Home Work 2

Xingyu Yan 50291116

1 BFS tree

The first part is the code I used to count tree height and sibling edge. Following the code, trees and numbers are printed, and the results are given. I assume the node "A" corresponds to "1", node "B" corresponds to "2", and so on.

In [1]:
```
edges = [[1,2], [1,3], [1,4], [2,4], [3,4], [3,17], [4,13], [4,12], [
5,9],
         [5,10], [5,14], [5,15], [5,16], [5,18], [6,7], [6,8], [6,9],
[6,16],
         [7,8], [7,12], [7,15], [7,16], [7,17], [10,15], [10,17], [11
,12], [11,18]]

class Node:
    def __init__(self,x):
        self.val = x
        self.nei = [];

node_set = set()
node_dict = {}

for n in range(1, 19):
    cur = Node(n)
    node_set.add(cur)
    node_dict[n] = cur

for edge in edges:
    left = node_dict[edge[0]]
    right = node_dict[edge[1]]
    left.nei.append(right)
    right.nei.append(left)
```

In [2]:
```
def BFS(root):
    height = 0
    visited = set()
    queue = []
    queue.append(root)
    sib_edge = 0

    while(queue):
        height += 1
        size = len(queue)
        level_node = set()

        for node in queue:
            print(node.val, end = " ")
            level_node.add(node)
        print(" ")

        for _ in range(size):
            node = queue.pop(0)
            for nei in node.nei:
                if nei in level_node:
                    sib_edge += 1
                elif not (nei in visited):
                    queue.append(nei)
                    visited.add(nei)

    return height, int(sib_edge/2)
```

In [3]:
```
def check_all(node_dict):
    h_max = -999
    h_min = 999
    sib_min = 999
    h_max_root = 0
    h_min_root = 0
    sib_min_root = 0

    for val, root in node_dict.items():
        print("\nBFS tree use node", val, "as root:")
        h, s = BFS(root)
        print("It has tree height = ", h, "sibling edge number = ", s
)
        if h > h_max:
            h_max = h
            h_max_root = val
        if h < h_min:
            h_min = h
            h_min_root = val
        if s < sib_min:
            sib_min = s
            sib_min_root = val
    return h_max, h_max_root, h_min, h_min_root, sib_min, sib_min_roo
t
```

In [4]:
```
h_max, h_max_root, h_min, h_min_root, sib_min, sib_min_root = check_a
ll(node_dict)
```

```
BFS tree use node 1 as root:
1
2 3 4
1 17 13 12
7 10 11
6 8 15 16 5 18
9 14
It has tree height =  6 sibling edge number =  7

BFS tree use node 2 as root:
2
1 4
2 3 13 12
17 7 11
10 6 8 15 16 18
5 9
14
It has tree height =  7 sibling edge number =  6

BFS tree use node 3 as root:
3
1 4 17
2 3 13 12 7 10
11 6 8 15 16 5
18 9 14
It has tree height =  5 sibling edge number =  6

BFS tree use node 4 as root:
4
1 2 3 13 12
4 17 7 11
10 6 8 15 16 18
5 9
14
It has tree height =  6 sibling edge number =  7

BFS tree use node 5 as root:
5
9 10 14 15 16 18
5 6 17 7 11
8 3 12
1 4
2 13
It has tree height =  6 sibling edge number =  4

BFS tree use node 6 as root:
6
7 8 9 16
6 12 15 17 5
4 11 10 3 14 18
1 2 13
It has tree height =  5 sibling edge number =  6

BFS tree use node 7 as root:
7
6 8 12 15 16 17
7 9 4 11 5 10 3
```

```
1 2 13 18 14
It has tree height =  4 sibling edge number =  6

BFS tree use node 8 as root:
8
6 7
8 9 16 12 15 17
5 4 11 10 3
14 18 1 2 13
It has tree height =  5 sibling edge number =  4

BFS tree use node 9 as root:
9
5 6
9 10 14 15 16 18 7 8
17 11 12
3 4
1 2 13
It has tree height =  6 sibling edge number =  7

BFS tree use node 10 as root:
10
5 15 17
9 10 14 16 18 7 3
6 11 8 12 1 4
2 13
It has tree height =  5 sibling edge number =  6

BFS tree use node 11 as root:
11
12 18
4 7 11 5
1 2 3 13 6 8 15 16 17 9 10 14
It has tree height =  4 sibling edge number =  8

BFS tree use node 12 as root:
12
4 7 11
1 2 3 13 12 6 8 15 16 17 18
9 5 10
14
It has tree height =  5 sibling edge number =  7

BFS tree use node 13 as root:
13
4
1 2 3 13 12
17 7 11
10 6 8 15 16 18
5 9
14
It has tree height =  7 sibling edge number =  7

BFS tree use node 14 as root:
14
5
9 10 14 15 16 18
```

```
6 17 7 11
8 3 12
1 4
2 13
It has tree height =  7 sibling edge number =  4

BFS tree use node 15 as root:
15
5 7 10
9 14 15 16 18 6 8 12 17
11 4 3
1 2 13
It has tree height =  5 sibling edge number =  6

BFS tree use node 16 as root:
16
5 6 7
9 10 14 15 16 18 8 12 17
11 4 3
1 2 13
It has tree height =  5 sibling edge number =  5

BFS tree use node 17 as root:
17
3 7 10
1 4 17 6 8 12 15 16 5
2 13 9 11 14 18
It has tree height =  4 sibling edge number =  7

BFS tree use node 18 as root:
18
5 11
9 10 14 15 16 18 12
6 17 7 4
8 3 1 2 13
It has tree height =  5 sibling edge number =  5
```

Based on the calculation above, we get the result:

- BFS tree rooted at node G, K, Q has shortest height 4
- BFS tree rooted at node B, M, N has longest height 7
- BFS tree rooted at node E, H, N has leaset sibling edge 4

The nodes in each level of the tree has been printed in above. We can also print the parent/child relation of the tree.

2 Design Algorithm

The algorithm is designed and used in Question 1.

- We first set one node as root.
- Then use BFS to traverse the graph.
- In each level of the BFS tree, we construct a set of nodes to save the nodes in current level.
- During dequeue, we can check the neighbors of current poped node. If its neighbor is in the set, it means this is a sibling edge. Else if it has not been discovered, we enqueue it.
- We can set every node in the graph as the root, then do BFS and get every tree. The tree with least sibling edges can be find.

$$G(V,E), |V| = n, |E| = m$$

Analysis:

- The above single BFS takes $O(n)$ in space, $O(m+n)$ in time (because each node is added to a set in each level).
- Because we take each node as root, and do the BFS, we have time complexity totaly
$$O(n^2 + n * m)$$
or
$$O(n^3)$$

3 K-core & K-trusses

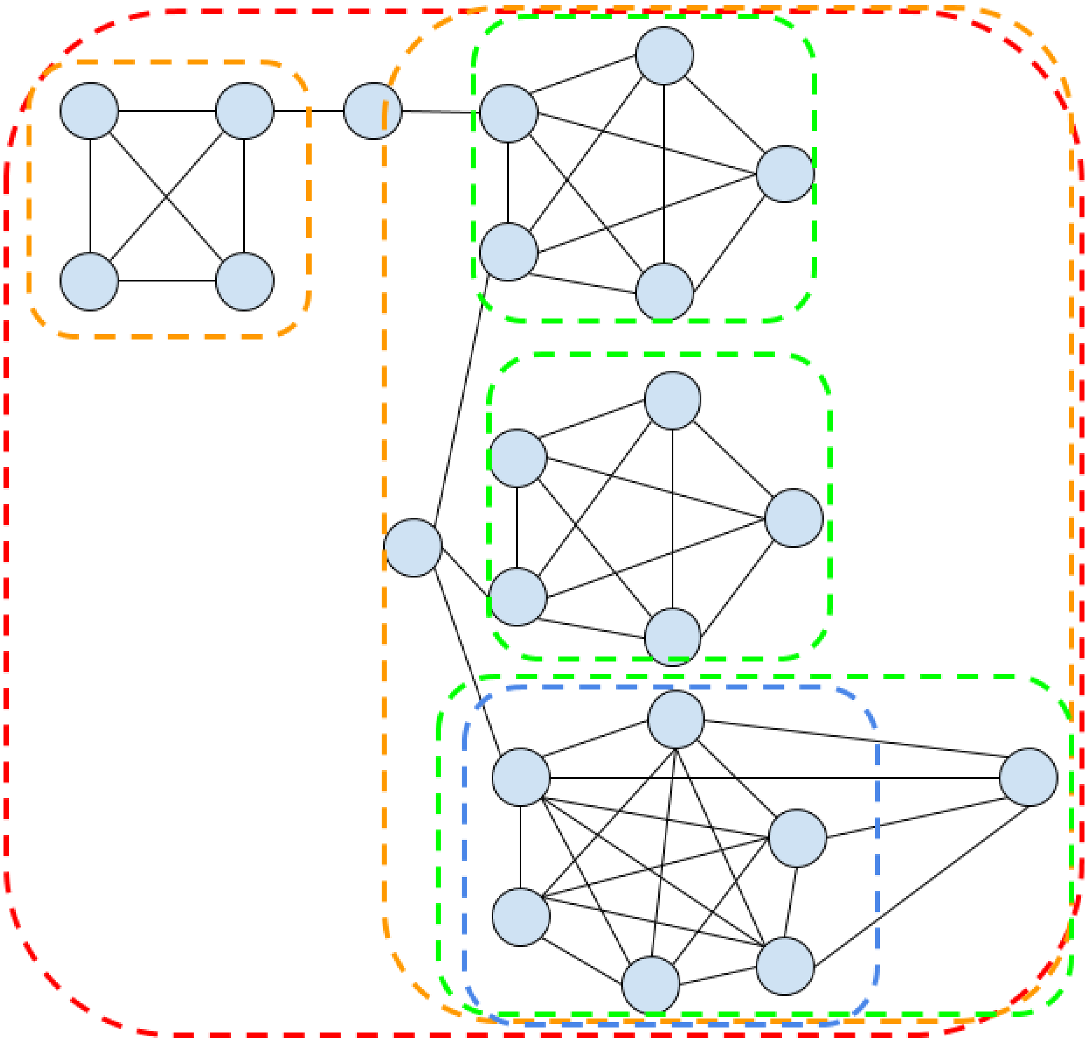
(a) K-core

Red: k = 2

Yellow: k = 3

Green: k = 4

Blue: k= 5



(b) K-truss

Red: k = 0

Yellow: k = 1

Green: k = 2

Blue: k = 3