BCA Semester–V

DisciplineSpecific Course (DSCC-10)

Course Title: Practical inDesign and Analysis of Algorithms
Course Code: 055BCA012

Type of Course	Theory /Practical	Credits	Instruction hour per week	Total No.of Lectures/Hours /Semester	Duration of Exam	Formative Assessment Marks	Summative assessment Marks	Total Marks
DSCC-10	**Practical**	**02**	**04**	**56hrs.**	**3hrs.**	**25**	**25**	**50**

Course Outcomes (COs):Attheend ofthecourse,studentswillbeableto:

CO1: Able to calculate complexity of an algorithm.
CO2: Select appropriate design techniques to solve real world problems.
CO3: Apply the dynamic programming technique to solve the problems.

Program Nos	Programs	56.hrs/ sem
1	Write a program to sort a list of N elements using Selection Sort Technique.	
2	Write a program to perform Travelling Sales man Problem	
3	Write program to implement Dynamic Programming algorithm for the 0/1 Knapsack problem.	
4	Write program to implement the DFS and BFS algorithm for a graph.	
5	Write a program to find minimum and maximum value in an array using divide and conquer.	
6	Write a test program to implement Divide and Conquer Strategy. Eg: Quick sort algorithm for sorting list of integers in ascending order.	
7	Write a program to implement Merge sort algorithm for sorting a list of integers in ascending order.	
8	Write C program that accepts the vertices and edges for a graph and stores it as an adjacency matrix.	
9	ImplementfunctiontoprintIn-Degree,Out-Degreeandtodisplaythatadjacencymatrix	
10	Write a program to perform Knapsack Problem using GreedySolution	
11	Write program to implement backtracking algorithm for solving problems like Nqueens.	
12	Write a program to implement the backtracking algorithm for the sum of subsets problem	
13	Write program to implement greedy algorithm for job sequencing with deadlines.	
14	WriteprogramtoimplementDynamicProgrammingalgorithmfortheOptimalBinary Search Tree Problem.	
15	WriteaprogramthatimplementsPrim'salgorithmtogenerateminimumcostspanningTree.	
16	Write a program that implements Kruskal's algorithm to generate minimum cost spanning tree.	

Program 1:

Write a program to sort a list of N elements using Selection Sort Technique.

```
def selection_sort(arr):
    n = len(arr)
    # Traverse through all elements
    for i in range(n - 1):
    # Assume the current element is the minimum
        min_index = i
        # Find the minimum element in remaining unsorted array
        for j in range(i + 1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        # Swap the found minimum element with the first element
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr
# Driver code
N = int(input("Enter number of elements: "))
elements = []
print("Enter the elements:")
for _ in range(N):
    elements.append(int(input()))
print("Original List:", elements)
sorted_list = selection_sort(elements)
print("Sorted List:", sorted_list)
```

Program 2:

Write a program to perform Travelling Salesman Problem

```
from itertools import permutations

def calculate_distance(route, distances):
    total_distance = 0
    for i in range(len(route) - 1):
        total_distance += distances[route[i]][route[i + 1]]
    # return to the starting city
    total_distance += distances[route[-1]][route[0]]
    return total_distance

def brute_force_tsp(distances, start):
    n = len(distances)
    cities = [i for i in range(n) if i != start]  # exclude the start city
    min_distance = float('inf')
    shortest_route = None
    print("\nAll Possible Routes and Their Distances:\n")
    for perm in permutations(cities):
        current_route = [start] + list(perm) + [start]  # start and end at chosen city
        current_distance = calculate_distance(current_route, distances)
        print(f"Route {current_route} → Distance = {current_distance}")
        if current_distance < min_distance:
            min_distance = current_distance
            shortest_route = current_route
    return shortest_route, min_distance
```

```
# ---------------- MAIN PROGRAM ----------------
# Read number of cities
n = int(input("Enter number of cities: "))
# Read adjacency matrix
print("Enter the distance matrix row by row (use spaces between values):")
distances = []
for i in range(n):
    row = list(map(int, input(f"Row {i+1}: ").split()))
    distances.append(row)
# Read starting node
start = int(input(f"Enter the starting node (0 to {n-1}): "))
# Run TSP brute force
route, total_distance = brute_force_tsp(distances, start)
print("\nShortest Route:", route)
print("Minimum Distance:", total_distance)
```

Program 3:

Write program to implement Dynamic Programming algorithm for the 0/1 Knapsack problem.

```
from itertools import combinations
def knapsack_bruteforce_all_subsets(weights, values, capacity):
    n = len(values)
    max_profit = 0
    best_combination = None
    print(f"\nKnapsack Capacity = {capacity}\n")
    print("All Possible Subsets:")
    # Generate all subsets (including empty set)
    for r in range(0, n+1):
        for subset in combinations(range(n), r):
            total_weight = sum(weights[i] for i in subset)
            total_value = sum(values[i] for i in subset)
            items = [i+1 for i in subset]  # 1-based item numbering
            if total_weight <= capacity:
                status = "Considered"
                if total_value > max_profit:
                    max_profit = total_value
                    best_combination = (items, total_weight, total_value)
            else:
                status = "Not Considered (Exceeds Capacity)"
            print(f'Items: {items}, Weight: {total_weight}, Value: {total_value} --> {status}")
    # Display best solution
    print("\nBest Combination Found:")
```

```
    print(f"Items: {best_combination[0]}, Weight: {best_combination[1]}, Profit: {best_combination[2]}")
    print(f"\nMaximum Profit Achievable = {max_profit} (with Capacity = {capacity})")
    return max_profit


# ---------- MAIN PROGRAM ----------
if __name__ == "__main__":
    n = int(input("Enter number of items: "))
    weights = []
    values = []
     print("\nEnter weights and values for each item:")
    for i in range(n):
        w = int(input(f"Weight of item {i+1}: "))
        v = int(input(f"Value of item {i+1}: "))
        weights.append(w)
        values.append(v)
    capacity = int(input("\nEnter knapsack capacity: "))
    knapsack_bruteforce_all_subsets(weights, values, capacity)
```