Graph Basics

Data Structures and Algorithms

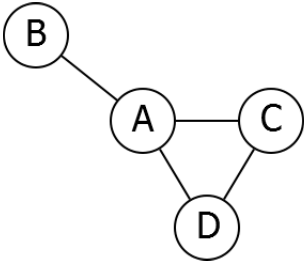
Formal Definition

Definition

An (undirected) Graph i s a collection V of vertices, and a collection E of edges each of which connects a part of vertices.

Drawing Graphs

Vertices: Points. Edges: Lines.

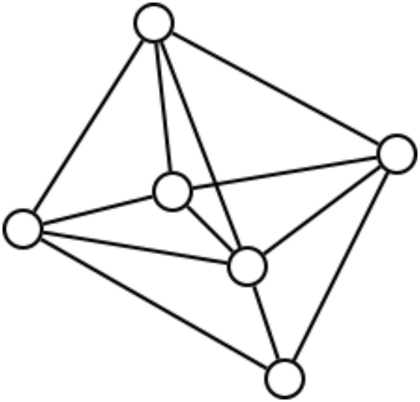


Vertices: A, B, C, D

Edges: (A, B), (A, C), (A, D), (C, D)

Problem

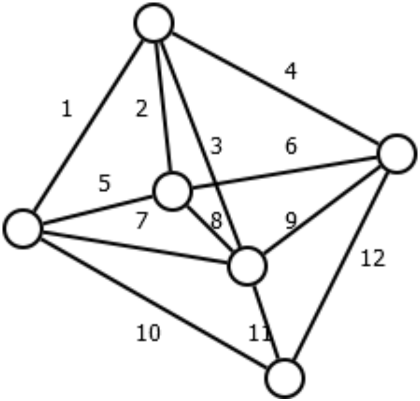
How many edges are i n t h e graph given below?



Answer

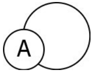
12.

.



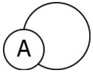
Loops and Multiple Edges

Loops connect a vertex to i tself.

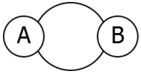


Loops and Multiple Edges

Loops connect a vertex to i tself.

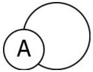


Multiple edges between same vertices.

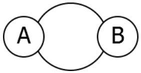


Loops and Multiple Edges

Loops connect a vertex to i tself.



Multiple edges between same vertices.



if a graph has neither, it i s simple.

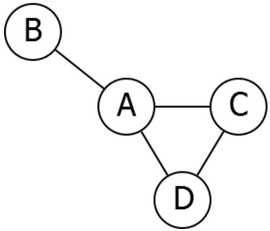
Representing Graphs

To compute things about graphs we frst need to represent them.

There are many ways to do this.

Edge List

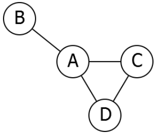
List of all edges:



Edges: (A, B), (A, C), (A, D), (C, D)

Adjacency Matrix

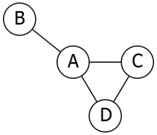
Matrix. Entries 1 i f there i s an edge, 0 i f there i s not.



$$\begin{array}{ccccc} & A & B & C & D \\ A & 0 & 1 & 1 & 1 \\ B & 1 & 0 & 0 & 0 \\ C & 1 & 0 & 0 & 1 \\ D & 1 & 0 & 1 & 0 \end{array}$$

Adjacency List

For each vertex, a list of adjacent vertices.



A adjacent to *B, C, D*

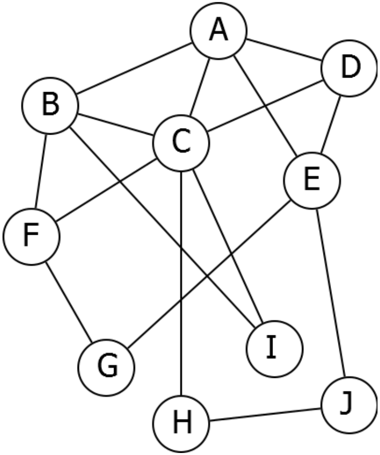
B adjacent to *A*

C adjacent to *A, D*

D adjacent to *A, C*

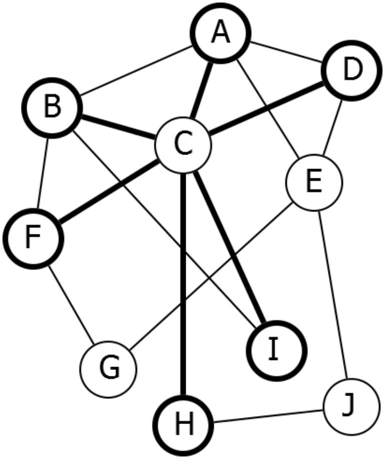
Problem

What are the neighbors of *C*?



Solution

A, B, D, F, H, I .

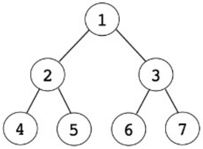


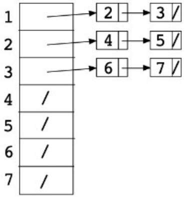
Question

Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that vertices are numbered from 1 to 7 as in a binary heap. (Edges are directed from parent to child)

Question

Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that vertices are numbered from 1 to 7 as in a binary heap. (Edges are directed from parent to child)





Adjacency Matrix:

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	0	0	1	1
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Algorithm Runtimes

Graph algorithm runtimes depend on $|V|$ and $|E|$.

Algorithm Runtimes

Graph algorithm runtimes depend on $|V|$ and $|E|$.

For example, $O(|V| + |E|)$ (linear time), $O(|V||E|)$, $O(|V|^{3/2})$, $O(|V| \log(|V|) + |E|)$.

Density

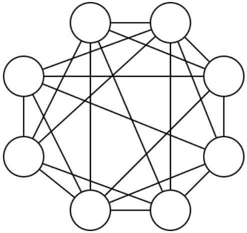
Which is faster, $O(|V|^{3/2})$ or $O(|E|)$?

Density

Wh ch s faster, $O(|V|^{3/2})$ or $O(|E|)$7
Depends on graph! Depends on the density, namely how many edges you have i nterms of the number of vertices.

Dense Graphs

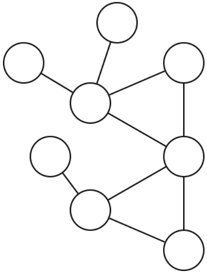
in dense graphs, $|E| \approx |V|^2$.



A large fraction of pairs of vertices are connected by edges.

Sparse Graphs

in sparse graphs, $|E| \approx |V|$.



Each vertex has only a few edges.

Graph Traversal
Depth First Search (DFS)

A standard DFS implementation puts each vertex of the graph into one of two categories:

- Visited
- Not Visited

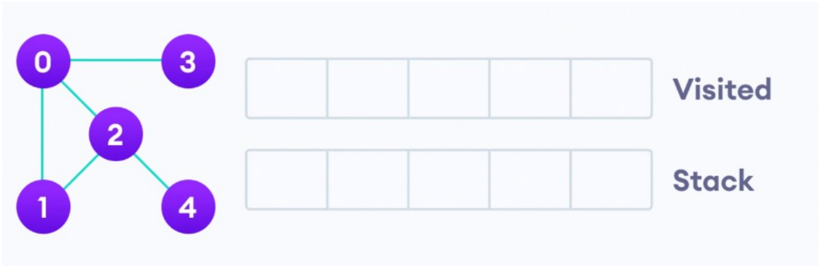
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

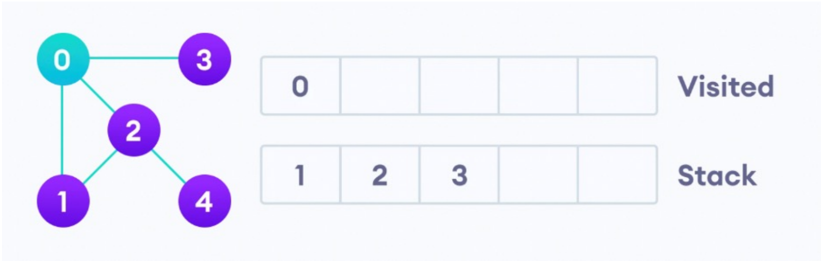
Example

Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



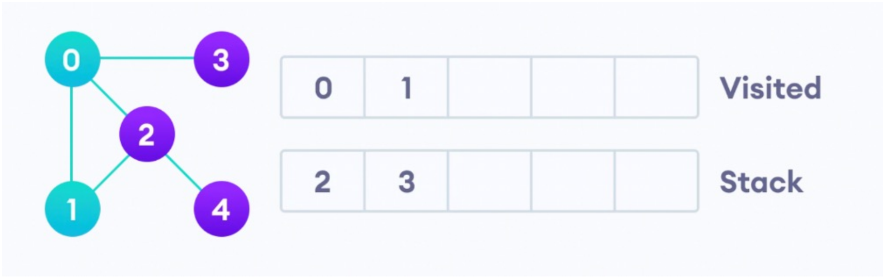
Example

We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.



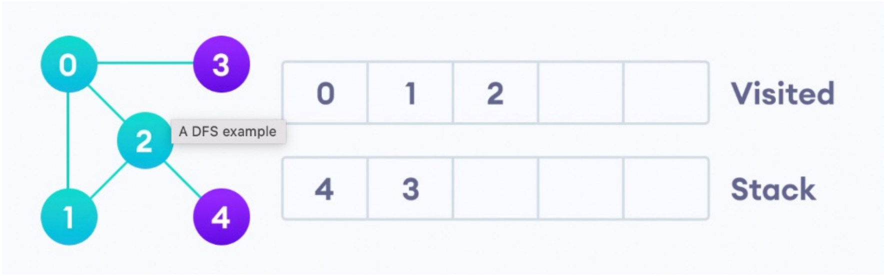
Example

Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.



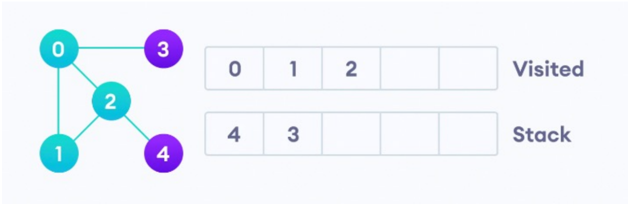
Example

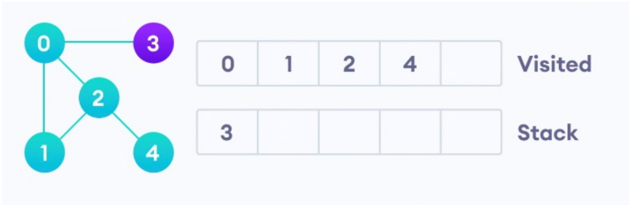
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



Example

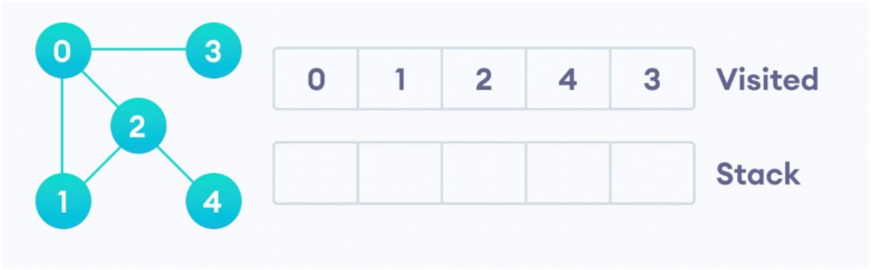
Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.





Example

After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



Complexity of Depth First Search

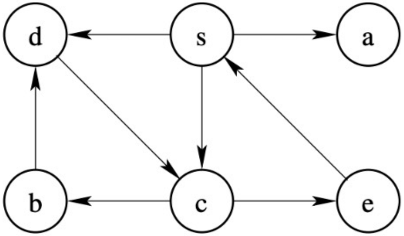
The time complexity of the DFS algorithm is represented in the form of $O(V + E)$, where V is the number of nodes and E is the number of edges.

The space complexity of the algorithm is $O(V)$.

Question

Give the visited node order for Depth First Search(DFS), starting with s, given the following adjacency lists and accompanying figure:

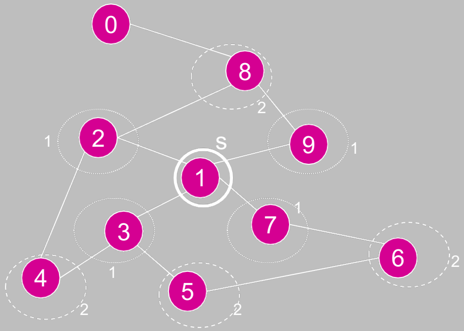
$$
\begin{aligned}
adj(s) &= [a, c, d],\\
adj(a) &= [\,],\\
adj(c) &= [e, b],\\
adj(b) &= [d],\\
adj(d) &= [c],\\
adj(e) &= [s].
\end{aligned}
$$



(b) Depth First Search Solution: s a c e b d (not unique!)

BFS and Shortest Path Problem

- Given any source vertex ***s***, BFS visits the other vertices at increasing distances away from s. In doing so, BFS discovers paths from s to other vertices
- What do we mean by "distance"? The number of edges on a path from s



Example

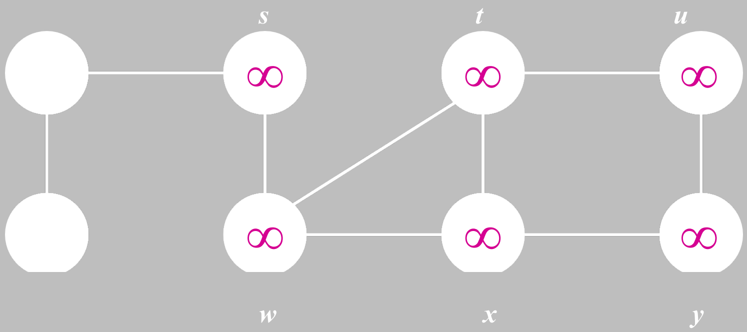
Consider s=vertex 1

Nodes at distance 1?
2, 3, 7, 9

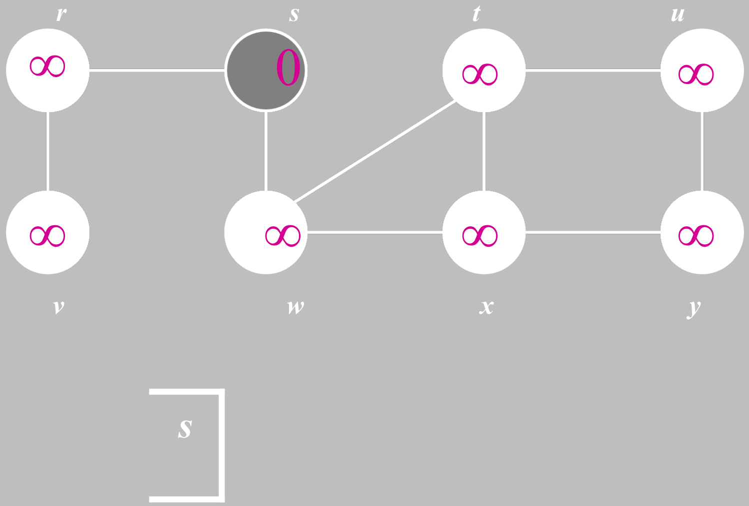
Nodes at distance 2?
8, 6, 5, 4

Nodes at distance 3?
0

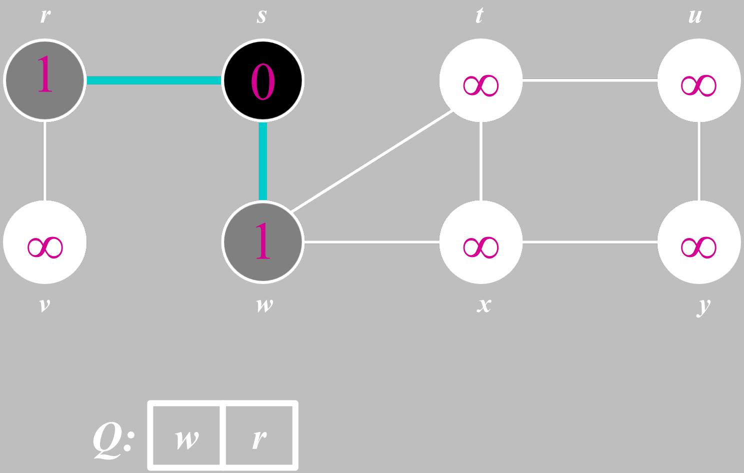
Breadth-First Search: Example



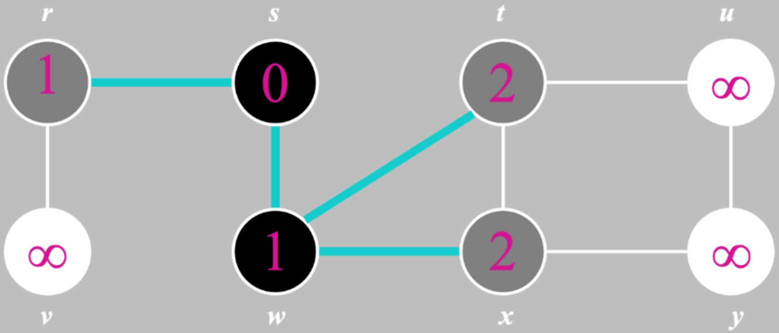
Breadth-First Search: Example



Breadth-First Search: Example

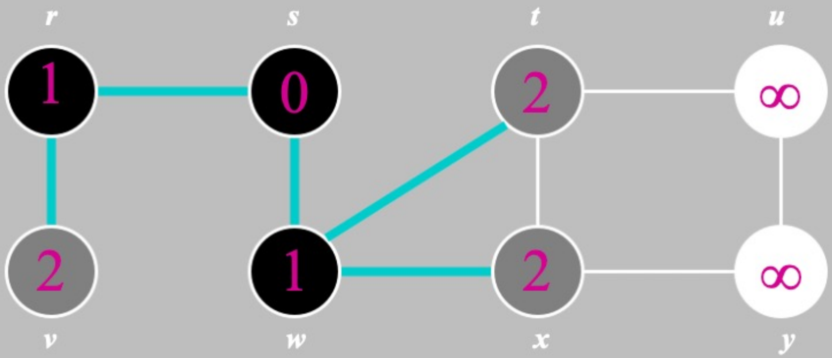


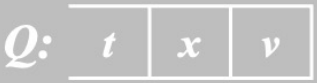
Breadth-First Search: Example



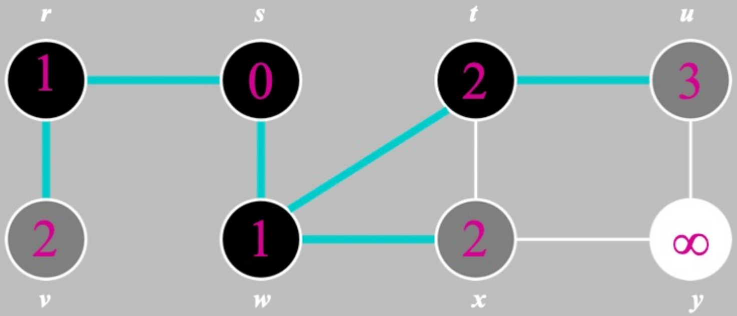
Q:	*r*	*t*	*x*

Breadth-First Search: Example



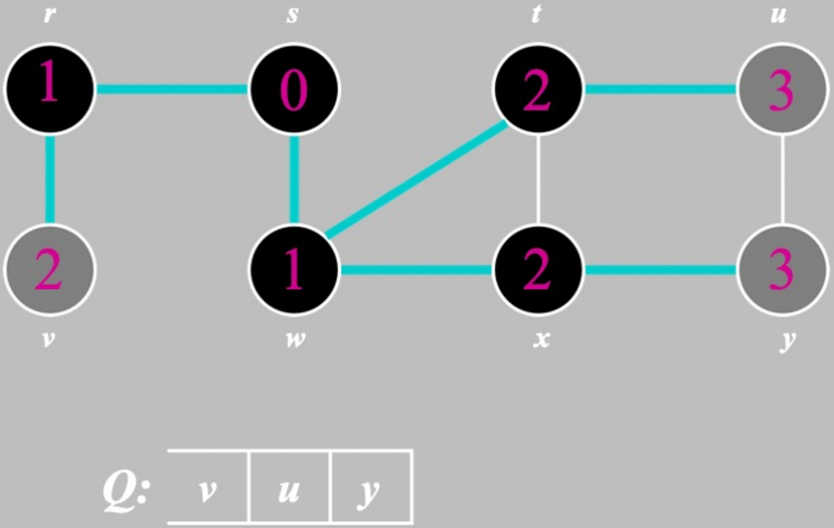


Breadth-First Search: Example

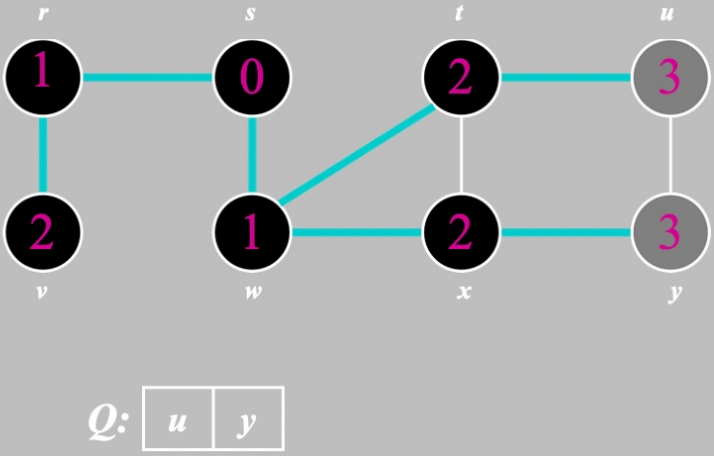


Q: | *x* | *v* | *u* |

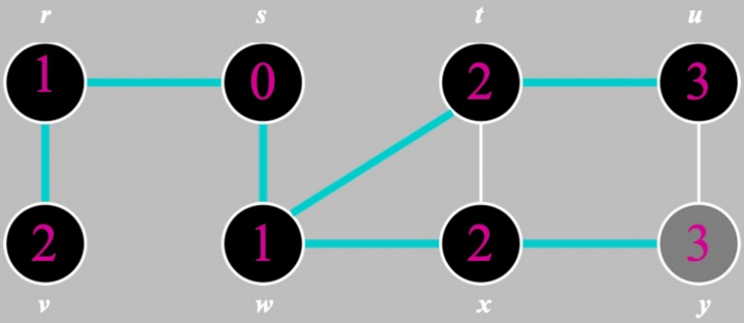
Breadth-First Search: Example

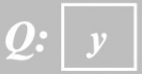


Breadth-First Search: Example

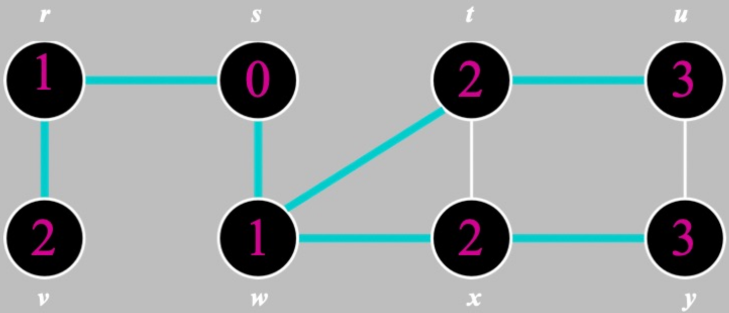


Breadth-First Search: Example





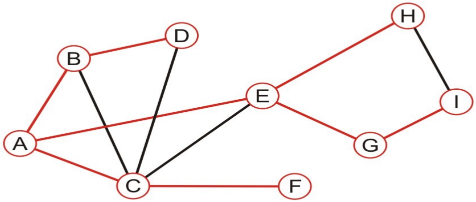
Breadth-First Search: Example



Q: Ø

Question

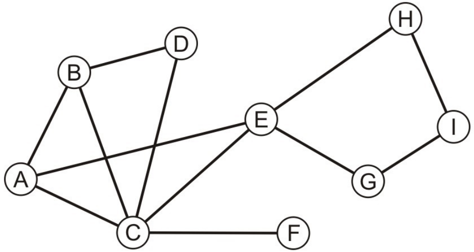
Give the visited node order for Breadth First Search(BFS), starting with A, given the accompanying figure:



(a) Breadth First Search Solution: A, B, C, E, D, F, G, H, I

Solution

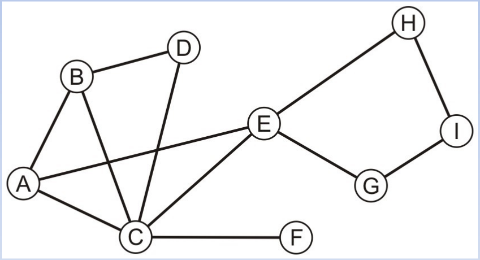
Consider this graph



Solution

Performing a breadth-first traversal

- Push the first vertex onto the queue



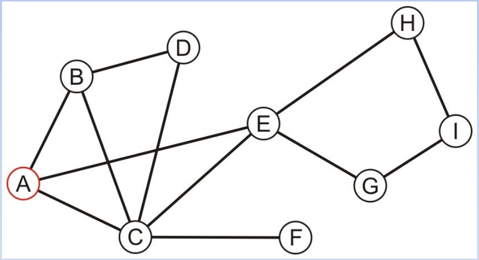
A					

Solution

Performing a breadth-first traversal

– Pop A and push B, C and E

A



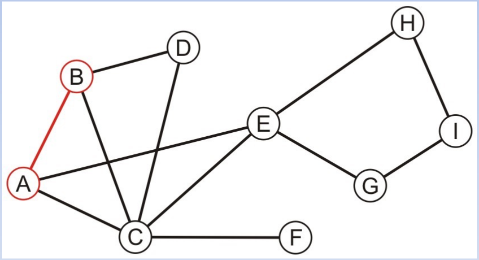
B	C	E			

Solution

Performing a breadth-first traversal:

– Pop B and push D

A, B



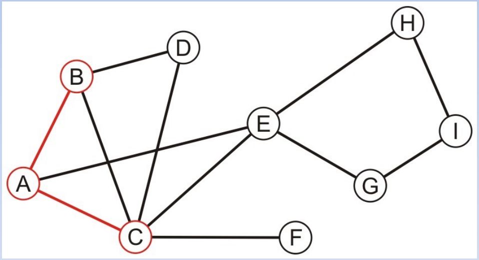
C	E	D			

Solution

Performing a breadth-first traversal:

– Pop C and push F

A, B, C



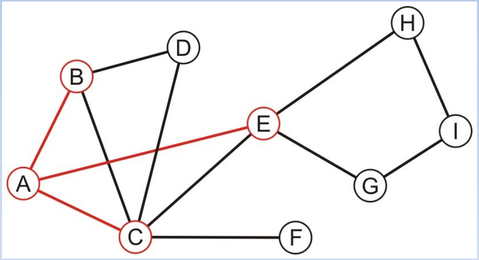
E	D	F			

Solution

Performing a breadth-first traversal:

- Pop E and push G and H

A, B, C, E



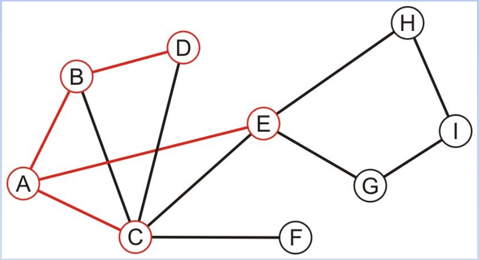
D	F	G	H		

Solution

Performing a breadth-first traversal:

- Pop D

A, B, C, E, D



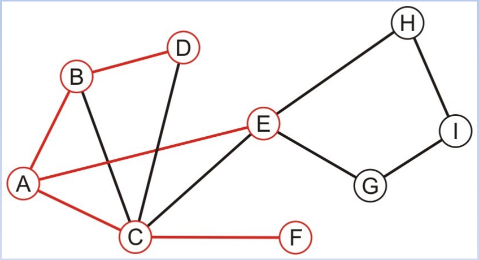
F	G	H			

Solution

Performing a breadth-first traversal:

- Pop F

A, B, C, E, D, F



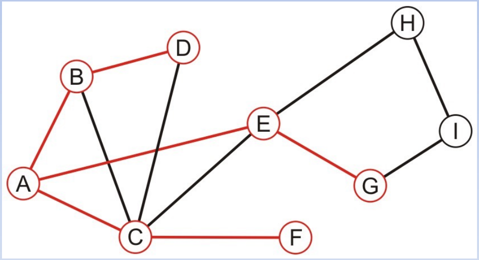
G	H				

Solution

Performing a breadth-first traversal:

- Pop G and push I

A, B, C, E, D, F, G



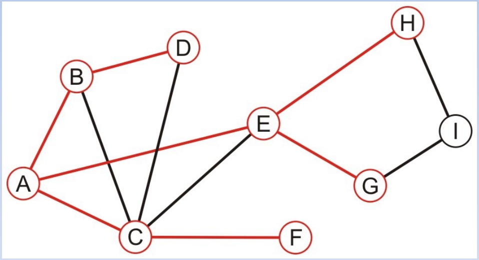
H	I				

Solution

Performing a breadth-first traversal:

– Pop H

A, B, C, E, D, F, G, H



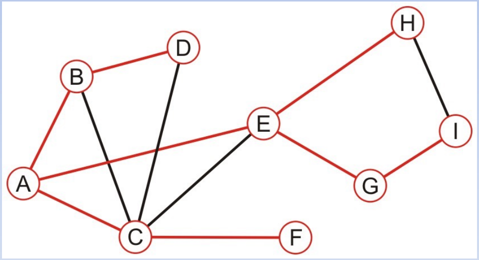
I					

Solution

Performing a breadth-first traversal:

– Pop I

A, B, C, E, D, F, G, H, I

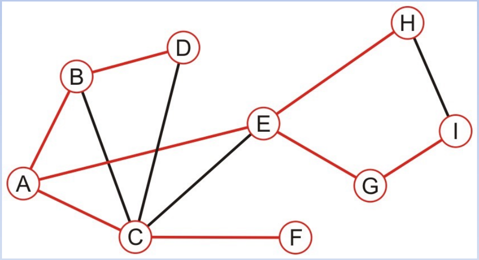


Solution

Performing a breadth-first traversal:

- The queue is empty: we are finished

A, B, C, E, D, F, G, H, I



Question

Give the visited node order for Breadth First Search(BFS), starting with s, given the following adjacency lists and accompanying figure:

$$\begin{aligned}
adj(s) &= [a, c, d],\\
adj(a) &= [\,],\\
adj(c) &= [e, b],\\
adj(b) &= [d],\\
adj(d) &= [c],\\
adj(e) &= [s].
\end{aligned}$$