In [1]:

```
global N
N = 4

def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

def isSafe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False

    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    for i, j in zip(range(row, N, 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQUtil(board, col):
    if col >= N:
        return True

    for i in range(N):
        if isSafe(board, i, col):
            board[i][col] = 1

            if solveNQUtil(board, col + 1) == True:
                return True

            board[i][col] = 0

    return False

def solveNQ():
    board = [[0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]]

    if solveNQUtil(board, 0) == False:
        print ("Solution does not exist")
        return False

    printSolution(board)
    return True

solveNQ()
```

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```

In [1]:

```
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]

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

```
220
```

In [3]:

```
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 = Item.fractionalKnapsack(W, arr)
    print(max_val)
```

```
240.0
```

In [22]:

```
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(self, node, val=''):
        newVal = val + str(node.huff)
        if(node.left):
            self.printNodes(self, node.left, newVal)
        if(node.right):
            self.printNodes(self, 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)

node.printNodes(node, nodes[0])
```

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

In [2]:

```
nterms = int(input("Enter number of terms "))
n1, n2 = 0, 1
count = 0

if nterms <= 0:
    print("Please enter a positive integer")

elif nterms == 1:
    print("Fibonacci sequence upto", nterms,":")
    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]:

```
def fibonacci(n):
    if(n <= 1):
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))

n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")

for i in range(n):
    print(fibonacci(i))
```

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