Basic Algorithms for Searching A Graph

Graphs: Definitions

- A **graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of $\mathcal{V}$ a set of vertices and $\mathcal{E}$ a set of edges.
- A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is **undirected** if $\mathcal{E}$ contains unordered pairs, that is, $\{u, v\} \in \mathcal{E}$ iff $\{v, u\} \in \mathcal{E}$. If $\{u, v\} \in \mathcal{E}$, we say that v is **adjacent** to u, and u is **adjacent** to v.
- A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is **directed** if $\mathcal{E}$ contains ordered pairs. If $(u, v) \in \mathcal{E}$, we sometimes write "$u \rightarrow v$ " and say that v is **adjacent** to u.
- Here we will consider directed graphs.

Graphs: Examples

Example

$G = (V, E)$ is undirected, where $V = \{1, 2, 3, 4\}$, and $E = \{\{1, 2\}; \{1, 3\}; \{2, 4\}\}$:

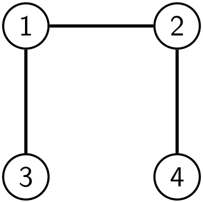


Figure: An Example Undirected Graph

Note that $\{1, 2\} = \{2, 1\}$.

Graphs: Examples

Example

The graph $G = (V, E)$ is directed, where $V = \{1, 2, 3, 4\}$, and $E = \{(1, 2); (2, 1); (3, 3); (4, 2)\}$:

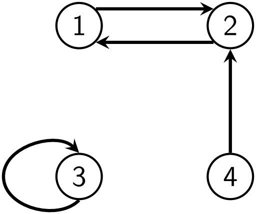


Figure: An Example Directed Graph

Note that $(1, 2) \neq (2, 1)$.

Graphs: Definitions

- A **path** of **length** k from vertex u to vertex v in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a sequence of edges in $\mathcal{E}$ connected to each other:
$$[(u, u_1), (u_1, u_2), ..., (u_{k-1}, v)].$$
- a path is **simple** if it contains no repeated edge or vertex.
- a vertex v is **reachable** from u is there exists a path from u to v.

Graphs Examples

Example

Given the following graph, two example paths in the graph are:

$$\{1,6\};\{6,3\};\{3,4\};\{4,5\}$$

$$\{6,5\};\{5,1\};\{1,5\};\{5,4\}$$

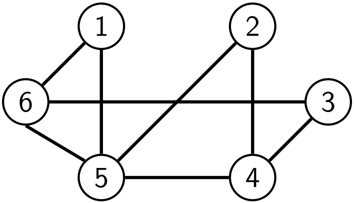
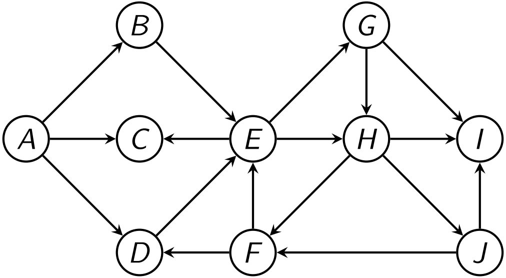


Figure: Another Example Undirected Graph

A Graph Example



- F, I, and J are adjacent to H. H is adjacent to E and G.
- E is reachable from A as there exists at least a path from A to E: $[(A, B), (B, E)]$, or $[(A, D), (D, E)]$.
- A is not reachable for any other vertex in the graph.

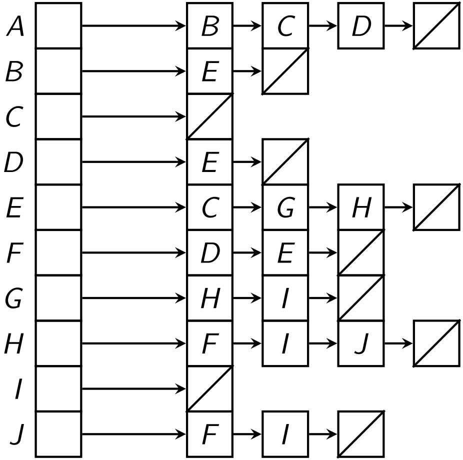
Graph Representation: Adjacency Matrix

A 2-D array with the same number of rows and columns as G contains vertices. The entry at row i column j simply indicates whether or not there is an edge in G from i to j.

	A	B	C	D	E	F	G	H	I	J
A		✓	✓	✓						
B					✓					
C										
D					✓					
E			✓				✓	✓		
F				✓	✓					
G								✓	✓	
H						✓			✓	✓
I										
J						✓			✓	

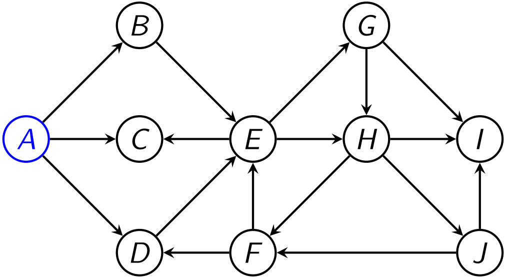
Graph Representation: Adjacency Lists

A list of linked-lists. For each vertex i in the graph, we store a list of the vertices adjacent to i.



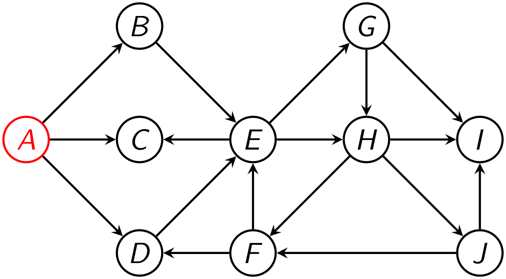
Search A Graph: Problem Statement

Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a specific vertex $v \in \mathcal{V}$, we want to traverse the graph, i.e., "reach" as many vertices as possible staring from v (e.g., vertex A is given below).



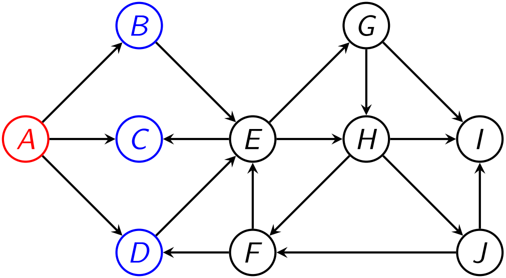
Search A Graph: Ideas

Reached



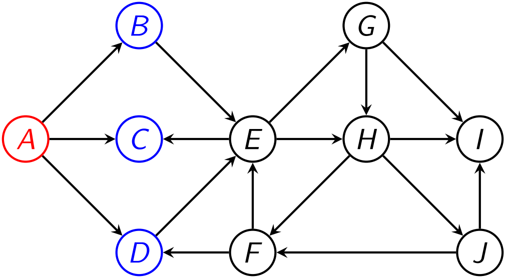
Search A Graph: Ideas

Reached *Discovered*



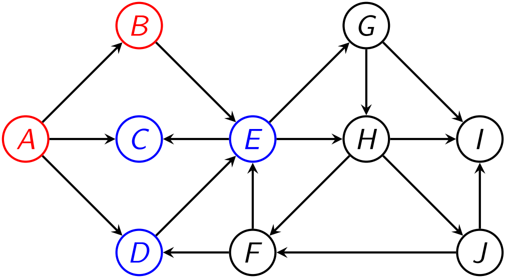
Search A Graph: Ideas

Reached *Discovered* *Others*



Search A Graph: Ideas

Reached *Discovered* *Others*



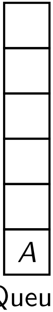
Basic Ideas for Searching A Graph

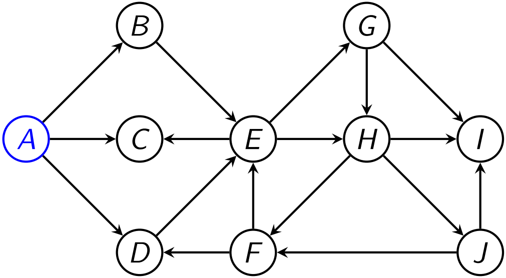
- We need to keep track of the vertices that have been reached already. We need an array of boolean values to indicate, for each vertex, whether or not its been reached.
- Deciding which discovered vertex to reach next is important. We can apply one of the following two basic strategies:
  - Breadth-First search (BFS) tries to reach vertices *as soon as possible*.
  - Depth-First search (DFS) tries to go *as far as possible* before looking at alternatives.

  To keep track of the order in which to reach vertices, we use
  - a first-in-first-out Queue to maintain the set of discovered vertices in BFS;
  - a first-in-last-out Stack to maintain the set of discovered vertices in DFS.

Breadth-First Search: Initialization

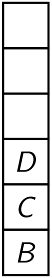
Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

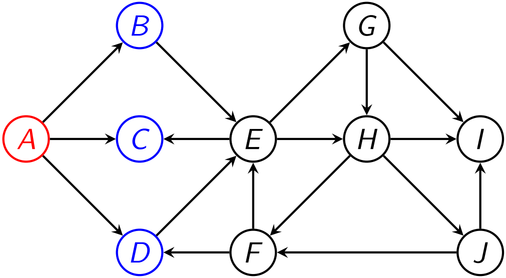




Breadth-First Search: *A* Reached

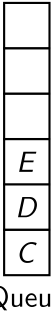
Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗

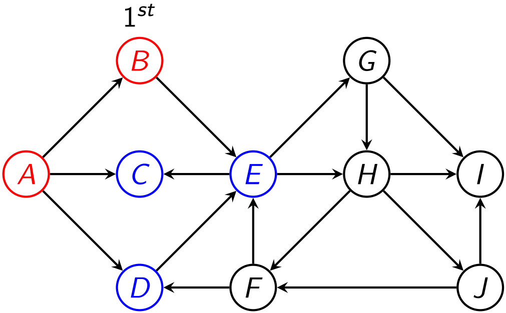




Breadth-First Search: B Reached

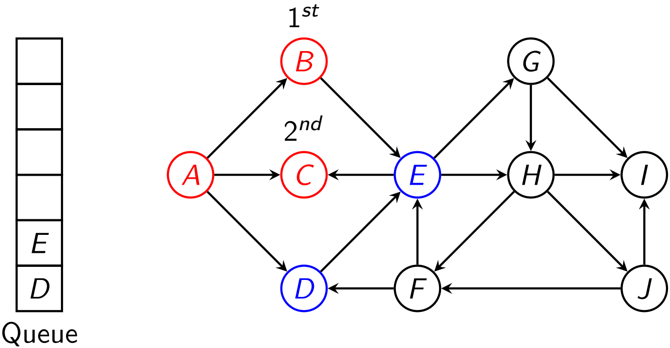
Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗





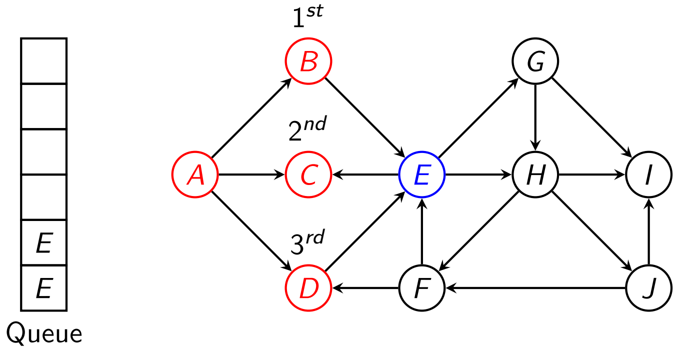
Breadth-First Search: C Reached

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗



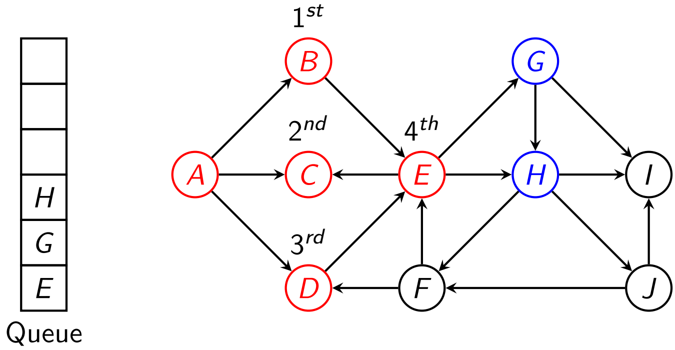
Breadth-First Search: D Reached

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗



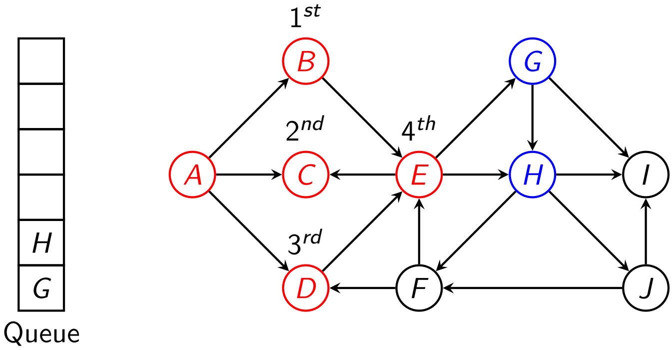
Breadth-First Search: E Reached

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗



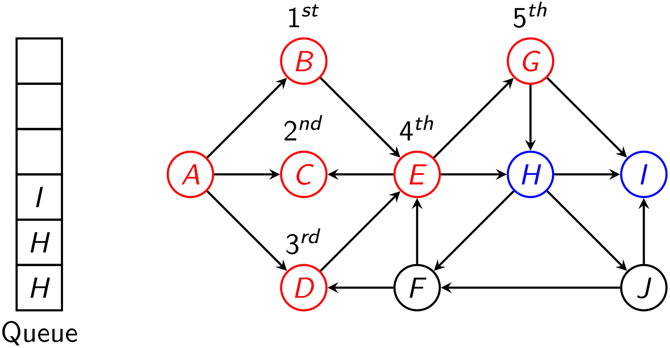
Breadth-First Search: Remove the other E

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗



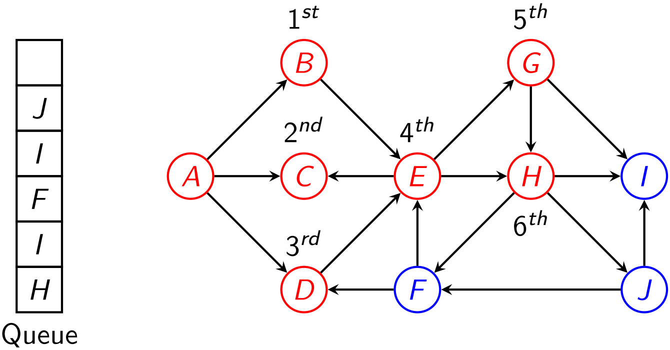
Breadth-First Search: G Reached

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗



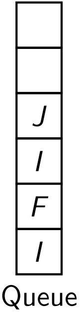
Breadth-First Search: *H* Reached

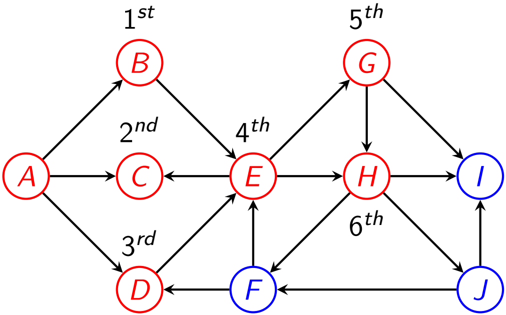
Vertex	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
Reached	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗



Breadth-First Search: Remove the Other H

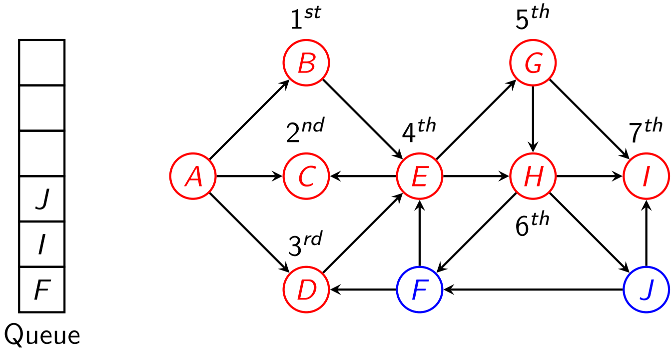
Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✗	✓	✓	✗	✗





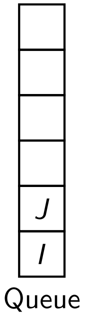
Breadth-First Search: *I* Reached

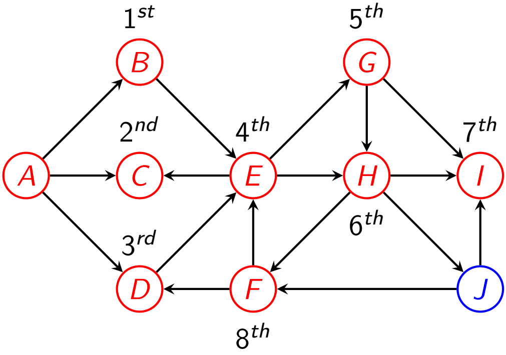
Vertex	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
Reached	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗



Breadth-First Search: *F* Reached

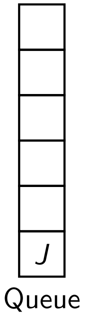
Vertex	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
Reached	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗

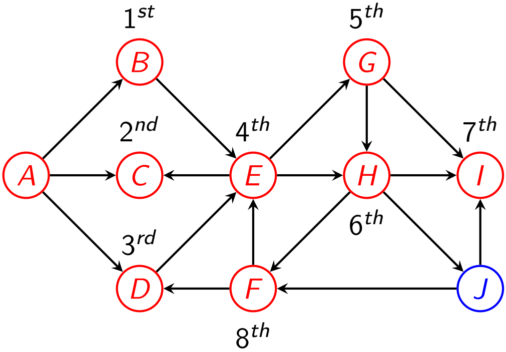




Breadth-First Search: Remove the Other I

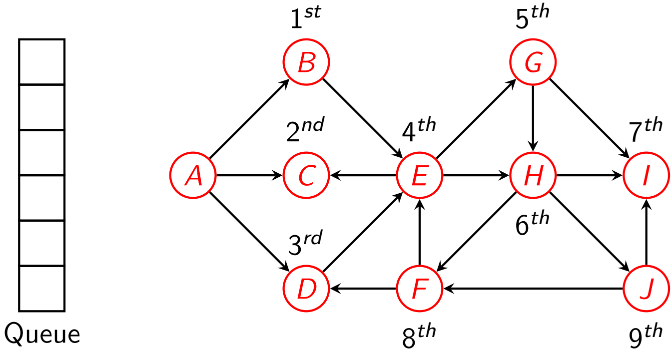
Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗





Breadth-First Search: J Reached

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓



Breadth-First Search: Algorithm

```
BFS(G, v) // search starts at vertex v.
    Queue Q = {} // start with an empty queue
    for each vertex u, set reached[u] = false;
    enqueue(Q v);
    while (Q is not empty)
        u = dequeue(Q);
        if (not reached[u])
            reached[u] = true;
            for each w adjacent to u and not reached
                enqueue(Q,w);
            end for
        end if
    end while
END
```

Breadth-First Search: Remarks

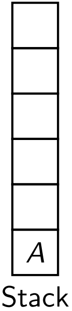
- BGS runs in linear time in the size of G.
- The edges that were used to visit a new vertex can be tracked in BFS:
  - For each vertex w, we use *predecessor*$[w]$ that stores the vertex w was reached from.
  - If w is to be enqueued when reaching u, we set *predecessor*$[w]$ to be equal to u.
- we can impose additional array of boolean values to distinguish discovered and undiscovered vertices. During the search, a vertex v can be in any one of the three states
  - *undiscoverd*, v has not been touched at all.
  - *discovered*, v has been put in the queue.
  - *reached*, v has been taken out of the queue.

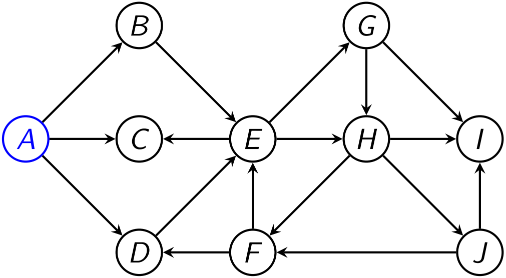
Depth-First Search: Algorithm

```
DFS(G, v) // search starts at vertex v.
    Stack Q = {} // start with an empty stack
    for each vertex u, set reached[u] = false;
    push(S v);
    while (S is not empty)
        u = pop(S);
        if (not reached[u])
            reached[u] = true;
            for each w adjacent to u and not reached
                push(S,w);
            end for
        end if
    end while
END
```

Depth-First Search: Initialization

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗





Depth-First Search: Completed

Vertex	A	B	C	D	E	F	G	H	I	J
Reached	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓