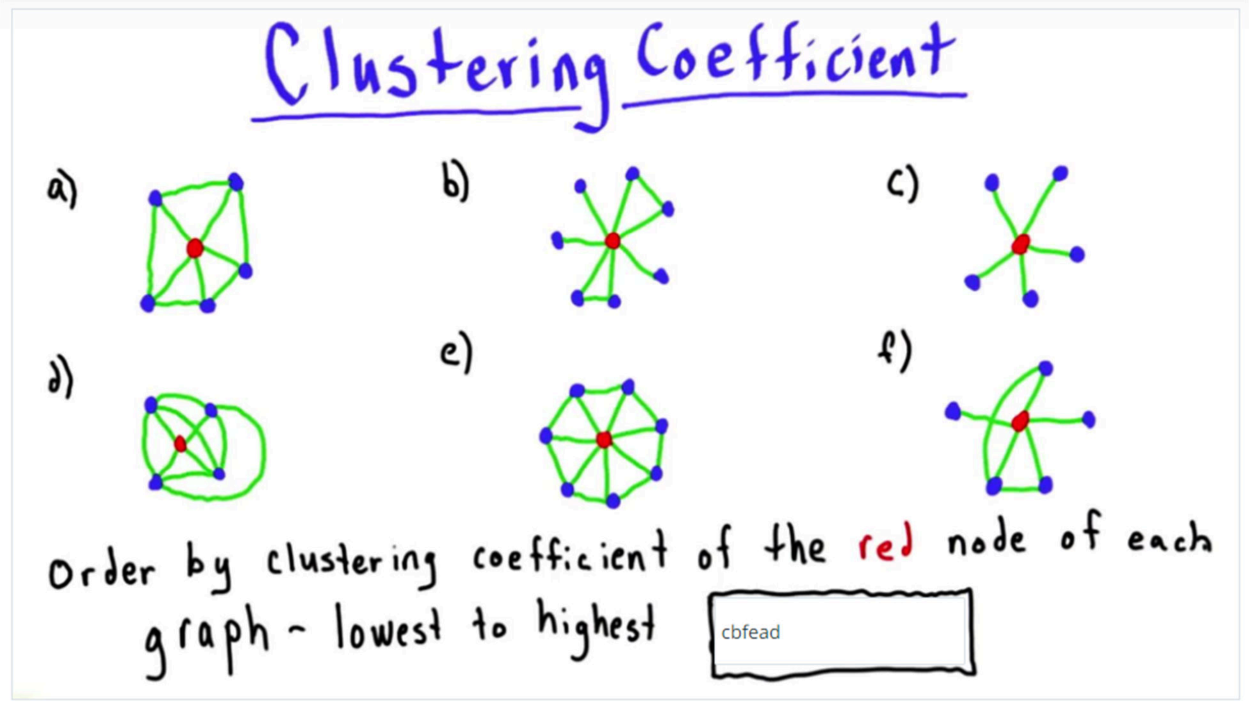
Clustering Coefficient

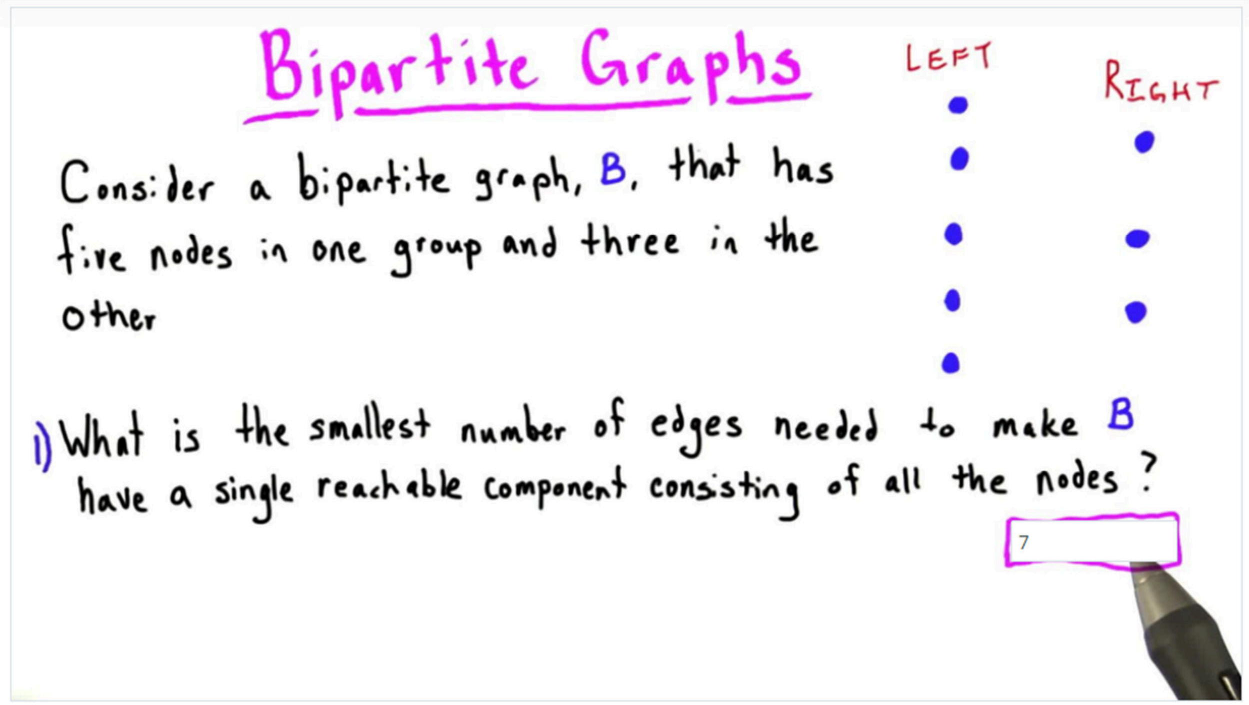


Clustering Coefficient

Order by clustering coefficient of the red node of each graph - lowest to highest

cbfead

Bipartite I



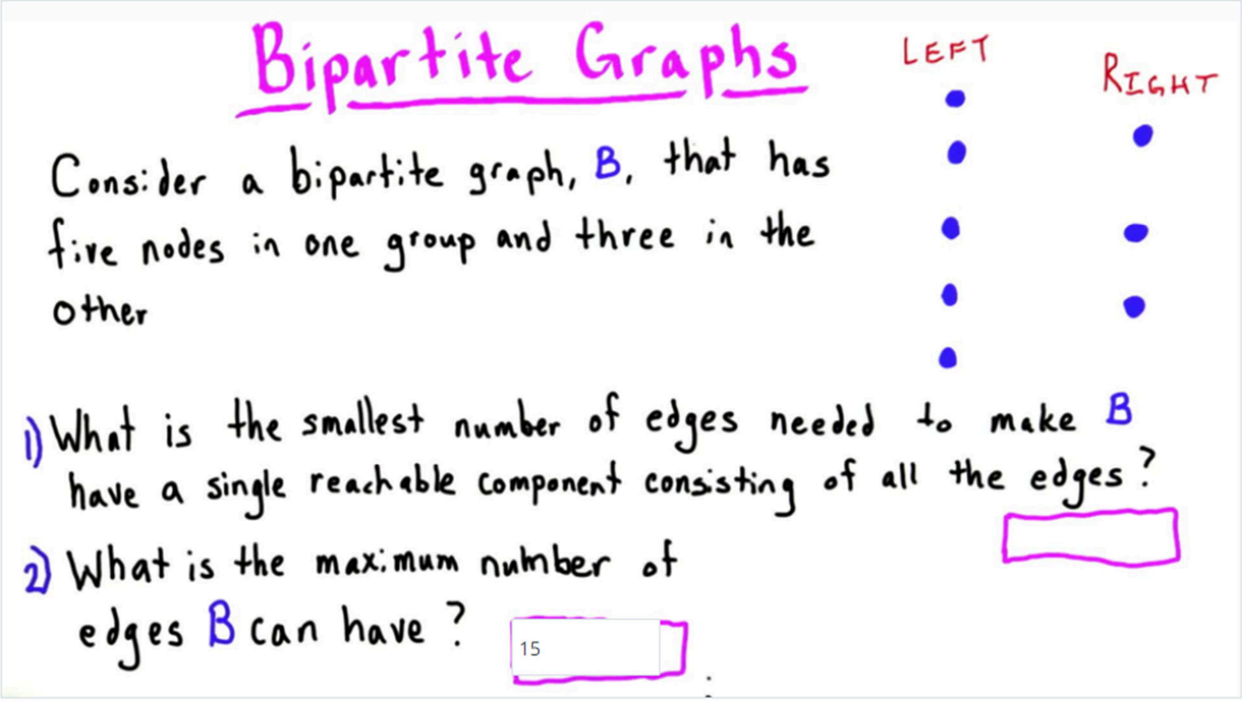
Bipartite Graphs

Consider a bipartite graph, B, that has five nodes in one group and three in the other

1) What is the smallest number of edges needed to make B have a single reachable component consisting of all the nodes?

7

Bipartite II



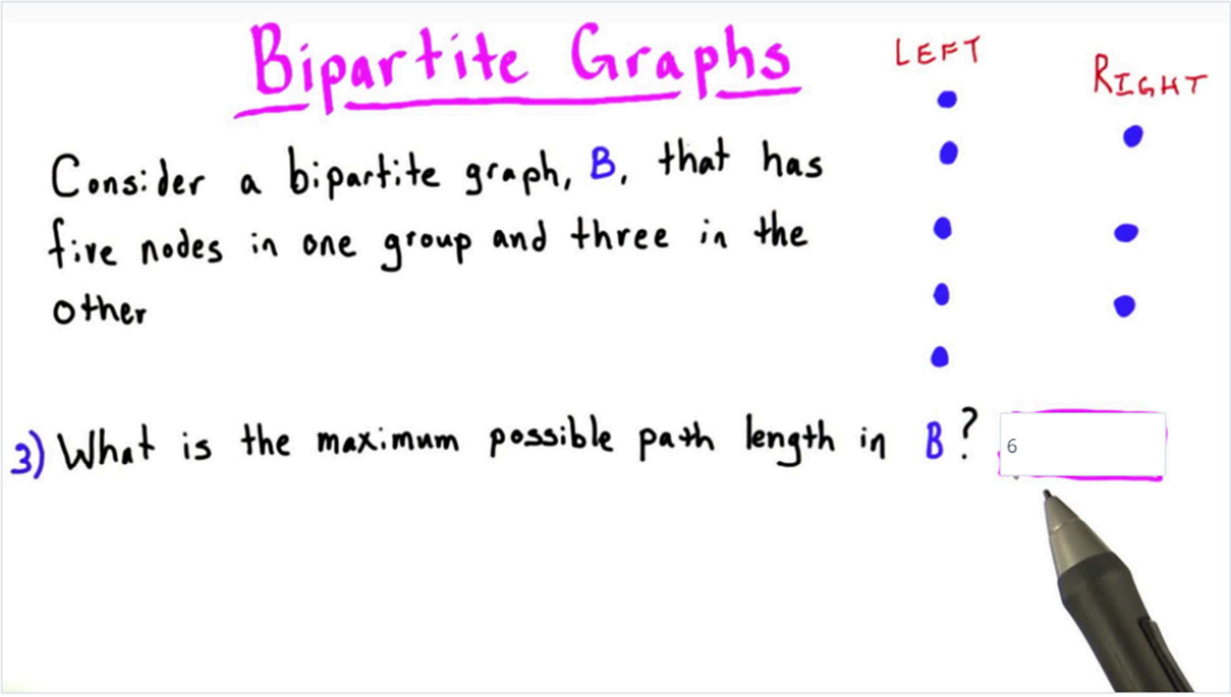
Bipartite Graphs

Consider a bipartite graph, B, that has five nodes in one group and three in the other

1) What is the smallest number of edges needed to make B have a single reachable component consisting of all the edges?

2) What is the maximum number of edges B can have? 15

Bipartite III

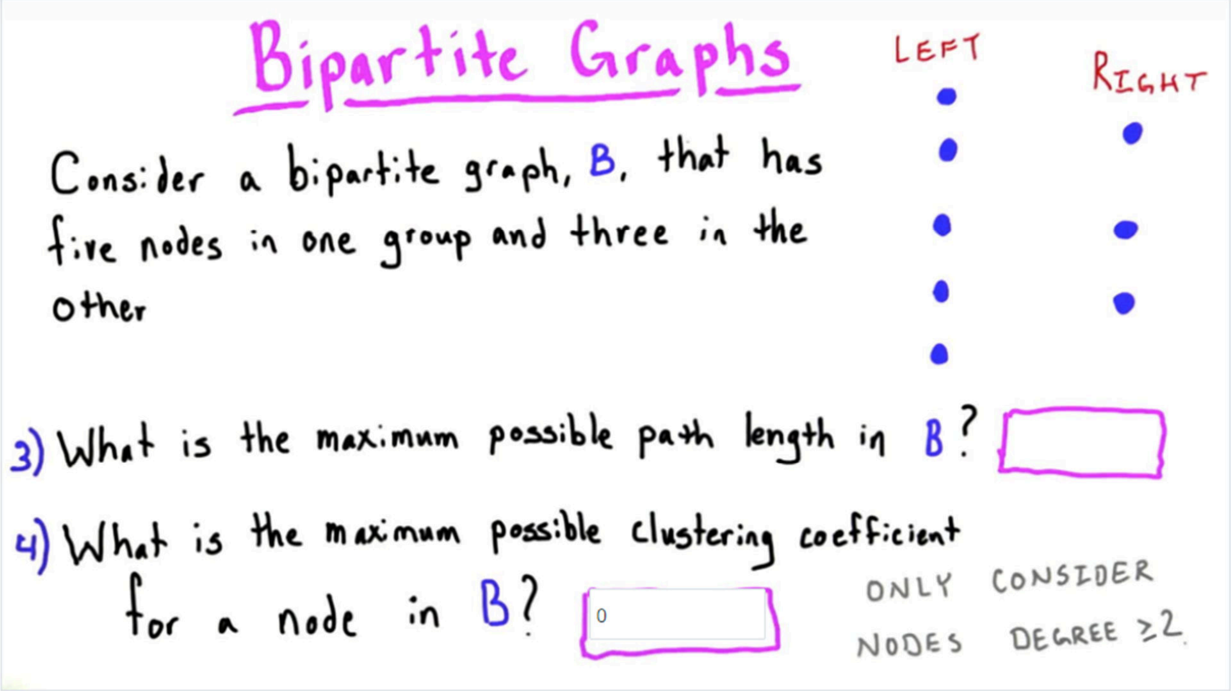


Bipartite Graphs

Consider a bipartite graph, B, that has five nodes in one group and three in the other

3) What is the maximum possible path length in B? 6

Bipartite IV



Bipartite Graphs

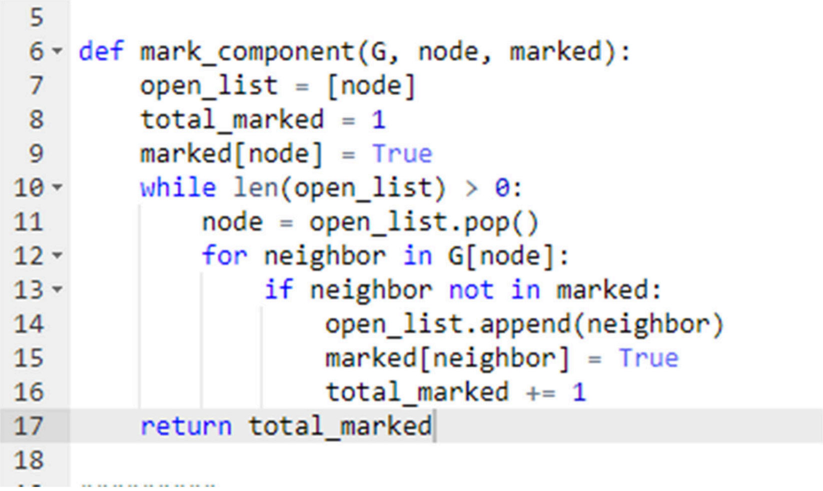
Consider a bipartite graph, B, that has five nodes in one group and three in the other

3) What is the maximum possible path length in B?

4) What is the maximum possible clustering coefficient for a node in B? 0

ONLY CONSIDER NODES DEGREE ≥2

Mark Component

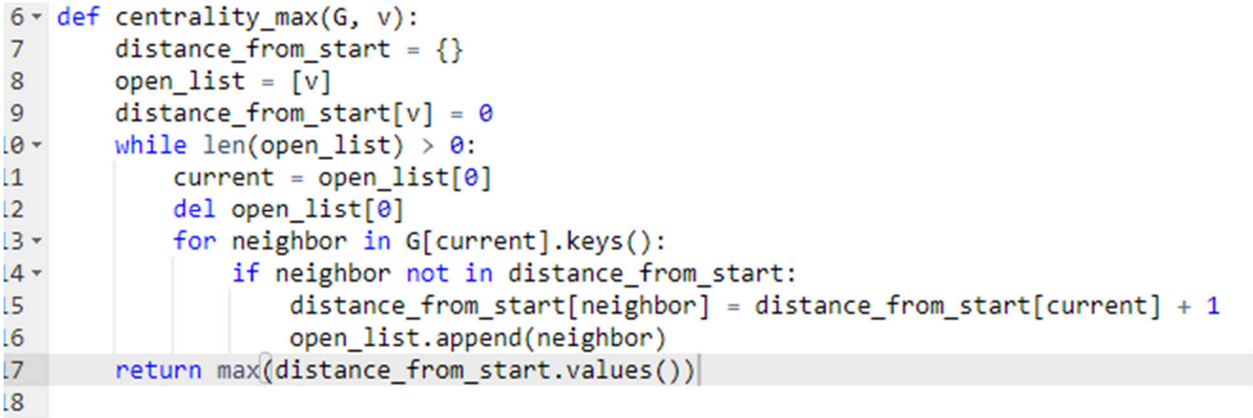


```

def mark_component(G, node, marked):
    open_list = [node]
    total_marked = 1
    marked[node] = True
    while len(open_list) > 0:
        node = open_list.pop()
        for neighbor in G[node]:
            if neighbor not in marked:
                open_list.append(neighbor)
                marked[neighbor] = True
                total_marked += 1
    return total_marked

```

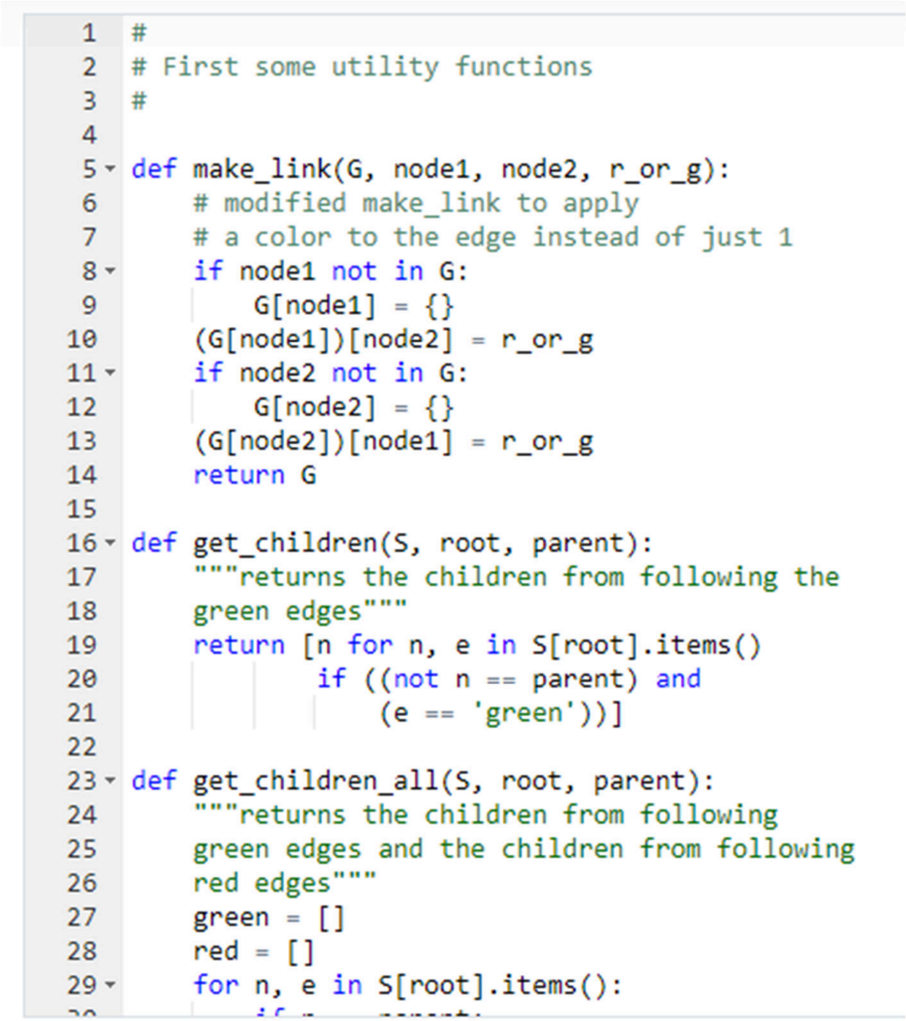
Centrality



```python
def centrality_max(G, v):
    distance_from_start = {}
    open_list = [v]
    distance_from_start[v] = 0
    while len(open_list) > 0:
        current = open_list[0]
        del open_list[0]
        for neighbor in G[current].keys():
            if neighbor not in distance_from_start:
                distance_from_start[neighbor] = distance_from_start[current] + 1
                open_list.append(neighbor)
    return max(distance_from_start.values())

```

Bridge Edges



```python
#
# First some utility functions
#

def make_link(G, node1, node2, r_or_g):
    # modified make_link to apply
    # a color to the edge instead of just 1
    if node1 not in G:
        G[node1] = {}
    (G[node1])[node2] = r_or_g
    if node2 not in G:
        G[node2] = {}
    (G[node2])[node1] = r_or_g
    return G

def get_children(S, root, parent):
    """returns the children from following the
    green edges"""
    return [n for n, e in S[root].items()
            if ((not n == parent) and
              (e == 'green'))]

def get_children_all(S, root, parent):
    """returns the children from following
    green edges and the children from following
    red edges"""
    green = []
    red = []
    for n, e in S[root].items():
```