**DAA**

1)Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

#include <iostream>

using namespace std;

int main() {

int n, t1 = 0, t2 = 1, nextTerm = 0;

cout << "Enter the number of terms: ";

cin >> n;

cout << "Fibonacci Series: ";

for (int i = 1; i <= n; ++i) {

// Prints the first two terms.

if(i == 1) {

cout << t1 << ", ";

continue;

}

if(i == 2) {

cout << t2 << ", ";

continue;

}

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

cout << nextTerm << ", ";

}

return 0;

}

#include <iostream>

using namespace std;

int fib(int x) {

if((x==1)||(x==0)) {

return(x);

}else {

return(fib(x-1)+fib(x-2));

}

}

int main() {

int x , i=0;

cout << "Enter the number of terms of series : ";

cin >> x;

cout << "

Fibonnaci Series : ";

while(i < x) {

cout << " " << fib(i);

i++;

}

return 0;

}

def iterative(n):

if(n==1):

return [0]

elif(n==2):

return [0,1]

else:

arr = [0,1]

for i in range(2,n):

arr.append(arr[i-1]+arr[i-2])

return arr

def utility(i):

if i <= 1:

return i

else:

return (utility(i - 1) + utility(i - 2))

def recursive(n):

for i in range(n):

print(utility(i), end=" ")

print(iterative(9))

print(recursive(9))

2)Write a program to implement Huffman Encoding using a greedy strategy.

class node:

def \_\_init\_\_(self, freq, symbol, left=None, right=None):

# frequency of symbol

self.freq = freq

# symbol name (character)

self.symbol = symbol

# node left of current node

self.left = left

# node right of current node

self.right = right

# tree direction (0/1)

self.huff = ''

# utility function to print huffman

# codes for all symbols in the newly

# created Huffman tree

def printNodes(node, val=''):

# huffman code for current node

newVal = val + str(node.huff)

# if node is not an edge node

# then traverse inside it

if(node.left):

printNodes(node.left, newVal)

if(node.right):

printNodes(node.right, newVal)

# if node is edge node then

# display its huffman code

if(not node.left and not node.right):

print(f"{node.symbol} -> {newVal}")

# characters for huffman tree

chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters

freq = [ 5, 9, 12, 13, 16, 45]

# list containing unused nodes

nodes = []

# converting characters and frequencies

# into huffman tree nodes

for x in range(len(chars)):

nodes.append(node(freq[x], chars[x]))

while len(nodes) > 1:

# sort all the nodes in ascending order

# based on their frequency

nodes = sorted(nodes, key=lambda x: x.freq)

# pick 2 smallest nodes

left = nodes[0]

right = nodes[1]

# assign directional value to these nodes

left.huff = 0

right.huff = 1

# combine the 2 smallest nodes to create

# new node as their parent

newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

# remove the 2 nodes and add their

# parent as new node among others

nodes.remove(left)

nodes.remove(right)

nodes.append(newNode)

# Huffman Tree is ready!

print("Huffman Tree Node")

printNodes(nodes[0])

3) Write a program to solve a fractional Knapsack problem using a greedy method

items = [('A',60,10),('B',100,20),('C',120,30),('D',1000,10)]

max\_wt = 50

def fractional\_knapsack(items,max\_wt):

ratios = []

for i in items:

ratios.append((i[0],i[1],i[2],i[1]/i[2]))

ratios.sort(key=lambda tup: tup[3] , reverse = True )

ans = {}

optimal\_profit = 0

for item in ratios:

if(item[2]<max\_wt):

max\_wt -= item[2]

ans[item[0]]=1

optimal\_profit += item[1]

else:

fraction = max\_wt/item[2]

ans[item[0]]=fraction

max\_wt = 0

optimal\_profit += fraction\*item[1]

break

return{"ans":ans, "profit": optimal\_profit}

print(fractional\_knapsack(items,max\_wt))

4) Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

def knapSack(W, wt,val, n):

K = [[0 for x in range(W + 1)] for x in range(n + 1)]

# Build table K[][] in bottom up manner

for i in range(n + 1):

for w in range(W + 1):

if i == 0 or w == 0:

K[i][w] = 0

elif wt[i-1] <= w:

K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w])

else:

K[i][w] = K[i-1][w]

return K[n][W]

# Driver program to test above function

val = [60, 100, 120]

wt = [10, 20, 30]

W = 50

n = len(val)

print(knapSack(W, wt, val, n))