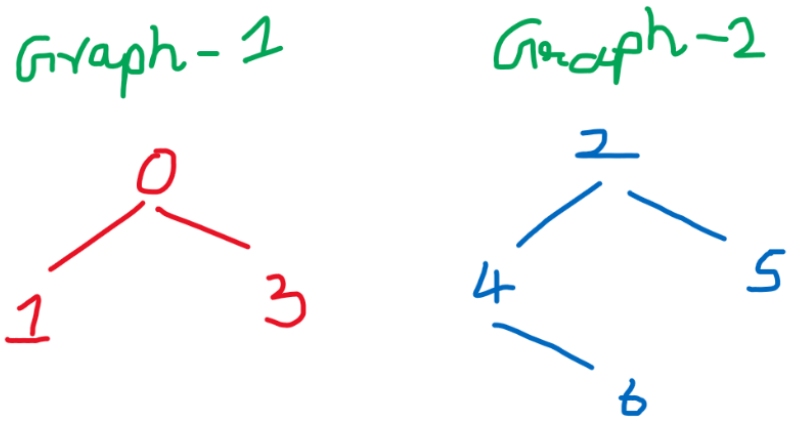
Assignment-2 **Prashanth.S**
19MID0020

DFS for disconnected graph

```
class Graph:
    def __init__(self,nVertices):
        self.nVertices = nVertices
        self.adjMatrix = [[0 for i in range(nVertices)] for j in range

    def addEdge(self,v1,v2):
        self.adjMatrix[v1][v2] = 1
        self.adjMatrix[v2][v1] = 1

    def removeEdge(self,v1,v2):  ## Before removing, check whether the
        if self.containsEdge(v1,v2) is False:
            return
        else:
            self.adjMatrix[v1][v2] = 0
            self.adjMatrix[v2][v1] = 0

    def containsEdge(self,v1,v2):   ## if there is an edge,then it wil
        if self.adjMatrix[v1][v2]>0:
            return True
        else:
            return False

class DFS_diconnected(Graph):
    def __dfsHelper(self,sv,visited):  ## private class
        print(sv,end=' ')
        visited[sv] = True
        for i in range(self.nVertices):
            ## if there is an edge and that edge is not visited
            if (self.adjMatrix[sv][i]>0) and (visited[i] is False):
                self.__dfsHelper(i,visited)

```

```
    def dfs(self):
        cnt = 0  ## to maintain the count of number of disconnected gr
        visited = [False for i in range(self.nVertices)]  ## maintaini
        for i in range(self.nVertices):
            if visited[i] is False:  ## if that vertex is not at all v
                cnt+=1
                print("\nGraph - {}".format(cnt))
                self.__dfsHelper(i,visited)

    def __str__(self):
        return str(self.adjMatrix)

class Graph:
    def __init__(self,nVertices):
        self.nVertices = nVertices
        self.adjMatrix = [[0 for i in range(nVertices)] for j in range


    def addEdge(self,v1,v2):
        self.adjMatrix[v1][v2] = 1
        self.adjMatrix[v2][v1] = 1

    def removeEdge(self,v1,v2):  ## Before removing, check whether the
        if self.containsEdge(v1,v2) is False:
            return
        else:
            self.adjMatrix[v1][v2] = 0
            self.adjMatrix[v2][v1] = 0

    def containsEdge(self,v1,v2):   ## if there is an edge,then it wil
        if self.adjMatrix[v1][v2]>0:
            return True
        else:
            return False

    def __str__(self):
        return str(self.adjMatrix)


if __name__ == '__main__':
    obj1 = DFS_diconnected(7)
    obj1.addEdge(0,1)
    obj1.addEdge(0,3)

    obj1.addEdge(2,4)
    obj1.addEdge(2,5)
    obj1.addEdge(4,6)

    obj1.dfs()

```



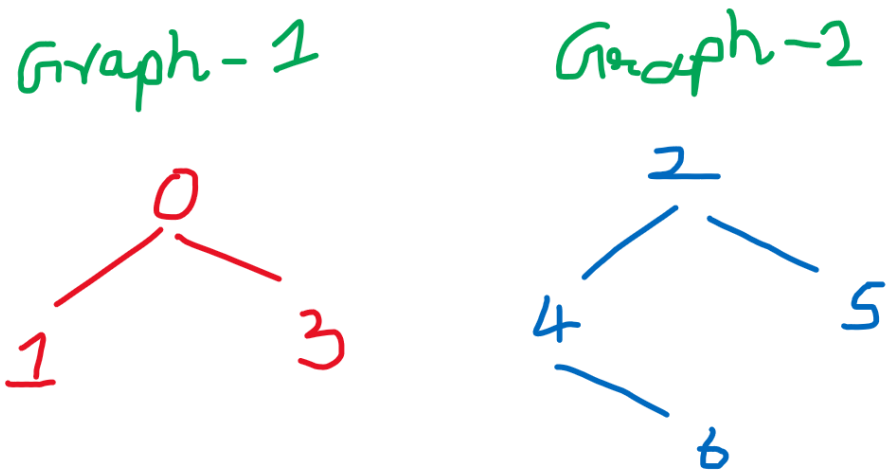
Output

```
Graph - 1
0 1 3
Graph - 2
2 4 6 5
***Repl Closed***
```

BFS for disconnected graph

```
1  import queue
2
3  class Graph:
4      def __init__(self,nVertices):
5          self.nVertices = nVertices
6          self.adjMatrix = [[0 for i in range(nVertices)] for j in range
7
8      def addEdge(self,v1,v2):
9          self.adjMatrix[v1][v2] = 1
10         self.adjMatrix[v2][v1] = 1
11
12     def removeEdge(self,v1,v2):  ## Before removing, check whether the
13         if self.containsEdge(v1,v2) is False:
14             return
15         else:
16             self.adjMatrix[v1][v2] = 0
17             self.adjMatrix[v2][v1] = 0
```

```
18
19      def containsEdge(self,v1,v2):   ## if there is an edge,then it wil
20          if self.adjMatrix[v1][v2]>0:
21              return True
22          else:
23              return False
24
25      def __str__(self):
26          return str(self.adjMatrix)
27
28  class BFS_disconnected(Graph):
29      def __bfsHelper(self,sv,visited):
30          q = queue.Queue()
31
32          q.put(sv)   # intially pushing 0 into the queue


33          visited[sv] = True  # and 0 is visited
34
35          while q.empty() is False:
36              u = q.get() ## After Dequeue,start exploring all the verti
37              print(u,end=' ')
38
39              for v in range(self.nVertices):  ## if a vertex is there a
40                  if (self.adjMatrix[u][v]>0 and visited[v] is False):
41                      q.put(v)
42                      visited[v] = True
43
44      def bfs(self):
45          cnt = 0  ## to maintain the count of number of disconnected gr
46          visited = [False for i in range(self.nVertices)]  ## maintaini
47          for i in range(self.nVertices):

48              if visited[i] is False:  ## if that vertex is not at all v
49                  cnt+=1
50                  print("\nGraph - {}".format(cnt))
51                  self.__bfsHelper(i,visited)
52
53      def __str__(self):
54          return str(self.adjMatrix)
55
56  if __name__ == '__main__':
57      obj1 = BFS_disconnected(7)
58      obj1.addEdge(0,1)
59      obj1.addEdge(0,3)
60
61      obj1.addEdge(2,4)
62      obj1.addEdge(2,5)
63      obj1.addEdge(4,6)
64
65      obj1.bfs()
```



Output:

```
Graph - 1
0 1 3
Graph - 2
2 4 5 6
***Repl Closed***
```

Sum of Subsets

```python
def Sum_Subset(index,list_total,weight):
    if sum(list_total)==weight:
        print(*list_total)
        return

    elif sum(list_total)>weight:
        return

    else:
        for i in range(index,len(list1)):
            list_total.append(list1[i])
            Sum_Subset(i+1,list_total,weight)
            list_total.pop()

if __name__ == '__main__':
    list1 = list(map(int,input("Enter the elements : ").split()))
    weight= int(input("Enter the total weight : "))
    list_total = []
    print("The possible weighs are : ")
    Sum_Subset(0,list_total,weight)
```

Output:

```
Enter the elements : 5 10 12 13 15 18
Enter the total weight : 30
The possible weighs are :
5 10 15
5 12 13
12 18

***Repl Closed***
```