

Test: CLA-2 T3

Date: 30/04/2024

Course Code & Title: 21CSC204J Design and Analysis of Algorithms

Duration: 1 hour 40 min

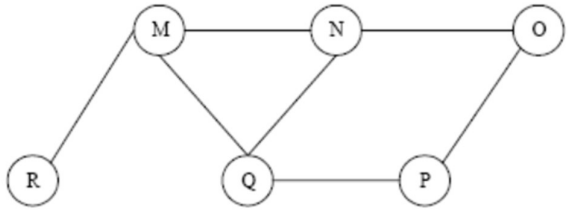
Year & Sem: II Year / IV Sem

Max. Marks: 50

Course Articulation Matrix:

| Course Outcome | PO 1 | PO 2 | PO 3 | PO 4 | PO 5 | PO 6 | PO 7 | PO 8 | PO 9 | PO 10 | PO 11 | PO 12 | Program Specific Outcomes | | |
|----------------|------|------|------|------|------|------|------|------|------|-------|-------|-------|---------------------------|-------|-------|
| | | | | | | | | | | | | | PSO-1 | PSO-2 | PSO-3 |
| CO1 | 2 | 1 | 2 | 1 | - | - | - | - | | 3 | - | 3 | 3 | 1 | - |
| CO2 | 2 | 1 | 2 | 1 | - | - | - | - | | 3 | - | 3 | 3 | 1 | - |
| CO3 | 2 | 1 | 2 | 1 | - | - | - | - | | 3 | - | 3 | 3 | 1 | - |
| CO4 | 2 | 1 | 2 | 1 | - | - | - | - | | 3 | - | 3 | 3 | 1 | - |
| CO5 | 2 | 1 | 2 | 1 | - | - | - | - | | 3 | - | 3 | 3 | 1 | - |

Part – A (8 x 1 = 8 Marks) Instructions: Answer all

| Q. No | Question | Marks | BL | CO | PO | PI Code |
|-------|---|-------|----|----|----|---------|
| 1 | Given two sequences "1232412" and "24312", what is the length of the longest common subsequence between them? a) 3 b) 4 c) 5 d) 6 | 1 | L3 | 3 | 2 | |
| 2 | What is the worst-case time complexity of Prim's algorithm if adjacency matrix is used? a) $(\log V)$ b) $\Omega(V^2)$ c) $O(E^2)$ d) $O(V \log E)$ | 1 | L2 | 3 | 2 | |
| 3 | If the keys of optimal binary search tree are {10, 20, 30}. How many numbers of trees are possible? a) 3 b) 7 c) 5 d) 4 | 1 | L3 | 3 | 2 | |
| 4 | The Breadth First Search algorithm has been implemented using the queue data structure. One possible order of visiting the nodes of the following graph is  | 1 | L3 | 4 | 2 | |

| | | | | | | |
|---|---|---|----|---|---|--|
| | a) MNOPQR b) NQMPOR c) QMNPRO d) QMNPOR | | | | | |
| 5 | Choose the correct statement from the following. a) branch and bound is more efficient than backtracking b) branch and bound is not suitable where a greedy algorithm is not applicable c) branch and bound divides a problem into at least 2 new restricted sub problems d) backtracking divides a problem into at least 2 new restricted sub problems | 1 | L1 | 4 | 2 | |
| 6 | What procedure is being followed in Floyd Warshall Algorithm? a) Top down b) Bottom up c) Neither Top down nor Bottom up d) Either one can be used | 1 | L1 | 4 | 2 | |
| 7 | Which of the following statement is correct? a) P is subset of NP b) NP is subset of P c) P and NP are equal d) NP is subset of NP hard | 1 | L2 | 5 | 2 | |
| 8 | Consider a simple graph G with 18 vertices. What will be the size of the maximum independent set of G, if the size of the minimum vertex cover of G is 10? a) 8 b) 18 c) 28 d) 10 | 1 | L3 | 5 | 2 | |

| Part – B (3 x 5 = 15 Marks) Instructions: Answer all | | | | | | |
|--|--|---|----|---|---|--|
| 9 | <p>Assume we have a weighted undirected disconnected sparse graph with edge weights. Which technique is suitable to compute Minimum Cost spanning Tree and provide the pseudocode for the same.</p> <p>Answer: Krushkal's Algorithm</p> <pre> #include <stdio.h> #include <stdlib.h> #include <string.h> // Structure to represent an edge in the graph struct Edge { int src, dest, weight; }; // Structure to represent a graph struct Graph { int V, E; // Number of vertices and edges struct Edge* edge; // Array of edges }; // Creates a graph with V vertices and E edges struct Graph* createGraph(int V, int E) { struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph)); graph->V = V; graph->E = E; graph->edge = (struct Edge*)malloc(E * sizeof(struct Edge)); return graph; } // A utility function to find the subset of an element i int find(int parent[], int i) { if (parent[i] == -1) return i; return find(parent, parent[i]); } // A utility function to do union of two subsets void Union(int parent[], int x, int y) { int xset = find(parent, x); int yset = find(parent, y); </pre> | 5 | L2 | 3 | 2 | |
| 10 | <p>In a distant galaxy, there are several planets connected by wormholes, forming a communications network for space travelers. Each wormhole allows for instant communication between two planets bidirectionally. However, due to variations in wormhole stability, the time taken to transmit a message through a wormhole can vary.</p> <p>You are tasked with designing a system that can efficiently calculate the minimum time it takes for a message to be transmitted between any two planets in the network. The network is represented as a weighted graph, where each planet</p> | 5 | L3 | 4 | 2 | |

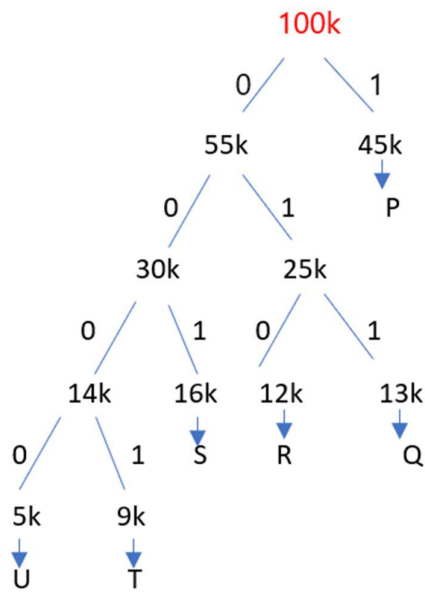
| | | | | | | |
|----|--|---|----|---|---|--|
| | <p>is a vertex and each wormhole is an edge with a weight representing the time taken for transmission. Identify the suitable algorithm to approach this problem to find the minimum transmission time between any two planets in the network. Provide a step-by-step explanation of the identified algorithm and analyze it's time complexity.</p> <p>Answer: Floyd Warshall algorithm</p> <pre> #include <stdio.h> #include <limits.h> #define V 4 // Number of vertices in the graph #define INF INT_MAX // Infinity void floydWarshall(int graph[V][V]) { int dist[V][V]; int i, j, k; // Initialize the distance matrix for (i = 0; i < V; i++) for (j = 0; j < V; j++) dist[i][j] = graph[i][j]; // Calculate shortest paths for (k = 0; k < V; k++) { // Pick all vertices as source one by one for (i = 0; i < V; i++) { // Pick all vertices as destination for the above picked source for (j = 0; j < V; j++) { // If vertex k is on the shortest path from i to j, // then update the value of dist[i][j] if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) dist[i][j] = dist[i][k] + dist[k][j]; } } } // Print the shortest distance matrix for (i = 0; i < V; i++) { for (j = 0; j < V; j++) { if (dist[i][j] == INF) printf("INF "); else printf("%d ", dist[i][j]); } printf("\n"); } } int main() { int graph[V][V] = { {0, 5, INF, 10}, {INF, 0, 3, INF}, {INF, INF, 0, 1}, } </pre> | | | | | |
| 11 | You are allocated with the task of sorting a large dataset efficiently. Your decided to use the improvised version of | 5 | L3 | 5 | 2 | |

| | | | | | | |
|--|--|--|--|--|--|--|
| | <p>quicksort algorithm to maintain its average-case time complexity as $O(n \log n)$. However, you need to ensure that the algorithm performs well even on worst-case inputs. Explain how you would modify the quicksort algorithm to handle worst-case scenarios more effectively. Brief the pseudocode steps you would take to implement and test this modification, ensuring that the algorithm remains efficient for large datasets</p> <p>Answer : Randomized Quick sort</p> <pre> #include <stdio.h> #include <stdlib.h> #include <time.h> // Function to swap two elements void swap(int* a, int* b) { int temp = *a; *a = *b; *b = temp; } // Function to partition the array using a random pivot int partition(int arr[], int low, int high) { // Generate a random index between low and high srand(time(NULL)); int random = low + rand() % (high - low + 1); // Swap the random element with the last element to use it as pivot swap(&arr[random], &arr[high]); int pivot = arr[high]; // Pivot int i = (low - 1); // Index of smaller element for (int j = low; j <= high - 1; j++) { // If current element is smaller than the pivot if (arr[j] < pivot) { i++; // Increment index of smaller element swap(&arr[i], &arr[j]); } } swap(&arr[i + 1], &arr[high]); return (i + 1); } // Function to implement quicksort void quicksort(int arr[], int low, int high) { if (low < high) { // Partition the array int pi = partition(arr, low, high); // Sort the subarrays quicksort(arr, low, pi - 1); quicksort(arr, pi + 1, high); } } // Function to print an array void printArray(int arr[], int size) { for (int i = 0; i < size; i++) </pre> | | | | | |
|--|--|--|--|--|--|--|

| | | | | | |
|--|--|--|--|--|--|
| <pre> printf("%d ", arr[i]); printf("\n"); } // Main function int main() { int arr[] = {10, 7, 8, 9, 1, 5}; int n = sizeof(arr) / sizeof(arr[0]); printf("Original array: \n"); printArray(arr, n); quicksort(arr, 0, n - 1); printf("Sorted array: \n"); printArray(arr, n); return 0; } </pre> | | | | | |
|--|--|--|--|--|--|

Part – C (3 x 9 = 27 Marks)

| | | | | | |
|--|---|----|---|---|--|
| <p>12. Suppose you are a software engineer working for a digital library that needs to efficiently store and transmit text documents. One of the key challenges is to minimize the amount of storage space required while ensuring fast access to the documents. To address this challenge, you are decided to implement a greedy algorithm. To start with Your client has provided a large document containing the minutes of important meetings, which needs to be stored efficiently. The document consists of 100k characters, with varying frequencies say 'P': 45k, 'Q': 13k, 'R': 12k, 'S': 16k, 'T': 9k, 'U': 5k times of each character. Develop a suitable algorithm to find a solution to reduce storage capacity and analysis its time complexity</p> <p>A</p> <p>Solution : 5 marks Time complexity : 1 mark</p> | 9 | L3 | 3 | 2 | |
|--|---|----|---|---|--|



Solution:

| ITEM | FREQUENCY | Huffman code |
|------|-----------|--------------|
| P | 45k | 1 |
| Q | 13k | 011 |
| R | 12k | 010 |
| S | 16k | 001 |
| T | 9k | 0001 |
| U | 5k | 0000 |

Time Complexity:

$O(n \log n)$

ALGORITHM: 4 marks

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_TREE_HT 100

// A Huffman tree node
struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
};

// A Min Heap
struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

// Function to create a new Min Heap node
struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp
        = (struct MinHeapNode*)malloc(sizeof(struct
MinHeapNode));

    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;

    return temp;
}

// Function to create a min heap
struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap
        = (struct MinHeap*)malloc(sizeof(struct MinHeap));

    minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array
        = (struct MinHeapNode**)malloc(minHeap->capacity
        * sizeof(struct MinHeapNode));
    return minHeap;
}
```

| | | | | | | |
|--------------|---|----------|-----------|----------|----------|--|
| | } | | | | | |
| (or) | | | | | | |
| 12. B | <p>Assume that you are a space explorer on a mission to a distant planet to collect valuable mineral samples. Your spaceship has limited cargo space, and you can only carry a maximum weight of 15 units and sample items cannot be broken into pieces. You have identified four mineral samples with the following weights and values:</p> <p>(i)Mineral sample 1: Weight 2 units, Value 3 (ii)Mineral sample 2: Weight 5 units, Value 7 (iii)Mineral sample 3: Weight 8 units, Value 10 (iv)Mineral sample 4: Weight 1 unit, Value 2</p> <p>You want to Develop a suitable dynamic programming algorithm which will maximize the total value of the samples you can bring back to Earth. Do determine the maximum total value of mineral samples you can carry back and specify the complexity of your developed algorithm.</p> <p>Answer: 0/1 knapsack Alorithm: 4 marks</p> <pre> int max (int a, int b) { return (a > b)? a : b; } int knapsack_01(int W, int weights[], int values[], int n) { int i, w; int dp[n+1][W+1]; for (i = 0; i <= n; i++) { for (w = 0; w <= W; w++) { if (i == 0 w == 0) dp[i][w] = 0; else 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]; } } return dp[n][W]; } int main() { int W = 15; int weights[] = {2, 5, 8, 1}; int values[] = {3, 7, 10, 2}; int n = sizeof(values) / sizeof(values[0]); int total_value = knapsack_01(W, weights, values, n); printf("Maximum total value: %d\n", total_value); return 0; } </pre> <p>Solution: 5 marks</p> | 9 | L3 | 3 | 2 | |

| values | weight | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|--------|--------|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 2 | 0 | 2 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 7 | 5 | 0 | 2 | 3 | 5 | 5 | 7 | 9 | 10 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| 10 | 8 | 0 | 2 | 3 | 5 | 5 | 7 | 9 | 10 | 12 | 12 | 13 | 15 | 15 | 15 | 15 | 20 |

The maximum profit is 20 and the items included are

- Mineral sample 1: Weight 2 units, Value 3
- Mineral sample 2: Weight 5 units, Value 7
- Mineral sample 3: Weight 8 units, Value 10

Time complexity:

The time complexity of the 0/1 knapsack problem solved using dynamic programming is $O(nW)$, where n is the number of items and W is the capacity of the knapsack

13. A In the kingdom of Chessland, the royal gardener is tasked with designing a new garden for the princess's palace. The garden is in the shape of a $M \times M$ square, where each square represents a potential location for a flower bed. However, due to the royal decree, no two flowers should be planted in the same row, column, or diagonal. Your task as the royal gardener's assistant is to determine how many different ways there are to plant M flowers in the garden such that no two flowers are in the same row, column, or diagonal. Write an algorithm to calculate the number of valid arrangements for $M = 8$, and provide the solution with its algorithm using backtracking designing strategy and analysis its time complexity.

Answer:

Solution Algorithm: 5 marks

```
#include <stdio.h>
```

```
#include <stdbool.h>
```

```
#define N 8
```

```
void printSolution(int board[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++)
            printf("%2d ", board[i][j]);
        printf("\n");
    }
}
```

```
bool isSafe(int board[N][N], int row, int col) {
    int i, j;

    for (i = 0; i < col; i++)
        if (board[row][i])
            return false;
```

9

L3

4

2

| | | | | | | |
|--------------|---|----------|-----------|----------|----------|--|
| | <pre> for (i = row, j = col; i >= 0 && j >= 0; i--, j--) if (board[i][j]) return false; for (i = row, j = col; j >= 0 && i < N; i++, j--) if (board[i][j]) return false; return true; } bool solveNQUtil(int board[N][N], int col) { if (col >= N) return true; for (int i = 0; i < N; i++) { if (isSafe(board, i, col)) { board[i][col] = 1; if (solveNQUtil(board, col + 1)) return true; board[i][col] = 0; } } return false; } bool solveNQ() { int board[N][N] = { {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, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0} }; if (solveNQUtil(board, 0) == false) { printf("Solution does not exist"); return false; } printSolution(board); return true; } int main() { solveNQ(); return 0; } </pre> <p>Ans: 4 marks {1, 5, 8, 6, 3, 7, 2, 4}</p> <p>Time Complexity: 1 mark O(M!)</p> | | | | | |
| (or) | | | | | | |
| 13. B | Assume that you are a delivery manager overseeing a team of drivers delivering packages to various cities given in the following table. The goal is to find the most efficient route that | 9 | L3 | 4 | 2 | |

visits each city exactly once and returns to the starting city. Design an algorithm with branch and bound approach to find the shortest route that allows you to deliver packages to all cities exactly once and return to the City Chennai. Your algorithm should output the sequence of cities to visit and the total distance travelled. And also depict the efficiency of the designed algorithm.

| From \ To | Chennai | Delhi | Hyderabad | Pune | Bangalore |
|-----------|---------|-------|-----------|------|-----------|
| Chennai | - | 20 | 30 | 10 | 11 |
| Delhi | 15 | - | 16 | 4 | 2 |
| Hyderabad | 3 | 5 | - | 2 | 4 |
| Pune | 19 | 6 | 18 | - | 3 |
| Bangalore | 16 | 4 | 7 | 16 | - |

Solution: 5 marks

| | 1 | 2 | 3 | 4 | 5 |
|---|----|----|----|----|---|
| 1 | ∞ | 10 | 17 | 0 | 1 |
| 2 | 12 | ∞ | 11 | 2 | 0 |
| 3 | 0 | 3 | ∞ | 0 | 2 |
| 4 | 15 | 3 | 12 | ∞ | 0 |
| 5 | 11 | 0 | 0 | 12 | ∞ |

→

| | 1 | 2 | 3 | 4 | 5 |
|---|----|----|----|----|---|
| 1 | ∞ | 10 | 17 | 0 | 1 |
| 2 | 12 | ∞ | 11 | 2 | 0 |
| 3 | 0 | 3 | ∞ | 0 | 2 |
| 4 | 15 | 3 | 12 | ∞ | 0 |
| 5 | 11 | 0 | 0 | 12 | ∞ |

reduced cost = 25

| | 1 | 2 | 3 | 4 | 5 |
|---|----|---|----|----|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | ∞ | ∞ | 11 | 2 | 0 |
| 3 | 0 | ∞ | ∞ | 0 | 2 |
| 4 | 15 | ∞ | 12 | ∞ | 0 |
| 5 | 11 | ∞ | 0 | 12 | ∞ |

②

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|----|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 1 | ∞ | ∞ | 2 | 0 |
| 3 | ∞ | 3 | ∞ | 0 | 2 |
| 4 | 4 | 3 | ∞ | ∞ | 0 |
| 5 | 0 | 0 | ∞ | 12 | 0 |

③

| | 1 | 2 | 3 | 4 | 5 |
|---|----|---|----|---|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 12 | ∞ | 11 | ∞ | 0 |
| 3 | 0 | 3 | ∞ | ∞ | 2 |
| 4 | ∞ | 3 | 12 | ∞ | 0 |
| 5 | 11 | 0 | 0 | ∞ | ∞ |

④

| | 1 | 2 | 3 | 4 | 5 |
|---|----|---|---|----|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 10 | ∞ | 9 | 0 | ∞ |
| 3 | 0 | 3 | ∞ | 0 | ∞ |
| 4 | 12 | 0 | 9 | ∞ | ∞ |
| 5 | ∞ | 0 | 0 | 12 | ∞ |

⑤

| | 1 | 2 | 3 | 4 | 5 |
|---|----|---|----|---|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | ∞ | ∞ | 11 | ∞ | 0 |
| 3 | 0 | ∞ | ∞ | ∞ | 2 |
| 4 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 5 | 11 | ∞ | 0 | ∞ | ∞ |

⑥

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 1 | ∞ | ∞ | ∞ | 0 |
| 3 | ∞ | 1 | ∞ | ∞ | 0 |
| 4 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 5 | 0 | 0 | ∞ | ∞ | ∞ |

⑦

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 2 | 1 | ∞ | 0 | ∞ | ∞ |
| 3 | 0 | 3 | ∞ | ∞ | ∞ |
| 4 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 5 | ∞ | 0 | 0 | ∞ | ∞ |

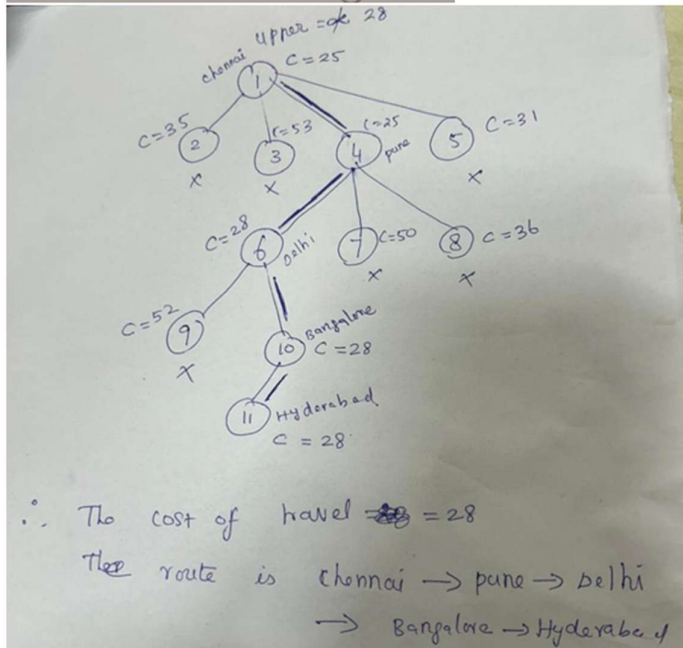
⑧

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 1 | ∞ | ∞ | ∞ | ∞ |
| 2 | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | ∞ | ∞ | 0 |
| 4 | ∞ | ∞ | ∞ | ∞ |
| 5 | 0 | ∞ | ∞ | ∞ |

(9)

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 1 | ∞ | ∞ | ∞ | ∞ |
| 2 | ∞ | ∞ | ∞ | ∞ |
| 3 | 0 | ∞ | ∞ | ∞ |
| 4 | ∞ | ∞ | ∞ | ∞ |
| 5 | ∞ | 0 | ∞ | ∞ |

(10)



Algorithm: 4 marks

```
#include <bits/stdc++.h>
using namespace std;
const int N = 4;
int final_path[N+1];
bool visited[N];
int final_res = INT_MAX;
void copyToFinal(int curr_path[])
{
    for (int i=0; i<N; i++)
        final_path[i] = curr_path[i];
    final_path[N] = curr_path[0];
}
int firstMin(int adj[N][N], int i)
{
    int min = INT_MAX;
    for (int k=0; k<N; k++)
        if (adj[i][k]<min && i != k)
            min = adj[i][k];
    return min;
}
int secondMin(int adj[N][N], int i)
{
    int first = INT_MAX, second = INT_MAX;
    for (int j=0; j<N; j++)
    {
        if (i == j)
            continue;

        if (adj[i][j] <= first)
```

| | | | | | | |
|--|--|--|--|--|--|--|
| | <pre> { second = first; first = adj[i][j]; } else if (adj[i][j] <= second && adj[i][j] != first) second = adj[i][j]; } return second; } void TSPRec(int adj[N][N], int curr_bound, int curr_weight, int level, int curr_path[]) { if (level==N) { if (adj[curr_path[level-1]][curr_path[0]] != 0) { int curr_res = curr_weight + adj[curr_path[level- 1]][curr_path[0]]; if (curr_res < final_res) { copyToFinal(curr_path); final_res = curr_res; } } return; } for (int i=0; i<N; i++) { if (adj[curr_path[level-1]][i] != 0 && visited[i] == false) { int temp = curr_bound; curr_weight += adj[curr_path[level- 1]][i]; if (level==1) curr_bound -= ((firstMin(adj, curr_path[level-1]) + firstMin(adj, i))/2); else curr_bound -= ((secondMin(adj, curr_path[level-1]) + firstMin(adj, i))/2); if (curr_bound + curr_weight < final_res) { </pre> | | | | | |
|--|--|--|--|--|--|--|

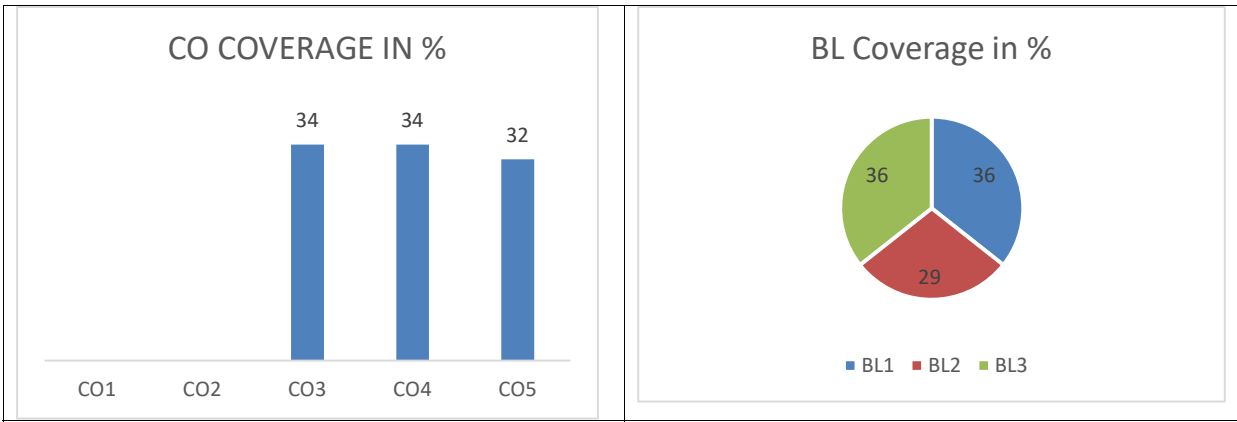
| | | | | | | |
|--|---|--|--|--|--|--|
| | <pre> curr_path[level] = i; visited[i] = true; TSPRec(adj, curr_bound, curr_weight, level+1, curr_path); } curr_weight -= adj[curr_path[level-1]][i]; curr_bound = temp; memset(visited, false, sizeof(visited)); for (int j=0; j<=level-1; j++) visited[curr_path[j]] = true; } } void TSP(int adj[N][N]) { int curr_path[N+1]; int curr_bound = 0; memset(curr_path, -1, sizeof(curr_path)); memset(visited, 0, sizeof(curr_path)); for (int i=0; i<N; i++) curr_bound += (firstMin(adj, i) + secondMin(adj, i)); curr_bound = (curr_bound&1)? curr_bound/2 + 1 : curr_bound/2; visited[0] = true; curr_path[0] = 0; TSPRec(adj, curr_bound, 0, 1, curr_path); } int main() { //Adjacency matrix for the given graph int adj[N][N] = { { }, { }, { }, { } }; TSP(adj); printf("Minimum cost : %d\n", final_res); printf("Path Taken : "); for (int i=0; i<=N; i++) printf("%d ", final_path[i]); </pre> | | | | | |
|--|---|--|--|--|--|--|

| | | | | | | |
|----------|--|---|----|---|---|--|
| | <pre> return 0; } </pre> <p>Time Complexity: $O(n!)$ (1 mark)</p> | | | | | |
| 14. A | <p>You are developing a document management system for a law firm that handles a large number of legal documents. One of the key features of the system is to quickly find similar documents based on a user-provided query document. The similarity between documents is determined by the presence of similar paragraphs or sections of text. How would you use the randomized algorithm to efficiently find similar documents in the database based on a user-provided query document? Consider the user query document is having the pattern "AABA" and the legal document database is having "ADBAACAADAABAAABAA". Does there exists a document similarity? If so, Specify the time complexity of the developed randomized algorithm.</p> <p>Answer:</p> <p>Algorithm: (5 marks)</p> <pre> void search(char *pat, char *txt) { int M = strlen(pat); int N = strlen(txt); int i, j; int p = 0; // Hash value for pattern int t = 0; // Hash value for txt int h = 1; srand(time(NULL)); int prime = rand() % 100 + 100; for (i = 0; i < M - 1; i++) h = (h * d) % prime; for (i = 0; i < M; i++) { p = (d * p + pat[i]) % prime; t = (d * t + txt[i]) % prime; } for (i = 0; i <= N - M; i++) { if (p == t) { for (j = 0; j < M; j++) { if (txt[i + j] != pat[j]) break; } if (j == M) printf("Pattern found at index %d\n", i); } if (i < N - M) { t = (d * (t - txt[i] * h) + txt[i + M]) % prime; if (t < 0) t = (t + prime); } } } </pre> | 9 | L3 | 5 | 2 | |

| | | | | | | |
|--------------|--|----------|-----------|----------|----------|--|
| | <pre> } } } int main() { char txt[] = " "; char pat[] = " "; search(pat, txt); return 0; } </pre> <p>Ans: Pattern match is there (4 marks)</p> <p>Time Complexity: $O(n+m)$ where n is the length of the text and m is the length of the pattern. (1 mark)</p> | | | | | |
| (or) | | | | | | |
| 14. B | <p>Explain about NP-Hard and NP-Complete.</p> <p>Answer</p> <p>NP-Hard and NP-Complete are two classes of problems in computational complexity theory, particularly in the context of decision problems (problems with yes/no answers).</p> <p>NP-Hard: A problem is NP-Hard if every problem in the NP (Nondeterministic Polynomial time) complexity class can be reduced to it in polynomial time. This means that if we had a polynomial-time algorithm for solving any NP-Hard problem, we could use it to solve all problems in NP in polynomial time. However, NP-Hard problems themselves may not be in NP and may not have efficient solutions.</p> <p>NP-Complete: A problem is NP-Complete if it is both NP and NP-Hard. This means that an NP-Complete problem is at least as hard as the hardest problems in NP, and any problem in NP can be reduced to it in polynomial time. NP-Complete problems are considered to be among the hardest problems in NP.</p> <p>The concept of NP-Complete problems is particularly important because if any one of them can be solved in polynomial time, then all problems in NP can be solved in polynomial time. This is known as the "P = NP" question, which remains one of the most famous unsolved problems in computer science.</p> <p>Examples of NP-Complete problems include the Traveling Salesman Problem, the Boolean Satisfiability Problem (SAT), and the Knapsack Problem.</p> | 9 | L3 | 5 | 2 | |

*Program Indicators are available separately for Computer Science and Engineering in AICTE examination reforms policy.

Course Outcome (CO) and Bloom's level (BL) Coverage in Questions



Approved by the Audit Professor/Course Coordinator