**Roll No…………….. Total No. of Pages:……**

**ST-4 (SET-IV)**

**6th SEMESTER 2023-24**

**CS192- Advanced Data Structures**

**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1) What is the time complexity of Bottom-Up DP for 1-Dimensional problems with n states?

1. O(1)
2. O(log n)
3. **O(n)**
4. O(n^2)

2) In grid DP, what is typically represented along the horizontal axis of the DP table?

1. Subproblem size
2. Subproblem index
3. Grid row
4. **Grid column**

3) In dynamic programming for the 0/1 Knapsack problem, what does the state "dp[i][w]" represent?

1. **The maximum value achievable by selecting items from the first "i" items with a maximum weight limit of "w."**
2. The total weight of items selected from the first "i" items.
3. The number of items selected from the first "i" items.
4. The maximum weight limit achievable by selecting items from the first "i" items.

4) What does the "Optimal Substructure" property mean in dynamic programming?

1. It describes problems that have a single solution.
2. **It states that the global optimal solution can be constructed from optimal solutions of subproblems.**
3. It indicates problems with no overlapping subproblems.
4. It refers to the order in which subproblems are solved.

5) In multidimensional dynamic programming, what is the role of the transition table?

1. **Storing subproblem solutions**
2. Defining the problem statement
3. Creating recursive functions
4. Identifying base cases

6) What is the primary disadvantage of using a recursive (top-down) approach in multidimensional dynamic programming?

1. It is more challenging to implement.
2. **It may result in exponential time complexity.**
3. It requires less memory.
4. It is less accurate.

7) Which of the following problems can be efficiently solved using Dynamic Programming on Trees?

1. Determining if a graph is connected
2. Finding the shortest path between two nodes in a graph
3. Finding the minimum spanning tree of a graph
4. **Calculating the diameter of a tree**

8) Which of the following problems is NOT solved using a greedy algorithm?

1. Fractional Knapsack Problem
2. Huffman coding
3. Shortest Path Problem
4. **Sorting an array**

9) How do you check if the rightmost (least significant) bit is set to 1 in an integer x?

1. **(x & 1) == 1**
2. (x & 1) == 0
3. (x | 1) == 1
4. (x ^ 1) == 1

10) Which theorem is used to calculate modular inverses when 'p' is a prime number?

1. Euler's Totient Theorem
2. **Fermat's Little Theorem**
3. Chinese Remainder Theorem
4. Extended Euclidean Theorem

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

11) What is the output of the following C++ code?

#include <iostream>

using namespace std;

int main() {

int dp[7] = {0};

dp[1] = 1;

for (int i = 2; i <= 6; i++) {

dp[i] = i;

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

dp[i] = min(dp[i], dp[j] + dp[i - j]);

}

}

cout << "Minimum number of coins needed: " << dp[6] << endl;

return 0;

}

a) Minimum number of coins needed: 1

b) Minimum number of coins needed: 2

c) Minimum number of coins needed: 3

**d) Minimum number of coins needed: 6**

12) The subset-sum problem is defined as follows. Given a set of n positive integers, S = {a1 ,a2 ,a3 ,…,an} and positive integer W, is there a subset of S whose elements sum to W? A dynamic program for solving this problem uses a 2-dimensional Boolean array X, with n rows and W+1 columns. X[i, j],1 <= i <= n, 0 <= j <= W, is TRUE if and only if there is a subset of {a1 ,a2 ,...,ai} whose elements sum to j. Which of the following is valid for 2 <= i <= n and ai <= j <= W?

a) X[i, j] = X[i - 1, j] ∨ X[i, j -ai]

**b) X[i, j] = X[i - 1, j] ∨ X[i - 1, j - ai]**

c) X[i, j] = X[i - 1, j] ∧ X[i, j - ai]

d) X[i, j] = X[i - 1, j] ∧ X[i -1, j - ai]

13) Consider a sequence F00 defined as : F00(0) = 1, F00(1) = 1 F00(n) = 10 ∗ F00(n – 1) + 100 F00(n – 2) for n ≥ 2 Then what shall be the set of values of the sequence F00 ?

**a) (1, 110, 1200)**

b) (1, 110, 600, 1200)

c) (1, 2, 55, 110, 600, 1200)

d) (1, 55, 110, 600, 1200)

14) Consider a graph G=(V, E), where V = { v1,v2,…,v100 }, E={ (vi, vj) ∣ 1≤ i < j ≤ 100} and weight of the edge (vi, vj) is ∣i–j∣. The weight of minimum spanning tree of G is \_\_\_\_\_\_\_\_. Note - This question was Numerical Type.

**a) 99**

b) 100

c) 98

d) 101

15) What does the following C expression do?

x = x & (x-1)

a) Sets all bits as 1

b) Makes x equals to 0

**c) Turns of the rightmost set bit**

d) Turns of the leftmost set bit

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Given the arrival and departure times of all trains that reach a railway station, the task is to find the minimum number of platforms required for the railway station so that no train waits. We are given two arrays that represent the arrival and departure times of trains that stop.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | arr[] = {900, 940, 950, 1100, 1500, 1800},  dep[] = {910, 1200, 1120, 1130, 1900, 2000} | arr[] = {900, 940},  dep[] = {910, 1200} | arr[] = { 100, 300, 600 }  dep[] = { 900, 400, 500 } |
| **Output** | 3 | 1 | 2 |

Solution :

**// C++ program to implement the above approach**

**#include <bits/stdc++.h>**

**using namespace std;**

**// Function to find the minimum number of platforms required**

**int findPlatformOptimized(int arr[], int dep[], int n)**

**{**

**int count = 0, maxPlatforms = 0;**

**// Find the maximum departure time**

**int maxDepartureTime = dep[0];**

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

**maxDepartureTime = max(maxDepartureTime, dep[i]);**

**}**

**// Create a vector to store the count of trains at each time**

**vector<int> v(maxDepartureTime + 2, 0);**

**// Increment the count at the arrival time and decrement at the departure time**

**for (int i = 0; i < n; i++) {**

**v[arr[i]]++;**

**v[dep[i] + 1]--;**

**}**

**// Iterate over the vector and keep track of the maximum sum seen so far**

**for (int i = 0; i <= maxDepartureTime + 1; i++) {**

**count += v[i];**

**maxPlatforms = max(maxPlatforms, count);**

**}**

**return maxPlatforms;**

**}**

**int main()**

**{**

**int arr[] = { 100, 300, 600 };**

**int dep[] = { 900, 400, 500 };**

**int n = sizeof(arr) / sizeof(arr[0]);**

**cout << findPlatformOptimized(arr, dep, n);**

**return 0;**

**}**

Q17) There are ‘p’ balls of type P, ‘q’ balls of type Q and ‘r’ balls of type R. Using the balls we want to create a straight line such that no two balls of same type are adjacent.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | p = 1, q = 1, r = 0 | p = 1, q = 1, r = 1 | p = 2, q = 1, r = 1 |
| **Output** | 2 | 6 | 12 |

Solution :

**#include <bits/stdc++.h>**

**using namespace std;**

**#define MAX 100**

**// Function to count the number of arrangements**

**int countUtil(int p, int q, int r)**

**{**

**int dp[MAX][MAX][MAX];**

**memset(dp, 0, sizeof(dp)); // Initializing the DP table with zeros**

**// Base cases**

**dp[1][0][0] = 1; // If only one 'p' is present**

**dp[0][1][0] = 1; // If only one 'q' is present**

**dp[0][0][1] = 1; // If only one 'r' is present**

**// Fill the DP table**

**for (int i = 0; i <= p; i++)**

**{**

**for (int j = 0; j <= q; j++)**

**{**

**for (int k = 0; k <= r; k++)**

**{**

**// Skip the base cases as they are already initialized**

**if (i == 1 && j == 0 && k == 0)**

**continue;**

**if (i == 0 && j == 1 && k == 0)**

**continue;**

**if (i == 0 && j == 0 && k == 1)**

**continue;**

**// Update DP values based on previous states**

**if (i - 1 >= 0)**

**dp[i][j][k] += dp[i - 1][j][k]; // Add the count when using 'p'**

**if (j - 1 >= 0)**

**dp[i][j][k] += dp[i][j - 1][k]; // Add the count when using 'q'**

**if (k - 1 >= 0)**

**dp[i][j][k] += dp[i][j][k - 1]; // Add the count when using 'r'**

**}**

**}**

**}**

**return dp[p][q][r]; // Return the count of required arrangements**

**}**

**int main()**

**{**

**int p = 2, q = 1, r = 1;**

**cout<<countUtil(p, q, r);**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) Given a cost matrix cost[][] and a position (M, N) in cost[][], write a function that returns cost of minimum cost path to reach (M, N) from (0, 0). Each cell of the matrix represents a cost to traverse through that cell. The total cost of a path to reach (M, N) is the sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j), cells (i+1, j), (i, j+1), and (i+1, j+1) can be traversed.

Note: You may assume that all costs are positive integers.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 1 2 3  4 8 2  1 5 3  m=2 n=2 | 1 2 3  4 8 2  1 5 3  m=1 n=2 | 2 2 3  9 8 2  7 5 3  m=1 n=3 |
| **Output** | 8 | 5 | 13 |

Solution :

**// A Dynamic Programming based solution for MCP problem**

**#include <bits/stdc++.h>**

**using namespace std;**

**#define R 3**

**#define C 3**

**int min(int x, int y, int z);**

**// Returns cost of minimum cost path from (0,0) to (m, n) in mat[R][C]**

**int minCostMemoized(int cost[R][C], int m, int n,**

**int memo[R][C])**

**{**

**if (n < 0 || m < 0)**

**return INT\_MAX;**

**else if (m == 0 && n == 0)**

**return cost[m][n];**

**if (memo[m][n] != -1)**

**return memo[m][n];**

**memo[m][n]**

**= cost[m][n]**

**+ min(minCostMemoized(cost, m - 1, n - 1, memo),**

**minCostMemoized(cost, m - 1, n, memo),**

**minCostMemoized(cost, m, n - 1, memo));**

**return memo[m][n];**

**}**

**// Returns cost of minimum cost path from (0,0) to (m, n) in mat[R][C]**

**int minCost(int cost[R][C], int m, int n)**

**{**

**int memo[R][C];**

**memset(memo, -1,**

**sizeof(memo)); // Initialize memo table with -1**

**return minCostMemoized(cost, m, n, memo);**

**}**

**// A utility function that returns minimum of 3 integers**

**int min(int x, int y, int z)**

**{**

**if (x < y)**

**return (x < z) ? x : z;**

**else**

**return (y < z) ? y : z;**

**}**

**int main()**

**{**

**int cost[R][C]**

**= { { 1, 2, 3 }, { 4, 8, 2 }, { 1, 5, 3 } };**

**cout << minCost(cost, 2, 2) << endl;**

**return 0;**

**}**