In [1]:
```
#Breadth first search
graph = {
    'A':['B','C'],
    'B':['D','E'],
    'C':['F'],
    'D':[],
    'E':['F'],
    'F':[]
}
visited=[]
queue=[]

def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)

    while queue:
        s=queue.pop(0)
        print(s,end='')

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
bfs(visited,graph,'A')
```

```
ABCDEF
```

In [2]:
```
#Depth first search
graph = {
    'A':['B','C'],
    'B':['D','E'],
    'C':['F'],
    'D':[],
    'E':['F'],
    'F':[]
}

visited=set()

def dfs(visited,graph,node):
    if node not in visited:
        print(node)
        visited.add(node)


        for neighbour in graph[node]:
            dfs(visited,graph,neighbour)
dfs(visited,graph,'A')
```

```
A
B
D
E
F
C
```

In [8]:
```
#Best first Search

from queue import PriorityQueue
v=14
graph=[[] for i in range (v)]

def best_first_search(actual_Src,target,n):
    visited=[False]*n
    pq=PriorityQueue()
    pq.put((0,actual_Src))
    visited[actual_Src]=True

    while pq.empty()==False:
        u=pq.get()[1]
        print(u,end=' ')
        if u==target:
            break

        for v,c in graph[u]:
            if visited[v]==False:
                visited[v]=True
                pq.put((c,v))
    print()

def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))
addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

```
0 1 3 2 8 9
```

In [10]:
```
#branch & bound (knapsack)
wt = [7, 4, 5, 3]
p = [100, 50, 60, 44]
c = 10
a=[]
for i in range(0,len(wt)):
    for j in range(i+1,len(wt)):
            if(wt[i]+wt[j]<=c): a.append(p[i]+p[j])
max(a)
```

Out[10]:
```
144
```

In [11]:
```
#Finding the Maximum Number in the list using Steepest-Ascent Hill Climbing
def steepest_ascent_hill_climbing(numbers):

    current_max = numbers[0]

    for num in numbers:

        if num > current_max:

            current_max = num

    return current_max

numbers = [1, 3, 7, 12, 9, 5]

max_number = steepest_ascent_hill_climbing(numbers)

print(f"The maximum number in the list is: {max_number}")
```

```
The maximum number in the list is: 12
```

In [12]:
```
#A* Search
from collections import defaultdict
jug1,jug2,aim=4,3,2
visited=defaultdict(lambda : False)

def waterjug(amt1,amt2):
    if (amt1==aim and amt2==0) or (amt2==aim and amt1==0):
        return True
    if visited[(amt1,amt2)]==False:
        print(amt1,amt2)
        visited[(amt1,amt2)]=True

        return (waterjug (0,amt2) or
                waterjug(amt1,0) or
                waterjug(jug1,amt2) or
                waterjug(amt1,jug2) or
                waterjug (amt1+min(amt2,(jug1-amt1)),
                          amt2-min(amt2,(jug1-amt1))) or
                waterjug (amt1-min(amt1,(jug2-amt2)),
                          amt2+min(amt1,(jug2-amt2))))
    else:
        return False
print("Steps")
waterjug(0,0)
```

```
Steps
0 0
4 0
4 3
0 3
3 0
3 3
4 2
```

Out[12]:
```
True
```

In [ ]: