Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No.1

Write a recursive program to generate Fibonacci numbers and count the number of recursive call (steps)

Code:

```
count = 0

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

n = int(input("Enter number of terms: "))
print("Fibonacci Series:", end=" ")

for i in range(n):
    print(fibonacci(i), end=" ")

print("\nTotal number of recursive calls:", count)
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Enter number of terms: 5
Fibonacci Series: 0 1 1 2 3
Total number of recursive calls: 19
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No.2

Write a Program to calculate the Factorial of a number using both recursion and iteration

Code:

```
# Recursive function

def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Iterative function

def factorial_iterative(n):
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact

n = int(input("Enter a number: "))

fact_rec = factorial_recursive(n)
print(f"Factorial of {n} using recursion: {fact_rec}")

fact_itr = factorial_iterative(n)
print(f"Factorial of {n} using iteration: {fact_itr}")
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Enter a number: 5
Factorial of 5 using recursion: 120
Factorial of 5 using iteration: 120
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No.3

Implement job sequencing with Deadlines using the greedy method.

Code:

```
class Job:
    def __init__(self, job_id, deadline, profit):
        self.job_id = job_id
        self.deadline = deadline
        self.profit = profit

def job_sequencing(jobs):
    # Sort jobs in decreasing order of profit
    jobs.sort(key=lambda x: x.profit, reverse=True)

    # Find maximum deadline
    max_deadline = max(job.deadline for job in jobs)

    # Initialize slots and result
    slot = [-1] * (max_deadline + 1)
    total_profit = 0
    job_sequence = []

    # Iterate through all jobs
    for job in jobs:
        # Find a free slot for this job (from its deadline backward)
        for t in range(job.deadline, 0, -1):
            if slot[t] == -1:
                slot[t] = job.job_id
                job_sequence.append(job.job_id)
                total_profit += job.profit
                break

    # Print results
    print("Job Sequence:", job_sequence)
    print("Total Profit:", total_profit)

# Driver Code
n = int(input("Enter number of jobs: "))
jobs = []

for i in range(n):
    job_id = input(f"Enter Job ID {i+1}: ")
    deadline = int(input("Enter deadline: "))
    profit = int(input("Enter profit: "))
    jobs.append(Job(job_id, deadline, profit))

job_sequencing(jobs)
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Output:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_
Enter number of jobs: 4
Enter Job ID 1: J1
Enter deadline: 2
Enter profit: 90
Enter Job ID 2: J2
Enter deadline: 1
Enter profit: 56
Enter Job ID 3: J3
Enter deadline: 2
Enter profit: 67
Enter Job ID 4: J4
Enter deadline: 1
Enter profit: 25
Job Sequence: ['J1', 'J3']
Total Profit: 157
PS C:\Users\DELL\Desktop\data_science>
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No.4

Write a program using the greedy algorithm for fractional knapsack problems.

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


def fractional_knapsack(items, capacity):
    # Calculate value-to-weight ratio for each item
    for item in items:
        item.ratio = item.value / item.weight


    # Sort items by ratio in descending order
    items.sort(key=lambda x: x.ratio, reverse=True)


    total_value = 0.0  # Total profit

    for item in items:
        if capacity >= item.weight:
            capacity -= item.weight
            total_value += item.value
        else:
            total_value += item.value * (capacity / item.weight)
            break

    return total_value


# Driver code

n = int(input("Enter number of items: "))
items = []
for i in range(n):
    value = float(input(f"Enter value of item {i+1}: "))
    weight = float(input(f"Enter weight of item {i+1}: "))
    items.append(Item(value, weight))


capacity = float(input("Enter capacity of knapsack: "))

max_value = fractional_knapsack(items, capacity)
print("\nMaximum value in Knapsack =", round(max_value, 2))
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Enter number of items: 6
Enter value of item 1: 67
Enter weight of item 1: 20
Enter value of item 2: 89
Enter weight of item 2: 60
Enter value of item 3: 90
Enter weight of item 3: 67
Enter value of item 4: 69
Enter weight of item 4: 10
Enter value of item 5: 84
Enter weight of item 5: 40
Enter value of item 6: 100
Enter weight of item 6: 67
Enter capacity of knapsack: 150
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No.5

Construct a solution for the 0/1 knapsack problems using Dynamic Programming

Code:

```
def knapsack(values, weights, capacity):
    n = len(values)
    dp = [[0 for _ in range(capacity + 1)] for _ in range(n + 1)]


    for i in range(1, n + 1):
        for w in range(1, capacity + 1):
            if weights[i - 1] <= w:
                dp[i][w] = max(
                    values[i - 1] + dp[i - 1][w - weights[i - 1]],
                    dp[i - 1][w]
                )
            else:
                dp[i][w] = dp[i - 1][w]

    print("\nDP Table:")
    for row in dp:
        print(row)

    return dp[n][capacity]

values = [60, 100, 120]
weights = [10, 20, 30]
capacity = 50

max_value = knapsack(values, weights, capacity)
print("\nMaximum value that can be obtained =", max_value)
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"

DP Table:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60,
0, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 100, 100, 100, 100, 100, 100, 100,
60, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160, 160]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 100, 100, 100, 100, 100, 100, 100,
60, 160, 160, 160, 180, 180, 180, 180, 180, 180, 180, 180, 180, 180, 220]

Maximum value that can be obtained = 220
PS C:\Users\DELL\Desktop\data_science> []
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No.6

Implementation of Binomial Coefficients using Backtracking

Code:

```
count = 0  # global counter

def backtrack(start, n, k, combination):
    global count
    # If combination length = k, we found a valid subset
    if len(combination) == k:
        count += 1
        return

    # Explore further elements
    for i in range(start, n + 1):
        combination.append(i)      # choose
        backtrack(i + 1, n, k, combination)
        combination.pop()          # un-choose


def binomial_coefficient(n, k):
    global count
    count = 0  # reset counter
    backtrack(1, n, k, [])
    return count


# Driver code
n = int(input("Enter n: "))
k = int(input("Enter k: "))

print("C(n, k) =", binomial_coefficient(n, k))
```

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Enter n: 15
Enter k: 8
C(n, k) = 6435
PS C:\Users\DELL\Desktop\data_science>
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No. 7

Implement the Bellman-Ford algorithm using Dynamic Programming to find the shortest paths from a given source vertex to all other vertices in a weighted, directed graph.

```
def bellman_ford(vertices, edges, source):
    # Step 1: Initialize distances
    dist = [float('inf')] * vertices
    dist[source] = 0

    # Step 2: Relax edges |V| - 1 times
    for _ in range(vertices - 1):
        for u, v, w in edges:
            if dist[u] != float('inf') and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w

    # Step 3: Check for negative-weight cycles
    for u, v, w in edges:
        if dist[u] != float('inf') and dist[u] + w < dist[v]:
            print("Graph contains a negative-weight cycle!")
            return None
 return dist

# Driver code
vertices = 5
edges = [
    (0, 1, -1),
    (0, 2, 4),
    (1, 2, 3),
    (1, 3, 2),
    (1, 4, 2),
    (3, 2, 5),
    (3, 1, 1),
    (4, 3, -3)
]

source = 0
result = bellman_ford(vertices, edges, source)

print("Shortest distances from source vertex", source)
print(result)
```

Output:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Shortest distances from source vertex 0
[0, -1, 2, -2, 1]
PS C:\Users\DELL\Desktop\data_science>
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment No. 8

Write a program to solve the travelling salesman problem using the least cost Branch and Bound method.

Code:

```
import math
import heapq

class Node:
    def __init__(self, level, path, bound):
        self.level = level
        self.path = path
        self.bound = bound

    def __lt__(self, other):
        return self.bound < other.bound


def calculate_bound(matrix, path):
    n = len(matrix)
    bound = 0
    visited = [False] * n

    # Mark visited cities
    for i in path:
        visited[i] = True

    # Add minimum edge cost from each unvisited city
    for i in range(n):
        if not visited[i]:
            min_cost = math.inf
            for j in range(n):
                if i != j:
                    min_cost = min(min_cost, matrix[i][j])
            bound += min_cost

    return bound


def tsp_branch_and_bound(matrix):
    n = len(matrix)
    pq = []

    initial_path = [0]
    initial_bound = calculate_bound(matrix, initial_path)
    root = Node(0, initial_path, initial_bound)

    heapq.heappush(pq, root)
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

```
    best_cost = math.inf
    best_path = []

    while pq:
        current = heapq.heappop(pq)

        if current.bound >= best_cost:
            continue

        if current.level == n - 1:
            total_cost = 0
            for i in range(len(current.path) - 1):
                total_cost += matrix[current.path[i]][current.path[i + 1]]
            total_cost += matrix[current.path[-1]][0]

            if total_cost < best_cost:
                best_cost = total_cost
                best_path = current.path + [0]
            continue

        for next_city in range(n):
            if next_city not in current.path:
                new_path = current.path + [next_city]
                new_bound = calculate_bound(matrix, new_path)

                if new_bound < best_cost:
                    heapq.heappush(
                        pq,
                        Node(current.level + 1, new_path, new_bound)
                    )

    return best_path, best_cost


# Example cost matrix
matrix = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]

path, cost = tsp_branch_and_bound(matrix)
print("Optimal Path:", path)
print("Minimum Cost:", cost)
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Output:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Optimal Path: [0, 2, 3, 1, 0]
Minimum Cost: 80
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment-9

Write a program to calculate the shortest path using Dijkstra's Algorithm.

```
import heapq

def dijkstra(graph, source):
    # Number of vertices
    n = len(graph)
    dist = [float('inf')] * n
    dist[source] = 0

    pq = [(0, source)]  # (distance, vertex)

    while pq:
        current_dist, u = heapq.heappop(pq)

        if current_dist > dist[u]:
            continue

        for v, weight in graph[u]:
            if dist[u] + weight < dist[v]:
                dist[v] = dist[u] + weight
                heapq.heappush(pq, (dist[v], v))

    return dist


graph = [
    [(1, 4), (2, 1)],
    [(3, 1)],
    [(1, 2), (3, 5)],
    []
]
source = 0
result = dijkstra(graph, source)

print("Shortest distances from source vertex", source)
print(result)
```

Output:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DA
Shortest distances from source vertex 0
[0, 3, 1, 4]
PS C:\Users\DELL\Desktop\data_science>
```

Name: Krishna Mansaram Sarovar
Roll No: 08
Date:

Assignment-10

Write a program to implement Kruskal's Algorithm to find the MST of a given weighted graph. Display the total cost and selected edges

```
class DisjointSet:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [0] * n

    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self, x, y):
        rootX = self.find(x)
        rootY = self.find(y)

        if rootX != rootY:
            if self.rank[rootX] < self.rank[rootY]:
                self.parent[rootX] = rootY
            elif self.rank[rootX] > self.rank[rootY]:
                self.parent[rootY] = rootX
            else:
                self.parent[rootY] = rootX
                self.rank[rootX] += 1
            return True

        return False


def kruskal(n, edges):
    edges.sort(key=lambda x: x[2])  # sort by weight
    ds = DisjointSet(n)
    mst = []
    total_cost = 0

    for u, v, w in edges:
        if ds.union(u, v):
            mst.append((u, v, w))
            total_cost += w
```

```
        if len(mst) == n - 1:
            break

    return mst, total_cost



# Example graph
edges = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]

n = 4
mst, cost = kruskal(n, edges)

print("Selected edges in the MST:")
for u, v, w in mst:
    print(f"{u} -- {v} , weight = {w}")

print("Total cost of MST:", cost)
```

Output:

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

```
PS C:\Users\DELL\Desktop\data_science> python -u "c:\Users\DELL\Desktop\data_science\DAA.py"
Selected edges in the MST:
2 -- 3 , weight = 4
0 -- 3 , weight = 5
0 -- 1 , weight = 10
Total cost of MST: 19
```