In [1]:
```
#Fibbonacci without recursion
nterms = int(input("Enter number of terms: "))
n1, n2 = 0, 1
count = 0
if nterms<=0:
    print("Please enter positive number!!")
elif nterms==1:
    print("Fibbonacci sequence: ")
    print(n1)
else:
    print("fibonacci sequence: ")
    while count< nterms:
        print(n1)
        nth = n1+n2
        n1 = n2
        n2=nth
        count+=1
```

```
Enter number of terms: 5
fibonacci sequence:
0
1
1
2
3
```

In [3]:
```
#fibonacci with recursion
def fibo(n):
    if(n<=1):
        return n
    else:
        return (fibo(n-1)+fibo(n-2))
n = int(input("Enter number of terms: "))
print("Fibonacci sequence: ")
for i in range(n):
    print(fibo(i))
```

```
Enter number of terms: 5
Fibonacci sequence:
0
1
1
2
3
```

In [7]:
```
#huffman coding
import heapq
class node:
    def __init__(self, freq, symbol, left=None, right=None):
        self.freq = freq
        self.symbol = symbol
        self.left = left
        self.right = right
        self.huff = ''
    def __lt__(self, nxt):
        return self.freq < nxt.freq
def printNodes(node, val=''):
    newVal = val+str(node.huff)
    if(node.left):
        printNodes(node.left, newVal)
    if(node.right):
        printNodes(node.right, newVal)
    if(not node.left and not node.right):
        print(f"{node.symbol} -> {newVal}")

chars = ['a','b','c','d','e','f']
freq = [5,9,12,13,16,45]
nodes = []


for x in range(len(chars)):
    heapq.heappush(nodes,node(freq[x], chars[x]))

while len(nodes)>1:
    left = heapq.heappop(nodes)
    right = heapq.heappop(nodes)
    left.huff = 0
    right.huff = 1

    newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)
    heapq.heappush(nodes, newNode)
printNodes(nodes[0])
```

```
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
```

In [2]:

```
#Fractional Knapsack
class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight

def fractionalKnapsack(W, arr):
    arr.sort(key=lambda x:(x.value/x.weight), reverse=True)
    finalValue = 0.0
    for item in arr:
        if item.weight <= W:
            W -= item.weight
            finalValue += item.value
        else:
            finalValue += item.value * W/item.weight
            break
    return finalValue

if __name__ == "__main__":
    W=50
    arr = [Item(60, 10), Item(100, 20), Item(120, 30)]
    max_val = fractionalKnapsack(W, arr)
    print(max_val)
```

```
240.0
```

In [7]:

```
#0/1 Knapsack problem
def knapSack(W, wt, val, n):

    dp = [0 for i in range(W+1)]

    for i in range(1, n+1):
        for w in range(W, 0, -1):
            if wt[i-1] <= w:
                dp[w] = max(dp[w], dp[w-wt[i-1]]+val[i-1])

    return dp[W]
```

```
if __name__ == "__main__":
    val = [60, 100, 120]
    wt = [10, 20, 30]
    W = 50
    n = len(val)
    print(knapSack(W, wt, val, n))
```

```
220
```

In [11]:

```
#N-Queens
print("Enter number of queen: ")
N = int(input())
board = [[0]*N for _ in range(N)]
def is_attack(i, j):
    for k in range(0, N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    for k in range(0, N):
        for l in range(0, N):
            if(k+l == i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False

def N_queen(n):
    if n==0:
        return True
    for i in range(0, N):
        for j in range(0, N):
            if(not(is_attack(i,j))) and (board[i][j]!=1):
                board[i][j]=1
                if N_queen(n-1) == True:
                    return True
                board[i][j] = 0
    return False

N_queen(N)
for i in board:
    print(i)
```

```
Enter number of queen:
4
[0, 1, 0, 0]
[0, 0, 0, 1]
[1, 0, 0, 0]
[0, 0, 1, 0]
```