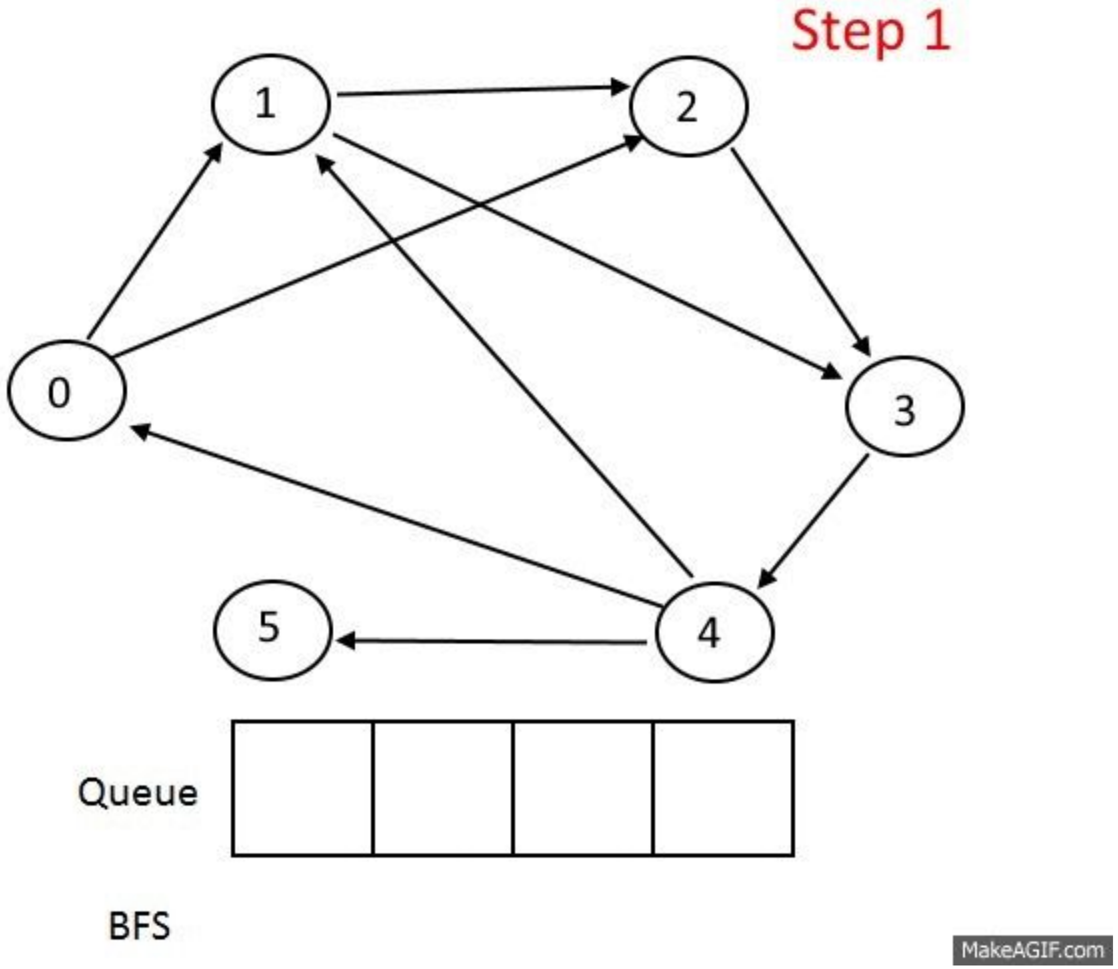
Graphs

DFS



BFS




Step 1
1
2
0
3
5
4
Queue
BFS
MakeAGIF.com

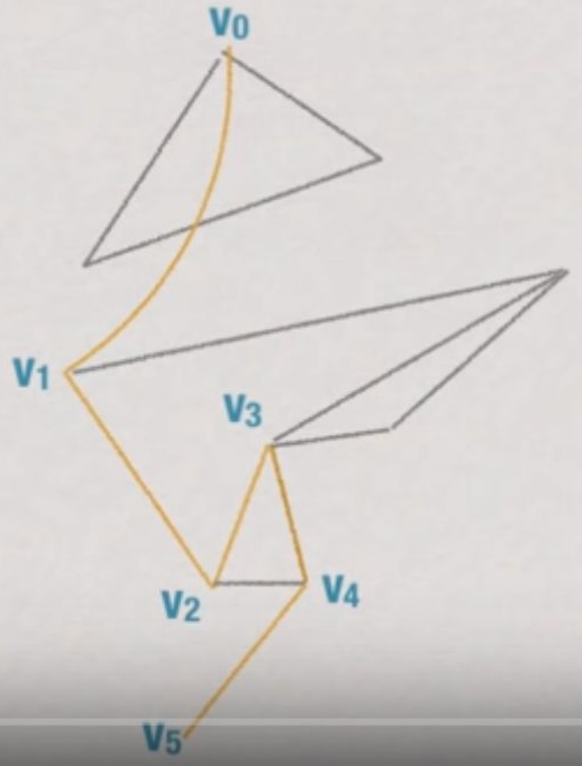
Graphs, formally

$G = (V,E)$

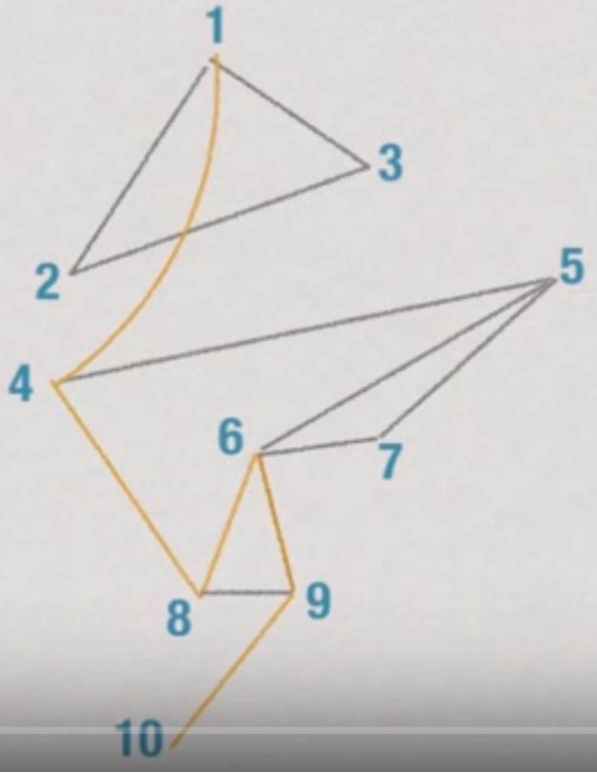
* Set of vertices V
* Set of edges E
  * E is a subset of pairs (v,v'): $E \subseteq V \times V$
  * Undirected graph: (v,v') and (v',v) are the same edge
  * Directed graph:
    * (v,v') is an edge from v to v'
    * Does not guarantee that (v',v) is also an edge

Finding a route

- Find a sequence of vertices $v_0, v_1, \ldots, v_k$ such that
  - v_0 is **source**
  - Each (v_i, v_{i+1}) is an edge in E
  - v_k is **target**

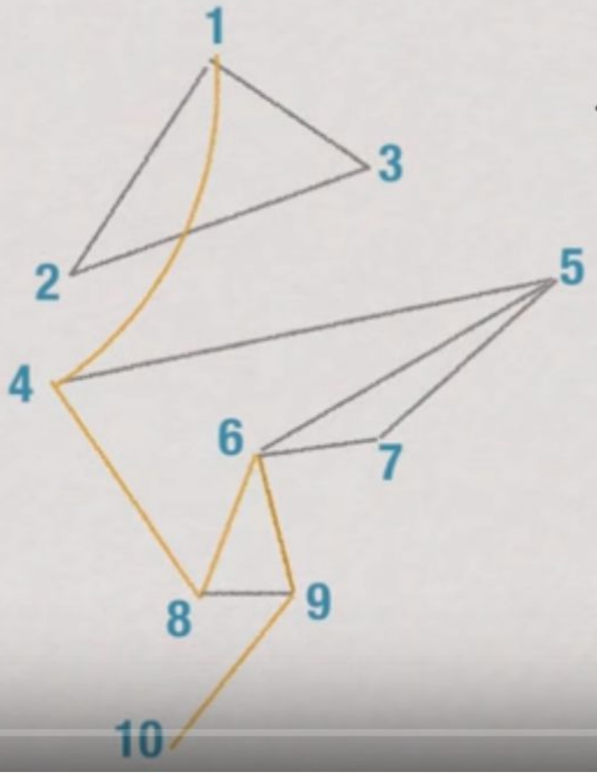


Adjacency matrix



	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	1	0	0	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	0	1	1	1	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	1	0	1	0	0	1	0
9	0	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	0	0	0	1	0

Adjacency list



- For each vertex, maintain a list of its neighbours

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

Finding a path

- Mark vertices that have been visited
- Keep track of vertices whose neighbours have already been explored
  - Avoid going round indefinitely in circles
- Two fundamental strategies: breadth first and depth first

Breadth first search

* Explore the graph level by level
  * First visit vertices one step away
  * Then two steps away
  * ...
* Remember which vertices have been **visited**
* Also keep track of vertices visited, but whose neighbours are yet to be **explored**

Breadth first search

- Recall that $V = \{1,2,\ldots,n\}$
- Array visited[i] records whether i has been visited
- When a vertex is visited for the first time, add it to a **queue** .
  - Explore vertices in the order they reach the queue

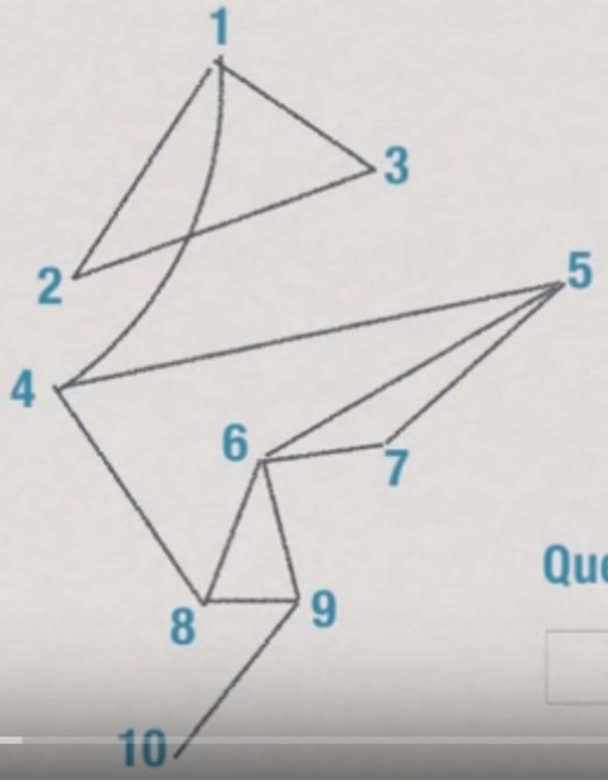
Breadth first search

- Exploring a vertex i:

```
for each edge (i,j)
  if visited[j] == 0
    visited[j] = 1
    append j to queue
```

- Initially, queue contains only source vertex
- At each stage, explore vertex at the head of the queue
- Stop when the queue becomes empty

Breadth first search

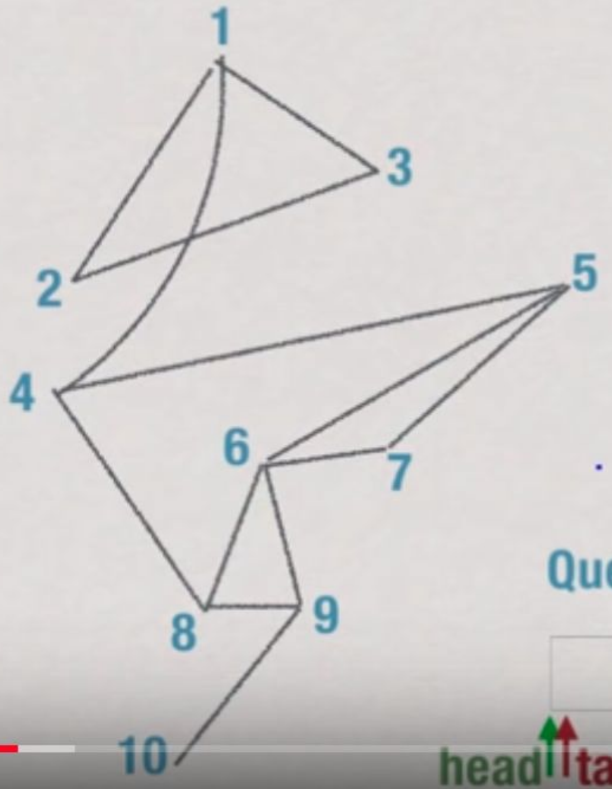


Visited

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Queue

Breadth first search

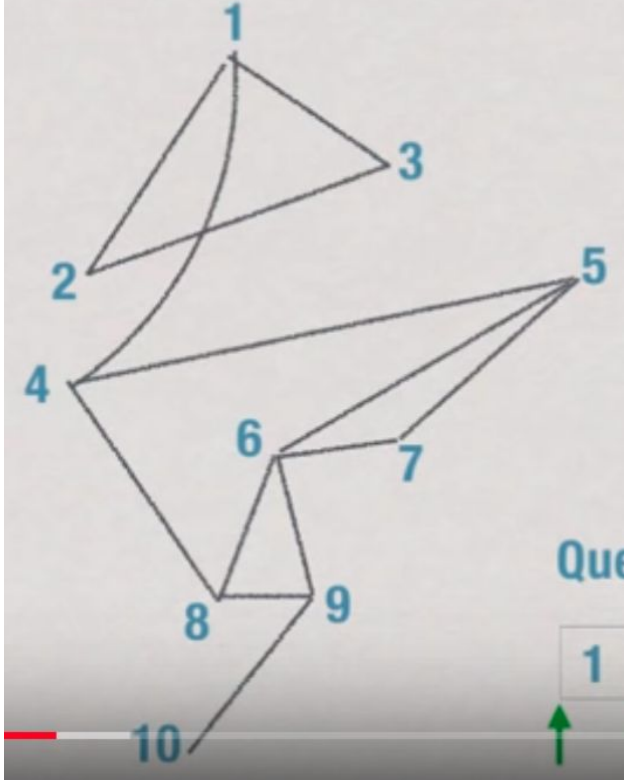


Visited

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Queue

Breadth first search



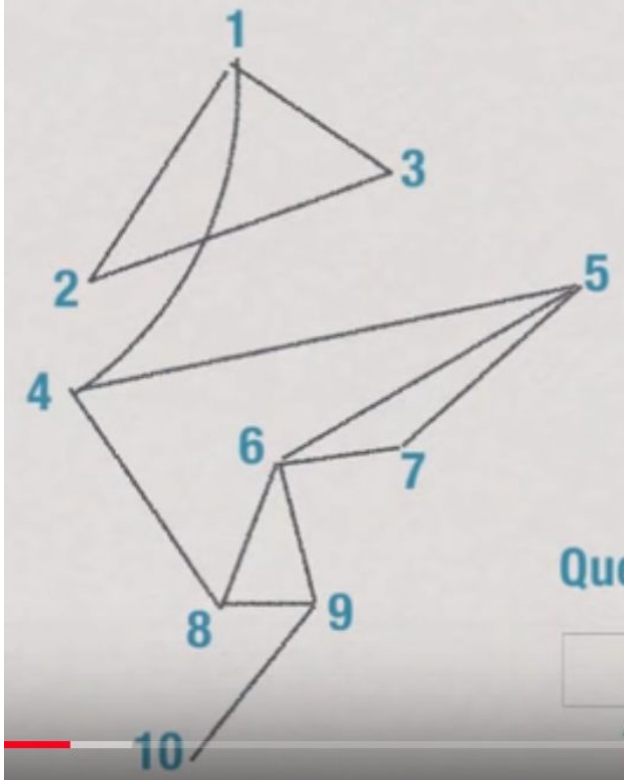
Visited

1	1
2	
3	
4	
5	
6	
7	
8	
9	
10	

Queue

1									

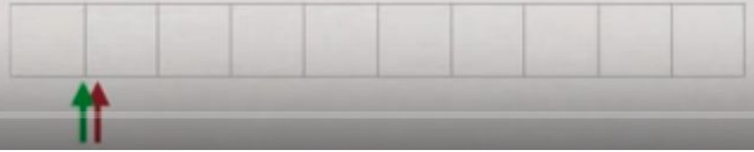
Breadth first search



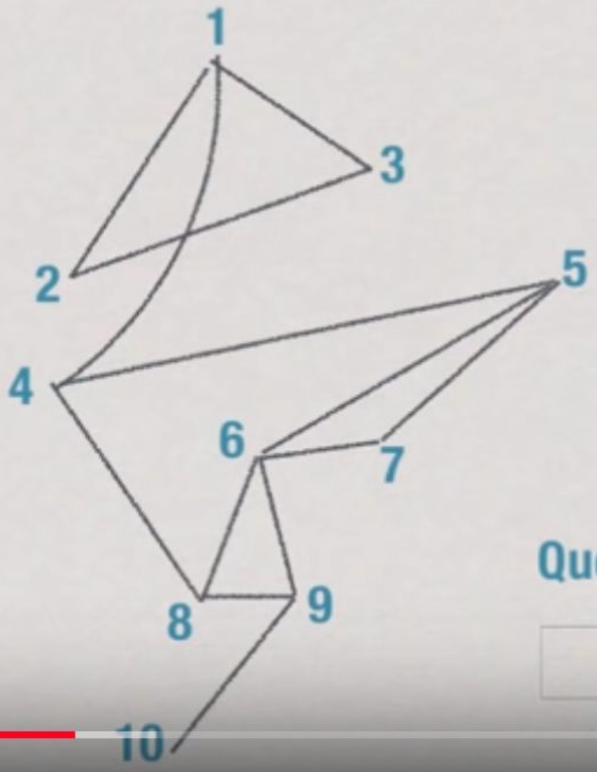
Visited

1	1
2	
3	
4	
5	
6	
7	
8	
9	
10	

Queue



Breadth first search



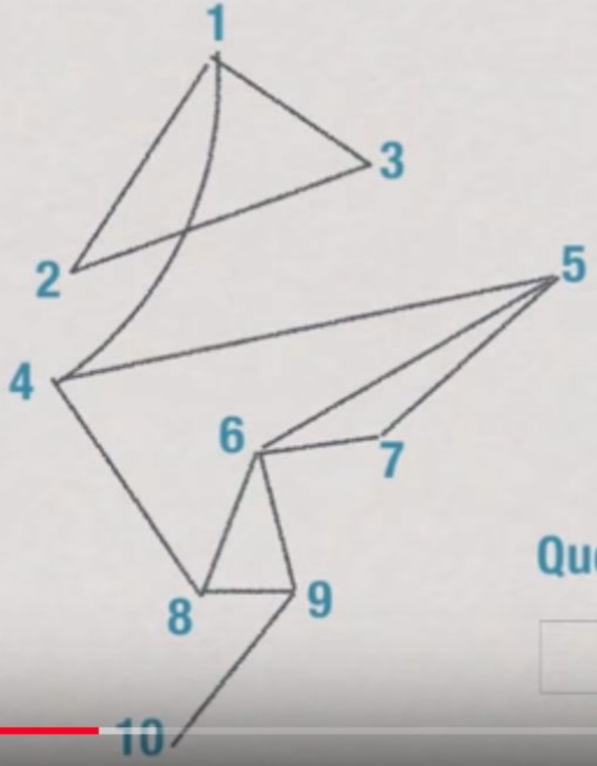
Visited

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

Queue

	2								

Breadth first search



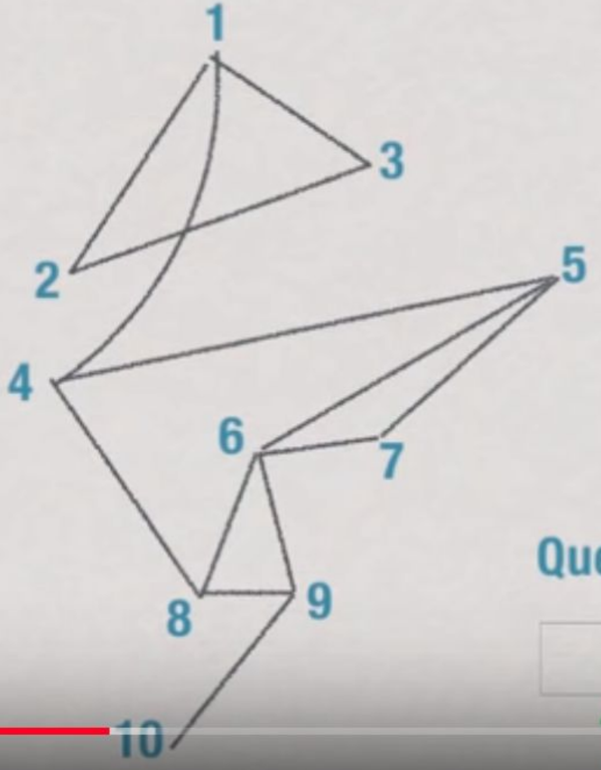
Visited

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

Queue

	2	3							

Breadth first search



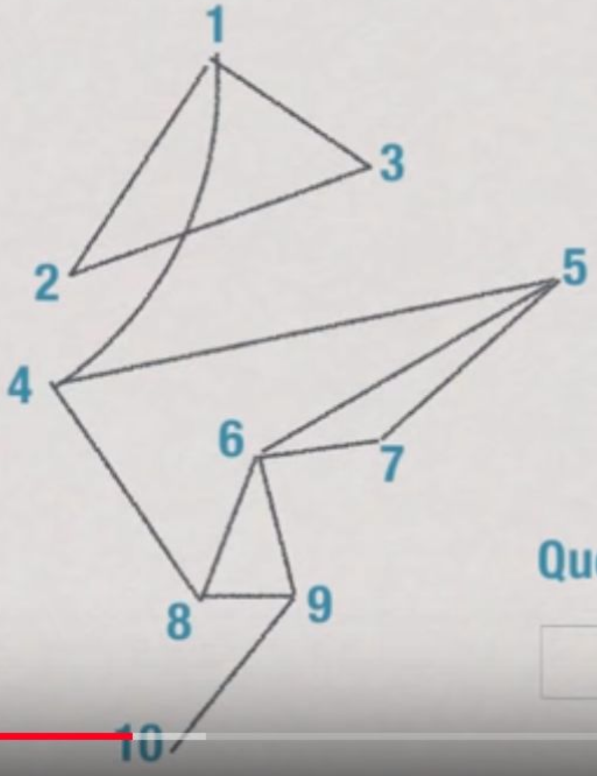
Visited

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

Queue

	2	3	4						

Breadth first search



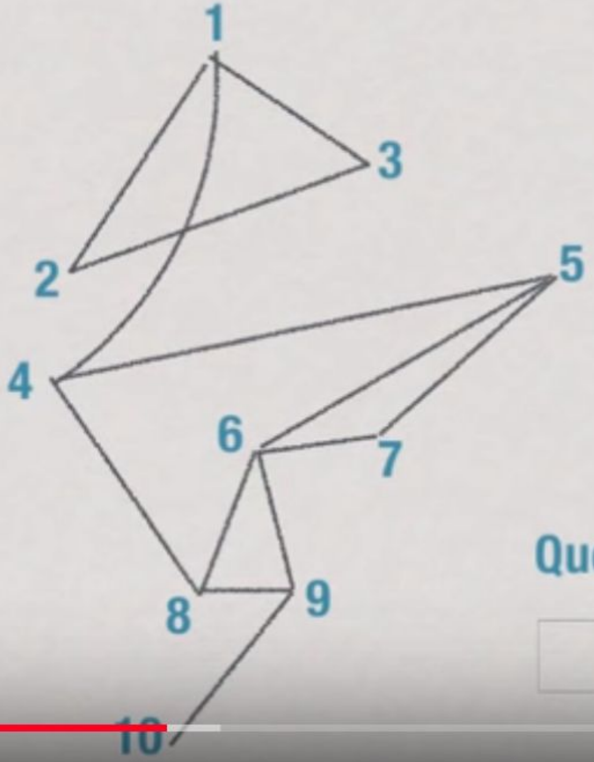
Visited

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

Queue

		3	4						

Breadth first search



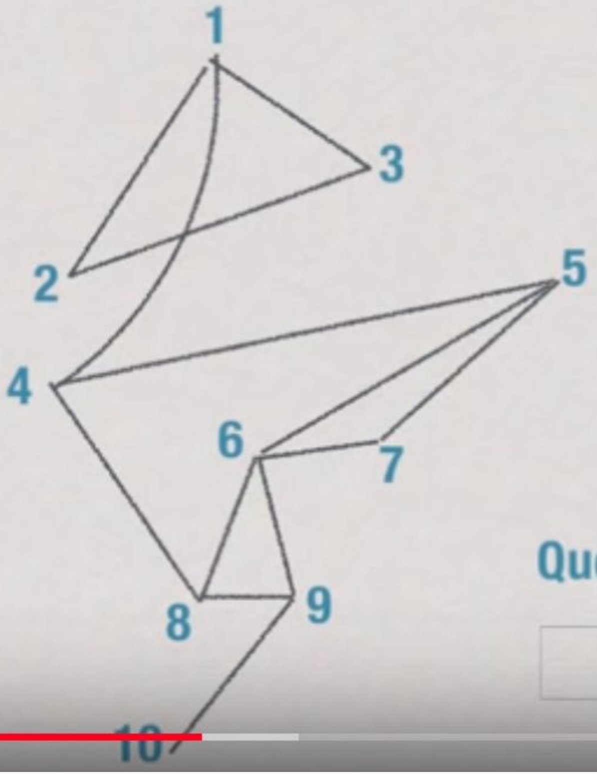
Visited

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

Queue

			4						

Breadth first search

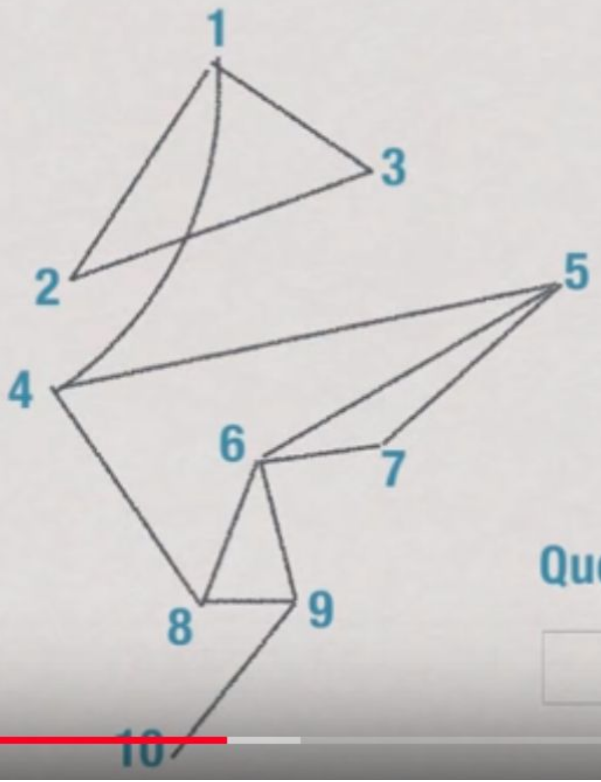


Visited

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

Queue

Breadth first search



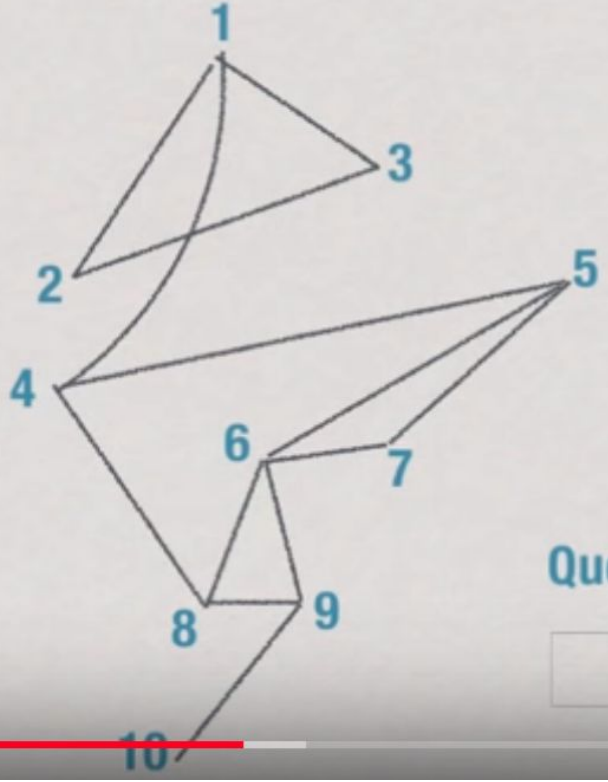
Visited

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

Queue

				5					

Breadth first search



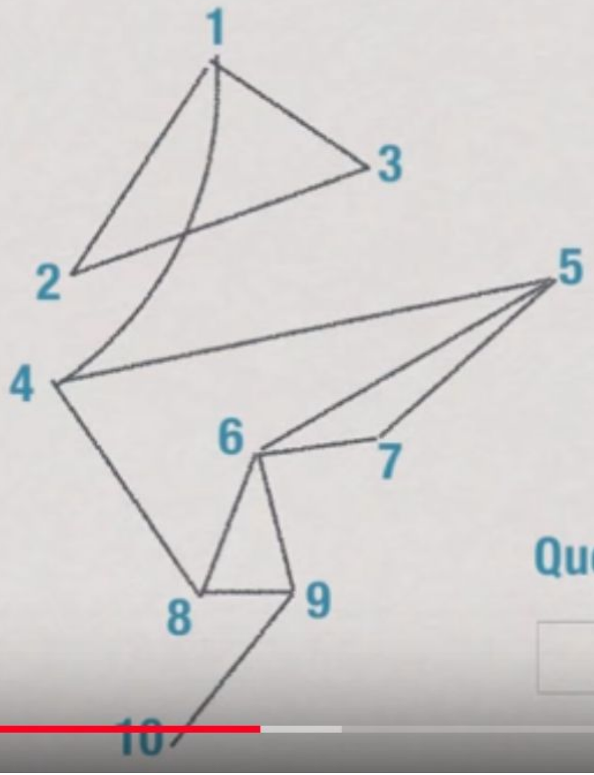
Visited

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

Queue

				5	8				

Breadth first search



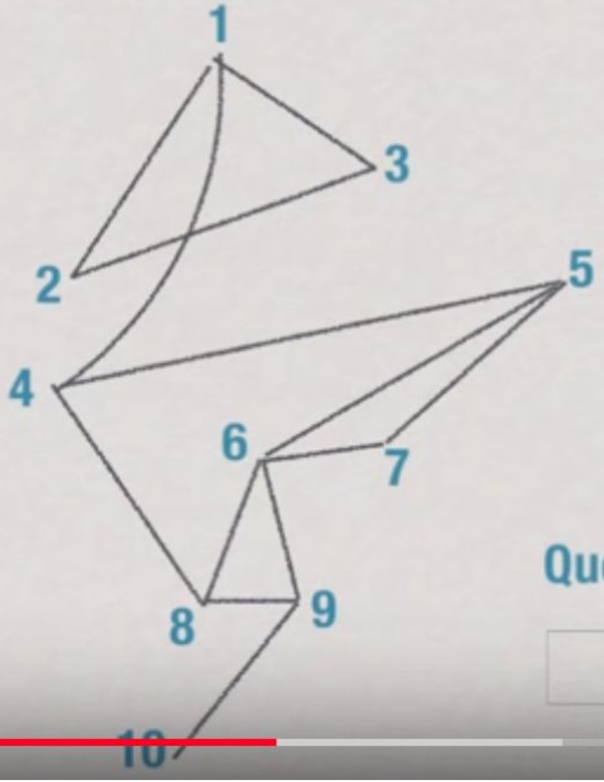
Visited

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

Queue

					8				

Breadth first search



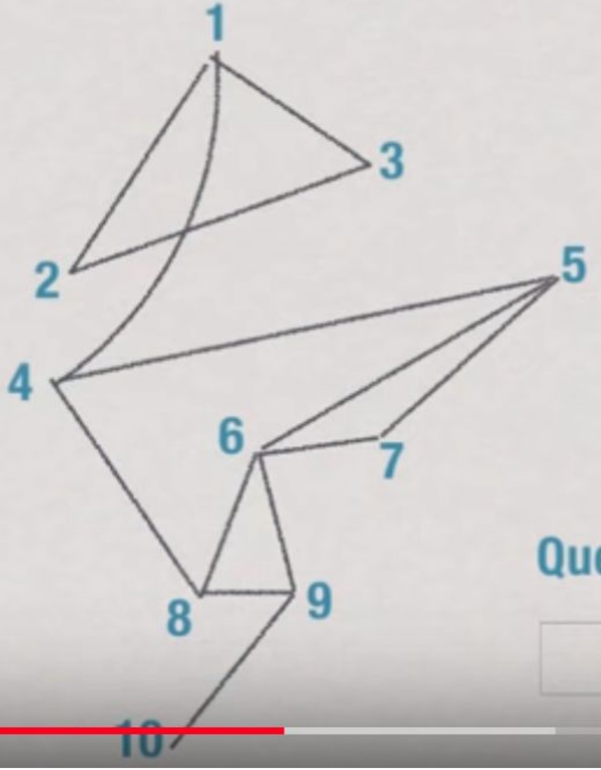
Visited

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

Queue

					8	6			

Breadth first search



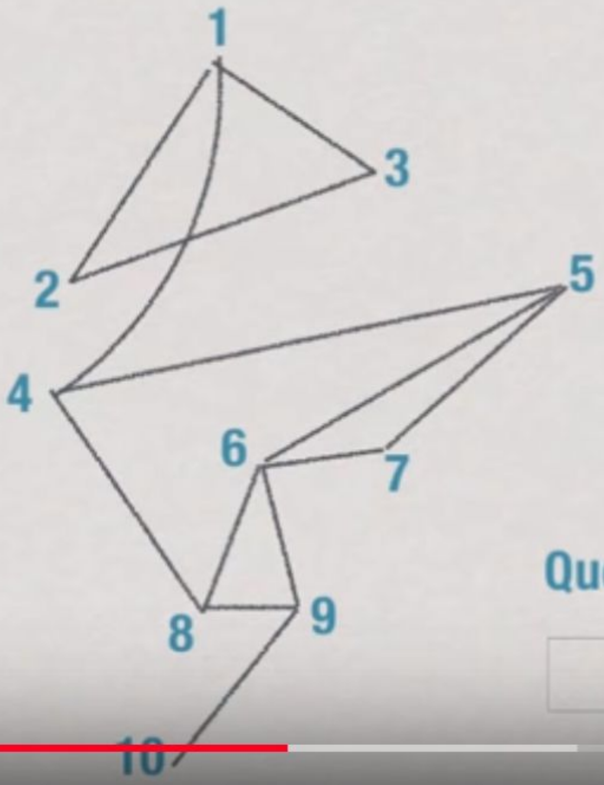
Visited

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

Queue

					8	6	7		

Breadth first search



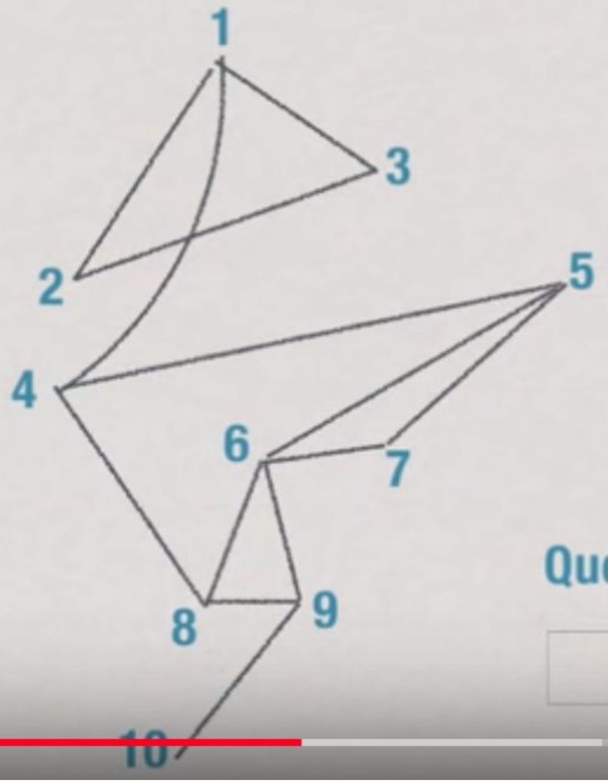
Visited

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

Queue

					8	6	7		

Breadth first search



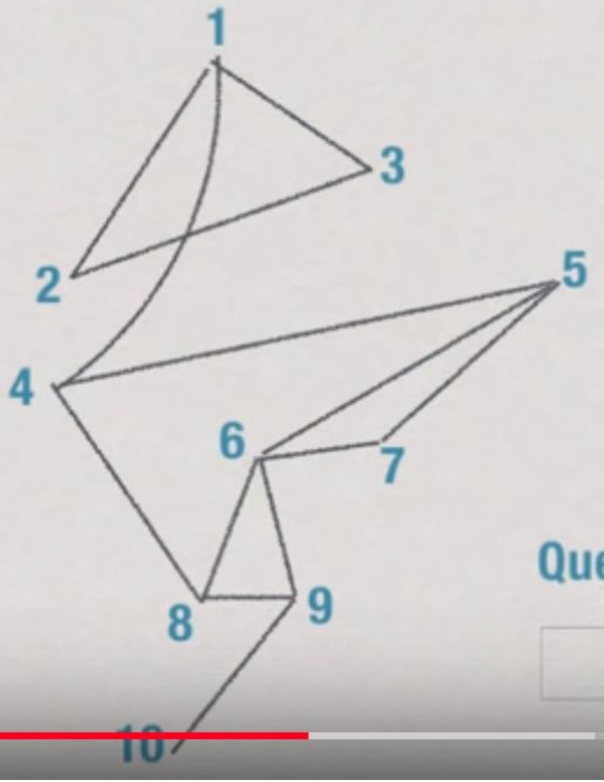
Visited

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

Queue

						6	7		

Breadth first search



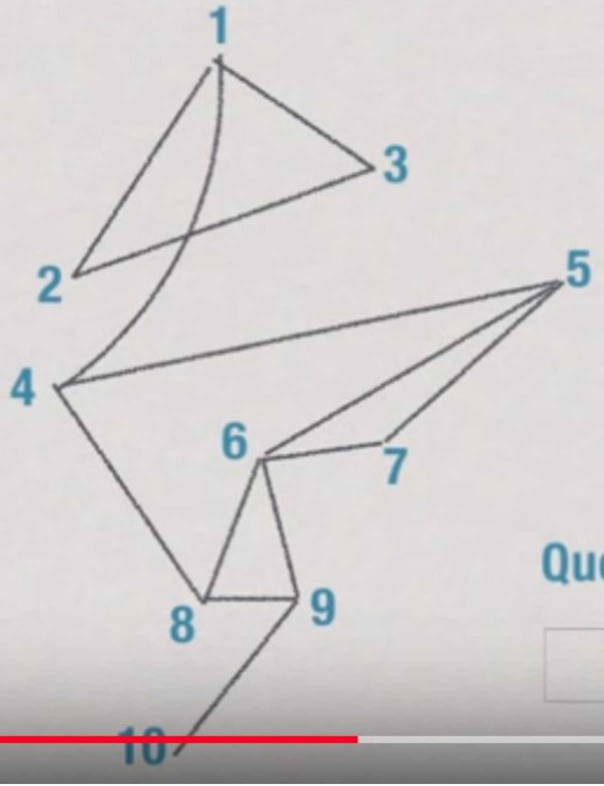
Visited

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

Queue

						6	7	9	

Breadth first search



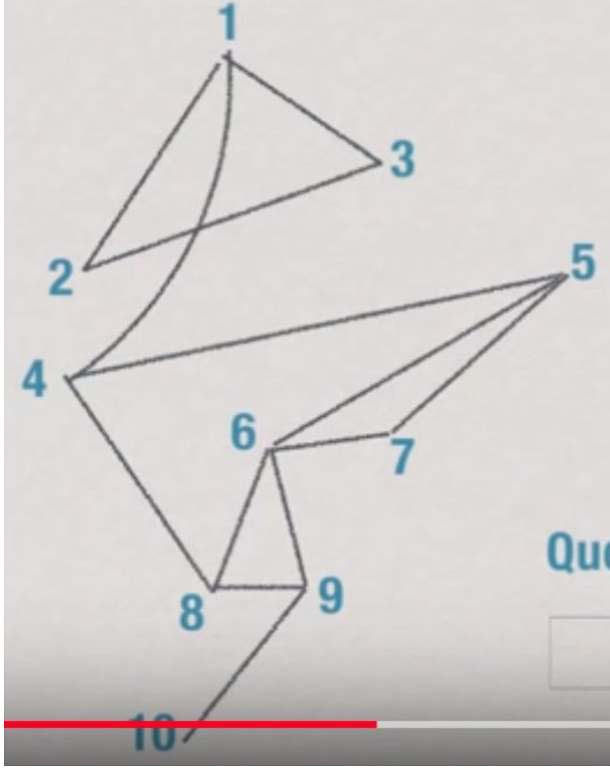
Visited

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

Queue

							7	9	

Breadth first search



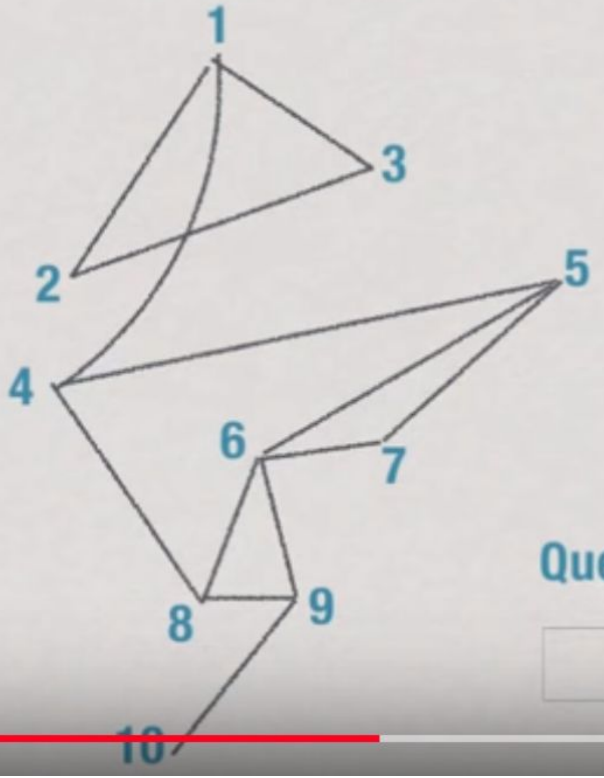
Visited

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

Queue

								9	

Breadth first search

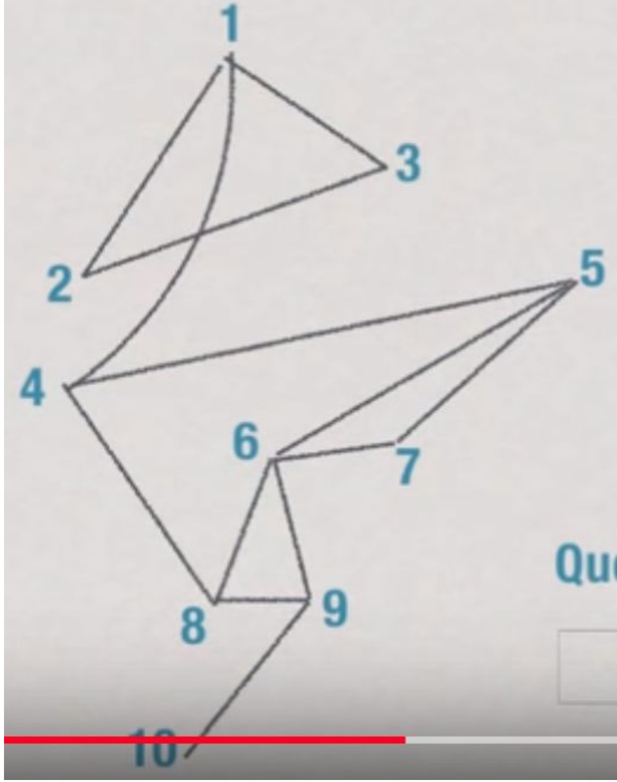


Visited

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

Queue

Breadth first search



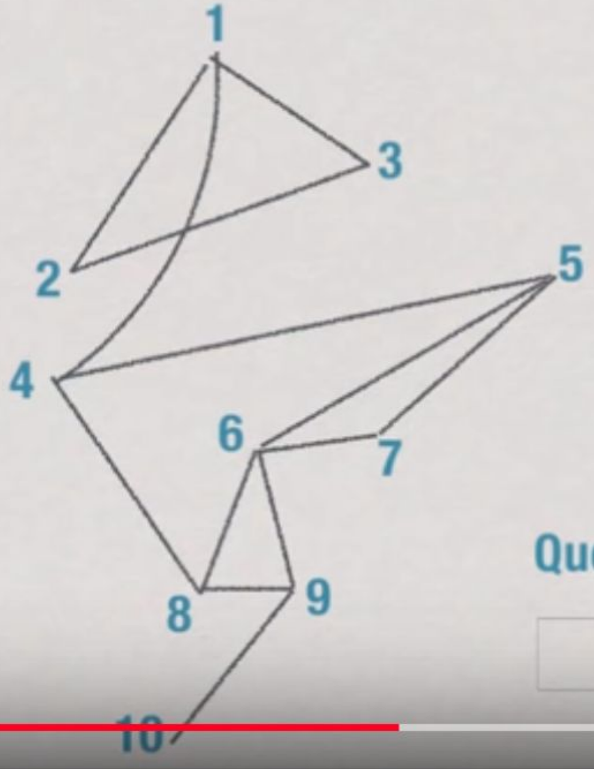
Visited

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

Queue

									10

Breadth first search



Visited

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

Queue

Breadth first search

```
function BFS(i) // BFS starting from vertex i

   //Initialization
   for j = 1..n {visited[j] = 0}; Q = []

   //Start the exploration at i
   visited[i] = 1; append(Q,i)

   //Explore each vertex in Q
   while Q is not empty
      j = extract_head(Q)
      for each (j,k) in E
         if visited[k] == 0
            visited[k] = 1; append(Q,k)
```

Complexity of BFS

- Each vertex enters Q exactly once
- If graph is connected, loop to process Q iterated n times
  - For each j extracted from Q, need to examine all neighbours of j
  - In adjacency matrix, scan row j: n entries
- Hence, overall $O(n^2)$

Complexity of BFS

- Let m be the number of edges in E. What if $m << n^2$?
- Adjacency list: scanning neighbours of j takes time proportional to number of neighbours (**degree** of j)
- Across the loop, each edge (i,j) is scanned twice, once when exploring i and again when exploring j
  - Overall, exploring neighbours takes time O(m)
- Marking n vertices visited still takes O(n)
- Overall, O(n+m)

Complexity of BFS

- For graphs, O(m+n) is considered the best possible
    - Need to see each edge and vertex at least once
- O(m+n) is considered to be **linear** in the size of the graph