Graph Representations

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Graphs with weights
- Directed graphs
- Edge lists
- Adjacency matrices
- Adjacency lists

Exercise

- Friend Circles

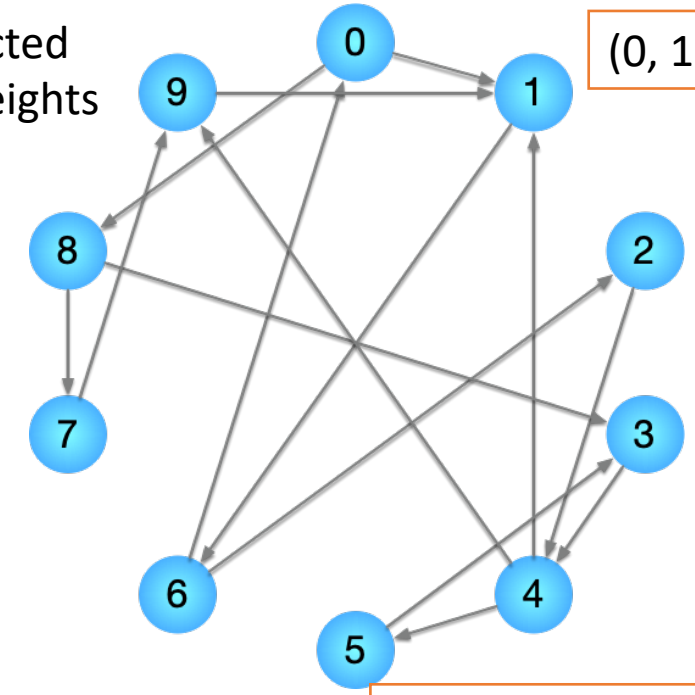
Comparisons

For each representation, we are going to ask the following questions:

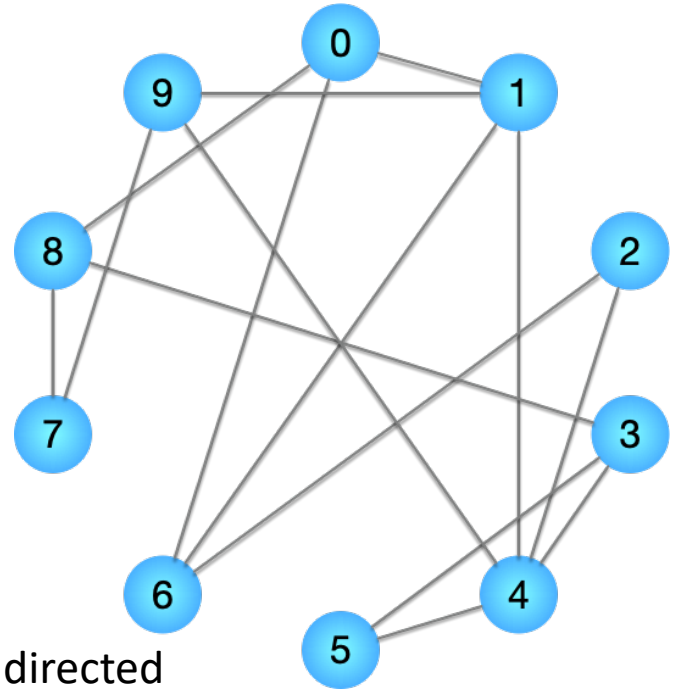
1. How do we count the number of vertices, and how long does it take?
2. How do we count the number of edges, and how long does it take?
3. How do we add vertices, and how long does it take?
4. How do we add edges, and how long does it take?
5. How do you check for the existence of an edge, and how long does it take?
6. How do you find all neighbors of a vertex, and how long does it take?
7. How much memory is needed to store the graph?

Directed
No Weights

(0, 1)

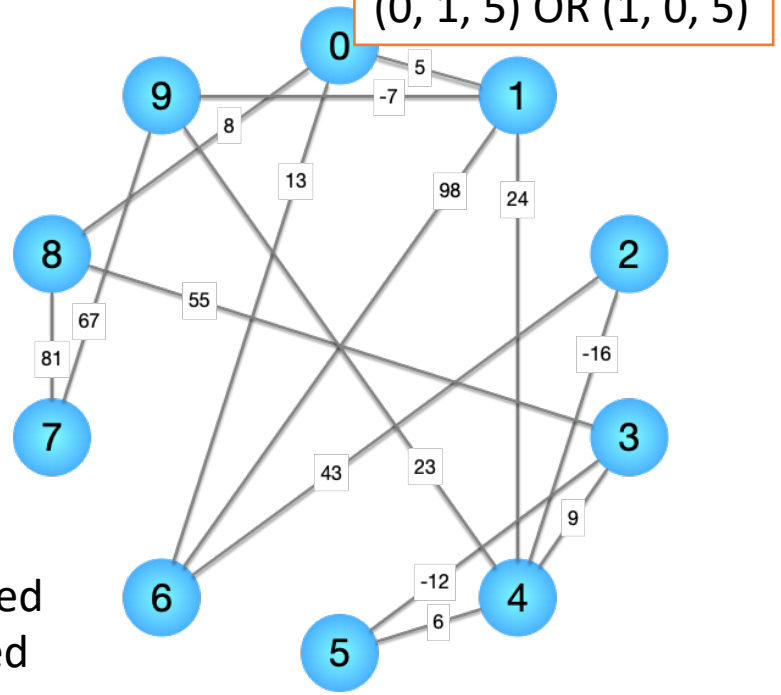


(0, 1) OR (1, 0)



Undirected
No Weights

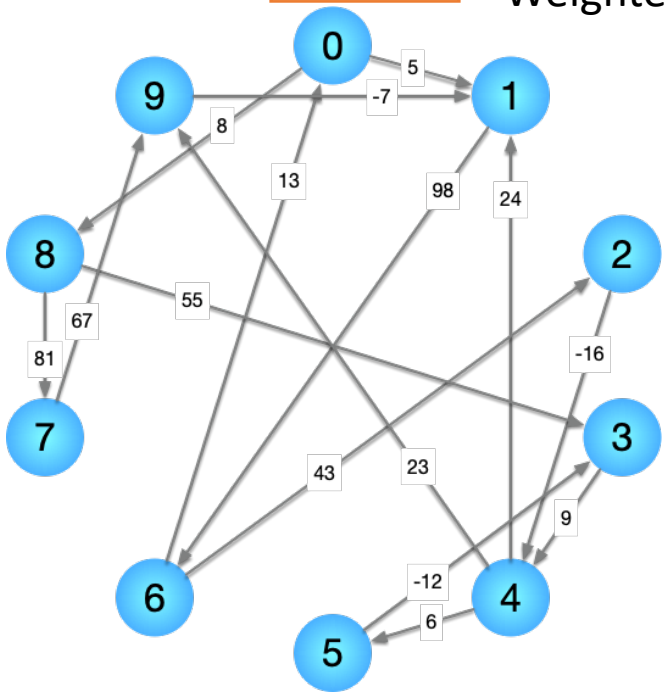
(0, 1, 5) OR (1, 0, 5)



Undirected
Weighted

(0, 1, 5)

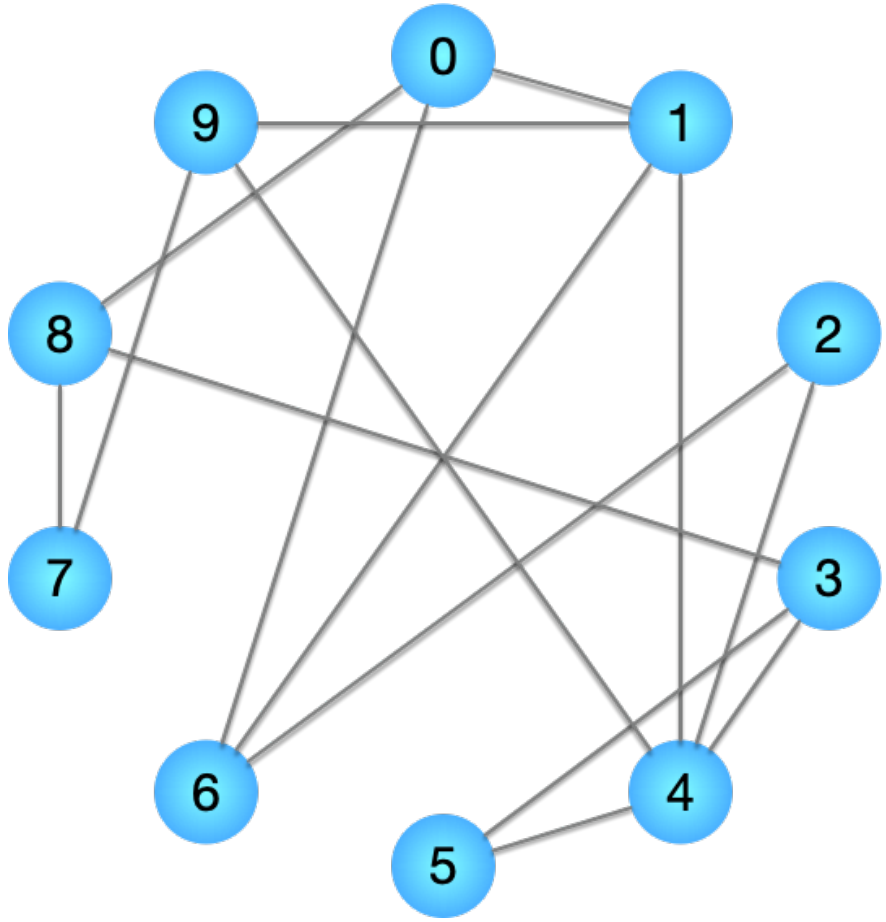
Directed
Weighted



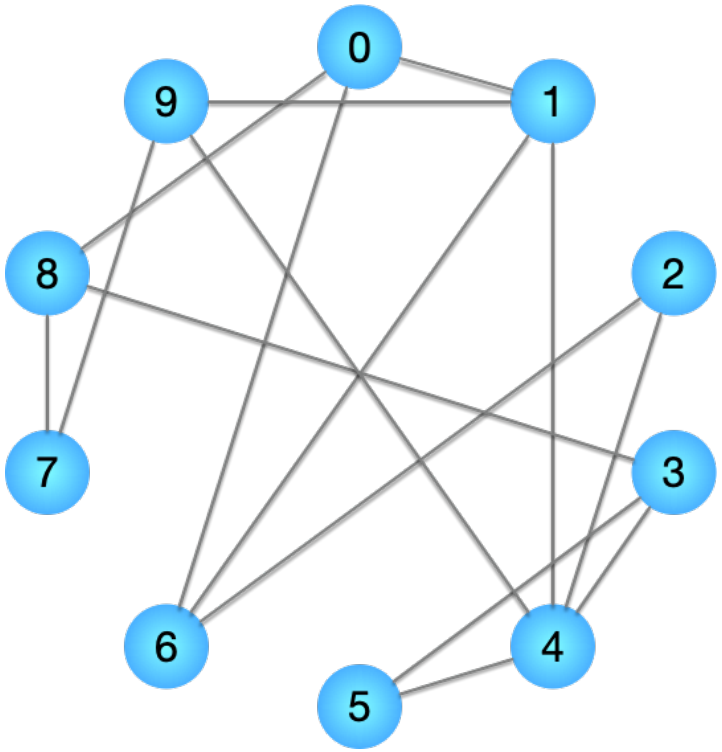
Weights? Directedness?

Edge List Representation

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```

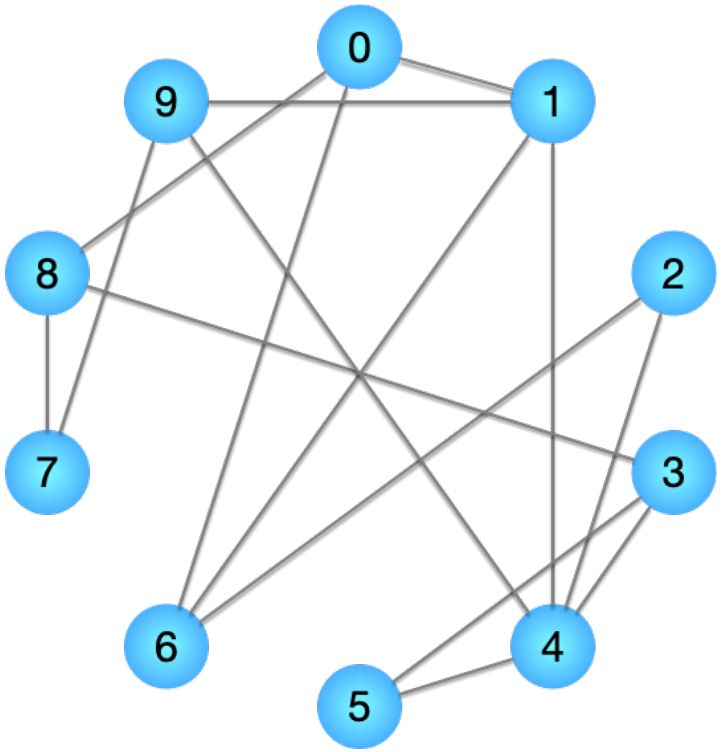


```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```



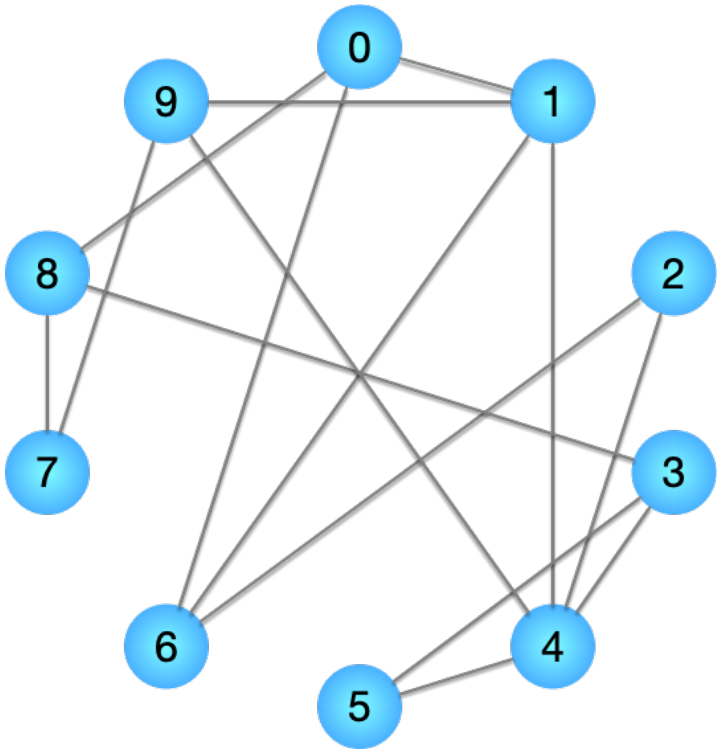

```python
# 1. How do we count the number of vertices, and how
long does it take?
def count_vertices_el(el):
    '''Retun the number of vertices in an edge list.

    Directedness does not matter.

    This procedure loops over all m vertices. So:
    T(n, m) = O(m)
    '''
    V = set()

    for e in el:

            v1, v2 = e[0], e[1]

        # We have to add both v1 and v2 since v2
        # might never show up as the first vertex in
        # an edge.
        V.add(v1)
        V.add(v2)

    return len(V)
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```



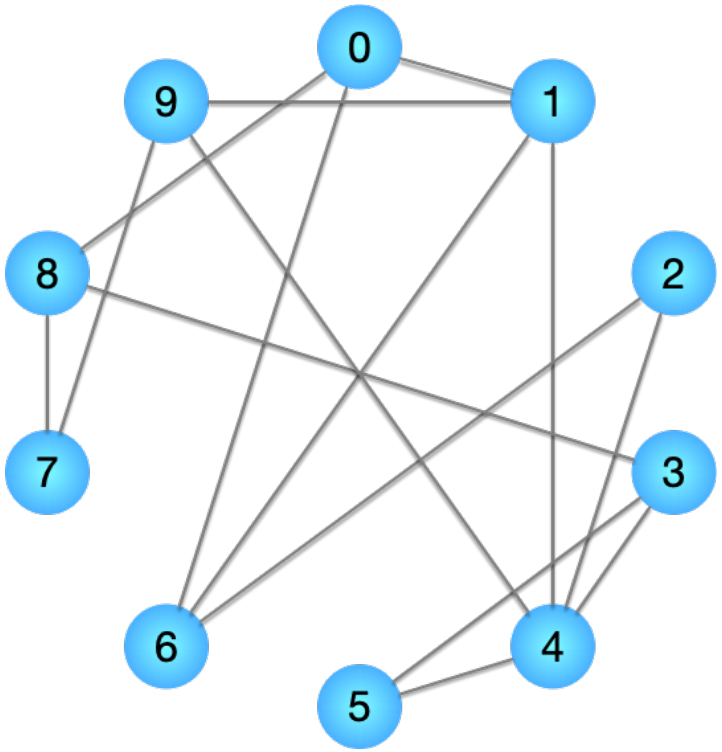

```python
# 2. How do we count the number of edges, and how
long does it take?
def count_edges_el(el):
    '''Return the number of edges in an edge list.

    This function assumes that the edge list does
    not contain any self-connections (an edge
    connecting a vertex to itself).

    T(n, m) = O(1)
    '''
    return len(el)
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```



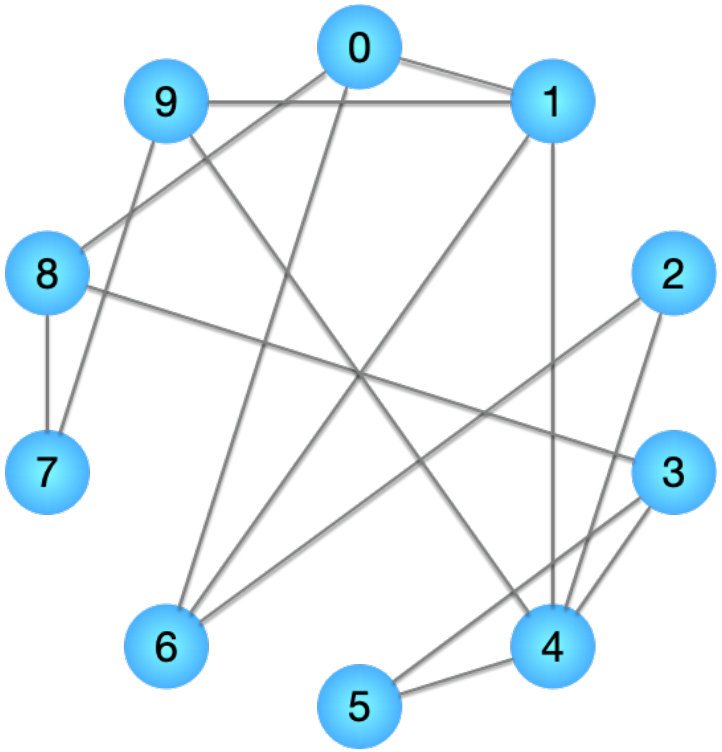

```
# 3. How do we add vertices, and how long does it take?
def add_vertex_el(el, v, is_weighted=False):
    '''Add a new vertex to an edge list.

    This function assumes that v is not already in the edge list.

    T(n, m) = O(1)
    '''
    new_edge = (v, v, 0) if is_weighted else (v, v)
    return add_edge_el(new_edge)
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```



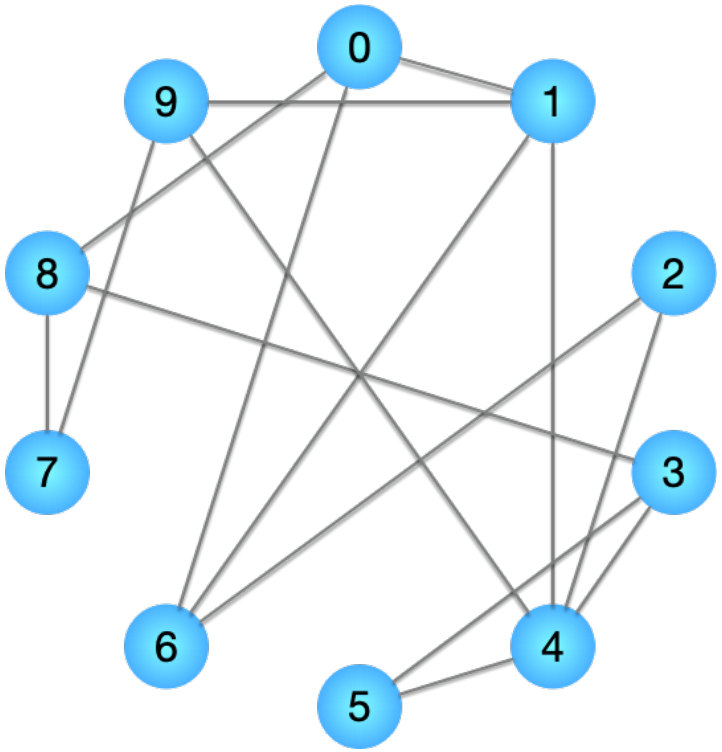

```python
# 4. How do we add edges,and how long does it take?
def add_edge_el(el, e):
    '''Add a new edge to an edge list.

    This function assumes that e is not already in
    the edge list.

    T(n, m) = O(1)
    '''
    el.append(e)
    return el
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```



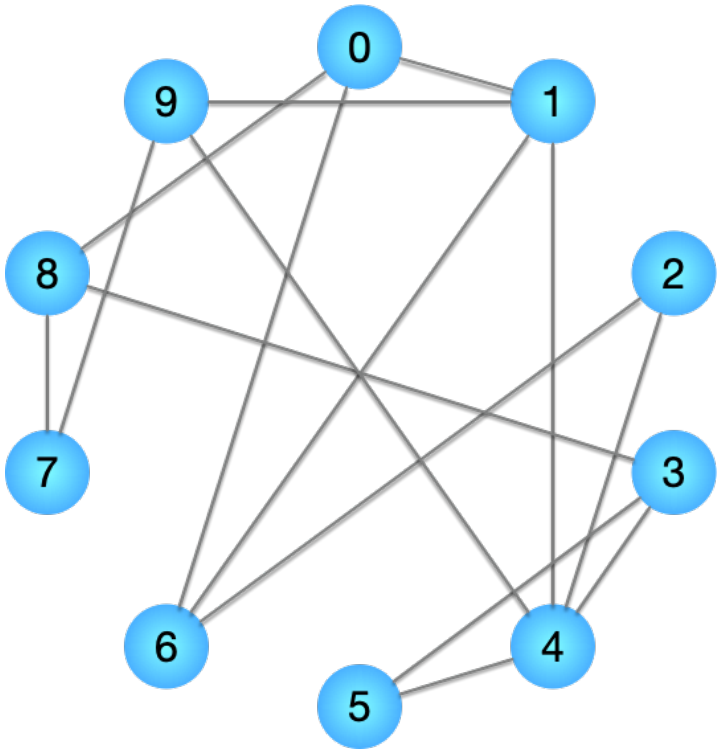

```
# 5. How do you check for the existence of an edges,
and how long does it take?
def find_edge_el(el, e, is_ordered=False):
    '''Check for existence of edge in edge list.

    If is_ordered is True, then all edges are in
    sorted order.

    This function does not assume directed or
    undirected edges, so the order
    (v1, v2) vs (v2, v1) DOES matter.

    This function will not work well with floating-
    point weights.

    Must search through all edges in order, or using
    a binary search.
    T(n, m) = O(m), or
    T(n, m) = O(lg(m))
    '''
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```



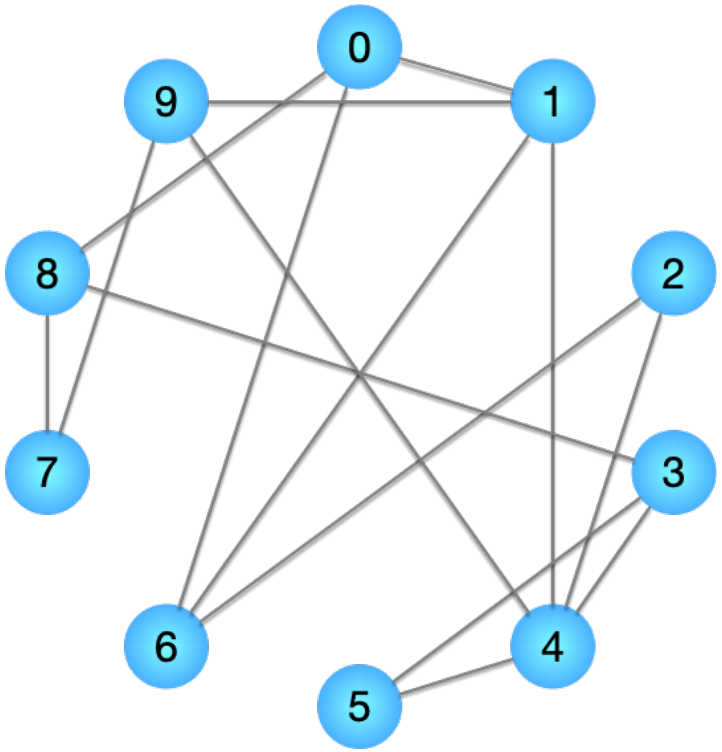

```
# Binary search
if is_ordered:
    left, right = 0, len(el) - 1
    while left <= right:
        mid = (left + right) // 2


        if e == el[mid]:
            return True
        elif e < el[mid]:
            right = mid - 1
        else:
            left = mid + 1


# Linear search
else:
    for edge in el:
        if e == edge:
            return True

return False
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```




```python
# 6. How do you find all neighbors of a vertex, and
how long does it take?
def find_neighbors_el(el, v, is_directed=False):
    '''Return all neighbors of a given vertex.

    This function does not assume edges are in sorted
    order.

    T(n, m) = O(m)
    '''
    neighbors = []
    for edge in el:
        v1, v2 = edge[0], edge[1]
        if v1 == v:
            neighbors.append(v2)
        elif not is_directed and v2 == v:
            neighbors.append(v1)

    return neighbors
```

```
edge_list = [
    (0, 1), (0, 6), (0, 8),
    (1, 4), (1, 6), (1, 9),
    (2, 4), (2, 6),
    (3, 4), (3, 5), (3, 8),
    (4, 5), (4, 9),
    (7, 8), (7, 9),
]
```

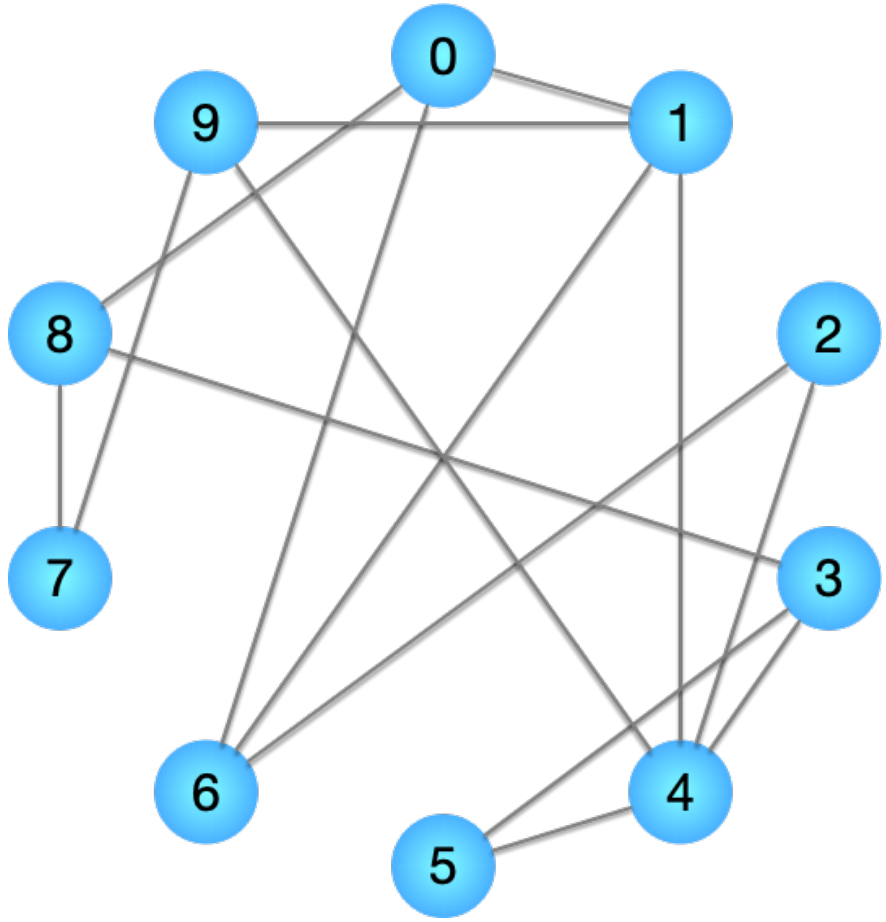



```python
# 7. How much memory is needed to store the graph?
def calc_memory_el(el):
    '''Calculate the approximate amount of memory
       used by el in bytes.

    Edge lists use 2 or 3 numbers for representing
    undirected, or directed graphs, respectively.

    M(n, m) = O(m)
    '''
    m = count_edges_el(el)

    # Memory used for storing vertices
    vertices_per_edge = 2
    bytes_per_int = 4
    memory_usage = m * vertices_per_edge * bytes_per_int

    # Check for edge weights
    if len(el[0]) == 3:
        bytes_per_float = 8
        memory_usage += m * bytes_per_float

    return memory_usage
```

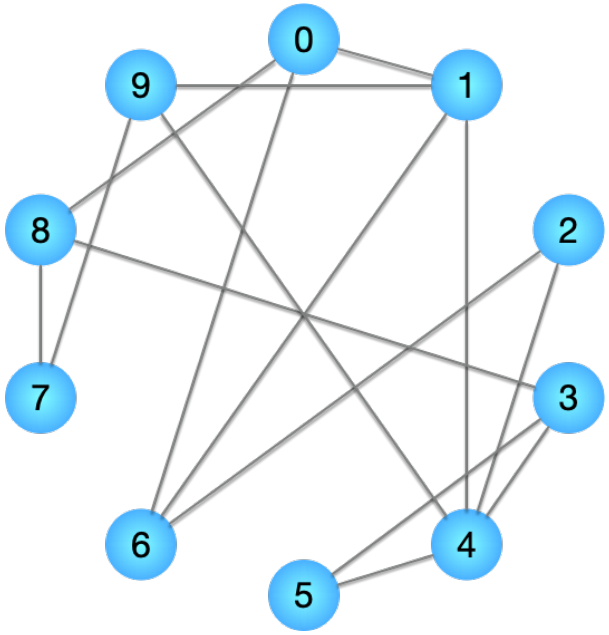
Weights? Directedness?

Adjacency Matrix Representation

```
adjacency_matrix = [
    [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
    [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
    [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
    [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
    [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
    [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
# Or T/F
# Symmetric matrix for undirected graphs
```

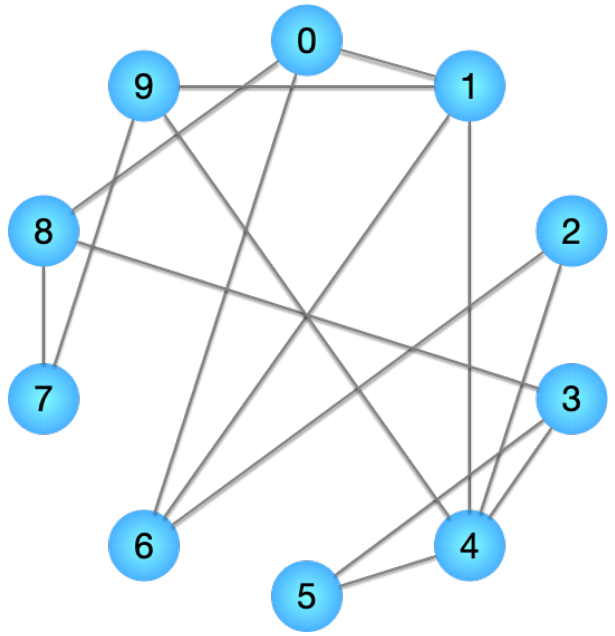


```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```



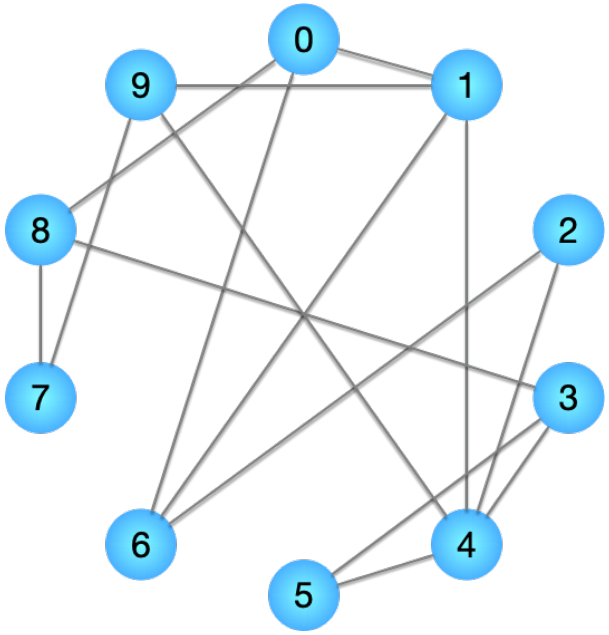

```python
# 1. How do we count the number of vertices, and how
long does it take?
def count_vertices_am(am):
    '''Retun the number of vertices in an adjacency
       matrix.

    T(n, m) = O(1)
    '''
    return len(am)
```

```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```



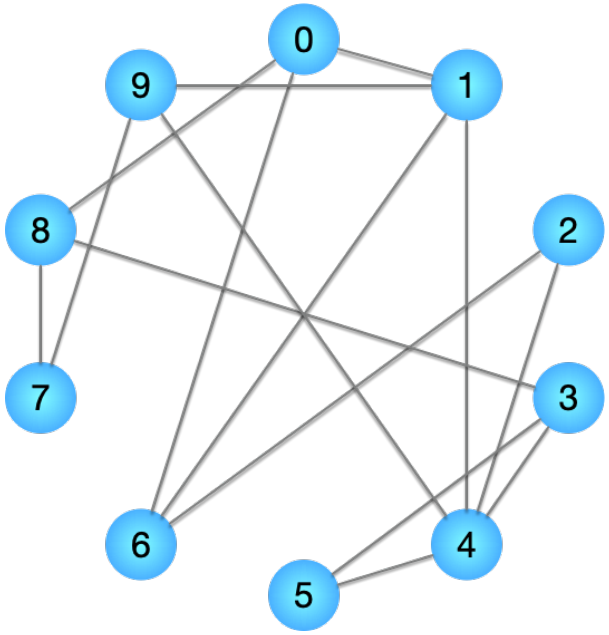

```python
# 2. How do we count the number of edges, and how
long does it take?
def count_edges_am(am, is_weighted=False, is_directed=False):
    '''Return the number of.

    We must loop over the entire matrix and look for
    non-zero entries.

    If is_weighted is True, then assume all entries
    are None or a floating-point number.

    If is_directed is False, then the matrix is
    symmetric.

    T(n, m) = O(n^2)
    '''
    if is_weighted:
        num_edges = sum(sum(1 for val in row
            if val is not None) for row in am)
    else:
        num_edges = sum(sum(row) for row in am)

    return num_edges if is_directed else num_edges // 2
```

```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```



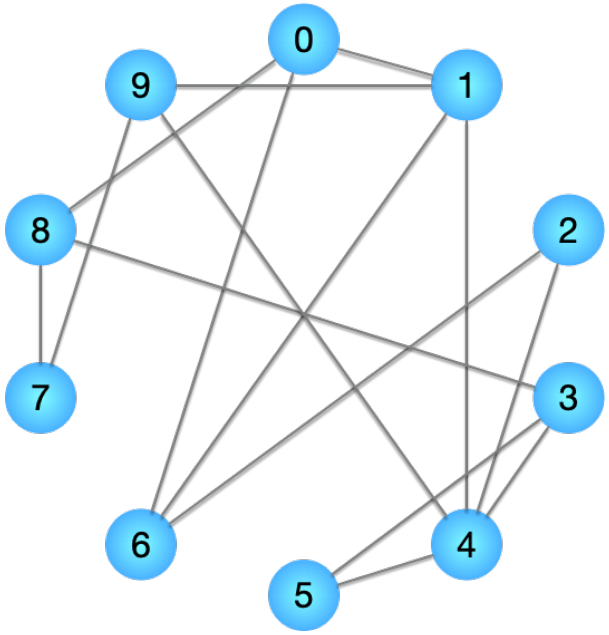

```python
# 3. How do we add vertices, and how long does it take?
def add_vertex_am(am, is_weighted=False):
    '''Add a new vertex to an adjacency matrix.

    Use the next available index.

    We must create a new row and a new column

    T(n, m) = O(n)
    '''
    default_value = None if is_weighted else 0

    # Add the new column
    for row in am:
        row.append(default_value)

    # Add the new row
    new_n = count_edges_am(am)
    am.append([default_value] * new_n)
    return am
```

```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```



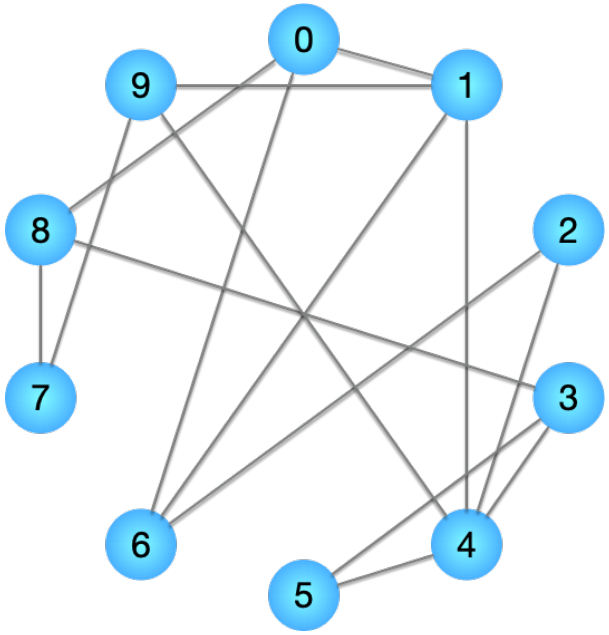

```python
# 4. How do we add edges,and how long does it take?
def add_edge_am(am, e, is_directed=False):
    '''Add a new edge to an adjacency matrix.

    If is_directed is False, then the matrix is
    symmetric.

    T(n, m) = O(1)
    '''
    # Check for edge weight
    if len(e) == 2:
        e = (e[0], e[1], 1)

    v1, v2, w = e
    am[v1][v2] = w
    if not is_directed:
        am[v2][v1] = w

    return am
```

```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```



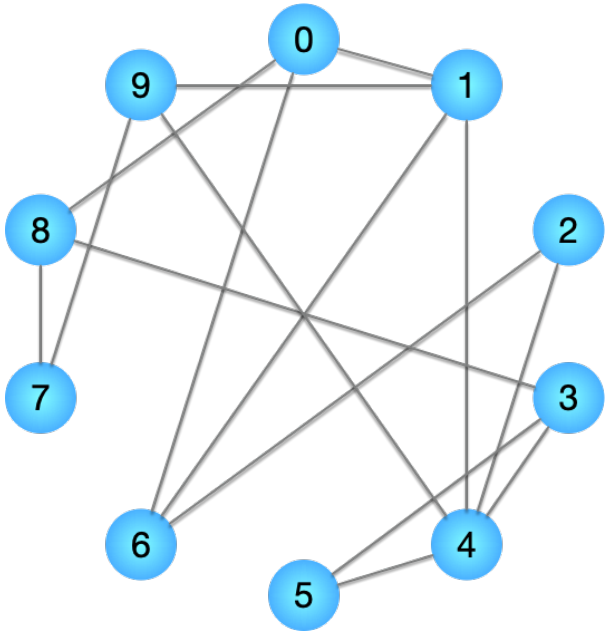

```python
# 5. How do you check for the existence of an edges,
and how long does it take?
def find_edge_am(am, e, is_weighted=False):
    '''Check for existence of edge in adjacency
       matrix.

    This function assumes that edges contain only v1
    and v2.

    T(n, m) = O(1)
    '''
    if is_weighted:
        v1, v2, _ = e
        return am[v1][v2] != None
    else:
        v1, v2 = e
        return am[v1][v2] == 1
```

```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```




```python
# 6. How do you find all neighbors of a vertex, and
how long does it take?
def find_neighbors_am(am, v, is_weighted=False):
    '''Return all neighbors of a given vertex.

    T(n, m) = O(n)
    '''

    default_value = None if is_weighted else 0
    neighbors = []

    for v2, w in enumerate(am[v]):
        if w != default_value:
            neighbors.append(v2)

    return neighbors
```

```
adjacency_matrix = [
 [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
 [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
 [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
]
```

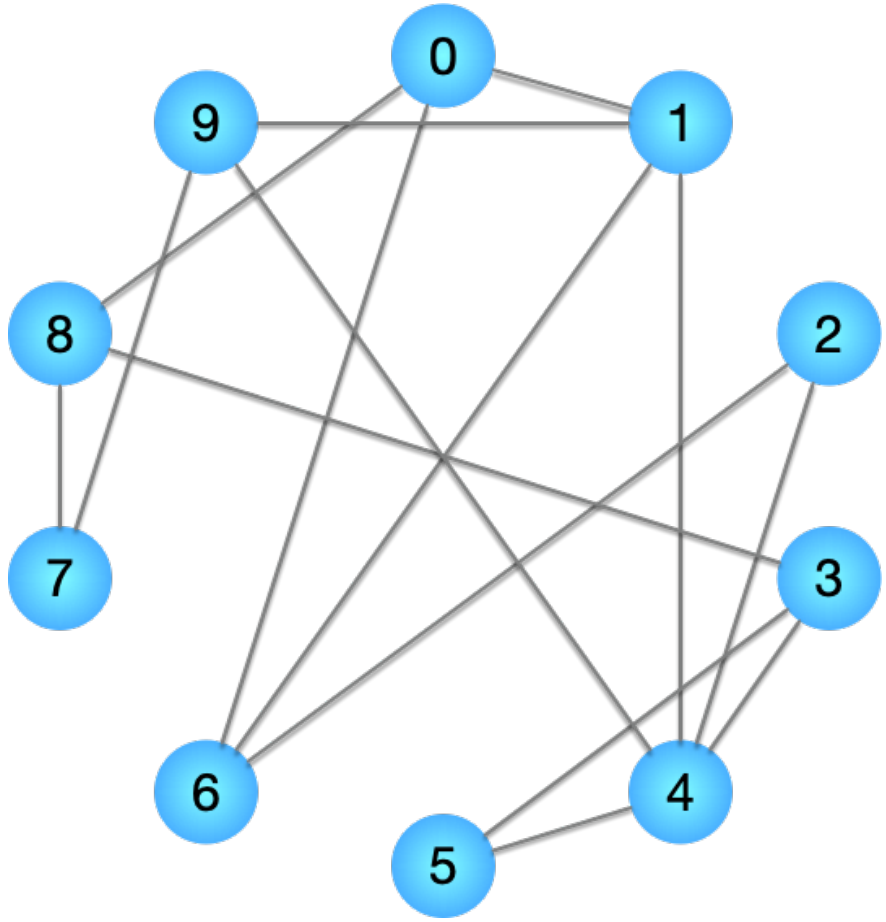



```python
# 7. How much memory is needed to store the graph?
def calc_memory_am(am, is_weighted=False):
    '''Calculate the approximate amount of memory
       used by am in bytes.

    Matrices store booleans or floats for
    representing un-weighted,
    or weighted graphs, respectively.

    M(n, m) = O(n^2)
    '''

    n = count_vertices_am(am)

    bytes_per_bool = 1
    bytes_per_float = 8
    bytes_per_cell = bytes_per_float if is_weighted
                     else bytes_per_bool
    return n * n * bytes_per_cell
```

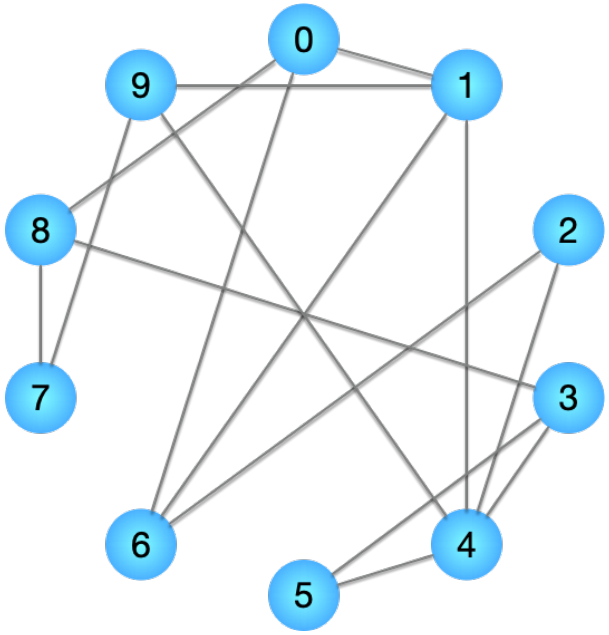
Weights? Directedness?

Adjacency List Representation

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```

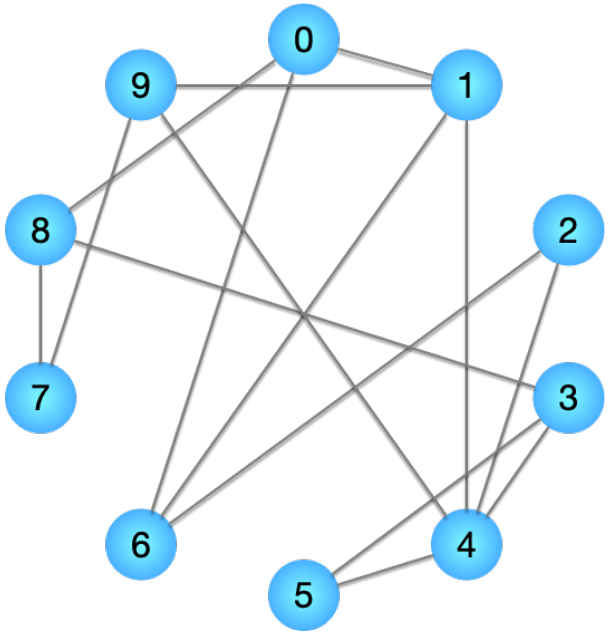


```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```



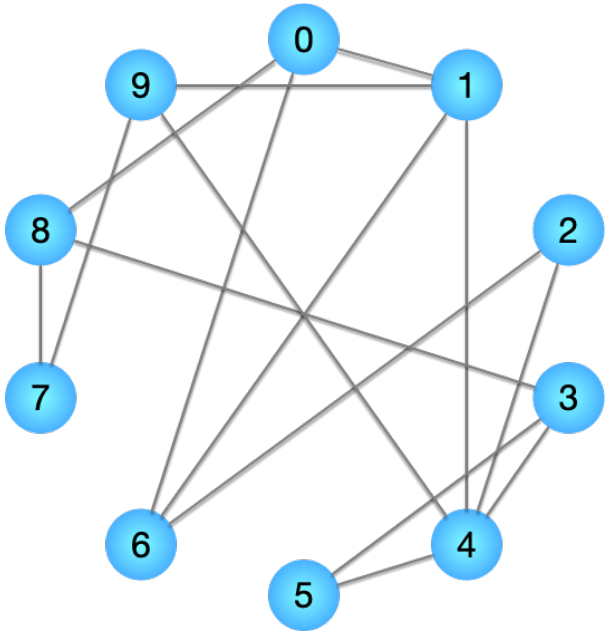

```python
# 1. How do we count the number of vertices, and how long does it take?
def count_vertices_al(al):
    '''Retun the number of vertices in an adjacency
       list.

    T(n, m) = O(1)
    '''
    return len(al)
```

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```



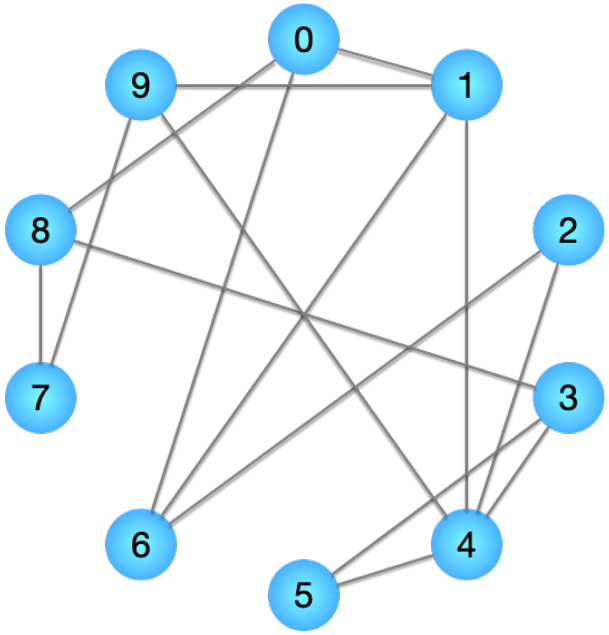

```python
# 2. How do we count the number of edges, and how long does it take?
def count_edges_al(al, is_directed=False):
    '''Return the number of edges in an adjacency list.

    We must loop over each vertex and count edges.

    T(n, m) = O(n)
    '''
    if is_directed:
        return sum(len(l) for l in al)
    else:
        return sum(len(l) for l in al) // 2
```

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```



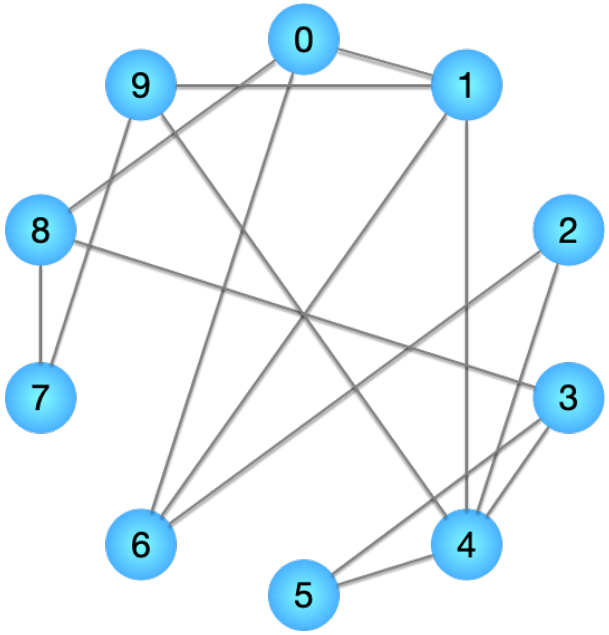

```python
# 3. How do we add vertices, and how long does it take?
def add_vertex_al(al):
    '''Add a new vertex to an adjacency list.

    Use the next available index.

    This function assumes that v is not already in the adjacency list.

    T(n, m) = O(1)
    '''
    al.append([])
    return al
```

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```



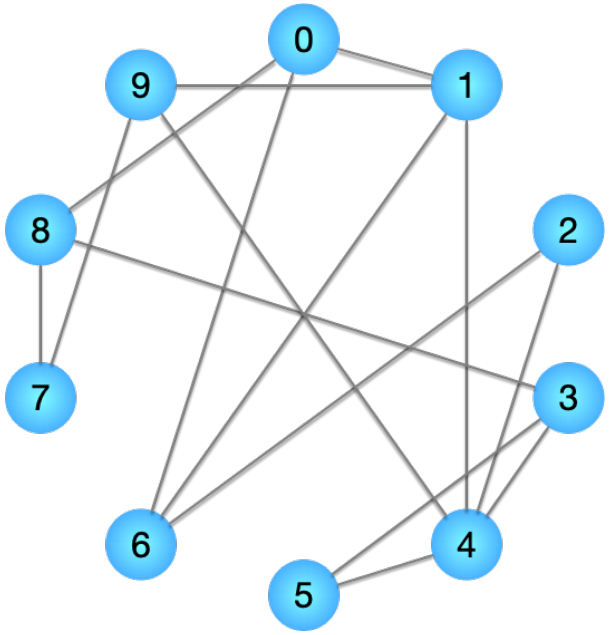

```python
# 4. How do we add edges,and how long does it take?
def add_edge_al(al, e):
    '''Add a new edge to an adjacency list.

    T(n, m) = O(1)
    '''
    v1, v2 = e[0], e[1]

    # Check for edge weight
    if len(e) == 3:
        v2 = (v2, e[2])

    al[v1].append(v2)
    return al
```

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```



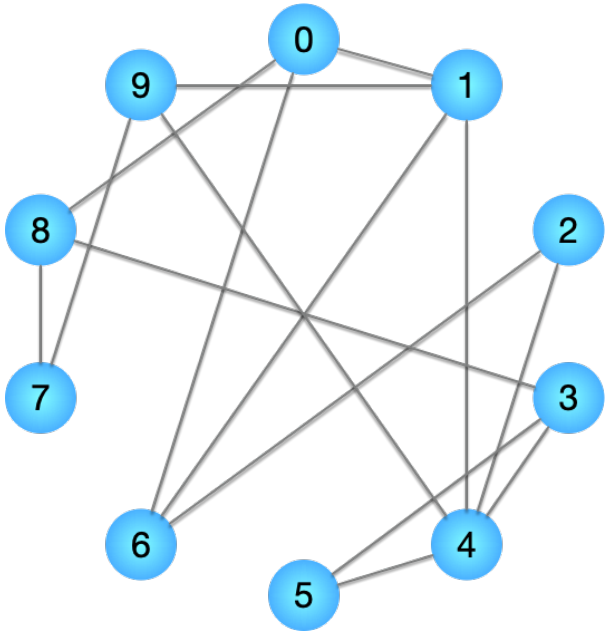

```python
# 5. How do you check for the existence of an edges,
and how long does it take?
def find_edge_al(al, e, is_weighted=False):
    '''Check for existence of edge in adjacency list.

    This function assumes that edges contain only v1
    and v2.

    T(n, m) = O(m)
    '''
    v1 = e[0]
    vw = (e[1], e[2]) if is_weighted else e[1]
    return vw in al[v1]
```

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```



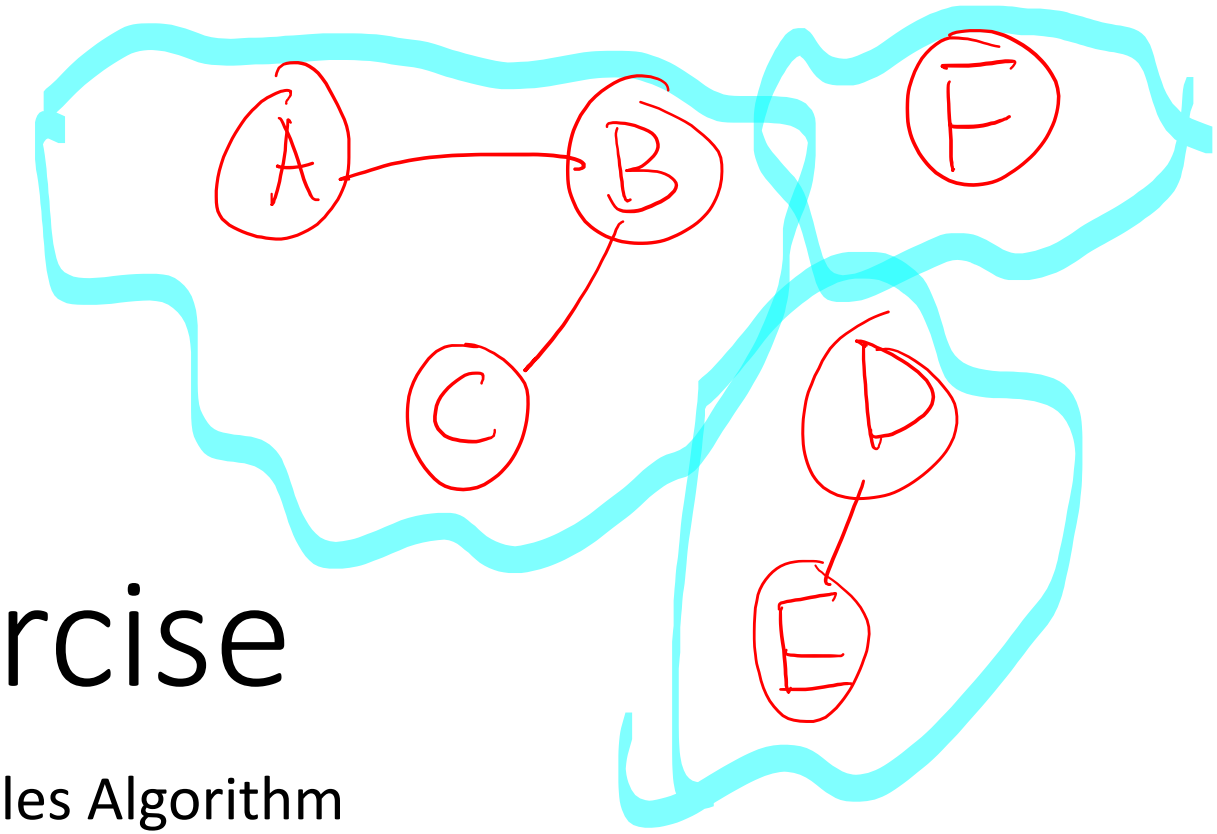

```python
# 6. How do you find all neighbors of a vertex, and
how long does it take?
def find_neighbors_al(al, v, is_weighted=False):
    '''Return all neighbors of a given vertex.

    T(n, m) = O(1)
    '''
    return al[v]
```

```
adjacency_list = [
    [1, 6, 8],
    [0, 4, 6, 9],
    [4, 6],
    [4, 5, 8],
    [1, 2, 3, 5, 9],
    [3, 4],
    [0, 1, 2],
    [8, 9],
    [0, 3, 7],
    [1, 4, 7],
]
```




```python
# 7. How much memory is needed to store the graph?
def calc_memory_al(al, is_weighted=False):
    '''Calculate the approximate amount of memory
       used by al in bytes.

    Adjacency lists use 1 or 2 numbers for
    representing each edge in weighted or unweighted
    graphs, respectively.

    M(n, m) = O(n + m)
    '''
    m = count_edges_al(al)

    # Memory used for storing vertices
    vertices_per_edge = 1
    bytes_per_int = 4
    memory_usage = m * vertices_per_edge * bytes_per_int

    # Check for edge weights
    if is_weighted:
        bytes_per_float = 8
        memory_usage += m * bytes_per_float
    return memory_usage
```

m ? n	Edge List	Adjacency Matrix	Adjacency List
Count Vertices	O(m)	O(1)	O(1)
Count Edges	O(1)	$O(n^2)$	O(n)
Add Vertex	O(1)	O(n)	O(1)
Add Edge	O(1)	(1)	O(1)
Check Edge	O(m) or O(lg(m))	O(1)	O(m)
Find Neighbors	O(m)	O(n)	O(1)
Memory Usage	O(m)	$O(n^2)$	O(n + m)



Exercise

Friend Circles Algorithm

BFS

call DFS on each node.

RFS