**GRAPH THEORY**

**Types**

1. Directed graph: One way street.
2. Undirected graph: Two-way Street.

**Neighbour nodes:** They are nodes which accessible by a node through an edge by obeying the direction of an edge.

**Code**

* Adjacency list:

adjacency list

{

    a: [b, c],

    b: [d],

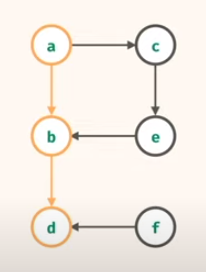
    c: [e],

    d: [],

    e: [b],

    f: [d]

}



**Traversal**

* Depth first search: a, b, d, c, e,
* Breadth first search: a, b, c, d, e,

**Depth first search**

* It uses a stack.

Algorithm Pseudo code

1. Push the starting node onto the stack.
2. Pop node from the stack and immediately look at its neighbor’s and push onto the stack.
3. If the stack has elements, pop the element and push its neighbor’s onto the stack.
4. Repeat step 3 until the stack is empty.

**Important note:** We use array of JavaScript as stack and uses push and pop method.

**Breadth first search**

* It uses a queue.

Algorithm Pseudo code

1. Initialize the queue with starting node.
2. Remove the node from the queue and insert its neighbor’s elements.
3. If the queue has elements, remove the front of queue and insert its neighbor elements into the queue.
4. Repeat step 3 until the queue becomes empty.

**Important note:** array.shift is used as dequeue and array.push is used as enqueue.

**Problems**

1. Has Paths

{

f: [g, I ],

g: [h],

h: [],

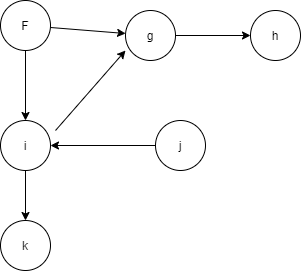
i: [g, k],

j: [ I ],

k: []

}

Visualization –



**Acyclic**

No cycles.

**Cyclic**

You end up to that node where you started, infinite loop.

**Time Complexity**

N = nodes

E = edges

Time: O(e)

Space: O(n)

1. Undirected problem: Is there a path given an undirected graph with cyclic behavior present in the graph or not present.

Edges: [

[i, j],

[k, i],

[m, k],

[k, l],

[o, n]

]

* Every pair in this edge list represents a connection between two nodes.

Converting it to adjacency list.

Adjacency list: {

i: [j, k],

j: [ i ],

k: [ i, m, l],

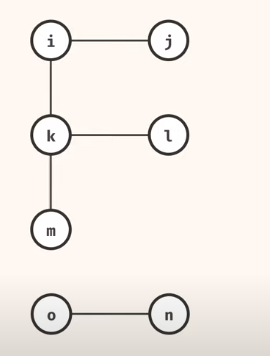
m: [k],

l: [k],

o: [n],

n: [o]

}



**Important note:** There may exist a cycle if a new edge exist which connects node j and node k. We have to watch out for a cycle because if we don’t do any special handling, then we may get trapped in an infinite traversal.

**Solution Algorithm:**

1. Convert edge pair to adjacency list.
2. Starting with the source node, keep a track of visited nodes list.
3. If the neighbor nodes are already visited, return false else add to visited list.
4. If all nodes are visited, end the program.

**Time Complexity**

N = nodes

E = edges

Time: O(e)

Space: O(n)