Algorithms

Reading

Required. Asymptotic Analysis:

Sections 2.1-2.4. Chapter 2 Solved Exercises.

Solve exercises 1,3,6 of Chapter 2.

Suggested. Asymptotic Analysis:

Sections 2.1-2.5 of Roughgarden's video lectures.

Notes on Asymptotic Notation.
(uploaded on Brightspace)

All exercises of Chapter 2.

Asymptotic Analysis: Big-Oh Notation

The order notation is not symmetric.

We can say $2n + 5 = O(n)$ but not $O(n) = 2n + 5$.

We can however say:

$$O(n) = O(n^2)$$

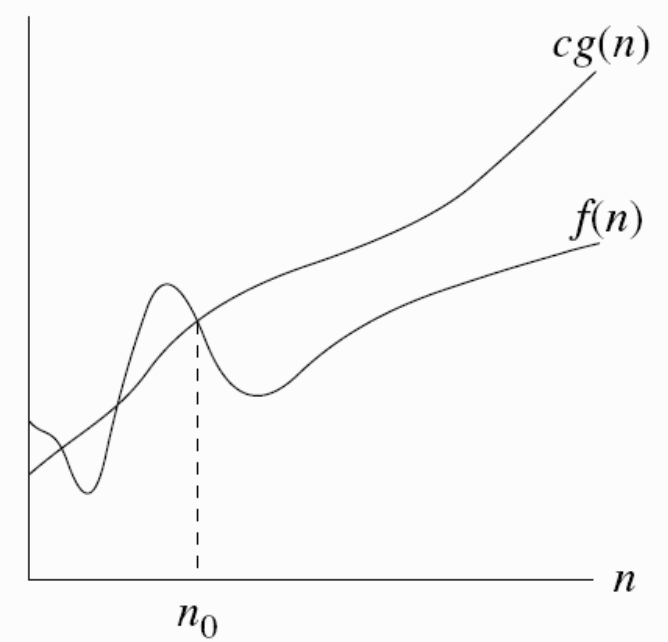
stands for a function that is $O(n)$

Asymptotic Analysis: Big-Oh Notation

$f(n)$ is $O(g(n))$ if $\exists$ constants c and n_0 s.t. $f(n) \leq c \cdot g(n)$ for $n \geq n_0$

In other words it is enough to find a constant c$\geq$ 0, such that

$$lim_{n\to\infty} \frac{f(n)}{g(n)} = \text{c}$$



Abuse of notation:

When $f(n)$ is $O(g(n))$, we will often write $f(n) = O(g(n))$.

This is incorrect but frequently used notation.

Asymptotic Notation

Upper bound order

Saying that $f(n) = O(g(n))$ is somewhat like saying "$f \leq g$".

Lower bound

$f(n) = \Omega(g(n))$ "$f \geq g$" same as $g(n) = O(f(n))$

Asymptotically tight bound

For ..? Worst case

$f(n) = \Theta(g(n))$ "$f = g$" same as $f(n) = O(g(n))$ **and** $g(n) = O(f(n))$

$f(n) = o(g(n))$ "$f < g$" same as $f(n) = O(g(n))$ **but not** $g(n) = O(f(n))$

$f(n) = \omega(g(n))$ "$f > g$" same as $g(n) = o(f(n))$

Examples

$n^2 + 100n = O(n^2)$ *since* $n^2 + 100n \leq 2n^2$ *for* $n \geq 100$

$n^2 + 100n = \Omega(n^2)$ *since* $n^2 = O(n^2 + 100n)$

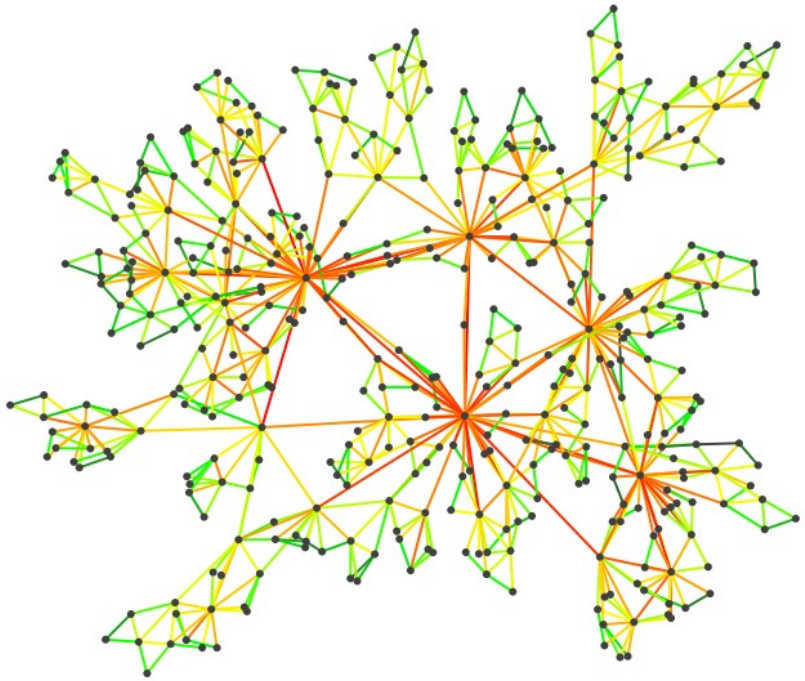
or equivalently, since $n^2 + 100n \geq n^2$ *for* $n \geq 0$

$n^2 + 100n = \Theta(n^2)$ *follows from the above two statements*

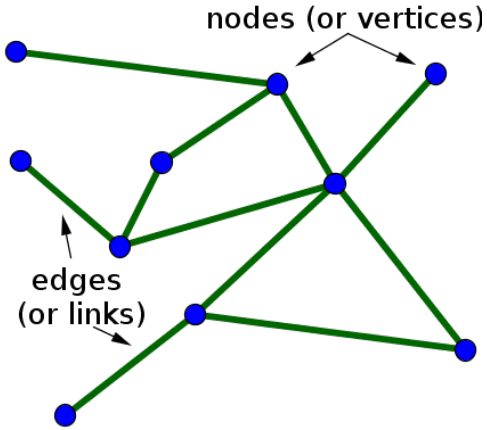
Is $n \log n + 10n = \Theta(n \log n)$? *Yes.*

if $\Theta(f(n) = O(f(n) = g(n)$ *then what can we say about* $\lim_{n\to\infty} \frac{f(n)}{g(n)}$? >0

Graphs and Graph Traversal



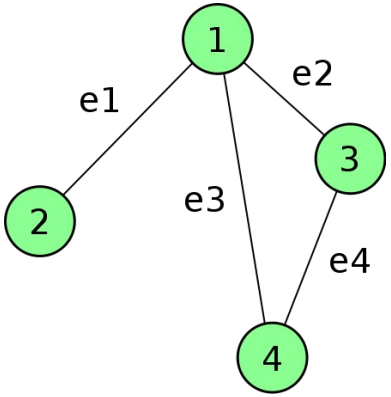
Graphs



A graph can be used to model relationships between entities.

Examples.
web graph, social network, road network, communication network, network of neurons in the brain, economic networks, biomedical networks etc.

Graphs



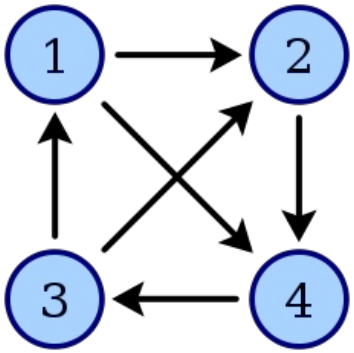
undirected graph

$\mathbf{G} = (\mathbf{V}, \mathbf{E})$

V : set of vertices

E : set of edges

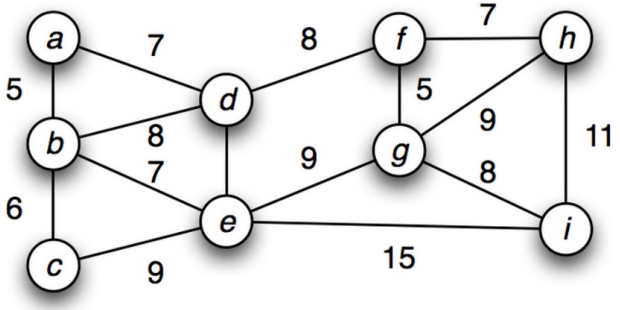
Each edge is a pair of vertices.



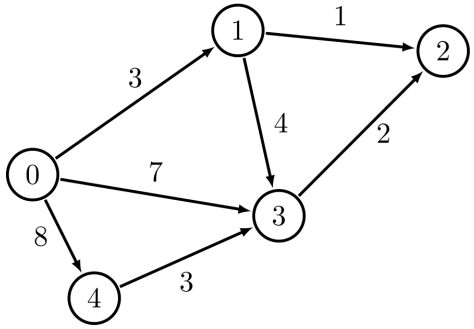
directed graph

A graph in which edges have directions.
That is edges are **ordered pairs.**

Weighted Graphs



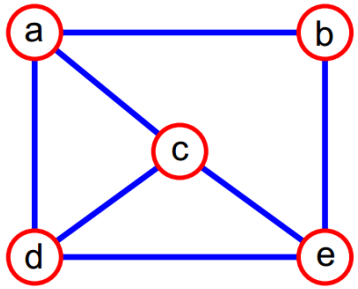
weighted undirected graph



weighted directed graph

Each edge has an associated weight.

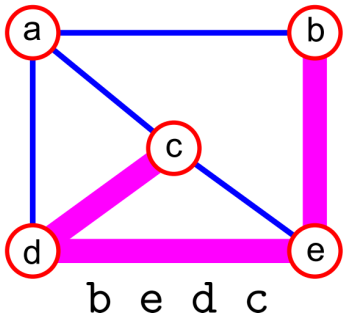
Paths



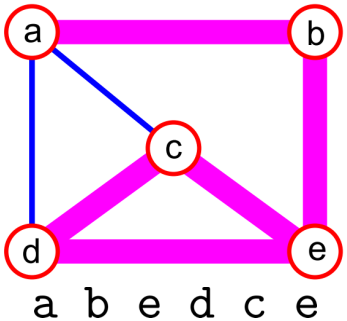
$\mathbf{G} = (\mathbf{V}, \mathbf{E})$

Path. Sequence of vertices $v_1, \cdots, v_k$ s.t. $(v_i, v_{i+1}) \in E$ for all $1 \leq i < k$.

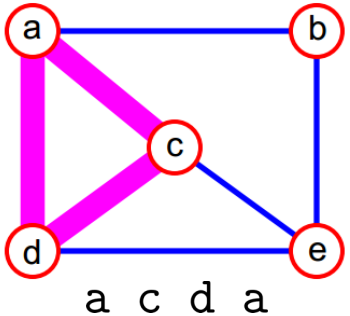
Examples:



simple path



non-simple path

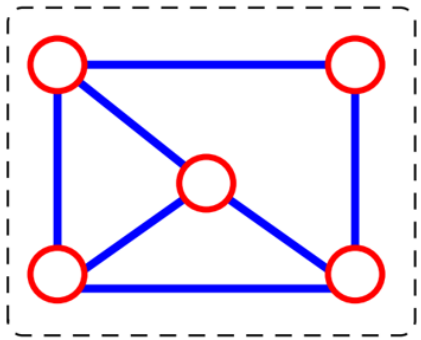


cycle

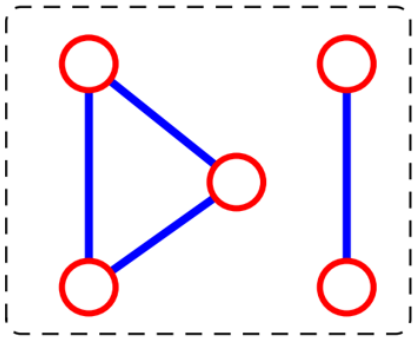
$a\ c\ b\ e$ is **not** a path.

Connectedness

A graph is connected if for every pair of vertices u and v, there is a path joining them.



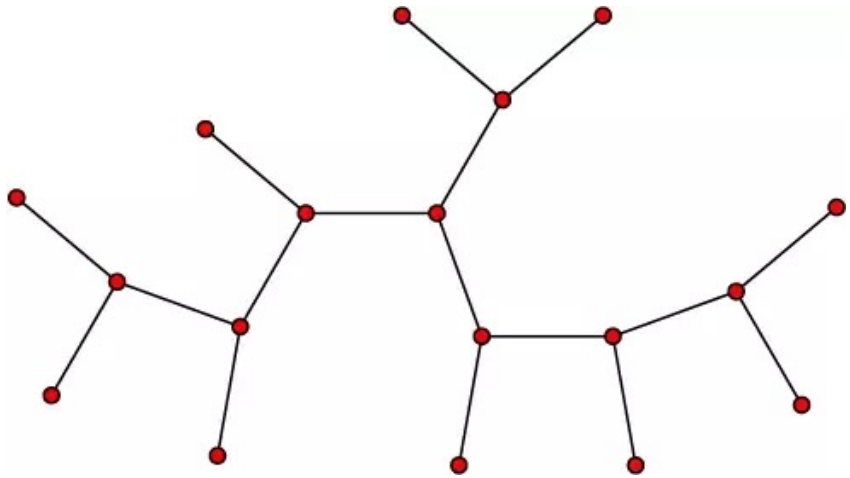
connected graph



not connected
has two connected components

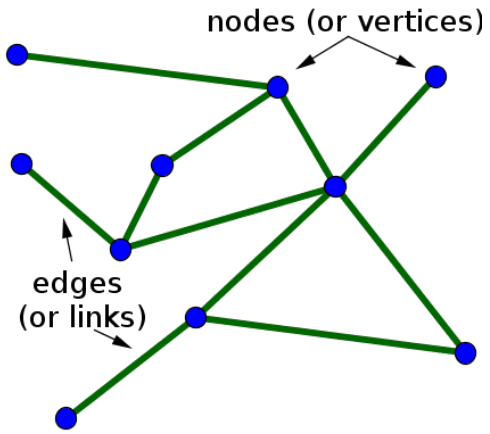
A connected component is a maximal connected subgraph.

Trees



A tree is connected graph without cycles.

Graph Representation



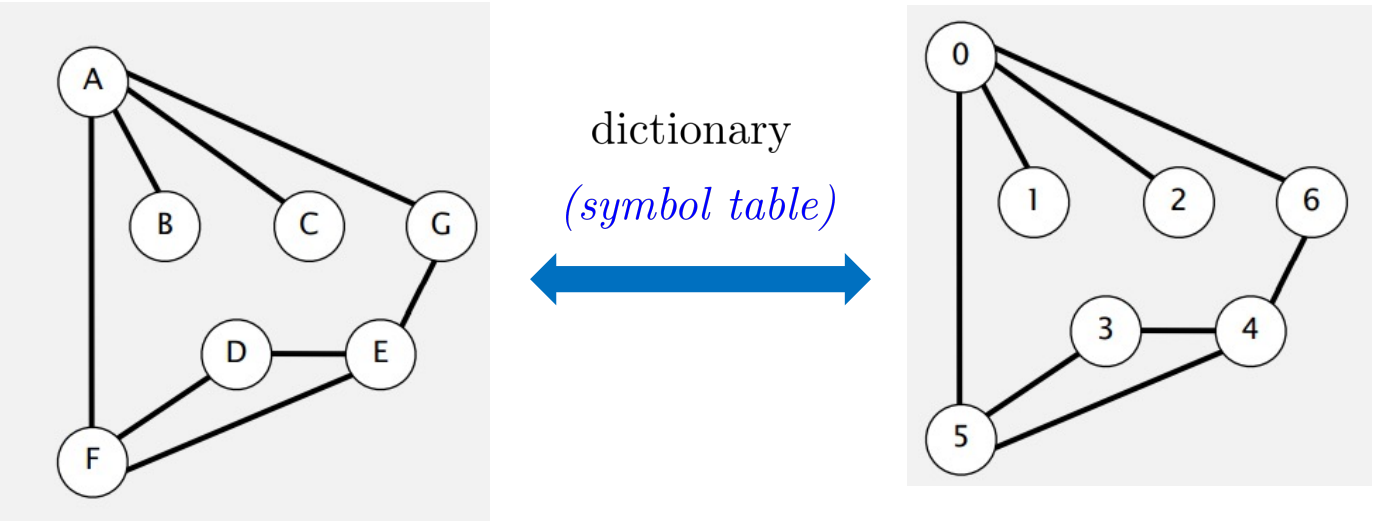
How do we represent a graph in the memory?

Typical operations we need to support:

- Are vertices u and v adjacent?
- Enumerate the neighbors of a vertex v
- Add/Remove a vertex
- Add/Remove an edge

Graph Representation

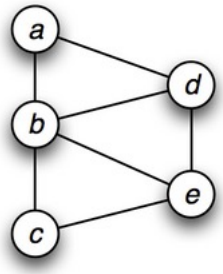
Vertex Representation: Typically an integer between 0 and $n-1$, where n is the number of vertices.



In applications where vertex names must to be used, a symbol table can be used to convert from the names to integers in $\{0, \cdots, n-1\}$.

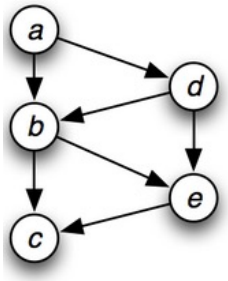
Adjacency Matrix

Undirected Graph



	a	b	c	d	e
a	0	1	0	1	0
b	1	0	1	1	1
c	0	1	0	0	1
d	1	1	0	0	1
e	0	1	1	1	0

Directed Graph



	a	b	c	d	e
a	0	1	0	1	0
b	0	0	1	0	1
c	0	0	0	0	0
d	0	1	0	0	1
e	0	0	1	0	0

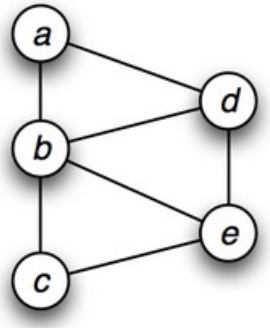
Cell (u, v) has a 1 if the edge (u, v) is present, and 0 otherwise.

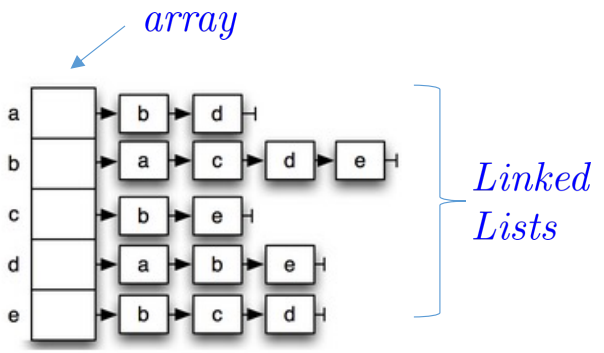
How much memory does this require for n vertices and m edges? $\Theta(n^2)$

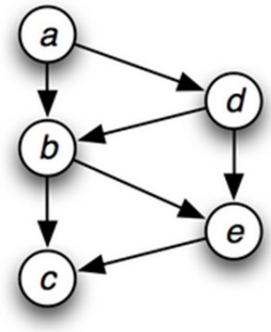
How much time is required to find the neighbors of a vertex? $\Theta(n)$

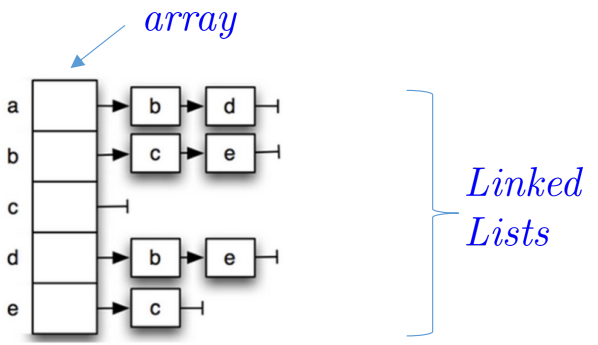
How much time to add a vertex? $\Theta(n^2)$

Adjacency List









How much memory does this require for n vertices and m edges? $\Theta(m+n)$

How much time is required to find the neighbors of a vertex? $\Theta(1)$ *time per neighbor*

How much time to add a vertex? $\Theta(n)$

Graph Traversal

Given a graph, we would like to visit all the vertices.

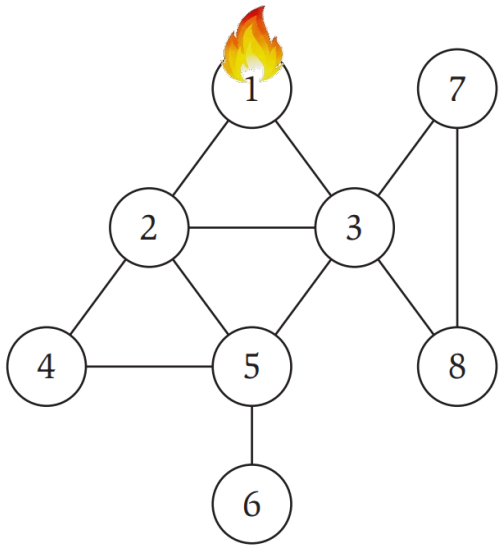
Two basic traversal algorithms:

- Breadth First Search (BFS)
- Depth First Search (DFS)

We will describe the traversal algorithms for connected graphs.

If the graph has multiple connected components, we simply run the algorithm on each component separately.

Breadth First Search (BFS)



Suppose that we start a fire at the vertex 1 and the fire takes one unit of time to traverse any edge. In what order to the vertices catch fire?

(A) 1, 2, 3, 4, 5, 6, 7, 8

(B) 1, 2, 3, 4, 5, 7, 8, 6

(C) 1, 2, 3, 7, 8, 5, 4, 6

(D) 1, 3, 7, 8, 2, 5, 6, 4

Pick all that apply

BFS starts at an arbitrary vertex v and visits the vertices in the order in which they will catch fire if we start a fire at v.

BFS chooses an arbitrary order among vertices that catch fire at the same time.

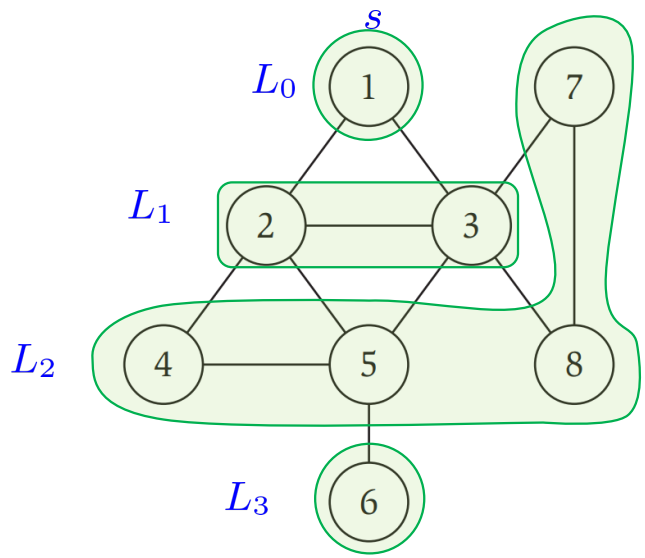
Breadth First Search (BFS)

We start at a vertex s and explore the graph one "layer" at a time.

Layer L_0 consists of just the vertex s.

For $i \geq 1$, layer L_i consists of all the vertices that are adjacent to a node in layer L_{i-1} but do not belong to previous layers.

Example.

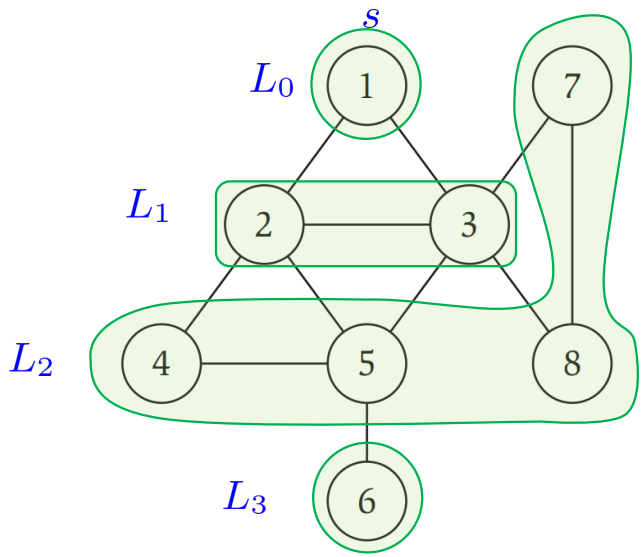


Vertices within a layer can be visited in any order.

Visualization: https://www.cs.usfca.edu/~galles/visualization/BFS.html

Breadth First Search (BFS)

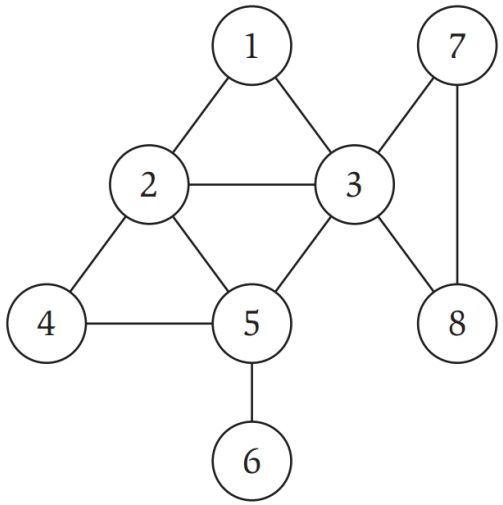
Example.



Observation. Layer L_i consists of the vertices to which the shortest path from s consists of i edges.

Breadth First Search (BFS)

Example.



What are the layers if we start from vertex 7?

$L_0 : \{7\}$ $L_1 : \{3, 8\}$ $L_2 : \{1, 2, 5\}$ $L_3 : \{4, 6\}$

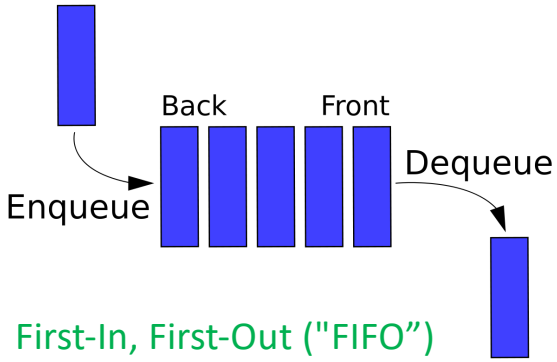
How many layers do we have if we start at vertex 2?

three : $L_0 = \{2\}, L_1 = \{1, 3, 4, 5\}$ *and* $L_2 = \{6, 7, 8\}$

Breadth First Search (BFS)

How do we implement BFS?

We need a very simple data structure called a **Queue**.



$Enqueue(x)$: inserts element x at the rear of the queue

$Dequeue()$: removes and returns the element at the front of the queue

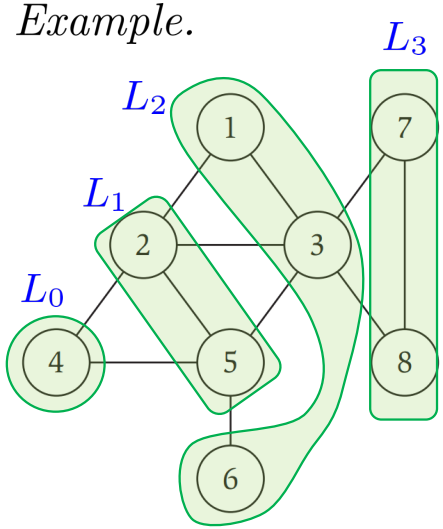
$Empty()$: returns `true` if the queue is empty, `false` otherwise

Easily implemented using linked lists s.t. all operations take $O(1)$ time.

Breadth First Search (BFS)

```
Q = {}  # empty queue

BFS(s):
    Q.enqueue(s)
    mark s as "discovered"

    while not Q.empty():

        u = Q.dequeue()
        visit u and mark it "visited"

        for each edge (u,v):
            if v is not "discovered":
                Q.enqueue(v)
                mark v as "discovered"
```

Example.



Execution of $BFS(4)$:

Front *Rear*

~~4~~ ~~2~~ ~~5~~ ~~1~~ ~~3~~ ~~6~~ ~~7~~ ~~8~~

vertices are visited in the order they are queued

Breadth First Search (BFS)

Why does this work?

Initially, we enqueue the vertex s and mark it "discovered".

The queue at this point contains exactly L_0.

So, we visit the vertices (actually just one vertex) of L_0 first and during this, we discover and enqueue all the vertices of L_1.

Thus, we visit the vertices of L_1 next and while do this, we discover and enqueue the vertices of L_2.

$\cdots$ and so on $\cdots$

While we visit the vertices of L_i we discover and enqueue the vertices of L_{i+1}.

Breadth First Search (BFS)

```
Q = {}  # empty queue

BFS(s):
    Q.enqueue(s)
    mark s as "discovered"

    while not Q.empty():

        u = Q.dequeue()
        visit u and mark it "visited"

        for each edge (u,v):
            if v is not "discovered":
                Q.enqueue(v)
                mark v as "discovered"
```

What is the running time if the graph has n vertices and m edges?

Homework