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

**ST-4 (SET-III)**

**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 key characteristic of a "Counting" type 1-Dimensional DP problem?
2. Minimizing a value
3. Maximizing a value
4. **Counting the number of valid solutions**
5. Finding a specific solution

2) What is the primary goal of dynamic programming in grid-based problems?

1. To create complex algorithms
2. To find the optimal solution to any problem
3. To reduce the time complexity of solving problems
4. **To solve problems by breaking them into smaller, overlapping subproblems**

3) In the Unbounded Knapsack problem, what does the term "unbounded" signify?

1. There are no constraints on item selection.
2. Each item can be selected at most once.
3. **Each item can be selected multiple times.**
4. The knapsack capacity is unlimited.

4) The 0/1 Knapsack problem is an example of which dynamic programming pattern?

1. Top-down
2. **Bottom-up**
3. Memoization
4. Greedy

5) What is the primary advantage of using the bottom-up approach in multidimensional dynamic programming?

1. It is easier to implement.
2. **It reduces space complexity.**
3. It avoids overlapping subproblems.
4. It ensures optimal solutions.

6) In Dynamic Programming on Trees, what is the main benefit of solving subproblems efficiently?

1. It reduces the number of nodes in the tree.
2. **It speeds up the execution of the program.**
3. It avoids using memoization.
4. It allows for more complex problems to be solved.

7) In Huffman coding, what is the main idea behind assigning shorter codes to frequent characters?

1. To improve encryption
2. **To save memory space**
3. To speed up decoding
4. To improve sorting

8) What is the binary representation of 10 in two's complement?

1. **1111 0110**
2. 1010 0101
3. 0011 1010
4. 1101 1011

9) What is the primary purpose of the extended Euclidean algorithm?

1. Finding prime numbers
2. **Solving Diophantine equations**
3. Calculating modular exponentiation
4. Computing Euler's totient function

10) Mala has a colouring book in which each English letter is drawn two times. She wants to paint each of these 52 prints with one of k colours, such that the colour-pairs used to colour any two letters are different. Both prints of a letter can also be coloured with the same colour. What is the minimum value of k that satisfies this requirement ?

1. 9
2. 8
3. **7**
4. 6

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

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

11) A sub-sequence of a given sequence is just the given sequence with some elements (possibly none or all) left out. We are given two sequences X[m] and Y[n] of lengths m and n respectively, with indexes of X and Y starting from 0. We wish to find the length of the longest common sub-sequence(LCS) of X[m] and Y[n] as l(m,n), where an incomplete recursive definition for the function l(i,j) to compute the length of The LCS of X[m] and Y[n] is given below:

l(i,j) = 0, if either i=0 or j=0

= expr1, if i,j > 0 and X[i-1] = Y[j-1]

= expr2, if i,j > 0 and X[i-1] != Y[j-1]

a) expr1 ≡ l(i-1, j) + 1

b) expr1 ≡ l(i, j-1)

**c) expr2 ≡ max(l(i-1, j), l(i, j-1))**

d) expr2 ≡ max(l(i-1,j-1),l(i,j))

12) Consider two strings A = "qpqrr" and B = "pqprqrp". Let x be the length of the longest common subsequence (not necessarily contiguous) between A and B and let y be the number of such longest common subsequences between A and B. Then x + 10y = \_\_\_.

a) 33

b) 23

c) 43

**d) 34**

13) Six files F1, F2, F3, F4, F5 and F6 have 100, 200, 50, 80, 120, 150 records respectively. In what order should they be stored so as to optimize act. Assume each file is accessed with the same frequency

**a) F3, F4, F1, F5, F6, F2**

b) F2, F6, F5, F1, F4, F3

c) F1, F2, F3, F4, F5, F6

d) Ordering is immaterial as all files are accessed with the same frequency

14) Consider the following code snippet for checking whether a number is power of 2 or not.

/\* Incorrect function to check if x is power of 2\*/

bool isPowerOfTwo (unsigned int x)

{

return (!(x&(x-1)));

}

What is wrong with above function?

a) It does reverse of what is required

b) It works perfectly fine for all values of x.

**c) It does not work for x = 0**

d) It does not work for x = 1

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

#include <iostream>

using namespace std;

int main() {

int arr[] = {1, 3, 2, 4, 6};

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

int target = 8;

int dp[9] = {0};

dp[0] = 1;

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

for (int j = arr[i]; j <= target; j++) {

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

}

}

cout << "Number of ways to reach the target: " << dp[target] << endl;

return 0;

}

a) Number of ways to reach the target: 18

b) Number of ways to reach the target: 21

**c) Number of ways to reach the target: 17**

d) Number of ways to reach the target: 4

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

Q16) You are given a list of N coins of different denominations. You can pay an amount equivalent to any 1 coin and can acquire that coin. In addition, once you have paid for a coin, we can choose at most K more coins and can acquire those for free. The task is to find the minimum amount required to acquire all the N coins for a given value of K.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | coin[] = {100, 20, 50, 10, 2, 5},  k = 3 | coin[] = {1, 2, 5, 10, 20, 50},  k = 3 | coin[] = {8, 5, 3, 10,2, 1, 15, 25}  k = 2 |
| **Output** | 7 | 3 | 6 |

Solution :

**// C++ program to acquire all n coins**

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

**using namespace std;**

**// function to calculate min cost**

**int minCost(int coin[], int n, int k)**

**{**

**// sort the coins value**

**sort(coin, coin + n);**

**// calculate no. of coins needed**

**int coins\_needed = ceil(1.0 \* n / (k + 1));**

**// calculate sum of all selected coins**

**int ans = 0;**

**for (int i = 0; i <= coins\_needed - 1; i++)**

**ans += coin[i];**

**return ans;**

**}**

**int main()**

**{**

**int coin[] = {8, 5, 3, 10,2, 1, 15, 25};**

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

**int k = 3;**

**cout << minCost(coin, n, k);**

**return 0;**

**}**

Q17) Compute the parity of a number using XOR and table look-up Parity of a number refers to whether it contains an odd or even number of 1-bits. The number has “odd parity”, if it contains odd number of 1-bits and is “even parity” if it contains even number of 1-bits.

1 --> parity of the set is odd

0 --> parity of the set is even

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | 254 | 7 | 125 |
| **Output** | Odd Parity | Even Parity | Even Parity |

Solution :

**// CPP program to illustrate Compute the parity of a number using XOR**

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

**// Generating the look-up table while pre-processing**

**#define P2(n) n, n ^ 1, n ^ 1, n**

**#define P4(n) P2(n), P2(n ^ 1), P2(n ^ 1), P2(n)**

**#define P6(n) P4(n), P4(n ^ 1), P4(n ^ 1), P4(n)**

**#define LOOK\_UP P6(0), P6(1), P6(1), P6(0)**

**// LOOK\_UP is the macro expansion to generate the table**

**unsigned int table[256] = { LOOK\_UP };**

**// Function to find the parity**

**int Parity(int num)**

**{**

**// Number is considered to be of 32 bits**

**int max = 16;**

**// Dividing the number into 8-bit chunks while performing X-OR**

**while (max >= 8) {**

**num = num ^ (num >> max);**

**max = max / 2;**

**}**

**// Masking the number with 0xff (11111111)to produce valid 8-bit result**

**return table[num & 0xff];**

**}**

**int main()**

**{**

**unsigned int num = 125;**

**// Result is 1 for odd parity, 0 for even parity**

**bool result = Parity(num);**

**// Printing the desired result**

**result ? std::cout << "Odd Parity" :**

**std::cout << "Even Parity";**

**return 0;**

**}**

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

Q18) Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays, val[0..n-1] and wt[0..n-1] represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the items such that sum of the weights of those items of a given subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don’t pick it (0-1 property).

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | val[] = {60, 100, 120}  wt[] = {10, 20, 30}  W = 50 | val[] = {40, 100, 50, 60}  wt[] = {20, 10, 40, 30}  W = 60 | val[] = { 200, 500, 320 }  int wt[] = { 10, 50, 40 }  W = 250 |
| **Output** | 220 //maximum value that can be obtained  30 20 //weights 20 and 30 are included. | 200  30 20 10 | 1020  40 50 10 |

Solution :

**// CPP code for Dynamic Programming based solution for 0-1 Knapsack problem**

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

**#include <iostream>**

**using namespace std;**

**// A utility function that returns maximum of two integers**

**int max(int a, int b) { return (a > b) ? a : b; }**

**// Prints the items which are put in a knapsack of capacity W**

**void printknapSack(int W, int wt[], int val[], int n)**

**{**

**int i, w;**

**int K[n + 1][W + 1];**

**// Build table K[][] in bottom up manner**

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

**for (w = 0; w <= W; w++) {**

**if (i == 0 || w == 0)**

**K[i][w] = 0;**

**else if (wt[i - 1] <= w)**

**K[i][w] = max(val[i - 1] +**

**K[i - 1][w - wt[i - 1]], K[i - 1][w]);**

**else**

**K[i][w] = K[i - 1][w];**

**}**

**}**

**// stores the result of Knapsack**

**int res = K[n][W];**

**cout<< res << endl;**

**w = W;**

**for (i = n; i > 0 && res > 0; i--) {**

**// either the result comes from the top (K[i-1][w]) or from (val[i-1] + K[i-1]**

**// [w-wt[i-1]]) as in Knapsack table. If it comes from the latter one/ it means the item is included.**

**if (res == K[i - 1][w])**

**continue;**

**else {**

**// This item is included.**

**cout<<" "<<wt[i - 1] ;**

**// Since this weight is included its value is deducted**

**res = res - val[i - 1];**

**w = w - wt[i - 1];**

**}**

**}**

**}**

**int main()**

**{**

**int val[] = { 60, 100, 120 };**

**int wt[] = { 10, 20, 30 };**

**int W = 50;**

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

**printknapSack(W, wt, val, n);**

**return 0;**

**}**