Page No.
Date

Assignment No: 01

Kavita Sahu
BA0191109

Title: Fibonacci series

Date of completion:-

Objective:-
To write a non-recursive and recursive program to calculate fibonacci numbers.

Problem statement:-
To write a non-recursive and recursive program to calculate fibonacci numbers and analyze the time and space complexity

Software and Hardware Requirements:-

Theory:-
The Fibonacci series can be described using the mathematical equation
$F(n) = f(n-1) + F(n-2)$ $F(n) = f(n-1) \neq f(n-2)$, with the condition $F(1) = 0$ and $F(2) = 1$
The sum of the previous two numbers in the sequence is the next number of the sequence. The fibonacci series can be implemented in using recursion or without using recursion.

1) Using Recursion -

In the code, we created a function named fib(), which has an integer as the parameter and return value data type. In the main function, we iterate over a loop from 1 to n times, and at each iteration called the fib() function. In fib() function, if the parameter passed is 0 or 1., we will return the same value otherwise we will return the sum of recursive calls with parameter values one and two less than our current parameter, which is fib(i-1) + fib(i-2).

Time and space complexity of Recursive method -

Time Complexity - T(2^N).

Space Complexity - O(N).

2) Without Recursion -

In the fib() function. we create an array of size n, in our case an array of size 5 to hold the Fibonacci numbers. The first and second element in the array is initialized to 0 and 1, respectively. Later we used for loop to find the other elements of the array i.e the fibonacci numbers using the formula arr[i] = arr[i-1] + arr[i-2]. Later we

iii) print all the elements in the array.

Time complexity and Space complexity of Dynamic Programming -
Time Complexity is T(N).
Space Complexity is O(N)

Conclusion -
We have implemented the non-recursive and recursive program for calculating the Fibonacci numbers.

Assignment No. 2

Page No.
Date

Kavita Sahu
BAC019109

Title: Huffman Encoding

Date of completion:-

Objective:-
To implement Huffman Encoding.

Problem statement-
Write a program to implement Huffman Encoding using a greedy strategy.

Software and Hardware Requirements-

Theory-
Huffman coding is a lossless data compression algorithm. The idea is to assign variable length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

Huffman coding is a technique of compressing data to reduce the size without losing any of the details.

Steps to build Huffman Tree:-

Page No.
Date

Input is an array of unique characters along with their frequency of occurrences and the output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat step 2 and 3 until heap contains only one node.

The remaining node is the root node and the tree is complete.

Conclusion -

We have implemented the Huffman coding using greedy strategy.

Assignment No. 3

Page No.
Date

Kavita Sahu
BAC019103

Title:- Knapsack problem.

Date of completion:-

Objective:-
To maximize the sum of the profits of the items selected in the knapsack with sum of the weights less than or equal to the knapsack capacity.

Problem statement:-
Write a program to solve a fractional knapsack problem using greedy method.

Software and Hardware Requirements:-

Theory-
The fractional knapsack problem using the greedy method is an efficient method to solve it, where you need to sort the items according to their ratio of value/weight. In a fractional knapsack, we can break items to maximize the knapsack's total value.
This problem in which we can break an item is also called the fractional knapsack problem.

Knapsack problem using Greedy Method -
In this method, the knapsack's filling is done so that the maximum capacity of the knapsack is utilized so that maximum profit can be earned from it.
The knapsack problem using the Greedy Method is referred to as:
Given a list of n objects, say $(I_1, I_2, \ldots I_n)$ and a knapsack (or bag).
The capacity of the knapsack is M.
Each object I_j has a weight w_j and a profit of P_j.

Maximize the profit =

$$\sum_{j=1}^{n} P_j x_j = \sum_{j=1}^{n} w_j \cdot x_j \leq M \text{ and } x_i \in \{0, \ldots 1\}, 1 \leq j \leq n.$$

Conclusion -
Have implemented the knapsack problem using greedy method.

Assignment No. 4.

Kavita Sahu
BAC019103

Title:- 0/1 Knapsack problem

Date of completion:-

Objective:-
To find a subset of items leading to the maximum total profit while respecting the capacity constraint.

Problem statement -
Write a program to solve a 0-1 knapsack problem using dynamic programming or branch and bound strategy.

Software and Hardware Requirements:-

Theory -
The 0/1 knapsack problem means that the items are either completely or no items are filled in a knapsack.
For example -
We have two items having weights 2kg and 3kg, respectively. If we pick the 2kg item then we cannot pick 1kg item from the 2kg item.
We have to pick the 2kg item completely.

This is a 0/1 knapsack problem in which either we pick the item (profit) or we don't pick that item.
The 0/1 knapsack problem is solved by the dynamic programming.

Conclusion -
Implemented the 0/1 knapsack problem using dynamic programming.

Assignment No. 5.

Kavita Sahu
BAC019109

Title:- N Queen problem.

Date of Completion:-

Objective:-
To suitably place N number of Queen on an N x N chessboard in a way that there is no conflict between them due to the arrangement.

Problem statement:-
Design n Queen matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen's matrix.

Software and Hardware Requirements-

Theory:-
The N-Queen is the problem of placing N chess queen on a N x N chessboard so that no two queens attacks each other.

The idea is to place queens one by one in different columns, starting from the leftmost column.

When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

1) Start in the leftmost column.
2) If all queens are placed return true.
3) Try all rows in the current column. Do following for every tried row:-
a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to solution.
b) If placing the queen in [row, column] leads to a solution then return true.
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack and go to step (a) to try other row.
4) If all rows have been tried and nothing worked, return false to trigger backtracking.

Conclusion:
Implemented n-Queen matrix using Backtracking