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

**ST-4 (SET-VI)**

**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. In 1-Dimensional DP, what is the primary use of memoization?
   1. Reducing the time complexity of the problem
   2. Keeping track of the number of recursive calls
   3. **Storing the results of subproblems to avoid redundant computations**
   4. Converting recursive solutions to iterative solutions
2. Which of the following problems can be solved using grid DP?
   1. Finding the shortest path in a graph
   2. Calculating the Fibonacci sequence
   3. **Counting the number of ways to reach a target in a grid**
   4. Sorting an array
3. What is the space complexity of solving the Knapsack problem using dynamic programming?
   1. O(n)
   2. O(n log n)
   3. O(n^2)
   4. **O(nW), where W is the knapsack capacity**
4. Dynamic programming can be applied to problems with which type of time complexity?
   1. O(1)
   2. O(n)
   3. O(log n)
   4. **O(n^2)**
5. When solving multidimensional dynamic programming problems, what is the importance of identifying overlapping subproblems?
   1. **It helps reduce time complexity.**
   2. It ensures that the problem is solved bottom-up.
   3. It simplifies the problem statement.
   4. It indicates that the problem cannot be solved with dynamic programming.
6. In Dynamic Programming on Trees, what is the primary role of the "memo" table or data structure?
   1. **To store intermediate results and avoid redundant calculations**
   2. To represent the tree's structure
   3. To store the values of all tree nodes
   4. To keep track of the root node
7. Which greedy algorithm is used to find the shortest path in a weighted graph?
   1. **Dijkstra's algorithm**
   2. Prim's algorithm
   3. Kruskal's algorithm
   4. Bellman-Ford algorithm
8. What is the result of 12 | 9 in binary?
   1. **13**
   2. 8
   3. 12
   4. 9
9. What is the primary use of Fermat's little theorem?
   1. Finding prime numbers
   2. Solving Diophantine equations
   3. **Calculating modular inverses**
   4. Calculating Euler's totient function
10. Which C++ library is commonly used for handling big integers?
    1. <math.h>
    2. <cstdlib>
    3. <iomanip>
    4. **<boost/multiprecision/cpp\_int.hpp>**

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

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

11) What is the main goal of the Word Break Problem?

a) To break a given string into individual characters.

**b) To determine if a string can be segmented into a sequence of words from a dictionary.**

c) To find the longest subsequence of a string.

d) To reverse a given string.

12) Which of the following problems is closely related to the Subsets Sum problem and can also be solved using dynamic programming?

a) Longest Common Subsequence Problem

b) Longest Palindromic Subsequence Problem

**c) Coin Change Problem**

d) Optimal Binary Search Tree Problem

13) Consider a job scheduling problem with 4 jobs J1, J2, J3, J4 and with corresponding deadlines: ( d1, d2, d3, d4) = (4, 2, 4, 2). Which of the following is not a feasible schedule without violating any job schedule?

a) J2, J4, J1, J3

**b) J4, J1, J2, J3**

c) J4, J2, J1, J3

d) J4, J2, J3, J1

14) What does the following C expression do? x = (x<<1) + x + (x>>1);

a) Multiplies an integer with 7

**b) Multiplies an integer with 3.5**

c) Multiplies an integer with 3

d) Multiplies an integer with 8

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

#include <iostream>

using namespace std;

int main() {

int n = 5;

int dp[6] = {0};

dp[0] = 1;

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

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

dp[j] += dp[j - i];

}

}

cout << "Number of ways to make change: " << dp[n] << endl;

return 0;

}

a) Number of ways to make change: 6

**b) Number of ways to make change: 7**

c) Number of ways to make change: 8

d) Number of ways to make change: 10

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

Q16) Given a number N, generate bit patterns from 0 to 2^N-1 such that successive patterns differ by one bit.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | N = 2 | N = 3 | N = 2 |
| **Output** | 00 01 11 10 | 000 001 011 010 110 111 101 100 | 0 1 |

Solution :

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

**using namespace std;**

**void GreyCode(int n)**

**{**

**// power of 2**

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

**{**

**// Generating the decimal values of gray code then using**

**// bitset to convert them to binary form**

**int val = (i ^ (i >> 1));**

**// Using bitset**

**bitset<32> r(val);**

**// Converting to string**

**string s = r.to\_string();**

**cout << s.substr(32 - n) << " ";**

**}**

**}**

**// Driver Code**

**int main()**

**{**

**int n;**

**n = 4;**

**// Function call**

**GreyCode(n);**

**return 0;**

**}**

Q17) Every positive fraction can be represented as sum of unique unit fractions. A fraction is unit fraction if numerator is 1 and denominator is a positive integer, for example 1/3 is a unit fraction. Such a representation is called Egyptian Fraction as it was used by ancient Egyptians.

Following are a few examples:

Egyptian Fraction Representation of 2/3 is 1/2 + 1/6

Egyptian Fraction Representation of 6/14 is 1/3 + 1/11 + 1/231

Egyptian Fraction Representation of 12/13 is 1/2 + 1/3 + 1/12 + 1/156

Write a C++ program to print a fraction in Egyptian Form using Greedy Algorithm

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | numerator = 6,  denominator = 14 | numerator = 2, denominator = 9 | numerator = 12,  denominator = 1 |
| **Output** | Egyptian Fraction representation of 6/14 is  1/3 + 1/11 + 1/231 | Egyptian Fraction representation of 2/9 is  1/5 + 1/45 | Egyptian Fraction representation of 12/1 is  12 |

Solution :

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

**using namespace std;**

**void egyptianFraction(int n, int d)**

**{**

**//When Both Numerator and denominator becomes zero then we simply return;**

**if (d == 0 || n == 0)**

**return;**

**if (d % n == 0) {**

**cout << "1/" << d / n;**

**return;**

**}**

**if (n % d == 0) {**

**cout << n / d;**

**return;**

**}**

**if (n > d) {**

**cout << n / d << " + ";**

**egyptianFraction(n % d, d);**

**return;**

**}**

**int x = d / n + 1;**

**cout << "1/" << x << " + ";**

**egyptianFraction(n \* x - d, d \* x);**

**}**

**int main()**

**{**

**int numerator = 6, denominator = 14;**

**cout << "Egyptian Fraction representation of "**

**<< numerator << "/" << denominator << " is"**

**<< endl;**

**egyptianFraction(numerator, denominator);**

**return 0;**

**}**

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

Q18) A stable tower of height n is a tower consisting of exactly n tiles of unit height stacked vertically in such a way,

that no bigger tile is placed on a smaller tile. We have an infinite number of tiles of sizes 1, 2, …, m. The task is to calculate the number of the different stable towers of height n that can be built from these tiles, with a restriction that you can use at most k tiles of each size in the tower.

n (height of the tower), m (number of available blocks), and k (maximum consecutive blocks that can be placed)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | n = 3, m = 3, k = 1. | n = 3, m = 3, k = 2. | n = 10, m = 4, k = 2. |
| **Output** | 1 | 7 | 0 |

Solution :

**// CPP program to find number of ways to make stable tower of given height.**

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

**using namespace std;**

**#define N 100**

**int possibleWays(int n, int m, int k)**

**{**

**int dp[N][N];**

**int presum[N][N];**

**memset(dp, 0, sizeof dp);**

**memset(presum, 0, sizeof presum);**

**// Initializing 0th row to 0.**

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

**dp[0][i] = 0;**

**presum[0][i] = 1;**

**}**

**// Initializing 0th column to 0.**

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

**presum[i][0] = dp[i][0] = 1;**

**// For each row from 1 to m**

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

**// For each column from 1 to n.**

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

**// Initializing dp[i][j] to presum of (i - 1, j).**

**dp[i][j] = presum[i - 1][j];**

**if (j > k) {**

**dp[i][j] -= presum[i - 1][j - k - 1];**

**}**

**}**

**// Calculating presum for each i, 1 <= i <= n.**

**for (int j = 1; j < n + 1; j++)**

**presum[i][j] = dp[i][j] + presum[i][j - 1];**

**}**

**return dp[m][n];**

**}**

**int main()**

**{**

**int n = 3, m = 3, k = 2;**

**cout << possibleWays(n, m, k) << endl;**

**return 0;**

**}**